**12 几何着色器**

如果不启用曲面细分这一环节，那么几何着色器这个可选阶段便会位于顶点着色器与像素着色器之间。

顶点着色器以顶点作为输入数据，而**几何着色器的输入数据则是完整的图元**。例如，如果要绘制三角形列表，则几何着色器程序实际将对列表中的每个三角形执行操作。几何着色器以每个三角形的3个顶点作为输入，且**输出的是对应的图元列表**。

**几何着色器的亮点是可以创建或销毁几何图形**，此功能使GPU可以实现一些有趣的效果。

比如，借助几何着色器可以将输入的图元扩展为一个或更多其他类型的图元，或者根据某些条件而选择不输出图元。几何着色器的输出图元类型不一定与输入图元的类型相同。例如，几何着色器的一个拿手好戏就是将一个点扩展为一个四边形。

几何着色器所输出的图元由顶点列表定义而成。**在退出顶点着色器时，必须将顶点的位置变换到齐次裁剪空间。**换言之，经过几何着色器阶段的处理后，我们就得到了位于其次裁剪空间中由一系列顶点所定义的多个图元。这些顶点会同样经历投影(齐次除法)与光栅化等后续步骤。

**12.1 编写几何着色器**

几何着色器的编写方式比较接近于顶点着色器和像素着色器，当然也存在区别。

[maxvertexcount(N)]

void ShaderName(PrimitiveType InputVertexType InputName[NumElements],

inout StreamOutputObject<OutputVertexType> OutputName)

**我们必须先指定几何着色器单次调用所输出的顶点数量最大值**(每个图元都会调用一次几何着色器)。可以使用属性语法来设置着色器定义之前的最大顶点数量：

[maxvertexcount(N)]

其中N是几何着色器单次调用所输出的顶点数量最大值。几何着色器每次输出的顶点个数都可能各不相同，但是这个数量却不能超过之前定义的最大值。处于对性能方面的考量，我们应当令maxvertexcount尽可能小。maxvertexcount在1-20时，性能较好。27-40时，性能将下降到峰值性能的50%。

**几何着色器有输入，输出共两个参数**(实际上它可以拥有更多的参数，但这是另一个主题了)几何着色器的输入参数必须是一个定义有特定图元的顶点数组-点应输入一个顶点，线条要输入两个顶点，三角形需输入3个顶点，线及其相邻图元为4个顶点，三角形及其相邻图元为6个顶点。**几何着色器的输入顶点类型即为顶点着色器输出的顶点类型。输入参数一定要以图元类型作为前缀，用以描述输入到几何着色器的具体图元类型**，前缀可以是以下类型之一：

1,point 2,line 3,triangle 4,lineadj 5,triangleadj

**输出参数一定要标有inout修饰符。另外，它必须是一种流类型**(即某种类型的流输出对象)。流类型存有一系列顶点，它们定义了几何着色器输出的几何图形。

几何着色器可以通过内置方法Append向输出流列表添加单个顶点:

void StreamOutputObject<OutputVertexType>::Append(OutputVertexType v);

流类型本质上是一种模板类型，其模板参数用以指定输出顶点类型的具体类型。流类型有如下3种：

1,PointStream<OutputVertexType>：一系列顶点所定义的点列表

2,LineStream<OutputVertexType>：一系列顶点所定义的线列表

3,TriangleStream<OutputVertexType>：一系列顶点所定义的三角形带

几何着色器输出的多个顶点会构成图元，图元的输出类型由流类型(即PointStream，LineStream与TriangleStream)来指定。

对于线条与三角形来说，几何着色器输出的对应图元必定是线条带与三角形带。而线条列表与三角形列表可借助内置函数RestartStrip来实现：

void StreamOutputObject<OutputVertexType>::RestartStrip();

比如，如果希望输出三角形列表，则需要在每次向输出流追加3个顶点之后调用RestartStrip。

(例子：将输入的三角形进行细分，并输出细分后的4个小三角形)

要指定的几何着色器作为流水线状态对象的一部分，以此将它绑定到渲染流水线上。

**12.2 以公告牌技术实现森林效果**

SV\_PrimitiveID语义

若指定该语义，则输入装配器阶段会自动为每个图元生成图元ID。在绘制n个图元的调用执行过程中，第一个图元被标记为0，以此类推。

**12.3 纹理数组**

12.3.1 概述

顾名思义，纹理数组即为存放纹理的数组。在C++代码中，纹理数组也由ID3D12Resource接口来表示。在创建ID3D12Resource对象时，可以通过设置DepthOrArraySize属性来指定纹理数组所存储的元素个数(对于3D纹理来说，此项设定的则为深度值)。

如果查看Common/DDSTextureLoader.cpp中的CreateD3Dresource12函数，便会明白这些代码究竟是如何来创建纹理数组与体纹理的。

在HLSL文件中，**纹理数组**是通过Texture2DArray类型来表示的。

12.3.2 对纹理数组进行采样

float3 uvw = float3(pin.TexC, pin.PrimID % 4)

float4 diffuseAlbedo = gTreeMapArray.Sample(gsamAnisotropicWrap, uvw) \* gDiffuseAlbedo;

使用纹理数组共需要3个坐标值：前两个坐标就是普通的2D纹理坐标，第三个坐标则是纹理数组的索引。

12.3.3 加载纹理数组

位于Common/DDSTextureLoader.h/.cpp文件中的代码，可以加载存有纹理数组的DDS文件。因此重点就落在了创建含有纹理数组的DDS文件上。为此，我们需要使用微软公司所提供的texassemble工具。

12.3.4 纹理子资源

Direct3D API用术语数组切片来表示纹理数组中的某个纹理及其mipmap链，又用术语mip切片来表示纹理数组中特定层级的所有mipmap。子资源则是指纹理数组中某纹理的单个mipmap层级。

这就是说，若给出纹理数组的索引以及mipmap层级，我们就能访问纹理数组中的相应子资源。查看D3D12CalcSubresource方法。

**12.4 alpha-to-coverage技术**

那到底有没有办法通知Direct3D，使它在计算覆盖情况时考虑alpha通道这个因素呢？回答是肯定的，这便是被称之为alpha-to-coverage。

在开启MSAA与alpha-to-coverage后，硬件就会检测像素着色器所返回的alpha值，并将其用于确定覆盖的情况。例如，在使用4xMSAA时，若像素着色器返回的alpha值为0.5，则我们即可认为此像素里4个子像素中的2个位于多边形的范围之外，并据此来创建平滑的图像边缘。

一般来说，在以alpha遮罩的方式来裁剪树叶与围栏这类纹理时，建议总是使用alpha-to-coverage技术。当然，前提是需要开启MSAA。