告知我们有关下载 PDF 体验的信息。

高级着色器语言 (HLSL)

项目 • 2022/09/22

HLSL 是与 DirectX 中的可编程着色器一起使用的类似 C 的高级着色器语言。

例如，可以使用 HLSL 编写 顶点着色器或 像素着色器，并在 Direct3D 应用程序中呈现器的实现中使用这些着色器。

或者，可以使用 HLSL 编写计算着色器，或许可以实现物理模拟。 但是，例如，如果你倾向于在计算着色器中将自己的卷积运算符 (写入图像处理) 为 HLSL，则在使用 Direct 机器学习 (DirectML) 时，你将获得更好的性能。

从 DirectX 9) 开始， (创建 HLSL 来设置可编程的 3D 管道。 可以使用 HLSL 指令对整个管道进行编程。

下一步到哪里

- HLSL 编程指南
- HLSL 参考

HLSL 编程指南

有关 HLSL 的概念性介绍，请参阅 HLSL 编程指南。

编程指南讨论了不同类型的着色器阶段，以及如何创建、编译、优化、绑定和链接着色器。

你还将在此处找到有关已发布的着色器模型版本的连续几代概述和发行说明，回溯到 HLSL 着色器模型 5。

HLSL 参考

有关 HLSL 参考文档，请参阅 HLSL 参考。

参考部分包含语言语法和内置于 HLSL 的内部函数的完整列表，以便简化编码要求。

此外，还可以找到着色器模型与配置文件的讨论，以及着色器模型引用内容回退到 HLSL 着色器模型 1。 还有内容涵盖程序集指令、D3DCompiler 工具以及着色器可以返回的错误和警告信息。

HLSL 编程指南

项目 • 2022/09/22

数据以基元流的形式进入图形管道，并由着色器阶段处理。 实际的着色器阶段取决于 Direct3D 的版本，但当然包括顶点、像素和几何阶段。 其他阶段包括用于分割的外壳着色器和域着色器，以及计算着色器。 这些阶段使用高级底纹语言 (HLSL) 完全可编程。

可以在作者或运行时编译 HLSL 着色器，并在运行时设置为适当的管道阶段。 Direct3D 9 着色器可以使用着色 器模型 1、 着色器模型 2 和 着色器模型 3 进行设计：Direct3D 10 着色 器只能在着色器模型 4 上设计。 Direct3D 11 着色 器可以在着色器模型 5 上设计。 Direct3D 11.3 和 Direct3D 12 可以在 着色器模型 5.1 上设计，Direct3D 12 也可以在 着色器模型 6 上设计。

在本节中

主题	描述
使用着色器链接	我们演示如何创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。
在 Direct3D 9 中编写 HLSL 着色器	
在 Direct3D 9 中使用着色器	
在 Direct3D 10 中使用着色器	
优化 HLSL 着色器	
在 Visual Studio 中调试着色器	用于调试着色器的最新工具现在作为Microsoft Visual Studio中的一项功能提供，称为Visual Studio图形调试器。
编译着色器	现在，我们来了解各种方法来编译着色器代码和着色器代码的文件扩展名约定。
指定编译器目标	下面列出了 **D3DCompile*** 函数和 HLSL 编译器支持的各种配置文件的目标。
用于In-Place图像编辑的解压缩和打包DXGI_FORMAT	
使用 HLSL 最小精度	从Windows 8开始，图形驱动程序可以使用大于或等于指定位精度的任何精度来实现最小精度 HLSL 标量数据类型。

主题	描述
HLSL 着色器模型 5	
HLSL 着色器模型 5.1	本部分介绍着色器模型 5.1 的功能，因为它们在实践中适用于 D3D12 和 D3D11.3。 所有 DirectX 12 硬件都支持着色器模型 5.1。
HLSL 着色器模型 6.0	描述添加到 HLSL 着色器模型 6.0 中的波形操作内部函数。
HLSL 着色器模型 6.4	介绍添加到 HLSL 着色器模型 6.4 的机器学习内部函数。

相关主题

- HLSL
- HLSL 参考

使用着色器链接

项目 • 2022/09/22

我们演示如何创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。 从 Windows 8.1 开始，支持着色器链接。

目的： 了解如何使用着色器链接。

先决条件

我们假定你熟悉 C++。 你还需要具有图形编程概念方面的基本经验。

完成总时间： 60 分钟。

下一步

另请参阅 HLSL 编译器 API。

我们将向你展示如何：

- 编译着色器代码
- 将编译的代码加载到着色器库中
- 将资源从源槽绑定到目标槽
- 为着色器构造函数链接图 (FLG)
- 使用着色器库链接着色器图，以生成 Direct3D 运行时可以使用的着色器 Blob

接下来，我们将创建着色器库，并将资源从源槽绑定到目标槽。

打包着色器库

相关主题

HLSL 编程指南

Direct3D 11 图形

DXGI

打包着色器库

项目 • 2022/09/23

下面介绍如何编译着色器代码、将编译的代码加载到着色器库中，并将资源从源槽绑定到目标槽。

目的： 打包用于着色器链接的着色器库。

先决条件

我们假定你熟悉 C++。 你还需要具有图形编程概念方面的基本经验。

完成时间： 30 分钟。

说明

1. 编译着色器代码

使用其中一个编译函数编译着色器代码。 例如，此代码片段使用 D3DCompile。

C++

```
    string source;

    ComPtr<ID3DBlob> codeBlob;
    ComPtr<ID3DBlob> errorBlob;
    HRESULT hr = D3DCompile(
        source.c_str(),
        source.size(),
        "ShaderModule",
        NULL,
        NULL,
        NULL,
        ("lib" + m_shaderModelSuffix).c_str(),
        D3DCOMPILE_OPTIMIZATION_LEVEL3,
        0,
        &codeBlob,
        &errorBlob
        );
```

源字符串包含未编译的 ASCII HLSL 代码。

2.将编译的代码加载到着色器库中。

调用 D3DLoadModule 函数，将编译的代码 (ID3DBlob) 加载到表示着色器库的 **模块** (ID3D11Module) 。

C++

```
    // Load the compiled library code into a module object.
    ComPtr<ID3D11Module> shaderLibrary;
    DX::ThrowIfFailed(D3DLoadModule(codeBlob->GetBufferPointer(), codeBlob-
>GetBufferSize(), &shaderLibrary));
```

3.将资源从源槽绑定到目标槽。

调用 ID3D11Module：：CreateInstance 方法以创建库 (ID3D11ModuleInstance) 实例，以便为实例定义资源绑定。

调用 ID3D11ModuleInstance 的绑定方法，将所需的资源从源槽绑定到目标槽。 资源可以是纹理、缓冲区、采样器、常量缓冲区或 UAV。 通常，将使用与源库相同的槽。

C++

```
    // Create an instance of the library and define resource bindings for
it.
    // In this sample we use the same slots as the source library however
this is not required.
    ComPtr<ID3D11ModuleInstance> shaderLibraryInstance;
    DX::ThrowIfFailed(shaderLibrary->CreateInstance("",
&shaderLibraryInstance));
    // HRESULTs for Bind methods are intentionally ignored as compiler
optimizations may eliminate the source
    // bindings. In these cases, the Bind operation will fail, but the final
shader will function normally.
    shaderLibraryInstance->BindResource(0, 0, 1);
    shaderLibraryInstance->BindSampler(0, 0, 1);
    shaderLibraryInstance->BindConstantBuffer(0, 0, 0);
    shaderLibraryInstance->BindConstantBuffer(1, 1, 0);
    shaderLibraryInstance->BindConstantBuffer(2, 2, 0);
```

此 HLSL 代码显示源库使用与 ID3D11ModuleInstance 上述绑定方法中使用的槽相同的槽 (t0、s0、b0、b1 和 b2) 。

syntax

```
// This is the default code in the fixed header section.
Texture2D<float3> Texture : register(t0);
SamplerState Anisotropic : register(s0);
cbuffer CameraData : register(b0)
{
    float4x4 Model;
```

```
    float4x4 View;
    float4x4 Projection;
};
cbuffer TimeVariantSignals : register(b1)
{
    float SineWave;
    float SquareWave;
    float TriangleWave;
    float SawtoothWave;
};

// This code is not displayed, but is used as part of the linking process.
cbuffer HiddenBuffer : register(b2)
{
    float3 LightDirection;
};
```

总结和后续步骤

我们编译了着色器代码，将编译的代码加载到着色器库中，并将资源从源槽绑定到目标槽。

接下来，我们将构造函数链接图 (FLG) 着色器，将它们链接到已编译的代码，并生成 Direct3D 运行时可以使用的着色器 blob。

构造函数链接关系图并将其链接到已编译的代码

相关主题

使用着色器链接

构造函数链接关系图并将其链接到已编译的代码

项目 • 2022/09/22

下面介绍如何为着色器构造函数链接图 (FLG) ，以及如何将这些着色器链接到着色器库，以生成 Direct3D 运行时可以使用的着色器 blob。

目的： 构造函数链接关系图并将其链接到已编译的代码。

先决条件

我们假定你熟悉 C++。 你还需要具有图形编程概念方面的基本经验。

我们还假设你通过 打包着色器库。

完成时间： 30 分钟。

说明

1.为顶点着色器构造函数链接图。

调用 D3DCreateFunctionLinkingGraph 函数以创建 function-linking-graph (ID3D11FunctionLinkingGraph) 来表示顶点着色器。

使用 D3D11_PARAMETER_DESC 结构的数组来定义顶点着色器的输入参数。 输入汇编器阶段将输入参数馈送到顶点着色器。 顶点着色器的输入参数布局与编译代码中的顶点着色器的布局匹配。 定义输入参数后，调用 ID3D11FunctionLinkingGraph：：SetInputSignature 方法来定义顶点着色器的输入节点 (ID3D11LinkingNode) 。

C++

```
            ComPtr<ID3D11FunctionLinkingGraph> vertexShaderGraph;
            DX::ThrowIfFailed(D3DCreateFunctionLinkingGraph(0,
&vertexShaderGraph));

            // Define the main input node which will be fed by the Input
Assembler pipeline stage.
            static const D3D11_PARAMETER_DESC vertexShaderInputParameters[]
=
            {
                {"inputPos",  "POSITION0", D3D_SVT_FLOAT, D3D_SVC_VECTOR, 1,
3, D3D_INTERPOLATION_LINEAR, D3D_PF_IN, 0, 0, 0, 0},
                {"inputTex",  "TEXCOORD0", D3D_SVT_FLOAT, D3D_SVC_VECTOR, 1,
```

```
2, D3D_INTERPOLATION_LINEAR, D3D_PF_IN, 0, 0, 0, 0},
                {"inputNorm", "NORMAL0",   D3D_SVT_FLOAT, D3D_SVC_VECTOR, 1,
3, D3D_INTERPOLATION_LINEAR, D3D_PF_IN, 0, 0, 0, 0}
            };
            ComPtr<ID3D11LinkingNode> vertexShaderInputNode;
            LinkingThrowIfFailed(vertexShaderGraph-
>SetInputSignature(vertexShaderInputParameters,
ARRAYSIZE(vertexShaderInputParameters),
                &vertexShaderInputNode), vertexShaderGraph.Get());
```

调用 ID3D11FunctionLinkingGraph：：CallFunction 方法为主顶点着色器函数创建节点，并调用 ID3D11FunctionLinkingGraph：:P assValue ，将输入节点中的值传递给主顶点着色器函数的节点。

C++

```
            // Create a node for the main VertexFunction call using the
output of the helper functions.
            ComPtr<ID3D11LinkingNode> vertexFunctionCallNode;
            LinkingThrowIfFailed(vertexShaderGraph->CallFunction("",
shaderLibrary.Get(), "VertexFunction", &vertexFunctionCallNode),
                vertexShaderGraph.Get());

            // Define the graph edges from the input node and helper
function nodes.
            LinkingThrowIfFailed(vertexShaderGraph-
>PassValue(homogenizeCallNodeForPos.Get(), D3D_RETURN_PARAMETER_INDEX,
                vertexFunctionCallNode.Get(), 0), vertexShaderGraph.Get());
            LinkingThrowIfFailed(vertexShaderGraph-
>PassValue(vertexShaderInputNode.Get(), 1, vertexFunctionCallNode.Get(), 1),
                vertexShaderGraph.Get());
            LinkingThrowIfFailed(vertexShaderGraph-
>PassValue(homogenizeCallNodeForNorm.Get(), D3D_RETURN_PARAMETER_INDEX,
                vertexFunctionCallNode.Get(), 2), vertexShaderGraph.Get());
```

使用 D3D11_PARAMETER_DESC 结构的数组来定义顶点着色器的输出参数。 顶点着色器将其输出参数馈送给像素着色器。 顶点着色器的输出参数的布局与编译代码中的像素着色器的布局匹配。 定义输出参数后，调用 ID3D11FunctionLinkingGraph：：SetOutputSignature 方法，为顶点着色器定义输出节点 (ID3D11LinkingNode) 。

C++

```
            // Define the main output node which will feed the Pixel Shader
pipeline stage.
            static const D3D11_PARAMETER_DESC vertexShaderOutputParameters[]
=
            {
                {"outputTex",  "TEXCOORD0",   D3D_SVT_FLOAT, D3D_SVC_VECTOR,
1, 2, D3D_INTERPOLATION_UNDEFINED, D3D_PF_OUT, 0, 0, 0, 0},
                {"outputNorm", "NORMAL0",     D3D_SVT_FLOAT, D3D_SVC_VECTOR,
```

```
1, 3, D3D_INTERPOLATION_UNDEFINED, D3D_PF_OUT, 0, 0, 0, 0},
                {"outputPos",  "SV_POSITION", D3D_SVT_FLOAT, D3D_SVC_VECTOR,
1, 4, D3D_INTERPOLATION_UNDEFINED, D3D_PF_OUT, 0, 0, 0, 0}
            };
            ComPtr<ID3D11LinkingNode> vertexShaderOutputNode;
            LinkingThrowIfFailed(vertexShaderGraph-
>SetOutputSignature(vertexShaderOutputParameters,
ARRAYSIZE(vertexShaderOutputParameters),
                &vertexShaderOutputNode), vertexShaderGraph.Get());
```

调用 ID3D11FunctionLinkingGraph：:P assValue ，将主顶点着色器函数的节点的值传递到输出节点。

C++

```
            LinkingThrowIfFailed(vertexShaderGraph-
>PassValue(vertexFunctionCallNode.Get(), 0, vertexShaderOutputNode.Get(),
2),
                vertexShaderGraph.Get());
            LinkingThrowIfFailed(vertexShaderGraph-
>PassValue(vertexFunctionCallNode.Get(), 1, vertexShaderOutputNode.Get(),
0),
                vertexShaderGraph.Get());
            LinkingThrowIfFailed(vertexShaderGraph-
>PassValue(vertexFunctionCallNode.Get(), 2, vertexShaderOutputNode.Get(),
1),
                vertexShaderGraph.Get());
```

调用 ID3D11FunctionLinkingGraph：：CreateModuleInstance 方法以完成顶点着色器图形。

C++

```
            // Finalize the vertex shader graph.
            ComPtr<ID3D11ModuleInstance> vertexShaderGraphInstance;
            LinkingThrowIfFailed(vertexShaderGraph-
>CreateModuleInstance(&vertexShaderGraphInstance, nullptr),
vertexShaderGraph.Get());
```

2. 链接顶点着色器

调用 D3DCreateLinker 函数以创建一个链接器 (ID3D11Linker) ，该链接器库可用于链接在 打包着色器库中创建的着色器库 的实例以及在上一步中创建的顶点着色器图形的实例。 调用 ID3D11Linker：：UseLibrary 方法以指定要用于链接的着色器库。 调用 ID3D11Linker：：Link 方法，将着色器库与顶点着色器图链接，并生成指向可用于访问已编译顶点着色器代码的 ID3DBlob 接口的指针。 然后，可以将此编译的顶点着色器代码

传递给 ID3D11Device：：CreateVertexShader 方法，以创建顶点着色器对象，并将 ID3D11Device：：CreateInputLayout 方法传递给创建输入布局对象。

C++

```
            // Create a linker and hook up the module instance.
            ComPtr<ID3D11Linker> linker;
            DX::ThrowIfFailed(D3DCreateLinker(&linker));
            DX::ThrowIfFailed(linker-
>UseLibrary(shaderLibraryInstance.Get()));

            // Link the vertex shader.
            ComPtr<ID3DBlob> errorBlob;
            if (FAILED(linker->Link(vertexShaderGraphInstance.Get(), "main",
("vs" + m_shaderModelSuffix).c_str(), 0, &vertexShaderBlob,
                &errorBlob)))
            {
                throw errorBlob;
            }


    ComPtr<ID3D11VertexShader> vertexShader;
    DX::ThrowIfFailed(
        device->CreateVertexShader(
            vertexShaderBlob->GetBufferPointer(),
            vertexShaderBlob->GetBufferSize(),
            nullptr,
            &vertexShader
            )
        );
    context->VSSetShader(vertexShader.Get(), nullptr, 0);
    D3D11_INPUT_ELEMENT_DESC inputLayoutDesc[] =
    {
        { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
D3D11_INPUT_PER_VERTEX_DATA, 0 },
        { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT,    0, 12,
D3D11_INPUT_PER_VERTEX_DATA, 0 },
        { "NORMAL",   0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 20,
D3D11_INPUT_PER_VERTEX_DATA, 0 }
    };
    ComPtr<ID3D11InputLayout> inputLayout;
    DX::ThrowIfFailed(device->CreateInputLayout(inputLayoutDesc,
ARRAYSIZE(inputLayoutDesc), vertexShaderBlob->GetBufferPointer(),
vertexShaderBlob->GetBufferSize(), &inputLayout));
    context->IASetInputLayout(inputLayout.Get());
```

3.为像素着色器构造函数链接图。

调用 D3DCreateFunctionLinkingGraph 函数以创建函数链接图形 (ID3D11FunctionLinkingGraph) 来表示像素着色器。

使用 D3D11_PARAMETER_DESC 结构的数组来定义像素着色器的输入参数。 顶点着色器阶段将输入参数馈送到像素着色器。 像素着色器的输入参数的布局与编译代码中的像素着色器的布局匹配。 定义输入参数后，调用 ID3D11FunctionLinkingGraph：：SetInputSignature 方法，为像素着色器定义输入节点 (ID3D11LinkingNode) 。

C++

```
            ComPtr<ID3D11FunctionLinkingGraph> pixelShaderGraph;
            DX::ThrowIfFailed(D3DCreateFunctionLinkingGraph(0,
&pixelShaderGraph));

            // Define the main input node which will be fed by the vertex
shader pipeline stage.
            static const D3D11_PARAMETER_DESC pixelShaderInputParameters[] =
            {
                {"inputTex",  "TEXCOORD0",    D3D_SVT_FLOAT, D3D_SVC_VECTOR,
1, 2, D3D_INTERPOLATION_UNDEFINED, D3D_PF_IN, 0, 0, 0, 0},
                {"inputNorm", "NORMAL0",      D3D_SVT_FLOAT, D3D_SVC_VECTOR,
1, 3, D3D_INTERPOLATION_UNDEFINED, D3D_PF_IN, 0, 0, 0, 0},
                {"inputPos",  "SV_POSITION", D3D_SVT_FLOAT, D3D_SVC_VECTOR,
1, 4, D3D_INTERPOLATION_UNDEFINED, D3D_PF_IN, 0, 0, 0, 0}
            };
            ComPtr<ID3D11LinkingNode> pixelShaderInputNode;
            LinkingThrowIfFailed(pixelShaderGraph-
>SetInputSignature(pixelShaderInputParameters,
ARRAYSIZE(pixelShaderInputParameters),
                &pixelShaderInputNode), pixelShaderGraph.Get());
```

调用 ID3D11FunctionLinkingGraph：：CallFunction 方法，为主像素着色器函数创建节点，并调用 ID3D11FunctionLinkingGraph：:P assValue ，将输入节点中的值传递给主像素着色器函数的节点。

C++

```
            // Create a node for the main ColorFunction call and connect it
to the pixel shader inputs.
            ComPtr<ID3D11LinkingNode> colorValueNode;
            LinkingThrowIfFailed(pixelShaderGraph->CallFunction("",
shaderLibrary.Get(), "ColorFunction", &colorValueNode),
                pixelShaderGraph.Get());

            // Define the graph edges from the input node.
            LinkingThrowIfFailed(pixelShaderGraph-
>PassValue(pixelShaderInputNode.Get(), 0, colorValueNode.Get(), 0),
                pixelShaderGraph.Get());
            LinkingThrowIfFailed(pixelShaderGraph-
>PassValue(pixelShaderInputNode.Get(), 1, colorValueNode.Get(), 1),
                pixelShaderGraph.Get());
```

使用 D3D11_PARAMETER_DESC 结构的数组来定义像素着色器的输出参数。 像素着色器将其输出参数馈送到 输出合并阶段。 定义输出参数后，调用 ID3D11FunctionLinkingGraph：：SetOutputSignature 方法以定义像素着色器的输出节点 (ID3D11LinkingNode)，并调用 ID3D11FunctionLinkingGraph：:P assValue 将像素着色器函数节点中的值传递给输出节点。

```
C++
```

```
            // Define the main output node which will feed the Output Merger
pipeline stage.
            D3D11_PARAMETER_DESC pixelShaderOutputParameters[] =
            {
                {"outputColor", "SV_TARGET", D3D_SVT_FLOAT, D3D_SVC_VECTOR,
1, 4, D3D_INTERPOLATION_UNDEFINED, D3D_PF_OUT, 0, 0, 0, 0}
            };
            ComPtr<ID3D11LinkingNode> pixelShaderOutputNode;
            LinkingThrowIfFailed(pixelShaderGraph-
>SetOutputSignature(pixelShaderOutputParameters,
ARRAYSIZE(pixelShaderOutputParameters),
                &pixelShaderOutputNode), pixelShaderGraph.Get());
            LinkingThrowIfFailed(pixelShaderGraph-
>PassValue(fillAlphaCallNode.Get(), D3D_RETURN_PARAMETER_INDEX,
pixelShaderOutputNode.Get(), 0),
                pixelShaderGraph.Get());
```

调用 ID3D11FunctionLinkingGraph：：CreateModuleInstance 方法以完成像素着色器图形。

```
C++
```

```
            // Finalize the pixel shader graph.
            ComPtr<ID3D11ModuleInstance> pixelShaderGraphInstance;
            LinkingThrowIfFailed(pixelShaderGraph-
>CreateModuleInstance(&pixelShaderGraphInstance, nullptr),
pixelShaderGraph.Get());
```

4.链接像素着色器

调用 D3DCreateLinker 函数以创建一个链接器 (ID3D11Linker)，该链接器库可用于链接在 打包着色器库中创建的着色器库 的实例以及在上一步中创建的像素着色器图形的实例。 调用 ID3D11Linker：：UseLibrary 方法以指定要用于链接的着色器库。 调用 ID3D11Linker：：Link 方法，将着色器库与像素着色器图链接，并生成指向 ID3DBlob 接口的指针，可用于访问编译的像素着色器代码。 然后，可以将此编译的像素着色器代码传递给 ID3D11Device：：CreatePixelShader 方法以创建像素着色器对象。

```
C++
```

```
            // Create a linker and hook up the module instance.
            ComPtr<ID3D11Linker> linker;
            DX::ThrowIfFailed(D3DCreateLinker(&linker));
            DX::ThrowIfFailed(linker-
>UseLibrary(shaderLibraryInstance.Get()));

            // Link the pixel shader.
            ComPtr<ID3DBlob> errorBlob;
            if (FAILED(linker->Link(pixelShaderGraphInstance.Get(), "main",
("ps" + m_shaderModelSuffix).c_str(), 0, &pixelShaderBlob, &errorBlob)))
            {
                throw errorBlob;
            }


    ComPtr<ID3D11PixelShader> pixelShader;
    DX::ThrowIfFailed(
        device->CreatePixelShader(
            pixelShaderBlob->GetBufferPointer(),
            pixelShaderBlob->GetBufferSize(),
            nullptr,
            &pixelShader
            )
        );
    context->PSSetShader(pixelShader.Get(), nullptr, 0);
```

总结

我们使用 ID3D11FunctionLinkingGraph 方法构造顶点和像素着色器图形，并以编程方式指定着色器结构。

这些图形构造由预编译函数调用序列组成，这些调用相互传递值。 FLG 节点 (ID3D11LinkingNode) 表示预编译库函数的输入和输出着色器节点和调用。 注册函数调用节点的顺序定义调用序列。 必须首先指定输入节点 (ID3D11FunctionLinkingGraph：：SetInputSignature) ，最后一个 (ID3D11FunctionLinkingGraph：：SetOutputSignature) 。 FLG 边缘定义如何将值从一个节点传递到另一个节点。 传递值的数据类型必须相同;没有隐式类型转换。 形状和重排规则遵循 HLSL 行为。 值只能在此序列中向前传递。

我们还使用 ID3D11Linker 方法将着色器库与着色器图链接，并为要使用的 Direct3D 运行时生成着色器 Blob。

祝贺你！ 现在，你已准备好在自己的应用中使用着色器链接。

相关主题

使用着色器链接

在 Direct3D 9 中编写 HLSL 着色器

项目 • 2022/09/23

- 顶点着色器基础知识
- 像素着色器基础知识
  - 纹理阶段和采样器状态
  - 像素着色器输入
  - 像素着色器输出
- 着色器输入和着色器变量
  - 声明着色器变量
  - 统一着色器输入
  - 不同的着色器输入和语义
  - 采样器和纹理对象
- 编写函数
- 流控制
- 相关主题

Vertex-Shader基础知识

在操作过程中，可编程顶点着色器取代 Microsoft Direct3D 图形管道完成的顶点处理。 使用顶点着色器时，固定函数管道会忽略有关转换和照明操作的状态信息。 禁用顶点着色器并返回固定函数处理时，将应用所有当前状态设置。

应在执行顶点着色器之前对高阶基元进行分割。 着色器处理后执行图面分割的实现必须以对应用程序和着色器代码不明显的方式执行。

至少，顶点着色器必须在同质剪辑空间中输出顶点位置。 （可选）顶点着色器可以输出纹理坐标、顶点颜色、顶点照明、雾因子等。

Pixel-Shader基础知识

像素处理由像素着色器对单个像素执行。 像素着色器与顶点着色器协同工作;顶点着色器的输出为像素着色器提供输入。 其他像素操作 (雾混合、模具操作和呈现目标混合) 执行着色器后发生。

纹理阶段和采样器状态

像素着色器完全取代了由多纹理混合器指定的像素混合功能，包括以前由纹理阶段状态定义的操作。 可以在着色器中初始化由标准纹理阶段状态控制的纹理采样和筛选操作，以

便缩小、放大、mip 筛选和包装寻址模式。 应用程序可以自由更改这些状态，而无需重新生成当前绑定着色器。 如果着色器在效果中设计，则可以更轻松地设置状态。

像素着色器输入

对于像素着色器版本ps_1_1 - ps_2_0、漫射和反射颜色在着色器使用前，将范围 0 到 1 中的 (固定) 饱和。

假定像素着色器的颜色值输入正确，但不能保证所有硬件) (。 从纹理坐标采样的颜色以正确的角度迭代，并在迭代期间固定到 0 到 1 范围。

像素着色器输出

对于像素着色器版本ps_1_1 - ps_1_4，像素着色器发出的结果是寄存器 r0 的内容。 无论着色器何时完成处理，它都会发送到雾阶段和呈现目标混合器。

对于像素着色器版本ps_2_0及更高版本，输出颜色从 oC0 - oC4 发出。

着色器输入和着色器变量

- 声明着色器变量
- 统一着色器输入
- 不同的着色器输入和语义
- 采样器和纹理对象

声明着色器变量

最简单的变量声明包括类型和变量名称，例如此浮点声明：

```
float fVar;
```

可以在同一语句中初始化变量。

```
float fVar = 3.1f;
```

可以声明变量数组，

```
```

```
int iVar[3];
```

或在同一语句中声明和初始化。

```
int iVar[3] = {1,2,3};
```

下面是一些声明，演示高级着色器语言 (HLSL) 变量的许多特征：

```
float4 color;
uniform float4 position : POSITION;
const float4 lightDirection = {0,0,1};
```

数据声明可以使用任何有效类型，包括：

- DirectX HLSL) (数据类型
- 矢量类型 (DirectX HLSL)
- 矩阵类型 (DirectX HLSL)
- 着色器类型 (DirectX HLSL)
- 采样器类型 (DirectX HLSL)
- 用户定义的类型 (DirectX HLSL)

着色器可以具有顶级变量、参数和函数。

```
// top-level variable
float globalShaderVariable;

// top-level function
void function(
in float4 position: POSITION0 // top-level argument
              )
{
  float localShaderVariable; // local variable
  function2(...)
}

void function2()
{
  ...
}
```

顶级变量在所有函数之外声明。 顶级参数是顶级函数的参数。 顶级函数是由应用程序 (调用的任何函数，而不是由另一个函数) 调用的函数。

统一着色器输入

顶点和像素着色器接受两种类型的输入数据：不同和统一。 不同的输入是着色器每次执行时唯一的数据。 对于顶点着色器，不同的数据 (例如：位置、法线等) 来自顶点流。 统一数据 (例如：材料颜色、世界转换等) 是着色器多次执行的常量。 对于熟悉程序集着色器模型的模型，统一数据由常量寄存器指定，v 和 t 寄存器会改变数据。

统一数据可由两种方法指定。 最常见的方法是声明全局变量并在着色器中使用它们。 着色器中任何全局变量的使用都会将该变量添加到该着色器所需的统一变量列表中。 第二种方法是将顶级着色器函数的输入参数标记为统一。 此标记指定应将给定变量添加到统一变量列表中。

着色器使用的统一变量通过常量表传回应用程序。 常量表是符号表的名称，用于定义着色器使用的统一变量如何适应常量寄存器。 与全局变量不同，统一函数参数显示在常量表中，前面有美元符号 ($) 。 需要美元符号，以避免同名本地统一输入与全局变量之间的名称冲突。

常量表包含着色器使用的所有统一变量的常量寄存器位置。 如果指定，该表还包含类型信息和默认值。

不同的着色器输入和语义

顶级着色器函数 (不同的输入参数) 必须使用语义或统一关键字来标记，指示该值是执行着色器的常量。 如果未使用语义或统一关键字标记顶级着色器输入，则着色器将无法编译。

输入语义是一个名称，用于将给定输入链接到图形管道上一部分的输出。 例如，顶点着色器使用输入语义 POSITION0 来指定应链接顶点缓冲区中的位置数据的位置。

像素和顶点着色器具有不同的输入语义集，因为馈送到每个着色器单元的图形管道的不同部分。 顶点着色器输入语义描述每个顶点信息 (例如：位置、普通、纹理坐标、颜色、正切、二进制等) 要从顶点缓冲区加载到顶点着色器可以使用的形式。 输入语义直接映射到顶点声明用法和使用情况索引。

像素着色器输入语义描述光栅化单元为每个像素提供的信息。 通过内插当前基元的每个顶点的顶点着色器的输出来生成数据。 基本像素着色器输入语义将输出颜色和纹理坐标信息链接到输入参数。

可以通过两种方法将输入语义分配给着色器输入：

- 将冒号和语义名称追加到参数声明。
- 使用分配给每个结构成员的输入语义定义输入结构。

顶点和像素着色器向后续图形管道阶段提供输出数据。 输出语义用于指定着色器生成的数据应如何链接到下一阶段的输入。 例如，顶点着色器的输出语义用于链接光栅器中内插器的输出，以生成像素着色器的输入数据。 像素着色器输出是提供给每个呈现目标或写入深度缓冲区的 alpha 混合单元的值。

顶点着色器输出语义用于将着色器链接到像素着色器和光栅器阶段。 光栅器使用的顶点着色器，不向像素着色器公开的顶点着色器必须生成最小位置数据。 生成纹理坐标和颜色数据的顶点着色器在内插完成后向像素着色器提供该数据。

像素着色器输出语义将像素着色器的输出颜色与正确的呈现目标绑定。 像素着色器输出颜色链接到 alpha 混合阶段，该阶段决定如何修改目标呈现目标。 像素着色器深度输出可用于更改当前光栅位置的目标深度值。 仅某些着色器模型支持深度输出和多个呈现目标。

输出语义的语法与指定输入语义的语法相同。 语义可以直接在声明为“out”参数的参数上指定，也可以在返回为“out”参数或函数的返回值的结构定义期间分配。

语义标识数据的来源。 语义是用于标识着色器输入和输出的可选标识符。 语义出现在以下三个位置之一：

- 结构成员之后。
- 在函数的输入参数列表中参数之后。
- 函数的输入参数列表之后。

此示例使用结构提供一个或多个顶点着色器输入，另一个结构提供一个或多个顶点着色器输出。 每个结构成员都使用语义。

```
vector vClr;

struct VS_INPUT
{
    float4 vPosition : POSITION;
    float3 vNormal : NORMAL;
    float4 vBlendWeights : BLENDWEIGHT;
};

struct VS_OUTPUT
{
    float4  vPosition : POSITION;
    float4  vDiffuse : COLOR;

};
```

```
float4x4 mWld1;
float4x4 mWld2;
float4x4 mWld3;
float4x4 mWld4;

float Len;
float4 vLight;

float4x4 mTot;

VS_OUTPUT VS_Skinning_Example(const VS_INPUT v, uniform float len=100)
{
    VS_OUTPUT out;

    // Skin position (to world space)
    float3 vPosition =
        mul(v.vPosition, (float4x3) mWld1) * v.vBlendWeights.x +
        mul(v.vPosition, (float4x3) mWld2) * v.vBlendWeights.y +
        mul(v.vPosition, (float4x3) mWld3) * v.vBlendWeights.z +
        mul(v.vPosition, (float4x3) mWld4) * v.vBlendWeights.w;
    // Skin normal (to world space)
    float3 vNormal =
        mul(v.vNormal, (float3x3) mWld1) * v.vBlendWeights.x +
        mul(v.vNormal, (float3x3) mWld2) * v.vBlendWeights.y +
        mul(v.vNormal, (float3x3) mWld3) * v.vBlendWeights.z +
        mul(v.vNormal, (float3x3) mWld4) * v.vBlendWeights.w;

    // Output stuff
    out.vPosition    = mul(float4(vPosition + vNormal * Len, 1), mTot);
    out.vDiffuse  = dot(vLight,vNormal);

    return out;
}
```

输入结构标识顶点缓冲区中的数据，该缓冲区将提供着色器输入。 此着色器将数据从顶点缓冲区的位置、普通元素和混合重量元素映射到顶点着色器寄存器。 输入数据类型不必与顶点声明数据类型完全匹配。 如果未完全匹配，顶点数据将在写入着色器寄存器时自动转换为 HLSL 的数据类型。 例如，如果正常数据被应用程序定义为 UINT 类型，则当着色器读取时，它将转换为 float3。

如果顶点流中的数据包含的组件少于相应的着色器数据类型，则缺少的组件将初始化为 0（，但 w 除外，该组件初始化为 1）。

输入语义类似于 **D3DDECLUSAGE** 中的值。

输出结构标识位置和颜色的顶点着色器输出参数。 这些输出将由管道用于基元处理（的三角形光栅化）。 标记为位置数据的输出表示同质空间中顶点的位置。 至少，顶点着色器必须生成位置数据。 通过在顶点着色器完成（x、y、z）坐标除以 w 来计算屏幕空间位置。 在屏幕空间中，-1 和 1 是视区边界的最小值和最大 x 和 y 值，而 z 用于 z 缓冲区测试。

输出语义也类似于 D3DDECLUSAGE 中的值。 一般情况下，顶点着色器的输出结构也可以用作像素着色器的输入结构，前提是像素着色器不会从带有位置、点大小或雾语义标记的任何变量中读取。 这些语义与像素着色器不使用的每个顶点标量值相关联。 如果像素着色器需要这些值，则可以将其复制到使用像素着色器语义的另一个输出变量中。

全局变量由编译器自动分配给注册。 全局变量也称为统一参数，因为每次调用着色器时处理的所有像素的内容都是相同的。 寄存器包含在常量表中，可以使用 ID3DXConstantTable 接口进行读取。

像素着色器的输入语义将值映射到特定硬件寄存器，以便在顶点着色器和像素着色器之间传输。 每个寄存器类型都有特定的属性。 由于颜色和纹理坐标目前只有两个语义，因此大多数数据通常标记为纹理坐标，即使它不是。

请注意，顶点着色器输出结构使用具有位置数据的输入，该输入不由像素着色器使用。 HLSL 允许顶点着色器的有效输出数据，该着色器不是像素着色器的有效输入数据，前提是它未在像素着色器中引用。

输入参数也可以是数组。 每个数组元素的编译器会自动递增语义。 例如，请考虑以下显式声明：

```
struct VS_OUTPUT
{
    float4 Position   : POSITION;
    float3 Diffuse    : COLOR0;
    float3 Specular   : COLOR1;
    float3 HalfVector : TEXCOORD3;
    float3 Fresnel    : TEXCOORD2;
    float3 Reflection : TEXCOORD0;
    float3 NoiseCoord : TEXCOORD1;
};

float4 Sparkle(VS_OUTPUT In) : COLOR
```

上述显式声明等效于编译器自动递增语义的以下声明：

```
float4 Sparkle(float4 Position : POSITION,
                float3 Col[2] : COLOR0,
                float3 Tex[4] : TEXCOORD0) : COLOR0
{
   // shader statements
   ...
```

与输入语义一样，输出语义标识像素着色器输出数据的数据使用情况。 许多像素着色器只写入一个输出颜色。 像素着色器还可以将深度值写入一个或多个呈现目标，同时 (最多四个) 。 与顶点着色器一样，像素着色器使用结构返回多个输出。 此着色器将 0 写入颜色组件以及深度组件。

```
struct PS_OUTPUT
{
    float4 Color[4] : COLOR0;
    float  Depth  : DEPTH;
};

PS_OUTPUT main(void)
{
    PS_OUTPUT out;

   // Shader statements
   ...

  // Write up to four pixel shader output colors
  out.Color[0] =  ...
  out.Color[1] =  ...
  out.Color[2] =  ...
  out.Color[3] =  ...

  // Write pixel depth
  out.Depth =  ...

    return out;
}
```

像素着色器输出颜色的类型必须为 float4。 编写多个颜色时，必须连续使用所有输出颜色。 换句话说， COLOR1 不能是输出，除非已编写 COLOR0 。 像素着色器深度输出的类型必须为 float1。

采样器和纹理对象

采样器包含采样器状态。 采样器状态指定要采样的纹理，并控制采样期间完成的筛选。 对纹理进行采样需要三件事：

- 纹理
- 具有采样器状态) 的采样器 (
- 采样指令

可以使用纹理和采样器状态初始化采样器，如下所示：

```
sampler s = sampler_state
{
  texture = NULL;
  mipfilter = LINEAR;
};
```

下面是用于采样 2D 纹理的代码示例：

```
texture tex0;
sampler2D s_2D;

float2 sample_2D(float2 tex : TEXCOORD0) : COLOR
{
  return tex2D(s_2D, tex);
}
```

纹理使用纹理变量 tex0 声明。

在此示例中，声明了名为s_2D的采样器变量。 采样器包含大括号内的采样器状态。 这包括要采样的纹理，以及（可选）筛选状态（，即包装模式、筛选模式等）。 如果省略采样器状态，则会应用默认采样器状态，用于指定纹理坐标的线性筛选和包装模式。 采样器函数采用双分量浮点纹理坐标，并返回双分量颜色。 这用 float2 返回类型表示，表示红色和绿色组件中的数据。

(定义四种类型的采样器，请参阅关键字)，纹理查找由内部函数执行：**tex1D (s， t) (DirectX HLSL)** 、**tex2D (s、t) (DirectX HLSL)** 、**tex3D (s、t) (DirectX HLSL)** 、**texCUBE (s、t) (DirectX HLSL)** 。 下面是 3D 采样的示例：

```
texture tex0;
sampler3D s_3D;

float3 sample_3D(float3 tex : TEXCOORD0) : COLOR
{
  return tex3D(s_3D, tex);
}
```

此采样器声明使用筛选器设置和地址模式的默认采样器状态。

下面是相应的多维数据集采样示例：

```
texture tex0;
samplerCUBE s_CUBE;

float3 sample_CUBE(float3 tex : TEXCOORD0) : COLOR
{
  return texCUBE(s_CUBE, tex);
}
```

最后，下面是 1D 采样示例：

```
texture tex0;
sampler1D s_1D;

float sample_1D(float tex : TEXCOORD0) : COLOR
{
  return tex1D(s_1D, tex);
}
```

由于运行时不支持 1D 纹理，因此编译器将使用 2D 纹理，了解 y 坐标不重要。 由于 **tex1D (s，t) (DirectX HLSL)** 作为 2D 纹理查找实现，因此编译器可以高效选择 y 组件。 在某些罕见情况下，编译器不能选择有效的 y 组件，在这种情况下，它将发出警告。

```
texture tex0;
sampler s_1D_float;

float4 main(float texCoords : TEXCOORD) : COLOR
{
    return tex1D(s_1D_float, texCoords);
}
```

此特定示例效率低下，因为编译器必须将输入坐标移到另一个寄存器 (因为 1D 查找作为 2D 查找实现，纹理坐标声明为 float1) 。 如果使用 float2 输入而不是 float1 重写代码，编译器可以使用输入纹理坐标，因为它知道 y 已初始化为某些内容。

```
texture tex0;
sampler s_1D_float2;

float4 main(float2 texCoords : TEXCOORD) : COLOR
{
    return tex1D(s_1D_float2, texCoords);
}
```

所有纹理查找都可以追加“bias”或“proj”（，即 tex2Dbias (DirectX HLSL) 、 texCUBEproj (DirectX HLSL) ）。 使用“proj”后缀时，纹理坐标被 w-component 除以。 使用“偏差”时，mip 级别由 w 组件移动。 因此，所有带后缀的纹理查找始终采用 float4 输入。 tex1D (s、t) (DirectX HLSL) 和 tex2D (s、t) (DirectX HLSL) 分别忽略 yz 和 z 组件。

采样器也可以在数组中使用，尽管目前没有后端支持对采样器的动态数组访问。 因此，以下内容有效，因为它可以在编译时解析：

```
tex2D(s[0],tex)
```

但是，此示例无效。

```
tex2D(s[a],tex)
```

采样器的动态访问主要用于编写具有文本循环的程序。 以下代码演示了采样器数组访问：

```
sampler sm[4];

float4 main(float4 tex[4] : TEXCOORD) : COLOR
{
    float4 retColor = 1;

    for(int i = 0; i < 4;i++)
    {
        retColor *= tex2D(sm[i],tex[i]);
    }

    return retColor;
}
```

> **备注**
>
> 使用 Microsoft Direct3D 调试运行时可帮助捕获纹理中的组件数与采样器之间的不匹配。

编写函数

函数将大型任务分解为较小的任务。 小任务更易于调试，一旦得到证实，就可以重复使用。 函数可用于隐藏其他函数的详细信息，这使得由函数组成的程序更易于遵循。

HLSL 函数以多种方式类似于 C 函数：它们都包含定义和函数正文，它们都声明返回类型和参数列表。 与 C 函数一样，HLSL 验证在着色器编译期间对参数、参数类型和返回值执行类型检查。

与 C 函数不同，HLSL 入口点函数使用语义将函数参数绑定到着色器输入和输出，（调用的 HLSL 函数在内部忽略语义）。 这样，可以轻松地将缓冲区数据绑定到着色器，并将着色器输出绑定到着色器输入。

函数包含声明和正文，声明必须位于正文前面。

```
float4 VertexShader_Tutorial_1(float4 inPos : POSITION ) : POSITION
{
    return mul(inPos, WorldViewProj );
};
```

函数声明包括大括号前面的所有内容：

```
float4 VertexShader_Tutorial_1(float4 inPos : POSITION ) : POSITION
```

函数声明包含：

- 返回类型
- 函数名称
- 参数列表 (可选)
- 输出语义 (可选)
- 注释 (可选)

返回类型可以是任何 HLSL 基本数据类型，例如 float4：

```
float4 VertexShader_Tutorial_1(float4 inPos : POSITION ) : POSITION
{
   ...
}
```

返回类型可以是已定义的结构：

```
struct VS_OUTPUT
{
    float4  vPosition        : POSITION;
    float4  vDiffuse         : COLOR;
};

VS_OUTPUT VertexShader_Tutorial_1(float4 inPos : POSITION )
{
   ...
}
```

如果函数不返回值，则 void 可用作返回类型。

```
void VertexShader_Tutorial_1(float4 inPos : POSITION )
{
   ...
}
```

返回类型始终首先出现在函数声明中。

```
float4 VertexShader_Tutorial_1(float4 inPos : POSITION ) : POSITION
```

参数列表将输入参数声明为函数。 它还可能声明将返回的值。 某些参数既是输入和输出参数。 下面是采用四个输入参数的着色器示例。

```
float4 Light(float3 LightDir : TEXCOORD1,
             uniform float4 LightColor,
             float2 texcrd : TEXCOORD0,
             uniform sampler samp) : COLOR
{
    float3 Normal = tex2D(samp,texcrd);

    return dot((Normal*2 - 1), LightDir)*LightColor;
}
```

此函数返回最终颜色，该颜色是纹理样本和浅色的混合。 该函数采用四个输入。 两个输入具有语义：LightDir 具有 TEXCOORD1 语义，texcrd 具有 TEXCOORD0 语义。 语义意

味着这些变量的数据将来自顶点缓冲区。 尽管 LightDir 变量具有 TEXCOORD1 语义，但参数可能不是纹理坐标。 TEXCOORDn 语义类型通常用于为未预定义的类型提供语义，(光线方向) 没有顶点着色器输入语义。

其他两个输入 LightColor 和 samp 使用 统一 关键字进行标记。 这些是不会在绘图调用之间更改的统一常量。 这些参数的值来自着色器全局变量。

参数可以标记为具有 in 关键字的输入，并使用 out 关键字输出参数。 参数不能通过引用传递;但是，如果使用 inout 关键字声明参数，则参数可以是输入和输出。 传递给用 inout 关键字标记的函数的参数被视为原始函数的副本，直到函数返回，并复制它们。 下面是使用 inout 的示例：

```
void Increment_ByVal(inout float A, inout float B)
{
    A++; B++;
}
```

此函数递增 A 和 B 中的值并返回它们。

函数正文是函数声明后的所有代码。

```
float4 VertexShader_Tutorial_1(float4 inPos : POSITION ) : POSITION
{
    return mul(inPos, WorldViewProj );
};
```

正文由大括号包围的语句组成。 函数体使用变量、文本、表达式和语句实现所有功能。

着色器正文执行两项操作：它执行矩阵乘法并返回 float4 结果。 矩阵乘法是使用 mul (DirectX HLSL) 函数完成的，该函数执行 4x4 矩阵乘法。 **mul (DirectX HLSL)** 称为内部函数，因为它已内置于函数的 HLSL 库中。 下一部分将更详细地介绍内部函数。

矩阵乘法将输入向量 Pos 和复合矩阵 WorldViewProj 组合在一起。 结果是将数据转换为屏幕空间。 这是我们可以做的最小顶点着色器处理。 如果使用固定函数管道而不是顶点着色器，则执行此转换后，可以绘制顶点数据。

函数正文中的最后一个语句是 return 语句。 与 C 一样，此语句将控件从函数返回到调用函数的语句。

函数返回类型可以是 HLSL 中定义的任意简单数据类型，包括 bool、int half、float 和 double。 返回类型可以是复杂数据类型之一，例如矢量和矩阵。 引用对象的 HLSL 类型

不能用作返回类型。 这包括像素阴影器、顶点阴影器、纹理和采样器。

下面是一个函数示例，该函数使用返回类型的结构。

```
float4x4 WorldViewProj : WORLDVIEWPROJ;

struct VS_OUTPUT
{
    float4 Pos  : POSITION;
};

VS_OUTPUT VS_HLL_Example(float4 inPos : POSITION )
{
    VS_OUTPUT Out;

    Out.Pos = mul(inPos,  WorldViewProj );

    return Out;
};
```

float4 返回类型已替换为结构VS_OUTPUT，该结构现在包含单个 float4 成员。

return 语句向函数的末尾发出信号。 这是最简单的 return 语句。 它将控件从函数返回到调用程序。 它不返回任何值。

```
void main()
{
    return ;
}
```

return 语句可以返回一个或多个值。 此示例返回文本值：

```
float main( float input : COLOR0) : COLOR0
{
    return 0;
}
```

此示例返回表达式的标量结果：

```
return  light.enabled;
```

此示例返回从局部变量和文本构造的 float4：

```
return  float4(color.rgb, 1) ;
```

此示例返回一个 float4，该浮点是从内部函数返回的结果构造的，以及几个文本值：

```
float4 func(float2 a: POSITION): COLOR
{
    return float4(sin(length(a) * 100.0) * 0.5 + 0.5, sin(a.y * 50.0), 0,
1);
}
```

此示例返回包含一个或多个成员的结构：

```
float4x4 WorldViewProj;

struct VS_OUTPUT
{
    float4 Pos  : POSITION;
};

VS_OUTPUT VertexShader_Tutorial_1(float4 inPos : POSITION )
{
    VS_OUTPUT out;
    out.Pos = mul(inPos, WorldViewProj );
    return out;
};
```

流控制

大多数最新的顶点和像素着色器硬件旨在逐行运行着色器行，每次执行一次指令。 HLSL 支持流控制，其中包括静态分支、谓词指令、静态循环、动态分支和动态循环。

以前，使用 if 语句会导致程序集语言着色器代码实现 if 端和代码流的另一端。 下面是为 vs_1_1 编译的 HLSL 代码的示例：

```
if (Value > 0)
    oPos = Value1;
```

```
else
    oPos = Value2;
```

下面是生成的程序集代码：

```

// Calculate linear interpolation value in r0.w
mov r1.w, c2.x
slt r0.w, c3.x, r1.w
// Linear interpolation between value1 and value2
mov r7, -c1
add r2, r7, c0
mad oPos, r0.w, r2, c1
```

某些硬件允许静态或动态循环，但大多数硬件需要线性执行。 在不支持循环的模型中，必须取消滚动所有循环。 例如， DepthOfField 示例 示例使用未滚动循环，即使对于ps_1_1着色器也是如此。

HLSL 现在包括对这些类型的流控制的支持：

- static branching
- 谓词说明
- 静态循环
- 动态分支
- 动态循环

静态分支允许基于布尔着色器常量打开或关闭着色器代码块。 这是一种基于当前呈现的对象类型启用或禁用代码路径的便捷方法。 在绘图调用之间，可以决定想要支持当前着色器的功能，然后设置获取该行为所需的布尔标志。 在着色器执行期间，将跳过布尔常量禁用的任何语句。

最熟悉的分支支持是动态分支。 使用动态分支时，比较条件驻留在变量中，这意味着在运行时对每个顶点或每个像素执行比较（，而不是在编译时发生的比较，或者在两个绘图调用之间）。 性能命中是分支的成本，加上分支一侧的说明成本。 动态分支在着色器模型 3 或更高版本中实现。 优化使用这些模型的着色器类似于优化在 CPU 上运行的代码。

相关主题

HLSL 编程指南

在 Direct3D 9 中使用着色器

项目 • 2022/09/22

- 为特定硬件编译着色器
- 初始化着色器常量
- 将着色器参数绑定到特定寄存器
- 呈现可编程着色器
- 调试着色器
- 相关主题

为特定硬件编译着色器

着色器首先添加到 DirectX 8.0 中的 Microsoft DirectX。 当时，定义了多个虚拟着色器计算机，每个计算机大致对应于由前 3D 图形供应商生成的特定图形处理器。 对于每个虚拟着色器计算机，设计了一种程序集语言。 写入着色器模型的程序 (名称vs_1_1和ps_1_1 - ps_1_4) 相对较短，通常由开发人员直接使用适当的汇编语言编写。 应用程序将使用 **D3DXAssembleShader** 将此可读的程序集语言代码传递给 D3DX 库，并返回着色器的二进制表示形式，后者反过来会使用 **CreateVertexShader** 或 **CreatePixelShader** 传递。 有关详细信息，请参阅软件开发工具包 (SDK) 。

Direct3D 9 中的情况类似。 应用程序使用 **D3DXCompileShader** 将 HLSL 着色器传递给 D3DX，并获取编译的着色器的二进制表示形式，后者又使用 **CreatePixelShader** 或 **CreateVertexShader** 传递到 Microsoft Direct3D。 运行时对 HLSL 没有任何了解，只有二进制程序集着色器模型。 这很好，因为它意味着 HLSL 编译器可以独立于 Direct3D 运行时进行更新。 还可以使用 fxc 脱机编译着色器。

除了开发 HLSL 编译器外，Direct3D 9 还引入了程序集级着色器模型来公开最新一代图形硬件的功能。 应用程序开发人员可以在这些新模型 (vs_2_0、vs_3_0、ps_2_0、ps_3_0) 工作，但我们希望大多数开发人员迁移到 HLSL 进行着色器开发。

当然，编写 HLSL 程序来表达特定着色算法的功能不会自动使其在任何给定硬件上运行。 应用程序调用 D3DX 将着色器编译为 **具有 D3DXCompileShader** 的二进制程序集代码。 此入口点的一个限制是一个参数，用于定义哪些程序集级模型 (或编译目标) HLSL 编译器应用于表达最终着色器代码。 如果应用程序在运行时执行 HLSL 着色器编译 (而不是编译时间或脱机) ，则应用程序可以检查 Direct3D 设备的功能并选择要匹配的编译目标。 如果以 HLSL 着色器表示的算法过于复杂，无法在所选编译目标上执行，编译将失败。 这意味着，虽然 HLSL 对着色器开发具有巨大好处，但它不会将开发人员从将游戏传送到具有不同功能的图形设备的目标受众的现实中解脱出来。 作为游戏开发人员，你仍然需要管理视觉对象的分层方法;这意味着为支持更强大的图形卡编写更好的着色器，并为旧卡编写更多基本版本。 然而，由于编写良好的 HLSL，这种负担可以显著缓解。

许多开发人员在应用程序加载时间或首次使用时，而不是在客户计算机上使用 D3DX 编译 HLSL 着色器，而是选择在交付之前将着色器从 HLSL 编译为二进制程序集代码。 这使他们的 HLSL 源代码远离窥探眼睛，并确保其应用程序将运行的所有着色器都经过其内部质量保证过程。 脱机编译着色器的便捷实用工具是 fxc。 此工具有许多选项可用于编译指定编译目标的代码。 如果要优化着色器，或者通常更详细地了解虚拟着色器计算机的功能，那么在开发过程中，研究反汇编输出可能非常具有教育性。 下面汇总了这些选项：

初始化着色器常量

着色器常量包含在常量表中。 这可以通过 ID3DXConstantTable 接口进行访问。 可以在着色器代码中初始化全局着色器变量。 这些通过调用 SetDefaults 在运行时进行初始化。

将着色器参数绑定到特定寄存器

编译器会自动将寄存器分配给全局变量。 编译器会将环境分配给采样器寄存器 s0、SparkleNoise 到采样器寄存器 s1，并将k_s常量寄存器 c0 (假设尚未为以下三个全局变量分配其他采样器或常量寄存器)：

```
sampler Environment;
sampler SparkleNoise;
float4 k_s;
```

还可以将变量绑定到特定寄存器。 若要强制编译器分配给特定寄存器，请使用以下语法：

```
register(RegisterName)
```

其中 RegisterName 是特定寄存器的名称。 以下示例演示了特定的寄存器分配语法，其中采样器环境将绑定到采样器寄存器 s1，SparkleNoise 将绑定到采样器寄存器 s0，k_s将绑定到常量寄存器 c12：

```
sampler Environment : register(s1);
sampler SparkleNoise : register(s0);
float4 k_s : register(c12);
```

呈现可编程着色器

着色器通过设置设备中的当前着色器、初始化着色器常量、告知设备从何处获取不同的输入数据，最后呈现基元来呈现着色器。 可以通过分别调用以下方法来实现上述每个方法：

- SetVertexShader
- SetVertexShaderConstantF
- SetStreamSource
- DrawPrimitive

调试着色器

Microsoft Visual Studio .NET 的 DirectX 扩展在 Visual Studio .NET 集成开发环境 (IDE) 中提供完全集成的 HLSL 调试器。 若要准备着色器调试，必须在计算机上安装正确的工具，（请参阅 Direct3D 9)) Visual Studio (Direct3D 9 中的调试着色器 。

相关主题

HLSL 编程指南

在 Direct3D 10 中使用着色器

项目 • 2022/09/22

管道具有三个着色器阶段，每个阶段都使用 HLSL 着色器编程。 所有 Direct3D 10 着色器都以 HLSL 编写，面向着色器模型 4。

Direct3D 9 和 Direct3D 10 之间的差异：

- 与可以在中间程序集语言中创作的 Direct3D 9 着色器模型不同，着色器模型 4.0 着色器仅在 HLSL 中创作。 仍支持将着色器脱机编译为设备易耗字节码，并推荐用于大多数方案。

此示例仅使用顶点着色器。 由于所有着色器都是从通用着色器核心构建的，因此了解如何使用顶点着色器与使用几何图形或像素着色器非常相似。

编写 HLSL 着色器 (此示例使用顶点着色器 HLSLWithoutFX.vsh) 后，需要为将使用它的特定管道阶段做好准备。 若要执行此操作，需要：

- 编译着色器
- 创建着色器对象
- 设置着色器对象
- 对所有 3 个着色器阶段重复

对于管道中的每个着色器，需要重复这些步骤。

编译着色器

第一步是编译着色器，检查是否已正确编码 HLSL 语句。 这是通过调用 D3D10CompileShader 并为其提供多个参数来完成的，如下所示：

```
    IPD3D10Blob * pBlob;


    // Compile the vertex shader from the file
    D3D10CompileShader( strPath, strlen( strPath ), "HLSLWithoutFX.vsh",
        NULL, NULL, "Ripple", "vs_4_0", dwShaderFlags, &pBlob, NULL );
```

此函数采用以下参数：

- 文件的名称（，名称字符串长度（以字节为单位）），其中包含着色器。 此示例仅使用文件 HLSLWithoutFX.vsh 文件中的顶点着色器（，其中文件扩展名 .vsh 是顶点着

色器的缩写) 。

- 着色器函数名称。 此示例从波纹函数编译顶点着色器，该函数采用单个输入并返回输出结构，（函数来自 HLSLWithoutFX 示例）：

```
VS_OUTPUT Ripple( in float2 vPosition : POSITION )
```

- 指向着色器使用的所有宏的指针。 使用D3D10_SHADER_MACRO来帮助定义宏;只需创建一个名称字符串，其中包含所有宏名称 (，每个名称用空格) 分隔，定义字符串 (，每个宏正文用空格分隔) 。 这两个字符串都需要为 NULL 终止。

- 指向要编译着色器的任何其他文件的指针。 这使用 ID3D10Include 接口，该接口具有两种用户实现的方法：打开和关闭。 若要执行此操作，需要实现 Open 和 Close 方法的正文;在 Open 方法中添加用于打开所需的任何包含文件的代码，在 Close 函数中添加代码以在完成文件时关闭文件。

- 要编译的着色器函数的名称。 此着色器编译波纹函数。

- 编译时要面向的着色器配置文件。 由于可以将函数编译为顶点、几何图形或像素着色器，因此配置文件会告知编译器哪种类型的着色器以及要比较代码的着色器模型。

- 着色器编译器标志。 这些标志告知编译器要将哪些信息放入编译的输出中，以及希望输出代码优化的方式：用于速度、调试等。有关可用标志的列表，请参阅 Direct3D 10) (效果常量 。 此示例包含一些代码，可用于设置项目的编译器标志值 - 这主要是是否要生成调试信息的问题。

- 指向包含已编译着色器代码的缓冲区的指针。 缓冲区还包含编译器标志请求的任何嵌入式调试和符号表信息。

- 指向包含编译期间遇到的错误和警告列表的缓冲区的指针，如果在编译着色器时运行调试器，则在调试输出中看到的相同消息。 如果不希望错误返回到缓冲区，**则** NULL 是可接受的值。

如果着色器成功编译，则返回指向着色器代码的指针作为 ID3D10Blob 接口。 它被称为 Blob 接口，因为指针是内存中由 DWORD 数组构成的位置。 提供了接口，以便你可以获取指向下一步中所需的已编译着色器的指针。

从 2006 年 12 月 SDK 开始，DirectX 10 HLSL 编译器现在是 DirectX 9 和 DirectX 10 中的默认编译器。 有关详细信息，请参阅 Effect-Compiler 工具 。

获取指向已编译着色器的指针

多个 API 方法需要指向已编译着色器的指针。 此参数通常称为 *pShaderBytecode* ，因为它指向表示为字节代码序列的已编译着色器。 若要获取指向已编译着色器的指针，请先通过调用 D3D10CompileShader 或类似函数来编译着色器。 如果编译成功，则会在 ID3D10Blob 接口中返回已编译的着色器。 最后，使用 GetBufferPointer 方法返回指针。

创建着色器对象

编译着色器后，调用 CreateVertexShader 创建着色器对象：

```
ID3D10VertexShader ** ppVertexShader
ID3D10Blob pBlob;


// Create the vertex shader
hr = pd3dDevice->CreateVertexShader( (DWORD*)pBlob->GetBufferPointer(),
    pBlob->GetBufferSize(), &ppVertexShader );

// Release the pointer to the compiled shader once you are done with it
pBlob->Release();
```

若要创建着色器对象，请将指针传递给编译的着色器到 CreateVertexShader。 由于必须首先成功编译着色器，因此此调用几乎肯定会通过，除非计算机上存在内存问题。

可以根据需要创建任意数量的着色器对象，只需保留指向它们的指针。 这一机制适用于几何图形和像素着色器，假设在调用 create 方法) 时调用编译方法) 接口名称 (时 (着色器配置文件 (。

设置着色器对象

最后一步是将着色器设置为管道阶段。 由于管道中有三个着色器阶段，因此需要对每个阶段进行三个 API 调用。

```
// Set a vertex shader
pd3dDevice->VSSetShader( g_pVS10 );
```

对 VSSetShader 的调用将指针指向步骤 1 中创建的顶点着色器。 这会在设备中设置着色器。 顶点着色器阶段现在使用其顶点着色器代码进行初始化，所有这些代码都是初始化任何着色器变量。

对所有 3 个着色器阶段重复

重复上述一组步骤，生成任何顶点或像素着色器，甚至生成输出到像素着色器的几何图形着色器。

相关主题

编译着色器

相关主题

HLSL 编程指南

优化 HLSL 着色器

项目 • 2022/09/24

本部分介绍可用于优化着色器的常规用途策略。 可以将这些策略应用于在任何平台上以任何语言编写的着色器。

- 了解执行着色器计算的位置
- 跳过不必要的说明
- 包变量和内插
- 减少着色器复杂性
- 相关主题
- 相关主题

了解执行着色器计算的位置

顶点着色器执行包括提取顶点和执行顶点数据的矩阵转换的操作。 通常，每个顶点执行一次顶点着色器。

像素着色器执行包括提取纹理数据和执行照明计算的操作。 通常，对于给定的几何图形，每个像素执行一次像素着色器。

通常，像素数超过场景中的顶点数，因此像素着色器执行的频率高于顶点着色器。

设计着色器算法时，请记住以下事项：

- 如果可能，请在顶点着色器上执行计算。 对像素着色器执行的计算比在顶点着色器上执行的计算要贵得多。
- 请考虑使用每顶点计算来提高密集网格等情况下的性能。 对于密集网格，每个顶点计算可能会生成与使用每像素计算生成的结果直观不区分的结果。

跳过不必要的说明

在 HLSL 中，动态分支提供限制执行指令数的功能。 因此，动态分支可以帮助加快着色器执行时间。 如果未显示几何图形或像素，请使用动态分支退出着色器或限制指令。 例如，如果像素未亮起，则执行照明算法时没有点。

下表列出了某些情况，你可以在着色器中测试条件，并使用动态分支来跳过不必要的说明。 表不全面。 相反，它旨在为你提供优化代码的想法。

要检查的条件	着色器中的响应

要检查的条件	着色器中的响应
Alpha 检查确定不会看到像素。	跳过着色器的其余部分。
像素或几何图形完全迷雾。	跳过着色器的其余部分。
皮肤重量为零。	跳过骨骼。
光衰减为零。	跳过照明。
非正兰伯田术语。	跳过照明。

包变量和内插

请注意着色器数据所需的空间。 尽可能将信息打包到变量或内插中。 有时，两个变量中的信息可以打包到单个变量的内存空间中。

减少着色器复杂性

保持着色器较小且简单。 通常，使用较少指令的着色器比具有更多指令的着色器更快地执行。 调试和优化更小、更复杂的着色器也更容易。

相关主题

HLSL 编程指南

相关主题

HLSL 编程指南

在 Visual Studio 中调试着色器

项目 • 2022/09/23

用于调试着色器的最新工具现在作为Microsoft Visual Studio中的一项功能提供，称为Visual Studio图形调试器。 此新工具是 PIX for Windows 工具的替代工具。 Visual Studio图形调试器大大提高了可用性、对 Windows 8 和 Direct3D 11.1 的支持，并与传统的Visual Studio功能（如调用堆栈和 HLSL 调试窗口）集成。 有关此新功能的详细信息，请参阅 调试 DirectX 图形。

相关主题

HLSL 编程指南

编译着色器

项目 • 2022/09/24

> **备注**
>
> 本主题介绍 `FXC.EXE` 用于着色器模型 2 到 5.1 的编译器。 对于着色器模型 6，请改用 `DXC.EXE` ，如 **使用 dxc.exe 和 dxcompiler.dll** 中所述。 为 HLSL 属性页配置选择着色器模型 6 时，Visual Studio将自动使用`DXC.EXE`。

Microsoft Visual Studio可以从 C++ 项目中包括的 *.hlsl 和 *.fx 文件编译着色器代码。

作为生成过程的一部分，Visual Studio使用 fxc.exe 或 dxc.exe HLSL 代码编译器将 HLSL 着色器源文件编译为二进制着色器对象文件或头文件中定义的字节数组。 HLSL 代码编译器如何编译项目中的每个着色器源文件取决于如何为该文件指定 **Ouput Files** 属性。 有关 HLSL 属性页的详细信息，请参阅 HLSL 属性页。

使用的编译方法通常取决于 HLSL 着色器源文件的大小。 如果在标头中包含大量字节代码，则增加应用的大小和初始加载时间。 即使创建着色器，也强制所有字节代码驻留在内存中，这会浪费资源。 但是，在标头中包含字节代码时，可以减少代码复杂性并简化着色器创建。

现在，我们来了解各种方法来编译着色器代码和着色器代码的文件扩展名约定。

- 使用着色器代码文件扩展名
- 在生成时编译到对象文件
- 在生成时编译到头文件
- 使用 D3DCompileFromFile 进行编译
- 相关主题
- 相关主题

使用着色器代码文件扩展名

若要符合 Microsoft 约定，请将以下文件扩展名用于着色器代码：

- 扩展名为 .hlsl 的文件保留高级着色语言 (HLSL) 源代码。 也支持较旧的 .fx 扩展，但通常与 旧版效果系统相关联。
- 扩展名为 .cso 的文件保留经过编译的着色器对象。
- 扩展名为 .h 的文件是头文件，但在着色器代码上下文中，此头文件定义保留着色器数据的字节数组。 HLSL 着色器代码标头的其他常见扩展包括 .hlsli 和 .fxh。

在生成时编译到对象文件

如果将 .hlsl 文件编译为二进制着色器对象文件，则应用需要从这些对象文件读取数据，(.cso 是这些对象文件的默认扩展名)，将数据分配给字节数组，并从这些字节数组创建着色器对象。 例如，若要创建顶点着色器 (ID3D11VertexShader**)，请使用包含已编译顶点着色器字节代码的字节数组调用 ID3D11Device：：CreateVertexShader 方法。 在此示例代码中，SimpleVertexShader.hlsl 文件的 **Ouput Files** 属性指定要编译为 SimpleVertexShader.cso 对象文件。

C++

```
        auto vertexShaderBytecode = ReadData("SimpleVertexShader.cso");
        ComPtr<ID3D11VertexShader> vertexShader;
        DX::ThrowIfFailed(
            m_d3dDevice->CreateVertexShader(
                vertexShaderBytecode->Data,
                vertexShaderBytecode->Length,
                nullptr,
                &vertexShader
                )
```

> 此处的 ReadData 帮助程序将查找当前工作目录，以及与当前进程 EXE 文件相同的目录，与 .cso 文件通常与其他 VS 生成产品一起找到。 有关示例实现，请参阅 ReadData.h 。

在生成时编译到头文件

如果将 .hlsl 文件编译为标头文件中定义的字节数组，则需要在代码中包含这些头文件。在此示例代码中，PixelShader.hlsl 文件的 **Ouput Files** 属性指定要编译为 PixelShader.h 头文件中定义的 *g_psshader* 字节数组。

C++

```
namespace
{
       include "PixelShader.h"
}
...
        ComPtr<ID3D11PixelShader> m_pPixelShader;
        hr = pDevice->CreatePixelShader(g_psshader, sizeof(g_psshader),
nullptr, &m_pPixelShader);
```

使用 D3DCompileFromFile 进行编译

还可以在运行时使用 **D3DCompileFromFile 函数编译** Direct3D 11 的着色器代码。 有关如何执行此操作的详细信息，请参阅 如何：编译着色器。

> **备注**
>
> Windows Microsoft Store应用支持使用 **D3DCompileFromFile** 进行开发，但不支持部署。

“相关主题”

HLSL 编程指南

相关主题

HLSL 编程指南

指定编译器目标

项目 • 2022/09/24

调用 D3DCompile、D3DCompile2 或 D3DCompileFromFile 函数时，需要指定着色器目标（着色器功能集）进行编译。 下面列出了 D3DCompile* 函数和 HLSL 编译器支持的各种配置文件的目标。

- Direct3D 11.0 和 11.1 功能级别
- Direct3D 10.1 功能级别
- Direct3D 10.0 功能级别
- Direct3D 9.1、9.2 和 9.3 功能级别
- 旧版 Direct3D 9 着色器模型 3.0
- 旧版 Direct3D 9 着色器模型 2.0
- 旧版 Direct3D 9 着色器模型 1.x
- 旧效果
- 备注
- 相关主题

Direct3D 11.0 和 11.1 功能级别

下面是 Direct3D 11.0 和 11.1 功能级别 支持的着色器目标。

目标	描述
cs_5_0	DirectCompute 5.0 (计算着色器)
ds_5_0	域着色器
gs_5_0	几何着色器
hs_5_0	外壳着色器
ps_5_0	像素着色器
vs_5_0	顶点着色器

Direct3D 10.1 功能级别

下面是 Direct3D 10.1 功能级别 支持的着色器目标。

目标	描述
cs_4_1	DirectCompute 4.1 (计算着色器) $ $ $
gs_4_1	几何着色器
ps_4_1	像素着色器
vs_4_1	顶点着色器

Direct3D 10.0 功能级别

下面是 Direct3D 10.0 功能级别 支持的着色器目标。

目标	描述
cs_4_0	DirectCompute 4.0 (计算着色器) —
gs_4_0	几何着色器
ps_4_0	像素着色器
vs_4_0	顶点着色器

Direct3D 9.1、9.2 和 9.3 功能级别

下面是 Direct3D 9.1、9.2 和 9.3 功能级别 支持的着色器目标。

> **备注**
>
> 使用 *_4_0_level_9_x HLSL 着色器配置文件时，隐式使用 **着色器模型** 2.x 配置文件来支持支持 Direct3D 9 的硬件。 着色器模型 2.x 配置文件支持比 **着色器模型** 4.x 和更高版本的配置文件更有限的流控制行为。

目标	描述

目标	描述
ps_4_0_level_9_1	9.1 和 9.2 像素着色器 (与ps_2_0) 类似的限制 • 64 算术和 32 个纹理指令 • 12 个临时寄存器 • 4 个从属读取级别
ps_4_0_level_9_3	9.3 像素着色器 (与 ps_2_x 一 类似的限制，) • 512 说明 • 32 个临时寄存器 • 静态流控制 (最大深度为 4) • 动态流控制 (最大深度为 24) • D3DPS20CAPS_ARBITRARYSWIZZLE • D3DPS20CAPS_GRADIENTINSTRUCTIONS • D3DPS20CAPS_PREDICATION • D3DPS20CAPS_NODEPENDENTREADLIMIT • D3DPS20CAPS_NOTEXINSTRUCTIONLIMIT
vs_4_0_level_9_1	9.1 和 9.2 的顶点着色器 (类似于vs_2_0) • 256 条说明 • 12 个临时寄存器 • 静态流控制 (最大深度为 1)
vs_4_0_level_9_3	9.3 (的顶点着色器类似于 vs_2_a 一，具有其他着色器功能和实例化) • 256 条说明 • 32 个临时寄存器 • 静态流控制 (最大深度为 4) • D3DVS20CAPS_PREDICATION

旧版 Direct3D 9 着色器模型 3.0

下面是旧版 Direct3D 9 着色器模型 3.0 马达加斯加的着色器目标。

目标	描述
ps_3_0	像素着色器 3.0
ps_3_sw	像素着色器 3.0 (软件)
vs_3_0	顶点着色器 3.0

目标	描述
vs_3_sw	顶点着色器 3.0 (软件)

旧版 Direct3D 9 着色器模型 2.0

下面是旧版 Direct3D 9 着色器模型 2.0 马达加斯加的着色器目标。

目标	描述
ps_2_0	像素着色器 2.0
ps_2_a	像素着色器 2a
ps_2_b	像素着色器 2b
ps_2_sw	像素着色器 2.0 软件
vs_2_0	顶点着色器 2.0
vs_2_a	顶点着色器 2a
vs_2_sw	顶点着色器 2.0 软件

旧版 Direct3D 9 着色器模型 1.x

下面是旧版 Direct3D 9 着色器模型 1.x[4] 的着色器目标。

目标	描述
tx_1_0	旧版 D3DX9[5] 函数 **D3DXCreateTextureShader** 和 **D3DXFillTextureTX** 使用的纹理着色器配置文件
vs_1_1	顶点着色器 1.1

旧效果

下面是旧效果的效果目标。

目标	描述
fx_2_0	效果 (D3DX9 中 Direct3D 9 的 FX) [5]
fx_4_0	D3DX10[5] 中 Direct3D 10.0 的效果 (FX)
fx_4_1	D3DX10[5] 中 Direct3D 10.1 的效果 (FX)
fx_5_0	Direct3D 11[5] 的 FX) (效果

说明

下面是上述部分引用的一些说明：

1. 功能级别 10.0 和 10.1 设备可以选择支持 DirectCompute。 若要验证支持，请使用 **ID3D11Device：：CheckFeatureSupport** 和 **D3D11_FEATURE_D3D10_X_HARDWARE_OPTIONS**。
2. 功能级别 9.3 实际上需要符合 旧版 Direct3D 9 着色器模型 3.0 的要求的硬件，但此功能级别不使用vs_3_0或ps_3_0目标。
3. 仅对 Direct3D 9 API 使用旧版 Direct3D 9 着色器模型。 请改用 Direct3D 10.x 和 11.x API 的 9.x 配置文件。
4. 当前的 HLSL 着色器 **D3DCompile*** 函数不支持旧版 1.x 像素着色器。 2006 年 10 月版 DirectX SDK 中，支持这些目标的 HLSL 的最后一个版本是 D3DX9。
5. D3DX 和 DirectX SDK 的所有版本都已弃用。 有关详细信息，请参阅 DirectX SDK 在哪里？

相关主题

HLSL 编程指南

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

项目 • 2022/09/22

D3DX_DXGIFormatConvert.inl 文件包含可在 Direct3D 11 硬件上的计算着色器或像素着色器中使用的内联格式转换函数。 可以在应用程序中使用这些函数同时读取和写入纹理。 也就是说，可以执行就地图像编辑。 若要使用这些内联格式转换函数，请在应用程序中包括 D3DX_DXGIFormatConvert.inl 文件。

> D3DX_DXGIFormatConvert.inl 标头随附在旧版 DirectX SDK 中。 它还包含在 Microsoft.DXSDK.D3DX NuGet包中。

Direct3D 11 的无序访问视图 (UAV) Texture1D、Texture2D 或 Texture3D 资源支持从计算着色器或像素着色器随机访问读取和写入内存。 但是，Direct3D 11 支持同时读取和写入到DXGI_FORMAT_R32_UINT纹理格式。 例如，Direct3D 11 不支持同时读取和写入其他更有用的格式，例如DXGI_FORMAT_R8G8B8A8_UNORM。 只能使用 UAV 来随机访问写入此类其他格式，或者只能使用着色器资源视图 (SRV) 来随机访问从此类其他格式读取。 格式转换硬件不能同时读取和写入此类其他格式。

但是，只要资源的原始格式支持强制转换为DXGI_FORMAT_R32_UINT，你仍然可以同时读取和写入此类其他格式，方法是在创建 UAV 时将纹理强制转换为DXGI_FORMAT_R32_UINT纹理格式。 每个元素的大多数 32 位格式都支持转换为DXGI_FORMAT_R32_UINT。 通过在创建 UAV 时将纹理强制转换为DXGI_FORMAT_R32_UINT纹理格式，那么只要着色器在读取和写入时对读取和打包执行手动格式打包，就可以同时执行对纹理的读取和写入操作。

将纹理强制转换为DXGI_FORMAT_R32_UINT纹理格式的好处是，稍后可以使用适当的格式 (，例如，在同一纹理上与其他视图DXGI_FORMAT_R16G16_FLOAT) ，例如呈现目标视图 (RTV) 或 SRV。 因此，硬件可以执行典型的自动格式解压缩和打包，可以执行纹理筛选，等等，其中没有硬件限制。

以下方案要求应用程序执行以下操作序列来执行就地图像编辑。

假设要创建一个纹理，你可以使用像素着色器或计算着色器执行就地编辑，并且希望纹理数据以以下 TYPELESS 格式之一的后代的形式存储：

- DXGI_FORMAT_R10G10B10A2_TYPELESS
- DXGI_FORMAT_R8G8B8A8_TYPELESS
- DXGI_FORMAT_B8G8R8A8_TYPELESS
- DXGI_FORMAT_B8G8R8X8_TYPELESS
- DXGI_FORMAT_R16G16_TYPELESS

例如，DXGI_FORMAT_R10G10B10A2_UNORM格式是DXGI_FORMAT_R10G10B10A2_TYPELESS格式的后代。 因此，DXGI_FORMAT_R10G10B10A2_UNORM支持按以下顺序描述的使用模式。 从DXGI_FORMAT_R32_TYPELESS（如DXGI_FORMAT_R32_FLOAT）降序的格式是微不足道的，无需以下序列中所述的任何格式转换帮助。

执行就地图像编辑

1. 创建一个纹理，其中包含在上一方案中指定的相应 TYPELESS 依赖格式以及所需的绑定标志，例如D3D11_BIND_UNORDERED_ACCESS |D3D11_BIND_SHADER_RESOURCE。

2. 若要进行就地图像编辑，请使用DXGI_FORMAT_R32_UINT格式创建 UAV。 Direct3D 11 API 通常不允许在不同的格式"系列"之间强制转换。但是，Direct3D 11 API 会以DXGI_FORMAT_R32_UINT格式发出异常。

3. 在计算着色器或像素着色器中，使用D3DX_DXGIFormatConvert.inl 文件中提供的相应内联格式包和解压缩函数。 例如，假设纹理的 DXGI_FORMAT_R32_UINT UAV 确实包含DXGI_FORMAT_R10G10B10A2_UNORM格式的数据。 应用程序将 UAV 中的 uint 读取到着色器后，它必须调用以下函数来解压缩纹理格式：

   ```
   XMFLOAT4 D3DX_R10G10B10A2_UNORM_to_FLOAT4(UINT packedInput)
   ```

   然后，若要在同一着色器中写入 UAV，应用程序调用以下函数将着色器数据打包到应用程序可以写入 UAV 的 uint 中：

   ```
   UINT D3DX_FLOAT4_to_R10G10B10A2_UNORM(hlsl_precise XMFLOAT4
   unpackedInput)
   ```

4. 然后，应用程序可以创建具有所需格式的其他视图，例如 SRV。 例如，如果资源创建为 DXGI_FORMAT_R10G10B10A2_TYPELESS，则应用程序可以创建DXGI_FORMAT_R10G10B10A2_UNORM格式的 SRV。 当着色器访问该 SRV 时，硬件可以像往常一样执行自动类型转换。

> ⓘ **备注**
>
> 如果着色器必须仅写入 UAV 或读取为 SRV，则无需执行此转换工作，因为可以使用完全类型化的 UAV 或 SRV。 只有在你想要执行纹理的 UAV 同时读取和写入时，D3DX_DXGIFormatConvert.inl 中提供的格式转换函数才可能很有用。

下面是 D3DX_DXGIFormatConvert.inl 文件中包含的格式转换函数的列表。 这些函数按解压缩和打包的DXGI_FORMAT进行分类。 每个受支持的格式都从上述方案中列出的 TYPELESS 格式之一下降，并支持转换为 UAV DXGI_FORMAT_R32_UINT。

DXGI_FORMAT_R10G10B10A2_UNORM

```
XMFLOAT4 D3DX_R10G10B10A2_UNORM_to_FLOAT4(UINT packedInput)
UINT     D3DX_FLOAT4_to_R10G10B10A2_UNORM(hlsl_precise XMFLOAT4
unpackedInput)
```

DXGI_FORMAT_R10G10B10A2_UINT

```
XMUINT4 D3DX_R10G10B10A2_UINT_to_UINT4(UINT packedInput)
UINT    D3DX_UINT4_to_R10G10B10A2_UINT(XMUINT4 unpackedInput)
```

DXGI_FORMAT_R8G8B8A8_UNORM

```
XMFLOAT4 D3DX_R8G8B8A8_UNORM_to_FLOAT4(UINT packedInput)
UINT     D3DX_FLOAT4_to_R8G8B8A8_UNORM(hlsl_precise XMFLOAT4 unpackedInput)
```

DXGI_FORMAT_R8G8B8A8_UNORM_SRGB

```
XMFLOAT4 D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4_inexact(UINT packedInput) *
XMFLOAT4 D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4(UINT packedInput)
UINT     D3DX_FLOAT4_to_R8G8B8A8_UNORM_SRGB(hlsl_precise XMFLOAT4
unpackedInput)
```

ⓘ 备注

_inexact 类型函数使用没有足够精度的着色器指令来给出确切答案。 备用函数使用存储在着色器中的查阅表来提供精确的 SRGB 浮>点转换。

DXGI_FORMAT_R8G8B8A8_UINT

```
XMUINT4 D3DX_R8G8B8A8_UINT_to_UINT4(UINT packedInput)
XMUINT  D3DX_UINT4_to_R8G8B8A8_UINT(XMUINT4 unpackedInput)
```

DXGI_FORMAT_R8G8B8A8_SNORM

```
XMFLOAT4 D3DX_R8G8B8A8_SNORM_to_FLOAT4(UINT packedInput)
UINT     D3DX_FLOAT4_to_R8G8B8A8_SNORM(hlsl_precise XMFLOAT4 unpackedInput)
```

DXGI_FORMAT_R8G8B8A8_SINT

```
XMINT4 D3DX_R8G8B8A8_SINT_to_INT4(UINT packedInput)
UINT   D3DX_INT4_to_R8G8B8A8_SINT(XMINT4 unpackedInput)
```

DXGI_FORMAT_B8G8R8A8_UNORM

```
XMFLOAT4 D3DX_B8G8R8A8_UNORM_to_FLOAT4(UINT packedInput)
UINT     D3DX_FLOAT4_to_B8G8R8A8_UNORM(hlsl_precise XMFLOAT4 unpackedInput)
```

DXGI_FORMAT_B8G8R8A8_UNORM_SRGB

```
XMFLOAT4 D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4_inexact(UINT packedInput) *
XMFLOAT4 D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4(UINT packedInput)
UINT     D3DX_FLOAT4_to_R8G8B8A8_UNORM_SRGB(hlsl_precise XMFLOAT4
unpackedInput)
```

ⓘ 备注

_inexact 类型函数使用没有足够精度的着色器指令来给出确切答案。 备用函数使用存储在着色器中的查阅表来提供精确的 SRGB 浮>点转换。

DXGI_FORMAT_B8G8R8X8_UNORM

```
XMFLOAT3 D3DX_B8G8R8X8_UNORM_to_FLOAT3(UINT packedInput)
UINT     D3DX_FLOAT3_to_B8G8R8X8_UNORM(hlsl_precise XMFLOAT3 unpackedInput)
```

DXGI_FORMAT_B8G8R8X8_UNORM_SRGB

```
XMFLOAT3 D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3_inexact(UINT packedInput) *
XMFLOAT3 D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3(UINT packedInput)
UINT     D3DX_FLOAT3_to_B8G8R8X8_UNORM_SRGB(hlsl_precise XMFLOAT3
unpackedInput)
```

ⓘ 备注

_inexact 类型函数使用没有足够精度的着色器指令来给出确切答案。 备用函数使用存储在着色器中的查阅表来提供精确的 SRGB 浮>点转换。

DXGI_FORMAT_R16G16_FLOAT

```
XMFLOAT2 D3DX_R16G16_FLOAT_to_FLOAT2(UINT packedInput)
UINT     D3DX_FLOAT2_to_R16G16_FLOAT(hlsl_precise XMFLOAT2 unpackedInput)
```

DXGI_FORMAT_R16G16_UNORM

```
XMFLOAT2 D3DX_R16G16_UNORM_to_FLOAT2(UINT packedInput)
UINT     D3DX_FLOAT2_to_R16G16_UNORM(hlsl_precise FLOAT2 unpackedInput)
```

DXGI_FORMAT_R16G16_UINT

```
XMUINT2 D3DX_R16G16_UINT_to_UINT2(UINT packedInput)
UINT    D3DX_UINT2_to_R16G16_UINT(XMUINT2 unpackedInput)
```

DXGI_FORMAT_R16G16_SNORM

```
XMFLOAT2 D3DX_R16G16_SNORM_to_FLOAT2(UINT packedInput)
UINT     D3DX_FLOAT2_to_R16G16_SNORM(hlsl_precise XMFLOAT2 unpackedInput)
```

DXGI_FORMAT_R16G16_SINT

```
XMINT2 D3DX_R16G16_SINT_to_INT2(UINT packedInput)
UINT   D3DX_INT2_to_R16G16_SINT(XMINT2 unpackedInput)
```

相关主题

HLSL 编程指南

相关主题

HLSL 编程指南

使用 HLSL 最小精度

项目 • 2022/09/23

从Windows 8开始，图形驱动程序可以使用大于或等于指定位精度的任何精度实现最小精度 HLSL 标量数据类型。 在实现 HLSL 最小值精度的硬件上使用 HLSL 最小精度着色器代码时，使用内存带宽更少，因此也使用更少的系统电源。

可以通过使用 **D3D11_FEATURE_SHADER_MIN_PRECISION_SUPPORT** 值调用 **ID3D11Device：：CheckFeatureSupport** 来查询图形驱动程序提供的最小精度支持。 有关详细信息，请参阅 HLSL 最低精度支持。

- 使用最小精度数据类型声明变量
- 测试最小精度着色器代码
- 相关主题

使用最小精度数据类型声明变量

若要在 HLSL 着色器代码中使用最小精度，请将单个变量声明为矢量) 、**min16int**、**min10float** 等类型的单个变量，例如 **min16float** (**min16float4** 等。 使用这些变量时，着色器代码指示它不需要比变量所指示的精度更高。 但硬件可以忽略最小精度指示器，并完全以 32 位精度运行。 当着色器代码用于利用最小精度的硬件上时，使用更少的内存带宽，因此，只要着色器代码不需要比指定的精度更高的系统电源，你也会使用更少的系统电源。

无需创作多个着色器，且不使用最小精度。 相反，如果图形驱动程序报告它不支持任何最小精度，则创建具有最小精度的着色器，并且最小精度变量的行为完全为 32 位精度。 HLSL 最小精度着色器在早于Windows 8的操作系统上不起作用，因此，如果你打算面向早期操作系统，则需要创作多个着色器，有些着色器不使用最小精度。

> **备注**
>
> 不要在着色器中的不同精度级别之间进行数据切换，因为这些类型的转换浪费并降低性能。 例外情况是着色器常量仍然始终为 32 位，但供应商可以设计图形硬件，这些硬件可以自由地向下转换为 HLSL 指令读取可能使用的任何较低精度。

通过使用最小精度，可以控制着色器代码的各个部分的计算精度。

HLSL 最小精度的规则类似于 C/C++，其中表达式中的类型确定操作的精度，而不是最终写入的类型。

测试最小精度着色器代码

参考光栅器 (**D3D_DRIVER_TYPE_REFERENCE**) 让你大致了解 HLSL 着色器代码中最小精度的行为方式，方法是将每个 HLSL 指令量化为指定的精度。 这有助于发现可能意外依赖于最低精度的代码。 当 HLSL 着色器代码使用最小精度时，引用光栅器不会更快地运行，但你可以使用它来验证代码的正确性。 WARP (**D3D_DRIVER_TYPE_WARP**) 不支持在 HLSL 着色器代码中使用最小精度;WARP 只需以完整的 32 位精度运行。

相关主题

HLSL 编程指南

功能级别 9 硬件上的用户剪辑平面

项目 • 2022/09/22

从Windows 8开始，Microsoft 高级着色器语言 (HLSL) 支持可用于 Microsoft Direct3D 11 API 的语法，以指定功能级别9_x及更高级别的用户剪辑平面。 可以使用此剪辑平面语法编写着色器，然后将该着色器对象与 Direct3D 11 API 一起使用，以在所有 Direct3D 功能级别上运行。

- 背景
- 语法
- 在功能级别 9 及更高级别的剪辑空间中创建剪辑平面
  - 背景阅读
  - 10Level9 功能级别
  - 剪裁平面数学
  - 在视图空间中剪裁
  - 投影矩阵
  - 剪辑空间剪辑平面
- 相关主题

背景

可以通过 **IDirect3DDDevice9：：SetClipPlane** 和 **IDirect3DDevice9：：GetClipPlane 方法访问 Microsoft Direct3D** 9 API 中的用户剪辑平面。 在 Microsoft Direct3D 10 及更高版本中，可以通过 SV_ClipDistance 语义访问用户剪辑平面。 但在Windows 8之前，SV_ClipDistance不适用于 Direct3D 10 或 Direct3D 11 API 中的功能级别9_x硬件。 因此，在Windows 8之前，使用功能级别9_x硬件访问用户剪辑平面的唯一方法是通过 Direct3D 9 API。 Direct3D Windows Microsoft Store应用无法使用 Direct3D 9 API。 下面介绍可用于在功能级别9_x及更高版本上通过 Direct3D 11 API 访问用户剪辑平面的语法。

应用使用剪辑平面在 3D 世界中定义一组不可见平面，这些平面剪辑 (丢弃) 所有绘制基元。 Windows不会绘制任何剪辑平面负端的任何像素。 然后，应用可以使用剪裁平面来呈现平面反射。

语法

使用此语法可将剪辑平面声明为 函数声明中的函数属性。 例如，此处我们在顶点着色器片段上使用语法：

```
syntax
```

```
cbuffer ClipPlaneConstantBuffer
{
       float4 clipPlane1;
       float4 clipPlane2;
};

[clipplanes(clipPlane1,clipPlane2)]
VertexShaderOutput main(VertexShaderInput input)
{
       // the rest of the vertex shader doesn't refer to the clip plane

       …

       return output;
}
```

对于顶点着色器片段，此示例表示两个剪辑平面。 它显示，需要在顶点着色器的返回值之前立即将新的 **剪贴板** 属性放在方括号内。 在 **剪辑平面** 属性后面的括号内，提供最多 6 个 **float4** 常量的列表，用于定义每个活动剪辑平面的平面系数。 该示例还表明，需要使每个平面的系数驻留在常量缓冲区中。

> **备注**
>
> 没有可用于动态禁用剪辑平面的语法。 必须重新编译没有 **剪辑平面** 属性的其他相同着色器，或者应用可以将常量缓冲区中的系数设置为零，以便平面不再影响任何几何图形。

此语法适用于任何 4.0 或更高版本的顶点着色器目标，其中包括vs_4_0_level_9_1和vs_4_0_level_9_3。

在功能级别 9 及更高级别的剪辑空间中创建剪辑平面

下面介绍如何在 功能级别 9_x及更高版本上在剪辑空间中创建剪辑平面。

背景阅读

Frank D. Luna 的“DirectX 10 3D 游戏编程简介”介绍了所需的图形数学背景 (章节 1、2 和 3) ，以及顶点着色器 (第 5.6 和 5.8 节中发生的各种空格和空间转换) 。

10Level9 功能级别

在 Direct3D 10 及更高版本中，可以在任何有意义的空间中剪辑，通常在世界空间或查看空间中。 但 Direct3D 9 使用剪辑空间，这是预先透视划分投影空间。 当顶点着色器将它们传递到 图形管道中的阶段时，矢量处于剪辑空间中。

编写Windows Microsoft Store应用时，必须使用 10Level9 功能级别 (功能级别9_x) ，以便应用可以在功能级别9_x和更高硬件上运行。 由于应用支持功能级别9_x及更高，因此还必须使用在剪辑空间中应用剪辑平面的常见功能。

使用 vs_4_0_level_9_1 或更高版本编译顶点着色器时，该顶点着色器可以使用 **剪辑平面** 属性。 Direct3D 10 或更高版本的对象具有所发出的顶点积，其中包含属性中指定的每个 **float4** 全局常量。 Direct3D 9 对象有足够的元数据，导致 10Level9 运行时发出对 IDirect3DDevice9：：SetClipPlane 的适当调用。

剪裁平面数学

剪辑平面由具有 4 个组件的向量定义。 前三个组件定义一个 x、y、z 向量，该矢量从我们想要剪辑的空间中的原点产生。 此向量表示一个平面，垂直于矢量，并通过原点运行。 Windows保留平面向量端的所有像素，并剪裁平面后面的所有像素。 (负 w 组件将平面推回，并导致Windows (负 w 导致Windows沿向量线剪裁更多) 。 如果 x、y、z 组件构成一个单元 (规范化) 向量，则 w 将平面 w 单位推回。

图形处理单元 (GPU) 执行的数学运算是顶点向量 (x、y、z、1) 和剪裁平面向量之间的简单点积。 此数学运算在剪辑平面向量上创建投影长度。 负点产品显示要位于平面的剪裁端的顶点。

在视图空间中剪裁

下面是视图空间中的顶点：

$$\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z & 1 \end{bmatrix}$$

下面是视图空间中的剪辑平面：

$$\mathbf{C} = \begin{bmatrix} C_x & C_y & C_z & C_w \end{bmatrix}$$

下面是视图空间中顶点平面和剪辑平面的点积：

ClipDistance = $\mathbf{v} \cdot \mathbf{C} = v_x C_x$ +vyCyvzCzCw + +

此数学运算适用于 Direct3D 10 或更高版本的对象，但不适用于 Direct3D 9 对象。 对于 Direct3D 9，必须首先通过投影转换转换为剪辑空间。

投影矩阵

投影矩阵从视图空间转换顶点，（其中原点为查看器的眼睛，+x 位于右侧，+y 向上，+z 正向前）为剪辑空间。 投影矩阵读取硬件剪辑和 光栅化阶段的顶点。 下面是一个标准透视矩阵，（其他投影需要不同的数学）：
r 窗口宽度/高度 *α* 视角 *f* 从查看器到远平面 *n* 距离，从查看器到近平面
![投影矩阵] (图像/projection-matrix.png)

下一个矩阵是上一个矩阵的简化版本。 我们演示了简化的矩阵，以便稍后在矩阵乘法运算中使用它。

$$\mathbf{P} = \begin{bmatrix} P_x & 0 & 0 & 0 \\ 0 & P_y & 0 & 0 \\ 0 & 0 & A & 1 \\ 0 & 0 & B & 0 \end{bmatrix}$$

现在，我们将视图空间顶点转换为带矩阵相乘的剪辑空间：

$$\mathbf{v}\,\mathbf{P} = \begin{bmatrix} v_x P_x & v_y P_y & v_z A_y + B & v_z \end{bmatrix}$$

在矩阵乘法运算中，x 和 y 分量仅略有调整，但 z 和 w 分量相当混乱。 我们的剪辑平面不会给我们更多的希望。

剪辑空间剪辑平面

在这里，我们希望创建一个剪辑空间剪辑平面，其点产品与剪辑空间顶点给我们的值与 **v 相同。** “视图空间”部分中的剪辑 中的 C。

$$\mathbf{C_P} = \begin{bmatrix} C_{P_x} & C_{P_y} & C_{P_z} & C_{P_w} \end{bmatrix}$$

v ·C = v P ·CP

$v_x C_x$ +vyCyvzCwv$_x$P$_x$CP$_x$vyCPyvzAyCPzBCPzvz + + + + + = + $_{CPw}$

现在，可以通过顶点分量将前面的数学运算分解为四个单独的公式：

$$v_x\, C_x = v_x\, P_x\, C_{p_x} \rightarrow C_{p_x} = \frac{C_x}{P_x}$$

$$v_y\, C_y = v_y\, P_y\, C_{p_y} \rightarrow C_{p_y} = \frac{C_y}{P_y}$$

$$C_w = B\, C_{p_z} \rightarrow C_{p_z} = \frac{C_w}{B}$$

$$v_z\, C_z = v_z\, A_y\, C_{p_z} + v_z\, C_{p_w} \rightarrow C_z = A_y\, C_{p_z} + C_{p_w} \rightarrow C_{p_w} = C_z - A_y\, C_{p_z} \rightarrow C_{p_w} = C_z - \frac{C_w \mathrm{A}}{B}$$

我们的视图空间剪辑平面和投影矩阵派生并给我们剪辑空间剪辑平面。

$$\mathbf{C_P} = \begin{bmatrix} \frac{C_x}{P_x} & \frac{C_y}{P_y} & \frac{C_w}{B} & C_z - \frac{C_w \mathrm{A}}{B} \end{bmatrix}$$

相关主题

HLSL 编程指南

函数声明语法

HLSL 着色器模型 5

项目 • 2022/09/23

本部分包含High-Level着色器语言的概述材料，特别是 Microsoft Direct3D 11 中引入的着色器模型 5 中的新功能。

本节内容

项	说明
动态链接	动态链接允许运行时在绘图时 (做出决策，而不是编译时) 要运行的代码路径。 这减少了着色器扩散问题，着色器具有几乎相同的输入签名。
几何着色器功能	新的几何着色器功能，包括：实例化，在流中基元的顺序无关紧要时提供性能提升，以及多个点输出流，以便着色器可以在多个流上输出顶点。
镶嵌	Direct3D 11 运行时支持实现分割的三个新阶段，该阶段将低细节细分图面转换为 GPU 上的更详细基元。 分割将高阶图面平铺（或分裂）成合适的结构以进行渲染。 这三个分割阶段是外壳着色器、分割器和域着色器阶段。

此外，参考部分涵盖着色器模型 5 的许多新 API 元素，包括：属性、内部函数、着色器模型 5 对象和方法和系统值。

相关主题

HLSL 编程指南

动态链接

项目 • 2022/09/22

图形开发人员有时会创建大型通用着色器，这些着色器可供各种场景项使用。 在运行时，着色器有条件地运行适合给定情况的代码。 遗憾的是，这些大型通用着色器使用常规用途寄存器 (GPR) 效率低下，并且比更小、更有针对性的着色器要慢得多。

着色器模型 5 通过引入动态着色器链接来解决此性能问题。 动态链接使用接口和虚拟函数分隔着色器代码片段，并允许应用程序选择在绘图时要使用的片段。 这通过仅绑定所需的着色器代码而不是整个大型常规用途着色器来提高性能。

本节内容

项	说明
存储要共享的着色器的变量和类型	描述用于存储多个着色器可以共享的变量和类型的类链接对象。
接口和类	介绍如何使用 HLSL 接口和类实现动态链接。
接口使用限制	介绍着色器代码中接口的使用限制。

相关主题

HLSL

存储要共享的着色器的变量和类型

项目 • 2022/09/22

类链接对象是多个着色器可以共享的变量和类型的命名空间。 在调用中传递类链接对象以创建着色器时，运行时会收集一个变量和类型列表，这些变量和类型可以在着色器中实现每个接口，并将这些变量和类型的名称存储在类链接对象中。

因此，调用 ID3D11ClassLinkage：：GetClassInstance 方法以从类链接对象生成类实例时，运行时可以检索与每个着色器中提供的名称相对应的变量或类型， (如果该名称对给定着色器有效) 且使用给定类链接对象创建。

例如，假设你有一个用于实现 Color 接口的 Light 类，并在顶点着色器和像素着色器中使用此类。 例如，通过调用 ID3D11Device：：CreatePixelShader) 创建着色器 (时，运行时确定 Light 类类型在顶点着色器和像素着色器中都可用，并将 Light 类类型添加到类链接对象。 然后，可以在所需位置创建 **浅** 色实例，绑定两个着色器的资源，并在将着色器设置为设备 ((例如，通过调用 ID3D11DeviceContext：:P SSetShader)) 在类实例数组中传递此实例。 然后，运行时将执行以下序列：

1. 验证是否已使用同一类链接对象创建实例。
2. 验证 Light 类类型在顶点着色器和像素着色器中是否可用。
3. 选择正确的函数表，该表对于顶点和像素着色器可能有所不同。
4. 发送实例提供的偏移量。

类链接对象最终是类型和变量名称的存储库。 每个项的最大名称数 (类型和变量) 为 64K。 类型和变量名称越长，存储要求越高，每个着色器存储的接口元数据就越高。 这是因为运行时必须为每个着色器存储这些名称的映射。

相关主题

动态链接

相关主题

动态链接

接口和类

项目 • 2022/09/24

动态着色器链接使用高级着色器语言 (HLSL) 接口和语法上类似于其 C++ 对应项的类。 这允许着色器在编译时引用抽象接口实例，并将这些实例的解析保留为运行时应用程序的具体类。

以下部分详细介绍了如何设置着色器以使用接口和类，以及如何在应用程序代码中初始化接口实例。

- 声明接口
- 声明类
- 着色器中的接口实例声明
- 着色器中的类实例声明
- 在应用程序中初始化接口实例
- 相关主题

声明接口

接口函数的方式与 C++ 中的抽象基类类似。 接口使用接口关键字在着色器中声明，并且仅包含方法声明。 在接口中声明的方法都是从接口派生的任何类中的虚拟方法。 派生类必须实现接口中声明的所有方法。 请注意，接口是声明虚拟方法的唯一方法，C++ 中没有虚拟关键字，类内涵声明虚拟方法。

以下示例着色器代码声明两个接口。

```
interface iBaseLight
{
   float3 IlluminateAmbient(float3 vNormal);
   float3 IlluminateDiffuse(float3 vNormal);
   float3 IlluminateSpecular(float3 vNormal, int specularPower );
};

interface iBaseMaterial
{
   float3 GetAmbientColor(float2 vTexcoord);

   float3 GetDiffuseColor(float2 vTexcoord);

   int GetSpecularPower();

};
```

声明类

类的行为方式与 C++ 中的类类似。 类使用类关键字声明，可以包含成员变量和方法。 类可以从零个或一个类和零个或多个接口继承。 类必须为其继承链中的所有接口实现或继承实现，否则类无法实例化。

以下示例着色器代码演示如何从接口和另一个类派生类。

```
class cAmbientLight : iBaseLight
{
   float3            m_vLightColor;
   bool     m_bEnable;
   float3 IlluminateAmbient(float3 vNormal);
   float3 IlluminateDiffuse(float3 vNormal);
   float3 IlluminateSpecular(float3 vNormal, int specularPower );
};

class cHemiAmbientLight : cAmbientLight
{
   float4   m_vGroundColor;
   float4   m_vDirUp;
   float3 IlluminateAmbient(float3 vNormal);
};
```

着色器中的接口实例声明

接口实例充当提供接口方法实现的类实例的位置持有者。 使用接口的实例，着色器代码可以调用方法，而无需知道将调用该方法的哪个实现。 着色器代码为它定义的每个接口声明一个或多个实例。 这些实例以类似于 C++ 基类指针的方式在着色器代码中使用。

以下示例着色器代码演示了声明多个接口实例并在着色器代码中使用它们。

```
// Declare interface instances
iBaseLight     g_abstractAmbientLighting;
iBaseLight     g_abstractDirectLighting;
iBaseMaterial  g_abstractMaterial;

struct PS_INPUT
{
    float4 vPosition : SV_POSITION;
    float3 vNormal   : NORMAL;
```

```
    float2 vTexcoord : TEXCOORD0;
};

float4 PSMain( PS_INPUT Input ) : SV_TARGET
{
    float3 Ambient = (float3)0.0f;
    Ambient = g_abstractMaterial.GetAmbientColor( Input.vTexcoord ) *
        g_abstractAmbientLighting.IlluminateAmbient( Input.vNormal );

    float3 Diffuse = (float3)0.0f;
    Diffuse += g_abstractMaterial.GetDiffuseColor( Input.vTexcoord ) *
        g_abstractDirectLighting.IlluminateDiffuse( Input.vNormal );

    float3 Specular = (float3)0.0f;
    Specular += g_abstractDirectLighting.IlluminateSpecular( Input.vNormal,
        g_abstractMaterial.GetSpecularPower() );

    float3 Lighting = saturate( Ambient + Diffuse + Specular );

    return float4(Lighting,1.0f);
}
```

着色器中的类实例声明

将用于代替接口实例的每个类都必须在常量缓冲区中声明为变量，或使用 ID3D11ClassLinkage：：CreateClassInstance 方法在运行时由应用程序创建。 接口实例将指向应用程序代码中的类实例。 类实例可以像任何其他变量一样在着色器代码中引用，但派生自接口的类通常仅用于接口实例，并且不会直接由着色器代码引用。

以下示例着色器代码演示了声明多个类实例。

```

cbuffer cbPerFrame : register( b0 )
{
   cAmbientLight     g_ambientLight;
   cHemiAmbientLight g_hemiAmbientLight;
   cDirectionalLight g_directionalLight;
   cEnvironmentLight g_environmentLight;
   float4            g_vEyeDir;
};
```

在应用程序中初始化接口实例

接口实例通过将包含接口分配的动态链接数组传递给 ID3D11DeviceContext SetShader 方法之一，在应用程序代码中初始化。

若要创建动态链接数组，请执行以下步骤

1. 使用 CreateClassLinkage 创建类链接对象。

```
ID3D11ClassLinkage* g_pPSClassLinkage = NULL;
pd3dDevice->CreateClassLinkage( &g_pPSClassLinkage );
```

2. 创建将使用动态类链接的着色器，将类链接对象作为参数传递给着色器的 create 函数。

```
pd3dDevice->CreatePixelShader( pPixelShaderBuffer->GetBufferPointer(),
    pPixelShaderBuffer->GetBufferSize(), g_pPSClassLinkage,
&g_pPixelShader ) );
```

3. 使用 D3DReflect 函数创建 ID3D11ShaderReflection 对象。

```
ID3D11ShaderReflection* pReflector = NULL;
D3DReflect( pPixelShaderBuffer->GetBufferPointer(),
    pPixelShaderBuffer->GetBufferSize(),
    IID_ID3D11ShaderReflection, (void**) &pReflector) );
```

4. 使用着色器反射对象使用 ID3D11ShaderReflection：：GetNumInterfaceSlots 方法获取着色器中的接口实例数。

```
g_iNumPSInterfaces = pReflector->GetNumInterfaceSlots();
```

5. 创建足够大的数组，以容纳着色器中的接口实例数。

```
ID3D11ClassInstance** g_dynamicLinkageArray = NULL;
g_dynamicLinkageArray =
    (ID3D11ClassInstance**) malloc( sizeof(ID3D11ClassInstance*) *
g_iNumPSInterfaces );
```

6. 使用 ID3D11ShaderReflection：：GetVariableByName 和 ID3D11ShaderReflectionVariable：：GetInterfaceSlot 确定对应于每个接口实例的索引。

```
ID3D11ShaderReflectionVariable* pAmbientLightingVar =
    pReflector->GetVariableByName("g_abstractAmbientLighting");
    g_iAmbientLightingOffset = pAmbientLightingVar-
>GetInterfaceSlot(0);
```

7. 使用 ID3D11ClassLinkage：：GetClassInstance 获取从着色器中的接口派生的每个类对象的类实例。

```
g_pPSClassLinkage->GetClassInstance( "g_hemiAmbientLight", 0,
    &g_pHemiAmbientLightClass );
```

8. 通过在动态链接数组中设置相应的条目，将接口实例设置为类实例。

```
g_dynamicLinkageArray[g_iAmbientLightingOffset] =
g_pHemiAmbientLightClass;
```

9. 将动态链接数组作为参数传递给 SetShader 调用。

```
pd3dImmediateContext->PSSetShader( g_pPixelShader,
g_dynamicLinkageArray, g_iNumPSInterfaces );
```

相关主题

动态链接

接口使用限制

项目 • 2022/09/23

当前 GPU 硬件不支持着色器运行时的不同槽信息。 因此，不能在条件表达式（如 if 或 switch 语句）内修改接口引用。

以下着色器代码演示了此限制何时发生，以及可能的备用方法。

给定以下接口声明：

```
interface A
{
    float GetRatio();
    bool IsGood();
};

interface B
{
    float GetValue();
};

A arrayA[6];
B arrayB[6];
```

给定以下类声明：

```
class C1 : A
{
    float var;
    float GetRatio() { return 1.0f; }
    bool IsGood() { return true; }
};

class C2 : C1, B
{
    float GetRatio() { return C1::GetRatio() * 0.33f; }
    float GetValue() { return 5.0f; }
    bool IsGood() { return false; }
};

class C3 : B
{
    float var;
    float GetValue() { return -1.0f; }
};
```

```
class C4 : A, B
{
    float var;
    float GetRatio() { return var; }
    float GetValue() { return var * 2.0f; }
    bool IsGood() { return var > 0.0f; }
};
```

不能在条件表达式中修改接口引用，(if 语句)：

```
float main() : wicked
{
    float rev;
    {
        A a = arrayA[0];
        for( uint i = 0; i < 6; ++i )
        {
            if( arrayA[i].IsGood() )
            {
                // This forces the loop to be unrolled,
                // since the slot information is changing.
                a = arrayA[i];
                rev -= arrayA[i-2].GetRatio();
            }
            else
            {
                // This causes an error since the compiler is
                // unable to determine the interface slot
                rev += arrayB[i].GetValue() + a.GetRatio();
            }
        }
    }
    return rev;
}
```

鉴于同一接口和类声明，可以使用索引来提供相同的功能，并避免强制循环取消滚动。

```
float main() : wicked
{
    float rev;
    {
        uint index = 0;
        for( uint i = 0; i < 6; ++i )
        {
            if( arrayA[i].IsGood() )
            {
                index = i;
```

```
                rev -= arrayA[i-2].GetRatio();
            }
            else
            {
                rev += arrayB[i].GetValue() + arrayA[index].GetRatio();
            }
        }
    }
    return rev;
}
```

相关主题

动态链接

几何着色器功能

项目 • 2022/09/22

本部分包含着色器模型 5 中引入的新几何着色器功能的概述材料。

本节内容

项	说明
如何：为多个输出流编制索引	使用索引在单个几何着色器中声明最多四个输出流。
如何：实例几何着色器	为每个基元执行几何着色器的一个或多个实例。

相关主题

HLSL 着色器模型 5

如何：为多个输出流编制索引

项目 • 2022/09/24

在着色器模型 5 中，几何着色器最多可以支持 4 个单独的流。 这意味着单个着色器可以在一到四个输出流之间输出，具体取决于声明的流数。

为多个输出流编制索引

1. 使用流模板类型定义数据流。

```
    inout PointStream<OutVertex1> myStream1,
```

2. 使用流模板类型定义第二个数据流。

```
    inout PointStream<OutVertex2> myStream2 )
```

3. 使用流输出对象内部 (函数（如 Append 或 RestartStrip））将数据输出到 (或两个) 流。

```
void MyGS(
    InVertex verts[2],
    inout PointStream<OutVertex1> myStream1,
    inout PointStream<OutVertex2> myStream2 )
{
    OutVertex1 myVert1 = TransformVertex1( verts[0] );
    OutVertex2 myVert2 = TransformVertex2( verts[1] );
    myStream1.Append( myVert1 );
    myStream2.Append( myVert2 );
}
```

使用单个输出流时，可以发出三角形条带、线条条或点列表。 将三角形和线条条存储在流出缓冲区中时，它们将分别扩展到三角形列表和线条列表。 还可以对一个流进行光栅化，而不将其发送到内存缓冲区。

使用多个输出流时，所有流都必须包含点，最多可以将一个输出流发送到光栅器。 更常见的是，应用程序不会对任何流进行光栅化。

将数据流式传输到缓冲区后，可以使用该数据来呈现任何基元类型，而不仅仅是用于填充缓冲区的基元类型。

几何着色器的总输出限制为 1024 标量。 当存在多个流时，将从最大流类型计算标量数乘以最大顶点计数。

着色器模型 4 和着色器模型 5 之间的差异：
着色器模型 4：

- 流输出的最大标量数为 64。
- 每个组件寄存器掩码必须跨索引范围匹配。

着色器模型 5：

- 流输出的最大标量数为 128。
- 每个组件寄存器掩码不需要在索引范围内匹配。
- 输出的动态索引在所有流中都必须是合法的。
- 内插模式不需要匹配流。

相关主题

几何着色器功能

如何：实例几何图形着色器

项目 • 2022/09/23

几何着色器实例化允许对每个基元执行相同几何着色器的多个执行。 若要实例化几何着色器，请将实例属性添加到主着色器函数，并在着色器函数正文中标识实例索引参数。

实例几何着色器：

1. 将 实例属性 添加到主函数。

   ```
   [instance(24)]
   ```

   这定义每个基元最多运行 32 个) 的实例数 (。

2. 将 SV_GSInstanceID 系统值附加到函数参数列表中的变量，该变量可用于跟踪要执行的实例的 ID。

   ```
   uint InstanceID : SV_GSInstanceID
   ```

3. 像编译和创建着色器一样，就像使用任何其他几何图形着色器一样。

其他详细信息包括：

- 最大实例计数为 32。
- 最大顶点计数是每个实例的最大顶点计数。
- 每个实例调用 (与任何几何着色器调用一样，) 增加调用计数并生成隐式 RestartStrip () 。

相关主题

几何着色器功能

HLSL 着色器模型 5.1

项目 • 2022/09/22

本部分介绍着色器模型 5.1 的功能，因为它们在实践中适用于 D3D12。 所有 DirectX 12 硬件都支持着色器模型 5.1。

在本节中

主题	描述
SM5.1 中的字节码更改	SM5.1 更改了在指令中声明和引用资源寄存器的方式。
HLSL 指定的根签名	根签名 (D3D12) 的键表和其他元素表，可以在 HLSL 中指定为字符串。

有关着色器语言的语法更改的详细信息，请参阅 着色器模型 5.1。

相关主题

HLSL 编程指南

SM5.1 中的字节码更改

项目 • 2022/09/22

SM5.1 更改了在指令中声明和引用资源寄存器的方式。

SM5.1 将转向声明寄存器“变量”，这类似于组共享内存寄存器的完成方式，以下示例所示：

syntax

```
Texture2D<float4> tex0          : register(t5,  space0);
Texture2D<float4> tex1[][5][3]  : register(t10, space0);
Texture2D<float4> tex2[8]       : register(t0,  space1);
SamplerState samp0              : register(s5, space0);

float4 main(float4 coord : COORD) : SV_TARGET
{
    float4 r = coord;
    r += tex0.Sample(samp0, r.xy);
    r += tex2[r.x].Sample(samp0, r.xy);
    r += tex1[r.x][r.y][r.z].Sample(samp0, r.xy);
    return r;
}
```

以下示例的反汇编如下：

syntax

```
// Resource Bindings:
//
// Name                                 Type  Format         Dim Space Slot
Elements
// ------------------------------ ---------- ------- ----------- ----- ----
---------
// samp0                             sampler      NA          NA     0    5
1
// tex0                              texture  float4          2d     0    5
1
// tex1[0][5][3]                     texture  float4          2d     0   10
unbounded
// tex2[8]                           texture  float4          2d     1    0
8
//
//
//
// Input signature:
//
// Name                 Index   Mask Register SysValue  Format   Used
// -------------------- ----- ------ -------- -------- ------- ------
// COORD                    0   xyzw        0     NONE   float   xyzw
```

```
//
//
// Output signature:
//
// Name                 Index   Mask Register SysValue  Format   Used
// -------------------- ----- ------ -------- -------- ------- ------
// SV_TARGET                0   xyzw        0   TARGET   float   xyzw
//
ps_5_1
dcl_globalFlags refactoringAllowed
dcl_sampler s0[5:5], mode_default, space=0
dcl_resource_texture2d (float,float,float,float) t0[5:5], space=0
dcl_resource_texture2d (float,float,float,float) t1[10:*], space=0
dcl_resource_texture2d (float,float,float,float) t2[0:7], space=1
dcl_input_ps linear v0.xyzw
dcl_output o0.xyzw
dcl_temps 2
sample r0.xyzw, v0.xyxx, t0[0].xyzw, s0[5]
add r0.xyzw, r0.xyzw, v0.xyzw
ftou r1.x, r0.x
sample r1.xyzw, r0.xyxx, t2[r1.x + 0].xyzw, s0[5]
add r0.xyzw, r0.xyzw, r1.xyzw
ftou r1.xyz, r0.zyxz
imul null, r1.yz, r1.zzyz, l(0, 15, 3, 0)
iadd r1.y, r1.z, r1.y
iadd r1.x, r1.x, r1.y
sample r1.xyzw, r0.xyxx, t1[r1.x + 10].xyzw, s0[5]
add o0.xyzw, r0.xyzw, r1.xyzw
ret
// Approximately 12 instruction slots used
```

每个着色器资源范围现在都有一个 ID， (着色器字节码中的名称) 。 例如，tex1 纹理数组在着色器字节代码中变为“t1”。 将唯一 ID 分配到每个资源范围可以实现两个目的：

- 明确标识哪些资源范围 (在指令中dcl_resource_texture2d) 正在编制索引， (请参阅示例指令) 。
- 将属性集附加到声明，例如元素类型、步幅大小、光栅操作模式等。

请注意，范围 ID 与 HLSL 下限声明无关。

反射资源绑定和着色器声明指令的顺序与标识 HLSL 变量和字节码 ID 之间的对应关系相同。

SM5.1 中的每个声明指令使用 3D 操作数来定义范围 ID、下限和上限。 将发出一个附加的标记来指定寄存器空间。 还可能会发出其他标记来传递范围的其他属性，例如，cbuffer 或结构化缓冲区声明指令会发出 cbuffer 或结构的大小。 可以在 d3d12TokenizedProgramFormat.h 和 D3D10ShaderBinary::CShaderCodeParser 中找到确切的编码详细信息。

SM5.1 指令不会发出附加的资源操作数信息作为指令的一部分（与在 SM5.0 中一样）。 此信息现已移至声明说明。 在 SM5.0 中，指令索引资源要求在扩展的操作码标记中描述资源属性，因为索引编制操作会模糊化与声明的关联。 在 SM5.1 中，每个 ID（（例如't1'））都明确与描述所需资源信息的单个声明相关联。 因此，不再发出在指令中用来描述资源信息的扩展操作码标记。

在非声明指令中，采样器、SRV 和 UAV 的资源操作数是一个 2D 操作数。 第一个索引是指定范围 ID 的文本常量。 第二个索引代表索引的线性化值。 该值是相对于对应寄存器空间的开头（而不是相对于逻辑范围的开头）计算的，以更好地与根签名相关联，并降低调整索引所造成的驱动程序编译器负担。

CBV 的资源操作数是一个 3D 操作数：范围的文本 ID、cbuffer 索引、偏移量到 cbuffer 的特定实例中。

相关主题

Direct3D 12 的 HLSL 着色器模型 5.1 功能

Shader Model 5.1（着色器模型 5.1）

HLSL 指定的根签名

项目 • 2022/09/22

根签名 (D3D12) 的资源和其他元素的键表可以指定为字符串。

有关在 HLSL 中指定根签名 的完整详细信息和示例代码，请参阅 D3D12 文档。

相关主题

Direct3D 12 的 HLSL 着色器模型 5.1 功能

Shader Model 5.1（着色器模型 5.1）

HLSL 着色器模型 6.0

项目 • 2022/09/24

描述添加到 HLSL 着色器模型 6.0 中的波形操作内部函数。

- 着色器模型 6.0
- 术语
- 底纹语言内部函数
  - 波形查询
  - 波形投票
  - 波形广播
  - 波形减少
  - 波形扫描和前缀
  - 四宽随机运算
- 硬件功能
- 相关主题

着色器模型 6.0

对于早期的着色器模型，HLSL 编程只公开一个执行线程。 从模型 6.0 开始，提供了新的波形级操作，以显式利用当前 GPU 的并行度-许多线程可以同时在同一内核上执行锁步操作。 例如，模型 6.0 内部函数允许消除屏障构造（当同步范围在 SIMD 处理器的宽度内）或已知彼此相对为原子的一组其他线程时消除屏障构造。

潜在的用例包括：流压缩、减少、块转置、比特排序或快速傅立叶转换 (FFT) 、装箱、流重复和类似方案。

大多数内部函数都出现在像素着色器和计算着色器中，但对于每个函数），（存在一些异常。 这些函数已添加到 API 级别 12 下 DirectX 功能级别 12.0 的要求。

*<这些函数的类型>*参数和返回值意味着表达式的类型，支持的类型是以下列表中*也存*在于应用的目标着色器模型中的类型：

- half、half2、half3、half4
- float、float2、float3、float4
- double、double2、double3、double4
- int、int2、int3、int4
- uint、uint2、uint3、uint4
- short、short2、short3、short4
- ushort、ushort2、ushort3、ushort4
- uint64_t、uint64_t2、uint64_t3、uint64_t4

某些操作 (例如位运算符) 仅支持整数类型。

术语

条款	定义
通道	单个执行线程。 版本 6.0 之前的着色器模型仅在语言级别公开其中一个模型，使扩展完全由实现进行并行 SIMD 处理。
Wave	一组通道 (线程) 同时在处理器中执行。 无需显式屏障才能保证它们并行执行。 类似的概念包括“扭曲”和“波浪线”。
非活动通道	未执行的通道，例如，由于控制流或工作不足，无法填充波形的最小大小。
活动通道	正在执行执行的通道。 在像素着色器中，它可能包含任何帮助程序像素通道。
四边形	一组 4 个相邻通道，对应于以 2x2 方块排列的像素。 它们用于通过 x 或 y 中的差异来估计渐变。 波可能由多个象限组成。 活动象限中的所有像素 (执行，可能是“活动通道”)，但不生成可见结果的像素称为“帮助程序通道”。
帮助程序通道	仅在像素着色器象限中执行渐变的通道。 此类通道的输出将被丢弃，因此不会呈现到目标图面。

底纹语言内部函数

此着色器模型的所有操作都已添加到一系列内部函数中。

波形查询

用于查询单波的内部函数。

Intrinsic	说明	像素着色器	计算着色器
WaveGetLaneCount	返回当前波中的通道数。	*	*
WaveGetLaneIndex	返回当前波中当前通道的索引。	*	*
WaveIsFirstLane	仅返回当前波中具有最小索引的活动通道的 true	*	*

波形投票

这组内部函数比较当前从当前波处于活动状态的线程之间的值。

Intrinsic	说明	像素着色器	计算着色器
WaveActiveAnyTrue	如果表达式在当前波形中的任何活动通道中为 true，则返回 true。	*	*
WaveActiveAllTrue	如果表达式在当前波的所有活动通道中为 true，则返回 true。	*	*
WaveActiveBallot	返回指定波形中所有活动通道的布尔表达式计算的 64 位无符号整数位掩码。	*	*

波形广播

这些内部函数使当前波中的所有活动通道都能从指定的通道接收值，从而有效地广播它。无效通道中的返回值未定义。

Intrinsic	说明	像素着色器	计算着色器
WaveReadLaneAt	返回指定波中给定通道索引的表达式值。	*	*
WaveReadLaneFirst	返回具有最小索引的当前波的活动通道的表达式值。	*	*

波形减少

这些内部函数计算波中所有活动通道的指定操作，并将最终结果广播到所有活动通道。因此，最终输出保证在波中保持一致。

Intrinsic	说明	像素着色器	计算着色器
WaveActiveAllEqual	如果当前波中每个活动通道的表达式 (相同，因此在) 中保持一致，则返回 true。	*	*
WaveActiveBitAnd	返回当前波中所有活动通道中表达式的所有值的按位 AND，并将结果复制到波中的所有通道。	*	*
WaveActiveBitOr	返回当前波中所有活动通道中表达式的所有值的按位 OR，并将结果复制到波中的所有通道。	*	*

Intrinsic	说明	像素着色器	计算着色器
WaveActiveBitXor	返回当前波中所有活动通道中表达式的所有值的按位排他或，并将结果复制到波中的所有通道。	*	*
WaveActiveCountBits	计算在当前波中所有活动通道中计算结果为 true 的布尔变量数，并将结果复制到波中的所有通道。	*	*
WaveActiveMax	计算当前波形中所有活动通道的表达式最大值，并将结果复制到波中的所有通道。	*	*
WaveActiveMin	计算当前波形中所有活动通道的表达式最小值，并将结果复制到波中的所有通道。	*	*
WaveActiveProduct	将表达式的值乘以当前波中的所有活动通道，并将结果复制到波中的所有通道。	*	*
WaveActiveSum	总结当前波形中所有活动通道的表达式值，并将其复制到当前波中的所有通道，并将结果复制到波中的所有通道。	*	*

波形扫描和前缀

这些内部函数将操作应用于每个通道，并将计算的每个部分结果保留在相应的通道中。

Intrinsic	说明	像素着色器	计算着色器
WavePrefixCountBits	返回所有指定布尔变量的总和，这些变量在所有活动通道中设置为 true，其索引小于当前通道。	*	*
WavePrefixSum	返回活动通道中所有值的总和，其索引比此索引小。	*	*
WavePrefixProduct	返回通道中所有值的乘积，然后返回指定波形之一之前的所有值。	*	*

四宽随机操作

这些内部函数对已知包含像素着色器象限的波中的值执行交换操作，如下所示。 象限中像素的索引以扫描行或读取顺序定义 - 其中四边形中的坐标为：

+---------> X

|[0] [1]

|[2] [3]

v

Y

这些例程适用于计算着色器或像素着色器。 在计算着色器中，它们以定义为 SIMD 波内 4 的均匀划分组的象限运行。 在像素着色器中，它们应在 WaveQuadLanes 捕获的波上使用，否则结果未定义。

Intrinsic	说明	像素着色器	计算着色器
QuadReadLaneAt	返回从由 quadLaneID [0..3] 标识的当前象限的通道中读取的指定源值，该值必须在象限中保持一致。	*	
QuadReadAcrossDiagonal	返回从此象限中对角相反的通道读取的指定本地值。	*	
QuadReadAcrossX	返回从此四向 X 方向的其他通道读取的指定源值。	*	
QuadReadAcrossY	返回从 Y 方向的此象限中从另一个通道读取的指定源值。	*	

硬件功能

若要检查波形操作功能在任何特定硬件上是否可用，请调用 ID3D12Device：：CheckFeatureSupport，指出 D3D12_FEATURE_DATA_D3D12_OPTIONS1 结构的说明和使用。

相关主题

- HLSL 编程指南
- 着色器模型 6 内部函数

HLSL 着色器模型 6.4

项目 • 2022/09/24

介绍添加到 HLSL 着色器模型 6.4 的机器学习内部函数。

着色器模型 6.4

这些内部函数是着色器模型 6.4 的必需/支持功能。 因此，除了确保使用着色器模型 6.4 之外，不需要单独的功能位检查。 这些例程支持的最低客户端为版本 1903 Windows 10。

底纹语言内部函数

4 个元素和累积的无符号整数Dot-Product

syntax

```
uint32 dot4add_u8packed(uint32 a, uint32 b, uint32 acc); // ubyte4 a, b;
```

带添加的 4 维无符号整数点积。 将两个输入 DWORD 中无符号 8 位字节的每个对应配对相乘，并将结果求和到 32 位无符号整数累加器中。 此指令在单个 32 位宽 SIMD 通道中运行。 输入也假定为 32 位数量。

4 个元素和累积的带符号整数Dot-Product

syntax

```
int32 dot4add_i8packed(uint32 a, uint32 b, int32 acc); // signed byte4 a, b;
```

带添加的 4 维有符号整数点积。 将两个输入 DWORD 中带符号 8 位字节的相应配对相乘，并将结果求和到 32 位有符号整数累积器中。 此指令在单个 32 位宽 SIMD 通道中运行。 输入也假定为 32 位数量。

单精度浮点 2 元素Dot-Product和累积

syntax

```
float dot2add( half2 a, half2 b, float acc );
```

具有添加的半 2 向量的二维浮点点积。 将两个半精度浮点输入向量的元素相乘，并将结果求和到 32 位浮点累积器中。 此指令在单个 32 位宽 SIMD 通道内运行。 输入是打包到同一通道中的 16 位数量。

这在低精度功能位 (下介绍，指示本机半和短支持) 存在。

SV_ShadingRate

syntax

```
uint shadingRate : SV_ShadingRate
```

一个无符号整数，表示每个像素着色器调用所写入的目标像素数。 有效值属于枚举值集 D3D12_SHADING_RATE。

此系统值适用于 D3D12_VARIABLE_SHADING_RATE_TIER_2 或更高版本的平台。 它最多可以从顶点或几何着色器阶段之一写入。 可以从像素着色器阶段读取它。 有关详细信息，请参阅 可变速率着色。

HLSL 参考

项目 • 2022/09/22

HLSL 参考文档指定语言特征。 它分为多个部分。

- 语言语法 (DirectX HLSL) - HLSL 中的编程着色器要求你了解语言语法，即如何编写 HLSL 代码。 这包括用于声明和初始化变量的代码、编写用户定义的着色器函数，以及添加流控制语句以使函数更加强大。
- 着色器模型与着色器配置文件 - HLSL 编译器基于着色器模型实现规则和限制。 如果使用 Direct3D 10) 和像素着色器，则每个顶点着色器、几何着色器 (中的代码根据编译时提供的着色器模型进行验证。
- **内部函数 (DirectX HLSL) - HLSL** 具有许多内部函数。 这些已实现并测试，以便你能够使用它们，知道它们已调试，并且性能良好。 如果选择编写自己的函数，请参阅用于编写用户定义的函数的语言语法部分。
- Asm 着色器参考 - 可用于编程和调试着色器的程序集说明。
- D3DCompiler 参考 - 编译原始 HLSL 源。
- 内联格式转换参考 - D3DX_DXGIFormatConvert.inl 文件包含可在 Direct3D 11 硬件上的计算着色器或像素着色器中使用的内联格式转换函数。 可以在应用程序中使用这些函数同时读取和写入纹理。 也就是说，可以执行就地图像编辑。 若要使用这些内联格式转换函数，请在应用程序中包括 D3DX_DXGIFormatConvert.inl 文件。
- 附录 (DirectX HLSL) - 附录包含完整性。 它包括关键字和保留字的列表;这些单词不能用作程序中的标识符。 它还包括用于参考的语言语法的列表。
- **HLSL 错误和警告** - 提供着色器可以返回的错误和警告代码。

相关主题

HLSL

HLSL 编程指南

语言语法

项目 • 2022/09/22

HLSL 着色器由变量组成，函数又由语句组成。 语言语法记录了如何定义和声明变量、添加流控制，以便着色器可以根据变量做出运行时决策，以及编写自定义函数。

- 变量
- 流控制
- 函数
- 语句
- 语义

相关主题

HLSL 参考

变量

项目 • 2022/09/22

HLSL 变量类似于 C 编程语言中定义的变量。 与 C 类似，变量具有一些命名限制，具有范围属性，这些属性取决于声明的位置，并且可以将用户元数据附加到它们。 与 C 一样，有几种标准数据类型。 与 C 不同，HLSL 还定义了其他数据类型，以帮助最大化使用矩阵数学对 3D 图形数据进行操作的 4 分量向量的性能。

- 变量语法
- 数据类型
- 语义

相关主题

DirectX HLSL) (语言语法

变量语法

项目 • 2022/09/22

使用以下语法规则声明 HLSL 变量。

[*Storage_Class*][*Type_Modifier*] *类型名称*[*索引*] [*：语义*] [*：Packoffset*] [*：Register*];[*批注*] [= *Initial_Value*]

参数

Storage_Class

可选的存储类修饰符，用于提供有关变量作用域和生存期的编译器提示;可以按任意顺序指定修饰符。

值	说明
extern	将全局变量标记为着色器的外部输入;这是所有全局变量的默认标记。 不能与 **静态**组合。
nointerpolation	不要在将顶点着色器的输出传递给像素着色器之前插入输出。
精确	应用于变量时 **，精确** 关键字将限制用于以下列方式生成分配给该变量的值的任何计算：* 单独操作保持独立。 例如，如果 mul 和 add 操作可能已融合到疯狂的操作中， **则精确** 强制操作保持独立。 相反，必须显式使用疯狂的内部函数。 * 维护操作顺序。 如果指令顺序可能已随机排列以提高性能， **则精确** 确保编译器保留写入顺序。* IEEE 不安全操作受到限制。 如果编译器可能使用了不考虑 NaN 的快速数学运算， (不是数字) ，INF (无限) 值， **则精确** 强制遵守有关 NaN 和 INF 值的 IEEE 要求。 如果不 **精确**，这些优化和数学运算不是 IEEE 安全的。* 限定变量 **精确** 不会使使用该变量 **的精确**运算。 由于 **精确** 传播仅传播给分配给 **精确**限定变量的值的操作，因此，正确进行所需的 **计算可能很** 棘手，因此我们建议将着色器输出直接标记在声明它们的 **精确** 位置，无论是在结构字段上，还是在输出参数上，还是输入函数的返回类型。通过禁用可能会影响最终结果的优化（由于累积精度差异的差异）来以这种方式控制优化，从而保持修改的输出变量的结果不变。 当希望着色器进行分割以保持水密补丁接缝或匹配多个传递的深度值时，它非常有用。示例代码 ： `HLSLmatrix g_mWorldViewProjection;void main(in float3 InPos : Position, out precise float4 OutPos : SV_Position){ // operation is precise because it contributes to the precise parameter OutPos OutPos = mul( float4( InPos, 1.0 ), g_mWorldViewProjection );}`
共享	标记用于在效果之间共享的变量;这是编译器的提示。
groupshared	为计算着色器标记线程组共享内存的变量。 在 D3D10 中，具有组共享存储类的所有变量的最大总大小为 16kb，在 D3D11 中，最大大小为 32kb。 请参阅示例。

值	说明
static	标记局部变量，使其一次初始化并在函数调用之间保留。 如果声明不包含初始值设定项，则该值设置为零。 标记为 **静态** 的全局变量对应用程序不可见。
uniform	在着色器（（例如顶点着色器中的材料颜色）的执行过程中，标记其数据是常量的变量）;默认情况下，全局变量被视为 **统一** 。
volatile	标记经常更改的变量;这是编译器的提示。 此存储类修饰符仅适用于局部变量。 [！注意] HLSL 编译器当前忽略此存储类修饰符。

Type_Modifier

可选的变量类型修饰符。

值	说明
const	标记无法由着色器更改的变量，因此，必须在变量声明中对其进行初始化。 默认情况下，全局变量被视为 **常量** ，（通过将 /Gec 标志提供给编译器）来抑制此行为。
row_major	标记一个变量，该变量将四个组件存储在单个行中，以便它们可以存储在单个常量寄存器中。
column_major	标记一个变量，该变量将 4 个组件存储在单个列中以优化矩阵数学。

ⓘ 备注

如果未指定类型修饰符值，编译器将使用 column_major 作为默认值。

类型

数据类型中列出的任何 HLSL 类型 (DirectX HLSL) 。

Name[*Index*]

唯一标识着色器变量的 ASCII 字符串。 若要定义可选数组，请使用 **数组大小的索引** ，即正整数 = 1。

语义

编译器用于链接着色器输入和输出的可选参数使用情况信息。 顶点和像素着色器有多个预定义 语义 。 编译器将忽略语义，除非它们在全局变量上声明，否则会忽略传递给着色

器的参数。

Packoffset

手动打包着色器常量的可选关键字。 请参阅 packoffset (DirectX HLSL) 。

注册

用于将着色器变量手动分配给特定寄存器的可选关键字。 请参阅 (DirectX HLSL) 注册 。

批注 (s)

可选元数据，以字符串的形式附加到全局变量。 效果框架使用注释，由 HLSL 忽略;若要查看更详细的语法，请参阅 批注语法。

Initial_Value

可选初始值 (s) ;值数应与 *Type* 中的组件数匹配。 每个标记为 **extern** 的全局变量都必须使用文本值进行初始化;每个标记为 **静态** 的变量都必须使用常量进行初始化。

未标记为 **静态** 或 **外部** 的全局变量不会编译到着色器中。 编译器不会自动为全局变量设置默认值，并且不能将它们用于优化。 若要初始化这种类型的全局变量，请使用反射获取其值，然后将该值复制到常量缓冲区。 例如，可以使用 ID3D11ShaderReflection：：GetVariableByName 方法获取变量，使用 ID3D11ShaderReflectionVariable：：GetDesc 方法获取着色器变量说明，并从D3D11_SHADER_VARIABLE_DESC结构的 DefaultValue 成员获取初始值。 若要将值复制到常量缓冲区，必须确保使用 CPU 写入访问创建缓冲区 (D3D11_CPU_ACCESS_WRITE) 。 有关如何创建常量缓冲区的详细信息，请参阅 如何：创建常量缓冲区。

还可以使用 效果框架 自动处理反射和设置初始值。 例如，可以使用 ID3DX11EffectPass：：Apply 方法。

示例

下面是着色器变量声明的几个示例。

```
float fVar;
```

```
float4 color;
float fVar = 3.1f;
```

```
int iVar[3];

int iVar[3] = {1,2,3};

uniform float4 position : SV_POSITION;
const float4 lightDirection = {0,0,1};
```

共享组

HLSL 允许计算着色器的线程通过共享内存交换值。 HLSL 提供屏障基元（如 GroupMemoryBarrierWithGroupSync），等等，以确保对着色器中共享内存的读取和写入进行正确的排序，并避免数据争用。

> **备注**
>
> 硬件在组中执行线程 (扭曲或波形前线），有时可以省略屏障同步以提高性能，前提是仅同步属于同一组的线程是正确的。 但出于以下原因，我们非常不鼓励这种遗漏：
>
> - 这种遗漏会导致非可移植代码，这些代码在某些硬件上可能不起作用，并且不适用于通常以较小的组执行线程的软件光栅器。
> - 与使用全线程屏障相比，你可能通过此遗漏实现的性能改进是次要的。

在 Direct3D 10 中，写入 **组共享**时线程没有同步，这意味着每个线程仅限于数组中的单个位置进行写入。 写入时，使用 SV_GroupIndex 系统值为此数组编制索引，以确保两个线程无法碰撞。 在读取方面，所有线程都有权访问整个数组进行读取。

```
struct GSData
{
    float4 Color;
    float Factor;
}

groupshared GSData data[5*5*1];

[numthreads(5,5,1)]
void main( uint index : SV_GroupIndex )
{
    data[index].Color = (float4)0;
    data[index].Factor = 2.0f;
    GroupMemoryBarrierWithGroupSync();
    ...
}
```

包装

打包矢量和标量子的子组件，其大小足够大，无法跨越寄存器边界。 例如，这些都是有效的：

```
cbuffer MyBuffer
{
    float4 Element1 : packoffset(c0);
    float1 Element2 : packoffset(c1);
    float1 Element3 : packoffset(c1.y);
}
```

不能混合包装类型。

与 register 关键字一样，packoffset 可以是特定的目标。 子组件打包仅适用于 packoffset 关键字，而不适用于 register 关键字。 在 cbuffer 声明中，将忽略 Direct3D 10 目标的 register 关键字，因为它假定是跨平台兼容性。

打包的元素可能会重叠，编译器不会显示错误或警告。 在此示例中，Element2 和 Element3 将与 Element1.x 和 Element1.y 重叠。

```
cbuffer MyBuffer
{
    float4 Element1 : packoffset(c0);
    float1 Element2 : packoffset(c0);
    float1 Element3 : packoffset(c0.y);
}
```

使用 packoffset 的示例为： HLSLWithoutFX10 示例 。

相关主题

DirectX HLSL) (变量

packoffset

项目 • 2022/09/23

可选的着色器常量打包关键字，该关键字使用以下语法：

：packoffset (c[Subcomponent][.component])

参数

项	说明
packoffset	必需关键字。
C	打包仅适用于常量寄存器 (c) 。
[子组件][.component]	可选子组件和组件。 子组件是一个寄存器号，它是一个整数。 组件采用 [.xyzw] 的形式。

注解

使用此关键字在 声明变量类型时手动打包着色器常量。

打包常量时，不能混合常量类型。

对于全局常量和统一常量，编译器的行为略有不同：

- 全局常量。 全局变量作为全局常量添加到编译器 *$Global* cbuffer 中。 自动打包的元素 (声明没有 packoffset 的元素) 将显示在最后一个手动打包的变量之后。 打包全局常量时，可能会混合类型。
- 统一常量。 当着色器在效果框架外部编译着色器时，函数参数列表中的统一参数将被编译器添加到 *$Param* 常量缓冲区。 在效果框架内部编译时，统一常量必须解析为全局范围内定义的统一变量。 无法手动偏移统一常量;其推荐用途仅用于着色器专用化，其中着色器将别名回全局，而不是作为将应用程序数据传递到着色器的方法。

下面是一些其他示例： 使用着色器模型 4 打包常量。

示例

下面是手动打包着色器常量的示例。

打包矢量和标量子组件，其大小足够大，无法跨越寄存器边界。 例如，这些都是有效的：

```
cbuffer MyBuffer
{
    float4 Element1 : packoffset(c0);
    float1 Element2 : packoffset(c1);
    float1 Element3 : packoffset(c1.y);
}
```

另请参阅

变量语法

(DirectX HLSL) 变量

register

项目 • 2022/09/23

将着色器变量分配给特定寄存器的可选关键字，该寄存器使用以下语法：

```
: register ( [shader_profile] , Type#[subcomponent] )
```

parameters

注册

必需关键字。

[shader_profile]

可选 着色器配置文件，可以是着色器目标，也可以只是 ps 或 vs。

Type#[subcomponent]

注册类型、数字和子组件声明。

- 类型为以下项之一：

类型	注册说明
b	常量缓冲区
t	纹理和纹理缓冲区
c	缓冲区偏移量
s	取样器
u	无序访问视图

- # 是寄存器号，它是整数。
- *子组件*是可选的整数。

注解

可以将一个或多个寄存器赋值添加到同一变量声明中，用空格分隔。

对于全局范围内的 Direct3D 10 变量， **寄存器** 关键字的行为与 packoffset (DirectX HLSL) 关键字相同。

示例

下面是一些示例：

```
sampler myVar : register( ps_5_0, s );
```

```
sampler myVar : register( vs, s[8] );
```

```
sampler myVar : register( ps, s[2] )
              : register( ps_5_0, s[0] )
              : register( vs, s[8] );
```

另请参阅

变量语法

DirectX HLSL) (变量

数据类型 (HLSL)

项目 • 2022/09/22

HLSL 支持许多不同的内部数据类型。 此表显示用于定义着色器变量的类型。

使用此内部类型	定义此着色器变量
Scalar	单组件标量
矢量、 矩阵	多组件向量或矩阵
采样器、 纹理 或 缓冲区	采样器、纹理或缓冲区对象
结构 ， 用户定义的	自定义结构或 typedef
Array	声明包含大多数其他类型的文本标量表达式
State 对象	状态对象的 HLSL 表示形式

为了帮助你更好地了解如何在 HLSL 中使用矢量和矩阵，你可能想要阅读有关 HLSL 如何使用 每分量 数学的此背景信息。

相关主题

DirectX HLSL) (变量

缓冲区类型

项目 • 2022/12/14

使用以下语法声明缓冲区变量。

```
缓冲区<类型> 名称;
```

参数

缓冲区

必需关键字。

类型

标量、矢量和一些矩阵 HLSL 类型之一。 只要它符合 4 32 位数量，就可以使用矩阵声明缓冲区变量。 因此，可以编写 `Buffer<float2x2>`。 但 `Buffer<float4x4>` 太大，编译器将生成错误。

名字

唯一标识变量名称的 ASCII 字符串。

示例

下面是缓冲区声明的示例。

```
Buffer<float4> g_Buffer;
```

使用 **Load** HLSL 内部函数的重载版本从缓冲区读取数据，该函数采用一个输入参数 (整数索引) 。 像元素数组一样访问缓冲区;因此，此示例读取第二个元素。

```
float4 bufferData = g_Buffer.Load( 1 );
```

使用 流输出阶段 将数据输出到缓冲区。

备注

需要兼容类型化缓冲区着色器资源视图 (SRV) 才能从缓冲区正确加载。 负载可以选择执行类型转换，例如 **，可以将**RGBA8_UNORM 缓冲区加载到 `float4` 变量中。 对于缓冲区串联结构，请改用 **StructuredBuffer** 。

另请参阅

DirectX HLSL) (数据类型

标量类型

项目 • 2022/09/22

HLSL 支持多种标量数据类型：

- **bool** - true 或 false。
- **int** - 32 位有符号整数。
- **uint** - 32 位无符号整数。
- **dword** - 32 位无符号整数。
- **half** - 16 位浮点值。 此数据类型仅用于语言兼容性。 Direct3D 10 着色器目标将所有半数据类型映射到浮点数据类型。 如果需要此功能），则不能在统一全局变量上使用半数据类型 (使用 /Gec 标志。
- **float** - 32 位浮点值。
- **double** - 64 位浮点值。 不能将双精度值用作流的输入和输出。 若要在着色器之间传递双精度值，请将每个**双精度**值声明为 **uint** 数据类型的配对。 然后，使用 **asuint** 函数将每个**双精度**值打包到 **uint** 的配对，使用**双精度**函数将 **uint** 的配对解压缩回**双精度**值。

从 Windows 8 HLSL 开始，还支持最小精度标量数据类型。 图形驱动程序可以使用大于或等于指定位精度的任何精度实现最小精度标量数据类型。 建议不要依赖于依赖于特定基础精度的固定或包装行为。 例如，图形驱动程序可能以完全 32 位精度对 **min16float** 值执行算术。

- **min16float** - 最小 16 位浮点值。
- **min10float** - 最小 10 位浮点值。
- **min16int** - 最小 16 位有符号整数。
- **min12int** - 最小 12 位有符号整数。
- **min16uint** - 最小 16 位无符号整数。

有关标量文本的详细信息，请参阅 语法。

Direct3D 9 和 Direct3D 10 之间的差异：
在 Direct3D 10 中，以下类型是浮点类型的修饰符。

- **snorm float** - IEEE 32 位带符号规范化浮点范围 -1 到 1（含 1）。
- **unorm float** - IEEE 32 位无符号规范化浮点，范围为 0 到 1（含 1）。

例如，下面是一个 4 分量有符号规范化的浮变量声明。

```
snorm float4 fourComponentIEEEFloat;
```

字符串类型

HLSL 还 **支持字符串类型** ，即 ASCII 字符串。 没有接受字符串的操作或状态，但效果可以查询字符串参数和批注。

示例

C

```
// top-level variable
float globalShaderVariable;

// top-level function
void function(
in float4 position: POSITION0 // top-level argument
              )
{
  float localShaderVariable; // local variable
  function2(...)
}

void function2()
{
  ...
}
```

另请参阅

- 声明标量类型
- DirectX HLSL) (数据类型

矢量类型

项目 • 2022/09/23

向量包含一个和四个标量组件：向量的每个组件必须具有相同的类型。

TypeNumber 名称

```
TypeComponents Name
```

组件

项	说明
TypeComponents	包含两个部分的单个名称。 第一部分是 标量 类型之一。 第二部分是组件数，必须介于 1 和 4 之间（含 1 到 4）。
名字	唯一标识变量名称的 ASCII 字符串。

示例

下面是一些示例：

```
bool    bVector;   // scalar containing 1 Boolean
int1    iVector = 1;
float3  fVector = { 0.2f, 0.3f, 0.4f };
```

还可以使用此语法声明向量：

```
vector <Type, Number> VariableName
```

下面是一些示例：

```
vector <int,    1> iVector = 1;
vector <double, 4> dVector = { 0.2, 0.3, 0.4, 0.5 };
```

另请参阅

DirectX HLSL) (数据类型

矩阵类型

项目 • 2022/09/22

矩阵是一种特殊的数据类型，包含一到十六个组件。 矩阵的每个组件必须具有相同的类型。

TypeComponents 名称

组件

项	说明
TypeComponents	包含三个部分的单个名称。 第一部分是 标量 类型之一。 第二部分是行数。 第三部分是列数。 行数和列数是介于 1 和 4 之间的正整数（含 1 到 4）。
名字	唯一标识变量名称的 ASCII 字符串。

示例

下面是一些示例：

```
int1x1    iMatrix;   // integer matrix with 1 row,  1 column
int4x1    iMatrix;   // integer matrix with 4 rows, 1 column
int1x4    iMatrix;   // integer matrix with 1 row, 4 columns
double3x3 dMatrix;   // double matrix with 3 rows, 3 columns

float2x2 fMatrix = { 0.0f, 0.1, // row 1
                     2.1f, 2.2f // row 2
                   };
```

还可以使用此语法声明矩阵：

```
matrix <Type, Number> VariableName
```

矩阵类型使用尖括号指定类型、行数和列数。 此示例创建一个浮点矩阵，其中包含两行和两列。 可以使用任何标量数据类型。

下面是一个示例：

```
matrix <float, 2, 2> fMatrix = { 0.0f, 0.1, // row 1
                                 2.1f, 2.2f // row 2
                               };
```

另请参阅

DirectX HLSL) (数据类型

Per-Component数学运算

项目 • 2022/09/22

使用 HLSL，可以在算法级别对着色器进行编程。 若要了解语言，需要了解如何声明变量和函数、使用内部函数、定义自定义数据类型并使用语义将着色器参数连接到其他着色器和管道。

了解如何在 HLSL 中创作着色器后，需要了解 API 调用，以便可以：编译特定硬件的着色器、初始化着色器常量，并在必要时初始化其他管道状态。

- 向量类型
- 矩阵类型
  - 矩阵排序
- 示例
- 相关主题

向量类型

向量是一个数据结构，其中包含一到四个组件。

```
bool    bVector;   // scalar containing 1 Boolean
bool1   bVector;   // vector containing 1 Boolean
int1    iVector;   // vector containing 1 int
float3  fVector;   // vector containing 3 floats
double4 dVector;   // vector containing 4 doubles
```

数据类型后面的整数是矢量上的组件数。

初始值设定项也可以包含在声明中。

```
bool    bVector = false;
int1    iVector = 1;
float3  fVector = { 0.2f, 0.3f, 0.4f };
double4 dVector = { 0.2, 0.3, 0.4, 0.5 };
```

或者，矢量类型可用于进行相同的声明：

```
vector <bool,   1> bVector = false;
vector <int,    1> iVector = 1;
vector <float,  3> fVector = { 0.2f, 0.3f, 0.4f };
vector <double, 4> dVector = { 0.2, 0.3, 0.4, 0.5 };
```

矢量类型使用尖括号来指定组件的类型和数量。

矢量最多包含四个组件，每个组件都可以使用两个命名集之一进行访问：

- 位置集：x，y，z，w
- 颜色集：r、g、b、a

这些语句都返回第三个组件中的值。

```
// Given
float4 pos = float4(0,0,2,1);

pos.z    // value is 2
pos.b    // value is 2
```

命名集可以使用一个或多个组件，但它们不能混合。

```
// Given
float4 pos = float4(0,0,2,1);
float2 temp;

temp = pos.xy  // valid
temp = pos.rg  // valid

temp = pos.xg  // NOT VALID because the position and color sets were used.
```

读取组件时指定一个或多个矢量组件，称为重排。 例如：

```
float4 pos = float4(0,0,2,1);
float2 f_2D;
f_2D = pos.xy;   // read two components
f_2D = pos.xz;   // read components in any order
f_2D = pos.zx;

f_2D = pos.xx;   // components can be read more than once
f_2D = pos.yy;
```

屏蔽控制写入的组件数。

```
float4 pos = float4(0,0,2,1);
float4 f_4D;
f_4D    = pos;     // write four components

f_4D.xz = pos.xz;  // write two components
f_4D.zx = pos.xz;  // change the write order

f_4D.xzyw = pos.w; // write one component to more than one component
f_4D.wzyx = pos;
```

无法多次将分配写入同一组件。 因此，此语句的左侧无效：

```
f_4D.xx = pos.xy;   // cannot write to the same destination components
```

此外，组件名称空间不能混合。 这是一个无效的组件写入：

```
f_4D.xg = pos.rgrg;    // invalid write: cannot mix component name spaces
```

以标量形式访问矢量将访问矢量的第一个组件。 以下两个语句等效。

```
f_4D.a = pos * 5.0f;
f_4D.a = pos.r * 5.0f;
```

矩阵类型

矩阵是包含数据行和列的数据结构。 数据可以是任何标量数据类型，但是，矩阵的每个元素都是相同的数据类型。 使用追加到数据类型的逐行字符串指定行数和列数。

```
int1x1    iMatrix;   // integer matrix with 1 row,  1 column
int2x1    iMatrix;   // integer matrix with 2 rows, 1 column
...
int4x1    iMatrix;   // integer matrix with 4 rows, 1 column
...
int1x4    iMatrix;   // integer matrix with 1 row, 4 columns
```

```
double1x1 dMatrix;   // double matrix with 1 row,  1 column
double2x2 dMatrix;   // double matrix with 2 rows, 2 columns
double3x3 dMatrix;   // double matrix with 3 rows, 3 columns
double4x4 dMatrix;   // double matrix with 4 rows, 4 columns
```

最大行数或列数为 4;最小数字为 1。

声明矩阵时，可以初始化矩阵：

```
float2x2 fMatrix = { 0.0f, 0.1, // row 1
                     2.1f, 2.2f // row 2
                   };
```

或者，矩阵类型可用于进行相同的声明：

```
matrix <float, 2, 2> fMatrix = { 0.0f, 0.1, // row 1
                                 2.1f, 2.2f // row 2
                               };
```

矩阵类型使用尖括号来指定类型、行数和列数。 此示例创建一个浮点矩阵，其中包含两行和两列。 可以使用任何标量数据类型。

此声明定义浮点值的矩阵， (32 位浮点数) 包含两行和三列：

```
matrix <float, 2, 3> fFloatMatrix;
```

矩阵包含按行和列组织的值，可以使用结构运算符“.”访问这些值，后跟两个命名集之一：

- 从零开始的行列位置：
  - _m00、_m01、_m02、_m03
  - _m10、_m11、_m12、_m13
  - _m20、_m21、_m22、_m23
  - _m30、_m31、_m32、_m33
- 基于一个行列的位置：
  - _11, _12, _13, _14
  - _21, _22, _23, _24
  - _31, _32, _33, _34
  - _41, _42, _43, _44

每个命名集以下划线开头，后跟行号和列号。 从零开始的约定还包括行号和列号之前的字母“m”。 下面是使用两个命名集访问矩阵的示例：

```
// given
float2x2 fMatrix = { 1.0f, 1.1f, // row 1
                     2.0f, 2.1f  // row 2
                   };

float f_1D;
f_1D = matrix._m00; // read the value in row 1, column 1: 1.0
f_1D = matrix._m11; // read the value in row 2, column 2: 2.1

f_1D = matrix._11;  // read the value in row 1, column 1: 1.0
f_1D = matrix._22;  // read the value in row 2, column 2: 2.1
```

与向量一样，命名集可以使用任一命名集中的一个或多个组件。

```
// Given
float2x2 fMatrix = { 1.0f, 1.1f, // row 1
                     2.0f, 2.1f  // row 2
                   };
float2 temp;

temp = fMatrix._m00_m11 // valid
temp = fMatrix._m11_m00 // valid
temp = fMatrix._11_22   // valid
temp = fMatrix._22_11   // valid
```

还可以使用数组访问表示法访问矩阵，这是从零开始的索引集。 每个索引位于方括号内。 使用以下索引访问 4x4 矩阵：

- [0][0], [0][1], [0][2], [0][3]
- [1][0], [1][1], [1][2], [1][3]
- [2][0], [2][1], [2][2], [2][3]
- [3][0], [3][1], [3][2], [3][3]

下面是访问矩阵的示例：

```
float2x2 fMatrix = { 1.0f, 1.1f, // row 1
                     2.0f, 2.1f  // row 2
                   };
float temp;
```

```
temp = fMatrix[0][0] // single component read
temp = fMatrix[0][1] // single component read
```

请注意，结构运算符“.”不用于访问数组。 数组访问表示法不能使用重排读取多个组件。

```

float2 temp;
temp = fMatrix[0][0]_[0][1] // invalid, cannot read two components
```

但是，数组访问可以读取多组件向量。

```

float2 temp;
float2x2 fMatrix;
temp = fMatrix[0] // read the first row
```

与矢量一样，读取多个矩阵组件称为重排。 可以分配多个组件，前提是仅使用一个名称空间。 这些都是有效的分配：

```

// Given these variables
float4x4 worldMatrix = float4( {0,0,0,0}, {1,1,1,1}, {2,2,2,2}, {3,3,3,3} );
float4x4 tempMatrix;

tempMatrix._m00_m11 = worldMatrix._m00_m11; // multiple components
tempMatrix._m00_m11 = worldMatrix.m13_m23;

tempMatrix._11_22_33 = worldMatrix._11_22_33; // any order on swizzles
tempMatrix._11_22_33 = worldMatrix._24_23_22;
```

屏蔽控制写入的组件数。

```

// Given
float4x4 worldMatrix = float4( {0,0,0,0}, {1,1,1,1}, {2,2,2,2}, {3,3,3,3} );
float4x4 tempMatrix;

tempMatrix._m00_m11 = worldMatrix._m00_m11; // write two components
tempMatrix._m23_m00 = worldMatrix._m00_m11;
```

无法多次将分配写入同一组件。 因此，此语句的左侧无效：

```

```

```
// cannot write to the same component more than once
tempMatrix._m00_m00 = worldMatrix._m00_m11;
```

此外，组件名称空间不能混合。 这是一个无效的组件写入：

```

// Invalid use of same component on left side
tempMatrix._11_m23 = worldMatrix._11_22;
```

矩阵排序

默认情况下，统一参数的矩阵打包顺序设置为列主。 这意味着矩阵的每个列都存储在单个常量寄存器中。 另一方面，一个行主矩阵将矩阵的每一行打包在单个常量寄存器中。 可以使用 **#pragmapack_matrix** 指令或 **row_major** 或 **column_major** 关键字更改矩阵打包。

矩阵中的数据在着色器运行时加载到着色器常量寄存器中。 对于矩阵数据读取方式有两种选择：按行主顺序或列主顺序。 列主顺序表示每个矩阵列将存储在单个常量寄存器中，而行主顺序意味着矩阵的每一行将存储在单个常量寄存器中。 对于矩阵使用多少常量寄存器，这是一个重要考虑因素。

行主矩阵布局如下：

11
21
31
41

12
22
32
42

13
23
33
43

14
24

34
44

列主矩阵的布局如下：

11
12
13
14

21
22
23
24

31
32
33
34

41
42
43
44

行主矩阵和列主矩阵排序确定矩阵组件从着色器输入读取的顺序。 将数据写入常量寄存器后，矩阵顺序不会影响数据在着色器代码中使用或访问的方式。 此外，着色器正文中声明的矩阵不会打包到常量寄存器中。 行主和列主包装顺序对构造函数的包装顺序没有影响， (始终遵循行主排序) 。

矩阵中的数据顺序可以在编译时声明，或者编译器会在运行时对数据进行排序，以便最高效的使用。

示例

HLSL 使用两种特殊类型（一种矢量类型和矩阵类型）来简化 2D 和 3D 图形编程。 每种类型都包含多个组件;向量最多包含四个分量，一个矩阵最多包含 16 个组件。 在标准 HLSL 公式中使用矢量和矩阵时，执行的数学运算旨在按分量工作。 例如，HLSL 实现此乘法：

```
float4 v = a*b;
```

作为四分量相乘。 结果是四个标量：

```
float4 v = a*b;

v.x = a.x*b.x;
v.y = a.y*b.y;
v.z = a.z*b.z;
v.w = a.w*b.w;
```

这是四个乘法，其中每个结果存储在 v 的单独组件中。 这称为四分乘。 HLSL 使用组件数学，使编写着色器非常高效。

这与通常作为生成单个标量的点积实现的乘法大不相同：

```
v = a.x*b.x + a.y*b.y + a.z*b.z + a.w*b.w;
```

矩阵还使用 HLSL 中的每组件操作：

```
float3x3 mat1,mat2;
...
float3x3 mat3 = mat1*mat2;
```

结果是两个矩阵的按分量乘以（，而不是标准 3x3 矩阵相乘）。 每个分量矩阵乘以此第一个术语：

```
mat3.m00 = mat1.m00 * mat2._m00;
```

这不同于 3x3 矩阵乘法，这将产生第一个术语：

```
// First component of a four-component matrix multiply
mat.m00 = mat1._m00 * mat2._m00 +
```

```
            mat1._m01 * mat2._m10 +
            mat1._m02 * mat2._m20 +
            mat1._m03 * mat2._m30;
```

乘法函数的重载版本处理一个操作数是向量，另一个操作数是矩阵。 例如：vector * vector、vector * matrix、matrix * vector 和 matrix * matrix。 例如：

```
float4x3 World;

float4 main(float4 pos : SV_POSITION) : SV_POSITION
{
    float4 val;
    val.xyz = mul(pos,World);
    val.w = 0;

    return val;
}
```

生成的结果与：

```
float4x3 World;

float4 main(float4 pos : SV_POSITION) : SV_POSITION
{
    float4 val;
    val.xyz = (float3) mul((float1x4)pos,World);
    val.w = 0;

    return val;
}
```

此示例使用 (float1x4) 强制转换将 pos 向量强制转换为列向量。 通过强制转换更改向量，或交换提供给乘法的参数的顺序相当于转置矩阵。

自动强制转换会导致乘积函数和点内部函数返回与此处使用的结果相同：

```
{
  float4 val;
  return mul(val,val);
}
```

乘法的此结果为 1x4 * 4x1 = 1x1 向量。 这相当于点产品：

```
{
  float4 val;
  return dot(val,val);
}
```

返回单个标量值。

相关主题

DirectX HLSL) (数据类型

采样器类型

项目 • 2022/09/22

使用以下语法声明采样器状态和采样器比较状态。

Direct3D 9 和 Direct3D 10 及更高版本之间的差异：
下面是 Direct3D 9 中采样器的语法。

```
sampler NameSamplerType = { Texture = <texture_variable>;[state_name = state_value;]... };
```

Direct3D 10 及更高版本中采样器的语法略有变化，以支持纹理对象和采样器数组。

```
SamplerType Name[Index]{ [state_name = state_value;] ... };
```

参数

采样

仅 Direct3D 9。 必需关键字。

名字

唯一标识采样器变量名称的 ASCII 字符串。

[索引]

仅 Direct3D 10 及更高版本。 可选数组大小;大于或等于 1 的正整数。

SamplerType

[in]采样器类型，属于以下类型之一：*sampler*、*sampler1D*、*sampler2D*、*sampler3D*、*samplerCUBE*、*sampler_state*、*SamplerState*。

Direct3D 9 和 Direct3D 10 及更高版本之间的差异：

- Direct3D 10 及更高版本支持另一种采样器类型： *SamplerComparisonState*。

纹理 = <texture_variable>;

仅 Direct3D 9。 纹理变量。 *左侧需要 Texture* 关键字;变量名称位于尖括号中的表达式右侧。

state_name = state_value

[in]可选状态分配 () 。 赋值左侧是状态名称，右侧是状态值。 所有状态赋值都必须显示在语句块中，（大括号）。 每个语句用分号分隔。 下表列出了可能的状态名称。

```
// sampler state
AddressU
AddressV
AddressW
BorderColor
Filter
MaxAnisotropy
MaxLOD
MinLOD
MipLODBias

// sampler-comparison state
ComparisonFunc
```

每个表达式的右侧是分配给每个状态的值。 有关 Direct3D 11 可能的状态值 **，请参阅 D3D11_SAMPLER_DESC** 结构。 状态名称和结构成员之间存在 1 到 1 的关系。 请参阅以下示例。

备注

实现效果时，采样器状态是可能需要在管道中设置的几种状态之一来呈现。 有关可在效果中设置的所有可能状态的列表，请参阅：

- Direct3D 10 使用 状态组。
- Direct3D 9 使用单个 状态。

示例

Direct3D 9 和 Direct3D 10 之间的差异：
下面是 BasicHLSL 示例中 Direct3D 9 采样器的部分示例。

```
sampler MeshTextureSampler =
sampler_state
{
    Texture = <g_MeshTexture>;
    MipFilter = LINEAR;
    MinFilter = LINEAR;
```

```
    MagFilter = LINEAR;
};
```

下面是 BasicHLSL10 示例中 Direct3D 10 采样器的部分示例。

```
SamplerState MeshTextureSampler
{
    Filter = MIN_MAG_MIP_LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};
```

下面是声明采样器比较状态的部分示例，并在 Direct3D 10 中调用比较采样器。

```
SamplerComparisonState ShadowSampler
{
   // sampler state
   Filter = COMPARISON_MIN_MAG_LINEAR_MIP_POINT;
   AddressU = MIRROR;
   AddressV = MIRROR;

   // sampler comparison state
   ComparisonFunc = LESS;
};

float3 vModProjUV;
  ...
float fShadow = g_ShadowMap.SampleCmpLevelZero( ShadowSampler,
vModProjUV.xy, vModProjUV.z);
```

另请参阅

DirectX HLSL) (数据类型

着色器类型

项目 • 2022/09/23

在效果中声明着色器变量的语法已从 Direct3D 9 更改为 Direct3D 10。

Direct3D 10 的着色器类型

使用着色器类型语法在 Direct3D 10) 中声明效果传递 (的着色器变量：

SetPixelShader 编译 (ShaderTarget、ShaderFunction) ; SetGeometryShader 编译 (ShaderTarget、ShaderFunction) ; SetVertexShader 编译 (ShaderTarget、ShaderFunction) ;

参数

项	说明
SetXXXShader	用于创建着色器对象的 Direct3D API 调用。 可以是： SetPixelShader 或 SetGeometryShader 或 SetVertexShader。
ShaderTarget	要编译的着色器模型。 这适用于所有目标，包括所有 Direct3D 9 目标以及 着色器模型 4 目标：vs_4_0、gs_4_0和ps_4_0。
ShaderFunction	包含着色器入口点函数名称的 ASCII 字符串;这是在调用着色器时开始执行的函数。 (...) 表示着色器参数;这些参数与传递给着色器创建 API 的参数相同：VSSetShader 或 GSSetShader 或 PSSetShader。

示例

下面是为特定着色器模型编译的顶点着色器和像素着色器对象的示例。 在 Direct3D 10 示例中，没有几何着色器，因此指针设置为 NULL。

```
// Direct3D 10
technique10 Render
{
    pass P0
    {
        SetVertexShader( CompileShader( vs_4_0, VS() ) );
        SetGeometryShader( NULL );
        SetPixelShader( CompileShader( ps_4_0, PS() ) );
    }
}
```

Direct3D 9 的着色器类型

使用着色器类型语法在 Direct3D 9) 的效果传递 (内声明着色器变量：

PixelShader = 编译 ShaderTarget ShaderFunction (...) ;**顶点**Shader = 编译 ShaderTarget ShaderFunction (...) ;

参数

项	说明
XXXShader	一个着色器变量，表示已编译的着色器。 可以是 ：PixelShader 或 VertexShader。
ShaderTarget	要编译的 着色器模型 ;取决于着色器变量的类型。
ShaderFunction (...)	包含着色器入口点函数名称的 ASCII 字符串;这是在调用着色器时开始执行的函数。 (...) 表示着色器参数;这些参数与传递给着色器创建 API 的参数相同：SetVertexShader 或 SetPixelShader。

示例

下面是为特定着色器模型编译的顶点着色器对象和像素着色器对象的示例。

```
// Direct3D 9
technique RenderSceneWithTexture1Light
{
    pass P0
    {
        VertexShader = compile vs_2_0 RenderSceneVS( 1, true, true );
        PixelShader  = compile ps_2_0 RenderScenePS( true );
    }
}
```

另请参阅

(DirectX HLSL) 的数据类型

纹理类型

项目 • 2022/09/22

使用以下语法声明纹理变量。

类型名称;

参数

项	说明
类型	以下类型之一：纹理 (非类型化，以实现向后兼容性) 、Texture1D、Texture1DArray、Texture2D、Texture2D、Texture2D、Texture3D、TextureCube。 元素大小必须适合 4 个 32 位数量。
名字	唯一标识变量名称的 ASCII 字符串。

注解

使用纹理有三个部分。

1. 声明纹理变量。 这与上面所示的语法一起完成。 例如，这些声明是有效的声明。

   ```
   texture g_MeshTexture;
   ```

   - 或 -

   ```
   Texture2D g_MeshTexture;
   ```

2. 声明和初始化采样器对象。 这是在 Direct3D 9 和 Direct3D 10 中略有不同语法完成的。 有关采样器对象语法的详细信息，请参阅 Sampler 类型 (DirectX HLSL) 。

3. 在着色器中调用纹理函数。

Direct3D 9 和 Direct3D 10 之间的差异：

Direct3D 9 使用 内部纹理函数 来执行纹理操作。 此示例来自 BasicHLSL 示例 ，使用 tex2D (s、t) (DirectX HLSL) 来执行纹理采样。

```
Output.RGBColor = tex2D(MeshTextureSampler, In.TextureUV) * In.Diffuse;
```

Direct3D 10 改用 模板化纹理对象 。 下面是等效纹理操作的示例。

```
Output.RGBColor = g_MeshTexture.Sample(MeshTextureSampler, In.TextureUV) *
In.Diffuse;
```

另请参阅

(DirectX HLSL) 的数据类型

结构类型

项目 • 2022/09/23

使用以下语法使用 HLSL 声明结构。

结构*名称*{ [*InterpolationModifier*] *Type*[*RxC*]*MemberName*; ... };

参数

名字

唯一标识结构名称的 ASCII 字符串。

[*InterpolationModifier*]

指定内插类型的可选修饰符。 有关详细信息，请参阅“备注”。

Type[*RxC*]

具有可选行 (*R*) x 列的成员类型， (*C*) 数组大小。 结构至少包含一个元素;如果它包含多个元素，则元素都是同一类型。 行数和列是介于 1 到 4 之间的无符号整数（含 1 和 4）。

MemberName

唯一标识成员名称的 ASCII 字符串。

注解

可以在任何结构成员或像素着色器函数的参数上指定内插修饰符。 如果两个位置都出现修饰符，则外部修饰符 (像素着色器参数修饰符) (结构修饰符) 重写内部修饰符。

编译着色器或效果时，着色器编译器会根据 HLSL 打包规则打包结构成员。

着色器模型 4 中引入的内插修饰符

用于像素着色器输入的顶点着色器输出会线性内插，以便在光栅化期间获取每像素值。 若要设置内插方法，请使用 着色器模型 4 或更高版本支持的任何以下值。 在未用作像素着色器输入的任何顶点着色器输出上忽略修饰符。

内插修饰符	说明
线性	着色器输入之间的内插;如果未指定内插修饰符， **则线性** 为默认值。

内插修饰符	说明
centroid	在像素覆盖区域内的样本之间内插 (这可能需要从像素中心) 推断终结点。 如果部分覆盖像素 (即使像素中心未覆盖) ，质心采样也可能提高抗锯齿。 **质心**修饰符必须与**线性**修饰符或**无指定**修饰符组合在一起。
nointerpolation	请勿内插。
noperspective	在内插期间不要执行透视更正。 noperspective 修饰符可以与**质心**修饰符组合在一起。
样品	**在着色器模型 4.1 及更高版本中可用**在示例位置而不是像素中心内插。 这会导致像素着色器执行每个样本而不是每像素。 导致按样本执行的另一种方法是具有语义SV_SampleIndex的输入，该输入指示当前示例。 仅包含指定 **样本** 的输入 (或输入SV_SampleIndex) 在像素中的着色器调用之间有所不同，而其他输入则不指定修饰符 (例如，如果在不同输入上混合修饰符，) 仍插在像素中心。 像素着色器调用和深度/模具测试都发生在像素中每个覆盖样本中。 这有时称为 *超级采样*。 相比之下，在没有采样频率调用（称为 *多重采样* ）的情况下，无论覆盖了多少个样本，每个像素都会调用像素着色器一次，而深度/模具测试以采样频率进行。 这两种模式都提供等效的边缘抗锯齿。 但是，超级采样通过调用像素着色器更频繁地提供更好的着色质量。

1. 使用 int/uint 类型时，唯一有效的选项是“nointerpolation”。

内插修饰符可以应用于结构成员或 函数参数，也可以同时应用于这两者。

示例

下面是一些示例结构声明。

```
struct struct1
{
  int    a;
}
```

此声明包括数组。

```
struct struct2
{
  int    a;
  float  b;
  int4x4 iMatrix;
}
```

此声明包括内插修饰符。

```
struct In
{
  centroid float2 Texcoord;
};
```

另请参阅

DirectX HLSL) (数据类型

用户定义类型

项目 • 2022/09/23

使用以下语法声明用户定义的类型。

typedef **[const] Type Name[Index]**;

parameters

项	说明
[const]	可选。 此关键字将类型显式标记为常量。
类型	标识数据类型;必须是 HLSL 内部数据类型之一。
名字	唯一标识变量名称的 ASCII 字符串。
指数	可选数组大小。 必须是介于 1 和 4 之间的无符号整数。

除了内置内部数据类型外，HLSL 还支持遵循以下语法的用户定义或自定义类型：

注解

用户定义的类型不区分大小写。 为方便起见，以下类型会在超全局范围内自动定义。

```
typedef vector <bool, #> bool#;
typedef vector <int, #> int#;
typedef vector <uint, #> uint#;
typedef vector <half, #> half#;
typedef vector <float, #> float#;
typedef vector <double, #> double#;

typedef matrix <bool, #, #> bool#x#;
typedef matrix <int, #, #> int#x#;
typedef matrix <uint, #, #> uint#x#;
typedef matrix <half, #, #> half#x#;
typedef matrix <float, #, #> float#x#;
typedef matrix <double, #, #> double#x#;
```

磅号 (#) 表示介于 1 和 4 之间的整数数字。

为了与 DirectX 8 效果兼容，将在超全局范围内自动定义以下类型：

```
typedef int DWORD;
typedef float FLOAT;
typedef vector <float, 4> VECTOR;
typedef matrix <float, 4, 4> MATRIX;
typedef string STRING;
typedef texture TEXTURE;
typedef pixelshader PIXELSHADER;
typedef vertexshader VERTEXSHADER;
```

相关主题

(DirectX HLSL) 的数据类型

状态对象

项目 • 2022/09/22

借助着色器模型 6.3 及更高版本，除了使用 Direct3D 12 API 外，应用程序还能够方便地灵活地直接在 HLSL 着色器代码中定义 DXR 状态对象。

在 HLSL 中，使用以下语法声明状态对象：

syntax

```
Type Name =
{
    Field1,
    Field2,
    ...
};
```

项	说明
类型	标识子对象的类型。 必须是受支持的 HLSL 子对象类型之一。
名字	唯一标识变量名称的 ASCII 字符串。
Field[1， 2， ...]	子对象的字段。 下面介绍了每种子对象类型的特定字段。

子对象类型的列表：

- StateObjectConfig
- GlobalRootSignature
- LocalRootSignature
- SubobjectToExportsAssocation
- RaytracingShaderConfig
- RaytracingPipelineConfig
- TriangleHitGroup
- ProceduralPrimitiveHitGroup

StateObjectConfig

StateObjectConfig 子对象类型对应于 D3D12_STATE_OBJECT_CONFIG 结构。

它有一个字段，一个按位标志，即一个或两个

- STATE_OBJECT_FLAGS_ALLOW_LOCAL_DEPENDENCIES_ON_EXTERNAL_DEFINITONS
- STATE_OBJECT_FLAGS_ALLOW_EXTERNAL_DEPENDENCIES_ON_LOCAL_DEFINITIONS

或者，对于这两个都不为零。

例如：

```
StateObjectConfig MyStateObjectConfig =
{
    STATE_OBJECT_FLAGS_ALLOW_LOCAL_DEPENDENCIES_ON_EXTERNAL_DEFINITONS
};
```

GlobalRootSignature

GlobalRootSignature 对应于 D3D12_GLOBAL_ROOT_SIGNATURE 结构。

这些字段由描述根签名部分的一些字符串组成。 有关此方面的参考，请参阅 在 HLSL 中指定根签名。

例如：

```
GlobalRootSignature MyGlobalRootSignature =
{
    "DescriptorTable(UAV(u0))," // Output texture
    "SRV(t0)," // Acceleration
structure
    "CBV(b0)," // Scene constants
    "DescriptorTable(SRV(t1, numDescriptors = 2))" // Static index and
vertex buffers.
};
```

LocalRootSignature

LocalRootSignature 对应于 D3D12_LOCAL_ROOT_SIGNATURE 结构。

与全局根签名子对象一样，字段由描述根签名部分的一些字符串组成。 有关此方面的参考，请参阅 在 HLSL 中指定根签名。

例如：

```
LocalRootSignature MyLocalRootSignature =
{
```

```
    "RootConstants(num32BitConstants = 4, b1)"  // Cube constants
};
```

SubobjectToExportsAssocation

默认情况下，仅与导出相同的库中声明的子对象能够应用于该导出。 但是，应用程序能够替代该对象并获取特定子对象与哪些导出相关的内容。 在 HLSL 中，此"显式关联"是使用 SubobjectToExportsAssocation 完成的。

SubobjectToExportsAssocation 对应于 D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION 结构。

此子对象使用语法声明

```
syntax

SubobjectToExportsAssocation Name =
{
    SubobjectName,
    Exports
};
```

项	说明
名字	唯一标识变量名称的 ASCII 字符串。
SubobjectName	标识导出的子对象的字符串。
出口	包含以分号分隔的导出列表的字符串。

例如：

```
SubobjectToExportsAssociation MyLocalRootSignatureAssociation =
{
    "MyLocalRootSignature",    // Subobject name
    "MyHitGroup;MyMissShader"  // Exports association
};
```

请注意，这两个字段都使用 *导出* 的名称。 如果应用程序选择进行导出重命名，则导出的名称可能与 HLSL 中的原始名称不同。

RaytracingShaderConfig

RaytracingShaderConfig 对应于 D3D12_RAYTRACING_SHADER_CONFIG 结构。

此子对象使用语法声明

syntax

```
RaytracingShaderConfig Name =
{
    MaxPayloadSize,
    MaxAttributeSize
};
```

项	描述
名字	唯一标识变量名称的 ASCII 字符串。
MaxPayloadSize	标量的最大存储的数值 (计算在关联光线跟踪着色器的光线有效负载中每个) 4 个字节。
MaxAttributeSize	最大标量数的数值 (计算为 4 个字节，每个) 可用于关联光线跟踪着色器中的属性。 该值不能超过 D3D12_RAYTRACING_MAX_ATTRIBUTE_SIZE_IN_BYTES。

例如：

```
RaytracingShaderConfig MyShaderConfig =
{
    16,  // Max payload size
    8    // Max attribute size
};
```

RaytracingPipelineConfig

RaytracingPipelineConfig 对应于 D3D12_RAYTRACING_PIPELINE_CONFIG 结构。

此子对象使用语法声明

syntax

```
RaytracingPipelineConfig Name =
{
    MaxTraceRecursionDepth
};
```

项	说明

项	说明
名字	唯一标识变量名称的 ASCII 字符串。
MaxTraceRecursionDepth	用于光线跟踪管道中光线递归的数字限制。 它是介于 0 和 31 之间的数字，包括 0 和 31。

例如：

```
RaytracingPipelineConfig MyPipelineConfig =
{
    1  // Max trace recursion depth
};
```

由于光线跟踪递归会产生性能成本，因此应用程序应使用所需的结果所需的最低递归深度。

如果着色器调用尚未达到最大递归深度，则可以调用 TraceRay 任意次数。 但是，如果它们达到或超过最大递归深度，则调用 TraceRay 会将设备置于删除状态。 因此，如果跟踪着色器满足或超过最大递归深度，应注意停止调用 TraceRay。

TriangleHitGroup

TriangleHitGroup 对应于类型字段设置为D3D12_HIT_GROUP_TYPE_TRIANGLES的D3D12_HIT_GROUP_DESC结构。

此子对象使用语法声明

```
syntax

TriangleHitGroup Name =
{
    AnyHitShader,
    ClosestHitShader
};
```

项	说明
名字	唯一标识变量名称的 ASCII 字符串。
AnyHitShader	命中组的 anyhit 着色器的字符串名称或空字符串。
ClosestHitShader	命中组最近的命中着色器或空字符串的字符串名称。

例如：

```
TriangleHitGroup MyHitGroup =
{
    "",                     // AnyHit
    "MyClosestHitShader",   // ClosestHit
};
```

请注意，这两个字段都使用 *导出* 的名称。 如果应用程序选择进行导出重命名，则导出的名称可能与 HLSL 中的原始名称不同。

ProceduralPrimitiveHitGroup

ProceduralPrimitiveHitGroup 对应于类型字段设置为 D3D12_HIT_GROUP_TYPE_PROCEDURAL_PRIMITIVE的D3D12_HIT_GROUP_DESC结构。

此子对象使用语法声明

syntax

```
ProceduralPrimitiveHitGroup Name =
{
    AnyHitShader,
    ClosestHitShader,
    IntersectionShader
};
```

项	描述
名字	唯一标识变量名称的 ASCII 字符串。
AnyHitShader	命中组的 anyhit 着色器的字符串名称或空字符串。
ClosestHitShader	命中组最近的命中着色器或空字符串的字符串名称。
IntersectionShader	命中组的交集着色器的字符串名称或空字符串。

例如：

```
ProceduralPrimitiveHitGroup MyProceduralHitGroup
{
    "MyAnyHit",       // AnyHit
    "MyClosestHit",   // ClosestHit
```

```
    "MyIntersection"  // Intersection
};
```

请注意，这三个字段使用 *导出* 的名称。 如果应用程序选择进行导出重命名，则导出的名称可能与 HLSL 中的原始名称不同。

注解

子对象具有“关联”或“哪个子对象与哪个导出”的概念。

通过着色器代码指定子对象时，选择“哪个子对象随哪个导出一起”遵循 DXR 规范 中所述的规则。 具体而言，假设应用程序有一些导出。 如果应用程序通过着色器代码将导出与根签名 A 相关联，并通过应用程序代码将根签名 B 关联到根签名 B，则 B 是已使用的签名。 使用 B 而不是“生成错误”的设计使应用程序能够方便地使用应用程序代码替代 DXIL 关联，而不是被迫重新编译着色器来解决不匹配的问题。

相关主题

DirectX 开发人员博客帖子“D3D12 中的新增功能 - DirectX 光线跟踪 (DXR) 现在支持库子对象”

DirectX 光线跟踪 (DXR) 功能规格

示例：D3D12RaytracingLibrarySubobjects

流控制

项目 • 2022/09/23

大多数硬件旨在逐行运行着色器代码行，并执行每个 HLSL 语句一次。 流控制语句在运行时确定接下来要执行的 HLSL 语句块。 使用流控制语句，着色器可以循环访问一组语句，或者跳转 (分支) 到下一行中的指令以外的指令。 某些流控制语句支持在编译时指定的静态控制;其他人提供谓词控件，这是运行时做出的按组件决策，而其他人则支持动态控制，这是基于变量内容在运行时做出的决策。

HLSL 支持以下流控制语句。

- break
- continue
- 丢弃
- do
- for
- if
- switch
- while

相关主题

DirectX HLSL) (语言语法

break 语句

项目 • 2022/09/23

退出周围循环 (这样做，同时) 。

break;

参数

无

请参阅

Flow控件

continue 语句

项目 • 2022/09/22

停止执行当前循环 (执行，同时)，更新循环条件，然后从循环顶部开始执行。

continue;

parameters

无

请参阅

Flow控件

discard 语句

项目 • 2022/09/22

不要输出当前像素的结果。

放弃;

参数

无

备注

只能从像素着色器调用此语句;几何着色器或顶点着色器中不支持它。

另请参阅

Flow控件

do 语句

项目 • 2022/09/22

持续执行一系列语句，直到条件表达式失败。

[*属性*] 执行 { *语句块*; } 时 (*条件*) ;

参数

属性

一个可选参数，用于控制语句的编译方式。

属性	说明
fastopt	减少编译时间，但生成攻击性较低的优化。 如果使用此属性，编译器将不会取消滚动循环。 此属性仅影响支持 中断 指令的着色器模型目标。 此属性在着色器模型 vs_2_x 和 着色器模型 3 及更高版本中可用。 编译器编译循环时，着色 器模型 4 及更高版本尤其有用。 默认情况下，编译器模拟循环，以评估是否可以取消注册循环。 如果不希望编译器取消滚动循环，请使用此属性来减少编译时间。

语句块

一个或多个 语句。

条件

条件 表达式。 在计算表达式之前执行语句块。 当表达式的计算结果为 false 时，将退出循环。

要求

要求	Value
标头	Ocidl.h

另请参阅

Flow控件

for 语句

项目 • 2022/09/23

基于条件表达式的计算，以迭代方式执行一系列语句。

[*属性*] 用于 (*初始值设定项*;*条件*;*迭代器*) { *语句块*; }

参数

属性

一个可选参数，用于控制如何编译语句。 如果未指定任何属性，编译器将首先尝试发出循环的滚动版本，如果失败，或者如果取消注册循环，某些操作会更容易，则回退到循环的未注册版本。

属性	说明
取消滚动 (x)	取消滚动循环，直到停止执行。 可以选择指定循环执行的最大次数。 与 [loop] 属性不兼容。
loop	生成使用流控制执行循环每次迭代的代码。 与 [unroll] 属性不兼容。
fastopt	减少编译时间，但产生不太积极的优化。 如果使用此属性，编译器将不会取消滚动循环。 此属性仅影响支持 中断 指令的着色器模型目标。 此属性可用于着色器模型 vs_2_x 和 着色器模型 3 及更高版本。 当编译器编译循环时，着色 器模型 4 和更高版本尤其有用。 默认情况下，编译器模拟循环，以评估它是否可以取消注册它们。 如果不希望编译器取消滚动循环，请使用此属性来减少编译时间。
allow_uav_condition	允许计算着色器循环终止条件基于 UAV 读取。 循环不得包含同步内部函数。

初始 化

循环计数器的初始值。

条件

条件 表达式。 如果条件表达式的计算结果为 true，则执行语句块。 当表达式计算结果为 false 时，循环将结束。

迭 代

更新循环计数器的值。

语句块

一个或多个 HLSL 语句。

注解

[unroll] 和 [loop] 属性互斥，并在指定两者时生成编译器错误。

如果指定 [unroll] ，**则忽略** [fastopt] 和 [allow_uav_condition] 属性。

另请参阅

Flow控件

if 语句

项目 • 2022/09/24

根据条件表达式的计算，有条件地执行一系列语句。

[*属性*] 如果 (*条件*) { *语句块*; }

参数

属性

一个可选参数，用于控制如何编译语句。

Attribute	说明
branch	仅评估 if 语句的一侧，具体取决于给定条件。 [！注意] 使用 着色器模型 2.x 或 着色器模型 3.0 时，每次使用动态分支时，都会消耗资源。 因此，如果针对这些配置文件时过度使用动态分支，则可以收到编译错误。
平展 (flatten)	计算 if 语句的两侧，并在两个结果值之间进行选择。

条件

条件 表达式。 计算表达式，如果为 true，则执行语句块。

语句块

一个或多个 HLSL 语句。

注解

当编译器使用分支方法编译 if 语句时，它将生成仅评估 if 语句的一端的代码，具体取决于给定条件。 例如，在 if 语句中：

```
[branch] if(x)
{
    x = sqrt(x);
}
```

if 语句具有等效于 x = x 的隐式 else 块。 由于我们已告知编译器将分支方法与前面的分支属性一起使用，因此编译的代码将评估 x 并仅执行应执行的一侧：如果 x 为零，则它将执行 **另一** 端，如果为非零，则执行 **该** 端。

相反，如果使用 **平展** 属性，则编译的代码将计算 **if** 语句的两侧，并使用 x 的原始值在两个生成的值之间进行选择。 下面是平展属性用法的示例：

```
[flatten] if(x)
{
    x = sqrt(x);
}
```

在某些情况下，使用分支或平展属性可能会生成编译错误。 如果 if 语句的任一端包含渐变函数（如 tex2D ），则分支属性可能会失败。 如果 if 语句的任一端包含流追加语句或任何其他具有副作用的语句，则平展属性可能会失败。

if 语句还可以使用可选其他块。 如果 **if** 表达式为 true，则处理与 if 语句关联的语句块中的代码。 否则，将处理与可选其他块关联的语句块。

另请参阅

Flow控件

switch 语句

项目 • 2022/09/23

根据选择器的值，将控件传输到开关正文中的其他语句块。

[*属性*] 开关 (*Selector*) { case 0 ： { *StatementBlock*; } break; case 1 ： { *StatementBlock*; } break; case n ： { *StatementBlock*; } break; default ： { *StatementBlock*; } break;

参数

属性

一个可选参数，用于控制如何编译语句。 如果未指定任何属性，编译器可以使用硬件开关或发出一系列 if 语句。

属性	说明
平展 (flatten)	将语句编译为一系列 if 语句，每个语句都具有 **平展** 属性。
branch	将语句编译为一系列 if 语句，每个语句都有 **分支** 属性。 [！注意] 使用 着色器模型 2.x 或 着色器模型 3.0 时，每次使用动态分支时，都会消耗资源。 因此，如果针对这些配置文件时过度使用动态分支，则可以收到编译错误。
forcecase	强制硬件中的 switch 语句。 [！注意] 需要 功能级别 10_0 或更高版本的硬件。
call	交换机中各个事例的正文将移动到硬件子例程中，交换机将是一系列子例程调用。 [！注意] 需要 功能级别 10_0 或更高版本的硬件。

选择

变量。 大括号内的 case 语句将分别检查此变量，以查看 SwitchValue 是否与其特定 CaseValue 匹配。

StatementBlock

一个或多个 语句。

注解

```
[branch] switch(a)
{
    case 0:
        return 0;
    case 1:
        return 1;
    case 2:
        return 3;
    default:
        return 6;
}
```

等效于：

```
[branch] if( a == 2 )
    return 3;
else if( a == 1 )
    return 1;
else if( a == 0 )
    return 0;
else
    return 6;
```

下面是 forcecase 和调用流控制属性的示例用法：

```
[forcecase] switch(a)
{
    case 0:
        return 0;
    case 1:
        return 1;
    case 2:
        return 3;
```

```
    default:
        return 6;
}

[call] switch(a)
{
    case 0:
        return 0;
    case 1:
        return 1;
    case 2:
        return 3;
    default:
        return 6;
}
```

要求

要求	Value
标头	Urlmon.h

另请参阅

Flow控件

while 语句

项目 • 2022/09/22

执行语句块，直到条件表达式失败。

[*属性*] 同时 (*条件*) { *语句块*; }

parameters

属性

一个可选参数，用于控制语句的编译方式。

属性	说明
unroll (x)	取消滚动循环，直到它停止执行。 （可选）可以指定循环可以执行的最大次数。
loop	在编译的着色器中使用流控制语句;不要取消滚动循环。
fastopt	减少编译时间，但生成攻击性较低的优化。 如果使用此属性，编译器将不会取消滚动循环。 此属性仅影响支持 中断 指令的着色器模型目标。 此属性在着色器模型 vs_2_x 和 着色器模型 3 及更高版本中可用。 编译器编译循环时，着色 器模型 4 及更高版本尤其有用。 默认情况下，编译器模拟循环，以评估是否可以取消注册循环。 如果不希望编译器取消滚动循环，请使用此属性来减少编译时间。
allow_uav_condition	允许计算着色器循环终止条件基于 UAV 读取。 循环不得包含同步内部函数。

条件

条件 表达式。 如果表达式的计算结果为 true，则执行语句块。 当表达式的计算结果为 false 时，循环将结束。

语句块

一个或多个 语句。

另请参阅

Flow控件

函数 (HLSL 引用)

项目 • 2022/09/22

函数封装 HLSL 语句。 这使你可以调试一组函数，然后在着色器或效果之间重复使用它们。 你可能想要创建一个函数，该函数封装顶点着色器、像素着色器或纹理着色器的功能。 其他情况下，你可能想要编写一个执行一些常用任务的帮助程序函数，然后从着色器函数调用该帮助程序函数。 编写 HLSL 着色器函数的规则与编写 C 函数非常相似。

- 语法
- 参数
- Return 语句
- 签名

HLSL 还具有许多内置 **内部函数 (DirectX HLSL)** 。 由于所有内部函数都经过测试和性能优化，因此最好尽可能使用内部函数，而不是创建自己的函数。

相关主题

DirectX HLSL) (语言语法

函数声明语法

项目 • 2022/09/22

使用以下语法声明 HLSL 函数。

[*StorageClass*][clipplanes ()][精确]Return_Value *名称* ([*ArgumentList*]) [: *Semantic*] { [*StatementBlock*] };

参数

StorageClass

重新定义函数声明的修饰符。 **内联** 当前是唯一的修饰符值。 修饰符值必须 **内联** ，因为它也是默认值。 因此，无论是否指定 **内联**，HLSL 中的所有函数都是内联函数，都是内联函数。 内联函数在为每个函数调用编译) 时生成函数正文 (的副本。 这样做是为了降低调用函数的开销。

剪贴板

剪辑平面的可选列表，最多包含 6 个用户指定的剪辑平面。 这是一种替代机制，适用于功能级别9_x及更高级别的SV_ClipDistance。

名字

唯一标识着色器函数名称的 ASCII 字符串。

ArgumentList

可选参数列表，它是传入函数 的参数 的逗号分隔列表。

语义

用于标识返回数据的预期用法的可选字符串 (请参阅 语义 (DirectX HLSL)) 。

StatementBlock

构成函数主体的可选 语句 。 未定义正文的函数称为函数原型;必须先在其他地方定义原型函数的主体，然后才能调用该函数。

返回值

返回类型可以是其中任一 HLSL 类型。

注解

此页上的语法几乎描述了每种类型的 HLSL 函数，其中包括顶点着色器、像素着色器以及帮助程序函数。 虽然几何着色器还通过函数实现，但其语法稍微复杂一些，因此有一个单独的页面定义几何着色器函数声明，（请参阅 Geometry-Shader 对象 (DirectX HLSL) ）。

只要函数具有参数类型和/或参数顺序的唯一组合，就可以重载函数。 HLSL 还实现许多内置函数或 **内部函数**。

可以使用 **剪辑平面** 属性指定用户特定的剪辑平面。 Windows将这些剪辑平面应用于绘制的所有基元。 **剪辑平面**属性的工作方式类似于SV_ClipDistance，但适用于所有硬件功能级别9_x及更高版本。 有关详细信息，请参阅 功能级别 9 硬件上的用户剪辑平面。

示例

此示例来自 BasicHLSL10.fx 中的 BasicHLSL10 示例。

hlsl

```
struct VS_OUTPUT
{
    float4 Position   : SV_POSITION;
    float4 Diffuse    : COLOR0;
    float2 TextureUV  : TEXCOORD0;
};

VS_OUTPUT RenderSceneVS( float4 vPos : POSITION,
                         float3 vNormal : NORMAL,
                         float2 vTexCoord0 : TEXCOORD,
                         uniform int nNumLights,
                         uniform bool bTexture,
                         uniform bool bAnimate )
{
    VS_OUTPUT Output;
    ...
    return Output;
}
```

来自 AdvancedParticles.fx 的 AdvancedParticles 示例的示例演示了使用返回类型的语义。

hlsl

```
//
// PS for particles
//
```

```
float4 PSPointSprite(PSSceneIn input) : SV_Target
{
    return g_txDiffuse.Sample( g_samLinear, input.tex ) * input.color;
}
```

相关主题

函数 (DirectX HLSL)

函数参数

项目 • 2022/09/23

函数采用一个或多个输入参数;使用以下语法声明每个参数。

[InputModifier]类型名称 [： 语义] [InterpolationModifier] [= Initializers]

[修饰符]类型名称 [： 语义] [： 内插修饰符] [= 初始值设定项 (s)]

如果有多个函数参数，则用逗号分隔它们。

参数

项	说明
InputModifier	可选术语，用于将参数标识为输入、输出或两者。 值 — 说明 in — 仅输入 inout — 输入和输出 out — 仅输出 uniform — 仅输入常量数据 参数始终按值传递。 in 指示在函数开始之前应从调用应用程序复制参数的值。 out 指示应复制参数的最后一个值，并在函数返回时返回到调用应用程序。 inout 是用于指定这两者的简写。 统一值来自常量寄存器;每个顶点着色器或像素着色器调用都看到统一变量的相同初始值。 全局变量被视为声明统一的变量。 对于非顶级函数，统一与 in 同义词。 如果未指定任何参数用法，则假定参数用法处于 **中**。
类型	参数类型;可以是任何有效的 HLSL 类型。
名字	唯一标识着色器函数名称的 ASCII 字符串。
语义	用于标识数据 (预期用途的可选字符串，请参阅 DirectX HLSL)) (语义 。
InterpolationModifier	可选的 内插修饰符 ，允许着色器确定内插方法。 函数参数上的内插修饰符仅适用于用作像素着色器函数输入的参数。
初始值设定项	初始化的可选值;初始化多组件数据类型需要多个值。

备注

函数参数列在函数声明中逗号分隔的参数列表中。 与 C 函数一样，每个参数都必须声明参数名称和类型;HLSL 函数的参数（可选）可以包含语义、初始值和像素着色器输入可以包括内插类型。

函数参数 *的类型* 可以是一个结构，该结构可能包括每成员内插修饰符。 如果函数参数还具有内插修饰符，函数参数修饰符将替代在 Type 中声明的内插修饰符。

示例

此示例 (从 BasicHLSL10 示例) 演示顶点着色器函数的统一和非统一输入。

```
VS_OUTPUT RenderSceneVS(
  float4 vPos : POSITION,
  float3 vNormal : NORMAL,
  float2 vTexCoord0 : TEXCOORD,
  uniform int nNumLights,
  uniform bool bTexture,
  uniform bool bAnimate )
{
  ...
}
```

此示例 (内容流示例) 使用输入结构将参数传递给像素着色器函数。

```
VSBasicIn input
struct VSBasicIn
{
  float4 Pos    : POSITION;
  float3 Norm   : NORMAL;
  float2 Tex    : TEXCOORD0;
};

PSBasicIn VSBasic(VSBasicIn input)
{
  ...
}
```

相关主题

DirectX HLSL) 函数 (

return 语句

项目 • 2022/09/23

return 语句向函数的末尾发出信号。

返回 [value];

最简单的 return 语句将函数中的控件返回到调用程序;它不返回任何值。

```
void main()
{
    return ;
}
```

但是，return 语句可以返回一个或多个值。 此示例返回文本值：

```
float main( float input : COLOR0) : COLOR0
{
    return 0;
}
```

此示例返回表达式的标量结果。

```
return  light.enabled = true ;
```

此示例返回从局部变量和文本构造的四分量向量。

```
return  float4(color.rgb, 1) ;
```

此示例返回一个四分量向量，该向量由从内部函数返回的结果以及文本值一起构造。

```
float4 func(float2 a: POSITION): COLOR
{
```

```
    return float4(sin(length(a) * 100.0) * 0.5 + 0.5, sin(a.y * 50.0), 0,
1);
}
```

此示例返回包含一个或多个成员的结构。

```

float4x4 WorldViewProj;

struct VS_OUTPUT
{
    float4 Pos  : POSITION;
};

VS_OUTPUT VertexShader_Tutorial_1(float4 inPos : POSITION )
{
    VS_OUTPUT out;
    out.Pos = mul(inPos, WorldViewProj );
    return out;
};
```

另请参阅

函数 (DirectX HLSL)

签名

项目 • 2022/09/23

着色器签名是着色器函数的输入或输出的参数列表。 在 Direct3D 10 中，相邻阶段有效地共享寄存器数组，其中输出着色器 (或管道阶段) 将数据写入寄存器数组中的特定位置，并且输入着色器必须从同一位置读取数据。 API 使用着色器签名将着色器输出与输入绑定，而无需语义解析开销。

在 Direct3D 10 中，输入签名是从着色器输入声明生成的，输出签名是从着色器输出声明生成的。 当输出签名是输入签名的严格子集 (参数类型和顺序匹配) 输入签名时，据说输入签名与输出签名兼容。 实现此目的的最简单方法是通过同一结构类型链接相应的着色器输入和输出。

下面是兼容签名的示例。

```
// Vertex Shader Output Signature
Struct VSOut
{
  float4 Pos: SV_Position;
  float3 MyNormal: Normal;
  float2 MyTex : Texcoord0;
}

// Pixel Shader Input Signature
Struct PSInWorks
{
  float4 Pos: SV_Position;
  float3 MyNormal: Normal;
}
```

下面是不兼容签名的示例;输入签名中的参数顺序与输出签名中的顺序不匹配。

```
// Vertex Shader Output Signature
Struct VSOut
{
  float4 Pos: SV_Position;
  float3 MyNormal: Normal;
  float2 MyTex : Texcoord0;
}

// Pixel Shader Input Signature
Struct PSInFails
{
  float3 MyNormal: Normal;
```

```
  float4 Pos: SV_Position;
}
```

PSInWorks 是 VSOut 的兼容子集， (前两个条目与 VSOut) 中的前两个条目匹配类型和顺序。 但是，PSInFails 不兼容，因为排序与 VSOut 不匹配。

相关主题

函数

语句

项目 • 2022/09/22

HLSL 语句是构成表达式的变量和函数的组合;与句子一样，HLSL 语句要求运算符确定表达式的计算方式。 可以在着色器中创建单个语句，也可以在名为语句块的组中创建多个语句。 若要创建 HLSL 语句，需要知道用于编写表达式的语法，以及用于合并表达式的运算符。

- 表达式
- 运算符
- 语句块

相关主题

DirectX HLSL (语言语法)

表达式

项目 • 2022/09/23

表达式是由运算符标点的变量和文本序列。 文本是一个显式数据值，例如整数的 1 或浮点数的 2.1。 文本通常用于向变量赋值。

后跟分号 (;) 的表达式称为语句。 语句的复杂性范围从简单表达式到完成一系列操作的语句块。 Flow控制语句确定执行 order 语句。

语句块还指示子范围。 语句块中声明的变量仅在块内识别。 HLSL 语句确定计算表达式的顺序。 每个表达式可以是下列表达式之一。

- 表达式
- 语句块
- return 语句
- 流控制语句

相关主题

语句 (DirectX HLSL)

运算符

项目 • 2022/09/22

表达式是由运算符标点的变量和文本序列。 运算符确定变量和文本如何组合、比较、选定等。 运算符包括：

操作员名称	运算符
累加运算符和乘法运算符	+, -, *, /, %
数组运算符	[i]
赋值运算符	=, +=, -=, *=, /=, %=
二进制转换	bool 的 float 和 int、C 规则或 HLSL 内部函数的 C 规则
位运算符	~, <<, >>, &, |, ^, <<=, >>=, &=, |=, ^=
布尔数学运算符	&&, ||, ?:
转换运算符	(类型)
逗号运算符	,
比较运算符	<, >, ==, !=, <=, >=
前缀或后缀运算符	++, --
结构运算符	.
一元运算符	!, -, +

许多运算符都是每个分量，这意味着操作是针对每个变量的每个组件独立执行的。 例如，单个组件变量执行了一个操作。 另一方面，四分量变量执行了四个操作，每个组件各执行一个操作。

对值执行某些操作的所有运算符（例如 + 和 * ）按组件工作。

比较运算符需要单个组件才能工作，除非对多组件变量使用 **所有** 函数 **或任何** 内部函数。 以下操作失败，因为 if 语句需要单个布尔值，但收到 bool4：

```
if (A4 < B4)
```

以下操作成功：

```
if ( any(A4 < B4) )
if ( all(A4 < B4) )
```

二进制强制转换运算符 asfloat、 asint 等每个组件的工作，除了记录其特殊规则的 **双倍** 。

选择运算符（如句点、逗号和数组括号）不能按组件工作。

强制转换运算符更改组件数。 以下强制转换操作显示其等效性：

```
(float) i4 ->   float(i4.x)
(float4)i ->   float4(i, i, i, i)
```

累加运算符和乘法运算符

累加运算符和乘法运算符为：+、-、*、/、%

```
int i1 = 1;
int i2 = 2;
int i3 = i1 + i2;  // i3 = 3
i3 = i1 * i2;        // i3 = 1 * 2 = 2
```

```
i3 = i1/i2;       // i3 = 1/3 = 0.333. Truncated to 0 because i3 is an
integer.
i3 = i2/i1;       // i3 = 2/1 = 2
```

```
float f1 = 1.0;
float f2 = 2.0f;
float f3 = f1 - f2; // f3 = 1.0 - 2.0 = -1.0
f3 = f1 * f2;         // f3 = 1.0 * 2.0 = 2.0
```

```
f3 = f1/f2;         // f3 = 1.0/2.0 = 0.5
f3 = f2/f1;         // f3 = 2.0/1.0 = 2.0
```

模数运算符返回除法的余数。 当使用整数和浮点数时，这会产生不同的结果。 将截断小数部分的整数余数。

```
int i1 = 1;
int i2 = 2;
i3 = i1 % i2;       // i3 = remainder of 1/2, which is 1
i3 = i2 % i1;       // i3 = remainder of 2/1, which is 0
i3 = 5 % 2;         // i3 = remainder of 5/2, which is 1
i3 = 9 % 2;         // i3 = remainder of 9/2, which is 1
```

使用整数时，取模运算符截断小数余数。

```
f3 = f1 % f2;       // f3 = remainder of 1.0/2.0, which is 0.5
f3 = f2 % f1;       // f3 = remainder of 2.0/1.0, which is 0.0
```

仅当双方为正或双方为负的情况下，才定义 % 运算符。 与 C 不同，它还对浮点数据类型以及整数进行操作。

数组运算符

数组成员选择运算符“[i]”选择数组中的一个或多个组件。 它是一组包含从零开始的索引的方括号。

```
int arrayOfInts[4] = { 0,1,2,3 };
arrayOfInts[0] = 2;
arrayOfInts[1] = arrayOfInts[0];
```

数组运算符还可用于访问向量。

```
float4 4D_Vector = { 0.0f, 1.0f, 2.0f, 3.0f  };
float 1DFloat = 4D_Vector[1];          // 1.0f
```

通过添加其他索引，数组运算符还可以访问矩阵。

```
float4x4 mat4x4 = {{0,0,0,0}, {1,1,1,1}, {2,2,2,2}, {3,3,3,3} };
mat4x4[0][1] = 1.1f;
float 1DFloat = mat4x4[0][1];      // 0.0f
```

第一个索引是从零开始的行索引。 第二个索引是从零开始的列索引。

赋值运算符

赋值运算符包括：=、+=、-=、*=、/=

可以为变量赋值文本值：

```
int i = 1;
float f2 = 3.1f;
bool b = false;
string str = "string";
```

还可以为变量分配数学运算的结果：

```
int i1 = 1;
i1 += 2;           // i1 = 1 + 2 = 3
```

可以在等号的任一端使用变量：

```
float f3 = 0.5f;
f3 *= f3;          // f3 = 0.5 * 0.5 = 0.25
```

浮点变量的除法按预期计算，因为小数余数不是问题。

```
float f1 = 1.0;
f1 /= 3.0f;        // f1 = 1.0/3.0 = 0.333
```

如果使用可能会被除的整数，请小心，尤其是在截断影响结果时。 此示例与上一个示例相同，但数据类型除外。 截断会导致一个非常不同的结果。

```
int i1 = 1;
i1 /= 3;           // i1 = 1/3 = 0.333, which gets truncated to 0
```

二进制转换

int 和 float 之间的强制转换操作会将数值转换为以下 C 规则中的相应表示形式，以截断 int 类型。 将值从浮点转换为 int，然后转换回浮点将导致基于目标的精度进行丢失转换。

也可以使用 **内部函数 (DirectX HLSL)** 执行二进制转换，后者将数字的位表示形式重新解释为目标数据类型。

```
asfloat() // Cast to float
asint()   // Cast to int
asuint()  // Cast to uint
```

按位运算符

HLSL 支持以下按位运算符，该运算符与其他运算符遵循与 C 相同的优先级。 下表描述了运算符。

> ⓘ **备注**
>
> 按位运算符需要具有 Direct3D 10 及更高硬件的 **着色器模型** 4_0 。

运算符	函数
~	逻辑非
<<	左移
>>	右移
&	逻辑与
|	逻辑或

运算符	函数
^	逻辑 Xor
<<=	左移相等
>>=	右移相等
&=	相等
|=	或等于
^=	Xor 等于

按位运算符定义为仅对 int 和 uint 数据类型进行操作。 尝试对浮点或结构数据类型使用按位运算符将导致错误。

布尔数学运算符

布尔数学运算符包括： &&、||、 ？ ：

```
bool b1 = true;
bool b2 = false;
bool b3 = b1 && b2 // b3 = true AND false = false
b3 = b1 || b2                 // b3 = true OR false = true
```

与||和 ？的 &&短路计算不同：在 C 中，HLSL 表达式永远不会对计算进行短路，因为它们是矢量操作。 始终计算表达式的所有方面。

布尔运算符基于每个组件函数。 这意味着，如果比较两个向量，则结果为包含每个组件的比较布尔结果的向量。

对于使用布尔运算符的表达式，在操作发生之前，每个变量的大小和组件类型将提升为相同。 提升的类型确定操作的发生分辨率以及表达式的结果类型。 例如，int3 + float 表达式将提升为 float3 + float3 进行计算，其结果为 float3 类型。

转换运算符

括号中类型名称前面的表达式是显式类型强制转换。 类型强制转换将原始表达式转换为强制转换的数据类型。 一般情况下，可以使用提升强制转换) 将简单数据类型转换为更复杂的数据类型 (，但只有一些复杂数据类型可以转换为简单数据类型， (降级强制转换) 。

只有右侧类型铸造是合法的。 例如，表达式（例如 `(int)myFloat = myInt;` 是非法的）。 请改用 `myFloat = (float)myInt;`。

编译器还会执行隐式类型强制转换。 例如，以下两个表达式等效：

```
int2 b = int2(1,2) + 2;
int2 b = int2(1,2) + int2(2,2);
```

逗号运算符

逗号运算符（，）分隔要按顺序计算的一个或多个表达式。 序列中最后一个表达式的值用作序列的值。

下面是值得关注的一个案例。 如果构造函数类型意外离开等号右侧，则右侧现在包含四个表达式，用三个逗号分隔。

```
// Instead of using a constructor
float4 x = float4(0,0,0,1);

// The type on the right side is accidentally left off
float4 x = (0,0,0,1);
```

逗号运算符计算从左到右的表达式。 这会将右侧减少为：

```
float4 x = 1;
```

在这种情况下，HLSL 使用标量提升，因此结果如下所示：

```
float4 x = float4(1,1,1,1);
```

在此实例中，从右侧离开 float4 类型可能是编译器无法检测到的错误，因为这是有效的语句。

比较运算符

比较运算符包括： <、 、 = >=、 ！=、 <=、 >=。

比较大于 (或小于) 任何标量值的值：

```
if( dot(lightDirection, normalVector) > 0 )
   // Do something; the face is lit
```

```
if( dot(lightDirection, normalVector) < 0 )
   // Do nothing; the face is backwards
```

或者，比较等于 (或不等于) 任何标量值的值：

```
if(color.a == 0)
   // Skip processing because the face is invisible

if(color.a != 0)
   // Blend two colors together using the alpha value
```

或者合并并比较大于或等于 (或小于或等于) 任何标量值的值：

```
if( position.z >= oldPosition.z )
   // Skip the new face because it is behind the existing face
```

```
if( currentValue <= someInitialCondition )
   // Reset the current value to its initial condition
```

每个比较都可以使用任何标量数据类型完成。

若要对矢量和矩阵类型使用比较运算符，请使用**所有或任何**内部函数。

此操作失败，因为 if 语句需要单个 bool，但接收 bool4：

```
if (A4 < B4)
```

这些操作成功：

```
if ( any(A4 < B4) )
if ( all(A4 < B4) )
```

前缀或后缀运算符

前缀和后缀运算符包括：++、--. 前缀运算符在计算表达式之前更改变量的内容。 后缀运算符在计算表达式后更改变量的内容。

在这种情况下，循环使用 i 的内容跟踪循环计数。

```
float4 arrayOfFloats[4] = { 1.0f, 2.0f, 3.0f, 4.4f };

for (int i = 0; i<4; )
{
    arrayOfFloats[i++] *= 2;
}
```

由于使用后缀递增运算符 (++)，因此在递增 i 之前，arrayOfFloats[i] 乘以 2。 这可能会稍微重新排列，以使用前缀递增运算符。 虽然这两个示例都是等效的，但很难阅读。

```
float4 arrayOfFloats[4] = { 1.0f, 2.0f, 3.0f, 4.4f };

for (int i = 0; i<4; )
{
    arrayOfFloats[++i - 1] *= 2;
}
```

由于使用前缀运算符 (++)，因此在递增 i 后，arrayOfFloats[i+1 - 1] 乘以 2。

前缀递减和后缀递减运算符 (--) 按与递增运算符相同的顺序应用。 区别在于减去 1，而不是加 1。

结构运算符

结构成员选择运算符 (.) 是句点。 鉴于此结构：

```
struct position
{
float4 x;
float4 y;
float4 z;
};
```

可以如下所示阅读：

```
struct position pos = { 1,2,3 };

float 1D_Float = pos.x
1D_Float = pos.y
```

可以使用结构运算符读取或写入每个成员：

```
struct position pos = { 1,2,3 };
pos.x = 2.0f;
pos.z = 1.0f;       // z = 1.0f
pos.z = pos.x      // z = 2.0f
```

一元运算符

一元运算符为：!, -, +

一元运算符对单个操作数进行操作。

```
bool b = false;
bool b2 = !b;      // b2 = true
int i = 2;
int i2 = -i;       // i2 = -2
int j = +i2;       // j = +2
```

运算符优先级

当表达式包含多个运算符时，运算符优先级确定计算顺序。 HLSL 的运算符优先级遵循与 C 相同的优先级。

注解

大括号 () {;} 开始和结束语句块。 当语句块使用单个语句时，大括号是可选的。

相关主题

语句 (DirectX HLSL)

语句块

项目 • 2022/09/22

语句块包含一个或多个语句;每个语句由 表达式 和 运算符组成。

```
{
   statement 1;
   statement 2;
   ...
   statement n;
}
```

语句块还指示子范围。 语句块中声明的变量只能在块内识别。

相关主题

语句 (DirectX HLSL)

语义

项目 • 2022/09/22

语义是附加到着色器输入或输出的字符串，用于传达有关参数预期使用的信息。 着色器阶段之间传递的所有变量都需要语义。 此处显示了向着色器变量添加语义的语法，（变量语法 (DirectX HLSL) ）。

通常，管道阶段之间传递的数据是完全通用的，并且不是系统唯一解释的：允许具有特殊含义的任意语义。 Direct3D 10 及更高版本中 (的参数) 包含这些特殊语义的参数称为 系统值语义。

Direct3D 9 和 Direct3D 10 及更高版本中支持的语义

Direct3D 9 和 Direct3D 10 及更高版本中都支持以下类型的语义。

- 顶点着色器语义
- 像素着色器语义

顶点着色器语义

当附加到顶点着色器参数时，这些语义具有意义。 Direct3D 9 和 Direct3D 10 及更高版本中都支持这些语义。

输入	说明	类型
BINORMAL[n]	二进制	float4
BLENDINDICES[n]	混合索引	uint
BLENDWEIGHT[n]	混合权重	FLOAT
COLOR[n]	漫射和反射颜色	float4
NORMAL[n]	普通向量	float4
POSITION[n]	对象空间中的顶点位置。	float4
POSITIONT	转换的顶点位置。	float4
PSIZE[n]	点大小	FLOAT
TANGENT[n]	正切	float4

输入	说明	类型
TEXCOORD[n]	纹理坐标	float4

输出	说明	类型
COLOR[n]	漫射或反射颜色	float4
雾	顶点雾	FLOAT
POSITION[n]	顶点在同质空间中的位置。 通过将 (x，y，z) 除以 w，在屏幕空间中计算位置。 每个顶点着色器都必须写出具有此语义的参数。	float4
PSIZE	点大小	FLOAT
TESSFACTOR[n]	分割因子	FLOAT

`n` 是介于 0 和支持的资源数之间的可选整数。 例如，POSITION0、TEXCOORD1 等。

像素着色器语义

当附加到像素着色器输入参数时，这些语义具有意义。 Direct3D 9 和 Direct3D 10 及更高版本中都支持这些语义。

输入	说明	类型
COLOR[n]	漫射或反射颜色。	float4
TEXCOORD[n]	纹理坐标	float4
VFACE	指示面向背的基元的浮点标量。 负值向后面，而正值则面对相机。 [！注意] Direct3D 9 着色器模型 3.0 中提供了此语义。 对于 Direct3D 10 及更高版本，请改用 SV_IsFrontFace 。	FLOAT
VPOS	屏幕空间中的像素位置 (x，y)。 若要将此语义 () 的 Direct3D 9 着色器转换为 Direct3D 10 及更高版本的着色器，请参阅 Direct3D 9 VPOS 和 Direct3D 10 SV_Position)	float2

输出	说明	类型
COLOR[n]	输出颜色	float4
DEPTH[n]	输出深度	FLOAT

`n` 是介于 0 和支持的资源数之间的可选整数。 例如，PSIZE0、COLOR1 等。

COLOR 语义仅在着色器兼容性模式下有效， (即使用 D3D10_SHADER_ENABLE_BACKWARDS_COMPATIBILITY) 创建着色器时。

仅支持 Direct3D 10 和更新的语义。

为 Direct3D 10 引入了以下类型的语义，不适用于 Direct3D 9。

- 系统值语义

System-Value语义

系统值语义是 Direct3D 10 的新增功能。 所有系统值都以SV_前缀开头，一个常见示例是 SV_POSITION，该示例由光栅器阶段解释。 系统值在管道的其他部分有效。 例如，可以将SV_Position指定为顶点着色器和输出的输入。 像素着色器只能写入具有SV_Depth和 SV_Target系统值语义的参数。

其他系统值 (SV_VertexID，SV_InstanceID，SV_IsFrontFace) 只能输入到管道中的第一个活动着色器中，这些着色器可以解释特定值：之后，着色器函数必须将值传递给后续阶段。

SV_PrimitiveID是此规则的例外，即仅输入管道中的第一个活动着色器，该着色器可以解释特定值;硬件可以提供与船体着色器阶段、域着色器阶段的输入相同的 ID 值，并在该阶段之后启用哪个阶段：几何着色器阶段或像素着色器阶段。

如果启用了分割，则存在外壳着色器阶段和域着色器阶段。 对于给定的修补程序，相同的 PrimitiveID 适用于修补程序的外壳着色器调用，以及所有细化域着色器调用。 相同的 PrimitiveID 还会传播到下一个活动阶段：如果已启用 geometry-shader 阶段或像素着色器阶段。

如果几何着色器输入SV_PrimitiveID，并且因为它可以为每个调用输出零个或多个基元输出，则着色器必须为每个输出基元编程自己的SV_PrimitiveID值（如果后续像素着色器输入SV_PrimtiveID）。

作为另一个示例，顶点着色器阶段无法解释SV_PrimitiveID，因为顶点可以是多个基元的成员。

这些语义已添加到 Direct3D 10;它们在 Direct3D 9 中不可用。

光栅器阶段的系统值语义。

System-Value语义	说明	类型

System-Value语义	说明	类型
SV_ClipDistance[n]	剪辑距离数据。 SV_ClipDistance值都假定为平面的 float32 带符号距离。 基元设置仅对内插平面距离 () >为 0 的像素调用光栅化。 通过将一个或多个顶点元素的多个组件声明为 SV_ClipDistance，可以同时实现多个剪辑平面 ()。 组合的剪辑和剔除距离值最多是 D3D#_CLIP_OR_CULL_DISTANCE_COUNT 组件，大多数为 D3D#_CLIP_OR_CULL_DISTANCE_ELEMENT_COUNT 寄存器。 可供读取或写入的所有着色器使用，但顶点着色器可以写入值，但不能将其用作输入。 **剪辑平面**属性的工作方式类似于SV_ClipDistance，但适用于所有硬件功能级别9_x及更高版本。 有关详细信息，请参阅功能级别 9 硬件上的用户剪辑平面。	FLOAT
SV_CullDistance[n]	剔除距离数据。 如果为 (顶点元素 () 的组件），则每个值都假定为平面的 float32 带符号距离。 如果基元中的所有顶点的平面距离 () 为 < 0，则完全放弃基元。 可以通过将一个或多个顶点元素的多个组件 () 声明为SV_CullDistance来同时使用多个布尔平面。 组合的剪辑和剔除距离值最多是 D3D#_CLIP_OR_CULL_DISTANCE_COUNT 组件，大多数为 D3D#_CLIP_OR_CULL_DISTANCE_ELEMENT_COUNT 寄存器。 可供读取或写入的所有着色器使用，但顶点着色器可以写入值，但不能将其用作输入。	FLOAT
SV_Coverage	可以在输入、输出或两个像素着色器上指定的掩码。 对于像素着色器上的SV_Coverage，ps_4_1或更高版本支持 OUTPUT。 对于像素着色器上的SV_Coverage，INPUT 需要ps_5_0或更高版本。	uint
SV_Depth	深度缓冲区数据。 可以通过像素着色器编写。	FLOAT
SV_DepthGreaterEqual	在像素着色器中，只要深度大于或等于光栅器确定的值，就可以输出深度。 启用调整深度而不禁用早期 Z。	FLOAT
SV_DepthLessEqual	在像素着色器中，只要深度小于或等于光栅器确定的值，就可以输出深度。 启用调整深度而不禁用早期 Z。	FLOAT
SV_DispatchThreadID	定义组的每个维度的 Dispatch 调用中的全局线程偏移量。 可用作计算着色器的输入。 (只读)	uint3
SV_DomainLocation	定义正在计算的当前域点外壳上的位置。 可用作域着色器的输入。 (只读)	float2|3
SV_GroupID	定义 Dispatch 调用中的组偏移量，每个调度调用的维度。 可用作计算着色器的输入。 (只读)	uint3
SV_GroupIndex	为给定组中的给定线程提供平展索引。 可用作计算着色器的输入。 (只读)	uint

System-Value语义	说明	类型
SV_GroupThreadID	定义组中每个维度的线程偏移量。 可用作计算着色器的输入。 (只读)	uint3
SV_GSInstanceID	定义几何着色器的实例。 可用作几何着色器的输入。 需要实例，因为同一几何图形基元上最多可以调用几何着色器 32 次。	uint
SV_InnerCoverage	表示 (低估的保守光栅化信息，即是否保证像素完全覆盖) 。 可以由像素着色器读取或写入。	
SV_InsideTessFactor	定义修补图面中的细化量。 在外壳着色器中可用于写入，并在域着色器中可用于读取。	float|float[2]
SV_InstanceID	运行时自动生成的每个实例标识符 (请参阅 使用 Direct3D 1) 0) direct3D 10 System-Generated (值 。 适用于所有着色器。	
SV_IsFrontFace	指定三角形是否正面。 对于行和点，IsFrontFace 的值为 true。 例外是 (线框模式) 绘制出三角形的线条，该模式设置 IsFrontFace 的方式与在实心模式下光栅化三角形的方式相同。 可以由几何着色器写入，并由像素着色器读取。	bool
SV_OutputControlPointID	定义通过调用外壳着色器的主要入口点来操作的控制点 ID 的索引。 只能由外壳着色器读取。	uint
SV_Position	当为着色器输入声明SV_Position时，它可以指定两种内插模式之一：linearNoPerspective 或 linearNoPerspectiveCentroid，后者会导致在多重采样抗锯齿时提供质心贴靠 xyzw 值。 在着色器中使用时，SV_Position 描述像素位置。 在所有着色器中可用，以获取具有 0.5 偏移量的像素中心。	float4
SV_PrimitiveID	运行时自动生成的每基元标识符 (请参阅 使用 direct3D 1) 0) 使用 System-Generated 值 (Direct3D 10 。 可以由几何图形或像素着色器写入，并由几何图形、像素、外壳或域着色器读取。	uint
SV_RenderTargetArrayIndex	呈现目标数组索引。 应用于几何着色器输出，并指示基元将由像素着色器绘制到的呈现目标数组切片。 仅当呈现目标为数组资源时，SV_RenderTargetArrayIndex才有效。 此语义仅适用于基元;如果基元有多个顶点，则使用前导顶点中的值。 此值还指示深度/模具视图的数组切片用于读/写。 可以从几何着色器写入，并由像素着色器读取。 如果为 D3D11_FEATURE_DATA_D3D11_OPTIONS3：：VPAndRTArrayIndexFromAnyShaderFeedingRasterizer `true`，则SV_RenderTargetArrayIndex应用于馈送光栅器的任何着色器。	uint
SV_SampleIndex	采样频率索引数据。 只能由像素着色器读取或写入。	uint

System-Value语义	说明	类型
SV_StencilRef	表示当前像素着色器模具引用值。 只能由像素着色器编写。	uint
SV_Target[n]，其中 0 <= n <= 7	将存储在呈现目标的输出值。 索引指示要写入的 8 个可能绑定的呈现目标中的哪一个。 该值适用于所有着色器。	float[2|3|4]
SV_TessFactor	定义补丁的每个边缘上的分割量。 可用于在外壳着色器中写入，并在域着色器中读取。	float[2|3|4]
SV_VertexID	运行时自动生成的每个顶点标识符 (请参阅 使用 Direct3D 10 (Direct3D 10)) System-Generated值 。 仅作为顶点着色器的输入提供。	uint
SV_ViewportArrayIndex	视区数组索引。 应用于几何图形着色器输出，并指示当前要写入的基元使用的视区。可以由像素着色器读取。 在将基元传递给光栅器之前，将针对索引指定的视区进行转换和剪裁。 此语义仅适用于基元;如果基元有多个顶点，则使用前导顶点的值。 如果D3D11_FEATURE_DATA_D3D11_OPTIONS3：：VPAndRTArrayIndexFromAnyShaderFeedingRasterizer`true`，则SV_ViewportArrayIndex应用于为光栅器馈送的任何着色器。	uint
SV_ShadingRate	通过底纹速率 值定义由一个像素着色器调用为 可变着色速率第 2 层或更高版本设备写入的像素数。 可以从像素着色器读取。 可以从顶点或几何图形着色器编写。	uint

编写SV_Depth时的限制：

- 当多重采样 (MultisampleEnable 在**D3D10_RASTERIZER_DESC)** 为 **TRUE**，并使用像素着色器) 写入深度值 (时，也会在深度测试中使用单个值：因此，当多重采样时，在更高的分辨率下呈现基元边缘的能力会丢失。
- 使用动态流控制时，在编译时无法确定是否 SV_Depth保证在每次执行中写入 SV_Depth SV_Depth的着色器。 声明后无法写入SV_Depth会导致未定义的行为（，这可能或可能包括放弃像素）。
- 任何 float32 值（包括 +/-INF 和 NaN）都可以写入SV_Depth。
- 执行双源颜色混合时，编写SV_Depth仍然有效。

从 Direct3D 9 迁移到 Direct3D 10 及更高版本

将代码从 Direct3D 9 迁移到 Direct3D 10 及更高版本时，应考虑以下问题：

映射到 Direct3D 9 语义

一些 Direct3D 10 及更高版本的语义直接映射到 Direct3D 9 语义。

Direct3D 10 语义	Direct3D 9 等效语义
SV_Depth	DEPTH
SV_Position	POSITION
SV_Target	COLOR

> [!]请注意 Direct3D 9 开发人员：对于 Direct3D 9 目标，着色器语义必须映射到有效的 Direct3D 9 语义。 为了向后兼容 POSITION0 (及其变体名称) 被视为SV_Position，COLOR 将被视为SV_TARGET。

- 映射到 Direct3D 9 语义
- Direct3D 9 VPOS 和 Direct3D 10 SV_Position
- HLSL 中的用户剪辑平面

Direct3D 9 VPOS 和 Direct3D 10 SV_Position

D3D10 语义SV_Position提供与 Direct3D 9 着色器模型 3 VPOS 语义类似的功能。 例如，在 Direct3D 9 中，以下语法用于使用屏幕空间坐标的像素着色器：

```
HLSL

float4 psMainD3D9( float4 screenSpace : VPOS ) : COLOR
{
    // code here
}
```

为着色器模型 3 支持添加了 VPOS，以指定屏幕空间坐标，因为 POSITION 语义适用于对象空间坐标。

在 Direct3D 10 及更高版本中，在像素着色器) 上下文中使用的SV_Position语义 (指定屏幕空间坐标 (偏移量为 0.5) 。 因此，Direct3D 9 着色器大致等效 (，而不考虑以下 0.5 偏移量) ：

```
HLSL

float4 psMainD3D10( float4 screenSpace : SV_Position ) : COLOR
{
    // code here
}
```

从 Direct3D 9 迁移到 Direct3D 10 及更高版本时，在转换着色器时需要注意这一点。

HLSL 中的用户剪辑平面

从Windows 8开始，可以在 HLSL 函数声明中使用**剪辑平面**函数属性，而不是 SV_ClipDistance使着色器在功能级别9_x以及功能级别 10 及更高。 有关详细信息，请参阅 功能级别 9 硬件上的用户剪辑平面。

相关主题

- 语言语法
- (DirectX HLSL) 变量

SV_DispatchThreadID

项目 • 2022/09/22

计算着色器正在在其中执行合并线程和线程组的索引。 SV_DispatchThreadID是 SV_GroupID * numthreads 和 GroupThreadID 的总和。 它因 **Dispatch** 和 numthreads 中指定的范围而异。 例如，如果在计算着色器上调用 Dispatch (2，2，2) ，该着色器 (3，3，3) SV_DispatchThreadID的每个维度的范围为 0..5。

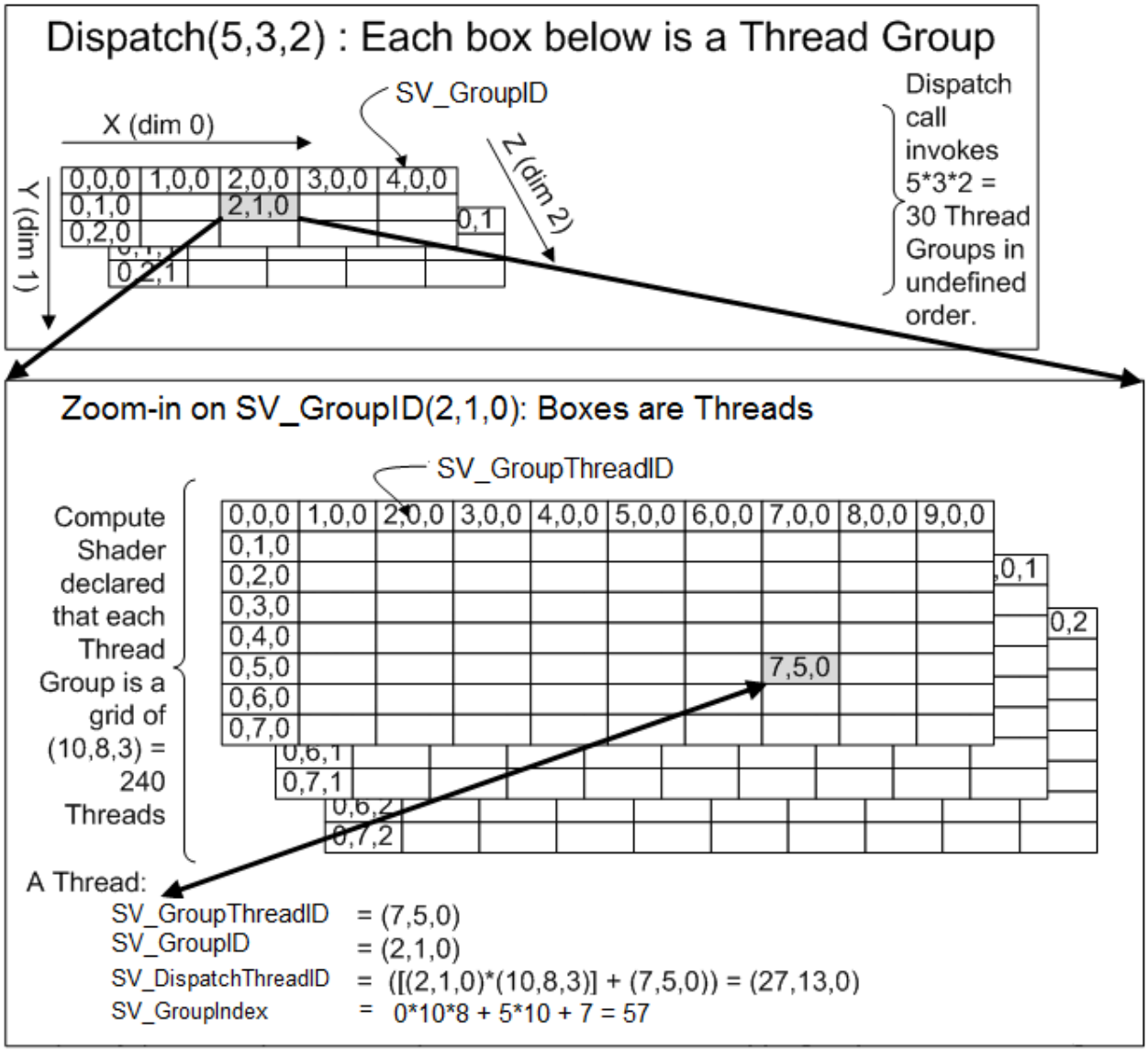
类型

类型
uint3

注解

此系统值是可选的。

下图显示了传递给 **Dispatch** 的参数之间的关系， 调度 (5，3，2) 、 numthreads 属性中指定的值、numthreads (10，8，3) ，以及将传递给线程相关系统值的计算着色器的值 (SV_GroupIndex，SV_DispatchThreadID，SV_GroupThreadID，SV_GroupID) 。



以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
					x

另请参阅

语义

着色器模型 5

SV_DomainLocation

项目 • 2022/09/23

定义正在计算的当前域点外壳上的位置。

类型

类型	输入拓扑
float2	quad patch
float3	tri patch
float2	等值 线

注解

此系统值是必需的。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
		x			

另请参阅

语义

着色器模型 5

SV_TessFactor

项目 • 2022/09/22

定义补丁的每个边缘上的分割量。

类型

类型	输入拓扑
float[4]	quad patch
float[3]	tri patch
float[2]	等值 线

分割因子必须声明为数组;它们不能打包到单个向量中。

注解

必须在外壳着色器的修补常量函数期间定义细化因子的值。

如果使用象限或三个修补程序，则外壳着色器所需的输出值。 此值也是域着色器在分割阶段之间匹配修补常量数据签名所需的输入值。

对于 isoline，SV_TessFactor中的第一个值是折线密度细化因子，第二个值是折线细化因子。

三个补丁分割因子

第一个组件为修补程序的 u==0 边缘提供细化因子。 第二个组件为修补程序的 v==0 边缘提供细化因子。 第三个组件为修补程序的 w==0 边缘提供细化因子。

四次修补分割因子

第一个组件为修补程序的 u==0 边缘提供细化因子。 第二个组件为修补程序的 v==0 边缘提供细化因子。 第三个组件为修补程序的 u==1 边缘提供细化因子。 第四个组件为修补程序的 v==1 边缘提供细化因子。 边缘的排序是顺时针的，从 u==0 边缘开始，即修补程序左侧，从 v==0 边缘开始，即补丁的顶部。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x	x			

另请参阅

语义

着色器模型 5

SV_GroupID

项目 • 2022/09/24

正在对其执行计算着色器的线程组的索引。 索引是整个组而不是单个线程。 可能的值因作为参数传递给 **Dispatch** 的范围而异。 例如，调用 Dispatch (2，1，1) 会导致可能的值为 0，0，0 和 1，0，0。

定义 **Dispatch** 调用中的组偏移量，每个调度调用的维度。

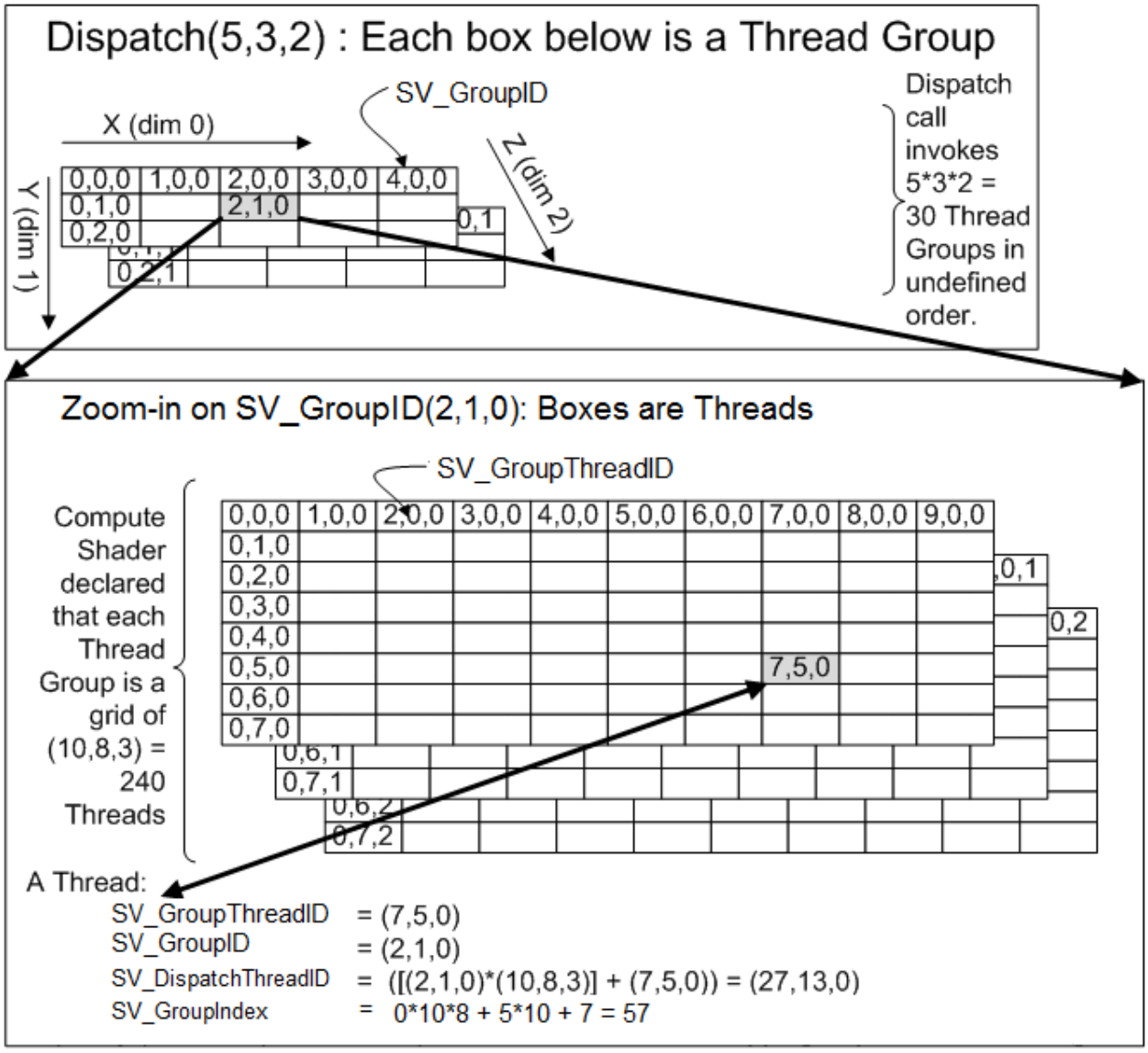
类型

类型
uint3

注解

此系统值是可选的。

下图显示了传递给 **Dispatch**、Dispatch (5，3，2) 的参数之间的关系、 numthreads 属性中指定的值、numthreads (10，8，3) ，以及将传递给线程相关系统值的计算着色器的值 (SV_GroupIndex，SV_DispatchThreadID，SV_GroupThreadID，SV_GroupID) 。



以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
					x

另请参阅

语义

着色器模型 5

SV_GroupIndex

项目 • 2022/09/23

线程组中计算着色器线程的“平展”索引，该索引将多维SV_GroupThreadID转换为 1D 值。 SV_GroupIndex从 0 到 (numthreadsX * numthreadsY * numThreadsZ) – 1。

类型

类型
uint

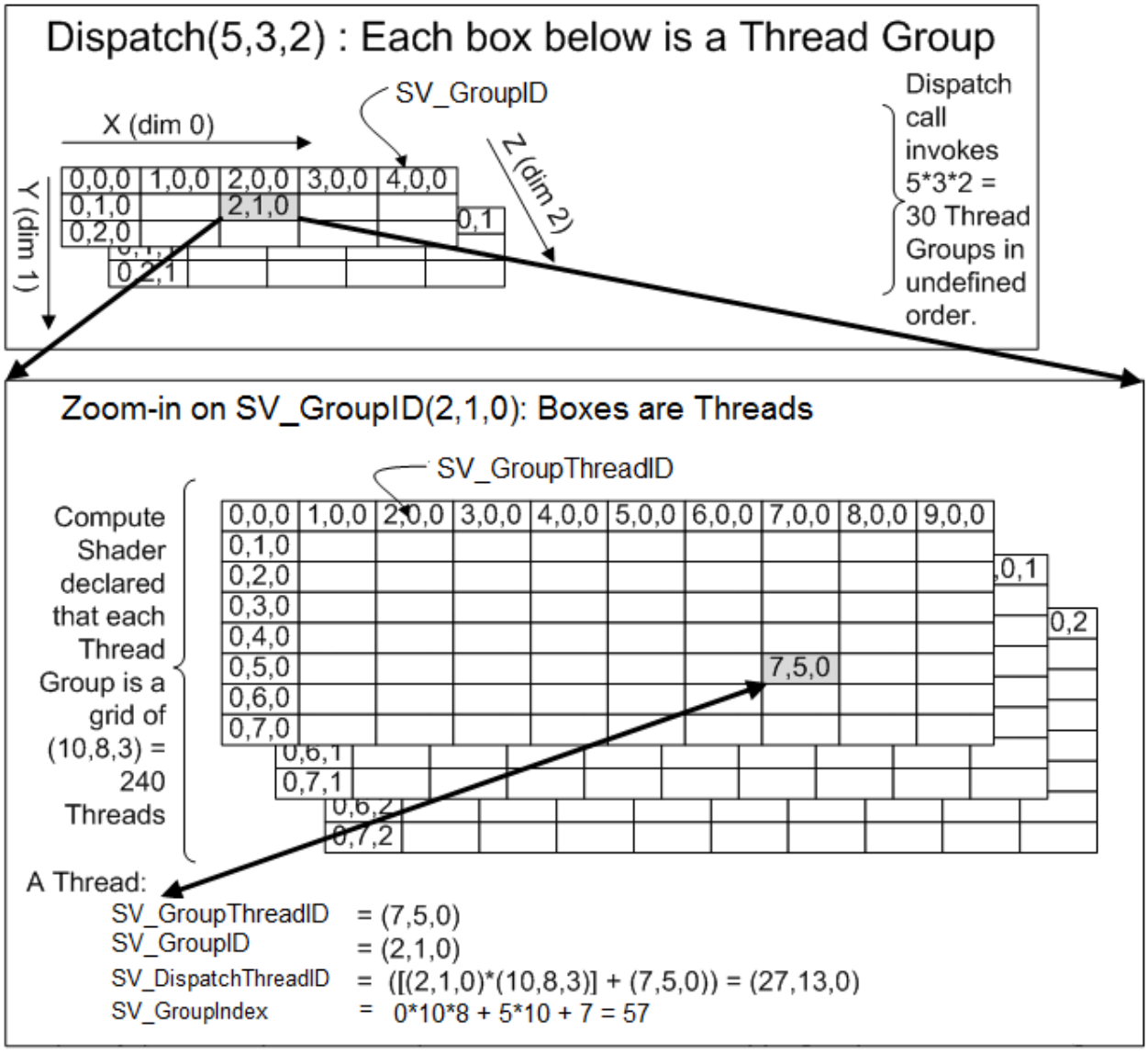
注解

```
SV_GroupIndex = SV_GroupThreadID.z*dimx*dimy +
                        SV_GroupThreadID.y*dimx +
                        SV_GroupThreadID.x
```

其中 dimx 和 dimy 是入口点的 numthreads 属性中指定的维度。

此系统值是可选的。 但是，它的使用可确保线程只写入其分配的组共享变量中的内存区域。

下图显示了传递给 **ID3D11DeviceContext：:D ispatch** 的参数之间的关系， 调度 (5，3，2) 、 numthreads 属性中指定的值、numthreads (10，8，3) ，以及将传递给线程相关系统值的计算着色器的值 (SV_GroupIndex，SV_DispatchThreadID，SV_GroupThreadID，SV_GroupID) 。



以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
					x

另请参阅

语义

着色器模型 5

SV_GroupThreadID

项目 • 2022/09/22

计算着色器正在在其中执行的线程组中单个线程的索引。 SV_GroupThreadID因 numthreads 属性中计算着色器指定的范围而异。 例如，如果 (为SV_GroupThreadID输入值指定了 3，2，1) 的 numthreads (0-2，0-1，0) 的值范围。

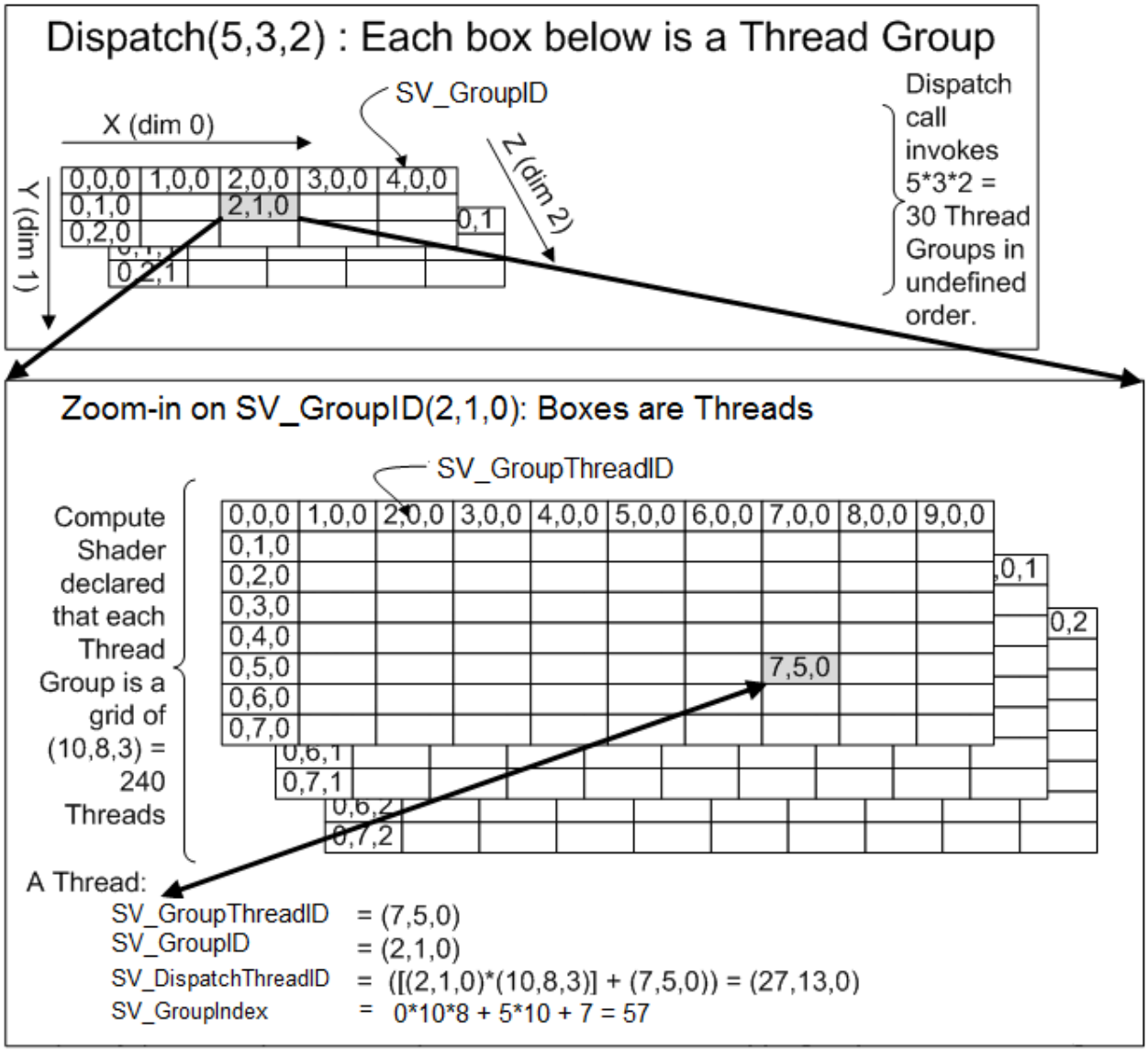
类型

类型
uint3

注解

此系统值是可选的，始终在传递到 numthreads 属性的值的边界内。

下图显示了传递给 **Dispatch** 的参数之间的关系， 调度 (5，3，2) 、 numthreads 属性中指定的值、numthreads (10，8，3) ，以及将传递给线程相关系统值的计算着色器的值 (SV_GroupIndex，SV_DispatchThreadID，SV_GroupThreadID，SV_GroupID) 。



以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
					x

另请参阅

语义

着色器模型 5

SV_GSInstanceID

项目 • 2022/09/22

定义几何着色器的 实例 。

类型

类型
uint

注解

此系统值是可选的。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
			x		

另请参阅

语义

着色器模型 5

SV_InsideTessFactor

项目 • 2022/09/22

定义修补图面中的细化量。

类型

类型	输入拓扑
float[2]	quad patch
FLOAT	tri patch
unused	等值 线

分割因子必须声明为数组;它们不能打包到单个向量中。

注解

必须在外壳着色器的修补常量函数期间定义此值。

如果使用象限或三个修补程序，则外壳着色器所需的输出值。 此值是域着色器所需的输入，以便硬件通过分割器匹配签名。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x	x			

另请参阅

语义

着色器模型 5

SV_OutputControlPointID

项目 • 2022/09/22

定义通过调用外壳着色器的主要入口点来操作的控制点 ID 的索引。

类型

类型
uint

注解

此系统值是可选的。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
	x				

另请参阅

语义

着色器模型 5

着色器模型与着色器配置文件

项目 • 2022/09/23

DirectX 的高级底纹语言实现了一系列着色器模型。 使用 HLSL，可以创建用于 Direct3D 管道的类似于 C 的可编程着色器。 每个着色器模型都基于模型的功能构建在模型之前，实现更多限制较少的功能。

着色器模型 1 以 DirectX 8 开头，包含程序集级别和类似 C 的说明。 此模型有许多由早期可编程着色器硬件引起的限制。 着色器模型 2 和 3 在指令数量上大大扩展，常量着色器可以使用。 它们比着色器模型 1 更强大，但仍具有第一个着色器模型的一些现有限制。

从 Windows Vista 开始，着色器模型 4 是一个完整的重新设计。 它允许在计算机) 的硬件约束内 (无限指令和常量，具有模板化对象，使纹理采样更加简洁高效，并且对任何着色器模型的限制最少。 但是，它需要仅在 Windows Vista (或更高版本) 操作系统上可用的 Windows驱动程序模型。

着色器配置文件

着色器配置文件是编译着色器的目标;下表列出了每个着色器模型支持的着色器配置文件。

着色器模型	着色器配置文件
着色器模型 1	vs_1_1
着色器模型 2	ps_2_0、ps_2_x、vs_2_0、vs_2_x、ps_4_0_level_9_0、ps_4_0_level_9_1、ps_4_0_level_9_3、vs_4_0_level_9_0、vs_4_0_level_9_1、vs_4_0_level_9_3、lib_4_0_level_9_1、lib_4_0_level_9_3

着色器模型	着色器配置文件
着色器模型 3	ps_3_0、vs_3_0
着色器模型 4	cs_4_0、gs_4_0、ps_4_0、vs_4_0、cs_4_1、gs_4_1、ps_4_1、vs_4_1、lib_4_0、lib_4_1
着色器模型 5	cs_5_0、ds_5_0、gs_5_0、hs_5_0、ps_5_0、vs_5_0、lib_5_0 (尽管gs_4_0、gs_4_1、ps_4_0、ps_4_1、vs_4_0和vs_4_1是在着色器模型 4.0 中引入的，着色器模型 5 增加了对这些结构化缓冲区和字节地址缓冲区的着色器配置文件的支持。)
着色器模型 6	cs_6_0、ds_6_0、gs_6_0、hs_6_0、ps_6_0、vs_6_0、lib_6_0

Direct3D 9 和 Direct3D 10 之间的差异：

- Direct3D 9 引入了着色器模型 1、2 和 3。
- Direct3D 10 引入了着色器模型 4。
- Direct3D 10.1 引入了着色器模型 4.1。

效果配置文件

效果配置文件是编译效果/着色器的目标;下表列出了 Direct3D 的每个版本支持的效果配置文件。

Direct3D 9 和 Direct3D 10 之间的差异：

- Direct3D 9 引入了效果框架配置文件fx_1_0和fx_2_0。
- Direct3D 10 引入了效果框架配置文件fx_4_0。
- Direct3D 10.1 引入了 effect-framework 配置文件fx_4_1。
- Direct3D 11 引入了效果框架配置文件fx_5_0。

> **备注**
>
> 这些旧效果配置文件已弃用。

相关主题

HLSL 参考

着色器模型 1

项目 • 2022/09/22

着色器模型 1 是在 DirectX 中创建的第一个着色器模型。 它向可编程管道的第一个实现引入了顶点和像素着色器。

特性	功能
指令集	• **HLSL 函数** • 顶点着色器程序集指令 (请参阅 说明 - vs_1_1) 。 (ps_1_x) 已弃用像素着色器说明。 若要将ps_1_x着色器编译为ps_2_0着色器，请参阅 编译着色器模型 1。
注册集	• 顶点着色器寄存器
顶点着色器最大值	128 指令
着色器配置文件	vs_1_1

有关着色器模型 1 的更多详细信息，请参阅：

- 顶点着色器

相关主题

着色器模型与着色器配置文件

着色器模型 2

项目 • 2022/09/22

着色器模型 2 向 着色器模型 1 添加了其他功能。

特性	功能
指令集	• **HLSL 函数** • 程序集说明 (请参阅 说明 - vs_2_0， 指令 - vs_2_x， ps_2_0说明， ps_2_x 指令)
注册集	• 像素着色器寄存器 (请参阅 ps_2_0寄存器， ps_2_x寄存器) • 顶点着色器寄存器 (请参阅 寄存器 - vs_2_0， 寄存器 - vs_2_x)
像素着色器最大值	• ps_2_0 - 32 纹理 + 64 算术 • ps_2_x - 96 个最小值，最多为 D3DCAPS9 中的槽数。 D3DPSHADERCAPS2_0.NumInstructionSlots。 请参阅D3DPSHADERCAPS2_0
顶点着色器最大值	256 条说明
着色器配置文件	ps_2_0、ps_2_x、vs_2_0、vs_2_x

有关着色器模型 2 的更多详细信息，请参阅：

- 像素着色器 2.0、 像素着色器 2.x
- 顶点着色器 2.0、 顶点着色器 2.x

相关主题

着色器模型与着色器配置文件

着色器模型 3 (HLSL 参考)

项目 • 2022/09/23

着色器模型 3 向 着色器模型 2 添加了其他功能。

特性	功能
指令集	• **HLSL 函数** • 程序集说明 (请参阅 ps_3_0说明、 说明 - vs_3_0)
注册集	• 像素着色器注册 (请参阅 ps_3_0寄存器) • 顶点着色器寄存器 (请参阅 寄存器 - vs_3_0)
像素着色器最大值	最低为 512，最大为 D3DCAPS9 中的槽数。MaxPixelShader30InstructionSlots (请参阅 **D3DPSHADERCAPS2_0)** 。
顶点着色器最大值	最低为 512，最大为 D3DCAPS9 中的槽数。MaxVertexShader30InstructionSlots (请参阅 **D3DCAPS9**) 。
着色器配置文件	ps_3_0、vs_3_0

有关模型 3 着色器的更多详细信息，请参阅：

- 像素着色器 3.0
- 顶点着色器 3.0

相关主题

着色器模型与着色器配置文件

着色器模型 3 (HLSL 参考)

项目 • 2022/09/24

顶点着色器和像素着色器在早期着色器版本中大大简化。 如果在硬件中实现着色器，则不能将vs_3_0或ps_3_0用于任何其他着色器版本，并且不能将任一着色器类型用于固定函数管道。 通过这些更改，可以简化驱动程序和运行时。 唯一的例外是，仅软件vs_3_0着色器可与任何像素着色器版本一起使用。 此外，如果将仅软件vs_3_0着色器与以前的像素着色器版本配合使用，顶点着色器只能使用与灵活顶点格式兼容的输出语义，（FVF）代码。

顶点着色器输出中使用的语义必须在像素着色器输入上使用。 语义用于将顶点着色器输出映射到像素着色器输入，类似于顶点声明映射到顶点着色器输入寄存器和以前的着色器模型的方式。 请参阅 vs 3.0 和 ps 3.0 着色器的匹配语义。

添加了其他换行模式呈现状态，以涵盖此新方案中其他纹理坐标的可能性。 当设置相应的 **D3DRS_WRAP*** 时，具有D3DDECLUSAGE_TEXCOORD和使用情况索引（从 0 到 15）的属性在换行模式下内插。

- 顶点着色器模型 3 功能
- 像素着色器模型 3 功能
- 匹配vs_3_0和ps_3_0着色器的语义
- 雾、深度和底纹模式更改
- 浮点转换和整数转换
- 指定全精度或部分精度
- 软件顶点和像素着色器

顶点着色器模型 3 功能

顶点着色器输出寄存器类型已折叠为十二个寄存器，（请参阅 输出寄存器）。 需要使用 dcl 指令和语义 (声明每个寄存器，例如，dcl_color0 o0.xyzw) 。

3_0 顶点着色器模型 (vs_3_0) 扩展了 vs_2_0具有更强大的寄存器索引、一组简化的输出寄存器、采样顶点着色器中的纹理的功能以及控制着色器输入初始化速率的功能。

为任何寄存器编制索引

所有寄存器 (输入寄存器和输出寄存器) 都可以使用Loop计数器寄存器编制索引，（只能在早期版本中为常量寄存器编制索引。）

在为其编制索引之前，必须声明输入和输出寄存器。 但是，不能为使用位置或点大小语义声明的任何输出寄存器编制索引。 事实上，如果使用索引编制，则必须分别在 o0 和

o1 寄存器中声明位置和 psize 语义。

只允许为连续的寄存器范围编制索引;也就是说，不能跨尚未声明的寄存器编制索引。 尽管此限制可能不方便，但它允许进行硬件优化。 尝试跨非连续寄存器编制索引将产生未定义的结果。 着色器验证不会强制实施此限制。

简化输出寄存器

所有类型的输出寄存器已折叠为 12 个输出寄存器：1 表示位置，2 表示颜色，8 表示纹理，1 表示雾或点大小。 这些寄存器将内插它们为像素着色器包含的任何数据。 输出寄存器声明是必需的，并将语义分配给每个寄存器。

寄存器可以按如下所示进行细分：

- 至少一个寄存器必须声明为四分量位置寄存器。 这是唯一必需的顶点着色器寄存器。
- 着色器使用的前 10 个寄存器最多可以使用 4 个组件 (xyzw) 最大值。
- 最后一个 (或第十二个) 寄存器只能包含标量 (，例如点大小) 。

有关寄存器的列表，请参阅 寄存器 - vs_3_0。

顶点着色器中的纹理示例

顶点着色器 3_0 支持使用 texldl 与顶点着色器中的纹理查找。

像素着色器模型 3 功能

像素着色器颜色和纹理寄存器已折叠成十个输入寄存器， (请参阅 输入寄存器类型) 。 人脸寄存器是浮点标量寄存器。 只有此寄存器的符号有效。 如果符号为负数，则基元为后脸。 例如，这可以在像素着色器内使用，以实现双面照明。 位置寄存器引用当前 (x，y) 像素。

可以使用以下命令设置着色器常量寄存器：

- SetPixelShaderConstantB
- SetPixelShaderConstantI
- SetPixelShaderConstantF

匹配vs_3_0和ps_3_0着色器的语义

vs_3_0和ps_3_0语义用法存在一些限制。 通常，在使用与着色器输出中使用的语义匹配的着色器输入的语义时，需要小心。

例如，此像素着色器将多个名称打包到一个寄存器中：

```
ps_3_0
dcl_texcoord0 v0.x
dcl_texcoord1 v0.yz // Valid to pack multiple names into one register
dcl_texcoord2_centroid v1.w
...
```

每个寄存器都有不同的语义。 请注意，由于使用写入掩码，还可以使用不同的 (多个) 语义命名 v0.x 和 v0.yz。

给定像素着色器，以下vs_3_0着色器无法与它配对：

```
vs_3_0
...
dcl_texcoord0 o5.x
dcl_texcoord1 o6.yzw
...
```

这两个着色器与使用 D3DDECLUSAGE_TEXCOORD0 和 D3DDECLUSAGE_TEXCOORD1 语义冲突。

重写如下所示的顶点着色器，以避免语义冲突：

```
vs_3_0
...
dcl_texcoord2 o3
dcl_texcoord3 o9
...
```

同样，在像素着色器 (v0 和 v1 中的不同输入寄存器上声明的语义名称) 不能在此顶点着色器中的单个输出寄存器中使用。 例如，无法将此顶点着色器与像素着色器配对，因为D3DDECLUSAGE_TEXCOORD1用于两个像素着色器输入寄存器 (v0、v1) 和顶点着色器输出寄存器 o3。

```
vs_3_0
...
dcl_texcoord0 o3.x
dcl_texcoord1 o3.yz
```

```
dcl_texcoord2 o3.w // BAD! Would be valid if this were not o3
dcl_texcoord3 o9 ...
```

另一方面，此顶点着色器不能与像素着色器配对，因为具有给定语义的参数的输出掩码不提供像素着色器请求的数据：

```
vs_3_0
...
dcl_texcoord0 o5.x
dcl_texcoord1 o5.yzw
dcl_texcoord2 o7.yz // BAD! Would be valid if w were included
dcl_texcoord3 o9
...
```

此顶点着色器不提供包含像素着色器请求的语义名称之一的输出，因此着色器配对无效：

```
vs_3_0
...
dcl_texcoord0 o5.x
dcl_texcoord1 o5.yzw
dcl_texcoord3 o9
// The pixel shader wants texcoord2, with a w component,
// but it isn't output by this vertex shader at all!
...
```

雾、深度和底纹模式更改

当在剪辑和三角形光栅化期间为平面底纹设置D3DRS_SHADEMODE时，带有D3DDECLUSAGE_COLOR的属性将内插为平面着色。 如果使用颜色语义声明寄存器的任何组件，但同一寄存器的其他组件具有不同的语义，则不带颜色语义的寄存器中的组件上将不定义平面底纹内插 (线性与平面) 。

如果需要雾呈现，vs_3_0和ps_3_0着色器必须实现雾。 着色器之外不会执行雾计算。 vs_3_0中没有雾寄存器，并且已添加用于计算每个顶点) 的雾混合因子的其他D3DDECLUSAGE_FOG (语义，D3DDECLUSAGE_DEPTH (用于向像素着色器传递深度值以计算雾混合因子) 。

使用像素着色器 3.0 时，将忽略纹理阶段状态D3DTSS_TEXCOORDINDEX。

添加了以下值以适应这些更改：

```
// Fog and Depth usages
D3DDECLUSAGE_FOG
D3DDECLUSAGE_DEPTH

// Additional wrap states for vs_3_0 attributes with D3DDECLUSAGE_TEXCOORD
D3DRS_WRAP8
D3DRS_WRAP9
D3DRS_WRAP10
D3DRS_WRAP11
D3DRS_WRAP12
D3DRS_WRAP13
D3DRS_WRAP14
D3DRS_WRAP15
```

浮点转换和整数转换

浮点数学以不同的精度和范围发生，(管道不同部分的 16 位、24 位和 32 位) 。 大于输入该管道 (的管道的动态范围的值，ps_2_0) 将 32 位浮点纹理映射采样到 24 位浮点管道中，从而创建未定义的结果。 对于可预测的行为，应将此类值固定到动态范围最大值。

从浮点值到整数的转换发生在多个位置，例如：

- 遇到 mova - 与 指令时。
- 在纹理寻址期间。
- 写出到非浮点呈现目标时。

指定全精度或部分精度

ps_3_0和ps_2_x都为两个精度级别提供支持：

ps_3_0	ps_2_0	精度	值
x		完整	fp32 或更高版本
x		部分精度	fp16=s10e5
x	x	完整	fp24=s16e7 或更高版本
x	x	部分精度	fp16=s10e5

ps_3_0支持精度高于ps_2_0。 默认情况下，所有操作在完全精度级别发生。

部分精度 (看到 像素着色器寄存器修饰符) 是通过将_pp修饰符添加到着色器代码 (请求的，前提是基础实现支持它) 。 实现始终可以随意忽略修饰符，并完全精确执行受影响的操作。

_pp修饰符可以在两个上下文中发生：

- 在纹理坐标声明中，将部分精度纹理坐标传递给像素着色器。 当纹理坐标将颜色数据中继到像素着色器时，这可能比某些实现中的全精度更快。
- 在请求使用部分精度的任何指令上，包括纹理加载指令。 这表示允许实现以部分精度执行指令，并存储部分精度结果。 如果没有显式修饰符，指令必须以完全精度 (执行，而不考虑输入操作数) 的精度。

应用程序可能会故意选择权衡性能的精度。 有多种类型的着色器输入数据是部分精度处理的自然候选项：

- 颜色迭代器由部分精度值表示。
- 大多数格式的纹理值可以通过部分精度值准确表示 (从 32 位浮点格式纹理采样的值) 。
- 常量可以按部分精度表示形式来表示，适合着色器。

在所有这些情况下，开发人员可以选择指定部分精度来处理数据，知道不会丢失任何输入数据精度。 在某些情况下，着色器可能需要以完全精度执行计算的内部步骤，即使输入和最终输出值没有超过部分精度也是如此。

软件顶点和像素着色器

软件实现 (顶点着色器的运行时和参考，以及像素着色器的引用，) 版本 2_0 着色器及更高版本的着色器引用具有一些宽松的验证。 这对于调试和原型制作非常有用。 应用程序向运行时/汇编程序指示，它需要使用汇编程序 (中的_sw标志来放宽某些验证，例如 vs_2_sw) 。 软件着色器不适用于硬件。

vs_2_sw是放松vs_2_x的最大上限;同样，ps_2_sw是放宽ps_2_x的最大上限。 具体而言，以下验证是宽松的：

着色器模型	资源	限制
vs_2_sw、vs_3_sw、ps_2_sw、ps_3_sw	指令计数	无限制
vs_2_sw、vs_3_sw、ps_2_sw、ps_3_sw	Float 常量寄存器	8192
vs_2_sw、vs_3_sw、ps_2_sw、ps_3_sw	整数常量寄存器	2048

着色器模型	资源	限制
vs_2_sw、vs_3_sw、ps_2_sw、ps_3_sw	布尔常量寄存器	2048
ps_2_sw	从属读取深度	无限制
vs_2_sw	流控制指令和标签	无限制
vs_2_sw、vs_3_sw、ps_2_sw、ps_3_sw	Loop start/step/counts	代表和循环指令的迭代开始和迭代步骤大小是 32 位有符号整数。 计数最多可以MAX_INT/64。
vs_2_sw、vs_3_sw、ps_2_sw、ps_3_sw	端口限制	所有注册文件的端口限制都放宽。
vs_3_sw	内插器数	vs_3_sw中的 16 个输出寄存器。
ps_3_sw	内插器数	14 (16-2) 输入寄存器用于ps_3_sw。

相关主题

着色器模型 3 (DirectX HLSL)

片段声明语法 (Direct3D 9 HLSL)

项目 • 2022/09/22

每个 Microsoft 高级着色器语言 (HLSL) 函数都可以通过添加片段声明转换为着色器片段。

语法

```
fragmentKeyword FragmentName = compile_fragment shaderProfile
FunctionName();
```

其中：

值	说明
fragmentKeyword	必需关键字。 像素整理或顶点整理。
FragmentName	一个 ASCII 文本字符串，指定已编译的片段名称。
compile_fragment	必需关键字。
shaderProfile	要编译的着色器模型。 任何有效的顶点着色器配置文件 (请参阅 D3DXGetVertexShaderProfile) 或像素着色器配置文件， (请参阅 D3DXGetPixelShaderProfile) 。
FunctionName ()	着色器函数名称，后跟括号。

共享片段参数通过向其语义添加“r_”前缀来标记。

```
void AmbientDiffuse( float3 vPosWorld: r_PosWorld,
                     float3 vNormalWorld: r_NormalWorld,
                     out float4 vColor: COLOR0 )
{
    // Compute the light vector
    float3 vLight = normalize( g_vLightPosition - vPosWorld );

    // Compute the ambient and diffuse components of illumination
    vColor = g_vLightColor * g_vMaterialAmbient;
    vColor += g_vLightColor * g_vMaterialDiffuse * saturate( dot( vLight,
vNormalWorld ) );
}
```

```
vertexfragment AmbientDiffuseFragment = compile_fragment vs_1_1
AmbientDiffuse();
```

在此示例中，r_PosWorld和r_NormalWorld语义标识这两个参数是其他片段之间的共享参数。

> **备注**
>
> 片段链接器是 D3DX 9 中的 Microsoft Direct3D 9 技术。 片段链接器是工具 (Flink.exe) 、D3DX 9 API 和 HLSL 增强功能。 自 2009 年 8 月 DirectX SDK 版本起，片段链接器已删除。 片段链接器从不应用于 Microsoft Direct3D 10、Microsoft Direct3D 10.1 或 Microsoft Direct3D 11。

相关主题

着色器模型 3 (DirectX HLSL)

着色器模型 4

项目 • 2022/09/22

着色器模型 4 是 着色器模型 3 中功能的超集，但着色器模型 4 不支持着色器模型 1 中的功能。 它使用通用着色器核心进行设计，它为所有可编程着色器提供一组通用功能，这些着色器仅使用 HLSL 进行可编程。

特性	功能
指令集	**HLSL 函数**
注册集	注册集可通过常量缓冲区和纹理缓冲区中的成员访问，这些成员使用 HLSL 语义进行组件打包等操作。 • 像素着色器寄存器 (请参阅 寄存器 - ps_4_0 和 寄存器 - ps_4_1) • 顶点着色器寄存器 (请参阅 寄存器 - vs_4_0 和 寄存器 - vs_4_1) • 几何着色器寄存器 (请参阅 寄存器 - gs_4_0 和 寄存器 - gs_4_1)
顶点着色器最大值	无限制
像素着色器最大值	无限制
添加了新的着色器配置文件	gs_4_0、ps_4_0、vs_4_0、gs_4_1*、ps_4_1*、gs_4_1*
添加了新的Effect-Framework配置文件	fx_4_0，fx_4_1*

* - Direct3D 10.1 或更高版本支持gs_4_1、ps_4_1、vs_4_1和fx_4_1。

着色器模型 4 支持新的管道阶段（几何着色器阶段），可用于创建或修改现有几何图形。 它还包括两个新对象类型：一个流输出对象，用于将数据流式传输到几何阶段，以及实现纹理采样函数的模板化纹理对象。

- Common-Shader Core
- 常量
- Geometry-Shader 对象
- Stream-Output 对象
- 纹理对象

着色器模型 4 支持打包规则，这些规则决定了在存储数据时可以排列得有多紧密。 这些规则在常量变量的打包规则中介绍

着色器模型 4 程序集部分介绍着色器模型 4 和着色器模型 4.1 支持的程序集说明。

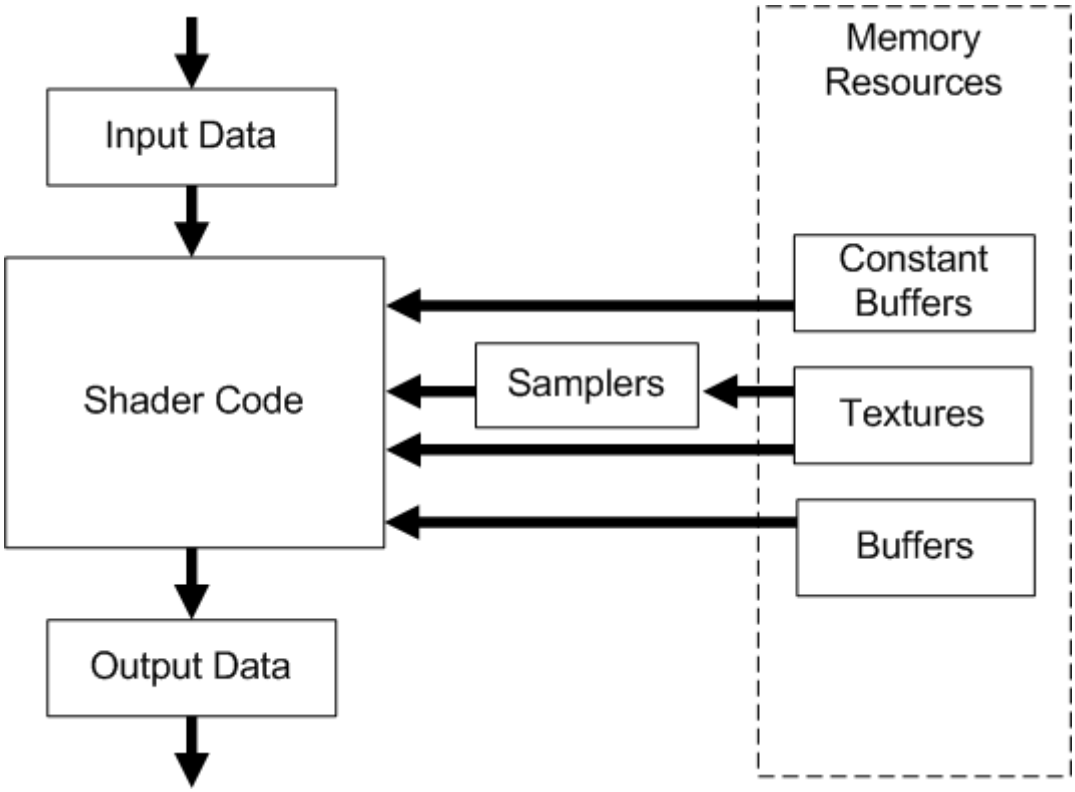
相关主题

着色器模型与着色器配置文件

Common-Shader核心

项目 • 2022/09/22

在着色器模型 4 中，所有着色器阶段都使用通用着色器核心实现相同的基本功能。 此外，这三个着色器阶段 (顶点、几何图形和像素) 为每个阶段提供唯一的功能，例如能够从几何着色器阶段生成新基元或放弃像素着色器阶段中的特定像素。 下图显示了数据如何流经着色器阶段，以及公共着色器核心与着色器内存资源的关系。



- **输入数据**：顶点着色器从输入汇编器阶段接收其输入;几何图形和像素着色器从上一着色器阶段接收其输入。 其他输入包括 系统值语义，这些语义可供管道中的第一个单元使用，它们适用。
- **输出数据**：着色器生成输出结果，以传递到管道中的后续阶段。 对于几何图形着色器，单个调用的数据输出量可能会有所不同。 某些输出由公共着色器核心 (解释，例如顶点位置和呈现目标数组索引) ，其他输出由应用程序设计为解释。
- **着色器代码：着色器**可以从内存中读取、执行矢量浮点和整数算术运算或流控制操作。 着色器中可以实现的语句数没有限制。
- **采样器**：采样器定义如何采样和筛选纹理。 多达 16 个采样器可以同时绑定到着色器。
- **纹理**：可以使用采样器筛选纹理，也可以使用 负载 内部函数直接按纹素读取纹理。
- **缓冲区**：从不筛选缓冲区，但可以直接使用 加载 内部函数从内存中读取缓冲区。 (组合) 可以同时绑定到着色器，多达 128 个纹理和缓冲区资源。
- **常量缓冲区**：常量缓冲区针对着色器常量变量进行优化。 多达 16 个常量缓冲区可以同时绑定到着色器阶段。 它们旨在通过 CPU 进行更频繁的更新;因此，它们具有其他大小、布局和访问限制。

Direct3D 9 和 Direct3D 10 之间的差异：

- 在 Direct3D 9 中，每个着色器单元都有一个小型常量寄存器文件，用于存储所有常量着色器变量。 容纳具有此有限常量空间的所有着色器需要 CPU 频繁回收常量。
- 在 Direct3D 10 中，常量存储在内存中的不可变缓冲区中，并像任何其他资源一样进行管理。 应用程序可以创建的常量缓冲区数没有限制。 通过将常量按更新和使用情况的频率将常量组织到缓冲区中，可以显著减少更新常量以容纳所有着色器所需的带宽量。

整数和按位支持

常见的着色器核心提供一组符合 IEEE 的 32 位整数和按位运算。 这些操作使图形硬件示例中的新算法类包括压缩和打包技术、FFT 和位字段程序流控制。

Direct3D 10 HLSL 中的 **int** 和 **uint** 数据类型映射到硬件中的 32 位整数。

Direct3D 9 和 Direct3D 10 之间的差异： 在标记为 HLSL 中的整数的 Direct3D 9 流输入中，被解释为浮点。 在 Direct3D 10 中，标记为整数的流输入解释为 32 位整数。 此外，布尔值现在是所有位集或所有位未设置。 如果值不等于 0.0f，则转换为 **布尔** 的数据将解释为 true，（允许正零和负零为 false），否则为 false。

位运算符

常见的着色器核心支持以下按位运算符：

运算符	**函数**
~	逻辑非
<<	左移
>>	右移
&	逻辑与
|	逻辑或
^	逻辑 Xor
<<=	左移等于
>>=	右移相等
&=	和等于

运算符	函数
|=	或等于
^=	Xor Equal

按位运算符定义为仅对 int 和 uint 数据类型进行操作。 尝试对 **浮点** 或 **结构** 数据类型使用按位运算符将导致错误。 对于其他运算符，按位运算符遵循与 C 相同的优先级。

二进制转换

整数和浮点类型之间的强制转换将转换遵循 C 截断规则的数值。 将值从 **浮点**转换为 int，并返回 **浮点** 是一个损失转换，具体取决于目标数据类型的精度。 下面是一些转换函数： asfloat (DirectX HLSL) 、 asint (DirectX HLSL) 、 asuint (DirectX HLSL) 。

也可以使用 HLSL 内部函数执行二进制转换。 这会导致编译器将数字的位表示形式重新解释为目标数据类型。

相关主题

着色器模型 4

着色器常量 (HLSL)

项目 • 2022/09/22

在着色器模型 4 中，着色器常量存储在内存中的一个或多个缓冲区资源中。 它们可分为两种类型的缓冲区：常量缓冲区 (cbuffers) ，纹理缓冲区 (tbuffers) 。 常量缓冲区针对常量变量使用情况进行了优化，其特点是低延迟访问和 CPU 更新频率更高。 因此，其他大小、布局和访问限制适用于这些资源。 纹理缓冲区会像纹理一样访问，并更好地为任意索引的数据执行。 无论使用哪种类型的资源，应用程序可以创建的常量缓冲区或纹理缓冲区数没有限制。

声明常量缓冲区或纹理缓冲区看起来非常类似于 C 中的结构声明，并添加了用于手动分配寄存器或打包数据的 **register** 和 **packoffset** 关键字。

BufferType [*Name*] [: **register** (b#)] { *VariableDeclaration* [: **packoffset** (c#.xyzw)]; ... };

参数

BufferType

[in]缓冲区类型。

BufferType	**描述**
cbuffer	常量缓冲区
tbuffer	纹理缓冲区

名字

[in]可选，包含唯一缓冲区名称的 ASCII 字符串。

register (b#)

[in]可选关键字，用于手动打包常量数据。 常量只能在常量缓冲区中打包，其中起始寄存器由寄存器号 () # 提供。

VariableDeclaration

[in]变量声明，类似于结构成员声明。 这可以是任何 HLSL 类型或效果对象， (纹理或采样器对象) 。

packoffset (c#.xyzw)

[in]可选关键字，用于手动打包常量数据。 常量可以打包到任何常量缓冲区中，其中寄存器号由 () # 提供。 使用 xyzw 重排) 的子组件打包 (可用于大小适合单个寄存器的常量，(不跨寄存器边界) 。 例如，从 y 组件开始的单个寄存器中无法打包 float4，因为它不适合四分量寄存器。

注解

常量缓冲区允许将着色器常量组合在一起并同时提交，而不是逐个调用来单独提交每个常量，这减少了更新着色器常量所需的带宽。

常量缓冲区是一种专用缓冲区资源，它像缓冲区一样访问。 每个常量缓冲区最多可以容纳 4096 向量;每个向量最多包含四个 32 位值。 每个管道阶段最多可以绑定 14 个常量缓冲区， (另外 2 个槽用于内部使用) 。

纹理缓冲区是像纹理一样访问的专用缓冲区资源。 与缓冲区访问) 相比，纹理访问 (可以对任意索引的数据具有更好的性能。 每个管道阶段最多可以绑定 128 个纹理缓冲区。

缓冲区资源旨在最大程度地减少设置着色器常量开销。 效果框架 (看到 **ID3D10Effect 接口**) 将管理更新常量和纹理缓冲区，也可以使用 Direct3D API 更新缓冲区， (请参阅 复制和访问资源数据 (Direct3D 10) 以获取信息) 。 应用程序还可以将数据从另一个缓冲区 ((例如呈现目标或流输出目标) 复制到常量缓冲区) 。

有关在 D3D10 应用程序中使用常量缓冲区的详细信息，请参阅 资源类型 (Direct3D 10) 和 创建缓冲区资源 (Direct3D 10) 。

有关在 D3D11 应用程序中使用常量缓冲区的详细信息，请参阅 Direct3D 11 中的缓冲区简介 和 如何：创建常量缓冲区。

常量缓冲区不需要 将视图 绑定到管道。 但是，纹理缓冲区需要视图，并且必须在 (或使用效果) 时与 **SetTextureBuffer** 绑定。

有两种方法可以打包常量数据：使用 寄存器 (DirectX HLSL) 和 packoffset (DirectX HLSL) 关键字。

Direct3D 9 和 Direct3D 10 和 11 之间的差异：

- 与 Direct3D 9 中的常量自动分配不同，该常量未执行打包，而是将每个变量分配给一组 float4 寄存器，HLSL 常量变量遵循 Direct3D 10 和 11 中的打包规则。

组织常量缓冲区

常量缓冲区允许将着色器常量组合在一起并同时提交，而不是逐个调用来单独提交每个常量，这减少了更新着色器常量所需的带宽。

有效使用常量缓冲区的最佳方法是，根据它们的更新频率将着色器变量组织成常量缓冲区。 这样，应用程序就可以最大程度地减少更新着色器常量所需的带宽。 例如，着色器可以声明两个常量缓冲区，并根据更新频率在每个对象中组织数据：需要按对象更新的数据，如世界矩阵 () 分组为可为每个对象更新的常量缓冲区。 这与描述场景特征的数据分开，因此，当场景更改) 时，（更新频率可能要少得多。

```
cbuffer myObject
{
    float4x4 matWorld;
    float3   vObjectPosition;
    int      arrayIndex;
}

cbuffer myScene
{
    float3   vSunPosition;
    float4x4 matView;
}
```

默认常量缓冲区

有两个默认常量缓冲区可用，$Global和$Param。 使用用于 cbuffer 的相同打包方法，将全局范围内放置的变量隐式添加到 $Global cbuffer。 在效果框架外部编译着色器时，函数参数列表中的统一参数显示在$Param常量缓冲区中。 在效果框架内部编译时，所有制服都必须解析为全局范围内定义的变量。

示例

下面是 Skinning10 Sample 中的一个示例，它是由矩阵数组组成的纹理缓冲区。

```
tbuffer tbAnimMatrices
{
    matrix g_mTexBoneWorld[MAX_BONE_MATRICES];
};
```

此示例声明手动分配常量缓冲区以在特定寄存器处开始，并按子组件打包特定元素。

```
cbuffer MyBuffer : register(b3)
{
    float4 Element1 : packoffset(c0);
    float1 Element2 : packoffset(c1);
    float1 Element3 : packoffset(c1.y);
}
```

相关主题

着色器模型 4

Geometry-Shader 对象

项目 • 2022/09/22

几何着色器对象处理整个基元。 使用以下语法声明几何着色器对象。

[maxvertexcount (*NumVerts*)] void *ShaderName* (*PrimitiveType DataType Name [NumElements]* ， inout *StreamOutputObject*) ;

参数

[maxvertexcount (*NumVerts*)]

[in]要创建的最大顶点数的声明。

- [maxvertexcount ()] - 必需关键字;括号和括号是正确语法所必需的字符。
- *NumVerts* - 表示顶点数的整数。

ShaderName

[in]包含几何着色器函数的唯一名称的 ASCII 字符串。

PrimitiveType DataType 名称 [NumElements]

PrimitiveType - *基元* 类型，用于确定基元数据的顺序。

基元类型	说明
点	点列表
line	行列表或线条条带
三角形	三角形列表或三角形条带
lineadj	带相邻线或带相邻线的行列表
三角形adj	带相邻或带相邻的三角形列表

DataType - [in] 输入数据类型;可以是任何 HLSL 数据类型。

Name - 参数名称;这是 ASCII 字符串。

NumElements - 输入的数组大小，具体取决于 *PrimitiveType* ，如下表所示。

基元类型	NumElements

基元类型	NumElements
点	[1] 一次只运行一个点。
line	[2] 一行需要两个顶点。
三角形	[3] 三角形需要三个顶点。
lineadj	[4] lineadj 有两端：因此，它需要四个顶点。
三角形adj	[6] 三角形边框再三个三角形：因此，它需要六个顶点。

StreamOutputObject

流输出对象的声明。

返回值

无

备注

下图显示了几何着色器对象的各种基元类型。

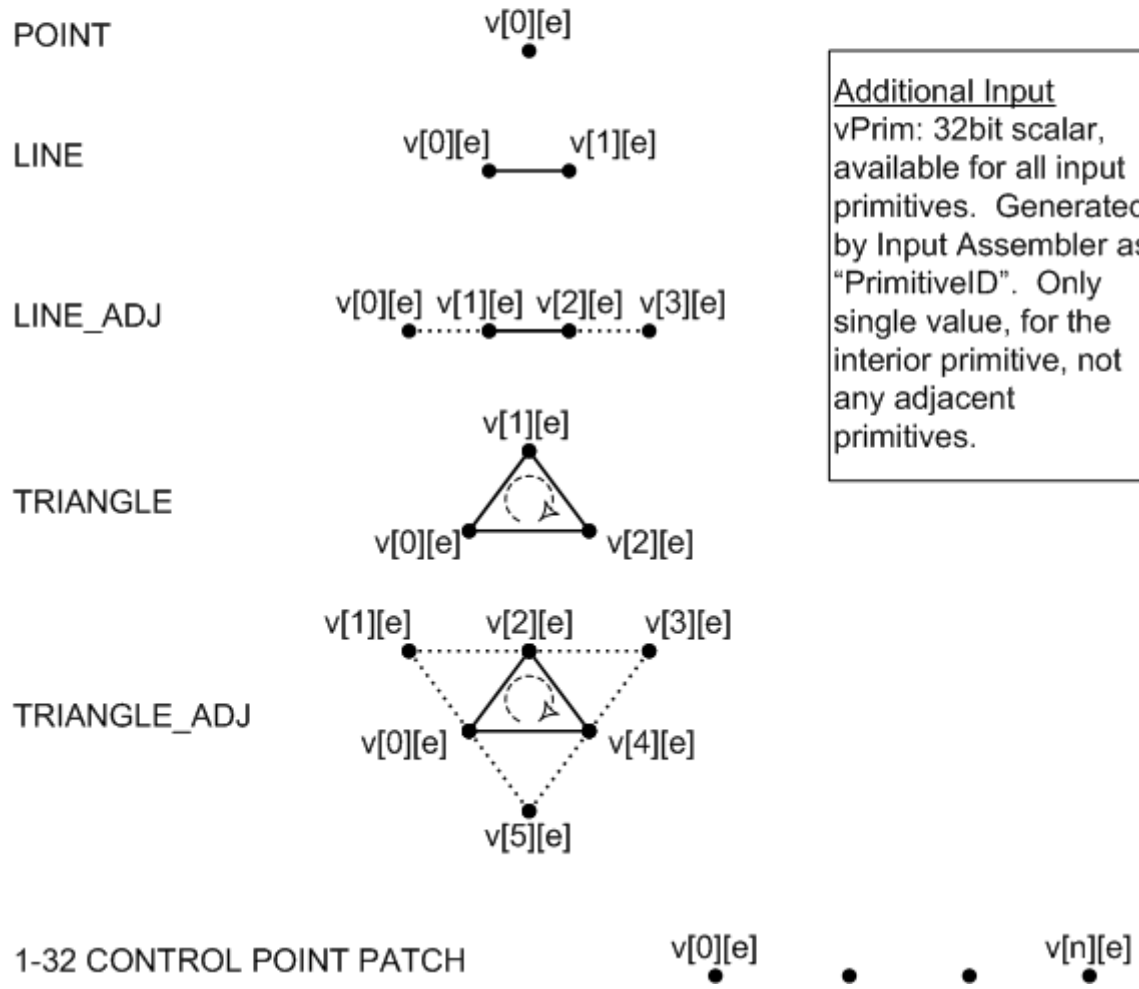
Geometry Shader Input Primitives

Legend:

- = Vertex

= Winding direction

v[n][e] = Input vertex n, element e (n and e independently indexable)

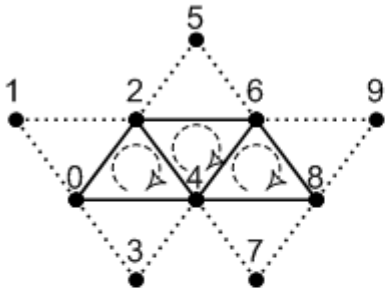


Additional Input
vPrim: 32bit scalar, available for all input primitives. Generated by Input Assembler as "PrimitiveID". Only single value, for the interior primitive, not any adjacent primitives.

下图显示了几何着色器调用。

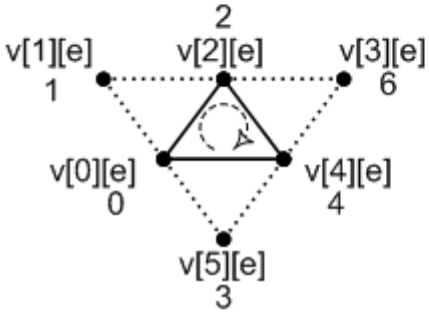
Example: GS Invocations From TriStrip w/Adjacency

Triangle strip with adjacency, generated by Input Assembler :



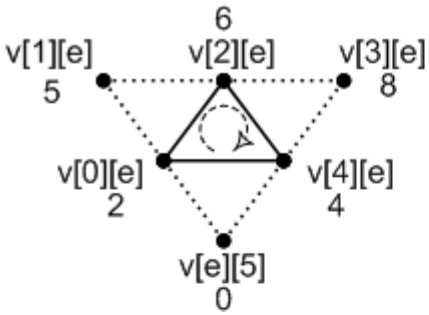
Resulting Geometry Shader Invocations (TRIANGLE_ADJ) :

First invocation:



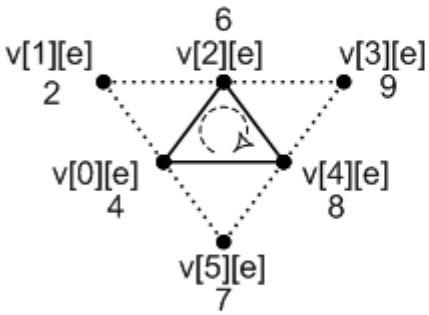
vPrim = 0 (assuming the strip is first in Draw() call)

Second invocation:



vPrim = 1

Third invocation:



vPrim = 2

In general, for each GS invocation, v[0][#] is initialized with the leading vertex for the primitive, and other vertices are obtained by traversing around the primitive in the direction of the winding order.

示例

此示例来自 Direct3D 10 着色器模型 4.0 研讨会 的练习 1。

```
[maxvertexcount(3)]
void GSScene( triangleadj GSSceneIn input[6], inout
TriangleStream<PSSceneIn> OutputStream )
{
    PSSceneIn output = (PSSceneIn)0;

    for( uint i=0; i<6; i+=2 )
    {
        output.Pos = input[i].Pos;
        output.Norm = input[i].Norm;
        output.Tex = input[i].Tex;

        OutputStream.Append( output );
    }

    OutputStream.RestartStrip();
}
```

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 4 和更高着色器模型	是

相关主题

着色器模型 4

Stream-Output 对象

项目 • 2022/09/22

流输出对象是一个模板化对象，用于将数据流出 几何着色器阶段。 使用以下语法声明流输出对象。

inout *StreamOutputObjectDataTypeName*<>;

参数

StreamOutputObject<*DataType*>*名字*

流输出对象 (SO) 声明。

Stream-Output对象类型	描述
PointStream	点基元序列
LineStream	行基元序列
TriangleStream	三角形基元序列

DataType - 输出数据类型;可以是任何 HLSL 数据类型。 必须用尖括号括起来。

名称 - 变量名称;唯一标识对象的 ASCII 字符串。

示例

这是流输出对象声明的示例，它流式传输出三角形基元，其数据由PS_CUBEMAP_IN结构定义。 几何着色器仅限于生成 18 个顶点。

```
struct PS_CUBEMAP_IN
{
    float4 Pos : SV_POSITION;     // Projection coord
    float2 Tex : TEXCOORD0;       // Texture coord
    uint RTIndex : SV_RenderTargetArrayIndex;
};
```

```
[maxvertexcount(18)]
void main( inout TriangleStream<PS_CUBEMAP_IN> CubeMapStream, triangle
PS_CUBEMAP_INT[3] )
{
    ...
}
```

这是 CubeMapGS 示例中 的代码片段。

Stream-Output对象方法

使用以下语法调用 stream-output-object 方法。

```
Object.Method
```

实现以下方法。

方法	描述
追加	将输出数据追加到现有流。
RestartStrip	结束当前基元条带并启动新的基元条带。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 4 及更高着色器模型	是

相关主题

着色器模型 4

追加 (DirectX HLSL Stream-Output 对象)

项目 • 2022/09/23

将 geometry-shader-output 数据追加到现有流。

追加 (*StreamDataType*) ;

参数

项	说明
StreamDataType	数据输入说明。 此说明必须与名为 DataType 的 stream-object 模板参数匹配。

返回值

无

示例

此代码片段 (多 维数据集MapGS 示例) 显示了向流输出对象追加三角形条基元的部分示例。

```
[maxvertexcount(18)]
void GS_CubeMap( triangle GS_CUBEMAP_IN input[3],
                 inout TriangleStream<PS_CUBEMAP_IN> CubeMapStream )
{
    for( int f = 0; f < 6; ++f )
    {
        // Compute screen coordinates
        PS_CUBEMAP_IN output;
        output.RTIndex = f;
        for( int v = 0; v < 3; v++ )
        {
            output.Pos = mul( input[v].Pos, g_mViewCM[f] );
            output.Pos = mul( output.Pos, mProj );
            output.Tex = input[v].Tex;
            CubeMapStream.Append( output );
        }
        CubeMapStream.RestartStrip();
    }
}
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

Stream-Output 对象

RestartStrip (DirectX HLSL Stream-Output 对象)

项目 • 2022/09/22

结束当前基元条带并启动一个新条带。 如果当前条带没有足够的顶点发出来填充基元拓扑，则会丢弃末尾不完整的基元。

RestartStrip () ;

参数

项	说明
没有	

返回值

无

备注

条带剪切会导致当前条带结束，并启动一个新条带。 可以通过显式调用此方法来完成条带剪切，也可以仅通过呈现最大索引值 1 (1 来完成，32 位索引0xffffffff，或者 16 位索引) 0xffff。 索引实例绘制的每个实例都会自动生成条带剪切。 即使拓扑不是三角形条带，也是如此。

> ⓘ **备注**
>
> 仅 **功能级别** 10.0 或更高版本设备上才支持重启和 1 个“magic 值”。

输出始终假定为三角形条带。 若要使输出成为三角形列表，必须在每个三角形之间调用 RestartStrip。 不支持三角形风扇。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

Stream-Output 对象

纹理对象

项目 • 2022/09/22

在 Direct3D 10 中，可以单独指定采样器和纹理;纹理采样是使用模板化纹理对象实现的。 此模板化纹理对象具有特定格式，返回特定类型，并实现多种方法。

Direct3D9 和 Direct3D10 之间的差异：

- 在 Direct3D 9 中，采样器绑定到特定纹理。
- 在 Direct3D 10 中，纹理和采样器是独立的对象。 每个模板化纹理对象实现纹理采样方法，该方法将纹理和采样器作为输入参数。

下面是用于创建除多重采样对象) 以外的所有纹理对象的语法 (。

```
Object1 [<类型>] 名称;
```

多重采样对象 (Texture2DMS 和 Texture2DMSArray) 要求显式声明纹理大小并将其表示为样本数。

```
Object2 [<类型， 示例>] 名称;
```

参数

项	说明

项	说明
对象	纹理对象。 必须是下列类型之一。

Object1 类型	说明
Buffer	Buffer
Texture1D	1D 纹理
Texture1DArray	1D 纹理数组
Texture2D	2D 纹理
Texture2DArray	2D 纹理数组
Texture3D	3D 纹理
TextureCube	立方体纹理
TextureCubeArray	多维数据集纹理数组
Object2 类型	说明
Texture2DMS	2D 多重采样纹理
Texture2DMSArray	2D 多重采样纹理数组

1. 缓冲区类型支持除 GetDimensions 以外的大多数纹理对象方法。
2. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
3. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

类型	可选。 任何 标量 HLSL 类型 或 **矢量 HLSL 类型**，都用尖括号括起来。 默认类型为 **float4**。
名字	一个指定纹理对象名称的 ASCII 字符串。
样品	(样本数介于 1 和 128) 之间。

示例 1

下面是声明纹理对象的示例。

```
Texture2D <float4> MyTex;

Texture2DMS <float4, 128> MyMSTex;
```

纹理对象方法

每个纹理对象实现某些方法;下面是列出所有方法的表。 查看每个方法的参考页，了解哪些对象可以使用该方法。

Texture 方法	说明	vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
CalculateLevelOfDetail	计算 LOD，返回固定结果。				x		
CalculateLevelOfDetailUnclamped	计算 LOD，返回未采样的结果。				x		
Gather	获取四个样本 (红色分量，仅在对纹理进行采样时用于双线性内插的) 。		x		x		x
GetDimensions	获取指定 mipmap 级别的纹理维度。	x	x	x	x	x	x
GetDimensions (MultiSample)	获取指定 mipmap 级别的纹理维度。		x		x		x
GetSamplePosition	获取指定样本的位置。		x		x		x
加载	在不进行任何筛选或采样的情况下加载数据。	x	x	x	x	x	x

Texture 方法	说明	vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
加载 (多重采样)	在不进行任何筛选或采样的情况下加载数据。		x	x	x		x
示例	采样纹理。			x	x		
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。			x	x		
SampleCmp	使用比较值对纹理进行采样以拒绝样本。			x	x		
SampleCmpLevelZero	仅使用比较值来拒绝样本，对纹理 (mipmap 级别 0 采样) 。	x	x	x	x	x	x
SampleGrad	使用渐变对纹理进行采样，以影响计算样本位置的方式。	x	x	x	x	x	x
SampleLevel	对指定 mipmap 级别的纹理进行采样。	x	x	x	x	x	x

返回类型

除非另有指定，否则纹理对象方法的返回类型为 float4，但多采样反别名纹理对象除外，这些对象始终需要指定的类型和样本计数。 返回类型与纹理资源类型相同，(DXGI_FORMAT) 。 换句话说，它可以是以下任何类型。

类型	说明
FLOAT	32 位浮点 (请参阅 浮点规则 ，了解 IEEE 浮点) 的差异
int	32 位带符号整数
unsigned int	32 位无符号整数
snorm	范围 -1 到 1（含 32 位）的 32 位浮点 (请参阅浮点规则 ，了解 IEEE 浮点)
unorm	32 位浮点范围 0 到 1（含 1） (请参阅 浮点规则 ，了解 IEEE 浮点)
任何纹理类型或结构	返回的组件数必须介于 1 和 3 之间（含 1 到 3）。

此外，返回类型可以是包含结构的任何纹理类型，但它必须小于 4 个组件，例如返回一个组件的 float1 类型。

纹理中缺少组件的默认值

纹理资源类型中缺少组件的默认值为零，除 alpha 组件 (A) ;缺少 A 的默认值为 1。 此着色器显示的方式取决于纹理资源类型。 它采用实际存在于纹理资源类型中的第一个类型化组件的形式， (从 RGBA 顺序) 从左侧开始。 如果此窗体为 UNORM 或 FLOAT，则缺少 A 的默认值为 1.0f。 如果窗体为 SINT 或 UINT，则缺少 A 的默认值0x1。

例如，当着色器读取 DXGI_FORMAT_R24_UNORM_X8_TYPELESS 纹理资源类型时，G 和 B 的默认值为零，A 的默认值为 1.0f;当着色器读取 DXGI_FORMAT_R16G16_UINT 纹理资源类型时，B 的默认值为零，A 的默认值为0x00000001;当着色器读取 DXGI_FORMAT_R16_SINT 纹理资源类型时，G 和 B 的默认值为零，A 的默认值0x00000001。

示例 2

下面是使用纹理方法进行纹理采样的示例。

```
sampler MySamp;
Texture2D <float4> MyTex;

float4 main( float2 TexCoords[2] : TEXCOORD ) : SV_Target
{
```

```
    return MyTex.Sample( MySamp, TexCoords[0] ));
}
```

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 4 及更高着色器模型	是

另请参阅

着色器模型 4

CalculateLevelOfDetail (DirectX HLSL 纹理对象)

项目 • 2022/09/24

计算详细信息级别。

ret Object.CalculateLevelOfDetail (sampler_state S、 float x) ;

参数

项	说明
对象	除 Texture2DMS 和 Texture2DMSArray) 之外，任何 纹理对象 类型 (。
S	[in] 采样器状态。 这是在包含状态分配的效果文件中声明的对象。
X	[in]线性内插值或值，它是介于 0.0 和 1.0 之间的浮点数（含 1.0）。 组件数取决于纹理对象类型。 **Texture-Object类型** / **参数类型** Texture1D、Texture1DArray / float1 Texture2D、Texture2DArray / float2 Texture3D、TextureCube、TextureCubeArray / float3

返回值

返回计算的 LOD，一个浮点值。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
			x		

1. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。

2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

相关主题

Texture-Object

CalculateLevelOfDetailUnclamped (DirectX HLSL 纹理对象)

项目 • 2022/09/22

计算 LOD 而不固定结果。

```
ret Object.CalculateLevelOfDetailUnclamped ( sampler_state S， float x ) ;
```

此函数的工作方式与 CalculateLevelOfDetail (DirectX HLSL 纹理对象) 相同，只不过没有固定计算的 LOD。

相关主题

Texture-Object

收集 DirectX HLSL 纹理对象 ()

项目 • 2022/09/22

获取 (红色分量 (四个样本) ，该样本仅在采样纹理时用于双线性内插。

<Template Type4> Object.Gather (sampler_state S， float2|3|4 位置 [， int2 Offset]) ;

参数

<table>
<tr><th>项</th><th>描述</th></tr>
<tr><td>对象</td><td>支持以下 纹理对象 类型：Texture2D、Texture2DArray、TextureCube、TextureCubeArray。</td></tr>
<tr><td>S</td><td>[in] 采样器状态。 这是在包含状态分配的效果文件中声明的对象。</td></tr>
<tr><td>位置</td><td>[in]纹理坐标。 参数类型依赖于纹理对象类型。
<table>
<tr><th>Texture-Object类型</th><th>参数类型</th></tr>
<tr><td>Texture2D</td><td>float2</td></tr>
<tr><td>Texture2DArray，TextureCube</td><td>float3</td></tr>
<tr><td>TextureCubeArray</td><td>float4</td></tr>
</table></td></tr>
<tr><td>抵消</td><td>[in]可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 参数类型依赖于纹理对象类型。 对于面向着色器模型 5.0 及更高版本的着色器，每个偏移值的 6 个最小有效位被视为带符号值，从而产生 [-32..31] 范围。 对于以前的着色器模型着色器，偏移量必须是 -8 到 7 之间的即时整数。
<table>
<tr><th>Texture-Object类型</th><th>参数类型</th></tr>
<tr><td>Texture2D、Texture2DArray</td><td>int2</td></tr>
<tr><td>TextureCube、TextureCubeArray</td><td>不支持</td></tr>
</table></td></tr>
</table>

返回值

一个四分量向量，其中包含四个红色数据的组件，其类型与纹理的模板类型相同。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
	x		x		x

1. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

示例

```
Texture2D<int1> Tex2d;
Texture2DArray<int2> Tex2dArray;
TextureCube<int3> TexCube;
TextureCubeArray<float2> TexCubeArray;

SamplerState s;

int4 main (float4 f : SV_Position) : SV_Target
{
    int2 iOffset = int2(2,3);

    int4 i1 = Tex2d.Gather(s, f.xy);
    int4 i2 = Tex2d.Gather(s, f.xy, iOffset);

    int4 i3 = Tex2dArray.Gather(s, f.xyz);
    int4 i4 = Tex2dArray.Gather(s, f.xyz, iOffset);

    int4 i5 = TexCube.Gather(s, f.xyzw);

    float4 f6 = TexCubeArray.Gather(s, f.xyzw);

    return i1+i2+i3+i4+i5+int4(f6);
}
```

相关主题

Texture-Object

GetDimensions (DirectX HLSL 纹理对象)

项目 • 2022/09/22

获取纹理大小信息。 语法块显示方法声明中可能的所有参数。 “备注”部分中的表显示了为每个纹理对象类型实现的参数。

void Object.GetDimensions (UINT MipLevel， typeX Width， typeX Height， typeX Elements， typeX Depth， typeX NumberOfLevels， typeX NumberOfSamples) ;

typeX 表示有两种可能的类型： **uint** 或 **float**。

参数

项	说明
对象	除 **Buffer** 对象之外的任何纹理对象类型。
MipLevel	[in]一个从零开始的索引，用于标识 mipmap 级别。 如果未使用此参数，则假定第一个 mip 级别。
宽度	[out]纹理宽度（以纹素为单位）。
高度	[out]纹理高度（以纹素为单位）。
元素	[out]数组中的元素数。
深度	[out]纹理深度（以纹素为单位）。
NumberOfLevels	[out]mipmap 级别的数目。
NumberOfSamples	[out]样本数。

返回值

无

重载方法

此表列出方法的所有不同版本;版本因输入参数数而异。 请注意，对于采用整数参数的每个方法，都有一个重载的方法，该方法采用浮点参数。

Texture-Object类型	输入参数

Texture-Object类型	输入参数
Texture1D	UINT MipLevel、UINT Width、UINT NumberOfLevels
Texture1D	UINT 宽度
Texture1D	UINT MipLevel， float Width， float NumberOfLevels
Texture1D	float Width
Texture1DArray	UINT MipLevel、UINT Width、UINT 元素、UINT NumberOfLevels
Texture1DArray	UINT 宽度、UINT 元素
Texture1DArray	UINT MipLevel， float Width， float Elements， float NumberOfLevels
Texture1DArray	float Width、float Elements
Texture2D	UINT MipLevel、UINT Width、UINT Height、UINT NumberOfLevels
Texture2D	UINT 宽度、UINT 高度
Texture2D	UINT MipLevel， float Width， float Height， float NumberOfLevels
Texture2D	浮点宽度、浮点高度
Texture2DArray	UINT MipLevel、UINT Width、UINT Height、UINT 元素、UINT NumberOfLevels
Texture2DArray	UINT 宽度、UINT 高度、UINT 元素
Texture2DArray	UINT MipLevel， float Width， float Height， float Elements， float NumberOfLevels
Texture2DArray	float Width、float Height、float Elements
Texture3D	UINT MipLevel、UINT Width、UINT Height、UINT Depth、UINT NumberOfLevels
Texture3D	UINT 宽度、UINT 高度、UINT 深度
Texture3D	UINT MipLevel， float Width， float Height， float Depth， float NumberOfLevels
Texture3D	浮点宽度、浮点高度、浮点深度
TextureCube	UINT MipLevel、UINT Width、UINT Height、UINT NumberOfLevels
TextureCube	UINT 宽度、UINT 高度
TextureCube	UINT MipLevel， float Width， float Height， UINT NumberOfLevels

Texture-Object类型	输入参数
TextureCube	浮点宽度、浮点高度
TextureCubeArray	UINT MipLevel、UINT Width、UINT Height、UINT 元素、UINT NumberOfLevels
TextureCubeArray	UINT 宽度、UINT 高度、UINT 元素
TextureCubeArray	UINT MipLevel， float Width， float Height， float Elements， float NumberOfLevels
TextureCubeArray	float Width、float Height、float Elements
Texture2DMS	UINT 宽度、UINT 高度、UINT 示例
Texture2DMS	float Width、float Height、float Samples
Texture2DMSArray	UINT 宽度、UINT 高度、UINT 元素、UINT 示例
Texture2DMSArray	float Width、float Height、float Elements、float Samples

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
x	x	x	x	x	x

1. 返回最大 () mipmap 级别的维度。
2. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
3. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

相关主题

Texture-Object

GetSamplePosition (DirectX HLSL 纹理对象)

项目 • 2022/09/24

获取指定样本的位置。

ret Object.GetSamplePosition (int s) ;

参数

项	说明
对象	Texture2DMS 或 Texture2DMSArray texture-object 类型。
s	[in]从零开始的示例索引。

返回值

返回 (x，y) 样本位置，一个双分量的浮点向量。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
	x		x		x

- 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

注解

可以按采样频率评估像素着色器， (每个样本) 运行像素着色器一次，或者以像素频率计算， (每像素运行一次像素着色器) 。 将SV_SampleIndex语义附加到像素着色器输入，以采样频率调用像素着色器，然后在采样呈现目标时将输入值用作示例索引。

可以通过多种方式内插像素着色器输入。 若要在：

- 像素中心，不使用任何语义。
- 示例使用SV_SampleIndex语义。

- 质心位置，使用 _centroid 修饰符。

相关主题

Texture-Object

加载 (DirectX HLSL 纹理对象)

项目 • 2022/11/22

读取纹素数据，无需任何筛选或采样。

```
ret Object.Load (
typeX Location、
[typeX SampleIndex， ]
[typeX Offset ]
);
```

typeX 表示有四种可能的类型： **int**、 **int2**、 **int3** 或 **int4**。

参数

对象

纹理对象类型 (TextureCube 或 TextureCubeArray) 除外。

位置

[in]纹理坐标;最后一个组件指定 mipmap 级别。 此方法使用从 0 开始的坐标系，而不是 0.0-1.0 UV 系统。 参数类型依赖于纹理对象类型。

对象类型	参数类型
Buffer	int
Texture1D、Texture2DMS	int2
Texture1DArray、Texture2D、Texture2DMSArray	int3
Texture2DArray、Texture3D	int4

例如，若要访问 2D 纹理，请为前两个分量提供整数纹素坐标，为第三个分量提供 mipmap 级别。

备注

当 *Location* 中的一个或多个坐标超过纹理的 u、v 或 w mipmap 级别尺寸时， **Load** 在所有分量中返回零。 Direct3D 保证为超出边界访问的任何资源返回零。

SampleIndex

[in]采样索引。 对于多样本纹理是必需的。 不支持其他纹理。

纹理类型	**参数类型**
Texture1D、Texture1DArray、Texture2D、Texture2DArray、Texture3D、Texture2DArray、TextureCube、TextureCubeArray	不支持
Texture2DMS、Texture2DMSArray[1]	int

抵消

[in]采样前应用于纹理坐标的可选偏移量。 偏移类型取决于纹理对象类型，并且需要是静态的。

纹理类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray、Texture2DMS、Texture2DMSArray	int2
Texture3D	int3

备注

必须始终首先使用多样本纹理指定 *SampleIndex*。

返回值

返回类型与 *Object* 声明中的类型匹配。 例如，声明为“Texture2d<uint4> myTexture;”的 Texture2D 对象具有 **uint4** 类型的返回值。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1[1]	ps_4_0	ps_4_1[1]	gs_4_0	gs_4_1[1]
x	x	x	x	x	x

- 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

示例

此部分代码示例来自 AdvancedParticles 示例中 的 Paint.fx 文件。

```
// Object Declarations
Buffer<float4> g_ParticleBuffer;

// Shader body calling the intrinsic function
float4 PSPaint(PSQuadIn input) : SV_Target
{
    ...
    for( int i=g_ParticleStart; i<g_NumParticles; i+=g_ParticleStep )
    {
        ...
        // load the particle
        float4 particlePos = g_ParticleBuffer.Load( i*4 );
        float4 particleColor = g_ParticleBuffer.Load( (i*4) + 2 );
        ...
    }
    ...
}
```

相关主题

Texture-Object

示例 (DirectX HLSL Texture Object)

项目 • 2022/09/23

采样纹理。

<模板类型> Object.Sample (sampler_state S，浮点位置 [， int Offset]) ;

参数

项	说明
对象	除 Texture2DMS 和 Texture2DMSArray) 之外，任何 纹理对象 类型 (。
S	[in] 采样器状态。 这是在包含状态分配的效果文件中声明的对象。
位置	[in]纹理坐标。 参数类型依赖于纹理对象类型。 **Texture-Object类型** / **参数类型** Texture1D / FLOAT Texture1DArray、Texture2D / float2 Texture2DArray、Texture3D、TextureCube / float3 TextureCubeArray / float4
抵消	[in]可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 纹理偏移量必须是静态的。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用纹理坐标偏移量。 **Texture-Object类型** / **参数类型** Texture1D、Texture1DArray / int Texture2D、Texture2DArray / int2 Texture3D / int3 TextureCube、TextureCubeArray / 不支持

返回值

纹理的模板类型，可以是单组件或多组件向量。 格式基于纹理的 DXGI_FORMAT。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
		x	x		

1. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

示例

此部分代码示例基于 BasicHLSL11 示例中的 BasicHLSL11.fx 文件。

```
// Object Declarations
Texture2D g_MeshTexture;            // Color texture for mesh

SamplerState MeshTextureSampler
{
    Filter = MIN_MAG_MIP_LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};

struct VS_OUTPUT
{
    float4 Position   : SV_POSITION; // vertex position
    float4 Diffuse    : COLOR0;      // vertex diffuse color (note that
COLOR0 is clamped from 0..1)
    float2 TextureUV  : TEXCOORD0;   // vertex texture coords
};

VS_OUTPUT In;

// Shader body calling the intrinsic function
   ...
        Output.RGBColor = g_MeshTexture.Sample(MeshTextureSampler,
In.TextureUV) * In.Diffuse;
```

备注

纹理采样使用纹素位置查找纹素值。 可在查找之前将偏移量应用于位置。 采样器状态包含采样和筛选选项。 此方法可以在像素着色器中调用，但顶点着色器或几何着色器不支持此方法。

仅在整数错误位处使用偏移量;否则，可能会根据硬件实现或驱动程序设置获得不同的结果。

计算纹素位置

纹理坐标是引用纹理数据的浮点值，也称为规范化纹理空间。 按此顺序应用地址换行模式， (纹理坐标 + 偏移量 + 包装模式) 以修改 [0...1] 范围之外的纹理坐标。

对于纹理数组，location 参数中的附加值指定纹理数组中的索引。 此索引被视为缩放浮点值 (而不是标准纹理坐标的规范化空间) 。 对整数索引的转换按以下顺序完成， (float + 舍入到最接近的整数 + 固定到数组范围) 。

应用纹理坐标偏移量

偏移参数在纹素空间中修改纹理坐标。 即使纹理坐标是规范化的浮点数，偏移量也会应用整数偏移量。 另请注意，纹理偏移量必须是静态的。

返回的数据格式由纹理格式确定。 例如，如果使用 DXGI_FORMAT_A8B8G8R8_UNORM_SRGB格式定义纹理资源，则采样操作会将采样的纹素从 gamma 2.0 转换为 1.0，筛选并将结果写入为 [0..1] 范围内的浮点值。

相关主题

Texture-Object

SampleBias (DirectX HLSL 纹理对象)

项目 • 2022/09/23

将输入偏差应用于 mipmap 级别后，对纹理采样。

<Template Type> Object.SampleBias (sampler_state S、 float Location、 float Bias [, int Offset]) ;

参数

<table>
<tr><th>项</th><th>说明</th></tr>
<tr><td>对象</td><td>除 Texture2DMS 和 Texture2DMSArray) 之外，任何 纹理对象 类型 (。</td></tr>
<tr><td>S</td><td>[in] 采样器状态。 这是在包含状态分配的效果文件中声明的对象。</td></tr>
<tr><td>位置</td><td>[in]纹理坐标。 参数类型依赖于纹理对象类型。
<table>
<tr><th>Texture-Object类型</th><th>参数类型</th></tr>
<tr><td>Texture1D</td><td>FLOAT</td></tr>
<tr><td>Texture1DArray、Texture2D</td><td>float2</td></tr>
<tr><td>Texture2DArray、Texture3D、TextureCube</td><td>float3</td></tr>
<tr><td>TextureCubeArray</td><td>float4</td></tr>
</table></td></tr>
<tr><td>偏见</td><td>[in]偏差值（介于 -16.0 和 15.99 之间的浮点数）在采样前应用于 mip 级别。</td></tr>
<tr><td>抵消</td><td>[in]可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 纹理偏移量必须是静态的。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用纹理坐标偏移量。
<table>
<tr><th>Texture-Object类型</th><th>参数类型</th></tr>
<tr><td>Texture1D、Texture1DArray</td><td>int</td></tr>
<tr><td>Texture2D、Texture2DArray</td><td>int2</td></tr>
<tr><td>Texture3D</td><td>int3</td></tr>
<tr><td>TextureCube、TextureCubeArray</td><td>不支持</td></tr>
</table></td></tr>
</table>

返回值

纹理的模板类型，可以是单组件或多组件向量。 格式基于纹理的 DXGI_FORMAT。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
		x	x		

1. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

相关主题

Texture-Object

SampleCmp (DirectX HLSL 纹理对象)

项目 • 2022/09/22

对纹理采样，并将单个组件与指定的比较值进行比较。

```
float Object.SampleCmp (
SamplerComparisonState S，
float Location，
float CompareValue，
[int Offset]
);
```

比较是存储在纹理中的第一个组件与传递到方法中的比较值之间的单分量比较。

只能从像素着色器调用此方法;顶点或几何着色器不支持它。

参数

对象

除 Texture2DMS、Texture2DMSArray 或 Texture3D) 外，任何 纹理对象 类型 (。

S

[in]采样器比较状态，即采样器状态加上比较状态，(比较函数和比较筛选器) 。 有关详细信息和示例，请参阅 采样器类型 。

位置

[in]纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArrayM，TextureCube	float3
TextureCubeArrayM	float4

CompareValue

要用作比较值的浮点值。

抵消

[in]可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 纹理偏移量必须是静态的。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用纹理坐标偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArrayM	int2
TextureCube、TextureCubeArrayM	不支持

返回值

返回范围 [0..1] 中的浮点值。

对于根据筛选模式 () 采样器配置提取的每个纹素， **SampleCmp** 对着色器中输入) 的第三个分量 (z 值进行比较，如果比较通过，则从着色器 (1;否则为 0) 。 然后，**SampleCmp** 将每个纹素的 0 和 1 结果混合在一起，就像在普通纹理筛选中一样， (不是平均) ，并将生成的 [0..1] 值返回到着色器。

注解

比较筛选提供了一种基本筛选操作，可用于更深入的百分比筛选。

在浮点资源上使用此方法 (而不是) 带符号规范化或无符号规范化格式时，比较值不会在 0.0 和 1.0 之间自动固定。 因此，对于常见的阴影技术，可能需要手动固定比较值。

仅在整数位处使用偏移量;否则，可能会根据硬件实现或驱动程序设置获得不同的结果。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1 —	ps_4_0	ps_4_1 —	gs_4_0	gs_4_1 —
		xM	x		

1. Texture2DArray 和 TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

> **备注**
>
> 使用 Direct3D **功能级别 9 中**介绍的技术时，ps 4_0_level_9_1 和 4_0_level_9_3 中也提供了 **SampleCmp。**

相关主题

Texture-Object

SampleCmpLevelZero (DirectX HLSL 纹理对象)

项目 • 2022/09/23

对纹理进行采样并将结果与比较值进行比较。 此函数与仅在 mipmap 级别 0 上调用 **SampleCmp** 相同。

相关主题

Texture-Object

SampleGrad (DirectX HLSL 纹理对象)

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响样本位置的计算方式。

<Template Type> Object.SampleGrad (sampler_state S， float Location， float DDX，float DDY [， int Offset]) ;

参数

项	说明
对象	除 Texture2DMS 和 Texture2DMSArray) 之外，任何 纹理对象 类型 (。
S	[in] 采样器状态。 这是在包含状态分配的效果文件中声明的对象。
位置	[in]纹理坐标。 参数类型依赖于纹理对象类型。 **Texture-Object类型** / **参数类型** Texture1D / FLOAT Texture1DArray、Texture2D / float2 Texture2DArray、Texture3D、TextureCube / float3 TextureCubeArray / float4 Texture2DMS、Texture2DMSArray / 不支持
DDX	[in]x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。 **Texture-Object类型** / **参数类型** Texture1D、Texture1DArray / FLOAT Texture2D、Texture2DArray / float2 Texture3D、TextureCube、TextureCubeArray / float3 Texture2DMS、Texture2DMSArray / 不支持

项	说明
DDY	[in]y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。 **Texture-Object类型** / **参数类型** Texture1D、Texture1DArray / FLOAT Texture2D、Texture2DArray / float2 Texture3D、TextureCube、TextureCubeArray / float3 Texture2DMS、Texture2DMSArray / 不支持
抵消	[in]可选纹理坐标偏移量，可用于任何纹理对象类型。 偏移量将应用于采样之前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅应用整数偏移量。 **Texture-Object类型** / **参数类型** Texture1D、Texture1DArray / int Texture2D、Texture2DArray / int2 Texture3D / int3 TextureCube、TextureCubeArray / 不支持 Texture2DMS、Texture2DMSArray / 不支持

返回值

纹理的模板类型，可以是单组件向量或多组件向量。 格式基于纹理 **的**DXGI_FORMAT。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
x	x	x	x	x	x

1. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

示例

此部分代码示例来自 MotionBlur10 示例中的 MotionBlur.fx 文件。

```
// Object Declarations
Texture2D g_txDiffuse;

SamplerState g_samLinear
{
    Filter = ANISOTROPIC;
    MaxAnisotropy = 8;
    AddressU = Wrap;
    AddressV = Wrap;
};

struct VSSceneOut
{
    float4 Pos : SV_POSITION;
    float4 Color : COLOR0;
    float2 Tex : TEXCOORD;
    float2 Aniso : ANISOTROPY;
};

float4 PSSceneMain( VSSceneOut Input ) : SV_TARGET
{
    float2 ddx = Input.Aniso;
    float2 ddy = Input.Aniso;

    // Shader body calling the intrinsic function
    float4 diff = g_txDiffuse.SampleGrad( g_samLinear, Input.Tex, ddx, ddy);

    ...
}
```

相关主题

Texture-Object

SampleLevel (DirectX HLSL 纹理对象)

项目 • 2022/09/22

使用 mipmap 级别偏移对纹理进行采样。

<Template Type> Object.SampleLevel (sampler_state S、float Location、float LOD [, int Offset]) ;

此函数类似于 Sample ，只不过它在位置参数的最后一个组件中使用 LOD 级别 (,) 选择 mipmap 级别。 例如，2D 纹理使用前两个组件进行 uv 坐标，使用 mipmap 级别的第三个分量。

参数

<table>
<tr><th>项</th><th>说明</th></tr>
<tr><td>对象</td><td>除 Texture2DMS 和 Texture2DMSArray) 之外，任何 纹理对象 类型 (。</td></tr>
<tr><td>S</td><td>[in] 采样器状态。 这是在包含状态分配的效果文件中声明的对象。</td></tr>
<tr><td>位置</td><td>[in]纹理坐标。 参数类型依赖于纹理对象类型。
<table>
<tr><th>Texture-Object类型</th><th>参数类型</th></tr>
<tr><td>Texture1D</td><td>FLOAT</td></tr>
<tr><td>Texture1DArray、Texture2D</td><td>float2</td></tr>
<tr><td>Texture2DArray、Texture3D、TextureCube</td><td>float3</td></tr>
<tr><td>TextureCubeArray</td><td>float4</td></tr>
</table>
如果纹理对象是数组，则最后一个组件是数组索引。</td></tr>
<tr><td>LOD</td><td>[in]一个指定 mipmap 级别的数字。 如果值为 = 0，则使用第 (个最大映射) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。</td></tr>
</table>

项	说明
抵消	[in]可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 纹理偏移量必须是静态的。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用纹理坐标偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

返回值

纹理的模板类型，可以是单组件向量或多组件向量。 格式基于纹理 **的DXGI_FORMAT**。

最小着色器模型

以下着色器模型中支持此函数。

vs_4_0	vs_4_1	ps_4_0	ps_4_1	gs_4_0	gs_4_1
x	x	x	x	x	x

1. TextureCubeArray 在着色器模型 4.1 或更高版本中可用。
2. 着色器模型 4.1 在 Direct3D 10.1 或更高版本中可用。

示例

此部分代码示例来自 Instancing10 示例中的 Instancing.fx 文件。

```
// Object Declarations
Texture1D g_txRandom;

SamplerState g_samPoint
{
    Filter = MIN_MAG_MIP_POINT;
    AddressU = Wrap;
```

```
    AddressV = Wrap;
};


// Shader body calling the intrinsic function
float3 RandomDir(float fOffset)
{
    float tCoord = (fOffset) / 300.0;
    return g_txRandom.SampleLevel( g_samPoint, tCoord, 0 );
   ...
```

相关主题

Texture-Object

常量变量的打包规则

项目 • 2022/09/22

打包规则决定了在存储数据时可以排列得有多紧密。 HLSL 为 VS 输出数据、GS 输入和输出数据以及 PS 输入和输出数据实现打包规则。 (数据未打包用于 VS 输入，因为 IA 阶段无法解压缩 data.)

HLSL 打包规则类似于使用 **Visual Studio 执行#pragma包 4**，后者将数据打包成 4 字节边界。 此外，HLSL 会打包数据，使其不跨越 16 字节边界。 变量打包到给定的四分量向量中，直到变量将跨越 4 向量边界：下一个变量将弹跳到下一个四分量向量。

每个结构强制下一个变量从下一个四分量向量开始。 这有时会为结构数组生成填充。 任何结构的结果大小始终通过大小 (*四分量向量*) 均匀地分割。

默认情况下，数组不会打包在 HLSL 中。 为了避免强制着色器对偏移计算产生 ALU 开销，数组中的每个元素都存储在四分量向量中。 请注意，可以使用强制转换实现数组的打包 (并产生寻址计算) 。

下面是给定结构及其相应打包 (大小的示例： **float1** 占用 4 个字节) ：

```
//  2 x 16byte elements
cbuffer IE
{
    float4 Val1;
    float2 Val2;  // starts a new vector
    float2 Val3;
};

//  3 x 16byte elements
cbuffer IE
{
    float2 Val1;
    float4 Val2;  // starts a new vector
    float2 Val3;  // starts a new vector
};

//  1 x 16byte elements
cbuffer IE
{
    float1 Val1;
    float1 Val2;
    float2 Val3;
};

//  1 x 16byte elements
cbuffer IE
```

```
{
    float1 Val1;
    float2 Val2;
    float1 Val3;
};

//  2 x 16byte elements
cbuffer IE
{
    float1 Val1;
    float1 Val1;
    float1 Val1;
    float2 Val2;    // starts a new vector
};


//  1 x 16byte elements
cbuffer IE
{
    float3 Val1;
    float1 Val2;
};

//  1 x 16byte elements
cbuffer IE
{
    float1 Val1;
    float3 Val2;
};

//  2 x 16byte elements
cbuffer IE
{
    float1 Val1;
    float1 Val1;
    float3 Val2;        // starts a new vector
};


// 3 x 16byte elements
cbuffer IE
{
    float1 Val1;

    struct     {
        float4 SVal1;    // starts a new vector
        float1 SVal2;    // starts a new vector
    } Val2;
};

// 3 x 16byte elements
cbuffer IE
{
    float1 Val1;
    struct     {
```

```
        float1 SVal1;     // starts a new vector
        float4 SVal2;     // starts a new vector
    } Val2;
};

// 3 x 16byte elements
cbuffer IE
{
    struct     {
        float4 SVal1;
        float1 SVal2;    // starts a new vector
    } Val1;

    float1 Val2;
};
```

更激进的打包

可以更积极地打包数组。 例如，给定浮点变量数组：

```
float4 array[16];
```

可以选择像这样打包它，而无需数组中的任何空格：

```
static float2 aggressivePackArray[32] = (float2[32])array;
```

更严格的包装是一种权衡，而需要额外的着色器指令进行地址计算。

相关主题

着色器模型 4

寄存器 - ps_4_0

项目 • 2022/09/23

本部分包含像素着色器版本 4_0 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
R#		4096 (r#+x#[n])	R/W	4	否	无	是
x#[n]		4096 (r#+x#[n])	R/W	4	是	无	是
V#		32	R	4	是	无	是
t#		128	R	1	否	无	是
s#		16	R	1	否	无	是
cb#[index]		15	R	4	是 (内容)	无	是
icb[index]		1	R	4	是 (内容)	无	是

输出寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
Null	放弃结果	空值	W	不适用	不可用	空值	否
o#	输出寄存器	8	W	空值	不适用	4	否
oDepth	输出深度	1	W	空值	不可用	1	不适用

相关主题

着色器模型 4

寄存器 - ps_4_1

项目 • 2022/09/22

本部分包含像素着色器版本 4_1 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
R#		4096 (r#+x#[n])	R/W	4	否	无	是
x#[n]		4096 (r#+x#[n])	R/W	4	是	无	是
V#		32	R	4	是	无	是
t#		128	R	1	否	无	是
s#		16	R	1	否	无	是
cb#[index]		15	R	4	是 (内容)	无	是
icb[index]		1	R	4	是 (内容)	无	是

输出寄存器

注册	名称	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
Null	放弃结果	空值	W	空值	空值	空值	否
o#	输出寄存器	8	W	空值	不适用	4	否
oDepth	输出深度	1	W	空值	空值	1	不适用
oMask	输出 MSAA 掩码	1	W	空值	空值	1	不适用

相关主题

着色器模型 4

寄存器 - vs_4_0

项目 • 2022/09/23

本部分包含顶点着色器版本 4_0 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
R#		4096 (r#+x#[n])	R/W	4	否	无	是
x#[n]		4096 (r#+x#[n])	R/W	4	是	无	是
V#		16	R	4	是	无	是
t#		128	R	1	否	无	是
s#		16	R	1	否	无	是
cb#[index]		15	R	4	是 (内容)	无	是
icb[index]		1	R	4	是 (内容)	无	是

输出寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
Null	放弃结果	空值	W	空值	不可用	空值	否
o#	输出寄存器	16	W	空值	不适用	4	是

相关主题

着色器模型 4

寄存器 - vs_4_1

项目 • 2022/09/22

本部分包含顶点着色器版本 4_1 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
R#		4096 (r#+x#[n])	R/W	4	否	无	是
x#[n]		4096 (r#+x#[n])	R/W	4	是	无	是
V#		32	R	4	是	无	是
t#		128	R	1	否	无	是
s#		16	R	1	否	无	是
cb#[index]		15	R	4	是 (内容)	无	是
icb[index]		1	R	4	是 (内容)	无	是

输出寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
Null	放弃结果	空值	W	空值	不适用	空值	否
o#	输出寄存器	32	W	空值	不适用	4	是

相关主题

着色器模型 4

寄存器 - gs_4_0

项目 • 2022/09/23

本部分包含几何着色器版本 4_0 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
R#		4096 (r#+x#[n])	R/W	4	否	无	是
x#[n]		4096 (r#+x#[n])	R/W	4	是	无	是
v#[顶点][元素]		32	R	4 (comp) *6 (转)	是	无	是
vprim		1	R	1	否	无	是
t#		128	R	1	否	无	是
s#		16	R	1	否	无	是
cb#[index]		15	R	4	是 (内容)	无	是
icb[index]		1	R	4	是 (内容)	无	是

输出寄存器

注册	名称	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
Null	放弃结果	空值	W	空值	空值	空值	否
o#	输出寄存器	32	W	空值	不适用	4	是

相关主题

着色器模型 4

寄存器 - gs_4_1

项目 • 2022/09/24

本部分包含由几何着色器版本 4_0 和 4_1 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
R#		4096 (r#+x#[n])	R/W	4	否	无	是
x#[n]		4096 (r#+x#[n])	R/W	4	是	无	是
v#[顶点][元素]		32	R	4 (comp) *6 ()	是	无	是
vprim		1	R	1	否	无	是
t#		128	R	1	否	无	是
s#		16	R	1	否	无	是
cb#[index]		15	R	4	是 (内容)	无	是
icb[index]		1	R	4	是 (内容)	无	是

输出寄存器

注册	名称	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
Null	放弃结果	不适用	W	不适用	不适用	不适用	否
o#	输出寄存器	32	W	不适用	不适用	4	是

相关主题

着色器模型 4

着色器模型 5

项目 • 2022/09/22

本部分包含 HLSL 着色器模型 5 的参考页。

着色器模型 5 是 着色器模型 4 中功能的超集。 它已使用通用着色器核心进行设计，它为所有可编程着色器提供一组通用功能，这些着色器仅使用 HLSL 进行可编程。

特性	功能
指令集	**HLSL 内部函数**
顶点着色器最大值	无限制
像素着色器最大值	无限制
添加了新的着色器配置文件	cs_4_0、gs_4_0*、ps_4_0*、vs_4_0*、cs_4_1、gs_4_1*、ps_4_1*、vs_4_1*、cs_5_0、ds_5_0、gs_5_0、hs_5_0、ps_5_0、vs_5_0

* - 着色器模型 4.0 中引入了gs_4_0、gs_4_1、ps_4_0、ps_4_1、vs_4_0和vs_4_1，但 DirectX 11 向 DirectX 10 硬件上运行的着色器模型 4 添加了对 结构化缓冲区 和字节地址缓冲区的支持。

着色器模型 5 引入了提供高速通用计算的 计算着色器 。

着色器模型 5 功能的完整列表包含在 Direct3D 11 功能的列表中。

着色器模型 5 程序集部分介绍了着色器模型 5 支持的程序集说明。

本节内容

项	说明
着色器模型 5 属性	着色器模型 5 属性的参考页。
着色器模型 5 内部函数	着色器模型 5 内部函数的参考页。
着色器模型 5 对象	着色器模型 5 对象和方法的参考页。
着色器模型 5 系统值	着色器模型 5 系统值的参考页。

相关主题

着色器模型与着色器配置文件

着色器模型 5 属性

项目 • 2022/09/22

着色器模型 5 从着色器模型 4 和以下 (查看 DirectX SDK) 以及以下新属性：

- domain
- earlydepthstencil
- 实例
- maxtessfactor
- numthreads
- outputcontrolpoints
- outputtopology
- 分区
- patchconstantfunc

除 **实例** 属性) 外，所有这些属性都需要 (。 无法使用它们将导致编译错误。

相关主题

着色器模型 5

域

项目 • 2022/09/22

定义 HS 中使用的修补程序类型。

```
domain(X)
```

备注

X 为 **tri**、 **quad** 或 **isoline**。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
	x	x			

相关主题

着色器模型 5 属性

着色器模型 5

earlydepthstencil

项目 • 2022/12/14

在执行着色器之前强制进行深度模具测试。

```
earlydepthstencil
```

注解

深度模具测试通常由像素着色器在像素处理期间完成。 强制进行早期深度模具测试会导致在着色器执行之前完成测试;目的是通过剔除/减少/避免不必要的像素处理来提高每像素性能。

应用程序可以通过提供属性来强制进行早期深度模具测试，或者硬件可以执行早期深度模具测试，前提是没有写入深度值，并且着色器中不会执行无序访问操作作为优化。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
				x	

相关主题

着色器模型 5 属性

着色器模型 5

实例

项目 • 2022/09/23

使用此属性来实例几何着色器。

```
instance(X)
```

注解

X 是一个整数索引，指示要为每个绘制项执行的实例数 (例如，每个三角形) 。 使用此属性时，请使用 SV_GSInstanceID 标识要执行的几何着色器的实例。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
			x		

相关主题

着色器模型 5 属性

着色器模型 5

maxtessfactor

项目 • 2022/09/23

指示外壳着色器将返回任何细化因子的最大值。

```
maxtessfactor(X)
```

注解

此属性对请求的分割量施加上限，以帮助驱动程序确定分割所需的最大资源量。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
	x				

相关主题

着色器模型 5 属性

着色器模型 5

numthreads

项目 • 2022/09/24

定义在调度计算着色器时要在单个线程组中执行的线程数，（请参阅 ID3D11DeviceContext：:D ispatch）。

```
numthreads(X, Y, Z)
```

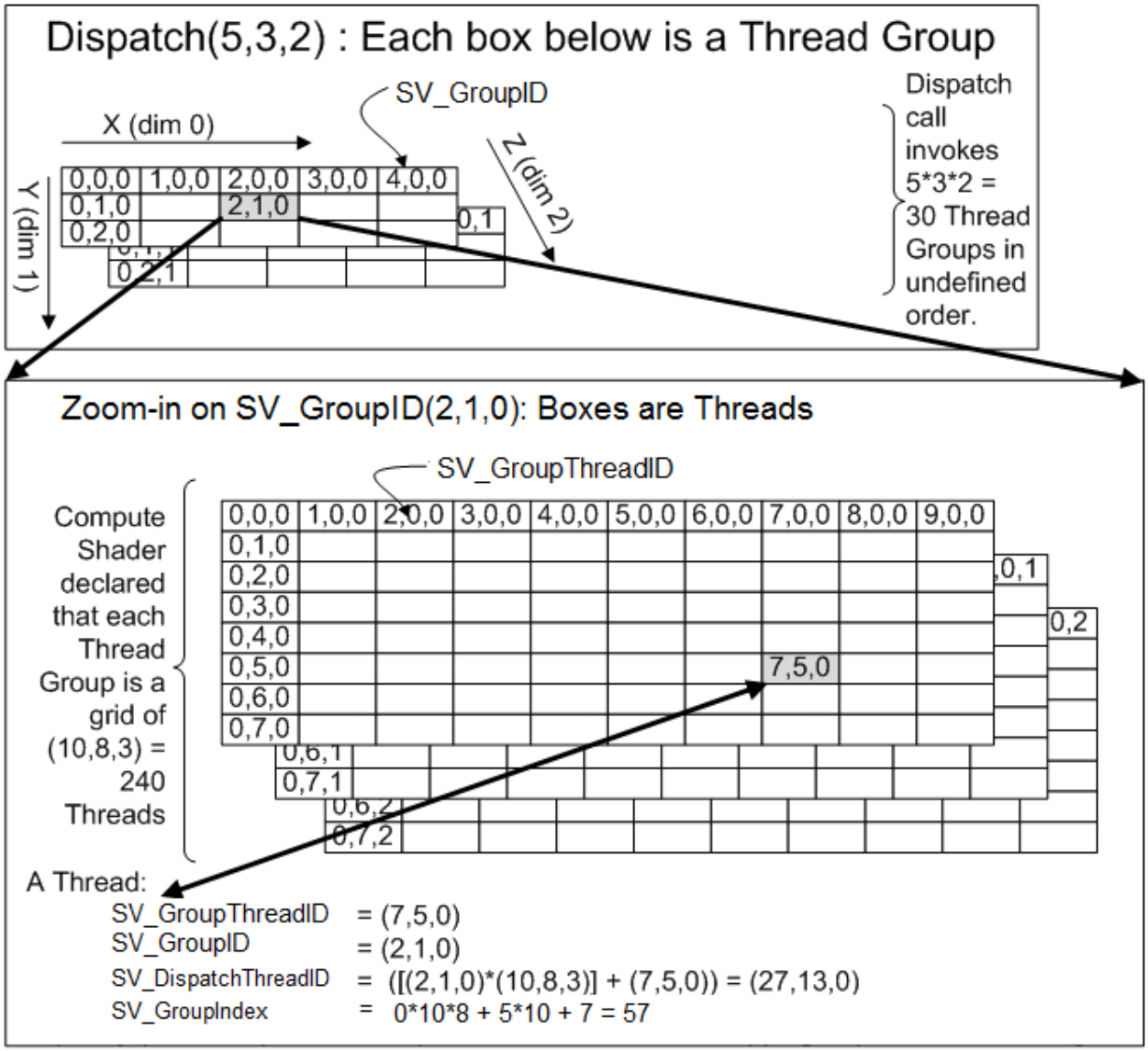
X、Y 和 Z 值指示特定方向的线程组的大小，X*Y*Z 的总数为组中的线程数。 指定跨三个维度的线程组大小的功能允许以逻辑 2D 和 3D 数据结构的方式访问各个线程。

例如，如果计算着色器正在添加 4x4 矩阵，则可以将数字线程设置为 numthreads (4，4，1) ，并且单个线程中的索引将自动匹配矩阵项。 计算着色器还可以使用数字线程 (16，1，1) 来声明线程组 (16，16) ，但随后必须基于当前线程数计算当前矩阵条目。

numthread 的可允许参数值取决于计算着色器版本。

计算着色器	最大 Z	最大线程数 (X*Y*Z)
cs_4_x	1	768
cs_5_0	64	1024

下图显示了传递给 ID3D11DeviceContext：:D ispatch 的参数之间的关系。 调度 (5，3，2) 、numthreads 属性中指定的值、numthreads (10，8，3) ，以及传递给线程相关系统值的计算着色器的值，（SV_GroupIndex，SV_DispatchThreadID，SV_GroupThreadID，SV_GroupID）。



以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
					x

相关主题

着色器模型 5 属性

着色器模型 5

outputcontrolpoints

项目 • 2022/09/22

定义将在外壳着色器中创建的每个线程 (输出控制点数) 。

```
outputcontrolpoints(X)
```

注解

这是执行主函数的次数。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
	x				

相关主题

着色器模型 5 属性

着色器模型 5

outputtopology

项目 • 2022/09/23

定义细化器的输出基元类型。

```
outputtopology(X)
```

备注

X 必须是 **点**、 **行**、 **triangle_cw**或 **triangle_ccw** ，并且必须位于引号内。 下面是此属性的有效选项：

```
[outputtopology("point")]
[outputtopology("line")]
[outputtopology("triangle_cw")]
[outputtopology("triangle_ccw")]
```

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
	x				

相关主题

着色器模型 5 属性

着色器模型 5

partitioning

项目 • 2022/09/22

定义要在外壳着色器中使用的细化方案。

```
partitioning(X)
```

备注

可以是 **整数**、 fractional_even、 fractional_odd或 pow2。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
	x				

相关主题

着色器模型 5 属性

着色器模型 5

patchconstantfunc

项目 • 2022/12/14

定义用于计算修补常量数据的函数。

```
patchconstantfunc("function_name")
```

注解

function_name 是输出修补程序常量数据的单独函数的名称。

以下类型的着色器支持此属性：

顶点	外壳	域	几何结构	像素	计算
	x				

相关主题

着色器模型 5 属性

着色器模型 5

着色器模型 5 内部函数

项目 • 2022/09/23

着色器模型 5 从着色器模型 4 和以下实现内部函数，（请参阅 **内部函数 (DirectX HLSL)** ，获取受支持的函数的完整列表），以及以下新函数：

- AllMemoryBarrier
- AllMemoryBarrierWithGroupSync
- asdouble
- asuint
- countbits
- ddx_coarse
- ddy_coarse
- DeviceMemoryBarrier
- DeviceMemoryBarrierWithGroupSync
- EvaluateAttributeCentroid
- EvaluateAttributeAtSample
- f16tof32
- f32tof16
- firstbithigh
- firstbitlow
- fma
- GroupMemoryBarrier
- GroupMemoryBarrierWithGroupSync
- InterlockedAdd
- InterlockedAnd
- InterlockedCompareExchange
- InterlockedCompareStore
- InterlockedExchange
- InterlockedMax
- InterlockedMin
- InterlockedOr
- InterlockedXor
- msad4
- Process2DQuadTessFactorsAvg
- Process2DQuadTessFactorsMax
- Process2DQuadTessFactorsMin
- ProcessIsolineTessFactors
- ProcessQuadTessFactorsAvg
- ProcessQuadTessFactorsMax

- ProcessQuadTessFactorsMin
- ProcessTriTessFactorsAvg
- ProcessTriTessFactorsMax
- ProcessTriTessFactorsMin
- rcp
- reversebits

相关主题

着色器模型 5

着色器模型 5 对象

项目 • 2022/09/22

着色器模型 5 从着色器模型 4 和以下实现内部函数 (查看 DirectX SDK) ，以及这些新函数：

- AppendStructuredBuffer
- Buffer
- ByteAddressBuffer
- ConsumeStructuredBuffer
- InputPatch
- OutputPatch
- RWBuffer
- RWByteAddressBuffer
- RWStructuredBuffer
- RWTexture1D
- RWTexture1DArray
- RWTexture2D
- RWTexture2DArray
- RWTexture3D
- StructuredBuffer
- Texture1D
- Texture1DArray
- Texture2D
- Texture2DArray
- Texture2DMS
- Texture2DMSArray
- Texture3D
- TextureCube
- TextureCubeArray

相关主题

着色器模型 5

AppendStructuredBuffer

项目 • 2022/09/24

显示为着色器可能追加到的流的输出缓冲区。 只有结构化缓冲区才能采用结构中的 T 类型。

方法	说明
追加	将值追加到缓冲区的末尾。
GetDimensions	获取资源维度。

需要使用DXGI_FORMAT_UNKNOWN格式创建绑定到此资源的 UAV 格式。

绑定到此资源的 UAV 必须使用 D3D11_BUFFER_UAV_FLAG_APPEND创建。

有关追加结构化缓冲区的详细信息，请参阅这两个部分： 追加和使用缓冲区 和 结构化缓冲区。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

着色器模型 5 对象

Append 函数

项目 • 2022/09/24

将值追加到缓冲区末尾。

语法

```
syntax
```

```
void Append(
  in T value
);
```

参数

值 [in]

类型： **T**

输入值。

返回值

无

备注

T 可以是任何数据类型，包括结构。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

AppendStructuredBuffer

AppendStructuredBuffer：：GetDimensions 函数

项目 • 2022/09/22

获取资源维度。

语法

syntax

```
void GetDimensions(
  out uint numStructs,
  out uint stride
);
```

parameters

numStructs [out]

类型： **uint**

结构数。

stride [out]

类型： **uint**

每个元素中的字节数。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

AppendStructuredBuffer

着色器模型 5

Buffer

项目 • 2022/09/23

着色器模型 4 中存在的缓冲区类型以及资源变量和缓冲区信息。

方法	说明
GetDimensions	获取资源维度。
加载	读取缓冲区数据。
运算符[]	获取只读资源变量。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Buffer： ：GetDimensions 函数

项目 • 2023/03/29

着色器资源视图中设置的 Buffer 的长度（以元素为单位）。

语法

syntax

```
void GetDimensions(
  out uint dim
);
```

参数

dim [out]

类型： **uint**

缓冲区的长度（以字节为单位）。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

请参阅

Buffer

着色器模型 5

Buffer： ：Load 方法

项目 • 2022/09/22

读取 **缓冲区** 数据。

重载列表

方法	说明
加载 (int、uint)	读取缓冲区数据并返回操作的状态。
加载 int) (	读取缓冲区数据。

另请参阅

Buffer

Buffer：：Load (int，uint) 函数

项目 • 2022/09/22

读取缓冲区数据并返回操作的状态。

语法

```
syntax
```

```
 Load(
  in  int Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Buffer 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

示例

此示例演示如何使用 **Load**：

syntax

```
Buffer<float4> myBuffer;
float loc;
uint status;
float4 myColor = myBuffer.Load( loc , status );
```

另请参阅

加载方法

Buffer： ：Operator 函数

项目 • 2022/09/22

返回只读资源变量。

语法

```
syntax

R Operator[](
  in uint pos
);
```

parameters

pos [in]

类型： **uint**

索引位置。

返回值

类型： **R**

只读资源变量。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Buffer

ByteAddressBuffer

项目 • 2022/09/22

以字节为单位编制索引的只读缓冲区。

方法	说明
GetDimensions	获取资源维度。
加载	获取一个值。
Load2	获取两个值。
Load3	获取三个值。
Load4	获取四个值。

使用原始缓冲区时，可以使用 **ByteAddressBuffer** 对象类型。 有关缓冲区的原始查看的详细信息，请参阅 缓冲区的原始视图。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色 器模型着色器模型 4 (可通过 Direct3D 11 API 使用，方法是在支持计算着色器的设备上使用 10.0 或 10.1 功能级别 (D3D_FEATURE_LEVEL_10_X) 。 有关下层硬件上的计算着色器支持的详细信息，请参阅 Downlevel Hardware.)	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

有关字节地址缓冲区的详细信息，请参阅 字节可寻址资源类型。

着色器模型 5 还实现 读写字节地址缓冲区。

另请参阅

着色器模型 5 对象

ByteAddressBuffer：：GetDimensions 函数

项目 • 2022/09/22

获取缓冲区的长度。

语法

syntax

```
void GetDimensions(
  out uint dim
);
```

参数

dim [out]

类型： **uint**

缓冲区的长度（以字节为单位）。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

ByteAddressBuffer

加载方法

项目 • 2022/09/22

从以字节为单位的只读缓冲区中获取一个值。

重载列表

方法	说明
加载 int、uint) (	获取一个值和操作的状态。
加载 int) (	获取一个值。

另请参阅

ByteAddressBuffer

ByteAddressBuffer： ：Load (int) 函数

项目 • 2022/09/24

获取一个值。

语法

syntax

```
uint Load(
  in int address
);
```

parameters

地址 [in]

类型： **int**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint**

一个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

加载方法

ByteAddressBuffer：：Load (int， uint) 函数

项目 • 2022/09/23

获取一个值和操作的状态。

语法

syntax

```
uint Load(
  in  int Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

输入地址（以字节为单位），该地址必须是 4 的倍数。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint**

一个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

加载方法

ByteAddressBuffer

Load2 方法

项目 • 2022/09/22

从以字节为单位的只读缓冲区中获取两个值。

重载列表

方法	说明
Load2 (uint)	获取两个值。
Load2 (uint、uint)	获取两个值和操作的状态。

另请参阅

ByteAddressBuffer

ByteAddressBuffer：：Load2 (uint) 函数

项目 • 2022/09/22

获取两个值。

语法

```
syntax

uint2 Load2(
  in uint address
);
```

parameters

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint2**

两个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Load2 方法

Load2 (uint、uint) 函数

项目 • 2022/09/23

获取两个值和操作的状态。

语法

```
syntax
```

```
uint2 Load2(
  in  uint Location,
  out uint Status
);
```

parameters

位置 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint2**

两个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Load2 方法

ByteAddressBuffer

Load3 方法

项目 • 2022/09/23

从以字节为单位编制索引的只读缓冲区中获取三个值。

重载列表

方法	说明
Load3 (uint)	获取三个值。
Load3 (uint、uint)	获取三个值和操作的状态。

另请参阅

ByteAddressBuffer

ByteAddressBuffer：：Load3 (uint) 函数

项目 • 2022/09/24

获取三个值。

语法

syntax

```
uint3 Load3(
  in uint address
);
```

parameters

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint3**

三个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Load3 方法

Load3 (uint、uint) 函数

项目 • 2022/09/22

获取三个值和操作的状态。

语法

```
syntax
```

```
uint3 Load3(
  in  uint Location,
  out uint Status
);
```

参数

位置 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint3**

三个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Load3 方法

ByteAddressBuffer

Load4 方法

项目 • 2022/09/24

从以字节为单位的只读缓冲区中获取四个值。

重载列表

方法	说明
Load4 (uint)	获取四个值。
Load4 (uint、uint)	获取四个值和操作的状态。

另请参阅

ByteAddressBuffer

ByteAddressBuffer：：Load4 (uint) 函数

项目 • 2022/09/22

获取四个值。

语法

```
syntax

uint4 Load4(
  in uint address
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint4**

四个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Load4 方法

Load4 (uint、uint) 函数

项目 • 2022/09/22

获取四个值和操作的状态。

语法

```
syntax
```

```
uint4 Load4(
  in  uint Location,
  out uint Status
);
```

parameters

位置 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint4**

四个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Load4 方法

ByteAddressBuffer

ConsumeStructuredBuffer

项目 • 2022/09/22

作为流显示的输入缓冲区，着色器可能会从中拉取值。 只有结构化缓冲区才能采用结构中的 T 类型。

方法	说明
消费	从缓冲区末尾移除值。
GetDimensions	获取资源维度。

需要使用DXGI_FORMAT_UNKNOWN格式创建绑定到此资源的 UAV 格式。

绑定到此资源的 UAV 必须使用 D3D11_BUFFER_UAV_FLAG_APPEND创建。

有关使用结构化缓冲区的详细信息，请参阅这两个部分： 追加和使用缓冲区 和 结构化缓冲区。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

使用函数

项目 • 2022/09/24

从缓冲区末尾删除值。

语法

```
syntax

T Consume(void);
```

parameters

此函数没有参数。

返回值

类型： **T**

(删除的值可以是结构) 。

注解

T 可以是任何数据类型，包括结构。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

ConsumeStructuredBuffer

着色器模型 5

ConsumeStructuredBuffer：：GetDimensions 函数

项目 • 2022/09/22

获取资源维度。

语法

syntax

```
void GetDimensions(
  out uint numStructs,
  out uint stride
);
```

parameters

numStructs [out]

类型： **uint**

结构数。

stride [out]

类型： **uint**

每个元素的步幅（以字节为单位）。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

ConsumeStructuredBuffer

着色器模型 5

InputPatch

项目 • 2022/09/22

表示作为输入的外壳着色器可用的控制点数组。

方法	说明
运算符[]	获取只读资源变量。

InputPatch 类还支持以下属性：

属性	类型	说明
长度	uint	控制点数。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
	x		x		

另请参阅

着色器模型 5 对象

InputPatch：：Operator 函数

项目 • 2022/09/24

返回修补程序中的*第 n*个控制点。

语法

```
syntax
```

```
T Operator[](
  in uint n
);
```

参数

n [in]

类型： **uint**

输入索引。

返回值

类型： **T**

第 *n*个控制点。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

InputPatch

OutputPatch

项目 • 2022/09/22

表示可用于外壳着色器的修补常量函数以及域着色器的输出控制点数组。

方法	说明
运算符[]	获取只读资源变量。

此外，InputPatch 类还支持以下属性：

属性	类型	描述
长度	uint	控制点数。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
	x	x			

另请参阅

着色器模型 5 对象

OutputPatch：：Operator 函数

项目 • 2022/09/24

返回修补程序中的*第 n*个控制点。

语法

```
syntax
```

```
T Operator[](
  in uint n
);
```

parameters

n [in]

类型： **uint**

输入索引。

返回值

类型： **T**

第 *n*个控制点。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x	x			

另请参阅

OutputPatch

RWBuffer

项目 • 2022/09/22

读/写缓冲区。

方法	说明
GetDimensions	获取资源维度。
加载	获取读写缓冲区中的一个值。
运算符[]	返回资源变量。

还可以将资源变量传递到任何无序或互锁操作中。

RWBuffer 对象可以前缀为存储类 **全局一致性**。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅在当前组中刷新 UAV。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWBuffer：：Load 方法

项目 • 2022/09/23

从 **RWBuffer** 获取一个值。

重载列表

方法	说明
加载 int、uint) (	读取缓冲区数据并返回操作的状态。
加载 int) (	读取缓冲区数据。

另请参阅

RWBuffer

RWBuffer：：Load (int) 函数

项目 • 2022/09/24

读取缓冲区数据。

语法

syntax

```
 Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

返回值

类型：

返回类型与 **RWBuffer** 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

加载方法

RWBuffer：：Load (int，uint) 函数

项目 • 2022/09/23

读取缓冲区数据并返回操作的状态。

语法

```
syntax
```

```
 Load(
  in  int  Location,
  out uint Status
);
```

parameters

位置 [in]

类型： **int**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWBuffer 对象的声明中的类型匹配。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

请参阅

加载方法

RWBuffer：：GetDimensions 函数

项目 • 2023/03/29

获取缓冲区的长度。

语法

syntax

```
void GetDimensions(
  out uint dim
);
```

参数

dim [out]

类型： **uint**

无序资源视图中设置的 Buffer 的长度（以元素为单位）。

返回值

无

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

请参阅

RWBuffer

着色器模型 5

RWBuffer：：Operator 函数

项目 • 2022/09/22

返回资源变量。

语法

syntax

```
R Operator[](
  in uint pos
);
```

参数

pos [in]

类型： **uint**

索引位置。

返回值

类型： **R**

资源变量。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

RWBuffer

RWByteAddressBuffer

项目 • 2022/09/23

索引（以字节为单位）的读/写缓冲区。

方法	说明
GetDimensions	获取资源维度。
InterlockedAdd	以原子方式添加。
InterlockedAnd	AND，从原子上讲。
InterlockedCompareExchange	以原子方式比较和交换。
InterlockedCompareStore	以原子方式比较和存储。
InterlockedExchange	以原子方式交换。
InterlockedMax	以原子方式查找最大值。
InterlockedMin	以原子方式查找最小值。
InterlockedOr	OR，从原子上讲。
InterlockedXor	XOR，从原子上讲。
加载	获取一个值。
Load2	获取两个值。
Load3	获取三个值。
Load4	获取四个值。
存储	设置一个值。
Store2	设置两个值。
Store3	设置三个值。
Store4	设置四个值。

RWByteAddressBuffer 对象可以前缀为存储类 **globallycoherent**。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅在当前组中刷新 UAV。

需要使用DXGI_FORMAT_R32_TYPELESS格式创建绑定到此资源的 UAV 格式。

必须已使用 D3D11_BUFFER_UAV_FLAG_RAW创建绑定到此资源的 UAV。

使用原始缓冲区时，可以使用 **RWByteAddressBuffer** 对象类型。 有关缓冲区的原始查看的详细信息，请参阅 缓冲区的原始视图。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色 器模型着色器模型 4 (可通过 Direct3D 11 API 使用，方法是在支持计算着色器的设备上使用 10.0 或 10.1 功能级别 **(D3D_FEATURE_LEVEL**_10_X) 。 有关下层硬件上的计算着色器支持的详细信息，请参阅 Downlevel Hardware 上的计算着色器。)	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWByteAddressBuffer：：GetDimensions 函数

项目 • 2022/09/22

获取缓冲区的长度。

语法

syntax

```
void GetDimensions(
  out uint dim
);
```

参数

dim [out]

类型： **uint**

缓冲区的长度（以字节为单位）。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

RWByteAddressBuffer

InterlockedAdd 函数

项目 • 2022/09/22

以原子方式添加值。

语法

syntax

```
void InterlockedAdd(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： **UINT**

目标地址。

值 [in]

类型： **UINT**

输入值。

original_value [out]

类型： **UINT**

原始值。

返回值

此函数不返回值。

注解

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行原子值加法。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行值原子添加。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数会减少到存储在 dest 中的 dest 值和值之和。 重载函数具有一个附加的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

要求

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedAnd 函数

项目 • 2022/09/22

以原子方式表示值。

语法

```
syntax
```

```
void InterlockedAnd(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

parameters

dest [in]

类型： **UINT**

目标地址。

值 [in]

类型： **UINT**

输入值。

original_value [out]

类型： **UINT**

原始值。

返回值

此函数不返回值。

注解

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行原子值和值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子值和值。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数会减少到 dest 和值的值，存储在 dest 中。 重载函数具有一个附加的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedCompareExchange 函数

项目 • 2022/09/23

将输入与比较值进行比较，并以原子方式交换结果。

语法

```
syntax
```

```
void InterlockedCompareExchange(
  in  UINT dest,
  in  UINT compare_value,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： **UINT**

目标地址。

compare_value [in]

类型： **UINT**

比较值。

值 [in]

类型： **UINT**

输入值。

original_value [out]

类型： **UINT**

原始值。

返回值

此函数不返回值。

注解

以原子方式将 *dest* 中的值与 *compare_value* 进行比较，如果值匹配，请将值存储在 *dest* 中，在 *original_value* 中返回 *dest* 的原始值。 此操作只能在 **int** 或 *uint* 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 *dest* 引用的共享内存寄存器执行操作。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 *dest* 引用的资源位置执行操作。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数会减少到使用本地操作执行的操作。 仅当 R 可读且可写时，此操作才可用。

> **备注**
>
> 如果在**计算**着色器循环中或**计算**着色器循环中调用 InterlockedCompareExchange，则必须在该循环上使用 [allow_uav_condition] 属性。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedCompareStore 函数

项目 • 2022/09/22

以原子方式将输入与比较值进行比较。

语法

```
syntax
```

```
void InterlockedCompareStore(
  in UINT dest,
  in UINT compare_value,
  in UINT value
);
```

parameters

dest [in]

类型： **UINT**

目标地址。

compare_value [in]

类型： **UINT**

比较值。

值 [in]

类型： **UINT**

输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行操作。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行操作。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数会减少到使用本地操作执行的操作。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedExchange 函数

项目 • 2022/09/23

以原子方式交换值。

语法

syntax

```
void InterlockedExchange(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： **UINT**

目标地址。

值 [in]

类型： **UINT**

输入值。

original_value [out]

类型： **UINT**

原始值。

返回值

此函数不返回值。

注解

此操作只能对标量类型的资源和共享内存变量执行。 此函数有三种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行操作。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行操作。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数会减少到使用本地操作执行的操作。 仅当 R 可读且可写时，此操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedMax 函数

项目 • 2022/09/22

以原子方式查找最大值。

语法

syntax

```
void InterlockedMax(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： **UINT**

目标地址。

value [in]

类型： **UINT**

输入值。

original_value [out]

类型： **UINT**

原始值。

返回值

此函数不返回值。

备注

此操作只能在 int 和 uint 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行值的最大原子最大值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行值的最大原子最大值。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数将减少到存储在 dest 中的最大 dest 值和值。 重载函数具有一个额外的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedMin 函数

项目 • 2022/09/22

以原子方式查找最小值。

语法

```
syntax
```

```
void InterlockedMin(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： **UINT**

目标地址。

value [in]

类型： **UINT**

输入值。

original_value [out]

类型： **UINT**

原始值。

返回值

无

注解

此操作只能在 int 和 uint 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行值的最低原子最小值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行值原子最小值。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数将减少到存储在 dest 中的 dest 和值的值的最小值。 重载函数具有一个额外的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedOr 函数

项目 • 2022/09/22

对值执行原子 OR 。

语法

syntax

```
void InterlockedOr(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： UINT

目标地址。

value [in]

类型： UINT

输入值。

original_value [out]

类型： UINT

原始值。

返回值

无

注解

此操作只能在 **INT** 或 **UINT** 类型资源和共享内存变量上执行。 此函数有三种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数执行原子 **OR** ，其中包含 *dest* 引用的共享内存寄存器的值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数使用 *dest* 引用的资源位置的值执行原子 **OR**。 最后，第三种情况是 R 是局部变量类型。 在此方案中，函数会减少为*具有* *dest* 和 *value* 值的 **OR**。 操作的结果将替换 *dest* 的值。 重载函数具有一个附加的输出变量，该变量将设置为 *dest* 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

InterlockedXor 函数

项目 • 2022/09/23

对值执行原子 XOR 。

语法

```
syntax
```

```
void InterlockedXor(
  in  UINT dest,
  in  UINT value,
  out UINT original_value
);
```

参数

dest [in]

类型： UINT

目标地址。

值 [in]

类型： UINT

输入值。

original_value [out]

类型： UINT

原始值。

返回值

此函数不返回值。

注解

此操作只能在 INT 或 UINT 类型化资源和共享内存变量上执行。 此函数有三种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数执行原子 XOR ，其中包含 *dest* 引用的共享内存寄存器的值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数使用 *dest* 引用的资源位置的值执行原子 XOR。 最后，第三种情况是 R 是局部变量类型。 在此方案中，该函数会减少到 *dest* 和 *value* 值的 XOR。 操作的结果将替换 *dest* 中的值。 重载函数具有一个附加的输出变量，该变量将设置为 *dest* 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

以下类型的着色器支持此函数：

VS	HS	DS	GS	PS	CS
x	x	x	x	x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

RWByteAddressBuffer：：Load 方法

项目 • 2022/09/22

从 RWByteAddressBuffer 获取一个值。

重载列表

方法	说明
加载 int、uint) (	获取一个值并返回操作的状态。
加载 int) (	获取一个值。

另请参阅

RWByteAddressBuffer

RWByteAddressBuffer：：Load (int) 函数

项目 • 2022/12/14

获取一个值。

语法

syntax

```
uint Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

返回值

类型： **uint**

一个值。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWByteAddressBuffer：：Load (int，uint) 函数

项目 • 2022/09/22

获取一个值并返回操作的状态。

语法

syntax

```
uint Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint**

一个值。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWByteAddressBuffer：：Load2 方法

项目 • 2022/09/22

从 RWByteAddressBuffer 获取两个值。

重载列表

方法	说明
Load2 (uint)	获取两个值。
Load2 (uint，uint)	获取两个值并返回操作的状态。

另请参阅

RWByteAddressBuffer

RWByteAddressBuffer：：Load2 (uint) 函数

项目 • 2022/09/22

获取两个值。

语法

syntax

```
uint2 Load2(
  in uint address
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint2**

两个值。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Load2 方法

着色器模型 5

Load2 (uint，uint) 函数

项目 • 2022/09/24

获取两个值并返回操作的状态。

语法

```
syntax
```

```
uint2 Load2(
  in  uint Location,
  out uint Status
);
```

参数

位置 [in]

类型： **uint**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint2**

两个值。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Load2 方法

RWByteAddressBuffer：：Load3 方法

项目 • 2022/09/22

从 RWByteAddressBuffer 获取三个值。

重载列表

方法	说明
Load3 (uint)	获取三个值。
Load3 (uint，uint)	获取三个值并返回操作的状态。

另请参阅

RWByteAddressBuffer

RWByteAddressBuffer：：Load3 (uint) 函数

项目 • 2022/09/22

获取三个值。

语法

syntax

```
uint3 Load3(
  in uint address
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint3**

三个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Load3 方法

着色器模型 5

Load3 (uint，uint) 函数

项目 • 2022/09/23

获取三个值并返回操作的状态。

语法

```
syntax
```

```
uint3 Load3(
  in  uint Location,
  out uint Status
);
```

参数

位置 [in]

类型： **uint**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint3**

三个值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Load3 方法

RWByteAddressBuffer：：Load4 方法

项目 • 2022/09/22

从 RWByteAddressBuffer 获取四个值。

重载列表

方法	说明
Load4 (uint)	获取四个值。
Load4 (uint，uint)	获取四个值并返回操作的状态。

另请参阅

RWByteAddressBuffer

RWByteAddressBuffer：：Load4 (uint) 函数

项目 • 2022/09/22

获取四个值。

语法

syntax

```
uint4 Load4(
  in uint address
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

返回值

类型： **uint4**

四个值。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Load4 方法

着色器模型 5

Load4 (uint，uint) 函数

项目 • 2022/09/22

获取四个值并返回操作的状态。

语法

syntax

```
uint4 Load4(
  in  uint Location,
  out uint Status
);
```

参数

位置 [in]

类型： **uint**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **uint4**

四个值。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Load4 方法

Microsoft Store函数

项目 • 2022/09/22

设置一个值。

语法

```
syntax
```

```
void Store(
  in uint address,
  in uint value
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

value [in]

类型： **uint**

一个输入值。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

Store2 函数

项目 • 2022/09/23

设置两个值。

语法

syntax

```
void Store2(
  in uint address,
  in uint2 values
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

值 [in]

类型： **uint2**

两个输入值。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

Store3 函数

项目 • 2022/09/22

设置三个值。

语法

syntax

```
void Store3(
  in uint address,
  in uint3 values
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

值 [in]

类型： **uint3**

三个输入值。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

Store4 函数

项目 • 2022/09/23

设置四个值。

语法

syntax

```
void Store4(
  in uint address,
  in uint4 values
);
```

参数

地址 [in]

类型： **uint**

输入地址（以字节为单位），该地址必须是 4 的倍数。

值 [in]

类型： **uint4**

四个输入值。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWByteAddressBuffer

着色器模型 5

RWStructuredBuffer

项目 • 2022/09/23

可采用结构 T 类型的读/写缓冲区。

方法	说明
DecrementCounter	递减对象的隐藏计数器。
GetDimensions	获取资源维度。
IncrementCounter	递增对象的隐藏计数器。
加载	读取缓冲区数据。
运算符[]	返回资源变量。

资源变量也可以传递到任何无序操作或互锁操作中。

RWStructuredBuffer 对象可以前缀为存储类 **globallycoherent**。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步只会刷新当前组中的 UAV。

需要使用DXGI_FORMAT_UNKNOWN格式创建绑定到此资源的 UAV 格式。

若要详细了解 结构化缓冲区，请参阅概述材料。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色 器模型着色器模型 4 (可通过 Direct3D 11 API 使用，方法是在支持计算着色器的设备上使用 10.0 或 10.1 功能级别 (D3D_FEATURE_LEVEL_10_X) 。 有关下层硬件上的计算着色器支持的详细信息，请参阅 Downlevel Hardware 上的计算着色器。)	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

DecrementCounter 函数

项目 • 2022/09/23

递减对象的隐藏计数器。

语法

```
syntax
```

```
uint DecrementCounter(void);
```

parameters

此函数没有参数。

返回值

类型： **uint**

递减后的计数器值。

备注

在创建此方法期间，绑定的无序访问视图必须设置 D3D11_BUFFER_UAV_FLAG_COUNTER 。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

RWStructuredBuffer

着色器模型 5

RWStructuredBuffer：：GetDimensions 函数

项目 • 2022/09/24

获取资源维度。

语法

syntax

```
void GetDimensions(
  out uint numStructs,
  out uint stride
);
```

参数

numStructs [out]

类型： **uint**

资源中的结构数。

stride [out]

类型： **uint**

每个结构元素的步幅（以字节为单位）。

返回值

无

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWStructuredBuffer

着色器模型 5

IncrementCounter 函数

项目 • 2022/09/22

递增对象的隐藏计数器。

语法

```
syntax

uint IncrementCounter(void);
```

parameters

此函数没有参数。

返回值

类型： **uint**

预先递增的计数器值。

备注

绑定的无序访问视图必须在创建过程中设置 D3D11_BUFFER_UAV_FLAG_COUNTER 才能使此方法正常工作。

可以根据结构的组件名称进一步为结构化资源编制索引。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

RWStructuredBuffer

着色器模型 5

RWStructuredBuffer：：Load 方法

项目 • 2022/09/24

将缓冲区数据读入 RWStructuredBuffer。

重载列表

方法	说明
加载 int、uint) (	读取缓冲区数据并返回有关操作的状态。
加载 int) (	读取缓冲区数据。

另请参阅

RWStructuredBuffer

RWStructuredBuffer：：Load (int) 函数

项目 • 2022/09/22

读取缓冲区数据。

语法

syntax

```
 Load(
  in int Location
);
```

parameters

位置 [in]

类型： **int**

缓冲区的位置。

返回值

类型：

返回类型与 RWStructuredBuffer 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

加载方法

RWStructuredBuffer：：Load (int，uint) 函数

项目 • 2022/09/22

读取缓冲区数据并返回有关操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWStructuredBuffer 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

加载方法

RWStructuredBuffer：：Operator 函数

项目 • 2022/09/24

返回资源变量。

语法

syntax

```
R Operator[](
  in uint pos
);
```

参数

pos [in]

类型： **uint**

索引位置。

返回值

类型： **R**

资源变量。

注解

可以根据结构的组件名称进一步为结构化资源编制索引。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWStructuredBuffer

着色器模型 5

RWTexture1D

项目 • 2022/09/24

读/写资源。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
运算符[]	获取资源变量。

可以将 RWTexture1D 对象作为 **全局共同**存储类的前缀。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅在当前组中刷新 UAV。

RWTexture1D 对象需要对象的声明语句中的元素类型。 例如，以下声明正确：

```
RWTexture1D<float> tex;
```

由于 RWTexture1D 对象是 UAV 类型对象，因此其属性不同于着色器资源视图 (SRV) 类型对象，例如 Texture1D 对象。 例如，可以读取和写入 RWTexture1D 对象，但只能从 Texture1D 对象读取。

RWTexture1D 对象不能使用 Texture1D 对象的方法，例如 Sample。 但是，由于可以将多个视图类型创建到同一资源，因此可以将多个纹理类型声明为多个着色器中的单个纹理。 例如，可以在计算着色器中声明和使用 RWTexture1D 对象作为纹素，然后在像素着色器中声明并使用 Texture1D 对象作为纹。

> **备注**
>
> 当为同一资源创建多个视图类型时，运行时会强制实施某些使用模式。 例如，运行时不允许同时对资源具有 UAV 映射和同一资源活动的 SRV 映射。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWTexture1D： ：GetDimensions 函数

项目 • 2022/09/24

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width
);
```

参数

宽度 [out]

类型： UINT

资源宽度（以纹素为单位）。

返回值

无

注解

这是此方法重载版本的列表。

```
void GetDimensions (out UINT Width);

void GetDimensions(out float Width);
```

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

RWTexture1D

着色器模型 5

RWTexture1D：：Load 方法

项目 • 2022/09/23

从 RWTexture1D 读取纹理数据。

重载列表

方法	说明
加载 int、uint) (	读取纹理数据并返回有关操作的状态。
加载 int) (	读取纹理数据。

另请参阅

RWTexture1D

RWTexture1D： ：Load (int) 函数

项目 • 2022/09/22

读取纹理数据。

语法

```
syntax
```

```
 Load(
  in int Location
);
```

parameters

位置 [in]

类型： **int**

纹理的位置。

返回值

类型：

返回类型与 RWTexture1D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture1D：：Load (int，uint) 函数

项目 • 2022/09/24

读取纹理数据并返回有关操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWTexture1D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

加载方法

RWTexture1D： ：Operator 函数

项目 • 2022/09/22

返回资源变量。

语法

```
syntax
```

```
R Operator[](
  in uint pos
);
```

parameters

pos [in]

类型： **uint**

索引位置。 包含 x 坐标。

返回值

类型： **R**

资源变量。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture1D

RWTexture1DArray

项目 • 2022/09/22

读/写资源。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
运算符[]	获取资源变量。

可以将 RWTexture1DArray 对象前缀为存储类 **全局一致性**。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅在当前组中刷新 UAV。

RWTexture1DArray 对象需要对象声明语句中的元素类型。 例如，以下声明是正确的：

```
RWTexture1DArray<float> tex;
```

由于 RWTexture1DArray 对象是 UAV 类型对象，因此其属性不同于着色器资源视图 (SRV) 类型对象（如 Texture1DArray 对象）。 例如，可以读取和写入 RWTexture1DArray 对象，但只能从 Texture1DArray 对象读取。

RWTexture1DArray 对象不能使用 Texture1DArray 对象中的方法，例如 Sample。 但是，由于可以为同一资源创建多个视图类型，因此可以在多个着色器中将多个纹理类型声明为单个纹理。 例如，可以在计算着色器中声明并使用 RWTexture1DArray 对象作为 *tex*，然后在像素着色器中声明并使用 Texture1DArray 对象作为纹。

> **备注**
>
> 当为同一资源创建多个视图类型时，运行时会强制实施某些使用模式。 例如，运行时不允许同时为资源创建 UAV 映射和同时处于活动状态的同一资源的 SRV 映射。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWTexture1DArray：：GetDimensions 函数

项目 • 2022/09/24

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width,
  out UINT Elements
);
```

parameters

宽度 [out]

类型： UINT

资源宽度（以纹素为单位）。

元素 [out]

类型： UINT

数组中的元素数。

返回值

此函数不返回值。

注解

这是此方法重载版本的列表。

```

void GetDimensions (out UINT Width,
  out UINT Elements);
```

```
void GetDimensions(out float Width,
  out UINT Elements);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture1DArray

着色器模型 5

RWTexture1DArray： ：Load 方法

项目 • 2022/09/22

从 RWTexture1DArray 读取纹理数据。

重载列表

方法	说明
加载 int、uint) (	读取纹理数据并返回有关操作的状态。
加载 int) (	读取纹理数据。

另请参阅

RWTexture1DArray

RWTexture1DArray：：Load (int) 函数

项目 • 2022/09/22

读取纹理数据。

语法

syntax

```
 Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

返回值

类型：

返回类型与 RWTexture1DArray 对象的声明中的类型匹配。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture1DArray：：Load (int，uint) 函数

项目 • 2022/09/22

读取纹理数据并返回有关操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWTexture1DArray 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

加载方法

RWTexture1DArray： ：Operator 函数

项目 • 2022/09/22

返回资源变量。

语法

```
syntax
```

```
R Operator[](
  in uint2 pos
);
```

parameters

pos [in]

类型： **uint2**

索引位置。 第一个组件包含 x 坐标。 第二个组件指示所需的数组切片。

返回值

类型： **R**

资源变量。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture1DArray

RWTexture2D

项目 • 2022/09/22

读/写资源。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
运算符[]	获取资源变量。

可以将 RWTexture2D 对象前缀为全局 **共同**存储类。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅刷新当前组中的未排序访问视图 (UAV) 。

RWTexture2D 对象需要对象的声明语句中的元素类型。 例如，以下声明不正确：

```
// The following declaration is incorrectly coded.
RWTexture2D myTexture;
```

以下声明正确：

```
// The following declaration is correctly coded.
RWTexture2D<float> tex;
```

由于 RWTexture2D 对象是 UAV 类型对象，因此其属性不同于着色器资源视图 (SRV) 类型对象（例如 Texture2D 对象）。 例如，可以读取和写入 RWTexture2D 对象，但只能从 Texture2D 对象读取。

RWTexture2D 对象不能使用 Texture2D 对象的方法，例如 Sample。 但是，由于可以将多个视图类型创建到同一资源，因此可以将多个纹理类型声明为多个着色器中的单个纹理。 例如，以下代码片段演示如何在计算着色器中声明和使用 RWTexture2D 对象作为纹，然后在像素着色器中声明并使用 Texture2D 对象作为纹。

备注

当为同一资源创建多个视图类型时，运行时会强制实施某些使用模式。 例如，运行时不允许同时对资源具有 UAV 映射和同一资源活动的 SRV 映射。

以下代码适用于计算着色器：

```
RWTexture2D<float> tex;
[numthreads(groupDim_x, groupDim_y, 1)]
void main(
    uint3 groupId : SV_GroupID,
    uint3 groupThreadId : SV_GroupThreadID,
    uint3 dispatchThreadId : SV_DispatchThreadID,
    uint groupIndex : SV_GroupIndex)
{
    tex [dispatchThreadId.xy] = <something>;
}
```

以下代码适用于像素着色器：

```
struct PixelShaderInput
{
    float4 pos : SV_POSITION;
    float2 tex : TEXTURE;
};

Texture2D<float> tex;
float4 main(PixelShaderInput input) : SV_TARGET
{
    return tex.Sample(TextureSampler, input.tex);
}
```

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWTexture2D：：GetDimensions 函数

项目 • 2022/09/24

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width,
  out UINT Height
);
```

parameters

宽度 [out]

类型： UINT

资源宽度（以纹素为单位）。

高度 [out]

类型： UINT

资源高度（以纹素为单位）。

返回值

此函数不返回值。

注解

这是此方法重载版本的列表。

```

void GetDimensions (out UINT Width,
  out UINT Height);


```

```
void GetDimensions(out float Width,
  out float Height);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture2D

着色器模型 5

RWTexture2D：：Load 方法

项目 • 2022/09/22

从 RWTexture2D 读取纹理数据。

重载列表

方法	说明
加载 int、uint) (	读取纹理数据并返回有关操作的状态。
加载 int) (	读取纹理数据。

另请参阅

RWTexture2D

RWTexture2D：：Load (int) 函数

项目 • 2022/09/22

读取纹理数据。

语法

```
syntax
```

```
 Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

返回值

类型：

返回类型与 RWTexture2D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture2D：：Load (int，uint) 函数

项目 • 2022/09/23

读取纹理数据并返回有关操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  out uint Status
);
```

parameters

位置 [in]

类型： **int**

纹理的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWTexture2D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture2D：：Operator 函数

项目 • 2022/09/22

返回资源变量。

语法

syntax

```
R Operator[](
  in uint2 pos
);
```

参数

pos [in]

类型： **uint2**

索引位置。 包含 (x、y) 坐标。

返回值

类型： **R**

资源变量。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture2D

RWTexture2DArray

项目 • 2022/09/23

读/写资源。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
运算符[]	获取资源变量。

可以将 RWTexture2DArray 对象前缀为存储类 **全局共和**。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅在当前组中刷新 UAV。

RWTexture2DArray 对象需要对象的声明语句中的元素类型。 例如，以下声明正确：

```
RWTexture2DArray<float> tex;
```

由于 RWTexture2DArray 对象是 UAV 类型对象，因此其属性不同于着色器资源视图 (SRV) 类型对象，例如 Texture2DArray 对象。 例如，可以读取和写入 RWTexture2DArray 对象，但只能从 Texture2DArray 对象读取。

RWTexture2DArray 对象不能使用 Texture2DArray 对象（例如 Sample）中的方法。 但是，由于可以将多个视图类型创建到同一资源，因此可以将多个纹理类型声明为多个着色器中的单个纹理。 例如，可以在计算着色器中声明和使用 RWTexture2DArray 对象作为 *tex*，然后在像素着色器中声明并使用 Texture2DArray 对象作为 *纹*。

> **备注**
>
> 当为同一资源创建多个视图类型时，运行时会强制实施某些使用模式。 例如，运行时不允许同时对资源具有 UAV 映射和同一资源活动的 SRV 映射。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWTexture2DArray：：GetDimensions 函数

项目 • 2022/09/23

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width,
  out UINT Height,
  out UINT Elements
);
```

parameters

宽度 [out]

类型： UINT

资源宽度（以纹素为单位）。

高度 [out]

类型： UINT

资源高度（以纹素为单位）。

元素 [out]

类型： UINT

数组中的元素数。

返回值

此函数不返回值。

注解

这是此方法的重载版本的列表。

```
void GetDimensions (out UINT Width,
  out UINT Height,
  out UINT Elements);

void GetDimensions(out float Width,
  out float Height,
  out float Elements);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture2DArray

着色器模型 5

RWTexture2DArray：：Load 方法

项目 • 2022/09/22

从 RWTexture2DArray 读取纹理数据。

重载列表

方法	说明
加载 int、uint) (	读取纹理数据并返回有关操作的状态。
加载 int) (	读取纹理数据。

另请参阅

RWTexture2DArray

RWTexture2DArray：：Load (int) 函数

项目 • 2022/09/22

读取纹理数据。

语法

```
syntax
```

```
 Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

返回值

类型：

返回类型与 RWTexture2DArray 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture2DArray：：Load (int，uint) 函数

项目 • 2022/09/22

读取纹理数据并返回有关操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWTexture2DArray 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture2DArray：：Operator 函数

项目 • 2022/09/22

返回资源变量。

语法

```
syntax
```

```
R Operator[](
  in uint3 pos
);
```

参数

pos [in]

类型： **uint3**

索引位置。 第一个和第二个组件包含 (x、y) 坐标。 第三个组件指示所需的数组切片。

返回值

类型： **R**

资源变量。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture2DArray

RWTexture3D

项目 • 2022/09/24

读/写资源。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
运算符[]	获取资源变量。

可以使用存储类 **globallycoherent** 为 **RWTexture3D** 对象添加前缀。 此存储类会导致内存屏障和同步跨整个 GPU 刷新数据，以便其他组可以看到写入。 如果没有此说明符，内存屏障或同步将仅在当前组中刷新 UAV。

RWTexture3D 对象需要对象声明语句中的元素类型。 例如，以下声明是正确的：

```
RWTexture3D<float> tex;
```

由于 **RWTexture3D** 对象是 UAV 类型对象，因此其属性不同于着色器资源视图 (SRV) 类型对象（如 Texture3D 对象）。 例如，可以读取和写入 **RWTexture3D** 对象，但只能从 **Texture3D** 对象读取。

RWTexture3D 对象不能使用 Texture3D 对象中的方法，例如 Sample。 但是，由于可以为同一资源创建多个视图类型，因此可以在多个着色器中将多个纹理类型声明为单个纹理。 例如，可以在计算着色器中声明并使用 **RWTexture3D** 对象作为 *tex*，然后在像素着色器中声明并使用 **Texture3D** 对象作为 *纹*。

> **备注**
>
> 当为同一资源创建多个视图类型时，运行时会强制实施某些使用模式。 例如，运行时不允许同时为资源创建 UAV 映射和同时处于活动状态的同一资源的 SRV 映射。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

着色器模型 5 对象

RWTexture3D：：GetDimensions 函数

项目 • 2022/09/22

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width,
  out UINT Height,
  out UINT Depth
);
```

参数

宽度 [out]

类型： UINT

资源宽度（以纹素为单位）。

高度 [out]

类型： UINT

资源高度（以纹素为单位）。

深度 [out]

类型： UINT

以纹素表示的资源深度。

返回值

此函数不返回值。

注解

这是此方法的重载版本的列表。

```
void GetDimensions (out UINT Width,
  out UINT Height,
  out UINT Depth);

void GetDimensions(out float Width,
  out float Height,
  out float Depth);
```

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

RWTexture3D

着色器模型 5

RWTexture3D：：Load 方法

项目 • 2022/09/22

从 RWTexture3D 读取纹理数据。

重载列表

方法	说明
加载 int、uint) (	读取纹理数据并返回有关操作的状态。
加载 int) (	读取纹理数据。

另请参阅

RWTexture3D

RWTexture3D：：Load (int) 函数

项目 • 2022/09/22

读取纹理数据。

语法

syntax

```
 Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

返回值

类型：

返回类型与 RWTexture3D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

RWTexture3D：：Load (int，uint) 函数

项目 • 2022/09/24

读取纹理数据并返回有关操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 RWTexture3D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

加载方法

RWTexture3D：：Operator 函数

项目 • 2022/09/24

返回 RWTexture3D 的资源变量。

语法

```
syntax
```

```
R Operator[](
  in uint3 pos
);
```

参数

pos [in]

类型： **uint3**

索引位置。 包含 (x、y、z) 坐标。

返回值

类型： **R**

资源变量。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

RWTexture3D

StructuredBuffer

项目 • 2022/09/23

只读缓冲区，该缓冲区可以采用属于结构的 T 类型。

方法	说明
GetDimensions	获取资源维度。
加载	读取缓冲区数据。
运算符[]	返回只读资源变量。

需要使用 DXGI_FORMAT_UNKNOWN 格式创建绑定到此资源的 SRV 格式。

若要了解有关 结构化缓冲区的详细信息，请参阅概述材料。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色 器模型着色器模型 4 (可用于某些 Direct3D 10 设备上的 Direct3D 11 中的计算和像素着色器。)	是

以下类型的着色器支持此对象。

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

StructuredBuffer：：GetDimensions 函数

项目 • 2022/09/24

获取资源维度。

语法

syntax

```
void GetDimensions(
  out uint numStructs,
  out uint stride
);
```

参数

numStructs [out]

类型： **uint**

资源中的结构数。

stride [out]

类型： **uint**

每个结构元素的步幅（以字节为单位）。

返回值

此函数不返回值。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

StructuredBuffer

着色器模型 5

StructuredBuffer： ：Load 方法

项目 • 2022/09/23

从 StructuredBuffer 读取缓冲区数据。

重载列表

方法	说明
加载 int、uint) (	读取缓冲区数据并返回有关操作的状态。
加载 int) (	读取缓冲区数据。

另请参阅

StructuredBuffer

StructuredBuffer： ：Load (int) 函数

项目 • 2022/09/24

读取缓冲区数据。

语法

```
syntax
```

```
 Load(
  in int Location
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

返回值

类型：

返回类型与 StructuredBuffer 对象的声明中的类型匹配。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

StructuredBuffer：：Load (int，uint) 函数

项目 • 2022/09/22

读取缓冲区数据并返回有关操作的状态。

语法

```
syntax
```

```
 Load(
  in  int  Location,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

缓冲区的位置。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 StructuredBuffer 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

加载方法

StructuredBuffer：：Operator 函数

项目 • 2022/09/22

返回 StructuredBuffer 的只读资源变量。

语法

syntax

```
R Operator[](
  in uint pos
);
```

参数

pos [in]

类型： **uint**

索引位置。

返回值

类型： **R**

只读资源变量。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

StructuredBuffer

Texture1D

项目 • 2022/09/22

着色 器模型 4) 加上资源变量中存在 1D 纹理类型 (。 除了着色器模型 4 中的方法外，此纹理对象还支持以下方法。

方法	描述
GetDimensions	获取资源维度。
加载	读取纹理数据。
运算符[]	获取只读资源变量。
Mips。Operator[][]	获取只读资源变量。
示例	采样纹理。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleCmp	使用比较值来采样纹理，以拒绝样本。
SampleCmpLevelZero	仅) 采样纹理 (mipmap 级别 0，使用比较值来拒绝样本。
SampleGrad	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleLevel	对指定 mipmap 级别上的纹理采样。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture1D： ：GetDimensions 函数

项目 • 2022/09/23

返回资源的维度。

语法

syntax

```
void GetDimensions(
  in  UINT MipLevel,
  out UINT Width,
  out UINT NumberOfLevels
);
```

参数

MipLevel [in]

类型： **UINT**

可选。 如果使用 *NumberOfLevels*），则必须指定 Mipmap 级别 (。

宽度 [out]

类型： **UINT**

资源宽度（以纹素为单位）。

NumberOfLevels [out]

类型： **UINT**

mipmap 级别数 (还需要 *MipLevel*) 。

返回值

此函数不返回值。

注解

这是此方法重载版本的列表。

```
void GetDimensions(UINT MipLevel,
  out UINT Width,
  out UINT NumberOfLevels);

void GetDimensions (out UINT Width);

void GetDimensions(UINT MipLevel,
  out float Width,
  out float NumberOfLevels);

void GetDimensions(out float Width);
```

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

Texture1D

着色器模型 5

Texture1D：：Load 方法

项目 • 2022/09/23

从 Texture1D **读取纹理**数据。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture1D

Texture1D：：Load (int，int，uint) 函数

项目 • 2022/09/24

读取纹理数据并返回操作的状态。

语法

```
syntax
```

```
 Load(
  in  int  Location,
  in  int  Offset,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理坐标。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture1D 对象的声明中的类型匹配。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

加载方法

Texture1D

Texture1D：：mips。运算符函数

项目 • 2022/09/22

返回只读资源变量或 Texture1D。

语法

```
syntax
```

```
R mips.Operator[][](
  in uint mipSlice,
  in uint pos
);
```

parameters

mipSlice [in]

类型： **uint**

mip 切片索引。

pos [in]

类型： **uint**

索引位置。 包含 x 坐标。

返回值

类型： **R**

只读资源变量。

注解

用法示例

```
```

```
Texture1D<float4> tex;
uint mip = 2;
uint pos = 1234;
float4 f = tex.mips[mip][pos];
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture1D

着色器模型 5

Texture1D： ：Operator 函数

项目 • 2022/09/23

返回 Texture1D 的只读资源变量。

语法

syntax

```
R Operator[](
  in uint pos
);
```

参数

pos [in]

类型： **uint**

索引位置。 包含 x 坐标。

返回值

类型： **R**

只读资源变量。

注解

此方法始终访问第一个 mip 级别。 若要指定其他 mip 级别，请改用 mip.operator[][] 。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture1D

着色器模型 5

Texture1D：：Sample 方法

项目 • 2022/09/22

采样 **纹理 1D**。

重载列表

方法	说明
示例 (S、float、int)	对纹理进行采样。
示例 (S、float、int、float)	将具有可选值的纹理采样，以将样本详细级别固定 (LOD) 值。
示例 (S，float，int，float，uint)	使用可选值对纹理采样采样， (LOD) 值固定，并返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture1D

Texture1D：：Sample (S，float，int，float) 函数

项目 • 2022/09/24

将具有可选值的纹理采样，以将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float        Location,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

示例方法

Texture1D：：示例 (S、float、int、float、uint) 函数

项目 • 2022/09/23

使用可选值采样纹理，以固定示例详细级别 (LOD) 值，并返回操作的状态。

语法

syntax

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

示例方法

Texture1D：：SampleBias 方法

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对 Texture1D 采样。

重载列表

方法	说明
SampleBias (S，float，float，int)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S，float，float，int，float)	将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。
SampleBias (S，float，float，int，float，uint)	将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture1D

SampleBias：：SampleBias (S、float、float、int、float) 函数 for Texture1D

项目 • 2022/09/23

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in int          Offset,
  in float        Clamp
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

SampleBias： ：SampleBias (S、float、float、int、float、uint) function for Texture1D

项目 • 2022/09/24

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

Texture1D：：SampleCmp 方法

项目 • 2022/09/22

使用比较值对 Texture1D 采样，以拒绝样本。

重载列表

方法	说明
SampleCmp (S，float，float，int)	使用比较值对纹理进行采样，以拒绝样本。
SampleCmp (S，float，float，int，float)	使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。
SampleCmp (S，float，float，int，float，uint)	使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture1D

SampleCmp： ：Texture1D 的 SampleCmp (S、float、float、int、float) 函数

项目 • 2022/09/22

使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

SampleCmp： ：SampleCmp (S、float、float、int、float、uint) 函数 for Texture1D

项目 • 2022/09/24

使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样， (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

Texture1D：：SampleCmpLevelZero 方法

项目 • 2022/09/22

仅对 mipmap 级别 0 上的 Texture1D 采样，并将结果与比较值进行比较。

重载列表

方法	说明
SampleCmpLevelZero (S，float，float，int)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。
SampleCmpLevelZero (S，float，float，int，uint)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture1D

SampleCmpLevelZero： ： SampleCmpLevelZero (S，float，float，int，uint) function for Texture1D

项目 • 2022/09/24

仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmpLevelZero(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleCmpLevelZero 方法

Texture1D：：SampleGrad 方法

项目 • 2022/09/22

使用渐变对 Texture1D 采样，以影响示例位置的计算方式。

重载列表

方法	说明
SampleGrad (S、float、float、float、int)	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleGrad (S、float、float、float、int、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。
SampleGrad (S、float、float、float、int、float、uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture1D

SampleGrad：：SampleGrad (S、float、float、float、int、float) 函数 for Texture1D

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

以 x 方向变化表面几何图形的速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

SampleGrad：：Texture1D 的 SampleGrad (S、float、float、float、int、float、uint) 函数

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2

Texture-Object类型	参数类型
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture1D：：SampleLevel 方法

项目 • 2022/09/22

对指定 mipmap 级别上的 Texture1D 采样。

重载列表

方法	说明
SampleLevel (S，float，float，int)	对指定 mipmap 级别的纹理采样。
SampleLevel (S，float，float，int，uint)	对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture1D

SampleLevel：：Texture1D 的 SampleLevel (S、float、float、int、uint) 函数

项目 • 2022/09/23

对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  in  int          Offset,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果该值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleLevel 方法

Texture1DArray

项目 • 2022/09/24

Texture1DArray 类型 (，因为它存在于着色器模型 4) 加上资源变量中。 除了着色器模型 4 中的方法外，此纹理对象还支持以下方法。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
Mips。Operator[][]	获取只读资源变量。
运算符[]	获取只读资源变量。
示例	采样纹理。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleCmp	使用比较值来采样纹理，以拒绝样本。
SampleCmpLevelZero	仅) 采样纹理 (mipmap 级别 0，使用比较值来拒绝样本。
SampleGrad	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleLevel	对指定 mipmap 级别上的纹理采样。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture1DArray：：GetDimensions 函数

项目 • 2022/09/24

返回资源的维度。

语法

syntax

```
void GetDimensions(
  in  UINT MipLevel,
  out UINT Width,
  out UINT Elements,
  out UINT NumberOfLevels
);
```

parameters

MipLevel [in]

类型： **UINT**

可选。 如果使用 *NumberOfLevels*），则必须指定 Mipmap 级别 (。

宽度 [out]

类型： **UINT**

资源宽度（以纹素为单位）。

元素 [out]

类型： **UINT**

数组中的元素数。

NumberOfLevels [out]

类型： **UINT**

mipmap 级别数 (还需要 *MipLevel*) 。

返回值

此函数不返回值。

注解

这是此方法重载版本的列表。

```
void GetDimensions(UINT MipLevel,
  out UINT Width,
  out UINT Elements,
  out UINT NumberOfLevels);

void GetDimensions (out UINT Width,
  out UINT Elements);

void GetDimensions(UINT MipLevel,
  out float Width,
  out UINT Elements,
  out float NumberOfLevels);

void GetDimensions(out float Width,
  out UINT Elements);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Texture1DArray

着色器模型 5

Texture1DArray：：Load 方法

项目 • 2022/09/24

读取纹理数据。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture1DArray

Texture1DArray：：Load (int，int，uint) 函数

项目 • 2022/09/24

读取纹理数据并返回操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  in  int  Offset,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理坐标。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture1DArray 对象的声明中的类型匹配。

另请参阅

加载方法

Texture1DArray：：mips。运算符函数

项目 • 2022/09/22

返回只读资源变量。

语法

```
syntax
```

```
R mips.Operator[][](
  in uint mipSlice,
  in uint2 pos
);
```

参数

mipSlice [in]

类型： **uint**

mip 切片索引。

pos [in]

类型： **uint2**

索引位置。 第一个组件包含 x 坐标。 第二个组件指示所需的数组切片。

返回值

类型： **R**

只读资源变量。

备注

用法示例

```
```

```
Texture1DArray<float4> tex;
uint mip = 2;
uint2 pos_x_and_array = {1234, 3};
float4 f = tex.mips[mip][pos_x_and_array];
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture1DArray

着色器模型 5

Texture1DArray：：Operator 函数

项目 • 2022/09/22

返回只读资源变量。

语法

syntax

```
R Operator[](
  in uint2 pos
);
```

参数

pos [in]

类型： **uint2**

索引位置。 第一个组件包含 x 坐标。 第二个组件指示所需的数组切片。

返回值

类型： **R**

只读资源变量。

注解

此方法始终访问第一个 mip 级别。 若要指定其他 mip 级别，请改用 mip.operator[][] 方法。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture1DArray

着色器模型 5

Texture1DArray：：Sample 方法

项目 • 2022/09/22

采样纹理。

重载列表

方法	说明
示例 (S、float、int)	采样纹理。
示例 (S、float、int、float)	将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。
示例 (S、float、int、float、uint)	对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回有关操作的状态。

另请参阅

Texture1DArray

Texture-Object

Texture1DArray：：Sample (S，float，int，float) 函数

项目 • 2022/09/24

将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float        Location,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

示例方法

Texture1DArray：：Sample (S，float，int，float，uint) 函数

项目 • 2022/09/22

使用可选值采样纹理，以固定示例详细级别 (LOD) 值，并返回操作的状态。

语法

syntax

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

示例方法

Texture1DArray：：SampleBias 方法

项目 • 2022/09/24

将偏差值应用于 mipmap 级别后，对纹理采样。

重载列表

方法	说明
SampleBias (S，float，float，int)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S、float、float、int、float)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。
SampleBias (S、float、float、int、float、uint)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture1DArray

Texture-Object

SampleBias：：Texture1DArray 的 SampleBias (S、float、float、int、float) 函数

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（0.0 到 1.0（含 0.0）之间的浮点数在采样之前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleBias 方法

SampleBias：：SampleBias (S、float、int、float、uint) function for Texture1DArray

项目 • 2022/09/24

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

Texture1DArray：：SampleCmp 方法

项目 • 2022/09/22

使用比较值对纹理进行采样，以拒绝样本。

重载列表

方法	说明
SampleCmp (S，float，float，int)	使用比较值对纹理进行采样，以拒绝样本。
SampleCmp (S，float，float，int，float)	使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。
SampleCmp (S，float，float，int，float，uint)	使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture1DArray

Texture-Object

SampleCmp： ：Texture1DArray 的 SampleCmp (S、float、float、int、float) 函数

项目 • 2022/09/22

使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

SampleCmp： ：Texture1DArray 的 SampleCmp (S、float、float、int、float、uint) 函数

项目 • 2022/09/22

使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴中提取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

Texture1DArray：：SampleCmpLevelZero 方法

项目 • 2022/09/23

仅对 mipmap 级别 0 上的纹理采样，使用比较值来拒绝样本。

重载列表

方法	说明
SampleCmpLevelZero (S、float、float、int)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。
SampleCmpLevelZero (S、float、float、int、uint)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

另请参阅

Texture1DArray

Texture-Object

SampleCmpLevelZero： ：SampleCmpLevelZero (S，float，float，int，uint) function for Texture1DArray

项目 • 2022/09/22

仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmpLevelZero(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleCmpLevelZero 方法

Texture1DArray：：SampleGrad 方法

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响样本位置的计算方式。

重载列表

方法	说明
SampleGrad (S，float，float，float，int)	使用渐变对纹理进行采样，以影响样本位置的计算方式。
SampleGrad (S，float，float，float，int，float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。
SampleGrad (S，float，float，float，int，float，uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture1DArray

Texture-Object

SampleGrad：：Texture1DArray 的 SampleGrad (S、float、float、float、int、float) 函数

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

SampleGrad：：SampleGrad (S，float，float，float，int，float，uint) function for Texture1DArray

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2

Texture-Object类型	参数类型
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

DDX [in]

类型： **float**

以 x 方向变化表面几何图形的速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture1DArray：：SampleLevel 方法

项目 • 2022/09/23

对指定 mipmap 级别上的纹理采样。

重载列表

方法	说明
SampleLevel (S，float，float，int)	对指定 mipmap 级别上的纹理采样。
SampleLevel (S、float、float、int、uint)	对指定 mipmap 级别上的纹理采样，并返回有关操作的状态。

另请参阅

Texture1DArray

Texture-Object

SampleLevel： ：SampleLevel (S、float、float、int、uint) 函数 (适用于 Texture1DArray)

项目 • 2022/09/24

对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果该值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleLevel 方法

Texture2D

项目 • 2022/09/22

纹理 2D 类型 (， 因为它存在于着色器模型 4) 加上资源变量中。 除了着色器模型 4 中的方法外，此纹理对象还支持以下方法。

方法	说明
Gather	返回将在双线性筛选操作中使用的四个纹素值。
GatherRed	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
GatherGreen	返回将在双线性筛选操作中使用的四个纹素值的绿色分量。
GatherBlue	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherAlpha	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherCmp	对于要在双线性筛选操作中使用的四个纹素值，将返回它们与比较值进行比较。
GatherCmpRed	对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值进行比较。
GatherCmpGreen	对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值进行比较。
GatherCmpBlue	对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值进行比较。
GatherCmpAlpha	对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值的比较。
GetDimensions	获取资源维度。
加载	读取纹理数据。
Mips。Operator[][]	获取只读资源变量。
运算符[]	获取只读资源变量。
示例	对纹理进行采样。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleCmp	使用比较值对纹理进行采样，以拒绝样本。
SampleCmpLevelZero	仅使用比较值来拒绝样本，将纹理 (mipmap 级别 0 采样) 。
SampleGrad	使用渐变对纹理进行采样，以影响样本位置的计算方式。

方法	说明
SampleLevel	对指定 mipmap 级别的纹理采样。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture2D：：Gather 方法

项目 • 2022/09/24

采样 **Texture2D** 并返回所有四个组件。

有关 gather4 的文档，了解有关基础 DXBC 指令的详细信息。

重载列表

方法	说明
收集 (S、float、int)	采样纹理并返回所有四个组件。
收集 (S、float、int、uint)	采样纹理并返回所有四个组件以及操作状态。

另请参阅

Texture2D

Texture2D：：收集 (S、float、int) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值。

语法

syntax

```
TemplateType Gather(
  in sampler s,
  in float2 location,
  in int2 offset
);
```

parameters

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样之前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

收集方法

着色器模型 5

Texture2D：：Gather (S，float，int，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值，以及图块映射状态。

语法

syntax

```
TemplateType Gather(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

收集方法

Texture2D：：GatherAlpha 方法

项目 • 2022/09/23

采样 Texture2D 并返回 alpha 组件。

重载列表

方法	说明
GatherAlpha (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherAlpha (S、float、int、uint)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 组件，以及图块映射状态。
GatherAlpha (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherAlpha (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 组件，以及图块映射状态。

另请参阅

Texture2D

Texture2D：：GatherAlpha (S，float，int) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。

语法

```
syntax
```

```
TemplateType GatherAlpha(
  in sampler s,
  in float2 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

着色器模型 5

Texture2D：：GatherAlpha (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。

语法

syntax

```
TemplateType GatherAlpha(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

Texture2D：：GatherAlpha (S、float、int、uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherAlpha(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

Texture2D：：GatherAlpha (S，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherAlpha(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

Texture2D：：GatherBlue 方法

项目 • 2022/09/23

采样 Texture2D 并返回蓝色分量。

重载列表

方法	说明
GatherBlue (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherBlue (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量，以及图块映射状态。
GatherBlue (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherBlue (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量，以及图块映射状态。

另请参阅

Texture2D

Texture2D：：GatherBlue (S，float，int) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。

语法

syntax

```
TemplateType GatherBlue(
  in sampler s,
  in float2 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

着色器模型 5

GatherBlue (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。

语法

syntax

```
TemplateType GatherBlue(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

Texture2D：：GatherBlue (S，float，int，uint) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

语法

syntax

```
TemplateType GatherBlue(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

Texture2D：：GatherBlue (S，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherBlue(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

Texture2D：：GatherCmp 方法

项目 • 2022/09/24

对于将在双线性筛选操作中使用的 **Texture2D** 的四个纹素值，将返回它们与比较值进行比较。

有关 gather4_c 的文档，请参阅描述基础 DXBC 指令的详细信息。

重载列表

方法	说明
GatherCmp (S，float，float，int)	采样并比较纹理并返回所有四个组件。
GatherCmp (S，float，float，int，uint)	采样并比较纹理，并返回所有四个组件以及操作状态。

另请参阅

Texture2D

Texture2D：：GatherCmp (S、float、float、int) 函数

项目 • 2022/09/24

对于要在双线性筛选运算中使用的四个纹素值，返回它们与比较值进行比较。

语法

syntax

```
float4 GatherCmp(
  in SamplerComparisonState s,
  in float2 location,
  in float compare_value,
  in int2 offset
);
```

parameters

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmp 方法

着色器模型 5

Texture2D：：GatherCmp (S、float、float、int、uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回它们与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int2**

采样之前应用于纹理坐标的纹素中的偏移量。 必须是文本值。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmp 方法

Texture2D：：GatherCmpAlpha 方法

项目 • 2022/09/24

采样并比较 Texture2D 并返回 alpha 组件。

重载列表

方法	说明
GatherCmpAlpha (S，float，float，int)	采样和比较纹理并返回 alpha 组件。
GatherCmpAlpha (S，float，float，int，uint)	采样和比较纹理并返回 alpha 组件以及有关操作的状态。
GatherCmpAlpha (S，float，float，int2，int2，int2，int2)	采样和比较纹理并返回 alpha 组件。
GatherCmpAlpha (S，float，float，int2，int2，int2，int2，uint)	采样和比较纹理并返回 alpha 组件以及有关操作的状态。

另请参阅

Texture2D

Texture2D：：GatherCmpAlpha (S，float，float，int) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回其 alpha 分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpAlpha(
  in SamplerComparisonState s,
  in float2 location,
  in float compare_value,
  in int2 offset
);
```

参数

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

着色器模型 5

Texture2D：：GatherCmpAlpha (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值的比较。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

Texture2D：：GatherCmpAlpha (S，float，float，int，uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 组件的比较与比较值以及磁贴映射状态。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

Texture2D：：GatherCmpAlpha (S，float，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

Texture2D：：GatherCmpBlue 方法

项目 • 2022/09/22

采样并比较 Texture2D 并返回蓝色分量。

重载列表

方法	说明
GatherCmpBlue (S，float，float，int)	采样并比较纹理并返回蓝色分量。
GatherCmpBlue (S，float，float，int，uint)	采样并比较纹理，并返回蓝色组件以及操作状态。
GatherCmpBlue (S，float，float，int2，int2，int2，int2)	采样并比较纹理并返回蓝色分量。
GatherCmpBlue (S，float，float，int2，int2，int2，int2，uint)	采样并比较纹理，并返回蓝色组件以及操作状态。

另请参阅

Texture2D

Texture2D：：GatherCmpBlue (S，float，float，int) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpBlue(
  in SamplerComparisonState s,
  in float2 location,
  in float compare_value,
  in int2 offset
);
```

parameters

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

着色器模型 5

Texture2D：：GatherCmpBlue (S，float，float，int2，int2，int2，int2，int2) 函数

项目 • 2022/09/23

对于要在双线性筛选运算中使用的四个纹素值，返回其蓝色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

Texture2D：：GatherCmpBlue (S、float、float、int、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

Texture2D：：GatherCmpBlue (S，float，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

Texture2D：：GatherCmpGreen 方法

项目 • 2022/09/24

采样并比较 Texture2D 并返回绿色分量。

重载列表

方法	说明
GatherCmpGreen (S，float，float，int)	采样并比较纹理并返回绿色分量。
GatherCmpGreen (S，float，float，int，uint)	采样并比较纹理，并返回绿色组件以及有关操作的状态。
GatherCmpGreen (S，float，float，int2，int2，int2，int2)	采样并比较纹理并返回绿色分量。
GatherCmpGreen (S，float，float，int2，int2，int2，int2，uint)	采样并比较纹理，并返回绿色组件以及有关操作的状态。

另请参阅

Texture2D

Texture2D：：GatherCmpGreen (S，float，float，int) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpGreen(
  in SamplerComparisonState s,
  in float2 location,
  in float compare_value,
  in int2 offset
);
```

parameters

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

着色器模型 5

Texture2D：：GatherCmpGreen (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

Texture2D：：GatherCmpGreen (S、float、float、int、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

Texture2D：：GatherCmpGreen (S，float，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

Texture2D：：GatherCmpRed 方法

项目 • 2022/09/22

采样并比较 Texture2D 并返回红色分量。

重载列表

方法	说明
GatherCmpRed (S，float，float，int)	采样和比较纹理并返回红色分量。
GatherCmpRed (S，float，float，int，uint)	采样和比较纹理，并返回红色组件以及有关操作的状态。
GatherCmpRed (S，float，float，int2，int2，int2，int2)	采样和比较纹理并返回红色分量。
GatherCmpRed (S，float，float，int2，int2，int2，int2，uint)	采样和比较纹理，并返回红色组件以及有关操作的状态。

另请参阅

Texture2D

Texture2D：：GatherCmpRed (S、float、float、int) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpRed(
  in SamplerComparisonState s,
  in float2 location,
  in float compare_value,
  in int2 offset
);
```

参数

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

着色器模型 5

Texture2D：：GatherCmpRed (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

Texture2D：：GatherCmpRed (S，float，float，int，uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

Texture2D：：GatherCmpRed (S，float，float，int2，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

Texture2D：：GatherGreen 方法

项目 • 2022/09/22

采样 Texture2D 并返回绿色组件。

重载列表

方法	说明
GatherGreen (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的绿色分量。
GatherGreen (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的绿色分量，以及图块映射状态。
GatherGreen (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的绿色分量。
GatherGreen (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的绿色分量，以及图块映射状态。

另请参阅

Texture2D

Texture2D：：GatherGreen (S，float，int) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherGreen(
  in sampler s,
  in float2 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

着色器模型 5

Texture2D：：GatherGreen (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的绿色分量。

语法

syntax

```
TemplateType GatherGreen(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

Texture2D：：GatherGreen (S，float，int，uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的绿色组件，以及图块映射状态。

语法

syntax

```
TemplateType GatherGreen(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

Texture2D：：GatherGreen (S，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的绿色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherGreen(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

Texture2D：：GatherRed 方法

项目 • 2022/09/23

返回将在双线性筛选操作中使用的 Texture2D 四个纹素值的红色组件。

重载列表

方法	说明
GatherRed (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
GatherRed (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。
GatherRed (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
GatherRed (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

另请参阅

Texture2D

Texture2D：：GatherRed (S，float，int) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherRed(
  in sampler s,
  in float2 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float2**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

着色器模型 5

Texture2D：：GatherRed (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量。

语法

syntax

```
TemplateType GatherRed(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

Texture2D：：GatherRed (S，float，int，uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherRed(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

Texture2D：：GatherRed (S，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherRed(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

Texture2D：：GetDimensions 函数

项目 • 2022/09/24

返回资源的维度。

语法

syntax

```
void GetDimensions(
  in
            uint MipLevel,
  out
            uint Width,
  out uint Height,
  out
            uint NumberOfLevels
);
```

参数

MipLevel [in]

类型： **uint**

可选。 如果使用 *NumberOfLevels*），则必须指定 mipmap 级别 (。

宽度 [out]

类型： **uint**

资源宽度（以纹素为单位）。

高度 [out]

类型： **uint**

资源高度（以纹素为单位）。

NumberOfLevels [out]

类型： **uint**

mipmap 级别数 (还需要 *MipLevel*) 。

返回值

无

注解

这是此方法重载版本的列表。

```
void GetDimensions(uint MipLevel,
  out uint Width,
  out uint Height,
  out uint NumberOfLevels);

void GetDimensions (out uint Width,
  out uint Height);

void GetDimensions(uint MipLevel,
  out float Width,
  out float Height,
  out float NumberOfLevels);

void GetDimensions(out float Width,
  out float Height);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Texture2D

着色器模型 5

Texture2D：：Load 方法

项目 • 2022/09/22

从 Texture2D **读取纹理**数据。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture2D

Texture2D：：Load (int，int，uint) 函数

项目 • 2022/09/22

读取纹理数据并返回操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  in  int  Offset,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理坐标。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture2D 对象的声明中的类型匹配。

另请参阅

加载方法

Texture2D：：mips。运算符函数

项目 • 2022/09/22

返回只读资源变量。

语法

```
syntax
```

```
R mips.Operator[][](
  in uint mipSlice,
  in uint2 pos
);
```

参数

mipSlice [in]

类型： **uint**

mip 切片索引。

pos [in]

类型： **uint2**

索引位置。 包含 (x、y) 坐标。

返回值

类型： **R**

只读资源变量。

注解

用法示例

```
```

```
Texture2D<float4> tex;
uint mip = 2;
uint2 pos_xy = {123, 456};
float4 f = tex.mips[mip][pos_xy];
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2D

着色器模型 5

Texture2D：：Operator 函数

项目 • 2022/09/24

返回只读资源变量。

语法

syntax

```
R Operator[](
  in uint2 pos
);
```

parameters

pos [in]

类型： **uint2**

索引位置。 包含 (x、y) 坐标。

返回值

类型： **R**

只读资源变量。

注解

此方法始终访问第一个 mip 级别。 若要指定其他 mip 级别，请改用 mip.operator[][] 方法。

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2D

着色器模型 5

Texture2D：：Sample 方法

项目 • 2022/09/23

采样 **Texture2D**。

重载列表

方法	说明
示例 (S、float、int)	采样纹理。
示例 (S、float、int、float)	将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。
示例 (S、float、int、float、uint)	对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture2D

示例 (S、float、int、float) 函数 (HLSL 引用)

项目 • 2022/09/22

使用可选值对 Texture2D 采样，以将样本详细级别固定 (LOD) 值。

> ⓘ **备注**
>
> 需要 **着色器模型 5** 或更高版本。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float Location,
  in int Offset,
  in float Clamp
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 纹理偏移量必须是静态的。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用纹理坐标偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

备注

纹理采样使用纹素位置查找纹素值。 在查找之前，可以向位置应用偏移量。 采样器状态包含采样和筛选选项。 可以在像素着色器中调用此方法，但顶点着色器或几何着色器不支持此方法。

仅在整数位处使用偏移量;否则，可能会根据硬件实现或驱动程序设置获得不同的结果。

计算纹素位置

纹理坐标是引用纹理数据的浮点值，也称为规范化纹理空间。 按此顺序应用地址换行模式 (纹理坐标 + 偏移量 + 换行模式) 以修改 [0...1] 范围之外的纹理坐标。

对于纹理数组，location 参数中的附加值指定纹理数组中的索引。 此索引被视为缩放浮点值 (而不是标准纹理坐标的规范化空间) 。 对整数索引的转换按以下顺序完成， (float +

舍入到最接近的整数 + 固定到数组范围) 。

应用纹理坐标偏移量

偏移参数在纹素空间中修改纹理坐标。 即使纹理坐标是规范化的浮点数，偏移量也会应用整数偏移量。 另请注意，纹理偏移量必须是静态的。

返回的数据格式由纹理格式确定。 例如，如果使用 DXGI_FORMAT_A8B8G8R8_UNORM_SRGB 格式定义纹理资源，则采样操作会将采样纹素从 gamma 2.0 转换为 1.0，筛选结果，并将结果作为浮点值写入 [0..1]。

仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。

另请参阅

示例方法

Texture-Object

示例 (S、float、int、float、uint) 函数 (HLSL 引用)

项目 • 2022/09/22

使用可选值对 Texture2D 采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

> **备注**
>
> 需要 **着色器模型 5** 或更高版本。

语法

```
syntax
```

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float Location,
  in  int Offset,
  in  float Clamp,
  out uint Status
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏移量 [in]

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 纹理偏移量必须是静态的。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用纹理坐标偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

注解

纹理采样使用纹素位置查找纹素值。 可在查找之前将偏移量应用于位置。 采样器状态包含采样和筛选选项。 此方法可以在像素着色器中调用，但顶点着色器或几何着色器不支持此方法。

仅在整数错误位处使用偏移量;否则，可能会根据硬件实现或驱动程序设置获得不同的结果。

计算纹素位置

纹理坐标是引用纹理数据的浮点值，也称为规范化纹理空间。 按此顺序应用地址换行模式，（纹理坐标 + 偏移量 + 包装模式）以修改 [0...1] 范围之外的纹理坐标。

对于纹理数组，location 参数中的附加值指定纹理数组中的索引。 此索引被视为缩放浮点值（而不是标准纹理坐标的规范化空间）。 对整数索引的转换按以下顺序完成，（float + 舍入到最接近的整数 + 固定到数组范围）。

应用纹理坐标偏移量

偏移参数在纹素空间中修改纹理坐标。 即使纹理坐标是规范化的浮点数，偏移量也会应用整数偏移量。 另请注意，纹理偏移量必须是静态的。

返回的数据格式由纹理格式确定。 例如，如果使用 DXGI_FORMAT_A8B8G8R8_UNORM_SRGB格式定义纹理资源，则采样操作会将采样的纹素从 gamma 2.0 转换为 1.0，筛选并将结果写入为 [0..1] 范围内的浮点值。

另请参阅

示例方法

Texture-Object

Texture2D：：SampleBias 方法

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对 Texture2D 采样。

重载列表

方法	说明
SampleBias (S，float，float，int)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S，float，float，int，float)	将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。
SampleBias (S，float，float，int，float，uint)	将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture2D

SampleBias：：Texture2D 的 SampleBias (S、float、float、int、float) 函数

项目 • 2022/09/23

在将偏差值应用于 mipmap 级别后，对 **Texture2D** 采样，其中可选值用于将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（0.0 到 1.0（含 0.0）之间的浮点数在采样之前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleBias 方法

SampleBias： ：SampleBias (S、float、int、float、float、uint) function for Texture2D

项目 • 2022/09/23

将偏差值应用于 mipmap 级别后，对 Texture2D 采样，并提供可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

Texture2D：：SampleCmp 方法

项目 • 2022/09/22

使用比较值来拒绝样本，对 Texture2D 采样。

重载列表

方法	说明
SampleCmp (S，float，float，int)	使用比较值来采样纹理，以拒绝样本。
SampleCmp (S，float，float，int，float)	使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样，(LOD) 值。
SampleCmp (S，float，float，int，float，uint)	使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样，(LOD) 值。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture2D

SampleCmp： ：SampleCmp (S、float、float、int、float) 函数 for Texture2D

项目 • 2022/09/24

使用比较值来拒绝样本，使用可选值对 Texture2D 进行采样，以便将样本详细级别固定到 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

SampleCmp： ：SampleCmp (S、float、int、float、float、uint) function for Texture2D

项目 • 2022/09/22

使用比较值来拒绝样本，使用可选值对 Texture2D 进行采样，以便将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

Texture2D：：SampleCmpLevelZero 方法

项目 • 2022/09/24

仅对 mipmap 级别 0 上的 Texture2D 采样，使用比较值来拒绝样本。

重载列表

方法	说明
SampleCmpLevelZero (S，float，float，int)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。
SampleCmpLevelZero (S，float，float，int，uint)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

备注

请参阅 Texture-Object。

另请参阅

Texture2D

SampleCmpLevelZero： ：SampleCmpLevelZero (S，float，float，int，uint) 函数用于 Texture2D

项目 • 2022/09/22

仅对 mipmap 级别 0 上的 **Texture2D** 采样，并将结果与比较值进行比较。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmpLevelZero(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleCmpLevelZero 方法

Texture2D：：SampleGrad 方法

项目 • 2022/09/23

使用渐变对 Texture2D 采样，以影响计算样本位置的方式。

重载列表

方法	说明
SampleGrad (S、float、float、float、int)	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleGrad (S、float、float、float、int、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。
SampleGrad (S、float、float、float、int、float、uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture2D

SampleGrad：：Texture2D 的 SampleGrad (S、float、float、float、int、float) 函数

项目 • 2022/09/22

使用渐变对 Texture2D 采样，以影响样本位置的计算方式，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in int          Offset,
  in float        Clamp
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

SampleGrad：：SampleGrad (S、float、float、float、int、float、uint) function for Texture2D

项目 • 2022/09/23

使用渐变对 Texture2D 采样，以影响样本位置的计算方式，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2

Texture-Object类型	参数类型
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture2D：：SampleLevel 方法

项目 • 2022/09/24

对指定 mipmap 级别的 Texture2D 采样。

重载列表

方法	说明
SampleLevel (S，float，float，int)	对指定 mipmap 级别的纹理采样。
SampleLevel (S，float，float，int，uint)	对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

注解

请参阅 Texture-Object。

另请参阅

Texture2D

SampleLevel：：SampleLevel (S、float、float、int、uint) function for Texture2D

项目 • 2022/09/22

对指定 mipmap 级别上的 Texture2D 采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleLevel 方法

Texture2DArray

项目 • 2022/09/22

Texture2DArray 类型 (，因为它存在于着色器模型 4) 加上资源变量中。 除了着色器模型 4 中的方法外，此纹理对象还支持以下方法。

方法	说明
Gather	返回将在双线性筛选操作中使用的四个纹素值。
GatherRed	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
GatherGreen	返回将在双线性筛选操作中使用的四个纹素值的绿色分量。
GatherBlue	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherAlpha	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherCmp	对于要在双线性筛选操作中使用的四个纹素值，将返回它们与比较值进行比较。
GatherCmpRed	对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值进行比较。
GatherCmpGreen	对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值进行比较。
GatherCmpBlue	对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值进行比较。
GatherCmpAlpha	对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值的比较。
GetDimensions	获取资源维度。
加载	读取纹理数据。
Mips。Operator[][]	获取只读资源变量。
运算符[]	获取只读资源变量。
示例	对纹理进行采样。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleCmp	使用比较值对纹理进行采样，以拒绝样本。
SampleCmpLevelZero	仅使用比较值来拒绝样本，将纹理 (mipmap 级别 0 采样) 。
SampleGrad	使用渐变对纹理进行采样，以影响样本位置的计算方式。

方法	说明
SampleLevel	对指定 mipmap 级别的纹理采样。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture2DArray：：Gather 方法

项目 • 2022/09/22

返回将在双线性筛选操作中使用的 Texture2DArray 的四个纹素值。

有关 gather4 的文档，了解有关基础 DXBC 指令的详细信息。

重载列表

方法	说明
收集 (S、float、int)	返回将在双线性筛选操作中使用的四个纹素值。
收集 (S、float、int、uint)	返回将在双线性筛选操作中使用的四个纹素值，以及图块映射状态。

另请参阅

Texture2DArray

Texture2DArray：：Gather (S，float，int) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值。

语法

syntax

```
TemplateType Gather(
  in sampler s,
  in float3 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

收集方法

着色器模型 5

Texture2DArray：：Gather (S，float，int，uint) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值。

语法

syntax

```
TemplateType Gather(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

收集方法

Texture2DArray：：GatherAlpha 方法

项目 • 2022/12/15

返回 **纹理** 2DArray 的四个纹素值的 alpha 分量，该值将用于双线性筛选操作。

重载列表

方法	说明
GatherAlpha (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherAlpha (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量，以及图块映射状态。
GatherAlpha (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherAlpha (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量，以及图块映射状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherAlpha (S，float，int) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。

语法

syntax

```
TemplateType GatherAlpha(
  in sampler s,
  in float3 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

着色器模型 5

Texture2DArray：：GatherAlpha (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。

语法

syntax

```
TemplateType GatherAlpha(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

Texture2DArray：：GatherAlpha (S，float，int，uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值，以及磁贴映射状态。

语法

syntax

```
TemplateType GatherAlpha(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

Texture2DArray：：GatherAlpha (S，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherAlpha(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

Texture2DArray：：GatherBlue 方法

项目 • 2022/09/24

返回将在双线性筛选操作中使用的 Texture2DArray 四个纹素值的蓝色组件。

重载列表

方法	说明
GatherBlue (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherBlue (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。
GatherBlue (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherBlue (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherBlue (S，float，int) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。

语法

syntax

```
TemplateType GatherBlue(
  in sampler s,
  in float3 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

着色器模型 5

GatherBlue (S，float，int2，int2，int2，int2) 函数 (HLSL 参考)

项目 • 2023/03/06

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。

语法

syntax

```
TemplateType GatherBlue(
  in SamplerState S,
  in float3       Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

Texture2DArray：：GatherBlue (S、float、int、uint) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherBlue(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

Texture2DArray：：GatherBlue (S，float，int2，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

语法

syntax

```
TemplateType GatherBlue(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DArray：：GatherCmp 方法

项目 • 2022/09/22

对于将在双线性筛选操作中使用的 Texture2DArray 的四个纹素值，将返回它们与比较值进行比较。

有关 gather4_c 的文档，请参阅描述基础 DXBC 指令的详细信息。

重载列表

方法	说明
GatherCmp (S，float，float，int)	采样并比较纹理并返回所有四个组件。
GatherCmp (S，float，float，int，uint)	采样并比较纹理，并返回所有四个组件以及操作状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherCmp (S、float、float、int) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回它们与比较值进行比较。

语法

syntax

```
float4 GatherCmp(
  in SamplerComparisonState s,
  in float3 location,
  in float compare_value,
  in int2 offset
);
```

参数

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmp 方法

着色器模型 5

Texture2DArray：：GatherCmp (S、float、float、int、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，返回它们与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmp 方法

Texture2DArray：：GatherCmpAlpha 方法

项目 • 2022/09/22

采样并比较 Texture2DArray 并返回 alpha 分量。

重载列表

方法	说明
GatherCmpAlpha (S，float，float，int)	采样并比较纹理并返回 alpha 分量。
GatherCmpAlpha (S，float，float，int，uint)	采样并比较纹理并返回 alpha 组件以及操作状态。
GatherCmpAlpha (S，float，float，int2，int2，int2，int2)	采样并比较纹理并返回 alpha 分量。
GatherCmpAlpha (S，float，float，int2，int2，int2，int2，uint)	采样并比较纹理并返回 alpha 组件以及操作状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherCmpAlpha (S，float，float，int) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其 alpha 分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpAlpha(
  in SamplerComparisonState s,
  in float3 location,
  in float compare_value,
  in int2 offset
);
```

参数

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

着色器模型 5

Texture2DArray：：GatherCmpAlpha (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值的比较。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

Texture2DArray：：GatherCmpAlpha (S、float、float、int、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 组件的比较与比较值以及磁贴映射状态。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

Texture2DArray：：GatherCmpAlpha (S，float，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 组件的比较与比较值以及磁贴映射状态。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

Texture2DArray：：GatherCmpBlue 方法

项目 • 2022/09/22

示例并比较 Texture2DArray 并返回蓝色组件。

重载列表

方法	说明
GatherCmpBlue (S，float，float，int)	采样和比较纹理并返回蓝色组件。
GatherCmpBlue (S，float，float，int，uint)	采样和比较纹理并返回蓝色组件以及操作状态。
GatherCmpBlue (S，float，float，int2，int2，int2，int2)	采样和比较纹理并返回蓝色组件。
GatherCmpBlue (S，float，float，int2，int2，int2，int2，uint)	采样和比较纹理并返回蓝色组件以及操作状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherCmpBlue (S，float，float，int) 函数

项目 • 2022/09/22

对于要在双线性筛选运算中使用的四个纹素值，返回其蓝色分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpBlue(
  in SamplerComparisonState s,
  in float3 location,
  in float compare_value,
  in int2 offset
);
```

parameters

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

着色器模型 5

Texture2DArray：：GatherCmpBlue (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

对于要在双线性筛选运算中使用的四个纹素值，返回其蓝色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

Texture2DArray：：GatherCmpBlue (S，float，float，int，uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回其蓝色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

Texture2DArray：：GatherCmpBlue (S，float，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

Texture2DArray：：GatherCmpGreen 方法

项目 • 2022/09/22

采样并比较 Texture2DArray 并返回绿色组件。

重载列表

方法	说明
GatherCmpGreen (S，float，float，int)	采样并比较纹理并返回绿色分量。
GatherCmpGreen (S，float，float，int，uint)	采样并比较纹理，并返回绿色组件以及有关操作的状态。
GatherCmpGreen (S，float，float，int2，int2，int2，int2)	采样并比较纹理并返回绿色分量。
GatherCmpGreen (S，float，float，int2，int2，int2，int2，uint)	采样并比较纹理，并返回绿色组件以及有关操作的状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherCmpGreen (S，float，float，int) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpGreen(
  in SamplerComparisonState s,
  in float3 location,
  in float compare_value,
  in int2 offset
);
```

参数

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

着色器模型 5

Texture2DArray：：GatherCmpGreen (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

Texture2DArray：：GatherCmpGreen (S、float、float、int、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

Texture2DArray：：GatherCmpGreen (S，float，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

Texture2DArray：：GatherCmpRed 方法

项目 • 2022/09/23

采样并比较 Texture2DArray 并返回红色组件。

重载列表

方法	说明
GatherCmpRed (S，float，float，int)	采样和比较纹理并返回红色分量。
GatherCmpRed (S，float，float，int，uint)	采样和比较纹理，并返回红色组件以及有关操作的状态。
GatherCmpRed (S，float，float，int2，int2，int2，int2)	采样和比较纹理并返回红色分量。
GatherCmpRed (S，float，float，int2，int2，int2，int2，uint)	采样和比较纹理，并返回红色组件以及有关操作的状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherCmpRed (S，float，float，int) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值进行比较。

语法

syntax

```
float4 GatherCmpRed(
  in SamplerComparisonState s,
  in float3 location,
  in float compare_value,
  in int2 offset
);
```

参数

s [in]

类型： **SamplerComparisonState**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

compare_value [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **float4**

一个四分量值，每个组件是每个组件比较的结果。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

着色器模型 5

Texture2DArray：：GatherCmpRed (S，float，float，int2，int2，int2，int2) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值进行比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

Texture2DArray：：GatherCmpRed (S，float，float，int，uint) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

Texture2DArray：：GatherCmpRed (S，float，float，int2，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

Texture2DArray：：GatherGreen 方法

项目 • 2022/12/15

返回将在双线性筛选操作中使用的 Texture2DArray 四个纹素值的绿色组件。

重载列表

方法	说明
GatherGreen (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的绿色组件。
GatherGreen (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的绿色组件，以及图块映射状态。
GatherGreen (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的绿色组件。
GatherGreen (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的绿色组件，以及图块映射状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherGreen (S，float，int) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的绿色分量。

语法

syntax

```
TemplateType GatherGreen(
  in sampler s,
  in float3 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

着色器模型 5

Texture2DArray：：GatherGreen (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的绿色分量。

语法

syntax

```
TemplateType GatherGreen(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

Texture2DArray：：GatherGreen (S，float，int，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的绿色组件，以及图块映射状态。

语法

syntax

```
TemplateType GatherGreen(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

Texture2DArray：：GatherGreen (S，float，int2，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的绿色组件，以及图块映射状态。

语法

syntax

```
TemplateType GatherGreen(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DArray：：GatherRed 方法

项目 • 2022/09/24

返回用于双线性筛选操作的 Texture2DArray 四个纹素值的红色分量。

重载列表

方法	说明
GatherRed (S，float，int)	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
GatherRed (S，float，int，uint)	返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。
GatherRed (S，float，int2，int2，int2，int2)	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
GatherRed (S，float，int2，int2，int2，int2，uint)	返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

另请参阅

Texture2DArray

Texture2DArray：：GatherRed (S，float，int) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的红色分量。

语法

syntax

```
TemplateType GatherRed(
  in sampler s,
  in float3 location,
  in int2 offset
);
```

参数

s [in]

类型： **采样器**

从零开始的采样器索引。

位置 [in]

类型： **float3**

示例坐标 (u，v) 。

offset [in]

类型： **int2**

采样前应用于纹理坐标的偏移量。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

着色器模型 5

Texture2DArray：：GatherRed (S，float，int2，int2，int2，int2) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量。

语法

syntax

```
TemplateType GatherRed(
  in SamplerState S,
  in float        Location,
  in int2         Offset1,
  in int2         Offset2,
  in int2         Offset3,
  in int2         Offset4
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样前应用于纹理坐标的第一个偏移量组件。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量组件。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量组件。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

Texture2DArray：：GatherRed (S，float，int，uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherRed(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

偏移量 [in]

类型： **int**

采样之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

Texture2DArray：：GatherRed (S，float，int2，int2，int2，int2，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherRed(
  in  SamplerState S,
  in  float        Location,
  in  int2         Offset1,
  in  int2         Offset2,
  in  int2         Offset3,
  in  int2         Offset4,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

Offset1 [in]

类型： **int2**

采样之前应用于纹理坐标的第一个偏移量。

Offset2 [in]

类型： **int2**

采样前应用于纹理坐标的第二个偏移量组件。

Offset3 [in]

类型： **int2**

采样前应用于纹理坐标的第三个偏移量。

Offset4 [in]

类型： **int2**

采样前应用于纹理坐标的第四个偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DArray：：GetDimensions 函数

项目 • 2022/09/22

返回资源的维度。

语法

syntax

```
void GetDimensions(
  in  UINT MipLevel,
  out UINT Width,
  out UINT Height,
  out UINT Elements,
  out UINT NumberOfLevels
);
```

参数

MipLevel [in]

类型： **UINT**

可选。 如果使用 *NumberOfLevels*），则必须指定 mipmap 级别 (。

宽度 [out]

类型： **UINT**

资源宽度（以纹素为单位）。

高度 [out]

类型： **UINT**

资源高度（以纹素为单位）。

元素 [out]

类型： **UINT**

数组中的元素数。

NumberOfLevels [out]

类型： UINT

mipmap 级别数 (还需要 *MipLevel*) 。

返回值

无

备注

这是此方法的重载版本的列表。

```
void GetDimensions(UINT MipLevel,
  out UINT Width,
  out UINT Height,
  out UINT Elements,
  out UINT NumberOfLevels);

void GetDimensions (out UINT Width,
  out float Height,
  out float Elements);

void GetDimensions(UINT MipLevel,
  out float Width,
  out float Height,
  out float Elements,
  out float NumberOfLevels);

void GetDimensions(out float Width,
  out float Height,
  out float Elements);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Texture2DArray

Texture2DArray：：Load 方法

项目 • 2022/09/23

读取 Texture2DArray 数据。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture2DArray

Texture2DArray：：Load (int，int，uint) 函数

项目 • 2022/09/23

读取纹理数据并返回操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  in  int  Offset,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理坐标。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture2DArray 对象的声明中的类型匹配。

另请参阅

加载方法

Texture2DArray：：mips。运算符函数

项目 • 2022/09/24

返回只读资源变量。

语法

```
syntax
```

```
R mips.Operator[][](
  in uint mipSlice,
  in uint3 pos
);
```

参数

mipSlice [in]

类型： **uint**

mip 切片索引。

pos [in]

类型： **uint3**

索引位置。 第一个和第二个组件包含 (x、y) 坐标。 第三个组件指示所需的数组切片。

返回值

类型： **R**

只读资源变量。

注解

用法示例

```
Texture2DArray<float4> tex;
uint mip = 2;
uint2 pos_xy_and_array = {123, 456, 3};
float4 f = tex.mips[mip][pos_xy_and_array];
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DArray

着色器模型 5

Texture2DArray：：Operator 函数

项目 • 2022/09/23

返回只读资源变量。

语法

syntax

```
R Operator[](
  in uint3 pos
);
```

参数

pos [in]

类型： **uint3**

索引位置。 第一个和第二个组件包含 (x、y) 坐标。 第三个组件指示所需的数组切片。

返回值

类型： **R**

只读资源变量。

注解

此方法始终访问第一个 mip 级别。 若要指定其他 mip 级别，请改用 mip.operator[][] 方法。

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DArray

着色器模型 5

Texture2DArray：：Sample 方法

项目 • 2022/09/22

采样 Texture2DArray。

重载列表

方法	说明
示例 (S、float、int)	采样纹理。
示例 (S、float、int、float)	将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。
示例 (S、float、int、float、uint)	对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

另请参阅

Texture2DArray

Texture2DArray：：示例 (S、float、int、float) 函数

项目 • 2022/09/22

将具有可选值的纹理采样，以将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float        Location,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹

理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

示例方法

Texture2DArray：：Sample (S，float，int，float，uint) 函数

项目 • 2022/09/22

对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

语法

syntax

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

示例方法

Texture2DArray：：SampleBias 方法

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对 Texture2DArray 采样。

重载列表

方法	说明
SampleBias (S，float，float，int)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S、float、float、int、float)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。
SampleBias (S、float、float、int、float、uint)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture2DArray

SampleBias：：SampleBias (S、float、float、int、float) function for Texture2DArray

项目 • 2022/09/22

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

SampleBias：：Texture2DArray 的 SampleBias (S、float、float、int、float、uint) 函数

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（0.0 到 1.0（含 0.0）之间的浮点数在采样之前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

Texture2DArray：：SampleCmp 方法

项目 • 2022/09/23

使用比较值来采样 Texture2DArray，以拒绝样本。

重载列表

方法	说明
SampleCmp (S，float，float，int)	使用比较值来采样纹理，以拒绝样本。
SampleCmp (S，float，float，int，float)	使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样，(LOD) 值。
SampleCmp (S，float，float，int，float，uint)	使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样，(LOD) 值。 返回有关操作的状态。

另请参阅

Texture2DArray

SampleCmp： ：Texture2DArray 的 SampleCmp (S、float、float、int、float) 函数

项目 • 2022/09/22

使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

SampleCmp： ：Texture2DArray 的 SampleCmp (S、float、float、int、float、uint) 函数

项目 • 2022/09/22

使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴中提取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

Texture2DArray

Texture2DArray： ： SampleCmpLevelZero 方法

项目 • 2022/12/15

仅对 mipmap 级别 0 上的 Texture2DArray 采样，使用比较值来拒绝样本。

重载列表

方法	说明
SampleCmpLevelZero (S、float、float、int)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。
SampleCmpLevelZero (S、float、float、int、uint)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

另请参阅

Texture2DArray

SampleCmpLevelZero： ： SampleCmpLevelZero (S，float，float，int，uint) function for Texture2DArray

项目 • 2022/12/14

仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmpLevelZero(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  int          Offset,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleCmpLevelZero 方法

Texture2DArray

Texture2DArray：：SampleGrad 方法

项目 • 2022/09/24

使用渐变对 Texture2DArray 采样以影响示例位置的计算方式。

重载列表

方法	说明
SampleGrad (S、float、float、float、int)	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleGrad (S、float、float、float、int、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。
SampleGrad (S、float、float、float、int、float、uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture2DArray

SampleGrad：：Texture2DArray 的 SampleGrad (S、float、float、float、int、float) 函数

项目 • 2022/09/23

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture2DArray

SampleGrad：：SampleGrad (S，float，float，float，int，float，uint) function for Texture2DArray

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2

Texture-Object类型	参数类型
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

DDX [in]

类型： **float**

以 x 方向变化表面几何图形的速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture2DArray

Texture2DArray：：SampleLevel 方法

项目 • 2022/09/24

对指定 mipmap 级别上的 Texture2DArray 采样。

重载列表

方法	说明
SampleLevel (S，float，float，int)	对指定 mipmap 级别的纹理采样。
SampleLevel (S，float，float，int，uint)	对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

另请参阅

Texture2DArray

SampleLevel：：SampleLevel (S、float、float、int、uint) function for Texture2DArray

项目 • 2022/09/24

对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  in  int          Offset,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果该值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleLevel 方法

Texture2DArray

Texture3D

项目 • 2022/09/23

纹理 3D 类型 (，因为它存在于着色器模型 4) 加上资源变量中。 除了着色器模型 4 中的方法外，此纹理对象还支持以下方法。

方法	说明
GetDimensions	获取资源维度。
加载	读取纹理数据。
Mips。Operator[][]	获取只读资源变量。
运算符[]	获取只读资源变量。
示例	对纹理进行采样。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleGrad	使用渐变对纹理进行采样，以影响样本位置的计算方式。
SampleLevel	对指定 mipmap 级别的纹理采样。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture3D：：GetDimensions 函数

项目 • 2022/09/23

返回资源的维度。

语法

syntax

```
void GetDimensions(
  in  UINT MipLevel,
  out UINT Width,
  out UINT Height,
  out UINT Depth,
  out UINT NumberOfLevels
);
```

parameters

MipLevel [in]

类型： **UINT**

可选。 如果使用 *NumberOfLevels*），则必须指定 Mipmap 级别 (。

宽度 [out]

类型： **UINT**

资源宽度（以纹素为单位）。

高度 [out]

类型： **UINT**

资源高度（以纹素为单位）。

深度 [out]

类型： **UINT**

资源深度（以纹素为单位）。

NumberOfLevels [out]

类型： UINT

mipmap 级别数 (还需要 *MipLevel*) 。

返回值

无

注解

这是此方法重载版本的列表。

```
void GetDimensions(UINT MipLevel,
  out UINT Width,
  out UINT Height,
  out UINT Depth,
  out UINT NumberOfLevels);

void GetDimensions (out UINT Width,
  out UINT Height,
  out UINT Depth);

void GetDimensions(UINT MipLevel,
  out float Width,
  out float Height,
  out float Depth,
  out float NumberOfLevels);

void GetDimensions(out float Width,
  out float Height,
  out float Depth);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Texture3D

Texture3D：：Load 方法

项目 • 2022/09/23

读取纹理数据。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture3D

Texture3D：：Load (int，int，uint) 函数

项目 • 2022/09/22

读取纹理数据并返回操作的状态。

语法

```
syntax

 Load(
  in  int  Location,
  in  int  Offset,
  out uint Status
);
```

parameters

位置 [in]

类型： **int**

纹理坐标。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture3D 对象的声明中的类型匹配。

另请参阅

加载方法

Texture3D

Texture3D：：mips。运算符函数

项目 • 2022/09/22

返回只读资源变量。

语法

```
syntax
```

```
R mips.Operator[][](
  in uint mipSlice,
  in uint3 pos
);
```

parameters

mipSlice [in]

类型： **uint**

mip 切片索引。

pos [in]

类型： **uint3**

索引位置。 包含 (x、y、z) 坐标。

返回值

类型： **R**

只读资源变量。

注解

用法示例

```
```

```
Texture3D<float4> tex;
uint mip = 2;
uint3 pos_xyz = {123, 456, 789};
float4 f = tex.mips[mip][pos_xyz];
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture3D

着色器模型 5

Texture3D：：Operator 函数

项目 • 2022/09/23

返回只读资源变量。

语法

syntax

```
R Operator[](
  in uint3 pos
);
```

parameters

pos [in]

类型： **uint3**

索引位置。 包含 (x、y、z) 坐标。

返回值

类型： **R**

只读资源变量。

注解

此方法始终访问第一个 mip 级别。 若要指定其他 mip 级别，请改用 mip.operator[][] 。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture3D

着色器模型 5

Texture3D：：Sample 方法

项目 • 2022/09/24

采样纹理。

重载列表

方法	说明
示例 (S、float、int)	采样纹理。
示例 (S、float、int、float)	将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。
示例 (S、float、int、float、uint)	对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

另请参阅

Texture3D

Texture-Object

Texture3D：：示例 (S、float、int、float) 函数

项目 • 2022/09/22

将具有可选值的纹理采样，以将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float        Location,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹

理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

示例方法

Texture3D

Texture3D：：Sample (S，float，int，float，uint) 函数

项目 • 2022/09/22

对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

语法

syntax

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float        Location,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏移量 [in]

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴获取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

示例方法

Texture3D

Texture3D：：SampleBias 方法

项目 • 2022/09/24

将偏差值应用于 mipmap 级别后，对纹理采样。

重载列表

方法	说明
SampleBias (S，float，float，int)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S，float，float，int，float)	将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。
SampleBias (S，float，float，int，float，uint)	将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture3D

Texture-Object

SampleBias：：Texture3D 的 SampleBias (S、float、float、int、float) 函数

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in int          Offset,
  in float        Clamp
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（0.0 到 1.0（含 0.0）之间的浮点数在采样之前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleBias： ：SampleBias (S、float、float、int、float、uint) 函数 for Texture3D

项目 • 2022/09/23

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

偏移量 [in]

类型： **int**

可选的纹理坐标偏移量，可用于任何纹理对象类型;偏移量应用于采样前的位置。 仅在整数错误位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

Texture3D

Texture3D：：SampleGrad 方法

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响样本位置的计算方式。

重载列表

方法	说明
SampleGrad (S，float，float，float，int)	使用渐变对纹理进行采样，以影响样本位置的计算方式。
SampleGrad (S，float，float，float，int，float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。
SampleGrad (S，float，float，float，int，float，uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。 返回有关操作的状态。

另请参阅

Texture3D

Texture-Object

SampleGrad：：Texture3D 的 SampleGrad (S、float、float、float、int、float) 函数

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in int          Offset,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture3D

SampleGrad： ：SampleGrad (S、float、float、float、int、float、uint) 函数 (适用于 Texture3D)

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  int          Offset,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2

Texture-Object类型	参数类型
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleGrad 方法

Texture3D

Texture3D：：SampleLevel 方法

项目 • 2022/09/24

对指定 mipmap 级别上的纹理采样。

重载列表

方法	说明
SampleLevel (S，float，float，int)	对指定 mipmap 级别上的纹理采样。
SampleLevel (S、float、float、int、uint)	对指定 mipmap 级别上的纹理采样，并返回有关操作的状态。

另请参阅

Texture3D

Texture-Object

SampleLevel：：SampleLevel (S、float、float、int、uint) function for Texture3D

项目 • 2022/09/22

对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  in  int          Offset,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果该值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

偏移量 [in]

类型： **int**

可选纹理坐标偏移量，可用于任何纹理对象类型：偏移量应用于采样前的位置。 仅在整数位处使用偏移量;否则，可能会获得无法很好地转换为硬件的结果。 参数类型依赖于纹理对象类型。 有关详细信息，请参阅 应用整数偏移量。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	int
Texture2D、Texture2DArray	int2
Texture3D	int3
TextureCube、TextureCubeArray	不支持

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

请参阅

SampleLevel 方法

Texture3D

Texture2DMS

项目 • 2022/09/23

Texture2DMS 类型（，因为它存在于着色器模型 4) 加上资源变量中。

在本节中

主题	说明
GetDimensions	返回资源的维度。
GetSamplePosition	返回提供的示例索引的示例位置。
加载方法	从提供的位置和示例索引的资源中检索值。
样品。Operator[][]	从提供的位置和示例索引的资源中检索值。
运算符[]	从示例索引 0 提供的位置的资源检索值。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 4 及更高版本	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture2DMS：：GetDimensions 函数

项目 • 2022/09/23

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width,
  out UINT Height,
  out UINT NumberOfSamples
);
```

参数

宽度 [out]

类型： **UINT**

资源宽度（以纹素为单位）。

高度 [out]

类型： **UINT**

资源高度（以纹素为单位）。

NumberOfSamples [out]

类型： **UINT**

示例位置数。

返回值

此函数不返回值。

注解

这是此方法的重载版本的列表。

```
void GetDimensions(out float Width,
  out float Height,
  out float NumberOfSamples);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Texture2DMS

着色器模型 5

Texture2DMS：：GetSamplePosition 函数

项目 • 2022/09/23

返回提供的示例索引的示例位置。

语法

```
syntax
```

```
float2 GetSamplePosition(
  in int sampleindex
);
```

参数

sampleindex [in]

类型： **int**

示例位置的从零开始的索引。

返回值

类型： **float2**

示例位置。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DMS

着色器模型 5

Texture2DMS： ：Load 方法

项目 • 2022/09/22

从提供的位置和示例索引的资源中检索值。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、int)	读取纹理数据。
加载 (int、int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture2DMS

Texture2DMS：：Load (int，int) 函数

项目 • 2022/09/24

获取一个值。

语法

```
syntax
```

```
T Load(
  in int2 coord,
  in int sampleindex
);
```

参数

coord [in]

类型： **int2**

输入位置。

sampleindex [in]

类型： **int**

示例索引。

返回值

类型： **T**

一个值，其类型与纹理模板类型匹配。

备注

这是此方法的重载版本的列表。

```

```

```
void Load(in int2 coord,
  in int sampleindex,
```

```
    in int2 offset);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

着色器模型 5

Texture2DMS：：Load (int、int、int、uint) 函数

项目 • 2022/09/22

读取纹理数据并返回操作的状态。

语法

syntax

```
 Load(
  in  int2 Location,
  in  int  sampleindex,
  in  int2 Offset,
  out uint Status
);
```

参数

位置 [in]

类型： **int2**

纹理坐标。

sampleindex [in]

类型： **int**

示例索引。

偏移量 [in]

类型： **int2**

加载之前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture2DMS 对象的声明中的类型匹配。

另请参阅

加载方法

Texture2DMS

Texture2DMS：：sample。运算符函数

项目 • 2022/09/23

从提供的位置和示例索引的资源中检索值。

语法

syntax

```
R sample.Operator[][](
  in uint sampleSlice,
  in uint2 pos
);
```

参数

sampleSlice [in]

类型： **uint**

示例切片索引。

pos [in]

类型： **uint2**

索引位置。 这些组件包含 (x、y) 坐标。

返回值

类型： **R**

只读资源变量。

注解

用法示例

```
Texture2DMS<float4, 8> tex;

float4 main( float3 tcoord : texturecoord0 ) : SV_Target
{
     return s_msTexture.sample[2][tcoord];
}
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DMS

着色器模型 5

Texture2DMS： ：Operator 函数

项目 • 2022/09/24

从示例索引 0 提供的位置的资源检索值。

语法

```
syntax
```

```
R Operator[](
  in uint2 pos
);
```

参数

pos [in]

类型： **uint2**

索引位置。 包含 (x、y) 坐标。

返回值

类型： **R**

只读资源变量。

注解

若要选择特定示例，请参阅 **示例。Operator[][].**

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DMS

着色器模型 5

Texture2DMSArray

项目 • 2022/09/22

Texture2DMSArray 类型 (，因为它存在于着色器模型 4) 加上资源变量中。

方法	说明
GetDimensions	获取资源维度。
GetSamplePosition	获取指定样本的位置。
加载	获取一个值。
样品。 Operator[][]	获取只读资源变量。

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

Texture2DMSArray：：GetDimensions 函数

项目 • 2022/09/24

返回资源的维度。

语法

syntax

```
void GetDimensions(
  out UINT Width,
  out UINT Height,
  out UINT Elements,
  out UINT NumberOfSamples
);
```

参数

宽度 [out]

类型： **UINT**

资源宽度（以纹素为单位）。

高度 [out]

类型： **UINT**

资源高度（以纹素为单位）。

元素 [out]

类型： **UINT**

数组中的元素数。

NumberOfSamples [out]

类型： **UINT**

示例位置数。

返回值

无

注解

这是此方法的重载版本的列表。

```
void GetDimensions(out float Width,
  out float Height,
  out float Elements,
  out float NumberOfSamples);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

Texture2DMSArray

着色器模型 5

Texture2DMSArray：：GetSamplePosition 函数

项目 • 2022/09/23

获取指定样本的位置。

语法

syntax

```
float2 GetSamplePosition(
  in int sampleindex
);
```

参数

sampleindex [in]

类型： **int**

示例位置的从零开始的索引。

返回值

类型： **float2**

示例位置。

注解

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DMSArray

着色器模型 5

Texture2DMSArray： ：Load 方法

项目 • 2022/09/22

读取纹理数据。

重载列表

方法	说明
加载 (int，int)	读取纹理数据。
加载 (int、int、int)	读取纹理数据。
加载 (int、int、int、uint)	读取纹理数据并返回操作的状态。

另请参阅

Texture2DMSArray

Texture2DMSArray：：Load (int，int) 函数

项目 • 2022/09/22

获取一个值。

语法

```
syntax
```

```
T Load(
  in int3 coord,
  in int sampleindex
);
```

参数

coord [in]

类型： **int3**

输入位置。

sampleindex [in]

类型： **int**

示例索引。

返回值

类型： **T**

一个值，其类型与纹理模板类型匹配。

注解

这是此方法重载版本的列表。

```
```

```
void Load(in int3 coord,
  in int sampleindex,
  in int2 offset);
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

加载方法

着色器模型 5

Texture2DMSArray：：Load (int、int、int、uint) 函数

项目 • 2022/09/22

读取纹理数据并返回操作的状态。

语法

syntax

```
 Load(
  in  int  Location,
  in  int  sampleindex,
  in  int  Offset,
  out uint Status
);
```

参数

位置 [in]

类型： **int**

纹理坐标。

sampleindex [in]

类型： **int**

示例索引。

偏移量 [in]

类型： **int**

采样前应用于纹理坐标的偏移量。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型：

返回类型与 Texture2DMSArray 对象的声明中的类型匹配。

另请参阅

加载方法

Texture2DMSArray

Texture2DMSArray：：sample。运算符函数

项目 • 2022/09/22

返回只读资源变量。

语法

syntax

```
R sample.Operator[][](
  in uint sampleSlice,
  in uint3 pos
);
```

参数

sampleSlice [in]

类型： **uint**

示例切片索引。

pos [in]

类型： **uint3**

索引位置。 第一个和第二个组件包含 (x、y) 坐标。 第三个组件指示所需的数组切片。

返回值

类型： **R**

只读资源变量。

注解

用法示例

```
Texture2DMSArray<float4, 4> alpha;

float4 main( float3 tcoord : texturecoord0 ) : SV_Target
{
     return s_msTexture.sample[2][tcoord];
}
```

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

Texture2DMSArray

着色器模型 5

TextureCube 对象

项目 • 2022/09/22

TextureCube 类型 (，因为它存在于着色器模型 4) 加上资源变量中。 除了着色器模型 4 中的方法外，此纹理对象还支持这些方法。

- 方法

方法

TextureCube 对象具有这些方法。

方法	说明
Gather	返回将在双线性筛选操作中使用的四个纹素值。
GatherAlpha	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherBlue	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherCmp	对于要在双线性筛选运算中使用的四个纹素值，返回它们与比较值进行比较。
GatherCmpAlpha	对于要在双线性筛选操作中使用的四个纹素值，返回其 alpha 分量与比较值进行比较。
GatherCmpBlue	对于要在双线性筛选运算中使用的四个纹素值，返回其蓝色分量与比较值进行比较。
GatherCmpGreen	对于要在双线性筛选操作中使用的四个纹素值，返回其绿色分量与比较值进行比较。
GatherCmpRed	对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值进行比较。
GatherGreen	返回将在双线性筛选操作中使用的四个纹素值的绿色组件。
GatherRed	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
示例	采样纹理。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleCmp	使用比较值来采样纹理，以拒绝样本。
SampleCmpLevelZero	仅) 采样纹理 (mipmap 级别 0，使用比较值来拒绝样本。
SampleGrad	使用渐变对纹理进行采样，以影响计算样本位置的方式。

方法	说明
SampleLevel	对指定 mipmap 级别上的纹理采样。

注解

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

TextureCube： ：Gather 方法

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值。

有关 gather4 的文档，了解有关基础 DXBC 指令的详细信息。

重载列表

方法	说明
收集 (S，float)	对纹理采样并返回四个样本，（红色分量仅）用于双线性内插。
收集 (S、float、uint)	采样纹理并返回所有四个组件以及操作状态。

另请参阅

TextureCube

TextureCube：：Gather (S、float、uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值。

语法

syntax

```
TemplateType Gather(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

收集方法

TextureCube

TextureCube：：GatherAlpha 方法

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。

重载列表

方法	说明
GatherAlpha (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量，以及图块映射状态。

另请参阅

TextureCube

TextureCube：：GatherAlpha (S、float、uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的 alpha 组件，以及图块映射状态。

语法

syntax

```
TemplateType GatherAlpha(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

TextureCube

TextureCube： ：GatherBlue 方法

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。

重载列表

方法	说明
GatherBlue (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

另请参阅

TextureCube

TextureCube：：GatherBlue (S、float、uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

语法

syntax

```
TemplateType GatherBlue(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

TextureCube

TextureCube： ：GatherCmp 方法

项目 • 2022/09/23

对于将在双线性筛选操作中使用的 TextureCube 的四个纹素值，将返回它们与比较值进行比较。

有关 gather4_c 的文档，请参阅描述基础 DXBC 指令的详细信息。

重载列表

方法	说明
GatherCmp (S，float，float，uint)	采样并比较纹理，并返回所有四个组件以及操作状态。

另请参阅

TextureCube

TextureCube：：GatherCmp (S、float、float、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选运算中使用的四个纹素值，返回它们与比较值进行比较。

语法

syntax

```
TemplateType GatherCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，

CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmp 方法

TextureCube

TextureCube：：GatherCmpAlpha 方法

项目 • 2022/09/22

采样并比较纹理并返回 alpha 分量。

重载列表

方法	说明
GatherCmpAlpha (S，float，float，uint)	采样并比较纹理并返回 alpha 组件以及操作状态。

另请参阅

TextureCube

TextureCube：：GatherCmpAlpha (S、float、float、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

TextureCube

TextureCube：：GatherCmpBlue 方法

项目 • 2022/09/24

采样并比较纹理并返回蓝色分量。

重载列表

方法	说明
GatherCmpBlue (S，float，float，uint)	采样并比较纹理，并返回蓝色组件以及操作状态。

另请参阅

TextureCube

TextureCube：：GatherCmpBlue (S、float、float、uint) 函数

项目 • 2022/12/15

对于要在双线性筛选操作中使用的四个纹素值，返回其蓝色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

TextureCube

TextureCube：：GatherCmpGreen 方法

项目 • 2022/09/23

采样并比较纹理并返回绿色分量。

重载列表

方法	说明
GatherCmpGreen (S、float、float、uint)	采样并比较纹理，并返回绿色组件以及有关操作的状态。

另请参阅

TextureCube

TextureCube：：GatherCmpGreen (S、float、float、uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

TextureCube

TextureCube：：GatherCmpRed 方法

项目 • 2022/09/22

采样并比较纹理并返回红色分量。

重载列表

方法	说明
GatherCmpRed (S，float，float，uint)	采样并比较纹理并返回红色组件以及操作状态。

另请参阅

TextureCube

TextureCube：：GatherCmpRed (S、float、float、uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值以及图块映射状态的比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

TextureCube

TextureCube： ：GatherGreen 方法

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的绿色组件。

重载列表

方法	说明
GatherGreen (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的绿色组件，以及图块映射状态。

另请参阅

TextureCube

TextureCube：：GatherGreen (S，float，uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的绿色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherGreen(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

TextureCube

TextureCube：：GatherRed 方法

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值的红色分量。

重载列表

方法	说明
GatherRed (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

另请参阅

TextureCube

TextureCube：：GatherRed (S、float、uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherRed(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

TextureCube

TextureCube：：Sample 方法

项目 • 2022/09/24

采样纹理。

重载列表

方法	说明
示例 (S，float)	采样纹理。
示例 (S、float、float)	将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。
示例 (S、float、float、uint)	对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

另请参阅

TextureCube

Texture-Object

TextureCube：：Sample (S，float，float) 函数

项目 • 2022/09/23

将具有可选值的纹理采样，以固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float        Location,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

示例方法

TextureCube

TextureCube：：示例 (S、float、float、uint) 函数

项目 • 2022/09/24

对具有可选值的纹理采样，以固定示例详细级别 (LOD) 值，并返回操作的状态。

语法

syntax

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float        Location,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

固定 [in]

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

示例方法

TextureCube

TextureCube：：SampleBias 方法

项目 • 2022/09/23

将偏差值应用于 mipmap 级别后，对纹理采样。

重载列表

方法	说明
SampleBias (S、float、float)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S、float、float、float)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。
SampleBias (S、float、float、float、uint)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

另请参阅

TextureCube

Texture-Object

SampleBias：：TextureCube 的 SampleBias (S、float、float、float) 函数

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（0.0 到 1.0（含 0.0）之间的浮点数在采样之前应用于 mip 级别。

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

TextureCube

SampleBias： ：SampleBias (S、float、float、float、float、uint) function for TextureCube

项目 • 2022/09/24

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleBias 方法

TextureCube

TextureCube：：SampleCmp 方法

项目 • 2022/09/23

使用比较值来采样纹理，以拒绝样本。

重载列表

方法	说明
SampleCmp (S、float、float)	使用比较值来采样纹理，以拒绝样本。
SampleCmp (S、float、float、float)	使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样，(LOD) 值。
SampleCmp (S、float、float、float、uint)	使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样，(LOD) 值。 返回有关操作的状态。

另请参阅

TextureCube

Texture-Object

SampleCmp： ：TextureCube 的 SampleCmp (S、float、float、float) 函数

项目 • 2022/09/22

使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样， (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

TextureCube

SampleCmp： ：TextureCube 的 SampleCmp (S、float、float、float、uint) 函数

项目 • 2022/09/22

使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleCmp 方法

TextureCube

TextureCube：：SampleCmpLevelZero 方法

项目 • 2022/09/23

仅对 mipmap 级别 0 上的纹理采样，使用比较值来拒绝样本。

重载列表

方法	说明
SampleCmpLevelZero (S、float、float)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。
SampleCmpLevelZero (S、float、float、uint)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

另请参阅

TextureCube

Texture-Object

SampleCmpLevelZero： ： SampleCmpLevelZero (S，float，float，uint) function for TextureCube

项目 • 2022/09/22

仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmpLevelZero(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 **CheckAccessFullyMapped** 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴获取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmpLevelZero 方法

TextureCube

TextureCube：：SampleGrad 方法

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响计算样本位置的方式。

重载列表

方法	说明
SampleGrad (S、float、float、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleGrad (S、float、float、float、float、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。
SampleGrad (S、float、float、float、float、uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

另请参阅

TextureCube

Texture-Object

SampleGrad：：SampleGrad (S、float、float、float、float、float) 函数 for TextureCube

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

以 x 方向变化表面几何图形的速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

TextureCube

SampleGrad： ：SampleGrad (S、float、float、float、float、float、uint) function for TextureCube

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

以 x 方向变化表面几何图形的速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 **CheckAccessFullyMapped** 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴获取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

TextureCube

TextureCube：：SampleLevel 方法

项目 • 2022/09/24

对指定 mipmap 级别上的纹理采样。

重载列表

方法	说明
SampleLevel (S、float、float)	对指定 mipmap 级别上的纹理采样。
SampleLevel (S、float、float、uint)	对指定 mipmap 级别上的纹理采样，并返回有关操作的状态。

另请参阅

TextureCube

Texture-Object

SampleLevel：：SampleLevel (S、float、float、uint) function for TextureCube

项目 • 2022/09/23

对指定 mipmap 级别上的纹理采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 **CheckAccessFullyMapped** 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴获取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleLevel 方法

TextureCube

TextureCubeArray 对象

项目 • 2022/09/22

TextureCubeArray 类型 (，因为它存在于着色器模型 4) 加上资源变量中。 除了着色器模型 4 中的方法外，此纹理对象还支持这些方法。

- 方法

方法

TextureCubeArray 对象具有这些方法。

方法	说明
Gather	返回将在双线性筛选操作中使用的四个纹素值。
GatherAlpha	返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。
GatherBlue	返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。
GatherCmp	对于要在双线性筛选操作中使用的四个纹素值，将返回它们与比较值进行比较。
GatherCmpAlpha	对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 分量与比较值的比较。
GatherCmpBlue	对于要在双线性筛选操作中使用的四个纹素值，将返回其蓝色分量与比较值进行比较。
GatherCmpGreen	对于要在双线性筛选操作中使用的四个纹素值，将返回其绿色分量与比较值进行比较。
GatherCmpRed	对于要在双线性筛选操作中使用的四个纹素值，将返回其红色分量与比较值进行比较。
GatherGreen	返回将在双线性筛选操作中使用的四个纹素值的绿色分量。
GatherRed	返回将在双线性筛选操作中使用的四个纹素值的红色分量。
示例	对纹理进行采样。
SampleBias	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleCmp	使用比较值对纹理进行采样，以拒绝样本。
SampleCmpLevelZero	仅使用比较值来拒绝样本，将纹理 (mipmap 级别 0 采样) 。
SampleGrad	使用渐变对纹理进行采样，以影响样本位置的计算方式。

方法	说明
SampleLevel	对指定 mipmap 级别的纹理采样。

注解

最小着色器模型

以下着色器模型中支持此对象。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此对象：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

着色器模型 5 对象

TextureCubeArray：：Gather 方法

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量。

有关 gather4 的文档，了解有关基础 DXBC 指令的详细信息。

重载列表

方法	说明
收集 (S，float)	返回四个纹素值的红色分量， (红色分量仅) 用于双线性筛选操作。
收集 (S、float、uint)	返回四个纹素值的红色分量，这些值将用于双线性筛选操作，以及图块映射状态。

另请参阅

TextureCubeArray

TextureCubeArray：：Gather (S，float，uint) 函数

项目 • 2022/09/23

返回将在双线性筛选操作中使用的四个纹素值，以及图块映射状态。

语法

syntax

```
TemplateType Gather(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

收集方法

TextureCubeArray

TextureCubeArray：：GatherAlpha 方法

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的 alpha 分量。

重载列表

方法	说明
GatherAlpha (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的 alpha 组件，以及图块映射状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherAlpha (S、float、uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的 alpha 组件，以及图块映射状态。

语法

syntax

```
TemplateType GatherAlpha(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherAlpha 方法

TextureCubeArray

TextureCubeArray：：GatherBlue 方法

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的蓝色分量。

重载列表

方法	说明
GatherBlue (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherBlue (S、float、uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的蓝色组件，以及磁贴映射状态。

语法

syntax

```
TemplateType GatherBlue(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherBlue 方法

TextureCubeArray

TextureCubeArray：：GatherCmp 方法

项目 • 2022/09/24

对于将在双线性筛选操作中使用的 TextureCubeArray 的四个纹素值，请根据比较值返回其比较值。

有关 gather4_c 的文档，请参阅描述基础 DXBC 指令的详细信息。

重载列表

方法	说明
GatherCmp (S，float，float，uint)	采样和比较纹理并返回所有四个组件以及有关操作的状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherCmp (S、float、float、uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，请根据比较值以及磁贴映射状态返回其比较值。

语法

syntax

```
TemplateType GatherCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmp 方法

TextureCubeArray

TextureCubeArray：：GatherCmpAlpha 方法

项目 • 2022/09/22

采样和比较纹理并返回 alpha 组件。

重载列表

方法	说明
GatherCmpAlpha (S、float、float、uint)	采样和比较纹理并返回 alpha 组件以及有关操作的状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherCmpAlpha (S，float，float，uint) 函数

项目 • 2022/09/22

对于要在双线性筛选操作中使用的四个纹素值，将返回其 alpha 组件的比较与比较值以及磁贴映射状态。

语法

syntax

```
TemplateType GatherCmpAlpha(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpAlpha 方法

TextureCubeArray

TextureCubeArray：：GatherCmpBlue 方法

项目 • 2022/09/22

采样并比较纹理并返回蓝色分量。

重载列表

方法	说明
GatherCmpBlue (S，float，float，uint)	采样并比较纹理，并返回蓝色组件以及操作状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherCmpBlue (S、float、float、uint) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其蓝色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpBlue(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpBlue 方法

TextureCubeArray

TextureCubeArray：：GatherCmpGreen 方法

项目 • 2022/09/24

采样和比较纹理并返回绿色组件。

重载列表

方法	说明
GatherCmpGreen (S、float、float、uint)	采样和比较纹理，并返回绿色组件以及有关操作的状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherCmpGreen (S、float、float、uint) 函数

项目 • 2022/09/23

对于要在双线性筛选操作中使用的四个纹素值，返回其绿色组件与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpGreen(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpGreen 方法

TextureCubeArray

TextureCubeArray：：GatherCmpRed 方法

项目 • 2022/09/22

采样和比较纹理并返回红色分量。

重载列表

方法	说明
GatherCmpRed (S，float，float，uint)	采样和比较纹理，并返回红色组件以及有关操作的状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherCmpRed (S、float、float、uint) 函数

项目 • 2022/09/24

对于要在双线性筛选操作中使用的四个纹素值，返回其红色分量与比较值以及磁贴映射状态的比较。

语法

syntax

```
TemplateType GatherCmpRed(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

CompareValue [in]

类型： **float**

一个值，用于将每个值与每个采样值进行比较。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： TemplateType

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherCmpRed 方法

TextureCubeArray

TextureCubeArray：：GatherGreen 方法

项目 • 2022/09/22

对纹理进行采样并返回绿色分量。

重载列表

方法	说明
GatherGreen (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的绿色分量，以及图块映射状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherGreen (S、float、uint) 函数

项目 • 2022/09/24

返回将在双线性筛选操作中使用的四个纹素值的绿色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherGreen(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

备注

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherGreen 方法

TextureCubeArray

TextureCubeArray：：GatherRed 方法

项目 • 2022/12/15

返回将在双线性筛选操作中使用的四个纹素值的红色分量。

重载列表

方法	说明
GatherRed (S、float、uint)	返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

另请参阅

TextureCubeArray

TextureCubeArray：：GatherRed (S、float、uint) 函数

项目 • 2022/09/22

返回将在双线性筛选操作中使用的四个纹素值的红色分量，以及图块映射状态。

语法

syntax

```
TemplateType GatherRed(
  in  SamplerState S,
  in  float        Location,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

从零开始的采样器索引。

位置 [in]

类型： **float**

示例坐标 (u，v) 。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： **TemplateType**

一个四分量值，其类型与模板类型相同。

注解

纹理样本可用于双线性内插。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

GatherRed 方法

TextureCubeArray

TextureCubeArray：：Sample 方法

项目 • 2022/09/22

对纹理进行采样。

重载列表

方法	说明
示例 (S，float)	对纹理进行采样。
示例 (S、float、float)	将具有可选值的纹理采样，以将样本详细级别固定 (LOD) 值。
示例 (S、float、float、uint)	使用可选值采样纹理，以固定示例详细级别 (LOD) 值，并返回操作的状态。

另请参阅

TextureCubeArray

Texture-Object

TextureCubeArray：：Sample (S，float，float) 函数

项目 • 2022/09/23

将具有可选值的纹理采样，以将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT Sample(
  in SamplerState S,
  in float        Location,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

示例方法

TextureCubeArray

TextureCubeArray：：Sample (S，float，float，uint) 函数

项目 • 2022/12/15

使用可选值采样纹理，以固定示例详细级别 (LOD) 值，并返回操作的状态。

语法

syntax

```
DXGI_FORMAT Sample(
  in  SamplerState S,
  in  float        Location,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

固定 [in]

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴中提取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

示例方法

TextureCubeArray

TextureCubeArray：：SampleBias 方法

项目 • 2022/09/22

将偏差值应用于 mipmap 级别后，对纹理采样。

重载列表

方法	说明
SampleBias (S、float、float)	将偏差值应用于 mipmap 级别后，对纹理采样。
SampleBias (S、float、float、float)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。
SampleBias (S、float、float、float、uint)	对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

另请参阅

TextureCubeArray

Texture-Object

SampleBias：：TextureCubeArray 的 SampleBias (S、float、float、float) 函数

项目 • 2022/09/23

将偏差值应用于 mipmap 级别后，对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in SamplerState S,
  in float        Location,
  in float        Bias,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（0.0 到 1.0（含 0.0）之间的浮点数在采样之前应用于 mip 级别。

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleBias 方法

TextureCubeArray

SampleBias： ：TextureCubeArray 的 SampleBias (S、float、float、float、uint) 函数

项目 • 2022/09/22

对纹理采样，在将偏差值应用于 mipmap 级别后，使用可选值将样本详细级别固定到 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleBias(
  in  SamplerState S,
  in  float        Location,
  in  float        Bias,
  in  float        Clamp,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

偏差 [in]

类型： **float**

偏差值（包含 0.0 到 1.0 之间的浮点数）在采样前应用于 mip 级别。

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleBias 方法

TextureCubeArray

TextureCubeArray：：SampleCmp 方法

项目 • 2022/09/23

使用比较值对纹理进行采样，以拒绝样本。

重载列表

方法	说明
SampleCmp (S，float，float)	使用比较值对纹理进行采样，以拒绝样本。
SampleCmp (S，float，float，float，float)	使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。
SampleCmp (S，float，float，float，uint)	使用比较值拒绝样本对纹理进行采样，可选值将样本详细级别固定 (LOD) 值。 返回有关操作的状态。

另请参阅

TextureCubeArray

Texture-Object

SampleCmp： ：TextureCubeArray 的 SampleCmp (S、float、float、float) 函数

项目 • 2022/09/23

使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样， (LOD) 值。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in SamplerState S,
  in float        Location,
  in float        CompareValue,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmp 方法

TextureCubeArray

SampleCmp： ：TextureCubeArray 的 SampleCmp (S、float、float、float、uint) 函数

项目 • 2022/09/24

使用比较值来拒绝样本，使用可选值对纹理采样采样进行采样， (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmp(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  in  float        Clamp,
  out uint         Status
);
```

parameters

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

固定 [in]

类型： **float**

要将示例 LOD 值固定到的可选值。 例如，如果传递 2.0f 作为固定值，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 CheckAccessFullyMapped 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，CheckAccessFullyMapped 将返回 TRUE。 如果从未映射的磁贴获取任何值，CheckAccessFullyMapped 将返回 FALSE。

返回值

类型： DXGI_FORMAT

纹理格式，它是 DXGI_FORMAT中列出的类型化值之一。

另请参阅

SampleCmp 方法

TextureCubeArray

TextureCubeArray：：SampleCmpLevelZero 方法

项目 • 2022/09/24

仅对 mipmap 级别 0 上的纹理采样，使用比较值来拒绝样本。

重载列表

方法	说明
SampleCmpLevelZero (S、float、float)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。
SampleCmpLevelZero (S、float、float、uint)	仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

另请参阅

TextureCubeArray

Texture-Object

SampleCmpLevelZero： ： SampleCmpLevelZero (S，float，float，uint) function for TextureCubeArray

项目 • 2022/09/22

仅对 mipmap 级别 0 上的纹理采样，并将结果与比较值进行比较。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleCmpLevelZero(
  in  SamplerState S,
  in  float        Location,
  in  float        CompareValue,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

CompareValue [in]

类型： **float**

要用作比较值的浮点值。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 **CheckAccessFullyMapped** 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴中提取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleCmpLevelZero 方法

TextureCubeArray

TextureCubeArray：：SampleGrad 方法

项目 • 2022/09/22

使用渐变对纹理进行采样，以影响计算样本位置的方式。

重载列表

方法	说明
SampleGrad (S、float、float、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式。
SampleGrad (S、float、float、float、float、float)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。
SampleGrad (S、float、float、float、float、uint)	使用渐变对纹理进行采样，以影响计算样本位置的方式，使用可选值来固定样本详细级别 (LOD) 值。 返回有关操作的状态。

另请参阅

TextureCubeArray

Texture-Object

SampleGrad：：TextureCubeArray 的 SampleGrad (S、float、float) 函数

项目 • 2022/09/24

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。

语法

```
syntax
```

```
DXGI_FORMAT SampleGrad(
  in SamplerState S,
  in float        Location,
  in float        DDX,
  in float        DDY,
  in float        Clamp
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

TextureCubeArray

SampleGrad：：TextureCubeArray 的 SampleGrad (S、float、float、float、float、uint) 函数

项目 • 2022/09/23

使用渐变对纹理进行采样，以影响计算样本位置的方式，可选值用于固定样本详细级别 (LOD) 值。 返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleGrad(
  in  SamplerState S,
  in  float        Location,
  in  float        DDX,
  in  float        DDY,
  in  float        Clamp,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	**参数类型**
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3

Texture-Object类型	参数类型
TextureCubeArray	float4

DDX [in]

类型： **float**

x 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

DDY [in]

类型： **float**

y 方向表面几何图形的变化速率。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D、Texture1DArray	FLOAT
Texture2D、Texture2DArray	float2
Texture3D、TextureCube、TextureCubeArray	float3
Texture2DMS、Texture2DMSArray	不支持

固定 [in]

类型： **float**

要固定示例 LOD 值的可选值。 例如，如果为固定值传递 2.0f，请确保没有单个示例访问小于 2.0f 的 mip 级别。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 **CheckAccessFullyMapped** 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴中提取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleGrad 方法

TextureCubeArray

TextureCubeArray：：SampleLevel 方法

项目 • 2022/09/23

对指定 mipmap 级别的纹理采样。

重载列表

方法	说明
SampleLevel (S，float，float)	对指定 mipmap 级别的纹理采样。
SampleLevel (S，float，float，uint)	对指定 mipmap 级别的纹理进行采样，并返回有关操作的状态。

另请参阅

TextureCubeArray

Texture-Object

SampleLevel：：SampleLevel (S、float、float、uint) 函数

项目 • 2022/09/22

对指定 mipmap 级别上的纹理采样，并返回有关操作的状态。

语法

syntax

```
DXGI_FORMAT SampleLevel(
  in  SamplerState S,
  in  float        Location,
  in  float        LOD,
  out uint         Status
);
```

参数

S [in]

类型： **SamplerState**

采样器状态。 这是在包含状态分配的效果文件中声明的对象。

位置 [in]

类型： **float**

纹理坐标。 参数类型依赖于纹理对象类型。

Texture-Object类型	参数类型
Texture1D	FLOAT
Texture1DArray、Texture2D	float2
Texture2DArray、Texture3D、TextureCube	float3
TextureCubeArray	float4

LOD [in]

类型： **float**

[in]一个指定 mipmap 级别的数字。 如果值≤ 0，则使用 mipmap 级别 0 (最大的地图) 。 如果提供的) 用于在两个 mipmap 级别之间内插，则小数部分值 (。

状态 [out]

类型： **uint**

操作的状态。 无法直接访问状态;而是将状态传递给 **CheckAccessFullyMapped** 内部函数。 如果相应**示例**、**收集**或**加载**操作中的所有值都访问了平铺资源中的映射磁贴，**CheckAccessFullyMapped** 将返回 **TRUE**。 如果从未映射的磁贴获取任何值，**CheckAccessFullyMapped** 将返回 **FALSE**。

返回值

类型： **DXGI_FORMAT**

纹理格式，它是 **DXGI_FORMAT**中列出的类型化值之一。

另请参阅

SampleLevel 方法

TextureCubeArray

着色器模型 5 系统值

项目 • 2022/09/23

着色器模型 5 从着色器模型 4 及更早版本以及以下新系统值实现 系统 值：

- SV_DispatchThreadID
- SV_DomainLocation
- SV_GroupID
- SV_GroupIndex
- SV_GroupThreadID
- SV_GSInstanceID
- SV_InsideTessFactor
- SV_OutputControlPointID
- SV_TessFactor

相关主题

着色器模型 5

寄存器 - vs_5_0

项目 • 2022/09/24

以下输入和输出寄存器在顶点着色器版本 5_0 中实现。

输入寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 Temp (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入 (v#)	32	R	4	是	无	是
输入资源中的元素 (t#)	128	R	1	否	无	是
采样器 (s#)	16	R	1	否	无	是
ConstantBuffer 引用 (cb#[index])	15	R	4	是 (内容)	无	是
iImmediate ConstantBuffer 引用 (icb[index])	1	R	4	是 (内容)	无	是

输出寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
NULL (放弃结果，对于具有多个结果的操作非常有用)	不适用	W	不适用	不适用	不适用	否
32 位输出顶点数据元素 (o#)	32	W	4	不适用	不适用	是

相关主题

寄存器 - ps_5_0

项目 • 2022/09/22

以下输入和输出寄存器在像素着色器版本 5_0 中实现。

输入寄存器

注册类型	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位临时 (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入属性 (v#)	32	R	4	是	无	是
输入资源中的元素 (t#)	128	R	1	否	无	是
Sampler (s#)	16	R	1	否	无	是
constantBuffer 引用 (cb#[index])	15	R	4	是 (内容)	无	是
icb[index] (即时 ConstantBuffer 引用)	1	R	4	是 (内容)	无	是

输出寄存器

注册类型	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
NULL (放弃结果，对于具有多个结果的操作非常有用)	空值	W	空值	空值	空值	否

注册类型	计数	R/W	维度	可按 r 编制索引 #	默认值	需要 DCL
32 位输出元素 (o#)	8	W	4	空值	空值	否
无序访问视图 (u#)	8 - 呈现目标 #	R/W	D3D11_PS_CS_UAV_REGISTER_COMPONENTS	否	否	是
32 位 [0.0f..1.0f] (oDepth) 浮点输出深度	1	W	1	不适用	空值	是
32 位 UINT 输出示例掩码 (oMask)	1	W	1	不适用	空值	是

相关主题

着色器模型 5

寄存器 - gs_5_0

项目 • 2022/09/22

以下输入和输出寄存器在几何图形着色器版本 5_0 中实现。

输入寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 Temp (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入 (v[顶点][元素])	32	R	4 (comp) *32 (vert)	是	无	是
32 位输入基元 ID (vPrim)	1	R	1	否	无	是
32 位输入实例 ID (vInstanceID)	1	R	1	否	无	是
输入资源中的元素 (t#)	128	R	1	否	无	是
采样器 (s#)	16	R	1	否	无	是
ConstantBuffer 引用 (cb#[index])	15	R	4	是 (内容)	无	是
(icb[index]) 即时 ConstantBuffer 引用	1	R	4	是 (内容)	无	是

输出寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
NULL (放弃结果，对于使用多个结果) 操作非常有用	不适用	W	不适用	不适用	不适用	否
32 位输出顶点数据元素 (o#)	32	W	不适用	不适用	4	是

相关主题

着色器模型 5

寄存器 - hs_5_0

项目 • 2022/09/22

外壳着色器由三个不同的阶段组成：控制点阶段、分支阶段和联接阶段。 每个阶段都有自己的输入和输出寄存器集。

控制点阶段

hs_control_point_phase是具有以下寄存器模型的着色器程序。

输入寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 Temp (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入 (v[顶点][元素])	32 (元素) *32 ()	R	4	是	无	是
32 位 UINT 输入 vOutputControlPointID (23.7)	1	R	1	否	无	是
32 位 UINT 输入基元 ID (vPrim)	1	R	1	否	不适用	是
输入资源中的元素 (t#)	128	R	128	是	无	是
采样器 (s#)	16	R	1	是	无	是
ConstantBuffer 引用 (cb#[index])	15	R	4	是	无	是
(icb[index]) 即时 ConstantBuffer 引用	1	R	4	是 (内容)	无	是

输出寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位输出顶点数据元素 (o#)	32	W	4	是	无	是

每个外壳着色器控制点阶段输出寄存器最多为 4 向量，最多可以声明 32 个寄存器。 声明的输出控制点也从 1 到 32 个，可缩放所需的存储量。 让我们将所有外壳着色器控制点阶段输出中允许的最大标量聚合数称为#cp_output_max。

#cp_output_max = 3968 标量。

此限制基于此处 4096*32 位存储的某些硬件的设计点。 控制点输出量为 3968=4096-128，即 32 (个控制点) *4 个 (组件) *32 个 (元素) - 4 个 (组件) *32 个 (元素) 。 减法保留 128 标量 (一个控制点) 用于外壳着色器阶段 2 和 3 的空间。 选择为修补常量保留 128 个标量，而不是允许任何 4096 标量存储量都由输出控制点使用-以适应另一个特定硬件设计的限制。 控制点阶段可以声明 32 个输出控制点，但它们只是不能完全 32 个元素，每个元素有 4 个组件，因为总存储会过高。

分叉阶段

以下寄存器在hs_fork_phase模型中可见。

输入资源 (t#) 、采样器 (s#) 、常量缓冲区 (cb#) 和下面的即时常量缓冲区 (icb) 均与其他所有外壳着色器阶段共享状态。 也就是说，从 API/DDI 的角度来看，外壳着色器对所有阶段都有一组输入资源状态。 这与从 API/DDI 的角度来看，外壳着色器是单个原子着色器：其中的各个阶段是实现详细信息。

输入寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 Temp (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入控制点 (vicp[顶点][元素]) (预控制点阶段)	32 请参阅以下说明	R	4 (组件) *32 (元素) *32 ()	是	无	是
32 位输出控制点 (vocp[顶点][元素]]) (帖子控制点阶段)	32 请参阅以下说明	R	4 (组件) *32 (元素) *32 ()	是	无	是
32 位 UINT 输入基元 ID (vPrim)	1	R	1	否	不适用	是

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 UINT 输入 ForkInstanceID (23.8) (vForkInstanceID)	1	R	1	否	不适用	是
输入资源中的元素 (t#)	128	R	128	是	无	是
采样器 (s#)	16	R	1	是	无	是
ConstantBuffer 引用 (cb#[index])	15	R	4	是	无	是
(icb[index]) 即时 ConstantBuffer 引用	1	R	4	是 (内容)	无	是

ⓘ 备注

外壳着色器分叉阶段的输入控制点寄存器 (vicp) 声明必须是外壳着色器控制点输入的任何子集（沿 [element] 轴），(预控制点阶段)。 同样，用于输入输出控制点 (vocp) 的声明必须是外壳着色器输出控制点的任何子集 (帖子控制点)。

沿 [顶点] 轴，要为每个 vicp 和 vocp 读取的控制点数同样必须是外壳着色器输入控制点计数和外壳着色器输出控制点计数的子集。 例如，如果 vocp 寄存器的顶点轴用 n 个顶点声明，则使控制点阶段的输出控制点 [0..n-1] 可用作分叉阶段的只读输入。

输出寄存器

注册	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位输出修补常量数据元素 (o#)	32 请参阅下面的注释 1	W	4	是	无	是

ⓘ 备注

外壳着色器分叉和联接阶段输出是 4 个 4 向量寄存器的共享集。 每个分支或联接阶段程序的输出不能相互重叠。 系统解释的值（如 TessFactors）走出了此空间。

加入阶段

以下寄存器在hs_join_phase模型中可见。 有三组输入寄存器：控制点阶段输入控制点 (vicp) 、vocp 控制点阶段输出控制点 (vocp) ，以及 (vcp) 修补常量。 azure 是所有外壳着色器分叉阶段程序的聚合输出。 外壳着色器联接阶段输出 o# 寄存器与外壳着色器分支阶段输出位于相同的寄存器空间中。

输入资源 (t#) 、采样器 (s#) 、常量缓冲区 (cb#) 和下面的即时常量缓冲区 (icb) 均与其他所有外壳着色器阶段共享状态。 也就是说，从 API/DDI 的角度来看，外壳着色器对所有阶段都有一组输入资源状态。 这与从 API/DDI 的角度来看，外壳着色器是单个原子着色器：其中的各个阶段是实现详细信息。

输入寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 Temp (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入控制点 (vicp[顶点][元素]) (预控制点阶段)	32 请参阅下面的说明 1	R	4 (组件) *32 (元素) *32 ()	是	无	是
32 位输出控制点 (vocp[顶点][元素]) (帖子控制点阶段)	32 请参阅下面的说明 1	R	4 (组件) *32 (元素) *32 ()	是	无	是
32 位输入 (() (修补常量数据)	32 请参阅下面的说明 2	R	4	是	无	是
32 位 UINT 输入基元 ID (vPrim)	1	R	1	否	不适用	是
32 位 UINT 输入 JoinInstanceID (vJoinInstanceID)	1	R	1	否	不适用	是
输入资源中的元素 (t#)	128	R	128	是	无	是
采样器 (s#)	16	R	1	是	无	是
ConstantBuffer 引用 (cb#[index])	15	R	4	是	无	是

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
(icb[index]) 即时 ConstantBuffer 引用	1	R	4	是 (内容)	无	是

注意 1： 外壳着色器联接阶段的输入控制点寄存器 (vicp) 声明必须是任何子集（沿 [element] 轴）的外壳着色器控制点输入 (预控制点阶段)。 同样，用于输入输出控制点 (vocp) 的声明必须是外壳着色器输出控制点的任何子集 (帖子控制点)。

沿 [顶点] 轴，要为每个 vicp 和 vocp 读取的控制点数同样必须是外壳着色器输入控制点计数和外壳着色器输出控制点计数的子集。 例如，如果将 vocp 寄存器的顶点轴声明为 n 个顶点，则使控制点阶段的输出控制点 [0..n-1] 可用作联接阶段的只读输入。

注意 2： 除了控制点输入之外，外壳着色器联接阶段还被视为由外壳着色器分支阶段程序 () 计算的修补常量数据。 这显示在外壳着色器分叉阶段，因为vp# 寄存器。 外壳着色器联接阶段的输入vp# 寄存器共享与外壳着色器分叉阶段输出 o# 寄存器相同的寄存器空间。 o# 寄存器的声明不得与任何外壳着色器分支阶段程序 o# 输出声明重叠;外壳着色器联接阶段正在为外壳着色器添加聚合修补常量数据输出。

输出寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位输出修补常量数据元素 (o#)	32 请参阅以下说明	W	4	是	无	是

> **① 备注**
>
> 外壳着色器分叉和联接阶段输出是 4 个 4 向量寄存器的共享集。 每个分支或联接阶段程序的输出不能相互重叠。 系统解释的值（如 TessFactors）走出了此空间。

相关主题

着色器模型 5

寄存器 - ds_5_0

项目 • 2022/09/24

域着色器版本 5_0 中实现了以下输入和输出寄存器。

输入寄存器

注册类型	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位 Temp (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位输入控制点 (vcp[顶点][元素])	32 请参阅下面的说明 1。	R	4 (组件) *32 (元素) *32 ()	是	无	是
32 位输入修补程序常量 ()	32 请参阅下面的说明 2。	R	4	是	无	是
域 (vDomain.xy， vDomain.xyz)) 中的 32 位输入位置	1	R	3	否	空值	是
32 位 UINT 输入基元 ID (vPrim)	1	R	1	否	空值	是
输入资源中的元素 (t#)	128	R	128	是	无	是
采样器 (s#)	16	R	1	是	无	是
iConstantBuffer 参考 (cb#[index])	15	R	4	是	无	是
iImmediate ConstantBuffer 引用 (icb[index])	1	R	4	是 (内容)	无	是

注意 1： 域着色器在 2 个单独的寄存器集中看到外壳着色器输出。 vcp 寄存器可以看到所有外壳着色器的输出控制点。 网寄存器可以看到所有外壳着色器的修补程序常量输出数据。

注意 2： 由于外壳着色器修补程序常量分支或联接阶段的代码使用名称（如 SV_TessFactor）输出 TessFactors，因此，如果域着色器希望查看这些值，域着色器必须

与等效的vp输入上的声明匹配。

输出寄存器

注册类型	计数	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位输出顶点数据元素 (o#)	32	W	4	是	无	是

相关主题

着色器模型 5

寄存器 - cs_5_0

项目 • 2022/09/23

计算着色器版本 5_0 中实现了以下输入和输出寄存器。

输入寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
32 位临时 (r#)	4096 (r#+x#[n])	R/W	4	否	无	是
32 位可索引临时数组 (x#[n])	4096 (r#+x#[n])	R/W	4	是	无	是
32 位线程组共享内存 (g#[n])	线程组) 所有共享内存的 8192 (总和	R/W	1 (可以通过各种方式声明)	是	无	是
输入资源中的元素 (t#)	128	R	1	否	无	是
Sampler (s#)	16	R	1	否	无	是
constantBuffer 引用 (cb#[index])	15	R	4	是 (内容)	无	是
icb[index] (即时 ConstantBuffer 引用)	1	R	4	是 (内容)	无	是
ThreadID (vThreadID.xyz)	1	R	3	否	空值	是
ThreadGroupID (vThreadGroupID.xyz)	1	R	3	否	空值	是
ThreadIDInGroup (vThreadIDInGroup.xyz)	1	R	3	否	空值	是
ThreadIDInGroupFlattened (vThreadIDInGroupFlattened)	1	R	1	否	空值	是

输出寄存器

注册类型	Count	R/W	维度	可按 r 编制索引#	默认值	需要 DCL
NULL (放弃结果，对于具有多个结果) 操作非常有用	空值	W	空值	空值	空值	否
无序访问视图 (u#)	8	R/W	1	否	否	是

相关主题

着色器模型 5

Shader Model 5.1 (着色器模型 5.1)

项目 • 2022/09/23

本部分包含 HLSL 着色器模型 5.1 (D3D12 引入) 的参考页。

着色器模型 5.1 的功能非常类似于 着色器模型 5，主要更改通过允许从着色器内部索引描述符数组来更灵活地选择资源。

特性	功能
指令集	**HLSL 内部函数**
顶点着色器最大值	无限制
像素着色器最大值	无限制
添加了新的着色器配置文件	cs_5_1、ds_5_1、gs_5_1、hs_5_1、ps_5_1、vs_5_1、rootsig_1_0

在本节中

主题	说明
着色器模型 5.1 对象	已将以下对象添加到着色器模型 5.1。
着色器模型 5.1 系统值	着色器模型 5.1 添加了以下新系统值。

相关主题

着色器模型与着色器配置文件

Direct3D 12 的 HLSL 着色器模型 5.1 功能

着色器模型 5.1 对象

项目 • 2022/09/22

以下对象已添加到着色器模型 5.1。

对于 D3D11.3 和 D3D12) 中提供的 光栅器顺序视图 (，以下对象是新的，并且仅允许在像素着色器中使用。 请注意，它们支持的方法与相应的 UAV 对象相同。

光栅器对象	UAV 对象
RasterizerOrderedBuffer	RWBuffer
RasterizerOrderedByteAddressBuffer	RWByteAddressBuffer
RasterizerOrderedStructuredBuffer	RWStructuredBuffer
RasterizerOrderedTexture1D	RWTexture1D
RasterizerOrderedTexture1DArray	RWTexture1DArray
RasterizerOrderedTexture2D	RWTexture2D
RasterizerOrderedTexture2DArray	RWTexture2DArray
RasterizerOrderedTexture3D	RWTexture3D

相关主题

Shader Model 5.1（着色器模型 5.1）

Direct3D 12 的 HLSL 着色器模型 5.1 功能

着色器模型 5.1 系统值

项目 • 2022/09/22

着色器模型 5.1 添加了以下新系统值。

- SV_InnerCoverage
- SV_StencilRef

相关主题

Shader Model 5.1（着色器模型 5.1）

Direct3D 12 的 HLSL 着色器模型 5.1 功能

SV_InnerCoverage

项目 • 2022/09/24

SV_InnerCoverage表示被低估的保守光栅化信息（，即是否保证像素完全覆盖）。

类型

类型
uint

注解

此系统生成的值对于第 3 层支持是必需的，并且仅在该层中可用。 此输入与 SV_Coverage互斥 - 两者都不能使用。

若要访问SV_InnerCoverage，必须将它声明为像素着色器输入寄存器之一中的单个组件。 声明中的内插模式必须是常量（内插不适用）。

保守光栅化在 D3D11.3 和 D3D12 中都可用。 请参阅：

- D3D11.3 保守光栅化
- D3D12 保守光栅化

另请参阅

着色器模型 5

着色器模型 5.1 系统值

SV_StencilRef

项目 • 2022/09/22

SV_StencilRef表示当前像素着色器模具引用值。

类型

类型
uint

注解

在像素着色器中指定着色器引用值在 D3D11.3 和 D3D12 中都可用。 请参阅：

- D3D11.3 着色器指定的模具引用值
- D3D12 着色器指定的模具引用值

另请参阅

着色器模型 5

着色器模型 5.1 系统值

着色器模型 6

项目 • 2022/09/23

所有非象限相关的波形内部函数在所有着色器阶段都可用。 象波内部函数仅在像素和计算着色器中可用。

在本节中

主题	说明
QuadReadAcrossDiagonal	返回从此象限中对角相反的通道读取的指定本地值。
QuadReadLaneAt	从当前象限中的通道 ID 标识的通道中返回指定的源值。
QuadReadAcrossX	返回从 X 方向上的另一个通道中读取的指定本地值。
QuadReadAcrossY	返回从 Y 方向的此象限中的其他通道中读取的指定源值。
WaveActiveAllEqual	如果当前波中每个活动通道的表达式 (且因此) 统一，则返回 true。
WaveActiveBitAnd	返回当前波中所有活动通道中表达式的所有值的按位 AND，并将其复制回所有活动通道。
WaveActiveBitOr	返回当前波中所有活动通道中表达式的所有值的按位 OR，并将其复制回所有活动通道。
WaveActiveBitXor	返回当前波中所有活动通道中表达式的所有值的按位 XOR，并将其复制回所有活动通道。
WaveActiveCountBits	计算在当前波中所有活动通道中计算结果为 true 的布尔变量数，并将结果复制到波中的所有通道。
WaveActiveMax	返回当前波中所有活动通道的表达式的最大值，并将其复制回所有活动通道。
WaveActiveMin	返回当前波中所有活动通道中的表达式的最小值，将其复制回所有活动通道。
WaveActiveProduct	将表达式的值乘以当前波中的所有活动通道，并将其复制回所有活动通道。
WaveActiveSum	对当前波中所有活动通道的表达式值求和，并将其复制到当前波中的所有通道。
WaveActiveAllTrue	如果表达式在当前波的所有活动通道中为 true，则返回 true。
WaveActiveAnyTrue	如果表达式在当前波中的任何活动通道中为 true，则返回 true。

主题	说明
WaveActiveBallot	返回指定波中所有活动通道的布尔表达式计算的 4 位无符号整数位掩码。
WaveGetLaneCount	返回此体系结构上的波形中的通道数。
WaveGetLaneIndex	返回当前波中当前通道的索引。
WaveIsFirstLane	仅返回当前波中具有最小索引的活动通道的 true。
WavePrefixCountBits	返回所有指定布尔变量的总和，这些变量在所有活动通道中设置为 true，其索引小于当前通道。
WavePrefixProduct	返回此波中活动通道中所有值乘积，索引小于此通道。
WavePrefixSum	返回活动通道中所有值的总和，其索引比此值小。
WaveReadLaneFirst	返回具有最小索引的当前波的活动通道的表达式值。
WaveReadLaneAt	返回指定波中给定通道索引的表达式值。

相关主题

着色器模型 6 概述

着色器模型与着色器配置文件

QuadReadAcrossDiagonal 函数

项目 • 2022/09/24

返回从此象限中对角相反的通道读取的指定本地值。

语法

syntax

```
<type> QuadReadAcrossDiagonal(
   <type> localValue
);
```

参数

localValue

请求的类型。

返回值

从此象限中对角相反的通道读取的指定本地值。

备注

有关象限的详细信息，请参阅 着色器模型 6 概述。

仅像素着色器模型 6.0 和计算着色器支持此函数。

另请参阅

着色器模型 6

QuadReadLaneAt 函数

项目 • 2022/09/23

从由当前象限中的通道 ID 标识的通道中返回指定的源值。

语法

syntax

```
<type> QuadReadLaneAt(
   <type> sourceValue,
   uint   quadLaneID
);
```

参数

sourceValue

请求的类型。

quadLaneID

通道 ID;此值将从 0 到 3。

返回值

指定的源值。 此函数的结果在整个象限中是统一的。 如果源通道处于非活动状态，则结果未定义。

注解

有关象限的详细信息，请参阅 着色器模型 6 概述。

仅在像素着色器和计算着色器中，着色器模型 6.0 才支持此函数。

另请参阅

着色器模型 6

QuadReadAcrossX 函数

项目 • 2022/09/24

返回从 X 方向上的另一个通道中读取的指定本地值。

语法

syntax

```
<type> QuadReadAcrossX(
   <type> localValue
);
```

parameters

localValue

请求的类型。

返回值

指定的本地值。 如果源通道处于非活动状态，则结果未定义。

注解

有关象限的详细信息，请参阅 着色器模型 6 概述。

仅像素着色器模型 6.0 和计算着色器支持此函数。

另请参阅

着色器模型 6

QuadReadAcrossY 函数

项目 • 2022/09/24

返回从 Y 方向的此象限中的其他通道中读取的指定源值。

语法

syntax

```
<type> QuadReadAcrossY(
   <type> localValue
);
```

parameters

localValue

请求的类型。

返回值

指定的源值。 如果源通道处于非活动状态，则结果未定义。

注解

有关象限的详细信息，请参阅 着色器模型 6 概述。

仅像素着色器模型 6.0 和计算着色器支持此函数。

另请参阅

着色器模型 6

WaveActiveAllEqual 函数

项目 • 2022/09/24

如果当前波中每个活动通道的表达式 (且因此) 统一，则返回 true。

语法

syntax

```
bool WaveActiveAllEqual(
   <type> expr
);
```

参数

expr

要计算的表达式。

返回值

如此 如果当前波中每个活动通道的表达式都相同。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveBitAnd 函数

项目 • 2022/09/24

返回当前波中所有活动通道中表达式的所有值的按位 AND，并将其复制回所有活动通道。

语法

syntax

```
<int_type> WaveActiveBitAnd(
   <int_type> expr
);
```

参数

expr

要计算的表达式。

返回值

按位 AND 值。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveBitOr 函数

项目 • 2022/09/22

返回当前波中所有活动通道中表达式的所有值的按位 OR，并将其复制回所有活动通道。

语法

syntax

```
<int_type> WaveActiveBitOr(
   <int_type> expr
);
```

参数

expr

要计算的表达式。

返回值

按位 OR 值。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveBitXor 函数

项目 • 2022/09/24

返回当前波中所有活动通道中表达式的所有值的按位 XOR，并将其复制回所有活动通道。

语法

syntax

```
<int_type> WaveActiveBitXor(
   <int_type> expr
);
```

参数

expr

要计算的表达式。

返回值

按位 XOR 值。

注解

在所有着色器阶段中，着色器模型 6.0 支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveCountBits 函数

项目 • 2022/09/23

计算在当前波中所有活动通道中计算结果为 true 的布尔变量数，并将结果复制到波中的所有通道。

语法

syntax

```
uint WaveActiveCountBits(
   bool bBit
);
```

parameters

bBit

要计算的布尔变量。 提供显式真正的布尔值将返回活动通道数。

返回值

布尔变量计算结果为 true 的通道数，在当前波中的所有活动通道中。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

这比完整的 WaveActiveSum 更高效地实现，如以下示例中所述：

syntax

```
result = WaveActiveCountBits( WaveActiveBallot( bBit ) );
```

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveMax 函数

项目 • 2022/09/23

返回当前波中所有活动通道的表达式的最大值，并将其复制回所有活动通道。

语法

syntax

```
<type> WaveActiveMax(
   <type> expr
);
```

参数

expr

要计算的表达式。

返回值

最大值。

注解

操作顺序未定义。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

syntax

```
 float3 maxPos = WaveActiveMax( myPoint.position );
    BoundingBox.max = max( maxPos, BoundingBox.max );
```

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveMin 函数

项目 • 2022/09/22

返回当前波中所有活动通道中的表达式的最小值，将其复制回所有活动通道。

语法

syntax

```
<type> WaveActiveMin(
   <type> expr
);
```

参数

expr

要计算的表达式。

返回值

最小值。

注解

操作顺序未定义。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

syntax

```
 float3 minPos = WaveActiveMin( myPoint.position );
    BoundingBox.min = min( minPos, BoundingBox.min );
```

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveProduct 函数

项目 • 2022/09/24

将表达式的值乘以当前波中的所有活动通道，并将其复制回所有活动通道。

语法

syntax

```
<type> WaveActiveProduct(
   <type> expr
);
```

参数

expr

要计算的表达式。

返回值

产品值。

注解

操作顺序未定义。

在所有着色器阶段中，着色器模型 6.0 支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveSum 函数

项目 • 2022/09/24

对当前波中所有活动通道的表达式值求和，并将其复制到当前波中的所有通道。

语法

```
syntax
```

```
<type> WaveActiveSum(
   <type> expr
);
```

参数

expr

要计算的表达式。

返回值

总和值。

注解

操作顺序未定义。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

```
syntax
```

```
float3 total = WaveActiveSum( position ); // sum positions in wave
float3 center = total/count;           // compute average of these positions
```

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveAllTrue 函数

项目 • 2022/09/22

如果表达式在当前波的所有活动通道中为 true，则返回 true。

语法

syntax

```
bool WaveActiveAllTrue(
   bool expr
);
```

参数

expr

要计算的布尔表达式。

返回值

如果表达式在所有通道中为 true，则为 True。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveAnyTrue 函数

项目 • 2022/09/22

如果表达式在当前波中的任何活动通道中为 true，则返回 true。

语法

syntax

```
bool WaveActiveAnyTrue(
   bool expr
);
```

parameters

expr

要计算的布尔表达式。

返回值

如果表达式在任何通道中为 true，则为 True。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveActiveBallot 函数

项目 • 2022/09/24

返回一个 uint4，其中包含当前波中所有活动通道的布尔表达式计算的位掩码。

语法

syntax

```
uint4 WaveBallot(
   bool expr
);
```

参数

expr

要计算的布尔表达式。

返回值

一个 uint4，包含当前波中所有活动通道的布尔表达式计算的位掩码。 最小有效位对应于索引为零的通道。 对应于非活动通道的位将为零。 大于或等于 WaveGetLaneCount 的位将为零。

注解

不同的 GPU 具有不同的 SIMD 处理器宽度，（通道计数）。 其中大多数 **WaveXXX** 函数都可以在隐藏 SIMD 机器宽度的抽象级别运行。 若要最大化跨 GPU 的代码可移植性，请使用不依赖于计算机宽度的内部函数。 例如，使用：

syntax

```
uint result = WaveActiveCountBits( bBit );
```

不是：

syntax

```
uint result = countbits( WaveActiveBallot( bBit ) );
```

在所有着色器阶段中，着色器模型 6.0 支持此函数。

示例

syntax

```
// get a bitwise representation of the number of currently active lanes:
uint4 waveBits = WaveActiveBallot( true ); // convert to bits
```

另请参阅

着色器模型 6 概述

着色器模型 6

WaveGetLaneCount 函数

项目 • 2022/09/22

返回此体系结构上的波形中的通道数。

语法

```
syntax
```

```
uint WaveGetLaneCount(void);
```

parameters

此函数没有参数。

返回值

结果介于 4 和 128 之间，包括所有波形：活动、非活动通道和/或帮助程序通道。 此函数返回的结果可能因驱动程序实现而异。

注解

在所有着色器阶段中，着色器模型 6.0 支持此函数。

示例

```
syntax
```

```
 uint laneCount = WaveGetLaneCount();    // number of lanes in wave
```

另请参阅

着色器模型 6 概述

着色器模型 6

WaveGetLaneIndex 函数

项目 • 2022/09/23

返回当前波中当前通道的索引。

语法

```
syntax

uint WaveGetLaneIndex(void);
```

parameters

此函数没有参数。

返回值

当前通道索引。 结果介于 0 和 从 WaveGetLaneCount 返回的结果之间。

注解

在所有着色器阶段中，着色器模型 6.0 支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveIsFirstLane 函数

项目 • 2022/09/23

仅返回当前波中具有最小索引的活动通道的 true。

语法

syntax

```
bool WaveIsFirstLane(void);
```

parameters

此函数没有参数。

返回值

True 仅适用于当前波中具有最小索引的活动通道。

注解

此函数可用于标识每波只执行一次的操作。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

syntax

```
 if ( WaveIsFirstLane() )
{
    . . . // once per-wave code
}
```

另请参阅

着色器模型 6 概述

着色器模型 6

WavePrefixCountBits 函数

项目 • 2022/09/24

返回所有指定布尔变量的总和，这些变量在所有活动通道中设置为 true，其索引小于当前通道。

语法

syntax

```
uint WavePrefixCountBits(
   bool bBit
);
```

参数

bBit

指定的布尔变量。

返回值

所有指定的布尔变量的总和在所有活动通道中设置为 true，索引小于当前通道。

注解

在所有着色器阶段中，着色器模型 6.0 支持此函数。

示例

以下代码介绍如何实现对有序流的压缩写入，其中每个通道写入的元素数为 1 或 0。

syntax

```
bool bDoesThisLaneHaveAnAppendItem = <expr>;
// compute number of items to append for the whole wave
uint laneAppendOffset = WavePrefixCountBits( bDoesThisLaneHaveAnAppendItem
);
uint appendCount = WaveActiveCountBits( bDoesThisLaneHaveAnAppendItem);
```

```
// update the output location for this whole wave
uint appendOffset;
if ( WaveIsFirstLane () )
{
    // this way, we only issue one atomic for the entire wave, which reduces
contention
    // and keeps the output data for each lane in this wave together in the
output buffer
    InterlockedAdd(bufferSize, appendCount, appendOffset);
}
appendOffset = WaveReadLaneFirst( appendOffset ); // broadcast value
appendOffset += laneAppendOffset; // and add in the offset for this lane
buffer[appendOffset] = myData; // write to the offset location for this lane
```

另请参阅

着色器模型 6 概述

着色器模型 6

WavePrefixProduct 函数

项目 • 2022/09/23

返回此波中活动通道中所有值乘积，索引小于此通道。

语法

syntax

```
<type> WavePrefixProduct(
   <type> value
);
```

参数

value

要相乘的值。

返回值

所有值的乘积。

注解

无法保证此例程上的操作顺序。 因此，实际上，在它内忽略 [precise] 标志。

可以通过将前缀产品乘以当前通道的值来计算后缀产品。

请注意，具有最低索引的活动通道将始终为其前缀产品接收 1。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

hlsl

```
uint numToMultiply = 2;
uint prefixProduct = WavePrefixProduct( numToMultiply );
```

在波形大小为 8 的计算机上，除通道 0 和 4 之外的所有通道都处于活动状态，以下值将从 WavePrefixProduct 返回。

lane 索引	状态	prefixProduct
0	非活跃	不适用
1	活动	= 1
2	活动	= 1*2
3	活动	= 1*2*2
4	非活跃	不适用
5	活动	= 1*2*2*2
6	活动	= 1*2*2*2*2*2
7	活动	= 1*2*2*2*2*2*2

另请参阅

着色器模型 6 概述

着色器模型 6

WavePrefixSum 函数

项目 • 2022/09/22

返回活动通道中所有值的总和，其索引比此值小。

语法

syntax

```
<type> WavePrefixSum(
   <type> value
);
```

参数

value

要求和的值。

返回值

值之和。

注解

无法保证此例程上的操作顺序。 因此，实际上，在它内忽略 [precise] 标志。

通过将前缀和添加到当前通道的值，可以计算后缀和。

请注意，具有最低索引的活动通道将始终为其前缀总和接收 0。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

示例

hlsl

```
uint numToSum = 2;
uint prefixSum = WavePrefixSum( numToSum );
```

在波形大小为 8 的计算机上，除通道 0 和 4 之外的所有通道都处于活动状态，以下值将从 WavePrefixSum 返回。

lane 索引	status	prefixSum
0	非活跃	不适用
1	活动	= 0
2	活动	= 0+2
3	活动	= 0+2+2
4	非活跃	不适用
5	活动	= 0+2+2+2
6	活动	= 0+2+2+2+2+2
7	活动	= 0+2+2+2+2+2+2

另请参阅

着色器模型 6 概述

着色器模型 6

WaveReadLaneFirst 函数

项目 • 2022/09/24

返回具有最小索引的当前波的活动通道的表达式值。

语法

syntax

```
<type> WaveReadLaneFirst(
   <type> expr
);
```

参数

expr

要计算的表达式。

返回值

生成的值在波中是统一的。

注解

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

着色器模型 6 概述

着色器模型 6

WaveReadLaneAt 函数

项目 • 2022/09/22

返回指定波中给定通道索引的表达式值。

语法

syntax

```
<type> WaveReadLaneAt(
   <type> expr,
   uint laneIndex
);
```

参数

expr

要计算的表达式。

laneIndex

将为其返回 *expr* 结果的通道的索引。

返回值

生成的值为 *expr* 的结果。 如果 *laneIndex* 是统一的，它将是统一的。

注解

此函数实际上是 *laneIndex'th* 通道中值的广播。

在所有着色器阶段中，着色器模型 6.0 都支持此函数。

另请参阅

- 着色器模型 6 概述
- 着色器模型 6

内部函数

项目 • 2022/09/22

下表列出了 HLSL 中可用的内部函数。 每个函数都有简短说明，以及指向引用页的链接，其中包含有关输入参数和返回类型的更多详细信息。

名称	说明	最小着色器模型
中止	终止正在执行的当前绘图或调度调用。	4
Abs	每个组件) 的绝对值 (。	1M
acos	返回 x 的每个组件的反正弦值。	1M
所有	测试 x 的所有组件是否为非零。	1M
AllMemoryBarrier	阻止在完成所有内存访问之前执行组中的所有线程。	5
AllMemoryBarrierWithGroupSync	阻止执行组中的所有线程，直到所有内存访问都已完成，组中的所有线程都已到达此调用。	5
任何	测试 x 的任何组件是否为非零。	1M
asdouble	将强制转换值重新解释为双精度值。	5
asfloat	将输入类型转换为浮点数。	4
asin	返回 x 的每个组件的反正弦值。	1M
asint	将输入类型转换为整数。	4
asuint	将 64 位类型的位模式重新解释为 uint。	5
asuint	将输入类型转换为无符号整数。	4
atan	返回 x 的反正切值。	1M
atan2	返回 (x，y) 两个值的反正切值。	1M
ceil	返回大于或等于 x 的最小整数。	1M
CheckAccessFullyMapped	确定是否在平铺资源中访问**了示例**或**加载**操作中的所有值都访问了映射磁贴。	5
钳	将 x 固定到范围 [min， max]。	1M
剪辑	如果 x 的任何组件小于零，则放弃当前像素。	1M

名称	说明	最小着色器模型
因为	返回 x 的余弦值。	1M
cosh	返回 x 的双曲余弦值。	1M
countbits	计算输入整数中每个组件) (位数。	5
cross	返回两个三维向量的交叉乘积。	1M
D3DCOLORtoUBYTE4	重排和缩放 4D 向量 xto 的组件可补偿某些硬件中缺少 UBYTE4 支持。	1M
ddx	返回 x 相对于屏幕空间 x 坐标的部分派生。	2M
ddx_coarse	计算与屏幕空间 x 坐标相关的低精度部分派生。	5
ddx_fine	计算与屏幕空间 x 坐标相关的高精度分部派生。	5
ddy	返回 x 相对于屏幕空间 y 坐标的部分派生。	2M
ddy_coarse	计算与屏幕空间 y 坐标相关的低精度部分派生。	5
ddy_fine	计算与屏幕空间 y 坐标相关的高精度分部派生。	5
度	将 x 从弧度转换为度。	1M
行列 式	返回平方矩阵 m 的确定性。	1M
DeviceMemoryBarrier	阻止执行组中的所有线程，直到所有设备内存访问都已完成。	5
DeviceMemoryBarrierWithGroupSync	阻止执行组中的所有线程，直到完成所有设备内存访问，并且该组中的所有线程都已达到此调用。	5
distance	返回两个点之间的距离。	1M
点	返回两个向量的点积。	1
Dst	计算距离向量。	5
errorf	将错误消息提交到信息队列。	4
EvaluateAttributeCentroid	以像素质心计算。	5
EvaluateAttributeAtSample	在已编制索引的示例位置求值。	5

名称	说明	最小着色器模型
EvaluateAttributeSnapped	以带偏移量的像素质心求值。	5
exp	返回 base-e 指数。	1M
exp2	每个组件的基数 2 指数 () 。	1M
f16tof32	将存储在 uint 低半部分的 float16 转换为浮点。	5
f32tof16	将输入转换为 float16 类型。	5
faceforward	返回 -n * 符号 (点 (i， ng)) 。	1M
firstbithigh	获取从最高顺序位开始的第一个集位的位置，并按组件向下工作。	5
firstbitlow	返回从最低顺序位开始的第一个集位的位置，并按组件向上工作。	5
地板	返回小于或等于 x 的最大整数。	1M
fma	返回 * b + c 的双精度融合加法。	5
fmod	返回 x/y 的浮点余数。	1M
压裂	返回 x 的小数部分。	1M
frexp	返回 x 的 mantissa 和指数。	2M
fwidth	返回 abs (ddx (x)) + abs (ddy (x))	2M
GetRenderTargetSampleCount	返回呈现目标样本的数目。	4
GetRenderTargetSamplePosition	返回给定样本索引 (x，y) 的示例位置。	4
GroupMemoryBarrier	阻止执行组中的所有线程，直到所有组共享访问都已完成。	5
GroupMemoryBarrierWithGroupSync	阻止执行组中的所有线程，直到所有组共享访问都已完成，组中的所有线程都已达到此调用。	5
InterlockedAdd	对 dest 资源变量执行有保证的原子值添加。	5
InterlockedAnd	执行有保证的原子和。	5
InterlockedCompareExchange	以原子方式将输入与比较值进行比较，并交换结果。	5
InterlockedCompareStore	以原子方式将输入与比较值进行比较。	5

名称	说明	最小着色器模型
InterlockedExchange	为 dest 赋值并返回原始值。	5
InterlockedMax	执行有保证的原子最大值。	5
InterlockedMin	执行有保证的原子最小值。	5
InterlockedOr	执行有保证的原子或。	5
InterlockedXor	执行有保证的原子 xor。	5
isfinite	如果 x 是有限值，则返回 true;否则返回 false。	1M
isinf	如果 x 为 +INF 或 -INF，则返回 true;否则返回 false。	1M
isnan	如果 x 为 NAN 或 QNAN，则返回 true，否则返回 false。	1M
ldexp	返回 x * 2exp	1M
长度	返回向量 v 的长度。	1M
lerp	返回 x + s (y - x) 。	1M
点燃	返回照明矢量 (环境、漫射、反射、1)	1M
日志	返回 x 的 base-e 对数。	1M
log10	返回 x 的 base-10 对数。	1M
log2	返回 x 的 base-2 对数。	1M
疯狂	对三个值执行算术乘法/加运算。	5
max	选择 x 和 y 的更大项。	1M
分钟	选择 x 和 y 的较小值。	1M
modf	将值 x 拆分为小数部分和整数部分。	1M
msad4	比较 4 字节引用值和 8 字节源值，并累积 4 个总和的向量。	5
mul	使用 x 和 y 执行矩阵乘法。	1
噪声	使用 Perlin-noise 算法生成随机值。	1M
规范化	返回规范化向量。	1M

名称	说明	最小着色器模型
战俘	返回 x^y。	1M
printf	将自定义着色器消息提交到信息队列。	4
Process2DQuadTessFactorsAvg	为象限补丁生成更正的细化因子。	5
Process2DQuadTessFactorsMax	为象限补丁生成更正的细化因子。	5
Process2DQuadTessFactorsMin	为象限补丁生成更正的细化因子。	5
ProcessIsolineTessFactors	为 isoline 生成舍入细化因子。	5
ProcessQuadTessFactorsAvg	为象限补丁生成更正的细化因子。	5
ProcessQuadTessFactorsMax	为象限补丁生成更正的细化因子。	5
ProcessQuadTessFactorsMin	为象限补丁生成更正的细化因子。	5
ProcessTriTessFactorsAvg	为三个修补程序生成更正的细化因子。	5
ProcessTriTessFactorsMax	为三个修补程序生成更正的细化因子。	5
ProcessTriTessFactorsMin	为三个修补程序生成更正的细化因子。	5
弧度	将 x 从度转换为弧度。	1
rcp	计算一个快速、近似、每分量对等。	5
reflect	返回反射向量。	1
折射	返回折射向量。	1 次
reversebits	按组件反转位的顺序。	5
轮	将 x 舍入为最接近的整数	1 次
rsqrt	返回 1 / sqrt (x)	1 次
饱和	将 x 固定到范围 [0 ， 1]	1
标志	计算 x 的符号。	1 次
罪	返回 x 的正弦值	1 次
sincos	返回 x 的正弦和余弦值。	1 次
sinh	返回 x 的双曲正弦值	1 次
smoothstep	返回介于 0 和 1 之间的平滑 Hermite 内插。	1 次

名称	说明	最小着色器模型
sqrt	每个组件的平方根 ()	1 次
步	返回 (x >=) ？ 1 : 0	1 次
潭	返回 x 的正切值	1 次
tanh	返回 x 的双曲正切值	1 次
tex1D (s、t)	1D 纹理查找。	1
tex1D (s、t、ddx、ddy)	1D 纹理查找。	2[1]
tex1Dbias	带偏差的 1D 纹理查找。	2[1]
tex1Dgrad	具有渐变的 1D 纹理查找。	2[1]
tex1Dlod	具有 LOD 的 1D 纹理查找。	3M
tex1Dproj	带投影除法的 1D 纹理查找。	2[1]
tex2D (s、t)	2D 纹理查找。	1 次
tex2D (s、t、ddx、ddy)	2D 纹理查找。	2[1]
tex2Dbias	带偏差的 2D 纹理查找。	2[1]
tex2Dgrad	具有渐变的 2D 纹理查找。	2[1]
tex2Dlod	具有 LOD 的 2D 纹理查找。	3
tex2Dproj	带投影除法的二维纹理查找。	2[1]
tex3D (s、t)	三维纹理查找。	1 次
tex3D (s、t、ddx、ddy)	三维纹理查找。	2[1]
tex3Dbias	具有偏差的 3D 纹理查找。	2[1]
tex3Dgrad	具有渐变的 3D 纹理查找。	2M
tex3Dlod	使用 LOD 进行 3D 纹理查找。	3M
tex3Dproj	具有投影除法的 3D 纹理查找。	2M
texCUBE (s、 t)	多维数据集纹理查找。	1M
texCUBE (s、t、ddx、ddy)	多维数据集纹理查找。	2M
texCUBEbias	带偏差的多维数据集纹理查找。	2M

名称	说明	最小着色器模型
texCUBEgrad	具有渐变的多维数据集纹理查找。	2M
texCUBElod	使用 LOD 进行多维数据集纹理查找。	3M
texCUBEproj	具有投影除法的多维数据集纹理查找。	2M
transpose	返回矩阵 m 的转置。	1
trunc	截断浮点值 () 整数值 (s)	1

M 请参阅参考页了解限制。

组件和模板类型

HLSL 内部函数声明使用组件类型和模板类型作为输入参数参数和返回值。 下表列出了可用的类型。

这些模板类型	说明	支持这些数据类型
矩形图	最多 16 个组件，具体取决于声明	基本 HLSL 类型
object	sampler 对象	*sampler*、 *sampler1D*、 *sampler2D*、 *sampler3D*、 *samplerCUBE*
标量 (scalar)	1 个组件	基本 HLSL 类型
向量	最小 1 个组件，最多 4 个组件 ((含))	基本 HLSL 类型

请参阅

HLSL 参考

abort 函数

项目 • 2022/09/22

将错误消息提交到信息队列，并终止正在执行的当前绘图或调度调用。

语法

```
syntax

void abort(

);
```

参数

无

返回值

此函数不返回值。

注解

此操作对不支持光栅器的光栅器不执行任何操作。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 (DirectX HLSL) 或更高版本。	是

要求

要求	Value
标头	Corecrt_terminate.h

另请参阅

内部函数

abs

项目 • 2022/09/24

返回指定值的绝对值。

ret abs (*x*)

参数

项	说明
X	[in]指定的值。

返回值

x 参数的绝对值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float、 int	any
Ret	与输入 *x* 相同	与输入 *x* 相同	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL (内部函数)

acos

项目 • 2022/09/22

返回指定值的反余弦值。

***ret* acos (*x*)**

参数

项	说明
X	[in]指定的值。 每个组件应为 -1 到 1 范围内的浮点值。

返回值

x 参数的反余弦值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	仅vs_1_1

另请参阅

DirectX HLSL) 内部函数 (

all

项目 • 2022/09/23

确定指定值的所有组件是否为非零。

***重试* 所有 (*x*)**

参数

项目	说明
X	[in]指定的值。

返回值

如果 *x* 参数的所有组件均为非零，则为 **True**;否则为 **false**。

备注

此函数类似于 **任何** HLSL 内部函数。 **任何**函数确定指定值的任何组件是否为非零，而**所有**函数确定指定值的所有组件是否为非零。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	**float**、 **int**、 **bool**	any
Ret	**标量**	**bool**	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL) 内部函数 (

AllMemoryBarrier 函数

项目 • 2022/09/22

阻止执行组中的所有线程，直到所有内存访问都已完成。

语法

```
syntax

void AllMemoryBarrier(void);
```

parameters

此函数没有参数。

返回值

此函数不返回值。

注解

内存屏障可确保未完成的内存操作。 线程在 GroupSync 屏障处同步。 如果内存操作正在进行，这可能会停止线程或线程。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
					x

另请参阅

内部函数

着色器模型 5

AllMemoryBarrierWithGroupSync 函数

项目 • 2022/09/24

阻止执行组中的所有线程，直到所有内存访问都已完成，组中的所有线程都已到达此调用。

语法

```
syntax

void AllMemoryBarrierWithGroupSync(void);
```

parameters

此函数没有参数。

返回值

此函数不返回值。

注解

内存屏障可保证未完成的内存操作。 线程在 GroupSync 屏障处同步。 如果内存操作正在进行，这可能会停止线程或线程。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算

顶点	外壳	域	几何结构	像素	计算
					x

另请参阅

内部函数

着色器模型 5

any

项目 • 2022/09/23

确定指定值的任何组件是否为非零。

***重试* 任何 (*x*)**

参数

项目	说明
X	[in]指定的值。

返回值

如果 *x* 参数的任何组件为非零，则为 **True**;否则为 **false**。

注解

此函数类似于 **所有** HLSL 内部函数。 **任何**函数确定指定值的任何组件是否为非零，而**所有**函数确定指定值的所有组件是否为非零。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	**float**、 **int**、 **bool**	any
Ret	**标量**	**bool**	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL) 内部函数 (

asdouble 函数

项目 • 2022/09/22

将转换值重新解释为双精度值 (两个 32 位值) 。

语法

```
syntax

double asdouble(
  in uint lowbits,
  in uint highbits
);
```

参数

lowbits [in]

类型： **uint**

输入值的低 32 位模式。

highbits [in]

类型： **uint**

输入值的高 32 位模式。

返回值

类型： **double**

输入 (两个 32 位值，) 重播为双精度值。

注解

还可以使用以下重载版本：

```
syntax

double2 asdouble(uint2 lowbits, uint2 highbits);
```

如果输入值为两个 32 位组件，则返回类型将包含一个双精度值。 如果输入值为 4 个 32 位组件，则返回类型将包含两个双精度值。 如果输入值为 64 位类型，则返回的值将具有与输入值相同的组件数。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

asfloat

项目 • 2022/09/24

将 *x* 的位模式解释为浮点数。

ret asfloat (*x*)

参数

项	描述
X	[in]输入值。

返回值

解释为浮点数的输入。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float、 int、 uint	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

函数重载

“floatx< asfloat (floatx value) ;” 'floatx><>> asfloat (intx<> value) ;' 'floatx<>< asfloat (uintx<> value) ;'

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 和更高着色器模型	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

注解

旧编译器被错误地允许 `asfloat(bool)`，但请注意，不支持布尔输入。

另请参阅

DirectX HLSL) 内部函数 (

asin

项目 • 2022/09/22

返回指定值的反正弦值。

***ret* asin (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的反正弦值。

注解

x 参数的每个组件都应在 -π/2 到 π/2 的范围内。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

asint 函数

项目 • 2022/09/22

将输入值的位模式解释为整数。 有关 **asint** 内部函数的详细信息，请参阅 asint (DirectX HLSL) 。

语法

syntax

```
int asint(
  in  value
);
```

parameters

value [in]

输入值。

返回值

类型： **int**

解释为整数的输入。

另请参阅

DirectX HLSL) 内部函数 (

asint

项目 • 2022/09/22

将 *x* 的位模式解释为整数。

ret asint (*x*)

参数

项	说明
X	[in]输入值。

返回值

解释为整数的输入。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float、 uint	any
Ret	与输入 *x* 相同	Int	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 和更高着色器模型	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

asuint 函数

项目 • 2022/09/23

将 64 位值的位模式重新解释为两个无符号 32 位整数。

语法

syntax

```
void asuint(
  in  double value,
  out uint lowbits,
  out uint highbits
);
```

参数

值 [in]

类型： **double**

输入值。

lowbits [out]

类型： **uint**

*值的*低 32 位模式。

highbits [out]

类型： **uint**

高 32 位 *值*模式。

返回值

此函数不返回值。

注解

此函数是早期着色器模型中提供的 asuint 内部函数的备用版本，已针对着色器模型 5 引入。 HLSL 编译器中通过不同的签名 () 识别的原始函数仍可供着色器模型 5 使用。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

asuint (DirectX HLSL)

着色器模型 5

asuint

项目 • 2022/09/22

将 *x* 的位模式解释为无符号整数。

ret asuint (*x*)

参数

项	说明
X	[in]输入值。

返回值

解释为无符号整数的输入。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	**float**、 **int**	any
Ret	与输入 *x* 相同	**uint**	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 和更高着色器模型	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

atan

项目 • 2022/09/24

返回指定值的反正切值。

***重试* atan (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的反正切值。 此值在 -π/2 到 π/2 的范围内。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

atan2

项目 • 2022/09/23

返回两个值 (x，y) 的反正切值。

***ret* atan2 (*y*， *x*)**

参数

项	说明
Y	[in]y 值。
X	[in]x 值。

返回值

(y，x) 的反正切值。

备注

x 和 *y* 参数的符号用于确定 -π 到π范围内的返回值的象限。 **atan2** HLSL 内部函数对于除原点以外的每个点都定义得很好，即使 *y* 等于 0 且 *x* 不等于 0。

类型说明

名称	模板类型	组件类型	大小
y	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

要求

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

ceil

项目 • 2022/09/22

返回大于或等于指定值的最小整数值。

***ret* ceil (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

作为浮点类型返回的最小整数值 (大于或等于 *x* 参数的浮点) 类型。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

CheckAccessFullyMapped 函数

项目 • 2022/09/22

确定是否在平铺资源中访问**了示例**、**收集**或**加载**操作中的所有值。

语法

syntax

```
bool CheckAccessFullyMapped(
  in uint_only status
);
```

参数

status [in]

类型： **uint_only**

从 **示例**、 **收集**或 **加载** 操作返回的状态值。 由于无法直接访问此状态值，因此需要将其传递给 **CheckAccessFullyMapped**。

返回值

类型： **bool**

返回一个**布尔**值，该值指示是否在平铺资源中访问**了示例**、**收集**或**加载**操作中的所有值。 如果操作中的所有值都访问映射磁贴，则返回 **TRUE** ;否则，如果从未映射的磁贴中获取任何值，则返回 **FALSE** 。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	x

另请参阅

内部函数

钳

项目 • 2022/09/22

将指定的值固定到指定的最小值和最大值范围。

***ret* 固定 (*x*， *min*， *max*)**

参数

项目	说明
X	[in]要固定的值。
min	[in]指定的最小范围。
麦克斯	[in]指定的最大范围。

返回值

x 参数的固定值。

备注

对于 -INF 或 INF 的值，固定将按预期方式运行。 但是，对于 NaN 的值，结果是未定义的。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float、 int	any
min	与输入 *x* 相同	float、 int	与输入 *x* 相同的维度 (s)
max	与输入 *x* 相同	float、 int	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	float、 int	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL (内部函数)

clip

项目 • 2022/09/22

如果指定的值小于零，则放弃当前像素。

剪辑 (*x*)

参数

项	说明
X	[in]指定的值。

返回值

无。

备注

如果 *x* 参数的每个组件表示与平面的距离，则使用**剪辑** HLSL 内部函数模拟剪裁平面。

此外，使用 **剪辑** 函数测试 alpha 行为，如以下示例所示：

```
clip( Input.Color.A < 0.1f ? -1:1 );
```

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 1 (DirectX HLSL)	是 (仅像素着色器)

另请参阅

DirectX HLSL) 内部函数 (

cos

项目 • 2022/09/22

返回指定值的余弦值。

***ret* cos (*x*)**

参数

项	说明
X	[in]指定的值，以弧度为单位。

返回值

x 参数的余弦值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

cosh

项目 • 2022/09/22

返回指定值的双曲余弦值。

***ret* cosh (*x*)**

参数

项	说明
X	[in]指定的值，以弧度为单位。

返回值

x 参数的双曲余弦值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

countbits 函数

项目 • 2022/12/14

计算输入整数中设置的每个组件 (位数) 。

语法

```
syntax

uint countbits(
  in uint value
);
```

parameters

value [in]

类型： **uint**

输入值。

返回值

类型： **uint**

位数。

注解

以下重载版本也可用：

```
syntax

uint count_bits(uint value);
uint2 count_bits(uint2 value);
uint3 count_bits(uint3 value);
uint4 count_bits(uint4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

交叉

项目 • 2022/09/22

返回两个浮点 3D 向量的交叉乘积。

***ret* cross (*x*， *y*)**

参数

项	说明
X	[in]第一个浮点 3D 向量。
Y	[in]第二个浮点 3D 向量。

返回值

x 参数和 *y* 参数的交叉乘积。

类型说明

名称	模板类型	组件类型	大小
x	向量	float	3
y	向量	float	3
Ret	向量	float	3

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL (内部函数)

D3DCOLORtoUBYTE4

项目 • 2022/09/22

将 D3DCOLOR 设置的浮点 4D 向量转换为 UBYTE4。

***Ret* D3DCOLORtoUBYTE4 (*x*)**

此函数重排和缩放 *x* 参数的组件。 使用此函数补偿某些硬件中缺少 UBYTE4 支持。

参数

项	说明
X	[in]要转换的浮点向量 4。

返回值

x 参数的 UBYTE4 表示形式。

类型说明

名称	模板类型	组件类型	大小
x	向量	float	4
Ret	向量	integer	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

ddx

项目 • 2022/09/22

返回指定值相对于屏幕空间 x 坐标的部分派生。

***ret* ddx (*x*)**

此函数计算与屏幕空间 x 坐标相关的部分派生。 若要计算屏幕空间 y 坐标的部分派生，请使用 ddy 函数。

此函数仅在像素着色器中受支持。

参数

项	说明
X	[in]指定的值。

返回值

x 参数的部分派生。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	是

着色器模型	支持
着色器模型 2 (DirectX HLSL)	是，在 ps_2_x;ps_2_0不支持。
着色器模型 1 (DirectX HLSL)	否

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

DirectX HLSL) 内部函数 (

ddx_coarse函数

项目 • 2022/09/22

计算与屏幕空间 x 坐标相关的低精度部分派生。

语法

```
syntax

float ddx_coarse(
  in float value
);
```

参数

value [in]

类型： **float**

输入值。

返回值

类型： **float**

值的低精度部分派生。

备注

以下重载版本也可用：

```
syntax

float2 ddx_coarse(float2 value);
float3 ddx_coarse(float3 value);
float4 ddx_coarse(float4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

内部函数

着色器模型 5

ddx_fine函数

项目 • 2022/09/23

计算与屏幕空间 x 坐标相关的高精度分部派生。

语法

```
syntax

float ddx_fine(
  in float value
);
```

参数

值 [in]

类型： **float**

输入值。

返回值

类型： **float**

*值的*精确部分派生值。

注解

还可以使用以下重载版本：

```
syntax

float2 ddx_fine(float2 value);
float3 ddx_fine(float3 value);
float4 ddx_fine(float4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

内部函数

着色器模型 5

ddy

项目 • 2022/09/23

返回指定值相对于屏幕空间 y 坐标的部分派生值。

***ret* ddy (*x*)**

此函数计算屏幕空间 y 坐标的部分派生。 若要计算屏幕空间 x 坐标的部分派生，请使用 ddx 函数。

此函数仅在像素着色器中受支持。

参数

项	说明
X	[in]指定的值。

返回值

x 参数的部分派生。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	是

着色器模型	支持
着色器模型 2 (DirectX HLSL)	是，在 ps_2_x;在 ps_2_0 中不受支持。
着色器模型 1 (DirectX HLSL)	否

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	

另请参阅

DirectX HLSL (内部函数)

ddy_coarse函数

项目 • 2022/09/22

计算与屏幕空间 y 坐标相关的低精度部分派生。

语法

```
syntax

float ddy_coarse(
  in float value
);
```

参数

值 [in]

类型： **float**

输入值。

返回值

类型： **float**

*值的*低精度部分派生值。

注解

还可以使用以下重载版本：

```
syntax

float2 ddy_coarse(float2 value);
float3 ddy_coarse(float3 value);
float4 ddy_coarse(float4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

内部函数

着色器模型 5

ddy_fine函数

项目 • 2022/09/22

计算与屏幕空间 x 坐标相关的高精度分部派生。

语法

```
syntax

float ddy_fine(
  in float value
);
```

parameters

值 [in]

类型： **float**

输入值。

返回值

类型： **float**

*值的*精确部分派生值。

注解

还可以使用以下重载版本：

```
syntax

float2 ddy_fine(float2 value);
float3 ddy_fine(float3 value);
float4 ddy_fine(float4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

内部函数

着色器模型 5

度

项目 • 2022/09/22

将指定的值从弧度转换为度。

***ret* degrees (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

将 *x* 参数从弧度转换为度的结果。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

行列 式

项目 • 2022/09/22

返回指定浮点平方矩阵的确定性。

***重试* 确定性 (*m*)**

参数

项	说明
m	[in]指定的值。

返回值

表示 *m* 参数确定值的浮点标量值。

类型说明

名称	模板类型	组件类型	大小
m	矩阵	float	任何 (行数 = 列数)
Ret	标量	FLOAT	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL) 内部函数 (

DeviceMemoryBarrier 函数

项目 • 2022/09/22

阻止执行组中的所有线程，直到所有设备内存访问都已完成。

语法

```
syntax

void DeviceMemoryBarrier(void);
```

parameters

此函数没有参数。

返回值

此函数不返回值。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	x

另请参阅

内部函数

着色器模型 5

DeviceMemoryBarrierWithGroupSync 函数

项目 • 2022/09/22

阻止执行组中的所有线程，直到所有设备内存访问都已完成，并且组中的所有线程都已达到此调用。

语法

```
syntax

void DeviceMemoryBarrierWithGroupSync(void);
```

parameters

此函数没有参数。

返回值

此函数不返回值。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
					x

另请参阅

内部函数

着色器模型 5

distance

项目 • 2022/09/22

返回两个向量之间的距离标量。

***ret* distance (*x*， *y*)**

参数

项	说明
X	[in]要比较的第一个浮点向量。
Y	[in]要比较的第二个浮点向量。

返回值

一个浮点标量值，表示 *x* 参数与 *y* 参数之间的距离。

类型说明

名称	模板类型	组件类型	大小
x	向量	float	any
y	向量	float	与输入 *x* 相同的维度 (s)
Ret	标量	FLOAT	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

dot

项目 • 2022/09/22

返回两个向量的点积。

***ret* dot (*x* , *y*)**

参数

项	说明
X	[in]第一个向量。
Y	[in]第二个向量。

返回值

x 参数和 *y* 参数的点积。

类型说明

名称	模板类型	组件类型	大小
x	向量	float、 int	any
y	向量	float、 int	与输入 *x* () 相同的尺寸
Ret	标量	float、 int	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是

另请参阅

DirectX HLSL (内部函数)

dst 函数

项目 • 2022/09/23

计算距离向量。

语法

```
syntax

fVector dst(
  in fVector src0,
  in fVector src1
);
```

参数

src0 [in]

类型： fVector

第一个向量。

src1 [in]

类型： fVector

第二个向量。

返回值

类型： fVector

计算的距离向量。

备注

此内部函数提供与顶点着色器指令 dst 相同的功能。

另请参阅

内部函数

errorf 函数

项目 • 2022/09/23

将错误消息提交到信息队列。

语法

syntax

```
void errorf(
   string format,
    argument ...
);
```

参数

format

格式字符串。

参数。。。

可选参数。

返回值

此函数不返回值。

备注

此操作对不支持此操作的设备不执行任何操作。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 (DirectX HLSL) 或更高版本。	是

另请参阅

内部函数

EvaluateAttributeCentroid 函数

项目 • 2022/09/22

以像素质心求值。

语法

```
syntax

numeric EvaluateAttributeCentroid(
  in attrib numeric value
);
```

参数

value [in]

类型： **attrib numeric**

输入值。

注解

内插模式可以是 **线性** 模式，也可以是变量上的 linear_no_perspective 。 忽略 **质心** 或 **样本** 的使用。 还允许使用常量内插的属性，在这种情况下，将忽略示例索引。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
				x	

请参阅

内部函数

着色器模型 5

EvaluateAttributeAtSample 函数

项目 • 2022/09/23

在索引的示例位置求值。

语法

```
syntax
```

```
numeric EvaluateAttributeAtSample(
  in attrib numeric value,
  in uint sampleindex
);
```

参数

value [in]

类型： **attrib numeric**

输入值。

sampleindex [in]

类型： **uint**

示例位置。

注解

内插模式可以是 **线性** 模式，也可以是变量上的 linear_no_perspective 。 忽略 **质心** 或 **样本** 的使用。 还允许使用常量内插的属性，在这种情况下，将忽略示例索引。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

内部函数

着色器模型 5

EvaluateAttributeSnapped 函数

项目 • 2022/09/22

以像素质心求值（偏移量）求值。

语法

syntax

```
numeric EvaluateAttributeSnapped(
  in attrib numeric value,
  in
            int2 offset
);
```

参数

value [in]

类型： **attrib numeric**

输入值。

offset [in]

类型： **int2**

使用 16x16 网格从像素中心偏移量 2D。

注解

*偏移*参数的范围必须由以下字节代码定义。

仅使用前两个组件 (U 的最小有效 4 位，使用像素偏移量的 V) 。 从 4 位固定点到浮点的转换如下所示 (MSB...LSB) ，其中 MSB 既是分数的一部分，又确定符号：

- 1000 = -0.5f (-8 / 16)
- 1001 = -0.4375f (-7 / 16)
- 1010 = -0.375f (-6 / 16)
- 1011 = -0.3125f (-5 / 16)
- 1100 = -0.25f (-4 / 16)
- 1101 = -0.1875f (-3 / 16)

- 1110 = -0.125f (-2 / 16)
- 1111 = -0.0625f (-1 / 16)
- 0000 = 0.0f (0 / 16)
- 0001 = 0.0625f (1 / 16)
- 0010 = 0.125f (2 / 16)
- 0011 = 0.1875f (3 / 16)
- 0100 = 0.25f (4 / 16)
- 0101 = 0.3125f (5 / 16)
- 0110 = 0.375f (6 / 16)
- 0111 = 0.4375f (7 / 16)

> **备注**
>
> 像素的左边缘和上边缘包含在偏移量中;但是，不包括底部和右边缘。 忽略 32 位整数和 V 偏移值中的所有其他位。

实现可以采用着色器提供的偏移量，并通过执行以下计算，获取完整的 32 位固定点值 (28.4) ，该值跨越有效范围：

```
iU = (iU<<28)>>28  // keep lowest 4 bits and sign extend, which yields
[-8..7]
```

如果实现必须将偏移量映射到浮点偏移量，则执行以下计算：

```
fU = ((float)iU)/16
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
				x	

另请参阅

内部函数

着色器模型 5

exp

项目 • 2022/09/22

返回指定值的 base-e 指数或 e^x。

***ret* exp (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的 base-e 指数。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL (内部函数)

exp2

项目 • 2022/09/23

返回指定值的基 2 指数或 2 倍。

***ret* exp2 (*x*)**

参数

项目	说明
X	[in]指定的浮点值。

返回值

x 参数的基 2 指数。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

f16tof32 函数

项目 • 2022/09/23

将存储在 uint 低半部分的 float16 转换为浮点。

语法

```
syntax
```

```
float f16tof32(
  in uint value
);
```

parameters

值 [in]

类型： **uint**

输入值。

返回值

类型： **float**

转换后的值。

注解

还可以使用以下重载版本：

```
syntax
```

```
float2 f16tof32(uint2 value);
float3 f16tof32(uint3 value);
float4 f16tof32(uint4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 及更高着色器模型	是

尽管此函数在 着色器模型 4 和更高着色器模型中可用，但由于它在 4.0 和 4.1 中得到模拟，但这些着色器模型上的 perfomant 小于 着色器模型 5。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

f32tof16 函数

项目 • 2022/09/22

将输入转换为 float16 类型。

语法

```
syntax
```

```
uint f32tof16(
  in float value
);
```

参数

值 [in]

类型： **float**

输入值。

返回值

类型： **uint**

转换后的值，存储在 uint 的下半部分。

备注

还可以使用以下重载版本：

```
syntax
```

```
uint2 f32tof16(float2 value);
uint3 f32tof16(float3 value);
uint4 f32tof16(float4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 及更高着色器模型	是

尽管此函数在 着色器模型 4 和更高着色器模型中可用，但由于它在 4.0 和 4.1 中得到模拟，但这些着色器模型上的 perfomant 小于 着色器模型 5。

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

faceforward

项目 • 2022/09/23

如果需要，) 面对与 i 相反的方向翻转表面法线 (;以 n 为单位返回结果。

***ret* faceforward (*n*， *i*， *ng*)**

此函数使用以下公式：*-n* 符号 (点 (*i*， *ng*)) 。

参数

项	说明
n	[in]生成的浮点图面法向量。
我	[in]从视图位置指向底纹位置的浮点事件向量。
议员	[in]浮点图面法向量。

返回值

面向视图方向的浮点图面法向量。

类型说明

名称	模板类型	组件类型	大小
n	向量	float	any
i	向量	float	与输入 *n* 相同的维度 (s)
议员	向量	float	与输入 *n* 相同的维度
Ret	向量	float	与输入 *n* 相同的维度

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1和ps_1_4

另请参阅

DirectX HLSL (内部函数)

firstbithigh 函数

项目 • 2022/09/23

获取从最高顺序位开始的第一个集位的位置，并按组件向下工作。

语法

```
syntax

int firstbithigh(
  in int value
);
```

参数

值 [in]

类型： **int**

输入值。

返回值

类型： **int**

第一个集位的位置。

注解

对于带符号整数，第一个有效位是负数的零。

还可以使用以下重载版本：

```
syntax

int2 firstbithigh(int2 value);
int3 firstbithigh(int3 value);
int4 firstbithigh(int4 value);
uint firstbithigh(uint value);
uint2 firstbithigh(uint2 value);
uint3 firstbithigh(uint3 value);
uint4 firstbithigh(uint4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

firstbitlow 函数

项目 • 2022/09/23

返回从最低顺序位开始的第一个集位的位置，并按组件向上工作。

语法

syntax

```
int firstbitlow(
  in int value
);
```

参数

值 [in]

类型： **int**

输入值。

返回值

类型： **int**

第一个集位的位置。

注解

还可以使用以下重载版本：

syntax

```
uint2 firstbitlow(uint2 value);
uint3 firstbitlow(uint3 value);
uint4 firstbitlow(uint4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

floor

项目 • 2022/09/22

返回小于或等于指定值的最大整数。

***ret* floor (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

作为浮点类型 (返回的最大整数值) 小于或等于 *x* 参数。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

fma

项目 • 2022/09/23

返回 * b + c 的双精度融合加法。

```
ret fma (double a , b , c) ;
```

参数

a

[in]融合乘法中的第一个值。

B

[in]融合乘法中的第二个值。

C

[in]融合乘法中的第三个值。

返回值

双精度融合加法参数 *abc* + * 。 返回的值必须准确到 0.5 单位的最小精度， (ULP) 。

注解

fma 内部函数必须支持 NaN、INF 和 Denorms。

若要在着色器代码中使用 **fma** 内部函数，请使用 D3D11_FEATURE_D3D11_OPTIONS 调用 ID3D11Device：：CheckFeatureSupport 方法，验证 Direct3D 设备是否支持 ExtendedDoublesShaderInstructions 功能选项。 **fma** 内部函数需要 WDDM 1.2 显示驱动程序，并且所有 WDDM 1.2 显示驱动程序都必须支持 **fma**。 如果应用创建 功能级别 为 11.0 或 11.1 的呈现设备，并且编译目标为着色器模型 5 或更高版本，则 HLSL 源代码可以使用 **fma** 内部函数。

类型说明

名称	模板类型	组件类型	大小
a	**标量**、 **矢量**或 **矩阵**	double	any

名称	模板类型	组件类型	大小
b	与输入 *相同*	double	*与输入*相同的维度
c	与输入 *相同*	double	*与输入*相同的维度
Ret	与输入 *相同*	double	*与输入*相同的维度

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 或更高版本	是

要求

要求	值
最低受支持的客户端	Windows 8 [桌面应用|UWP 应用]
最低受支持的服务器	Windows Server 2012 [桌面应用|UWP 应用]
标头	Corecrt_math.h

另请参阅

内部函数

fmod

项目 • 2022/09/23

返回 x/y 的浮点余数。

***ret* fmod (*x* , *y*)**

参数

项	说明
X	[in]浮点股息。
Y	[in]浮点除数。

返回值

x 参数的浮点余数除以 *y* 参数。

注解

浮点余数的计算结果为 x = i * y + f，其中 i 是整数，f 的符号与 x 相同，f 的绝对值小于 y 的绝对值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
y	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

要求

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

压裂

项目 • 2022/09/22

返回 x 部分的小数 (或十进制) 部分;大于或等于 0 且小于 1。

另请参阅 截断。

***ret* frac (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的小数部分。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	vs_1_1

另请参阅

DirectX HLSL) 内部函数 (

frexp

项目 • 2022/09/23

返回指定浮点值的 mantissa 和指数。

***ret* frexp (*x* , *exp*)**

返回值为 mantissa， *exp* 参数返回的值是指数。

参数

项	说明
X	[in]指定的浮点值。 如果 *x* 参数为 0，则此函数返回 mantissa 和指数的 0。
exp	[out] *x* 参数的返回指数。

返回值

x 参数的 mantissa。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
exp	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 3 (DirectX HLSL) 及更高着色器模型	是
着色器模型 2 (DirectX HLSL)	是 (ps_2_*x*仅)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

备注

要求

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

fwidth

项目 • 2022/09/23

返回指定值的分部派生的绝对值。

***ret* fwidth (*x*)**

此函数计算以下内容： abs (ddx (*x*)) + abs (ddy (*x*)) 。

此函数仅在像素着色器中受支持。

参数

项	描述
X	[in]指定的值。

返回值

x 参数的部分派生值的绝对值。

类型说明

名称	模板类型	组件类型	大小
x	标量、 矢量或 矩阵	浮动	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 3 (DirectX HLSL) 及更高着色器模型	是
着色器模型 2 (DirectX HLSL)	是 (ps_2_*x*仅)
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

GetRenderTargetSampleCount

项目 • 2022/09/22

获取呈现目标的示例数。

***UINT* GetRenderTargetSampleCount ()**

参数

项	说明
没有	

返回值

样本数。

注解

使用此函数和 GetRenderTargetSamplePosition 可找出呈现目标采样位置的数量和位置。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 和更高着色器模型	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

GetRenderTargetSamplePosition

项目 • 2022/09/22

获取给定样本索引的采样位置 (x，y) 。

float2<> GetRenderTargetSamplePosition (int1<> Index
);

参数

项	说明
指数	[in]从零开始的示例索引。

返回值

给定样本的 (x，y) 位置。

注解

使用此函数和 GetRenderTargetSampleCount 可找出呈现目标的采样位置的数量和位置。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 及更高着色器模型	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL (内部函数)

GroupMemoryBarrier 函数

项目 • 2022/09/23

阻止执行组中的所有线程，直到所有组共享访问都已完成。

语法

```
syntax

void GroupMemoryBarrier(void);
```

parameters

此函数没有参数。

返回值

此函数不返回值。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
					x

另请参阅

内部函数

着色器模型 5

GroupMemoryBarrierWithGroupSync 函数

项目 • 2022/09/23

阻止执行组中的所有线程，直到所有组共享访问都已完成，并且组中的所有线程都已达到此调用。

语法

syntax

```
void GroupMemoryBarrierWithGroupSync(void);
```

parameters

此函数没有参数。

返回值

此函数不返回值。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
					x

另请参阅

内部函数

着色器模型 5

InterlockedAdd 函数 (HLSL 引用)

项目 • 2022/09/23

对 dest 资源变量执行有保证的原子值添加。

语法

syntax

```
void InterlockedAdd(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

value [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

可选。 原始输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一个是 R 是共享内存变量类型。 在本例中，该函数对 dest 引用的共享内存寄存器执行原子值添加。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子值添加。 重载函数具有一个额外的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedAnd 函数 (HLSL 引用)

项目 • 2022/09/23

执行有保证的原子和。

语法

syntax

```
void InterlockedAnd(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

value [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

可选。 原始输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行原子值和值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子值和值。 重载函数具有一个额外的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedCompareExchange 函数 (HLSL 引用)

项目 • 2022/09/23

以原子方式比较目标与比较值。 如果它们相同，则使用输入值覆盖目标。 原始值设置为目标的原始值。

语法

syntax

```
void InterlockedCompareExchange(
  in  R dest,
  in  T compare_value,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

compare_value [in]

类型： **T**

比较值。

值 [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

原始值。

返回值

此函数不返回值。

备注

以原子方式比较 *dest* 引用的值与 *compare_value*，如果值匹配，请将*值*存储在 *dest* 引用的位置，并在 *original_value* 中返回 *dest* 的原始值。 此操作只能在 **int** 或 **uint** 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 *dest* 引用的共享内存寄存器执行操作。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 *dest* 引用的资源位置执行操作。 仅当 R 可读且可写时，此操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

> **备注**
>
> 如果在**计算**着色器循环中或**计算**着色器循环中调用 InterlockedCompareExchange，则必须在该循环上使用 [allow_uav_condition] 属性。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedCompareStore 函数 (HLSL 引用)

项目 • 2022/09/22

以原子方式将目标与比较值进行比较。 如果它们相同，则使用输入值覆盖目标。

语法

syntax

```
void InterlockedCompareStore(
  in R dest,
  in T compare_value,
  in T value
);
```

parameters

dest [in]

类型： **R**

目标地址。

compare_value [in]

类型： **T**

比较值。

value [in]

类型： **T**

输入值。

返回值

此函数不返回值。

备注

原子将 *dest* 引用的值与 *compare_value* 进行比较，并将 *值* 存储在 *dest* 引用的位置（如果值匹配）。 此操作只能在 **int** 或 **uint** 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数对 *dest* 引用的共享内存寄存器执行操作。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 *dest* 引用的资源位置执行操作。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedExchange 函数 (HLSL 引用)

项目 • 2022/09/22

为 dest 赋值并返回原始值。

语法

syntax

```
void InterlockedExchange(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

值 [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

原始值。

返回值

此函数不返回值。

备注

此操作只能对标量类型的资源和共享内存变量执行。 此函数有两种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行操作。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行操作。 仅当 R 可读且可写时，此操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedMax 函数 (HLSL 引用)

项目 • 2022/09/22

执行有保证的原子最大值。

语法

syntax

```
void InterlockedMax(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

value [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

可选。 原始输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 和 uint 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行原子最大值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子最大值。 重载函数具有一个额外的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedMin 函数 (HLSL 引用)

项目 • 2022/09/22

执行有保证的原子最小值。

语法

syntax

```
void InterlockedMin(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

值 [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

可选。 原始输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 和 uint 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行原子最小值值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子最小值值。 重载函数具有一个附加的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

InterlockedOr 函数 (HLSL 引用)

项目 • 2022/09/22

执行有保证的原子或。

语法

syntax

```
void InterlockedOr(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

value [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

可选。 原始输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一个是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行原子或值。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子或值。 重载函数具有一个额外的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

interlockedXor 函数 (HLSL 引用)

项目 • 2022/09/23

执行有保证的原子 xor。

语法

syntax

```
void InterlockedXor(
  in  R dest,
  in  T value,
  out T original_value
);
```

parameters

dest [in]

类型： **R**

目标地址。

值 [in]

类型： **T**

输入值。

original_value [out]

类型： **T**

可选。 原始输入值。

返回值

此函数不返回值。

备注

此操作只能在 int 或 uint 类型化资源和共享内存变量上执行。 此函数有两种可能用途。 第一种是 R 是共享内存变量类型。 在这种情况下，该函数对 dest 引用的共享内存寄存器执行值原子 XOR。 第二种情况是 R 是资源变量类型。 在此方案中，该函数对 dest 引用的资源位置执行原子 XORof 值。 重载函数具有一个附加的输出变量，该变量将设置为 dest 的原始值。 仅当 R 可读且可写时，此重载操作才可用。

互锁操作并不意味着任何内存围栏/屏障。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何图形	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

isfinite

项目 • 2022/09/23

确定指定的浮点值是否有限。

***ret* isfinite (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

返回与输入大小相同的值，如果 *x* 参数有限，则值为 **True**;否则**为 False**。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	**标量**、 **矢量**或 **矩阵**	bool	作为输入

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL (内部函数)

isinf

项目 • 2022/09/23

确定指定的值是否为无限值。

***ret* isinf (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

返回与输入大小相同的值，如果 *x* 参数为 +INF 或 -INF，则值设置为 **True**。 否则为 **False**。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	**标量**、 **矢量**或 **矩阵**	bool	作为输入

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

isnan

项目 • 2022/09/24

确定指定的值为 NAN 还是 QNAN。

***ret* isnan (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

返回与输入大小相同的值，如果 *x* 参数为 NAN 或 QNAN，则值为 **True**。 否则为 **False**。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
Ret	**标量**、 **矢量**或 **矩阵**	bool	作为输入

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

ldexp

项目 • 2022/09/23

返回将指定值乘以 2 的结果，结果被提升为指定指数的幂。

***ret* ldexp (*x*， *exp*)**

此函数使用以下公式： $x * 2^{exp}$

参数

项	说明
X	[in]指定的值。
exp	[in]指定的指数。

返回值

将 *x* 参数乘以 2 的结果，将结果提升为 *exp* 参数的幂。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
exp	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL (内部函数)

length

项目 • 2022/09/23

返回指定的浮点向量长度。

***ret* length (*x*)**

参数

项	说明
X	指定的浮点向量。

返回值

一个浮点标量，表示 *x* 参数的长度。

类型说明

名称	模板类型	组件类型	大小
x	向量	float	any
Ret	标量	FLOAT	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

lerp

项目 • 2022/09/22

执行线性内插。

***ret* lerp (*x* , *y* , *s*)**

参数

项	说明
X	[in]第一个浮点值。
Y	[in]第二个浮点值。
s	[in]在 *x* 参数和 *y* 参数之间线性内插的值。

返回值

线性内插的结果。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
y	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
s	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

注解

线性内插基于以下公式：x* (1-s) + y*s，这相当于可以写入 x + s (y-x) 。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1和ps_1_1)

另请参阅

DirectX HLSL) 内部函数 (

点燃

项目 • 2022/09/22

返回照明系数向量。

ret* lit *(n_dot_l , n_dot_h , m)

此函数返回照明系数向量 (环境、漫射、反射、1) ，其中：

- ambient = 1
- 漫射 = n ·l < 0 ？ 0 ： n ·我
- 反射 = n ·l < 0 || n · h < 0 ？ 0 ： (n ·h) ^ m

其中向量 n 是正向量，l 是光的方向，h 是半向量。

参数

项	说明
n_dot_l	[in]规范化表面法线和光向量的点积。
n_dot_h	[in]半角向量和表面法线的点积。
m	[in]一个反射指数。

返回值

照明系数向量。

类型说明

名称	模板类型	组件类型	大小
n_dot_l	标量	FLOAT	1
n_dot_h	标量	FLOAT	1
m	标量	FLOAT	1
Ret	向量	float	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

log

项目 • 2022/09/22

返回指定值的 base-e 对数。

***ret* log (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的 base-e 对数。 如果 *x* 参数为负数，此函数将无限期返回。 如果 *x* 参数为 0，则此函数返回 -INF。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

log10

项目 • 2022/09/22

返回指定值的 base-10 对数。

重试* log10 *(x)

参数

项	说明
X	[in]指定的值。

返回值

x 参数的 base-10 对数。 如果 *x* 参数为负数，此函数将无限期返回。 如果 *x* 为 0，则此函数返回 -INF。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

log2

项目 • 2022/09/24

返回指定值的 base-2 对数。

***重试* log2 (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的 base-2 对数。 如果 *x* 参数为负数，此函数将无限期返回。 如果 *x* 参数为 0，则此函数返回 +INF。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

mad 函数

项目 • 2022/09/22

对三个值执行算术乘/加运算。

语法

syntax

```
numeric mad(
  in numeric mvalue,
  in numeric avalue,
  in numeric bvalue
);
```

参数

mvalue [in]

类型： **数值**

乘法值。

avalue [in]

类型： **数值**

第一个加法值。

bvalue [in]

类型： **数值**

第二个加法值。

返回值

类型： **数值**

mvalueavaluebvalue + * 的结果。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

着色器作者可以使用 **疯狂** 的内生体显式定位编译着色器输出中的 **疯狂** 硬件指令，这对于使用 精确 关键字标记结果的着色器特别有用。 **疯狂**指令可以在硬件中实现为“融合”，其精度高于实现 mul 指令，后跟**添加**指令，或者作为**多条 + 指令实现。**

如果着色器作者使用 **疯狂** 的内生体来计算标记为精确着色器的结果，则它们向硬件指示使用 **疯狂** 指令的任何有效实现，（融合或不）只要实现在该硬件上的任何着色器中的所有使用 **这种疯狂** 内部函数都是一致的。 然后，着色器可以使用本机**疯狂**指令（与某些硬件上的**多 +** ）来利用潜在的性能改进。 执行本机 **疯狂** 硬件指令的结果可能与执行 mul 后跟**添加**不同。 但是，无论结果是什么，结果都必须一致，才能在同一操作的多个着色器或着色器的不同部分发生。

另请参阅

内部函数

着色器模型 5

max

项目 • 2022/09/24

选择 x 和 y 的更大项。

ret max (x， y)

参数

项	说明
X	[in]x 输入值。
Y	[in]y 输入值。

返回值

x 或 *y* 参数，以最大值为准。

注解

非规范处理方式如下：

src0 src1->	-inf	F	+inf	NAN
-inf	-inf	src1	+inf	-inf
F	src0	src0 或 src1	+inf	src0
+inf	+inf	+inf	+inf	+inf
NaN	-inf	src1	+inf	NaN

F 表示有限实数。

类型说明

名称	输入/输出	模板类型	组件类型	大小
x	in	标量、 矢量或 矩阵	float、 int	any

名称	输入/输出	模板类型	组件类型	大小
y	in	与输入 x 相同	float、 int	与输入 x 相同的维度 ()
Ret	返回类型	与输入 x 相同	float、 int	与输入 x 相同的维度 ()

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1和ps_1_4)

另请参阅

DirectX HLSL (内部函数)

DirectX 功能规范

min

项目 • 2022/09/22

选择 x 和 y 的较小值。

ret min (x， y)

参数

项	说明
X	[in]x 输入值。
Y	[in]y 输入值。

返回值

x 或 *y* 参数，以最小值为准。

注解

非规范如下处理：

src0 src1->	-inf	F	+inf	NAN
-inf	-inf	-inf	-inf	-inf
F	-inf	src0 或 src1	src0	src0
+inf	-inf	src1	+inf	+inf
NaN	-inf	src1	+inf	NaN

F 表示有限实数。

类型说明

名称	输入/输出	模板类型	组件类型	大小
x	in	标量、 矢量或 矩阵	float、 int	any

名称	输入/输出	模板类型	组件类型	大小
y	in	与输入 x 相同	float、 int	与输入 x 相同的维度 (s)
Ret	返回类型	与输入 x 相同	float、 int	与输入 x 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1和ps_1_4)

另请参阅

DirectX HLSL) 内部函数 (

DirectX 功能规范

modf

项目 • 2022/09/22

将值 x 拆分为小数部分和整数部分，每个部分的符号与 x 相同。

ret modf (x， out ip)

参数

项	说明
X	[in]x 输入值。
Ip	[out] *x* 的整数部分。

返回值

x 的带符号小数部分。

类型说明

名称	输入/输出	模板类型	组件类型	大小
x	in	**标量**、 **矢量**或 **矩阵**	float、 int	any
ip	out	与输入 x 相同	float、 int	与输入 x 相同的维度 ()
Ret	out	与输入 x 相同	float、 int	与输入 x 相同的维度 ()

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL (内部函数)

msad4

项目 • 2022/09/22

比较 4 字节引用值和 8 字节源值，并累积 4 个总和的向量。 每个总和对应于引用值与源值之间不同字节对齐的绝对差的掩码总和。

```
uint4 result = msad4 (uint reference， uint2 source， uint4 accum) ;
```

参数

参考

[in]一个 **uint** 值中 4 个字节的引用数组。

源

[in]两个 **uint2** 值中的 8 个字节的源数组。

accum

[in]4 个值的向量。 **msad4** 将此向量添加到引用值与源值之间不同字节对齐的掩码绝对差异之和。

返回值

4 个求和的向量。 每个总和对应于引用值与源值之间不同字节对齐的绝对差异的掩码总和。 如果掩码（（即）引用字节为 0），**则 msad4** 不包括总和的差异。

注解

若要在着色器代码中使用 **msad4** 内部函数，请使用 D3D11_FEATURE_D3D11_OPTIONS调用 ID3D11Device：：CheckFeatureSupport 方法，以验证 Direct3D 设备是否支持 SAD4ShaderInstructions 功能选项。 **msad4** 内部函数需要 WDDM 1.2 显示驱动程序，所有 WDDM 1.2 显示驱动程序都必须支持 **msad4**。 如果应用创建 功能级别 为 11.0 或 11.1 的呈现设备，并且编译目标为着色器模型 5 或更高版本，HLSL 源代码可以使用 **msad4** 内部函数。

返回值仅准确到 65535。 如果使用可能导致返回值大于 65535 的输入调用 **msad4** 内部函数， **msad4** 将生成未定义的结果。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 或更高版本	是

示例

下面是 msad4 的示例结果计算：

```
reference = 0xA100B2C3;
source.x = 0xD7B0C372
source.y = 0x4F57C2A3
accum = {1,2,3,4}
result.x alignment source: 0xD7B0C372
result.x = accum.x + |0xD7   0xA1| + 0 (masked) + |0xC3   0xB2| + |0x72
0xC3| = 1 + 54 + 0 + 17 + 81 = 153
result.y alignment source: 0xA3D7B0C3
result.y = accum.y + |0xA3   0xA1| + 0 (masked) + |0xB0   0xB2| + |0xC3
0xC3| = 2 + 2 + 0 + 2 + 0 = 6
result.z alignment source: 0xC2A3D7B0
result.z = accum.z + |0xC2   0xA1| + 0 (masked) + |0xD7   0xB2| + |0xB0
0xC3| = 3 + 33 + 0 + 37 + 19 = 92
result.w alignment source: 0x57C2A3D7
result.w = accum.w + |0x57   0xA1| + 0 (masked) + |0xA3   0xB2| + |0xD7
0xC3| = 4 + 74 + 0 + 15 + 20 = 113
result = {153,6,92,113}
```

下面是如何使用 msad4 在缓冲区中搜索引用模式的示例：

```
uint4 accum = {0,0,0,0};
for(uint i=0;i<REF_SIZE;i++)
    accum = msad4(
        buf_ref[i],
        uint2(buf_src[DTid.x+i], buf_src[DTid.x+i+1]),
        accum);
buf_accum[DTid.x] = accum;
```

要求

要求	值
最低受支持的客户端	Windows 8 [桌面应用|UWP 应用]
最低受支持的服务器	Windows Server 2012 [桌面应用|UWP 应用]

另请参阅

内部函数

mul

项目 • 2022/09/23

使用矩阵数学乘以 x 和 y。 内部维度 x 列和 y 行必须相等。

ret mul (x， y)

参数

项	说明
X	[in]x 输入值。 如果 x 是矢量，则它被视为行向量。
Y	[in]y 输入值。 如果 y 是矢量，则它被视为列向量。

返回值

x times y 的结果。 结果具有维度 x 行 x y 列。

类型说明

此函数有 9 个重载版本;重载版本处理输入参数的类型和大小的不同情况。

版本	名称	目的	模板类型	组件类型	大小
1					
	x	in	标量 (scalar)	float、int	1
	y	in	标量 (scalar)	与输入 x 相同	1
	Ret	out	标量 (scalar)	与输入 x 相同	1
2					
	x	in	标量 (scalar)	float、int	1
	y	in	向量	float、int	any

版本	名称	目的	模板类型	组件类型	大小
	Ret	out	向量	float、int	与输入 y 相同的维度 (s)
3					
	x	in	标量 (scalar)	float、int	1
	y	in	矩形图	float、int	any
	Ret	out	矩形图	与输入 y 相同	与输入 y 相同的维度 (s)
4					
	x	in	向量	float、int	any
	y	in	标量 (scalar)	float、int	1
	Ret	out	向量	float、int	与输入 x 相同的维度 ()
5					
	x	in	向量	float、int	any
	y	in	向量	float、int	与输入 x 相同的维度 ()
	Ret	out	标量 (scalar)	float、int	1
6					
	x	in	向量	float、int	any
	y	in	矩形图	float、int	行 = 与输入 x 相同的维度 () ，列 = 任意
	Ret	out	向量	float、int	与输入 y 列相同的维度 ()
7					
	x	in	矩形图	float、int	any
	y	in	标量 (scalar)	float、int	1
	Ret	out	矩形图	float、int	与输入 x 相同的维度 ()
8					
	x	in	矩形图	float、int	any

版本	名称	目的	模板类型	组件类型	大小
	y	in	向量	float、int	输入 x 中的列数
	Ret	out	向量	float、int	输入 x 中的行数
9					
	x	in	矩形图	float、int	any
	y	in	矩形图	float、int	rows = input x 中的列数
	Ret	out	矩形图	float、int	rows = input x 中的行数，列 = 输入 y 中的列数

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 1 (DirectX HLSL) 和更高的着色器模型	是

另请参阅

DirectX HLSL (内部函数)

噪音

项目 • 2022/09/23

使用 Perlin-noise 算法生成随机值。

***(x)* 重试噪音**

参数

项	说明
X	[in]从中生成 Perlin 噪声的浮点向量。

返回值

Perlin 噪声值介于 -1 和 1 之间。

注解

Perlin 噪音值从一个点到另一个点在空间上顺利变化，从而产生自然外观的随机生成的值。 可以使用 Perlin 噪音为烟雾和火等效果生成过程纹理。

类型说明

名称	模板类型	组件类型	大小
x	向量	FLOAT	any
Ret	标量	FLOAT	1

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	否
着色器模型 1 (DirectX HLSL)	是 (tx_1_0仅)

另请参阅

DirectX HLSL) 内部函数 (

规范化

项目 • 2022/09/22

根据 x/length (x) 规范化指定的浮点向量。

***ret* 规范化 (*x*)**

参数

项	说明
X	[in]指定的浮点向量。

返回值

规范化的 *x* 参数。 如果 *x* 参数的长度为 0，则结果无限期。

注解

规范化 HLSL 内部函数使用以下公式：**xlength** / (*x*) 。

类型说明

名称	模板类型	组件类型	大小
x	**向量**	FLOAT	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

pow

项目 • 2022/09/24

返回引发到指定幂的指定值。

***ret* pow (*x* , *y*)**

parameters

项	描述
X	[in]指定的值。
Y	[in]指定的电源。

返回值

向 *y* 参数的幂引发的 *x* 参数。

注解

pow HLSL 内部函数计算 *xy*。

X	Y	结果
< 0	any	NAN
> 0	== 0	1
== 0	> 0	0
== 0	< 0	inf
> 0	< 0	$1/^{X-Y}$
== 0	== 0	取决于特定图形处理器 0、1 或 NAN

类型说明

名称	模板类型	组件类型	大小

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
y	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

printf 函数

项目 • 2022/09/22

将自定义着色器消息提交到信息队列。

语法

syntax

```
void printf(
   string format,
    argument ...
);
```

参数

format

格式字符串。

参数。。。

可选参数。

返回值

此函数不返回值。

备注

此操作对不支持此操作的设备不执行任何操作。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4 (DirectX HLSL) 或更高版本。	是

另请参阅

内部函数

Process2DQuadTessFactorsAvg 函数

项目 • 2022/09/23

为象限修补程序生成更正后的细化因子。

语法

syntax

```
void Process2DQuadTessFactorsAvg(
  in  float4 RawEdgeFactors,
  in  float2 InsideScale,
  out float4 RoundedEdgeTessFactors,
  out float2 RoundedInsideTessFactors,
  out float2 UnroundedInsideTessFactors
);
```

parameters

RawEdgeFactors [in]

类型： **float4**

边缘分割因素已传递到细化器阶段。

InsideScale [in]

类型： **float2**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float4**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float2**

由细化器阶段计算的舍入细化因子（由内边缘的细化器阶段计算）。

UnundedInsideTessFactors [out]

类型： **float2**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为四方补丁生成更正后的细化因子，将内部分割因子计算为边缘细化因子的平均值。 使用域的对方平均值独立计算你和 V 内部分割因子，然后由 InsideScale 缩放。 然后，根据分区模式对结果进行舍入，但未解决的结果可以使用 UnroundedInsideTessFactors 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

Process2DQuadTessFactorsMax 函数

项目 • 2022/09/22

为象限补丁生成更正的细化因子。

语法

syntax

```
void Process2DQuadTessFactorsMax(
  in  float4 RawEdgeFactors,
  in  float2 InsideScale,
  out float4 RoundedEdgeTessFactors,
  out float2 RoundedInsideTessFactors,
  out float2 UnroundedInsideTessFactors
);
```

参数

RawEdgeFactors [in]

类型： **float4**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float2**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float4**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float2**

细化器阶段为内边缘计算的舍入细化因子。

UnroundedInsideTessFactors [out]

类型： **float2**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为四个补丁生成更正的细化因子，将内部细化因子计算为边缘分割因子的最大值。 使用域的对立面的最大值来独立计算你和 V 内部分割因子，然后由 InsideScale 进行缩放。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 UnroundedInsideTessFactors 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

Process2DQuadTessFactorsMin 函数

项目 • 2022/09/24

为象限补丁生成更正的细化因子。

语法

syntax

```
void Process2DQuadTessFactorsMin(
  in  float4 RawEdgeFactors,
  in  float2 InsideScale,
  out float4 RoundedEdgeTessFactors,
  out float2 RoundedInsideTessFactors,
  out float2 UnroundedInsideTessFactors
);
```

参数

RawEdgeFactors [in]

类型： **float4**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float2**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float4**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float2**

细化器阶段为内边缘计算的舍入细化因子。

UnroundedInsideTessFactors [out]

类型： **float2**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为四个补丁生成更正的细化因子，将内部细化因子计算为边缘分割因子的最小值。 使用域的相对端的最小值来独立计算你和 V 内部分割因子，然后由 InsideScale 进行缩放。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 UnroundedInsideTessFactors 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessIsolineTessFactors 函数

项目 • 2022/09/22

为 isoline 生成舍入细化因子。

语法

syntax

```
void ProcessIsolineTessFactors(
  in  float RawDetailFactor,
  in  float RawDensityFactor,
  out float RoundedDetailFactor,
  out float RoundedDensityFactor
);
```

参数

RawDetailFactor [in]

类型： **float**

所需的详细信息因子。

RawDensityFactor [in]

类型： **float**

所需的密度因子。

RoundedDetailFactor [out]

类型： **float**

固定到可被分割器使用的范围的圆圆细节因子。

RoundedDensityFactor [out]

类型： **float**

细化器可以使用固定到 rangethat 的圆密度因子。

返回值

此函数不返回值。

注解

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessQuadTessFactorsAvg 函数

项目 • 2022/09/22

为象限补丁生成更正的细化因子。

语法

syntax

```
void ProcessQuadTessFactorsAvg(
  in  float4 RawEdgeFactors,
  in  float InsideScale,
  out float4 RoundedEdgeTessFactors,
  out float2 RoundedInsideTessFactors,
  out float2 UnroundedInsideTessFactors
);
```

参数

RawEdgeFactors [in]

类型： **float4**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float4**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float2**

细化器阶段为内边缘计算的舍入细化因子。

UnroundedInsideTessFactors [out]

类型： **float2**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为四个补丁生成更正的细化因子，将内部细化因子计算为边缘分割因子的平均值。 内部细化因子将是由 InsideScale 缩放的所有四个边缘的平均值确定的相同值。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 UnroundedInsideTessFactors 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessQuadTessFactorsMax 函数

项目 • 2022/09/22

为象限补丁生成更正的细化因子。

语法

syntax

```
void ProcessQuadTessFactorsMax(
  in  float4 RawEdgeFactors,
  in  float InsideScale,
  out float4 RoundedEdgeTessFactors,
  out float2 RoundedInsideTessFactors,
  out float2 UnroundedInsideTessFactors
);
```

参数

RawEdgeFactors [in]

类型： **float4**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float4**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float2**

细化器阶段为内边缘计算的舍入细化因子。

UnroundedInsideTessFactors [out]

类型： **float2**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为四个补丁生成更正的细化因子，将内部细化因子计算为边缘分割因子的最大值。 内部 tess 因子将由 InsideScale 缩放的所有四个边缘的最大值确定的相同值。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 *UnroundedInsideTessFactors* 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessQuadTessFactorsMin 函数

项目 • 2022/09/24

为象限补丁生成更正的细化因子。

语法

syntax

```
void ProcessQuadTessFactorsMin(
  in  float4 RawEdgeFactors,
  in  float InsideScale,
  out float4 RoundedEdgeTessFactors,
  out float2 RoundedInsideTessFactors,
  out float2 UnroundedInsideTessFactors
);
```

参数

RawEdgeFactors [in]

类型： **float4**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float4**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float2**

细化器阶段为内边缘计算的舍入细化因子。

UnroundedInsideTessFactors [out]

类型： **float2**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为四个补丁生成更正的细化因子，将内部细化因子计算为边缘分割因子的最小值。 内部 tess 因子将由 InsideScale 缩放的所有四个边缘的最小值确定的相同值。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 UnroundedInsideTessFactors 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessTriTessFactorsAvg 函数

项目 • 2022/09/23

为三个修补程序生成更正的细化因子。

语法

syntax

```
void ProcessTriTessFactorsAvg(
  in  float3 RawEdgeFactors,
  in  float InsideScale,
  out float3 RoundedEdgeTessFactors,
  out float RoundedInsideTessFactor,
  out float UnroundedInsideTessFactor
);
```

parameters

RawEdgeFactors [in]

类型： **float3**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float3**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactor [out]

类型： **float**

细化器阶段计算的细化因子，并舍入。

UnroundedInsideTessFactor [out]

类型： **float**

由分割阶段计算的原始、未得到的 UV 分割因子。

返回值

此函数不返回值。

注解

为三个补丁生成更正的细化因子，将内部细化因子计算为边缘细化因子的平均值，然后由 InsideScale 进行缩放。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 UnroundedInsideTessFactor 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessTriTessFactorsMax 函数

项目 • 2022/09/22

为三个修补程序生成更正后的细化因子。

语法

syntax

```
void ProcessTriTessFactorsMax(
  in  float3 RawEdgeFactors,
  in  float InsideScale,
  out float3 RoundedEdgeTessFactors,
  out float RoundedInsideTessFactor,
  out float UnroundedInsideTessFactor
);
```

parameters

RawEdgeFactors [in]

类型： **float3**

边缘分割因素已传递到细化器阶段。

InsideScale [in]

类型： **float**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float3**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactor [out]

类型： **float**

细化器阶段计算的细化因子，并舍入。

UnundedInsideTessFactor [out]

类型： **float**

由分割阶段计算的原始、未得到的 UV 细化因子。

返回值

此函数不返回值。

注解

为三个补丁生成更正后的细化因子，将内部细化因子计算为边缘细化因子的最大值，然后由 InsideScale 缩放。 然后，根据分区模式对结果进行舍入，但未解决的结果可以使用 UnroundedInsideTessFactor 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

ProcessTriTessFactorsMin 函数

项目 • 2022/09/22

为三个修补程序生成更正的细化因子。

语法

syntax

```
void ProcessTriTessFactorsMin(
  in  float3 RawEdgeFactors,
  in  float InsideScale,
  out float3 RoundedEdgeTessFactors,
  out float RoundedInsideTessFactors,
  out float UnroundedInsideTessFactors
);
```

参数

RawEdgeFactors [in]

类型： **float3**

边缘分割因子，传递到分割器阶段。

InsideScale [in]

类型： **float**

应用于由分割阶段计算的 UV 细化因子的刻度因子。 InsideScale 允许的范围为 0.0 到 1.0。

RoundedEdgeTessFactors [out]

类型： **float3**

细化器阶段计算的舍入边缘细化因子。

RoundedInsideTessFactors [out]

类型： **float**

细化器阶段为内边缘计算的舍入细化因子。

UnroundedInsideTessFactors [out]

类型： **float**

细化因子由细化器阶段计算的内边缘。

返回值

此函数不返回值。

注解

为三个补丁生成更正的细化因子，将内部细化因子计算为边缘细化因子的最小值，然后由 InsideScale 进行缩放。 然后，将根据分区模式舍入结果，但未得到的结果可以使用 UnroundedInsideTessFactor 参数进行舍入。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	几何结构	像素	计算
	x				

另请参阅

内部函数

着色器模型 5

radians

项目 • 2022/09/24

将指定的值从度转换为弧度。

***ret* radians (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

从度转换为弧度的 *x* 参数。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 1 (DirectX HLSL) 和更高着色器模型	是

另请参阅

DirectX HLSL) 内部函数 (

rcp

项目 • 2022/09/23

计算一个快速、近似、每分量倒数。

***ret* rcp (*x*)**

参数

X

[in]输入值。

返回值

x 参数的倒数。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float 或 double	any
Ret	与输入 *x* 相同	float 或 double	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 及更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

reflect

项目 • 2022/09/22

使用事件光线和表面法线返回反射矢量。

ret* reflect (*i* , *n)

参数

项	说明
我	[in]浮点事件向量。
n	[in]浮点法向量。

返回值

浮点反射向量。

注解

此函数使用以下公式计算反射矢量：v = i - 2 * n * 点 (i n) 。

类型说明

名称	模板类型	组件类型	大小
i	向量	float	any
n	向量	float	与*输入 i* 相同的维度 (s)
Ret	向量	float	与*输入 i* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 1 (DirectX HLSL) 和更高的着色器模型	是

另请参阅

DirectX HLSL (内部函数)

折射

项目 • 2022/09/22

使用输入光线、表面法线和折射索引返回折射矢量。

***ret* refract (*i*， *n*， ？)**

参数

项	说明
我	[in]浮点光线方向向量。
n	[in]浮点表面法向量。
？	[in]浮点折射索引标量。

返回值

浮点折射向量。 如果输入射线 i 和表面正态 n 之间的角度对于给定的折射索引而言太大，则返回值 (0，0，0) 。

类型说明

名称	模板类型	组件类型	大小
i	**向量**	**float**	any
n	**向量**	**float**	与*输入 i* 相同的维度 (s)
？	**标量**	**FLOAT**	1
refraction vector	**向量**	**float**	与*输入 i* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是

着色器模型	支持
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

reversebits 函数

项目 • 2022/09/22

按组件反转位的顺序。

语法

syntax

```
uint reversebits(
  in uint value
);
```

参数

value [in]

类型： **uint**

输入值。

返回值

类型： **uint**

输入值，并反转位顺序。

注解

以下重载版本也可用：

syntax

```
uint2 reversebits(uint2 value);
uint3 reversebits(uint3 value);
uint4 reversebits(uint4 value);
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5 和更高着色器模型	是

以下类型的着色器支持此函数：

顶点	外壳	域	Geometry	像素	计算
x	x	x	x	x	x

另请参阅

内部函数

着色器模型 5

round

项目 • 2022/09/22

将指定的值舍入到最接近的整数。 一半的案例被舍入到最接近的偶数。

***ret* round (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数，舍入到浮点类型内最接近的整数。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

要求

要求	Value

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

rsqrt

项目 • 2022/09/22

返回指定值的平方根的对等值。

***ret* rsqrt (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x 参数的平方根的对等。

注解

此函数使用以下公式：1/ **sqrt** (*x*) 。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

饱和 (HLSL 参考)

项目 • 2022/09/22

将指定值固定在 0 到 1 的范围内。

***ret* 饱和 (*x*)**

参数

项	说明
X	[in]指定的值。

返回值

x *参数*，固定在 0 到 1 的范围内。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 1 (DirectX HLSL) 和更高的着色器模型	是

另请参阅

DirectX HLSL (内部函数)

签名

项目 • 2022/09/23

返回 *x* 的符号。

***重新* 签名 (*x*)**

参数

项	描述
X	[in]输入值。

返回值

如果 *x* 小于零，则返回 -1;如果 *x* 等于零，则为 0;如果 *x* 大于零，则为 1。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	**float**、 **int**	any
Ret	与输入 *x* 相同	**int**	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1和ps_1_4)

另请参阅

DirectX HLSL) 内部函数 (

sin

项目 • 2022/09/23

返回指定值的正弦值。

***ret* sin (*x*)**

参数

项	说明
X	[in]指定的值，以弧度为单位。

返回值

x 参数的正弦值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

sincos

项目 • 2022/09/22

返回 x 的正弦和余弦值。

sincos (*x*， out *s*， out *c*)

参数

项	描述
X	[in]指定的值（以弧度为单位）。
s	[out]返回 x 的正弦值。
C	[out]返回 x 的余弦值。

返回值

无。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
s	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
c	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

sinh

项目 • 2022/09/23

返回指定值的双曲正弦值。

***ret* sinh (*x*)**

参数

项	说明
X	[in]指定的值（以弧度为单位）。

返回值

x 参数的双曲正弦值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	**浮动**	any
Ret	与输入 *x* 相同	FLOAT	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

smoothstep

项目 • 2022/09/22

如果 *x* 在 [*min*， *max*] 范围内，则返回介于 0 和 1 之间的平滑隐士内插。

***ret* smoothstep (*min*， *max*， *x*)**

参数

项	说明
min	[in] *x* 参数的最小范围。
麦克斯	[in] *x* 参数的最大范围。
X	[in]要内插的指定值。

返回值

如果 *x* 小于 *最小值*，则返回 0;如果 *x* 大于 *最大值*，则为 1;否则，如果 *x* 位于 [*min*，*max*] 范围内，则值为 0 到 1。

备注

使用 **平滑步骤** HLSL 内部函数在两个值之间创建平滑转换。 例如，可以使用此函数平滑混合两种颜色。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
min	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
max	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

sqrt

项目 • 2022/09/23

返回每个组件的指定浮点值的平方根。

***ret* sqrt (*x*)**

参数

项	说明
X	[in]指定的浮点值。

返回值

每个组件的 *x* 参数的平方根。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

步骤

项目 • 2022/09/23

比较两个值，根据哪个值返回 0 或 1。

***ret* 步骤 (*y*、 *x*)**

参数

项	说明
Y	[in]要比较的第一个浮点值。
X	[in]要比较的第二个浮点值。

返回值

如果 *x* 参数大于或等于 *y* 参数，则为 1;否则为 0。

注解

此函数使用以下公式： (*x*>= *y*) ？ 1 : 0. 该函数返回 0 或 1，具体取决于 *x* 参数是否大于 *y* 参数。 若要计算介于 0 和 1 之间的平滑内插，请使用 **平滑步骤** HLSL 内部函数。

类型说明

名称	模板类型	组件类型	大小
y	**标量**、 **矢量**或 **矩阵**	float	any
x	与输入 *y* 相同	float	与输入 *y* 相同的维度 (s)
Ret	与输入 *y* 相同	float	与输入 *y* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1和ps_1_4)

另请参阅

DirectX HLSL) 内部函数 (

tan

项目 • 2022/09/22

返回指定值的正切值。

***ret* tan (*x*)**

参数

项	说明
X	[in]指定的值（以弧度为单位）。

返回值

x 参数的正切值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL) 内部函数 (

tanh

项目 • 2022/09/23

返回指定值的双曲正切值。

***ret* tanh (*x*)**

参数

项	说明
X	[in]指定的值，以弧度为单位。

返回值

x 参数的双曲正切值。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	float	any
Ret	与输入 *x* 相同	float	与输入 *x* 相同的维度 (s)

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 2 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 1 (DirectX HLSL)	是 (vs_1_1仅)

另请参阅

DirectX HLSL (内部函数)

tex1D (HLSL 引用)

项目 • 2022/09/23

采样 1D 纹理。

ret tex1D (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler1D	1
t	in	**标量**	**FLOAT**	1
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅) ，但在编译时必须使用 旧编译选项 。
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	是 (像素着色器仅)

另请参阅

DirectX HLSL (内部函数)

tex1D (HLSL 引用) - 选择 mip 级别

项目 • 2022/09/23

使用渐变来采样 1D 纹理以选择 mip 级别。

ret tex1D (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]x 方向上表面几何图形的变化率。
ddy	[in]y 方向表面几何图形的变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler1D	1
t	in	**向量**	FLOAT	1
ddx	in	**向量**	FLOAT	1
ddy	in	**向量**	FLOAT	1
Ret	out	**向量**	float	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS 设置 D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

注解

当流控制存在于着色器中时，给定分支路径中请求的渐变计算的结果在相邻像素可能会向下移动单独的流控制路径时不明确。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造中发生渐变计算，该操作可能会因给定基元光栅化而有所不同。 如果具有分支属性的 **if** 语句的任一端都使用渐变函数，则可能会生成编译器错误。 查看语句 (DirectX HLSL) 。

另请参阅

DirectX HLSL (内部函数)

tex1Dbias

项目 • 2022/09/23

按 t.w 对 mip 级别进行偏差后，对 1D 纹理采样。

ret tex1Dbias (s， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler1D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

tex1Dgrad

项目 • 2022/09/22

使用渐变采样 1D 纹理以选择 mip 级别。

ret tex1Dgrad (s , t , ddx , ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]以 x 方向变化表面几何图形的速率。
ddy	[in]y 方向表面几何图形变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler1D	1
t	in	**向量**	FLOAT	1
ddx	in	**向量**	FLOAT	1
ddy	in	**向量**	FLOAT	1
Ret	out	**向量**	float	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS设置D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

注解

当流控制存在于着色器中时，给定分支路径中请求的渐变计算结果不明确，当相邻像素可能会向下走单独的流控制路径时。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造内的某个位置进行渐变计算，该操作可能会因给定基元光栅化而有所不同。 如果具有分支属性的 **if** 语句的任一端都使用渐变函数，则可能会生成编译器错误。 查看 语句是否 (DirectX HLSL) 。

另请参阅

DirectX HLSL) 内部函数 (

tex1Dlod

项目 • 2022/09/22

使用 mipmap 对 1D 纹理采样。 mipmap LOD 在 t.w 中指定。

ret tex1Dlod (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler1D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

tex1Dproj

项目 • 2022/09/23

使用投影除法对 1D 纹理采样;查找之前，纹理坐标被 t.w 除以。

ret tex1Dproj (s， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler1D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

tex2D (HLSL 参考)

项目 • 2022/09/24

对二维纹理进行采样。

ret tex2D (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler2D	1
t	in	**向量**	**浮动**	2
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅) ，但在编译时必须使用 旧编译选项 。
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	是 (像素着色器仅)

另请参阅

DirectX HLSL (内部函数)

tex2D (HLSL 引用) - 选择 mip 级别

项目 • 2022/09/24

使用渐变来采样 2D 纹理以选择 mip 级别。

ret tex2D (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]x 方向上表面几何图形的变化率。
ddy	[in]y 方向表面几何图形的变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler2D	1
t	in	**向量**	**float**	2
ddx	in	**向量**	**float**	2
ddy	in	**向量**	**float**	2
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS 设置 D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

注解

当流控制存在于着色器中时，给定分支路径中请求的渐变计算的结果在相邻像素可能会向下移动单独的流控制路径时不明确。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造中发生渐变计算，该操作可能会因给定基元光栅化而有所不同。 如果具有分支属性的 **if** 语句的任一端都使用渐变函数，则可能会生成编译器错误。 查看语句 (DirectX HLSL) 。

另请参阅

DirectX HLSL (内部函数)

tex2Dbias

项目 • 2022/09/23

通过 t.w 对 mip 级别进行偏差后，对 2D 纹理采样。

ret tex2Dbias (s， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler2D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL (内部函数)

tex2Dgrad

项目 • 2022/09/23

使用渐变来采样 2D 纹理以选择 mip 级别。

ret tex2Dgrad (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]x 方向上表面几何图形的变化率。
ddy	[in]y 方向表面几何图形的变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler2D	1
t	in	**向量**	**float**	2
ddx	in	**向量**	**float**	2
ddy	in	**向量**	**float**	2
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS 设置 D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

注解

当流控制存在于着色器中时，给定分支路径中请求的渐变计算的结果在相邻像素可能会向下移动单独的流控制路径时不明确。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造中发生渐变计算，该操作可能会因给定基元光栅化而有所不同。 如果具有分支属性的 **if** 语句的任一端都使用渐变函数，则可能会生成编译器错误。 查看语句 (DirectX HLSL) 。

另请参阅

DirectX HLSL (内部函数)

tex2Dlod

项目 • 2022/12/14

使用 mipmap 对 2D 纹理进行采样。 mipmap LOD 在 t.w 中指定。

ret tex2Dlod (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler2D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 3 (DirectX HLSL) 和更高的着色器模型	是
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

注解

从 Direct3D 10 开始，可以使用新的 HLSL 语法访问纹理和其他资源。 可以将内部样式的纹理查找函数（如 **tex2Dlod**）替换为更面向对象的样式。 在此面向对象的样式中，纹理与采样器分离，并具有加载和采样的方法。

若要采样 2D 纹理，而不是使用 **tex2Dlod** ，如以下代码所示：

```
sampler S;
...
color = tex2Dlod(S, Location);
```

使用**纹理对象的**SampleLevel 方法，如以下代码所示：

```
Texture2D MyTexture;
SamplerState MySampler;
...
color = MyTexture.SampleLevel(MySampler, Location, LOD);
```

若要将内部样式的纹理查找函数（如 **tex2Dlod**）与 着色器模型 4 及更高版本配合使用，请使用 D3DCOMPILE_ENABLE_BACKWARDS_COMPATIBILITY 进行编译。 但是，如果想要将着色器模型 4 及更高版本 (甚至 *_4_0_level_9_*) 与面向对象的较新的样式代码一起，请迁移到较新的 HLSL 语法。

另请参阅

DirectX HLSL (内部函数)

tex2Dproj

项目 • 2022/09/23

使用投影除法对二维纹理采样;查找之前，纹理坐标被 t.w 除以。

ret tex2Dproj (s， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler2D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

tex3D (HLSL 参考)

项目 • 2022/09/23

采样 3D 纹理。

ret tex3D (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler3D	1
t	in	**向量**	**float**	3
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是的 (像素着色器仅) ，但编译时必须使用 旧编译选项 。
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	是 (仅像素着色器)

另请参阅

DirectX HLSL) 内部函数 (

tex3D (HLSL 引用) - 选择 mip 级别

项目 • 2022/09/23

使用渐变采样 3D 纹理以选择 mip 级别。

ret tex3D (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]x 方向上表面几何图形的变化率。
ddy	[in]y 方向表面几何图形的变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler3D	1
t	in	**向量**	**float**	3
ddx	in	**向量**	**float**	3
ddy	in	**向量**	**float**	3
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS 设置 D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

注解

当流控制存在于着色器中时，给定分支路径中请求的渐变计算的结果在相邻像素可能会向下移动单独的流控制路径时不明确。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造中发生渐变计算，该操作可能会因给定基元光栅化而有所不同。 如果具有分支属性的 **if** 语句的任一端都使用渐变函数，则可能会生成编译器错误。 查看语句 (DirectX HLSL) 。

另请参阅

DirectX HLSL (内部函数)

tex3Dbias

项目 • 2022/09/22

通过 t.w 对 mip 级别进行偏差后，对 3D 纹理采样。

ret tex3Dbias (s， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler3D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL (内部函数)

tex3Dgrad

项目 • 2022/09/22

使用渐变采样 3D 纹理以选择 mip 级别。

ret tex3Dgrad (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]x 方向上表面几何图形的变化率。
ddy	[in]y 方向表面几何图形的变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler3D	1
t	in	**向量**	**FLOAT**	3
ddx	in	**向量**	**FLOAT**	3
ddy	in	**向量**	**FLOAT**	3
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS 设置 D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

注解

当流控制存在于着色器中时，给定分支路径中请求的渐变计算的结果在相邻像素可能会向下移动单独的流控制路径时不明确。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造中发生渐变计算，该操作可能会因给定基元光栅化而有所不同。 如果具有分支属性的 **if** 语句的任一端都使用渐变函数，则可能会生成编译器错误。 查看语句 (DirectX HLSL) 。

另请参阅

DirectX HLSL (内部函数)

tex3Dlod

项目 • 2022/09/23

使用 mipmap 对 3D 纹理采样。 mipmap LOD 在 t.w 中指定。

ret tex3Dlod (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler3D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

tex3Dproj

项目 • 2022/09/22

使用投影除法对三维纹理进行采样;查找之前，纹理坐标被 t.w 除以。

```
ret tex3Dproj (s， t)
```

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	sampler3D	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

texCUBE (HLSL 参考)

项目 • 2022/09/22

采样多维数据集纹理。

ret texCUBE (s , t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	samplerCUBE	1
t	in	**向量**	**float**	3
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	是 (仅像素着色器)

另请参阅

DirectX HLSL) 内部函数 (

texCUBE (HLSL 参考) - 选择 mip 级别

项目 • 2022/09/22

使用渐变采样多维数据集纹理以选择 mip 级别。

ret texCUBE (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]以 x 方向变化表面几何图形的速率。
ddy	[in]y 方向表面几何图形变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	samplerCUBE	1
t	in	**向量**	**float**	3
ddx	in	**向量**	**float**	3
ddy	in	**向量**	**float**	3
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS设置D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

另请参阅

DirectX HLSL) 内部函数 (

texCUBEbias

项目 • 2022/12/14

按 t.w 对 mip 级别进行偏差后对多维数据集纹理采样。

ret texCUBEbias (s ， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	samplerCUBE	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	是 (仅像素着色器)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

texCUBEgrad

项目 • 2022/09/23

使用渐变来采样多维数据集纹理以选择 mip 级别。

ret texCUBEgrad (s， t， ddx， ddy)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。
ddx	[in]x 方向上表面几何图形的变化率。
ddy	[in]y 方向表面几何图形的变化率。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	samplerCUBE	1
t	in	**向量**	**FLOAT**	3
ddx	in	**向量**	**FLOAT**	3
ddy	in	**向量**	**FLOAT**	3
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 1 (DirectX HLSL)	否

1. 对重大代码重新排序是为了在流控制之外移动渐变计算。
2. 如果使用 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS 设置 D3DPSHADERCAPS2_0上限，编译器会将此函数映射到 texldd。

另请参阅

DirectX HLSL (内部函数)

texCUBElod

项目 • 2022/09/22

使用 mipmap 对多维数据集纹理采样。 mipmap LOD 在 t.w 中指定。

ret texCUBElod (s ， t)

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	samplerCUBE	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (仅像素着色器)
着色器模型 3 (DirectX HLSL)	是 (仅像素着色器)
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL) 内部函数 (

texCUBEproj

项目 • 2022/09/22

使用投影除法对多维数据集纹理进行采样;在查找发生之前，纹理坐标除以 t.w。

```
ret texCUBEproj (s ， t)
```

参数

项	说明
s	[in]采样器状态。
t	[in]纹理坐标。

返回值

纹理数据的值。

类型说明

名称	输入/输出	模板类型	组件类型	大小
s	in	**对象**	samplerCUBE	1
t	in	**向量**	**float**	4
Ret	out	**向量**	**float**	4

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 4	是 (像素着色器仅)
着色器模型 3 (DirectX HLSL)	是 (像素着色器仅)
着色器模型 2 (DirectX HLSL)	是 (像素着色器仅)

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

请参阅

DirectX HLSL (内部函数)

transpose

项目 • 2022/09/23

转置指定的输入矩阵。

ret 转置 (*x*)

参数

项	说明
X	[in]指定的矩阵。

返回值

x 参数的转置值。

注解

如果源矩阵的维度为*行*，*列*，则生成的矩阵为*列*。

类型说明

名称	模板类型	组件类型	大小
x	矩阵	float、 int、 bool	any
Ret	矩阵	float、 int、 bool	rows = 与输入 *x* 相同的列数，列 = 与输入 *x* 相同的行数

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 1 (DirectX HLSL) 和更高着色器模型	是

另请参阅

DirectX HLSL) 内部函数 (

trunc

项目 • 2022/09/23

截断整数组件的浮点值。

ret trunc (*x*)

参数

项	说明
X	[in]指定的输入。

返回值

截断为整数组件的输入值。

注解

此函数将浮点值截断到整数组件。 给定浮点值 1.6，截断函数将返回 1.0，其中，当 **轮 (DirectX HLSL)** 函数将返回 2.0。

类型说明

名称	模板类型	组件类型	大小
x	**标量**、 **矢量**或 **矩阵**	FLOAT	any
Ret	与输入 x 相同的类型	FLOAT	与输入 x 相同的维度 ()

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 1 (DirectX HLSL) 和更高着色器模型	是

要求

要求	Value
标头	Corecrt_math.h

另请参阅

DirectX HLSL) 内部函数 (

Asm 着色器参考

项目 • 2022/09/22

着色器驱动可编程图形管道。

顶点着色器参考

- vs_1_1
- vs_2_0
- vs_2_x
- vs_3_0

顶点着色器差异 汇总顶点着色器版本之间的差异。

像素着色器参考

- ps_1_1、ps_1_2、ps_1_3、ps_1_4
- ps_2_0
- ps_2_x
- ps_3_0

像素着色器差异 汇总了像素着色器版本之间的差异。

着色器模型 4 和 5 参考

着色器模型 4 程序集和着色器模型 5 程序集部分介绍了着色器模型 4 和 5 支持的说明。

程序集着色器中常量寄存器的行为

在程序集着色器中设置常量寄存器有两种方法：

- 使用 def* 指令之一在程序集代码中声明着色器常量。
- 使用 Set***ShaderConstant* API 方法之一。

Direct3D 9 着色器常量

在 Direct3D 9 中，给定着色器中定义的常量生存期仅限于仅执行该着色器 (且不可重写) 。 Direct3D 9 中定义的常量在着色器之外没有副作用。

下面是使用 Direct3D 9 的示例：

```
Given:
    Create shader1 which references c4 and defines it with the def
instruction

Scenario 1:
    Call Set***Shader shader1
    Call Set***ShaderConstant* to set c4
    Call Draw
    Result: The shader will see the def'd value in c4


Given:
    Scenario 1 has just completed
    Create shader2 (which references c4 but does not use the def instruction
      to define it)

Scenario 2:
    Call Set***Shader shader2
    Call Draw
    Result: The shader will see the value last set in c4 by
     Set***ShaderConstant* in scenario 1. This is because shader 2
     didn't def c4.
```

在 Direct3D 9 中，调用 Get***ShaderConstant* 将仅检索通过 Set***ShaderConstant*设置的常量值。

Direct3D 8 着色器常量

此行为在 Direct3D 8.x 中有所不同。

```
Given:
    Create shader1 which references c4 and defines it with the def
instruction

Scenario 1 (repeated with Direct3D 8):
    Call Set***Shader with shader1
    Call Set***ShaderConstant to set c4
    Call Draw
    Result: The shader will see the value in c4 from Set***ShaderConstant
```

在 Direct3D 8.x Set***ShaderConstant 中立即生效。 请考虑以下方案：

```
```

```
Given:
    Create shader1 which references c4 and defines it with the def
instruction

Scenario 3:
    Call Set***Shader with shader1
    Call Draw
    Result: The shader will see the def'd value in c4

Given:
    Scenario 3 has just completed
    Create shader2 (which references c4 but does not use the def instruction
      to define it)

Scenario 4 :
    Call Set***Shader with shader2
    Call Draw
    Result: The shader will see the def'd value in c4 (set by def in shader
1)
```

不可取的结果是，着色器设置的顺序可能会影响各个着色器观察到的行为。

着色器驱动程序模型要求

Direct3D 9 接口仅限于设备驱动程序接口， (DirectX 7 级别及更高级别的 DDI) 驱动程序。 若要检查 DDI 级别，请运行 DirectX 诊断工具 并检查保存的文本文件。

有关参考，Direct3D 8 接口仅适用于 DirectX 6 级别及更高级别的 DDI 驱动程序。

着色器二进制格式

着色器指令流的按位布局在 D3d9types.h 中定义。 如果要设计自己的着色器编译器或构造工具，并且想要详细了解着色器令牌流，请参阅 Direct3D 9 驱动程序开发工具包 (DDK) 。

类似于 C 的着色器语言

请参阅 HLSL 参考 来体验类似于 C 的着色器语言。

相关主题

HLSL 参考

顶点着色器

项目 • 2022/09/22

- vs_1_1
- vs_2_0
- vs_2_x
- vs_3_0

- 顶点着色器差异

- 顶点着色器指令
- 顶点着色器寄存器

若要查看着色器模型 3 中引入的最新功能，请参阅：

- 着色器模型 3 (DirectX HLSL)

相关主题

Asm 着色器参考

vs_1_1

项目 • 2022/12/14

可编程顶点着色器由一组对顶点数据进行操作的指令组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

- 说明 - vs_1_1 包含可用说明的列表。
- 寄存器 - vs_1_1 列出顶点着色器 ALU 使用的不同类型的寄存器。
- 顶点着色器寄存器修饰符 用于修改指令的工作方式。
- 顶点着色器源寄存器修饰符 在指令运行之前更改源寄存器数据。
- 源寄存器重排 可控制哪些寄存器组件是读取、复制或写入的。
- 目标注册掩码 确定要写入的目标寄存器的哪些组件。

相关主题

顶点着色器

vs_2_0

项目 • 2022/09/22

可编程顶点着色器由一组对顶点数据进行操作的指令组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

- 说明 - vs_2_0 包含可用说明的列表。
- 寄存器 - vs_2_0 列出顶点着色器 ALU 使用的不同类型的寄存器。
- 顶点着色器寄存器修饰符 用于修改指令的工作方式。
- 指令运行前，顶点着色器源寄存器修饰符会更改源寄存器数据。
- 源寄存器重排 可更好地控制哪些寄存器组件是读取、复制或写入的。
- 目标寄存器掩码 确定要写入的目标寄存器的哪些组件。

指令计数

每个顶点着色器最多可以存储 256 条指令。 由于循环/代表支持 () ，运行指令的数量可能要高得多，并且受 D3DCAPS9 限制。MaxVShaderInstructionsExecuted，至少应0xFFFF。

相关主题

顶点着色器

vs_2_x

项目 • 2022/09/22

可编程顶点着色器由一组对顶点数据进行操作的指令组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

顶点着色器版本vs_2_x扩展vs_2_0支持的功能集。 每个附加功能由 D3DCAPS9 结构中的 D3DCAPS20CAPS 中的相应上限表示。 若要使用这些上限表示的任何增强功能，必须将顶点着色器版本指定为vs_2_x。

- 说明 - vs_2_x 包含可用说明的列表。
- 寄存器 - vs_2_x 列出顶点着色器 ALU 使用的不同类型的寄存器。
- 顶点着色器寄存器修饰符 用于修改指令的工作方式。
- 指令运行前，顶点着色器源寄存器修饰符会更改源寄存器数据。
- 源寄存器重排 可更好地控制哪些寄存器组件是读取、复制或写入的。
- 目标寄存器掩码 确定要写入的目标寄存器的哪些组件。

新功能

新功能如下所示：

动态Flow控件

如果 D3DVS20CAPS> 0，则支持以下动态流控制指令：

- if_comp - vs
- break - vs
- break_comp - vs

如果还设置了 D3DVS20CAPS ，则支持以下其他流控制指令：

- setp_comp - vs
- 如果 pred - vs
- callnz pred - vs
- breakp - vs

动态流控制深度的值范围为 0 到 24，等于动态流控制指令的嵌套深度， (请参阅Flow控制嵌套限制，了解详细信息) 。 如果此上限为零，则设备不支持动态流控制指令。

临时寄存器数

D3DVS20CAPS 表示设备支持的 临时寄存器数。 此上限的值范围为 12 到 32。

静态Flow控制嵌套深度

D3DVS20CAPS 表示两种类型的静态流控制指令的嵌套深度：loop - vsrep/- vscallnz bool - vscallnz/bool - vs/. loop - vs/rep - vs 指令可以嵌套到 D3DVS20CAPS 深度。 独立调用 - vs/callnz bool - 与指令可以嵌套到 D3DVS20CAPS 深度。 如果还设置了 D3DVS20CAPS，则 callnz pred - vs 将计入调用的嵌套深度 - vs/callnz bool - vs/if bool - vs (请参阅Flow控制嵌套限制的详细信息) 。

预测

如果设置了 D3DVS20CAPS ，则设备支持 setp_comp和 指令谓词。 如果 D3DVS20CAPS 也大于 0，则支持以下其他动态流控制指令：

- 如果 pred - vs
- callnz pred - vs
- breakp - vs

指令计数

每个顶点着色器最多可以存储 256 条指令。 由于循环/代表支持 () ，运行指令的数量可能要高得多，并且受 **D3DCAPS9** 限制，这至少应0xFFFF。

相关主题

顶点着色器

vs_3_0

项目 • 2022/09/23

可编程顶点着色器由一组对顶点数据进行操作的指令组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

顶点着色器版本vs_3_0扩展vs_2_x支持的功能集。 vs_2_X中需要设置上限的每个功能都可以在vs_3_0中使用，而无需上限。

- 说明 - vs_3_0 包含可用说明的列表。
- 寄存器 - vs_3_0 列出顶点着色器 ALU 使用的不同类型的寄存器。
- 顶点着色器寄存器修饰符 用于修改指令的工作方式。
- 指令运行前，顶点着色器源寄存器修饰符会更改源寄存器数据。
- 源寄存器重排 可更好地控制哪些寄存器组件是读取、复制或写入的。
- 目标寄存器掩码 确定要写入的目标寄存器的哪些组件。

新功能

以下部分列出了顶点着色器版本vs_3_0的新功能。

索引寄存器

在前面的着色器模型中，只能对常量寄存器库编制索引。 在此模型中，可以使用循环计数器寄存器 (aL) 为以下寄存器编制索引：

- 输入寄存器 (v#)
- 输出寄存器 (o#)

顶点纹理

此着色器模型支持使用 texldl 在顶点着色器中查找纹理。 顶点引擎具有四个纹理采样器阶段，（不同于位移地图采样器，像素引擎中的纹理采样器) 可用于在这些阶段设置纹理。 请参阅 DirectX HLSL) 中的vs_3_0 (顶点纹理 。

顶点流频率

此功能允许以与每个顶点一次不同的速率初始化输入寄存器的子集。 请参阅 绘制非索引几何图形。

着色器输出

与vs_2_0类似，着色器的输出可能因静态流控制而异。 请注意动态分支，因为这可能会导致着色器输出因顶点而异。 这将在不同硬件上产生不可预知的结果。

动态流控制

支持所有动态流控制指令。 允许的最大嵌套深度值为 24。 (有关详细信息，请参阅Flow控制嵌套限制。)

临时寄存器

总共支持 32 个临时寄存器， (r#) 。

静态Flow控件

循环的最大嵌套深度 - vsrep/- 与 4。 调用的最大嵌套深度 - vscallnz/bool - vscallnz/pred - vs 为 4。 对于 bool - vs，允许的最大嵌套深度值为 24。 (有关详细信息，请参阅Flow控制嵌套限制。)

预测

支持指令谓词。 使用 setp_comp - 与 设置谓词寄存器。

指令计数

允许每个顶点着色器从 512 到 **D3DCAPS9** 中的 MaxVertexShader30InstructionSlots 中的槽数。 由于循环/代表支持，运行指令的数量可能要高得多：但是，这由 D3DCAPS9 中 MaxVShaderInstructionsExecuted 封盖，至少应0xFFFF。

设备上限

如果顶点着色器 3_0 受支持，则硬件 (至少支持以下上限) ：

笔帽	功能
着色器上限	• DynamicFlowControlDepth 为 24 • NumTemps 为 32 • StaticFlowControlDepth 为 4 • 支持谓词。

笔帽	功能
GuardBandLeft、GuardBandTop、GuardBandRight、GuardBandBottom	8K
VertexShaderVersion	3_0
MaxVertexShaderConst	256
MaxVertexShader30InstructionSlots	512
雾支持	D3DPRASTERCAPS_FOGVERTEX
VertexTextureFilterCaps	• D3DPTFILTERCAPS_MINFPOINT • D3DPTFILTERCAPS_MAGFPOINT
D3DDEVCAPS2_VERTEXELEMENTSCANSHARESTREAMOFFSET	顶点声明中的顶点元素可以共享相同的流偏移量。
顶点格式	• D3DDECLTYPE_UBYTE4 • D3DDECLTYPE_UBYTE4N • D3DDECLTYPE_SHORT2N • D3DDECLTYPE_SHORT4N • D3DDECLTYPE_FLOAT16_2 • D3DDECLTYPE_FLOAT16_4

相关主题

顶点着色器

顶点着色器差异

项目 • 2022/09/22

指令槽

每个版本都支持不同的最大指令槽数。

版本	最大指令槽数
vs_1_1	128
vs_2_0	256
vs_2_x	256
vs_3_0	最低为 512，最大为 D3DCAPS9 中的槽数。MaxVertexShader30InstructionSlots。 请参阅 D3DCAPS9。

有关软件着色器限制的信息，请参阅 软件着色器。

Flow控制嵌套限制

- 请参阅Flow控制嵌套限制。

vs_1_1功能

新说明：

请参阅 说明 - vs_1_1。

新寄存器：

请参阅 寄存器 - vs_1_1。

vs_2_0功能

新功能：

- 静态流控制
- 地址寄存器的所有四个组件 (a0) 都可用。

新说明：

- 设置说明 - defb - vs， defi - vs
- 算术指令 - abs - vs， crs - vs， lrp - vs， mova - vs， nrm - vs， pow - vs， sgn - vs， sincos - vs
- 静态流控制指令 - 调用 - vs， callnz bool - vs， else - vs， endif - vs， endloop - vs， endrep - vs， if bool - vs， label - vs， loop - vs， rep - vs， ret - vs

新寄存器：

- 常量布尔寄存器 (b#)
- 常量整数寄存器 (i#)
- Loop计数器寄存器 (aL)

vs_2_x功能

新功能 (D3DCAPS9。VS20Caps) ：

- 动态流控制
- 动态和静态流控制指令的嵌套
- (r#) 增加 的临时寄存器数
- 预测

新说明：

- 动态流控制指令 - break - vs， break_comp - vs， breakp - vs， callnz pred - vs， if_comp - vs， if pred - vs， setp_comp - vs

新寄存器：

- 谓词寄存器 (p0)

vs_3_0功能

新功能：

- 纹理查找
- 可索引 输出寄存器 (o#)
- (r#) 临时寄存器数增加到 32

新说明：

- 设置说明 - dcl_samplerType (sm3 - vs asm)
- 纹理指令 - texldl - vs

新寄存器：

- Sampler (Direct3D 9 asm-vs) (s#)

相关主题

顶点着色器

顶点着色器指令

项目 • 2022/09/23

- 说明 - vs_1_1
- 说明 - vs_2_0
- 说明 - vs_2_x
- 说明 - vs_3_0

- 顶点着色器寄存器修饰符
- Flow控制嵌套限制

相关主题

顶点着色器

说明 - vs_1_1

项目 • 2022/09/23

本部分包含顶点着色器版本 1_1 指令的参考信息。

有多种类型的顶点着色器指令，如表中所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 安装程序 - 非算术指令。 每个着色器都必须有一个版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	新建
add - vs	添加两个向量	1		x	x
dcl_usage输入 (sm1、sm2、sm3 - vs asm)	声明输入顶点寄存器 (请参阅 寄存器 - vs_1_1)	0	x		x
def - vs	定义常量	0	x		x
dp3 - vs	三分量点积	1		x	x
dp4 - vs	四分量点积	1		x	x
dst - vs	计算距离向量	1		x	x
exp - vs	全精度 2 倍	10		x	x
expp - vs	部分精度 2x	1		x	x
frc - vs	小数部分	3		x	x
lit - vs	部分照明计算	1		x	x
log - vs	全精度 log一 (x)	10		x	x
logp - vs	部分精度 log一 (x)	1		x	x
m3x2 - vs	3x2 乘法	2		x	x
m3x3 - vs	3x3 乘法	3		x	x
m3x4 - vs	3x4 乘法	4		x	x

名称	说明	指令槽	设置	算术	新建
m4x3 - vs	4x3 乘法	3		x	x
m4x4 - vs	4x4 乘法	4		x	x
mad - vs	相乘和添加	1		x	x
max - vs	最大值	1		x	x
min - vs	最小值	1		x	x
mov - vs	移动	1		x	x
mul - vs	乘	1		x	x
nop - vs	无操作	1		x	x
rcp - vs	互惠	1		x	x
rsq - vs	倒方方根	1		x	x
sge - vs	大于或等于比较	1		x	x
slt - vs	小于比较	1		x	x
sub - vs	减	1		x	x
与	版本	0	x		x

相关主题

顶点着色器指令

说明 - vs_2_0

项目 • 2022/09/22

本部分包含顶点着色器版本 2_0 指令的参考信息。

有多种类型的顶点着色器指令，如表中所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 安装程序 - 非算术指令。 每个着色器都必须有一个版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- Flow控制 - 这些指令添加流控制功能，例如循环...endloop，如果...否则...endif - vs 和子例程调用。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	流量控制	新建
abs - vs	绝对值	1		x		x
add - vs	添加两个向量	1		x		
call - vs	调用子例程	2			x	x
callnz bool - vs	如果布尔寄存器不为零，则调用子例程	3			x	x
crs - vs	跨产品	2		x		x
dcl_usage输入 (sm1、sm2、sm3 - vs asm)	声明输入顶点寄存器 (请参阅寄存器 - vs_2_0)	0	x			
def - vs	定义常量	0	x			
defb - vs	定义布尔常量	0	x			x
defi - vs	定义整数常量	0	x			x
dp3 - vs	三分量点积	1		x		
dp4 - vs	四分量点积	1		x		
dst - vs	计算距离向量	1		x		

名称	说明	指令槽	设置	算术	流量控制	新建
else - vs	开始 其他 - 与 块	1			x	x
endif - vs	结束 一个 if bool - vs...else - vs block	1			x	x
endloop - vs	循环结束 - 与块	2			x	x
endrep - vs	重复块的结束	2			x	x
exp - vs	全精度 2 倍	1		x		
expp - vs	部分精度 2x	1		x		
frc - vs	小数部分	1		x		
if bool - vs	如果 bool - vs block (使用布尔条件)	3			x	x
label - vs	标签	0			x	x
lit - vs	部分照明计算	3		x		
log - vs	全精度 log一 (x)	1		x		
logp - vs	部分精度 log一 (x)	1		x		
loop - vs	Loop	3			x	x
lrp - vs	线性内插	2		x		x
m3x2 - vs	3x2 乘法	2		x		
m3x3 - vs	3x3 乘法	3		x		
m3x4 - vs	3x4 乘法	4		x		
m4x3 - vs	4x3 乘法	3		x		
m4x4 - vs	4x4 乘法	4		x		
mad - vs	相乘和添加	1		x		
max - vs	最大值	1		x		
min - vs	最小值	1		x		
mov - vs	移动	1		x		

名称	说明	指令槽	设置	算术	流量控制	新建
mova - vs	将数据从浮点寄存器移动到地址寄存器 (a0)	1		x		x
mul - vs	乘	1		x		
nop - vs	无操作	1		x		
nrm - vs	规范化 4D 向量	3		x		x
pow - vs	xy	3		x		x
rcp - vs	互惠	1		x		
rep - vs	Repeat	3			x	x
ret - vs	子例程或 main 的结尾	1			x	x
rsq - vs	倒数方根	1		x		
sge - vs	大于或等于比较	1		x		
sgn - vs	签名	3		x		x
sincos - vs	正弦和余弦	8		x		x
slt - vs	小于比较	1		x		
sub - vs	减	1		x		
与	版本	0	x			

相关主题

顶点着色器指令

说明 - vs_2_x

项目 • 2022/09/22

本部分包含顶点着色器版本2_x说明的参考信息。

有多种类型的顶点着色器指令，如表中所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 安装程序 - 非算术指令。 每个着色器都必须有一个版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- Flow控制 - 这些指令添加流控制功能，如循环 - vs...endloop - vs， if bool - vs...否则...endif 和子例程调用。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	流量控制	新建
abs - vs	绝对值	1		x		
add - vs	添加两个向量	1		x		
break - vs	中断 循环 - vs...endloop - vs 或 rep...endrep 块	1			x	x
break_comp - vs	有条件地突破 循环 - vs...endloop - vs 或 rep...endrep 块，带有比较	3			x	x
breakp - vs	中断 循环 - vs...endloop - vs 或 rep...endrep 块，基于谓词	3			x	x
call - vs	调用子例程	2			x	
callnz bool - vs	如果布尔寄存器不为零，则调用子例程	3			x	
callnz pred - vs	如果谓词寄存器不是零，则调用子例程	3			x	x
crs - vs	跨产品	2		x		
dcl_usage输入 (sm1、sm2、sm3 - vs asm)	声明输入顶点寄存器 (请参阅 寄存器 - vs_2_x)	0	x			
def - vs	定义常量	0	x			

名称	说明	指令槽	设置	算术	流量控制	新建
defb - vs	定义布尔常量	0	x			
defi - vs	定义整数常量	0	x			
dp3 - vs	三分量点积	1		x		
dp4 - vs	四分量点积	1		x		
dst - vs	计算距离向量	1		x		
else - vs	开始 其他 - 与 块	1			x	
endif - vs	结束 一个 if bool - vs...else - vs block	1			x	
endloop - vs	循环结束 - 与块	2			x	
endrep - vs	重复块的结束	2			x	
exp - vs	全精度 ^{2x}	1		x		
expp - vs	分部精度 ^{2x}	1		x		
frc - vs	小数部分	1		x		
if bool - vs	使用布尔条件) 开始 bool - vs block (	3			x	
if_comp - vs	使用 比较开始一个 if bool - vs block	3			x	x
如果 pred - vs	开始 一个布尔值 - 与 具有谓词条件的块	3			x	x
label - vs	标签	0			x	
lit - vs	部分照明计算	3		x		
log - vs	全精度 log一 (x)	1		x		
logp - vs	分精度 log一 (x)	1		x		
loop - vs	Loop	3			x	
lrp - vs	线性内插	2		x		
m3x2 - vs	3x2 乘法	2		x		
m3x3 - vs	3x3 乘法	3		x		
m3x4 - vs	3x4 乘法	4		x		
m4x3 - vs	4x3 乘法	3		x		

名称	说明	指令槽	设置	算术	流量控制	新建
m4x4 - vs	4x4 乘法	4		x		
mad - vs	乘法和添加	1		x		
max - vs	最大值	1		x		
min - vs	最小值	1		x		
mov - vs	移动	1		x		
mova - vs	将数据从浮点寄存器移到地址寄存器 (a0)	1		x		
mul - vs	乘	1		x		
nop - vs	无操作	1		x		
nrm - vs	规范化 4D 向量	3		x		
pow - vs	^{xy}	3		x		
rcp - vs	互惠	1		x		
rep - vs	Repeat	3			x	
ret - vs	子例程或 main 的结尾	1			x	
rsq - vs	倒方方根	1		x		
setp_comp - vs	设置谓词寄存器	1			x	x
sge - vs	大于或等于比较	1		x		
sgn - vs	签名	3		x		
sincos - vs	正弦和余弦	8		x		
slt - vs	小于比较	1		x		
sub - vs	减	1		x		
与	版本	0	x			

相关主题

顶点着色器指令

说明 - vs_3_0

项目 • 2022/09/23

本部分包含顶点着色器版本 3_0 指令的参考信息。

有多种类型的顶点着色器指令，如表中所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 安装程序 - 非算术指令。 每个着色器必须具有版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- 纹理 - 这些指令支持纹理地址查找。
- Flow控制 - 这些指令添加流控制，例如循环、重复和布尔 - 与...else...endif 比较。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
abs - vs	绝对值	1		x			
add - vs	添加两个向量	1		x			
break - vs	中断 循环 - vs...endloop - vs 或 rep...endrep 块	1				x	
break_comp - vs	有条件地中断 循环 - 与...endloop - vs 或 rep...endrep 块，带有比较	3				x	
breakp - vs	中断 循环 - vs...endloop - vs 或 rep...endrep 块，基于谓词	3				x	
call - vs	调用子例程	2				x	
callnz bool - vs	如果布尔寄存器不为零，则调用子例程	3				x	
callnz pred - vs	如果谓词寄存器不为零，则调用子例程	3				x	
crs - vs	跨产品	2		x			
dcl_usage输入 (sm1、sm2、sm3 - vs asm)	声明输入顶点寄存器 (请参阅 寄存器 - vs_3_0)	0	x				

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
dcl_samplerType (sm3 - vs asm)	声明采样器的纹理维度	0	x				x
def - vs	定义常量	0	x				
defb - vs	声明布尔常量	0	x				
defi - vs	声明整数常量	0	x				
dp3 - vs	三分量点产品	1		x			
dp4 - vs	四分量点产品	1		x			
dst - vs	距离	1		x			
else - vs	开始 其他 块	1				x	
endif - vs	结束 一个 if bool - vs...else 块	1				x	
endloop - vs	循环结束 - 与块	2				x	
endrep - vs	重复块的结束	2				x	
exp - vs	全精度 2^x	1		x			
expp - vs	分部精度 2^x	1		x			
frc - vs	小数部分	1		x			
if bool - vs	使用布尔条件) 开始 bool - vs block (	3				x	
if_comp - vs	使用 比较开始一个 if bool - vs block	3				x	
如果 pred - vs	开始 一个布尔值 - 与 具有谓词条件的块	3				x	
label - vs	标签	0				x	
lit - vs	计算照明	3		x			
log - vs	全精度 log一 (x)	1		x			
logp - vs	分精度 log一 (x)	1		x			
loop - vs	Loop	3				x	
lrp - vs	线性内插	2		x			

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
m3x2 - vs	3x2 乘法	2		x			
m3x3 - vs	3x3 乘法	3		x			
m3x4 - vs	3x4 乘法	4		x			
m4x3 - vs	4x3 乘法	3		x			
m4x4 - vs	4x4 乘法	4		x			
mad - vs	乘法和添加	1		x			
max - vs	最大值	1		x			
min - vs	最小值	1		x			
mov - vs	移动	1		x			
mova - vs	将数据从浮点寄存器移动到整数寄存器	1		x			
mul - vs	乘	1		x			
nop - vs	无操作	1		x			
nrm - vs	规范化	3		x			
pow - vs	xy	3		x			
rcp - vs	互惠	1		x			
rep - vs	Repeat	3				x	
ret - vs	子例程结束	1				x	
rsq - vs	倒数方根	1		x			
setp_comp - vs	设置谓词寄存器	1				x	
sge - vs	大于或等于比较	1		x			
sgn - vs	签名	3		x			
sincos - vs	正弦和余弦	8		x			
slt - vs	小于比较	1		x			
sub - vs	减	1		x			

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
texldl - vs	使用用户可调整的详细信息级别进行纹理加载	请参阅注释 1			x		x
与	版本	0	x				

注意：

- 如果纹理是多维数据集映射，则槽 = 5;否则槽 = 2

相关主题

顶点着色器指令

Flow控制嵌套限制

项目 • 2022/09/23

顶点着色器流控制指令有两个特殊限制。 嵌套深度限制可以相互调用的指令数。 此外，每个指令都有一个指令槽计数，该计数适用于着色器可以支持的最大指令数。

> **备注**
>
> 使用 *_4_0_level_9_x HLSL 着色器配置文件时，隐式使用 **着色器模型** 2.x 配置文件来支持支持 Direct3D 9 的硬件。 着色器模型 2.x 配置文件支持比 **着色器模型** 4.x 及更高版本的配置文件更有限的流控制行为。

每个指令vs_2_0的深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 下表显示了每个指令从现有深度添加或减去的深度计数：

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套	静态流计数
if bool - vs	0	0	0	0	1
if_comp - vs	不适用	不适用	不适用	不适用	不适用
如果 pred - vs	不适用	不适用	不适用	不适用	不适用
else - vs	0	0	0	0	1 (如果布尔 - 仅与)
endif - vs	-1	0	0	0	0
rep - vs	0	0	1	0	1
endrep - vs	0	0	-1	0	0
loop - vs	0	0	1	0	1
endloop - vs	0	0	-1	0	0
break - vs	不适用	不适用	不适用	不适用	不适用
break_comp - vs	不适用	不适用	不适用	不适用	不适用
breakp - vs	不适用	不适用	不适用	不适用	不适用
call - vs	0	0	0	1	1

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套	静态流计数
callnz bool - vs	0	0	0	1	1
callnz pred - vs	不适用	不适用	不适用	不适用	不适用
ret - vs	0	0	0	-1	0
setp_comp - vs	不适用	不适用	不适用	不适用	不适用

嵌套深度

嵌套深度定义可以相互调用的指令数。 每种类型的指令都有一个或多个嵌套限制：

指令类型	最大值
静态嵌套	仅受静态流计数的限制
动态嵌套	不适用
loop/rep 嵌套	1
调用嵌套	1
静态流计数	16

vs_2_x每个指令的深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 下表显示了每个指令从现有深度添加或减去的深度计数：

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
if bool - vs	1	0	0	0	1
if_comp - vs	0	1	0	0	0
如果 pred - vs	0	1	0	0	0
else - vs	0	0	0	0	1 (如果布尔 - 仅与)

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
endif - vs	-1 (如果布尔 - vs)	-1 ((如果 pred - vs 或 if_comp - vs)	0	0	0
rep - vs	0	0	1	0	1
endrep - vs	0	0	-1	0	0
loop - vs	0	0	1	0	1
endloop - vs	0	0	-1	0	0
break - vs	0	0	0	0	0
break_comp - vs	0	1, -1	0	0	0
breakp - vs	0	0	0	0	0
call - vs	0	0	0	1	1
callnz bool - vs	0	0	0	1	1
callnz pred - vs	0	1	0	1	0
ret - vs	0	-1 (callnz pred - vs)	0	-1	0
setp_comp - vs	0	0	0	0	0

嵌套深度

嵌套深度定义可以相互调用的指令数。 每种类型的指令都有一个或多个嵌套限制：

指令类型	最大值
静态嵌套	仅受静态流计数的限制
动态嵌套	0 或 24，请参阅 D3DCAPS9。VS20Caps.DynamicFlowControlDepth
loop/rep 嵌套	1 到 4，请参阅 D3DCAPS9。VS20Caps.StaticFlowControlDepth

指令类型	最大值
调用嵌套	1 到 4，请参阅 D3DCAPS9。VS20Caps.StaticFlowControlDepth (独立于循环/代表限制)
静态流计数	16

每个指令的深度计数，用于vs_2_sw

每个指令针对一个或多个嵌套深度限制进行计数。 下表显示了每个指令从现有深度添加或减去的深度计数：

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
if bool - vs	1	0	0	0	不适用
if_comp - vs	0	1	0	0	不适用
如果 pred - vs	0	1	0	0	不适用
else - vs	0	0	0	0	不适用
endif - vs	-1 (如果布尔 - vs)	-1 ((如果 pred - vs 或 if_comp - vs)	0	0	不适用
rep - vs	0	0	1	0	不适用
endrep - vs	0	0	-1	0	不适用
loop - vs	0	0	1	0	不适用
endloop - vs	0	0	-1	0	不适用
break - vs	0	0	0	0	不适用
break_comp - vs	0	1, -1	0	0	不适用
breakp - vs	0	0	0	0	不适用
call - vs	0	0	0	1	不适用
callnz bool - vs	0	0	0	1	不适用

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
callnz pred - vs	0	1	0	1	不适用
ret - vs	0	-1 (callnz pred - vs)	0	-1	不适用
setp_comp - vs	0	0	0	0	不适用

嵌套深度

嵌套深度定义可以相互调用的指令数。 每种类型的指令都有一个或多个嵌套限制：

指令类型	最大值
静态嵌套	24
动态嵌套	24
loop/rep 嵌套	4
调用嵌套	4
静态流计数	无限制

每个指令的深度计数vs_3_0

每个指令针对一个或多个嵌套深度限制进行计数。 下表显示了每个指令从现有深度添加或减去的深度计数：

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
if bool - vs	1	0	0	0	不适用
if_comp - vs	0	1	0	0	不适用
如果 pred - vs	0	1	0	0	不适用
else - vs	0	0	0	0	不适用

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
endif - vs	-1 (如果布尔 - vs)	-1 ((如果 pred - vs 或 if_comp - vs)	0	0	不适用
rep - vs	0	0	1	0	不适用
endrep - vs	0	0	-1	0	不适用
loop - vs	0	0	1	0	不适用
endloop - vs	0	0	-1	0	不适用
break - vs	0	0	0	0	不适用
break_comp - vs	0	1, -1	0	0	不适用
breakp - vs	0	0	0	0	不适用
call - vs	0	0	0	1	不适用
callnz bool - vs	0	0	0	1	不适用
callnz pred - vs	0	1	0	1	不适用
ret - vs	0	-1 (callnz pred - vs)	0	-1	不适用
setp_comp - vs	0	0	0	0	不适用

嵌套深度

嵌套深度定义可以相互调用的指令数。 每种类型的指令都有一个或多个嵌套限制：

指令类型	最大值
静态嵌套	24
动态嵌套	24
loop/rep 嵌套	4
调用嵌套	4

指令类型	最大值
静态流计数	无限制

vs_3_sw每个指令的深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 下表显示了每个指令从现有深度添加或减去的深度计数：

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
if bool - vs	1	0	0	0	不适用
if_comp - vs	0	1	0	0	不适用
如果 pred - vs	0	1	0	0	不适用
else - vs	0	0	0	0	不适用
endif - vs	-1 (如果布尔 - vs)	-1 ((如果 pred - vs 或 if_comp - vs)	0	0	不适用
rep - vs	0	0	1	0	不适用
endrep - vs	0	0	-1	0	不适用
loop - vs	0	0	1	0	不适用
endloop - vs	0	0	-1	0	不适用
break - vs	0	0	0	0	不适用
break_comp - vs	0	1, -1	0	0	不适用
breakp - vs	0	0	0	0	不适用
call - vs	0	0	0	1	不适用
callnz bool - vs	0	0	0	1	不适用
callnz pred - vs	0	1	0	1	不适用
ret - vs	0	-1 (callnz pred - vs)	0	-1	不适用

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套	静态流计数
setp_comp - vs	0	0	0	0	不适用

嵌套深度

嵌套深度定义可以相互调用的指令数。 每种类型的指令都有一个或多个嵌套限制：

指令类型	最大值
静态嵌套	24
动态嵌套	24
循环/rep 嵌套	4
调用嵌套	4
静态流计数	无限制

相关主题

顶点着色器指令

指令修饰符 (HLSL VS 参考)

项目 • 2022/09/23

指令修饰符在写入目标寄存器之前影响指令的结果。

_sat

在写入目标寄存器之前，（或固定将指令结果）到 [0，1] 范围的饱和。

例如：

```
add_sat dst, src0, src1
```

其中：

dst = clamp_between_0_and_1 (src0 + src1)

_sat指令修饰符不收取额外的指令槽费用。

如果受支持，_sat指令修饰符可与除 frc - vs、 sincos - vs 和 texldl - vs 以外的任何指令一起使用。

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
_sat					x	x

相关主题

顶点着色器指令

abs - vs

项目 • 2022/09/23

计算绝对值。

语法

abs dst , src

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
abs		x	x	x	x	x

此指令的工作方式如下。

```
dest.x = abs(src.x)
dest.y = abs(src.y)
dest.z = abs(src.z)
dest.w = abs(src.w)
```

相关主题

顶点着色器指令

add - vs

项目 • 2022/09/22

添加两个向量。

语法

```
add dst , src0 , src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
add	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x = src0.x + src1.x;
dest.y = src0.y + src1.y;
dest.z = src0.z + src1.z;
dest.w = src0.w + src1.w;
```

相关主题

顶点着色器指令

break - vs

项目 • 2022/12/14

在最接近的 endloop - vs 或 endrep - vs. 处中断当前循环。

语法

break

注解

以下版本支持此指令。

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
break			x	x	x	x

相关主题

顶点着色器指令

break_comp - vs

项目 • 2022/09/22

有条件地在最接近的 endloop - vs 或 endrep - vs. 上中断当前循环。

语法

break_comp src0、src1

其中：

- _comp是两个源寄存器之间的比较。 该参数可以是下列值之一：

语法	**比较**
_gt	大于
_lt	小于
_ge	大于或等于
_le	小于或等于
_eq	等于
_ne	不等于

- src0 是源寄存器。 需要复制重排才能选择单个组件。
- src1 是源寄存器。 需要复制重排才能选择单个组件。

注解

以下版本支持此指令。

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
break_comp			x	x	x	x

当比较为 true 时，它会中断当前循环，如下所示。

```
if (src0 comparison src1)
   jump to the corresponding endloop or endrep instruction;
```

相关主题

顶点着色器指令

breakp - vs

项目 • 2022/09/23

有条件地在最接近的 endloop - vs 或 endrep - vs. 上中断当前循环。使用谓词寄存器的一个组件作为条件来确定是否执行指令。

语法

breakp [!]P。{x|y|z|w}

其中：

- [!]可选的布尔 NOT。
- p0 是谓词寄存器。 请参阅 谓词寄存器。
- {x|y|z|w} 是 p0 上所需的复制重排。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
breakp			x	x	x	x

相关主题

顶点着色器指令

call - vs

项目 • 2022/09/22

执行对标有标签的指令的函数调用。

语法

call l#

其中 l# 是 一个标签 - 与 标记要调用的子例程的开头。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
call		x	x	x	x	x

此指令执行以下操作：

1. 下一个指令的推送地址到返回地址堆栈。
2. 从标签标记的指令继续执行。

在顶点着色器 2_0 中，不允许嵌套调用。

在顶点着色器2_x中，嵌套深度受 **D3DVSHADERCAPS2_0** 结构的 StaticFlowControlDepth 元素的限制。 有关详细信息，请参阅 **GetDeviceCaps**。

在顶点着色器 3_0 中，允许四个级别的调用嵌套。

仅允许转接呼叫。 这意味着，顶点着色器内的标签位置应在引用它的调用指令之后。

如果在循环中调用调用调用指令...endloop 块，Loop计数器寄存器 (aL) 的值可在子例程内访问。

如果子例程引用位于子例程外部的 Loop Counter Register (aL) ，则调用此子例程的每个实例都应被循环包围...endloop 块。

相关主题

顶点着色器指令

callnz bool - vs

项目 • 2022/09/23

如果不是零，则调用。 对标签索引标记的指令执行条件调用。

语法

```
callnz l# ,  [ ! ]B#
```

其中：

- 其中 l# 是一个 标签 - 与 标记要调用的子例程的开头。
- [!]是可选的布尔非修饰符。
- b# 标识 常量布尔寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
callnz bool		x	x	x	x	x

此指令执行以下操作：

```
if (specified boolean register is not zero)
{
    Push address of the next instruction to the return address stack.
    Continue execution from the instruction marked by the label.
}
```

此指令使用一个顶点着色器指令槽。

相关主题

顶点着色器指令

callnz pred - vs

项目 • 2022/09/24

如果不是零，则调用谓词。 对标签索引标记的指令执行条件调用。 谓词使用布尔值来确定是否不执行指令。

语法

```
callnz l# , [ ! ]P。{x|y|z|w}
```

其中：

- l# 是一个 标签 - 与 标记要调用的子例程的开头。
- [!]是可选的反修饰符。
- p0 是 谓词寄存器。
- {x|y|z|w} 是 p0 上所需的复制重排。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
callnz pred			x	x	x	x

此指令执行以下操作：

```
if (specified register component is not zero)
{
    Push address of the next instruction to the return address stack.
    Continue execution from the instruction marked by the label.
}
```

此指令使用一个顶点着色器指令槽。

相关主题

顶点着色器指令

crs - vs

项目 • 2022/09/22

使用右侧规则计算跨产品。

语法

crs dst、src0、src1

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

备注

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
Crs		x	x	x	x	x

此指令的工作方式如下。

```
dest.x = src0.y * src1.z - src0.z * src1.y;
dest.y = src0.z * src1.x - src0.x * src1.z;
dest.z = src0.x * src1.y - src0.y * src1.x;
```

对使用的一些限制：

- src0 不能与 dest 相同的寄存器。
- src1 不能与 dest 相同的寄存器。
- src0 不能有除默认重排 (.xyzw) 以外的任何重排。
- src1 不能有除默认重排 (.xyzw) 以外的任何重排。
- dest 必须有以下七个掩码之一：.x | .y | .z | .xy | .xz | .yz | .xyz。
- dest 必须是临时寄存器。
- dest 不得与 src0 或 src1 相同的寄存器

相关主题

顶点着色器指令

dcl_samplerType (sm3 - vs asm)

项目 • 2022/09/22

声明顶点着色器采样器。

语法

dcl_samplerType s#

其中：

- _samplerType定义采样器数据类型。 这将确定采样时每个纹理坐标所需的坐标数。 定义了以下纹理坐标维度。
  - _2d
  - _cube
  - _volume
- s# 标识采样器，其中采样器是采样器的缩写，#是采样器编号。 采样器 (Direct3D 9 asm-vs) 是伪寄存器，因为你无法直接读取或写入它们。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
dcl_samplerType					x	x

所有dcl_samplerType指令都必须显示在第一个可执行指令之前。

相关主题

顶点着色器指令

dcl_usage输入 (sm1、sm2、sm3 - vs asm)

项目 • 2022/09/22

声明顶点元素用法与顶点着色器输入寄存器的使用索引之间的关联。

语法

dcl_usage[usage_index] v#

其中：

- dcl_usage标识注册数据的使用方式。 此值与 **没有 D3DDECLUSAGE 前缀的 D3DDECLUSAGE** 的成员相同。
- usage_index是介于 0 和 15 之间的可选整数索引。 它修改使用情况数据。 索引与顶点声明中的使用索引匹配。 请参阅 Direct3D 9 (顶点声明) 。 索引追加到没有空格的使用情况值 (dcl_usage) 。 如果未提供，则假定为 0。
- v# 是 输入寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
dcl_usage	x	x	x	x	x	x

所有dcl_usage指令都必须显示在第一个可执行指令之前。

相关主题

顶点着色器指令

dcl_usage输出 (sm1、sm2、sm3 - vs asm)

项目 • 2022/09/22

各种类型的输出寄存器已折叠为 12 个输出寄存器， (2 个用于颜色，8 个用于纹理，1 个用于位置，1 个用于雾和点大小) 。 这些可用于用户想要为像素着色器内插的任何内容：纹理坐标、颜色、雾等。

输出寄存器需要包含语义的声明。 例如，旧位置和点大小寄存器通过声明具有位置或点大小语义的输出寄存器来替换。

在十二个输出寄存器中，任何 10 个 (不一定 o0 到 o9) 有 4 个组件 (xyzw) ，另一个组件必须声明为位置 (，并且还必须包括所有四个组件) ，并且可以选择再包含一个组件可以是标量点大小。

语法

声明输出寄存器的语法类似于输入寄存器的声明：

- dcl_semantics o[.write_mask]

其中：

- dcl_semantics可以使用与输入声明相同的语义集。 语义名称来自 D3DDECLUSAGE (，并与索引 (如 position3)) 配对。 当未用于处理顶点时，始终必须有一个输出寄存器与 positiont0 语义。 positiont0 semantic 和 pointsize0 语义是唯一具有含义的语义，仅允许从顶点到像素着色器的链接。 对于具有流控制的着色器，假定声明最差的情况输出。 如果着色器实际上没有输出由于流控制) 而声明它应 (的内容，则不存在默认值。
- o 是输出寄存器。 请参阅 Output_Registers。
- write_mask指示可以多次声明 (的输出寄存器，以便每次使用唯一写入掩码将不同的语义应用于各个组件) 。 但是，在声明中不能多次使用同一语义。 这意味着向量必须是四个分量或更少，并且不能跨越 (单个寄存器) 的四个分量寄存器边界。 使用点大小语义时，它应具有完整的写入掩码，因为它被视为标量。 使用位置语义时，它应具有完整的写入掩码，因为必须写入所有四个组件。

注解

顶点着色器版本	3_0	3_sw

顶点着色器版本	3_0	3_sw
dcl_usage	x	x

所有 dcl_usage 指令都必须显示在第一个可执行指令之前。

声明示例

```
vs_3_0
dcl_color4     o3.x    // color4 is a semantic name.
dcl_texcoord3  o3.yz   // Different semantics can be packed into one
register.
dcl_fog        o3.w
dcl_tangent    o4.xyz
dcl_position   o7.xyzw // position must be declared to some unique register
                       //   in a vertex shader, with all 4 components.

dcl_psize      o6      // Pointsize cannot have a mask
                       //   (that is, mask is full .xyzw)
                       // This is an implied scalar register.
                       // No other semantics can be assigned to any
components
                       //   of this register.
                       // If pointsize declaration is not used (typical),
                       //   only 11 "out" registers are available, not 12.
                       // Pixel shaders cannot see this value.
```

相关主题

顶点着色器指令

def - vs

项目 • 2022/09/22

定义顶点着色器常量。

语法

```
def dst , float1 , float2 , float3 , float4
```

其中

- dst 是目标寄存器。
- float1、float2、float3、float4 是四个浮点数。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
def	x	x	x	x	x	x

def 指令定义着色器常量，该常量在着色器设置为设备时加载其值。 这些常量称为即时常量。 即时常量优先于 API 方法 SetVertexShaderConstantF 设置的常量。

可通过两种方法在着色器中设置常量。

1. 使用 def - 与直接在着色器内定义常量。

   def - vs 只能定义浮点常量。

2. 使用 API 方法设置常量。

   - 使用 SetVertexShaderConstantF 设置浮点常量。

相关主题

顶点着色器指令

defi - vs

defb - vs

项目 • 2022/09/23

定义一个布尔常量值，应在着色器设置为设备时加载该值。

语法

defb dest、booleanValue

其中

- dst 是目标寄存器。
- booleanValue 是布尔值，值为 True 或 False。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
defb		x	x	x	x	x

defb - vs 指令定义一个布尔着色器常量，该常量在着色器设置为设备时加载其值。 这些常量称为即时常量。 即时常量优先于 API 方法 SetVertexShaderConstantB 设置的常量。

可通过两种方法在着色器中设置布尔常量。

1. 使用 defb - 与直接在着色器内定义常量。
2. 使用 API 方法设置常量。
    - 使用 SetVertexShaderConstantB 设置布尔常量。

相关主题

顶点着色器指令

def - vs

defi - vs

defi - vs

项目 • 2022/09/22

定义整数常量值，应在着色器设置为设备时加载该值。

语法

```
defi dst， integerValue0， integerValue1， integerValue2， integerValue3
```

- dst 是目标寄存器。
- integerValue# 是常量整数值。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
defi		x	x	x	x	x

defi 指令定义一个整数着色器常量，该常量在着色器设置为设备时加载其值。 这些常量称为即时常量。 即时常量优先于 API 方法 SetVertexShaderConstantI 设置的常量。

可通过两种方法在着色器中设置整数常量。

1. 使用 defi 直接在着色器内定义整数常量向量。
2. 使用 API 方法设置常量。
    - 使用 SetVertexShaderConstantI 设置整数常量。

相关主题

顶点着色器指令

def - vs

defb - vs

dp3 - vs

项目 • 2022/09/22

计算源寄存器的三分量点积。

语法

```
dp3 dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
dp3	x	x	x	x	x	x

以下代码片段显示了执行的操作：

```
dest.w = (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z);
dest.x = dest.y = dest.z = dest.w;
```

相关主题

顶点着色器指令

dp4 - vs

项目 • 2022/09/22

计算源寄存器的四分量点积。

语法

```
dp4 dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
dp4	x	x	x	x	x	x

以下代码片段显示执行的操作：

```
dest.w = (src0.x * src1.x) + (src0.y * src1.y) +
         (src0.z * src1.z) + (src0.w * src1.w);
dest.x = dest.y = dest.z = dest.w;
```

相关主题

顶点着色器指令

dst - vs

项目 • 2022/09/22

计算距离向量。

语法

```
dst dest、src0、src1
```

其中

- dest 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
dst	x	x	x	x	x	x

以下代码片段显示执行的操作：

```
dest.x = 1;
dest.y = src0.y * src1.y;
dest.z = src0.z;
dest.w = src1.w;
```

第一个源操作数 (src0) 假定为矢量 (被忽略、d*d、d*d、忽略) 和第二个源操作数 (src1) 假定为被忽略的向 (量，1/d、忽略、1/d) 。 目标 (dest) 是结果向量 (1、d、d*d、1/d) 。

相关主题

顶点着色器指令

else - vs

项目 • 2022/09/22

其他块"开始"菜单。

语法

else

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
else		x	x	x	x	x

如果相应 布尔 - vs 语句中的条件为 true，则布尔值 - vs 语句和匹配 的 endif 将运行由 bool - vs 语句括起来的代码。 否则，其他代码将括起来...endif 语句正在运行。

相关主题

顶点着色器指令

if bool - vs

endif - vs

endif - vs

项目 • 2022/09/24

标记布尔值 - vs...else block.

语法

endif

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
endif		x	x	x	x	x

相关主题

顶点着色器指令

if bool - vs

else - vs

endloop - vs

项目 • 2022/09/24

循环结束...endloop 块。

语法

endloop

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
endloop		x	x	x	x	x

此指令的工作方式如下。

```
LoopCounter += LoopStep;
LoopInterationCount = LoopIterationCount - 1;
if (LoopIterationCount > 0)
    Continue execution at the StartLoopOffset
```

endloop 必须遵循 循环与 块的最后一条指令。

示例

```
loop aL, i3
    add r1, r0, c2[ aL ]
endloop
```

相关主题

顶点着色器指令

endrep - vs

项目 • 2022/09/22

结束 代表 - vs...endrep 块。

语法

endrep

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
endrep		x	x	x	x	x

示例

```
rep i2
    add r0, r0, c0
endrep
```

相关主题

顶点着色器指令

exp - vs

项目 • 2022/09/22

提供全精度指数 2 倍。

语法

```
exp dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。 请参阅 源注册重排。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
exp	x	x	x	x	x	x

此指令提供至少 21 位的精度。

以下代码片段显示执行的操作：

```
dest.x = dest.y = dest.z = dest.w = (float)pow(2,
src.replicateSwizzleComponent);
```

相关主题

顶点着色器指令

expp - vs

项目 • 2022/09/22

提供部分精度指数 2^x。

语法

```
expp dst， src.{x|y|z|w}
```

其中：

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 .x、.y、.z、.w 重排(组件之一或 .r、.g、.b、.a 等效项) 。
- {x|y|z|w} 是源寄存器上所需的复制重排。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
expp	x	x	x	x	x	x

vs_1_1

exp - vs 指令根据顶点着色器版本以不同的方式运行。

在vs_1_1中，expp 指令提供以下结果：

```
v = the scalar value from the source register with a replicate swizzle

dest.x = pow(2, floor(v))
dest.y = v - floor(v)
dest.z = pow(2, v) (partial-precision)
dest.w = 1
```

在vs_2_0和向上，expp 指令提供以下结果：

```
v = the scalar value from the source register with a replicate swizzle

dest.x = dest.y = dest.z = dest.y = pow(2, v) (partial-precision)
```

vs_2_0

在vs_2_0和向上，指令的工作方式如下所示：

```
float V = the scalar value from the source register with a replicate swizzle

dest.x = dest.y = dest.z = dest.y = pow( 2, V ) (partial-precision)
```

该指令提供至少 10 位精度。

相关主题

顶点着色器指令

frc - vs

项目 • 2022/09/24

返回每个输入组件的分数部分。

语法

```
frc dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
Frc	x	x	x	x	x	x

以下代码片段从概念上演示指令的操作方式。

```
dest.x = src.x - (float)floor(src.x);
dest.y = src.y - (float)floor(src.y);
dest.z = src.z - (float)floor(src.z);
dest.w = src.w - (float)floor(src.w);
```

floor 函数将传入的参数转换为小于 (或等于) 参数的最大整数。 这会转换为浮点，然后减去原始值。 生成的小数值范围为 0.0 到 1.0。

对于版本 1_1，允许的写入掩码为 .y，不允许) .xy (.x。

相关主题

顶点着色器指令

if bool - vs

项目 • 2022/09/23

启动 if...否则...endif - vs block。

语法

if bool

其中 bool 是布尔寄存器号。 请参阅 常量布尔寄存器。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
if bool		x	x	x	x	x

如果 if 语句中的源布尔寄存器为 true，则运行 if 语句和匹配项所括的代码。 否则，其他代码将括起来...endif - vs 语句正在运行。 此指令使用一个指令槽。

如果块可以嵌套，

如果块无法跨越循环块。

示例

此指令提供条件静态流控制。

```
defb b2, TRUE

...

if b2
// Instructions to run if b2 is nonzero

else
// Instructions to run otherwise
```

```
endif
```

相关主题

顶点着色器指令

else - vs

endif - vs

if_comp - vs

项目 • 2022/09/24

"开始"菜单布尔值 - vs...else - vs...endif - vs block，条件基于可以在着色器中计算的值。 此指令用于根据条件跳过代码块。

语法

if_comp src0、src1

其中：

- _comp是两个源寄存器之间的比较。 该参数可以是下列值之一：

语法	**比较**
_gt	大于
_lt	小于
_ge	大于或等于
_le	小于或等于
_eq	等于
_ne	不等于

- src0 是源寄存器。 选择组件需要复制重排。
- src1 是源寄存器。 选择组件需要复制重排。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
if_comp			x	x	x	x

此指令用于根据条件跳过代码块。

```
if_lt src0, src1
   jump to the corresponding else or endif instruction;
```

请注意，在浮点数上使用相等和不等于比较模式。 由于在浮点计算期间发生舍入，因此可以对小非零数) (epsilon 值进行比较以避免错误。

具体限制包括：

- if_comp...else - vs...endif - 与 块 (以及谓词，如果块) 可以嵌套到最多 24 层深。
- src0 和 src1 寄存器需要复制重排。
- if_comp块必须以 其他 （ vs 或 endif - 与指令 ） 结尾。
- if_comp...else - vs...endif - 与 块无法对循环块进行分层。 if_comp块必须完全位于循环内部或外部 - 与 块。

示例

此指令提供条件动态流控制。

```
if_lt r3.x, r4.y
// Instructions to run if r3.x < r4.y

else
// Instructions to run otherwise

endif
```

相关主题

顶点着色器指令

如果 pred - vs

项目 • 2022/09/22

"开始"菜单 if pred - vs...else - vs...endif - vs block，其条件取自谓词寄存器的内容。

语法

```
if [ ! ]pred.replicateSwizzle
```

其中：

- [!]可选的 NOT 修饰符。 这会修改谓词寄存器中的值。
- pred 是谓词寄存器 p0。 请参阅 谓词寄存器。
- replicateSwizzle 是 (复制或复制到所有四个组件) (重排) 的单个组件。 有效组件包括：x、y、z、w 或 r、g、b、a。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
if pred			x	x	x	x

此指令用于基于谓词寄存器的通道跳过代码块。 每个if_pred块必须以其他或 endif 指令结尾。

具体限制包括：

可以嵌套if_pred块。 这计入动态嵌套深度和 if_comp 块。

if_pred块不能跨越循环块，它应该完全位于它内部或包围它。

相关主题

顶点着色器指令

label - vs

项目 • 2022/09/23

将下一个指令标记为具有标签索引。

语法

label l#

where # 标识标签号。

对于vs_2_0和vs_2_x，标签号必须介于 0 到 15 之间。

对于vs_2_sw、vs_3_0和vs_3_sw，标签号必须介于 0 到 2047 之间。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
标签		x	x	x	x	x

此指令定义位于下一个着色器指令处的标签。 标签指令只能在 重新 指令 (之后直接存在，该指令定义上一个子例程或主程序) 的结尾。

相关主题

顶点着色器指令

lit - vs

项目 • 2022/09/23

通过计算两个点产品和指数中的照明系数，为照明提供部分支持。

语法

lit dst， src

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
点燃	x	x	x	x	x	x

假定源向量包含以下伪代码中显示的值。

```
src.x = N*L        ; The dot product between normal and direction to light
src.y = N*H        ; The dot product between normal and half vector
src.z = ignored    ; This value is ignored
src.w = exponent   ; The value must be between -128.0 and 128.0
```

以下代码片段显示执行的操作。

```
dest.x = 1;
dest.y = 0;
dest.z = 0;
dest.w = 1;

float power = src.w;
const float MAXPOWER = 127.9961f;
```

```
if (power < -MAXPOWER)
    power = -MAXPOWER;          // Fits into 8.8 fixed point format
else if (power > MAXPOWER)
    power = MAXPOWER;          // Fits into 8.8 fixed point format

if (src.x > 0)
{
    dest.y = src.x;
    if (src.y > 0)
    {
        // Allowed approximation is EXP(power * LOG(src.y))
        dest.z = (float)(pow(src.y, power));
    }
}
```

在评估目标 y 组件时，可以接受精度降低的算术 (dest.y) 。 实现必须支持电源参数中至少八个分数位。 点乘以规范化向量计算，固定限制为 -128 到 128。

错误应对应于 logp - vs 和 exp - 与 组合，或不超过 8 位颜色组件的大约一个有效位。

相关主题

顶点着色器指令

log - vs

项目 • 2022/09/23

全精度 log一 (x) 。

语法

log dst、src

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
log	x	x	x	x	x	x

以下代码片段显示执行的操作。

```
float v = abs(src);
if (v != 0)
{
    dest.x = dest.y = dest.z = dest.w =
        (float)(log(v)/log(2));
}
else
{
    dest.x = dest.y = dest.z = dest.w = -FLT_MAX;
}
```

此指令接受忽略其符号位的标量源。 结果将复制到所有四个通道。

此指令提供 21 位精度。

相关主题

顶点着色器指令

logp - vs

项目 • 2022/09/23

部分精度 logp一 (x) 。

语法

```
logp dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
logp	x	x	x	x	x	x

以下代码片段显示执行的操作。

```
float f = abs(src);
if (f != 0)
    dest.x = dest.y = dest.z = dest.w = (float)(log(f)/log(2));
else
    dest.x = dest.y = dest.z = dest.w = -FLT_MAX;
```

此指令提供对数基 2 部分精度，最多 10 位。

相关主题

顶点着色器指令

loop - vs

项目 • 2022/09/23

"开始"菜单循环...endloop 块。

语法

循环 aL， i#

其中：

- aL 是持有当前循环计数的Loop计数器寄存器。
- i# 是 常量整数寄存器。 请参阅备注。

备注

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
loop		x	x	x	x	x

- Loop计数器寄存器 (aL) 保存当前循环计数，并可用于在循环块内将常量整数寄存器 (c#) 或输出寄存器 (o#) 相对寻址。
- i#.x 指定迭代计数。 法律范围为 [0， 255]。 请注意，此指令不会递增或递减 i#.x 的值。
- i#.y 指定 Loop Counter Register (aL) 寄存器的初始值。 法律范围为 [0， 255]。 请注意，此指令不会递增或递减 i#.y 的值。
- i#.z 指定步骤/步幅大小。 法律范围为 [-128， 127]。
- i#.w 未使用，必须设置为 0。
- Loop块可以嵌套。 请参阅Flow控制嵌套限制。
- 嵌套时，Loop计数器寄存器的值 (aL) 引用直接封闭循环块。
- Loop块可以完全位于 if* 块内或完全围绕它。 不允许流浪。

示例

```
```

```
loop aL, i3
    add r1, r0, c2[aL]
endloop
```

相关主题

顶点着色器指令

lrp - vs

项目 • 2022/09/22

按第一个源寄存器中指定的比例在第二和第三个源寄存器之间线性内插。

语法

lrp dst， src0， src1， src2

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。
- src2 是源寄存器。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
Lrp		x	x	x	x	x

此指令基于以下公式执行线性内插。

```
dest.x = src0.x * (src1.x - src2.x) + src2.x;
dest.y = src0.y * (src1.y - src2.y) + src2.y;
dest.z = src0.z * (src1.z - src2.z) + src2.z;
dest.w = src0.w * (src1.w - src2.w) + src2.w;
```

相关主题

顶点着色器指令

m3x2 - vs

项目 • 2022/09/23

将 3 分量向量乘以 3x2 矩阵。

语法

m3x2 dst、src0、src1

其中

- dst 是目标寄存器。 结果是一个 2 分量的向量。
- src0 是表示 3 分量矢量的源寄存器。
- src1 是表示 3x2 矩阵的源寄存器，对应于连续 2 个寄存器中的第一个。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
m3x2	x	x	x	x	x	x

目标寄存器需要 xy 掩码。 对于 src0，但不允许 src1 使用反门和重排修饰符。

以下公式显示了指令的运行方式：

```
dest.x = (src0.x * src1.x) + (src0.x * src1.y) + (src0.x * src1.z);
dest.y = (src0.x * src2.x) + (src0.y * src2.y) + (src0.z * src2.z);
```

输入向量位于寄存器 src0 中。 输入 3x2 矩阵位于寄存器 src1 中，同一寄存器文件中的下一个更高的寄存器，如以下扩展所示。 生成 2D 结果，使目标寄存器的其他元素 (dest.z 和 dest.w) 不受影响。

此操作通常用于 2D 转换。 此指令作为点产品的配对实现，如下所示。

```
m3x2   r0.xy, r1, c0   which will be expanded to:

dp3   r0.x, r1, c0
dp3   r0.y, r1, c1
```

src1 寄存器的重排和否定修饰符无效。 dest 和 src0 寄存器或 src1+i 寄存器中的任何一个寄存器不能相同。

相关主题

顶点着色器指令

m3x3 - vs

项目 • 2022/09/22

将 3 分量向量乘以 3x3 矩阵。

语法

```
m3x3 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为 3 分量向量。
- src0 是表示 3 分量矢量的源寄存器。
- src1 是表示 3x3 矩阵的源寄存器，对应于连续 3 个寄存器中的第一个。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
m3x3	x	x	x	x	x	x

目标寄存器需要 xyz 掩码。 对于 src0，但不允许 src1 使用反门和重排修饰符。

以下代码片段显示执行的操作。

```
dest.x = (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z);
dest.y = (src0.x * src2.x) + (src0.y * src2.y) + (src0.z * src2.z);
dest.z = (src0.x * src3.x) + (src0.y * src3.y) + (src0.z * src3.z);
```

输入向量位于寄存器 src0 中。 输入 3x3 矩阵位于寄存器 src1 中，接下来的两个更高的寄存器，如下面的扩展所示。 生成 3D 结果，使目标寄存器的另一个元素 (dest.w) 不受影响。

此操作通常用于在照明计算期间转换普通向量。 此指令作为点产品的配对实现，如下所示。

```
m3x3 r0.xyz, r1, c0  which will be expanded to:

dp3   r0.x, r1, c0
dp3   r0.y, r1, c1
dp3   r0.z, r1, c2
```

相关主题

顶点着色器指令

m3x4 - vs

项目 • 2022/09/22

将 3 分量向量乘以 3x4 矩阵。

语法

```
m3x4 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为 4 分量向量。
- src0 是表示 3 分量矢量的源寄存器。
- src1 是表示 3x4 矩阵的源寄存器，对应于连续 4 个寄存器中的第一个。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
m3x4	x	x	x	x	x	x

目标寄存器需要 xyzw (默认) 掩码。 对于 src0，但不允许 src1 使用反门和重排修饰符。

以下代码片段显示执行的操作。

```
dest.x = (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z);
dest.y = (src0.x * src2.x) + (src0.y * src2.y) + (src0.z * src2.z);
dest.z = (src0.x * src3.x) + (src0.y * src3.y) + (src0.z * src3.z);
dest.w = (src0.x * src4.x) + (src0.y * src4.y) + (src0.z * src4.z);
```

输入向量位于寄存器 src0 中。 输入 3x4 矩阵位于寄存器 src1 中，接下来的三个更高的寄存器，如下面的扩展所示。

此操作通常用于通过具有投影效果的矩阵转换位置向量，但不应用任何翻译。 此指令作为点产品的配对实现，如下所示。

```
m3x4   r0.xyzw, r1, c0   will be expanded to:

dp3 r0.x, r1, c0
dp3 r0.y, r1, c1
dp3 r0.z, r1, c2
dp3 r0.w, r1, c3
```

相关主题

顶点着色器指令

m4x3 - vs

项目 • 2022/09/23

将 4 分量向量乘以 4x3 矩阵。

语法

```
m4x3 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为三分量向量。
- src0 是表示 4 分量的源寄存器。
- src1 是表示 4x3 矩阵的源寄存器，对应于连续 3 个寄存器中的第一个。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
m4x3	x	x	x	x	x	x

目标寄存器需要 xyz 掩码。 src0 允许反门和重排修饰符，但不允许用于 src1。

以下代码片段显示了执行的操作。

```
dest.x = (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z) + (src0.w
* src1.w);
dest.y = (src0.x * src2.x) + (src0.y * src2.y) + (src0.z * src2.z) + (src0.w
* src2.w);
dest.z = (src0.x * src3.x) + (src0.y * src3.y) + (src0.z * src3.z) + (src0.w
* src3.w);
```

输入向量位于寄存器 src0 中。 输入 4x3 矩阵位于寄存器 src1 中，接下来的两个更高的寄存器，如下面的扩展所示。 生成 3D 结果，使目标寄存器的另一个元素 (dest.w) 不受影响。

此操作通常用于通过没有投影效果的矩阵转换位置向量，例如在模型空间转换中发生。 此指令作为点产品的配对实现，如下所示。

```
m4x3   r0.xyz, r1, c0    will be expanded to:

dp4   r0.x, r1, c0
dp4   r0.y, r1, c1
dp4   r0.z, r1, c2
```

src1 寄存器的重排和否定修饰符无效。 dst 和 src0 寄存器不能相同。

相关主题

顶点着色器指令

m4x4 - vs

项目 • 2022/09/22

将 4 分量向量乘以 4x4 矩阵。

语法

```
m4x4 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为 4 分量向量。
- src0 是表示 4 分量的源寄存器。
- src1 是表示 4x4 矩阵的源寄存器，对应于连续 4 个寄存器中的第一个。

备注

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
m4x4	x	x	x	x	x	x

目标寄存器需要 xyzw (默认) 掩码。 src0 允许反门和重排修饰符，但不允许 src1 使用。

src0 寄存器的重排和否定修饰符无效。 dest 和 src0 寄存器不能相同。

以下代码片段显示执行的操作。

```
dest.x = (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z) +
        (src0.w * src1.w);
dest.y = (src0.x * src2.x) + (src0.y * src2.y) + (src0.z * src2.z) +
        (src0.w * src2.w);
dest.z = (src0.x * src3.x) + (src0.y * src3.y) + (src0.z * src3.z) +
        (src0.w * src3.w);
dest.w = (src0.x * src4.x) + (src0.y * src4.y) + (src0.z * src4.z) +
        (src0.w * src4.w);
```

输入向量位于寄存器 src0 中。 输入 4x4 矩阵位于寄存器 src1 中，接下来的三个更高的寄存器，如下面的扩展所示。

此操作通常用于通过投影矩阵转换位置。 此指令作为一系列点产品实现，如下所示。

```
m4x4   r0.xyzw, r1, c0    will be expanded to:

dp4   r0.x, r1, c0
dp4   r0.y, r1, c1
dp4   r0.z, r1, c2
dp4   r0.w, r1, c3
```

相关主题

顶点着色器指令

mad - vs

项目 • 2022/09/23

乘以并添加源。

语法

```
mad dst , src0 , src1 , src2
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。
- src2 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
疯狂	x	x	x	x	x	x

以下代码片段显示执行的操作。

```
dest.x = src0.x * src1.x + src2.x;
dest.y = src0.y * src1.y + src2.y;
dest.z = src0.z * src1.z + src2.z;
dest.w = src0.w * src1.w + src2.w;
```

相关主题

顶点着色器指令

max - vs

项目 • 2022/09/22

计算源的最大数目。

语法

```
max dst , src0 , src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
max	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x=(src0.x >= src1.x) ? src0.x : src1.x;
dest.y=(src0.y >= src1.y) ? src0.y : src1.y;
dest.z=(src0.z >= src1.z) ? src0.z : src1.z;
dest.w=(src0.w >= src1.w) ? src0.w : src1.w;
```

相关主题

顶点着色器指令

min - vs

项目 • 2022/09/22

计算源的最小值。

语法

min dst、src0、src1

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
min	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x=(src0.x < src1.x) ? src0.x : src1.x;
dest.y=(src0.y < src1.y) ? src0.y : src1.y;
dest.z=(src0.z < src1.z) ? src0.z : src1.z;
dest.w=(src0.w < src1.w) ? src0.w : src1.w;
```

相关主题

顶点着色器指令

mov - vs

项目 • 2022/09/22

在寄存器之间移动浮点数据。

语法

```
mov dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
mov	x	x	x	x	x	x

可用于浮点数据。 对于版本vs_1_1，它还可用于写入地址寄存器。 用于更新地址寄存器时，值将使用舍入转换为最接近的浮点。

以下代码片段显示执行的操作。

```
if(dest is an integer register)
{
    int intSrc = RoundToNearest(src.w);
    dest = intSrc;
}
else
{
    dest = src;
}
```

相关主题

顶点着色器指令

mova - vs

项目 • 2022/09/22

将数据从浮点寄存器移动到 地址寄存器，a0。

语法

```
mova dst , src
```

其中

- dst 必须是 地址寄存器，a0。
- src 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
mova		x	x	x	x	x

将浮点数据移动到整数寄存器。 这些值使用舍入转换为最接近的浮点。

地址寄存器是允许的唯一目标寄存器。

以下代码片段显示了执行的操作。

```
if(dest is an integer register)
{
    int intSrc = RoundToNearest(src);
    dest = intSrc;
}
else
{
    dest = src;
}
```

对于版本2_x及更高版本，地址寄存器是组件向量。 因此，允许任何写入掩码。

```
mova a0.xz, r0
```

相关主题

顶点着色器指令

mul - vs

项目 • 2022/09/24

将源乘以目标。

语法

```
mul dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
mul	x	x	x	x	x	x

以下代码片段显示执行的操作。

```
dest.x = src0.x * src1.x;
dest.y = src0.y * src1.y;
dest.z = src0.z * src1.z;
dest.w = src0.w * src1.w;
```

相关主题

顶点着色器指令

nop - vs

项目 • 2022/09/22

不执行任何操作。

语法

```
nop
```

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
nop	x	x	x	x	x	x

此指令执行无操作或无操作。 调用它的语法如下所示：

```
nop
```

相关主题

顶点着色器指令

nrm - vs

项目 • 2022/09/22

规范化 3D 向量。

语法

```
nrm dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
nrm		x	x	x	x	x

此说明在概念上有效，如下所示。

squareRootOfTheSum = (src0.x*src0.x + src0.y*src0.y + src0.z*src0.z) $^{1/2}$;

```
dest.x = src0.x * (1 / squareRootOfTheSum);
dest.y = src0.y * (1 / squareRootOfTheSum);
dest.z = src0.z * (1 / squareRootOfTheSum);
dest.w = src0.w * (1 / squareRootOfTheSum);
```

dest 和 src 寄存器不能相同。 dest 寄存器必须是临时寄存器。

此指令根据以下公式执行线性内插。

```
float f = src0.x*src0.x + src0.y*src0.y + src0.z*src0.z;
if (f != 0)
```

```
    f = (float)(1/sqrt(f));
else
    f = FLT_MAX;

dest.x = src0.x*f;
dest.y = src0.y*f;
dest.z = src0.z*f;
dest.w = src0.w*f;
```

相关主题

顶点着色器指令

pow - vs

项目 • 2022/09/22

全精度 abs (src0) ^{src1}。

语法

pow dst、src0、src1

其中

- dst 是目标寄存器。
- src0 是输入源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。
- src1 是输入源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
pow		x	x	x	x	x

此说明的工作方式如下。

```
dest = pow(abs(src0), src1);
```

这是标量指令，因此源寄存器应具有复制重排来指示使用哪个通道。

标量结果将复制到所有四个输出通道。

此指令可以扩展为 exp (src1 * log (src0)) 。

精度不低于 15 位。

dest 寄存器应该是临时寄存器，不应与 src1 相同的寄存器。

相关主题

顶点着色器指令

rcp - vs

项目 • 2022/09/23

计算源标量的对等。

语法

```
rcp dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
rcp	x	x	x	x	x	x

以下代码片段显示执行的操作。

```
float f = src0;
if(f == 0.0f)
{
    f = FLT_MAX;
}
else
{
    if(f != 1.0)
    {
        f = 1/f;
    }
}

dest = f;
```

如果输入正好为 1.0，则输出必须正好为 1.0。 0.0 的源产生无穷大。

精度应至少为 1.0/ (2一一) 在 (1.0、2.0) 范围内出现绝对误差，因为常见的实现将分离 mantissa 和指数。

如果源没有下标，则使用 x 组件。

相关主题

顶点着色器指令

rep - vs

项目 • 2022/09/24

"开始"菜单代表...endrep 块。

语法

rep i#

其中 i# 是一个整数寄存器，用于指定 .x 组件中的重复计数。 请参阅 常量整数寄存器。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
代表		x	x	x	x	x

- i#.x 指定迭代计数。 法律范围为 [0， 255]。 请注意，此指令不会递增或递减 i#.x 的值。
- 重复块不使用 i#.yzw。
- 重复块可以嵌套。 请参阅Flow控制嵌套限制。
- 允许重复块完全位于 if* 块内或完全围绕它。 不允许流浪。
- 对不同的或嵌套的代表指令使用相同的 i# 是正常的 - 每个循环都将基于指定的计数循环访问。

示例

```
rep i2
    add r0, r0, c0
endrep
```

相关主题

顶点着色器指令

ret - vs

项目 • 2022/09/22

从子例程或主函数返回。

语法

Ret

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
Ret		x	x	x	x	x

此指令从返回地址堆栈获取指令的地址，并继续从该地址堆栈执行。 对于主函数，此指令将停止着色器执行。

重试指令使用一个顶点着色器指令槽。

如果着色器不包含子例程，则使用主程序末尾的 ret 是可选的。

主程序或任何子例程不允许使用多个返回语句，第一个返回语句被视为主程序或子例程的结尾。

相关主题

顶点着色器指令

rsq - vs

项目 • 2022/09/24

计算源标量 (正) 的对方根。

语法

```
rsq dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
rsq	x	x	x	x	x	x

以下代码片段显示执行的操作。

```
float f = abs(src0);
if (f == 0)
    f = FLT_MAX
else
{
    if (f != 1.0)
        f = 1.0/(float)sqrt(f);
}

dest.z = dest.y = dest.z = dest.w = f;
```

在处理之前，将采用绝对值。

精度应至少为 1.0/ (2E，) 绝对误差超过 1.0、4.0) (，因为常见的实现将分离 mantissa 和指数。

如果源没有下标，则使用 x 组件。 如果输入正好为 1.0，则输出必须正好为 1.0。 0.0 的源产生无穷大。

相关主题

顶点着色器指令

setp_comp - vs

项目 • 2022/12/14

设置谓词寄存器。

语法

setp_comp dst、src0、src1

其中：

- _comp是两个源寄存器之间的每通道比较。 该参数可以是下列值之一：

语法	**比较**
_gt	大于
_lt	小于
_ge	大于或等于
_le	小于或等于
_eq	等于
_ne	不等于

- dst 是 谓词寄存器 ，p0。
- src0 是源寄存器。
- src1 是源寄存器。

备注

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
setp_comp			x	x	x	x

此指令按以下方式运行：

```
per channel in destination write mask
{
  dst.channel = src0.channel cmp src1.channel
}
```

对于可以根据目标写入掩码写入的每个通道，请在源修饰符重排) 解析后，在 src0 和 src (1 的相应通道之间保存比较操作的布尔结果。

源寄存器允许指定任意组件重排。

目标寄存器允许任意写入掩码。

dest 寄存器必须是谓词寄存器。

应用谓词寄存器

使用 setp 初始化谓词寄存器后，它可用于控制每个组件的指令。 下面是语法：

```
([!]p0[.swizzle]) instruction dest, srcReg, ...
```

其中：

- [!]是可选的布尔 NOT
- p0 是谓词寄存器
- [.swizzle] 是一个可选重排，用于在使用它屏蔽指令之前应用于谓词寄存器的内容。 可用重排为：如果未指定任何），或者任何复制重排：.x/.r、.y/.g、.z/.b 或 .a/.w，则为 .xyzw (默认值。
- 指令是任何有符号或纹理指令。 这不能是静态或动态流控制指令。
- dest，srcReg，...是指令所需的寄存器

假设谓词寄存器已使用 (true、true、false、false) 组件值进行设置，则可以应用于以下指令：

```
// given r0 = 0,0,1,1
// given r1 = 1,1,0,0
setp_le p0, r0, r1
(p0) add r2, r3, r4
```

若要执行 2 个组件添加，则为 。

```
r2.x = r3.x + r4.x
r2.y = r3.y + r4.y
```

r2 的 x 和 y 组件不会写入，因为谓词寄存器在 components z 和 w 中包含 false。

将谓词寄存器应用于算术或纹理指令时，其指令槽计数增加 1。

谓词寄存器也可以应用于 pred - vs， callnz pred - vs 和 breakp - vs 指令。 使用谓词寄存器时，这些流控制指令在指令槽计数中没有任何增加。

相关主题

顶点着色器指令

sge - vs

项目 • 2022/09/22

如果第一个源大于或等于第二个源，则计算该符号。

语法

```
sge dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
sge	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x = (src0.x >= src1.x) ? 1.0f : 0.0f;
dest.y = (src0.y >= src1.y) ? 1.0f : 0.0f;
dest.z = (src0.z >= src1.z) ? 1.0f : 0.0f;
dest.w = (src0.w >= src1.w) ? 1.0f : 0.0f;
```

相关主题

顶点着色器指令

sgn - vs

项目 • 2022/09/22

计算输入的符号。

语法

sgn dst、src0、src1、src2

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是保存中间结果的临时寄存器。 执行后，内容未定义。
- src2 是一个临时寄存器，用于保存中间结果。 执行后，内容未定义。

注解

顶点着色器版本	**1_1**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
sgn		x	x	x	x	x

此指令的工作原理如下所示。

```
for each component in src0
{
   if (src0.component < 0)
       dest.component = -1;
   else
       if (src0.component == 0)
           dest.component = 0;
       else
           dest.component = 1;
}
```

src1 和 src2 必须是不同的 临时寄存器。

相关主题

顶点着色器指令

sincos - vs

项目 • 2022/09/22

以弧度计算正弦和余弦。

语法

vs_2_0和vs_2_x

```
sincos dst.{x|y|xy}，src0。{x|y|z|w}， src1， src2
```

其中：

- dst 是目标寄存器，必须是 临时注册 (r#)。 目标寄存器必须具有以下三个掩码之一：.x | .y | .xy。
- src0 是一个源寄存器，它提供输入角度，必须位于 [-pi， +pi] 内。 {x|y|z|w} 是所需的复制重排。
- src1 和 src2 是源寄存器，必须是两个不同的 常量浮点寄存器 (c#)。 src1 和 src2 的值必须是宏 D3DSINCOSCONST1 和 D3DSINCOSCONST2 的值。

vs_3_0

```
sincos dst.{x|y|xy}，src0。{x|y|z|w}
```

其中：

- dst 是目标寄存器，必须是 临时注册 (r#)。 目标寄存器必须具有以下三个掩码之一：.x | .y | .xy。
- src0 是一个源寄存器，它提供输入角度，必须位于 [-pi， +pi] 内。 {x|y|z|w} 是所需的复制重排。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
sincos		x	x	x	x	x

vs_2_0和vs_2_x备注

对于vs_2_0和vs_2_x，sincos 可用于谓词，但对 谓词寄存器 (p0) 的重排有一个限制：仅允许复制重排 (.x | .y | .z | .w) 。

对于vs_2_0和vs_2_x，指令按如下所示操作： (V = 具有复制重排) 的 src0 的标量值：

- 如果写入掩码为 .x：

```
dest.x = cos( V )
dest.y is undefined when the instruction completes
dest.z is undefined when the instruction completes
dest.w is not touched by the instruction
```

- 如果写入掩码为 .y：

```
dest.x is undefined when the instruction completes
dest.y = sin( V )
dest.z is undefined when the instruction completes
dest.w is not touched by the instruction
```

- 如果写入掩码为 .xy：

```
dest.x = cos( V )
dest.y = sin( V )
dest.z is undefined when the instruction completes
dest.w is not touched by the instruction
```

vs_3_0备注

对于vs_3_0，sincos 可以与谓词一起使用，没有任何限制。 请参阅 谓词寄存器。

对于vs_3_0，指令按如下所示操作： (V = src0 中的标量值，其中包含复制重排)

- 如果写入掩码为 .x：

```
dest.x = cos( V )
dest.y is not touched by the instruction
dest.z is not touched by the instruction
dest.w is not touched by the instruction
```

- 如果写入掩码为 .y：

```
dest.x is not touched by the instruction
dest.y = sin( V )
dest.z is not touched by the instruction
dest.w is not touched by the instruction
```

- 如果写入掩码为 .xy：

```
dest.x = cos( V )
dest.y = sin( V )
dest.z is not touched by the instruction
dest.w is not touched by the instruction
```

应用程序可以使用以下着色器伪代码将任意角度 (以弧度) 映射到范围 [-pi， +pi] ：

```
def c0, pi, 0.5, 2*pi, 1/(2*pi)
mad r0.x, input_angle, c0.w, c0.y
frc r0.x, r0.x
mad r0.x, r0.x, c0.z, -c0.x
```

相关主题

顶点着色器指令

slt - vs

项目 • 2022/09/24

如果第一个源小于第二个源，则计算该符号。

语法

```
slt dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
slt	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x = (src0.x < src1.x) ? 1.0f : 0.0f;
dest.y = (src0.y < src1.y) ? 1.0f : 0.0f;
dest.z = (src0.z < src1.z) ? 1.0f : 0.0f;
dest.w = (src0.w < src1.w) ? 1.0f : 0.0f;
```

相关主题

顶点着色器指令

sub - vs

项目 • 2022/09/22

减去源。

语法

```
sub dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

备注

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
sub	x	x	x	x	x	x

此指令执行此公式中显示的减法。

```
dest.x = src0.x - src1.x
dest.y = src0.y - src1.y
dest.z = src0.z - src1.z
dest.w = src0.w - src1.w
```

相关主题

顶点着色器指令

texldl - vs

项目 • 2022/09/22

使用特定采样器对纹理进行采样。 要采样的特定 mipmap 详细级别必须指定为纹理坐标的第四个组件。

语法

```
texldl dst , src0 , src1
```

其中：

- dst 是目标寄存器。
- src0 是一个源寄存器，它提供纹理样本的纹理坐标。
- src1 标识源采样器寄存器 (s#) ，其中 # 指定要采样的纹理采样器编号。 采样器已将其关联到纹理和 D3DSAMPLERSTATETYPE 枚举定义的控件状态 (，例如 D3DSAMP_MINFILTER) 。

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
texldl					x	x

texldl 在 src1 引用的采样器阶段查找纹理集。 从 src0.w 中选择详细信息级别。 此值可以是负值，在这种情况下，所选的详细级别为“零一”，(使用 MAGFILTER 的最大地图) 。 由于 src0.w 是浮点值，因此如果 MIPFILTER 在两个 mip 级别之间) ，则小数值用于内插 (。 采样器状态 MIPMAPLODBIAS 和 MAXMIPLEVEL 受到尊重。 有关采样器状态的详细信息，请参阅 D3DSAMPLERSTATETYPE。

如果采样器中的着色器程序样本没有纹理集，则会在目标寄存器中获取 0001。

这是引用设备算法的近似值。

```
LOD = src0.w + LODBIAS;
if (LOD <= 0 )
```

```
{
   LOD = 0;
   Filter = MagFilter;
   tex = Lookup( MAX(MAXMIPLEVEL, LOD), Filter );
}
else
{
   Filter = MinFilter;
   LOD = MAX( MAXMIPLEVEL, LOD);
   tex = Lookup( Floor(LOD), Filter );
   if( MipFilter == LINEAR )
   {
      tex1 = Lookup( Ceil(LOD), Filter );
      tex = (1 - frac(src0.w))*tex + frac(src0.w)*tex1;
   }
}
```

限制：

- 纹理坐标不应按纹理大小缩放。
- dst 必须是 临时注册 (r#) 。
- dst 可以接受写入掩码。 请参阅 目标注册掩码。
- 缺少组件的默认值为 0 或 1，具体取决于纹理格式。
- src1 必须是 Sampler (Direct3D 9 asm-vs) (s#) 。 src1 不得使用反门修饰符。 src1 可以使用重排，在遵循写入掩码之前在采样后应用。 采样器必须使用 dcl_samplerType (sm3 (声明 (- 着色器开头的 asm)) 。
- 执行纹理样本所需的坐标数取决于采样器声明方式。 如果声明为多维数据集，则需要三分量的纹理坐标 (.rgb) 。 验证强制向 texldl 提供的坐标足以满足为采样器声明的纹理维度。 但是，不能保证应用程序实际上通过 API (设置纹理，其维度等于为采样器声明的维度) 。 在这种情况下，运行时将尝试检测 (可能仅在调试) 不匹配。 允许对维度小于纹理坐标中存在的纹理采样，并假定忽略额外的纹理坐标组件。 相反，不允许采样具有比纹理坐标中存在的维度多的纹理。
- 如果 src0 (纹理坐标) 是 临时寄存器 (r#) ，则必须事先编写上述查找 (所需的组件) 。
- 采样无符号 RGB 纹理将导致浮点值介于 0.0 和 1.0 之间。
- 采样有符号纹理将导致 -1.0 到 1.0 之间的浮点值。
- 采样浮点纹理时，Float16 表示数据将在MAX_FLOAT16范围内。 Float32 表示将使用管道的最大范围。 未定义这两个范围之外的采样。
- 没有依赖读取限制。

相关主题

顶点着色器指令

vs

项目 • 2022/09/23

此指令指定着色器版本号。 此指令适用于所有着色器版本。

语法

```
vs_mainVer_subVer
```

输入参数

输入参数包含具有单个子版本号的单个主版本号。 允许的组合列在下表中。

主版本	子版本
1	1
2	0、sw (软件) 、x (扩展)
3	0 ， sw (软件)

注解

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
vs	x	x	x	x	x	x

此指令必须是顶点着色器中的第一个非注释指令。

所有顶点着色器版本都支持此指令。

在版本号) 中没有_sw，硬件加速版本的软件 (版本可以处理具有硬件提升的顶点或使用软件顶点处理。 仅在软件中处理顶点) 版本号中具有_sw的软件版本 (版本。

示例

此部分示例声明版本 1_1 顶点着色器。

```
vs_1_1
```

本部分示例声明版本 2 软件顶点着色器。

```
vs_2_sw
```

相关主题

顶点着色器指令

顶点着色器寄存器

项目 • 2022/09/22

- 寄存器 - vs_1_1
- 寄存器 - vs_2_0
- 寄存器 - vs_2_x
- 寄存器 - vs_3_0
- 顶点着色器寄存器修饰符

相关主题

顶点着色器

顶点着色器寄存器修饰符

项目 • 2022/09/22

- 顶点着色器源寄存器修饰符
- 源注册重排
- 目标注册掩码

相关主题

顶点着色器寄存器

顶点着色器源寄存器修饰符

项目 • 2022/09/22

源修饰符可用于在指令使用数据之前修改从源寄存器读取的数据。

Negate

否定源寄存器的内容。

组件修饰符	说明
- r	源否定

不能在这些指令的第二个源寄存器上使用反门修饰符： m3x2 - vs、 m3x3 - vs、 m3x4 - vs、 m4x3 - vs、 m4x4 - vs。

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
-	x	x	x	x	x	x

绝对值

获取寄存器的绝对值。

顶点着色器版本	1_1	2_0	2_x	2_sw	3_0	3_sw
abs					x	x

如果从一个或多个常量浮点寄存器 (c#) 读取任何版本 3 着色器，则以下任一类型必须为 true。

- 所有常量浮点寄存器都必须使用 abs 修饰符。
- 常量浮点寄存器都不能使用 abs 修饰符。

相关主题

顶点着色器寄存器修饰符

源注册重排 (HLSL VS 参考)

项目 • 2022/12/14

在指令运行之前，源寄存器中的数据将复制到临时寄存器。 重排是指将任何源寄存器组件复制到任何临时注册组件的功能。 重排不会影响源寄存器数据。

组件重排

如下表所示，可向源寄存器数据的各个组件应用重排， (其中是有效顶点着色器输入寄存器之一 - vs_1_1) 。

组件修饰符	说明
r.[xyzw][xyzw][xyzw][xyzw]	源重排

- 始终复制所有四个组件。 如果指定了少于四个组件，则最后一个组件重复 (xy 表示 .xyyy) 。 如果未指定任何组件，则会重复 x (.xxxx) 。
- 组件可以按任意顺序显示。 v0.ywx 导致 v0.ywxx。
- rgba 组件分别可用于 x for x、g for b 等的 xyzw (r) 。
- 这些说明实现源寄存器单组件重排：exp、expp、logp、pow、rcp、rsq。 这些指令的结果将复制到所有四个目标寄存器组件。

不能在 m3x2 - vs、 m3x3 - vs、 m4x3 - vs 和 m4x4 - 指令 上使用重排。

相关主题

顶点着色器寄存器修饰符

目标注册掩码

项目 • 2022/09/23

掩码指定将使用指令结果更新目标寄存器的哪些组件。 纹理寄存器有一组规则，非文本寄存器有另一组规则。

- dx9_graphics_reference_asm_vs_registers_modifiers_masking - 本部分包含将掩码应用于目标寄存器的规则。
- Texture_Register_Masks - 纹理寄存器具有一些唯一的规则。

目标注册掩码

如下表所示，掩码 (其中一个有效 顶点着色器顶点着色器寄存器) 可应用于矢量数据的单个组件。

组件修饰符	说明
r.{x}{y}{z}{w}	目标掩码

- 通常，指定目标寄存器写入掩码是良好的编码样式。 它使代码更易于读取和维护。
- 只要 x 位于 y、y 之前，并且 z 位于 w 之前，就可以指定任何组件组合，（包括无）。
- oPts 和 oFog 输出寄存器只能使用一个掩码。
- 某些说明要求目标寄存器使用单个写入掩码：exp、expp、logp、pow、rcp 和 rsq。
- 在版本 1.0 中，frc 指令需要以下掩码组合之一：.x 或 .y 或 .xy。 版本 2.0 没有掩码限制。
- sincos 需要以下掩码组合之一：.x 或 .y 或 .xyz。
- m3x2 需要 .xy 写入掩码。
- m3x3 和 m4x3 需要 .xyz 写入掩码。
- m3x4 和 m4x4 需要 .xyz 写入掩码或默认写入掩码 (xyzw) 。

纹理寄存器掩码

在纹理坐标寄存器上使用修饰符的验证规则比其他寄存器的验证规则更严格。

- 如果编写 oTn，则所有以前的寄存器 (oTn-1 ~ oT0) 也必须编写。
- 任何 oT# 寄存器的“组合”写入掩码必须正好是以下各项之一：
  - .x

- .xy
- .xyz
- .xyzw（，这等效于不使用任何组件修饰符）

例如，顶点着色器可以在单独的指令中输出到纹理寄存器。

```
oT1.y
oT0.y
oT2
oT0.xz
oT1.x
```

或者，可以组合说明。

```
oT0.xyz
oT1.xy
oT2.xyzw
```

这些限制仅适用于纹理坐标寄存器。

相关主题

顶点着色器寄存器修饰符

寄存器 - vs_1_1

项目 • 2022/09/23

本部分包含顶点着色器版本 1_1 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	# 读取端口	# 读取/inst	维度	RelAddr	默认值	需要 DCL
a0	地址注册	1	R/W	1	无限制	请参阅注释 3	否	无	否
c#	常量浮点寄存器	查看注释 2	R	1	无限制	4	a0.x	(0、0、0、0)	否
V#	输入寄存器	16	R	1	无限制	4	否	请参阅注释 1	是
R#	临时寄存器	12	R/W	3	无限制	4	否	无	否

注意：

1. 部分 (0、0、0、1) - 如果仅更新了一部分通道，则剩余通道将默认 (0、0、0、1) 。
2. 等于 D3DCAPS9。maxVertexShaderConst (至少 96 for vs_1_1) 。
3. 只有 .x 通道可用。

输出寄存器

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
oPos	位置寄存器	1	W	4	否	无	否
oFog	雾寄存器	1	W	1	否	无	否
选择	点大小寄存器	1	W	1	否	无	否
Od#	颜色寄存器;请参阅注释 1	2	W	4	否	无	否
oT#	纹理坐标寄存器	8	W	4	否	无	否

注意：

- oD0 是漫射颜色输出;oD1 是反射颜色输出。

相关主题

顶点着色器寄存器

寄存器 - vs_2_0

项目 • 2022/09/22

本部分包含顶点着色器版本 2_0 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	# 读取端口	# 读取/inst	维度	RelAddr	默认值	需要 DCL
V#	输入寄存器	16	R	1	无限制	4	否	请参阅注释 1	是
R#	临时寄存器	12	R/W	3	无限制	4	否	无	否
c#	常量浮点寄存器	查看注释 2	R	1	2	4	a0 / aL	(0、0、0、0)	否
a0	地址注册	1	R/W	1	2	4	否	无	否
B#	常量布尔寄存器	16	R	1	1	1	否	false	否
我#	常量整数寄存器	16	R	1	1	4	否	(0、0、0、0)	否
铝	Loop计数器寄存器	1	R	1	2	1	否	无	否

注意：

1. 部分 (0、0、0、1) - 如果仅更新了一部分通道，则剩余通道将默认 (0、0、0、1) 。
2. 等于 D3DCAPS9。maxVertexShaderConst (至少 256 for vs_2_0) 。

输出寄存器

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
oPos	位置寄存器	1	W	4	否	无	否
oFog	雾寄存器	1	W	1	否	无	否
选择	点大小寄存器	1	W	1	否	无	否

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
Od#	颜色寄存器;请参阅说明 1	2	W	4	否	无	否
oT#	纹理坐标寄存器	8	W	4	否	无	否

注意：

- oD0 是漫射颜色输出;oD1 是反射颜色输出。

相关主题

顶点着色器寄存器

寄存器 - vs_2_x

项目 • 2022/09/23

本部分包含顶点着色器版本2_x实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	# 读取端口	# 读取/inst	维度	RelAddr	默认值	需要 DCL
V#	输入寄存器	16	R	1	无限制	4	否	请参阅注释 1	是
R#	临时寄存器	查看注释 2	R/W	3	无限制	4	否	无	否
c#	常量浮点寄存器	请参阅注释 3	R	1	2	4	a0 / aL	(0、0、0、0)	否
a0	地址注册	1	R/W	1	2	4	否	无	否
B#	常量布尔寄存器	16	R	1	1	1	否	false	否
我#	常量整数寄存器	16	R	1	1	4	否	(0、0、0、0)	否
铝	Loop计数器寄存器	1	R	1	2	1	否	无	否
P	谓词寄存器	1	R/W	1	1	4	否	无	否

注意：

1. 部分 (0、0、0、1) - 如果仅更新了一部分通道，则剩余通道将默认 (0、0、0、1) 。
2. 等于 D3DCAPS9。VS20Caps.NumTemps (至少 12 个用于vs_2_x) 。
3. 等于 D3DCAPS9。maxVertexShaderConst (至少 256 for vs_2_x) 。

输出寄存器

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
oPos	位置寄存器	1	W	4	否	无	否

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
oFog	雾寄存器	1	W	1	否	无	否
选择	点大小寄存器	1	W	1	否	无	否
Od#	颜色寄存器;请参阅注释 1	2	W	4	否	无	否
oT#	纹理坐标寄存器	8	W	4	否	无	否

注意：

- oD0 是漫射颜色输出;oD1 是反射颜色输出。

相关主题

顶点着色器寄存器

寄存器 - vs_3_0

项目 • 2022/09/22

本部分包含顶点着色器版本 3_0 实现的输入和输出寄存器的参考信息。

输入寄存器

注册	名称	Count	R/W	# 读取端口	# 读取/inst	维度	RelAddr	默认值	需要 DCL
V#	输入寄存器	16	R	1	无限制	4	a0/aL	请参阅注释 1	是
R#	临时寄存器	32	R/W	3	无限制	4	否	无	否
c#	常量浮点寄存器	查看注释 2	R	1	无限制	4	a0/aL	(0、0、0、0)	否
a0	地址注册	1	R/W	1	无限制	4	否	无	否
B#	常量布尔寄存器	16	R	1	1	1	否	false	否
我#	常量整数寄存器	16	R	1	1	4	否	(0、0、0、0)	否
铝	Loop计数器寄存器	1	R	1	无限制	1	否	无	否
P	谓词寄存器	1	R/W	1	1	4	否	none	否
s#	Sampler (Direct3D 9 asm-vs)	4	R	1	1	4	否	请参阅说明 3	是

注意：

1. 部分 (0、0、0、1) - 如果仅更新了一部分通道，剩余通道将默认为 (0、0、0、0、1) 。
2. 等于 D3DCAPS9。MaxVertexShaderConst (至少 256 for vs_3_0) 。
3. 采样器查找的默认值存在，但值取决于纹理格式。

输出寄存器

输出寄存器已折叠为 12 o# (输出) 寄存器。 这些可用于用户想要为像素着色器内插的任何内容：纹理坐标、颜色、雾等。

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
o#	输出寄存器	12	W	4	铝	无	是

相关主题

顶点着色器寄存器

相对寻址 (HLSL VS 引用)

项目 • 2022/09/23

[ ] 语法只能在某些着色器模型中相对寻址的寄存器类型中使用。 支持的 [ ] 语法形式如下：

其中：

- “R”表示可以相对寻址的任何寄存器类型。
- “A”表示任何可用作索引的寄存器，以相对处理其他寄存器。
- n0 - ni， m0 - mj， k 是整数 >= 0。

[ ] 语法	**有效索引**	**示例**
R[A + m0 + … + mj]	A + m0 + … + mj	c[a0.x + 3 + 7]
R[k] (= Rk)	k	c[10] (= c10)
R[A]	A	c[a0.y]
Rk[n0 + … + ni + A + m0 + … + mj]	A + k + n0 + … + ni + m0 + … + mj	c8[3 + 2 + a0.w + 5 + 6 + 1]
R[n0 + … + ni + A + m0 + … + mj]	A + n0 + … + ni + m0 + … + mj	c[2 + 1 + aL + 3 + 4 + 5]
Rk[A]	A + k	c12[aL]， c0[a0.z]
Rk[A + m0 + … + mj]	A + k + m0 + … + mj	v1[aL + 4 + 8]
R[n0 + … + ni + A]	A + n0 + … + ni	o[3 + 1 + aL]
Rk[n0 + … + ni + A]	A + k + n0 + … + ni	o1[2 + 1 + 3 + aL]

注册在以下版本中可用：

注册类型	**顶点着色器版本**
a0	全部
铝	vs_2_0及更高版本
cn	vs_1_1及更高版本
on	vs_3_0

相关主题

顶点着色器寄存器

地址注册

项目 • 2022/09/22

a0 寄存器是地址寄存器。 单个寄存器在版本vs_1_1中可用。 vs_1_1中指定为 a0.x 的地址寄存器可用作带符号整数偏移量，用于相对寻址到常量寄存器文件中。 对于版本vs_2_0及更高版本，（.x、.y、.z、.w) 的所有四个组件都可用于相对寻址。

```
c[a0.x + n]
```

地址寄存器不能由顶点着色器读取，它只能用于常量寄存器的相对寻址。 读取法律范围之外的值将返回 (0.0、0.0、0.0、0.0) 。 地址寄存器只能是 mov - 与 指令的目标。 如果将浮点数移动到整数寄存器中，则会发生舍入到最接近的转换。

所有着色器在使用前都必须初始化地址寄存器。 对于版本vs_2_0及更高版本， mova - vs 指令可以将浮点值移动到地址寄存器。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
地址注册	x	x	x	x	x	x

相关主题

顶点着色器寄存器

常量浮点寄存器 (HLSL VS 引用)

项目 • 2022/09/23

四个分量的浮点常量的顶点着色器输入寄存器。 使用 def 和SetVertexShaderConstantF 设置常量寄存器。

常量寄存器文件从顶点着色器的角度来看是只读的。 任何单个指令只能访问一个常量寄存器。 但是，该指令中的每个源都可以在读取时独立重排和否定该向量。

着色器常量的行为在 Direct3D 8 和 Direct3D 9 之间发生了变化。

- 对于 Direct3D 9，常量设置为着色器常量空间的 defx 赋值。 使用 defx 声明的常量生存期仅限于该着色器的执行。 相反，使用 API SetXXXShaderConstantX 在全局空间中初始化常量设置常量。 在调用 SetxxxShaderConstants 之前（，全局空间中的常量不会复制到对着色器) 可见的本地空间。
- 对于 Direct3D 8，使用 defx 或 API 设置的常量都为着色器常量空间赋值。 每次执行着色器时，当前着色器都会使用常量，而不考虑用于设置它们的技术。

常量寄存器被指定为绝对寄存器或相对寄存器：

```
c[n]           ; absolute
c[a0.x + n]    ; relative - supported only in version 1_1
```

因此，可以使用绝对索引或地址寄存器中的相对索引读取常量寄存器。 从范围外寄存器读取 (0.0、0.0、0.0、0.0、0.0) 。

示例

下面是在着色器中声明两个浮点常量的示例。

```
def c40, 0.0f,0.0f,0.0f,0.0f;
```

每次调用 SetVertexShader 时都会加载这些常量。

下面是使用 API 的示例。

```
    // Set up the vertex shader constants.
    {
        D3DXMATRIXA16 mat;
        D3DXMatrixMultiply( &mat, &m_matView, &m_matProj );
        D3DXMatrixTranspose( &mat, &mat );

        D3DXVECTOR4 vA( sinf(m_fTime)*15.0f, 0.0f, 0.5f, 1.0f );
        D3DXVECTOR4 vD( D3DX_PI, 1.0f/(2.0f*D3DX_PI), 2.0f*D3DX_PI, 0.05f );

        // Taylor series coefficients for sin and cos.
        D3DXVECTOR4 vSin( 1.0f, -1.0f/6.0f, 1.0f/120.0f, -1.0f/5040.0f );
        D3DXVECTOR4 vCos( 1.0f, -1.0f/2.0f, 1.0f/ 24.0f, -1.0f/ 720.0f );

        m_pd3dDevice->SetVertexShaderConstantF(  0, (float*)&mat,  4 );
        m_pd3dDevice->SetVertexShaderConstantF(  4, (float*)&vA,   1 );
        m_pd3dDevice->SetVertexShaderConstantF(  7, (float*)&vD,   1 );
        m_pd3dDevice->SetVertexShaderConstantF( 10, (float*)&vSin, 1 );
        m_pd3dDevice->SetVertexShaderConstantF( 11, (float*)&vCos, 1 );
    }
```

如果要使用 API 设置常量值，则不需要着色器声明。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
常量寄存器	x	x	x	x	x	x

相关主题

顶点着色器寄存器

常量整数寄存器 (HLSL VS 引用)

项目 • 2022/12/14

常量整数寄存器仅由循环使用 - vs 和 rep - 与。

可以使用 defi 设置它们 - vs 或 **SetVertexShaderConstantI**。

用作 循环的参数时 - 与 指令：

- .x 是迭代计数。 (代表 - vs 仅使用此组件) 。
- .y 是循环计数器的初始值。
- .z 是循环计数器的增量步骤。

着色器常量的行为在 Direct3D 8 和 Direct3D 9 之间发生了变化。

- 对于 Direct3D 9，使用 defx 将值设置为着色器常量空间的常量。 使用 defx 声明的常量的生存期仅限于执行该着色器。 相反，使用 API SetXXXShaderConstantX 在全局空间中初始化常量设置常量。 在调用 SetxxxShaderConstants 之前，全局 (空间中的常量不会复制到对着色器可见的本地空间) 。
- 对于 Direct3D 8，使用 defx 或 API 设置的常量都向着色器常量空间赋值。 每次执行着色器时，当前着色器都会使用常量，而不考虑用于设置它们的技术。

相关主题

顶点着色器寄存器

常量布尔寄存器 (HLSL VS 引用)

项目 • 2022/09/22

此寄存器是静态流控制 (指令中使用的位集合，例如，如果 bool -vselse - vsendif - - - vs) 。 其中 16 个，因此着色器可以具有 16 个独立的分支条件。 可以使用 defb 设置它们 - vs 或 **SetVertexShaderConstantB**。

着色器常量的行为在 Direct3D 8 和 Direct3D 9 之间发生了变化。

- 对于 Direct3D 9，使用 defx 将值设置为着色器常量空间的常量。 使用 defx 声明的常量的生存期仅限于执行该着色器。 相反，使用 API SetXXXShaderConstantX 在全局空间中初始化常量设置常量。 在调用 SetxxxShaderConstants 之前，全局 (空间中的常量不会复制到对着色器可见的本地空间) 。
- 对于 Direct3D 8，使用 defx 或 API 设置的常量都向着色器常量空间赋值。 每次执行着色器时，当前着色器都会使用常量，而不考虑用于设置它们的技术。

相关主题

顶点着色器寄存器

颜色寄存器

项目 • 2022/12/14

这些顶点着色器输出寄存器包含颜色值。

注册	说明
oD0	漫射颜色寄存器。
oD1	反射颜色寄存器。

oD0 值内插，并写入像素着色器颜色寄存器 0 (v0) 。 oD1 值内插并写入像素着色器颜色寄存器 1 (v1) 。 有关像素着色器颜色寄存器的详细信息，请参阅像素着色器 输入颜色寄存器 主题。

备注

示例

```
min oD0, r0, c1.x
```

相关主题

顶点着色器寄存器

雾寄存器

项目 • 2022/09/22

此顶点着色器输出寄存器包含每顶点雾色。

寄存器由确定每个寄存器的行为的属性组成。

属性	说明
名称	oFog
Count	一个矢量，其中只能使用一个组件，必须由组件掩码指定
I/O 权限	只写。

输出雾值寄存器。 该值是要内插的雾因子，然后路由到雾表。 仅使用雾的标量 x 分量。

相关主题

顶点着色器寄存器

输入寄存器

项目 • 2022/12/14

顶点着色器输入寄存器。

使用一个或多个输入顶点流从每个顶点 (的数据，) 在运行顶点着色器之前加载到顶点输入寄存器中。 输入寄存器由 16 个四分量的浮点向量组成，指定为 v0 到 v15。 这些寄存器是只读的。 输入寄存器通过顶点声明绑定到顶点数据。

以下寄存器属性控制每个寄存器的行为方式：

属性	说明
名称	v[n] - n 是可选的寄存器号。 0 是使用的默认值（如果省略）。
Count	最多 16 个寄存器，v0 - v15。
I/O 权限	只读。 此寄存器不能由 API 或着色器内编写。
读取端口	1. 这是可以在单个指令中读取寄存器的次数。 请参阅下文。

任何单个指令只能访问一个顶点输入寄存器。 但是，指令中的每个源都可以在读取时独立重排和否定该向量。

示例

下面是使用顶点声明绑定 2D 顶点位置数据以注册 v0 的示例。

顶点声明属于应用程序：

```
D3DVERTEXELEMENT9 decl[] =
{
    { 0, 0, D3DDECLTYPE_FLOAT2, D3DDECLMETHOD_DEFAULT,
D3DDECLUSAGE_POSITION, 0 },
      D3DDECL_END()
};
```

下面是相应的顶点着色器声明：

```
dcl_position v0
```

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
位置寄存器	x	x	x	x	x	x

相关主题

顶点着色器寄存器

Loop计数器寄存器 (HLSL VS 引用)

项目 • 2022/09/23

此银行中唯一的寄存器是 (aL) 寄存器的当前循环计数器。 它在循环的每个执行中自动递增 - vs...endloop - vs block。 因此，如果需要，可以在块中使用它进行相对寻址，并在循环外部使用它无效。

相关主题

顶点着色器寄存器

输出寄存器

项目 • 2022/09/23

- 顶点颜色寄存器
- 雾寄存器
- Position_Register
- Point_Size_Register
- Texture_Coordinate_Register

寄存器名称前面有小写字母 o，指示输出寄存器是仅写的。

顶点颜色寄存器 - oD0、oD1

oD0 是漫射颜色寄存器。 oD1 是反射颜色寄存器。 oD0 值是内插的，并写入到像素着色器的输入颜色寄存器 0 (v0) 。 oD1 值是内插的，并写入到像素着色器的输入颜色寄存器 1 (v1) 。 有关像素着色器颜色寄存器的详细信息，请参阅寄存器。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
顶点颜色寄存器	x	x	x	x		

雾寄存器 - oFog

输出雾值寄存器。 该值是要内插的雾因子，然后路由到雾表。 仅使用雾的标量 x 分量。 在传递给光栅器之前，值将固定在零到 1 之间。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
雾寄存器	x	x	x	x		

位置寄存器 - oPos

输出位置寄存器。 该值是同质剪辑空间中的位置。 此值必须由顶点着色器编写。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
位置寄存器	x	x	x	x		

点大小寄存器 - oPts

输出点大小寄存器。 仅使用点大小的标量 x 分量。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
点大小寄存器	x	x	x	x		

纹理坐标寄存器 - oT0 到 oT7

输出纹理坐标寄存器。 具体而言，这些是输出数据寄存器的数组，这些寄存器通过纹理采样阶段将数据路由到像素着色器，以迭代并用作纹理坐标。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
纹理坐标寄存器	x	x	x	x		

写入纹理坐标寄存器时，建议仅传递任意数量的浮点值作为相应纹理映射的维度。 控制使用修饰符传递的值。 例如，对二维纹理地图使用 .xy。

为纹理阶段启用纹理投影时，必须将所有四个浮点值写入相应的纹理寄存器。

使用可编程管道时，任何 D3DTTFF* 纹理转换标志都应为零。

纹理坐标范围

对象顶点数据提供输入纹理坐标。 不使用平铺纹理的对象通常具有范围 [0，1] 中的纹理坐标。 使用平铺纹理的对象（如地形）通常具有介于 [-?,+ ？] 范围内的纹理坐标 在哪里？ 可以是较大的浮点数。

纹理坐标内插对顶点数据执行光栅化。 在光栅化期间，纹理坐标在对象顶点之间内插，通过纹理换行和按纹理大小缩放 (还考虑到纹理地址模式) 生成整数索引。 然后，该索引用于执行纹理查找。 MaxTextureRepeat 可用于确定纹理可平铺的次数。

如果使用 texcoord 或 texcrd) 将纹理坐标直接读取到像素着色器 (，则纹理坐标范围取决于指令和像素着色器版本。

相关主题

顶点着色器寄存器

点大小寄存器

项目 • 2022/09/22

此顶点着色器输出寄存器包含每个顶点大小数据。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
点大小寄存器	x	x	x	x	x	x

寄存器由确定每个寄存器的行为方式的属性组成。

properties	说明
名称	选择
Count	1 向量，其中只能使用 1 个组件，必须由组件掩码指定。
I/O 权限	只写。

仅使用点大小的标量 x 分量。

相关主题

顶点着色器寄存器

位置寄存器

项目 • 2022/09/23

此顶点着色器输出寄存器包含每个顶点位置数据。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
位置寄存器	x	x	x	x	x	x

寄存器由确定每个寄存器的行为的属性组成。

属性	说明
名称	oPos
计数	1 向量
I/O 权限	只写。

该值是同质剪辑空间中的位置。 此值必须由顶点着色器编写。

示例

```
    dcl_position v0

    def c40, 0.0f,0.0f,0.0f,0.0f;
    // transform into projection space
    m4x4 r0,v0,c8
    max r0.z,c40.z,r0.z //clamp to 0
    max r0.w,c12.x,r0.w //clamp to near clip plane
    mov oPos,r0
```

相关主题

顶点着色器寄存器

谓词寄存器 (HLSL VS 参考)

项目 • 2022/09/22

此顶点着色器输出寄存器包含每通道布尔值。

以下版本支持谓词寄存器。

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
谓词寄存器				x	x	x

下面是寄存器属性。

注册类型	Count	R/W	# 读取端口	# Reads/inst	维度	RelAddr	默认值	需要 DCL
谓词 (p)	1	R/W	1	1	4	不可用	None	N

可以使用 setp_comp - vs. 修改谓词寄存器。此寄存器没有默认值，应用程序需要在使用寄存器之前设置寄存器。

相关主题

顶点着色器寄存器

采样器 (Direct3D 9 asm-vs)

项目 • 2022/09/22

采样器是顶点着色器的输入伪寄存器，用于标识采样阶段。 有四个顶点着色器采样器：s0 到 s3。 可以在单个着色器传递中读取四个纹理图面。

采样器 (Direct3D 9 asm-vs) 是伪寄存器，因为你无法直接读取或写入它们。

采样单元对应于纹理采样阶段，封装 **SetSamplerState** 提供的特定于采样的状态。 每个采样器唯一标识单个纹理图面，该图面使用 **SetTexture** 设置为相应的采样器。 但是，可以在多个采样器上设置相同的纹理图面。

在绘制时，无法同时将纹理设置为呈现目标和阶段的纹理。

由于有四个采样器，因此可以在单个着色器传递中读取最多四个纹理图面。 采样器可能显示为纹理加载指令中的唯一参数： texldl - vs.

在vs_3_0中，如果使用采样器，则需要使用 dcl_samplerType (sm3 - 与 asm) 指令在着色器程序的开头声明它。

顶点着色器版本	**1_1**	**2_0**	**2_sw**	**2_x**	**3_0**	**3_sw**
取样器					x	x

相关主题

顶点着色器寄存器

Vs_3_0 (DirectX HLSL) 中的顶点纹理

临时注册 (HLSL VS 参考)

项目 • 2022/09/22

顶点着色器临时寄存器用于保存中间结果。

在使用临时寄存器之前，必须先初始化临时寄存器。 每个临时寄存器都具有单写入和三次读取访问权限。 这意味着单个着色器指令可以使用多达三个临时寄存器作为输入。

不能使用临时寄存器中保留的顶点着色器前面的调用中的值。

寄存器由确定每个寄存器的行为方式的属性组成。

属性	描述
名称	r[n]。 n 是可选的寄存器号。 默认值为 0，如果未指定任何值，则为使用的值。
Count	最多 12 个寄存器。
I/O 权限	读/写。 此寄存器可由 API 或着色器读取或写入。
读取端口	可以在单个指令中读取寄存器的次数为 3。 临时寄存器是唯一可以在单个指令中读取和写入的寄存器。

每个临时寄存器都具有单写入和三次读取访问权限。 因此，指令在其输入源操作数集中可以有多达三个临时寄存器。

不能使用临时寄存器中保留的顶点着色器前面的调用中的值。 在写入临时寄存器之前从临时寄存器读取值的顶点着色器将失败 Direct3D API 调用以创建顶点着色器。

示例

下面是使用临时寄存器的示例：

```
def c4, 0,0,0,1
...
// Decompress position
mov r0.x, v0.x
mov r0.y, c4.w       // 1
mov r0.z, v0.y
mov r0.w, c4.w       // 1
```

```
// Compute theta from distance and time
mov r4.xz, r0        // xz
```

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
临时寄存器	x	x	x	x	x	x

相关主题

顶点着色器寄存器

纹理坐标寄存器 (HLSL VS 参考)

项目 • 2022/09/22

此顶点着色器输出寄存器包含每个顶点纹理坐标。

寄存器由确定每个寄存器的行为方式的属性组成。

属性	描述
名称	oT0 - oT7
Count	八个向量
I/O 权限	只写

输出纹理坐标寄存器是输出数据寄存器的数组。 寄存器数据由纹理采样阶段迭代并用作纹理坐标，以便向像素着色器提供数据。

写入纹理坐标寄存器时，建议仅传递与相应纹理地图维度相同的浮点值。 控制使用修饰符传递的值。 例如，将 .xy 用于 2D 纹理地图。

如果使用的是可编程顶点着色器，则固定函数顶点管道标志 D3DTEXTURETRANSFORMFLAGS (D3DTTFF_COUNT1、D3DTTFF_COUNT2、D3DTTFF_COUNT3、D3DTTFF_COUNT4) 应设置为零。

对象顶点数据提供输入纹理坐标。 不使用平铺纹理的对象通常具有区域 [0，1] 中的纹理坐标。 使用平铺纹理的对象（如地形）通常具有介于 [-n，+n] 的纹理坐标，其中 n 可以是任何浮点数。

纹理坐标内插对顶点数据执行光栅化。 在光栅化期间，纹理坐标在对象顶点之间内插，通过纹理换行修改，并按纹理大小缩放 (还考虑到纹理寻址模式) 生成整数索引。 然后，该索引用于执行纹理查找。 使用 D3DCAPS9 中的 MaxTextureRepeat 值来确定纹理可以平铺的次数。

示例

声明纹理坐标寄存器。

```
dcl_texcoord v7
```

将每个顶点纹理坐标复制到输出寄存器。

```
mov oT0, v7
```

顶点着色器版本	1_1	2_0	2_sw	2_x	3_0	3_sw
纹理坐标寄存器	x	x	x	x	x	x

相关主题

顶点着色器寄存器

像素着色器

项目 • 2022/09/24

- ps_1_1、ps_1_2、ps_1_3、ps_1_4
- ps_2_0
- ps_2_x
- ps_3_0
- 寄存器
- 像素着色器说明
- 像素着色器差异

相关主题

Asm 着色器参考

ps_1_1、ps_1_2、ps_1_3、ps_1_4

项目 • 2022/09/23

像素着色器汇编程序由一组指令组成，这些指令对寄存器中包含的像素数据进行操作。 操作表示为由运算符和一个或多个操作数组成的指令。 说明使用寄存器来传入和传出像素着色器 ALU 中的数据。 寄存器也可以由一些说明用来保存临时结果。

> **备注**
>
> HLSL 对像素着色器 1.x 的支持已弃用。

说明

像素着色器指令有两个主要类别：算术指令和纹理寻址指令。 算术指令修改颜色数据。 纹理寻址操作处理纹理坐标数据，在大多数情况下，对纹理进行采样。 像素着色器指令按像素运行;也就是说，他们不知道管道中的其他像素。

纹理寻址指令每个使用一个槽，但可以配对算术指令，以启用颜色组件 (RGB) 和单个槽中的 alpha 组件指令。

ps_1_1、ps_1_2、ps_1_3，ps_1_4说明 包含可用说明的列表。

启用多重采样后，像素着色器每像素仅运行一次，对于每个子像素，不运行一次。 多重采样只会增加多边形边缘的分辨率，以及深度和模具测试。 例如，如果启用了 3x3 多重采样，并且发现要光栅化的三角形覆盖特定像素的 9 个子像素中的 5 个，则像素着色器将运行一次，并将相同的颜色结果应用于所有五个子像素。

寄存器

ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器 列出了着色器 ALU 使用的不同寄存器。

修饰符

ps_1_X的修饰符 可用于更改指令的功能，或者从寄存器读取或写入到寄存器的数据。

Direct3D 9 要求中间计算在所有图面格式中至少保持 8 位精度。 对于阶段内数学，（12 位），建议在纹理阶段之间饱和度达到 8 位。 不支持可修改的舍入模式或异常。 应通过舍入到最接近的精度来支持乘法，以将精度损失保持在最低水平。

采样器计数

可用的纹理采样器数为：

- 对于 ps_1_0 - ps_1_3，最大值为 4。
- 对于ps_1_4，最大值为 6。

相关主题

像素着色器

ps_2_0

项目 • 2022/09/23

可编程像素着色器由一组对像素数据进行操作的说明组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

- ps_2_0说明 包含可用说明的列表。
- ps_2_0寄存器 列出了顶点着色器 ALU 使用的不同类型的寄存器。
- 修饰 符 用于修改指令的工作方式。
- 目标寄存器写入掩码 确定要写入的目标寄存器的哪些组件。
- 像素着色器源寄存器修饰符 在指令运行之前更改源寄存器数据。
- 源寄存器重排 可更好地控制哪些寄存器组件是读取、复制或写入的。

指令计数

着色器对最大指令计数有限制。 指令槽总数：96 个 (64 个算术和 32 个纹理) 。

采样器计数

可用的纹理采样器数为 16。

相关主题

像素着色器

ps_2_x

项目 • 2022/09/22

可编程像素着色器由一组对像素数据进行操作的指令组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

- ps_2_x说明 包含可用说明的列表。
- ps_2_x寄存器 列出了顶点着色器 ALU 使用的不同类型的寄存器。
- 修饰 符 用于修改指令的工作方式。
- 目标寄存器写入掩码 确定目标寄存器的哪些组件将被写入。
- 像素着色器源寄存器修饰符 在指令运行之前更改源寄存器数据。
- 源注册重排 可对哪些寄存器组件进行读取、复制或写入的其他控制。

动态Flow控件

DynamicFlowControlDepth 表示动态流控制指令的嵌套深度：如果、if_comp、if_pred、break - ps 和 break_comp - ps。 该值等于if_comp块的嵌套深度。 如果此上限为零，则设备不支持动态流控制指令。

临时寄存器数

设备支持的临时寄存器数。 范围从 12 到 32。

静态Flow控件嵌套深度

StaticFlowControlDepth 表示两种类型的静态流控制指令的嵌套深度：looprep / 和 callcallnz /。 循环 /rep 指令可以嵌套到 **StaticFlowControlDepth** 深度。 独立调用 /callnz 指令可以嵌套到 **StaticFlowControlDepth** 深度。

指令槽数

指令槽数可以介于 96 到最多 512 个范围内，并由 **MaxPixelShaderInstructionSlots** 指定。 可以运行的指令总数由 **MaxPixelShaderInstructionsExecuted** 定义。 这可以大于循环和子例程调用导致的指令槽数。

任意重排

如果设置了 D3DD3DPSHADERCAPS2_0_ARBITRARYSWIZZLE ，则支持任意重排。 请参阅 源注册重排。

渐变指令

如果设置了 D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS ，则支持渐变指令。 请参阅 dsx - ps、 dsy - ps 和 texldd - ps。

预测

如果设置了 D3DD3DPSHADERCAPS2_0_PREDICATION ，则支持指令谓词。 请参阅 谓词寄存器。

依赖读取限制

如果设置了 D3DD3DPSHADERCAPS2_0_NODEPENDENTREADLIMIT ，则没有依赖读取限制。

纹理指令限制

如果设置了 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT ，则纹理指令没有限制。

采样器计数

可用的纹理采样器数为 16。

相关主题

像素着色器

ps_3_0

项目 • 2022/09/22

可编程像素着色器由一组对像素数据进行操作的指令组成。 注册传入和传出 ALU 的数据。 可以应用其他控件来修改指令、结果或写入的数据。

- ps_3_0说明 包含可用说明的列表。
- ps_3_0寄存器 列出了像素着色器 ALU 使用的不同类型的寄存器。
- 修饰 符 用于修改指令的工作方式。
- 目标寄存器写入掩码 确定目标寄存器的哪些组件将被写入。
- 像素着色器源寄存器修饰符 在指令运行之前更改源寄存器数据。
- 源注册重排 可对哪些寄存器组件进行读取、复制或写入的其他控制。

新功能

添加人脸寄存器。 添加位置寄存器。 颜色寄存器 (v#) 现已完全浮点，纹理坐标寄存器 (t#) 已合并。 输入声明采用使用名称，给定寄存器的组件允许使用多个用法。

动态Flow控件

如果 bool - ps、break - ps 和 break_comp - ps) ，设备支持动态流控制 (。 嵌套深度范围为 0 到 24。

临时寄存器数

支持的临时寄存器数为 32。

静态Flow控件嵌套深度

调用 -pscallnz //call_pred可以嵌套到最大深度为 4。 独立循环 -psrep - ps/ 指令可以嵌套到最大深度为 4。

任意重排

支持任意重排。 请参阅 源注册重排。

渐变指令

支持渐变指令。 请参阅 dsx - ps、 dsy - ps 和 texldd - ps。

预测

支持指令谓词。 请参阅 谓词寄存器。

依赖读取限制

没有依赖读取限制。

纹理指令限制

纹理指令没有限制。

指令计数

允许每个像素着色器从 512 到 MaxPixelShader30InstructionSlots 中的槽数， (不超过 32768) 。 由于循环支持，运行指令的数量可能要高得多。
MaxPShaderInstructionsExecuted 应至少为 2^16。

采样器计数

可用的纹理采样器数为 16。

设备上限

如果支持ps_3_0，硬件 (至少支持以下上限) ：

笔帽	值
MaxTextureWidth、MaxTextureHeight	每个 4K 个
MaxTextureRepeat	8K
MaxAnisotropy	16
PixelShaderVersion	3_0
MaxPixelShader30InstructionSlots	512

笔帽	值
设置了以下基元上限：	D3DPMISCCAPS_BLENDOP、D3DPMISCCAPS_CLIPPLANESCALEDPOINTS、D3DPMISCCAPS_CLIPTLVERTS、D3DPMISCCAPS_CULLCCW、D3DPMISCCAPS_CULLCW、D3DPMISCCAPS_CULLNONE、D3DPMISCCAPS_FOGINFVF、D3DPMISCCAPS_MASKZ
设置了以下光栅上限：	D3DPRASTERCAPS_MIPMAPLODBIAS、D3DPRASTERCAPS_ANISOTROPY、D3DPRASTERCAPS_COLORPERSPECTIVE、D3DCAPS9 中的 D3DPRASTERCAPS_SCISSORTEST
完全支持深度偏差，包括：	D3DPRASTERCAPS_SLOPESCALEDEPTHBIAS，D3DPRASTERCAPS_DEPTHBIAS
深度和 alpha 测试的完整比较集，包括：	D3DCAPS9 中的所有 D3DPCMPCAPS。
源混合模式	除D3DPBLENDCAPS_SRCALPHASAT、D3DPBLENDCAPS_BOTHSRCALPHA和D3DPBLENDCAPS_BOTHINVSRCALPHA) 外，支持所有混合模式作为源 (。
支持以下纹理上限：	D3DPTEXTURECAPS_CUBEMAP、D3DPTEXTURECAPS_MIPCUBEMAP、D3DPTEXTURECAPS_MIPMAP、D3DPTEXTURECAPS_MIPVOLUMEMAP、D3DPTEXTURECAPS_PERSPECTIVE、D3DPTEXTURECAPS_PROJECTED、D3DPTEXTURECAPS_TEXREPEATNOTSCALEDBYSIZE、D3DPTEXTURECAPS_VOLUMEMAP
纹理筛选器帽、卷纹理筛选器帽和多维数据集纹理筛选器上限支持以下内容：	D3DPTFILTERCAPS_MINFPOINT、D3DPTFILTERCAPS_MINFLINEAR、D3DPTFILTERCAPS_MINFANISOTROPIC (VolumeTextureFilterCaps 和 CubeTextureFilterCaps) 、D3DPTFILTERCAPS_MIPFPOINT、D3DPTFILTERCAPS_MIPFLINEAR、D3DPTFILTERCAPS_MAGFPOINT、D3DPTFILTERCAPS_MAGFLINEAR
顶点和像素阶段支持以下纹理地址模式：	D3DPTADDRESSCAPS_WRAP、D3DPTADDRESSCAPS_MIRROR、D3DPTADDRESSCAPS_CLAMP、D3DPTADDRESSCAPS_BORDER、D3DPTADDRESSCAPS_INDEPENDENTUV、D3DPTADDRESSCAPS_MIRRORONCE

笔帽	值
支持所有像素着色器上限。	DynamicFlowControlDepth = 24，NumTemps = 32，StaticFlowControlDepth = 4，NumInstructionSlots = 512。 支持以下功能：谓词、任意重排和渐变指令。 没有依赖读取限制，对纹理和数学指令的混合没有限制。
支持所有模具操作。 这包括两侧模具。	请参阅 **D3DSTENCILOP**
每个顶点的设备支持点大小	D3DCAPS9 中的D3DFVFCAPS_PSIZE
2 个纹理支持的非电源。	完全支持或条件非 pow-2 支持;设备不应仅具有方形纹理限制，如D3DPTEXTURECAPS_SQUAREONLY所示。
如果设备支持多个呈现目标，则支持以下上限：	D3DPMISCCAPS_INDEPENDENTWRITEMASKS，D3DPMISCCAPS_MRTPOSTPIXELSHADERBLENDING
如果支持vs_3_0	D3DCAPS9 中的 MaxUserClipPlanes 为 6

相关主题

像素着色器

像素着色器差异

项目 • 2022/09/22

指令槽

每个版本都支持不同的最大指令槽数。

版本	最大指令槽数
ps_1_1	4 纹理 + 8 算术
ps_1_2	4 纹理 + 8 算术
ps_1_3	4 纹理 + 8 算术
ps_1_4	每个阶段 6 个纹理 + 8 个算术
ps_2_0	32 纹理 + 64 算术
ps_2_x	96 个最小值，最大为 D3DCAPS9 中的槽数。 D3DPSHADERCAPS2_0.NumInstructionSlots。 请参阅D3DPSHADERCAPS2_0。
ps_3_0	最小 512 个，最多是 D3DCAPS9 中的槽数。MaxPixelShader30InstructionSlots。 请参阅 D3DPSHADERCAPS2_0。

有关软件着色器限制的信息，请参阅 软件着色器。

Flow控制嵌套限制

- 请参阅Flow控制限制。

ps_1_x功能

新说明：

请参阅 ps_1_1、ps_1_2、ps_1_3、ps_1_4说明。

新寄存器：

请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4注册。

ps_2_0功能

新功能：

- 三个新的重排 - .yzxw、.zxyw、.wzyx
- 临时注册 (r#) 增加到 12
- 常量浮点寄存器数 (c#) 增加到 32
- 纹理 坐标寄存器数 (t#) 增加到 8

新说明：

- 设置说明 - dcl - (sm2， sm3 - ps asm) ， dcl_samplerType (sm2， sm3 - ps asm)
- 算术指令 - abs - ps， crs - ps， dp2add - ps， exp - ps， frc - ps， log - ps， m3x2 - ps， m3x3 - ps， m3x4 - ps， m4x3 - ps， max - ps， max - ps， nrm - ps， pow - ps， rcp - ps， rsq -ps， sincos - ps
- 纹理指令 - texld - ps_2_0和向上 (不同的语法) ， texldb - ps， texldp - ps

新寄存器：

- Sampler (Direct3D 9 asm-ps) (s#)

ps_2_x功能

新功能 (请参阅 **D3DPSHADERCAPS2_0**.) ：

- 动态流控制
- 静态流控制
- 动态和静态流控制指令的嵌套
- (r#) 增加 的临时寄存器数
- 任意源重排
- 渐变指令
- 预测
- 无依赖纹理读取限制
- 无纹理指令限制

新说明：

- 静态流控制指令 - 如果 bool - ps， call - ps， callnz bool -ps， else - ps， endif - ps， rep - ps， endrep - ps， label - ps， ret - ps
- 动态流控制指令 - break - ps， break_comp - ps， breakp - ps， callnz pred - ps， if_comp - ps， if pred - ps， setp_comp - ps
- 算术指令 - dsx - ps， dsy - ps
- 纹理指令 - texldd - ps

新寄存器：

- 谓词注册 (p0)

ps_3_0功能

新功能：

- 合并的 10 个 输入寄存器 (v#)
- Loop计数器寄存器 (aL) ，可索引输入颜色寄存器 (v#)
- (r#) 临时寄存器数增加到 32
- 常量浮点寄存器数 (c#) 增加到 224

新说明：

- 设置说明 - dcl_semantics (sm3 - ps asm)
- 静态流指令 - 循环 - ps、 endloop - ps
- 算术指令 - sincos - ps (新语法)
- 纹理指令 - texldl - ps

新寄存器：

- 输入寄存器 (v#)
- Position Register (vPos)
- 人脸注册 (vFace)

相关主题

像素着色器

像素着色器说明

项目 • 2022/12/14

- ps_1_1、ps_1_2、ps_1_3、ps_1_4 说明
- ps_2_0说明
- ps_2_x说明
- ps_3_0说明

- ps_1_X的修饰符
- Flow控制限制

相关主题

像素着色器

ps_1_1、ps_1_2、ps_1_3、ps_1_4说明

项目 • 2022/09/22

本部分包含像素着色器版本1_X说明的参考信息。

有多种类型的像素着色器指令，如下表所示。

指令集

版本	说明	指令槽	1_1	1_2	1_3	1_4
ps	版本号	0	x	x	x	x
常量指令			1_1	1_2	1_3	1_4
def - ps	定义常量	0	x	x	x	x
阶段说明			1_1	1_2	1_3	1_4
阶段 - ps	阶段 1 和阶段 2 之间的转换	0				x
算术指令			1_1	1_2	1_3	1_4
add - ps	添加两个向量	1	x	x	x	x
bem - ps	应用假颠簸环境地图转换	2				x
cmp - ps	将源与 0 进行比较	1 次		x	x	x
cnd - ps	将源与 0.5 进行比较	1	x	x	x	x
dp3 - ps	三分量点积	1	x	x	x	x
dp4 - ps	四分量点积	1 次		x	x	x
lrp - ps	线性内插	1	x	x	x	x
mad - ps	相乘和添加	1	x	x	x	x
mov - ps	移动	1	x	x	x	x
mul - ps	乘	1	x	x	x	x
nop - ps	无操作	0	x	x	x	x
sub - ps	减	1	x	x	x	x
纹理说明			1_1	1_2	1_3	1_4

版本	说明	指令槽	1_1	1_2	1_3	1_4
tex - ps	采样纹理	1	x	x	x	
texbem - ps	应用假颠簸环境映射转换	1	x	x	x	
texbeml - ps	使用亮度更正应用假颠簸环境地图转换	1+1 Ctrl	x	x	x	
texcoord - ps	将纹理坐标数据解释为颜色数据	1	x	x	x	
texcrd - ps	将纹理坐标数据复制为颜色数据	1				x
texdepth - ps	计算深度值	1				x
texdp3 - ps	纹理数据和纹理坐标之间的三分量点积	1		x	x	
texdp3tex - ps	三分量点产品和 1D 纹理查找	1		x	x	
texkill - ps	根据比较取消像素的呈现	1	x	x	x	x
texld - ps_1_4	采样纹理	1				x
texm3x2depth - ps	计算每像素深度值	1			x	
texm3x2pad - ps	双行矩阵相乘的第一行矩阵	1	x	x	x	
texm3x2tex - ps	双行矩阵相乘的最终行矩阵乘法	1	x	x	x	
texm3x3 - ps	3x3 矩阵乘法	1		x	x	
texm3x3pad - ps	三行矩阵相乘的第一行或第二行乘法	1	x	x	x	
texm3x3spec - ps	三行矩阵乘法的最后一行乘法	1	x	x	x	
texm3x3tex - ps	使用 3x3 矩阵相乘查找纹理	1	x	x	x	
texm3x3vspec - ps	纹理查找使用 3x3 矩阵乘以非常量眼光矢量	1	x	x	x	
texreg2ar - ps	使用 alpha 和红色组件对纹理进行采样	1	x	x	x	
texreg2gb - ps	使用绿色和蓝色组件对纹理进行采样	1	x	x	x	
texreg2rgb - ps	使用红色、绿色和蓝色组件采样纹理	1		x	x	

1. ps_1_4中的 1 个槽;ps_1_2 和 ps_1_3 中的 2 个槽
2. 1 + 1 = 1 个算术指令 + 1 个纹理指令

相关主题

像素着色器说明

ps_2_0说明

项目 • 2022/09/23

本部分包含像素着色器版本 2_0 指令的参考信息。

有多种类型的像素着色器指令，如表所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 设置 - 像素着色器必须具有版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- 纹理 - 这些指令用于加载和采样纹理数据，以及修改纹理坐标。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	纹理	新建
abs - ps	绝对值	1		x		x
add - ps	添加两个向量	1		x		
cmp - ps	将源与 0 进行比较	1		x		
crs - ps	跨产品	2		x		x
dcl_samplerType (sm2，sm3 - ps asm)	声明采样器的纹理维度	0	x			x
dcl - (sm2， sm3 - ps asm)	声明顶点着色器输出寄存器与像素着色器输入寄存器之间的关联。	0	x			x
def - ps	定义常量	0	x			
dp2add - ps	2D 点产品并添加	2		x		x
dp3 - ps	三维点产品	1		x		
dp4 - ps	4D 点产品	1		x		
exp - ps	全精度 2^x	1		x		x
frc - ps	小数部分	1		x		x
log - ps	全精度 log一 (x)	1		x		x

名称	说明	指令槽	设置	算术	纹理	新建
lrp - ps	线性内插	2		x		
m3x2 - ps	3x2 乘法	2		x		x
m3x3 - ps	3x3 乘法	3		x		x
m3x4 - ps	3x4 乘法	4		x		x
m4x3 - ps	4x3 乘法	3		x		x
m4x4 - ps	4x4 乘法	4		x		x
mad - ps	乘法和添加	1		x		
max - ps	最大值	1		x		x
min - ps	最小值	1		x		x
mov - ps	移动	1		x		
mul - ps	乘	1		x		
nop - ps	无操作	1		x		
nrm - ps	规范化	3		x		x
pow - ps	^{xy}	3		x		x
ps	版本	0	x			
rcp - ps	互惠	1		x		x
rsq - ps	倒数方根	1		x		x
sincos - ps	正弦和余弦	8		x		x
sub - ps	减	1		x		
texkill - ps	终止像素呈现	1			x	
texld - ps_2_0和向上	采样纹理	1			x	x
texldb - ps	具有 w-component 中细节级别的偏差的纹理采样	1			x	x
texldp - ps	具有投影除以 w-component 的纹理采样	1			x	x

相关主题

像素着色器说明

ps_2_x说明

项目 • 2022/09/22

本部分包含像素着色器版本2_x说明的参考信息。

有多种类型的像素着色器指令，如表所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 设置 - 像素着色器必须具有版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- 纹理 - 这些指令用于加载和采样纹理数据，以及修改纹理坐标。
- Flow控制 - 这些指令为指令的执行提供静态和动态流控制。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
abs - ps	绝对值	1		x			
add - ps	添加两个向量	1		x			
break - ps	分手代表...endrep 块	1				x	x
break_comp - ps	有条件地突破代表...endrep 块，带有比较	3				x	x
breakp - ps	分手代表...endrep 块，基于谓词	3				x	x
call - ps	调用子例程	2				x	x
callnz bool - ps	如果布尔寄存器不为零，则调用子例程	3				x	x
callnz pred - ps	如果谓词寄存器不为零，则调用子例程	3				x	x
cmp - ps	将源与 0 进行比较	1		x			
crs - ps	跨产品	2		x			
dcl_samplerType (sm2， sm3 - ps asm)	声明采样器的纹理维度	0	x				

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
dcl - (sm2 ， sm3 - ps asm)	声明顶点着色器输出寄存器与像素着色器输入寄存器之间的关联。	0	x				
def - ps	定义常量	0	x				
defb - ps	定义布尔常量	0	x				x
defi - ps	定义整数常量	0	x				x
dp2add - ps	2D 点产品并添加	2		x			
dp3 - ps	三维点产品	1		x			
dp4 - ps	4D 点产品	1		x			
dsx - ps	x 方向变化率	2		x			x
dsy - ps	y 方向的变化率	2		x			x
else - ps	开始其他块	1				x	x
endif - ps	结束 if...else 块	1				x	x
endrep - ps	重复块的结束	2				x	x
exp - ps	全精度 2^x	1		x			
frc - ps	小数部分	1		x			
if bool - ps	开始 if 块	3				x	x
if_comp - ps	使用比较开始 if 块	3				x	x
如果 pred - ps	使用谓词开始 if 块	3				x	x
label - ps	标签	0				x	x
log - ps	全精度 log一 (x)	1		x			
lrp - ps	线性内插	2		x			
m3x2 - ps	3x2 乘法	2		x			
m3x3 - ps	3x3 乘法	3		x			
m3x4 - ps	3x4 乘法	4		x			
m4x3 - ps	4x3 乘法	3		x			

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
m4x4 - ps	4x4 乘法	4		x			
mad - ps	乘法和添加	1		x			
max - ps	最大值	1		x			
min - ps	最小值	1		x			
mov - ps	移动	1		x			
mul - ps	乘	1		x			
nop - ps	无操作	1		x			
nrm - ps	规范化	3		x			
pow - ps	^{xy}	3		x			
ps	版本	0	x				
rcp - ps	互惠	1		x			
rep - ps	Repeat	3				x	x
ret - ps	子例程结束	1				x	x
rsq - ps	倒数方根	1		x			
setp_comp	设置谓词寄存器	1				x	x
sincos - ps	正弦和余弦	8		x			
sub - ps	减	1		x			
texkill - ps	终止像素呈现	请参阅注释 1			x		
texld - ps_2_0和向上	采样纹理	查看注释 2			x		
texldb - ps	具有 w-component 中细节级别的偏差的纹理采样	请参阅注释 3			x		
texldd - ps	使用用户提供的渐变进行纹理采样	3			x		x
texldp - ps	具有投影除以 w-component 的纹理采样	请参阅注释 4			x		

注意：

1. 如果设置了 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT ，槽数 = 2;否则，槽数 = 1。
2. 如果设置了 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT 并且纹理是多维数据集映射，则槽 = 4;否则，槽 = 1。
3. 如果设置了 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT ，槽数 = 6;否则，槽数 = 1。
4. 如果未设置 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT ，槽数 = 1;否则：

    - 如果设置了 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT 并且纹理是多维数据集映射，则槽数 = 4。
    - 如果设置了 D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT 且纹理不是多维数据集映射，则槽数 = 3。

相关主题

像素着色器说明

ps_3_0说明

项目 • 2022/09/23

本部分包含像素着色器版本 3_0 指令的参考信息。

有多种类型的像素着色器指令，如表所示。 右侧的列表示以下内容：

- 指令槽 - 每个指令使用的指令槽数。
- 设置 - 像素着色器必须具有版本指令，并且必须是第一个指令。
- 算术 - 这些指令在着色器中提供数学运算。
- 纹理 - 这些指令用于加载和采样纹理数据，以及修改纹理坐标。
- Flow控制 - 这些指令为指令的执行提供静态和动态流控制。
- 新增 - 这些说明是此版本的新增功能。

指令集

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
abs - ps	绝对值	1		x			
add - ps	添加两个向量	1		x			
break - ps	中断循环...endloop 或 rep...endrep 块	1				x	
break_comp - ps	有条件地中断循环...endloop 或 rep...endrep 块，带有比较	3				x	
breakp - ps	中断循环...endloop 或 rep...endrep 块，基于谓词	3				x	
call - ps	调用子例程	2				x	
callnz bool - ps	如果布尔寄存器不为零，则调用子例程	3				x	
callnz pred - ps	如果谓词寄存器不为零，则调用子例程	3				x	
cmp - ps	将源与 0 进行比较	1		x			
crs - ps	跨产品	2		x			

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
dcl_samplerType (sm2， sm3 - ps asm)	声明采样器的纹理维度	0	x				
dcl_semantics (sm3 - ps asm)	声明输入和输出寄存器	0	x				x
def - ps	定义常量	0	x				
defb - ps	定义布尔常量	0	x				
defi - ps	定义整数常量	0	x				
dp2add - ps	2D 点产品并添加	2		x			
dp3 - ps	三维点产品	1		x			
dp4 - ps	4D 点产品	1		x			
dsx - ps	x 方向变化率	2		x			
dsy - ps	y 方向的变化率	2		x			
else - ps	开始其他块	1				x	
endif - ps	结束 if...else 块	1				x	
endloop - ps	结束循环	2				x	x
endrep - ps	重复块的结束	2				x	
exp - ps	全精度 2^x	1		x			
frc - ps	小数部分	1		x			
if bool - ps	开始 if 块	3				x	
if_comp - ps	使用比较开始 if 块	3				x	
如果 pred - ps	使用谓词开始 if 块	3				x	
label - ps	标签	0				x	
log - ps	全精度 log一 (x)	1		x			
循环 - ps	Loop	3				x	x
lrp - ps	线性内插	2		x			

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
m3x2 - ps	3x2 乘法	2		x			
m3x3 - ps	3x3 乘法	3		x			
m3x4 - ps	3x4 乘法	4		x			
m4x3 - ps	4x3 乘法	3		x			
m4x4 - ps	4x4 乘法	4		x			
mad - ps	乘法和添加	1		x			
max - ps	最大值	1		x			
min - ps	最小值	1		x			
mov - ps	移动	1		x			
mul - ps	乘	1		x			
nop - ps	无操作	1		x			
nrm - ps	规范化	3		x			
pow - ps	^{xy}	3		x			
ps	版本	0	x				
rcp - ps	互惠	1		x			
rep - ps	Repeat	3				x	
ret - ps	子例程结束	1				x	
rsq - ps	倒数方根	1		x			
setp_comp	设置谓词寄存器	1				x	
sincos - ps	正弦和余弦	8		x			
sub - ps	减	1		x			
texkill - ps	终止像素呈现	2			x		
texld - ps_2_0和向上	采样纹理	请参阅注释 1			x		

名称	说明	指令槽	设置	算术	纹理	流量控制	新建
texldb - ps	具有 w-component 中细节级别的偏差的纹理采样	6			x		
texldl - ps	使用 w-component 中的详细级别进行纹理采样	查看注释 2			x		x
texldd - ps	使用用户提供的渐变进行纹理采样	3			x		
texldp - ps	具有投影除以 w-component 的纹理采样	请参阅注释 3			x		

注意：

1. 如果纹理是多维数据集地图，则槽 = 4;否则，槽数 = 1。
2. 如果纹理是多维数据集地图，则槽 = 5;否则，槽数 = 2。
3. 如果纹理是多维数据集地图，则槽 = 4;否则，槽数 = 3。

相关主题

像素着色器说明

ps_1_X的修饰符

项目 • 2022/09/22

指令修饰符在写入目标寄存器之前影响指令的结果。 例如，使用它们将结果乘以或除以两个因子，或将结果固定在零和一之间。 指令修饰符在指令运行后应用，但在将结果写入目标寄存器之前。

下面显示了修饰符的列表。

修饰符	说明	语法	版本 1_1	版本 1_2	版本 1_3	版本 1_4
_x2	乘以 2	instruction_x2	X	X	X	X
_x4	乘以 4	instruction_x4	X	X	X	X
_x8	乘以 8	instruction_x8				X
_d2	除以 2	instruction_d2	X	X	X	X
_d4	除以 4	instruction_d4				X
_d8	除以 8	instruction_d8				X
_sat	饱和 (夹从 0 和 1)	instruction_sat	X	X	X	X

- 乘法修饰符在读取寄存器数据后乘以两个幂。 这与左移相同。
- 除法修饰符在读取寄存器数据后将寄存器数据除以两个幂。 这与右移相同。
- 饱和修饰符将寄存器值的范围从零固定到一个。

指令修饰符可用于算术指令。 它们不能用于纹理地址指令。

乘法修饰符

此示例使用源操作数 (src0 和 src) 1 中的两种颜色之和加载目标寄存器 (dest) ，并将结果乘以两。

```
add_x2 dest, src0, src1
```

此示例合并了两个指令修饰符。 首先，将添加源操作数中的两种颜色， (src0 和 src1) 。 然后，结果乘以 2，并为每个组件固定在 0.0 到 1.0 之间。 结果保存在目标寄存器中。

```
add_x2_sat dest, src0, src1
```

除号修饰符

此示例使用源操作数 (src0 和 src) 1 中的两种颜色之和加载目标寄存器 (dest) ，并将结果除以两个。

```
add_d2 dest, src0, src1
```

饱和修饰符

对于算术指令，饱和修饰符会将此指令的结果固定到每个组件的 0.0 到 1.0 范围内。 以下示例演示如何使用此指令修饰符。

```
dp3_sat r0, t0_bx2, v0_bx2    ; t0 is bump, v0 is light direction
```

此操作发生在任何乘法或除指令修饰符之后。 _sat最常用于固定点产品结果。 但是，它还支持对多通道方法进行一致的模拟，其中帧缓冲区始终在 0 到 1 范围内，以及 DirectX 6 和 7.0 多文本语法的一致模拟，其中定义饱和度在每个阶段发生。

此示例使用源操作数 (src0 和 src1) 中的两种颜色之和加载目标寄存器 (dest) ，并将结果固定到每个组件的 0.0 到 1.0 范围内。

```
add_sat dest, src0, src1
```

此示例合并了两个指令修饰符。 首先，将添加源操作数中的两种颜色， (src0 和 src1) 。 结果乘以两个，每个组件的固定介于 0.0 到 1.0 之间。 结果保存在目标寄存器中。

```
add_x2_sat dest, src0, src1
```

相关主题

像素着色器说明

ps_2_0及更高版本的修饰符

项目 • 2022/12/14

指令修饰符在写入目标寄存器之前影响指令的结果。

本部分包含由像素着色器版本 2_0 及更高版本实现的指令修饰符的参考信息。

名称	语法	2_0	2_x	2_sw	3_0	3_sw
质心	_centroid	x	x	x	x	x
Partial_Precision	_pp	x	x	x	x	x
饱和	_sat	x	x	x	x	x

质心

质心修饰符是可选的修饰符，当基元未覆盖多采样像素中心时，将属性内插固定在基元范围内。 这可以在 质心采样 中看到。

可以将质心修饰符应用于程序集指令，如下所示：

```
dcl_texcoord0_centroid v0
```

还可以将质心修饰符应用于语义，如下所示：

```
float4 TexturePointCentroidPS( float4 TexCoord : TEXCOORD0_centroid ) :
COLOR0
{
    return tex2D( PointSampler, TexCoord );
}
```

此外，使用颜色语义声明的任何 输入颜色寄存器 (v#) 将自动应用质心。 从质心采样的属性计算的渐变不能保证准确。

部分精度

部分精度指令修饰符 (_pp) 指示部分精度可接受的区域，前提是基础实现支持它。 实现始终可以随意忽略修饰符，并完全精确执行受影响的操作。

_pp修饰符可以在两个上下文中发生：

- 在纹理坐标声明中，启用以部分精度形式将纹理坐标传递到像素着色器。 例如，这允许使用纹理坐标将颜色数据中继到像素着色器，这可能比某些实现中的全精度更快。 如果没有此修饰符，纹理坐标必须以完全精度传递。
- 在任何指令上，包括纹理加载指令。 这表示允许实现以部分精度执行指令，并存储部分精度结果。 在没有显式修饰符的情况下，必须以完全精度 (执行指令，而不考虑输入精度) 。

示例：

```
dcl_texcoord0_pp t1
cmp_pp r0, r1, r2, r3
```

饱和度

饱和指令修饰符 (_sat) 饱和 (或固定) 指令结果到范围 [0， 1] 之前写入目标寄存器。

_sat指令修饰符可与 frc - ps、 sincos - ps 和任何 tex* 指令以外的任何指令一起使用。

对于ps_2_0、ps_2_x和ps_2_sw，_sat指令修饰符不能与写入任何输出寄存器的说明一起使用， (输出颜色寄存器 或 输出深度寄存器) 。 此限制不适用于ps_3_0及更高版本。

例如：

```
dp3_sat r0, v0, c1
```

相关主题

像素着色器说明

Flow控制限制

项目 • 2022/12/14

像素着色器流控制指令具有限制，会影响指令中可以包含多少级别的嵌套。 此外，使用渐变指令实现每像素流控制还有一些限制。

> **备注**
>
> 使用 *_4_0_level_9_x HLSL 着色器配置文件时，隐式使用 **着色器模型** 2.x 配置文件来支持支持 Direct3D 9 的硬件。 着色器模型 2.x 配置文件支持比 **着色器模型** 4.x 及更高版本的配置文件更有限的流控制行为。

- 像素着色器指令深度计数
- Per-Pixel Flow控件与屏幕渐变的交互

像素着色器指令深度计数

ps_2_0不支持流控制。 下面列出了其他像素着色器版本的限制。

ps_2_x指令深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 下表列出了每个指令从现有深度添加或减去的深度计数。

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套
if bool - ps	1	0	0	0
if_comp - ps	0	1	0	0
如果 pred - ps	0	1	0	0
else - ps	0	0	0	0
endif - ps	-1 (布尔 - ps)	- (1 如果 pred - ps 或 if_comp - ps)	0	0
rep - ps	0	0	1	0
endrep - ps	0	0	-1	0
break - ps	0	0	0	0

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套
break_comp - ps	0	1, -1	0	0
breakp - ps	0	0	0	0
call - ps	0	0	0	1
callnz bool - ps	0	0	0	1
callnz pred - ps	0	1	0	1
ret - ps	0	-1 (callnz pred - ps)	0	-1
setp_comp - ps	0	0	0	0

嵌套深度

嵌套深度定义可从彼此内部调用指令数。 每种类型的指令都有一个或多个嵌套限制，如下表所示。

指令类型	最大值
静态嵌套	如果 (D3DCAPS9，则为 24。D3DPSHADERCAPS2_0.StaticFlowControlDepth > 0) ; 0 否则为 0
动态嵌套	0 到 24，请参阅 D3DCAPS9。D3DPSHADERCAPS2_0.DynamicFlowControlDepth
rep 嵌套	0 到 4，请参阅 D3DCAPS9。D3DPSHADERCAPS2_0.StaticFlowControlDepth
调用嵌套	0 到 4，请参阅 D3DCAPS9。D3DPSHADERCAPS2_0.StaticFlowControlDepth (独立于代表限制)

ps_2_sw指令深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 此表显示每个指令从现有深度添加或减去的深度计数。

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套
if bool - ps	1	0	0	0
如果 pred - ps	0	1	0	0
if_comp - ps	0	1	0	0
else - ps	0	0	0	0
endif - ps	-1 (布尔 - ps)	- (1 如果 pred - ps 或 if_comp - ps)	0	0
rep - ps	0	0	1	0
endrep - ps	0	0	-1	0
循环 - ps	不适用	不适用	不适用	不适用
endloop - ps	不适用	不适用	不适用	不适用
break - ps	0	0	0	0
break_comp - ps	0	1, -1	0	0
breakp - ps	0	0	0	0
call - ps	0	0	0	1
callnz bool - ps	0	0	0	1
callnz pred - ps	0	1	0	1
ret - ps	0	-1 (callnz pred - ps)	0	-1
setp_comp - ps	0	0	0	0

嵌套深度

嵌套深度定义可从彼此内部调用的指令数。 每种类型的指令都有一个或多个嵌套限制，如下表所示。

指令类型	最大值
静态嵌套	24
动态嵌套	24
rep 嵌套	4

指令类型	最大值
调用嵌套	4

ps_3_0指令深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 此表显示每个指令从现有深度添加或减去的深度计数。

指令	静态嵌套	动态嵌套	循环/rep 嵌套	调用嵌套
if bool - ps	1	0	0	0
如果 pred - ps	0	1	0	0
if_comp - ps	0	1	0	0
else - ps	0	0	0	0
endif - ps	-1 (布尔 - ps)	- (1 如果 pred - ps 或 if_comp - ps)	0	0
rep - ps	0	0	1	0
endrep - ps	0	0	-1	0
循环 - ps	0	0	1	0
endloop - ps	0	0	-1	0
break - ps	0	0	0	0
break_comp - ps	0	1, -1	0	0
breakp - ps	0	0	0	0
call - ps	0	0	0	1
callnz bool - ps	0	0	0	1
callnz pred - ps	0	1	0	1
ret - ps	0	-1 (callnz pred - ps)	0	-1
setp_comp - ps	0	0	0	0

嵌套深度

嵌套深度定义可从彼此内部调用的指令数。 每种类型的指令都有一个或多个嵌套限制，如下表所示。

指令类型	最大值
静态嵌套	24
动态嵌套	24
loop/rep 嵌套	4
调用嵌套	4

ps_3_sw指令深度计数

每个指令针对一个或多个嵌套深度限制进行计数。 此表显示每个指令从现有深度添加或减去的深度计数。

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套
if bool - ps	1	0	0	0
if pred - ps	0	1	0	0
if_comp - ps	0	1	0	0
else - ps	0	0	0	0
endif - ps	-1 (如果 bool - ps)	-1 ((如果 pred - ps 或 if_comp - ps)	0	0
rep - ps	0	0	1	0
endrep - ps	0	0	-1	0
loop - ps	0	0	1	0
endloop - ps	0	0	-1	0
break - ps	0	0	0	0
break_comp - ps	0	1, -1	0	0

指令	静态嵌套	动态嵌套	loop/rep 嵌套	调用嵌套
breakp - ps	0	0	0	0
call - ps	0	0	0	1
callnz bool - ps	0	0	0	1
callnz pred - ps	0	1	0	1
ret - ps	0	-1 (callnz pred - ps)	0	-1
setp_comp - ps	0	0	0	0

嵌套深度

嵌套深度定义可从彼此内部调用的指令数。 每种类型的指令都有一个或多个嵌套限制，如下表所示。

指令类型	最大值
静态嵌套	24
动态嵌套	24
loop/rep 嵌套	4
调用嵌套	4

Per-Pixel Flow控件与屏幕渐变的交互

像素着色器指令集包含多个指令，这些指令生成或使用与屏幕空间 x 和 y 相关的数量渐变。 渐变的最常见用途是计算纹理采样的详细级别计算，在异性筛选的情况下，选择各向异性轴上的样本。 通常，硬件实现在多个像素上运行像素着色器，同时（（如 2x2 网格）），以便着色器中计算的数量渐变可以合理近似为相邻像素中相同执行点的值增量。

当流控制存在于着色器中时，给定分支路径中请求的渐变计算结果不明确，当相邻像素可能执行单独的流控制路径时。 因此，使用任何像素着色器操作是非法的，该操作请求在流控制构造内的某个位置进行渐变计算，该操作可能会因给定基元光栅化而有所不同。

所有像素着色器指令都分区到允许的操作以及流控制中不允许的操作中：

- 方案 A：不允许在流控件内执行的操作，这些操作可能因基元中的像素而异。 其中包括下表中列出的操作。

指令	在Flow控件中允许在：
texld - ps_2_0和 up、 texldb - ps 和 texldp - ps	临时寄存器用于纹理坐标。
dsx - ps 和 dsy - ps	临时寄存器用于操作数。

- 方案 B：允许任何位置的操作。 其中包括下表中列出的操作。

指令	允许在：
texld - ps_2_0和 up、 texldb - ps 和 texldp - ps	只读数量用于纹理坐标 (可能会因像素而异，例如内插纹理坐标) 。
dsx - ps 和 dsy - ps	只读数量用于输入操作数 (可能会因像素而异，例如内插纹理坐标) 。
texldl - ps	用户提供详细级别作为参数，因此没有渐变，因此流控制没有问题。
texldd - ps	用户提供渐变作为输入参数，因此流控制没有问题。

在着色器验证中严格执行这些限制。 具有一个分支条件的方案，看起来像它会一致地跨基元分支，即使条件表达式中的操作数是像素着色器计算的数量，但仍属于方案 A，不允许使用。 同样，某些着色器计算数量 x 上从动态流控件中请求渐变的方案，但在其中，x 似乎未在任何分支中修改，但仍属于方案 A，不允许。

这些限制包含对流控制的限制，以便实现可以简单地将分支指令的实现与谓词指令交换。

用户可以一起使用 A 和 B 方案中的说明。 例如，假设用户需要给定着色器计算纹理坐标的异性纹理样本;但是，仅满足某些每像素条件的像素需要纹理加载。 为了满足这些要求，用户可以计算所有像素的纹理坐标，在每像素不同的流控制之外，使用 dsx - ps 和 dsy - ps 指令立即计算渐变。 然后，在每像素内（如果 bool -psendif - ps/ 块），用户可以使用 texlddd - ps (纹理加载与用户提供的渐变) ，传递预计算渐变。 描述此使用模式的另一种方法是，虽然基元中的所有像素都必须计算纹理坐标并涉及渐变计算，但仅采样纹理所需的像素实际这样做。

无论这些规则如何，用户仍会承担负担，以确保在计算任何渐变 (或执行隐式计算渐变) 的纹理样本之前，必须事先为所有执行路径初始化包含源数据的寄存器。 临时寄存器的

初始化通常不会进行验证或强制实施。

相关主题

像素着色器说明

abs - ps

项目 • 2022/09/23

计算绝对值。

语法

abs dst， src

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
abs					x	x	x	x	x

此说明的工作方式如下。

```
dest.x = abs(src.x)
dest.y = abs(src.y)
dest.z = abs(src.z)
dest.w = abs(src.w)
```

说明信息

要求	值
最低操作系统	Windows 98

相关主题

像素着色器说明

add - ps

项目 • 2022/09/24

添加两个向量。

语法

```
add dst , src0 , src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
add	x	x	x	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x = src0.x + src1.x;
dest.y = src0.y + src1.y;
dest.z = src0.z + src1.z;
dest.w = src0.w + src1.w;
```

说明信息

要求	值
最低操作系统	Windows 98

相关主题

像素着色器说明

bem - ps

项目 • 2022/09/23

应用假颠簸环境地图转换。

语法

bem dst.rg ， src0 ， src1

其中

- dst.rg dst 是目标寄存器。 必须使用红色和绿色组件写入掩码。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
bem				x					

此指令执行以下计算。

```
(Given n == dest register #)
dest.r = src0.r + D3DTSS_BUMPENVMAT00(stage n) * src1.r
                + D3DTSS_BUMPENVMAT10(stage n) * src1.g

dest.g = src0.g + D3DTSS_BUMPENVMAT01(stage n) * src1.r
                + D3DTSS_BUMPENVMAT11(stage n) * src1.g
```

使用 bem 的规则：

1. bem 必须出现在着色器 (的第一阶段，即阶段标记) 之前。
2. bem 使用两个算术指令槽。
3. 每个着色器只允许一次使用此指令。
4. 目标写掩码必须是 .rg /.xy。

5. 无法共同发出此指令。
6. 除了目标写入掩码为 .rg 的限制外，源 src0、src1 和指令修饰符上的修饰符不受约束。

说明信息

要求	值
最低操作系统	Windows 98

相关主题

像素着色器说明

break - ps

项目 • 2022/09/22

在最近的 endloop - ps 或 endrep - ps 处突破当前循环。

语法

break

注解

以下版本支持此说明。

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
break						x	x	x	x

要求	**值**
最低操作系统	Windows 98

相关主题

像素着色器说明

break_comp - ps

项目 • 2022/09/23

根据每个组件比较，在最接近 的 endloop - ps 或 endrep - ps 处突破当前循环。

语法

```
break_comp src0、src1
```

其中：

- _comp是两个源寄存器之间的标量 (或单个) 比较。 该参数可以是下列值之一：

语法	比较
_gt	大于
_lt	小于
_ge	大于或等于
_le	小于或等于
_eq	等于
_ne	不等于

- src0 是源寄存器。 如果选择单个组件，则需要复制重排。
- src1 是源寄存器。 如果选择单个组件，则需要复制重排。

注解

以下版本支持此说明。

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
break_comp						x	x	x	x

比较为 true 时，它会中断当前循环，如下所示。

```
if (!(src0 comparison src1))
   jump to the corresponding endloop or endrep instruction;
```

相关主题

像素着色器说明

breakp - ps

项目 • 2022/09/23

有条件地在最接近的 endloop - ps 或 endrep - ps 处中断当前循环。 使用谓词寄存器的一个组件作为条件来确定是否执行指令。

语法

```
breakp [ ! ]P。{x|y|z|w}
```

其中：

- [!]是可选的反修饰符。
- p0 是 谓词寄存器。
- {x|y|z|w} 是 p0 上所需的复制重排。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
breakp						x	x	x	x

相关主题

像素着色器说明

call - ps

项目 • 2022/12/14

执行对使用提供的标签标记的指令的函数调用。

语法

```
call l#
```

其中：

- l# 是一个 标签 - ps 标记要调用的子例程的开头。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
call						x	x	x	x

此指令执行以下操作：

1. 下一个指令的推送地址到返回地址堆栈。
2. 从标签标记的指令继续执行。

相关主题

像素着色器说明

callnz bool - ps

项目 • 2022/09/22

如果不是零，则调用。 对标签索引标记的指令执行条件调用。

语法

```
callnz l# , [ ! ]B#
```

其中：

- l# 是一个 标签 - ps 标记要调用的子例程的开头。
- [!]是可选的反修饰符。
- b# 标识 常量布尔寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
callnz bool						x	x	x	x

此指令执行以下操作：

```
if (specified Boolean register is not zero)
{
    Push address of the next instruction to the return address stack
    Continue execution from the instruction marked by the label
}
```

相关主题

像素着色器说明

callnz pred - ps

项目 • 2022/09/22

使用谓词（如果不是零）进行调用。 对标签索引标记的指令执行条件调用。 谓词使用布尔值来确定是否不执行指令。

语法

callnz l# , [!]P。{x|y|z|w}

其中：

- 其中 l# 是一个 标签 - ps 标记要调用的子例程的开头。
- [!]是可选的反修饰符。
- p0 是谓词寄存器。 请参阅 谓词寄存器。
- {x|y|z|w} 是 p0 上所需的复制重排。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
callnz pred						x	x	x	x

此指令执行以下操作：

```
if (specified register component is not zero)
{
    Push address of the next instruction to the return address stack
    Continue execution from the instruction marked by the label
}
```

相关主题

像素着色器说明

cmp - ps

项目 • 2022/09/24

如果 src0 >= 0，请选择 src1。 否则，请选择 src2。 每个通道的比较完成。

语法

```
cmp dst、src0、src1、src2
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。
- src2 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
Cmp		x	x	x	x	x	x	x	x

版本 1_2 和 1_3 还有其他一些限制：

- 每个着色器最多可以使用三个 cmp 指令。
- 目标寄存器不能与任何源寄存器相同。

此示例执行四通道比较。

```
ps_1_4
def c0, -0.6, 0.6, 0, 0.6
def c1  0,0,0,0
def c2  1,1,1,1

mov r1, c1
mov r2, c2

cmp r0, c0, r1, r2   // r0 is assigned 1,0,0,0 based on the following:
```

```
// r0.x = c2.x because c0.x < 0
// r0.y = c1.y because c0.y >= 0
// r0.z = c1.z because c0.z >= 0
// r0.w = c1.w because c0.w >= 0
```

相关主题

像素着色器说明

cnd - ps

项目 • 2022/09/22

根据比较 src0 > 0.5，有条件地在 src1 和 src2 之间进行选择。

语法

cnd dst， src0， src1， src2

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。
- src2 是源寄存器。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
cnd	x	x	x	x					

对于版本 1_1 到 1_3，src0 必须为 r0.a。

```
// Version 1_1 to 1_3
if (r0.a > 0.5)
  dest = src1
else
  dest = src2
```

版本 1_4 分别比较每个通道。

```
for each component in src0
{
   if (src0.component > 0.5)
```

```
      dest.component = src1.component
   else
      dest.component = src2.component
}
```

这些示例显示了在版本 1_4 着色器中完成的四通道比较，以及版本 1_1 着色器中可能的单通道比较。

```
ps_1_4
def c0, -0.5, 0.5, 0, 0.6
def c1,  0,0,0,0
def c2,  1,1,1,1

cnd r1, c0, c1, c2   // r0 contains 1,1,1,0 because
// r1.x = c2.x because c0.x <= 0.5
// r1.y = c2.y because c0.y <= 0.5
// r1.z = c2.z because c0.z <= 0.5
// r1.w = c1.w because c0.w >  0.5
```

版本 1_1 到 1_3 仅与 r0 的复制 alpha 通道进行比较。

```
ps_1_1
def c0, -0.5, 0.5, 0, 0.6
def c1,  0,0,0,0
def c2,  1,1,1,1
mov r0, c0
cnd r1, r0.a, c1, c2   // r1 gets assigned 0,0,0,0 because
                       // r0.a > 0.5, therefore r1.xyzw = c1.xyzw
```

此示例比较两个值 A 和 B。该示例假定 A 加载到 v0 中，B 加载到 v1 中。 A 和 B 必须介于 -1 到 +1 的范围内，并且由于颜色寄存器 (vn) 定义为介于 0 和 1 之间，因此本示例中会满足此限制。

```
ps_1_1                // Version instruction
sub r0, v0, v1_bias   // r0 = A - (B - 0.5)
cnd r0, r0.a, c0, c1  // r0 = ( A > B ? c0 : c1 )

// r0 = c0 if A > B, otherwise, r0 = c1
```

相关主题

像素着色器说明

crs - ps

项目 • 2022/09/22

使用右侧规则计算跨产品。

语法

crs dst、src0、src1

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
Crs					x	x	x	x	x

此指令的工作方式如下。

```
dest.x = src0.y * src1.z - src0.z * src1.y;
dest.y = src0.z * src1.x - src0.x * src1.z;
dest.z = src0.x * src1.y - src0.y * src1.x;
```

对使用的一些限制：

- src0 不能与 dest 相同的寄存器。
- src1 不能与 dest 相同的寄存器。
- src0 不能有除默认重排 (.xyzw) 以外的任何重排。
- src1 不能有除默认重排 (.xyzw) 以外的任何重排。
- dest 必须有以下七个掩码之一：.x | .y | .z | .xy | .xz | .yz | .xyz。
- dest 必须是临时寄存器。
- dest 不得与 src0 或 src1 相同的寄存器

相关主题

像素着色器说明

dcl - (sm2， sm3 - ps asm)

项目 • 2022/09/23

声明像素着色器输入寄存器。

语法

dcl[_pp] dest[.mask]

其中：

- [_pp] 是可选的分部精度。 请参阅 分部精度。
- dest 是目标寄存器。 它必须是 输入颜色寄存器 (vn) ，或者 纹理坐标寄存器 (tn) 。
- [.mask] 是一个可选的写入掩码，用于控制可以写入的目标寄存器的哪些组件。 请参阅 目标注册写入掩码。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dcl					x	x	x	x	x

所有 dcl 指令必须出现在第一个可执行指令之前。

相关主题

像素着色器说明

dcl_samplerType (sm2， sm3 - ps asm)

项目 • 2022/09/22

声明像素着色器采样器。

语法

dcl_samplerType s#

其中：

- _samplerType定义采样器数据类型。 这决定了采样时每个纹理坐标所需的坐标数。 定义了以下纹理坐标维度。
  - _2d
  - _cube
  - _volume
- s# 标识采样器，其中 s 是采样器的缩写，#是采样器编号。 采样器是伪寄存器，因为你无法直接读取或写入它们。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dcl_samplerType					x	x	x	x	x

所有dcl_samplerType指令都必须显示在第一个可执行指令之前。

示例

```
dcl_cube t0.rgb;  // Define a 3D texture map.

add r0, r0, t0;   // Perturb texture coordinates.
texld r0, s0, r0; // Load r0 with a color sampled from stage0
                  //   at perturbed texture coordinates r0.
                  // This is a dependent texture read.
```

相关主题

像素着色器说明

dcl_semantics (sm3 - ps asm)

项目 • 2022/09/23

声明顶点着色器输出与像素着色器输入之间的关联。

语法

dcl_semantics [_centroid] dst[.write_mask]

其中：

- _semantics：标识预期的数据使用情况，并且可以是 D3DDECLUSAGE (中任何值，而不使用D3DDECLUSAGE_前缀) 。 此外，可以将整数索引追加到语义中，以区分使用类似语义的参数。
- [_Centid] 是可选的指令修饰符。 声明输入寄存器和纹理查找指令的dcl_usage指令支持它。 质心追加无空格。
- dst：目标寄存器。 请参阅 ps_3_0注册。
- write_mask：每次使用唯一写入掩码 (时，可以多次声明相同的输出寄存器，以便将不同的语义应用于各个组件) 。 但是，在声明中不能多次使用同一语义。 这意味着向量必须是四个分量或更少，并且不能跨越 (单个输出寄存器) 的四分量寄存器边界。 使用_psize语义时，它应具有完整的写入掩码，因为它被视为标量。 使用_position语义时，它应具有完整的写入掩码，因为必须写入所有四个组件。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dcl_usage								x	x

所有dcl_usage指令都必须显示在第一个可执行指令之前。

声明示例

```
ps_3_0
```

```
; Declaring inputs
dcl_normal      v0.xyz
dcl_blendweight v0.w ; Must be same reg# as normal, matching vshader packing
dcl_texcoord1   v1.y ; Mask can be any subset of mask from vshader semantic
dcl_texcoord0   v1.zw; Has to be same reg# as texcoord1, to match vshader

; Declaring samplers
dcl_2d s0
dcl_2d s1

def c0, 0, 0, 0, 0

mov r0.x, v1.y ; texcoord1
mov r0.y, c0
texld r0, r0, s0

texld r1, v1.zw, s1
...
(output regs in ps_3_0 are same as ps_2_0: oC0-oC3, oDepth)
```

相关主题

像素着色器说明

Antialias 示例

def - ps

项目 • 2022/09/23

定义像素着色器浮点常量。

语法

```
def dst， fVvalue1， fValue2， fValue3， fValue4
```

其中：

- dst 是目标寄存器。
- fValue1 到 fValue4 是浮点值。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
def	x	x	x	x	x	x	x	x	x

可通过两种方法在像素着色器中设置浮点常量。

1. 使用 def 直接在着色器内定义常量。
2. 使用 API 使用 SetPixelShaderConstantF 设置常量。

def 定义着色器常量，该常量在着色器设置为设备时加载其值。 这些常量称为即时常量。 即时常量优先于 API 方法设置的常量。

- 必须在着色器中的第一个算术或寻址指令之前显示。
- 可与 dcl - (sm2， sm3 - ps asm) 指令混合，(这些指令位于着色器) 的开头。
- dst 寄存器必须是 常量寄存器。
- 写入掩码必须完全 (默认) 。
- 如果在着色器中多次定义 常量寄存器 ，则最后一个寄存器会保留。

相关主题

像素着色器说明

defb - ps

项目 • 2022/09/24

定义一个布尔常量值，该常量值应在着色器设置为设备时加载。

语法

```
defb dest、booleanValue
```

其中

- dst 是目标寄存器。
- booleanValue 是单个布尔值，为 true 或 false。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
defb						x	x	x	x

defb 指令定义一个布尔着色器常量，该常量在着色器设置为设备时加载其值。 这些常量称为即时常量。 即时常量优先于 API 方法 SetPixelShaderConstantB 设置的常量。

有两种方法可以在着色器中设置布尔的constant。

1. 使用 defb 直接在着色器内定义常量。
2. 使用 API 方法设置常量。
    - 使用 SetPixelShaderConstantB 设置布尔常量。

相关主题

像素着色器说明

def - ps

defi - ps

defi - ps

项目 • 2022/09/22

定义一个整数常量值，在着色器设置为设备时，应加载该值。

语法

```
defi dst， integerValue
```

- dst 是目标寄存器。
- integerValue 是常量整数值。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
defi						x	x	x	x

defi 指令定义一个着色器常量，该常量在将着色器设置为设备时加载其值。 这些常量称为即时常量。 即时常量优先于 API 方法 SetPixelShaderConstantB 设置的常量。

可通过两种方法在着色器中设置常量。

1. 使用 defi 直接在着色器内定义常量。
2. 使用 API 方法设置常量。
    - 使用 SetPixelShaderConstantB 设置布尔常量。
    - 使用 SetPixelShaderConstantF 设置浮点常量。
    - 使用 SetPixelShaderConstantI 设置整数常量。

相关主题

像素着色器说明

dp2add - ps

项目 • 2022/09/22

执行二维点积和标量加法。

语法

```
dp2add dst, src0, src1, src2.{x|y|z|w}
```

其中：

- dst 是目标寄存器。
- src0、src1 和 src2 是三个源寄存器。
- {x|y|z|w} 是 src2 上所需的复制重排。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dp2add					x	x	x	x	x

添加的标量值由 src2 上的复制重排选择。

以下代码片段显示了执行的操作。

```
dest = src0.r * src1.r + src0.g * src1.g + src2.replicate_swizzle
// The scalar result is replicated to the write mask components
```

相关主题

像素着色器说明

dp3 - ps

项目 • 2022/09/22

计算源寄存器的三分量点积。

语法

```
dp3 dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dp3	x	x	x	x	x	x	x	x	x

以下代码片段显示了执行的操作：

```
dest.x = dest.y = dest.z = dest.w =
  (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z);
```

此指令在矢量管道中运行，始终写出到颜色通道。 对于版本 1_4，此指令仍使用矢量管道，但可能会写入任何通道。

目标寄存器 .rgb (.xyz) 写入掩码的说明可能与 dp3 共同发出，如下所示。

```
dp3 r0.rgb, t0, v0            // Copy scalar result to color components
+mov r2.a, t0                 // Copy alpha component from t0 in parallel
```

可以使用 源寄存器签名缩放 输入参数修饰符修改 dp3 指令，（_bx2) 应用于其输入参数（如果它们尚未扩展到已签名的动态范围）。 对于照明着色器，饱和指令修饰符 (_sat) 通常用于将负值固定为黑色，如以下示例所示。

```
dp3_sat r0, t0_bx2, v0_bx2    // t0 is a bump map, v0 contains the light
direction
```

相关主题

像素着色器说明

dp4 - ps

项目 • 2022/09/23

计算源寄存器的四分量点积。

语法

```
dp4 dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dp4		x	x	x	x	x	x	x	x

以下代码片段显示了执行的操作：

```
dest.x = dest.y = dest.z = dest.w =
    (src0.x * src1.x) + (src0.y * src1.y) +
    (src0.z * src1.z) + (src0.w * src1.w);
```

ps_1_2和ps_1_3的限制：

- 每个着色器最多可以使用四个 dp4 指令。
- 每个 dp4 指令计数为两个算术指令。

1_X版本的限制：

- 无法共同发出此指令，因为 dp4 在矢量管道和 alpha 管道中运行。

相关主题

像素着色器说明

dsx - ps

项目 • 2022/12/14

计算呈现目标的 x 方向变化率。

语法

```
dsx dst、src
```

其中：

- dst 是目标寄存器。
- src 是输入源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dsx						x	x	x	x

从源寄存器计算的更改速率是对相邻像素中同一寄存器的内容的近似值， (s) 使用当前像素在锁步骤中运行像素着色器。

dsx And dsy 指令通过查看源寄存器的当前内容来计算其结果， (每个组件) 在锁步骤中执行的本地区域中的各个像素。 用于计算渐变的确切公式因硬件而异，但应与硬件执行与纹理采样级别计算过程的一部分相同的操作一致。

相关主题

像素着色器说明

texldd - ps

dsy - ps

项目 • 2022/09/22

计算呈现目标的 y 方向变化率。

语法

```
dsy dst , src
```

其中：

- dst 是目标寄存器。
- src 是输入源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
dsy						x	x	x	x

从源寄存器计算的更改速率是相邻像素中同一寄存器的内容的近似值， (s) 使用当前像素在锁定步骤中运行像素着色器。

dsx And dsy 指令通过查看源寄存器的当前内容来计算其结果， (每个组件) 在锁步骤中执行的本地区域中的各个像素。 用于计算渐变的确切公式因硬件而异，但应与硬件执行与纹理采样级别计算过程的一部分相同的操作的方式一致。

相关主题

像素着色器说明

texldd - ps

else - ps

项目 • 2022/09/22

其他块"开始"菜单。

语法

else

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
else						x	x	x	x

如果相应 if 语句中的条件为 true，则运行 if 语句和匹配项所括的代码。

相关主题

像素着色器说明

if bool - ps

endif - ps

endif - ps

项目 • 2022/09/22

标记 if...else block.

语法

endif

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
endif						x	x	x	x

相关主题

像素着色器说明

if bool - ps

else - ps

endloop - ps

项目 • 2022/09/23

循环结束 - ps...endloop 块。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
endloop								x	x

endloop 必须遵循循环的最后一条指令 - ps 块。

示例

```
loop aL, i3
    add r1, r0, c2[ aL ]
endloop
```

相关主题

像素着色器说明

endrep - ps

项目 • 2022/09/22

结束 代表 - ps...endrep 块。

语法

endrep

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
endrep						x	x	x	x

示例

```
rep i2
    add r0, r0, c0
endrep
```

相关主题

像素着色器说明

exp - ps

项目 • 2022/09/23

提供全精度指数 2^x。

语法

```
exp dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排;也就是说，必须指定 .x、.y、.z、.w 重排 (组件之一或 .r、.g、.b、.a 等效项) 。 请参阅 源注册重排。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
exp					x	x	x	x	x

以下代码片段显示了执行的操作：

```
dest.x = dest.y = dest.z = dest.w = (float)pow(2,
src.replicateSwizzleComponent);
```

相关主题

像素着色器说明

frc - ps

项目 • 2022/09/22

返回每个输入组件的分数部分。

语法

```
frc dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
Frc					x	x	x	x	x

以下代码片段在概念上演示指令的操作方式。

```
dest.x = src.x - (float)floor(src.x);
dest.y = src.y - (float)floor(src.y);
dest.z = src.z - (float)floor(src.z);
dest.w = src.w - (float)floor(src.w);
```

floor 函数将传入的参数转换为小于 (或等于) 参数的最大整数。 这会转换为浮点，然后减去原始值。 生成的小数值范围为 0.0 到 1.0。

相关主题

像素着色器说明

if bool - ps

项目 • 2022/09/23

"开始"菜单 if 块。

语法

if bool

其中：

- bool 是布尔 (布尔) 寄存器号。 请参阅 常量布尔寄存器。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
if bool						x	x	x	x

如果 if 语句中的源布尔寄存器为 true，则执行 if 语句和匹配 endif - ps 或其他 - ps 括起来的代码。 否则，由其他代码括起来的代码 - ps...endif - ps 语句执行。 此指令使用一个指令槽。

一个 if 块可以嵌套。

如果块不能跨越循环块。

if 块可以后跟语句块和/或 else - ps 指令和/或 endif - ps 指令。

示例

此指令提供条件静态流控制。

```
defb b3, true

if b3
```

```
// Instructions to run if b3 is nonzero
else
// Instructions to run otherwise
endif
```

相关主题

像素着色器说明

else - ps

endif - ps

if_comp - ps

项目 • 2022/09/22

"开始"菜单一个 if bool - ps...else - ps...endif - ps 块，条件基于可以在着色器中计算的值。 此指令用于根据条件跳过代码块。

语法

if_comp src0、src1

其中：

- _comp是两个源寄存器之间的比较。 该参数可以是下列值之一：

语法	**比较**
_gt	大于
_lt	小于
_ge	大于或等于
_le	小于或等于
_eq	等于
_ne	不等于

- src0 是源寄存器。 需要复制重排才能选择组件。
- src1 是源寄存器。 需要复制重排才能选择组件。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
if_comp						x	x	x	x

此指令用于根据条件跳过代码块。

```
if (src0 comparison src1)
   jump to the corresponding else or endif instruction;
```

小心对浮点数使用相等和不等于比较模式。 由于在浮点计算期间发生舍入，因此可以对 epsilon 值进行比较， (小非零数) 以避免错误。

具体限制包括：

- if_comp...else - ps...endif - ps 块 (以及谓词， 如果 块) 可以嵌套到最多 24 层深。
- src0 和 src1 寄存器需要复制重排。
- if_comp块必须以其他块结尾 - vs 或 endif - 与 指令。
- if_comp...else - ps...endif - ps 块无法跨越循环块。 if_comp块必须完全位于循环块内部或外部。

示例

此指令提供条件动态流控制。

```
if_lt r3.x, r4.y
// Instructions to run if r3.x < r4.y

else
// Instructions to run otherwise

endif
```

相关主题

像素着色器说明

如果 pred - ps

项目 • 2022/09/22

"开始"菜单一个 if bool - ps...else - ps...endif - ps 块，其条件取自谓词寄存器的内容。

语法

```
if [ ! ]pred.replicateSwizzle
```

其中：

- [!]是可选的 NOT 修饰符。 这会修改谓词寄存器中的值。
- pred 是 谓词寄存器。
- replicateSwizzle 是 (复制或复制到所有四个组件) (重排) 的单个组件。 有效组件包括：[x、y、z、w] 或 [r， g， b， a]。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
if_pred						x	x	x	x

此指令用于基于谓词寄存器的通道跳过代码块。 每个if_pred块必须以 其他 - ps 或 endif - ps 指令结尾。

具体限制包括：

可以嵌套if_pred块。 这计入动态嵌套深度和 if_comp 块。

if_pred块不能跨越循环块;它要么完全在里面，要么包围它。

相关主题

像素着色器说明

label - ps

项目 • 2022/09/23

将下一个指令标记为具有标签索引。

语法

label l#

where # 标识标签号。

对于ps_2_x，标签号必须介于 0 和 15 之间。

对于ps_2_sw、ps_3_0和ps_3_sw，标签号必须介于 0 到 2047 之间。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
标签						x	x	x	x

此指令定义位于下一个着色器指令处的标签。 标签指令只能在 重新 指令 (之后直接存在，该指令定义上一个子例程或主程序) 的结尾。

相关主题

像素着色器说明

loop - ps

项目 • 2022/09/22

启动循环...endloop - ps 块。

语法

循环 aL， i#

其中：

- aL 是持有当前循环计数的Loop计数器寄存器。
- i# 是 常量整数寄存器。 请参阅备注。

备注

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
loop								x	x

- Loop计数器寄存器 (aL) 保存当前循环计数，可用于在循环块内将输入颜色寄存器 (v#) 的相对寻址。
- i#.x 指定迭代计数。 法律范围为 [0， 255]。 请注意，此指令不会递增或递减 i#.x 的值。
- i#.y 指定 Loop Counter Register (aL) 寄存器的初始值。 法律范围为 [0， 255]。 请注意，此指令不会递增或递减 i#.y 的值。
- i#.z 指定步骤/步幅大小。 法律范围为 [-128， 127]。
- 循环块不使用 i#.w，必须为 0。
- Loop块可以嵌套。 请参阅Flow控制限制。
- 嵌套时，Loop计数器寄存器的值 (aL) 引用直接封闭循环块。
- 允许Loop块完全位于 if* 块内或完全围绕它。 不允许流浪。

示例

```
```

```
loop aL, i3
    add r1, r0, v2[ aL ]
endloop
```

相关主题

像素着色器说明

log - ps

项目 • 2022/09/23

全精度 log一 (x) 。

语法

log dst、src

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排;也就是说，必须指定 .x、.y、.z、.w 重排组件 (或 .r、.g、.b、.a 等值) 之一。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
log					x	x	x	x	x

以下代码片段显示了执行的操作。

```
float v = abs(src);
if (v != 0)
{
    dest.x = dest.y = dest.z = dest.w =
        (float)(log(v)/log(2));
}
else
{
    dest.x = dest.y = dest.z = dest.w = -FLT_MAX;
}
```

此指令接受忽略其符号位的标量源。 结果将复制到所有四个通道。

相关主题

像素着色器说明

lrp - ps

项目 • 2022/09/22

按第一个源寄存器中指定的比例在第二和第三个源寄存器之间线性内插。

语法

```
lrp dst , src0 , src1 , src2
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。
- src2 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
Lrp	x	x	x	x	x	x	x	x	x

此指令基于以下公式执行线性内插。

```
dest = src0 * src1 + (1-src0) * src2
// which is the same as
dest = src2 + src0 * (src1 - src2)
```

相关主题

像素着色器说明

m3x2 - ps

项目 • 2022/09/22

将 3 分量向量乘以 3x2 矩阵。

语法

```
m3x2 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果是一个 2 分量的向量。
- src0 是表示 3 分量矢量的源寄存器。
- src1 是表示 3x2 矩阵的源寄存器，对应于连续 2 个寄存器中的第一个。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
m3x2					x	x	x	x	x

目标寄存器需要 xy 掩码。 对于 src0，但不允许 src1 使用反门和重排修饰符。

以下公式显示了指令的运行方式：

```
dest.x = (src0.x*src1.x) + (src0.y*src1.y) + (src0.z*src1.z);
dest.y = (src0.x*src2.x) + (src0.y*src2.y) + (src0.z*src2.z);
```

输入向量位于寄存器 src0 中。 输入 3x2 矩阵位于寄存器 src1 中，同一寄存器文件中的下一个更高的寄存器，如以下扩展所示。 生成 2D 结果，使目标寄存器的其他元素 (dest.z 和 dest.w) 不受影响。

此操作通常用于 2D 转换。 此指令作为点产品的配对实现，如下所示。

```
m3x2   r0.xy, r1, c0   which will be expanded to:

dp3   r0.x, r1, c0
dp3   r0.y, r1, c1
```

src1 寄存器的重排和否定修饰符无效。 dst 和 src0 寄存器或任何 src1+i 寄存器不能相同。

相关主题

像素着色器说明

m3x3 - ps

项目 • 2022/09/23

将 3 分量向量乘以 3x3 矩阵。

语法

```
m3x3 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为三分量向量。
- src0 是表示三分量的源寄存器。
- src1 是表示 3x3 矩阵的源寄存器，对应于连续 3 个寄存器中的第一个。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
m3x3					x	x	x	x	x

目标寄存器需要 xyz 掩码。 对于 src0，但不允许 src1 使用反门和重排修饰符。

以下代码片段显示了执行的操作。

```
dest.x = (src0.x * src1.x) + (src0.y * src1.y) + (src0.z * src1.z);
dest.y = (src0.x * src2.x) + (src0.y * src2.y) + (src0.z * src2.z);
dest.z = (src0.x * src3.x) + (src0.y * src3.y) + (src0.z * src3.z);
```

输入向量位于寄存器 src0 中。 输入 3x3 矩阵位于寄存器 src1 中，接下来的两个更高的寄存器，如下面的扩展所示。 生成 3D 结果，使目标寄存器的另一个元素 (dest.w) 不受影响。

此操作通常用于在照明计算期间转换普通向量。 此指令作为一组点产品实现，如下所示。

```
m3x3 r0.xyz, r1, c0  which will be expanded to:

dp3   r0.x, r1, c0
dp3   r0.y, r1, c1
dp3   r0.z, r1, c2
```

相关主题

像素着色器说明

m3x4 - ps

项目 • 2022/09/22

将 3 分量向量乘以 3x4 矩阵。

语法

```
m3x4 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为 4 分量向量。
- src0 是表示 3 分量矢量的源寄存器。
- src1 是表示 3x4 矩阵的源寄存器，对应于连续 4 个寄存器中的第一个。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
m3x4					x	x	x	x	x

目标寄存器需要 xyzw (默认) 掩码。 对于 src0，但不允许 src1 使用反门和重排修饰符。

以下代码片段显示了执行的操作。

```
dest.x = (src0.x*src1.x) + (src0.y*src1.y) + (src0.z*src1.z);
dest.y = (src0.x*src2.x) + (src0.y*src2.y) + (src0.z*src2.z);
dest.z = (src0.x*src3.x) + (src0.y*src3.y) + (src0.z*src3.z);
dest.w = (src0.x*src4.x) + (src0.y*src4.y) + (src0.z*src4.z);
```

输入向量位于寄存器 src0 中。 输入 3x4 矩阵位于寄存器 src1 中，接下来的两个更高的寄存器，如下面的扩展所示。

此操作通常用于通过具有投影效果的矩阵转换位置向量，但不应用任何翻译。 此指令作为点产品的配对实现，如下所示。

```
m3x4   r0.xyzw, r1, c0   will be expanded to:

dp3 r0.x, r1, c0
dp3 r0.y, r1, c1
dp3 r0.z, r1, c2
dp3 r0.w, r1, c3
```

相关主题

像素着色器说明

m4x3 - ps

项目 • 2022/09/23

将 4 分量向量乘以 4x3 矩阵。

语法

```
m4x3 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为三分量向量。
- src0 是表示 4 分量的源寄存器。
- src1 是表示 4x3 矩阵的源寄存器，对应于连续 3 个寄存器中的第一个。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
m4x3					x	x	x	x	x

目标寄存器需要 xyz 掩码。 src0 允许反门和重排修饰符，但不允许用于 src1。

以下代码片段显示了执行的操作。

```
dest.x = (src0.x*src1.x) + (src0.y*src1.y) + (src0.z*src1.z) +
(src0.w*src1.w);
dest.y = (src0.x*src2.x) + (src0.y*src2.y) + (src0.z*src2.z) +
(src0.w*src2.w);
dest.z = (src0.x*src3.x) + (src0.y*src3.y) + (src0.z*src3.z) +
(src0.w*src3.w);
```

输入向量位于寄存器 src0 中。 输入 4x3 矩阵位于寄存器 src1 中，接下来的两个更高的寄存器，如下面的扩展所示。 生成 3D 结果，使目标寄存器的另一个元素 (dest.w) 不受影响。

此操作通常用于通过没有投影效果的矩阵转换位置向量，例如在模型空间转换中发生。 此指令作为一组点产品实现，如下所示。

```
m4x3   r0.xyz, r1, c0    will be expanded to:

dp4   r0.x, r1, c0
dp4   r0.y, r1, c1
dp4   r0.z, r1, c2
```

src1 寄存器的重排和否定修饰符无效。 dst 和 src0 寄存器不能相同。

相关主题

像素着色器说明

m4x4 - ps

项目 • 2022/09/24

将 4 分量向量乘以 4x4 矩阵。

语法

```
m4x4 dst、src0、src1
```

其中

- dst 是目标寄存器。 结果为 4 分量向量。
- src0 是表示 4 分量的源寄存器。
- src1 是表示 4x4 矩阵的源寄存器，对应于连续 4 个寄存器中的第一个。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
m4x4					x	x	x	x	x

目标寄存器需要 xyzw (默认) 掩码。 src0 允许反门和重排修饰符，但不允许 src1 使用。

src0 寄存器的重排和否定修饰符无效。 dst 和 src0 寄存器不能相同。

以下代码片段显示了执行的操作。

```
dest.x = (src0.x*src1.x) + (src0.y*src1.y) + (src0.z*src1.z) +
(src0.w*src1.w);
dest.y = (src0.x*src2.x) + (src0.y*src2.y) + (src0.z*src2.z) +
(src0.w*src2.w);
dest.z = (src0.x*src3.x) + (src0.y*src3.y) + (src0.z*src3.z) +
(src0.w*src3.w);
dest.w = (src0.x*src4.x) + (src0.y*src4.y) + (src0.z*src4.z) +
(src0.w*src4.w);
```

输入向量位于寄存器 src0 中。 输入 4x4 矩阵位于寄存器 src1 中，接下来的三个更高的寄存器，如下面的扩展所示。

此操作通常用于通过投影矩阵转换位置。 此指令作为一组点产品实现，如下所示。

```
m4x4   r0.xyzw, r1, c0    will be expanded to:

dp4   r0.x, r1, c0
dp4   r0.y, r1, c1
dp4   r0.z, r1, c2
dp4   r0.w, r1, c3
```

相关主题

像素着色器说明

mad - ps

项目 • 2022/09/22

相乘并添加指令。 将目标寄存器设置为 (src0 * src1) + src2。

语法

```
mad dst , src0 , src1 , src2
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。
- src2 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
疯狂	x	x	x	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x = src0.x * src1.x + src2.x;
dest.y = src0.y * src1.y + src2.y;
dest.z = src0.z * src1.z + src2.z;
dest.w = src0.w * src1.w + src2.w;
```

相关主题

像素着色器说明

mov - ps

项目 • 2022/09/23

在寄存器之间移动数据。

语法

```
mov dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
mov	x	x	x	x	x	x	x	x	x

相关主题

像素着色器说明

mul - ps

项目 • 2022/09/22

将源乘以目标。

语法

```
mul dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
mul	x	x	x	x	x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x = src0.x * src1.x;
dest.y = src0.y * src1.y;
dest.z = src0.z * src1.z;
dest.w = src0.w * src1.w;
```

相关主题

像素着色器说明

max - ps

项目 • 2022/09/24

计算源的最大值。

语法

```
max dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
max					x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x=(src0.x >= src1.x) ? src0.x : src1.x;
dest.y=(src0.y >= src1.y) ? src0.y : src1.y;
dest.z=(src0.z >= src1.z) ? src0.z : src1.z;
dest.w=(src0.w >= src1.w) ? src0.w : src1.w;
```

相关主题

像素着色器说明

min - ps

项目 • 2022/09/23

计算源的最小值。

语法

min dst、src0、src1

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
分钟					x	x	x	x	x

以下代码片段显示了执行的操作。

```
dest.x=(src0.x < src1.x) ? src0.x : src1.x;
dest.y=(src0.y < src1.y) ? src0.y : src1.y;
dest.z=(src0.z < src1.z) ? src0.z : src1.z;
dest.w=(src0.w < src1.w) ? src0.w : src1.w;
```

相关主题

像素着色器说明

nrm - ps

项目 • 2022/09/22

规范化 3D 向量。

语法

```
nrm dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
nrm					x	x	x	x	x

此说明在概念上有效，如下所示。

squareRootOfTheSum = (src0.x*src0.x + src0.y*src0.y + src0.z*src0.z) $^{1/2}$;

```
dest.x = src0.x * (1 / squareRootOfTheSum);
dest.y = src0.y * (1 / squareRootOfTheSum);
dest.z = src0.z * (1 / squareRootOfTheSum);
dest.w = src0.w * (1 / squareRootOfTheSum);
```

dest 和 src 寄存器不能相同。 dest 寄存器必须是临时寄存器。

此指令根据以下公式执行线性内插。

```
float f = src0.x*src0.x + src0.y*src0.y + src0.z*src0.z;
if (f != 0)
```

```
    f = (float)(1/sqrt(f));
else
    f = FLT_MAX;

dest.x = src0.x*f;
dest.y = src0.y*f;
dest.z = src0.z*f;
dest.w = src0.w*f;
```

相关主题

像素着色器说明

nop - ps

项目 • 2022/09/24

不执行任何操作。

语法

```
nop
```

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
nop	x	x	x	x	x	x	x	x	x

此指令执行无操作或无操作。 调用它的语法如下所示：

```
nop
```

相关主题

像素着色器说明

阶段 - ps

项目 • 2022/09/22

阶段指令标记阶段 1 和阶段 2 之间的转换。 如果没有阶段指令，则整个着色器将运行，就像它是第 2 阶段着色器一样。

此说明仅适用于版本 1_4。

语法

```
phase
```

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
phase				x					

阶段指令之前发生的着色器指令是阶段 1 指令。 所有其他指令都是阶段 2 指令。 通过为指令提供两个阶段，每个着色器的最大指令数将增加。

阶段转换的不幸副作用是 临时寄存器 的 alpha 组件不会在整个转换期间保留。 换句话说，在阶段指令之后必须重新初始化 alpha 组件。

示例

此示例演示如何将指令分组为着色器中的阶段 1 或阶段 2 指令。

阶段指令也称为阶段标记，因为它标记阶段 1 和 2 指令之间的转换。 在版本 1_4 像素着色器中，如果阶段标记不存在，则着色器将像在第 2 阶段中一样运行。 这一点很重要，因为阶段 1 和 2 指令和注册可用性之间存在差异。 在整个参考节中都记录了差异。

```
ps_1_4
  // Add phase 1 instructions here
```

```
phase
  // Add phase 2 instructions here
```

相关主题

像素着色器说明

pow - ps

项目 • 2022/09/23

全精度 abs (src0) ^{src1}。

语法

pow dst、src0、src1

其中

- dst 是目标寄存器。
- src0 是输入源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。
- src1 是输入源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

备注

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
pow					x	x	x	x	x

此说明的工作原理如下：

dest.x = dest.y = dest.z = dest.w = [abs (src0)]^{src1};

这是标量指令，因此源寄存器应具有复制重排来指示使用哪个通道。

输入电源 (src1) 必须是标量。

标量结果将复制到所有四个输出通道。

此指令可以扩展为 exp (src1 * log (src0)) 。

dst 寄存器应为临时寄存器，不应与 src1 相同。

相关主题

像素着色器说明

ps

项目 • 2022/09/24

此指令指定着色器版本号，并适用于所有着色器版本。

语法

```
ps_mainVer_subVer
```

输入参数

输入参数包含具有单个子版本号的单个主版本号。 允许的组合列在下表中。

主版本	子版本
1	1, 2, 3, 4
2	0、x (扩展) 、 (软件)
3	0 ， sw (软件)

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
ps	x	x	x	x	x	x	x	x	x

此指令必须是像素着色器中的第一个非注释指令。

在版本号) 中没有_sw，硬件加速版本的软件 (版本可以处理具有硬件提升的顶点或使用软件顶点处理。 仅在软件中处理顶点) 版本号中具有_sw的软件版本 (版本。

示例

此部分示例声明版本 1_1 像素着色器。

```
ps_1_1
```

本部分示例声明版本 1_4 像素着色器。

```
ps_1_4
```

相关主题

像素着色器说明

rcp - ps

项目 • 2022/09/24

计算源标量的对等。

语法

```
rcp dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 (或 .r、.g、.b、.a 等值) 的 .x、.y、.z、.w 重排组件之一。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
rcp					x	x	x	x	x

如果输入正好为 1.0，则输出必须正好为 1.0。 0.0 的源产生无穷大。

标量结果将复制到目标写入掩码中的所有通道。

精度应至少为 1.0/ (2一一) 在 (1.0、2.0) 范围内出现绝对误差，因为常见的实现将分离 mantissa 和指数。

相关主题

像素着色器说明

rep - ps

项目 • 2022/09/22

"开始"菜单代表...endrep - ps 块。

语法

rep i#

其中 i# 是一个整数寄存器，用于指定 .x 组件中的重复计数。 请参阅 常量整数寄存器。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
代表						x	x	x	x

- i#.x 指定迭代计数。 法律范围为 [0， 255]。 请注意，此指令不会递增或递减 i#.x 的值。
- 重复块不使用 i#.yzw。
- 重复块可以嵌套。 请参阅Flow控制限制。
- 允许重复块完全位于 if* 块内或完全围绕它。 不允许流浪。
- 对不同或嵌套的代表指令使用相同的 i# 是正常的 - 每个循环将基于指定的计数循环访问。

示例

```
rep i2
    add r0, r0, c0
endrep
```

相关主题

像素着色器说明

ret - ps

项目 • 2022/09/22

从返回地址堆栈获取指令的地址，并继续从该地址堆栈执行。 对于主函数，此指令将停止着色器执行。

语法

Ret

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
Ret						x	x	x	x

此指令从返回地址堆栈获取指令的地址，并继续从该地址堆栈执行。 对于主函数，此指令将停止着色器执行。

重试指令使用一个顶点着色器指令槽。

如果着色器不包含子例程，则使用主程序末尾的 ret 是可选的。

主程序或任何子例程不允许使用多个返回语句，第一个返回语句被视为主程序或子例程的结尾。

相关主题

像素着色器说明

rsq - ps

项目 • 2022/09/24

计算源标量 (正) 的倒数平方根。

语法

```
rsq dst，src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。 源寄存器需要显式使用复制重排，即必须指定 .x、.y、.z、.w 重排 (组件之一或 .r、.g、.b、.a 等效项) 。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
rsq					x	x	x	x	x

在处理之前将采用绝对值。

精度应至少为 1.0/ (2一一) 介于 1.0、4.0) (范围内的绝对误差，因为通用实现将分离 mantissa 和指数。

如果输入正好为 1.0，则输出必须正好为 1.0。 0.0 的源生成无穷大。

相关主题

像素着色器说明

setp_comp - ps

项目 • 2022/09/24

设置谓词寄存器。

语法

setp_comp dst、src0、src1

其中：

- _comp是两个源寄存器之间的每通道比较。 该参数可以是下列值之一：

语法	**比较**
_gt	大于
_lt	小于
_ge	大于或等于
_le	小于或等于
_eq	等于
_ne	不等于

- dst 是 谓词寄存器 ，p0。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
setp_comp						x	x	x	x

此指令按以下方式操作：

```
per channel in destination write mask
{
  dst.channel = src0.channel cmp src1.channel
}
```

对于可以根据目标写入掩码写入的每个通道，请在解析源修饰符重) 排后保存 src0 和 src1 (的相应通道之间的比较运算的布尔结果。

源寄存器允许指定任意组件重排。

目标寄存器允许任意写入掩码。

第 dst 寄存器必须是谓词寄存器。

应用谓词寄存器

使用 setp_comp 初始化谓词寄存器后，它可用于控制每个组件的指令。 下面是语法：

```
([!]p0[.swizzle]) instruction dest, src0, ...
```

其中：

- [!]是可选的布尔 NOT
- p0 是谓词寄存器
- [.swizzle] 是一个可选的重排，用于在使用该谓词寄存器屏蔽指令之前应用于谓词寄存器的内容。 可用重排为：如果未指定任何），或者任何复制重排：.x/.r、.y/.g、.z/.b 或 .a/.w，则为 .xyzw (默认值。
- 指令是任何写符号或纹理指令。 这不能是静态或动态流控制指令。
- dest，src0，...是指令所需的寄存器

假设谓词寄存器已设置为 (true、true、false、false) 组件值，则可以将其应用于以下指令：

```
(p0) add r1, r2, r3
```

若要执行 2 个组件添加，则为 。

```
r1.x = r2.x + r3.x
r1.y = r2.y + r3.y
```

r1 的 z 和 w 组件不会写入，因为谓词寄存器在 components z 和 w 中包含 false。

将谓词寄存器应用于算术或纹理指令会将其指令槽计数增加 1。

如果 pred - ps，callnz pred - ps 和 breakp - ps 指令，也可以应用于谓词寄存器。 使用谓词寄存器时，这些流控制指令在指令槽计数中没有任何增加。

相关主题

像素着色器说明

sincos - ps

项目 • 2022/09/22

以弧度计算正弦和余弦。

语法

ps_2_0和ps_2_x

```
sincos dst.{x|y|xy}，src0。{x|y|z|w}， src1， src2
```

其中：

- dst 是目标寄存器，必须是 临时注册 (r#)。 目标寄存器必须具有以下三个掩码之一：.x | .y | .xy。
- src0 是一个源寄存器，它提供输入角度，必须位于 [-pi， +pi] 内。 {x|y|z|w} 是所需的复制重排。
- src1 和 src2 是源寄存器，必须是两个不同的 常量浮点寄存器 (c#)。 src1 和 src2 的值必须是宏 D3DSINCOSCONST1 和 D3DSINCOSCONST2 的值。

ps_3_0

```
sincos dst.{x|y|xy}，src0。{x|y|z|w}
```

其中：

- dst 是目标寄存器，必须是 临时注册 (r#)。 目标寄存器必须具有以下三个掩码之一：.x | .y | .xy。
- src0 是一个源寄存器，它提供输入角度，必须位于 [-pi， +pi] 内。 {x|y|z|w} 是所需的复制重排。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
sincos					x	x	x	x	x

ps_2_0和ps_2_x

对于ps_2_0和ps_2_x，sincos 可用于谓词，但对 谓词寄存器 (p0) 的重排有一个限制：仅允许复制重排 (.x | .y | .z | .w) 。

对于ps_2_0和ps_2_x，指令按如下所示操作， (V = src0 中的标量值，其中包含复制重排) ：

- 如果写入掩码为 .x：

```
dest.x = cos(V)
dest.y is undefined when the instruction completes
dest.z is undefined when the instruction completes
dest.w is not touched by the instruction
```

- 如果写入掩码为 .y：

```
dest.x is undefined when the instruction completes
dest.y = sin(V)
dest.z is undefined when the instruction completes
dest.w is not touched by the instruction
```

- 如果写入掩码为 .xy：

```
dest.x = cos(V)
dest.y = sin(V)
dest.z is undefined when the instruction completes
dest.w is not touched by the instruction
```

ps_3_0

对于ps_3_0，sincos 可与谓词一起使用，没有任何限制。 请参阅 谓词寄存器。

对于ps_3_0，指令按如下所示操作， (V = 具有复制重排) 的 src0 的标量值：

- 如果写入掩码为 .x：

```
dest.x = cos(V)
dest.y is not touched by the instruction
dest.z is not touched by the instruction
dest.w is not touched by the instruction
```

- 如果写入掩码为 .y：

```
dest.x is not touched by the instruction
dest.y = sin(V)
dest.z is not touched by the instruction
dest.w is not touched by the instruction
```

- 如果写入掩码为 .xy：

```
dest.x = cos(V)
dest.y = sin(V)
dest.z is not touched by the instruction
dest.w is not touched by the instruction
```

应用程序可以使用以下着色器伪代码将任意角度 (以弧度) 映射到范围 [-pi， +pi] ：

```
def c0, pi, 0.5, 2*pi, 1/(2*pi)
mad r0.x, input_angle, c0.w, c0.y
frc r0.x, r0.x
mad r0.x, r0.x, c0.z, -c0.x
```

相关主题

像素着色器说明

sub - ps

项目 • 2022/09/22

减去源。

语法

```
sub dst、src0、src1
```

其中

- dst 是目标寄存器。
- src0 是源寄存器。
- src1 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
sub	x	x	x	x	x	x	x	x	x

此指令执行此公式中显示的减法。

```
dest = src0 - src1
```

相关主题

像素着色器说明

tex - ps

项目 • 2022/09/23

使用从纹理采样的 RGBA (RGBA) 加载目标寄存器。 纹理必须绑定到使用 SetTexture (n) 的特定纹理阶段。 纹理采样由 SetSamplerState 控制。

语法

```
tex dst
```

其中

- dst 是目标寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
tex	x	x	x						

目标寄存器编号指定纹理阶段号。

纹理采样使用纹理坐标查找或采样指定 (u、v、w、q) 坐标的颜色值，同时考虑纹理阶段状态属性。

纹理坐标数据从顶点纹理坐标数据内插，并与特定纹理阶段相关联。 默认关联是纹理阶段号与纹理坐标声明顺序之间的一对一映射。 这意味着，在顶点格式中定义的第一组纹理坐标默认与纹理阶段 0 相关联。

纹理坐标可以使用两种技术与任何阶段相关联。 使用固定函数顶点着色器或固定函数管道时，纹理阶段状态标志TSS_TEXCOORDINDEX可在 SetTextureStageState 中使用，将坐标关联到阶段。 否则，使用可编程顶点着色器着色器时，纹理坐标由顶点着色器 oTn 寄存器输出。

相关主题

像素着色器说明

texbem - ps

项目 • 2022/09/22

应用假颠簸环境地图转换。 这可以通过修改目标寄存器的纹理地址数据，使用地址干扰数据 (du、dv) 和 2D 凸起环境矩阵来实现。

语法

```
texbem dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texbem	x	x	x						

src 寄存器中的红色和绿色数据被解释为 (du，dv) 的扰动数据。

此指令使用 2D 凹凸环境映射矩阵转换源寄存器中的红色和绿色组件。 结果将添加到对应于目标寄存器号的纹理坐标集，并用于采样当前纹理阶段。

此操作始终将 du 和 dv 解释为已签名数量。 对于版本 1_0 和 1_1，在输入参数上不允许源 寄存器签名缩放 输入修饰符 (_bx2) 。

当输入纹理包含带符号格式数据时，此指令将生成定义的结果。 仅当前两个通道包含有符号数据时，混合格式数据才有效。 有关图面格式的详细信息，请参阅 D3DFORMAT。

这可用于基于地址干扰的各种技术，包括假每像素环境映射和漫射照明 (颠簸映射) 。

使用此指令时，纹理寄存器必须遵循以下顺序。

```
```

```
// The texture assigned to stage t(n) contains the (du,dv) data
// The texture assigned to stage t(m) is sampled
tex     t(n)
texbem  t(m),  t(n)      where m > n
```

在指令中完成的计算如下所示。

```
// 1. New values for texture addresses (u',v') are calculated
// 2. Sample the texture using (u',v')
```

u' = textureCoordinates (stage m) $_{u}$ + D3DTSS_BUMPENVMAT00 (stage m) *t (n) $_{R}$ + D3DTSS_BUMPENVMAT10 (stage m) *t (n) $_{G}$ v' = TextureCoordinates (stage m) $_{v}$ + D3DTSS_BUMPENVMAT01 (stage m) *t (n) $_{R}$ + D3DTSS_BUMPENVMAT11 (stage m) *t (n) $_{G}$ t (m) $_{RGBA}$ = TextureSample (stage m)

使用 (u'，v') 作为坐标

注册已由 texbem - ps 或 texbeml 读取的数据 - ps 指令以后无法读取，但其他 texbem - ps 或 texbeml - ps 除外。

```
// This example demonstrates the validation error caused by t0 being reread:
ps_1_1
tex t0
texbem t1, t0
add r0, t1, t0

(Instruction Error) (Statement 4) Register data that has been read by
texbem or texbeml instruction cannot be read by other instructions
```

示例

下面是一个示例着色器，其中包含标识的纹理映射和标识的纹理阶段。

```
ps_1_1
tex t0              ; Define t0 to get a 2-tuple DuDv
texbem t1, t0       ; Compute (u',v')
                    ; Sample t1 using (u',v')
mov r0, t1          ; Output result
```

texbem 在以下纹理阶段中需要以下纹理。

- 阶段 0 分配了带有 (du、dv) 扰动数据的颠簸地图。
- 阶段 1 使用具有颜色数据的纹理地图。
- 此指令在采样的纹理阶段设置矩阵数据。
- 这不同于固定函数管道的功能，其中扰动数据和矩阵占用相同的纹理阶段。

相关主题

像素着色器说明

texbeml - ps

项目 • 2022/09/24

应用具有亮度更正的假颠簸环境地图转换。 这是通过修改目标寄存器的纹理地址数据、使用地址干扰数据 (du、dv) 、2D 凸起环境矩阵和亮度来实现的。

语法

```
texbeml dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texbeml	x	x	x						

src 寄存器中的红色和绿色数据被解释为 du，dv) 的干扰数据 (。 src 寄存器中的蓝色数据被解释为亮度数据。

此指令使用 2D 凸起环境映射矩阵转换源寄存器中的红色和绿色组件。 结果将添加到对应于目标寄存器号的纹理坐标集中。 使用亮度值和偏差纹理阶段值应用亮度更正。 结果用于采样当前纹理阶段。

这可用于基于地址干扰的各种技术，例如假每像素环境映射。

此操作始终将 du 和 dv 解释为已签名数量。 对于版本 1_0 和 1_1，不允许输入参数 (_bx2) 源寄存器签名缩放 输入修饰符。

当输入纹理包含混合格式数据时，此指令将生成定义的结果。 有关图面格式的详细信息，请参阅 D3DFORMAT。

```
// When using this instruction, texture registers must follow
//   the following sequence:
// The texture assigned to stage tn contains the (du,dv) data
// The texture assigned to stage t(m) is sampled
tex     t(n)
texbeml t(m),  t(n)      where m > n
```

此示例演示指令中完成的计算。

```
// 1. New values for texture addresses (u',v') are calculated
// 2. Sample the texture using (u',v')
// 3. Luminance correction is applied
```

u' = textureCoordinates (阶段 m) $_{u}$ +

D3DTSS_BUMPENVMAT00 (阶段 m) *t (n) $_{R}$ +

D3DTSS_BUMPENVMAT10 (阶段 m) *t (n) $_{G}$

v' = TextureCoordinates (阶段 m) $_{v}$ +

D3DTSS_BUMPENVMAT01 (阶段 m) *t (n) $_{R}$ +

D3DTSS_BUMPENVMAT11 (阶段 m) *t (n) $_{G}$

t (m) $_{RGBA}$ = textureSample (阶段 m) 使用 (u'，v') 作为坐标

t (m) $_{RGBA}$ = t (m) $_{RGBA}$ *

[(t (n) $_{B}$ * D3DTSS_BUMPENVLSCALE (stage m)) +

D3DTSS_BUMPENVLOFFSET (阶段 m)]

注册已由 texbem 或 texbeml 指令读取的数据以后无法读取，但其他 texbem 或 texbeml 除外。

```
// This example demonstrates the validation error caused by
//   t0 being reread
ps_1_1
tex t0
texbem t1, t0
add r0, t1, t0

(Instruction Error) (Statement 4) Register data that has been read by
texbem or texbeml instruction cannot be read by other instructions
```

示例

下面是一个示例着色器，其中包含标识的纹理映射和标识的纹理阶段。

```
ps_1_1
tex t0              ; Define t0 to get a 2-tuple DuDv
texbeml t1, t0      ; Compute (u',v')
                    ; Apply luminance correction
                    ; Sample t1 using (u',v')
mov r0, t1          ; Output result
```

此示例需要以下纹理阶段中的以下纹理。

- 阶段 0 分配有 (du、dv) 扰动数据的凸起地图。
- 阶段 1 分配有颜色数据的纹理地图。
- texbeml 在采样的纹理阶段设置矩阵数据。 这不同于固定函数管道的功能，其中干扰数据和矩阵占用相同的纹理阶段。

相关主题

像素着色器说明

texcoord - ps

项目 • 2022/09/22

将纹理坐标数据解释为 (UVW1) 颜色数据， (RGBA) 。

语法

```
texcoord dst
```

其中

- dst 是目标寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texcoord	x	x	x						

本指令将 (UVW1) 的纹理坐标集解释为颜色数据， (RGBA) 。 如果纹理坐标集包含的组件少于 3 个，则缺少的组件设置为 0。 第四个组件始终设置为 1。 所有值都固定在 0 和 1 之间。

texcoord 的优点是，它提供了一种方法，以高分辨率直接插入到像素着色器中插入顶点数据。 但是，将数据写入目标寄存器时，会丢失一些精度，具体取决于硬件用于寄存器的位数。

此指令不对纹理进行采样。 只有在此纹理阶段设置的纹理坐标是相关的。

任何纹理数据（（如位置、普通和光源方向））都可以由顶点着色器映射到纹理坐标中。 这可以通过使用 SetTexture 将纹理与纹理寄存器相关联，并通过指定如何使用 SetTextureStageState 完成纹理采样来完成。 如果使用固定函数管道，请确保提供 TSS_TEXCOORDINDEX标志。

此说明如下所示：

```
```

```
texcoord tn
```

纹理坐标寄存器 (tn) 包含 RGBA) 四个颜色 (值。 数据也可以视为 (xyzw) 的向量数据。 texcoord 将从纹理坐标集 x 检索其中三个值 (xyz) ，第四个组件 (w) 设置为 1。 纹理地址从纹理坐标集 n 复制。 结果固定在 0 和 1 之间。

本例只用于演示。 着色器随附的 C 代码尚未针对性能进行优化。

下面是使用 texcoord 的示例着色器。

```
ps_1_1        ; version instruction
texcoord t0   ; declare t0 hold texture coordinates,
              ; which represent rgba values in this example
mov r0, t0    ; move the color in t0 to output register r0
```

下图显示了像素着色器的呈现输出。 (u、v、w、w、1) 坐标值映射到 (rgb) 通道。 alpha 通道设置为 1。 在插图的角落，坐标 (0，0，0，1) 被解释为黑色; (1，0，0，1) 为红色; (0，1，0，1) 为绿色;和 (1，1，0，1) 包含绿色和红色，产生黄色。

使用此着色器需要其他代码，下面显示了一个示例方案。

```
// This code creates the shader from a file. The contents of
// the shader file can also be supplied as a text string.
LPD3DXBUFFER        pCode;

// Assemble the vertex shader from the file
D3DXAssembleShaderFromFile(strPShaderPath, 0, NULL, &pCode, NULL);
m_pd3dDevice->CreatePixelShader((DWORD*)pCode->GetBufferPointer(),
                  &m_hPixelShader);
pCode->Release();

// This code defines the object vertex data
struct CUSTOMVERTEX
{
    FLOAT x, y, z;
    FLOAT tu1, tv1;
};
```

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_TEX1|TEXCOORD2(0))

static CUSTOMVERTEX g_Vertices[]=
{
    //  x      y     z     u1    v1
    { -1.0f, -1.0f, 0.0f, 0.0f, 0.0f, },
    { +1.0f, -1.0f, 0.0f, 1.0f, 0.0f, },
    { +1.0f, +1.0f, 0.0f, 1.0f, 1.0f, },
    { -1.0f, +1.0f, 0.0f, 0.0f, 1.0f, },
};
```

相关主题

像素着色器说明

texcrd - ps

项目 • 2022/09/24

将源纹理坐标迭代器中的纹理坐标数据复制为目标寄存器中的颜色数据。

语法

```
texcrd dst， src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texcrd				x					

此指令将坐标数据解释为颜色数据 (RGBA) 。

此指令不对纹理进行采样。 仅此纹理阶段上设置的纹理坐标是相关的。

使用 texcrd 时，请记住以下详细信息，了解如何将数据从源寄存器复制到目标寄存器。 源纹理坐标寄存器 (t#) 保存范围 [-D3DCAPS9] 中的数据。MaxTextureRepeat， D3DCAPS9。MaxTextureRepeat]，而目标寄存器 (r#) 只能将数据保存在 (可能更小的) 范围 [-D3DCAPS9 中。PixelShader1xMaxValue、D3DCAPS9。PixelShader1xMaxValue]。 请注意，对于像素着色器版本 1_4、D3DCAPS9。PixelShader1xMaxValue 必须至少为 8。 在固定目标寄存器范围外源数据的过程中，texcrd 指令在不同硬件上的行为可能会有所不同。 市场上的第一个像素着色器版本 1_4 硬件将针对超出范围的值执行特殊固定。 此固定器旨在生成一个可以容纳到目标寄存器中的数字，但也为了保留超出范围的数据的纹理寻址行为， (如果数据随后用于纹理采样，请参阅 D3DTEXTUREADDRESS) 。 但是，来自不同制造商的新硬件可能不会表现出此行为，可能只是切碎数据以适应目标注册范围。 因此，使用像素着色器版本 1_4 texcrd 时，最安全的方法是仅将纹理坐标数据提供给已位于 [-8，8] 范围内的像素着色器中，以便你不依赖硬件固定的方式。

与texcoord_不同，texcrd 不会将值固定在 0 和 1 之间。

使用 texcrd 的规则：

1. 必须在 texcrd 或 texld 指令中对单个 t (n) 寄存器的每个读取应用相同的 .xyz 或 .xyw 修饰符。
2. texcrd 的第四个通道结果在所有情况下都是未设置/未定义的。
3. 第三个通道针对xyw_dw事例未设置/未定义。

示例

下面显示了 texcrd 的完整允许语法集，考虑到所有有效的源修饰符/选择器和目标写入掩码组合。 请注意，可以互换使用 .rgba 和 .xyzw 表示法。

将纹理坐标迭代器寄存器的前三个通道（ t (n) ）复制到 r (m) 。 r (m) 的第四个通道未初始化。

```
texcrd  r(m).rgb, t(n).xyz

// Produces the same result as the previous instruction
texcrd  r(m).rgb, t(n)
```

将 t (n) 的第一个、第二和第四个组件放入 r (m) 的前三个通道。 r (m) 的第四个通道未初始化。

```
texcrd  r(m).rgb, t(n).xyw
```

下面是使用_dw修饰符的投影除法示例。

此示例将 x/w 和 y/w 从 t (n) 复制到 r (m) 的前两个通道中。 r (m) 的第三和第四个通道未初始化。 以前写入 r (m) 的第三个通道的任何数据都将丢失。 r (m) 的第四个通道中的数据由于阶段标记而丢失。 对于版本 1_4，将忽略D3DTTFF_PROJECTED标志。

```
texcrd  r(m).rg,  t(n)_dw.xyw
```

相关主题

像素着色器说明

texdepth - ps

项目 • 2022/09/22

计算要用于此像素的深度缓冲区比较测试中的深度值。

语法

texdepth dst

其中

- dst 是目标寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texdepth				x					

此指令在此像素的深度缓冲区比较测试中使用 r5.r/ r5.g。 将忽略蓝色和 alpha 通道中的数据。 如果 r5.g = 0，则 r5.r/ r5.g = 1.0 的结果。

临时寄存器 r5 是唯一可以使用此指令的寄存器。

执行此指令后，临时寄存器 r5 在着色器中不可用。

进行多重采样时，使用此指令可以消除更高分辨率深度缓冲区的大部分好处。 由于像素着色器每像素运行一次，因此 texm3x2depth - ps 或 texdepth 的单深度值将用于每个子像素深度比较测试。

示例

下面是使用 texdepth 的示例。

```
ps_1_4
texld  r0, t0        // Sample texture from texture stage 0 (dest
```

```
                    //   register number) into r0
                    // Use texture coordinate data from t0
texcrd r1.rgb, t1   // Load a second set of texture coordinate data into r1
add    r5.rg, r0, r1 // Add the two sets of texture coordinate data
phase               // Phase marker, required when using texdepth
instruction
texdepth  r5        // Calculate pixel depth as r5.r / r5.g
                    // Do other color calculations with shader output r0
```

相关主题

像素着色器说明

texdp3 - ps

项目 • 2022/09/22

在纹理寄存器号中的数据与对应于目标寄存器号的纹理坐标集之间执行三分量点积。

语法

```
texdp3 dst，src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texdp3		x	x						

纹理寄存器必须使用以下序列。

```
tex    t(n)        // Define tn as a standard 3-vector (tn must be
                   //   defined in some way before texdp3 uses it)
texdp3 t(m), t(n)  // where m > n
                   // Perform a three-component dot product between tn and
                   //   the texture coordinate set m. The scalar result is
                   //   replicated to all components of t(m)
```

下面更详细地介绍了点积的实现方式。

texdp3 指令执行三分量点产品，并将其复制到所有四个颜色通道。

$t(m)_{RGBA} = TextureCoordinates(stage\ m)_{UVW} * t(n)_{RGB}$

相关主题

像素着色器说明

texdp3tex - ps

项目 • 2022/09/23

执行三分量点积，并使用结果执行 1D 纹理查找。

语法

texdp3tex dst， src

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texdp3tex		x	x						

纹理寄存器必须使用以下序列。

```
tex       t(n)        // Define tn as a standard 3-vector (tn must be
                      // defined in some way before texdp3tex uses it)
texdp3tex t(m), t(n)  // where m > n
                      // Perform a three-component dot product between t(n)
and
                      // the texture coordinate set m. Use the scalar result
to
                      // do a 1D texture lookup at texturestage m and place
                      // the result in t(m)
```

下面是有关如何完成点产品和纹理查找的更多详细信息。

texdp3tex 指令执行三分量点积。

u' = textureCoordinates (stage m) $_{UVW}$ * t (n) $_{RGB}$

结果用于通过执行 1D 查找在纹理阶段 m 采样纹理。

$t(m)_{RGBA} = textureSample(stage\ m)_{RGBA}$ ，使用 (u'，0，0) 作为坐标

相关主题

像素着色器说明

texkill - ps

项目 • 2022/09/23

如果前三个组件中的任何一个 (UVW) 纹理坐标小于零，则取消当前像素的呈现。

语法

texkill dst

其中

- dst 是目标寄存器

注解

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
texkill	x	x	x	x	x	x	x	x	x

此指令对应于 HLSL 的 **剪辑** 函数。

texkill 不采样任何纹理。 它针对目标寄存器编号给出的纹理坐标的前三个组件进行操作。 对于ps_1_4，texkill 对目标寄存器的前三个组件中的数据进行操作。

可以使用此指令在光栅器中实现任意剪辑平面。

使用顶点着色器时，应用程序负责应用透视转换。 这可能会导致任意剪裁平面出现问题，因为如果它包含异态刻度因子，则还需要转换剪辑平面。 因此，最好在任意剪辑器中提供未投影的顶点位置，这是由 texkill 运算符标识的纹理坐标集。

此说明按如下所示使用：
texkill tn // 像素掩码按如下所示完成：如果（textureCoordinates 的 x，y，z 组件 (阶段 n) $_{UVWQ}$< 0) 取消像素呈现

对于像素着色器 1_1、1_2 和 1_3，texkill 对目标寄存器编号给定的纹理坐标进行运算。 但在版本 1_4 中，texkill 对 纹理坐标寄存器 中包含的数据执行操作， (tn) 或临时寄存器 (已指定为目标的 rn) 。

启用多重采样后，由于多重采样，在多边形边缘上实现的任何抗锯齿效果都不会沿 texkill 生成的任何边缘实现。 像素着色器每像素运行一次。

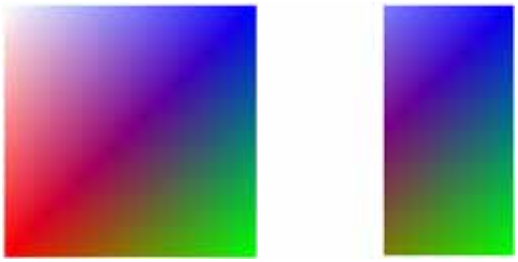
本例只用于演示。

此示例屏蔽具有负纹理坐标的像素。 像素颜色从顶点数据中提供的顶点颜色内插。

```
ps_1_1       // Version instruction
texkill t0   // Mask out pixel using texture coordinates from stage 0
mov r0, v0   // Move the diffuse color in v0 to r0

// The rendered output from the pixel shader is shown below
```

纹理坐标范围为 -0.5 到 0.5，v 中为 0.0 到 1.0。 此指令导致负 u 值被屏蔽。下图显示了应用于象限的顶点颜色，而未应用 texkill 指令。 下图显示了 texkill 指令的结果。 纹理坐标低于 0 的像素颜色 (x 从 -0.5 到 0.0) 被屏蔽。背景色 (白色) 用于屏蔽像素颜色的位置。



纹理坐标数据在本示例中的顶点数据声明中声明。

```
struct CUSTOMVERTEX
{
    FLOAT x, y, z;
    DWORD color;
    FLOAT tu1, tv1;
};

#define D3DFVF_CUSTOMVERTEX
(D3DFVF_XYZ|D3DFVF_DIFFUSE|D3DFVF_TEX1|D3DTEXCOORD2(0))

static CUSTOMVERTEX g_Vertices[]=
{
    //  x      y     z    color         u1,    v1
    { -1.0f, -1.0f, 0.0f, 0xffff0000, -0.5f,  1.0f, },
    {  1.0f, -1.0f, 0.0f, 0xff00ff00,  0.5f,  1.0f, },
    {  1.0f,  1.0f, 0.0f, 0xff0000ff,  0.5f,  0.0f, },
    { -1.0f,  1.0f, 0.0f, 0xffffffff, -0.5f,  0.0f, },

};
```

相关主题

像素着色器说明

texld - ps_1_4

项目 • 2022/09/22

使用源寄存器的内容作为纹理坐标 (RGBA) 加载目标寄存器。 采样纹理是与目标寄存器号关联的纹理。

texld dst，src

寄存器

值	说明	vn	快递 之 家	Tn	Rn	像素着色器版本
dst	目标注册				x	1_4
src	源寄存器			x		1_4 阶段 1
				x	x	1_4 阶段

使用 r (n) 作为源寄存器时， (XYZ) 的前三个组件必须在着色器的上一阶段初始化。

若要了解有关注册的详细信息，请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器。

注解

本指令对与目标寄存器号关联的纹理阶段中的纹理采样。 使用源寄存器中的纹理坐标数据对纹理采样。

texld 和 texcrd 指令的语法公开了对具有纹理寄存器修饰符的投影除法的支持。 对于像素着色器版本 1.4，始终忽略D3DTTFF_PROJECTED纹理转换标志。

使用 texld 的规则：

1. 必须在 texcrd 或 texld 指令中对单个 t (n) 寄存器的每个读取应用相同的 .xyz 或 .xyw 修饰符。 如果在 t (n) 寄存器读取上使用 .xyw，则可以将其与其他使用 .xyw_dw 的同一 t (n) 寄存器的其他读取混合。
2. _dz源修饰符仅在具有 r (n) 源 (寄存器的 texld 上有效，因此阶段 2 仅) 。
3. 每个着色器可以使用_dz源修饰符不超过两次。

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texld				x					

示例

texld 指令提供一些控制源纹理坐标数据的组件。 texld 的完整允许语法集如下所示，并包括所有有效的源寄存器修饰符、选择器和写入掩码组合。

```
texld  r(m), t(n).xyz
// Uses xyz from t(n) to sample 1D, 2D, or 3D texture
```

```
texld  r(m), t(n)
// Same as previous
```

```
texld  r(m), t(n).xyw
// Uses xyw (skipping z) from t(n) to sample 1D, 2D or 3D texture
```

```
texld  r(m), t(n)_dw.xyw
// Samples 1D or 2D texture at x/w, y/w from t(n). The result
// is undefined for a cube-map lookup.
```

```
texld  r(m), r(n).xyz
// Samples 1D, 2D, or 3D texture at xyz from r(m)
// This is possible in the second phase of the shader
```

```
texld  r(m), r(n)
// Same as previous
```

```
texld  r(m), r(n)_dz.xyz
// Samples 1D or 2D texture at x/z, y/z from r(m)
// Possible only in second phase
// The result is undefined for a cube-map lookup
```

```
texld  r(n), r(n)_dz
// Same as previous
```

相关主题

像素着色器说明

texld - ps_2_0和向上

项目 • 2022/09/22

使用提供的纹理坐标对特定采样器中的纹理采样。 此指令不同于 texld - ps_1_4 像素着色器版本 1_4 中使用的指令。

语法

```
texld dst、src0、src1
```

其中：

- dst 是目标寄存器。
- src0 是一个源寄存器，它提供纹理样本的纹理坐标。
- src1 标识 Sampler (Direct3D 9 asm-ps) (s#) ，其中 # 指定要采样的纹理采样器编号。 采样器已将其关联到纹理和 **D3DSAMPLERSTATETYPE** 定义的采样器状态。

ps_2_0和ps_2_x

dst 必须是 临时注册 (r#) ，仅允许 .xyzw 掩码 (默认掩码) 。

src0 必须是 纹理坐标寄存器 (t#) 或 临时寄存器 (r#) ，没有修饰符或重排。

src1 必须是 Sampler (Direct3D 9 asm-ps) (s#) ，没有修饰符或重排。

如果未 (在D3DPSHADERCAPS2_0) 中设置 D3DD3DPSHADERCAPS2_0_NODEPENDENTREADLIMIT上限位，则给定纹理指令 (texld、texldp、texldb - ps、texldd) 可能依赖于最多第三个顺序。 第一顺序依赖纹理指令是纹理指令，其中任一操作之一：

- src0 是临时 注册 (r#) 。
- dst 以前是写的，现在又被写了。

二阶依赖纹理指令定义为纹理指令，该指令读取或写入 临时寄存器 (r#) ，在执行纹理指令之前 (，其内容可能间接) 依赖于第一顺序依赖纹理指令的结果。 (n 个) 从 (n - 1) 顺序纹理指令派生。

ps_3_0

src1 必须是 Sampler (Direct3D 9 asm-ps) (s#) ，没有修饰符。 src0 或 src1 允许重排。 重排在纹理查找之前应用于纹理坐标。

注解

以下版本支持此说明：

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texld					x	x	x	x	x

src0 执行纹理样本所需的坐标数取决于 src1 的声明方式以及 .w 组件。 采样器类型声明为 dcl_samplerType (sm2、sm3 - ps asm) 。 如果 src1 声明为 2D 采样器，则 src0 必须包含 .xy 坐标;如果 src1 声明为多维数据集采样器或卷采样器，则 src0 必须包含 .xyz 坐标。 允许对维度小于纹理坐标中存在的纹理采样，因为忽略了额外的纹理坐标组件 () 。

如果源纹理包含的组件少于 4 个，则默认值将放置在缺少的组件中。 默认值取决于纹理格式，如下表所示：

纹理格式	默认值
D3DFMT_R5G6B5、D3DFMT_R8G8B8、D3DFMT_L8、D3DFMT_L16、D3DFMT_R3G3B2、D3DFMT_CxV8U8、D3DFMT_L6V5U5	A = 1.0
D3DFMT_V8U8、D3DFMT_V16U16、D3DFMT_G16R16、D3DFMT_G16R16F、D3DFMT_G32R32F	B = A = 1.0
D3DFMT_A8	R = G = B = 0.0
D3DFMT_R16F，D3DFMT_R32F	G = B = A = 1.0
所有深度/模具格式	R = B = 0.0，A = 1.0

相关主题

像素着色器说明

texldb - ps

项目 • 2022/09/23

有偏差的纹理加载指令。 此指令使用第四个元素 (.a 或 .w) 在采样之前对纹理采样级别进行偏差。

语法

```
texldb dst、src0、src1
```

其中：

- dst 是目标寄存器。
- src0 是一个源寄存器，提供纹理样本的纹理坐标。 请参阅 纹理坐标寄存器。
- src1 标识 Sampler (Direct3D 9 asm-ps) (s#) ，其中 # 指定要采样的纹理采样器编号。 采样器与它关联的纹理和 由 D3DSAMPLERSTATETYPE 定义的采样器状态。

有关使用 texldb 时的限制，请参阅 texld - ps_2_0和上 一说明。

ps_2_0和ps_2_x

dst 必须是 临时注册 (r#) ，并且只允许使用 .xyzw 掩码 (默认掩码) 。

src0 必须是 纹理坐标寄存器 (t#) 或 临时寄存器 (r#) ，没有修饰符或重排。

src1 必须是 Sampler (Direct3D 9 asm-ps) (s#) ，没有修饰符或重排。

如果在D3DPSHADERCAPS2_0) 中未 (设置 D3DD3DPSHADERCAPS2_0_NODEPENDENTREADLIMIT上限位，则给定纹理指令 (texld、texldp、texldb、texldd) 可能依赖于最多第三个顺序。 一阶依赖纹理指令是一种纹理指令，其中任一：

- src0 是一个 临时寄存器 (r#) 。
- dst 以前是写的，现在又被写了。

二阶依赖纹理指令定义为纹理指令，该指令读取或写入 临时寄存器 (r#) ，其内容在执行纹理指令之前，可能间接) 依赖于 (第一顺序依赖纹理指令的结果。 (n 个) 从 (n - 1) 次序纹理指令派生。

ps_3_0

src1 必须是 Sampler (Direct3D 9 asm-ps) (s#) ，没有修饰符。 src1 上允许重排，应用时，纹理查找结果会在写入到 dst 之前预先重排。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texldb					x	x	x	x	x

texldb 通过 src0.w 中 (有符号) 值，以正常方式计算 mipmap 详细级别。 正偏差值将导致选择较小的 mipmap，反之亦然。 对于ps_2_0和ps_2_x，偏差值可以位于 [-3.0， +3.0] 范围内。 对于ps_3_0，偏差值可以位于 [-16.0， +15.0] 范围内。 这些范围之外的偏差值会产生未定义的结果。 采样器状态D3DSAMP_MIPMAPLODBIAS仍受遵循，texldb 偏差会添加到此位置，但基于每个像素。 计算有偏差的详细信息级别后，仍会遵循D3DSAMP_MAXMIPLEVEL并发生纹理样本。 texldb 后，src0 的内容不受影响， (除非 dst 是相同的寄存器) 。

src0 执行纹理样本所需的坐标数取决于 src1 的声明方式，以及 .w 组件。 采样器类型使用 dcl_samplerType (sm2、sm3 - ps asm) 声明。 如果 src1 声明为 2D 采样器，则 src0 必须包含 .xyw 坐标;如果 src1 声明为多维数据集采样器或卷采样器，则 src0 必须包含 .xyzw 坐标。 允许使用 .xyzw 坐标对 2D 纹理采样， (忽略 .z 坐标) 。

如果源纹理包含的组件少于 4 个，则默认值将放置在缺少的组件中。 默认值取决于纹理格式，如下表所示：

纹理格式	默认值
D3DFMT_R5G6B5、D3DFMT_R8G8B8、D3DFMT_L8、D3DFMT_L16、D3DFMT_R3G3B2、D3DFMT_CxV8U8、D3DFMT_L6V5U5	A = 1.0
D3DFMT_V8U8、D3DFMT_V16U16、D3DFMT_G16R16、D3DFMT_G16R16F、D3DFMT_G32R32F	B = A = 1.0
D3DFMT_A8	R = G = B = 0.0
D3DFMT_R16F、D3DFMT_R32F	G = B = A = 1.0
所有深度/模具格式	R = B = 0.0，A = 1.0

相关主题

像素着色器说明

texldd - ps

项目 • 2022/09/24

使用其他渐变输入对纹理进行采样。

语法

```
texldd、dst、src0、src1、src2、src3
```

其中：

- dst 是目标寄存器。
- src0 是一个源寄存器，提供纹理样本的纹理坐标。 请参阅 纹理坐标寄存器。
- src1 标识源采样器寄存器 (s#) ，其中 # 指定要采样的纹理采样器编号。 采样器与它关联的纹理和 由 D3DSAMPLERSTATETYPE 枚举 (定义的控件状态。 D3DSAMP_MINFILTER) 。
- src2 是指定 x 渐变的输入源寄存器。
- src3 是一个输入源寄存器，用于指定 y 渐变。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texldd						x *	x	x	x

* 此指令仅受 ps_2_a 支持。 ps_2_b不支持它。 有关配置文件的详细信息，请参阅 D3DXGetPixelShaderProfile。

本指令使用 src0 的纹理坐标、src1 指定的采样器以及来自 src2 和 src3 的渐变 DSX 和 DSY 采样纹理。 x 和 y 渐变值用于选择纹理的相应 mipmap 级别进行采样。

所有源都支持任意重排。

所有写入掩码在目标上都有效。

相关主题

像素着色器说明

texldl - ps

项目 • 2022/09/23

使用特定采样器对纹理进行采样。 要采样的特定 mipmap 详细级别必须指定为纹理坐标的第四个组件。

语法

```
texldl dst , src0 , src1
```

其中：

- dst 是目标寄存器。
- src0 是一个源寄存器，它提供纹理样本的纹理坐标。 请参阅 纹理坐标寄存器。
- src1 标识源采样器寄存器 (s#) ，其中 # 指定要采样的纹理采样器编号。 采样器已将其关联到纹理和 D3DSAMPLERSTATETYPE 枚举定义的控件状态 (，例如 D3DSAMP_MINFILTER) 。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texldl								x	x

texldl 在 src1 引用的采样器阶段查找纹理集。 从 src0.w 中选择详细信息级别。 此值可以是负值，在这种情况下，选择的详细级别是使用 MAGFILTER 的最大地图) (第 1 个 (最大地图。 由于 src0.w 是浮点值，因此如果 MIPFILTER 在两个 mip 级别之间) ，则小数值用于内插 (。 采样器状态 MIPMAPLODBIAS 和 MAXMIPLEVEL 受到尊重。 有关采样器状态的详细信息，请参阅 D3DSAMPLERSTATETYPE。

如果采样器中的着色器程序样本没有纹理集，则会在目标寄存器中获取 0001。

下面是引用设备遵循的粗略算法：

```
LOD = src0.w + LODBIAS;
if (LOD <= 0 )
```

```
{
   LOD = 0;
   Filter = MagFilter;
   tex = Lookup( MAX(MAXMIPLEVEL, LOD), Filter );
}
else
{
   Filter = MinFilter;
   LOD = MAX( MAXMIPLEVEL, LOD );
   tex = Lookup( Floor(LOD), Filter );
   if( MipFilter == LINEAR )
   {
      tex1 = Lookup( Ceil(LOD), Filter );
      tex = (1 - frac(src0.w))*tex + frac(src0.w)*tex1;
   }
}
```

限制：

- 纹理坐标不应按纹理大小缩放。
- dst 必须是 临时注册 (r#) 。
- dst 可以接受写掩码。 请参阅 目标注册写入掩码。
- 缺少组件的默认值为 0 或 1，具体取决于纹理格式。
- src1 必须是 Sampler (Direct3D 9 asm-ps) (s#) 。 src1 可能不会使用反门修饰符，(请参阅 目标寄存器写入掩码) 。 src1 可以使用重排 (查看 源寄存器重 排) ，该) 在采样后应用，但在写入掩码之前， (看到目标注册写入掩码) 得到遵守。 采样器必须使用 dcl_samplerType (sm2、sm3 - ps asm) 着色器开头的) 声明 (。
- 执行纹理样本所需的坐标数取决于采样器声明方式。 如果声明为多维数据集，则需要三分量的纹理坐标 (.rgb) 。 验证强制要求提供给tex_ldl的坐标足以满足为采样器声明的纹理维度。 但是，不能保证应用程序实际上通过 API) 设置纹理 (，其维度等于为采样器声明的维度。 在这种情况下，运行时将尝试检测 (可能仅在调试) 不匹配。 允许采样维度小于纹理坐标中存在的纹理，并假定忽略额外的纹理坐标组件。 相反，对纹理坐标中具有比纹理坐标中存在的维度多的纹理采样是非法的。
- 如果 src0 (纹理坐标) 为 临时寄存器，则必须事先编写上述查找 () 所需的组件。
- 采样无符号 RGB 纹理将导致浮点值介于 0.0 和 1.0 之间。
- 采样有符号纹理将导致 -1.0 到 1.0 之间的浮点值。
- 采样浮点纹理时，Float16 表示数据将在MAX_FLOAT16范围内。 Float32 表示将使用管道的最大范围。 未定义这两个范围之外的采样。
- 没有依赖读取限制。

相关主题

像素着色器说明

texldp - ps

项目 • 2022/09/22

投影纹理加载指令。 此指令在采样之前将输入纹理坐标除以第四个元素 (.a 或 .w) 。

语法

```
texldp dst， src0， src1
```

其中

- dst 是目标寄存器。
- src0 是一个源寄存器，它提供纹理样本的纹理坐标。 请参阅 纹理坐标寄存器。
- src1 标识 Sampler (Direct3D 9 asm-ps) (s#) ，其中 # 指定要采样的纹理采样器编号。 采样器已将其关联到纹理和 **D3DSAMPLERSTATETYPE** 定义的采样器状态。

有关使用 texldp 时的限制集，请参阅 texld。

注解

在执行示例之前，texldp 对从 src0 读取的坐标执行投影。 每个纹理坐标被 src0.w 除以，然后对纹理进行采样。 当 texldp 完成时，src0 的内容不会受到影响， (除非 dst 是相同的寄存器) 。 使用 texldp 的替代方法是在着色器中手动执行投影除法。 但是，在着色器代码中执行除法通常比 texldp 指令执行时慢，因此尽可能避免此类额外的数学运算。

src0 执行纹理样本所需的坐标数取决于 src1 的声明方式以及 .w 组件。 采样器类型声明为 dcl_samplerType (sm2、sm3 - ps asm) 。 如果 src1 声明为 2D 采样器，则 src0 必须包含 .xyw 坐标;如果 src1 声明为多维数据集采样器或卷采样器，则 src0 必须包含 .xyzw 坐标。 允许使用 .xyzw 坐标的 2D 纹理采样 () 忽略 .z 坐标。

如果源纹理包含的组件少于 4 个，则默认值将放置在缺少的组件中。 默认值取决于如下表所示的纹理格式。

纹理格式	缺少组件的默认值
D3DFMT_R5G6B5、D3DFMT_R8G8B8、D3DFMT_L8、D3DFMT_L16、D3DFMT_R3G3B2、D3DFMT_CxV8U8、D3DFMT_L6V5U5	A = 1.0

纹理格式	缺少组件的默认值
D3DFMT_V8U8、D3DFMT_V16U16、D3DFMT_G16R16、D3DFMT_G16R16F、D3DFMT_G32R32F	B = A = 1.0
D3DFMT_A8	R = G = B = 0.0
D3DFMT_R16F，D3DFMT_R32F	G = B = A = 1.0
所有深度/模具格式	R = B = 0.0，A = 1.0

以下版本支持此说明：

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texldp					x	x	x	x	x

ps_2_0和ps_2_x

dst 必须是 临时注册 (r#) ，仅允许 .xyzw 掩码 (默认掩码) 。

src0 必须是 纹理坐标寄存器 (t#) 或 临时寄存器 (r#) ，没有修饰符或重排。

src1 必须是 Sampler (Direct3D 9 asm-ps) (s#) ，没有修饰符或重排。

如果未在 (D3DPSHADERCAPS2_0) 中设置 D3DD3DPSHADERCAPS2_0_NODEPENDENTREADLIMIT上限位，则给定纹理指令 (texld、texldp、texldb - ps、 texldd) 可能依赖于最多第三个顺序。 第一顺序依赖纹理指令是纹理指令，其中任一操作之一：

- src0 是 临时注册 (r#)
- dst 以前是写的，现在又被写了。

二阶依赖纹理指令定义为纹理指令，该指令读取或写入 临时寄存器 (r#) ，在执行纹理指令之前 (，其内容可能间接) 依赖于第一顺序依赖纹理指令的结果。 (n 个) 从 (n - 1) 顺序纹理指令派生。

ps_3_0

对于 ps_3_0，src1 必须是 Direct3D 9 asm-ps) (s#) 的采样 (器 ，没有修饰符。 在 src1 上允许重排，应用时，纹理查找结果会在写入到 dst 之前预先重排。

相关主题

像素着色器说明

texm3x2depth - ps

项目 • 2022/09/22

计算要在此像素的深度测试中使用的深度值。

语法

```
texm3x2depth dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x2depth			x						

此指令必须与 texm3x2pad - ps 指令一起使用。

使用这两个指令时，纹理寄存器必须使用以下序列。

```
tex t(n)                    // Define tn as a standard 3-vector.(tn must be
                            // defined in some way before it is used
texm3x2pad   t(m),   t(n)   // Where m > n
                            // Calculate z value
texm3x2depth t(m+1), t(n)   // Calculate w value; use both z and w to
                            // find depth
```

使用点积操作查找 z 和 w 后，深度计算完成。 下面是有关如何完成深度计算的更多详细信息。

texm3x2pad 指令计算 z。

z = textureCoordinates (stage m) $_{UVW}$ * t (n) $_{RGB}$

texm3x2depth 指令计算 w.

w = textureCoordinates (stage m+1) $_{UVW}$ * t (n) $_{RGB}$

计算深度并将结果存储在 t (m+1) 中。

```
if (w == 0)
  t(m+1) = 1.0
else
  t(m+1) = z/w
```

计算深度标记用于像素的深度测试，替换像素的现有深度测试值。

请务必将 z/w 固定在 (0-1) 范围内。 如果 z/w 超出此范围，则深度缓冲区中存储的结果将未定义。

运行 texm3x2depth 后，注册 t (m+1) 不再可用于着色器。

进行多重采样时，使用此指令可以消除分辨率较高的深度缓冲区的大部分好处。 由于像素着色器每像素运行一次，因此 texm3x2depth 或 texdepth 的 单个深度值输出将用于每个子像素深度比较测试。

相关主题

像素着色器说明

texm3x2pad - ps

项目 • 2022/09/22

执行双行矩阵乘法的第一行乘法。 此指令必须与 texm3x2tex - ps 或 texm3x2depth - ps 结合使用。 有关使用 texmpad 的详细信息，请参阅以下任一说明。

语法

```
tex3x2pad dst， src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x2pad	x	x	x						

无法自行使用此指令。

相关主题

像素着色器说明

texm3x2tex - ps

项目 • 2022/09/23

执行 3x2 矩阵乘法的最后一行，并使用结果执行纹理查找。 texm3x2tex 必须与 texm3x2pad - ps 指令结合使用。

语法

```
texm3x2tex dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x2tex	x	x	x						

该指令用作表示 3x2 矩阵乘法运算的两个指令之一。 此指令必须与 texm3x2pad - ps 指令一起使用。

使用这两个指令时，纹理寄存器必须使用以下序列。

```
tex t(n)                    // Define tn as a standard 3-vector (tn must
                            // be defined in some way before it is used)
texm3x2pad  t(m),   t(n)    // where m > n
                            // Perform first row of matrix multiply
texm3x2tex  t(m+1), t(n)    // Perform second row of matrix multiply
                            // to get (u,v) to sample texture
                            // associated with stage m+1
```

下面更详细地介绍了如何实现 3x2 乘法。

texm3x2pad 指令执行乘法的第一行以查找 u'。

u' = t (n) $_{RGB}$ * TextureCoordinates (阶段 m) $_{UVW}$

texm3x2tex 指令执行乘法的第二行以查找 v'。

v' = t (n) $_{RGB}$ * TextureCoordinates (阶段 m+1) $_{UVW}$

texm3x2tex 指令采样阶段 (m+1) 上的纹理，（u'，v'），并将结果存储在 t (m+1) 中。

t (m+1) $_{RGB}$ = TextureSample (stage m+1) $_{RGB}$ ，使用 (u'， v') 作为坐标

示例

下面是一个示例着色器，其中标识了纹理映射和纹理阶段。

```
ps_1_1
tex t0                  // Bind texture in stage 0 to register t0
texm3x2pad  t1,  t0     // First row of matrix multiply
texm3x2tex  t2,  t0     // Second row of matrix multiply to get (u,v)
                        // with which to sample texture in stage 2
mov r0, t2              // Output result
```

此示例需要以下纹理阶段中的以下纹理。

- 阶段 0 采用具有 (x，y，z) 扰动数据的地图。
- 阶段 1 保存纹理坐标。 纹理阶段不需要纹理。
- 阶段 2 保存纹理坐标以及该纹理阶段的二维纹理集。

相关主题

像素着色器说明

texm3x3 - ps

项目 • 2022/09/24

与两个 texm3x3pad 结合使用时执行 3x3 矩阵乘法 - ps 指令。

语法

```
texm3x3 dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x3		x	x						

此指令与 texm3x3tex - ps 指令相同，无需纹理查找。

此指令用作表示 3x3 矩阵乘法运算的三个指令中的最后一个指令。 3x3 矩阵由第三个纹理阶段的纹理坐标和上述两个纹理阶段组成。 将忽略分配给这三个纹理阶段中的任何一个纹理。

此指令必须与两个 texm3x3pad 指令一起使用。 纹理寄存器必须遵循以下顺序。

```
tex t(n)                 // Define tn as a standard 3-vector (tn must
                         // be defined in some way before it is used)
texm3x3pad t(m),   t(n)  // where m > n
                         // Perform first row of matrix multiply
texm3x3pad t(m+1), t(n)  // Perform second row of matrix multiply
texm3x3    t(m+2), t(n)  // Perform third row of matrix multiply to get a
                         // 3-vector result
```

下面是有关如何完成 3x3 乘法的更多详细信息。

第一个 texm3x3pad 指令执行乘法的第一行来查找 u'。

u' = textureCoordinates (stage m) $_{UVW}$ * t (n) $_{RGB}$

第二个 texm3x3pad 指令执行乘法的第二行以查找 v'。

v' = textureCoordinates (stage m+1) $_{UVW}$ * t (n) $_{RGB}$

texm3x3tex 指令执行乘法的第三行以查找 w'。

w' = textureCoordinates (stage m+2) $_{UVW}$ * t (n) $_{RGB}$

将矩阵的结果放在目标寄存器中。

t (m+2) $_{RGBA}$ = (u' ， v' ， w' ， 1)

相关主题

像素着色器说明

texm3x3pad - ps

项目 • 2022/09/24

执行三行矩阵相乘的第一行或第二行乘法。 此指令必须与 texm3x3 - ps、 texm3x3spec - ps、 texm3x3vspec - ps 或 texm3x3tex - ps 结合使用。

语法

```
texm3x3pad dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x3pad	x	x	x						

无法自行使用此指令。

相关主题

像素着色器说明

texm3x3spec - ps

项目 • 2022/09/23

执行 3x3 矩阵相乘，并使用结果执行纹理查找。 这可用于反射和环境映射。 texm3x3spec 必须与两个 texm3x3pad 结合使用 - ps 指令。

语法

```
texm3x3spec dst , src0 , src1
```

其中

- dst 是目标寄存器。
- src0 和 src1 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x3spec	x	x	x						

此指令执行 3x3 矩阵乘法的最后一行，使用生成的向量作为普通向量来反射眼光向量，然后使用反射向量执行纹理查找。 着色器从常量寄存器读取眼光矢量。 3x3 矩阵乘法通常用于将正向量定向到呈现表面的正确正切空间。

3x3 矩阵由第三个纹理阶段的纹理坐标和前面的两个纹理阶段组成。 生成的帖子反射矢量 (u、v、w) 用于对最终纹理阶段的纹理采样。 将忽略分配给上述两个纹理阶段的任何纹理。

此指令必须与两个 texm3x3pad 指令一起使用。 纹理寄存器必须使用以下序列。

```
tex t(n)                    // Define tn as a standard 3-vector (tn must
                            //   be defined in some way before it is used)
texm3x3pad t(m),   t(n)     //   where m > n
                            // Perform first row of matrix multiply
texm3x3pad  t(m+1), t(n)    // Perform second row of matrix multiply
```

```
texm3x3spec t(m+2), t(n), c0  // Perform third row of matrix multiply
                              // Then do a texture lookup on the texture
                              //   associated with texture stage m+2
```

第一个 texm3x3pad 指令执行乘法的第一行来查找 u'。

u' = textureCoordinates (stage m) $_{UVW}$ * t (n) $_{RGB}$

第二个 texm3x3pad 指令执行乘法的第二行以查找 v'。

v' = textureCoordinates (stage m+1) $_{UVW}$ * t (n) $_{RGB}$

texm3x3spec 指令执行乘法的第三行以查找 w'。

w' = textureCoordinates (stage m+2) $_{UVW}$ * t (n) $_{RGB}$

然后，texm3x3spec 指令执行反射计算。

(u' ， v' ， w') = 2*[(N*E) / (N*N)]*N - E

其中，常规 N 由

N = (u' ， v' ， w')

和眼光向量 E 由常量寄存器提供

E = c# (可以使用任何常量 register--c0、c1、c2 等---)

最后，texm3x3spec 指令示例 t (m+2) ， (u'，v'，w')，并将结果存储在 t (m+2) 。

t (m+2) $_{RGBA}$ = TextureSample (stage m+2) $_{RGBA}$ ，使用 (u' ， v' ， w') 作为坐标

示例

下面是一个示例着色器，其中包含纹理映射和标识的纹理阶段。

```
ps_1_1
tex t0                    // Bind texture in stage 0 to register t0 (tn must
                          //   be defined in some way before it is used)
texm3x3pad  t1,  t0       // First row of matrix multiply
texm3x3pad  t2,  t0       // Second row of matrix multiply
texm3x3spec t3,  t0,  c#  // Third row of matrix multiply to get a 3-vector
                          // Reflect 3-vector by the eye-ray vector in c#
                          // Use reflected vector to lookup texture in
                          //   stage 3
mov r0, t3                // Output the result
```

此示例需要以下纹理阶段设置。

- 阶段 0 分配有普通数据的纹理映射。 这通常称为颠簸地图。 每个纹素的数据 (XYZ) 法线。 阶段 n 处的纹理坐标定义对此普通地图进行采样的位置。
- 纹理坐标集 m 分配给 3x3 矩阵的第 1 行。 将忽略分配给阶段 m 的任何纹理。
- 纹理坐标集 m+1 分配给 3x3 矩阵的第 2 行。 将忽略分配给阶段 m+1 的任何纹理。
- 纹理坐标集 m+2 分配给 3x3 矩阵的第 3 行。 阶段 m+2 分配卷或多维数据集纹理地图。 纹理 (RGBA) 提供颜色数据。
- 眼光向量 E 由常量寄存器 E = c# 提供。

相关主题

像素着色器说明

texm3x3tex - ps

项目 • 2022/09/22

执行 3x3 矩阵相乘并使用结果执行纹理查找。 texm3x3tex 必须与两个 texm3x3pad 一起使用 - ps 指令。

语法

```
texm3x3tex dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x3tex	x	x	x						

此指令用作表示 3x3 矩阵乘法运算的三个指令中的最后一个指令，后跟纹理查找。 3x3 矩阵由第三个纹理阶段的纹理坐标和前面的两个纹理阶段组成。 生成的三分量向量 (u、v、w) 用于对阶段 3 中的纹理采样。 将忽略分配给上述两个纹理阶段的任何纹理。 3x3 矩阵乘法通常用于将正向量定向到呈现表面的正确正切空间。

此指令必须与 texm3x3pad 指令一起使用。 纹理寄存器必须使用以下序列。

```
tex t(n)                 // Define tn as a standard 3-vector (tn must
                         //   be defined in some way before it is used)
texm3x3pad t(m),   t(n)  //   where m > n
                         // Perform first row of matrix multiply
texm3x3pad t(m+1), t(n)  // Perform second row of matrix multiply
texm3x3tex t(m+2), t(n)  // Perform third row of matrix multiply to get a
                         //   3-vector with which to sample texture
                         //   associated with texture stage m+2
```

下面是有关如何完成 3x3 乘法的更多详细信息。

第一个 texm3x3pad 指令执行乘法的第一行来查找 u'。

u' = textureCoordinates (stage m) $_{UVW}$ * t (n) $_{RGB}$

第二个 texm3x3pad 指令执行乘法的第二行以查找 v'。

v' = textureCoordinates (stage m+1) $_{UVW}$ * t (n) $_{RGB}$

texm3x3spec 指令执行乘法的第三行以查找 w'。

w' = textureCoordinates (stage m+2) $_{UVW}$ * t (n) $_{RGB}$

最后，texm3x3tex 指令示例不会 (m+2) ，(u'、v'、w')，并将结果存储在 t (m+2) 中。

t (m+2) $_{RGBA}$ = TextureSample (stage m+2) $_{RGBA}$ ，使用 (u' ， v' ， w') 作为坐标。

示例

下面是一个示例着色器，其中包含标识的纹理映射和标识的纹理阶段。

```
ps_1_1
tex t0                  // Bind texture in stage 0 to register t0
texm3x3pad  t1,  t0     // First row of matrix multiply
texm3x3pad  t2,  t0     // Second row of matrix multiply
texm3x3tex  t3,  t0     // Third row of matrix multiply to get a
                        // 3-vector with which to sample texture at
mov r0, t3              // stage 3 output result
```

此示例需要以下纹理阶段设置。

- 阶段 0 分配有普通数据的纹理映射。 这通常称为颠簸地图。 每个纹素的数据 (XYZ) 法线。 纹理坐标集 0 定义如何采样此普通地图。
- 纹理坐标集 1 分配给 3x3 矩阵的第 1 行。 将忽略分配给阶段 1 的任何纹理。
- 纹理坐标集 2 分配给 3x3 矩阵的第 2 行。 将忽略分配给阶段 2 的任何纹理。
- 纹理坐标集 3 分配给 3x3 矩阵的第 3 行。 卷或多维数据集纹理应设置为阶段 3，以供转换的 3D 向量查找。

相关主题

像素着色器说明

texm3x3vspec - ps

项目 • 2022/09/24

执行 3x3 矩阵相乘，并使用结果执行纹理查找。 这可用于眼光矢量不常的反射和环境映射。 texm3x3vspec 必须与两个 texm3x3pad 结合使用 - ps 指令。 如果眼光矢量是常量， 则 texm3x3spec - ps 指令将执行相同的矩阵乘法和纹理查找。

语法

texm3x3vspec dst， src

其中

- dst 是目标寄存器。
- src 是源寄存器。

备注

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texm3x3vspec	x	x	x						

此指令执行 3x3 矩阵乘法运算的最后一行，将生成的向量解释为普通向量以反映眼光矢量，然后使用反射向量作为纹理查找的纹理地址。 它的工作方式就像 texm3x3spec - ps 一样，只不过眼光矢量是从纹理坐标的第四个组件中获取的。 3x3 矩阵乘法通常用于将正向量定向到呈现表面的正确正切空间。

3x3 矩阵由第三个纹理阶段的纹理坐标和前面的两个纹理阶段组成。 生成的帖子反射矢量 (UVW) 用于对阶段 3 中的纹理采样。 将忽略分配给上述两个纹理阶段的任何纹理。

此指令必须与 texm3x3pad 指令一起使用。 纹理寄存器必须使用以下序列。

```
tex t(n)                    // Define tn as a standard 3-vector (tn must
                            //   be defined in some way before it is used)
texm3x3pad   t(m),   t(n)   // where m > n
                            // Perform first row of matrix multiply
texm3x3pad   t(m+1), t(n)   // Perform second row of matrix multiply
```

```
texm3x3vspec t(m+2), t(n)   // Perform third row of matrix multiply
                            // Then do a texture lookup on the texture
                            // associated with texture stage m+2
```

第一个 texm3x3pad 指令执行乘法的第一行来查找 u'。

u' = textureCoordinates (stage m) $_{UVW}$ * t (n) $_{RGB}$

第二个 texm3x3pad 指令执行乘法的第二行以查找 v'。

v' = textureCoordinates (stage m+1) $_{UVW}$ * t (n) $_{RGB}$

texm3x3spec 指令执行乘法的第三行以查找 w'。

w' = textureCoordinates (stage m+2) $_{UVW}$ * t (n) $_{RGB}$

texm3x3vspec 指令还执行反射计算。

(u' ， v' ， w') = 2*[(N*E) / (N*N)]*N - E

其中，常规 N 由

N = (u' ， v' ， w')

和眼光向量 E 由

E = (TextureCoordinates (阶段 m) $_{Q}$ ，

textureCoordinates (stage m+1) $_{Q}$ ，

textureCoordinates (stage m+2) $_{Q}$)

最后，texm3x3vspec 指令示例 t (m+2) ， (u'，v'，w')，并将结果存储在 t (m+2) 。

t (m+2) $_{RGBA}$ = TextureSample (stage m+2) $_{RGBA}$ ，使用 (u' ， v' ， w') 作为坐标

示例

下面是一个示例着色器，其中包含标识的纹理映射和标识的纹理阶段。

```
ps_1_1
tex t0                // Bind texture in stage 0 to register t0
texm3x3pad   t1,  t0  // First row of matrix multiply
texm3x3pad   t2,  t0  // Second row of matrix multiply
texm3x3vspec t3,  t0  // Third row of matrix multiply to get a 3-vector
                      // Reflect 3-vector by the eye-ray vector
                      // Use reflected vector to do a texture lookup
```

```
                        //   at stage 3
mov r0, t3              // Output the result
```

此示例需要以下纹理阶段设置。

- 阶段 0 分配有普通数据的纹理映射。 这通常称为颠簸地图。 每个纹素的数据 (XYZ) 法线。 阶段 n 处的纹理坐标定义如何采样此普通地图。
- 纹理坐标集 m 分配给 3x3 矩阵的第 1 行。 将忽略分配给阶段 m 的任何纹理。
- 纹理坐标集 m+1 分配给 3x3 矩阵的第 2 行。 将忽略分配给阶段 m+1 的任何纹理。
- 纹理坐标集 m+2 分配给 3x3 矩阵的第 3 行。 阶段 m+2 分配卷或多维数据集纹理地图。 纹理 (RGBA) 提供颜色数据。
- 眼光向量 E 将传递到第四个分量的指令中， (q) 阶段 m、m+1 和 m+2 的纹理坐标数据。

相关主题

像素着色器说明

texreg2ar - ps

项目 • 2022/09/22

将源寄存器的 alpha 和红色组件解释为纹理地址数据 (u，v) 以采样对应于目标寄存器号的阶段中的纹理。 结果存储在目标寄存器中。

语法

```
texreg2ar dst， src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texreg2ar	x	x	x						

此说明对颜色空间重新映射操作很有用。

下面是指令遵循的序列示例：
tex t (n) texreg2ar t (m) ， t (n) 其中 m > n // 第一个指令使用 tn tex tn // 将纹理颜色 (RGBA) // 加载到寄存器 tn tex tn // 第二个指令将颜色 t (m) $_{RGBA}$ = TextureSample (阶段 m) $_{RGBA}$ 使用 t (n) $_{AR}$ 作为坐标重新映射颜色

_bx2不能用于 texreg2ar 或 texreg2gb 的 src 寄存器 - ps 指令。

对于此说明，源寄存器必须使用未签名的数据。 在源寄存器中使用已签名或混合数据将产生未定义的结果。 有关详细信息，请参阅 D3DFORMAT。

相关主题

像素着色器说明

texreg2gb - ps

项目 • 2022/09/23

将源寄存器的绿色和蓝色组件解释为纹理地址数据，以在对应于目标寄存器编号的阶段对纹理采样。

语法

```
texreg2gb dst， src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texreg2gb		x	x						

此指令可用于颜色空间重新映射操作。

下面是说明遵循的序列的示例。
tex t (n) texreg2gb t (m) ， t (n) 其中 m > n // 第一个指令加载纹理颜色 (RGBA) // 到寄存器 tn tex tn // 第二个指令将颜色 t (m) RGBA = TextureSample (阶段 m) $_{RGBA}$ 使用 t (n) $_{GB}$ 作为坐标重新映射颜色 t (m) $_{RGBA}$

_bx2不能用于 texreg2ar 的 src 寄存器 - ps 或 texreg2gb 指令。

对于本说明，源寄存器必须使用未签名的数据。 在源寄存器中使用已签名或混合数据将产生未定义的结果。 有关详细信息，请参阅 D3DFORMAT。

相关主题

像素着色器说明

texreg2rgb - ps

项目 • 2022/09/22

将源寄存器的红色、绿色和蓝色 (RGB) 颜色组件解释为纹理地址数据，以便在与目标寄存器编号对应的阶段对纹理采样。 结果存储在目标寄存器中。

语法

```
texreg2rgb dst , src
```

其中

- dst 是目标寄存器。
- src 是源寄存器。

注解

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
texreg2rgb		x	x						

此说明对颜色空间重新映射操作很有用。 它支持二维 (二维) 和三维 (三维) 坐标。 它可以与 texreg2ar - ps 或 texreg2gb - ps 一样用于重新映射 2D 数据。 但是，此指令还支持 3D 数据，以便它可以与多维数据集映射和 3D 卷纹理一起使用。

下面是说明所遵循的序列示例。

```
tex t(n)
texreg2rgb t(m), t(n)     where m > n
```

下面是有关如何完成重新映射的更多详细信息。
第一个指令将 RGBA) (纹理颜色加载到寄存器 tn tn // 第二个指令将颜色 t (m) $_{RGBA}$ = textureSample (阶段 m) $_{RGBA}$ 使用 t (n) $_{RGB}$ 作为坐标重新映射颜色

相关主题

像素着色器说明

寄存器

项目 • 2022/09/22

- ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器
- ps_2_0寄存器
- ps_2_x寄存器
- ps_3_0寄存器

相关主题

像素着色器

像素着色器寄存器修饰符

项目 • 2022/09/23

源寄存器修饰符：

- 源寄存器偏差
- 源寄存器反转
- 源寄存器反门
- 源寄存器重排
- 源寄存器缩放 x 2
- 源注册签名缩放
- 目标注册写入掩码

目标注册修饰符：

- 目标注册写入掩码

相关主题

寄存器

像素着色器源寄存器修饰符

项目 • 2022/09/23

在指令运行之前，使用源寄存器修饰符更改从寄存器读取的值。 源寄存器的内容保持不变。 修饰符可用于调整注册数据范围，以便为指令做准备。 一组称为选择器的修饰符将复制或复制单个通道中的数据，（r，g，b，）复制到其他通道。

ps_1_1 - ps_1_4

下表标识支持每个修饰符的版本：

源寄存器修饰符	语法	版本 1_1	版本 1_2	版本 1_3	版本 1_4
偏见	register_bias	X	X	X	X
转化	1 - 注册	X	X	X	X
negate	- 注册	X	X	X	X
按 2 缩放	register_x2				X
签名缩放	register_bx2	X	X	X	X
texld 和 texcrd 修饰符	register_d*	X	X	X	X
源寄存器重排	register.xyzw	X	X	X	X

源寄存器修饰符只能用于算术指令。 它们不能用于纹理地址指令。 这是 2 个修饰符 的缩放 例外。 对于版本 1_1，签名缩放可用于任何 texm* 指令的源参数。 对于版本 1_2 或 1_3，签名缩放可用于任何纹理地址指令的源参数。

某些修饰符特定的限制：

- 反门可以与偏差、有符号缩放或 scalex2 修饰符结合使用。 合并后，将最后运行反门。
- 反转不能与任何其他修饰符组合。
- 反转、否定、偏差、有符号缩放和 scalex2 可与任何选择器结合使用。
- 源寄存器修饰符不应用于常量寄存器，因为它们将导致未定义的结果。 对于版本 1_4，不允许常量上的修饰符，并且验证失败。

ps_2_0及更高版本

对于版本ps_2_0及上一版本，修饰符的数量已简化。

Negate

否定源寄存器的内容。

组件修饰符	说明
- r	源否定

不能在以下指令的第二个源寄存器上使用反门修饰符： m3x2 - ps、 m3x3 - ps、 m3x4 - ps、 m4x3 - ps 和 m4x4 - ps。

像素着色器版本	2_0	2_x	2_sw	3_0	3_sw
-	x	x	x	x	x

绝对值

获取寄存器的绝对值。

像素着色器版本	2_0	2_x	2_sw	3_0	3_sw
abs				x	x

如果从一个或多个常量浮点寄存器读取任何版本 3 着色器 (c#)，则以下任一版本必须为 true。

- 所有常量浮点寄存器都必须使用 abs 修饰符。
- 常量浮点寄存器都不能使用 abs 修饰符。

相关主题

像素着色器寄存器修饰符

源寄存器偏差

项目 • 2022/09/23

从所有组件中减去 0.5。

寄存器

源寄存器。 有关注册类型的详细信息，请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4注册。

注解

注册的内容不会更改。 修饰符仅适用于从寄存器读取的数据。 偏差应用于所有四个颜色通道，（RGBA），如下所示：

```
output = (input - 0.5)
```

效果是修改范围 0 到 1 中的数据，以在 -0.5 到 0.5 范围内。 将偏差应用于此范围之外的数据可能会产生未定义的结果。

> **备注**
>
> 此修饰符与 Source Register Invert 互斥，因此不能应用于同一寄存器。

此修饰符用于算术指令。

示例

本示例执行与 DirectX 6.0 和 7.0 多个纹理语法中的D3DTOP_ADDSIGNED相同的操作。

```
add r0, r0, t0_bias; Shift down by 0.5.
```

相关主题

像素着色器源寄存器修饰符

源寄存器反转

项目 • 2022/09/22

对指定寄存器的每个通道执行 (1 - 值) 计算。

语法

```
1 - register
```

寄存器

源注册。 有关注册类型的详细信息，请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器。

注解

注册的内容不会更改。 修饰符仅应用于从寄存器读取的数据。 反转操作将应用于 RGBA) (所有四个颜色通道。

此修饰符只能与算术指令一起使用。 此外，此修饰符不能与其他 目标寄存器写入掩码结合使用。

示例

此示例使用反转生成寄存器 r1 的补充。

```
mul r0, r0, 1-r1;
```

相关主题

像素着色器源寄存器修饰符

源寄存器反门

项目 • 2022/09/22

在所有寄存器组件上执行反 (y = -x) 。

语法

```
- register
```

寄存器

源注册。 有关注册类型的详细信息，请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器。

注解

注册的内容不会更改。 修饰符仅应用于从寄存器读取的数据。 否定运算应用于 RGBA () 的所有四个颜色通道。

此操作是在同一参数上存在的任何其他修饰符之后执行的。

此修饰符与 源寄存器反转 互斥，因此不能应用于同一寄存器。

此修饰符仅用于算术指令。

示例

以下示例演示如何使用此修饰符。

```
mul r0, r0, -v1;
```

相关主题

像素着色器源寄存器修饰符

源寄存器缩放 x 2

项目 • 2022/09/24

在指令中使用该值之前，请将值乘以 2。

语法

```
register_x2
```

注册

源寄存器。 有关注册类型的详细信息，请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4注册。

注解

注册的内容不会更改。 修饰符仅适用于从寄存器读取的数据。

此修饰符与 Source Register Invert 互斥，因此不能应用于同一寄存器。

此修饰符仅适用于版本 1_4 中的算术指令。

示例

此示例对纹理进行采样，将数据转换为 -1 到 +1 的范围，并计算点积。

```
mul r0, r0, r1_x2;
```

相关主题

像素着色器源寄存器修饰符

源注册签名缩放

项目 • 2022/09/22

从每个通道中减去 0.5，并将结果缩放为 2.0。 名称 bx2 来自偏差和缩放时间-2，即它执行的操作。

语法

```
source register_bx2
```

注册

源注册。 有关注册类型的详细信息，请参阅 ps_1_1__ps_1_2__ps_1_3__ps_1_4注册。

注解

此操作通常用于将数据从 [0.0 扩展到 1.0] 到 [-1.0 到 1.0]。 此修饰符旨在与算术指令一起使用。 此修饰符通常用于点产品指令的输入 (dp3 - ps) 。 对范围 0 到 1 以外的数据使用_bx2可能会产生未定义的结果。

签名的缩放操作将应用于在运行下一条指令之前从寄存器读取的数据。 此操作应用于 RGBA (所有四个颜色通道) ，如下所示：

```
y = 2(x - 0.5)
```

注册的内容不会更改。 修饰符仅适用于从寄存器读取的数据。

此修饰符与 源寄存器反转 互斥，因此无法将其应用于同一寄存器。

版本信息：

- 对于ps_1_0和ps_1_1，可以在任何源寄存器上使用_bx2，以获取表单 texm3x2* 和 texm3x3*的纹理说明。 _bx2不能用于任何其他纹理指令，如 texreg2ar - ps 或 texreg2gb - ps。
- 对于ps_1_2和ps_1_3，可以在任何 tex* 指令的任何源寄存器上使用_bx2，但除了：texreg2ar - ps、 texreg2gb - ps、 texbem - ps 或 texbeml - ps。

示例

此示例对纹理进行采样，将数据转换为 -1 到 +1 的范围，并计算点积。

```
tex t0                          ; Read a texture color.
dp3_sat r0, t0_bx2, v0_bx2      ; Calculate a dot product.
```

相关主题

像素着色器源寄存器修饰符

ps_1_4 texld、texcrd 的源寄存器修饰符

项目 • 2022/09/22

两个像素着色器版本 1_4 纹理地址指令， texld - ps_1_4 和 texcrd - ps，具有自定义语法。 这些说明支持自己的源寄存器修饰符、源寄存器选择器和目标注册写入掩码集，如下所示。

texld 和 texcrd 的源寄存器修饰符

这些修饰符通过将 x 和 y 值除以 z 或 w 值来提供投影除法功能。

源注册修饰符	说明	语法
_dz	将 x，y 组件除以 z	register_dz
_db	将 x，y 组件除以 z	register_db
_dw	将 x，y 组件除以 w	register_dw
_da	将 x，y 组件除以 w	register_da

注解

_dz或_db修饰符执行以下操作：

```
x' = x/z ( x' = 1.0 if z == 0)
y' = y/z ( y' = 1.0 if z == 0)
z' is undefined
w' is undefined
```

_dw或_da修饰符执行以下操作：

```
x' = x/w ( x' = 1.0 if w == 0)
y' = y/w ( y' = 1.0 if w == 0)
z' is undefined
w' is undefined
```

相关主题

像素着色器源寄存器修饰符

源寄存器重排 (HLSL PS 参考)

项目 • 2022/09/22

重排是指将任何源注册组件复制到任何临时注册组件的功能。 重排不会影响源寄存器数据。 在指令运行之前，源寄存器中的数据将复制到临时寄存器。

源重排

源重排允许源寄存器的各个组件在读取寄存器进行计算之前接受同一源寄存器四个组件中的任何一个组件的值。

例如，.zxxy 重排表示：

- .x 组件将接受 .z 组件的值
- .y 组件将接受 .x 组件的值
- .z 组件将接受 .x 组件的值
- .w 组件将接受 .y 组件的值

组件可以按任意顺序显示。 如果指定了少于四个组件，则重复最后一个组件。 例如： 。

```
.xy  = .xyyy
.wzx = .wzxx
.z   = .zzzz
```

如果未指定任何组件，则不会应用重排。

某些说明对源重排有限制。 它们列在所遵循的指令参考页中。

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
.x				x	x	x	x	x	x
.y				x	x	x	x	x	x
.z	x*	x*	x*	x	x	x	x	x	x
.w	x	x	x	x	x	x	x	x	x
.xyzw (默认)	x	x	x	x	x	x	x	x	x
.yzxw					x	x	x	x	x
.zxyw					x	x	x	x	x

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_x	2_sw	3_0	3_sw
.wzyx					x	x	x	x	x
任意重排						x	x	x	x

* 仅当目标写入掩码为 .w (.a) 时可用。

任意重排

重排可以按任意顺序应用于源寄存器;也就是说，任何源寄存器都可以按任意顺序采用任何组件掩码。

复制重排

复制重排将一个组件复制到另一个组件。 这正是 .x、.y、.z、.w 重排组件之一 (或 .r、.g、.b、.a 等效项) 必须指定。

相关主题

像素着色器源寄存器修饰符

ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器

ps_2_0寄存器

ps_2_x寄存器

ps_3_0寄存器

目标注册写入掩码

项目 • 2022/09/23

写入掩码控制在完成指令后写入目标寄存器的组件。 只要组件按 .rgba 或 .xyzw 的顺序排列，就允许输出写入掩码。 也就是说，.rba 和 .xw 是有效的掩码。 纹理寄存器有一组规则，非纹理寄存器有另一组规则。

语法

dst.writemask

其中

- dst 是目标寄存器。
- writemask 是一系列掩码： (x，y，z，w) 或 (红色、绿色、蓝色、alpha) 。

注解

以下目标写入掩码可用。

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_x**	**2_sw**	**3_0**	**3_sw**
.xyzw (默认)	x	x	x	x	x	x	x	x	x
.xyz	x	x	x	x	x	x	x	x	x
.w	x	x	x	x	x	x	x	x	x
任意掩码				x	x	x	x	x	x

任意掩码允许将任意一组通道组合在一起以生成掩码。 通道必须按 r、g、b、a - 例如 register.rba 的顺序列出，用于更新目标的红色、蓝色和 alpha 通道。 任意掩码在版本 1_4 或更高版本中可用。

如果未指定目标写入掩码，则目标写入掩码默认为 rgba 大小写。 换句话说，目标寄存器中的所有通道都会更新。

对于版本 1_0 到 1_3， dp3 - ps dp3 算术指令只能使用 .rgb 或 .rgba 输出写入掩码。

仅算术运算支持目标寄存器写入掩码。 它们不能用于纹理寻址指令，但版本 1_4 指令、texcrd - ps 和 texld - ps_2_0及上。

默认值是写入所有四个颜色通道。

```
// All four color channels can be written by explicitly listing them.
mul r0.rgba, t0, v0

// Or, the default mask can be used to write all four channels.
mul r0, t0, v0
```

alpha 写入掩码也称为标量写入掩码，因为它使用标量管道。

```
add r0.a, t1, v1
```

因此，此指令有效地将 t1 的 alpha 分量和 v1 的 alpha 分量的总和放入 r0.a 中。

颜色写入掩码用于控制写入到颜色通道。

```
// The color write mask is also referred to as the vector write mask,
//   because it uses the vector pipeline.
mul r0.rgb, t0, v0
```

对于版本 1_4，只要掩码按顺序排列 r、g、b、a，目标写入掩码就可以在任何组合中使用。

```
// This example updates the red, blue, and alpha channels.
mov r0.rba, r1
```

联合发出的指令允许同时发出两个可能不同的指令。 这可以通过在 alpha 管道和 RGB 管道中执行指令来实现。

```
  mul r0.rgb, t0, v0
+ add r1.a,   t1, c1
```

这样配对指令的优点是，它允许在向量管道和标量管道中并行执行不同的操作。

这些顶点着色器输出寄存器仅限于以下写入掩码：

注册类型	所需的写入掩码
oFog	此标量寄存器上不允许显式写入掩码
选择	此标量寄存器上不允许显式写入掩码
oPos	.xyzw（，这是默认）
oT#	组合掩码：.x | .xy | .xyz | .xyzw（，这是默认）

相关主题

像素着色器寄存器修饰符

ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器

项目 • 2022/09/22

像素着色器依赖于寄存器来获取顶点数据、输出像素数据、在计算期间保存临时结果以及识别纹理采样阶段。 有几种寄存器类型，每个寄存器都有唯一的功能。 本部分包含像素着色器版本1_X实现的输入和输出寄存器的参考信息。

注册要由像素着色器使用的数据。 以下各节将全面介绍寄存器。

- 寄存器类型描述可用的四种类型的寄存器及其用途。
- 读取端口限制详细介绍了在单个指令中使用多个寄存器的限制。
- 只读读写描述哪些寄存器可用于读取、写入或两者。
- 范围详细说明组件数据的范围。

注册类型

名称	类型	版本 1_1	版本 1_2	版本 1_3	版本 1_4
c#	常量寄存器	8	8	8	8
R#	临时寄存器	2	2	2	6
t#	纹理寄存器	4	4	4	6
V#	颜色寄存器	2	2	2	阶段 2 中的 2

- 常量寄存器包含常量数据。 可以使用 SetPixelShaderConstantF 将数据加载到常量寄存器中，也可以使用 def - ps 定义数据。 常量寄存器不能通过纹理地址指令使用。 唯一的例外是 texm3x3spec - ps 指令，该指令使用常量寄存器提供眼光矢量。

- 临时寄存器用于存储中间结果。 r0 还充当像素着色器输出。 着色器末尾的 r0 值是着色器的像素颜色。

  任何尝试从尚未由上一指令编写的临时寄存器读取的着色器上，着色器验证都将失败 CreatePixelShader 。 D3DXAssembleShader 将同样失败，假设验证已启用，(不使用D3DXSHADER_SKIPVALIDATION) 。

- 纹理寄存器

  对于像素着色器版本 1_1 到 1_3，纹理寄存器包含纹理数据或纹理坐标。 采样纹理时，纹理数据将加载到纹理寄存器中。 纹理采样使用纹理坐标查找或采样指定 (u、v、w、q) 坐标的颜色值，同时考虑纹理阶段状态属性。 纹理坐标数据从顶点纹理

坐标数据内插，并与特定纹理阶段相关联。 纹理阶段编号与纹理坐标声明顺序之间存在默认的一对一关联。 默认情况下，在顶点格式中定义的第一组纹理坐标与纹理阶段 0 相关联。

对于这些像素着色器版本，纹理寄存器的行为就像在算术指令使用时临时寄存器一样。

对于像素着色器版本 1_4，纹理寄存器 (t#) 包含只读纹理坐标数据。 这意味着纹理坐标集和纹理阶段编号彼此独立。 从中采样纹理) 的纹理阶段号 (由目标寄存器号 (r5) 确定。 对于纹理指令，纹理坐标集由源寄存器 (t0 到 t5) 确定，因此纹理坐标集可以映射到任何纹理阶段。 此外，源寄存器 (指定 texld 的纹理坐标) 也可以是临时寄存器 (r#) ，在这种情况下，临时寄存器的内容用作纹理坐标。

- 颜色寄存器包含每像素颜色值。 这些值是通过顶点数据中漫射和反射颜色值的按像素迭代获取的。 对于像素着色器版本 1_4 着色器，颜色寄存器仅在第二阶段可用。

  如果阴影模式设置为D3DSHADE_FLAT，则会禁用顶点颜色 (漫射和反射) 的迭代。 无论阴影模式如何，如果启用了像素雾，则管道仍会迭代雾。 请记住，在管道中应用雾比像素阴影器晚。

  通常，使用顶点漫射颜色数据加载 v0 寄存器。 通常，使用顶点反射颜色数据加载 v1 寄存器。

  输入颜色数据值 (饱和) 固定到范围 0 到 1，因为这是像素着色器中颜色寄存器的有效输入范围。

  像素着色器对颜色寄存器具有只读访问权限。 这些寄存器的内容是迭代值，但迭代的精度可能比纹理坐标低得多。

读取端口限制

读取端口限制指定可在单个指令中用作源寄存器的每个寄存器类型的不同寄存器数。

名称	类型	版本 1_1	版本 1_2	版本 1_3	版本 1_4
c#	常量寄存器	2	2	2	2
R#	临时寄存器	2	2	2	3
t#	纹理寄存器	2	3	3	1
V#	颜色寄存器	2	2	2	阶段 2 中的 2

例如，几乎所有版本的颜色寄存器都有两个读取端口限制。 这意味着单个指令最多可以使用两个不同的颜色寄存器 (v0 和 v1，例如) 作为源寄存器。 此示例显示了在同一指令中使用的两个颜色寄存器：

```
mad r0, v1, c2, v0
```

只读、读/写

根据下表中的只读 (RO) 功能或读/写 (RW) 功能标识寄存器类型。 只读寄存器只能用作指令中的源寄存器;它们永远不能用作目标寄存器。

名称	类型	版本 1_1	版本 1_2	版本 1_3	版本 1_4
名称	类型	1_1	1_2	1_3	1_4
c#	常量寄存器	RO	RO	RO	RO
R#	临时寄存器	RW	RW	RW	RW
t#	纹理寄存器	RW	RW	RW	请参阅以下说明
V#	颜色寄存器	RO	RO	RO	RO

支持 RW 的寄存器可用于存储中间结果。 这包括某些着色器版本的临时寄存器和纹理寄存器。

> **备注**
>
> - 对于像素着色器版本 1_4，纹理寄存器是用于纹理寻址指令的 RO，纹理寄存器既不能读取，也不能由算术指令写入。 此外，由于纹理寄存器已成为纹理坐标寄存器，因此具有 RO 访问不是以前功能的回归。

范围

范围是最大和最小寄存器数据值。 范围因寄存器类型而异。 可以使用 GetDeviceCaps 从设备上限查询某些寄存器的范围。

名称	类型	范围	版本
c#	常量寄存器	-1 到 +1	所有版本
R#	临时寄存器	- PixelShader1xMaxValue 到 + PixelShader1xMaxValue	所有版本
t#	纹理寄存器	- MaxTextureRepeat 到 + MaxTextureRepeat	所有版本
V#	颜色寄存器	0 到 1	所有版本

早期像素着色器硬件使用固定点数表示寄存器中的数据。 这会将精度限制为数字小数部分的大约 8 位。 设计着色器时请记住这一点。

对于像素着色器版本 1_1 到 1_3，MaxTextureRepeat 必须至少为 1。 对于 1_4，MaxTextureRepeat 必须至少为 8。

有关 PixelShader1xMaxValue 的详细信息，请参阅 **D3DCAPS9** 。

相关主题

寄存器

ps_2_0寄存器

项目 • 2022/09/22

像素着色器依赖于寄存器来获取顶点数据、输出像素数据、在计算期间保存临时结果以及识别纹理采样阶段。 有几种类型的寄存器，每个寄存器都具有独特的功能。 本部分包含像素着色器版本2_x实现的输入和输出寄存器的参考信息。

输入寄存器类型

注册	名称	Count	R/W	# 读取端口	# Reads/inst	维度	RelAddr	默认值	需要 DCL
V#	输入颜色寄存器	2	R	1	无限制	4	N	部分 (0001) 。请参阅注释 4	Y
R#	临时寄存器	请参阅注释 1	R/W	3	无限制	4	N	无	N
c#	常量浮点寄存器	32	R	1	2	4	N	0000	N
我#	常量整数寄存器	16	查看注释 2	1	1	4	N	0000	N
B#	常量布尔寄存器	16	查看注释 2	1	1	1	N	FALSE	N
P	谓词寄存器	1	查看注释 2	1	1	1	N	无	Y
s#	采样器 (Direct3D 9 asm-ps)	16	请参阅注释 3	1	1	4	N	参阅注释 5	Y
t#	纹理坐标寄存器	8	R	1	1	4	N	无	Y

注意：

1. 12 分钟/32 最大值：r# 寄存器数由 D3DPSHADERCAPS2_0.NumTemps (确定，范围为 12 到 32) 。

2. 仅可由流控制指令使用。
3. 只有纹理采样指令可用。
4. 部分 (x、y、z、w) - 如果寄存器中只更新了一部分通道，其余通道将默认为指定值 (x、y、z、w) 。
5. 采样器查找的默认值存在，但值取决于纹理格式。

读取端口数是可在单个指令中读取的每个寄存器类型 () 的不同寄存器数。

输出寄存器类型

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
Oc#	输出颜色寄存器	请参阅 Direct3D 9 (多元素纹理)	W	4	N	无	N
oDepth	输出深度寄存器	1	W	1	N	无	N

相关主题

寄存器

ps_2_x寄存器

项目 • 2022/09/22

像素着色器依赖于寄存器来获取顶点数据、输出像素数据、在计算期间保存临时结果以及识别纹理采样阶段。 有几种类型的寄存器，每个寄存器都具有独特的功能。 本部分包含像素着色器版本2_x实现的输入和输出寄存器的参考信息。

输入寄存器类型

注册	名称	Count	R/W	# 读取端口	# Reads/inst	维度	RelAddr	默认值	需要 DCL
V#	输入颜色寄存器	2	R	1	无限制	4	N	部分 (0001) 。请参阅注释 4	Y
R#	临时寄存器	请参阅注释 1	R/W	3	无限制	4	N	无	N
c#	常量浮点寄存器	32	R	1	2	4	N	0000	N
我#	常量整数寄存器	16	查看注释 2	1	1	4	N	0000	N
B#	常量布尔寄存器	16	查看注释 2	1	1	1	N	FALSE	N
P	谓词寄存器	1	查看注释 2	1	1	1	N	无	Y
s#	采样器 (Direct3D 9 asm-ps)	16	请参阅注释 3	1	1	4	N	参阅注释 5	Y
t#	纹理坐标寄存器	8	R	1	1	4	N	无	Y

注意：

1. 12 min/32 max：r# 寄存器数由 D3DPSHADERCAPS2_0.NumTemps (确定，范围为 12 到 32) 。

2. 只能由流控制指令使用。
3. 只有纹理采样指令可用。
4. 部分 (x、y、z、w) - 如果寄存器中只有一部分通道更新，其余通道将默认为指定值 (x、y、z、w) 。
5. 采样器查找的默认值存在，但值取决于纹理格式。

读取端口数是每个寄存器类型 (的不同寄存器数，) 可以在单个指令中读取。

输出寄存器类型

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
Oc#	输出颜色寄存器	请参阅 Direct3D 9 (多元素纹理)	W	4	N	无	N
oDepth	输出深度寄存器	1	W	1	N	无	N

相关主题

寄存器

ps_3_0寄存器

项目 • 2022/09/22

像素着色器依赖于寄存器来获取顶点数据、输出像素数据、在计算期间保存临时结果以及标识纹理采样阶段。 有几种寄存器类型，每个寄存器都有唯一的功能。 本部分包含像素着色器版本 3_0 实现的输入和输出寄存器的参考信息。

新建寄存器

输入寄存器

输入寄存器 (v#) 现已完全浮点， 纹理坐标寄存器 (t#) 已合并到其中。 dcl_semantics (sm3 - ps asm) 在着色器顶部用于描述特定Input_Register中包含的内容。 引入像素类型的语义 (类似于此模型的顶点侧) 。 当输入寄存器定义为颜色 ((如纹理坐标)) 时，不会执行固定。 定义为颜色的寄存器的计算与多重采样时纹理坐标不同。

人脸注册

此模型的人脸寄存器 (vFace) 是新的。 这是一个浮点标量寄存器，最终将包含基元区域。 但是，在ps_3_0中，只有此寄存器的符号有效。 因此，如果值小于零 (则) 基元设置为负 (区域为负，反时针) 。 因此，在ps_3_0将此寄存器与 0 (> 0 或 < 0) 进行比较才有意义。 在像素着色器中，应用程序可以决定使用哪种照明技术。 可以通过这种方式实现双向照明。 此寄存器需要声明，因此未声明的用法将标记为错误。 对于行和点基元，此寄存器未定义。 人脸寄存器只能用作以下说明的条件： setp_comp - ps、 if_comp - ps 或 break_comp - ps。

Loop计数器寄存器

Loop计数器寄存器 (aL) 是此模型的新增功能。 它在循环的每个执行 （ psendloop/ - ps 块) 中自动递增。 如果需要，可以在块中使用它进行相对寻址。 在循环外部使用Loop计数器寄存器无效。

位置寄存器

位置寄存器 (vPos) 是此模型的新增功能。 它包含相应通道中的当前像素 (x、y) 。 (z、w) 通道未定义。 此寄存器需要声明，因此未声明的用法将标记为错误。 声明后，此寄存器必须具有以下掩码之一：.x、.y、.xy。

输入寄存器类型

注册	名称	Count	R/W	# 读取端口	# 读取/inst	维度	RelAddr	默认值	需要 DCL
V#	输入寄存器	10	R	1	无限制	4	铝	无	是
R#	临时注册	32	R/W	3	无限制	4	否	无	否
c#	常量浮点寄存器	224	R	1	无限制	4	否	0000	否
我#	常量整数寄存器	16	R	1	1	4	否	0000	否
B#	常量布尔寄存器	16	R	1	1	1	否	false	否
P	谓词寄存器	1	R	1	1	1	否	无	否
s#	采样器 (Direct3D 9 asm-ps)	16	R	1	1	4	否	请参阅注释 1	是
vFace	Face_Register	1	R	1	无限制	1	否	无	是
vPos	Position_Register	1	R	1	无限制	4	否	无	是
铝	Loop_Counter_Register	1	R	1	无限制	1	不适用	无	否

注意：

- 采样器查找的默认值存在，但值取决于纹理格式。

读取端口数是每个寄存器类型 (的不同寄存器数，) 可以在单个指令中读取。

输出寄存器类型

注册	名称	Count	R/W	维度	RelAddr	默认值	需要 DCL
Oc#	输出颜色寄存器	请参阅 Direct3D 9 (多元素纹理)	W	4	否	无	否
oDepth	输出深度寄存器	1	W	1	否	无	否

相关主题

寄存器

相对寻址 (HLSL PS 参考)

项目 • 2022/09/23

[ ] 语法只能在某些着色器模型中相对寻址的寄存器类型中使用。 支持的 [ ] 语法形式如下所示：

其中：

- “R”表示任何可相对解决的寄存器类型。
- “A”表示任何可用作索引的寄存器，以相对处理其他寄存器。
- n0 - ni、m0 - mj 和 k 是整数 >= 0。

[ ] 语法	有效索引	示例
R[A + m0 + … + mj]	A + m0 + … + mj	c[a0.x + 3 + 7]
R[k] (= Rk)	k	c[10] (= c10)
R[A]	A	c[a0.y]
Rk[n0 + … + ni + A + m0 + … + mj]	A + k + n0 + … + ni + m0 + … + mj	c8[3 + 2 + a0.w + 5 + 6 + 1]
R[n0 + … + ni + A + m0 + … + mj]	A + n0 + … + ni + m0 + … + mj	c[2 + 1 + aL + 3 + 4 + 5]
Rk[A]	A + k	c12[aL] ， c0[a0.z]
Rk[A + m0 + … + mj]	A + k + m0 + … + mj	v1[aL + 4 + 8]
R[n0 + … + ni + A]	A + n0 + … + ni	o[3 + 1 + aL]
Rk[n0 + … + ni + A]	A + k + n0 + … + ni	o1[2 + 1 + 3 + aL]

寄存器在以下版本中可用：

注册类型	像素着色器版本
循环计数器：输入寄存器上的 aL	ps_3_0及更高版本

相关主题

寄存器

输入颜色寄存器

项目 • 2022/09/22

包含顶点颜色的像素着色器输入寄存器。

语法

```
dcl v#.writeMask
```

其中：

- dcl - (sm2， sm3 - ps asm) 是寄存器声明指令。
- v 是输入寄存器，#是寄存器号。 允许的寄存器数由着色器版本确定。
- writeMask 确定写入最多四个) (哪些组件。 有效组件包括： (x、y、z、w) 或 (r、g、b、a) 。

注解

颜色寄存器是只读寄存器。 每个寄存器包含从输入顶点迭代的四个分量 RGBA 值。 它们的精度低于大多数寄存器，保证在 (0、+1) 范围内有 8 位无符号数据。 不能在单个指令中使用多个。

相关主题

寄存器

ps_1_1__ps_1_2__ps_1_3__ps_1_4寄存器

ps_2_0寄存器

ps_2_x寄存器

ps_3_0寄存器

常量布尔寄存器 (HLSL PS 参考)

项目 • 2022/09/23

此寄存器是静态流控制 (指令中使用的位集合，例如，如果 bool -pselse - psendif - - psendif - - ps) 。 其中 16 个，因此着色器可以具有 16 个独立的分支条件。 可以使用 defb 设置它们 - ps 或 **SetPixelShaderConstantB**。

着色器常量的行为在 Direct3D 8 和 Direct3D 9 之间发生了变化。

- 对于 Direct3D 9，使用 defx 将值设置为着色器常量空间的常量。 使用 defx 声明的常量的生存期仅限于执行该着色器。 相反，使用 API SetXXXShaderConstantX 在全局空间中初始化常量设置常量。 在调用 SetxxxShaderConstants 之前，全局 (空间中的常量不会复制到对着色器可见的本地空间) 。
- 对于 Direct3D 8，使用 defx 或 API 设置的常量都向着色器常量空间赋值。 每次执行着色器时，当前着色器都会使用常量，而不考虑用于设置它们的技术。

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_sw**	**2_x**	**3_0**	**3_sw**
常量布尔寄存器							x	x	x

相关主题

寄存器

常量浮点寄存器 (HLSL PS 参考)

项目 • 2022/09/22

像素着色器输入寄存器用于 4D 浮点常量。

可以使用 def - ps 或 **SetPixelShaderConstantF** 设置它们。

着色器常量的行为在 Direct3D 8 和 Direct3D 9 之间发生了变化。

- 对于 Direct3D 9，常量设置为着色器常量空间的 defx 赋值。 使用 defx 声明的常量生存期仅限于该着色器的执行。 相反，使用 API SetXXXShaderConstantX 在全局空间中初始化常量设置常量。 在调用 SetxxxShaderConstants 之前（，全局空间中的常量不会复制到对着色器）可见的本地空间。
- 对于 Direct3D 8，使用 defx 或 API 设置的常量都为着色器常量空间赋值。 每次执行着色器时，当前着色器都会使用常量，而不考虑用于设置它们的技术。

示例

下面是在着色器中声明两个浮点常量的示例。

```
def c40, 0.0f,0.0f,0.0f,0.0f;
```

每次调用 **SetPixelShader** 时，都会加载这些常量。

如果要使用 API 设置常量值，则不需要着色器声明。

相关主题

寄存器

常量整数寄存器 (HLSL PS 引用)

项目 • 2022/09/22

常量整数寄存器仅用于 循环 - ps 和 rep - ps。

可以使用 defi 设置它们 - ps 或 **SetPixelShaderConstantI**。

用作循环的参数时 - ps 指令：

- .x 是迭代计数。 (rep - ps 仅使用此组件) 。
- .y 是循环计数器的初始值。
- .z 是循环计数器的增量步骤。

像素着色器版本	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_sw**	**2_x**	**3_0**	**3_sw**
常量整数寄存器							x	x	x

着色器常量的行为在 Direct3D 8 和 Direct3D 9 之间发生了变化。

- 对于 Direct3D 9，使用 defx 将值设置为着色器常量空间的常量。 使用 defx 声明的常量的生存期仅限于执行该着色器。 相反，使用 API SetXXXShaderConstantX 在全局空间中初始化常量设置常量。 在调用 SetxxxShaderConstants 之前，全局 (空间中的常量不会复制到对着色器可见的本地空间) 。
- 对于 Direct3D 8，使用 defx 或 API 设置的常量都向着色器常量空间赋值。 每次执行着色器时，当前着色器都会使用常量，而不考虑用于设置它们的技术。

相关主题

寄存器

Loop计数器寄存器 (HLSL PS 参考)

项目 • 2022/09/23

此银行中唯一的寄存器是 (aL) 寄存器的当前循环计数器。 它在循环的每个执行（ psendloop/ - ps 块）中自动递增。 因此，它可以在块中用于相对寻址（如果需要）且在循环外部使用它无效。

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_sw	2_x	3_0	3_sw
Loop计数器寄存器								x	x

相关主题

寄存器

谓词寄存器 (HLSL PS 参考)

项目 • 2022/09/22

此像素着色器输出寄存器包含每通道布尔值。

以下版本支持谓词寄存器：

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_sw	2_x	3_0	3_sw
谓词寄存器							x	x	x

下面是寄存器属性。

注册类型	Count	R/W	# 读取端口	# Reads/inst	维度	RelAddr	默认值	需要 DCL
谓词 (p)	1	R/W	1	1	4	不可用	None	N

可以使用 setp_comp - ps 修改谓词寄存器。 此寄存器没有默认值;应用程序在使用前需要设置寄存器。

相关主题

寄存器

Sampler (Direct3D 9 asm-ps)

项目 • 2022/09/23

采样器是像素着色器的输入伪寄存器，用于标识采样阶段。 有 16 个像素着色器采样阶段寄存器：s0 到 s15。 因此，可以在单个着色器传递中读取多达 16 个纹理图面。 使用采样器寄存器的说明为 texld 和 texldp。

在与 dcl_samplerType (sm2、sm3 - ps asm) 指令一起使用之前，必须声明采样器。

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_sw	2_x	3_0	3_sw
取样器					x	x	x	x	x

采样器是伪寄存器，因为你无法直接读取或写入它们。

采样单元对应于纹理采样阶段，封装 **SetSamplerState** 提供的特定于采样的状态。 每个采样器唯一标识单个纹理图面，该图面使用 **SetTexture** 设置为相应的采样器。 但是，可以在多个采样器上设置相同的纹理图面。

在绘图时，无法同时将纹理设置为呈现目标和阶段的纹理。

采样器可能显示为纹理加载指令中的唯一参数： texldl - ps。

在ps_3_0中，如果使用采样器，则需要使用 dcl_samplerType (sm2、sm3 - ps asm) 指令在着色器程序的开头声明它。

采样维度高于纹理坐标中存在的纹理是非法的。 采样尺寸低于纹理坐标的纹理将忽略额外的纹理坐标。

相关主题

寄存器

临时注册 (HLSL PS 参考)

项目 • 2022/09/22

像素着色器输入临时寄存器用于保存中间结果。

语法

```
no declaration is required
```

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_sw	2_x	3_0	3_sw
临时注册					x	x	x	x	x

- 有 12 个像素着色器临时寄存器：r0 到 r11。
- 这些寄存器用于在计算期间存储中间结果。
- 如果临时寄存器使用之前代码中未定义的组件，着色器验证将失败。
- 这些至少是浮点精度。
- 在单个指令中最多可以使用三个。

相关主题

寄存器

纹理坐标寄存器 (HLSL PS 参考)

项目 • 2022/09/22

包含纹理坐标的像素着色器输入寄存器。

像素着色器版本	1_1	1_2	1_3	1_4	2_0	2_sw	2_x	3_0	3_sw
纹理坐标寄存器					x	x	x	x	x

纹理坐标寄存器包含纹理坐标数据。 在使用纹理坐标寄存器之前，它必须由像素着色器声明声明。 有关如何声明纹理寄存器的详细信息，请参阅 dcl - (sm2， sm3 - ps asm) 。

此外，下面是纹理坐标寄存器的一些其他属性。

- 有八个像素着色器纹理坐标寄存器，t0 到 t7。
- 这些是只读寄存器。
- 它们包含从输入顶点迭代的四分量 RGBA 值。
- 它们包含从顶点数据内插的高精度、高动态范围数据值。 使用透视正确的内插生成值。 数据是浮点精度，并且已签名。
- 单个指令中最多有一个。
- 着色器中纹理坐标寄存器的多个读取必须使用相同的 目标寄存器写入掩码。
- 可选的部分精度修饰符 [_pp] 适用于依赖读取。 这是因为部分精度会影响涉及纹理坐标寄存器的算术运算。 它不会影响纹理地址指令的精度，因为它不会影响纹理坐标迭代器。

相关主题

寄存器

输出颜色寄存器

项目 • 2022/09/22

像素着色器颜色输出寄存器 (oC#) 是向多个呈现目标输出结果的仅写寄存器。

语法

Oc#

其中：

名称	说明
oC0	render target #0
oC1	render target #1
oC2	render target #2
oC3	render target #3

注解

- 如果 oCn 已写入，但没有相应的呈现目标，则忽略此写入 oCn。
- 如果) 适用，呈现状态D3DRS_COLORWRITEENABLE、D3DRS_COLORWRITEENABLE1、D3DRS_COLORWRITEENABLE2和D3DRS_COLORWRITEENABLE3确定 oCn 的哪些组件最终会写入混合后 (呈现目标。如果着色器写入由D3DRS_COLORWRITEENABLE* 呈现给定 oCn 寄存器定义的部分组件，则未写入的通道在相应的呈现目标中生成未定义的值。 如果未写入 oCn 的组件，则不能 (更新相应的呈现目标，如上述) 所述，因此D3DRS_COLORWRITEENABLE* 呈现状态不适用。

着色器模型 2 限制

- oCn 只能使用 mov - ps 指令编写。
- oC0 必须始终在着色器中写入。
- 写入任何 oCn 时，不允许 (源重排 (，但 .xyzw = 默认重排) 或源修饰符除外。
- 写入到任何 oCn 时，不允许使用目标写入掩码 (.xyzw = 默认掩码) 或指令修饰符。

- 如果编写 oCn，则还必须编写 oC0 - oCn-1。 例如，若要写入 oC2，还必须写入 oC0 和 oC1。
- 每个着色器最多允许对任何 oC# 进行一次写入。
- 对于ps_2_x和ps_3_0，无法写入动态流控制中的 oC# 和 oD# 寄存器，或者 (静态流控制内写入 oC# 的谓词) 。

着色器模型 3 限制

- 对于ps_3_0，输出寄存器 oC# 和 oD# 可以写入任意次数。 像素着色器的输出来自着色器执行结束时的输出寄存器的内容。 如果对输出寄存器的写入未发生，可能是由于流控制或着色器刚刚未写入，则相应的呈现目标也不会更新。 如果写入输出寄存器中的通道子集，则未定义的值将写入其余通道。
- 对于ps_3_0，可以使用任何写掩码编写 oC# 寄存器。
- 对于ps_2_x和ps_3_0，无法写入动态流控制中的 oC# 和 oD# 寄存器，或者 (静态流控制内写入 oC# 的谓词) 。
- 不能执行任何渐变计算 (或隐式调用渐变计算的操作，例如 texld - ps_2_0 和向上、texldb - ps、 texldp - ps - ps) 流控制语句内部，其分支条件根据每个基元而有所不同， (即：动态流控制指令) 。 指令谓词不被视为动态流控制。

相关主题

寄存器

多个呈现目标 (Direct3D 9)

输出深度寄存器

项目 • 2022/09/23

像素着色器输出深度寄存器 (oDepth) 是一个仅写标量寄存器，其范围 [0..1] 返回针对深度模具缓冲区的深度测试的新深度值。

语法

oDepth

其中：

名称	说明
oDepth	针对深度模具缓冲区的深度测试的新深度值

请务必注意，写入 oDepth 会导致任何特定于硬件的深度缓冲区优化算法（（即分层 Z）加速深度测试性能）丢失。

写入 oDepth 时，需要复制源重排 (.x | .y | .z | .w) 。 不允许显式写入掩码。

写入 oDepth 寄存器会替换内插深度值（，并忽略任何深度偏差/斜率呈现统计信息）。 如果未创建或附加到设备，则忽略写入 oDepth。

如果要对 oDepth 进行多重采样和写入，因为像素着色器每像素仅运行一次，因此深度值将针对所有覆盖的子样本位置进行复制。 深度测试仍按样本进行，但如果没有从像素着色器进行比较，但如果没有编写 oDepth，则不会对每个样本的深度值进行比较。

如果应用程序具有 w 缓冲区集作为其深度缓冲区，则在写入 oDepth 时需要考虑到这一点。 它可能需要将 w 范围信息发送到像素着色器，并计算 w 范围以缩放写入 oDepth 的 w 值。

ps_2_0和ps_2_x限制

- oDepth 只能使用 mov - ps 指令编写，并且只能执行一次。
- 写入 oDepth 时不允许使用源修饰符。
- 写入 oDepth 时不允许使用指令修饰符。
- 在流控制构造中或使用谓词时，不写入 oDepth。

ps_3_0限制

- 对于ps_3_0，输出寄存器 oC# 和 oD# 可以写入任意次数。 像素着色器的输出来自着色器执行结束时的输出寄存器的内容。 如果写入输出寄存器未发生，可能是由于流控制或着色器刚刚未写入，则相应的呈现目标也不会更新。 如果写入输出寄存器中的通道子集，则未定义的值将写入其余通道。
- 只要所有可能的路径最终写入寄存器，就可以在流控制或谓词中写入 oDepth。
- 不能执行任何渐变计算 (或隐式调用渐变计算的操作，例如 texld - ps_2_0 和向上、texldb - ps、 texldp - ps - ps) 流控制语句内部，其分支条件根据每个基元而有所不同，（即：动态流控制指令）。 指令谓词不被视为动态流控制。

相关主题

寄存器

着色器模型 4 程序集

项目 • 2023/03/09

着色器模型 4 要求在 HLSL 中对着色器进行编程。 但是，着色器编译器会将 HLSL 代码编译为在设备上运行的程序集。 如果使用 PIX for Windows 调试着色器，则可以选择在 HLSL 或程序集中显示着色器代码。 本部分列出了调试着色器时可能会遇到的着色器模型 4 和着色器模型 4.1 程序集说明。

指令修饰符
add
and
break
breakc
call
callc
case
continue
continuec
削减
dcl_constantBuffer
dcl_globalFlags
dcl_immediateConstantBuffer
dcl_indexableTemp
dcl_indexRange
dcl_input
dcl_input_sv
dcl_input vPrim
dcl_maxOutputVertexCount
dcl_output
dcl_output oDepth
dcl_output_sgv
dcl_output_siv
dcl_outputTopology
dcl_resource
dcl_sampler
dcl_temps
default
deriv_rtx
deriv_rty
丢弃

div
dne
dp2
dp3
dp4
else
发出
emitThenCut
endif
endloop
endswitch
eq
exp
Frc
ftoi
ftou
ge
iaddieq
if
lge
ilt
imad
imin
imul
ine
ineg
ishl
ishr
itof
label
ld
log
loop
lt
mad
max
min
mov
movc
mul

ne
nop
not
or
地心
Ret
retc
round_ne
round_ni
round_pi
round_z
rsq
sample
sample_b
sample_c
sample_c_lz
sample_d
sample_l
sincos
sqrt
switch
udiv
uge
ult
umad
umax
umin
umul
ushr
utof
xor

着色器模型 4.1 程序集

着色器模型 4.1 支持所有着色器模型 4.0 说明和以下附加说明：

gather4
ld2dms
Lod
sampleinfo
samplepos

相关主题

Asm 着色器参考

着色器模型 4

指令修饰符 (HLSL 参考)

项目 • 2022/09/24

在将指令写入寄存器之前，指令修饰符会影响指令的结果。 着色器模型 4 和着色器模型 4.1 支持以下指令修饰符。

源操作数修饰符

- 绝对值
- 否定

指令结果修饰符

- 饱和

相关主题

着色器模型 4 程序集 (DirectX HLSL)

绝对值

项目 • 2022/09/23

获取算术运算中使用的源操作数的绝对值。

_abs

此修饰符仅用于单精度和双精度浮点和指令。 _abs修饰符强制在源操作数正（包括 INF 值）上 (s) 的符号。

在 NaN 上应用 _abs 会保留 NaN，尽管未定义结果的特定 NaN 位模式。

当_abs与 反门 修饰符组合在一起时，组合会强制符号为负数，就像先应用 _abs 修饰符一样，再应用 **负值**。

最小着色器模型

以下着色器模型中支持此修饰符。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

指令修饰符

Negate

项目 • 2022/09/24

翻转算术运算中使用的源操作数的值的符号。

-

对于单精度和双精度浮点和指令，反精度修饰符在源操作数（包括 INF 值上）翻转数字 (s) 的符号。 在 NaN 上应用 **负值** 会保留 NaN，尽管未定义结果的特定 NaN 位模式。

对于整数指令， **反导** 修饰符采用源操作数中数字 () 的 2 的补数。

最小着色器模型

以下着色器模型中支持此修饰符。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

指令修饰符

饱和 (HLSL 参考)

项目 • 2022/09/22

将单精度浮点算术运算的结果固定到 [0.0f...1.0f] 范围。

_sat

饱和指令结果修饰符从_sat应用于它的浮点算术运算 (s) 对结果值执行以下操作：

`min(1.0f, max(0.0f, value))`

其中，上述表达式中的 min () 和 max () 的行为方式 为 min、 max、 dmin 或 dmax 。

`_sat(NaN)` 按最小值和最大值的规则返回 0。

最小着色器模型

以下着色器模型中支持此修饰符。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

指令修饰符

添加 (sm4 - asm)

项目 • 2022/09/22

以组件方式添加 2 个向量。

add[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要添加到 *src1* 的向量。
src1	[in]要添加到 *src0* 的向量。

注解

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。 F 表示有限实数。

src0 src1->	**-inf**	**-F**	**-denorm**	**-0**	**+0**	**denorm**	**+F**	**+inf**	**NaN**
-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	NaN	NaN
-F	-inf	-F	src0	src0	src0	src0	+-F 或 +-0	+inf	NaN
-denorm	-inf	src1	-0	-0	+0	+0	src1	+inf	NaN
-0	-inf	src1	-0	-0	+0	+0	src1	+inf	NaN
+0	i-inf	src1	+0	+0	+0	+0	src1	+inf	NaN
+denorm	-inf	src1	+0	+0	+0	+0	src1	+inf	NaN
+F	-inf	+-F 或 +-0	src0	src0	src0	src0	+ F	+inf	NaN
+inf	NaN	+inf	+inf	+inf	+inf	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

和 (sm4 - asm)

项目 • 2022/09/22

按位 AND。

and dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]使用 *src1* 将 32 位值设置为 AND。
src1	[in]使用 *src0* 将 32 位值设置为 AND。

注解

src0 和 src1 中每个 配对 32 位值的分量逻辑 AND。 32 位结果放置在 *dest* 中。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

中断 (sm4 - asm)

项目 • 2022/09/23

将执行点移到下一 个尾声或endwitch 之后的指令。

break

备注

令牌格式包含着色器中相应 **尾声** 或 **尾部** 指令的偏移量，以方便起见。

以下示例显示了 **中断** 说明。

```
loop
    // example of termination condition
    if_nz r0.x
        break
    endif
    ...
endloop
```

此指令必须出现在 loopendloop/ 中，或在 switchendswitch/ 中**出现**的情况。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

breakc (sm4 - asm)

项目 • 2022/09/22

有条件地将执行点移到下一 个 endloop 或 endwitch 之后的指令。

breakc{_z|_nz} src0.select_component

项	说明
src0	[in]要对其测试条件的组件。

注解

令牌格式包含着色器中相应 **尾声** 指令的偏移量，以方便起见。

以下示例显示了**breakc** 指令。

```
loop
    // example of termination condition
    breakc_z  r0.x // break if all bits in r0.x are 0
    breakc_nz r1.x // break if any bit in r1.x is nonzero
    ...
endloop
```

此指令必须出现在 **loopendloop/** 或 **switchendswitch/** 中。

src0 提供的 32 位寄存器在位级别进行测试。 如果任一位为非零位， **breakc_nz** 将执行中断。 如果所有位均为零， **breakc_z** 将执行中断。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

调用 sm4 (- asm)

项目 • 2022/09/24

调用标记为标签 l# 出现在程序中的子例程。

call l#

项	说明
我#	[in]子例程的标签。

备注

遇到 重试 时，请在此调用后将执行返回到指令。

标记格式包含着色器中相应标签的偏移量，作为方便。

以下示例显示了调用说明。

```
...
call l3
...
ret
label l3
    ...
    retc_nz r0.x
    ...
ret
```

限制

- 子例程可以嵌套 32 深。
- 通过实现以透明方式管理返回地址堆栈。
- 如果返回地址堆栈上已有 32 个条目，并且发出 **了调用** ，则会跳过调用。
- 没有自动参数堆栈。 应用程序可以使用可索引的临时寄存器数组 (x#[]) 手动实现堆栈。 但是，子例程调用返回地址不可见，与应用程序完成的任何手动堆栈管理是正交的。
- 不允许 *对 l#* 参数编制索引。
- 不允许递归。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

callc (sm4 - asm)

项目 • 2022/09/22

有条件地调用标记为标签 l# 出现在程序中的子例程。

callc{_z|_nz} src0.select_component，l#

项	说明
src0	[in]要对其测试条件的组件。
我#	[in]子例程的标签。

注解

遇到 重试 时，请在此调用后将执行返回到指令。

标记格式包含着色器中相应标签的偏移量，作为方便。

以下示例显示了调用说明。

```
...
callc_z  r1.y, l3 // if all bits in r0.x are 0, call l3
callc_nz r2.z, l3 // if any bit in r0.x is nonzero, call l3
...
ret
label l3
    ...
    retc_nz r0.x
    ...
ret
```

限制

- 子例程可以嵌套 32 深。
- 通过实现以透明方式管理返回地址堆栈。
- 如果返回地址堆栈上已有 32 个条目，并且发出 **了调用** ，则会跳过调用。
- 没有自动参数堆栈。 应用程序可以使用可索引的临时寄存器数组 (x#[]) 手动实现堆栈。 但是，子例程调用返回地址不可见，与应用程序完成的任何手动堆栈管理是正交的。

- 不允许 *对 l#* 参数编制索引。
- *src0* 提供的 32 位寄存器在位级别进行测试。 如果任一位为非零位， **callc_nz** 将执行调用。 如果所有位均为零， **callc_z** 将执行调用。
- 不允许递归。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

case (sm4 - asm)

项目 • 2022/09/22

开关指令中的标签。

case [32 位即时]

注解

由于仅当没有添加代码时才有效，因此多个**事例** (包括默认) 可以共享相同的代码块。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

continue (sm4 - asm)

项目 • 2022/09/23

在当前循环的开头继续执行。

continue

注解

继续 只能在 循环 或 尾声内使用。

以下示例演示如何使用 **继续** 说明。

```
                loop
                    if_na r0.x
                        break
                    endif
                    if_z r1.x
                        ...
                        continue
                    endif
                    ...
                endloop
```

令牌格式包含着色器中相应循环指令的偏移量，方便使用。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

continuec (sm4 - asm)

项目 • 2022/09/22

有条件地在当前循环的开头继续执行。

continuec{_z|_nz} src0.select_component

术语	说明
src0	[in]要对其测试条件的组件。

注解

continuec 只能在 循环 或 endloop 内使用。

以下示例演示如何使用 **continuec** 指令。

```
loop
    if_na r0.x
        break
    endif
    continuec_z r1.x  // if all bits of r1.x are zero then
                      // continue at beginning of loop.
    ...
    continuec_nz r3.y // if any bit in r3.y is set then
                      // continue at beginning of loop.

    ...
endloop
```

令牌格式包含着色器中相应循环指令的偏移量，方便使用。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

剪切 (sm4 - asm)

项目 • 2022/09/23

如果) 发出任何顶点，则完成当前基元拓扑的几何着色器指令 (，并启动几何着色器声明的类型的新拓扑。

剪切

注解

执行 **剪切** 时，发生的第一件事是，几何着色器调用之前发出的任何拓扑都已完成。 如果为以前的基元拓扑发出了不够的顶点，则会丢弃这些顶点。 由于几何着色器的唯一可用输出拓扑是点列表、折线线和三角形条，因此 **在剪切**时永远不会有任何剩余的顶点。

在上一个拓扑（如果有）完成后， **剪切** 会导致新的拓扑开始，并使用声明为 Geometry Shader 输出的拓扑。

限制

- **剪切**指令仅适用于几何着色器。
- **剪切** 可以在几何着色器中出现任意次数，包括在流控制中。
- 如果几何着色器结束并发出顶点，则生成它们的拓扑已完成，就像执行 **剪切** 作为最后一条指令一样。
- 如果已声明流，则必须使用 cut_stream 而不是 **剪切**。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
	x	

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_constantBuffer (sm4 - asm)

项目 • 2022/09/23

声明着色器常量缓冲区。

dcl_constantBuffer *cbN[size]*， *AccessPattern*

项	说明
cbN[size]	[in]着色器常量缓冲区，其中 N 是一个整数，表示常量-buffer-register 编号和大小是表示缓冲区中元素数的整数。
AccessPattern	[in]缓冲区将由着色器代码访问的方式，这是以下代码之一： **名称** / **说明** immediateIndexed / 使用文本值为缓冲区编制索引。 dynamic_indexed / 使用计算表达式的结果为缓冲区编制索引。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

此示例声明寄存器 cb0 的常量缓冲区，其中包含 19 个元素。 可以使用文本索引访问这些元素。

```
dcl_constantbuffer  cb0[19], immediateIndexed
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_globalFlags (sm4 - asm)

项目 • 2022/09/24

声明着色器全局标志。

dcl_globalFlags *标志*

标志

[in]全局着色器标志。 当前定义了一个标志。

- REFACTORING_ALLOWED - 允许驱动程序对优化进行重新排序算术运算，如下所示。

```
// Original code
a = b*c + b*d + b*e + b*f

// Reordered code
a = b*(c + d + e + f)
// or
a = dot4((b,b,b,b), (c,d,e,f))
```

> ⓘ **备注**
>
> 重新排序算术运算可能会生成不同的结果。

注解

此可选说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_immediateConstantBuffer (sm4 - asm)

项目 • 2022/09/23

声明着色器即时常量缓冲区。

dcl_immediateConstantBuffer ***值 (s)***

value (s)

[in]缓冲区必须至少包含一个值，但不超过 4096 个值。

注解

允许着色器一个即时常量缓冲区。 访问即时常量缓冲区就像使用动态索引的常量缓冲区一样。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_indexableTemp (sm4 - asm)

项目 • 2023/03/09

声明可索引的临时寄存器。

dcl_indexableTemp xN[size]， ComponentCount

项	说明
xN	[in]临时可索引寄存器。 • *N* 是标识寄存器号的整数。 • *[size]* 是可选的整数值。 寄存器数组中的元素数。
ComponentCount	[in]可选的整数值。寄存器数组中的组件数。

寄存器包含足够的空间来容纳 32 位四分量值;临时寄存器数组中的元素数 (可索引和非 索引) 不能超过 4096。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

包含此指令以帮助调试程序集中的着色器;无法使用着色器模型 4 以汇编语言创作着色器。

示例

下面是为可索引寄存器生成的一些代码示例。

```
dcl_indexableTemp x0[23], 2 ; // An indexable array of 23, 2-component, 32-bit elements
dcl_indexableTemp x1[16], 4 ; // An indexable array of 16, 4-component, 32-bit elements
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_indexRange (sm4 - asm)

项目 • 2023/03/09

声明将由索引 (着色器) 中计算的整数访问的寄存器范围。

dcl_indexRange minRegisterM、maxRegisterN

项	说明
minRegisterM	[in]要按索引访问的第一个寄存器。 • *minRegister* 对于顶点或像素着色器输入寄存器为 **v** ，对于顶点着色器输出寄存器为 **o** 。 • *M* 是表示寄存器编号的整数。
maxRegisterN	[in]按索引访问的最后一个寄存器。 与 *minRegister* 的格式相同，但 *N* 是寄存器编号。

以下限制适用于所有寄存器：

- 如果掩码声明），最小寄存器和最大寄存器的类型必须相同，并且具有相同的组件掩码 (。
- 寄存器可以有多个索引范围，只要它们不重叠。
- 最小寄存器号必须小于最大寄存器号。
- 索引寄存器不能包含 系统值。
- 在最大索引声明之外为寄存器编制索引会产生未定义的结果。

像素着色器输入寄存器必须使用相同的内插模式;像素着色器输出寄存器不可编制索引。

几何着色器输入寄存器具有两个维度 (顶点轴、属性轴) ：索引范围仅适用于属性轴，因为顶点轴始终是完全可编制索引的。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

包含此指令以帮助调试程序集中的着色器;无法使用着色器模型 4 以汇编语言创作着色器。

示例

以下是一个示例。

```
dcl_indexRange v1, v3
dcl_indexRange v4, v9
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_input (sm4 - asm)

项目 • 2023/03/09

声明着色器输入寄存器。

dcl_input vN[.mask][， interpolationMode]

项	说明
v*N*	[in]顶点数据寄存器。 • *N* 是标识寄存器号的整数。 • *[.mask]* 是一个可选的组件掩码 (.xyzw) ，用于指定要使用的寄存器组件。
interpolationMode	[in] 可选。 内插模式，仅在像素着色器输入寄存器上采用。 可以为下列值之一： • 常量 - 不要在寄存器值之间内插。 • linear - 在寄存器值之间线性内插。 • linearCentroid - 与线性相同，但在多重采样时夹紧质心。 • linearNoperspective - 与线性相同，但没有透视校正。 • linearNoperspectiveCentroid - 与线性相同，多采样时夹紧质心，无透视校正。

内插说明

默认情况下，执行多重采样抗锯齿时，从像素中心内插顶点属性。 如果未覆盖像素中心，则会在内插之前将属性推断为像素中心。

对于未完全覆盖的像素或未覆盖像素中心的属性，可以指定质心采样，强制采样发生在像素的覆盖区域内的某个位置。 由于样本掩码 (如果在计算质心之前应用) ，则无法选择样本掩码屏蔽的任何样本位置作为质心位置。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

若要将输入标识为系统值，请使用 dcl_input_sv (sm4 - asm) 。

包含此指令以帮助调试程序集中的着色器;不能使用着色器模型 4 以汇编语言创作着色器。

示例

下面是一些示例。

```
dcl_input v3.xyz

dcl_input v0.x, linearCentroid
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_input_sv (sm4 - asm)

项目 • 2023/03/09

声明要求从上一阶段提供 系统值的 着色器输入寄存器。

dcl_input_sv vN[.mask]， systemValueName[， interpolationMode]

项	说明
v*N*	[in]顶点数据寄存器。 • *N* 是标识寄存器号的整数。 • *[.mask]* 是可选的组件掩码 (.xyzw) ，用于指定要使用的寄存器组件。
systemValueName	[in]系统值名称是字符串， (不带"SV_"前缀) 查看 系统值语义 。
interpolationMode	[in] 可选。 影响光栅化期间计算值的内插模式;模式仅由像素着色器使用。 可以为下列值之一： • constant - 不在寄存器值之间内插。 • linear - 在寄存器值之间线性内插。 • linearCentroid - 与线性相同，但在多重采样时夹紧质心。 • linearNoperspective - 与线性相同，但没有透视更正。 • linearNoperspectiveCentroid - 与线性相同，但没有透视校正，在多重采样时夹住质心。

系统值声明的组件掩码可以是 [xyzw] 的任何适当子集;声明不能重叠， (每个声明都必须遵循 xyzw) 序列。 (剪裁距离和剔除距离（例如）) 声明标量系统值时，可以在单个寄存器中声明多个系统值。 如果这样做，请确保其他修饰符（如内插模式）匹配。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

包含此指令以帮助调试程序集中的着色器;无法使用着色器模型 4 以汇编语言创作着色器。

示例

下面是一些示例：

```
```

```
// valid
dcl_input v0.y, linear
dcl_input_sv v0.w, clipDistance
dcl_input_sv v0.z, cullDistance
```

```
// invalid declarations
dcl_input v0.y, linear
dcl_input_sv v0.x, clipDistance  // the y component was previously declared,
this declaration must use
                                 // either the z or w component

dcl_input v0.y, linearNoPerspective                 // the interpolation
mode is linear-no-perspective
dcl_input_sv v0.z, renderTargetArrayIndex, constant  // the interpolation
modes is constant
                                                    // the interpolation
modes must match
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_input vPrim (sm4 - asm)

项目 • 2022/09/22

声明几何着色器使用其标量输入寄存器 vPrim。

dcl_input *vPrim*

项	说明
vPrim	[in]一个 32 位标量，可应用于几何着色器中的每个内部基元。

标量不能应用于任何相邻基元。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
	x	

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_inputPrimitive (sm4 - asm)

项目 • 2022/09/23

声明几何着色器输入的基元类型。

dcl_inputPrimitive *类型*

项	**说明**
类型	[in]输入数据基元类型，如下所示： • *point* - point list • *line* - *行* 列表 • *三角形* - 三角形列表 • *line_adj* - 包含相邻数据的行列表 • *triangle_adj* - 带相邻数据的三角形列表

此说明适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
	x	

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

以下是一个示例。

```
dcl_inputPrimitive triangle
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	**支持**
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_maxOutputVertexCount (sm4 - asm)

项目 • 2022/09/24

声明可由几何着色器输出的最大顶点数。

dcl_maxOutputVertexCount *计数*

项	说明
计数	[in]介于 1 和 n 之间的 32 位无符号整数（含 1）。

几何着色器最多可以输出 1024 32 位值。 此最大值包括输入数据的大小以及着色器创建的数据的大小。

下面是一些限制：

- 如果在几何着色器完成执行之前到达顶点数，着色器将终止。
- 在输出任何顶点之前，几何着色器可以到达其程序的末尾：这是完全合法的。
- 如果要调试几何着色器，可以通过计算生成的发出指令数来判断生成的顶点数。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
	x	

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

下面是一些示例。

假设输入数据由位置 (.xyzw) 和颜色 (.rgb) 组成。 每个组件使用一个字节;给定 7 个字节，着色器可以生成的顶点的最大数目为 1024 / (4 + 3) = 146。

```
dcl_maxOutputVertexCount 146
```

假设几何着色器创建 32 个 4 分量的向量。 着色器可以生成的顶点的最大数目为 1024 / (32 * 4) = 8。

```
dcl_maxOutputVertexCount 8
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_output (sm4 - asm)

项目 • 2023/03/09

声明着色器输出寄存器。

dcl_output oN[.mask]

项	说明
oN	[in]输出数据寄存器; • *N* 是标识寄存器号的整数。 • *[.mask]* 是可选的组件掩码 (.xyzw) ，用于指定要使用的寄存器组件。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

包含此指令以帮助调试程序集中的着色器;无法使用着色器模型 4 以汇编语言创作着色器。

示例

下面是一些示例。

```
dcl_output o0.xyz
dcl_output o1
dcl_output o2.xw
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_output oDepth (sm4 - asm)

项目 • 2022/09/24

声明像素着色器将使用输出深度寄存器。

dcl_output oDepth

如果启用深度比较），则输出深度寄存器中的值在深度比较期间使用 (。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

下面是一些示例。

```
dcl_output oDepth
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_output_sgv (sm4 - asm)

项目 • 2023/03/09

声明包含 系统值 参数的输出寄存器。

dcl_output_sgv oN[.mask]， systemValueName

项	说明
oN	[in]输出数据寄存器; • *N* 是标识寄存器号的整数。 • *[.mask]* 是一个可选的组件掩码 (.xyzw) ，用于指定要使用的寄存器组件。
systemValueName	[in]系统值名称，它是一个字符串， (看到没有"SV_"前缀的系统 值语义) 。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
	x	

包含此指令以帮助调试程序集中的着色器;不能使用着色器模型 4 以汇编语言创作着色器。

示例

以下是一个示例。

```
dcl_output_sgv o4.x, primitiveID
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_output_siv (sm4 - asm)

项目 • 2023/03/09

声明包含 系统值 参数的输出寄存器。

dcl_output_siv oN[.mask]， systemValueName

项	说明
oN	[in]输出数据寄存器; • *N* 是标识寄存器号的整数。 • *[.mask]* 是可选的组件掩码 (.xyzw) ，用于指定要使用的寄存器组件。
systemValueName	[in]系统值名称是字符串， (不带"SV_"前缀) 查看 系统值语义 。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	

包含此指令以帮助调试程序集中的着色器;无法使用着色器模型 4 以汇编语言创作着色器。

示例

下面是一些示例。

```
dcl_output o[0].y
dcl_output_siv o[0].w, clipDistance
dcl_output_siv o[0].z, cullDistance
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_outputTopology (sm4 - asm)

项目 • 2023/03/09

声明基元类型 geometry-shader 输出数据。

dcl_outputTopology *类型*

项	说明
类型	[in]输出基元拓扑，它是以下值之一： • *pointlist* • *linestrip* • *trianglestrip*

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
	x	

包含此指令以帮助调试程序集中的着色器;无法使用着色器模型 4 以汇编语言创作着色器。

示例

以下是一个示例。

```
dcl_outputTopology trianglestrip
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_resource (sm4 - asm)

项目 • 2022/09/23

声明非多重采样着色器输入资源。

dcl_resource *tN*、 *resourceType*、*returnType (s)*

声明多采样着色器输入资源。

dcl_resource *tN* ， *resourceType[size]NN* ， *returnType (s)*

项	说明
tN	[in]纹理寄存器，其中 *N* 是表示寄存器号的整数。
resourceType	[in] 纹理对象 页中列出的任何对象类型。
resourceType[size]NN	[in]纹理 2D 或 Texture2DArray 对象类型 (查看 纹理对象 页) ; *size* 是一个可选整数，表示数组中的元素数; *NN* 是表示多重采样数的整数。
returnType (s)	[in]按组件返回类型，属于下列类型之一：UNORM、SNORM、SINT、UINT 或 FLOAT。 如果所有组件都是同一类型），则返回类型的数量可以高达 1 (，但最多可以四个。

使用 负载在 HLSL 中访问资源;还可以使用任何 HLSL 纹理对象 示例方法访问非多重采样纹理。

如果资源绑定到着色器阶段，则会根据返回类型验证资源格式。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

以下是一个示例。

```
```

```
dcl_resource t3, buffer, UNORM
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_sampler (sm4 - asm)

项目 • 2022/09/22

声明采样器寄存器。

dcl_sampler sN，模式

项	**说明**
sN	[in]采样器寄存器，其中 *N* 是表示寄存器号的整数。
模式	[in]采样器模式，它限制采样器状态 (列在 D3D10_SAMPLER_DESC) 的成员。 下表中列出了模式和状态。

模式	**采样器状态受表彰**
默认值	*筛选器* (不能使用_COMPARISON或_TEXT值) 、 *AddressU/V/W*、 *MinLOD/MaxLOD*、 *MipLODBias*、 *MaxAnisotropy*、 *BorderColor[4]*
比较	*Filter* ， *CompareFunction* ， *AddressU/V/W* ， *MinLOD* ，*MaxLOD* ， *MipLODBias* ， *MaxAnisotropy* ， *BorderColor[4]*
单	*筛选器* (必须是) 、 *MonoFilterWidth*、 *MonoFilterHeight* (这两种状态的_TEXT值之一是全局设备状态) 、 *MinLOD*、 *MipLODBias*、 *MaxAnisotropy*

该模式限制可以使用的示例说明;此表列出了每个模式支持的纹理对象方法。

在此模式下运行的采样器	**可以使用这些Texture-Object方法**
默认值	Sample、 SampleLevel、 SampleGrad
比较	SampleCmp、 SampleCmpLevelZero
单	SampleLevel

此说明适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
x	x	x*

* - 仅在像素着色器中支持在单声道模式下使用采样器。

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

以下是一个示例。

```
dcl_sampler s3, default
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dcl_temps (sm4 - asm)

项目 • 2022/09/23

声明临时寄存器。

dcl_temps N

项	说明
N	[in]临时寄存器数。

每个寄存器都有一个 32 位四分量值的空间。 临时和 可索引临时 寄存器的总数必须小于或等于 4096。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

此指令包含在程序集中帮助调试着色器;不能使用着色器模型 4 以程序集语言创作着色器。

示例

以下是一个示例。

```
dcl_temps 10; // Declare r0-r9
```

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

default (sm4 - asm)

项目 • 2022/09/24

switch 语句中的可选标签。

默认值

注解

此指令在 C 中的工作方式与 **默认值** 一样。仅当没有添加代码时，才有效，因此，(包括 **默认**) 可以共享同一代码块的多个事例。

开关构造中只允许使用一个**默认**语句。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

deriv_rtx (sm4 - asm)

项目 • 2022/09/22

对于 RenderTarget x 方向 (rtx) 或 RenderTarget y 方向，*src0* (帖子 重排) 的每个 float32 组件的内容更改率。

deriv_rtx[_sat] dest[.mask]， [-]src0[_abs][.swizzle]，

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

每个 2x2 像素标记只计算一个 x，y 派生配对。

此操作依赖于硬件。

三角形的参考光栅器实现：

- 像素着色器始终在锁步 (中超过 2x2 像素的着色器，甚至通过流控制运行着色器，屏蔽禁用的像素) 。
- 象限始终具有偶数像素坐标，（x 和 y）（ 对于左上角像素 ）。
- 如果基元太小而无法填充 2x2 象限，则虚拟像素会耗尽基元。
- **首先** 选择 2 像素来计算deriv_rtx：当前像素和其他像素与四边形中具有相同 y 坐标的另一个像素。 然后，结果计算为： *src0* (奇数 x 像素) - *src0* (偶 x 像素) [每分量]
- 首先 选择 2 像素计算deriv_rty：当前像素和其他像素与象限中具有相同 x 坐标的另一个像素。 然后，结果计算为： *src0* (奇数 y 像素) - *src0* (甚至 y 像素) [每分量]

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

deriv_rty (sm4 - asm)

项目 • 2022/09/23

呈现目标 y 等效于 deriv_rtx。

deriv_rty[_sat] dest[.mask]， [-]src0[_abs][.swizzle]，

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

每 2x2 个像素戳只计算一个 x，y 派生配对。

此操作依赖于硬件。

三角形的参考光栅器实现：

- 像素着色器始终在锁步 (以 2x2 像素为单位运行着色器，即使通过流控制、屏蔽禁用的像素) 。
- 象限始终具有偶数像素坐标， (x 和 y) 的左上角像素。
- 如果基元太小而无法填充 2x2 象限，则虚拟像素会耗尽基元。
- deriv_rtx 首先选择 2 个像素，即当前像素，另一个像素与象限中的 y 坐标相同。 然后，结果计算为： *src0* (奇数 x 像素) - *src0* (偶 x 像素) [每分量]
- **deriv_rty** 首先选择 2 个像素来计算：当前像素和具有同一 x 坐标的另一个像素从四边形。 然后，结果计算为： *src0* (奇数 y 像素) - *src0* (甚至 y 像素) [每组件]

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

放弃 (sm4 - asm)

项目 • 2022/09/22

在到达程序末尾时，将有条件地标记要丢弃的像素着色器的结果。

discard{_z|_nz} src0.select_component

项	说明
src0	[in]确定是否放弃正在处理的当前像素的值。

注解

此指令将当前像素标记为已终止，同时继续执行，以便如有必要，其他并行执行的像素可能会获取派生体。 即使执行仍在继续，所有像素着色器输出也会在 **放弃** 指令之前或之后写入。

对于 **discard_z**，如果 *src0.select_component* 中的所有位均为零，则丢弃像素。

对于 **discard_nz**，如果 *src0.select_component* 中的任何位都是非零位，则丢弃像素。

此外， **放弃** 指令可以存在于任何流控制构造中。

着色器中可能存在多个 **放弃** 指令，如果执行任何指令，则像素将终止。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

div (sm4 - asm)

项目 • 2023/03/09

按组件划分。

div[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in] 操作的结果。
src0	[输入] 被除数。
src1	[输入] 除数。

备注

下表显示了使用各种数字类执行指令时获得的结果，假定不会发生溢出或下溢。

应注意两个允许的除法实现：a/b 和* (1/b) 。

这一结果之一是下表中存在较大分母值 (大于 8.5070592e+37) 的例外情况，其中 1/分母是一个 denorm。 由于实现可能以* (1/b) 的形式执行除法，而不是直接执行 a/b，并且 1/[大值] 是可能被刷新的反数，因此表中的一些情况会产生不同的结果。 例如，(+/-) INF / (+/-) [value > 8.5070592e+37] 可能会在某些实现上生成 NaN，但在其他实现上 (+/-) INF

在此表中，F 表示有限实数。

**src0 src1 -> **	**-Inf**	**-F**	**-denorm**	**-0**	**+0**	**+denorm**	**+F**	**+inf**	**南**
-Inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	NaN	NaN
-F	-inf	-F	src0	src0	src0	src0	+-F 或 +-0	+inf	NaN
-denorm	-inf	src1	-0	-0	+0	+0	src1	+inf	NaN
-0	-inf	src1	-0	-0	+0	+0	src1	+inf	NaN
+0	-inf	src1	+0	+0	+0	+0	src1	+inf	NaN
+denorm	-inf	src1	+0	+0	+0	+0	src1	+inf	NaN

src0 src1 ->	-Inf	-F	-denorm	-0	+0	+denorm	+F	+inf	南
+F	-inf	+-F 或 +-0	src0	src0	src0	src0	+ F	+inf	NaN
+inf	NaN	+inf	+inf	+inf	+inf	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dp2 (sm4 - asm)

项目 • 2022/09/22

组件 rg、POS-重排的二维向量点积。

dp2[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 *dest = src0.r * src1.r + src0.g * src1.g*
src0	[in]操作中的组件。
src1	[in]操作中的组件。

注解

将标量结果复制到写入掩码中的组件。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dp3 (sm4 - asm)

项目 • 2022/09/22

组件的三维向量点积 rgb、POS-重排。

dp3[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]，

项	说明
dest	[in]操作的结果。 *dest = src0.r * src1.r + src0.g * src1.g + src0.b * src1.b*
src0	[in]操作中的组件。
src1	[in]操作中的组件。

注解

标量结果复制到写入掩码中的组件。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

dp4 (sm4 - asm)

项目 • 2022/09/24

组件 rgba、POS-重排的四维矢量点积。

dp4[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]，

项	说明
dest	[in]操作的结果。 *dest = src0.r * src1.r + src0.g * src1.g + src0.b * src1.b+ src0.a * src1.a*
src0	[in]操作中的组件。
src1	[in]操作中的组件。

注解

标量结果复制到写入掩码中的组件。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

else (sm4 - asm)

项目 • 2022/09/23

启动 **其他** 块。

else

注解

令牌格式包含着色器中相应 尾部 指令的偏移量，方便使用。

以下示例演示如何使用 **其他** 说明。

syntax

```
if     // any of the various forms of if* statements
   ...
else   // (optional)
   ...
endif
```

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

发出 (sm4 - asm)

项目 • 2022/09/22

发出顶点。

发出

注解

发出 会导致所有声明的 o# 寄存器读出 Geometry Shader 以生成顶点。

发出多个 **发出** 调用时，将生成基元。

发出 可以在 Geometry Shader 中出现任意次数，包括在流控制中。

如果已声明流，则必须使用 emit_stream。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
	x	

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

emitThenCut (sm4 - asm)

项目 • 2022/09/22

等效于 发出 命令，后跟 剪切 命令。

emitThenCut

备注

当有意将最后一个顶点输出到拓扑中时，此命令非常有用。

如果已声明流，则必须使用 emitthencut_stream。

此说明适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
	x	

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	**支持**
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

endif (sm4 - asm)

项目 • 2022/09/24

结束 if 语句。

endif

注解

以下示例演示如何使用 endif 指令。

syntax

```
if     // any of the various forms of if* statements
   ...
else   // (optional)
   ...
endif
```

标记格式包含在着色器 **中的指令作为** 方便时对应的偏移量。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

endloop (sm4 - asm)

项目 • 2022/09/24

结束循环语句。

endloop

注解

以下示例演示如何使用此指令。

syntax

```
loop
    // example of termination condition
    if_nz r0.x
        break
    endif
    ...
endloop
```

令牌格式包含着色器中相应循环指令的偏移量，方便使用。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

endswitch (sm4 - asm)

项目 • 2022/09/23

结束 switch 语句。

endswitch

注解

令牌格式包含着色器中相应开关指令的偏移量，以方便起见。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

eq (sm4 - asm)

项目 • 2022/09/23

分量向量浮点相等比较。

eq dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要理解 *src1* 的组件。
src1	[in]要昏迷于 *src0* 的组件。

备注

对每个组件执行浮点比较 (*src0src1* ==)，并将结果写入 *dest*。

如果比较为 true，则为该组件返回0xFFFFFFFF。 否则返回0x0000000。

在比较 (原始源寄存器未触摸) 之前刷新 Denorms。 +0 等于 -0。 与 NaN 的比较返回 false。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

exp (sm4 - asm)

项目 • 2023/05/05

组件 2exponent。

exp[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in] 操作的结果。 *dest* = 2 ^ *src0*
src0	[in]指数。

备注

此指令遵循限制理论。 最大相对误差为 2-21。

下表显示了使用各种数字类执行指令时获得的结果，假定不会发生溢出或下溢。 在此表中，F 表示有限实数。

src	-Inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	0	+ F	1	1	1	1	+ F	+inf	NaN

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

frc (sm4 - asm)

项目 • 2022/09/23

按组件方式提取小数部分。

frc[_sat] dest[.mask] ， [-] src0[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 *dest* = *src0* - round_ni *(src0)*
src0	[in]操作中的组件。

注解

下表显示了使用各种数字类执行指令时获取的结果。

src	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	NaN	[+0 到 1)	+0	+0	+0	+0	[+0 到 1)	NaN	NaN

F 表示有限实数。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ftoi (sm4 - asm)

项目 • 2022/09/23

浮点到有符号整数转换。

ftoi dest[.mask]， [-]src0[_abs][.swizzle]

项目	说明
dest	[in]操作结果的地址。 *dest* = round_z *(src0)*
src0	[in]要转换的组件。

注解

每个组件执行转换。 在从浮点转换为 int 的 C 约定后，始终向零执行舍入。需要不同舍入语义的应用程序可以在转换为整数之前调用 **舍入** 指令。

输入固定到范围 [-2147483648.999f...在转换之前为 2147483647.999f]，输入 NaN 值生成零结果。

可选反值和绝对值修饰符在转换前应用于源值。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ftou (sm4 - asm)

项目 • 2022/09/22

浮点到无符号整数转换。

ftou dest[.mask]， [-]src0[_abs][.swizzle]

ftoi dest[.mask]， [-]src0[_abs][.swizzle]

项	描述
dest	[in]操作结果的地址。
src0	[in]要转换的值。

注解

按组件执行转换。 在从浮点转换为 int 的 C 约定之后，始终向零执行舍入。

需要不同舍入语义的应用程序可以在转换为整数之前调用 **舍入** 指令。

输入固定到范围 [0.0f...转换前为 4294967295.999f]，输入 NaN 值生成零结果。

可选反值和绝对值修饰符在转换之前应用于源值。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

gather4 (sm4.1 - asm)

项目 • 2023/03/09

收集将在双线性筛选操作中使用的四个纹素，并将其打包到单个寄存器中。

gather4[_aoffimmi (u，v)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]， srcSampler.r

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]包含纹理坐标。
srcResource	[in]资源寄存器。 重排允许在将返回的值写入 *dest* 之前任意重排。
srcSampler	[in]采样器寄存器。 此参数必须具有 .r (红色) 重排，这指示将 R 通道的值复制到 *dest*。

备注

此操作仅适用于单通道 2D 或 CubeMap 纹理。 对于 2D 纹理，仅使用采样器的寻址模式，并使用任何 mip 棱锥图的顶层。

此指令的行为与 示例 指令类似，但不会生成筛选的样本。 有助于筛选的四个样本按逆时针顺序放置在 xyzw 中，从样本开始到查询位置的左下角。 这与以下位置 (u，v) 纹理坐标增量的点采样相同：(-、+) 、 (+、+) 、 (+、-) 、 (-,-) ，其中增量的幅度始终为纹素的半。

对于 CubeMap 纹理，当双线性足迹跨越相邻面的边缘纹素时，使用。 角使用与 **示例** 指令相同的规则;即不为人知道的角被视为三个指角的平均值。

适用于 **示例** 指令的纹理格式限制也适用于 **gather4** 指令。

对于硬件实现，传统双线性筛选中的优化（直接检测纹素上的样本并跳过读取权重为 0 的纹素）无法与 **gather4** 一起使用。 **gather4** 始终返回所有请求的纹素。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ge (sm4 - asm)

项目 • 2022/09/22

比或等于比较的分量向量浮点。

ge dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1 进行比较的*组件。
src1	[in]要与 *src0 进行比较的*组件。

注解

为每个组件执行浮点比较 (*src0*>= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则为该组件返回0xFFFFFFFF。 否则返回0x0000000。

在比较之前刷新 Denorms，原始源寄存器保持不变。 +0 等于 -0。 与 NaN 的比较返回 false。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

iadd (sm4 - asm)

项目 • 2022/09/22

整数加法。

iadd dest[.mask]， [-]src0[.swizzle]， [-]src1[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要添加到 *src1* 的数字。
src1	[in]要添加到 *src0* 的数字。

备注

分量添加 32 位操作数 *src0* 和 *src1*，将正确的 32 位结果置于 *dest* 中。 不会执行超出每个组件的 32 位值的携带或借用，因此此指令不区分其操作数的有符号性。

源操作数上的可选否定修饰符在执行操作之前需要 2 的补充。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ieq (sm4 - asm)

项目 • 2023/03/09

分量向量整数相等比较。

ieq dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

备注

此指令对每个组件执行整数比较 (*src0* == *src1*) ，并将结果写入 *dest*。

如果比较结果为 true，则会为该组件返回0xFFFFFFFF。 否则返回0x0000000

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

如果 (sm4 - asm)

项目 • 2022/09/23

基于逻辑 OR 结果的分支。

if{_z|_nz} src0.select_component

项	说明
src0	[in]包含要对其测试条件的组件。

注解

标记格式包含着色器中相应 endif 指令的偏移量，作为方便。

以下示例演示如何使用此指令。

syntax

```
                if_z r0.x // if all bits in r0.x are zero
                   ...
                else // (optional)
                   ...
                endif
                if_nz r1.x // if any bit in r0.x is nonzero
                   ...
                else // (optional)
                   ...
                endif
```

限制

- 如果 4 个组件向量) 必须使用单个组件选择器，则源操作数 (。
- *src0* 提供的 32 位寄存器在位级别进行测试。 如果任一位为非零位， **则if_z** 为 true。 如果所有位均为零， **则if_nz** 为 true。
- Flow控制块可以嵌套每个子例程 (和主) 最多 64 个深度。 HLSL 编译器不会生成超出此限制的子例程。 未定义每个子例程) 超过 64 个级别的控制流指令 (的行为。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ige (sm4 - asm)

项目 • 2022/09/22

分量向量整数大于或相等比较。

ige dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作的结果。
src0	[in]要与 *src1* 进行比较的组件。
src1	[in]要与 *src0* 进行比较的组件。

注解

为每个组件执行整数比较 (*src0* >= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则返回该组件的0xFFFFFFFF。 否则返回0x0000000。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ilt (sm4 - asm)

项目 • 2022/09/22

分量向量整数小于比较。

ilt dest[.mask]， src0[.swizzle]， src1[.swizzle]

项目	说明
dest	操作的结果。
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

备注

对每个组件执行整数比较 (*src0*<*src1*)，并将结果写入 *dest*。

如果比较为 true，则返回该组件的0xFFFFFFFF。 否则返回0x0000000。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

imad (sm4 - asm)

项目 • 2023/03/09

带符号整数乘法和加法。

imad dest[.mask]， [-]src0[.swizzle]， [-]src1[.swizzle]， [-]src2[.swizzle]

项	说明
dest	[in] 操作的结果。
src0	[in]要与 *src1* 相乘的值。
src1	[in]要与 *src0* 相乘的值。
src2	[in]要添加到 *src0* 和 *src1* 的乘积的值。

备注

32 位操作数 *src0* 和 *src1* (有符号) ，使结果的每个组件保持低 32 位 () ，然后添加*src2*，生成每个分量的正确低 32 位 () 结果。 32 位结果置于 *dest* 中。

源操作数上的可选反门修饰符在执行算术运算之前采用 2 的补码。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

imax (sm4 - asm)

项目 • 2022/09/22

组件型整数最大值。

imax dest[.mask]， [-]src0[.swizzle]， [-]src1[.swizzle]，

项	说明
dest	[in]操作的结果。 *dest = src0>src1 ？ src0 ： src1*
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

注解

源操作数上的可选否定修饰符在执行操作之前需要 2 的补充。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

imin (sm4 - asm)

项目 • 2022/09/23

最小分量整数。

imin dest[.mask]， [-]src0[.swizzle]， [-]src1[.swizzle]，

项	说明
dest	[in]操作的结果。 *dest = src0<src1 ? src0 : src1*
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

注解

源操作数上的可选否定修饰符在执行操作之前需要 2 的补充。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

imul (sm4 - asm)

项目 • 2022/09/23

带符号整数相乘。

imul destHI[.mask]， destLO[.mask]， [-]src0[.swizzle]， [-]src1[.swizzle]

项目	说明
destHI	[in]结果的 32 位高地址。
destLO	[in]结果的低 32 位的地址。
src0	[in]要与 *src1* 相乘的值。
src1	[in]要与 *src0* 相乘的值。

备注

32 位操作数 *src0* 和 *src1* (的分量乘以分量) 进行签名，为每个组件生成正确的完整 64 位 (,) 结果。 每个组件) 的低 32 位 (放置在 *destLO* 中。 每个组件 (高 32 位) 放置在 *destHI* 中。

如果不需要高或低 32 位的 64 位结果，则 *destHI* 或 *destLO* 可以指定为 NULL，而不是指定寄存器。

在执行算术运算之前，源操作数上的可选反门修饰符采用 2 的补充。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ine (sm4 - asm)

项目 • 2022/09/24

不相等的分量向量整数比较。

ine dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]包含要与 *src1* 进行比较的值。
src1	[in]包含要与 *src0* 进行比较的值。

注解

对每个组件执行整数比较 (*src0* ！= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则为该组件返回0xFFFFFFFF。 否则返回0x0000000

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ineg (sm4 - asm)

项目 • 2023/03/09

2 的补充。

ineg dest[.mask]， src0[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]包含操作的值。

备注

此指令对 *src0* 中每个 32 位值执行组件 2 的补充。 32 位结果存储在 *dest* 中。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ishl (sm4 - asm)

项目 • 2023/03/09

左移。

ishl dest[.mask]， src0[.swizzle]， src1.select_component

项	说明
dest	[in]操作结果的地址。
src0	[in]包含要移动的值。
src1	[in]包含移位量。

备注

此指令执行 *src0* 中每个 32 位值（由 LSB 5 位提供的无符号整数位计数 (0-31 范围）*src1.select_component*，并插入 0）。 每个分量 32 位的结果放置在 *dest* 中。 计数是应用于所有组件的标量值。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ishr (sm4 - asm)

项目 • 2022/09/22

算术右移 (符号扩展) 。

ishr dest[.mask]， src0[.swizzle]， src1.select_component

项	说明
dest	[in]包含操作的结果。
src0	[in]包含要移动的值。
src1	[in]包含班次金额。

注解

此指令通过 LSB 5 位提供的无符号整数位计数 (0-31 范围) 在 *src1.select_component* 中执行 *src0* 中每个 32 位值的分量算术移位，从而复制位 31 的值。 每个组件结果的 32 位放置在 *dest* 中。 计数是应用于所有组件的标量值。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

itof (sm4 - asm)

项目 • 2022/09/24

带符号整数到浮点转换。

itof dest[.mask]， [-]src0[.swizzle]

项	说明
dest	[in]包含操作的结果。
src0	[in]包含要转换的值。

注解

此带符号整数到浮点转换指令假定 *src0* 包含有符号 32 位整数 4 元组。 执行指令后，*dest* 将包含浮点 4 元组。

按组件执行转换。

如果整数输入值太大，无法完全以浮点格式表示，则强烈建议舍入到最接近的偶数模式，但不是必需的。

源操作数上的可选反门修饰符在执行算术运算之前采用 2 的补数。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

label (sm4 - asm)

项目 • 2022/09/23

指示子例程的开头。

label l#

项	说明
我#	[in]标签号。

注解

标签只能在任何流控制语句中未嵌套的**重试**指令之后直接显示。

程序中第一个 **标签** 之前的代码是主程序。 所有子例程都显示在程序末尾，由 **标签** 语句指示。

以下示例演示如何使用此指令。

syntax

```
...
call l3
...
ret
label l3
    ...
    if_nz r0.x
        ret
    endif
    ...
ret
```

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ld (sm4 - asm)

项目 • 2022/09/23

从指定的缓冲区或纹理中提取数据，而无需任何筛选 (，例如使用提供的整数地址) 点采样。 源数据可能来自除 TextureCube 以外的任何资源类型。

ld[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]

项目	说明
dest	[in]操作结果的地址。
srcAddress	[in]执行样本所需的纹理坐标。
srcResource	[in]纹理寄存器 (t#) ，必须声明标识要从中提取的纹理或缓冲区。

注解

此指令是 示例 指令的简化替代方法。 与 **示例**不同， **ld** 还能够从缓冲区提取数据。 **ld** 还可以从仅) 像素着色器上的多样本资源中提取 (。

srcAddress 提供以无符号整数形式执行样本所需的纹理坐标集。 如果 *srcAddress* 超出维度 -1)] 中的范围[0... (#texels]，则会调用超出边界的行为，其中 **ld** 以 *srcResource* 格式的所有非缺失组件返回 0，以及缺失组件的默认值。 希望对范围外地址行为进行更灵活的控制的应用程序应改用 **示例** 指令，因为它遵循定义为采样器状态的地址包装/镜像/固定/边框行为。

srcAddress.a (POS-重排) 始终提供无符号整数 mipmap 级别。 如果值超出 resource-1)] 中的范围 [0... (num miples]) ，则会调用超出边界的行为。 如果资源是一个缓冲区，该缓冲区不能有任何 mipmap，则忽略 *srcAddress.a*

对于缓冲区 (和非数组) ，*srcAddress.gb* (POS-重排) 将被忽略。 对于 texture1D 数组和 texture2D，*srcAddress.b* (POS-重排) 将被忽略。

对于 texture1D 数组， *srcAddress.g* (POS-重排) 以无符号整数的形式提供数组索引。 如果值超出可用数组索引的范围 [0... (数组大小-1)]，则会调用超出边界的行为。

对于 texture2D 数组， *srcAddress.b* (POS-重排) 提供数组索引，否则具有与 texture1D 相同的语义。

从 t# 提取，该 T # 没有任何限制，它为所有组件返回 0。

地址偏移量

可选的 [_aoffimmi (u，v，w)] 后缀 (按即时整数偏移) 指示 **ld** 的纹理坐标将由一组提供的即时纹素空间整数常量值偏移。 文本值是一组 4 位 2 的补数，具有整数范围 [-8，7]。 此修饰符仅针对 texture1D/2D/3D 定义，包括数组，而不是用于缓冲区。

偏移量将添加到纹理坐标（以纹素空间表示）相对于 **ld** 访问的错位。

地址偏移量不会沿 texture1D/2D 数组的数组轴应用。

纹理 1D 将忽略 *_aoffimmi v，w* 组件。

纹理 2D 将忽略 *_aoffimmi w* 组件。

由于 **ld** 的纹理坐标是无符号整数，因此如果偏移量导致地址低于零，则它将包装到大型地址，并导致超出边界的访问。

返回类型控件

ld 返回到目标寄存器的数据格式与**示例**说明中所述的方式相同：它基于绑定到 *srcResource* 参数的格式 (*t#*) 。

与 **示例** 指令一样， **ld** 的返回值为 4 向量，对于格式中不存在的组件，具有特定于格式的默认值。 *srcResource* 上的重排决定如何重排从纹理加载返回的 4 分量结果，之后 *dest* 上的 .mask 确定 *dest* 中的哪些组件得到更新。

当 *ld* 将 32 位浮点值读取到 32 位寄存器中时，这些位将保持不变：也就是说，非规范值保持非规范性。 这与 **示例** 指令不同。

其他详细信息

由于没有与 **ld** 指令关联的筛选，因此 LOD 偏差等概念不适用于 **ld**。 因此没有采样器 *s#* 参数。

限制

- *srcResource* 必须是 t# 寄存器，而不是 TextureCube。 *srcResource* 不能是常量缓冲区，不能绑定到 t# 寄存器。
- 不允许 *srcResource* 上的相对寻址。
- *srcAddress* 必须是 temp (r#/x#) 、常量 (cb#) 或输入 (v#) 寄存器。
- *dest* 必须是临时 (r#/x#) 或输出 (o*#) 寄存器。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ld2dms (sm4.1 - asm)

项目 • 2023/03/09

从二维多样本纹理中读取单个样本。

ld2dms[_aoffimmi (u，v)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]，sampleIndex

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]执行示例所需的纹理坐标。
srcResource	[in]纹理寄存器 (t#) ，它必须已声明用于标识要从中提取的纹理或缓冲区
sampleIndex	[in]将样本标识为从 *srcResource* 读取 (标量操作数) 。

备注

此指令是 示例 指令的简化替代方法。 它从指定的纹理中提取数据，而无需任何筛选 (例如使用提供的整数 *srcAddress* 和 *sampleIndex* 的点采样) 。

srcAddress 以无符号整数的形式提供执行示例所需的纹理坐标集。 如果 *srcAddress* 在维度 -1)] 中没有[0... (#texels]范围， **则 ld2dms** 在资源格式存在的所有组件中始终返回 0，对于缺失的组件，默认值 (0，0，0，1.0f/0x00000001) 。

sampleIndex 不一定是文本。 无需在纹理资源上指定多样本计数，它适用于深度视图或模具视图。

希望对超出范围地址行为进行更灵活的控制的应用程序应改用 **示例** 指令，因为它遵循定义为采样器状态的地址包装/镜像/固定/边框行为。

对于 Texture2D，将忽略 *srcAddress.b* (后重排) 。 如果值在可用数组索引 [0... (数组 size-1)] 的范围内， **则 ld2dms** 在资源格式存在的所有组件中始终返回 0，对于缺失的组件，默认值为 (0，0，0，1.0f/0x00000001) 。

对于 Texture2D 数组， *srcAddress.b* (后重排) 提供数组索引。 Oherwise 它的行为与 Texture2D 相同。

srcAddress.a (后重排) 始终被忽略。 HLSL 编译器永远不会在那里输出任何内容。

srcResource 是一个纹理寄存器 (t#) ，它必须已声明 (22.3.11) ，标识要从中提取的纹理。

从没有绑定到它的 t# 提取将返回所有组件的 0。

地址偏移量

可选的 [_aoffimmi (u，v，w)] 后缀 (地址偏移量为即时整数) 指示 **ld2dm 的** 纹理坐标将偏移一组提供的即时纹素空间整数常量值。 文本值是一组 4 位 2 的补数，具有整数范围 [-8，7]。

偏移量将添加到纹理坐标（纹素空间中）。

地址偏移不沿 Texture1D/2D 数组的数组轴应用。

对于 Texture1D，将忽略 *_aoffimmi v，w* 组件。

对于 Texture2D，将忽略 *_aoffimmi w* 组件。

由于 **ld2dms** 的纹理坐标是无符号整数，因此如果偏移量导致地址低于零，它将换行到一个大地址，并导致超出边界的访问，这就像 **ld** 在资源格式存在的所有组件中返回 0，而缺失组件的默认值 (0，0，0，1.0f/0x00000001) 。

示例编号

ld2dms 可用于任何资源。 **ld2dms** 的运行方式与 **ld** 相同，但 2D multsample 资源除外，方法是使用其他基于 0 的 () *sampleIndex* 操作数来标识要从资源读取哪个样本。

指定超过资源中样本数的 *sampleIndex* 的结果是未定义的，但不能返回设备上下文的地址空间之外的数据。

返回类型控件

ld2dms 返回的目标寄存器的数据格式与**示例**指令所述的相同方式确定。 它基于绑定到 *srcResource* 参数的格式 (t#) 。

与 **示例** 指令一样， **ld2dms** 的返回值是 4 向量，对于格式中不存在的组件，具有特定于格式的默认值。 *srcResource* 上的重排确定如何重排从纹理加载返回的 4 分量结果，然后 *dest* 上的 .mask 将确定*更新 dest* 中的哪些组件。

当 **ld2dms** 将 32 位浮点值读取到 32 位寄存器中时，位不会受到影响;也就是说，非正态值将保持非正态。 这与 **示例** 指令不同。

其他详细信息

由于没有与此指令关联的筛选，因此 LOD 偏差等概念不适用。 因此，没有 *采样器 s#* 参数。

限制

- *srcResource* 必须是 t# 寄存器，而不是 TextureCube、Texture1D 或 Texture1DArray。 *srcResource* 不能是 ConstantBuffer，它不能绑定到 t# 寄存器。
- 不允许在 *srcResource* 上进行相对寻址。
- *srcAddress* 和 *sampleIndex* 必须是临时 (r#/x#) 、常量 (cb#) 或输入 (v#) 寄存器。
- *dest* 必须是临时 (r#/x#) 或输出 (o*#) 寄存器。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

lod (sm4.1 - asm)

项目 • 2022/09/23

返回用于纹理筛选的详细信息级别 (LOD) 。

lod dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]， srcSampler

项	说明
dest	[in]结果的地址。
srcAddress	[in]一组纹理坐标。
srcResource	[in]纹理寄存器。
srcSampler	[in]采样器寄存器。

备注

这的行为类似于 示例 指令，但不会生成筛选的示例。 该指令计算以下矢量 (ClampedLOD、NonClampedLOD、0、0) 。 NonClampedLOD 是一个计算的 LOD 值，它忽略采样器或纹理 (的任何固定：它可以返回负值。) ClampedLOD 是实际 **示例** 指令将使用的计算 LOD 值。 *srcResource* 上的重排允许在将返回的值写入目标之前任意重排。

如果没有绑定到指定槽的资源，则返回 0。

如果采样器使用异性筛选，LOD 应根据椭圆占用的较小轴对应于小数级 mip 级别。

这适用于以下纹理类型：Texture1D、Texture2D、Texture3D 和 TextureCube。

当与指定点 mip 筛选的采样器一起使用时，不会定义 **lod** 指令，具体来说，任何以 MIP_POINT 结尾的D3D10_FILTER枚举。 (此示例为 D3D10_FILTER_MIN_MAG_MIP_POINT.)

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

log (sm4 - asm)

项目 • 2023/03/09

组件级日志基数 2。

log[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 dest = log2 (src0)
src0	[in]操作的值。

备注

限制

- 遵循限制理论。
- 容错：如果 *src0* 为 [0.5..2]，则绝对错误必须不超过 2-21。 如果 *src0* (0..0.5) 或 (2..+INF]，则相对误差必须不超过 2-21。

下表显示了使用各种数字类执行指令时获得的结果，假定不会发生溢出或下溢。 F 表示有限实数。

src	**-Inf**	**-F**	**-denorm**	**-0**	**+0**	**+denorm**	**+F**	**+inf**	**NaN**
dest	NaN	NaN	-inf	-inf	-inf	-inf	F	+inf	NaN

此指令适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	**支持**

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

loop (sm4 - asm)

项目 • 2022/09/23

指定循环，循环访问，直到遇到中断指令。

loop

注解

循环 可以无限期循环访问，尽管执行某些指令后，着色器的总体执行可能会被迫终止。

Flow控制块可以嵌套每个子例程和 main 最多 64 个深度。 HLSL 编译器不会生成超出此限制的子例程。 未定义每个子例程的 64 个级别以上控制流指令的行为。

令牌格式包含着色器中相应 尾声 指令的偏移量，以方便起见。

以下示例演示如何使用循环指令。

syntax

```
loop
    // example of termination condition
    if_nz r0.x
        break
    endif
    ...
endloop
```

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

lt (sm4 - asm)

项目 • 2022/09/22

分量向量浮点小于比较。

lt dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

注解

此指令对每个组件执行浮点比较 (*src0*<*src1*)，并将结果写入 *dest*。

如果比较为 true，则返回该组件的0xFFFFFFFF。 否则返回0x0000000

比较前刷新 Denorms;原始源寄存器不受触摸。

+0 等于 -0。

与 NaN 的比较返回 false。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

mad (sm4 - asm)

项目 • 2023/03/09

分量乘法 & 加法。

mad[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]， [-]src2[_abs][.swizzle]

项	说明
dest	[in] 操作的结果。 *dest = src0 * src1 + src2*
src0	[in]乘法。
src1	[in]乘数。
src2	[in]附录。

备注

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

max (sm4 - asm)

项目 • 2022/09/22

分量浮点数最大值。

max[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]，

项	说明
dest	[in]操作的结果。 *dest = src0>= src1 ？ src0 ： src1*
src0	[in]要与 *src1* 进行比较的组件。
src1	[in]要与 *src0* 进行比较的组件。

注解

> 使用 =，而不是 > 这样，如果最小 (x，y) = x，则最大 (x，y) = y。

NaN 具有特殊处理。 如果一个源操作数为 NaN，则返回另一个源操作数，并按组件进行选择。 如果两者均为 NaN，则返回任何 NaN 表示形式。

在比较之前，将刷新 Denorms 并保留符号。 但是，写入 *到 dest* 的结果可能或可能不会刷新非序。

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。 F 表示有限实数。

src0 src1->	**-inf**	**F**	**+inf**	**NaN**
-inf	-inf	src1	+inf	-inf
F	src0	src0 或 src1	+inf	src0
+inf	+inf	+inf	+inf	+inf
NaN	-inf	src1	+inf	NaN

此说明适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

min (sm4 - asm)

项目 • 2022/09/23

组件型浮点数最小值。

min[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]，

项	说明
dest	[in]操作的结果。 *dest = src0<src1 ？ src0 ： src1*
src0	[in]要与 *src1 进行比较的*组件。
src1	[in]要与 *src0 进行比较的*组件。

注解

> 使用 = 而不是 > 这样，如果最小 (x，y) = x，则最大 (x，y) = y。

NaN 具有特殊处理。 如果一个源操作数为 NaN，则返回另一个源操作数，并按组件进行选择。 如果两者均为 NaN，则返回任何 NaN 表示形式。 这符合新的 IEEE 754R 规则。

在比较之前，会刷新 Denorms，并保留符号。 但是，写入 *到 dest* 的结果可能或可能不会刷新非序数。

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。 F 表示有限实数。

src0 src1->	**-inf**	**F**	**+inf**	**NaN**
-inf	-inf	-inf	-inf	-inf
F	-inf	src0 或 src1	src0	src0
-inf	-inf	src1	+inf	+inf
NaN	-inf	src1	+inf	NaN

此说明适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

mov (sm4 - asm)

项目 • 2023/03/09

按组件移动。

mov[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 *dest = src0*
src0	[in]要移动的组件。

备注

除重排外，修饰符假定数据为浮点。 缺少修饰符只是移动数据而不更改位。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

movc (sm4 - asm)

项目 • 2022/09/24

按组件方式的条件移动。

movc[_sat] dest[.mask]， src0[.swizzle]， [-]src1[_abs][.swizzle]， [-]src2[_abs][.swizzle]，

项	说明
dest	[in]操作结果的地址。 如果 *src0*，则 *destsrc1* = else *destsrc2* =
src0	[in]要测试条件的组件。
src1	[in]要移动的组件。
src2	[in]要移动的组件。

注解

以下示例演示如何使用此指令。

```
syntax
```

```
                for each component in dest[.mask]
                    if the corresponding component in src0 (POS-swizzle)
                       has any bit set
                    {
                        copy this component (POS-swizzle) from src1 into
dest
                    }
                    else
                    {
                        copy this component (POS-swizzle) from src2 into
dest
                    }
                endfor
```

src1 和 *src2* 上的修饰符（除重排外）假定数据是浮点。 缺少修饰符只会移动数据而不更改位。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

mul (sm4 - asm)

项目 • 2022/09/24

按分量乘法。

mul[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作的结果。 dest = src0 * src1
src0	[in]乘法。
src1	[in]乘数。

备注

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。

F 表示有限实数。

src0 src1 ->	**-inf**	**-F**	**-1.0**	**-denorm**	**-0**	**+0**	**denorm**	**+1.0**	**+F**	**+inf**	**NaN**
-inf	+inf	+inf	+inf	NaN	NaN	NaN	NaN	-inf	-inf	-inf	NaN
-F	+inf	+ F	-src0	+0	+0	-0	-0	src0	-F	-inf	NaN
-1	+inf	-src1	+1.0	+0	+0	-0	-0	-1.0	-src1	-inf	NaN
-denorm	NaN	+0	+0	+0	+0	-0	-0	-0	-0	NaN	NaN
-0	NaN	+0	+0	+0	+0	-0	-0	-0	-0	NaN	NaN
+0	iNaN	-0	-0	-0	-0	+0	+0	+0	+0	NaN	NaN
+denorm	NaN	-0	-0	-0	-0	+0	+0	+0	+0	NaN	NaN
+1.0	-inf	src1	-1.0	-0	-0	+0	+0	+1.0	src1	+inf	NaN
+F	-inf	-F	-src0	-0	-0	+0	+0	src0	+ F	+inf	NaN
+inf	-inf	-inf	-inf	NaN	NaN	NaN	NaN	+inf	+inf	+inf	NaN

src0 src1 ->	-inf	-F	-1.0	-denorm	-0	+0	denorm	+1.0	+F	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ne (sm4 - asm)

项目 • 2022/09/23

分量向量浮点不相等比较。

ne dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作的结果。
src0	[in]要与 *src1* 进行比较的组件。
src1	[in]要与 *src0* 进行比较的组件。

注解

此指令对每个组件执行浮点比较 (*src0* ！= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则返回该组件的0xFFFFFFFF。 否则返回0x0000000。

在与原始源寄存器进行比较之前刷新 Denorms。

+0 等于 -0。

与 NaN 的比较返回 true。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

nop (sm4 - asm)

项目 • 2022/09/23

不执行任何操作。

nop

注解

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

不 (sm4 - asm)

项目 • 2022/09/22

按位不。

not dest[.mask] ， src0[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]原始组件。

注解

此指令对 *src0* 中每个 32 位值的补充执行分量。 32 位结果存储在 *dest* 中。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

或 (sm4 - asm)

项目 • 2022/09/22

按位或。

或 dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作的结果。
src0	[in]与 *src1* 配合使用的 OR 组件。
src1	[in]使用 *src0* 进行 OR 的组件。

注解

此指令对 *src0* 和 *src1* 中 32 位值的每个配对执行分量逻辑 OR。 32 位结果放置在 *dest* 中。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

(sm4 - asm)

项目 • 2022/09/24

查询给定输入资源的维度。

frameworkfo[_uint|_rcpFloat] dest[.mask]， srcMipLevel.select_component，srcResource[.swizzle]

项	说明
dest	[in]操作结果的地址。
srcMipLevel	[in]mip 级别。
srcResource	[in]要查询维度的 t# 或 u# 输入纹理。

注解

srcMipLevel 读取为无符号整数标量，因此源寄存器需要单个组件选择器（如果不是标量即时值）。

dest 接收由写入掩码选择的 [width、height、depth 或数组大小、total-mip-count]。。

返回的宽度、高度和深度值适用于 *srcMipLevel* 参数选择的 mip 级别值，并且与纹素数据大小无关，以纹素数为单位。 对于多采样资源 (texture2D[Array]MS#)，宽度和高度也会在纹素中返回，而不是样本。

dest.w 中返回的总 mip 计数不受 *srcMipLevel* 参数影响。

对于 UAV (u#)，mip 级别数始终为 1。

此指令的所有方面都基于在 t#/u# 上绑定的资源视图的特征，而不是基础基础资源。

返回的值都是浮点，除非使用_uint修饰符，在这种情况下，返回的值都是整数。 如果使用_rcpFloat修饰符，则返回的所有值都是浮点，并且宽度、高度和深度将作为相互 (1.0f/width、1.0f/height、1.0f/depth) 返回，包括 INF（如果宽度/高度/深度从超出范围的 *srcMipLevel* 行为为 0）。 _rcpFloat修饰符仅适用于宽度、高度和深度返回的值，不适用于设置为 0 的值，因此不返回，也不适用于数组大小返回。

srcResource 上的重排允许在将返回的值写入目标之前任意重排。

如果 *srcResource* 是 Texture1D，则在 *dest.x* 中返回宽度， *dest.yz* 设置为 0。

如果 *srcResource* 是 Texture1DArray，则宽度以 *dest.x* 返回，数组大小以 *dest.y* 返回，*dest.z* 设置为 0。

如果 *srcResource* 是 Texture2D，则在 *dest.xy* 中返回宽度和高度， *并且 dest.z* 设置为 0。

如果 *srcResource* 是 Texture2DArray，则在 *dest.xy* 中返回宽度和高度，并且数组大小在 *dest.z* 中返回。

如果 *srcResource* 是 Texture3D，则以 *dest.xyz* 返回宽度、高度和深度。

如果 *srcResource* 是 TextureCube，则在 *dest.xy* 中返回单个多维数据集人脸尺寸的宽度和高度， *dest.z* 设置为 0。

如果 *srcResource* 是 TextureCubeArray，则在 *dest.xy* 中返回单个多维数据集人脸尺寸的宽度和高度。 *dest.z* 设置为未定义值。

如果在 *srcResource* 上指定了每资源 mip 夹，则无论夹子如何，都始终返回视图中的 mipmap 总数。 但是，如果给定的误位的维度由脂化物请求，并且 miplevel 已被固定（，例如 2.2 的夹子意味着 mips 0 和 1 已被固定），则返回的维度是未定义的。 当误位超出范围时，某些实现将返回为塑料fo 指定的边界外行为。 其他实现将返回 mip 的维度，就像它尚未固定一样。

限制

- *srcResource* 必须是不是缓冲区的 t# 或 u# 寄存器，而是纹理*。
- 不允许 *对 srcResource* 进行相对寻址。
- *如果 srcMipLevel* 不是标量，则必须使用单个组件选择器。
- 从没有绑定到它的 t# 或 u# 提取返回宽度、高度、深度或数组大小和总 mip 计数的 0。 在这种情况下，仍会遵循_rcpFloat修饰符，从而返回适用的返回值的 INF。
- 如果 *srcMipLevel* 超出资源中可用数量的错误范围，则大小返回的行为 (*dest.xyz)* 与未绑定 t# 或 u# 资源的行为相同。 对于这种情况，仍以 *dest.w* 形式返回总 mip 计数。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ret (sm4 - asm)

项目 • 2022/09/23

Return 语句。

Ret

注解

如果在子例程内，请在调用后返回到指令。 如果未在子例程内，请终止程序执行。

以下示例演示如何使用此指令。

syntax

```
...
 call l3
 ...
 ret
 label l3
     ...
 ret
```

限制

- **ret** 可以在程序中的任意位置显示，多次。
- 如果 标签 指令出现在着色器中，则它必须前面有未嵌套在任何流控制语句中的 **ret** 命令。
- 如果着色器中有子例程，着色器中的最后一条指令必须是 **重试**。

此说明适用于以下着色器阶段：

顶点着色器	**几何着色器**	**像素着色器**
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

retc (sm4 - asm)

项目 • 2022/09/24

条件返回。

retc{_z|_nz} src0.select_component

项	说明
src0	[in]要测试条件的寄存器。

注解

如果在子例程中，此指令在调用后有条件地返回到指令。 如果不是子例程内部，则此指令将终止程序执行。

以下示例演示如何使用此指令。

syntax

```
            ...
            call l3
            ...
            ret
            label l3
                ...
                retc_nz r0.x  // If any bit in r0.x is nonzero, then return
                retc_z  r1.x  // If all bits in r0.x are zero, then return.
                ...
            ret
```

限制

- **retc** 可以在程序中的任意位置显示任意次数。
- 主程序或子例程中的最后一个指令不能是 **retc_z** 或 **retc_nz**。 相反，可以使用无条件 重试 。
- *src0* 提供的 32 位寄存器在位级别进行测试。 如果任一位为非零位， **ret_nz** 将返回。 如果所有位均为零， **retc_z** 将返回。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

round_ne (sm4 - asm)

项目 • 2022/09/23

浮点圆到整型浮点。

round_ne[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

此指令执行 *src0* 中值的分量浮点舍入，并将整型浮点值写入 *dest*。 **round_ne** 舍入到最接近的偶数。

下表显示了使用各种数字类执行指令时获取的结果。

F 表示有限实数。

src	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	-inf	-F	-0	-0	+0	+0	+ F	+inf	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

round_ni (sm4 - asm)

项目 • 2022/09/22

浮点圆到整型浮点。

round_ni[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	描述
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

此指令执行 *src0* 中值的分量浮点舍入，将整型浮点值写入 *dest*。 **round_ni** 向 -infinity 舍入，通常称为 floor () 。

下表显示了使用各种数字类执行指令时获取的结果。

F 表示有限实数。

src	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	-inf	-F	-0	-0	+0	+0	+ F	+inf	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

round_pi (sm4 - asm)

项目 • 2022/09/24

浮点圆到整型浮点。

round_pi[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

此指令执行 *src0* 中值的分量浮点舍入，并将整型浮点值写入 *dest*。

round_pi 向 +infinity 舍入，通常称为 ceil () 。

下表显示了使用各种数字类执行指令时获取的结果。

F 表示有限实数。

src	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	-inf	-F	-0	-0	+0	+0	+ F	+inf	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

round_z (sm4 - asm)

项目 • 2022/09/22

浮点圆到整型浮点。

round_z[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

此指令执行 *src0* 中值的分量浮点舍入，将整型浮点值写入 *dest*。

round_z 向零舍入。

下表显示了使用各种数字类执行指令时获取的结果。

src	**-inf**	**-F**	**-denorm**	**-0**	**+0**	**+denorm**	**+F**	**+inf**	**NaN**
dest	-inf	-F	-0	-0	+0	+0	+ F	+inf	NaN

F 表示有限实数。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是

着色器模型	支持
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

rsq (sm4 - asm)

项目 • 2022/09/23

按分量对等方根。

rsq[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]包含操作的结果。 *dest* = 1.0f / sqrt (*src0*)
src0	[in]操作的组件。

注解

最大相对误差为 2-21。

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。

F 表示有限实数。

src	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	NaN	NaN	-inf	-inf	+inf	+inf	+ F	+0	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

示例 (sm4 - asm)

项目 • 2022/09/22

使用指定的地址和给定采样器标识的筛选模式对指定元素/纹理中的数据采样。

sample[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]，srcSampler

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。 有关详细信息，请参阅**备注**部分。
srcResource	[in]纹理寄存器。 有关详细信息，请参阅**备注**部分。
srcSampler	[in]采样器寄存器。 有关详细信息，请参阅**备注**部分。

注解

源数据可能来自除缓冲区以外的任何资源类型。

srcAddress 提供执行样本所需的纹理坐标集，因为浮点值引用纹理中的规范化空间。 地址包装模式 (包装/镜像/固定/边框等等) 应用于 [0...1] 范围之外的纹理坐标，取自采样器状态 (s#) ，并在任何地址偏移量应用于纹理坐标后应用。

srcResource 是一个纹理寄存器， (t#) 。 这只是纹理的占位符，包括正在采样的资源的返回数据类型。 所有这些信息都在着色器前言中声明。 要采样的实际资源绑定到 t#) 的槽 # (外部着色器。

srcSampler 是一个采样器寄存器， (s) 。 这只是筛选控件集合的占位符，例如点与线性控件、误用控件和地址包装控件。

硬件执行采样所需的信息集拆分为两个正交部分。 首先，纹理寄存器提供源数据类型信息，例如，有关纹理是否包含 SRGB 数据的信息。 它还引用正在采样的实际内存。 其次，采样器寄存器定义要应用的筛选模式。

数组资源

对于 Texture1D 数组， *srcAddress* g 组件 (POS-重排) 选择要从中提取的数组切片。 这始终被视为缩放浮点值，而不是标准纹理坐标的规范化空间，对值应用圆到近的偶数，然后固定到可用的 BufferArray 范围。

对于 Texture2D 数组， *srcAddress* b 组件 (POS 重排) 选择要从中提取的数组切片，否则使用 Texture1D 数组描述的相同语义。

地址偏移量

可选的 [_aoffimmi (u，v，w)] 后缀 (按即时整数偏移) 指示样本的纹理坐标将由一组提供的即时纹素空间整数常量值偏移。 文本值是一组 4 位 2 的补数，具有整数范围 [-8，7]。 此修饰符针对所有资源（包括 Texture1D/2D 数组和 Texture3D）定义，但未为 TextureCube 定义。

硬件可以利用即时知识，即通过一组示例指令执行有关常见位置的纹素的遍历。 这可以使用 _aoffimmi (u、v、w) 进行传达。

相对于要访问的每个错位，偏移量将添加到纹理坐标（以纹素空间为单位）。 因此，即使纹理坐标作为规范化浮点值提供，偏移量也会应用纹素空间整数偏移量。

地址偏移量不会沿 Texture1D/2D 数组的数组轴应用。

对于 Texture1D，_aoffimmi v，w 组件将被忽略。

对于 Texture2D，_aoffimmi w 组件将被忽略。

在将任何地址偏移应用于纹理坐标后， (s#) 从采样器状态 (包装/镜像/固定/边框等) 地址包装模式。

返回类型控件

示例返回到目标寄存器的数据格式由绑定到 *srcResource* 参数 (t#) 的资源格式 (DXGI_FORMAT*) 确定。 例如，如果指定的 t# 与格式为 DXGI_FORMAT_A8B8G8R8_UNORM_SRGB的资源绑定，则采样操作会将采样纹素从 gamma 2.0 转换为 1.0，应用筛选，并将结果写入目标寄存器作为 [0..1] 范围内的浮点值。

返回的值是 4 向量 (，对于格式) 中不存在的组件，具有特定于格式的默认值。 *srcResource* 上的重排决定如何重排从纹理示例/筛选器返回的 4 分量结果，之后 *dest* 上的 .mask 确定 *dest* 中的哪些组件得到更新。

当样本将 32 位浮点值读取到 32 位寄存器中时，点采样 (没有筛选) ，则它可能或可能不会刷新非规范值，但数字未修改。 如果点采样非规范值的不确定性是应用程序的一个问题，请使用 ld 指令，这可以保证 32 位浮点值未修改。

LOD 计算

有关在确定 LOD 筛选过程中如何计算派生体的详细信息，请参阅 deriv_rtx 和 deriv_rty。 **示例**指令使用**派生**着色器指令使用的同一定义隐式计算纹理坐标上的派生值。 这不适用于 sample_l 或 sample_d 说明。 对于这些说明，LOD 或衍生产品由应用程序直接提供。

对于 **示例** 说明，实现可以选择在 2x2 戳 (中跨所有 4 像素共享相同的 LOD 计算，但没有更大的区域) ，或执行每像素 LOD 计算。

其他详细信息

对于 Buffer & Texture1D，将忽略 *srcAddress* .gba 组件 (POS-重排) 。 对于 Texture1D 数组，将忽略 *srcAddress* .ba 组件 (POS-重排) 。 对于 Texture2D， *将忽略 srcAddress* .a 组件 (POS-重排) 。

从没有任何绑定的输入槽提取为所有组件返回 0。

限制

- *srcResource* 必须是 t# 寄存器。 *srcResource* 不能是 ConstantBuffer，不能绑定到 t# 寄存器。
- *srcSampler* 必须是 s# 寄存器。
- 不允许在 *srcResource* 或 *srcSampler* 上进行相对寻址。
- *srcAddress* 必须是 temp (r#/x#) 、constantBuffer (cb#) 、输入 (v#) 寄存器或即时值 (s) 。
- *dest* 必须是临时 (r#/x#) 或输出 (o*#) 寄存器。
- textureCubes 不允许使用 _aoffimmi (u、v、w) 。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sample_b (sm4 - asm)

项目 • 2022/09/22

使用指定的地址和给定采样器标识的筛选模式从指定的元素/纹理中采样数据。

sample_b[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]，srcResource[.swizzle]， srcSampler， srcLODBias.select_component

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。 有关详细信息，请参阅 示例 说明。
srcResource	[in]纹理寄存器。 有关详细信息，请参阅 **示例** 说明。
srcSampler	[in]采样器寄存器。 有关详细信息，请参阅 **示例** 说明。
srcLODBias	[in]有关此参数的信息，请参阅 “**备注** ”部分。

注解

源数据可能来自除缓冲区以外的任何资源类型。 额外的偏差应用于在指令执行过程中计算的详细信息级别。

此指令的行为与 示例 指令类似，在选择 mip 映射 (s) 之前，将指定的 *srcLODBias* 值应用于作为指令执行的一部分计算的详细值级别。 *srcLODBias* 值将按像素添加到计算的 LOD，以及采样器 MipLODBias 值，然后再将固定到 MinLOD 和 MaxLOD。

限制

- **sample_b** 继承与 示例 指令相同的限制，以及其附加参数的其他限制。
- *srcLODBias* 的范围 (-16.0f 到 15.99f) ;超出此范围的值将生成未定义的结果。
- *如果 srcLODBias* 不是标量，则必须使用单个组件选择器。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sample_c (sm4 - asm)

项目 • 2023/03/09

执行比较筛选器。

sample_c[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]， srcResource.r，srcSampler， srcReferenceValue

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。 有关详细信息，请参阅 示例 说明。
srcResource	[in]纹理寄存器。 有关详细信息，请参阅 **示例** 说明。 必须为 .r 重排。
srcSampler	[in]采样器寄存器。 有关详细信息，请参阅 **示例** 说明。
srcReferenceValue	[in]选中了单个组件的寄存器，用于比较。

备注

本指令的主要目的是为Percentage-Closer深度筛选提供构建基块。 **sample_c** 中的"c"表示"比较"。

sample_c的操作数与示例指令相同，只是还有一个额外的 float32 源操作数 *srcReferenceValue*，该操作数必须是选择单组件寄存器或标量文本。

srcResource 参数必须具有 .r (红色) 重排。 **sample_c** 只对红色分量运行，并返回单个值。 *srcResource* 上的 .r 重排指示标量结果将复制到所有组件。

当深度缓冲区设置为输入纹理时，深度值将显示在红色分量中。

如果此指令与不是 Texture1D/2D/2DArray/Cube/CubeArray 的资源一起使用，则会生成未定义的结果。

执行此指令时，采样硬件会根据 *srcAddress* 中提供的坐标，使用当前采样器的 ComparisonFunction 将 *srcReferenceValue* 与每个筛选器"点击"位置 (纹素) 源资源的红色组件值进行比较。

在 *srcReferenceValue* 已量化为纹理格式的精度之后，将进行比较，其方式与在输出合并可见性测试中的深度比较之前将 z 量化为深度缓冲区精度完全相同。 这包括一个固定到格式范围 (例如 [0..1] 的 UNORM 格式) 。

将源纹素的 Red 分量与量化 *的 srcReferenceValue* 进行比较。 对于从资源掉落的纹素，红色分量值是通过应用地址模式 (和 BorderColorR（如果处于边框模式）确定的，) 采样器。 如果纹理格式为浮点，则比较遵循所有 D3D11 浮点比较规则。

每个传递的比较返回 1.0f 作为纹素的 Red 分量值，每个失败的比较返回 0.0f 作为纹理的 Red 值。 然后，筛选完全按照采样器状态的指定进行，仅在红色组件中运行，并将单个标量筛选器结果返回给着色器，并复制到所有掩码 *的 dest* 组件。

sample_c的使用与所有其他常规用途筛选控件是正交的。 **sample_c** 与其他常规用途筛选器模式无缝工作。 **sample_c** 更改常规用途筛选器的行为，使筛选的值由于比较结果而全部变为 1.0f 或 0.0f。

从未绑定任何内容的输入槽提取将为所有组件返回 0。

有关详细信息，请参阅 示例 说明。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sample_c_lz (sm4 - asm)

项目 • 2023/03/09

执行比较筛选器。 此指令的行为类似于 sample_c，但 LOD 为 0，而导数将被忽略。

sample_c_lz[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]， srcResource.r， srcSampler， srcReferenceValue

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。 有关详细信息，请参阅 示例 说明。
srcResource	[in]纹理寄存器。 有关详细信息，请参阅 **示例** 说明。 必须为 .r 重排。
srcSampler	[in]采样器寄存器。 有关详细信息，请参阅 **示例** 说明。
srcReferenceValue	[in]选中了单个组件的寄存器，用于比较。

备注

“lz”表示零级。 由于忽略导数，因此此指令在像素着色器以外的着色器中可用。

如果此指令与误贴纹理一起使用，则 LOD 0 将采样，除非采样器具有 LOD 夹（将 LOD 放在其他位置），或者存在 LOD 偏差，这只会从 0 开始偏置。 由于忽略导数，各向异性筛选的行为与各向同性筛选一样。

在像素着色器中，当纹理坐标派生在着色器中时，可以在不同的流控件中使用此指令，这与 sample_c不同。

从未绑定任何内容的输入槽提取将为所有组件返回 0。

为了保持一致性，此说明适用于所有着色器，而不仅仅是像素着色器。

顶点着色器	几何着色器	像素着色器
X	X	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sample_d (sm4 - asm)

项目 • 2023/03/09

使用指定的地址和给定采样器标识的筛选模式从指定的元素/纹理中采样数据。

sample_d[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]，srcResource[.swizzle]， srcSampler， srcXDerivatives[.swizzle]， srcYDerivatives[.swizzle]

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。 有关详细信息，请参阅 示例 说明。
srcResource	[in]纹理寄存器。 有关详细信息，请参阅 **示例** 说明。
srcSampler	[in]采样器寄存器。 有关详细信息，请参阅 **示例** 说明。
srcXDerivatives	[in]X 方向上源地址的导数。 有关详细信息，请参阅**备注**部分。
srcYDerivatives	[in]y 方向的源地址的导数。 有关详细信息，请参阅**备注**部分。

注解

此指令的行为类似于 示例 指令，但 x 方向和 y 方向的源地址的导数分别由额外的参数 *srcXDerivatives* 和 *srcYDerivatives* 提供。 这些导数位于规范化的纹理坐标空间中。

srcXDerivatives 的 r、g 和 b 组件 (POS-swizzle) 提供 du/dx、dv/dx 和 dw/dx。 将忽略 POS 重排) ("a"组件。

srcYDerivatives (POS-swizzle) 的 r、g 和 b 组件提供 du/dy、dv/dy 和 dw/dy。 将忽略 POS 重排) ("a"组件。

与允许跨 2x2 标记共享单个 LOD 计算 **的示例** 指令 **不同，sample_d** 在像素着色器中使用时，必须完全独立地按像素计算 LOD。

如果 **sample_d** 的导数输入来自像素着色器中的导数计算指令，并且值包括 INF/NaN，**则sample_d** 的行为可能与隐式计算导数 **的示例** 指令不匹配。 INF/NaN 值对 LOD 计算的影响可能不同。

从未绑定任何内容的输入槽提取将为所有组件返回 0。

限制

- **sample_d** 继承与 **示例** 指令相同的限制，以及下面对其附加参数的额外限制。
- *srcXDerivatives* 和 *srcYDerivatives* 必须为 temp (r#/x#) 、constantBuffer (cb#) 、输入 (v#) 寄存器或即时值 (s) 。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
X	X	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sample_l (sm4 - asm)

项目 • 2022/09/22

使用指定的地址和给定采样器标识的筛选模式从指定的元素/纹理中采样数据。

sample_l[_aoffimmi (u，v，w)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]， srcSampler， srcLOD.select_component

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。 有关详细信息，请参阅 示例 说明。
srcResource	[in]纹理寄存器。 有关详细信息，请参阅 **示例** 说明。
srcSampler	[in]采样器寄存器。 有关详细信息，请参阅 **示例** 说明。
srcLOD	[in]LOD。

注解

此指令与 示例相同，但 LOD 直接由应用程序作为标量值提供，表示无异性。 此说明在所有可支持着色器阶段都可用。

sample_l 使用 *srcLOD* 将纹理采样为 LOD。 如果 LOD 值为 <0，则选择第 (个最大地图) ，并应用了放大筛选器（如果根据筛选模式) (适用）。 由于 *srcLOD* 是浮点值，因此如果缩小筛选器是 LINEAR 或具有异性筛选，则小数值用于在两个 mip 级别之间内插。

sample_l 忽略地址派生体，因此筛选行为纯粹是异性。 由于忽略派生体，异性筛选的行为为异性筛选。

采样器状态 MIPLODBIAS 和 MAX/MINMIPLEVEL 受到尊重。

在像素着色器中使用时， **sample_l** 意味着 LOD 的选择是每像素，而相邻像素没有影响，例如在同一 2x2 戳中。

从没有绑定到输入槽的输入槽中为所有组件返回 0。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
X	X	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sampleinfo (sm4.1 - asm)

项目 • 2023/03/09

在给定着色器资源视图或光栅器中查询样本数。

sampleinfo[_uint] dest[.mask]， srcResource[.swizzle]

项	说明
dest	[in]操作结果的地址。
srcResource	[in]着色器资源。

备注

此指令返回给定资源或光栅器的样本数。 它仅对可以使用 ld2dms 加载的资源有效，除非光栅器指定为 *srcResource*。 *srcResource* 可以是 t# 寄存器 (着色器资源视图) 或光栅器寄存器。

指令计算向量 (SampleCount，0，0，0) 。

srcResource 上的重排允许在返回的值写入目标之前任意重排。 返回的值是浮点，除非使用_uint修饰符，在这种情况下，返回的值是整数。 如果没有资源绑定到指定槽，则返回0。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
X	X	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

samplepos (sm4.1 - asm)

项目 • 2023/03/09

查询样本在给定着色器资源视图或光栅器中的位置。

samplepos dest[.mask]， srcResource[.swizzle]， sampleIndex

项	说明
dest	[in]操作结果的地址。
srcResource	[in]着色器资源。
sampleIndex	[in]样本的索引 (标量操作数) 。

备注

此指令返回给定资源的 sample *sampleIndex* 的 2D 样本位置。 它仅对可以使用 ld2dms 加载的资源有效，除非将光栅器指定为 *srcResource*。

srcResource 可以是 t# 寄存器 (着色器资源视图) 或光栅器寄存器。

指令计算浮点矢量 (Xposition、Yposition、0、0) 。

srcResource 上的重排允许在将返回的值写入目标之前任意重排。 样本位置相对于像素的中心，基于像素坐标系。

如果 *sampleIndex* 超出边界，则返回零向量。 如果没有绑定到指定槽的资源，则返回 0。

samplepos 可用于着色器代码中的自定义解析等操作。

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
		x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sincos (sm4 - asm)

项目 • 2022/09/24

分量 sin (theta) 和 cos (theta) 弧度。

sincos[_sat] destSIN[.mask]， destCOS[.mask]， [-]src0[_abs][.swizzle]

项	说明
destSIN	[in]sin (*src0*) 的地址，每个组件计算。
destCOS	[in]cos (*src0*) 的地址，每个组件计算。
src0	[in]要计算 sin 和 cos 的组件。

注解

如果不需要结果，可以将 *destSIN* 和 *destCOS* 指定为 NULL，而不是指定寄存器。

该值可以是任何 IEEE 32 位浮点值。

从 -100*Pi 到 +100*Pi 的间隔中，最大绝对误差为 0.0008。

下表显示了使用各种数字类执行指令时获取的结果。

F 表示有限实数。

src	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
destSIN	NaN	[-1 到 +1]	-0	-0	+0	+0	[-1 到 +1]	NaN	NaN
destCOS	NaN	[-1 到 +1]	+1	+1	+1	+1	[-1 到 +1]	NaN	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

sqrt (sm4 - asm)

项目 • 2022/09/22

分量方根。

sqrt[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]操作的结果。 *dest* = sqrt (*src0*)
src0	[in]要为其获取平方根的组件。

注解

精度为 1 ulp。

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。

F 表示有限实数。

	-inf	-F	-denorm	-0	+0	+denorm	+F	+inf	NaN
dest	NaN	NaN	-0	-0	+0	+0	+ F	+inf	NaN

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

switch (sm4 - asm)

项目 • 2023/03/09

根据选择器的值，将控制权转移到开关正文中的不同语句块。

switch src0.select_component

项	说明
src0	[in]switch 语句的选择器。

备注

switch/endswitch 构造的行为与 C 语言中的 **switch** 构造完全相同，但以下例外情况如下：对于 D3D11 case/默认语句，如果默认语句**不**/带中断，则不会包含任何代码。 允许多个 **case** 语句（包括 **默认值**）按顺序显示，共享相同的代码块。

条件必须是 32 位寄存器组件或即时数量。 相等比较是按位 (整数) 。

与 D3D11 中的任何着色器指令一样，硬件可能直接实现开关构造，也可能不实现 **开关** 构造。

Switch 语句可以嵌套。 每个 **开关** 块根据每个子例程和 main 的流控制嵌套深度限制（64）计为 1 级，与 **事例** 语句的数量无关。 HLSL 编译器不会生成超过此限制的子例程。 未定义每个子例程超过 64 个级别的控制流指令的行为。

以下示例演示如何使用此命令。

syntax

```
...
switch r0.x
default: // falling through
case 3
    switch r1.x
    case 4
        ...
        break
    case 5
        ...
        break
    endswitch
    break
case 0
```

```
                    break
                endswitch
```

此指令适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

udiv (sm4 - asm)

项目 • 2022/09/23

无符号整数除法。

udiv destQUOT[.mask]， destREM[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
destQUOT	[in]生成的商的地址。
destREM	[in]生成的余数的地址。
src0	[in]要除以 *src1* 的组件。
src1	[in]按 whch 划分 *src0* 的组件。

注解

此指令执行 32 位操作数 *src0* 的分量无符号除以 32 位操作数 *src1*。 除数的结果是放置在 *destQUOT* 中的 32 位商和放置在 *destREM* 中的 32 位余数。

除以零表示商和余数0xffffffff。

如果需要商或余数，则可以将 *destQUOT* 或 *destREM* 指定为 NULL，而不是指定寄存器。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

uge (sm4 - asm)

项目 • 2022/09/22

分量向量无符号整数大于或等于比较。

uge dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1 进行比较的*组件。
src1	[in]要与 *src0 进行比较的*组件。

注解

此指令对每个组件执行无符号整数比较 (*src0*>= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则为该组件返回0xFFFFFFFF。 否则返回0x0000000。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ult (sm4 - asm)

项目 • 2022/09/23

组件向量无符号整数小于比较。

ult dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src1* 进行比较的值。

注解

对每个组件执行无符号整数比较 (*src0src1*<) ，并将结果写入 *dest*。

如果比较为 true，则此指令返回该组件的0xFFFFFFFF。 否则，它将返回0x0000000。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

umad (sm4 - asm)

项目 • 2022/09/22

无符号整数相乘并添加。

umad dest[.mask]， src0[.swizzle]， src1[.swizzle]， src2[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1* 相乘的值。
src1	[in]要与*src1* 相乘的值。
src2	[in]要添加到 *src0* 和 *src1* 的产品的值。

注解

32 位操作数 *src0* 和 *src1* 无符号的组件级 umul，使结果的每个组件保持低 32 位。 然后，此指令执行 *src2* 的 iadd，生成每个组件的正确低 32 位（，）结果。 32 位结果放置在 *dest* 中。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

umax (sm4 - asm)

项目 • 2022/09/23

分量无符号整数最大值。

umax dest[.mask]， src0[.swizzle]， src1[.swizzle]，

项	说明
dest	[in]操作结果的地址。 *dest = src0>src1 ? src0 : src1*
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

备注

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

umin (sm4 - asm)

项目 • 2022/09/23

最小分量无符号整数。

umin dest[.mask]， src0[.swizzle]， src1[.swizzle]，

项	说明
dest	[in]操作结果的地址。 *dest = src0<src1 ? src0 : src1*
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

注解

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

umul (sm4 - asm)

项目 • 2022/09/22

无符号整数相乘。

umul destHI[.mask]， destLO[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
destHI	[in]每个组件的 32 位结果的高 32 位。
destLO	[in]每个组件的结果的低 32 位。
src0	[in]要乘 *以 src1* 的组件。
src1	[in]要乘 *以 src0* 的组件。

注解

此指令执行无符号 32 位操作数 *src0* 和 *src1* 的分量乘法，为每个组件生成正确的完整 64 位结果。 每个组件的低 32 位放置在 *destLO* 中。 每个组件的高 32 位放置在 *destHI* 中。

如果不需要 64 位结果的高或低 32 位，则可以将 *destHI* 或 *destLO* 指定为 NULL，而不是指定寄存器。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

ushr (sm4 - asm)

项目 • 2022/09/22

向右移动。

ushr dest[.mask]， src0[.swizzle]， src1.select_component

项	说明
dest	[in]操作结果的地址。
src0	[in]要移动的组件。
src1	[in]转移 *src0* 的量。

注解

此指令通过 LSB 5 位提供的无符号整数位计数（（*src1.select_component* 中 0-31 范围））执行 *src0* 中每个 32 位值的分量移，插入 0。 每个组件结果 32 位放置在 *dest* 中。 计数是应用于所有组件的标量值。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

utof (sm4 - asm)

项目 • 2022/09/22

无符号整数到浮点转换。

utof dest[.mask]， src0[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要转换的组件。

注解

src0 必须包含无符号 32 位整数 4 元组。 执行指令后， *dest* 将包含浮点 4 元组。 按组件执行转换。

当整数输入值太大，无法完全以浮点格式表示时，舍入到最接近的偶数模式。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

xor (sm4 - asm)

项目 • 2022/09/24

按位 xor。

xor dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]操作的结果。
src0	[in]使用 *src1* 将组件连接到 XOR。
src1	[in]使用 *src0* 将组件连接到 XOR。

注解

此指令对 *src0* 和 *src1* 中 32 位值的每个配对执行组件逻辑 XOR。 32 位结果放置在 *dest* 中。

此说明适用于以下着色器阶段：

顶点着色器	几何着色器	像素着色器
x	x	x

最小着色器模型

以下着色器模型中支持此函数。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	是
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 4 程序集 (DirectX HLSL)

着色器模型 5 程序集

项目 • 2023/03/09

本部分介绍着色器模型 5 支持的说明。

指令修饰符
add
and
atomic_and
atomic_cmp_store
atomic_iadd
atomic_imax
atomic_imin
atomic_or
atomic_umax
atomic_umin
atomic_xor
Bfi
bfrev
break
breakc
bufinfo
call
callc
case
continue
continuec
countbits
削减
cut_stream
爸爸
dcl_constantBuffer
dcl_function_body
dcl_function_table
dcl_globalFlags
dcl_hs_fork_phase_instance_count
dcl_hs_join_phase_instance_count
dcl_hs_max_tessfactor
dcl_immediateConstantBuffer
dcl_indexableTemp

dcl_indexRange
dcl_input
dcl_input vForkInstanceID
dcl_input vGSInstanceID
dcl_input vJoinInstanceID
dcl_input vOutputControlPointID
dcl_input vPrim
dcl_input vThread
dcl_input_control_point_count
dcl_input_sv
dcl_inputPrimitive
dcl_interface
dcl_interface_dynamicindexed
dcl_maxOutputVertexCount
dcl_output
dcl_output oDepth
dcl_output oMask
dcl_output_control_point_count
dcl_output_sgv
dcl_output_siv
dcl_outputTopology
dcl_resource
dcl_resource原始
结构化dcl_resource
dcl_sampler
dcl_stream
dcl_temps
dcl_tessellator_domain
dcl_tessellator_output_primitive
dcl_tessellator_partitioning
dcl_tgsm_raw
dcl_tgsm_structured
dcl_thread_group
dcl_uav_raw
dcl_uav_structured
dcl_uav_typed
ddiv
default
deq
deriv_rtx_coarse

deriv_rtx_fine
deriv_rty_coarse
deriv_rty_fine
dfma
dge
丢弃
div
dlt
dmax
dmin
dmov
dmovc
dmul
dne
dp2
dp3
dp4
drcp
else
发出
emit_stream
emitThenCut
emitThenCut_stream
endif
endloop
endswitch
eq
exp
f16tof32
f32tof16
fcall
firstbit
Frc
ftod
ftoi
ftou
gather4
gather4_c
gather4_po
gather4_po_c

ge
hs_control_point_phase
hs_decls
hs_fork_phase
hs_join_phase
iadd
ibfe
ieq
if
lge
ilt
imad
imin
imm_atomic_alloc
imm_atomic_and
imm_atomic_cmp_exch
imm_atomic_consume
imm_atomic_exch
imm_atomic_iadd
imm_atomic_imax
imm_atomic_imin
imm_atomic_or
imm_atomic_umax
imm_atomic_umin
imm_atomic_xor
imul
ine
ineg
ishl
ishr
itof
label
ld
ld_raw
ld_structured
ld_uav_typed
ld2dms
Lod
log
loop

lt
mad
max
min
mov
movc
mul
ne
nop
not
or
rcp
树脂fo
Ret
retc
round_ne
round_ni
round_pi
round_z
rsq
sample
sample_b
sample_c
sample_c_lz
sample_d
sample_l
sampleinfo
samplepos
sincos
sqrt
store_raw
store_structured
store_uav_typed
swapc
switch

同步

uaddc
ubfe
udiv
uge

ult
umad
umax
umin
umul
ushr
usubb
utof
xor

相关主题

Asm 着色器参考

着色器模型 5 指令修饰符

项目 • 2022/09/23

指令修饰符在写入寄存器之前影响指令的结果。 着色器模型 5 支持以下指令修饰符。

源操作数修饰符

- 绝对值
- 否定

指令结果修饰符

- Precise
- 饱和

相关主题

着色器模型 5 程序集 (DirectX HLSL)

精确

项目 • 2022/12/14

按指令禁用算术重构。

精确 (组件掩码)

此修饰符需要全局着色器标志“REFACTORING_ALLOWED”。 当存在REFACTORING_ALLOWED时，单个指令的各个组件结果可能强制由编译器或驱动程序保持精确或不可重构。 如果 **疯狂** 指令的组件标记为 **精确**，硬件必须执行 **疯狂** 指令或完全相同的等效指令，并且不能将其拆分为后跟添加的乘法。 相反，后跟加法，其中一个或两者都标记为 **精确**，不能合并成融合 **的疯子**。

如果未指定REFACTORING_ALLOWED，则不允许 **精确** 修饰符。 不需要，因为一切都是精确的。 **精确的**修饰符会影响任何运算，而不仅仅是算术。

最小着色器模型

以下着色器模型中支持此修饰符。

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	是
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 指令修饰符

atomic_and (sm5 - asm)

项目 • 2023/03/09

原子按位 AND 到内存。

atomic_and dest、dstAddress[.swizzle]、src0[.select_component]

项	说明
dest	[in]使用 *src0* 与 AND 的组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，它也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]与 *dest* 一起指向 AND 的组件。

备注

此操作在 32 位每个组件地址 *dstAddress* 处以原子方式执行操作数 *src0* 的单个分量 32 位位 AND。

从地址获取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

着色器不会返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅作为派生物的实际像素/样本的帮助程序，则此指令根本不更改 *dest* 内存， (无提示) 。

u# 上的超出边界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且字节偏移到结构 (地址) 的第二个组件导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界寻址 (该特定 g# 的边界，而不是所有共享内存的边界，) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从Windows 8开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_cmp_store (sm5 - asm)

项目 • 2023/03/09

原子比较和写入内存。

atomic_cmp_store dest、dstAddress[.swizzle]、src0[.select_component]、src1[.select_component]

项	说明
dest	[in]要与 *src0 进行比较的*组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dest* 进行比较的 32 位值。
src1	[in]如果比较值相同，则要写入内存的值。

备注

此指令执行操作数 *src0* 与 *dest* 的单个组件 32 位值比较，每个组件地址 *dstAddress* 为 32 位。

如果比较值相同，则 *src1* 中的单组件 32 位值将写入目标内存。 否则，目标不会更改。

整个比较和写入操作以原子方式执行。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

从地址中提取的组件数由 dest u# 或 g# 的维数决定。

不会向着色器返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本指令根本不会改变 *dest* 内存， (无提示) 。

u# 上的越界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界外寻址 (该特定 g# 的边界，而不是所有共享内存) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_iadd (sm5 - asm)

项目 • 2023/03/09

原子整数添加到内存中。

atomic_iadd dest、dstAddress[.swizzle]、src0[.select_component]

项	说明
dest	[in]要通过 *src0* 添加的组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要添加到 *dest 的*组件。

备注

此指令以原子方式将操作数 *src0* 的单个分量 32 位整数加到 *dest* 中，每个组件地址 *dstAddress* 为 32 位。 此指令对签名不敏感。

从地址中提取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

不会向着色器返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅作为派生的真实像素/样本的帮助程序，则此指令根本不会更改 *dest* 内存， (无提示) 。

u# 上的越界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界外寻址 (该特定 g# 的边界，而不是所有共享内存) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_imax (sm5 - asm)

项目 • 2023/03/09

内存的原子带符号整数最大值。

atomic_imax dest、dstAddress[.swizzle]、src0[.select_component]

项	说明
dest	[in]要与 *src0 进行比较的*组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dest 进行比较的*组件。

备注

此操作以原子方式对每个组件地址 *dstAddress* 执行 32 位有符号最大操作数 *src0* 到 *dest* 的 32 位组件。

从地址中提取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

不会向着色器返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅作为派生的真实像素/样本的帮助程序，则此指令根本不会更改 *dest* 内存， (无提示) 。

u# 上的越界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界外寻址 (该特定 g# 的边界，而不是所有共享内存) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_imin (sm5 - asm)

项目 • 2023/03/09

从原子带符号整数最小值到内存。

atomic_imin dest，dstAddress[.swizzle]， src0[.select_component]

项	说明
dest	[in]要与 *src0 进行比较的*组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，它也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dest 进行比较的*组件。

备注

此指令以原子方式对每个组件地址 32 位的 *dstAddress* 执行单个组件 32 位有符号的最小操作数 *src0* 到 *dest*。

从地址获取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

着色器不会返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅用作实际像素/样本导数的帮助程序，则此指令根本不更改 *dest* 内存， (无提示) 。

u# 上的超出边界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且字节偏移到结构 (地址) 的第二个组件导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界寻址 (该特定 g# 的边界，而不是所有共享内存的边界，) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从Windows 8开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_or (sm5 - asm)

项目 • 2023/03/09

原子按位 OR 到内存。

atomic_or dest，dstAddress[.swizzle]， src0[.select_component]

项	说明
dest	[in]包含 *src0* 的 OR 的组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，它也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]具有 *dest* 的 OR 组件。

备注

此指令以原子方式对每个组件地址 *dstAddress* 执行 32 位操作数 *src0* 的单组件按位 OR 执行 *dest*。

从地址获取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

着色器不会返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅用作实际像素/样本导数的帮助程序，则此指令根本不更改 *dest* 内存， (无提示) 。

u# 上的超出边界寻址会导致不会写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界寻址 (该特定 g# 的边界，而不是所有共享内存的边界，) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从Windows 8开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_umax (sm5 - asm)

项目 • 2023/03/09

内存的原子无符号整数最大值。

atomic_umax dest、dstAddress[.swizzle]、src0[.select_component]

项	说明
dest	[in]要与 *src0 进行比较的*组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dest 进行比较的*组件。

备注

此指令以原子方式对每个组件地址 *dstAddress* 执行 32 位操作数 *src0* 到 *dest* 的 32 位无符号最大组件。

从地址中提取的组件数由 dest u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

不会向着色器返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅作为派生的真实像素/样本的帮助程序，则此指令根本不会更改 *dest* 内存， (无提示) 。

u# 上的越界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界外寻址 (该特定 g# 的边界，而不是所有共享内存) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_umin (sm5 - asm)

项目 • 2023/03/09

最小原子无符号整数到内存。

atomic_umin dest、dstAddress[.swizzle]、src0[.select_component]

项	说明
dest	[in]要与 *src0 进行比较的*组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dest 进行比较的*组件。

备注

此指令以原子方式对每个组件地址 *dstAddress* 执行 32 位无符号操作数 *src0* 到 *dest* 的 32 位无符号组件。

从地址中提取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

不会向着色器返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅作为派生的真实像素/样本的帮助程序，则此指令根本不会更改 *dest* 内存， (无提示) 。

u# 上的越界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界外寻址 (该特定 g# 的边界，而不是所有共享内存) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

atomic_xor (sm5 - asm)

项目 • 2023/03/09

原子按位 XOR 到内存。

atomic_xor dest、dstAddress[.swizzle]、src0[.select_component]

项	说明
dest	[in]具有 *src0* 的 XOR 的组件。 此值必须是 UAV) (u#) (无序访问视图。 在计算着色器中，它也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]具有 *dest* 的 XOR 的组件。

备注

此指令在 32 位每个组件地址 *dstAddress处以*原子方式执行操作数 *src0* 的单个分量 32 位 XOR。

从地址获取的组件数由 *dest* u# 或 g# 的维数决定。

如果 *dest* 是 u#，则可以将其声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为R32_UINT/_SINT。

如果 *dest* 为 g#，则必须将其声明为原始或结构化。

着色器不会返回任何内容。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅用作实际像素/样本导数的帮助程序，则此指令根本不更改 *dest* 内存， (无提示) 。

u# 上的超出边界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且字节偏移到结构 (地址) 的第二个组件导致越界访问，则 UAV 的整个内容将变得未定义。

g# 上的边界寻址 (该特定 g# 的边界，而不是所有共享内存的边界，) 会导致所有共享内存的整个内容变得未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从Windows 8开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

bfi (sm5 - asm)

项目 • 2022/09/24

给定数字 LSB 的位范围，请将该位数置于另一个数字中的任何偏移量。

bfi dest[.mask]， src0[.swizzle]， src1[.swizzle]， src2[.swizzle]， src3[.swizzle]

项	说明
dest	[in]结果的地址。
src0	[in]要从 *src2* 获取的位字段宽度。
src1	[in]用于替换 *src3* 中的位位的位字段偏移量。
src2	[in]位取自的数字。
src3	[in]要替换的位的数字。

注解

LSB 5 位 *src0* 提供从 *src2* 获取的位字段宽度 (0-31) 。

src1 的 LSB 5 位提供位字段偏移量 (0-31) ，以开始替换从 *src3* 读取的数字中的位。

syntax

```
Given width, offset:
                bitmask = (((1 << width)-1) << offset) & 0xffffffff
                dest = ((src2 << offset) & bitmask) | (src3 & ~bitmask)
```

此指令用于打包整数或标志。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

bfrev (sm5 - asm)

项目 • 2022/09/22

反转 32 位数字。

bfrev dest[.mask]， src0[.swizzle]

项	说明
dest	[in]反转位的 *src0* 地址。
src0	[in]要反向的 32 位数字。

注解

例如，给定0x12345678结果将0x1e6a2c48。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

bufinfo (sm5 - asm)

项目 • 2022/09/24

查询缓冲区 (而不是常量缓冲区) 的元素计数。

bufinfo dest[.mask]， srcResource

项	说明
dest	[in]结果的地址。
srcResource	[in]除常量缓冲区外，在 SRV (t#) 或 UAV (u#) 。

备注

dest 中的所有组件都接收缓冲区着色器资源视图中元素的整数数。 元素数取决于视图参数，例如内存格式。

对于类型化缓冲区 SRV 或 UAV，返回值是视图 (元素数，其中元素是类型化格式的一个单元) 。

对于原始缓冲区 SRV 或 UAV，返回值是视图中的字节数。

对于结构化缓冲区 SRV 或 UAV，返回值是视图中的结构数。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

countbits (sm5 - asm)

项目 • 2022/09/22

对数字中设置的位数进行计数。

countbits dest[.mask]， src0[.swizzle]

项	说明
dest	[in]结果的地址。
src0	[in]输入 32 位数字。

注解

此指令可用于计算着色器输入覆盖率百分比。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

cut_stream (sm5 - asm)

项目 • 2022/09/23

在指定流中完成当前基元拓扑（如果已发出任何顶点）的几何着色器指令，并启动该流中几何着色器声明的类型的新拓扑。

cut_stream streamIndex

项	说明
streamIndex	[in]流索引。

注解

执行此指令后，将完成几何着色器调用之前发出的任何拓扑。 如果为以前的基元拓扑发出了足够的顶点，则会丢弃它们。 由于几何着色器的唯一可用输出拓扑是点列表、线条条和三角形条，因此永远不会有任何剩余的顶点。

streamIndex 必须是声明流的即时值 [0..3]。

在上一个拓扑（如果有）完成之后，此指令将导致新的拓扑开始，使用声明为几何着色器的输出的拓扑。

限制

- 此指令仅适用于几何着色器。
- **cut_stream** 可以在几何着色器中显示任意次数，包括在流控制内。
- 如果几何着色器结束并发出顶点，则生成它们的拓扑已完成，就好像执行 **了 cut_stream** 指令作为最后一个指令。
- 如果未声明流，则必须使用 剪切 而不是 **cut_stream**。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
			X		

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

爸爸 (sm5 - asm)

项目 • 2022/09/22

分量双精度添加。

dadd[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要添加 *src1* 的组件。
src1	[in]使用 *src0* 添加的组件

注解

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 有效的 *dest* 掩码为 .xy、.zw 和 .xyzw。 以下映射帖子重排：

- *dest* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。

F 表示有限实数。

src0 src1->	**-inf**	**-F**	**-0**	**+0**	**+F**	**+inf**	**NaN**
-inf	-inf	-inf	-inf	-inf	-inf	NaN	NaN
-F	-inf	-F	src0	src0	+-F 或 +-0	+inf	NaN
-0	-inf	src1	-0	+0	src1	+inf	NaN
+0	-inf	src1	+0	+0	src1	+inf	NaN
+F	-inf	+-F 或 +-0	src0	src0	+ F	+inf	NaN
+inf	NaN	+inf	+inf	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_function_body (sm5 - asm)

项目 • 2022/09/22

声明函数正文。

dcl_function_body fb#

项目	说明
Fb#	[in]函数的显示位置的标签。

注解

此指令声明了一个唯一的函数正文，其代码将在标签 fb# 的程序中出现。

函数主体用于函数表声明。 有关详细信息，请参阅 dcl_function_table。

在外壳着色器和域着色器中，存在多个阶段 (控制点阶段、分叉阶段和联接阶段) ，所有功能体 (标签 fb#) 出现在所有阶段之后，而不是按阶段分组。

对于可以存在的函数体数没有限制。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_function_table (sm5 - asm)

项目 • 2022/09/22

声明函数表。

dcl_function_table ft# = {fb# , fb# , ...}

项	说明
英尺	[in]函数表条目。

注解

此函数将函数表声明为前面声明的一组函数主体。

这类似于 C++ vtable，但接口的每个调用站点有一个条目，而不是每个方法。

函数表中可以列出多少个函数主体没有限制。

在一个或多个函数表中多次引用给定函数正文 fb# 是有效的，作为共享常见代码的方式。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_hs_fork_phase_instance_count (sm5 - asm)

项目 • 2023/03/09

在外壳着色器分叉阶段中声明分叉阶段实例计数。

dcl_hs_fork_phase_instance_count N

项	说明
N	[in]实例计数 (UINT 大于 0) 。

备注

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_hs_join_phase_instance_count (sm5 - asm)

项目 • 2023/03/09

声明外壳着色器联接阶段中的联接阶段实例计数。

dcl_hs_join_phase_instance_count N

项	说明
N	[in]实例计数 (UINT 大于 0) 。

备注

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_hs_max_tessfactor (sm5 - asm)

项目 • 2023/03/09

声明修补程序的 maxTessFactor。

dcl_hs_max_tessfactor N

项	说明
N	[in]maxTessFactor。

备注

maxTessFactor 是范围 {1.0 ... 中的 float32 值。64.0}.

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_input vForkInstanceID (sm5 - asm)

项目 • 2022/09/22

在外壳着色器分支阶段声明实例 ID。

dcl_input vForkInstanceID

项	说明
vForkInstanceID	[in]实例 ID。

注解

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_input vGSInstanceID (sm5 - asm)

项目 • 2022/09/22

启用几何着色器实例化。

dcl_input vGSInstanceID， instanceCount

项目	说明
vGSInstanceID	[in]实例 ID。
instanceCount	[in]实例计数。

注解

声明的 *instanceCount* 参数指定几何着色器应为每个输入基元执行的实例数。 *instanceCount* 的最大值为 32。

通过 dcl_maxOutputVertexCount声明输出的最大顶点数分别应用于每个实例。

此声明中的实例计数乘以每个实例通过 dcl_maxOutputVertexCount的最大顶点计数，必须为 <1024。

给定几何着色器实例可以发出的数据量是 1024 标量最大值，通过计算为输入声明的所有标量，并乘以声明的输出顶点计数来验证。

使用几何着色器实例可有效地增加每个输入基元可以发出的数据总量。 单个实例的 1024 标量在单个输入基元的所有几何着色器实例中生成多达 1024*32 个输出数据的标量。 但是实例越多，每个实例可以发出的顶点就越少。 单个实例 (没有实例化) 可以发出 1024 个顶点。 如果声明 *32 实例，则表示每个实例只能输出 1024/32 = 32 个顶点。

几何着色器实例化声明向程序提供独立的 32 位整数输入寄存器 *vGSInstanceID*。 每个几何着色器实例由 *vGSInstanceID* [0，1，2...] 中包含的值进行标识。

vGSInstanceID 不是几何图形着色器输入顶点数组的一部分， (例如，在输入三角形) 时有 3 个顶点。 *vGSInstanceID* 寄存器本身，如 vPrimitiveID。

当每个几何着色器实例结束时，输出拓扑中存在隐式剪切，因此连续实例彼此不依赖。

尽管硬件可以并行执行每个几何图形着色器实例，但末尾的所有实例的输出将序列化，就像所有实例化几何着色器调用在循环中按顺序运行，循环访问 *vGSInstanceID* 从 0 到 *instanceCount-1*，每个实例末尾都有隐式输出拓扑削减。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
			X		

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_input vJoinInstanceID (sm5 - asm)

项目 • 2022/09/22

在外壳着色器联接阶段声明实例 ID。

dcl_input vJoinInstanceID

项目	说明
vJoinInstanceID	[in]实例 ID。

注解

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_input vOutputControlPointID (sm5 - asm)

项目 • 2022/09/24

在外壳着色器控制点阶段中声明输出控制点 ID。

dcl_input vOutputControlPointID

项	说明
vOutputControlPointID	[in]输出控制点 ID。

注解

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_input vThread (sm5 - asm)

项目 • 2023/03/09

声明计算着色器输入 ID。

dcl_input vThread

项	说明
vThread	[in]3 分量无符号 32 位整数 ID 值。 下列其中一项： • vThreadID.xyz • vThreadGroupID.xyz • vThreadIDInGroup.xyz • vThreadIDInGroupFlattened

dcl_input 是其他着色器阶段中的现有声明。 它在计算着色器中用于声明计算着色器特有的各种 3 分量无符号 32 位整数 ID 值。 它们是：

- vThreadID.xyz
- vGroupID.xyz
- vThreadIDInGroup.xyz
- vThreadIDInGroupFlattened (单组件)

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
					X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_input_control_point_count (sm5 - asm)

项目 • 2023/03/09

在外壳着色器声明部分中声明外壳着色器输入控制点计数。

dcl_input_control_point_count N

项	说明
N	[in]输入控制点计数 {1..32}。

备注

至少需要 1 个输入控制点;如果不需要，它可以为空。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_interface (sm5 - asm)

项目 • 2022/09/22

声明函数表指针 (接口) 。

dcl_interface fp#[arraySize][numCallSites] = {ft# , ft# , ...}

项	说明
Fp#	[in]函数表指针。

注解

在着色器可用之前，需要从 API 绑定每个接口。 绑定提供对其中一个函数表的引用，以便可以填充方法槽。 编译器不会为未引用的对象生成指针。

函数表指针具有一组完整的方法槽，以避免 C++ 指针到 vtable 表示形式需要的额外间接级别。 这还需要此指针为 5 元组。 在 HLSL 虚拟内联模型中，始终知道用于调用的全局变量/输入，以便我们可以为每个根对象设置表。

函数指针声明指示哪些函数表是合法的，可以使用它们。 这也允许派生方法关联信息。

接口声明的第一个 [] 是数组大小。 如果使用动态索引，声明将指示如下所示。 接口指针数组也可以静态编制索引，接口指针数组不一定意味着动态索引。

接口指针的编号从第一个声明的 0 开始，随后考虑数组大小，因此四个条目数组 fp0[4][1] 后面的第一个指针将是 fp4[][]。

接口声明的第二个 [] 是调用站点数，它必须与声明中引用的每个表中的正文数匹配。

接口声明中可以列出多少个函数表 (ft#) 选项。

给定函数表 (ft#) 可以在一个或多个接口声明中多次出现。

限制

- 着色器中的对象站点数（即其 [arraySize] 声明的所有 *fp#* 声明的总和）不得超过 253。 此数字对应于可以存在的 **此** 指针数。 运行时强制实施此 253 限制，以限制 DDI 的大小，以便传达此指针数据。
- 着色器中的呼叫站点数（即潜在分支目标数的所有 fcall 语句的总和）必须不超过 4096。

例如，对第一个 fp[][] 维度使用静态索引的 fcall 计数为 *fp[][]* 维度：

```
fcall fp0[0][0] // +1
```

使用动态索引的 **fcall** 将计算为dcl_interface) (第 一 个 [] 数组中的元素数：

```
dcl_interface_dynamicindexed fp1[2][1] = {ft2, ft3, ft4} ...

fcall fp1[r0.z + 0][1] // +2
```

此限制有助于某些实现在常量缓冲区般的存储中轻松拟合函数主体选择表。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

cs_4_0和cs_4_1支持 UAV 和 SRV 的此指令。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_interface_dynamicindexed (sm5 - asm)

项目 • 2022/09/23

) (接口声明函数表指针。

dcl_interface_dynamicindexed fp#[arraySize][numCallSites] = {ft#， ft#， ...}

项	说明
Fp#	[in]函数表指针。

注解

在着色器可用之前，需要从 API 绑定每个接口。 绑定提供对其中一个函数表的引用，以便可以填充方法槽。 编译器不会为未引用的对象生成指针。

函数表指针具有一组完整的方法槽，以避免 C++ 指针到 vtable 表示形式需要的额外间接级别。 这还需要此指针为 5 元组。 在 HLSL 虚拟内联模型中，始终知道用于调用的全局变量/输入，以便我们可以为每个根对象设置表。

函数指针声明指示哪些函数表是合法的，可用于它们。 这也允许派生方法关联信息。

接口声明的第一个 [] 是数组大小。 如果使用动态索引，声明将指示如下所示。 接口指针的数组也可以静态编制索引，接口指针数组不需要表示动态索引。

接口指针的编号从第一个声明的 0 开始，随后将数组大小考虑在内，因此四个条目数组 fp0[4][1] 之后的第一个指针将是 fp4[][]。

接口声明的第二个 [] 是调用站点数，它必须与声明中引用的每个表中的正文数匹配。

接口声明中可以列出多少个函数表 (ft#) 选项。

给定的函数表 (ft#) 可以在一个或多个接口声明中多次显示。

限制

- 着色器中的对象站点数（表示其 [arraySize] 声明的所有 *fp#* 声明的总和）必须不超过 253。 此数字对应于可以存在的 **此** 指针数。 运行时强制实施此 253 限制，以限制 DDI 的大小，以便通信此指针数据。

- 着色器中的调用站点数（即潜在分支目标数的所有 fcall 语句的总和）必须不超过 4096。

  例如，对第一个 *fp[][]* 维度使用静态索引的 fcall 计数为一个：

  ```
  fcall fp0[0][0] // +1
  ```

  使用动态索引的 **fcall** 将计算为 **dcl_interface)**（第一个 [] 数组中的元素数：

  ```
  dcl_interface_dynamicindexed fp1[2][1] = {ft2, ft3, ft4} ...
  ```

  ```
  fcall fp1[r0.z + 0][1] // +2
  ```

  此限制有助于某些实现轻松适应常量缓冲区存储中函数主体选择的表。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

cs_4_0和cs_4_1支持针对 UAV 和 SRV 的此说明。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_output oMask (sm5 - asm)

项目 • 2023/03/09

声明要由着色器编写的输出寄存器。

dcl_output o#[.mask]

项	说明
o#	[in]输出寄存器。 • # 是标识寄存器的名称。 • *[.mask]* 是一个可选的组件掩码 (.xyzw) ，用于指定要使用的寄存器组件。

备注

```
Example:
                dcl_output oMask[3].xyz
```

限制

- 组件掩码可以是 [xyzw] 的任何子集。 但是，在组件之间留下间隙会浪费空间。
- 声明组件掩码的超集是合法的，该组件掩码声明供下一阶段输入。 但是，不允许使用互斥掩码。 顶点着色器输出 o3.xy 意味着输入 v3.z 的像素着色器无效，但输入 v3.x 或 v3.y 或 v3.xy 有效。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_output_control_point_count (sm5 - asm)

项目 • 2022/09/22

声明外壳着色器输出控制点计数。

dcl_output_control_point_count N

项	说明
N	[in]一个从 0 到 32 的整数值，该值指定输出控制点计数。

注解

如果不需要，外壳着色器可以输出 0 个控制点。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	声明部分				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_resource_raw (sm5 - asm)

项目 • 2023/03/09

声明着色器资源输入并将其分配给 t# - 资源的占位符寄存器。

dcl_resource_raw dstSRV

项	说明
dstSRV	[in]声明为对原始缓冲区的 ShaderResourceView 的引用的 t# 寄存器。

备注

结构的内容没有类型;对内存执行的操作可能会隐式地将数据解释为具有 类型。

引用原始 t# 的指令采用 1D 地址，这是一个无符号的 32 位值，用于指定缓冲区中 32 位对齐位置的字节偏移量。 地址必须是 4 (字节) 的倍数。

绑定到声明为 raw 的 t# 的视图必须在创建时指定 RAW;否则，从着色器访问时的行为是不确定的。

cs_4_0和cs_4_1支持此指令。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_resource_structured (sm5 - asm)

项目 • 2023/03/09

声明着色器资源输入并将其分配给 t# - 资源的占位符寄存器。

dcl_resource_structured dstSRV，structByteStride

项	说明
dstSRV	[in]声明为对具有指定步幅的结构化缓冲区的 ShaderResourceView 的引用的 t# 寄存器，该步幅必须绑定到 API 上的 SRV 槽 #。
structByteStride	[in]一个 uint，用于指定要声明的缓冲区中结构的大小（以字节为单位）。 此值必须大于零。

备注

结构的内容没有类型：对内存执行的操作可能会隐式地将数据解释为具有 类型。

引用结构化 t# 的指令采用 2D 地址，其中第一个组件选取 [结构]，第二个组件选取 [结构中的偏移量，32 位的倍数]。

cs_4_0和cs_4_1支持此指令。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_stream (sm5 - asm)

项目 • 2023/03/09

声明几何着色器输出流。

dcl_stream mN

项	说明
锰	[in] *N* 为 0..3 (m0.的流。m3) 。

备注

给定的流只能声明一次。

如果未声明流，则假定输出和输出拓扑声明适用于流 0。

第一 **个dcl_stream** 不能出现在任何 **dcl_output** 或 **dcl_outputTopology** 语句之后。

任何 **dcl_stream** m # **语句之后** 的任何 **dcl_output或dcl_outputToplogy** 语句都定义流 m# 的输出。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
			X		

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_tessellator_domain (sm5 - asm)

项目 • 2023/03/09

在外壳着色器声明部分中声明细化器域和域着色器。

dcl_tessellator_domain域

项	说明
域	[in]域。 下列其中一项： • domain_isoline • domain_tri • domain_quad

备注

如果外壳着色器和域着色器提供不匹配的域或任何其他冲突的分位，则行为未定义。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	声明部分	X			

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_tessellator_output_primitive (sm5 - asm)

项目 • 2023/03/09

在外壳着色器声明部分中声明细化器输出基元类型。

dcl_tessellator_output_primitive类型

项	说明
类型	[in]输出基元类型。 下列其中一项： • output_point • output_line • output_triangle_cw • output_triangle_ccw

备注

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	声明部分				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_tessellator_partitioning (sm5 - asm)

项目 • 2023/03/09

在外壳着色器声明部分中声明细化器分区。

dcl_tessellator_partitioning模式

项	说明
模式	[in]细化器分区模式。 下列其中一项： • partitioning_integer • partitioning_pow2 • partitioning_fractional_odd • partitioning_fractional_even

备注

从硬件的角度来看，_pow2的行为就像_integer一样。 由 HLSL 着色器作者和/或编译器代码将 TessFactors 舍入为 2 的幂。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
	声明部分				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_tgsm_raw (sm5 - asm)

项目 • 2022/09/24

声明对可用于计算着色器线程组的共享内存空间区域的引用。

dcl_tgsm_raw g#、byteCount

项	说明
G#	[in]对非类型化共享内存的大小字节 *计数* 块的引用。
byteCount	[in]必须是 4 的倍数。

注解

所有 g# 的总存储必须 <= 每个线程组可用的共享内存量，即 32kB。

在极端情况下，可以声明 8192 个总 g# s，每个字节 *计数* 为 4。

在相反的极端，可以使用 *字节计数* 32768 声明单个 g# 。

> **备注**
>
> cs_4_0和cs_4_1支持 dcl_tgsm_structured，但不支持 **dcl_tgsm_raw**。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
					X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_tgsm_structured (sm5 - asm)

项目 • 2022/09/23

声明对可用于计算着色器线程组的共享内存空间区域的引用。 内存被视为结构数组。

dcl_tgsm_structured g#、structByteStride、structCount

项	说明
G#	[in]对大小为 *structByteStridestructCount* * 字节的共享内存块的引用。
structByteStride	[in]结构步幅。 此值是字节的 uint，必须是 4 的倍数。
structCount	[in]结构数。

注解

所有 g# 的总存储必须 <= 每个线程组可用的共享内存量，即 32kB 或 8192 32 位标量。

在极端情况下，可以声明 8192 个总 g# s（如果每个结构具有 4 *个结构* ，并且 *结构计数* 为 1）。

在相反的极端，你可以声明一个 g# ，其结构步幅为 32kB，结构计数为 1。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
					X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否

着色器模型	支持
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_thread_group (sm5 - asm)

项目 • 2022/09/22

声明线程组大小。

dcl_thread_group x、y、z

项目	说明
X	[in]一个 usigned 32 位整数。 1 <= x <= 1024
Y	[in]一个 usigned 32 位整数。 1 <= y <= 1024
Z	[in]一个 usigned 32 位整数。 1 <= z <= 64

备注

x*y*z <= 1024

此线程组声明必须在计算着色器中出现一次。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
					X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_uav_raw (sm5 - asm)

项目 • 2022/09/24

声明非排序的访问视图 (UAV) 以供着色器使用。

dcl_uav_raw[_glc] dstUAV

项	说明
dstUAV	[in]UAV。

注解

dstUAV 是一个 u# 寄存器，声明为对缓冲区的 UnorderedAccessView 的引用，其中缓冲区显示为 32 位非类型化条目的简单 1D 数组。

对内存执行的操作可能会隐式地将数据解释为具有类型。

_glc标志表示“全局一致”。 缺少_glc意味着 UAV 仅在计算着色器中声明为“组一致”，或者在单个像素着色器调用中声明为“本地一致”。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

> **备注**
>
> cs_4_0和cs_4_1支持此说明。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_uav_structured (sm5 - asm)

项目 • 2022/09/23

声明无序访问视图， (UAV) 以供着色器使用。

dcl_uav_structured[_glc] dstUAV， 结构ByteStride

项	说明
dstUAV	[in]UAV。
structByteStride	[in]结构的大小（以字节为单位）。

注解

dstUAV 是一个 u# 寄存器，它声明为对结构化缓冲区的 UnorderedAccessView 的引用，其步幅必须绑定到 API 上的 UAV 槽 #。

结构的内容没有类型;对内存执行的操作可能会隐式解释为具有类型的数据。

structByteStride 是声明的缓冲区中的结构大小（以字节为单位）。 此值必须大于零。 *structByteStride* 的类型为 uint，必须是 4 的倍数。

引用结构化 u# 的说明采用 2D 地址，其中第一个组件选取 [结构]，第二个组件选取 [结构中的偏移量，以对齐字节为单位]。

_glc标志表示“全局一致”。 缺少_glc意味着 UAV 仅在计算着色器中声明为“组一致”，或在单个像素着色器调用中声明为“本地一致”。

_opc标志是保留顺序计数器。 它指示如果 UAV 绑定到槽 # (u#) ，则必须使用 COUNTER 标志创建它。 这意味着着色器中的 imm_atomic_alloc 或 imm_atomic_consume 操作操作操作计数器，其值可以在着色器中用作对 UAV 中某个位置的永久引用。 着色器结束后，无法重新排序数据。

缺少_opc标志意味着，如果着色器使用imm_atomic_alloc 或imm_atomic_consume指令，并且 UAV 绑定到槽 # (u) ，则必须使用 APPEND 标志创建，该标志提供一个计数器，该计数器不能保证着色器调用后保留顺序。

如果_opc标志不存在，并且着色器不包含 imm_atomic_alloc 或 imm_atomic_consume 指令，则允许使用 COUNTER 标志创建 (UAV，) u) 允许使用 COUNTER 标志创建 UAV， (此着色器) 不使用计数器，则不 (没有计数器) ，但不允许使用 APPEND 标志。

> **备注**
>
> cs_4_0和cs_4_1支持 **dcl_tgsm_structured**，但不支持 dcl_tgsm_raw。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此说明适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始可用。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

> **备注**
>
> cs_4_0和cs_4_1支持此说明。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dcl_uav_typed (sm5 - asm)

项目 • 2022/09/24

声明非排序的访问视图 (UAV) 以供着色器使用。

dcl_uav_typed[_glc] dstUAV， 维度， 类型

项	说明
dstUAV	[in]UAV。
维 度	[in]指定访问 UAV 的说明提供多少个维度。
类型	[in]UAV 的类型。

注解

dstUAV 是一个 u# 寄存器，它被声明为对 UnorderedAccessView 的引用，必须在 API 上绑定到 UAV 槽 #。

维度必须是缓冲区、Texture1D、Texture1DArray、Texture2D、Texture2DArray 或 Texture3D。 这表示访问 UAV 的任何指令提供多少个维度：1 (Texture1D、Buffer) 、2 (Texture1DArray、Texture2D) 或 3 (Texture2DArray、Texture3D) 。

类型为 {UNORM，SNORM，UINT，SINT，FLOAT}。 使用声明的 u# 执行的操作必须与此处声明的类型兼容，并且绑定到槽 # 的 UAV 还必须具有相同的类型。

_glc标志代表“全球一致”。 缺少_glc意味着 UAV 仅在计算着色器中声明为“组一致”，或者在单个像素着色器调用中声明为“本地一致”。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

> **备注**
>
> 计算着色器 4.x 不支持此指令。

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ddiv (sm5 - asm)

项目 • 2023/03/09

计算分量级双精度除法。

ddiv[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in] 操作的结果。 结果值必须精确到 0.5 ULP。
src0	[输入] 被除数。
src1	[输入] 除数。

备注

每当将除运算符与 double 一起使用时，HLSL 编译器都会发出 DDIV 指令。 此指令的准确性要求为 0.5 ULP。

使用此指令的着色器将标有着色器标志，这将导致它们无法绑定，除非满足以下所有条件。

- 系统支持 DirectX 11.1。
- 系统包括 WDDM 1.2 驱动程序。
- 驱动程序通过D3D11_FEATURE_DATA_D3D11_OPTIONS报告此指令的支持 。 **ExtendedDoublesShaderInstructions** 设置为 **TRUE**。

下表显示了使用各种数字类执行指令时的结果，假定不会发生溢出或下溢。

在此表中，F 表示有限实数。

src0 src1 ->	**-Inf**	**-F**	**-1.0**	**-0**	**+0**	**+1.0**	**+F**	**+inf**	**NaN**
-Inf	NaN	+inf	+inf	+inf	-inf	-inf	-inf	NaN	NaN
-F	+0	+ F	-src0	+inf	-inf	src0	-F	-0	NaN
-0	+0	+0	+0	NaN	NaN	-0	-0	-0	NaN
+0	-0	-0	-0	NaN	NaN	+0	+0	+0	NaN
+F	-0	-F	-src0	-inf	+inf	src0	+ F	+0	NaN
+inf	NaN	-inf	-inf	-inf	+inf	+inf	+inf	NaN	NaN

src0 src1 ->	-Inf	-F	-1.0	-0	+0	+1.0	+F	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

deq (sm5 - asm)

项目 • 2022/09/22

分量双精度相等比较。

deq[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1 进行比较的*组件。
src1	[in]要与 *src0 进行比较的*组件。

注解

此指令对每个组件执行双精度浮点比较 (*src0src1 ==*），并将结果写入 *dest*。

如果比较为 true，则为该组件返回 32 位0xFFFFFFFF。 否则返回 32 位0x00000000。

与 NaN 的比较返回 false。

有效的 *dest* 掩码是任意一个或 2 个组件。
即：.x、.y、.z、.w、.xy、.xz、.xw、.yz、.yw、.zw 掩码中的第一个 *dest* 组件接收第一个双比较的 32 位结果。 掩码中的第二个组件（如果存在）接收第二次双重比较的 32 位结果。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 以下 *src* 映射帖子重排：

- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

deriv_rtx_coarse (sm5 - asm)

项目 • 2022/09/23

计算组件更改率。

deriv_rtx_coarse[_sat] dest[.mask]， [-]src0[_abs][.swizzle]，

项	描述
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

此指令计算 *src0* (帖子-重排) 的每个 float32 组件的内容更改速率，与 RenderTarget x 方向 (rtx) 或 RenderTarget y 方向 (请参阅deriv_rty_coarse) 。 每个 2x2 像素标记只计算一个 x，y 派生配对。

当前像素着色器调用中的数据可能或可能不会参与所请求的派生的计算，因为每 2x2 象限只计算一次派生。 例如，x 派生体可以是像素上行的增量，y 方向 (deriv_rty_coarse) 可能是像素左列中的增量。 确切的计算由硬件供应商决定。 也没有规范描述 2x2 象限如何对齐或平铺在基元上。

派生数以粗略级别计算，每 2x2 像素象限计算一次。 此说明和 deriv_rty_coarse 是 deriv_rtx_fine 和 deriv_rty_fine的替代方法。 这些_coarse和_fine派生指令是以前着色器模型中 **deriv_rtxderiv_rty** 的替代方法。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

deriv_rtx_fine (sm5 - asm)

项目 • 2022/09/23

计算组件更改率。

deriv_rtx_fine[_sat] dest[.mask]， [-]src0[_abs][.swizzle]，

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

此指令计算 *src0* (帖子-swizzle) 每个 float32 组件的内容更改速率，而 RenderTarget x 方向 (rtx) 或 RenderTarget y 方向 (看到deriv_rty_fine) 。 2x2 邮票中的每个像素获取 x/y 派生计算的唯一配对

当前像素着色器调用中的数据始终参与所请求派生的计算。 在 2x2 像素象限中，当前像素位于其中，x 派生是包含当前像素在内的 2 像素行的增量。 y 派生是包含当前像素的 2 像素列的增量。 没有规范描述 2x2 象限如何对齐或平铺在基元上。

派生在精细级别计算， (2x2 四) 中每个像素的 x/y 派生配对的唯一计算。 此指令和 deriv_rty_fine 是 deriv_rtx_coarse 和 deriv_rty_coarse的替代方法。 这些_coarse和_fine派生指令是**deriv_rtx这些_coarse**和_fine派生指令的替代方法，替代以前的着色器模型中**的deriv_rtx和deriv_rty。**

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

deriv_rty_coarse (sm5 - asm)

项目 • 2022/09/22

计算组件更改率。

deriv_rty_coarse[_sat] dest[.mask]， [-]src0[_abs][.swizzle]，

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

有关详细信息，请参阅 deriv_rtx_coarse。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

deriv_rty_fine (sm5 - asm)

项目 • 2022/09/22

计算组件更改率。

deriv_rty_fine[_sat] dest[.mask]， [-]src0[_abs][.swizzle]，

项	说明
dest	[in]操作结果的地址。
src0	[in]操作中的组件。

注解

有关详细信息，请参阅 deriv_rtx_fine。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dfma (sm5 - asm)

项目 • 2022/09/22

执行融合乘法添加。

dfma[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]，[-]src2[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 结果值必须准确到 0.5 ULP。 *dest = src0 * src1 + src2*
src0	[in]要与 *src1* 相乘的组件。
src1	[in]要与 *src0* 相乘的组件。
src2	[in]要添加到*src0src1 * 的*组件。

注解

使用此指令的着色器将标记为着色器标志，除非满足以下所有条件，否则它们将无法绑定。

- 系统支持 DirectX 11.1。
- 系统包含 WDDM 1.2 驱动程序。
- 驱动程序通过D3D11_FEATURE_DATA_D3D11_OPTIONS报告对此指令的支持 。 **ExtendedDoublesShaderInstructions** 设置为 **TRUE**。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dge (sm5 - asm)

项目 • 2022/09/22

分量双精度大于或等于比较。

dge[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项目	说明
dest	[in]操作结果的地址。
src0	要与 *src1 进行比较的*组件。
src1	要与 *src0 进行比较的*组件。

备注

此指令对每个组件执行双精度浮点比较 (*src0*>= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则为该组件返回 32 位0xFFFFFFFF。 否则返回 32 位0x00000000。

与 NaN 的比较返回 false。

有效的 *dest* 掩码是任意一个或两个组件。
即：.x、.y、.z、.w、.xy、.xz、.xw、.yz、.yw、.zw 掩码中的第一个 *dest* 组件接收第一个双比较的 32 位结果。 掩码中的第二个组件（如果存在）接收第二次双重比较的 32 位结果。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 以下 *src* 映射帖子重排：

- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dlt (sm5 - asm)

项目 • 2022/09/22

分量双精度小于比较。

dlt[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1* 进行比较的组件。
src1	[in]要与 *src0* 进行比较的组件。

注解

此指令对每个组件执行双精度浮点比较 (*src0src1*<) ，并将结果写入 *dest*。

如果比较为 true，则为该组件返回 32 位0xFFFFFFFF。 否则返回 32 位0x00000000。

与 NaN 的比较返回 false。

有效的 *dest* 掩码是任意一个或两个组件。 也就是
说：.x、.y、.z、.w、.xy、.xz、.xw、.yz、.yw、.zw 掩码中的第一个 *dest* 组件接收第一个双精度比较的 32 位结果。 如果存在) 则掩码中的第二个组件 (收到第二个双精度比较的 32 位结果。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 以下 *src* 映射帖子重排：

- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 的双 vec2。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 的双 vec2。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dmax (sm5 - asm)

项目 • 2022/09/22

分量双精度最大值。

dmax[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 *dest = src0>= src1 ？ src0 ： src1* >>使用 = 而不是这样，如果 min (x，y) = x，则 max (x，y) = y。
src0	[in]要与 *src1* 进行比较的值。
src1	[in]要与 *src0* 进行比较的值。

注解

NaN 具有特殊处理。 如果一个源操作数为 NaN，则返回另一个源操作数。 选择按组件进行。 如果两者均为 NaN，则返回任何 NaN 表示形式。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 有效的 *dest* 掩码是 .xy、.zw 和 .xyzw。 以下 *src* 映射帖子重排：

- *dest* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 的双 vec2。
- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 的双 vec2。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 的双 vec2。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dmin (sm5 - asm)

项目 • 2022/09/22

分量双精度最小值。

dmin[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 *dest = src0<src1 ？ src0 ： src1* < 使用而不是 <= ，以便如果最小 (x，y) = x，则最大 (x，y) = y。
src0	[in]要与 *src1 进行比较的*组件。
src1	[in]要与 *src0 进行比较的*组件。

注解

NaN 具有特殊处理。 如果一个源操作数为 NaN，则返回另一个源操作数。 选择按组件进行。 如果两者均为 NaN，则返回任何 NaN 表示形式。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 有效的 *dest* 掩码为 .xy、.zw 和 .xyzw。 以下 *src* 映射帖子重排：

- *dest* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dmov (sm5 - asm)

项目 • 2022/09/22

按组件移动。

dmov[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	描述
dest	[in]移动目标。 *dest = src0.*
src0	[in]要移动的组件。

注解

除重排外，修饰符假定数据是浮点。 缺少修饰符即可移动数据，而无需更改位。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 以下 *src* 映射帖子重排：

- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dmovc (sm5 - asm)

项目 • 2022/09/23

按组件进行条件移动。

dmovc[_sat] dest[.mask]， src0[.swizzle]， [-]src1[_abs][.swizzle]， [-]src2[_abs][.swizzle]，

项	说明
dest	[in]移动目标。 如果 *src0*，则 *destsrc1* = else *destsrc2* = 。
src0	[in]要测试条件的组件。
src1	[in]如果条件为 true，则要移动的组件。
src2	[in]如果条件为 false，则要移动的组件。

注解

以下示例演示如何使用此指令。

syntax

```
                if(the dest mask contains .xy)
                {
                    if(the first 32-bit component of src0, post-swizzle,
                       has any bit set)
                    {
                        copy the first double from src1 (post swizzle)
                        into dest.xy
                    }
                    else
                    {
                        copy the first double from src2 (post swizzle)
                        into dest.xy
                    }
                }
                if(the dest mask contains .zw)
                {
                    if(the second 32-bit component of src0, post-swizzle,
                       has any bit set)
                    {
                        copy the second double from src1 (post swizzle)
                        into dest.zw
                    }
                    else
                    {
```

```
                        copy the second double from src2 (post swizzle)
                        into dest.zw
                    }
                }
```

dest 的有效掩码是 .xy、.zw、.xyzw。

src0 的有效重排是任何内容。 前两个组件帖子重排用于缩进两个 32 位条件值。

src1 和包含双精度的 *src2* 的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 为 .xy、.zw 和 .xyzw。

以下 *src* 映射帖子重排：

- *dest* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src0* 是 x 和 y (zw 中忽略) 的 32 位/组件 vec2。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src2* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

src1 和 *src2* 上的修饰符（除重排外）假定数据是双精度的。 缺少修饰符即可移动数据，而无需更改位。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dmul (sm5 - asm)

项目 • 2022/09/23

分量双精度乘法。

dmul[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。 *dest = src0 * src1*
src0	[in]要与 *src1* 相乘的组件。
src1	[in]要与 *src0* 相乘的组件。

注解

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 有效的 *dest* 掩码为 .xy、.zw 和 .xyzw。 以下 *src* 映射帖子重排：

- *dest* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

下表显示了在使用各种数字类执行指令时获取的结果，前提是不会发生溢出或下溢。

F 表示有限实数。

src0 src1->	**-inf**	**-F**	**-1.0**	**-0**	**+0**	**+1.0**	**+F**	**+inf**	**NaN**
-inf	+inf	+inf	+inf	NaN	NaN	-inf	-inf	-inf	NaN
-F	+inf	+ F	-src0	+0	-0	src0	-F	-inf	NaN
-1.0F	+inf	-src1	+1.0	+0	-0	-1.0	-src1	-inf	NaN
-0	NaN	+0	+0	+0	-0	-0	-0	NaN	NaN
+0	NaN	-0	-0	-0	+0	+0	+0	NaN	NaN
+1.0	-inf	src1	-1.0	-0	+0	+1	src1	+inf	NaN
+F	-inf	-F	-src0	-0	+0	src0	+ F	+inf	NaN
+inf	-inf	-inf	-inf	NaN	NaN	+inf	+inf	+inf	NaN

src0 src1->	-inf	-F	-1.0	-0	+0	+1.0	+F	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dne (sm5 - asm)

项目 • 2022/09/22

分量双精度不相等比较。

dne[_sat] dest[.mask]， [-]src0[_abs][.swizzle]， [-]src1[_abs][.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]要与 *src1 进行比较的*组件。
src1	[in]要与 *src0 进行比较的*组件。

注解

此指令对每个组件执行双精度浮点比较 (*src0* ！= *src1*) ，并将结果写入 *dest*。

如果比较为 true，则为该组件返回 32 位0xFFFFFFFF。 否则返回 32 位0x00000000。

与 NaN 的比较返回 true。

有效的 *dest* 掩码是任意一个或两个组件。
即：.x、.y、.z、.w、.xy、.xz、.xw、.yz、.yw、.zw 掩码中的第一个 *dest* 组件接收第一个双比较的 32 位结果。 掩码中的第二个组件（如果存在）接收第二次双重比较的 32 位结果。

源参数的有效重排是 .xyzw、.xyxy、.zwxy、.zwzw。 以下 *src* 映射帖子重排：

- *src0* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。
- *src1* 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB 的双 vec2，w 32MSB) 。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

drcp (sm5 - asm)

项目 • 2023/03/09

计算分量级双精度倒数。

drcp[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]结果的地址 *dest* = **1.0** / *src0*。 结果值必须准确到 1.0 ULP
src0	[in]要取的倒数。

备注

仅当当使用 double 作为参数时，HLSL 编译器才通过 rcp () 内部函数显式调用 DRCP 指令。 此指令的准确性要求为 1.0 ULP。

使用此指令的着色器将标有着色器标志，这将导致它们无法绑定，除非满足以下所有条件。

- 系统支持 DirectX 11.1。
- 系统包括 WDDM 1.2 驱动程序。
- 驱动程序通过D3D11_FEATURE_DATA_D3D11_OPTIONS报告此指令的支持 。 **ExtendedDoublesShaderInstructions** 设置为 **TRUE**。

下表显示了使用各种数字类执行指令时获得的结果，假定不会发生溢出或下溢。

在此表中，F 表示有限实数。

src->	**-Inf**	**-F**	**-0**	**+0**	**+F**	**+inf**	**NaN**
dest->	-0	-F	-inf	+inf	+ F	+0	NaN

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

dtof (sm5 - asm)

项目 • 2023/03/09

从双精度浮点数据到单精度浮点数据的组件转换。

dtof dest[.mask]， [-]src0[.swizzle]，

项	说明
dest	[in]转换的数据的地址。
src0	[in]要转换的数据。

备注

使用舍入到最接近的舍入，将源的每个组件从双精度表示形式转换为单精度表示形式。

源参数的有效重排为 .xyzw、.xyxy、.zwxy、.zwzw。

有效的 *dest* 掩码是任意一个或两个组件。
即：.x、.y、.z、.w、.xy、.xz、.xw、.yz、.yw、.zw 第一个转换的结果将转到掩码中的第一个组件，而第二个组件的结果将进入掩码中的第二个组件（如果存在）。

dest 组件为 float32。

src0 是 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 后重排的双 vec2。

对于 float32< 双>精度转换，实现可以遵循 float32 denorms，也可以刷新它们。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是

着色器模型	支持
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

emit_stream (sm5 - asm)

项目 • 2022/09/23

向给定流发出顶点。

emit_stream streamIndex

项	说明
streamIndex	[in]流索引。

注解

此指令导致给定流的所有已声明 o# 寄存器从几何图形着色器中读取以生成顶点。 推断发出，所有流的输出寄存器中的所有数据都未初始化，而不仅仅是发出的流。

streamIndex 必须是声明流的即时值 [0..3]。

发出多个 **emit_stream** 调用时，会生成基元。

限制

- **emit_stream** 可以在几何着色器中显示任意次数，包括在流控制内。
- 如果未声明流，则必须使用 发出 而不是 **emit_stream**。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
			X		

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

emitThenCut_stream (sm5 - asm)

项目 • 2022/09/22

等效于 发出 命令，后跟 剪切 命令。

emitThenCut_stream streamIndex

项	说明
streamIndex	[in]流索引。

备注

当故意将最后一个顶点输出到拓扑中时，此操作非常有用。

如果未声明流，则必须使用 emitThenCut 而不是 **emitThenCut_stream**。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
			X		

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

f16tof32 (sm5 - asm)

项目 • 2023/03/09

分量 float16 到 float32 的转换。

f16tof32 dest[.mask]， [-]src0[.swizzle]

项	说明
dest	[in]float32 结果的地址。
src0	[in]要转换的 float16 值。

备注

此操作执行 float16 值从 LSB 位到 float32 结果的分量转换。

此操作遵循浮点转换的 D3D 规则。

使用此说明进行着色器驱动的数据解压缩。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

f32tof16 (sm5 - asm)

项目 • 2023/03/09

分量型 float16 到 float32 的转换。

f32tof16 dest[.mask]， [-]src0[.swizzle]

项	说明
dest	[in]float16 结果的地址。
src0	[in]要转换的 float32 值。

备注

此指令执行 float32 值到放置在 LSB 16 位中的 float16 值结果的组件级转换。

此指令遵循浮点转换的 D3D 规则。

使用此说明进行着色器驱动的数据压缩。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

fcall (sm5 - asm)

项目 • 2022/09/23

接口函数调用。

fcall fp#[arrayIndex][callSite]

项	说明
Fp#	[in]函数指针。
arrayIndex	[in] 可选。 指定函数指针数组的偏移量。 如果 fp# 未声明为可索引，则此参数必须是文本无符号整数。 否则，arrayIndex 可以是着色器寄存器的表单文本基数 + 偏移量。 例如，fcall fp1[r1.w + 0][0] 。
callSite	[in] 可选。 选定函数表中的文本无符号整数偏移量，选择要执行的函数正文 fb# 。

注解

fp#[*arrayIndex*][] 解析为从 *fp#* 声明中列出的函数表选项之外的 API 选择的特定函数表。

fp# 中的 #和 *arrayIndex* 的总和选择函数表。 例如，如果接口声明为 fp4[4][3] (数组大小为 4) ，则以下 **fcall** fp4[2][3] 和 fp5[1][3]，因为 4+2 = 5+1。

限制

- 如果 *arrayIndex* 使用动态索引，则如果 *arrayIndex* 在相邻着色器调用上发生分歧（可能在锁定步骤中执行），则行为是未定义的。 HLSL 编译器将尝试禁止这种情况。

  由于流控制，相邻调用可能处于非活动状态，因为它不会中断锁定执行。

- 如果 *fp# + arrayIndex* 指定边界外索引，则行为是未定义的。
- 对于此处所述的未定义情况，这意味着当前 D3D 设备的行为变得未定义，包括设备丢失的可能性。 但是，当前 D3D 设备之外不会访问或执行任何内存作为代码。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

firstbit (sm5 - asm)

项目 • 2022/09/24

从 LSB 或 MSB 查找数字中的第一个位集。

firstbit{_hi|_lo|_shi} dest[.mask]， src0[.swizzle]

项	说明
dest	[in] *src0* 中第一个位集的整数位置，从 LSB 开始firstbit_lo或 MSB 进行firstbit_hi。
src0	[in]输入整数。

备注

此操作返回从 LSB 开始firstbit_lo的 32 位输入中设置的第一位的整数位置，或从 firstbit_hi 的 MSB 开始。 例如，0x00000001上的firstbit_lo返回 0。 0x10000000上的firstbit_hi返回 3。

如果数字为负数，则firstbit_shi (表示已签名) 返回 MSB 的前 0 个;否则，它将从 MSB 返回前 1 个。

如果找不到匹配项，则指令的所有变体在 32 位) 寄存器中返回 ~0 (0xffffffff。

使用此指令快速枚举位字段中的设置位，或查找数字中 2 的最大功率。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

Mimimum 着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ftod (sm5 - asm)

项目 • 2023/03/09

按组件从单精度浮点数据转换为双精度浮点数据。

ftod dest[.mask]， [-]src0[.swizzle]，

项	说明
dest	[in]转换的数据的地址。
src0	[in]要转换的数据。

备注

源的每个组件都从单精度表示形式转换为双精度表示形式。

有效的 *dest* 掩码为 .xy、.zw 和 .xyzw。 .xy 接收第一次转换的结果，.zw 接收第二次转换的结果。

dest 是跨 (x 32LSB、y 32MSB) 和 (z 32LSB、w 32MSB) 的双 vec2。

src0 是跨 x 和 y (zw 的 float vec2，) (重排后) 忽略。

对于 float32< 双>精度转换，实现可以遵循 float32 非函数，也可以刷新它们。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

gather4 (sm5 - asm)

项目 • 2022/09/22

收集将在双线性筛选操作中使用的四个纹素，并将其打包到单个寄存器中。 此指令仅适用于 2D 或 CubeMap 纹理，包括数组。 仅使用采样器的寻址模式，并使用任何 mip 金字塔的顶层。

gather4[_aoffimmi (u，v)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]， srcSampler[.select_component]

项	描述
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。
srcResource	[in]纹理寄存器。
srcSampler	[in]采样器寄存器。

注解

此指令的行为类似于 **示例** 指令，但不会生成筛选的示例。 有助于筛选的四个样本按逆时针顺序放置在 xyzw 中，从样本到查询位置左下角开始。 这与以下位置的 (u、v) 纹理坐标增量的点采样相同：(、 +) 、 (+、 +) 、 (+、 -) 、 (-,-) ，其中增量的大小始终为半纹素。

对于 CubeMap 纹理，当双线性占用跨越边缘时，将使用邻近人脸中的纹素。 角使用与 **示例** 指令相同的规则;也就是说，未知的角被认为是三个击中人脸角的平均值。

有一些纹理格式限制适用于以格式列表表示的 **gather4** 。

srcResource 上的重排允许在将返回的值写入目标之前任意重排。

srcSampler 上的.select_component选择源纹理的哪个组件 (r/g/b/a) 读取 4 个纹素。

对于具有 float32 组件的格式，如果提取的值已规范化、非规范化、 +-0 或 +-INF，则会将其返回到着色器未更改。 NaN 作为 NaN 返回，但 NaN 的确切位表示形式可能会更改。 对于 TextureCubes，缺少的第 4 个纹素的一些合成必须在角处发生，因此合成纹素的返回位不变，并且可以刷新非序数。

对于硬件实现，传统双线性筛选中的优化可以直接检测纹素上的样本，并跳过具有权重 0 的纹素的读取，无法通过本指令来利用。 *gather4* 始终返回所有请求的纹素。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

gather4_c (sm5 - asm)

项目 • 2023/03/09

与 gather4 相同，只不过此干扰对纹素执行比较，类似于 sample_c。

gather4_c[_aoffimmi (u，v)] dest[.mask]， srcAddress[.swizzle]， srcResource[.swizzle]， srcSampler[.r]， srcReferenceValue

项	说明
dest	[in]操作结果的地址
srcAddress	[in]一组纹理坐标。
srcResource	[in]纹理寄存器。
srcSampler	[in]采样器寄存器。
srcReferenceValue	[in]选中了单个组件的寄存器，用于比较。

备注

有关 *srcReferenceValue* 如何与每个提取的纹素进行比较的说明，请参阅sample_c。 与 **sample_c**不同， **gather4_c** 返回每个比较结果，而不是筛选结果。

对于 TextureCube 角，其中有三个实际纹素，第四个必须合成，合成必须在比较步骤之后进行。 这意味着返回的联合纹素比较结果可以是 0、0.33、0.66 或 1。 对于合成纹素，某些实现可能只返回 0 或 1。 除了此可能的结果列表外，未指定合成纹素的方法。

对于具有 float32 分量的格式，如果要提取的值已规范化或 +-INF，则会在未更改的比较操作中使用该值。 NaN 在比较运算中用作 NaN，但 NaN 的确切位表示形式可能会更改。 在比较中，将 Denorms 刷新为零。 对于 TextureCubes，缺少的第 4 个纹素的一些合成必须在角处发生，因此对于合成纹素返回未更改的位的概念不适用。

*gather4_c*支持的格式与*sample_c*支持的格式相同。 这些是单组件格式，因此为 。r on *srcSampler*，而不是任意重排。 **未** 绑定资源上的gather4_c返回 0。

使用此指令进行自定义阴影贴图筛选。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

gather4_po (sm5 - asm)

项目 • 2022/09/22

集合 4 的变体，但不支持即时偏移 [-8..7]，偏移量是指令的参数，而且具有更大的 [-32..31] 范围。

gather4_po dest[.mask]， srcAddress[.swizzle]， srcOffset[.swizzle]， srcResource[.swizzle]， srcSampler[.select_component]

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。
srcOffset	[in]偏移量。
srcResource	[in]纹理寄存器。
srcSampler	[in]采样器寄存器。

注解

4 向量偏移参数的前两个组件提供 32 位整数偏移量。 忽略此参数的其他组件。

每个偏移值的 6 个最小有效位被视为有符号值，从而产生 [-32..31] 范围。

此指令仅适用于 2D 纹理，与 **gather4** 不同，后者也适用于 TextureCubes。

采样器中唯一遵循的模式是寻址模式。 仅使用资源视图中最详细的 mip。

如果地址位于纹素中心，这并不意味着其他纹素可以零除。

srcSampler 参数包括 [.select_component]，允许检索纹理的任何单个组件，包括返回缺失组件的默认值。

对于具有 float32 组件的格式，如果提取的值已规范化、非规范化、+-0 或 +-INF，则会将其返回到着色器未更改。 NaN 作为 NaN 返回，但 NaN 的确切位表示形式可能会更改。 对于 TextureCubes，缺少的第 4 个纹素的一些合成必须在角落发生，因此合成纹素返回位的概念不适用，并且可刷新非序值。

使用此指令将 **收集** 范围的偏移量范围扩展到更大且可编程。 名称上的“po”后缀表示“可编程偏移量”。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

gather4_po_c (sm5 - asm)

项目 • 2023/03/09

行为与 gather4_po相同，只是对纹素执行比较，类似于 sample_c。

gather4_po_c dest[.mask]， srcAddress[.swizzle]， srcOffset[.swizzle]，srcResource[.swizzle]， srcSampler[.r]， srcReferenceValue

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]一组纹理坐标。
srcOffset	[in]偏移量。
srcResource	[in]纹理寄存器。
srcSampler	[in]采样器寄存器。
srcReferenceValue	[in]选定的单个组件。

备注

有关如何将 *srcReferenceValue* 与每个提取的纹素进行比较的信息，请参阅sample_c。 与 **sample_c**不同， *gather4_po_c* 返回每个比较结果，而不是筛选结果。

与 **gather4_po** 一样，此指令仅适用于 2D 纹理。 这与 gather4_c 不同，后者也适用于 TextureCubes。

对于具有 float32 分量的格式，如果要提取的值已规范化或 +-INF，则会在未更改的比较操作中使用该值。 NaN 在比较操作中用作 NaN，但 NaN 的确切位表示形式可能会更改。 在比较中，将 Denorms 刷新为零。 对于 TextureCubes，缺失的第 4 个纹素的一些合成必须在角处进行，因此返回合成纹素不变的位的概念不适用。

gather4_po_c 支持的格式与**sample_c**支持的格式相同。 这些是单组件格式，因此 为 。R on srcSampler，而不是任意重排。

未 绑定资源的gather4_po_c返回 0。

使用此方法筛选阴影贴图。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

hs_control_point_phase (sm5 - asm)

项目 • 2022/09/22

在外壳着色器中"开始"菜单控制点阶段。

hs_control_point_phase

注解

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

hs_decls (sm5 - asm)

项目 • 2022/12/14

"开始"菜单外壳着色器中的声明阶段。

hs_decls

备注

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

hs_fork_phase (sm5 - asm)

项目 • 2022/09/22

在外壳着色器中"开始"菜单分支阶段。

hs_fork_phase

注解

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

hs_join_phase (sm5 - asm)

项目 • 2022/09/23

在外壳着色器中"开始"菜单联接阶段。

hs_join_phase

注解

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
	X				

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ibfe (sm5 - asm)

项目 • 2022/09/22

给定数字中的位范围，将这些位移到 LSB，并标记扩展该范围的 MSB。

ibfe dest[.mask]， src0[.swizzle]， src1[.swizzle]， src2[.swizzle]

项	说明
dest	[in]操作结果的地址。
src0	[in]位域宽度。
src1	[in]位字段偏移量。
src2	[in]要移动的值。

注解

LSB 5 位 src0 提供位字段宽度，（0-31）。

src1 的 LSB 5 位提供位字段偏移量，（0-31）。

以下示例演示如何使用此指令。

syntax

```
        Given width, offset:
                if( width == 0 )
                {
                    dest = 0
                }
                else if( width + offset < 32 )
                {
                    shl dest, src2, 32-(width+offset)
                    ishr dest, dest, 32-width
                }
                else
                {
                    ishr dest, src2, offset
                }
```

使用此指令解压缩带符号整数或标志。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_alloc (sm5 - asm)

项目 • 2023/03/09

使用 Count 或 Append 无序访问视图 (UAV) 以原子方式递增存储的隐藏 32 位计数器，返回原始值。

imm_atomic_alloc dest[.single_component_mask]， dstUAV

项	说明
dest	[in]包含返回的计数器值。
dstUAV	[in]具有 Count 或 Append 标志的结构化缓冲区 UAV。

备注

每个 Count 或 Append Buffer 视图都有一个隐藏的无符号 32 位整数计数器值，该值在视图绑定到管道时初始化，包括用于保留上一个值的选项。

此指令对计数器值执行原子递增，将原始值返回到 *dest*。

对于追加 UAV，返回的值仅在着色器调用期间有效。 之后，实现可能会重新排列内存布局。 基于返回值的任何内存寻址必须限制为着色器调用。

对于 Append UAV，在着色器调用中，HLSL 编译器可以使用返回的值作为用于访问结构化缓冲区的结构索引。 访问除调用 () **imm_atomic_alloc或**_consume调用返回的那些位置以外的任何结构索引会产生未定义的结果，即 UAV 中访问的内存位置是随机的，并且仅在着色器调用的生存期内固定。

对于 Count UAV，应用程序可以将返回的值保存为对 UAV 中某个固定位置的引用，该位置在着色器调用结束后有意义。 始终可以访问计数 UAV 中的任何位置，而不管计数值是什么。

没有计数的固定，因此它会在溢出时进行包装。

同一着色器不能尝试 **在同一** UAV 上imm_atomic_alloc和 **imm_atomic_consume** 。 此外，GPU 不能允许多个着色器调用在同一 UAV 上混合 **imm_atomic_alloc** 和 **imm_atomic_consume** 。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从Windows 8开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_and (sm5 - asm)

项目 • 2022/09/23

直接原子按位 AND 到内存。 返回 AND 之前的内存中的值。

imm_atomic_and dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	说明
dst0	[in]包含 AND 之前的 *dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]目标内存。
src0	使用 *dst* 的 AND 值。

注解

此指令执行单个分量 32 位位 AND 操作数 *src0*，每个组件地址 *dstAddress* 的 *dst1* 为 32 位。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

在 AND 返回到 *dst0* 之前 *dst1* 内存中的值。

整个操作以原子方式执行。

从地址中获取的组件数取决于 *在 dst1* 中声明的资源的维数。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生体的实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

在 u# 上寻址的边界外，不会将任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的整个内容将变为未定义。

u# 或 g# 上的边界外会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此说明适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始可用。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_cmp_exch (sm5 - asm)

项目 • 2023/03/09

立即比较并交换内存。

imm_atomic_cmp_exch dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]， src1[.select_component]

项	说明
dst0	[out]写入前包含 *dst1* 。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，这也可以是线程组共享内存 (g#) 。
dstAddress	[in]目标内存。
src0	[in]要与 *dst1* 进行比较的值。
src1	[in]如果比较值相同，则写入目标内存的值。

此指令执行操作数 *src0* 与 *dst1* 的单分量 32 位值比较，每个组件地址 *dstAddress* 为 32 位。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果已键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

如果比较值相同，则 *src1* 中的单分量 32 位值将写入目标内存。 否则，目标内存不会更改。

目标内存中的原始 32 位值始终写入 *dst0*。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果像素在执行之前已被丢弃，或者像素/样本调用仅存在以充当派生体的实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的越界寻址会导致不会将任何内容写入内存，除非 u# 是结构化的，并且结构 (地址) 的第二个组件中的字节偏移导致越界访问，则 UAV 的整个内容将变得未定义。

u# 或 g# 上的越界寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_consume (sm5 - asm)

项目 • 2023/03/09

使用“计数”或“追加”无序访问视图 (UAV) 以原子方式递减存储的隐藏 32 位计数器，返回新值。

imm_atomic_consume dest[.single_component_mask]， dstUAV

项	说明
dest	[in]包含返回的原始计数器值。
dstUAV	[in]具有 Count 或 Append 标志的结构化缓冲区 UAV。

备注

有关返回的计数值的有效性的讨论，请参阅 imm_atomic_alloc ，具体取决于 UAV 是 Count 还是 Append。 这同样适用于 **imm_atomic_consume**。

imm_atomic_consume 对计数器值执行原子递减，并将新值返回到 *dest*。

没有计数的固定，所以它包在下溢。

同一着色器不能尝试 **在同一** UAV 上imm_atomic_alloc和 **imm_atomic_consume** 。 此外，GPU 不能允许多个着色器调用在同一 UAV 上混合 **imm_atomic_alloc** 和 **imm_atomic_consume** 。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从Windows 8开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_exch (sm5 - asm)

项目 • 2022/09/22

立即将原子交换到内存。

imm_atomic_exch dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	说明
dst0	[in]包含写入前 *dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要写入 *dstAddress* 处*的 dst1* 的值。

注解

此指令在每个组件地址 *dstAddress* 处执行操作数 *src0* 到 *dst1* 的单个组件 32 位值写入 dst1。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

从地址获取的组件数取决于在 *dst1* 中声明的资源的维度。

目标内存中的原始 32 位值写入 *dst0*。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生派生实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_iadd (sm5 - asm)

项目 • 2022/09/23

立即将原子整数添加到内存中。 返回在添加之前内存中的值。

imm_atomic_iadd dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	描述
dst0	[in]包含写入前 *的 dst1* 中的值。
dst1	此值必须是 UAV () (u#) 的无序访问视图。 在计算着色器中，它也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存裸体。
src0	[in]要添加到 *dst1* 的值。

注解

此指令执行单个分量 32 位整数添加操作数 *src0*，每个组件地址 *dstAddress* 为 32 位的 *dst1*。 签名不敏感。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

添加到 *dst0* 之前，*dst1* 内存中的值。

从地址获取的组件数由 *dst1* 的维数决定。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生派生实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_imax (sm5 - asm)

项目 • 2022/09/23

立即将原子签名的最大值设置为内存。 在最大操作之前返回内存中的值。

imm_atomic_imax dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	说明
dst0	[in]包含此指令之前的 *dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dstAddress* 中的 *dst1* 进行比较的值。

注解

此指令执行单个组件 32 位带符号的最大操作数 *src0*，每个组件地址 *dstAddress* 的 *dst1* 为 32 位。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

在 max 返回到 *dst0* 之前，*dst1* 内存中的值。

从地址获取的组件数由 *dst1* 的维数决定。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生派生实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_imin (sm5 - asm)

项目 • 2022/09/22

立即将原子签名的最小值写入内存。 在最大操作之前返回内存中的值。

imm_atomic_imin dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	描述
dst0	[in]包含此指令之前 *dst1* 中的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dstAddress* 中的 *dst1* 进行比较的值。

注解

Thisi 指令执行单个组件 32 位带符号的最小操作数 *src0*，每个组件地址 *dstAddress* 的 *dst1* 为 32 位。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果键入，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

将最小值返回到 *dst0* 之前，*dst1* 内存中的值。

从地址中提取的组件数由 *dst1* 的维数决定。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生体的实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

在 u# 上寻址的边界外，不会将任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的整个内容将变为未定义。

u# 或 g# 上的边界外会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此说明适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始可用。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_or (sm5 - asm)

项目 • 2022/09/22

直接原子按位 OR 到内存。 返回 OR 之前内存中的值。

imm_atomic_or dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	说明
dst0	[in]包含 OR 之前的 *dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	具有 *dst1* 的 OR 值。

备注

此指令执行单个分量 32 位或操作数 *src0*，每个组件地址 *dstAddress* 为 32 位的 *dst1*。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

OR 返回到 *dst0* 之前 *dst1* 内存中的值。

整个操作以原子方式执行。

从地址获取的组件数取决于在 *dst1* 中声明的资源的维度。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生派生实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_umax (sm5 - asm)

项目 • 2022/09/22

即时原子无符号最大值到内存。 在最大操作之前返回内存中的值。

imm_atomic_umax dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	描述
dst0	[in]包含此指令之前的 *dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dstAddress* 中的 *dst1* 进行比较的值。

注解

此指令执行单个组件 32 位无符号最大操作数 *src0*，每个组件地址 *dstAddress* 的 *dst1* 为 32 位。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

在 max 返回到 *dst0* 之前，*dst1* 内存中的值。

从地址获取的组件数由 *dst1* 的维数决定。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生派生实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_umin (sm5 - asm)

项目 • 2022/09/22

立即将原子无符号最小值保存到内存中。 在最大操作之前返回内存中的值。

imm_atomic_umin dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项目	说明
dst0	[in]包含此指令之前的 *dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	[in]要与 *dstAddress* 中的 *dst1* 进行比较的值。

注解

此指令执行单个组件 32 位无符号最小操作数 *src0*，每个组件地址 *dstAddress* 为 32 位，为 *dst1*。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

将最小值返回到 *dst0* 之前 *dst1* 内存中的值。

从地址获取的组件数由 *dst1* 的维数决定。

整个操作以原子方式执行。

如果着色器调用处于非活动状态，例如，如果在执行之前放弃了像素，或者像素/样本调用仅用作派生体实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

imm_atomic_xor (sm5 - asm)

项目 • 2022/09/22

将原子按位 XOR 立即写入内存。 返回 XOR 之前的内存中的值。

imm_atomic_xor dst0[.single_component_mask]， dst1， dstAddress[.swizzle]， src0[.select_component]

项	说明
dst0	[in]包含 *XOR 之前的 dst1* 的值。
dst1	[in]UAV) (u#) (无序访问视图。 在计算着色器中，也可以是线程组共享内存 (g#) 。
dstAddress	[in]内存地址。
src0	值为 *dst1* 的 XOR。

注解

此指令执行单个组件 32 位位 XOR 操作数 *src0*，每个组件地址 *dstAddress 的 dst1 为 32* 位。

如果 *dst1* 是 u#，则它可能已声明为原始、类型化或结构化。 如果类型化，则必须将其声明为 UINT/SINT，绑定资源格式为 R32_UINT/_SINT。

如果 *dst1* 为 g#，则必须将其声明为原始或结构化。

在 XOR 返回到 *dst0* 之前，*dst1* 内存中的值。

整个操作以原子方式执行。

从地址获取的组件数取决于在 *dst1* 中声明的资源的维度。

如果着色器调用处于非活动状态，如果像素在执行之前被丢弃，或者像素/样本调用仅作为派生派生的实际像素/样本的帮助程序，则此指令根本不会更改 *dst1* 内存，并且返回的值未定义。

u# 上的边界外寻址会导致无任何内容写入内存，除非 u# 已结构化，并且字节偏移到地址 (第二个组件的结构) 导致边界外访问，则 UAV 的全部内容将变为未定义。

u# 或 g# 上的边界外寻址会导致未定义的结果返回到 *dst0* 中的着色器。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此指令适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ishl (sm5 - asm)

项目 • 2022/09/22

左移。

ishl dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]包含班次的结果。
src0	[in]要移动的 32 位值。
src1	[in]要移动的位数。

注解

此指令执行 *src0* 中每个 32 位值的分量换行，LSB 5 位提供的无符号整数位计数 (*src1* 中的 0-31 范围) ，插入 0。 每个组件结果的 32 位结果放置在 *dest* 中。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ishr (sm5 - asm)

项目 • 2023/03/09

算术右移 (符号扩展) 。

ishr dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]包含移位的结果。
src0	[in] 要位移的位数。
src1	[in]要移动的 32 位值。

备注

此指令通过 *src1* 中 LSB 5 位 (0-31) 范围提供的无符号整数位计数对 *src0* 右侧的每个 32 位值执行分量算术移位，复制位 31 的值。 每个分量 32 位的结果放置在 *dest* 中。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ld_raw (sm5 - asm)

项目 • 2023/03/09

从原始缓冲区随机读取 1-4 32 位组件。

ld_raw dest[.mask]、srcByteOffset[.select_component]、src0[.swizzle]

项	说明
dest	[in]操作结果的地址。
srcByteOffset	[in]指定要从中读取的偏移量。
src0	[in]要读取的组件。

备注

src0 必须是：

- 任何着色器阶段：SRV (t#) ld st
- 计算着色器或像素着色器：UAV (u#)
- 计算着色器：线程组共享内存 (g#)

srcByteOffset 为可以读取数据的 4 个连续 32 位值的窗口指定内存中的基本 32 位值，具体取决于其他参数的重排和掩码。

从原始缓冲区读取的数据等效于以下伪代码：其中，我们具有偏移量、地址、指向缓冲区内容的指针、源的步长以及以线性方式存储的数据。

syntax

```
                    BYTE *BufferContents;         // from src0
                    UINT srcByteOffset;           // from srcByteOffset
                    BYTE *ReadLocation;           // value to calculate
                    ReadLocation = BufferContents
                                + srcByteOffset;

                    UINT32 Temp[4];  // used to make code shorter

                    // apply the source resource swizzle on source data
                    Temp = read_and_swizzle(ReadLocation, srcSwizzle);

                    // write the components to the output based on mask
                    ApplyWriteMask(dstRegister, dstWriteMask, Temp);
```

任何给定 32 位组件的 u#/t# 上的边界寻址将返回该组件的 0。

g# 上的边界寻址 (该特定 g# 的边界，而不是任何给定 32 位组件的所有共享内存) 返回未定义的结果。

cs_4_0和cs_4_1支持适用于 UAV 和 SRV 的此说明。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本说明适用于 Direct3D 11.1 运行时的 UAV 的所有着色器阶段，从Windows 8开始提供。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

cs_4_0和cs_4_1支持适用于 UAV 和 SRV 的此说明。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ld_structured (sm5 - asm)

项目 • 2023/03/09

从结构化缓冲区随机读取 1-4 32 位组件。

ld_structured dest[.mask]、srcAddress[.select_component]、srcByteOffset[.select_component]、src0[.swizzle]

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]指定要读取的结构的索引。
srcByteOffset	[in]指定要从其开始读取的结构中的字节偏移量。
src0	要从中读取的缓冲区。 此参数必须是 SRV (t#) 、UAV (u#) 。 在计算着色器中，它也可以是线程组共享内存 (g#) 。

备注

从 结构中读取的数据等效于以下伪代码：其中，我们有偏移量、地址、指向缓冲区内容的指针、源的步长以及以线性方式存储的数据。

syntax

```
                    BYTE *BufferContents;         // from SRV or UAV
                    UINT BufferStride;            // from base resource
                    UINT srcAddress, srcByteOffset;   // from source
registers
                    BYTE *ReadLocation;           // value to calculate
                    ReadLocation = BufferContents
                                + BufferStride * srcAddress
                                + srcByteOffset;

                    UINT32 Temp[4];  // used to make code shorter

                    // apply the source resource swizzle on source data
                    Temp = read_and_swizzle(ReadLocation, srcSwizzle);

                    // write the components to the output based on mask
                    ApplyWriteMask(dstRegister, dstWriteMask, Temp);
```

此伪代码显示操作的工作原理，但实际数据不必以线性方式存储。 如果数据不是以线性方式存储的，则指令的实际操作需要与上述操作的行为相匹配。

任何给定 32 位组件的 u#/t# 上的边界寻址为该组件返回 0，除非 *srcByteOffset*，加上重排是导致对 you#/t# 进行边界访问的原因，则所有组件 (s) 的返回值均未定义。

g# 上的边界寻址 (该特定 g# 的边界，而不是任何给定 32 位组件的所有共享内存) 返回未定义的结果。

srcByteOffset 是 *与 srcAddress* 不同的参数，因为它通常是一个文本。 尚未对结构化内存上的原子执行此参数分离。

cs_4_0和cs_4_1支持 UAV、SRV 和 TGSM 的此说明。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本说明适用于 Direct3D 11.1 运行时的 UAV 的所有着色器阶段，从Windows 8开始提供。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

cs_4_0和cs_4_1支持 UAV、SRV 和 TGSM 的此说明。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ld_uav_typed (sm5 - asm)

项目 • 2023/03/09

从类型化无序访问视图中随机访问读取元素 (UAV) 。

ld_uav_typed dest[.mask]， srcAddress[.swizzle]， srcUAV[.swizzle]

项	说明
dest	[in]操作结果的地址。
srcAddress	[in]指定要从中读取的地址。
srcUAV	[in]要从中读取的源。

备注

此指令在 *srcAddress* 中的无符号整数地址处执行从 *srcUAV* 读取的 4 分量元素，根据格式转换为每个组件 32 位，然后写入着色器中的 *dest*。

srcUAV 是声明为类型化 (u#) 的 UAV。 但是，绑定资源的类型必须为 R32_UINT/SINT/FLOAT。

从地址获取的 32 位无符号整数分量的数目由 *srcUAV* 处声明的资源的维数决定。 寻址与 ld 指令相同。

边界外寻址与 **ld** 指令相同。

如果调用为 ld **dest[.mask]、srcAddress[.swizzle]、srcUAV[.swizzle]** ，则此指令的行为与 **ld** 指令相同。

在未声明为类型化 UAV 上使用此指令是无效且未定义的。 在结构化或无类型 UAV 上执行此操作无效。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

cs_4_0和cs_4_1针对 UAV、SRV 和 TGSM 支持此说明。

相关主题

着色器模型 5 程序集 (DirectX HLSL)

rcp (sm5 - asm)

项目 • 2022/09/24

分量倒数。

rcp[_sat] dest[.mask]， [-]src0[_abs][.swizzle]

项	说明
dest	[in]结果的地址 *dest* = **1.0f** / *src0*.
src0	[in]要采用倒数的数字。

注解

使用此指令可降低精度对等性，而与划分的严格要求无关。

最大相对误差为 2-21。 (容错只与 rsq) 匹配

下表显示了使用各种数字类执行指令时获取的结果。

src	**-inf**	**-F**	**-denorm**	**-0**	**+0**	**+denorm**	**+F**	**+inf**	**NaN**
dest	-0	-F	-inf	-inf	+inf	+inf	+ F	+0	NaN

此说明适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

store_raw (sm5 - asm)

项目 • 2023/03/09

将 1-4 32 位组件随机访问写入无类型内存。

store_raw dest[.write_mask]、 dstByteOffset[.select_component]、 src0[.swizzle]

项	说明
dest	[in]内存地址。
dstByteOffset	[in]偏移量。
src0	[in]要写入的组件。

备注

此指令在 *dstByteOffset* 中的偏移量处执行从 *src0* 写入到 *dest* 的 1-4 分量 *32 位组件。无需格式转换。

dest 必须是 UAV (u#) ，或者在计算着色器中，它也可以是线程组共享内存 (g#) 。

dstByteOffset 为 4 个连续 32 位值的窗口（根据其他参数的重排和掩码）指定内存中的基本 32 位值。

写入的数据的位置等效于以下伪代码，该伪代码显示地址、指向缓冲区内容的指针以及线性存储的数据。

syntax

```
BYTE *BufferContents;          // from src0
UINT dstByteOffset;            // source register
BYTE *WriteLocation;           // value to calculate

// calculate writing location
WriteLocation = BufferContents
            + dstByteOffset;

// calculate the number of components to write
switch (dstWriteMask)
{
    x:    WriteComponents = 1; break;
    xy:   WriteComponents = 2; break;
    xyz:  WriteComponents = 3; break;
    xyzw: WriteComponents = 4; break;
    default:  // only these masks are valid
```

```
                    }

                    // copy the data from the source register with
                    //    the swizzle applied
                    memcpy(WriteLocation, swizzle(src0, src0.swizzle),
                             WriteComponents * sizeof(UINT32));
```

此伪代码显示操作的工作方式，但实际数据不必以线性方式存储。 *dest* 只能具有以下项之一的写入掩码：.x、.xy、.xyz、.xyzw。 写入掩码确定要写入且没有间隔的 32 位组件的数量。

u# 上的越界寻址意味着不会向边界外内存写入任何内容;边界内的任何部分都正确写入。

g# 上的边界外寻址 (该特定 g# 的边界，而不是任何给定 32 位组件的所有共享内存) 会导致所有共享内存的整个内容变得未定义。

cs_4_0和cs_4_1支持此 UAV 指令。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

store_structured (sm5 - asm)

项目 • 2023/03/09

将 1-4 个 32 位组件随机访问写入结构化缓冲区无序访问视图，(UAV) 。

store_structured dest[.write_mask]、 dstAddress[.select_component]、 dstByteOffset[.select_component]、 src0[.swizzle]

项	说明
dest	[in]操作结果的地址。
dstAddress	[in]要写入的地址。
dstByteOffset	[in]要写入的 结构的索引。
src0	[in]要写入的组件。

备注

此指令在 *dstAddress* 和 dstByteOffset 中的地址处执行从 *src0* 写入到 *dest* 的 1-4 组件 *32 位组件。 无格式转换。

dest 必须是 UAV (u#) 。 在计算着色器中，也可以是线程组共享内存 (g#) 。

dstAddress 指定要写入的结构的索引。

写入的数据的位置等效于以下伪代码，其中显示了偏移量、地址、指向缓冲区内容的指针、源的步幅以及线性存储的数据。

syntax

```
                    BYTE *BufferContents;             // from dest
                    UINT BufferStride;                // from dest
                    UINT dstAddress, dstByteOffset;   // source registers
                    BYTE *WriteLocation;              // value to calculate

                    // calculate writing location
                     WriteLocation = BufferContents
                                + BufferStride * dstAddress
                                + dstByteOffset;

                    // calculate the number of components to write
                    switch (dstWriteMask)
                    {
                        x:    WriteComponents = 1; break;
                        xy:   WriteComponents = 2; break;
```

```
                        xyz:  WriteComponents = 3; break;
                        xyzw: WriteComponents = 4; break;
                        default:  // only these masks are valid
                    }

                    // copy the data from the source register with
                    //    the swizzle applied
                    memcpy(WriteLocation, swizzle(src0, src0.swizzle),
                             WriteComponents * sizeof(INT32));
```

此伪代码显示操作的工作方式，但实际数据不必以线性方式存储。 如果数据不是线性存储的，则指令的实际操作需要与上述操作的行为相匹配。

dest 只能具有以下项之一的写入掩码：.x、.xy、.xyz、.xyzw。 写入掩码确定要写入且没有间隔的 32 位组件的数量。

dstAddress 在 u# 上寻址的超出边界意味着不会向边界外内存写入任何内容。

如果 *dstByteOffset*（包括 *dstWriteMask*）导致对 you# 进行越界访问，则 UAV 的整个内容将变为未定义。

g# 上的边界外寻址 (该特定 g# 的边界，而不是任何给定 32 位组件的所有共享内存) 会导致所有共享内存的整个内容变得未定义。

dstByteOffset 是 *与 dstAddress* 不同的参数，因为它通常是一个文本。 尚未对结构化内存上的原子执行此参数分离。

cs_4_0和cs_4_1支持 UAV 和 TGSM 的此说明。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
				X	X

由于 UAV 可用于 Direct3D 11.1 的所有着色器阶段，因此本指令适用于 Direct3D 11.1 运行时的所有着色器阶段，该阶段从 Windows 8 开始可用。

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

store_uav_typed (sm5 - asm)

项目 • 2022/09/22

将元素随机访问写入类型化非排序访问视图， (UAV) 。

store_uav_typed dstUAV.xyzw， dstAddress[.swizzle]， src0[.swizzle]

项	说明
dstUAV	[in]包含操作的结果。
dstAddress	[in]要写入的地址。
src0	[in]要写入的组件。

注解

此指令在 *dstAddress* 的地址处执行从 *src0* 到 *dstUAV* 写入的 4 个组件 *32 位元素。 *dstUAV* 是类型化的 UAV (u#) 。

UAV 的格式决定了格式转换。

从地址中提取的 32 位无符号整数组件的数目由 *在 dstUAV* 声明的资源的维数确定。 此地址位于元素中。

超出边界寻址意味着不会写入内存。

dstUAV 始终具有 .xyzw 写入掩码。 必须写入所有组件。

在未声明为类型化的 UAV 上使用此指令无效且未定义。 也就是说，在结构化或无类型 UAV 上执行此操作是无效的。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 UAV 在 Direct3D 11.1 的所有着色器阶段都可用，因此此说明适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始可用。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此说明：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

swapc (sm5 - asm)

项目 • 2023/03/09

在两个输入寄存器之间对值执行按组件条件交换。

swapc dst0[.mask]， dst1[.mask]， src0[.swizzle]， src1[.swizzle]， src2[.swizzle]

项	说明
dst0	[in]使用任意非空写入掩码进行注册。 必须与 *dst1* 不同。
dst1	[in]使用任意非空写入掩码进行注册。 必须与 *dst0* 不同。
src0	[in]提供 4 个条件。 非零整数值表示 **true**。
src1	[in]要交换的值之一。
src2	[in]要交换的值之一。

备注

此指令的编码尝试在两个 4 分量寄存器之间紧凑地表达标量的多个并行条件交换，在交换所涉及的数字对的排列方面具有较小的灵活性。

src0、*src1* 和 *src2* 的寄存器和值的选择不受任何限制，例如 movc。

此指令的语义可以通过 **使用** movc 指令的等效操作来描述。 以下示例中显示了最坏情况，确保目标寄存器直到结束才会更新。

syntax

```
swapc dst0[.mask],
      dst1[.mask],
      src0[.swizzle],
      src1[.swizzle],
      src2[.swizzle]

expands to:

movc temp[dst0 s mask],
     src0[.swizzle],
     src2[.swizzle], src1[.swizzle]

movc dst1[.mask],
     src0[.swizzle],
     src1[.swizzle], src2[.swizzle]
```

```
                mov  dst0.mask, temp
```

可以选择如何处理任务（如果不直接）。 例如，可以通过最多 4 个简单标量条件交换序列（或如上所述的两个向量 **movc** 指令）以及任何开销来实现相同的效果，以确保源值不会在扩展期间被早期操作占用。

使用此说明进行排序。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

同步 (sm5 - asm)

项目 • 2022/09/22

线程组同步或内存屏障。

sync[_uglobal|_ugroup][_g][_t]

备注

同步 具有选项_uglobal、_ugroup、_g和_t。

在像素着色器中，仅允许 **sync_uglobal** 。

在计算着色器中，必须指定 (_uglobal或_ugroup*) 和/或_g。 此外，_t是可选的。

_uglobal

全局 u# (UAV) 内存围栏。

在此线程访问任何后续 u# 内存之前，此线程按程序顺序读取/写入的所有 u# 内存都对整个 GPU 上的所有线程可见。 定义的整个 GPU 部分在一种情况下被小于全局的范围所取代，如下所述。

这适用于当前着色器阶段绑定的所有 UAV 内存。

计算着色器或像素着色器中提供了_uglobal。

对于着色器未声明为全局一致性的任何绑定 UAV，_uglobal u# 内存围栏仅具有该 UAV 的当前计算着色器线程组中的可见性，就好像它是_ugroup而不是_uglobal。 此问题仅适用于计算着色器，因为像素着色器必须将所有 UAV 声明为全局一致。

_ugroup

线程组范围 u# (UAV) 内存围栏。

在线程访问任何后续 u# 内存之前，此线程按程序顺序读取或写入的所有 u# 内存都对线程组中的所有线程可见。

这适用于当前着色器阶段绑定的所有 UAV 内存。

_ugroup仅在计算着色器中可用。

如果公开_ugroup，则对于某些实现。 指定_ugroup而不是_uglobal的优点是 **同步** 操作可以更快地完成。

其他实现不区分_ugroup与_uglobal，因此两个操作都是等效的，行为类似于_uglobal。 应用程序可以通过请求所需的最窄的 **同步** 范围来指定其意向。

即使特定 UAV 被声明为全局一致，如果不需要全局屏障，_ugroup **同步** 操作也会更有效地在该 UAV 上运行。

_g

g# (线程组共享内存) 围栏。

在线程访问任何后续 g# 内存之前，此线程按程序顺序读取或写入的所有 g# 内存都对线程组中的所有线程可见。

这适用于当前线程组的所有 g# 共享内存。

_g仅在计算着色器中可用。

_t

线程组同步。单个线程组中的所有线程 (可以共享对一组常见共享注册空间的访问权限) 将执行到它们到达此指令的点，然后任何线程才能继续。

_t不能放置在动态流控制中， (分支，这些分支可能在线程组) 内变化，但可以存在于统一流控制中，其中组中的所有线程都选取相同的路径。

_t仅在计算着色器中可用。

下面是计算着色器“sync”变体的列表。

- sync_g
- sync_ugroup*
- sync_uglobal
- sync_g_t
- sync_ugroup_t*
- sync_uglobal_t
- sync_ugroup_g*
- sync_uglobal_g
- sync_ugroup_g_t*
- sync_uglobal_g_t

*根据上述_ugroup部分中的前面讨论，具有_ugroup的变体可能不是 HLSL 编译器的目标。

仅包含sync_uglobal像素着色器同步变体的列表。

内存围栏可防止编译器或硬件在围栏上重新排序受影响的指令。

可以通过着色器调用从同一地址进行多次读取，这些调用不由内存屏障或写入地址分隔，可以折叠在一起。 这同样适用于写入。 无法合并或移动由屏障分隔的访问。

不同的线程对给定地址执行原子操作时，不需要内存围栏才能正常工作。 当原子和/或加载/存储操作需要相互同步时，需要隔离区，因为它们从其他线程的角度来看出现在各个线程中。

在像素着色器中， 放弃 指令意味着sync_uglobal围栏，在该指令中不能在 **放弃**之间重新排序。 sync_uglobal (仅支持派生) 或丢弃像素运行的帮助程序像素可能会有任何影响或可能没有任何影响。 如果放弃后发出写入操作，则不允许帮助程序或丢弃的像素写入 UAV。 不允许从 UAV 返回的值参与派生计算。 因此，无论sync_u是否适用于帮助程序像素，还是在放弃后发出时都是无问题的。

cs_4_0和cs_4_1支持此指令。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
				X	X

由于 DIRECT3D 11.1 的所有着色器阶段都提供 UAV，因此此指令**的sync_uglobal**变体适用于 Direct3D 11.1 运行时的所有着色器阶段，从 Windows 8 开始提供。

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

uaddc (sm5 - asm)

项目 • 2023/03/09

无符号整数与携带相加。

uaddc dst0[.mask]， dst1[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dst0	[in]结果的地址。
dst1	[in] 如果生成携带，则为 1。 否则为 0。
src0	[in] 要添加的 32 位操作数。
src1	[in] 要添加的 32 位操作数。

备注

如果不需要携带，*则 dst1* 可以为 NULL。

使用此指令进行高精度算术。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否

着色器模型	支持
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ubfe (sm5 - asm)

项目 • 2022/09/22

给定数字中的一系列位，将这些位移动到 LSB 并将剩余位设置为 0。

ubfe dest[.mask]， src0[.swizzle]， src1[.swizzle]， src2[.swizzle]

项	说明
dest	[in]包含指令的结果。
src0	[in]LSB 5 位提供位字段宽度 (0-31) 。
src1	[in] *src1* 的 LSB 5 位提供位字段偏移量 (0-31) 。
src2	[in]要移动的数字。

注解

syntax

```
        Given width, offset:
                if( width == 0 )
                {
                    dest = 0
                }
                else if( width + offset < 32 )
                {
                    shl dest, src2, 32-(width+offset)
                    ushr dest, dest, 32-width
                }
                else
                {
                    ushr dest, src2, offset
                }
```

使用此指令解压缩无符号整数或标志。

此说明适用于以下着色器阶段：

顶点	外壳	域	几何结构	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

ushr (sm5 - asm)

项目 • 2023/03/09

向右移动。

ushr dest[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dest	[in]包含指令的结果。
src0	[in]要移动的 32 位值。
src1	[in]LSB 5 位提供 (0-31) 移位数。

此指令通过 *src1* 中 LSB 5 位 (0-31 范围提供的无符号整数位计数，) 对 *src0* 右侧每个 32 位值执行组件级移动，插入 0。 每个组件 32 位的结果置于 *dest* 中。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

usubb (sm5 - asm)

项目 • 2023/03/09

使用 borrow 减去无符号整数。

usubb dst0[.mask]， dst1[.mask]， src0[.swizzle]， src1[.swizzle]

项	说明
dst0	[in]包含指令的 LSAB 结果。
dst1	[in] *dst0* 的相应组件，指定是否生成借款。
src0	[in]要从中减去的值。
src1	[in]要从 *src0* 中减去的量。

备注

此指令执行从 *src0* 中减去 32 位操作数 *src1* 的组件无符号运算，将 32 位结果的 LSB 部分置于 *dst0* 中。

如果生成借用， *则 dst1* 中的相应组件使用 1 编写，否则为 0。

如果不需要借款，*则 dst1* 可以为 NULL。

此指令适用于以下着色器阶段：

顶点	外壳	域	Geometry	像素	计算
X	X	X	X	X	X

最小着色器模型

以下着色器模型中支持此指令：

着色器模型	支持
着色器模型 5	是
着色器模型 4.1	否
着色器模型 4	否

着色器模型	支持
着色器模型 3 (DirectX HLSL)	否
着色器模型 2 (DirectX HLSL)	否
着色器模型 1 (DirectX HLSL)	否

相关主题

着色器模型 5 程序集 (DirectX HLSL)

软件着色器

项目 • 2022/09/22

软件着色器实现，允许在没有基础硬件支持的情况下开发着色器。 它们支持完整的功能集。 因为它们是在软件中实现的，因此它们不会产生最佳性能。

版本	功能集	要求
vs_2_sw	vs_2_x的所有功能	仅软件顶点处理和引用设备支持。
vs_3_sw	vs_3_0的所有功能	仅软件顶点处理和引用设备支持。
ps_2_sw	ps_2_x的所有功能	仅引用设备支持。
ps_3_sw	ps_3_0的所有功能	仅引用设备支持。

对于软件着色器，某些验证很轻松。 这对于调试和原型制作非常有用。 以下验证是宽松的： (所有其他验证保持不变)

验证类型	放松
指令计数：	对于vs_2_sw、vs_3_sw和ps_2_sw来说，这是放松的，ps_3_sw。 允许无限制的说明。
浮点常量计数：	对于vs_2_sw、vs_3_sw和ps_2_sw来说，这是放松的，ps_3_sw。 最多允许 8192 个常量。
整数常量计数：	对于vs_2_sw、vs_3_sw和ps_2_sw来说，这是放松的，ps_3_sw。 最多允许 2048 个常量。
布尔常量计数：	对于vs_2_sw、vs_3_sw和ps_2_sw来说，这是放松的，ps_3_sw。 最多允许 2048 个常量。
从属读取深度：	对于ps_2_sw来说，这是放松的。 与vs_3_0和ps_3_0一样，允许无限依赖读取。
流控制指令和标签数：	对于vs_2_sw来说，这是放松的。 允许无限流控制指令和最多 2048 个标签。
Loop count/start/step：	对于vs_2_sw、vs_3_sw、ps_2_sw和ps_3_sw来说，这些都是宽松的。 代表和循环指令的迭代开始和交错步骤大小为 32 位有符号交错器。 交错计数最多可以MAX_INT/64。
读取端口限制：	vs_2_sw、vs_3_sw、ps_2_sw和ps_3_sw没有读取端口限制。
内插器数：	vs_3_sw中有 16 个寄存器 - vs_3_0 (o#) ，10 ps_3_0 注册 (v#) for ps_3_sw。

相关主题

Asm 着色器参考

D3DCompiler 参考

项目 • 2022/09/22

Direct3D API 定义了多个 API 元素来编译着色器代码。

- 函数
- 结构
- 枚举
- 常量

相关主题

HLSL 参考

编译器函数 (HLSL 引用)

项目 • 2022/09/22

本部分包含有关以下 Direct3D HLSL 编译器函数的信息：

在本节中

主题	说明
D3D11Reflect	获取指向反射接口的指针。
D3DCompile	将 HLSL 代码或效果文件编译为给定目标的字节码。
D3DCompile2	将 Microsoft 高级着色器语言 (HLSL) 代码编译为给定目标的字节码。
D3DCompileFromFile	[！注意] 可以使用此 API 开发Windows Microsoft Store应用，但在提交到Windows Microsoft Store的应用中不能使用它。 请参阅 D3DCompile2 备注中的“为 UWP 编译着色器”部分。 将 HLSL 代码编译为给定目标的字节码。
D3DCompressShaders	[！注意] 可以使用此 API 开发Windows Microsoft Store应用，但在提交到Windows Microsoft Store的应用中不能使用它。 将一组着色器压缩为更紧凑的形式。
D3DCreateBlob	创建缓冲区。
D3DCreateFunctionLinkingGraph	创建 function-linking-graph 接口。 [！注意] 此函数是 HLSL 着色器链接技术的一部分，可用于所有 Direct3D 11 平台来创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。

主题	说明
D3DCreateLinker	创建链接器接口。 [！注意] 此函数是 HLSL 着色器链接技术的一部分，可用于所有 Direct3D 11 平台来创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。
D3DDecompressShaders	[！注意] 可以使用此 API 开发Windows Microsoft Store应用，但在提交到Windows Microsoft Store的应用中不能使用它。 从压缩集解压缩一个或多个着色器。
D3DDisassemble	反汇编已编译的 HLSL 代码。
D3DDisassemble10Effect	从 Direct3D10 效果反汇编编译的 HLSL 代码。
D3DDisassemble11Trace	反汇编由着色器跟踪步骤指定的已编译的 HLSL 代码部分。
D3DDisassembleRegion	反汇编已编译的 HLSL 代码的特定区域。
D3DGetBlobPart	从编译结果中检索特定部件。
D3DGetDebugInfo	[！注意] 可以使用此 API 开发Windows Microsoft Store应用，但在提交到Windows Microsoft Store的应用中不能使用它。 获取着色器调试信息。

主题	说明
D3DGetInputAndOutputSignatureBlob	[！注意] D3DGetInputAndOutputSignatureBlob 可能会在 Windows 8.1后更改或不可用。 而是将 D3DGetBlobPart 与 D3D_BLOB_INPUT_AND_OUTPUT_SIGNATURE_BLOB 值一起使用。 从编译结果中获取输入和输出签名。
D3DGetInputSignatureBlob	[！注意] D3DGetInputSignatureBlob 可能会在Windows 8.1后更改或不可用。 而是将 D3DGetBlobPart 与 D3D_BLOB_INPUT_SIGNATURE_BLOB 值一起使用。 从编译结果中获取输入签名。
D3DGetOutputSignatureBlob	[！注意] D3DGetOutputSignatureBlob 可能会在Windows 8.1后更改或不可用。 请改用**具有** D3D_BLOB_OUTPUT_SIGNATURE_BLOB**值的** D3DGetBlobPart。 从编译结果获取输出签名。
D3DGetTraceInstructionOffsets	检索着色器代码部分中指令的字节偏移量。
D3DLoadModule	从着色器模块的源数据创建着色器模块接口。 [！注意] 此函数是 HLSL 着色器链接技术的一部分，可用于所有 Direct3D 11 平台来创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。
D3DPreprocess	预处理未编译的 HLSL 代码。

主题	说明
D3DReadFileToBlob	[！注意] 可以使用此 API 开发Windows Microsoft Store应用，但在提交到Windows Microsoft Store的应用中不能使用它。 将磁盘上的文件读取到内存中。
D3DReflect	获取指向反射接口的指针。
D3DReflectLibrary	从包含 HLSL 函数库的源数据创建库反射接口。 [！注意] 此函数是 HLSL 着色器链接技术的一部分，可用于所有 Direct3D 11 平台来创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。
D3DSetBlobPart	在编译结果中设置信息。
D3DStripShader	从编译结果中删除不需要的 Blob。
D3DWriteBlobToFile	[！注意] 可以使用此 API 开发Windows Microsoft Store应用，但在提交到Windows Microsoft Store的应用中不能使用它。 将内存 Blob 写入磁盘上的文件。

相关主题

D3DCompiler 参考

D3D11Reflect 函数

项目 • 2022/09/22

获取指向反射接口的指针。

语法

syntax

```
HRESULT D3D11Reflect(
  in  LPCVOID pSrcData,
  in  SIZE_T SrcDataSize,
  out ID3D11ShaderReflection ppReflector
);
```

参数

pSrcData [in]

类型： **LPCVOID**

指向作为已编译的 HLSL 代码的源数据的指针。

SrcDataSize [in]

类型： **SIZE_T**

pSrcData 的长度。

ppReflector [out]

类型： **ID3D11ShaderReflection****

指向 **ID3D11ShaderReflection** 接口的指针的地址。

返回值

类型： **HRESULT**

返回主题 Direct3D 11 返回代码中所述的返回代码之一。

备注

内联 D3D11Reflect 编译器函数是 D3DReflect 编译器函数的包装器。 D3D11Reflect 只能从着色器中检索 ID3D11ShaderReflection 接口。 D3DReflect 可以检索 ID3D11ShaderReflection 接口或 Direct3D 10 或 Direct3D 10.1 反射接口，例如 ID3D10ShaderReflection。

着色器代码包含可以使用反射 API 检查的元数据。

以下代码演示如何从着色器检索 ID3D11ShaderReflection 接口。

```
C++
```

```
pd3dDevice->CreatePixelShader( pPixelShaderBuffer->GetBufferPointer(),
                               pPixelShaderBuffer->GetBufferSize(),
g_pPSClassLinkage, &g_pPixelShader );

ID3D11ShaderReflection* pReflector = NULL;
D3D11Reflect( pPixelShaderBuffer->GetBufferPointer(), pPixelShaderBuffer-
>GetBufferSize(),
            &pReflector);
```

要求

要求	Value
标头	D3DCompiler.inl
库	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DCompile 函数 (d3dcompiler.h)

项目 • 2022/09/28

将 HLSL 代码或效果文件编译为给定目标的字节码。

语法

```
C++
```

```
HRESULT D3DCompile(
  [in]            LPCVOID                pSrcData,
  [in]            SIZE_T                 SrcDataSize,
  [in, optional]  LPCSTR                 pSourceName,
  [in, optional]  const D3D_SHADER_MACRO *pDefines,
  [in, optional]  ID3DInclude            *pInclude,
  [in, optional]  LPCSTR                 pEntrypoint,
  [in]            LPCSTR                 pTarget,
  [in]            UINT                   Flags1,
  [in]            UINT                   Flags2,
  [out]           ID3DBlob               **ppCode,
  [out, optional] ID3DBlob               **ppErrorMsgs
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向未编译着色器数据的指针;ASCII HLSL 代码或已编译的效果。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[in, optional] pSourceName`

类型： **LPCSTR**

可以将此参数用于指定错误消息的字符串。 如果未使用，则设置为 **NULL**。

`[in, optional] pDefines`

类型： **const D3D_SHADER_MACRO***

定义着色器宏 的D3D_SHADER_MACRO 结构的可选数组。 每个宏定义都包含一个名称和一个以 null 结尾的定义。 如果未使用，则设置为 **NULL**。 数组中的最后一个结构充当终止符，并且必须将所有成员设置为 **NULL**。

`[in, optional] pInclude`

类型： **ID3DInclude***

可选。 指向用于处理包含文件的 ID3DInclude 的指针。 如果着色器包含#include，则将其设置为 **NULL** 将导致编译错误。 可以传递 **D3D_COMPILE_STANDARD_FILE_INCLUDE** 宏，该宏是指向默认包含处理程序的指针。 此默认包含处理程序包含相对于当前目录的文件以及相对于初始源文件目录的文件。 使用 **D3D_COMPILE_STANDARD_FILE_INCLUDE**时，必须在 *pSourceName* 参数中指定源文件名称;编译器将从 *pSourceName* 派生初始相对目录。

```
C++

#define D3D_COMPILE_STANDARD_FILE_INCLUDE ((ID3DInclude*)(UINT_PTR)1)
```

`[in, optional] pEntrypoint`

类型： **LPCSTR**

着色器执行开始的着色器入口点函数的名称。 使用 fx 配置文件编译 (例如，fx_4_0、fx_5_0等) 时， **D3DCompile** 会忽略 *pEntrypoint*。 在这种情况下，我们建议将 *pEntrypoint* 设置为 **NULL** ，因为如果调用的函数不使用它，最好是将指针参数设置为 **NULL** 。 对于所有其他着色器配置文件，需要有效的 *pEntrypoint* 。

`[in] pTarget`

类型： **LPCSTR**

一个字符串，指定要编译的着色器目标或一组着色器功能。 着色器目标可以是着色器模型 2、着色器模型 3、着色器模型 4 或着色器模型 5。 目标也可以是效果类型 (，例如 fx_4_1) 。 有关各种配置文件支持的目标的信息，请参阅 指定编译器目标。

`[in] Flags1`

类型： **UINT**

由 D3D 编译常量定义的标志。

`[in] Flags2`

类型： **UINT**

由 D3D 编译效果常量定义的标志。 编译着色器而不是效果文件时， **D3DCompile** 将忽略 *Flags2*;建议将 *Flags2* 设置为零，因为如果调用的函数不使用，则最好将非点参数设置为零。

`[out] ppCode`

类型： **ID3DBlob****

指向可用于访问已编译代码的 ID3DBlob 接口的指针的变量的指针。

`[out, optional] ppErrorMsgs`

类型： **ID3DBlob****

指向一个变量的指针，该变量接收指向 ID3DBlob 接口的指针，可用于访问编译器错误消息;如果没有错误，则为 **NULL** 。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

D3DCompile 和 D3DCompile2 之间的区别在于，后一种方法采用一些可选参数，这些参数可用于控制如何生成字节码的某些方面。 如果不需要这种额外的灵活性，则使用 **D3DCompile2** 时没有性能提升。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	d3dcompiler.lib
DLL	d3dcompiler_47.dll

请参阅

函数

D3DCompile2 函数 (d3dcompiler.h)

项目 • 2022/09/27

将 Microsoft 高级着色器语言 (HLSL) 代码编译为给定目标的字节码。

语法

C++

```
HRESULT D3DCompile2(
  [in]            LPCVOID                pSrcData,
  [in]            SIZE_T                 SrcDataSize,
  [in, optional]  LPCSTR                 pSourceName,
  [in, optional]  const D3D_SHADER_MACRO *pDefines,
  [in, optional]  ID3DInclude            *pInclude,
  [in]            LPCSTR                 pEntrypoint,
  [in]            LPCSTR                 pTarget,
  [in]            UINT                   Flags1,
  [in]            UINT                   Flags2,
  [in]            UINT                   SecondaryDataFlags,
  [in, optional]  LPCVOID                pSecondaryData,
  [in]            SIZE_T                 SecondaryDataSize,
  [out]           ID3DBlob               **ppCode,
  [out, optional] ID3DBlob               **ppErrorMsgs
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向非编译着色器数据的指针，（ASCII HLSL 代码）。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 指向的内存块的大小（以字节为单位）。

`[in, optional] pSourceName`

类型： **LPCSTR**

指向常量以 null 结尾的字符串的可选指针，该字符串包含标识在错误消息中使用的源数据的名称。 如果未使用，则设置为 **NULL**。

`[in, optional] pDefines`

类型： **const D3D_SHADER_MACRO***

定义着色器宏 的D3D_SHADER_MACRO 结构的可选数组。 每个宏定义都包含一个名称和一个以 null 结尾的定义。 如果未使用，则设置为 **NULL**。 数组中的最后一个结构充当终止符，并且必须将所有成员设置为 **NULL**。

`[in, optional] pInclude`

类型： **ID3DInclude***

指向编译器用来处理包含文件的 ID3DInclude 接口的指针。 如果将此参数设置为 **NULL**，并且着色器包含#include，则会发生编译错误。 可以传递 **D3D_COMPILE_STANDARD_FILE_INCLUDE** 宏，该宏是指向默认包含处理程序的指针。 此默认包含处理程序包含相对于当前目录的文件以及相对于初始源文件目录的文件。 使用 **D3D_COMPILE_STANDARD_FILE_INCLUDE**时，必须在 *pSourceName* 参数中指定源文件名称;编译器将从 *pSourceName* 派生初始相对目录。

```
#define D3D_COMPILE_STANDARD_FILE_INCLUDE ((ID3DInclude*)(UINT_PTR)1)
```

`[in] pEntrypoint`

类型： **LPCSTR**

指向常量以 null 结尾的字符串的指针，其中包含着色器执行开始的着色器入口点函数的名称。 编译效果时， **D3DCompile2** 将忽略 *pEntrypoint*;我们建议将 *pEntrypoint* 设置为 **NULL** ，因为如果调用的函数不使用它，最好是将指针参数设置为 **NULL** 。

`[in] pTarget`

类型： **LPCSTR**

指向常量以 null 结尾的字符串的指针，该字符串指定要编译的着色器目标或一组着色器功能。 着色器目标可以是着色器模型 (例如着色器模型 2、着色器模型 3、着色器模型 4 或着色器模型 5) 。 目标也可以是效果类型 (，例如fx_4_1) 。 有关各种配置文件支持的目标的信息，请参阅 指定编译器目标。

`[in] Flags1`

类型： **UINT**

着色器 D3D 编译常量 的组合，这些常量是使用按位 **OR** 运算组合的。 生成的值指定编译器如何编译 HLSL 代码。

`[in] Flags2`

类型： **UINT**

使用按位 **OR** 操作组合的效果 D3D 编译效果常量。 生成的值指定编译器如何编译效果。 编译着色器而不是效果文件时， **D3DCompile2** 将忽略 *Flags2*;建议将 *Flags2* 设置为零，因为如果调用的函数不使用，则最好将非点参数设置为零。

`[in] SecondaryDataFlags`

类型： **UINT**

使用以下标志的组合，这些标志通过使用按位 **OR** 操作进行组合。 生成的值指定编译器如何编译 HLSL 代码。

标志	描述
D3DCOMPILE_SECDATA_MERGE_UAV_SLOTS (0x01)	合并非排序访问视图 (UAV) *pSecondaryData* 参数指向的辅助数据中的槽。
D3DCOMPILE_SECDATA_PRESERVE_TEMPLATE_SLOTS (0x02)	保留 *pSecondaryData* 参数指向的辅助数据中的模板槽。
D3DCOMPILE_SECDATA_REQUIRE_TEMPLATE_MATCH (0x04)	要求在编译器编译 HLSL 代码时 *pSecondaryData* 参数指向的辅助数据中的模板匹配。

如果 *pSecondaryData* 为 **NULL**，则设置为零。

`[in, optional] pSecondaryData`

类型： **LPCVOID**

指向辅助数据的指针。 如果不传递辅助数据，请设置为 **NULL**。 使用此辅助数据在两个着色器中对齐 UAV 槽。 假设着色器 A 具有 UAV，并且它们绑定到某些槽。 若要编译着色器 B，以便具有相同名称的 UAV 在 B 中映射到与 A 中相同的槽，请将 A 的字节代码作为辅助数据传递到 **D3DCompile2** 。

`[in] SecondaryDataSize`

类型： **SIZE_T**

pSecondaryData 指向的内存块的大小（以字节为单位）。 如果 *pSecondaryData* 为 **NULL**，则设置为零。

`[out] ppCode`

类型： **ID3DBlob****

指向可用于访问已编译代码的 ID3DBlob 接口的指针的变量的指针。

`[out, optional] ppErrorMsgs`

类型： **ID3DBlob****

指向一个变量的指针，该变量接收指向 ID3DBlob 接口的指针，可用于访问编译器错误消息;如果没有错误，则为 **NULL** 。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

D3DCompile2 和 D3DCompile 之间的区别在于，D3DCompile2 采用一些可选参数 (*SecondaryDataFlags*、*pSecondaryData 和 SecondaryDataSize*) ，这些参数可用于控制如何生成字节码的某些方面。 有关更多详细信息，请参阅这些参数的说明。 否则，对于 **在 D3DCompile2 和 D3DCompile** 之间生成的字节码的效率没有区别。

编译 UWP 的着色器

若要编译脱机着色器，建议使用 Effect-compiler 工具。 如果无法提前编译所有着色器，请考虑编译成本更高的着色器以及启动和大多数性能敏感路径所需的着色器，并在运行时编译其余着色器。 可以使用类似于以下内容的进程在 UWP 应用程序中编译已加载或生成的着色器，而无需阻止用户界面线程。

- 使用 Visual Studio 2015+ 开发 UWP 应用，添加新项“shader.hlsl”。
  - 在 Visual Studio **的解决方案文件夹**视图中，选择 **shaders.hlsl** 项，右键单击“**属性**”。
  - 确保项目 **内容** 设置为 “**是**”。
  - 确保 **项目类型** 设置为 “**文本**”。
  - 将按钮添加到 XAML，将此按钮命名为本示例中 (“TheButton”) ，并添加 **Click** 处理程序。

- 现在，将这些内容添加到 .cpp 文件：

```
syntax

#include <ppltasks.h>
#include <d3dcompiler.h>
#include <Robuffer.h>
```

- 使用以下代码调用 D3DCompile2。 请注意，此处没有错误检查或处理，此外，此代码还演示了可以在后台执行 I/O 和编译，这会使 UI 更具响应性。

C++/CX

```
void App1::DirectXPage::TheButton_Click(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e)
{
  std::shared_ptr<Microsoft::WRL::ComPtr<ID3DBlob>> blobRef =
std::make_shared<Microsoft::WRL::ComPtr<ID3DBlob>>();

  // Load a file and compile it.
  auto fileOp = Windows::ApplicationModel::Package::Current-
>InstalledLocation->GetFileAsync(L"shader.hlsl");
  create_task(fileOp).then([this](Windows::Storage::StorageFile^ file) ->
IAsyncOperation<Windows::Storage::Streams::IBuffer^>^
  {
    // Do file I/O in background thread (use_arbitrary).
    return Windows::Storage::FileIO::ReadBufferAsync(file);
  }, task_continuation_context::use_arbitrary())
    .then([this, blobRef](Windows::Storage::Streams::IBuffer^ buffer)
  {
    // Do compilation in background thread (use_arbitrary).

    // Cast to Object^, then to its underlying IInspectable interface.
    Microsoft::WRL::ComPtr<IInspectable>
insp(reinterpret_cast<IInspectable*>(buffer));

    // Query the IBufferByteAccess interface.
    Microsoft::WRL::ComPtr<Windows::Storage::Streams::IBufferByteAccess>
bufferByteAccess;
    insp.As(&bufferByteAccess);

    // Retrieve the buffer data.
    byte *pBytes = nullptr;
    bufferByteAccess->Buffer(&pBytes);

    Microsoft::WRL::ComPtr<ID3DBlob> blob;
    Microsoft::WRL::ComPtr<ID3DBlob> errMsgs;
    D3DCompile2(pBytes, buffer->Length, "shader.hlsl", nullptr, nullptr,
"main", "ps_5_0", 0, 0, 0, nullptr, 0, blob.GetAddressOf(),
errMsgs.GetAddressOf());
    *blobRef = blob;
  }, task_continuation_context::use_arbitrary())
```

```
    .then([this, blobRef]()
  {
    // Update UI / use shader on foreground thread.
    wchar_t message[40];
    swprintf_s(message, L"blob is %u bytes long", (unsigned)(*blobRef)-
>GetBufferSize());
    this->TheButton->Content = ref new Platform::String(message);
  }, task_continuation_context::use_current());
}
```

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

- 函数
- 在 HLSL 中指定 D3D12 根签名

D3DCompileFromFile 函数 (d3dcompiler.h)

项目 • 2022/09/29

> **注意** 可以使用此 API 开发 Windows 应用商店应用，但无法在提交到 Windows 应用商店的应用中使用它。 请参阅 **D3DCompile2** 的备注中的“为 UWP 编译着色器”部分。

将 Microsoft 高级着色器语言 (HLSL) 代码编译为给定目标的字节码。

语法

```
C++

HRESULT D3DCompileFromFile(
  [in]            LPCWSTR                 pFileName,
  [in, optional]  const D3D_SHADER_MACRO *pDefines,
  [in, optional]  ID3DInclude             *pInclude,
  [in]            LPCSTR                  pEntrypoint,
  [in]            LPCSTR                  pTarget,
  [in]            UINT                    Flags1,
  [in]            UINT                    Flags2,
  [out]           ID3DBlob                **ppCode,
  [out, optional] ID3DBlob                **ppErrorMsgs
);
```

参数

`[in] pFileName`

指向常量以 null 结尾的字符串的指针，该字符串包含包含着色器代码的文件的名称。

`[in, optional] pDefines`

定义着色器宏 的D3D_SHADER_MACRO 结构的可选数组。 每个宏定义都包含一个名称和一个以 null 结尾的定义。 如果未使用，则设置为 **NULL**。 数组中的最后一个结构充当终止符，并且必须将所有成员设置为 **NULL**。

`[in, optional] pInclude`

指向编译器用来处理包含文件的 ID3DInclude 接口的可选指针。 如果将此参数设置为 **NULL** ，并且着色器包含#include，则会发生编译错误。 可以传递 **D3D_COMPILE_STANDARD_FILE_INCLUDE** 宏，该宏是指向默认包含处理程序的指针。 此默认包含处理程序包含相对于当前目录的文件。

```
#define D3D_COMPILE_STANDARD_FILE_INCLUDE ((ID3DInclude*)(UINT_PTR)1)
```

`[in] pEntrypoint`

指向常量以 null 结尾的字符串的指针，其中包含着色器执行开始的着色器入口点函数的名称。 编译效果时， **D3DCompileFromFile** 将忽略 *pEntrypoint*;我们建议将 *pEntrypoint* 设置为 **NULL** ，因为如果调用的函数不使用它，最好是将指针参数设置为 **NULL** 。

`[in] pTarget`

指向常量以 null 结尾的字符串的指针，该字符串指定要编译的着色器目标或一组着色器功能。 着色器目标可以是着色器模型（，例如着色器模型 2、着色器模型 3、着色器模型 4 或着色器模型 5 及更高版本）。 目标也可以是效果类型（，例如fx_4_1）。 有关各种配置文件支持的目标的信息，请参阅 指定编译器目标。

`[in] Flags1`

使用按位 **OR** 操作组合的着色器编译选项。 生成的值指定编译器如何编译 HLSL 代码。

`[in] Flags2`

使用按位 **OR** 操作组合的效果编译选项。 生成的值指定编译器如何编译效果。 编译着色器而不是效果文件时， **D3DCompileFromFile** 将忽略 *Flags2*;建议将 *Flags2* 设置为零，因为如果调用的函数不使用，则最好将非点参数设置为零。

`[out] ppCode`

指向可用于访问已编译代码的 ID3DBlob 接口的指针的变量的指针。

`[out, optional] ppErrorMsgs`

指向一个变量的可选指针，该变量接收指向 ID3DBlob 接口的指针，可用于访问编译器错误消息;如果没有错误，则为 **NULL** 。

返回值

返回 Direct3D 11 返回代码之一。

注解

> **注意** 文件D3dcompiler_44.dll或更高版本包含 **D3DCompileFromFile** 编译器函数。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DCompressShaders 函数 (d3dcompiler.h)

项目 • 2022/09/29

> **注意** 可以使用此 API 开发 Windows 应用商店应用，但在提交到 Windows 应用商店的应用中不能使用它。

将一组着色器压缩为更紧凑的形式。

语法

```
C++

HRESULT D3DCompressShaders(
  [in]  UINT             uNumShaders,
  [in]  D3D_SHADER_DATA *pShaderData,
  [in]  UINT             uFlags,
  [out] ID3DBlob         **ppCompressedData
);
```

参数

`[in] uNumShaders`

类型： **UINT**

要压缩的着色器数。

`[in] pShaderData`

类型： D3D_SHADER_DATA*

描述要压缩的着色器集的 D3D_SHADER_DATA 结构的数组。

`[in] uFlags`

类型： **UINT**

指示如何压缩着色器的标志。 目前，仅定义D3D_COMPRESS_SHADER_KEEP_ALL_PARTS (0x00000001) 标志。

`[out] ppCompressedData`

类型： **ID3DBlob****

指向 ID3DBlob 接口的指针的地址，该接口用于检索压缩的着色器数据。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DCreateBlob 函数 (d3dcompiler.h)

项目 • 2022/10/05

创建缓冲区。

语法

```
C++

HRESULT D3DCreateBlob(
  [in]  SIZE_T   Size,
  [out] ID3DBlob **ppBlob
);
```

参数

`[in] Size`

类型： **SIZE_T**

Blob 中的字节数。

`[out] ppBlob`

类型： **ID3DBlob****

指向用于检索缓冲区的 ID3DBlob 接口的指针的地址。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

最新的D3dcompiler_nn.dll包含 **D3DCreateBlob** 编译器函数。 因此，不再需要使用包含在D3d10.dll中的 D3D10CreateBlob 函数来创建和使用任意长度的数据缓冲区。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DCreateFunctionLinkingGraph 函数 (d3dcompiler.h)

项目 • 2022/10/05

创建 function-linking-graph 接口。

> **注意** 此函数是 HLSL 着色器链接技术的一部分，可用于所有 Direct3D 11 平台来创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。

语法

```
C++

HRESULT D3DCreateFunctionLinkingGraph(
  [in]  UINT                         uFlags,
  [out] ID3D11FunctionLinkingGraph **ppFunctionLinkingGraph
);
```

参数

`[in] uFlags`

类型： **UINT**

预留

`[out] ppFunctionLinkingGraph`

类型： **ID3D11FunctionLinkingGraph****

指向接收指向 ID3D11FunctionLinkingGraph 接口的指针的变量的指针，该接口用于构造包含一系列预编译函数调用的着色器。

返回值

类型： **HRESULT**

如果成功，则返回S_OK;否则，返回 Direct3D 11 返回代码之一。

注解

> **注意** DLL 的 D3dcompiler_47.dll 或更高版本包含 **D3DCreateFunctionLinkingGraph** 函数。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

ID3D11FunctionLinkingGraph

D3DCreateLinker 函数 (d3dcompiler.h)

项目 • 2022/09/27

创建链接器接口。

> **注意** 此函数是 HLSL 着色器链接技术的一部分，可用于在所有 Direct3D 11 平台上创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整着色器。

语法

```
C++

HRESULT D3DCreateLinker(
  [out] ID3D11Linker **ppLinker
);
```

参数

`[out] ppLinker`

类型： **ID3D11Linker****

指向用于链接着色器模块的 ID3D11Linker 接口的指针的变量的指针。

返回值

类型： **HRESULT**

如果成功，则返回S_OK;否则，返回 Direct3D 11 返回代码之一。

注解

> **注意** DLL 的D3dcompiler_47.dll或更高版本包含 **D3DCreateLinker** 函数。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

ID3D11Linker

D3DDecompressShaders 函数 (d3dcompiler.h)

项目 • 2022/09/29

> **注意** 可以使用此 API 开发 Windows 应用商店应用，但在提交到 Windows 应用商店的应用中不能使用它。

从压缩集解压缩一个或多个着色器。

语法

```
C++

HRESULT D3DDecompressShaders(
  [in]            LPCVOID  pSrcData,
  [in]            SIZE_T   SrcDataSize,
  [in]            UINT     uNumShaders,
  [in]            UINT     uStartIndex,
  [in, optional]  UINT     *pIndices,
  [in]            UINT     uFlags,
  [out]           ID3DBlob **ppShaders,
  [out, optional] UINT     *pTotalShaders
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向未编译着色器数据的指针;ASCII HLSL 代码或已编译的效果。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 指向的未编译着色器数据的长度。

`[in] uNumShaders`

类型： **UINT**

要解压缩的着色器数。

`[in] uStartIndex`

类型： **UINT**

要解压缩的第一个着色器的索引。

`[in, optional] pIndices`

类型： **UINT***

表示要解压缩的着色器的索引数组。

`[in] uFlags`

类型： **UINT**

指示如何解压缩的标志。 目前未定义任何标志。

`[out] ppShaders`

类型： **ID3DBlob****

指向 ID3DBlob 接口的指针的地址，该接口用于检索解压缩的着色器数据。

`[out, optional] pTotalShaders`

类型： **UINT***

指向接收 **D3DDecompressShaders** 解压缩的着色器总数的变量的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h

Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DDisassemble 函数 (d3dcompiler.h)

项目 • 2022/09/27

反汇编已编译的 HLSL 代码。

语法

C++

```
HRESULT D3DDisassemble(
  [in]           LPCVOID  pSrcData,
  [in]           SIZE_T   SrcDataSize,
  [in]           UINT     Flags,
  [in, optional] LPCSTR   szComments,
  [out]          ID3DBlob **ppDisassembly
);
```

参数

`[in] pSrcData`

类型： LPCVOID

指向作为已编译的 HLSL 代码的源数据的指针。

`[in] SrcDataSize`

类型： SIZE_T

pSrcData 的长度。

`[in] Flags`

类型： UINT

影响 **D3DDisassemble** 行为的标志。 *标志* 可以是以下值中的零个或多个组合。

标志	描述
D3D_DISASM_ENABLE_COLOR_CODE	启用颜色代码的输出。
D3D_DISASM_ENABLE_DEFAULT_VALUE_PRINTS	启用默认值的输出。
D3D_DISASM_ENABLE_INSTRUCTION_NUMBERING	启用指令编号。

D3D_DISASM_ENABLE_INSTRUCTION_CYCLE	无效。
D3D_DISASM_DISABLE_DEBUG_INFO	禁用调试信息。
D3D_DISASM_ENABLE_INSTRUCTION_OFFSET	启用指令偏移量。
D3D_DISASM_INSTRUCTION_ONLY	仅反汇编说明。
D3D_DISASM_PRINT_HEX_LITERALS	在反汇编中使用十六进制符号。

`[in, optional] szComments`

类型： **LPCSTR**

着色器顶部的注释字符串，用于标识着色器常量和变量。

`[out] ppDisassembly`

类型： **ID3DBlob****

指向接收访问程序集文本的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DDisassemble10Effect 函数 (d3dcompiler.h)

项目 • 2022/09/27

从 Direct3D10 效果反汇编编译的 HLSL 代码。

语法

```
C++

HRESULT D3DDisassemble10Effect(
  [in]  ID3D10Effect *pEffect,
  [in]  UINT         Flags,
  [out] ID3DBlob     **ppDisassembly
);
```

参数

`[in] pEffect`

类型： **ID3D10Effect***

指向作为已编译的 HLSL 代码的源数据的指针。

`[in] Flags`

类型： **UINT**

着色器 编译选项。

`[out] ppDisassembly`

类型： **ID3DBlob****

指向接收包含反汇编文本的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DDisassemble11Trace 函数 (d3d11shadertracing.h)

项目 • 2022/10/05

(HLSL) 着色器跟踪步骤指定的已编译 Microsoft 高级着色器语言) 部分进行反汇编。

语法

```
C++
```

```
HRESULT D3DDisassemble11Trace(
  [in]  LPCVOID           pSrcData,
  [in]  SIZE_T            SrcDataSize,
  [in]  ID3D11ShaderTrace *pTrace,
  [in]  UINT              StartStep,
  [in]  UINT              NumSteps,
  [in]  UINT              Flags,
  [out] ID3D10Blob        **ppDisassembly
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向已编译着色器数据的指针。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 指向的内存块的大小（以字节为单位）。

`[in] pTrace`

类型： **ID3D11ShaderTrace***

指向着色器跟踪信息对象的 ID3D11ShaderTrace 接口的指针。

`[in] StartStep`

类型： **UINT**

从中启动反汇编的 D3DDisassemble11Trace 跟踪中的步骤数。

`[in] NumSteps`

类型： **UINT**

要反汇编的跟踪步骤数。

`[in] Flags`

类型： **UINT**

使用以下按位 OR 操作组合的零个或多个标志的组合。 生成的值指定 D3DDisassemble11Trace 如何反汇编已编译的着色器数据。

标志	描述
D3D_DISASM_ENABLE_COLOR_CODE (0x01)	启用颜色代码的输出。
D3D_DISASM_ENABLE_DEFAULT_VALUE_PRINTS (0x02)	启用默认值的输出。
D3D_DISASM_ENABLE_INSTRUCTION_NUMBERING (0x04)	启用指令编号。
D3D_DISASM_ENABLE_INSTRUCTION_CYCLE (0x08)	无效。
D3D_DISASM_DISABLE_DEBUG_INFO (0x10)	禁用调试信息的输出。
D3D_DISASM_ENABLE_INSTRUCTION_OFFSET (0x20)	启用指令偏移量的输出。
D3D_DISASM_INSTRUCTION_ONLY (0x40)	在 D3DDisassemble11Trace 中为每个步骤启用指令周期的输出。 此标志类似于 D3D_DISASM_ENABLE_INSTRUCTION_NUMBERING 和D3D_DISASM_ENABLE_INSTRUCTION_OFFSET标志。 此标志在 D3DDisassembleRegion 函数中不起作用。 循环信息来自跟踪;因此，循环信息仅在跟踪反汇编中可用。

`[out] ppDisassembly`

类型： **ID3D10Blob****

指向接收访问反汇编 HLSL 代码的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

此方法返回 HRESULT 错误代码。

注解

D3DDisassemble11Trace 演练着色器跟踪的步骤，并针对基于步骤指令索引的每个步骤输出适当的反汇编。 反汇编使用跟踪中的寄存器值信息进行批注。
D3DDisassemble11Trace 的行为不同于 D3DDisassemble，而不是 D3DDisassemble 执行的已编译着色器的静态反汇编，D3DDisassemble11Trace 提供基于着色器跟踪信息的执行跟踪。

要求

最低受支持的客户端	Windows 8 [桌面应用|UWP 应用]
最低受支持的服务器	Windows Server 2012 [桌面应用|UWP 应用]
目标平台	Windows
标头	d3d11shadertracing.h
DLL	D3D11SDKLayers.dll;D3D11_1SDKLayers.dll;D3D11_2SDKLayers.dll

另请参阅

着色器函数

D3DDisassembleRegion function (d3dcompiler.h)

Article10/13/2021

Disassembles a specific region of compiled Microsoft High Level Shader Language (HLSL) code.

Syntax

C++

```
HRESULT D3DDisassembleRegion(
  [in]            LPCVOID  pSrcData,
  [in]            SIZE_T   SrcDataSize,
  [in]            UINT     Flags,
  [in, optional]  LPCSTR   szComments,
  [in]            SIZE_T   StartByteOffset,
  [in]            SIZE_T   NumInsts,
  [out, optional] SIZE_T   *pFinishByteOffset,
  [out]           ID3DBlob **ppDisassembly
);
```

Parameters

`[in] pSrcData`

A pointer to compiled shader data.

`[in] SrcDataSize`

The size, in bytes, of the block of memory that *pSrcData* points to.

`[in] Flags`

A combination of zero or more of the following flags that are combined by using a bitwise **OR** operation. The resulting value specifies how **D3DDisassembleRegion** disassembles the compiled shader data.

Flag	Description
D3D_DISASM_ENABLE_COLOR_CODE (0x01)	Enable the output of color codes.
D3D_DISASM_ENABLE_DEFAULT_VALUE_PRINTS	Enable the output of default values.

(0x02)	
D3D_DISASM_ENABLE_INSTRUCTION_NUMBERING (0x04)	Enable instruction numbering.
D3D_DISASM_ENABLE_INSTRUCTION_CYCLE (0x08)	No effect.
D3D_DISASM_DISABLE_DEBUG_INFO (0x10)	Disable the output of debug information.
D3D_DISASM_ENABLE_INSTRUCTION_OFFSET (0x20)	Enable the output of instruction offsets.
D3D_DISASM_INSTRUCTION_ONLY (0x40)	This flag has no effect in **D3DDisassembleRegion**. Cycle information comes from the trace; therefore, cycle information is available only in D3DDisassemble11Trace's trace disassembly.

`[in, optional] szComments`

A pointer to a constant null-terminated string at the top of the shader that identifies the shader constants and variables.

`[in] StartByteOffset`

The number of bytes offset into the compiled shader data where **D3DDisassembleRegion** starts the disassembly.

`[in] NumInsts`

The number of instructions to disassemble.

`[out, optional] pFinishByteOffset`

A pointer to a variable that receives the number of bytes offset into the compiled shader data where **D3DDisassembleRegion** finishes the disassembly.

`[out] ppDisassembly`

A pointer to a buffer that receives the ID3DBlob interface that accesses the disassembled HLSL code.

Return value

Returns one of the Direct3D 11 return codes.

Remarks

> **Note** The D3dcompiler_44.dll or later version of the file contains the **D3DDisassembleRegion** compiler function.

Requirements

Target Platform	Windows
Header	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

See also

Functions

D3DGetBlobPart 函数 (d3dcompiler.h)

项目 • 2022/09/27

从编译结果中检索特定部件。

语法

C++

```
HRESULT D3DGetBlobPart(
  [in]  LPCVOID       pSrcData,
  [in]  SIZE_T        SrcDataSize,
  [in]  D3D_BLOB_PART Part,
  [in]  UINT          Flags,
  [out] ID3DBlob      **ppPart
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向未编译着色器数据的指针;ASCII HLSL 代码或已编译的效果。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 指向的未编译着色器数据的长度。

`[in] Part`

类型： **D3D_BLOB_PART**

指定要检索的缓冲区部分的 D3D_BLOB_PART类型化值。

`[in] Flags`

类型： **UINT**

指示如何检索 Blob 部件的标志。 目前未定义任何标志。

`[out] ppPart`

类型： **ID3DBlob****

指向 ID3DBlob 接口的指针的地址，该接口用于检索缓冲区的指定部分。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

D3DGetBlobPart 检索 blob (任意长度数据缓冲区)，其中包含 *Part* 参数指定的数据类型。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DGetDebugInfo 函数 (d3dcompiler.h)

项目 • 2022/10/05

> **注意** 可以使用此 API 开发 Windows 应用商店应用，但在提交到 Windows 应用商店的应用中不能使用它。

获取着色器调试信息。

语法

```
C++

HRESULT D3DGetDebugInfo(
  [in]  LPCVOID  pSrcData,
  [in]  SIZE_T   SrcDataSize,
  [out] ID3DBlob **ppDebugInfo
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向源数据的指针;未编译或编译的 HLSL 代码。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[out] ppDebugInfo`

类型： **ID3DBlob****

指向接收包含调试信息的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

调试信息在调用 D3DCompile 后嵌入着色器的正文中。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DGetInputAndOutputSignatureBlob 函数 (d3dcompiler.h)

项目 • 2022/09/29

> **请注意，D3DGetInputAndOutputSignatureBlob** 可能会在Windows 8.1后更改或不可用。 而是将 **D3DGetBlobPart** 与 **D3D_BLOB_INPUT_AND_OUTPUT_SIGNATURE_BLOB** 值一起使用。

从编译结果中获取输入和输出签名。

语法

```
C++

HRESULT D3DGetInputAndOutputSignatureBlob(
  [in]  LPCVOID  pSrcData,
  [in]  SIZE_T   SrcDataSize,
  [out] ID3DBlob **ppSignatureBlob
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

作为已编译的 HLSL 代码指向源数据的指针。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[out] ppSignatureBlob`

类型： **ID3DBlob****

指向接收包含已编译着色器的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DGetInputSignatureBlob 函数 (d3dcompiler.h)

项目 • 2022/09/29

> **请注意，D3DGetInputSignatureBlob** 可能会在Windows 8.1后更改或不可用。 请改用具有**D3D_BLOB_INPUT_SIGNATURE_BLOB**值的 **D3DGetBlobPart**。

从编译结果中获取输入签名。

语法

```
C++

HRESULT D3DGetInputSignatureBlob(
  [in]  LPCVOID  pSrcData,
  [in]  SIZE_T   SrcDataSize,
  [out] ID3DBlob **ppSignatureBlob
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

作为已编译的 HLSL 代码指向源数据的指针。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[out] ppSignatureBlob`

类型： **ID3DBlob****

指向接收包含已编译着色器的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DGetOutputSignatureBlob 函数 (d3dcompiler.h)

项目 • 2022/09/27

> **请注意，D3DGetOutputSignatureBlob 可能会在Windows 8.1后更改或不可用。 请改用具有D3D_BLOB_OUTPUT_SIGNATURE_BLOB值的 D3DGetBlobPart。**

从编译结果中获取输出签名。

语法

C++

```
HRESULT D3DGetOutputSignatureBlob(
  [in]  LPCVOID  pSrcData,
  [in]  SIZE_T   SrcDataSize,
  [out] ID3DBlob **ppSignatureBlob
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

作为已编译的 HLSL 代码指向源数据的指针。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[out] ppSignatureBlob`

类型： **ID3DBlob****

指向接收包含已编译着色器的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DGetTraceInstructionOffsets 函数 (d3dcompiler.h)

项目 • 2022/09/27

检索着色器代码部分中指令的字节偏移量。

语法

C++

```
HRESULT D3DGetTraceInstructionOffsets(
  [in]            LPCVOID pSrcData,
  [in]            SIZE_T  SrcDataSize,
  [in]            UINT    Flags,
  [in]            SIZE_T  StartInstIndex,
  [in]            SIZE_T  NumInsts,
  [out, optional] SIZE_T  *pOffsets,
  [out, optional] SIZE_T  *pTotalInsts
);
```

参数

`[in] pSrcData`

指向已编译着色器数据的指针。

`[in] SrcDataSize`

pSrcData 指向的内存块的大小（以字节为单位）。

`[in] Flags`

使用以下标志的组合，这些标志通过使用按位 **OR** 操作进行组合。 生成的值指定 **D3DGetTraceInstructionOffsets** 如何检索指令偏移量。

标志	描述
D3D_GET_INST_OFFSETS_INCLUDE_NON_EXECUTABLE (0x01)	在检索的信息中包含非可执行代码。

`[in] StartInstIndex`

编译的着色器数据中 **D3DGetTraceInstructionOffsets** 中指令的索引开始检索字节偏移量。

`[in] NumInsts`

D3DGetTraceInstructionOffsets 检索字节偏移量的说明数。

`[out, optional] pOffsets`

指向接收实际偏移量的变量的指针。

`[out, optional] pTotalInsts`

指向接收着色器代码部分中指令总数的变量的指针。

返回值

返回 Direct3D 11 返回代码之一。

注解

新类型的 Microsoft 高级着色器语言 (HLSL) 调试程序数据库 (PDB) 文件中的信息使用着色器 blob 中的指令字节偏移量，(任意长度的数据缓冲区) 。 使用 **D3DGetTraceInstructionOffsets** 转换到指令索引和从指令索引转换。

> **注意** D3dcompiler_44.dll或更高版本的文件包含 **D3DGetTraceInstructionOffsets** 编译器函数。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DLoadModule function (d3dcompiler.h)

Article10/13/2021

Creates a shader module interface from source data for the shader module.

> **Note** This function is part of the HLSL shader linking technology that you can use on all Direct3D 11 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

Syntax

C++

```
HRESULT D3DLoadModule(
  [in]  LPCVOID      pSrcData,
  [in]  SIZE_T       cbSrcDataSize,
  [out] ID3D11Module **ppModule
);
```

Parameters

`[in] pSrcData`

Type: **LPCVOID**

A pointer to the source data for the shader module.

`[in] cbSrcDataSize`

Type: **SIZE_T**

The size, in bytes, of the block of memory that *pSrcData* points to.

`[out] ppModule`

Type: **ID3D11Module****

A pointer to a variable that receives a pointer to the ID3D11Module interface that is used for shader resource re-binding.

Return value

Type: **HRESULT**

Returns S_OK if successful; otherwise, returns one of the Direct3D 11 Return Codes.

Remarks

> **Note** The D3dcompiler_47.dll or later version of the DLL contains the **D3DLoadModule** function.

Requirements

Target Platform	Windows
Header	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

See also

Functions

ID3D11Module

D3DPreprocess 函数 (d3dcompiler.h)

项目 • 2022/09/27

预处理未编译的 HLSL 代码。

语法

C++

```
HRESULT D3DPreprocess(
  [in]            LPCVOID                 pSrcData,
  [in]            SIZE_T                  SrcDataSize,
  [in, optional]  LPCSTR                  pSourceName,
  [in, optional]  const D3D_SHADER_MACRO *pDefines,
  [in, optional]  ID3DInclude             *pInclude,
  [out]           ID3DBlob                **ppCodeText,
  [out, optional] ID3DBlob                **ppErrorMsgs
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向未编译着色器数据的指针;ASCII HLSL 代码或已编译的效果。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[in, optional] pSourceName`

类型： **LPCSTR**

包含未编译的 HLSL 代码的文件的名称。

`[in, optional] pDefines`

类型： **const D3D_SHADER_MACRO***

NULL 终止宏定义的数组 (请参阅 D3D_SHADER_MACRO) 。

`[in, optional] pInclude`

类型： **ID3DInclude***

指向用于处理包含文件的 ID3DInclude 的指针。 如果着色器包含#include，则将其设置为 **NULL** 将导致编译错误。 可以传递 **D3D_COMPILE_STANDARD_FILE_INCLUDE** 宏，该宏是指向默认包含处理程序的指针。 此默认包含处理程序包括相对于当前目录的文件和相对于初始源文件目录的文件。 使用 **D3D_COMPILE_STANDARD_FILE_INCLUDE**时，必须在 *pSourceName* 参数中指定源文件名称;编译器将从 *pSourceName* 派生初始相对目录。

```
C++

#define D3D_COMPILE_STANDARD_FILE_INCLUDE ((ID3DInclude*)(UINT_PTR)1)
```

`[out] ppCodeText`

类型： **ID3DBlob****

包含已编译代码的 ID3DBlob 的地址。

`[out, optional] ppErrorMsgs`

类型： **ID3DBlob****

指向 ID3DBlob 的指针，其中包含编译器错误消息;如果没有错误，则为 **NULL** 。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

D3DPreprocess 输出 #line 指令并保留源输入的行号，以便输出行编号与输入源正确相关。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DReadFileToBlob 函数 (d3dcompiler.h)

项目2023/03/14

> **注意** 可以使用此 API 开发 Windows 应用商店应用，但在提交到 Windows 应用商店的应用中不能使用它。

将磁盘上的文件读取到内存中。

语法

```
C++

HRESULT D3DReadFileToBlob(
  [in]  LPCWSTR  pFileName,
  [out] ID3DBlob **ppContents
);
```

参数

`[in] pFileName`

指向常量以 null 结尾的字符串的指针，该字符串包含要读入内存的文件的名称。

`[out] ppContents`

指向接收指向 ID3DBlob 接口的指针的变量的指针，其中包含从 *pFileName* 文件读取 **D3DReadFileToBlob** 的信息。 可以使用此 **ID3DBlob** 接口访问文件信息并将其传递给其他编译器函数。

返回值

返回 Direct3D 11 返回代码之一。

注解

> **注意** 文件D3dcompiler_44.dll或更高版本包含 **D3DReadFileToBlob** 编译器函数。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DReflect 函数 (d3dcompiler.h)

项目 • 2022/09/27

获取指向反射接口的指针。

语法

```
C++

HRESULT D3DReflect(
  [in]  LPCVOID pSrcData,
  [in]  SIZE_T  SrcDataSize,
  [in]  REFIID  pInterface,
  [out] void    **ppReflector
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向作为已编译的 HLSL 代码的源数据的指针。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 的长度。

`[in] pInterface`

类型： **REFIID**

要使用的 COM 接口的引用 GUID。 例如， **IID_ID3D11ShaderReflection**。

`[out] ppReflector`

类型： **void****

指向反射接口的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

着色器代码包含可以使用反射 API 检查的元数据。

以下代码演示如何从着色器检索 ID3D11ShaderReflection 接口。

```
C++
```

```
pd3dDevice->CreatePixelShader( pPixelShaderBuffer->GetBufferPointer(),
                               pPixelShaderBuffer->GetBufferSize(),
g_pPSClassLinkage, &g_pPixelShader );

ID3D11ShaderReflection* pReflector = NULL;
D3DReflect( pPixelShaderBuffer->GetBufferPointer(), pPixelShaderBuffer-
>GetBufferSize(),
            IID_ID3D11ShaderReflection, (void**) &pReflector);
```

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DReflectLibrary 函数 (d3dcompiler.h)

项目 • 2022/09/27

从包含 HLSL 函数库的源数据创建库反射接口。

> **注意** 此函数是 HLSL 着色器链接技术的一部分，可用于所有 Direct3D 11 平台来创建预编译的 HLSL 函数，将它们打包到库中，并在运行时将它们链接到完整的着色器。

语法

```
C++

HRESULT D3DReflectLibrary(
  [in]  LPCVOID pSrcData,
  [in]  SIZE_T  SrcDataSize,
  [in]  REFIID  riid,
  [out] LPVOID  *ppReflector
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向源数据的指针作为函数的 HLSL 库。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 指向的内存块的大小（以字节为单位）。

`[in] riid`

类型： **REFIID**

要使用的 COM 接口的引用 GUID。 例如， **IID_ID3D11LibraryReflection**。

`[out] ppReflector`

类型： **LPVOID***

指向接收指向库反射接口 ID3D11LibraryReflection 的指针的变量的指针。

返回值

类型： **HRESULT**

如果成功，则返回S_OK;否则，返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

ID3D11LibraryReflection

D3DSetBlobPart 函数 (d3dcompiler.h)

项目 • 2022/09/26

在编译结果中设置信息。

语法

C++

```
HRESULT D3DSetBlobPart(
  [in]  LPCVOID       pSrcData,
  [in]  SIZE_T        SrcDataSize,
  [in]  D3D_BLOB_PART Part,
  [in]  UINT          Flags,
  [in]  LPCVOID       pPart,
  [in]  SIZE_T        PartSize,
  [out] ID3DBlob      **ppNewShader
);
```

参数

`[in] pSrcData`

类型： **LPCVOID**

指向已编译着色器数据的指针。

`[in] SrcDataSize`

类型： **SIZE_T**

pSrcData 指向的已编译着色器数据的长度。

`[in] Part`

类型： **D3D_BLOB_PART**

指定要设置的部件的 D3D_BLOB_PART类型化值。 目前，只能更新专用数据;也就是说，**D3DSetBlobPart** 目前仅支持 D3D_BLOB_PRIVATE_DATA 值。

`[in] Flags`

类型： **UINT**

指示如何设置 Blob 部件的标志。 目前未定义任何标志;因此，设置为零。

`[in] pPart`

类型： **LPCVOID**

指向在编译结果中设置的数据的指针。

`[in] PartSize`

类型： **SIZE_T**

pPart 指向的数据的长度。

`[out] ppNewShader`

类型： **ID3DBlob****

指向接收新部件数据集的新着色器的 ID3DBlob 接口的缓冲区的指针。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

D3DSetBlobPart 修改已编译着色器中的数据。 目前， **D3DSetBlobPart** 只能更新已编译着色器中的专用数据。 可以使用 **D3DSetBlobPart** 将任意未解释的数据附加到已编译的着色器。

> **注意** 文件D3dcompiler_44.dll或更高版本包含 **D3DSetBlobPart** 编译器函数。

要求

目标平台	Windows
标头	d3dcompiler.h

Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

D3DStripShader 函数 (d3dcompiler.h)

项目 • 2022/09/27

从编译结果中删除不需要的 Blob。

语法

```
C++

HRESULT D3DStripShader(
  [in]  LPCVOID  pShaderBytecode,
  [in]  SIZE_T   BytecodeLength,
  [in]  UINT     uStripFlags,
  [out] ID3DBlob **ppStrippedBlob
);
```

参数

`[in] pShaderBytecode`

类型： **LPCVOID**

作为已编译的 HLSL 代码指向源数据的指针。

`[in] BytecodeLength`

类型： **SIZE_T**

pSrcData 的长度。

`[in] uStripFlags`

类型： **UINT**

带状标志选项，由 D3DCOMPILER_STRIP_FLAGS表示。

`[out] ppStrippedBlob`

类型： **ID3DBlob****

指向一个变量的指针，该变量接收指向 ID3DBlob 接口的指针，可用于访问不需要的剥离着色器代码。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3dcompiler_47.lib
DLL	D3dcompiler_47.dll

请参阅

函数

D3DWriteBlobToFile 函数 (d3dcompiler.h)

项目 • 2022/09/27

> **注意** 可以使用此 API 开发 Windows 应用商店应用，但无法在提交到 Windows 应用商店的应用中使用它。

将内存 Blob 写入磁盘上的文件。

语法

```
C++

HRESULT D3DWriteBlobToFile(
  [in] ID3DBlob *pBlob,
  [in] LPCWSTR  pFileName,
  [in] BOOL     bOverwrite
);
```

参数

`[in] pBlob`

类型： **ID3DBlob***

指向 ID3DBlob 接口的指针，该接口包含要写入 *pFileName* 参数指定的文件的内存 blob。

`[in] pFileName`

类型： **LPCWSTR**

指向常量以 null 结尾的字符串的指针，其中包含要写入的文件的名称。

`[in] bOverwrite`

类型： **BOOL**

一个布尔值，该值指定是否覆盖 *pFileName* 文件中的信息。 TRUE 指定覆盖信息，FALSE 指定不覆盖信息。

返回值

类型： **HRESULT**

返回 Direct3D 11 返回代码之一。

注解

> **注意** 文件D3dcompiler_44.dll或更高版本包含 **D3DWriteBlobToFile** 编译器函数。

要求

目标平台	Windows
标头	d3dcompiler.h
Library	D3DCompiler.lib
DLL	D3DCompiler_47.dll

请参阅

函数

结构 (HLSL 参考)

项目 • 2022/09/23

本部分包含有关以下 Direct3D HLSL 编译器结构的信息：

结构	说明
D3D_SHADER_DATA	描述着色器数据。

相关主题

D3DCompiler 参考

D3D_SHADER_DATA结构 (d3dcompiler.h)

项目 • 2022/10/05

描述着色器数据。

语法

```
C++
```

```
typedef struct _D3D_SHADER_DATA {
  LPCVOID pBytecode;
  SIZE_T  BytecodeLength;
} D3D_SHADER_DATA;
```

成员

`pBytecode`

指向着色器数据的指针。

`BytecodeLength`

pBytecode 指向的着色器数据的长度。

注解

D3D_SHADER_DATA结构的数组传递给 D3DCompressShaders，以将着色器数据压缩为更紧凑的形式。

要求

标头	d3dcompiler.h

另请参阅

结构

枚举 (HLSL 引用)

项目 • 2022/09/22

本部分包含有关以下 Direct3D HLSL 编译器枚举的信息：

枚举	描述
D3D_BLOB_PART	标识 blob (任意长度数据缓冲区) 的各个部分。
D3DCOMPILER_STRIP_FLAGS	条带标志选项。

相关主题

D3DCompiler 参考

D3D_BLOB_PART枚举 (d3dcompiler.h)

项目 • 2022/09/26

标识任意长度数据缓冲区内容的部分的值。

语法

C++

```
typedef enum D3D_BLOB_PART {
  D3D_BLOB_INPUT_SIGNATURE_BLOB,
  D3D_BLOB_OUTPUT_SIGNATURE_BLOB,
  D3D_BLOB_INPUT_AND_OUTPUT_SIGNATURE_BLOB,
  D3D_BLOB_PATCH_CONSTANT_SIGNATURE_BLOB,
  D3D_BLOB_ALL_SIGNATURE_BLOB,
  D3D_BLOB_DEBUG_INFO,
  D3D_BLOB_LEGACY_SHADER,
  D3D_BLOB_XNA_PREPASS_SHADER,
  D3D_BLOB_XNA_SHADER,
  D3D_BLOB_PDB,
  D3D_BLOB_PRIVATE_DATA,
  D3D_BLOB_ROOT_SIGNATURE,
  D3D_BLOB_DEBUG_NAME,
  D3D_BLOB_TEST_ALTERNATE_SHADER = 0x8000,
  D3D_BLOB_TEST_COMPILE_DETAILS,
  D3D_BLOB_TEST_COMPILE_PERF,
  D3D_BLOB_TEST_COMPILE_REPORT
} ;
```

常量

`D3D_BLOB_INPUT_SIGNATURE_BLOB` Blob 部件是输入签名。
`D3D_BLOB_OUTPUT_SIGNATURE_BLOB` Blob 部件是输出签名。
`D3D_BLOB_INPUT_AND_OUTPUT_SIGNATURE_BLOB` Blob 部件是输入和输出签名。
`D3D_BLOB_PATCH_CONSTANT_SIGNATURE_BLOB` Blob 部件是修补程序常量签名。

`D3D_BLOB_ALL_SIGNATURE_BLOB` Blob 部件是所有签名。
`D3D_BLOB_DEBUG_INFO` Blob 部件是调试信息。
`D3D_BLOB_LEGACY_SHADER` Blob 部件是一个旧着色器。
`D3D_BLOB_XNA_PREPASS_SHADER` Blob 部件是 XNA 预通道着色器。
`D3D_BLOB_XNA_SHADER` Blob 部件是 XNA 着色器。
`D3D_BLOB_PDB` blob 部件是程序数据库 (PDB) 信息。 **注意** 文件D3dcompiler_44.dll或更高版本支持此值。
`D3D_BLOB_PRIVATE_DATA` Blob 部件是专用数据。 **注意** 文件D3dcompiler_44.dll或更高版本支持此值。
`D3D_BLOB_ROOT_SIGNATURE` Blob 部件是根签名。 有关将 Direct3D12 与 HLSL 配合使用的详细信息，请参阅 在 HLSL 中指定根签名 。 **注意** 文件D3dcompiler_47.dll或更高版本支持此值。

`D3D_BLOB_DEBUG_NAME`
Blob 部件是着色器的调试名称。 如果应用程序未指定调试名称本身，则会改为提供与着色器的 PDB 文件匹配的自动生成的名称。

> **注意**D3dcompiler_47.dll支持此值，如Windows 10 Fall Creators Update及其 SDK 或更高版本的文件提供。

`D3D_BLOB_TEST_ALTERNATE_SHADER`
值： *0x8000*
Blob 部件是测试备用着色器。

> **注意** 此值标识测试部件，仅由特殊编译器版本生成。 因此，此部件类型通常不在着色器中。

`D3D_BLOB_TEST_COMPILE_DETAILS`
Blob 部件是测试编译详细信息。

> **注意** 此值标识测试部件，仅由特殊编译器版本生成。 因此，此部件类型通常不在着色器中。

`D3D_BLOB_TEST_COMPILE_PERF`
Blob 部件是测试编译性能。

> **注意** 此值标识测试部件，仅由特殊编译器版本生成。 因此，此部件类型通常不在着色器中。

`D3D_BLOB_TEST_COMPILE_REPORT` Blob 部件是测试编译报告。 **注意** 此值标识测试部件，仅由特殊编译器版本生成。 因此，此部件类型通常不在着色器中。 **注意** 文件D3dcompiler_44.dll或更高版本支持此值。

注解

这些值将传递给 D3DGetBlobPart 或 D3DSetBlobPart 函数。

要求

标头	d3dcompiler.h

请参阅

枚举

D3DCOMPILER_STRIP_FLAGS枚举 (d3dcompiler.h)

项目 • 2022/09/29

条带标志选项。

语法

C++

```
typedef enum D3DCOMPILER_STRIP_FLAGS {
  D3DCOMPILER_STRIP_REFLECTION_DATA = 0x00000001,
  D3DCOMPILER_STRIP_DEBUG_INFO = 0x00000002,
  D3DCOMPILER_STRIP_TEST_BLOBS = 0x00000004,
  D3DCOMPILER_STRIP_PRIVATE_DATA = 0x00000008,
  D3DCOMPILER_STRIP_ROOT_SIGNATURE = 0x00000010,
  D3DCOMPILER_STRIP_FORCE_DWORD = 0x7fffffff
} ;
```

常量

`D3DCOMPILER_STRIP_REFLECTION_DATA` 值： *0x00000001* 删除反射数据。
`D3DCOMPILER_STRIP_DEBUG_INFO` 值： *0x00000002* 删除调试信息。
`D3DCOMPILER_STRIP_TEST_BLOBS` 值： *0x00000004* 删除测试 Blob 数据。

`D3DCOMPILER_STRIP_PRIVATE_DATA` 值： *0x00000008* **注意** 文件D3dcompiler_44.dll或更高版本支持此值。 删除专用数据。
`D3DCOMPILER_STRIP_ROOT_SIGNATURE` 值： *0x00000010* **注意** 文件D3dcompiler_47.dll或更高版本支持此值。 删除根签名。 有关将 Direct3D12 与 HLSL 配合使用的详细信息，请参阅 在 HLSL 中指定根签名 。
`D3DCOMPILER_STRIP_FORCE_DWORD` 值： *0x7fffffff* 强制此枚举编译为大小为 32 位。 如果没有此值，某些编译器将允许此枚举编译为 32 位以外的大小。 不使用此值。

注解

这些标志由 D3DStripShader 使用。

要求

标头	d3dcompiler.h

请参阅

枚举

D3DCompiler 常量

项目 • 2022/09/24

本部分包含有关以下 Direct3D HLSL 编译器常量的信息：

枚举	描述
D3DCOMPILE 常量	这些常量是着色器编译器选项。
D3DCOMPILE_EFFECT常量	这些常量是效果编译器选项。

相关主题

D3DCompiler 参考

D3DCOMPILE 常量

项目 • 2022/09/22

D3DCOMPILE 常量指定编译器如何编译 HLSL 代码。

返回的常量	说明	注意
D3DCOMPILE_DEBUG /ZI	指示编译器将调试文件/行/类型/符号信息插入到输出代码中。	请参阅D3DXSHADER_DEBUG
D3DCOMPILE_SKIP_VALIDATION /Vd	指示编译器不要根据已知功能和约束验证生成的代码。建议仅将此常量用于过去已成功编译的着色器。DirectX 始终在将着色器设置为设备之前对其进行验证。	请参阅D3DXSHADER_SKIPVALIDATION
D3DCOMPILE_SKIP_OPTIMIZATION /Od	指示编译器在代码生成过程中跳过优化步骤。建议仅针对调试目的设置此常量。	请参阅D3DXSHADER_SKIPOPTIMIZATION

返回的常量	说明	注意
D3DCOMPILE_PACK_MATRIX_ROW_MAJOR /Zpr	指示编译器在着色器的输入和输出上按行主顺序打包矩阵。	请参阅D3DXSHADER_PACKMATRIX_ROWMAJOR
D3DCOMPILE_PACK_MATRIX_COLUMN_MAJOR /Zpc	指示编译器在着色器的输入和输出上按列主顺序打包矩阵。 这种类型的包装通常更高效，因为一系列点产品随后可以执行向量矩阵乘法。	请参阅D3DXSHADER_PACKMATRIX_COLUMNMAJOR
D3DCOMPILE_PARTIAL_PRECISION /Gpp	指示编译器使用部分精度执行所有计算。 如果设置了此常量，则编译的代码可能会在某些硬件上运行速度更快。	请参阅D3DXSHADER_PARTIALPRECISION

返回的常量	说明	注意
D3DCOMPILE_FORCE_VS_SOFTWARE_NO_OPT	指示编译器为下一个最高着色器配置文件编译顶点着色器。 此常量打开调试并关闭优化。	此标志仅适用于 Direct3D 9。 请参阅 D3DXSHADER_FORCE_VS_SOFTWARE_NOOPT
D3DCOMPILE_FORCE_PS_SOFTWARE_NO_OPT	指示编译器为下一个最高着色器配置文件编译像素着色器。 此常量还会打开调试并关闭优化。	此标志仅适用于 Direct3D 9。 请参阅 D3DXSHADER_FORCE_PS_SOFTWARE_NOOPT
D3DCOMPILE_NO_PRESHADER /Op	指示编译器禁用预着色器。 如果设置了此常量，编译器不会提取用于计算的静态表达式。	此标志仅适用于旧版 Direct3D 9 和 Direct3D 10 效果 (FX) 。 请参阅D3DXSHADER_NO_PRESHADER
D3DCOMPILE_AVOID_FLOW_CONTROL /Gfa	指示编译器尽可能不使用流控制构造。	请参阅D3DXSHADER_AVOID_FLOW_CONTROL

返回的常量	说明	注意
D3DCOMPILE_ENABLE_STRICTNESS /Ges	强制严格编译，这可能不允许旧语法。默认情况下，编译器禁用已弃用语法的严格性。	
D3DCOMPILE_IEEE_STRICTNESS /Gis	强制 IEEE 严格编译，以避免可能中断 IEEE 规则的优化。	请参阅D3DXSHADER_IEEE_STRICTNESS
D3DCOMPILE_ENABLE_BACKWARDS_COMPATIBILITY /Gec	指示编译器使较旧的着色器能够编译为 5_0 目标。	请参阅 D3DXSHADER_ENABLE_BACKWARDS_COMPATIBILITY
D3DCOMPILE_OPTIMIZATION_LEVEL0 /O0	指示编译器使用最低优化级别。如果设置此常量，编译器可能会生成较慢的代码，但会更快地生成代码。在以迭代方式开发着色器时设置此常量。	请参阅D3DXSHADER_OPTIMIZATION_LEVEL0

返回的常量	说明	注意
D3DCOMPILE_OPTIMIZATION_LEVEL1 /O1	指示编译器使用第二个最低优化级别。	请参阅D3DXSHADER_OPTIMIZATION_LEVEL1
D3DCOMPILE_OPTIMIZATION_LEVEL2 /O2	指示编译器使用第二个最高优化级别。	请参阅D3DXSHADER_OPTIMIZATION_LEVEL2
D3DCOMPILE_OPTIMIZATION_LEVEL3 /O3	指示编译器使用最高优化级别。 如果设置此常量，编译器会生成最佳代码，但可能需要花费大量时间才能执行此操作。 当性能是最重要的因素时，为应用程序的最终生成设置此常量。	请参阅D3DXSHADER_OPTIMIZATION_LEVEL3

返回的常量	说明	注意
D3DCOMPILE_WARNINGS_ARE_ERRORS /WX	指示编译器在编译着色器代码时将所有警告视为错误。建议将此常量用于新的着色器代码，以便可以解决所有警告，并降低难以查找的代码缺陷的数量。	
D3DCOMPILE_RESOURCES_MAY_ALIAS /res_may_alias	指示编译器假定非排序访问视图 (UAV) 和着色器资源视图 (SRV) 可能会 cs_5_0 别名。	仅适用于 DirectX 12/着色器模型 5.1
D3DCOMPILE_ENABLE_UNBOUNDED_DESCRIPTOR_TABLES /enable_unbounded_descriptor_tables	指示编译器启用未绑定的描述符表。	仅适用于 DirectX 12/着色器模型 5.1
D3DCOMPILE_ALL_RESOURCES_BOUND /all_resources_bound	指示编译器以确保所有资源都绑定。	仅适用于 DirectX 12/着色器模型 5.1

返回的常量	说明	注意
D3DCOMPILE_DEBUG_NAME_FOR_SOURCE /Zss	生成调试 PDB 时，这会对哈希使用源文件和二进制文件。	
D3DCOMPILE_DEBUG_NAME_FOR_BINARY /Zsb	生成调试 PDB 时，这只对哈希使用二进制文件名。	

备注

`D3DCOMPILE_ENABLE_UNBOUNDED_DESCRIPTOR_TABLES`从 `D3DCOMPILE_RESOURCES_MAY_ALIAS`Windows 8.1 SDK 或更高版本随附的D3dcompiler_47.dll开始，编译器常量和`D3DCOMPILE_ALL_RESOURCES_BOUND`编译器常量都是新的。

备注

编译器`D3DCOMPILE_DEBUG_NAME_FOR_SOURCE` `D3DCOMPILE_DEBUG_NAME_FOR_BINARY`常量从 Windows 10 Fall Creator 的 Update SDK (版本 16299) 或更高版本随附的D3dcompiler_47.dll开始是新的。 请参阅**此博客帖子**。

备注

对于 DirectX 12，着色器模型 5.1、D3DCompile API 和 FXC 均已弃用。 请改为通过 DXIL 使用着色器模型 6。 请参阅GitHub。

要求

要求	Value
标头	D3DCompiler.h

另请参阅

D3DCompiler 常量

D3DCOMPILE_EFFECT常量

项目 • 2022/09/22

这些常量指示编译器如何编译效果文件或运行时如何处理效果文件。

D3DCOMPILE_EFFECT_CHILD_EFFECT

(1 << 0)

将效果 (.fx) 文件编译为子效果。 子效果对于任何共享值没有初始值设定项，因为这些子效果在主效果中初始化 (效果池) 。

> **备注**
>
> 效果池受效果 10 (FX10) 支持，但效果池不受效果 11 (FX11) 支持。 有关 Direct3D 10 中的效果池与 Direct3D 11 中效果组之间的差异的详细信息，请参阅 **效果池和组**。

D3DCOMPILE_EFFECT_ALLOW_SLOW_OPS

(1 1 <<)

禁用性能模式并允许可变状态对象。

默认情况下，启用性能模式。 性能模式禁止可变状态对象，防止非文本表达式出现在状态对象定义中。

要求

要求	Value
标头	D3DCompiler.h

另请参阅

D3DCompiler 常量

内联格式转换函数

项目 • 2022/09/24

本节包含以下部分

- 函数
- 结构

> D3DX_DXGIFormatConvert.inl 标头随附在旧版 DirectX SDK 中，并依赖于 XNAMath for C++ 支持。 它还包含在 Microsoft.DXSDK.D3DX NuGet包中。 最新版本使用 DirectXMath 进行 C++ 支持，所有函数都在 **DirectX** C++ 命名空间中定义。

相关主题

HLSL 参考

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

格式转换函数（HLSL 参考）

项目 • 2022/09/23

该部分包含计算和像素着色器中使用的格式转换函数。

- 转换器函数
- 相关主题

> D3DX_DXGIFormatConvert.inl 标头随附在旧版 DirectX SDK 中，并依赖于 XNAMath for C++ 支持。 它还包含在 Microsoft.DXSDK.D3DX NuGet包中。 最新版本使用 DirectXMath 进行 C++ 支持，所有函数都在 **DirectX** C++ 命名空间中定义。

转换器函数

DXGI_FORMAT_R10G10B10A2_UNORM

D3DX_R10G10B10A2_UNORM_to_FLOAT4
D3DX_FLOAT4_to_R10G10B10A2_UNORM

DXGI_FORMAT_R10G10B10A2_UINT

D3DX_R10G10B10A2_UINT_to_UINT4
D3DX_UINT4_to_R10G10B10A2_UINT

DXGI_FORMAT_R8G8B8A8_UNORM：

D3DX_R8G8B8A8_UNORM_to_FLOAT4
D3DX_FLOAT4_to_R8G8B8A8_UNORM

DXGI_FORMAT_R8G8B8A8_UNORM_SRGB

D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4_inexact
D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4
D3DX_FLOAT4_to_R8G8B8A8_UNORM_SRGB

DXGI_FORMAT_R8G8B8A8_UINT

D3DX_R8G8B8A8_UINT_to_UINT4
D3DX_UINT4_to_R8G8B8A8_UINT

DXGI_FORMAT_R8G8B8A8_SNORM

D3DX_R8G8B8A8_SNORM_to_FLOAT4
D3DX_FLOAT4_to_R8G8B8A8_SNORM

DXGI_FORMAT_R8G8B8A8_SINT

D3DX_R8G8B8A8_SINT_to_INT4
D3DX_INT4_to_R8G8B8A8_SINT

DXGI_FORMAT_B8G8R8A8_UNORM

D3DX_B8G8R8A8_UNORM_to_FLOAT4
D3DX_FLOAT4_to_B8G8R8A8_UNORM

DXGI_FORMAT_B8G8R8A8_UNORM_SRGB

D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4_inexact
D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4
D3DX_FLOAT4_to_R8G8B8A8_UNORM_SRGB

DXGI_FORMAT_B8G8R8X8_UNORM

D3DX_B8G8R8X8_UNORM_to_FLOAT3
D3DX_FLOAT3_to_B8G8R8X8_UNORM

DXGI_FORMAT_B8G8R8X8_UNORM_SRGB

D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3_inexact
D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3
D3DX_FLOAT3_to_B8G8R8X8_UNORM_SRGB

DXGI_FORMAT_R16G16_FLOAT

D3DX_R16G16_FLOAT_to_FLOAT2
D3DX_FLOAT2_to_R16G16_FLOAT

DXGI_FORMAT_R16G16_UNORM

D3DX_R16G16_UNORM_to_FLOAT2
D3DX_FLOAT2_to_R16G16_UNORM

DXGI_FORMAT_R16G16_UINT

D3DX_R16G16_UINT_to_UINT2
D3DX_UINT2_to_R16G16_UINT

DXGI_FORMAT_R16G16_SNORM

D3DX_R16G16_SNORM_to_FLOAT2
D3DX_FLOAT2_to_R16G16_SNORM

DXGI_FORMAT_R16G16_SINT

D3DX_R16G16_SINT_to_INT2
D3DX_INT2_to_R16G16_SINT

相关主题

内联格式转换函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4函数

项目 • 2022/09/22

将 DXGI_FORMAT_B8G8R8A8_UNORM_SRGB着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4_inexact函数

项目 • 2022/09/22

将 DXGI_FORMAT_B8G8R8A8_UNORM_SRGB着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4_inexact(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

注解

此函数使用没有足够高精度的着色器指令来给出确切答案。 替代函数 D3DX_B8G8R8A8_UNORM_SRGB_to_FLOAT4 使用存储在着色器中的查阅表来提供精确 SRGB 到浮点转换。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_B8G8R8A8_UNORM_to_FLOAT4 函数

项目 • 2022/12/14

将DXGI_FORMAT_B8G8R8A8_UNORM着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_B8G8R8A8_UNORM_to_FLOAT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3 函数

项目 • 2022/09/24

将DXGI_FORMAT_B8G8R8X8_UNORM_SRGB着色器数据解压缩到 XMFLOAT3。

语法

syntax

```
XMFLOAT3 D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3_inexact函数

项目 • 2022/09/23

将DXGI_FORMAT_B8G8R8X8_UNORM_SRGB着色器数据解压缩到 XMFLOAT3。

语法

syntax

```
XMFLOAT3 D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3_inexact(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

注解

此函数使用没有足够高精度的着色器指令来给出确切答案。 替代函数 D3DX_B8G8R8X8_UNORM_SRGB_to_FLOAT3 使用存储在着色器中的查阅表来提供精确 SRGB 到浮点转换。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_B8G8R8X8_UNORM_to_FLOAT3 函数

项目 • 2022/09/23

将DXGI_FORMAT_B8G8R8X8_UNORM着色器数据解压缩到 XMFLOAT3。

语法

```
syntax
```

```
XMFLOAT3 D3DX_B8G8R8X8_UNORM_to_FLOAT3(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT_to_INT函数

项目 • 2022/09/22

将 FLOAT 值转换为 INT。

语法

syntax

```
INT D3DX_FLOAT_to_INT(
   FLOAT _V,
   FLOAT _Scale
);
```

参数

_V

v 值。

_Scale

刻度值。

返回值

转换后的 FLOAT 值

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT_to_SRGB 函数

项目 • 2022/09/22

将 FLOAT 值转换为 SRGB。

语法

syntax

```
FLOAT D3DX_FLOAT_to_SRGB(
   FLOAT val
);
```

parameters

瓦尔

要转换的 FLOAT 值。

返回值

转换后的 SRGB 值。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT_to_UINT函数

项目 • 2022/09/23

将 FLOAT 值转换为 UINT。

语法

syntax

```
UINT D3DX_FLOAT_to_UINT(
   FLOAT _V,
   FLOAT _Scale
);
```

参数

_V

v 向量。

_Scale

刻度值。

返回值

转换后的 FLOAT 值

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT2_to_R16G16_FLOAT函数

项目 • 2022/09/22

将给定的 XMFLOAT2 打包回DXGI_FORMAT_R16G16_FLOAT。

语法

syntax

```
UINT D3DX_FLOAT2_to_R16G16_FLOAT(
   XMFLOAT2 unpackedInput
);
```

parameters

unpackedInput

解压缩的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT2_to_R16G16_SNORM 函数

项目 • 2022/09/23

将给定的 XMFLOAT2 打包回DXGI_FORMAT_R16G16_SNORM。

语法

syntax

```
UINT D3DX_FLOAT2_to_R16G16_SNORM(
   hlsl_precise XMFLOAT2 unpackedInput
);
```

参数

unpackedInput

解压缩的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT2_to_R16G16_UNORM函数

项目 • 2022/09/22

将给定的 XMFLOAT2 打包回DXGI_FORMAT_R16G16_UNORM。

语法

syntax

```
UINT D3DX_FLOAT2_to_R16G16_UNORM(
   XMFLOAT2 unpackedInput
);
```

参数

unpackedInput

解压缩的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT3_to_B8G8R8X8_UNORM函数

项目 • 2022/09/22

将给定的 XMFLOAT3 打包到 DXGI_FORMAT_B8G8R8X8_UNORM UINT 中。

语法

syntax

```
UINT D3DX_FLOAT3_to_B8G8R8X8_UNORM(
   hlsl_precise XMFLOAT3 unpackedInput
);
```

参数

unpackedInput

解压缩的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT3_to_B8G8R8X8_UNORM_SRGB函数

项目 • 2022/09/22

将给定的 XMFLOAT3 打包回DXGI_FORMAT_B8G8R8X8_UNORM_SRGB。

语法

syntax

```
UINT D3DX_FLOAT3_to_B8G8R8X8_UNORM_SRGB(
   hlsl_precise XMFLOAT3 unpackedInput
);
```

参数

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT4_to_R10G10B10A2_UNORM函数

项目 • 2022/09/24

将给定的 XMFLOAT4 打包回DXGI_FORMAT_R10G10B10A2_UNORM。

语法

syntax

```
UINT D3DX_FLOAT4_to_R10G10B10A2_UNORM(
   hlsl_precise XMFLOAT4 unpackedInput
);
```

参数

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT4_to_B8G8R8A8_UNORM函数

项目 • 2022/09/22

将给定的 XMFLOAT4 打包回DXGI_FORMAT_B8G8R8A8_UNORM。

语法

syntax

```
UINT D3DX_FLOAT4_to_B8G8R8A8_UNORM(
   hlsl_precise XMFLOAT4 unpackedInput
);
```

参数

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT4_to_B8G8R8A8_UNORM_SRGB函数

项目 • 2022/12/14

将给定的 XMFLOAT4 打包到 DXGI_FORMAT_B8G8R8A8_UNORM_SRGB UINT 中。

语法

```
syntax
```

```
UINT D3DX_FLOAT4_to_B8G8R8A8_UNORM_SRGB(
   hlsl_precise XMFLOAT4 unpackedInput
);
```

参数

unpackedInput

解压缩的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT4_to_R8G8B8A8_SNORM函数

项目 • 2022/09/22

将给定的 XMFLOAT4 打包回DXGI_FORMAT_R8G8B8A8_SNORM。

语法

syntax

```
UINT D3DX_FLOAT4_to_R8G8B8A8_SNORM(
   hlsl_precise XMFLOAT4 unpackedInput
);
```

参数

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT4_to_R8G8B8A8_UNORM函数

项目 • 2022/09/23

将DXGI_FORMAT_R8G8B8A8_UNORM着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_FLOAT4_to_R8G8B8A8_UNORM(
   UINT packedInput
);
```

parameters

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_FLOAT4_to_R8G8B8A8_UNORM_SRGB函数

项目 • 2022/09/22

将给定的 XMFLOAT4 打包回DXGI_FORMAT_R8G8B8A8_UNORM_SRGB。

语法

syntax

```
UINT D3DX_FLOAT4_to_R8G8B8A8_UNORM_SRGB(
   hlsl_precise XMFLOAT4 unpackedInput
);
```

parameters

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_INT_to_FLOAT函数

项目 • 2022/09/24

将 INT 值转换为 FLOAT。

语法

syntax

```
FLOAT D3DX_INT_to_FLOAT(
   INT _V,
   FLOAT _Scale
);
```

parameters

_V

v 值。

_Scale

刻度值。

返回值

转换后的 int 值。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_INT2_to_R16G16_SINT函数

项目 • 2022/09/23

将给定的 XMINT2 打包回DXGI_FORMAT_R16G16_SINT。

语法

syntax

```
UINT D3DX_INT2_to_R16G16_SINT(
   hlsl_precise XMINT2 unpackedInput
);
```

parameters

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_INT4_to_R8G8B8A8_SINT函数

项目 • 2022/09/22

将给定的 XMINT4 打包回DXGI_FORMAT_R8G8B8A8_SINT。

语法

syntax

```
UINT D3DX_INT4_to_R8G8B8A8_SINT(
   hlsl_precise XMINT4 unpackedInput
);
```

parameters

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_IsNan函数

项目 • 2022/09/22

确定该值是否为 NaN (不是数字) 。

语法

```
syntax

bool D3DX_IsNan(
   FLOAT _V
);
```

参数

_V

指定的值。

返回值

如果 NaN，则为 true;否则为 false。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R10G10B10A2_UINT_to_UINT4函数

项目 • 2022/09/22

将DXGI_FORMAT_R10G10B10A2_UINT着色器数据解压缩到 XMINT4。

语法

syntax

```
XMINT4 D3DX_R10G10B10A2_UINT_to_UINT4(
   UINT packedInput
);
```

parameters

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R10G10B10A2_UNORM_to_FLOAT4 函数

项目 • 2022/09/22

将 DXGI_FORMAT_R10G10B10A2_UNORM着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_R10G10B10A2_UNORM_to_FLOAT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R16G16_FLOAT_to_FLOAT2函数

项目 • 2022/09/24

将DXGI_FORMAT_B8G8R8X8_UNORM_SRGB着色器数据解压缩到 XMFLOAT2。

语法

syntax

```
XMFLOAT2 D3DX_R16G16_FLOAT_to_FLOAT2(
   UINT packedInput
);
```

parameters

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R16G16_SINT_to_INT2函数

项目 • 2022/09/22

将DXGI_FORMAT_R16G16_SINT着色器数据解压缩到 XMINT2。

语法

```
syntax

XMINT2 D3DX_R16G16_SINT_to_INT2(
   UINT packedInput
);
```

parameters

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R16G16_SNORM_to_FLOAT2函数

项目 • 2022/09/23

将DXGI_FORMAT_R16G16_SNORM着色器数据解压缩到 XMFLOAT2。

语法

syntax

```
XMFLOAT2 D3DX_R16G16_SNORM_to_FLOAT2(
   UINT packedInput
);
```

parameters

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R16G16_UINT_to_UINT2函数

项目 • 2022/09/22

将DXGI_FORMAT_R16G16_UINT着色器数据解压缩到 XMUINT2。

语法

syntax

```
XMUINT2 D3DX_R16G16_UINT_to_UINT2(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R16G16_UNORM_to_FLOAT2函数

项目 • 2022/09/23

将 DXGI_FORMAT_R16G16_UNORM着色器数据解压缩到 XMFLOAT2。

语法

syntax

```
XMFLOAT2 D3DX_R16G16_UNORM_to_FLOAT2(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R8G8B8A8_SINT_to_INT4函数

项目 • 2022/09/24

将DXGI_FORMAT_R8G8B8A8_SINT着色器数据解压缩到 XMINT4。

语法

syntax

```
XMINT4 D3DX_R8G8B8A8_SINT_to_INT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R8G8B8A8_SNORM_to_FLOAT4函数

项目 • 2022/09/22

将DXGI_FORMAT_R8G8B8A8_SNORM着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_R8G8B8A8_SNORM_to_FLOAT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R8G8B8A8_UINT_to_UINT4函数

项目 • 2022/09/23

将DXGI_FORMAT_R8G8B8A8_UINT着色器数据解压缩到 XMUINT4。

语法

syntax

```
XMUINT4 D3DX_R8G8B8A8_UINT_to_UINT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4函数

项目 • 2022/09/22

将DXGI_FORMAT_R8G8B8A8_UNORM_SRGB着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4_inexact函数

项目 • 2022/09/23

将DXGI_FORMAT_R8G8B8A8_UNORM_SRGB着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4_inexact(
   UINT packedInput
);
```

参数

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

注解

此函数使用没有足够高精度的着色器指令来给出确切答案。 替代函数 D3DX_R8G8B8A8_UNORM_SRGB_to_FLOAT4 使用存储在着色器中的查阅表来提供精确的 SRGB 浮点转换。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_R8G8B8A8_UNORM_to_FLOAT4 函数

项目 • 2022/09/24

将DXGI_FORMAT_R8G8B8A8_UNORM着色器数据解压缩到 XMFLOAT4。

语法

syntax

```
XMFLOAT4 D3DX_R8G8B8A8_UNORM_to_FLOAT4(
   UINT packedInput
);
```

parameters

packedInput

打包的着色器数据。

返回值

解压缩的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_Saturate_FLOAT 函数

项目 • 2022/09/22

检索给定 FLOAT 的饱和值。

语法

syntax

```
FLOAT D3DX_Saturate_FLOAT(
   FLOAT _V
);
```

parameters

_V

要饱和的值。

返回值

饱和值。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_SaturateSigned_FLOAT函数

项目 • 2022/09/24

从给定 FLOAT 中检索有符号饱和值。

语法

syntax

```
 D3DX_SaturateSigned_FLOAT(
   FLOAT _V
);
```

参数

_V

要饱和的值。

返回值

带符号饱和值。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_SRGB_to_FLOAT函数

项目 • 2022/09/22

将 SRGB 值转换为 FLOAT。

语法

syntax

```
FLOAT D3DX_SRGB_to_FLOAT(
   UINT val
);
```

参数

瓦尔

要转换的 SRGB 值。

返回值

转换后的 SRGB 值。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_SRGB_to_FLOAT_inexact函数

项目 • 2022/09/22

将 SRGB 值转换为 FLOAT。

语法

syntax

```
FLOAT D3DX_SRGB_to_FLOAT_inexact(
   hlsl_precise FLOAT val
);
```

参数

瓦尔

要转换的 SRGB 值。

返回值

转换后的 SRGB 值。

注解

此函数的精度不够高，无法给出确切答案。 备用函数 D3DX_SRGB_to_FLOAT 使用查阅表为精确 SRGB 提供浮点转换。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_Truncate_FLOAT函数

项目 • 2022/09/23

截断 FLOAT 值。

语法

syntax

```
FLOAT D3DX_Truncate_FLOAT(
   FLOAT _V
);
```

parameters

_V

要截断的值。

返回值

截断的值。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_UINT2_to_R16G16_UINT函数

项目 • 2022/09/22

将给定的 XMUINT2 打包回DXGI_FORMAT_R16G16_UINT。

语法

syntax

```
UINT D3DX_UINT2_to_R16G16_UINT(
   hlsl_precise XMUINT2 unpackedInput
);
```

参数

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_UINT4_to_R10G10B10A2_UINT 函数

项目 • 2022/09/22

将给定的 XMINT4 打包回DXGI_FORMAT_R10G10B10A2_UINT。

语法

syntax

```
UINT D3DX_UINT4_to_R10G10B10A2_UINT(
   hlsl_precise XMINT4 unpackedInput
);
```

parameters

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

D3DX_UINT4_to_R8G8B8A8_UINT 函数

项目 • 2022/09/22

将给定的 XMUINT4 打包回DXGI_FORMAT_R8G8B8A8_UINT。

语法

```
syntax
```

```
UINT D3DX_UINT4_to_R8G8B8A8_UINT(
   hlsl_precise XMUINT4 unpackedInput
);
```

参数

unpackedInput

要打包的着色器数据。

返回值

打包的着色器数据。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

请参阅

函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

(HLSL 引用) 格式转换结构

项目 • 2022/09/23

本节包含以下结构：

- XMINT2
- XMINT4
- XMUINT2
- XMUINT4

相关主题

内联格式转换函数

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

XMINT2 结构

项目 • 2022/09/23

描述二维整数向量。

语法

syntax

```
typedef struct _XMINT2 {
  INT x;
  INT y;
} XMINT2;
```

成员

x

矢量的 x 分量。

y

矢量的 y 分量。

注解

此结构在 DirectX SDK (2010 年 6 月的标头中 `D3DX\_DXGIFormatConvert.inl` 定义，) 供 C++ 使用。 Microsoft.DXSDK.D3DX NuGet包中此标头的最新版本不再定义它，而是依赖于 DirectXMath 中的 DirectX：：XMINT2。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

另请参阅

结构

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

XMINT4 结构

项目 • 2022/09/22

描述 4D 整数向量。

语法

syntax

```
typedef struct _XMINT4 {
  INT x;
  INT {
    INT {
      INT w;
    }z;
  }y;
} XMINT4;
```

成员

x

矢量的 x 分量。

y

矢量的 y 分量。

z

矢量的 z-component。

w

矢量的 w-component。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

注解

此结构在 DirectX SDK (2010 年 6 月的标头中 `D3DX\_DXGIFormatConvert.inl` 定义，) 供 C++ 使用。 Microsoft.DXSDK.D3DX NuGet 包中此标头的最新版本不再定义它，而是依赖于 DirectXMath 中的 DirectX：：XMINT4。

另请参阅

结构

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

XMUINT2 结构

项目 • 2022/09/24

描述 2D 无符号整数向量。

语法

syntax

```
typedef struct _XMUINT2 {
  UINT x;
  UINT y;
} XMUINT2;
```

成员

x

矢量的 x 分量。

y

矢量的 y 分量。

注解

此结构在 DirectX SDK (2010 年 6 月) 的标头中 `D3DX\_DXGIFormatConvert.inl` 定义，以便从 C++ 使用。 Microsoft.DXSDK.D3DX NuGet包中此标头的最新版本不再定义它，而是依赖于 DirectXMath 中的 DirectX：：XMUINT2。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

另请参阅

结构

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

XMUINT4 结构

项目 • 2022/09/22

描述 4D 无符号整数向量。

语法

syntax

```
typedef struct _XMUINT4 {
  UINT x;
  UINT {
    UINT {
      UINT w;
    }z;
  }y;
} XMUINT4;
```

成员

x

矢量的 x 分量。

y

矢量的 y 分量。

z

矢量的 z 分量。

w

矢量的 w 分量。

注解

此结构在 DirectX SDK (2010 年 6 月) 的标头中 `D3DX\_DXGIFormatConvert.inl` 定义，以便从 C++ 使用。 Microsoft.DXSDK.D3DX NuGet包中此标头的最新版本不再定义它，而是依赖于 DirectXMath 中的 DirectX：：XMUINT4。

要求

要求	Value
标头	D3DX_DXGIFormatConvert.inl

另请参阅

结构

用于In-Place图像编辑的解压缩和打包DXGI_FORMAT

附录 (HLSL)

项目 • 2022/09/22

HLSL 附录列出了其他信息，例如语言识别为关键字的单词，以及用于创建 HLSL 语句的语法。 它发布在这里是为了完整。

- Keywords (DirectX HLSL)
- DirectX HLSL (预处理器指令)
- DirectX HLSL) (保留字
- 语法 (DirectX HLSL)

相关主题

HLSL 参考

关键字

项目 • 2022/09/23

Microsoft 高级着色器语言 (HLSL) 将此部分中的单词识别为关键字。 关键字是具有特殊意义的预定义保留标识符。 不能将它们用作应用中的标识符。

- **AppendStructuredBuffer**，asm，asm_fragment
- BlendState、 bool、 break、 **Buffer**、 **ByteAddressBuffer**
- case， cbuffer， 质心， 类， column_major， 编译， **compile_fragment**， **CompileShader**， const， continue， ComputeShader， **ConsumeStructuredBuffer**
- default， DepthStencilState， DepthStencilView， discard， do， double， DomainShader， dword
- else， export， extern
- false， float， for，fxgroup
- GeometryShader， groupshared
- 半， 赫尔沙德
- if， in， inline， inout， **InputPatch**， int， interface
- line、lineadj、 linear、 LineStream
- matrix， min16float， min10float， min16int， min12int， min16uint
- namespace， nointerpolation， noperspective， NULL
- out， **OutputPatch**
- **packoffset**， pass， pixelfragment， PixelShader， point， PointStream， precise
- RasterizerState、RenderTargetView、return、register、row_major、**RWBuffer**、**RWByteAddressBuffer**、**RWStructuredBuffer**、**RWTexture1D**、**RWTexture1D**、**RWTexture2D**、**RWTexture2DArray**、**RWTexture3D**
- sample， sampler， SamplerState， SamplerComparisonState， shared， snorm， stateblock， stateblock_state， static， string， struct， switch， **StructuredBuffer**
- tbuffer， 技术， 技术 10， 技术 11， texture， Texture1DArray， Texture2D， Texture2DArray， Texture2DMS， Texture2DMSArray， Texture3D， TextureCube， TextureCubeArray， true， typedef， 三角形， 三角形，三角形流
- uint， uniform， unorm， unsigned
- vector， vertexfragment， VertexShader， void， volatile
- while

注解

这些数值类型具有标量、矢量和矩阵关键字扩展：

- float、int、uint、bool
- min10float、min16float
- min12int、min16int
- min16uint

这些数值类型的扩展遵循此模式，该模式使用 float 作为示例：

- Scalar
  FLOAT
- 向量
  float1、float2、float3、float4
- Matrix
  float1x1， float1x2， float1x3， float1x4 float2x1， float2x2， float2x3， float2x4
  float3x1， float3x2， float3x4 float4x1， float4x2， float4x3， float4x4

由于旧原因，HLSL 支持小写 纹理 和 采样器 。 相反，对于新应用，我们建议使用 HLSL 的新纹理对象 (Texture2D、 Texture3D 等) 和采样器对象 (SamplerState 和 SamplerComparisonState) 。

导出

使用 **导出** 标记打包到库中的函数。

下面是一个示例：

syntax

```
export float identity(float x)
{
    return x;
}
```

通过使用**导出**关键字标记**标识**函数，可从库中获取**标识**函数以供以后链接。 如果没有 **导出** 标记， **则标识** 函数不可用于以后的链接。

编译器忽略非库编译的 **导出** 关键字。

> ⓘ **备注**
>
> **导出**关键字需要D3dcompiler_47.dll或更高版本的 DLL。

相关主题

附录 (DirectX HLSL)

预处理器指令 (HLSL)

项目 • 2022/09/22

预处理器指令（如 #define 和 #ifdef）通常用于使源程序易于更改，并且易于在不同的执行环境中编译。 源文件中的指令告知预处理器执行特定操作。 例如，预处理器可以替换文本中的标记，将其他文件的内容插入源文件，或通过移除几个部分的文本来取消一部分文件的编译。 在扩展宏之前，将识别并执行预处理器行。 因此，如果宏扩展到类似于预处理器命令的内容，该预处理器无法识别此命令。

预处理器语句使用的字符集与源文件语句的相同，只不过转义序列不受支持。 预处理器语句中使用的字符集与执行字符集相同。 预处理器还可识别负字符值。

HLSL 预处理器识别以下指令：

- #define
- #elif
- #else
- #endif
- #error
- #if
- #ifdef
- #ifndef
- #include
- #line
- #pragma
- #undef

数字符号 (#) 必须是包含指令的行上的第一个非空白字符；空白字符可能出现在数字符号与指令的第一个字母之间。 某些指令包含参数或值。 所有跟在指令后面（指令包含的参数或值除外）的文本的前面必须有单行注释分隔符 (//) 或者必须括在注释分隔符 (/* */) 中。 包含预处理器指令的行可以通过紧靠在结束行标记前放置反斜杠 (\) 继续。

预处理器指令可以出现在源文件中的任何位置，但是它们仅应用于源文件的剩余部分。

相关主题

附录 (DirectX HLSL)

#define指令 (HLSL 引用)

项目 • 2022/09/23

定义常量或宏的预处理器指令。

重载列表

项	说明
#define标识符标记字符串	预处理器指令，用于将有意义的名称分配给应用程序中的常量。
#define标识符 (参数0、...、argumentN-1) token-string	用于创建类似函数的宏的预处理器指令。

另请参阅

DirectX HLSL (预处理器指令)

#define指令 (常量)

项目 • 2022/09/23

预处理器指令，用于将有意义的名称分配给应用程序中的常量。

#define *identifiertoken-string*

参数

项	说明
标识符	常量的标识符。
token-string [可选]	常量的值。 此参数由一系列标记组成，例如关键字、常量或完整语句。 一个或多个空格字符必须将此参数与 *标识符* 参数分开;此空格不被视为替换文本的一部分，也不会在文本的最后一个标记之后使用任何空格。 如果排除此参数，将从源文件中删除 *标识符* 参数的所有实例。 标识符保持不变，可以使用 定义的#if、#ifdef和#ifndef指令进行测试。

注解

源文件中#define指令之后发生的*标识符*参数的所有实例都将替换为 *token-string* 参数的值。 仅当标识符形成令牌时，才会替换标识符;例如，如果标识符出现在批注中、字符串内或作为较长标识符的一部分显示，则不会替换该标识符。

#undef指令指示预处理器忘记标识符的定义;有关详细信息，请参阅 DirectX HLSL) #undef 指令 (。

使用 /D 编译器选项定义常量的效果与在文件开头使用 #define 指令的效果相同。 可以使用 /D 选项定义最多 30 个常量。 有关如何使用此示例，请参阅 #ifdef和) 的示例部分。

示例

以下示例将标识符 WIDTH 定义为整数常量 80，然后在 WIDTH 和整数常量 10 中定义 LENGTH。

```
#define WIDTH       80
#define LENGTH      ( WIDTH + 10 )
```

LENGTH 的每个后续实例都替换为 (WIDTH + 10) ，并且每个后续的 WIDTH + 10 实例都替换为表达式 (80 + 10) 。 宽度 + 10 周围的括号很重要，因为它们控制如下语句中的解释。

```
var = LENGTH * 20;
```

预处理阶段后，语句将变为以下内容，计算结果为 1，800。

```
var = ( 80 + 10 ) * 20;
```

如果没有括号，结果将是以下结果，计算结果为 280。

```
var = 80 + 10 * 20;
```

相关主题

DirectX HLSL (预处理器指令)

#define重载

DirectX HLSL) #undef (指令

#define指令 (宏)

项目 • 2022/09/24

用于创建类似函数的宏的预处理器指令。

#define *标识符* (*argument0* , ... , *argumentN-1*) *token-string*

参数

项	描述
标识符	宏的标识符。 当前上下文中已存在的标识符的宏的第二个 #define 会生成错误，除非第二个标记序列与第一个标记序列相同。
(*argument0* , ... , *argumentN-1*)	宏的参数列表。 参数列表以逗号分隔，可以是任意长度，并且必须括在括号中。 列表中的每个参数名称都必须是唯一的。 没有空格可以分隔 *标识符* 参数和左括号。 使用换行将反斜杠 (\) 紧接在换行符之前，将长指令拆分为多个源行。
token-string [可选]	宏的值。 此参数由一系列标记组成，例如关键字、常量或完整语句。 一个或多个空格字符必须将此参数与 *标识符* 参数分开;此空格不被视为替换文本的一部分，也不会在文本的最后一个标记之后使用任何空格。 自由使用括号，以确保正确解释复杂的参数。 如果 *标识符* 参数的值发生在 *令牌字符串* 参数中 (，即使由于另一个宏扩展) 的结果，也不会扩展它。 如果排除此参数，将从源文件中删除 *标识符* 参数的所有实例。 标识符保持不变，可以使用 定义的#if、#ifdef和#ifndef指令进行测试。

注解

源文件中#define指令之后发生的*标识符*参数的所有实例构成宏调用，该标识符将替换为标记*字符串*参数的版本，该参数具有实际参数替代形式参数。 调用中的参数数必须与宏定义中的参数数匹配。 仅当标识符形成令牌时，才会替换标识符;例如，如果标识符出现在批注中、字符串内或作为较长标识符的一部分显示，则不会替换该标识符。

#undef指令指示预处理器忘记标识符的定义;有关详细信息，请参阅 DirectX HLSL) #undef 指令 (。

使用 /D 编译器选项定义常量的效果与在文件开头使用 #define 指令的效果相同。 可以使用 /D 选项定义最多 30 个宏。

宏调用中的实际参数与宏定义中的相应正式参数匹配。 标记字符串中的每个正式参数都由相应的实际参数替换，除非参数前面有字符串化 (#) 、charizing (#@) 或标记粘贴 (###) 运算符，否则后面是 ## 运算符。 实际自变量中的所有宏都将在指令替换形式参数之前展开。

在 HLSL 编译器中粘贴令牌与 C 编译器中的令牌粘贴略有不同，因此粘贴的令牌必须是自己的有效令牌。 例如，请考虑以下宏定义：

```
#define MERGE(a, b) a##b
MERGE(float, 4x4) test;
```

在 C 编译器中，这会导致以下内容：

```
float4x4 test
```

但是，在 HLSL 编译器中，这会导致以下内容：

```
float4 x4 test
```

可以改用以下宏定义来解决此问题。

```
MERGE(MERGE(float, 4), x4) test;
```

示例

以下示例使用宏来定义游标线。

```
#define CURSOR(top, bottom) (((top) << 8) | (bottom))
```

以下示例定义一个宏，该宏检索指定区域中的伪随机整数。

```
```

```
#define getrandom(min, max) \
((rand()%(int)(((max) + 1)-(min)))+ (min))
```

相关主题

DirectX HLSL (预处理器指令)

#define重载

DirectX HLSL) #undef (指令

#error 指令

项目 • 2022/09/23

生成编译器时错误消息的预处理器指令。

#error *token-string*

参数

项	说明
token-string	错误消息。 此参数由一系列标记组成，例如关键字、常量或完整语句。 令牌字符串受宏扩展的约束。

注解

#error指令最适用于检测程序员在预处理过程中不一致和违反约束的情况。 遇到#error指令时，编译将终止。

示例

以下示例演示预处理过程中的错误处理。

```
#if !defined(__cplusplus)
  #error C++ compiler required.
#endif
```

另请参阅

DirectX HLSL (预处理器指令)

#if、#elif、#else 和 #endif 指令

项目 • 2022/09/24

控制源文件部分编译的预处理器指令。

#if *ifCondition* ...
[#elif *elifCondition* ...]
[#else ...]
#endif

参数

项	说明
ifCondition	要评估的主要条件。 如果此参数的计算结果为非零值，则此#if指令与#elif、#else或#endif指令的下一个实例之间的所有文本都将保留在翻译单元中;否则，不会保留后续源代码。 条件可以使用定义的预处理器运算符来确定是否定义了特定的预处理器常量或宏;此用法等效于 使用 #ifdef 指令。 有关 *ifCondition* 参数的值的限制，请参阅"备注"部分。
elifCondition [可选]	要评估的其他条件。 如果 *ifCondition* 参数和所有以前的 #elif 指令的计算结果为零，并且此参数的计算结果为非零值，则此#elif指令与#elif、#else或#endif指令的下一个实例之间的所有文本都将保留在翻译单元中：否则，不会保留后续源代码。 条件可以使用定义的预处理器运算符来确定是否定义了特定的预处理器常量或宏;此用法等效于 使用 #ifdef 指令。 有关 *elifCondition* 参数值的限制，请参阅"备注"部分。

注解

源文件中的每个#if指令都必须匹配关闭#endif指令。 任何数量的#elif指令都可以出现在#if和#endif指令之间，但最多允许一个#else指令。 #else指令（如果存在）必须是#endif前的最后一个指令。 包含文件中包含的条件编译指令必须满足相同的条件。

#if、#elif、#else 和 #endif 指令可以嵌套在其他#if指令的文本部分中。 每个嵌套#else、#elif或#endif指令都属于最接近的上一个#if指令。

如果没有条件计算为非零值，预处理器将在#else指令后选择文本块。 如果省略#else子句，并且没有条件计算为非零值，则不会选择任何文本块。

ifCondition 和 *elifCondition* 参数非常满足以下要求：

- 条件编译表达式被视为 **有符号长** 值，这些表达式使用与 C++ 中的表达式相同的规则进行评估。
- 表达式必须具有整型，并且只能包含整数常量、字符常量和 defined 运算符。
- 表达式不能使用 **sizeof** 或 type-cast 运算符。
- 目标环境无法表示整数的所有范围。
- 翻译表示与**长**类型相同的类型 int，**而未签名的 int** 与**无符号长**相同。
- 转换器可以将字符常量转换为与目标环境的集不同的代码值集。 若要确定目标环境的属性，请从为目标环境生成的应用程序中的 LIMITS.H 中检查宏的值。
- 该表达式不得执行任何环境查询，并且必须不受与目标计算机上的实现详细信息的影响。

示例

本部分包含演示如何使用条件编译预处理器指令的示例。

使用定义的运算符

以下示例演示了已定义运算符的使用。 如果定义了标识符 CREDIT，则会编译对 **信用** 额度函数的调用。 如果定义了标识符 DEBIT，则编译对 **借记卡** 函数的调用。 如果未定义这两个标识符，则会编译 **对打印机** 函数的调用。 请注意，“CREDIT”和“credit”是 C 和 C++ 中的非重复标识符，因为它们的情况不同。

```
#if defined(CREDIT)
    credit();
#elif defined(DEBIT)
    debit();
#else
    printerror();
#endif
```

使用嵌套#if指令

以下示例演示如何嵌套#if指令。 此示例假定以前定义了名为 DLEVEL 的符号常量。 #elif和#else指令用于根据 DLEVEL 的值做出四个选项之一。 常量 STACK 设置为 0、100 或 200，具体取决于 DLEVEL 的定义。 如果 DLEVEL 大于 5，则不会定义 STACK。

```
#if DLEVEL > 5
    #define SIGNAL  1
    #if STACKUSE == 1
```

```
        #define STACK   200
    #else
        #define STACK   100
    #endif
#else
    #define SIGNAL  0
    #if STACKUSE == 1
        #define STACK   100
    #else
        #define STACK   50
    #endif
#endif
#if DLEVEL == 0
    #define STACK 0
#elif DLEVEL == 1
    #define STACK 100
#elif DLEVEL > 5
    display( debugptr );
#else
    #define STACK 200
#endif
```

用于包括头文件

条件编译的常见用途是防止多次包含同一个头文件。 在 C++ 中，类通常在标头文件中定义，可以使用条件编译构造来防止多个定义。 以下示例确定是否定义了符号常量 EXAMPLE_H。 如果是这样，则文件已包含且不需要重新处理;如果不是，则定义常量 EXAMPLE_H以指示该 EXAMPLE。H 已处理。

```

#if !defined( EXAMPLE_H )
#define EXAMPLE_H

class Example
{
...
};

#endif // !defined( EXAMPLE_H )
```

另请参阅

DirectX HLSL (预处理器指令)

#ifdef 和 #ifndef 指令

项目 • 2022/09/22

确定是否定义了特定的预处理器常量或宏的预处理器指令。

#ifdef *标识符* ...
#endif
#ifndef *标识符* ...
#endif

参数

项	说明
标识符	要检查的常量或宏的标识符。

注解

可以在使用#if的任何位置使用#ifdef和 #ifndef 指令。 #ifdef语句等效于) 指令。 这些指令仅检查使用 #define 指令定义的标识符是否存在，而不检查 C 或 C++ 源代码中声明的标识符。

提供这些指令只是为了实现与该语言的早期版本的兼容性。 首选将 定义的 运算符与 #if 指令一起使用。

#ifndef 指令检查 #ifdef 所检查的条件的相反值。 如果未定义标识符，则条件为 true (非零) ;否则，条件为 false (零) 。

示例

可使用 /D 选项从命令行中传递 identifier。 可使用 /D 指定最多 30 个宏。 这对于检查定义是否存在很有用，因为可以从命令行中传递定义。 以下示例使用此行为来确定是否在测试模式下运行应用程序。

```
// PROG.CPP
#ifndef test
  #define final
```

```
#endif
int main()
{
}
```

使用以下命令进行编译时，prog.cpp 将在测试模式下编译;否则，它将在最终模式下编译。

```
CL.EXE /Dtest prog.cpp
```

另请参阅

DirectX HLSL (预处理器指令)

#if、)

#include 指令

项目 • 2022/09/23

将指定文件的内容插入源程序中的预处理器指令，该指令出现在源程序中。

#include"*filename*"
<#include*文件名*>

参数

项	描述
filename	要包含的文件的文件名（可选）前面有目录规范。 文件名必须指定现有文件。

注解

#include指令会导致由指定文件的全部内容替换指令。 预处理器在找到具有指定名称的文件后立即停止搜索;如果为文件指定完整的明确路径规范，则预处理器仅搜索指定的路径。

> **备注**
>
> Effect-Compiler Tool 具有使用 /I 开关的内置包含处理程序。 但是，从 API 执行编译器时，可以通过实现 ID3DXInclude 接口来提供自定义的 include 处理程序。

这两种语法形式之间的差异是预处理器在未完全指定路径时搜索头文件的顺序，如下表所示。

语法形式	预处理器搜索模式

语法形式	预处理器搜索模式
#include "*filename*"	搜索包含文件： 1. 与包含#include指令的文件位于同一目录中。 2. 包含在包含#include指令的文件的#include指令的任何文件的目录中。 3. 在 /I 编译器选项指定的路径中，按列出顺序排列。 4. 由 INCLUDE 环境变量指定的路径中，按列出顺序排列。 > [!NOTE] > 在开发环境中忽略 INCLUDE 环境变量。 有关如何为项目设置包含路径的信息，请参阅开发环境的文档。
<#include文件名>	搜索包含文件： 1. 在 /I 编译器选项指定的路径中，按列出顺序排列。 2. 由 INCLUDE 环境变量指定的路径中，按列出顺序排列。 > [!NOTE] > 在开发环境中忽略 INCLUDE 环境变量。 有关如何为项目设置包含路径的信息，请参阅开发环境的文档。

示例

下面的示例使预处理器将 #include 指令替换为 stdio.h 的内容。 由于该示例使用尖括号格式，因此预处理器将仅在 /I 编译器选项和 INCLUDE 环境变量列出的目录中搜索文件。

C++

```
#include <stdio.h>
```

另请参阅

- DirectX HLSL (预处理器指令)
- ID3D10Include 接口
- 效果编译器工具

#line 指令

项目 • 2022/09/24

将编译器的内部存储行号和文件名设置为指定值的预处理器指令。

#line *lineNumber* "*filename*"

参数

项	说明
lineNumber	要设置的行号。 这可以是任何整数常量。 只要结果计算结果为正确的语法，就可以对预处理令牌执行宏替换。
文件名 [可选]	要设置的文件名。 文件名可以是字符的任意组合，并且必须用双引号 (") 括起来。 如果省略此参数，则上一个文件名保持不变。

注解

编译器使用行号和文件名来引用它在编译过程中查找的错误。 行号通常引用当前输入行，文件名引用当前输入文件。 处理每行后，行号将递增。 如果更改行号和文件名，编译器将忽略以前的值并继续处理新值。 #line指令通常由程序生成器用来导致错误消息引用原始源文件，而不是引用生成的程序。

翻译器使用行号和文件名来确定预定义宏的值__FILE__和__LINE__。 可以使用这些宏将自我描述性错误消息插入程序文本。 __FILE__宏将展开为文件名的字符串，用双引号 ("") 括起来。

示例

以下示例将行号设置为 151，文件名设置为"copy.c"。

```
#line 151 "copy.c"
```

在以下示例中，宏 ASSERT 使用预定义的宏__LINE__和__FILE__在指定的断言不为 true 时打印有关源文件的错误消息。

```
#define ASSERT(cond)

if( !(cond) )\
{printf( "assertion error line %d, file(%s)\n", \
__LINE__, __FILE__ );}
```

另请参阅

DirectX HLSL (预处理器指令)

#pragma 指令

项目 • 2022/09/22

预处理器指令，提供特定于计算机的或特定于操作系统的功能，同时保持与 C 和 C++ 语言的整体兼容性。

#pragma *token-string*

参数

项	描述
token-string	提供特定编译器指令和参数的一系列字符。 此参数受宏扩展的约束。

注解

如果编译器发现无法识别的杂注，则会发出警告，但编译会继续。

HLSL 编译器可识别以下杂注：

- def
- message
- pack_matrix
- warning

请参阅

DirectX HLSL (预处理器指令)

def pragma 指令

项目 • 2022/09/22

手动分配浮点着色器寄存器的 Pragma 指令。

#pragma def (*target* , *register* , *val1* , *val2* , *val3* , *val4*)

参数

项	说明
目标	包含要分配的寄存器的目标。
注册	要分配的浮点着色器寄存器。
val0	要分配给指定寄存器的值的第一个字节。
val1	要分配给指定寄存器的值的第二个字节。
val2	要分配给指定寄存器的值的第三个字节。
val3	要分配给指定寄存器的值的第四个字节。

注解

def 杂注允许开发人员预先填充具有指定值的浮点着色器寄存器。 此杂注很少使用。

另请参阅

DirectX HLSL (预处理器指令)

DirectX HLSL) (#pragma 指令

message pragma 指令

项目 • 2022/09/22

生成编译器时消息的 Pragma 指令。

#pragma消息 (“*token-string*”)

参数

项	描述
token-string	编译器消息。

示例

以下示例演示了预处理过程中的消息处理。

```
#pragma message "Debugging flag set."
```

另请参阅

DirectX HLSL (预处理器指令)

#pragma指令 (DirectX HLSL)

pack_matrix杂注指令

项目 • 2022/09/22

指定矩阵的打包对齐方式的 Pragma 指令。

#pragma pack_matrix (*对齐*)

参数

项	说明
对准	要为矩阵设置的对齐方式。 此参数可以采用下表中列出的值之一。 **值** / **说明** column_major / 默认。 将矩阵打包对齐方式设置为列主要。 row_major / 将矩阵打包对齐方式设置为行主对齐方式。

示例

以下示例将矩阵打包对齐方式设置为行主。

```
#pragma pack_matrix( row_major )
```

另请参阅

DirectX HLSL (预处理器指令)

#pragma指令 (DirectX HLSL)

warning pragma 指令

项目 • 2022/09/22

修改编译器警告消息行为的 Pragma 指令。

#pragma *警告 (warning-specifier* ： *warning-number-list* [; *warning-specifier* ： *warning-number-list*...])

参数

项	说明
warning-specifier	要为指定的警告设置的行为。 此参数可以采用下表中列出的值之一。 **值** / **说明** once / 仅显示一次具有指定数字的警告消息。 默认值 / 将具有指定数字的警告的行为重置为其默认值。 这也会影响在默认情况下关闭该警告。 警告将在默认级别生成。 1, 2, 3, 4 / 将指定的级别应用于具有指定数字的警告。 这也会影响在默认情况下关闭该警告。 disable / 不要发出具有指定数字的警告。 error / 将具有指定数字的警告报告为错误。
warning-number-list	用于修改其行为的警告编号的空格分隔列表。

备注

可以通过将更改与分号分隔在相同警告杂注中指定任意数量的不同警告行为更改。

编译器会将 4000 添加到介于 0 到 999 之间的任何警告号。 对于大于 4699 的警告编号，(与代码生成关联的警告) 警告杂注仅在放置在函数定义外部时才有效。 如果杂注指定大于 4699 的数字，并在函数内使用，则忽略杂注。

HLSL 警告杂注不支持 C++ 编译器中包含的警告杂注的 **推送** 和 **弹出** 功能。

示例

以下示例禁用警告 4507 和 4034，显示警告 4385 一次，并报告警告 4164 作为错误。

```
#pragma warning( disable : 4507 34; once : 4385; error : 164 )
```

前面的示例在功能上等效于以下内容：

```
#pragma warning( disable : 4507 34 )
#pragma warning( once : 4385 )
#pragma warning( error : 164 )
```

另请参阅

DirectX HLSL (预处理器指令)

#pragma指令 (DirectX HLSL)

#undef 指令

项目 • 2022/09/24

预处理器指令，用于删除以前使用 #define 指令定义的常量或宏的当前定义。

#undef *标识符*

参数

项	说明
标识符	要删除其定义的常量或宏的标识符。 如果未定义宏，请仅提供标识符，而不是参数列表。

注解

可以将#undef指令应用于没有先前定义的标识符;这可确保标识符未定义。 宏替换不会在#undef语句中执行。

#undef指令通常与 #define 指令配对，以在具有特殊含义的源程序中创建区域。 例如，源程序的特定函数可使用清单常量定义不会影响程序的其余部分的环境特定值。 #undef 指令还适用于 [] 指令来控制源程序的条件编译。

可以使用 /U 选项从命令行中取消定义常量和宏，后跟要未定义的标识符。 这相当于在源文件开头添加一系列#undef指令。

示例

以下示例演示如何使用 #undef 指令删除符号常量和宏的定义。

```
#define WIDTH           80
#define ADD( X, Y )     (X) + (Y)

#undef WIDTH
#undef ADD
```

另请参阅

DirectX HLSL (预处理器指令)

#define指令 (DirectX HLSL)

#if、)

保留字

项目 • 2022/09/22

以下单词保留供 HLSL 语言使用。 请勿使用它们命名 HLSL 代码中的变量或函数。

auto case， catch char， class， const_cast default， delete， dynamic_cast enum explicit friend goto long mutable new operator private， protected， public reinterpret_cast short， signed， sizeof， static_cast template， this， throw try， typename union， unsigned using virtual

相关主题

附录 (DirectX HLSL)

语法

项目 • 2022/09/22

使用以下语法规则构造 HLSL 语句。

- 空白
- 浮点数
- 整数数字
- 字符
- 字符串
- 标识符
- 运算符
- 相关主题

空格

以下字符被识别为空白。

- SPACE
- Tab
- EOL
- C 样式注释 (/* */)
- C++ 样式注释 (//)

浮点数

浮点数以 HLSL 表示，如下所示：

- fractional-constant exponent-part (opt) floating-suffix (opt)

  digit-sequence exponent-part floating-suffix (opt)

- fractional-constant：

  digit-sequence (opt) . digit-sequence

  digit-sequence .

- exponent-part：

  e sign (opt) digit-sequence

  e sign (opt) digit-sequence

- sign：一个

  + -

- digit-sequence：

  digit

  digit-sequence digit

- floating-suffix一个

  h H f F l L

  使用“L”后缀指定完整的 64 位精度浮点文本。 32 位浮点文本是默认值。

  例如，编译器将以下文本值识别为 32 位精度浮点文本，并忽略较低位：

  ```
  double x = -0.6473313946860445;
  ```

  编译器将以下文本值识别为 64 位精度浮点文本：

  ```
  double x = -0.6473313946860445L;
  ```

整数数字

整数在 HLSL 中表示，如下所示：

- integer-constant integer-suffix (opt)
- integer-constant：其中一个

  # (十进制数)

  0# (八进制数)

  0x# (十六进制数)

- integer-suffix 可以是以下任一项：

  u U l L

字符

字符以 HLSL 表示，如下所示：

字符	说明
“c”	(字符)
'\a' '\b' '\f' '\b' '\r' '\t' '\v'	(转义)
'\###'	(八进制转义，每个 # 都是八进制数字)
'\x#'	(十六进制转义，# 为十六进制数字，)
'\c'	(c 是其他字符，包括反斜杠和引号)

预处理器表达式不支持转义。

字符串

字符串以 HLSL 表示，如下所示：

“s” (s 是具有转义) 的任何字符串。

标识符

标识符以 HLSL 表示，如下所示：

```
    [A-Za-z_][A-Za-z0-9_]*
```

运算符

```
##, #@, ++, --, &, &, &, ||, ==, ::, <<, <<=, >>, >>=, ...,
<=, >=, !=, *=, /=, +=, -=, %=, &=, |=, ^=, ->
```

此外，与另一个规则不匹配的任何其他单个字符。

相关主题

附录 (DirectX HLSL)

HLSL 错误和警告

项目 • 2022/09/23

着色器可以返回的错误和警告代码。

常量/值	说明
ERR_COMMENTEOF 1001	注释继续超过文件末尾。
ERR_HEXTRUNCATED 1002	十六进制值被截断为 32 位。
ERR_OCTTRUNCATED 1003	八进制值截断为 32 位。
ERR_DECTRUNCATED 1004	小数值截断为 32 位。
ERR_STRINGEOL 1005	字符串继续超过行尾。
ERR_STRINGEOF 1006	字符串继续超过文件末尾。
ERR_CHAREOF 1007	字符继续超过文件末尾。
ERR_TOK_VERSION 1008	令牌版本中的错误。
ERR_PP_SYNTAX 1500	无效的预处理器语法。
ERR_UNEXPECTEDTOKENS 1501	预处理器指令之后出现了意外的令牌。
ERR_UNEXPECTEDEOF 1502	文件结束意外到达。
ERR_DIVZERO 1503	预处理器表达式中的除数为零。
ERR_INVALIDCOMMAND 1504	无效的预处理器命令。
ERR_INCLUDEFROMFILE 1505	从资源或内存支持#include所需的包含接口不起作用。

常量/值	说明
ERR_TOOMANYINCLUDES 1506	嵌套#includes太多。
ERR_FILE_OPEN 1507	指定的源文件无法打开。
ERR_ELIF 1508	发生了意外的#elif指令。
ERR_ELSE 1509	发生了意外的#else指令。
ERR_ENDIF 1510	发生了意外的#endif指令。
ERR_DUPLICATEPARAMATER 1511	向指定的宏提供重复参数。
ERR_RESOURCE_OPEN 1512	资源无法打开。
ERR_ELIF_ELSE 1513	意外的#elif指令遵循#else指令。
ERR_ELSE_ELSE 1514	意外的#else指令遵循#else指令。
ERR_UNEXPECTEDEOF_MACRO 1515	宏扩展中发生了意外的文件结束。
ERR_PARAMETERS_MACRO 1516	没有足够的实际参数提供给指定的宏。
ERR_PP_NOT_YET_IMPLEMENTED 1517	预处理器表达式中的函数定义尚未实现。
ERR_INVALID_INT_EXPR 1518	整数常量表达式无效或不受支持。
ERR_MACRO_REDEFINITION 1519	指定的宏需要重新定义。
ERR_LATE_FULL_PATH 1520	#hlsl_full_path指令必须是源文件中的第一个内容。
ERR_INVALID_FULL_PATH 1521	#hlsl_full_path指令格式不正确。
ERR_PARSE_SYNTAX 3000	分析着色器文件时发现了语法错误。

常量/值	说明
ERR_REDEFINITION 3003	指定的函数需要重新定义。
ERR_UNDECLARED_IDENTIFIER 3004	分析着色器文件时发现了未声明的标识符。
ERR_INVALID_USE 3005	分析着色器文件时发现了类型无效的使用。
ERR_EXTERN 3006	无法声明指定的变量。
ERR_STATIC 3007	无法将指定的变量声明为静态变量。
ERR_VOLATILE 3008	不能将指定的变量声明为可变变量。
ERR_INITIALIZERS 3009	指定的变量不能具有初始值设定项。
ERR_GROUPSHARED 3010	无法将指定的变量声明为组共享，或者组共享变量无法执行特定任务。
ERR_NONLITERAL_INITIALIZER 3011	指定的变量必须是文本表达式。
ERR_MISSING_INITIALIZERS 3012	指定的变量缺少初始值设定项。
ERR_ARGUMENTS 3013	该函数不采用指定的参数数。
ERR_ARGUMENTS_BASETYPE 3014	传递给数值类型构造函数的参数数不正确。
ERR_ARGUMENTS_INTRINSIC 3015	传递给内部函数的参数数不正确。
ERR_UNSUPPORTED_CAST 3017	不支持从一种类型转换为另一种类型。
ERR_SUBSCRIPT 3018	下标无效。
ERR_NUMERIC_EXPECTED 3019	预期数值（如浮点数）。

常量/值	说明
ERR_TYPE_MISMATCH 3020	出现类型不匹配。 例如，如果所有模板类型组件必须具有相同类型，但它们没有，则返回此错误。
ERR_PARSE_ARRAY_EXPECTED 3021	应有一个数组。
ERR_BASETYPE_EXPECTED 3022	预期有标量、矢量或矩阵。
ERR_DETERMINANT 3023	发生了确定性错误，例如错误观察。
ERR_VECTOR_EXPECTED 3024	预期有一个向量。
ERR_LVALUE_EXPECTED 3025	需要一个 l 值，该值指定 const 对象。
ERR_MATRIX_MULTIPLICATION 3026	发生矩阵乘法错误。
ERR_PARSE_ARRAY_INDEX_OUT_OF_BOUNDS 3030	数组的索引超出边界。
ERR_PARSE_IMAGINARY_SQUARE_ROOT 3031	分析着色器文件时发现了虚方根错误。
ERR_PARSE_INDEFINITE_LOG 3032	分析着色器文件时发现了无限期日志错误。
ERR_PARSE_DIVISION_BY_ZERO 3033	分析着色器文件时发现了除零错误。
ERR_CONST 3035	无法将指定的变量声明为 const。
ERR_REDEFINITION_FORMAL_PARAMETER 3036	指定正式参数的重新定义时出错。
ERR_UNSUPPORTED_TYPE_EXPR 3037	仅为数值基类型定义的构造函数。
ERR_NUMERIC 3038	指定的变量必须是数值。
ERR_PARSE_VERSION 3039	不能特定于目标。

常量/值	说明
ERR_ANNOTATIONS 3040	不能有批注。
ERR_SHADER_VERSION 3041	编译器目标不受支持。
ERR_PARSE_NOT_YET_IMPLEMENTED 3042	分析着色器文件时发现了尚未实现的错误。
ERR_SEMANTICS 3043	不能有语义。
ERR_MISSING_DEFAULT_PARAMETER 3044	缺少指定参数的默认值。
ERR_OUTPUT_INITIALIZER 3045	仅输出且无法初始化。
ERR_CONST_OUTPUT 3046	输出参数不能声明为 const。
ERR_UNIFORM 3047	无法将指定的变量声明为统一。
ERR_USAGE 3048	指定了重复的用法。
ERR_USAGE_VERSION 3049	不能特定于用法。
ERR_MATRIX_EXPECTED 3050	预期会有一个矩阵。
ERR_SCALAR_EXPECTED 3051	预期有标量。
ERR_VECTOR_SIZE 3052	矢量维度必须介于 1 和 MAX_VECTOR_SIZE 之间。
ERR_MATRIX_SIZE 3053	矩阵维度必须介于 1 和 MAX_VECTOR_SIZE 之间。
ERR_SHARED 3054	无法将指定的变量声明为共享。
ERR_INLINE 3055	无法将指定的变量声明为内联变量。
ERR_LITERAL_VARIABLE 3057	指定的变量是文本表达式。

常量/值	说明
ERR_ARRAY_LITERAL 3058	数组维度必须是文本标量表达式。
ERR_ARRAY_SIZE 3059	数组维度必须介于 1 和 65536 之间。
ERR_VECTOR_LITERAL 3060	矢量维度必须是文本标量表达式。
ERR_MATRIX_LITERAL 3061	矩阵维度必须是文本标量表达式。
ERR_UNIFORM_OUT 3062	指定的变量不能声明为“统一输出”。
ERR_SAMPLER 3063	指定的变量不能是采样器。
ERR_OBJECT_LITERALS 3064	对象文本表达式不允许在函数内使用。
ERR_OBJECT_ASSIGNMENTS 3065	对象分配不允许在函数内使用。
ERR_SAMPLER_EXPECTED 3066	预期有采样器。
ERR_AMBIGUOUS_FUNCTION_CALL 3067	函数调用不明确。
ERR_PROTOTYPE 3068	函数的返回值不同于函数原型的返回值。
ERR_FUNCTION_HAS_BODY 3069	该函数已有一个正文。
ERR_PARSE_INDEFINITE_ACOS 3070	分析无限期反余弦时发现了语法错误。
ERR_PARSE_INDEFINITE_ASIN 3071	分析无限期反正弦时发现了语法错误。
ERR_ARRAY_IMPLICIT 3072	此类型的数组维度必须是显式的。
ERR_ARRAY_IMPLICIT_ORDER 3073	辅助数组维度必须显式。
ERR_ARRAY_IMPLICIT_VALUE 3074	隐式数组缺少值。

常量/值	说明
ERR_ARRAY_IMPLICIT_SIZE 3075	隐式数组类型不匹配。
ERR_VOID_SEMANTIC 3076	void 函数无法附加语义。
ERR_USAGE_MATRIX 3077	无法将非矩阵类型声明为row_major或column_major。
ERR_REDEFINITION_LOOP_CONTROL 3078	在 for-loop 范围外部使用的循环控制变量与外部作用域中的上一声明冲突;使用了最新的定义。
ERR_RETURN_VOID 3079	Void 函数无法返回值。
ERR_RETURN_VALUE 3080	该函数必须返回一个值。
ERR_COMMA_EXPRESSION 3081	使用了逗号表达式，其中可能打算使用初始值设定项列表。
ERR_BINARYTYPE_EXPECTED 3082	按位运算符需要 int 或 unsigned int 类型。
ERR_GEOMETRY_CONFLICT 3083	存在冲突的几何图形类型。
ERR_ATTRIBUTE 3084	属性因参数错误而出错。
ERR_UNSIGNED_TYPE 3085	无符号类型不能与此变量类型一起使用。
ERR_DEPRECATED_IN_STRICT_MODE 3086	特定语法 (DirectX 9 语法) 或关键字 (像素着色器) 在严格模式下弃用。
ERR_NO_METHODS 3087	对象没有方法。
ERR_UNKNOWN_METHOD 3088	对象没有指定的方法。
ERR_TARGETUSAGE_INVALID 3089	着色器目标或用法无效。
ERR_NO_OBJECTS_IN_STRUCTS 3090	没有可写纹理、采样器或 UAV 可以是具有接口继承的复合类型的成员。

常量/值	说明
ERR_PACK_OFFSET_IN_INVALID_SCOPE 3091	仅允许在常量缓冲区中使用 Packoffset。
ERR_UNARY_NEGATE_OF_UNSIGNED 3092	无符号值的一元反正值仍为无符号。
ERR_OUT_OF_MEMORY 3093	内存不足将执行该操作。
ERR_NON_COMPOUND_BASE 3094	基类型不是结构、类或接口。
ERR_MULTI_CONCRETE_BASE 3095	指定了多个混凝土基类型。
ERR_NOT_TEMPLATE_TYPE 3096	指定的变量不是模板类型。
ERR_STATIC_METHOD_MEMBER_USE 3097	指定的静态方法不能引用实例成员。
ERR_NO_METHOD_PROTOTYPE 3098	在类中找不到该方法。
ERR_STATIC_METHOD_INSTANCE_CALL 3099	无法对对象调用指定的静态方法。
ERR_NO_STATIC_MEMBER_DECL 3100	在类中找不到指定的静态成员或不是静态变量。
ERR_STATIC_MEMBER_TYPE_MISMATCH 3101	声明类型不同于定义类型。
ERR_INVALID_STATIC_MEMBER_SCOPE 3102	静态成员只能在全局范围内定义。
ERR_MISSING_VARIABLE_DEFINITION 3103	指定变量已声明但未定义。
ERR_NO_DATA_IN_INTERFACES 3104	接口不能包含数据。
ERR_NO_STATIC_METHODS_IN_INTERFACES 3105	接口方法不能是静态的。
ERR_NO_INTERFACE_METHOD_BODIES 3106	接口方法不能在接口外部声明。
ERR_NO_INTERFACE_INHERITANCE 3107	接口无法从其他类型的继承。

常量/值	说明
ERR_CLASS_MISSING_INTERFACE_METHOD 3108	该类不实现指定的方法。
ERR_MISMATCHED_OVERRIDE_RETTYPE 3109	返回类型与重写的方法不匹配。
ERR_NO_INTERFACES_AS_MEMBERS 3110	接口不能是成员。
ERR_RECURSIVE_CONTAINMENT 3111	类型不能包含其自己的类型的成员。
ERR_NO_SWITCH 3112	不能对指定程序中的 switch 语句使用调用或 forcecase 属性。
ERR_NO_OUT_DEFAULTS 3113	
ERR_DEFAULT_REDEFINED 3114	默认参数只能在第一个原型中提供。
ERR_CONFLICTING_REGISTER_SEMANTICS 3115	指定的寄存器多次使用。
ERR_INVALID_API_CALL 3116	API 调用无效。
ERR_INVALID_COMPILE_EXPR_FLAG 3117	只能全局设置调试信息标志。
ERR_INTERFACE_OUT 3118	接口只能是输入。
ERR_MULTI_DIM_POINTER_ARRAY 3119	接口数组不能是多维数组。
ERR_INVALID_INDEX 3120	指定了索引的类型无效。 索引必须是标量或具有正确维度数的向量。
ERR_INDEXABLE_TYPE_EXPECTED 3121	索引表达式中需要数组、矩阵、矢量或可索引对象类型。
ERR_NON_SCALAR_VECTOR_ELEMENT 3122	向量元素类型必须是标量类型。
ERR_NON_SCALAR_MATRIX_ELEMENT 3123	矩阵元素类型必须是标量类型。
ERR_OBJECT_HAS_OBJECT_ELEMENT 3124	对象元素类型不能是对象类型。

常量/值	说明
ERR_INVALID_DOT_MIPS_USAGE 3125	.mips 类型只能用于双元素索引表达式，例如 .mips[mip][element]。
ERR_METHOD_IMPL_PROTO_MISMATCH 3126	指定的方法与类中的任何原型不匹配。
ERR_METHOD_IMPL_BODY_MISSING 3127	无法重新声明指定的方法。
ERR_NON_SIMPLE_STREAM 3128	流参数只能是单元素类型。
ERR_WARNING_AS_ERROR 3129	警告被视为错误。
ERR_FX_SINGLE 3130	指定的变量不能声明为“single”。
ERR_NO_STATIC_INTERFACE_INIT 3131	静态接口不能有初始值设定项。
ERR_NO_INTERFACES_IN_BUFFERS 3132	无法在缓冲区中声明接口。
WAR_TYPE_MISMATCH 3200	不建议使用类型不匹配。
WAR_NOFRAGMENTS 3201	不建议使用片段。
WAR_INVALID_SEMANTIC 3202	语义不适用，将被忽略。
WAR_SIGNED_UNSIGNED_COMPARE 3203	假定目标值与值之间发生有符号与无符号不匹配。
WAR_INT_TOO_LARGE 3204	无符号整数文本太大，因此被截断。
WAR_PRECISION_LOSS 3205	在从较大类型转换为较小类型时，可能会丢失数据。
WAR_ELT_TRUNCATION 3206	发生了向量类型的隐式截断。
WAR_CONST_INITIALIZER 3207	初始值设定项用于全局“const”变量。 这需要设置外部常量。 如果需要文本，请改用“static const”。

常量/值	说明
WAR_FAILED_COMPILING_10L9VS 3208	无法编译库函数) 顶点着色器版本的 10_level_9 (9_x功能级别。
WAR_FAILED_COMPILING_10L9PS 3209	无法编译库函数) 像素着色器版本的 10_level_9 (9_x功能级别。
ERR_COMP_NOT_YET_IMPLEMENTED 3500	尚未实现特定表达式。
ERR_ENTRYPOINT_NOT_FOUND 3501	找不到入口点函数。
ERR_MISSING_INPUT_SEMANTICS 3502	指定的输入参数缺少语义。
ERR_MISSING_OUTPUT_SEMANTICS 3503	指定的输出参数或函数返回值缺少语义。
ERR_COMP_ARRAY_INDEX_OUT_OF_BOUNDS 3504	数组的索引超出边界。
ERR_OLD_VERSION 3505	不再支持正在使用的版本;而是使用当前版本。
ERR_UNRECOGNIZED_VERSION 3506	无法识别编译器目标。
ERR_RETURN 3507	类型无法返回值。
ERR_OUT_UNINITIALIZED 3508	输出参数或返回值从未赋值。
ERR_DEPENDENT_TEX1D 3509	纹理示例被认为是依赖的，因为 texcoord 未至少声明为浮点数。
ERR_FUNCTION_MISSING_BODY 3510	函数缺少实现。
ERR_CANT_UNROLL 3511	循环无法取消注册，循环似乎不会及时终止，（指定的迭代数），或者未注册的循环太大。 使用 [unroll (n)] 属性强制增加数字。
ERR_ARRAY_INDEX_MUST_BE_LITERAL 3512	采样器数组的索引必须是文本表达式。
ERR_COMP_ARRAY_EXPECTED 3513	需要数组或特定数组维度。

常量/值	说明
ERR_GEOMETRY_INVALID 3514	指定的输入语义对于几何着色器基元无效，它必须是其自己的参数。
ERR_TARGET_INVALID 3515	目标无效。 例如，用户定义的缓冲区不能是特定的目标缓冲区，寄存器规范需要特定绑定。
ERR_TEXCUBE_OFFSET_INVALID 3516	Texcube 指令不能有整数偏移量。
ERR_UNDEFINED_VARIABLE 3517	变量未定义。
ERR_BREAK_OUTSIDE_LOOP 3518	中断必须位于循环中。
ERR_CONTINUE_OUTSIDE_LOOP 3519	继续必须位于循环中。
ERR_TEXPROJ_INVALID_TEXCOORD 3520	纹理投影不能有 texcoord 指令。
ERR_TEXTURE_TYPE 3521	纹理的返回类型太大。 它不能超过四个组件。
ERR_TEXTURE_OBJECTS_UNSUPPORTED 3522	旧目标不支持纹理对象或流。
ERR_COMPAT_MAKETEXTURE 3523	在 DirectX 9 兼容模式下未运行时，将禁用 DirectX 9 样式的内部函数。
ERR_DUPLICATE_ATTRIBUTE 3524	特定属性不能一起使用，如循环和取消滚动，或提供了重复属性。
ERR_NOT_SIMPLE_LOOP 3525	该循环无法映射到着色器目标，因为目标不支持中断。
ERR_GRADIENT_WITH_BREAK 3526	渐变指令不能用于带断符的循环中。
ERR_TEXTURE_OFFSET 3527	纹理访问需要文本偏移量和多采样索引。
ERR_CANT_BRANCH 3528	不能在此配置文件上使用Flow控制 (分支) 。
ERR_MUST_BRANCH 3529	在此特定情况下，无法平展流控制。

常量/值	说明
ERR_BIND_INVALID 3530	执行了无效的绑定操作。 例如，缓冲区只能绑定到一个槽或一个常量偏移量;由于预期存在特定绑定但未发生，因此注册规范无效;不能将 packoffset 元素与 cbuffer 中的非 packoffset 元素混合使用。
ERR_NEED_UNROLL_FORCED_LOOP 3531	无法取消使用循环属性标记的循环。
ERR_DUPLICATE_CASE 3532	switch 语句中出现重复的默认语句或 case 语句。
ERR_MUST_HAVE_BREAK 3533	非空 case 语句必须具有中断或返回。
ERR_LOW_PRECISION 3534	指定目标不支持部分精度。 最小精度类型可能提供类似的功能。
ERR_UNSUPPORTED_OPERATION 3535	执行了不受支持的操作。 例如，旧目标不支持按位操作; CheckAccessFullyMapped 需要着色器模型 5 或更高版本;平铺资源的 *TextureXxx* 方法需要着色器模型 5 或更高版本。
ERR_INCOMPATIBLE_DUP_SEMANTICS 3536	使用**剪辑平面**属性时，无法使用 SV_ClipDistance语义，或者复制的输入语义无法更改类型、大小或布局。
ERR_NO_FALLTHROUGH 3537	不支持 switch 语句中的倒退案例。 **情况 下**/不**带中断**的下一个 **casedefault**/**的默认语句不能**包含任何代码。
ERR_NON_LITERAL_SAMPLER 3538	采样器参数必须来自文本表达式。
ERR_OLDVERSION 3539	不再支持特定的着色器版本，例如ps_1_x;在 fxc.exe HLSL 代码编译器中使用 /Gec 自动升级到下一个着色器版本，例如，ps_2_0;或者，fxc 的 /LD 选项允许使用以前的编译器 DLL。
ERR_NO_GLOBAL_PACK_OFFSETS 3540	不支持全局 packoffset 变量。
ERR_INVALID_PACK_OFFSET_NAME 3541	指定 **了无效的** packoffset 位置。
ERR_PACK_OFFSET_CANT_HAVE_TARGET 3542	packoffset 变量不能有目标限定符。

常量/值	说明
ERR_REINTERPRET_UNSUPPORTED 3543	该操作无法重新解释提供的数据类型。
ERR_NO_INTERFACE_SUPPORT 3544	指定目标不支持抽象接口;接口引用必须解析为特定实例。
ERR_NO_IFACE_METHOD_IMPLS 3545	没有类实现指定的方法。
ERR_TBUFFER_UNSUPPORTED 3546	指定的目标不支持从纹理缓冲区读取。
ERR_NO_GLOBAL_COMPOUND_WRITES 3547	无法更改全局结构和类。
ERR_NO_NEGATIVE_EMULATED_UINTS 3548	指定的 uint 只能与已知正值一起使用，如果可能，请使用 int。
ERR_INTERLOCKED_TARGET 3549	互锁目标必须是组共享或 UAV 元素。 或者，指定的目标不支持互锁操作，例如，IncrementCounterDecrementCounter/ 仅在 RWStructuredBuffer 对象上有效。
WAR_ARRAY_INDEX_MUST_BE_LITERAL 3550	采样器数组的索引必须是文本表达式，因此循环强制取消滚动。
WAR_INFINITE_LOOP 3551	检测到无限循环，因此循环不会写入任何值。
WAR_NOT_SIMPLE_LOOP 3552	循环无法映射到着色器目标，因为目标不支持中断。
WAR_GRADIENT_WITH_BREAK 3553	不能在带中断的循环中使用渐变指令。
WAR_UNKNOWN_ATTRIBUTE 3554	对于指定的语句，该属性未知或无效。
WAR_INCOMPATIBLE_FLAGS 3555	标志与操作不兼容。
WAR_INT_DIVIDE_SLOW 3556	整数除数可能要慢得多，如果可能，请尝试使用 uint。
WAR_TOO_SIMPLE_LOOP 3557	该循环只执行有限数量的迭代，或者似乎不执行任何操作，因此请考虑将其删除或强制取消滚动。
WAR_ENDIF_UNINITIALIZED 3558	未初始化#endif指令。

常量/值	说明
WAR_LOOP_ASYMMETRIC_RETURN 3559	循环以非对称方式返回。
WAR_MUST_BRANCH 3560	如果包含超出边界的数组访问的语句无法平展。
WAR_OLDVERSION 3561	不再支持特定的着色器版本，例如ps_1_x;使用下一个着色器版本，例如ps_2_0。
WAR_OUTOFBOUNDS_LOOPSIM 3562	循环模拟超出界限。
WAR_OUTOFBOUNDS_LOOPUNROLL 3563	循环从边界中解开。
WAR_PRAGMA_RULEDISABLE 3564	为了获得更好的编译结果，请考虑重新启用指定的规则。
WAR_DID_NOT_SIMULATE 3565	Loop模拟提前完成，请使用 /O1 或更高版本来获取可能更好的 codegen。
WAR_NO_EARLY_BREAK 3566	Loop不会提前退出，请尝试确保循环条件尽可能严格。
WAR_IGNORING_REGISTER_SEMANTIC 3567	忽略寄存器语义。
WAR_UNKNOWN_PRAGMA 3568	忽略未知杂注指令。
WAR_LOOP_TOO_LONG 3569	循环执行的时间超过指定着色器目标的最大迭代次数，从而强制循环取消滚动。
WAR_GRADIENT_MUST_UNROLL 3570	渐变指令用于具有不同迭代的循环中，该迭代强制循环取消滚动。
WAR_POW_NOT_KNOWN_TO_BE_POSITIVE 3571	pow (f、e) 内部函数不适用于负 f、使用 abs (f) 或有条件地处理负值（如果预期）。
WAR_VARYING_INTERFACE 3572	接口引用必须解析为非不同对象。
WAR_TESSFACTORSCALE_OUTOFRANGE 3573	分割因子刻度固定到 [0， 1] 范围。
WAR_SYNC_IN_VARYING_FLOW 3574	线程同步操作不能用于不同的流控制。

常量/值	说明
WAR_BREAK_FROM_UAV 3575	已为循环禁用自动取消注册，请考虑使用 [unroll] 属性或手动取消注册。 或者，不同流控制中的循环终止条件，因此不能依赖于从 UAV 读取的数据。
WAR_OVERRIDDEN_SEMANTIC 3576	修补语义必须位于封闭类型中，因此忽略外部语义。 或者，类型中的语义由变量/函数或封闭类型重写。
WAR_KNOWN_NON_SPECIAL 3577	该值不能为无穷大，可能不需要对 isfinite 的调用。 /Gis 可能会强制执行。 或者，值不能为 NaN，对 isnan 的调用可能不需要。 /Gis 可能强制执行 **isnan** 。
WAR_TLOUT_UNINITIALIZED 3578	输出值未完全初始化。
WAR_GROUPSHARED_UNSUPPORTED 3579	指定的变量不支持 组共享 ，因此忽略 groupshared。
WAR_CONDITIONAL_SIDE_EFFECT 3580	||或 ？：运算符的两侧 &&始终进行计算，因此指定端的副作用不会是有条件的。
WAR_NO_UNSIGNED_ABS 3581	对无符号值的 abs 操作没有意义，因此将被忽略。
WAR_TEXTURE_OFFSET 3582	纹理访问必须具有文本偏移量和多重采样索引。
WAR_POTENTIAL_RACE_CONDITION_UAV 3583	检测到写入共享资源的争用条件，请注意线程正在写入相同的值，但由于争用，性能可能会降低。
WAR_POTENTIAL_RACE_CONDITION_GSM 3584	检测到写入共享内存的争用条件，请注意线程正在写入相同的值，但由于争用，性能可能会降低。
WAR_UNRELIABLE_SOURCE_MARK 3585	Source_mark在 /Od 版本中最有用。 如果没有 /Od source_mark，可以通过优化在最终着色器中四处移动。
WAR_NO_INTERFACE_SUPPORT 3586	指定的目标不支持抽象接口，因此接口引用必须解析为特定实例。
WAR_MIN10_RCP 3587	目标模拟 A/B 与 A * 对等 (B) 。 如果 B 的倒数不能在最小精度类型中表示，则结果在数学上可能不正确。

常量/值	说明
WAR_NO_CLIPPLANES_IN_LIBRARY 3588	库函数中忽略 **剪辑平面** 属性。
ERR_PRAGMA_DEF_OBSOLETE 3589	DirectX 10+ 上不再支持“#pragma def”指令，10_level_9 (9_x功能级别) 目标。 使用兼容性模式允许编译。
ERR_NO_32_BIT_HALF 3650	全局变量不能在指定的目标中使用“half”类型。 若要将此变量视为浮点数，请使用向后兼容性标志。
ERR_NO_32_BIT_DOUBLE 3651	指定的目标不支持双数据类型值。
ERR_NO_SMALL_INT 3652	指定的目标不支持 8 位或 16 位整数。
ERR_NO_64_BIT_INT 3653	指定的目标不支持 64 位整数。
ERR_NO_UNSIGNED_ABS 3654	不支持对无符号值的 abs 操作。
ERR_THREAD_GROUP_SIZE_INVALID 3655	线程组大小无效。
ERR_THREAD_GROUP_SIZE_MISSING 3656	线程组的大小缺失。
ERR_HSATTRIBUTE_INVALID 3657	预期指定参数为特定值，但已获取指定值。 或者，线条或三角形输出拓扑仅适用于 isoline 域。 或者，最大分割因子必须位于 [1，64] 范围内。
ERR_HS_PATCH_INVALID 3658	只允许使用一个 InputPatch 或 OutputPatch 参数。 或者， **InputPatch** 输入只能在外壳和几何图形中使用，（5_0+）着色器。 或者， **OutputPatch** 输入只能在域着色器和外壳着色器的修补常量函数中使用。
ERR_HS_TYPE_MISMATCH 3659	修补常量函数必须使用在控制点阶段声明的相同输入控制点类型。 或者，修补常量函数必须使用从控制点阶段返回的相同输出控制点类型。 或者，修补常量函数的输出修补程序输入应具有一定数量的元素，但具有指定的量。

常量/值	说明
ERR_INTERLOCKED_UNSUPPORTED 3660	指定的目标不支持互锁操作。
ERR_GROUPSHARED_UNSUPPORTED 3661	指定的变量不支持 组共享。
ERR_INDETERMINATE_DERIVATIVE 3662	渐变操作使用一个值，该值可能未为指定目标中的所有像素 (定义，UAV 加载无法参与) 的渐变操作。
ERR_SYNC_IN_VARYING_FLOW 3663	线程同步操作不能用于不同的流控制。
ERR_SYNC_UNSUPPORTED 3664	指定的目标不支持同步操作。
ERR_NO_APPEND_CONSUME 3665	指定的目标不支持追加/使用缓冲区。
ERR_NO_TYPED_UAVS 3666	指定的目标不支持类型化的 UAV。
ERR_NO_UAVS 3667	指定的目标不支持 UAV。
ERR_INDEX_IS_NOT_GROUP_INDEX 3668	要对指定目标的共享内存进行分组的存储只能由 SV_GroupIndex 编制索引。
ERR_NON_LITERAL_RESOURCE 3669	要编制索引的资源不能来自条件表达式，它们必须来自文本表达式。
ERR_NON_LITERAL_STREAM 3670	流参数必须来自文本表达式。
ERR_BREAK_FROM_UAV 3671	Loop不同流控制中的终止条件，因此不能依赖于从 UAV 读取的数据。
ERR_NO_PULL_MODEL 3672	指定的目标不支持拉取模型属性评估。
ERR_CANT_PULL_POSITION 3673	指定的目标不支持位置的拉取模型评估。
ERR_PULL_MUST_BE_INPUT 3674	只能对直接从输入获取的值执行属性评估。
ERR_LOOP_CONDITION_OUT_OF_BOUNDS 3675	不能在条件中使用超出边界的数组引用取消滚动循环。

常量/值	说明
ERR_TYPED_UAV_LOAD_MULTI_COMP 3676	仅单组件 32 位元素类型支持类型化 UAV 加载。
ERR_MULTIPLE_DEPTH_OUT 3677	指定的目标仅允许一个深度输出。
ERR_NO_ORDERED_ACCESS_IN_INTERFACE 3678	尚未实现包含 UAV 或组共享变量的接口可访问成员。
ERR_COMP_GLC_INVALID 3679	存储类 globallycoherent 只能与无序访问视图一起使用， (UAV) 缓冲区，不能与追加/使用缓冲区一起使用。
ERR_HS_UNKNOWN_OUTPUT_TYPE 3680	定义直通控制点着色器时，必须声明 InputPatch 对象，并且输出控制点的数目必须为零，或者必须与输入修补程序大小匹配。
ERR_ATOMIC_REQUIRES_INT 3681	指定的目标仅支持对标量 int 或 uint 数据执行互锁操作。
ERR_ATTRIBUTE_PARAM_SIDE_EFFECT 3682	具有副作用的表达式作为属性参数无效
ERR_INVALID_RESOURCE_CONTAINER 3683	组共享变量不能包含纹理、采样器或 UAV 等资源。 或者，纹理、采样器或 UAV 等资源不能包含其他资源。
ERR_UNSUPPORTED_DOUBLE_OPERATION 3684	指定的目标不支持双精度浮点。 或者，操作不能直接用于资源。 或者，操作不能与双精度值一起使用，首先强制转换为浮点。 或者，给定类型不支持该操作。
ERR_INVALID_TESS_FACTOR_SEMANTIC 3685	tessfactor 语义无序。 或者，冲突象限/三/isoline tessfactor 语义。 或者，tessfactor 语义必须位于同一组件中。
ERR_UNSUPPORTED_THIS_OBJECT 3686	不支持指定的对象。
ERR_INVALID_SHADER_IO 3687	双精度类型不能用作着色器输入或输出。 如果需要在着色器阶段之间传递双精度值，则必须将其作为两个 uint 传递，并使用 asuint 和 asdouble 在窗体之间转换。
ERR_INDEXED_DERIV 3688	尚未实现索引变量的派生。

常量/值	说明
ERR_ORDERED_ACCESS_CAST 3689	赋值左侧不能强制转换为可索引对象，因此请考虑在右侧使用 asuint、 asfloat 或 **加倍**。
ERR_RESOURCE_UNINITIALIZED 3690	正在编制索引的资源未初始化。
ERR_INVALID_STATIC_VAR_INIT 3691	静态变量初始值设定项中的变量引用无效。局部变量不能用于初始化静态变量。
ERR_NO_ABORT 3692	指定的目标不支持中止。
ERR_NO_MESSAGES 3693	指定的目标不支持消息。
ERR_GUARANTEED_RACE_CONDITION_UAV 3694	检测到写入共享资源的争用条件，因此请考虑使此操作写入条件。
ERR_GUARANTEED_RACE_CONDITION_GSM 3695	检测到写入共享内存的争用条件，因此请考虑使此操作写入条件。
ERR_INFINITE_LOOP 3696	检测到无限循环，因此循环永远不会退出。
ERR_TEMPLATE_VAR_CONFLICT 3697	指定的变量与模板着色器中的变量匹配，但类型布局不匹配。
ERR_RESOURCE_BIND_CONFLICT 3698	指定的资源与模板着色器有绑定冲突。
ERR_COMPLEX_TEMPLATE_RESOURCE 3699	放置持有者模板资源只能是简单的资源，因此不支持结构和数组。
ERR_RESOURCE_NOT_IN_TEMPLATE 3700	对于指定的资源，模板着色器中不存在绑定。
ERR_RESINDEX_UNSUPPORTED 3701	指定的目标不支持索引资源。
ERR_FMA_ONLY_DOUBLE 3702	fma 内部函数只能与双参数一起使用。
ERR_NO_MIN_PRECISION 3703	指定的目标不支持最小精度数据。
ERR_NO_F32_F16 3704	指定的目标不支持 16 位浮点转换。

常量/值	说明
ERR_NOT_ABLE_TO_FLATTEN 3705	如果包含副作用的语句无法平展。
ERR_INVALID_MININT 3706	最小精度类型不支持带符号整数除法。 强制转换为 int 以使用 32 位除法。
ERR_INVALID_MIN8FLOAT 3707	最小 8 位浮点值无效或不受支持。
ERR_CONTINUE_INSIDE_SWITCH 3708	不能在 switch 语句中使用 continue 语句。
ERR_DEBUG_NOT_SUPPORTED_FOR_MODERN 3709	不支持调试。
ERR_UNSUPPORTED_PARAM_TYPE 3710	不支持指定的函数参数。
ERR_DUPLICATE_FUNC_PARAM_SEMANTICS 3711	库函数参数和返回值不能具有重复的语义。
ERR_LIBRARY_FUNC_UNSUPPORTED 3712	库函数仅支持像素着色器和顶点着色器。
ERR_ENTRYPOINT_MUST_BE_EMPTY 3713	不能为库指定入口点。 使用导出关键字标记库入口点。
ERR_NO_STATIC_IN_LIBRARY 3714	指定的变量声明为静态变量，库尚不支持该变量。
ERR_NO_TBUFFER_IN_LIBRARY 3715	指定的变量声明为 tbuffer，库尚不支持该变量。
ERR_NO_INTERFACES_IN_LIBRARY 3716	库中不支持类和接口。
ERR_NO_DOUBLE_IN_LIBRARY 3717	双精度数据类型不能用作库函数输入或输出。 如果需要将双精度传递给库函数，则必须将其作为两个 uint 传递，并使用 asuint 和 asdouble 在窗体之间转换。
ERR_NO_OVERLOADING_FOR_LIB_FUNC 3718	库入口点无法重载。
ERR_RES_MAY_ALIAS_ONLY_IN_CS_5 3719	“resources_may_alias”选项仅适用于 cs_5_0+ 目标。
ERR_READ_BEFORE_WRITE 4000	指定变量在未完全初始化的情况下使用。

常量/值	说明
ERR_MID_DIVISION_BY_ZERO 4001	中间级预处理器表达式中的除数为零。
ERR_MID_INDEFINITE_LOG 4002	发生了无限期对数。
ERR_MID_IMAGINARY_SQUARE_ROOT 4003	发生虚方根。
ERR_TOO_COMPLEX 4004	程序过于复杂，因为有比寄存器更多的活动值。
ERR_INDEFINITE_ASIN 4005	发生无限期反正弦值。
ERR_INDEFINITE_ACOS 4006	发生无限期反余弦值。
ERR_ARRAY_INDEX_OUT_OF_BOUNDS 4007	数组索引超出边界。
WARN_FLOAT_DIVISION_BY_ZERO 4008	浮点除以零发生。
ERR_IDIV_DIVISION_BY_ZERO 4009	发生整数除以零。
ERR_UDIV_DIVISION_BY_ZERO 4010	发生无符号整数除数。
ERR_FTOI_OUTOFRANGE 4011	转换的整数范围中的浮点值。
ERR_FTOU_OUTOFRANGE 4012	转换的无符号整数范围的浮点值。
ERR_INDEFINITE_DSXY 4013	发生无限期派生计算。
ERR_GRADIENT_FLOW 4014	渐变操作不能在具有分流控制的循环中发生。
ERR_MID_SEMANTIC_TOO_LONG 4015	语义长度太长。
ERR_INVALID_SEMANTIC 4016	语义无效。 例如， SV_InstanceID**语义不能**与10_level_9 (9_x功能级别) 目标一起使用，也不支持零字符语义。

常量/值	说明
ERR_MID_INVALID_REGISTER_SEMANTIC 4017	同一个变量不能绑定到同一常量库中的多个常量。
ERR_TOO_MANY_PHASES 4018	着色器使用依赖项链中的纹理寻址操作，该操作对于特定目标着色器模型而言太复杂，无法处理。
ERR_CONSTANT_REG_COLLISION 4019	发现了多个变量，其中包含相同的用户指定位置。
ERR_TBUFFER_REG_COLLISION 4020	发现了多个变量，其中包含相同的用户指定位置。
ERR_DERIV_READ_BEFORE_WRITE 4021	派生是在定义之前使用的，因此请考虑在程序前面移动派生分配。
ERR_DERIV_INVALID_PREDICATE 4022	派生在流控制的不同分支中未定义，因此请考虑在任何流控制语句之前移动派生赋值。
ERR_DERIV_REDEFINITION 4023	已重新定义派生，并且只能分配一次派生。
ERR_DERIV_KNOWN_VALUE 4024	已知值的派生未实现。
ERR_DERIV_UNKNOWN 4025	无法计算指定值的派生值。
ERR_RACE_CONDITION_INDUCED_INV_SYNC 4026	线程同步操作必须处于非不同流控制中。 由于潜在的争用条件，此同步无效，因此，请考虑在读取控制着色器执行的任何值后添加同步。
ERR_ALIAS_ARRAY_INDEX_OUT_OF_BOUNDS 4027	数组索引超出边界。
ERR_MINPRECISION_PRECISE 4028	指定的变量具有最小精度类型，不能标记为精确。
ERR_LOOP_NEVER_BREAKS 4029	检测到无限循环，因此循环永远不会退出。
WARN_FTOI_OUTOFRANGE 4114	文本浮点值不是转换的整数范围。
WARN_FTOU_OUTOFRANGE 4115	文本浮点值与转换的无符号整数范围无关。
WARN_IDIV_DIVISION_BY_ZERO 4116	可能发生的整数除以零。

常量/值	说明
WARN_UDIV_DIVISION_BY_ZERO 4117	发生可能的无符号整数除以零。
WARN_IMAGINARY_SQUARE_ROOT 4118	发生虚方根操作。
WARN_INDEFINITE_LOG 4119	发生了无限期对数运算。
WARN_REPLACE_NOT_CONVERGE 4120	优化不会聚合。
WARN_HOISTING_GRADIENT 4121	必须从流控制中移出基于渐变的操作，以防止差异。 使用非渐变操作可以提高性能。
WARN_FLOAT_PRECISION_LOSS 4122	不能以双精度准确表示两个浮点值的总和。
WARN_FLOAT_CLAMP 4123	浮点操作将非序浮点文本刷新为零，因此指定的浮点值丢失精度 (此警告只会在每个编译) 显示一次。
ERR_GEN_NOT_YET_IMPLEMENTED 4500	尚未实现从重排向量剪辑等功能。
ERR_DUPLICATE_INPUT_SEMANTIC 4501	发生了不一致的语义定义。
ERR_INVALID_INPUT_SEMANTIC 4502	指定的输入语义无效。
ERR_INVALID_OUTPUT_SEMANTIC 4503	指定的输出语义无效。
ERR_DUPLICATE_OUTPUT_SEMANTIC 4504	出现重叠的输出语义。
ERR_MAX_TEMP_EXCEEDED 4505	超出最大临时寄存器索引。
ERR_MAX_INPUT_EXCEEDED 4506	超出输入的最大数目。
ERR_MAX_CONST_EXCEEDED 4507	超出最大常量寄存器索引。 尝试减少引用的常量数。
ERR_MAX_ADDR_EXCEEDED 4508	超出最大地址寄存器索引。

常量/值	说明
ERR_GEN_INVALID_REGISTER_SEMANTIC 4509	使用了无效的寄存器语义，或者变量必须绑定到多个寄存器银行。
ERR_MAX_SAMPLER_EXCEEDED 4510	超过采样器的最大数目。
ERR_REL_ADDRESS_NOT_SUP 4511	目标不支持相对寻址。
ERR_NO_W_ACCESS 4512	无法访问纹理坐标 w-component。
ERR_NO_DEP_FROM_COL 4513	无法执行任何方式基于颜色输入的依赖纹理读取操作。
ERR_PROGRAM_TOO_BIG 4514	该计划太大了。
ERR_CANNOT_BIND_SAMPLER 4515	采样器不能绑定到用户指定的阶段或采样器数组。
ERR_CANNOT_READ_SAME_TEX 4516	无法从采样器中读取用作输入的 texcoord。
ERR_CONFLICT_SAMP_BIND 4517	用户定义的采样器或采样器数组绑定冲突。如果两个采样器具有相同的用户绑定，则它们不能用于同一着色器。
ERR_MULTI_READ_SAMP_BIND 4518	无法从用户绑定或类似的数组访问采样器执行纹理查找两次。
ERR_TOO_MANY_TEXREADS 4519	纹理加载和读取次数过多，来自 texcoords。
ERR_NO_TEXCRD_SHARE 4520	texcoord 只能在 ps_1_4 和更高版本中读取和用于 texlookup。
ERR_OUT_OF_TEMP 4521	程序过于复杂，并且不是临时寄存器。
ERR_NO_REP_SWIZZLE 4522	复制重排仅在ps_1_4中受支持。
ERR_NO_DEP_MATCH 4523	此依赖纹理读取无法映射到ps_1_x，或者着色器无法编译为ps_1_x着色器，因为此模型无法匹配此着色器读取此着色器所需的所有依赖纹理。
ERR_TEXM_NO_SHARE 4524	无法匹配 texm，因为计算的texcoord 用于着色器。

常量/值	说明
ERR_TEXM_NOT_COR_STAGE 4525	无法匹配 texm*，因为源输入不在适当的纹理坐标中。 有关详细信息，请参阅 ps_1_x 程序集参考。
ERR_TEXM_NO_SOURCE_MOD 4526	无法匹配 texm*，因为 texm* 不能在输入 texcoord 上具有源修饰符。
ERR_TEXM_BX2_ONLY 4527	texm* 无法匹配，因为 texm* 只能在输入 texload 上具有 bx2 修饰符。
ERR_DEPTH_SCALAR 4528	DEPTH 必须是标量。
ERR_COLOR_4COMP 4529	语义 (SV_Target或 COLOR) 值必须是四分量向量。
ERR_WRITE_TO_COLOR0 4530	像素着色器必须至少写入语义 (SV_Target0 或 COLOR0) 值的所有四个组件。
ERR_DP4_NOT_SUP 4531	不支持 DP4。
ERR_NO_MATCH 4532	无法将表达式映射到着色器指令集。
ERR_NO_SWIZZLE_MATCH 4533	无法将重排映射到ps_1_x。
ERR_NO_DOUBLE_DEP 4534	无法在ps_1_x中执行双依赖纹理读取。
ERR_NO_TEX_SOURCE 4535	无法匹配 texreg2ar 或 texreg2gb 指令，因为不能有输入修饰符。
ERR_TEXRGB_NOT_SUPPORTED 4536	表达式只能映射到 texreg2rgb，但1_x不支持此指令。
ERR_CANT_EMMULLATE_WRITE 4537	无法模拟ps_1_x着色器模型的写入掩码。
ERR_COLOR_CONT 4538	SV_Target输出必须从SV_Target0连续到 SV_TargetN，或者 COLOR 输出必须从 COLOR0 连续到 COLORn。
ERR_SAMPLER_MISMATCH 4539	由于采样器使用不一致，导致采样器不匹配。
ERR_SEMANTIC_SCALER 4540	PSIZE 或 FOG 必须是标量。

常量/值	说明
ERR_WRITE_ALL_POS 4541	顶点着色器必须至少写入SV_Position或 POSITION 的所有四个组件。
ERR_TEXCOORD_CONT 4542	Texcoord 输出必须连续从 texcoord0 到 texcoordn。
ERR_NO_MULTI_SEM 4543	片段不支持多寄存器语义。
ERR_NO_4COMP_CLIP 4544	剪辑必须来自ps_1_x中的 3 个向量。
ERR_TEXTURE_NOT_SUPPORTED 4545	遇到指定目标的不受支持的纹理类型。
ERR_MAX_SAMP_EXCEEDED 4546	超过了最大采样器寄存器索引。
ERR_DEBUG_SIZE 4547	调试信息超出了最大注释大小，因此未发出调试信息。
ERR_CONSTANTTABLE_SIZE 4548	常量表信息超过最大注释大小。
ERR_MAX_PRED_EXCEEDED 4549	超过了最大谓词寄存器索引。
ERR_MAX_BOOL_EXCEEDED 4550	尝试减少常量分支的数量，从结构/数组中取出布尔值，或将其移动到结构的开头。
ERR_MAX_LOOP_EXCEEDED 4551	尝试减少循环数，将循环计数器从结构/数组中取出，或将它们移到结构的开头。
ERR_NOT_SIMPLE_FOR 4552	无法将常规循环映射到此指令集。
ERR_ADDRESS_TOO_DEEP 4553	相对地址引用太深。
ERR_CND_SCALAR 4554	无法在ps_1_x着色器模型中模拟矢量条件。
ERR_INVALID_TYPE 4555	用于指定语义的无效类型。
ERR_MAX_TEXTURE_EXCEEDED 4565	库超出了纹理槽的最大数目。
ERR_REQUIRE_INT_OFFSET 4566	偏移纹理指令必须采用偏移量，该偏移量可以解析为范围 -8 到 7 中的整数文本。

常量/值	说明
ERR_MAX_CBUFFER_EXCEEDED 4567	库超出最大常量缓冲区槽数。
ERR_INCORRECT_USAGE 4568	目标不支持使用。 例如，示例内插、无插值、无规范或整数输入用法可能不受支持。
ERR_POSITION_INCORRECTTYPE 4569	为 POSITION 值指定了不正确的类型。
ERR_MULTIPLE_STREAMS 4570	目标只能发出到特定数量的流。
ERR_MAX_OUTPUT_EXCEEDED 4571	超出输出限制。
ERR_NO_STREAMS_USED 4572	几何着色器未发出任何内容。
ERR_GEN_SEMANTIC_TOO_LONG 4573	语义长度过长，限制为指定字符数。
ERR_DUPLICATE_SYSVAL_SEMANTIC 4574	遇到重复的系统值语义定义。
ERR_READING_UNINITIALIZED 4575	读取了未初始化的值。
ERR_SIGNATURE_VALIDATION 4576	签名验证期间出错。
ERR_INCOMPLETE_POSITION 4577	并非所有SV_Position元素都是写入的。
ERR_DUPLICATE_CBUFFER_BANK 4578	指定的 cbuffer 寄存器多次使用。
ERR_INVALID_FP_LITERAL 4579	发生了无效的浮点文本。
ERR_UNWRITTEN_SI_VALUE 4580	指定的输出包含必须在着色器的每个执行路径中写入的系统解释值。 无条件初始化可能会有所帮助。
ERR_AUTOSAMPLER_ARRAY_UNIMPL 4581	尚未实现对10_level_9 (9_x功能级别) 目标上的纹理对象的采样器数组。
ERR_INVALID_TEXTURE_FORMAT 4582	无法从非浮点纹理格式采样。

常量/值	说明
ERR_INVALID_10L9_SEMANTIC 4583	10_level_9 (9_x功能级别) 目标不支持指定的语义。
ERR_MAX_IFACE_EXCEEDED 4584	超过接口指针的最大数目。
ERR_MAX_UAV_EXCEEDED 4585	库超过了 UAV 槽的最大数目。
ERR_MAX_GROUP_SHARED_MEMORY_EXCEEDED 4586	组共享内存的总量（以字节为单位）超过了目标的限制。
ERR_TOO_MANY_GROUP_SHARED_DATA 4587	为指定目标编译的着色器只能有单个组共享数据项。
ERR_INCORRECT_NUM_GROUP_SHARED_ELEMENTS 4588	对指定目标的共享数据进行分组必须具有等于线程组中线程数的元素计数。
ERR_CONTROL_POINT_COUNT_EXCEEDED 4589	超出目标的最大控制点计数
ERR_GROUP_SHARED_DATA_ELEMENT_TOO_LARGE 4590	对指定目标的共享数据进行分组太大，在编译指定数量的线程时，必须具有最多指定字节量的元素大小。
ERR_GROUP_SHARED_DATA_NOT_AN_ARRAY 4591	对指定目标的共享数据进行分组必须是元素数组。
ERR_MULTI_SO_NOT_POINT 4592	使用多个几何图形着色器输出流时，它们必须是点列表。
ERR_INVALID_SNAP_OFFSET 4593	目标的对齐偏移量必须介于 -8 到 7 范围内。
ERR_CLIPPLANE_TOO_COMPLICATED 4594	无法在指定目标中寻址剪辑平面;或者，剪辑平面必须是具有指定目标中标识重排的非文本常量。
ERR_ONLY_ONE_ALLOC_CONSUME 4595	RWStructuredBuffer 对象可以递增或递减其计数器，但不能递减两者。
ERR_TYPED_UAV_WRITE_MASK_MISMATCH 4596	类型化 UAV 存储必须写入所有声明的组件。
ERR_TEX1D_UNSUPPORTED 4596	指定目标不支持 Texture1D 类型。
ERR_RESINFO_Z_UNDEFINED_CUBEARRAY 4598	TextureCubeArray 对象上 GetDimensions 的数组元素计数在指定目标上不可用。

常量/值	说明
ERR_INVALID_STRUCTURED_ELEMENT_SIZE 4599	结构化缓冲区元素大小无效。 它必须是指定目标中的多个指定字节，或者不能大于指定目标中的指定字节。
ERR_MAX_ICB_REG_EXCEEDED 4600	已超出着色器的可索引文本值。 着色器使用过多的可索引文本值，因此请考虑使用较少的常量数组。
ERR_MAX_CBUFFER_SIZE_EXCEEDED 4601	指定常量缓冲区的大小是指定的数字 16 字节条目，超过允许的最大条目大小。
ERR_LIB_DEBUG_INST_UNSUPPORTED 4602	着色器库中不支持调试指令。
ERR_VARYING_INDEXED_INTERFACE 4603	接口调用不能使用不同的值编制索引。
WAR_GEN_NOT_YET_IMPLEMENTED 4700	尚未实现功能。
WAR_BIAS_MISSED 4701	错过了一个_bias机会， 因为源没有固定 0 到 1。
WAR_COMP_MISSED 4702	缺少补充机会，因为输入结果从 0 固定到 1。
WAR_LRP_MISSED 4703	无法匹配 Lerp，因为 lerp 因子不会_sat。
WAR_MAX_CONST_RANGE 4704	范围 -1 到 1 之外的文本值将固定在所有 ps_1_x底纹模型上。
WAR_DEPRECATED_INPUT_SEMANTIC 4705	指定的输入语义已被弃用;请改用指定的语义。
WAR_DEPRECATED_OUTPUT_SEMANTIC 4706	指定的输出语义已被弃用;请改用指定的语义。
WAR_TEXCOORD_CLAMP 4707	直接 (使用的 texcoord 输入，除了从着色器体中的纹理) 采样之外，ps_1_x始终从 0 固定到 1。
WAR_MIDLEVEL_VARNOTFOUND 4708	找不到中级 var。
WAR_OLD_SEMANTIC 4710	语义不再使用。
WAR_DUPLICATE_SEMANTIC 4711	遇到重复的非系统值语义定义。

常量/值	说明
WAR_CANT_MATCH_LOOP 4712	无法匹配循环，因为循环计数不是来自整数类型。
WAR_BIAS_CLAMPED 4713	样本偏差值限制为 [-16.00， 15.99] 范围，因此请使用指定的值而不是此值。
WAR_CS_TEMP_EXCEEDED 4714	临时寄存器和可索引临时寄存器的总和超过了建议的线程总数，因此性能可能会降低。
WAR_UNWRITTEN_SI_VALUE 4715	发出系统解释的值，不能在着色器的每个执行路径中写入。
WAR_PSIZE_HAS_NO_SPECIAL_MEANING 4716	指定的语义对目标) 特征级别10_level_9 (9_x没有特殊意义。
WAR_DEPRECATED_FEATURE 4717	D3DCompiler_47.dll或更高版本已弃用效果。

要求

要求	Value
标头	CompErrors.h