**第4章 Direct3D的初始化**

**1，组件对象模型ComPtr的概念和常用方法**

Microsoft::WRL::ComPtr类，可以把它当作是COM对象的智能指针。

**2，交换链和页面翻转的概念**

为了避免动画中出现画面闪烁的现象，最好将动画帧完整地绘制在一种称为后台缓冲区的离屏纹理内。为此，需要利用硬件管理两种纹理缓冲区：前台缓冲区和后台缓冲区。

当后台缓冲区中的动画帧绘制完成之后，两种缓冲区的角色互换。

前台缓冲区和后台缓冲区构成了交换链，在Direct3D中用IDXGISwapChain接口来表示。

**3，深度缓冲的概念**

深度缓冲区这种纹理资源存储是特定像素的深度信息。深度值的范围为0-1。0代表在视椎体中能看到离自己最近的物体，1代表能看到最远的物体。

深度缓冲区中的元素与后台缓冲区内的像素呈一一对应关系，如果后台缓冲区的分辨率为1280x1024，那么深度缓冲区中就应当有1280x1024个深度元素。

若使用了深度缓冲，则物体的绘制顺序也就变得无关紧要了。

深度缓冲技术的原理是计算每个像素的深度值，并执行深度测试，具有最小深度值的像素会获得最终的胜利，它将被写入后台缓冲区中。

**4，描述符的概念和作用以及分类**

**描述符的概念**： GPU资源并非直接与渲染流水线相绑定，而是要通过一种名为描述符的对象来对它间接引用。视图和描述符是同义词。

**本书常用的描述符如下：**

1，CBV/SRV/UAV描述符分别表示的是常量缓冲区视图，着色器资源视图和无序访问视图这3种资源；

2，采样器描述符表示的是采样器资源（用于纹理贴图）。

3，RTV描述符表示的是渲染目标视图资源。

4，DSV描述符表示的是深度/模板视图资源。

**描述符堆的概念**

**描述符堆**中存有一系列描述符，本质上是存放用户程序中某种特定类型描述符的一块内存。

我们需要为每一种类型的描述符都创建出单独的描述符堆。也可以为同一种描述符类型创建出多个描述符堆。

**5，多重采样技术的原理**

在不能提升显示器分辨率或在显示器分辨率受限的情况下，我们就可以运用各种反走样（也有译为抗锯齿）技术。

**超级采样**（SSAA）使用4倍于屏幕分辨率大小的后台缓冲区和深度缓冲区。当数据要从后台缓冲区调往屏幕显示的时候，会将后台缓冲区按4个像素一组进行解析。每组用求平均值的方法得到一种相对平滑的像素颜色。

**多重采样**（MSAA）同样使用4倍于屏幕分辨率的后台缓冲区和深度缓冲区。多重采样并不需要对每一个子像素都进行计算，而是仅计算一次像素中心处的颜色，再基于可见性和覆盖性将得到的颜色信息分享给其子像素。

**6，利用Direct3D进行多重采样**

根据给定的纹理格式和采样数量，我们就能用ID3D12Device::CheckFeatureSupport方法查询到对应的质量级别。

通常会把采样数量设定为4或8。如果不希望使用多重采样，则可将采样数量设为1，并令质量级别为0.

当创建后台缓冲区和深度缓冲区时，多重采样的有关设置一定要相同。

**7，功能级别**

功能级别为不同级别所支持的功能进行了严格的界定，每个功能级别所支持的特定功能可参见SDK文档。

如果用户的硬件不支持某特定功能级别，应用程序理当回退至版本更低的功能级别。

**8，DirectX图形基础结构**

IDXGIFactory可以枚举**显示适配器**。显通常来说，显示适配器是一种硬件设备（例如独立显卡），然而系统也可以用软件显示适配器来模拟硬件的图形处理功能。一个系统中可能会存在数个适配器（比如装有数块显卡），显示适配器用接口IDXGIAdapter来表示。

一个系统也可能装有数个显示设备，我们称每一台显示设备都是一个显示输出实例（适配器输出），用IDXGIOutput接口表示。每个适配器都与一组**显示输出**相关联。

每种显示设备都有一系列它所支持的**显示模式**，可以用DXGI\_MODE\_DESC结构体中的数据成员表示。

在进入全屏模式之时，枚举显示模式就显得尤为重要。为了获得最优的全屏性能，我们所指定的显示模式（包括刷新率）一定要与显示器支持的显示模式完全匹配。

**9，功能支持的检测**

HRESULT ID3D12Device::CheckFeatureSupport(

D3D12\_FEATURE Feature, void \*pFeatureSupportData,

UINT FeatureSupportDataSize

)

**10，资源驻留**

一般来说，资源在创建时就会驻留在显存中，而当它被销毁时则清出。但是通过下面方法，我们可以自己控制资源的驻留。

HRESULT ID3D12Device::MakeResident(UINT NumObjects, ID3D12Pageable \*const \*ppObjects)

HRESULT ID3D12Device::Evict(UINT NumObjects, ID3D12Pageable \*const \*ppObjects)

**4.2 CPU与GPU间的交互**

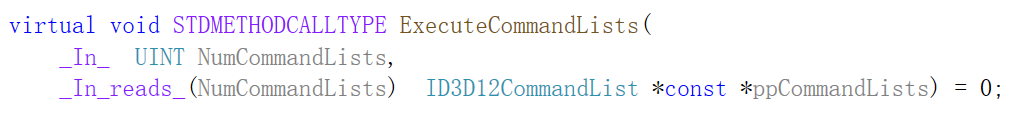
**1，命令队列**

**命令队列**：每个GPU都至少维护着一个命令队列。CPU可利用命令列表将命令提交到这个队列中去。

命令队列被抽象为ID3D12CommandQueue接口表示。

**2，命令列表**

利用ExecuteCommandLists可将命令列表里的命令添加到命令队列之中：

ID3D12GraphicsCommandList接口封装了一系列图形渲染命令，它实际上继承于ID3D12CommandList接口。

向命令列表中添加设置视口，清除渲染目标视图和发起绘制调用的命令：

mCommandList->RSSetViewports(1,&mScreenViewPort);

mCommandList->ClearRenderTargetView(mBackBufferView, Colors::LightSteelBlue, 0, nullptr)

mCommandList->DrawIndexedInstanced(36, 1, 0, 0, 0)

当命令都被加入命令列表之后，我们必须调用ID3D12GraphicsCommandList::Close方法来结束命令的记录。在调用ID3D12CommandQueue::ExecuteCommandLists方法提交命令列表之前，一定要将其关闭。

**3，命令分配器**

记录在命令列表内的命令，实际上是存储在与之关联的命令分配器ID3D12CommandAllocator上。

我们可以创建出多个关联于同一命令分配器的命令列表，但是不能同时用它们来记录命令。当其中一个命令列表在记录命令时，必须关闭同一命令分配器的其他命令列表。

重置命令列表并不会影响命令队列中的命令，因为相关的命令分配器仍在维护着其内存中被命令队列所引用的系列命令。

向GPU提交了一整帧的渲染命令后，我们可能还要为了绘制下一帧而复用命令分配器中的内存。ID3D12CommandAllocator::Reset方法由此应运而生。在没有确定GPU执行完命令分配器中的所有命令之前，千万不要重置命令分配器。

**4，CPU与GPU间的同步**

CPU更改了资源R的位置信息，等待GPU使用，但是在GPU使用之前，CPU再次改变了R的位置信息，会造成一个严重的错误。解决此问题的一种方法是：强制CPU等待，直到GPU完成所有命令的处理，达到某个指定的围栏点为止。

**5，资源转换**

对某个资源执行写操作时，需要将它的状态转换为渲染目标状态；而要对资源进行读操作时，再把它的状态变为着色器资源状态。

CD3DX12\_RESOURCE\_BARRIER继承自D3D12\_RESOURCE\_BARRIER结构体，并添加了一些辅助方法，定义在d3d12x.h头文件中，这个文件不属于SDK核心部分，需要自己下载。

mCommandList->ResourceBarrier(1,

&CD3CX12\_RESOURCE\_BARRIER::Transition(CurrentBackBuffer(),D3D12\_RESOURCE\_STATE\_PRESENT, D3D12\_RESOURCE\_STATE\_RENDER\_TARGET))

**6，命令与多线程**

在多线程环境中使用命令列表要注意的问题：

1，命令列表并非自由线程对象。

2，命令分配器也不是线程自由的。

3，命令队列是线程自由的。

4，出于性能原因，应用程序必须在初始化期间，指出用于并行记录命令的命令列表最大数量。

**4.3 初始化Direct3D**

我们对Direct3D进行初始化的过程可以分为以下几个步骤：

1. **创建设备**
2. **创建围栏并获取描述符的大小**
3. **检测对4X MSAA质量级别的支持**
4. **创建命令队列和命令列表**
5. **描述并创建交换链**
6. **创建描述符堆**
7. **创建渲染目标视图**
8. **创建深度/模板缓冲区及其视图**

**9 设置视口**

我们通常会将3D场景绘制到与整个屏幕（在全屏模式下）或整个窗口工作区大小相当的后台缓冲区中。但是，有时只是希望把3D场景绘制到后台缓冲区的某个矩形子区域当中。

我们把后台缓冲区中的这种矩形子区域叫做视口。

可以用视口技术来实现双人游戏的分屏模式。

**10，设置裁剪矩形**

我们可以相对于后台缓冲区定义一个裁剪矩形，在此矩形外的像素都将被剔除（即这些图像部分将不会被光栅化至后台缓冲区）。这个方法能用于优化程序的性能。例如，假设已知有一个矩形的UI元素覆于屏幕中某块区域的最上层，那么我们就无需对3D空间中哪些被他遮挡的像素进行处理类。

**4.4 计时与动画**

**1，性能计时器**

获取性能计时器测量当前时刻值（以计数为单位）：

\_\_int64 currTime;

QueryPerformanceCounter((LARGE\_INTEGER\*)&currTime);

获取性能计时器的频率（单位：计数/秒）：

\_\_int64 countsPerSec;

QueryPerformanceFrequency((LARGE\_INTEGER\*)&countsPerSec);