非框架方法

1. D3DApp：构造函数简单地将数据成员初始化为默认值。

2. ~D3DApp：析构函数释放D3DApp获取的COM接口，并刷新命令队列。我们需要在析构函数中刷新命令队列的原因是，我们需要等到 GPU 处理完队列中的命令后，才能销毁 GPU 仍在引用的任何资源。否则，当应用程序退出时，GPU 可能会崩溃。

3. AppInst：平凡访问函数返回应用程序实例句柄的副本。

4. MainWnd：平凡访问函数返回主窗口句柄的副本。

5. AspectRatio：纵横比定义为后台缓冲区宽度与其高度的比率。纵横比将在下一章中使用。它被简单地实现为：

6. Get4xMsaaState：如果启用了 4X MSAA，则返回 true，否则返回 false。

7. Set4xMsaaState：启用/禁用 4X MSAA。

8. 运行：该方法包装了应用程序消息循环。它使用 Win32 PeekMessage 函数，以便在不存在消息时处理我们的游戏逻辑。该函数的实现见§4.4.3。

9. InitMainWindow：初始化主应用程序窗口；我们假设读者熟悉基本的 Win32 窗口初始化。

10. InitDirect3D：通过实现第 4.3 节中讨论的步骤来初始化 Direct3D。

11. CreateSwapChain：创建交换链（第 4.3.5 节）。

12. CreateCommandObjects：创建命令队列、命令列表分配器和命令列表，如第 4.3.4 节所述。

13. FlushCommandQueue：强制 CPU 等待，直到 GPU 处理完队列中的所有命令（参见第 4.2.2 节）。

14. CurrentBackBuffer：返回一个ID3D12Resource到交换链中的当前后台缓冲区。

15. CurrentBackBufferView：将RTV（渲染目标视图）返回到当前后台缓冲区。

16. DepthStencilView：将 DSV（深度/模板视图）返回到主深度/模板缓冲区。

17.CalculateFrameStats：计算平均每秒帧数和平均每帧毫秒数。该方法的实现在第 4.4.4 节中讨论。

18. LogAdapters：枚举系统上的所有适配器（第 4.1.10 节）。

19. LogAdapterOutputs：枚举与适配器关联的所有输出（第 4.1.10 节）。

20. LogOutputDisplayModes：枚举输出支持给定格式的所有显示模式（第 4.1.10 节）。

框架方法

1.初始化：使用该方法放置应用程序的初始化代码，如分配资源、初始化对象、设置3D场景等。

该方法的D3DApp实现调用InitMainWindow和InitDirect3D；因此，您应该首先在派生实现中调用此方法的 D3DApp 版本，如下所示：

2. MsgProc：该方法实现主应用程序窗口的窗口过程功能。通常，只有当您需要处理 D3DApp::MsgProc 未处理（或不符合您的喜好）的消息时，您才需要覆盖此方法。

此方法的 D3DApp 实现在 §4.5.5 中进行了探讨。如果您覆盖此方法，则您未处理的任何消息都应转发到 D3DApp::MsgProc。

3. CreateRtvAndDsvDescriptorHeaps：您可以在其中创建应用程序所需的 RTV 和 DSV 描述符堆的虚拟函数。

默认实现创建一个带有 SwapChainBufferCount 多个描述符（用于交换链中的缓冲区）的 RTV 堆和一个带有一个描述符（用于深度/模板缓冲区）的 DSV 堆。

对于我们的许多演示，默认实现就足够了；对于使用多个渲染目标的更高级渲染技术，我们将不得不覆盖此方法。

4. OnResize：当接收到WM\_SIZE消息时，该方法由D3DApp::MsgProc调用。调整窗口大小时，某些 Direct3D 属性需要更改，因为它们取决于客户区尺寸。

特别是，需要重新创建后台缓冲区和深度/模板缓冲区以匹配窗口的新客户区。可以通过调用 IDXGISwapChain::ResizeBuffers 方法来调整后台缓冲区的大小。

深度/模板缓冲区需要销毁，然后根据新维度重新制作。此外，需要重新创建渲染目标和深度/模板视图。 OnResize 的 D3DApp 实现处理调整后端和深度/模板缓冲区大小所需的代码；

有关简单明了的详细信息，请参阅源代码。除了缓冲区之外，其他属性取决于客户区的大小（例如，投影矩阵），因此该方法是框架的一部分，因为在调整窗口大小时，客户端代码可能需要执行一些自己的代码.

5. 更新：这个抽象方法每帧都会被调用，并且应该用于随着时间的推移更新 3D 应用程序（例如，执行动画、移动相机、进行碰撞检测、检查用户输入等）。

6. Draw：这个抽象方法每帧都会被调用，我们在这里发出渲染命令以实际将当前帧绘制到后台缓冲区。绘制完框架后，我们调用 IDXGISwapChain::Present 方法将后台缓冲区呈现给屏幕。

帧统计

游戏和图形应用程序通常会测量每秒渲染的帧数 (FPS)。

为此，我们只需计算在某个指定时间段 t 内处理的帧数（并将其存储在变量 n 中）。

那么，时间段 t 内的平均 FPS 为 fpsavg = n/t。如果我们设置 t = 1，则 fpsavg = n/1 = n。

在我们的代码中，我们使用 t = 1（秒），因为它避免了除法，而且，一秒给出了相当好的平均值——它不太长也不太短。

计算 FPS 的代码由 D3DApp::CalculateFrameStats 方法提供：

有些方法我们还没有讨论。

ClearRenderTargetView 方法将指定的渲染目标清除为给定的颜色，ClearDepthStencilView 方法清除指定的深度/模板缓冲区。

我们总是在每帧绘制之前清除后台缓冲区渲染目标和深度/模板缓冲区，以重新开始图像。这些方法声明如下：

ClearRenderTargetView

1.RenderTargetView：RTV到我们要清除的资源。

2.ColorRGBA：定义要清除渲染目标的颜色。

3.NumRects：pRects 数组中的元素数。这可以是 0。

4.pRects：一个 D3D12\_RECT 数组，用于标识渲染目标上要清除的矩形区域。这可以是一个 nullptr 以指示清除整个渲染目标。

ClearDepthStencilView

1. DepthStencilView：DSV到深度/模板缓冲区清除。

2. ClearFlags：指示要清除深度/模板缓冲区的哪一部分的标志。这可以是 D3D12\_CLEAR\_FLAG\_DEPTH、D3D12\_CLEAR\_FLAG\_STENCIL，也可以是按位或运算在一起。

3. 深度：定义要清除深度值的值。

4. 模板：定义要清除模板值的值。

5. NumRects：pRects 数组中的元素数。这可以是 0。

6. pRects：一个 D3D12\_RECT 数组，用于标识渲染目标上要清除的矩形区域。这可以是一个 nullptr 以指示清除整个渲染目标。

另一个新方法是 ID3D12GraphicsCommandList::OMSetRenderTargets 方法。

此方法将我们要使用的渲染目标和深度/模板缓冲区设置为管道。

现在，我们想使用当前的后台缓冲区作为渲染目标和我们的主深度/模板缓冲区。

在本书的后面，我们将研究使用多个渲染目标的技术。该方法具有以下原型：

1. NumRenderTargetDescriptors：指定我们要绑定的 RTV 的数量。同时使用多个渲染目标用于一些高级技术。目前，我们总是使用一台 RTV。

2. pRenderTargetDescriptors：指向 RTV 数组的指针，这些 RTV 指定了我们要绑定到管道的渲染目标。

3. RTsSingleHandleToDescriptorRange：如果前面数组中的所有RTV在描述符堆中都是连续的，则指定为true。否则，指定 false。

4. pDepthStencilDescriptor：指向 DSV 的指针，该 DSV 指定了我们要绑定到管道的深度/模板缓冲区。

最后，IDXGISwapChain::Present 方法交换前后缓冲区。

当我们展示交换链以交换前后缓冲区时，我们还必须更新当前后台缓冲区的索引，以便我们在后续帧上渲染到新的后台缓冲区：

D3D12\_INPUT\_element\_DESC数组中的每个元素描述并对应于顶点结构中的一个组件。

因此，如果顶点结构有两个组件，那么相应的D3D12\_INPUT\_ELEMENT\_DESC数组将有两个元素。

D3D12\_输入\_元素\_描述结构定义为：

1.SemanticName：与元素关联的字符串。这可以是任何有效的变量名。语义用于将顶点结构中的元素映射到顶点着色器输入签名中的元素；参见图6.1

2.SemanticIndex：附加到语义的索引。其动机如图6.1，其中，例如，顶点结构可能具有多组纹理坐标；因此，我们不需要引入新的语义名称，只需在末尾附加一个索引来区分这两个纹理坐标集。着色器代码中未指定索引的语义默认为索引零；例如，位置相当于图6中的位置0。1.

3.Format：DXGI\_格式枚举类型的成员，指定此顶点元素到Direct3D的格式（即数据类型）；以下是一些常用格式的示例：

4.InputSlot：指定此元素将来自的输入插槽索引。Direct3D支持16个输入槽（索引范围为0-15），通过这些槽可以输入顶点数据。

现在，我们将只使用输入槽0（即，所有顶点元素都来自同一个输入槽）；练习2要求您尝试使用多个输入槽。

5.AlignedByteOffset：从指定输入槽的C顶点结构开始到顶点组件开始的偏移量，以字节为单位。

例如，在下面的顶点结构中，元素Pos的起点与顶点结构的起点重合，因此其偏移量为0字节；

元素Normal有一个12字节的偏移量，因为我们必须跳过Pos的字节才能到达Normal的开头；

元素Tex0有一个24字节的偏移量，因为我们需要跳过Pos和Normal的字节才能到达Tex0的开头；

元素Tex1有一个32字节的偏移量，因为我们需要跳过Pos、Normal和Tex0的字节才能到达Tex1的开头。

6.InputSlotClass：现在指定D3D12\_每个顶点的输入数据；另一个选项用于高级实例化技术。

7.InstanceDataStepRate：现在指定0；其他值仅用于高级实例化技术。