**6 利用Direct3D绘制几何体**

**6.1 顶点与输入布局**

除了空间位置，Direct3D中的顶点还可以存储其他属性数据。为了构建自定义的顶点格式，我们首先要创建一个结构体来容纳选定的顶点数据。

定义了顶点结构体之后，我们还需要向Direct3D提供该顶点结构体的描述，使它了解应怎样来处理结构体中的每个成员。用户提供给Direct3D的这种描述被称为输入布局描述，用结构体D3D12\_INPUT\_LAYOUT\_DESC来表示。

**6.2 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作顶点缓冲区。缓冲区的结构比纹理更为简单：即非多维资源，也不支持mipmap、过滤器以及多重采样等技术。

我们先通过填写D3D12\_RESOURCE\_DESC结构体来描述缓冲区资源，接着调用ID3D12Device::CreateCommittedResource方法去创建ID3D12Resource对象。

对于静态几何体(即每一帧都不会发生改变的几何体)而言，我们会将其顶点缓冲区置于默认堆(D3D12\_HEAP\_TYPE\_DEFAULT)中来优化性能。

除了创建顶点缓冲区资源本身之外，我们还需要D3D12\_HEAP\_TYPE\_UPLOAD这种堆类型来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。在创建了上传缓冲区之后，我们就可以将顶点数据从系统内存复制到上传缓冲区，而后再把顶点数据从上传缓冲区复制到真正的顶点缓冲区中。

**6.3 索引和索引缓冲区**

与顶点相似，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。如同顶点缓冲区视图一样，我们也无须为索引缓冲区视图创建描述符堆。索引缓冲区视图由结构体D3D12\_INDEX\_BUFFER\_VIEW表示。

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ID3D12GraphicsCommandList::IASetIndexBuffer方法即可将索引缓冲区绑定到输入装配器阶段。

**6.4 顶点着色器示例**

在Direct3D中，编写着色器的语言为高级着色语言HLSL。

**6.5 像素着色器示例**

**6.6 常量缓冲区**

**6.6.1 创建常量缓冲区**

常量缓冲区也是一种GPU资源，其数据内容可供着色器程序所引用。用cbuffer对象定义常量缓冲区。

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。

常量缓冲区的大小必为硬件最小分配空间(256B)的整数倍。

**6.6.2 更新常量缓冲区**

通过Map方法获得指向欲更新资源数据的指针。

利用memcpy函数将数据从系统内存复制到常量缓冲区。

更新完成后，我们应在释放映射内存之前对其进行Unmap操作。

**6.6.3 上传缓冲区辅助函数**

**6.6.4 常量缓冲区描述符**

常量缓冲区描述符都要存放在以D3D12\_DESCRIPTOR\_HEAP\_TYPE\_CBV\_SRV\_UAV类型所创建的描述符堆里。

通过填写D3D12\_CONSTANT\_BUFFER\_VIEW\_DESC实例，再调用ID3D12Device::CreateConstantBufferView方法吗，便可创建常量缓冲区。

**6.6.5 根签名和描述符表**

通常来讲，在绘制调用开始执行之前，我们应将不同的着色器程序所需的各种类型的资源绑定到渲染流水线上。实际上，不同类型的资源会被绑定到特定的寄存器槽上，以供着色器程序访问。

根签名定义的是：在执行绘制命令之前，那些应用程序将绑定到渲染流水线上的资源，它们会被映射到着色器对应的输入寄存器。根签名一定要与使用它的着色器相兼容(即在绘制开始之前，根签名一定要为着色器提供其执行期间需要绑定到渲染流水上的所有资源)，在创建流水线状态对象时会对此进行验证。不同的绘制调用可能会用到一组不同的着色器程序，这也就意味着要用到不同的根签名。

在Direct3D中，根签名有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。

描述符表指定的是描述符堆中存有描述符的一块连续区域。

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。只要率先通过命令列表设置好根签名，我们就能用ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable方法令描述符表与渲染流水线相绑定。

**6.7 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。

我们可在运行期间用D3DCompileFromFile函数进行编译。

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。

**6.7.1 离线编译**

我们不仅可以在运行期间编译着色器，还能够以单独的步骤离线地编译着色器。

我们通常用.cso作为已编译着色器的扩展名。

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。

使用LoadBinary将.cso文件加载到应用程序中。

**6.7.2 生成着色器汇编代码**

**6.7.3 利用Visual Studio离线编译着色器**

**6.8 光栅器状态**

当今渲染流水线中的大多阶段都是可编程的，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D3D12\_RASTERIZER\_DESC表示。

比较常用的3个成员：

FillMode：线框渲染或实体渲染；

CullMode：剔除模式；

FrontCounterClockwise：三角形正面绕序；

**6.9 流水线状态对象**

大多数控制图形流水线状态的对象呗统称为流水线状态对象，用ID3D12PipelineState接口表示。要创建PSO，我们首先要填写一份描述其细节的D3D12\_GRAPHICS\_PIPELINE\_STATE\_DESC结构体实例。

填写完毕后，我们用ID3D12Device::CreateGraphicsPipelineState方法来创建ID3D12PipelineState对象。

I D3D12PipelineState对象集合了大量的流水线状态信息，为了保证性能，我们将所有这些对象都集总在一起，一并送至渲染流水线。

并非所有的渲染状态都封装与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的状态，直到我们将其改变。

**6.10 几何图形辅助结构体**

**6.11 立方体演示程序**