**利用Direct3D绘制几何体**

**顶点与输入布局**

**What:**

**Why:**

**How:**

**如何构建自定义的顶点格式？**

首先要创建一个结构体来容纳选定的顶点数据。

然后，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。

这种描述被称为输入布局描述，用结构体D3D12\_INPUT\_LAYOUT\_DESC来表示。输入布局描述由一个以D3D12\_INPUT\_ELEMENT\_DESC元素构成的数组组成。D3D12\_INPUT\_ELEMENT\_DESC数组中的元素依次描述了顶点结构体中所对应的成员。

**顶点缓冲区**

**What：**

我们把存储顶点的缓冲区叫作顶点缓冲区。

**Why：**

为了GPU可以访问顶点数组，需要把它们放在称为缓冲区的GPU资源（ID3D12Resource）里。

**How：**

**顶点的处理流程？**

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

2，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。

3，CPU不能向默认堆中的顶点缓冲区写入数据。因此，我们还需用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。

4，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。我们无须为顶点缓冲区视图创建描述符堆。

5，在顶点缓冲区及其对应视图创建完成后，便可以将它与渲染流水线上的一个输入槽相绑定。这样一来，我们就能向流水线中的输入装配器阶段传递顶点数据了。

6，通过ID3D12GraphicsCommandList::DrawInstanced方法绘制顶点。

**顶点处理流程简述**

1，顶点结构体和输入布局

2，创建上传堆资源和默认堆资源

3，创建顶点缓冲区，ID3D12Resource对象

4，创建描述符，

5，将描述符绑定到流水线

6，绘制顶点数据

索引和索引缓冲区

What:

我们称存储索引的缓冲区为索引缓冲区。

Why:

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（ID3D12Resource）内。

How:

索引的处理流程

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（index buffer view）。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。

与顶点缓冲区相似（也包括其他的Direct3D资源在内），在使用之前，我们需要先将它们绑定到渲染流水线上。

在使用索引的时候，我们一定要用ID3D12GraphicsCommandList::DrawIndexedInstanced方法代替DrawInstanced方法进行绘制。

常量缓冲区

What:

常量缓冲区也是一种GPU资源（ID3D12Resource），其数据内容可供着色器程序所引用。

Why:

How:

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，这样做能使我们从CPU端更新常量。

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。

待到绘制物体的时候，只要将常量缓冲区视图（Constant Buffer View，CBV）绑定到存有物体相应常量数据的缓冲区子区域即可。

更新常量缓冲区

我们首先要获得指向欲更新资源数据的指针，可用Map方法来做到这一点。

我们利用memcpy函数将数据从系统内存（system memory，也就是CPU端控制的内存）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（取消映射）操作。

将上传缓冲区的相关操作简单地封装一下，使用起来会更加方便。我们在UploadBuffer.h文件中定义了下面这个类。

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类型所建的描述符堆里。这种堆内可以混合存储常量缓冲区描述符、着色器资源描述符和无序访问（unordered access）描述符。

根签名和描述符表

What：

根签名（root signature）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的对应输入寄存器。

Why：

How：

根参数

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（root parameter）定义而成。根参数可以是根常量（root constant）、根描述符（root descriptor）或者描述符表（descriptor table）。

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（command list）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。

编译着色器

What：

Why：

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。

How：

我们可以在运行期间用下列函数对着色器进行编译。

HRESULT D3DCompileFromFile(

LPCWSTR pFileName,

const D3D\_SHADER\_MACRO \*pDefines,

ID3DInclude \*pInclude,

LPCSTR pEntrypoint,

LPCSTR pTarget,

UINT Flags1,

UINT Flags2,

ID3DBlob \*\*ppCode,

ID3DBlob \*\*ppErrorMsgs);

离线编译

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。

利用Visual Studio离线编译着色器

光栅器状态

What：

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（rasterizer state）组由结构体D3D12\_RASTERIZER\_DESC来表示

Why：

How：

流水线状态对象

What：

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（Pipeline State Object，PSO），用ID3D12PipelineState接口来表示。

Why：

How：

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

换句话说，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。

实践：立方体演示程序