**6 利用Direct3D绘制几何体**

**1，顶点与输入布局**

为了构建自定义的顶点格式，我们首先要创建结构体：

struct Vertex1{

XMFLOAT3 Pos;

XMFLOAT4 Color;

}

还需要向Direct3D提供该顶点结构体的描述，用结构体**D3D12\_INPUT\_LAYOUT\_DESC**来表示。输入布局描述由**D3D12\_INPUT\_ELEMENT\_DESC**元素数组组成。

**2，顶点缓冲区**

为了使GPU可以访问顶点数组，就需要把它们放置在称为缓冲区的GPU资源(ID3D12Resource)里。我们把存储顶点的缓冲区叫做顶点缓冲区。

对于静态几何体而言(即每帧不会发生改变的几何体)，我们会将其顶点缓冲区置于默认堆中来优化性能。

CPU不能向默认堆中的顶点缓冲区写入数据。我们要创建一个处于中介位置的上传缓冲区资源。我们通过把资源提交至上传堆，才得以将数据从CPU复制到GPU显存中。

为了将顶点缓冲区绑定到渲染流水线上，我们需要给这种资源创建一个顶点缓冲区视图。无须为顶点缓冲区视图创建描述符堆。

在顶点缓冲区及其对应的视图创建完成后，便可以将它与渲染流水线上的一个输入槽相绑定。我们通过**IASetVertexBuffers**向流水线的输入装配器阶段传递顶点数据。

将顶点缓冲区设置到输入槽并不会对其执行实际的绘制操作，而是仅为顶点数据送至渲染流水线做好准备。通过**DrawInstanced**方法真正地绘制顶点。

**3，索引和索引缓冲区**

与顶点类似，为了使GPU可以访问索引数组，就需要将它们放置于GPU的缓冲区资源内。我们称存储索引的缓冲区为索引缓冲区。

为了使索引缓冲区与渲染流水线绑定，我们需要给索引缓冲区资源创建一个索引缓冲区视图。无须为索引缓冲区视图创建描述符堆。

与顶点缓冲区相似，在使用之前，我们需要先将它们绑定到渲染流水线上。通过IASetIndexBuffers方法即可将索引缓冲区绑定到输入装配器阶段。

在使用索引时，我们一定要用**DrawIndexedInstanced**方法代替DrawInstanced方法进行绘制。

**4，顶点着色器示例**

在Dirext3D中，编写着色器的语言为高级着色语言(HLSL)，其语法与C++十分相似，易于学习。一般情况下，着色器通常在以.hlsl为扩展名的文本文件中编写。

HLSL没有引用和指针概念，所以需要借助结构体或多个输入参数才能够从函数中返回多个数值。在HLSL中，所有的函数都是内联的。

SV\_POSITION修饰的顶点着色器输出元素存有齐次裁剪空间中的顶点位置信息。

我们可以把函数的返回类型和输入签名替换为结构体，从而取代过长的参数列表。

如果我们传入的顶点数据与顶点着色器所期望的输入不相符，便会导致错误。

**5，像素着色器示例**

为了计算三角形中每个像素的属性，我们会在光栅化处理期间对顶点着色器（或几何着色器）输出的顶点属性进行插值。随后，再将这些插值数据传至像素着色器中作为它的输入。

这些输入像素着色器的像素片段有可能最终不会传入或留存在后台缓冲区中。

**6，常量缓冲区**

**6.1，创建常量缓冲区**

与顶点缓冲区和索引缓冲区不同的是，常量缓冲区通常由CPU每帧更新一次。所以，我们会把常量缓冲区创建到一个上传堆而非默认堆中。

常量缓冲区对硬件也有特别的要求，即常量缓冲区的大小必为硬件最小分配空间(256B)的整数倍。

**6.2，更新常量缓冲区**

由于常量缓冲区是用D3D12\_HEAP\_TYPE\_UPLOAD这种堆类型来创建的，所以我们就能通过CPU为常量缓冲区资源更新数据。使用**Map**方法。

我们利用**memcpy**函数将数据从系统内存复制到常量缓冲区。

当常量缓冲区更新完成后，我们应在释放映射内存之前对其进行**Unmap**操作。

**6.3，根签名和描述符表**

在Direct3D中，根签名由**ID3D12RootSignature**接口来表示，并以一组描述绘制调用过程中着色器所需资源的根参数定义而成。

根参数可以是根常量，根描述符或者描述符表。

根签名只定义了应用程序要绑定到渲染流水线的资源，却没有真正地执行任何资源绑定操作。使用**SetGraphicsRootDescriptorTable**方法令描述符表与渲染流水线相绑定。

**7，编译着色器**

在Direct3D中，着色器程序必须先被编译为一种可移植的字节码。接下来，图形驱动程序将获取这些字节码，并将其重新编译为针对当前系统GPU所优化的本地指令。

我们可以在运行期间用**D3DCompileFromFile**函数对着色器进行编译。仅对着色器进行编译并不会使它与渲染流水线相绑定以供其使用。

离线编译：

除了在运行期间编译着色器，还能以单独的步骤离线地编译着色器。

为了以离线的方式编译着色器，我们将使用DirectX自带的FXC命令行编译工具或者Visual Studio离线编译着色器。

我们通常用.cso作为已编译着色器的扩展名。

通过查阅着色器的汇编代码，既可核对着色器的指令数量，也能了解生成的代码细节，这是为了验证编译器所生成的代码与我们预想的是否一致。

**8，光栅器状态**

如今渲染流水线中的大多数阶段都是可编程的，但是有些特定环节却只能接受配置。例如，用于配置渲染流水线中光栅化阶段的光栅器状态组由结构体**D3D12\_RASTERIZER\_DESC**表示。

**9，流水线状态对象**

大多数控制图形流水线状态的对象被统称为流水线状态对象(PSO)，用ID3D12PipelineState接口来表示。

并非所有的渲染状态都封装于PSO内，如视口和裁剪矩形等属性。

Direct3D实质上就是一种状态机，里面的事物会保持它们各自状态，直到我们将其改变。

如果把一个PSO与命令列表相绑定，那么，在我们设置另一个PSO或重置命令列表之前，会一直沿用当前的PSO绘制物体。

**10，实践：立方体演示程序**

**顶点，索引，常量，根签名**