04 Direct3D的初始化

4.1 预备知识

Direct3D 12较之上一个版本的主要改变在于其性能优化方面在大大减少了CPU开销的同时，又改进了对多线程的支持。为了达到这些性能目标，Direct3D 12的 API较之Direct3D 11更偏于底层。

组件对象模型

组件对象模型(COM)是一种令DirectX不受编程语言束缚，并且使之向后兼容的技术。我们通常将COM对象视为一种接口，但考虑当前变成的目的，遂将它当做一个C++类来使用。

要获取指向某COM接口的指针，需借助特定函数或另一COM接口的方法，而不是用C++语言中的关键字new去创建一个COM接口。

COM对象会统计其引用次数，因此，在使用完某接口时，我们便应调用它的Release方法，而不是用delete来删除，当COM对象的引用计数为0时，它将执行释放自己所占用的内存。

为了辅助用户管理COM对象的生命周期，Windows运行时库专门提供了Miscrosoft::WRL::ComPtr类，我们可以把它当作是COM对象的智能指针。

常用的3个ComPtr方法如下：

1. Get：返回一个指向此底层COM接口的指针。此方法常用于把原始的COM接口指针作为参数传递给函数；
2. GetAddressOf：返回指向此底层COM接口指针的地址。凭此方法即可利用函数参数返回COM接口的指针；
3. Reset：将此ComPtr实例设置为nullptr释放与之相关的所有引用。此方法的功能与将ComPtr实例赋值为nullptr效果相同。

纹理格式

纹理只能存储DXGI\_FORMAT枚举类型中描述的特定格式的数据元素。

1. DXGI\_FORMAT\_R32G32B32\_FLOAT;
2. DXGI\_FORMAT\_R16G16B16A16\_UNORM;
3. DXGI\_FORMAT\_R32G32\_UINT;
4. DXGI\_FORMAT\_R8G8B8\_UNORM;
5. DXGI\_FORMAT\_R8G8B8A8\_SNORM;
6. DXGI\_FORMAT\_R8G8B8A8\_SINT;
7. DXGI\_FORMAT\_R8G8B8A8\_UINT;

U表示无符号，S表示有符号，NORM表示[0,1]。

还有无类型格式的纹理，我们仅用它来预留内存。待纹理被绑定到渲染流水线之后，再具体解释它的数据类型。例如，DXGI\_FORMAT\_R16G16B16A16\_TYPELESS。

交换链和页面翻转

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。只要将指定动画帧的整个场景绘到后台缓冲区中，它就会以一个完整的帧画面展现在屏幕上；依照此法，观者便不会察觉出帧的绘制过程——而只会观赏到完整的动画帧。为此，需要利用由硬件管理的两种纹理缓冲区：即所谓的前台缓冲区（front buffer）和后台缓冲区（back buffer）。前台缓冲区存储的是当前显示在屏幕上的图像数据，而动画的下一帧则被绘制在后台缓冲区里。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：后台缓冲区变为前台缓冲区呈现新一帧的画面，而前台缓冲区则为了展示动画的下一帧转为后台缓冲区，等待填充数据。前后台缓冲的这种互换操作称为呈现。呈现是一种高效的操作，只需交换指向当前前台缓冲区和后台缓冲区的两个指针即可实现。

前台缓冲区和后台缓冲区构成了交换链（swap chain），在Direct3D中用IDXGISwapChain接口来表示。这个接口不仅存储了前台缓冲区和后台缓冲区两种纹理，而且还提供了修改缓冲区大小（IDXGISwapChain::ResizeBuffers）和呈现缓冲区内容（IDXGISwapChain::Present）的方法。

使用两个缓冲区（前台和后台）的情况称为双缓冲。

深度缓冲

深度缓冲区（depth buffer）这种纹理资源存储的并非图像数据，而是特定像素的深度信息。深度值的范围为0.0～1.0。0.0代表观察者在视锥体（view frustum，亦有译作视域体、视景体、视截体或视体等，意即观察者能看到的空间范围，形如从四棱锥中截取的四棱台，常称该形为平截头体（frustum，见图4.3，后文亦有详述））中能看到离自己最近的物体，1.0则代表观察者在视锥体中能看到离自己最远的物体。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（即后台缓冲区中第行第列的元素对应于深度缓冲区内第行第列的元素）。所以，如果后台缓冲区的分辨率为，那么深度缓冲区中就应当有个深度元素。

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。

总而言之，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（depth test）。而深度测试则用于对竞争写入后台缓冲区中同一像素的多个像素深度值进行比较。具有最小深度值的像素（说明该像素离观察者最近）会获得最终的胜利，它将被写入后台缓冲区中。这样做也是合乎情理的，因为离观察者较近的像素无疑会遮挡其后面的像素。

资源与描述符

在发出绘制命令之前，我们需要将于本次绘制调用相关的资源绑定到渲染流水线上。部分资源可能在每次绘制调用时都会有所变化，所以我们也就要每次按需更新绑定。但是，GPU资源并非直接与渲染流水线相绑定，而是通过一种名为描述符的对象来对它间接引用，我们可以把描述符视为一种对送往GPU的资源进行描述的轻量级结构。

为什么我们要额外使用描述符这个中间层呢？究其原因，GPU资源实质都是一些普通的内存块。由于资源的这种通用性，它们便能被设置到渲染流水线的不同阶段供其使用。一个常见的例子是先把纹理用作渲染目标，随后再将该纹理作为一个着色器资源。不管是充当渲染目标、深度/模板缓冲区还是着色器资源等角色，仅靠资源本身是无法体现出来的。而且，我们有时也许只希望将资源中的部分数据绑定至渲染流水线，但如何从整个资源中将它们选取出来呢？再者，创建一个资源可能用的是无类型格式，这样的话，GPU甚至不会知道这个资源的具体格式。

解决上述问题就是引入描述符的原因。除了指定资源数据，描述符还会为GPU解释资源：它们会告知Direct3D某个资源将如何使用（即此资源将被绑定在流水线的哪个阶段上），而且我们可借助描述符来指定欲绑定资源中的局部数据。这就是说，如果某个资源在创建的时候采用了无类型格式，那么我们就必须在为它创建描述符时指明其具体类型。

“常量缓冲区视图（constant buffer view）”与“常量缓冲区描述符（constant buffer descriptor）”表达的是同一事物。

每个描述符都有一种具体类型，此类型指明了资源的具体作用。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（constant buffer view）、着色器资源视图（shader resource view）和无序访问视图（unordered access view）这3种资源。

2．采样器（sampler，亦有译为取样器）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（render target view）。

4．DSV描述符表示的是深度/模板视图资源（depth/stencil view）。

描述符堆（descriptor heap）中存有一系列描述符（可将其看作是描述符数组），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。另外，也可以为同一种描述符类型创建出多个描述符堆。

我们能用多个描述符来引用同一个资源。例如，可以通过多个描述符来引用同一个资源中不同的局部数据。而且，前文曾提到过，一种资源可以绑定到渲染流水线的不同阶段。因此，对于每个阶段都需要设置独立的描述符。例如，当一个纹理需要被用作渲染目标与着色器资源时，我们就要为它分别创建两个描述符：一个RTV描述符和一个SRV描述符。类似地，如果以无类型格式创建了一个资源，又希望该纹理中的元素可以根据需求当作浮点值或整数值来使用，那么就需要为它分别创建两个描述符：一个指定为浮点格式，另一个指定为整数格式。

创建描述符的最佳时机为初始化期间。由于在此过程中需要执行一些类型的检测和验证工作，所以最好不要在运行时（runtime）才创建描述符。

多重采样技术的原理

**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)**

通过跨子像素共享一些计算信息，从而使它比超级采样的开销更低。这种技术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，再基于可视性和覆盖性将得到的颜色信息分享给其子像素。

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。只要了解所支持的功能集，就能知道有哪些Direct3D功能可供使用。

DirectX图形基础结构

DirectX图形基础结构是一种与Direct3D配合使用的API。设计DXGI的基本理念是使多种图形API中所共有的底层任务能借助一组通用的API来进行处理。

IDXGIFactory是DXGI中的关键接口之一，主要用于创建IDXGISwapChain接口以及枚举显示适配器。而显示适配器则真正实现了图形处理能力。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。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，用IDXGIOutput接口来表示。每个适配器都与一组显示输出相关联。

功能支持的检测

ID3D12Device::CheckFeatureSupport

资源驻留

应用程序通过控制资源在显存中的去留，主动管理资源的驻留情况。

HRESULT ID3D12Device::MakeResident()

HRESULT ID3D12Device::Evict()

CPU与GPU间的交互

为了获得最佳性能，最好的情况是让CPU和GPU尽量同时工作，少同步。同步意味着一种处理器要以空闲状态等待另一种处理器。同步破坏了两者并行工作的机制。

命令队列和命令列表

每个CPU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。当一系列命令被提交至命令队列时，它们并不会被GPU立即执行，理解这一点至关重要。由于GPU可能正在处理先前插入命令队列中的命令，因此，后来新到的命令会一直在这个队列之中等待执行。

在Direct3D 12中，命令队列被抽象为ID3D12CommandQueue接口来表示。要通过填写D3D12\_COMMAND\_QUEUE\_DESC结构体来描述队列，再调用ID3D12Device::CreateCommandQueue方法创建队列。

IID\_PPV\_ARGS辅助宏

得到的即为ID3D12CommandQueue接口的COM ID。IID\_PPV\_ARGS辅助函数的本质是将ppType强制转换为\*\*void类型。

利用ExecuteCommandLists接口，可将命令列表中的命令添加到命令队列中。

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。ID3D12GraphicsCommandList接口有数种方法向命令列表添加命令。

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。

ID3D12CommandAllocator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上。当通过ID3D12CommandQueue：：ExecuteCommandLists方法执行命令列表时，命令队列就会引用分配器里的命令。命令分配器通过ID3D12Device::CreateCommandAllocator创建。命令列表通过ID3D12Device::CreateCommandList接口创建。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，必须关闭其他命令列表。

在没有确定GPU执行完命令分配器中所有命令之前，千万不要重置命令分配器。

CPU与GPU间的同步

向命令队列添加命令之后，CPU不会阻塞，如果这时复写命令队列中数据的信息会造成严重错误。解决的方法是：强制CPU等待，直到GPU完成所有的命令处理，达到某个指定的围栏点为止。我们将这种方法称为刷新命令队列，可以通过围栏来实现这一点。围栏用ID3D12Fence表示，此技术能用于实现GPU和CPU间的同步。每个围栏对象都维护着一个UINT64类型的值，此为用来标志围栏点的整数。

资源转换

当GPU的写操作还没有完成亦或甚至还没有开始，却开始读取资源，便会导致资源冒险。为此，Direct3D专门针对资源设计了一组相关状态。资源在创建伊始会处于默认状态，该状态将一直持续到应用程序通过Direct3D将其转换为另一种状态为止。通过命令列表设置转换资源屏障数组，即可指定资源的转换。

CD3DX12作为前缀的变体添加了很多辅助方法。

初始化Direct3D

Direct3D进行初始化的几个步骤：

1．用D3D12CreateDevice函数创建ID3D12Device接口实例。

2．创建一个ID3D12Fence对象，并查询描述符的大小。

3．检测用户设备对4X MSAA质量级别的支持情况。

4．依次创建命令队列、命令列表分配器和主命令列表。

5．描述并创建交换链。

6．创建应用程序所需的描述符堆。

7．调整后台缓冲区的大小，并为它创建渲染目标视图。

8．创建深度/模板缓冲区及与之关联的深度/模板视图。

9．设置视口（viewport）和裁剪矩形（scissor rectangle）。

**性能计时器**

**What:**

**Why:**

精确地度量时间

**How:**

**性能计时器如何使用？**

QueryPerformanceCounter函数获取性能计时器测量的当前时刻值（以计数为单位）

再用QueryPerformanceFrequency函数来获取性能计时器的频率（单位：计数/秒）