**5 渲染流水线**

5.1 3D视觉即错觉

怎样将3D场景的空间深度感和立体感在2D平面显示器的屏幕中表现出来呢？

1，近大远小

2，光照和阴影

5.2 模型的表示

实体3D对象是用三角形网格来近似表示的，三角形是3D物体建模的基石。

除了三角形，点和线也有其用武之地。

5.3 计算机色彩基础

计算机显示器中的每个像素发出的都是红绿蓝三色混合光。

我们用3D向量(r,g,b)来表示颜色，其中0<=r,g,b<=1。

5.3.1 颜色运算

点积和叉积运算不适用于颜色向量。

颜色向量也有自己专属的颜色运算，即分量式乘法:对应分量相乘。

5.3.2 128位颜色

alpha分量常用于表示颜色的不透明度，它在混合技术中起到了至关重要的作用。

每种颜色刚好能用4D向量来表示，所以能用XMVECTOR类型来描述它们。

XMVECTOR XM\_CALLCONV XMColorModulate(FXMVECTOR c1, FXMVECTOR c2);

5.3.3 32位颜色

用32位数据表示一种颜色，每个分量仅能分配到1个字节。因此，每个占用8位字节的颜色分量就可以分别描述256种不同的颜色强度。

DirectXMath库用于存储32位颜色的结构：XMCOLOR。

XMVECTOR XM\_CALLCONV PackedVector::XMLoadColor(const XMCOLOR\* pSource);

void XM\_CALLCONV PackedVector::XMStoreColor(XMCOLOR\* pDestination, FXMVECTOR V);

存储在后台缓冲区中的像素颜色数据，往往都是以32位颜色值来表示。

5.4 渲染流水线概述

渲染流水线是以摄像机位观察视角而生成2D图像的一系列完整步骤。

输入装配器阶段，顶点着色器阶段，外壳着色器阶段，曲面细分阶段，域着色器阶段，几何着色器阶段，（流输出阶段），光栅化阶段，像素着色器阶段，输出合并阶段。

5.5 输入装配器阶段

输入装配器阶段会从显存中读取几何数据（顶点和索引），再将它们装配为几何图元（如三角形和线条等）。

简单来说，我们是通过索引来定义如何将顶点装配在一起，从而构成图元的方法。

5.5.1 顶点

顶点似乎仅是几何图元中的一种特殊点，但是，在Direct3D中，顶点的意义却不止于此。顶点还可以包含其他信息，使我们能够利用它来表现出更为复杂的渲染效果。

5.5.2 图元拓扑

在Direct3D中，我们要通过一种名为顶点缓冲区的特殊数据结构，将顶点与渲染流水线相绑定。顶点缓冲区利用连续的内存来存储一系列顶点。

但是，仅凭这一点并不能说明这些顶点究竟如何组成几何图元。例如，我们应将顶点缓冲区内的顶点两两一组解释成线段，还是3个一组解释成三角形。

对此，我们要通过指定图元拓扑来告知Direct3D如何用顶点数据来表示图元。

void ID3D12GraphicsCommandList::IASetPrimitiveTopology(D3D\_PRIMITIVE\_TOPOLOGY

PrimitiveTopology);

除了少数情况外，本书中大多使用三角形列表。

5.5.2.1 点列表

当使用点列表拓扑时，所有的顶点都将在绘制调用过程中被绘制为一个单独的点。

5.5.2.2 线条带

在使用线条带拓扑时，顶点将在绘制调用的过程中被连接为一系列的连续线段。在这种模式下，若有n+1个顶点就会生成n条线段。

5.5.2.3 线列表

每对顶点在绘制调用的过程中都会组成单独的线段，2n个顶点就会生成n条线段。

5.5.2.4 三角形带（图见书本）

所绘制的三角形将被连接成带状，在这种三角形连接的结构中，处于中间位置的顶点将被相邻的三角形所共同使用。n个顶点可生成n-2个三角形。

在三角形带中，次序为偶数的三角形与次序为奇数三角形的绕序是不同的（即装配图元的顶点顺序为逆时针或顺时针方向），这就是剔除问题的由来。

为了解决这个问题，GPU内部会对偶数三角形中前两个顶点的顺序进行调换，以此使它们与奇数三角形的绕序保持一致。

5.5.2.5 三角形列表

绘制调用的过程中会将每3个顶点装配成独立的三角形，所以每3n个顶点会生成n个三角形。

5.5.2.6 具有邻接数据的图元拓扑

对于存有邻接数据的三角形列表而言，每个三角形都有3个与之相邻的邻接三角形。

在几何着色器中，往往需要访问这些邻接三角形来实现特定的几何着色算法。

为了使几何着色器可以顺利地获得这些邻接三角形的信息，我们就需要借助顶点缓冲区与索引缓冲区将它们随主三角形一并提交至渲染流水线。

要将拓扑类型指定为D3D\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST\_ADJ，只有这样，渲染流水线才能得知如何以顶点缓冲区中的顶点来构建主三角形以及邻接三角形。

邻接图元的顶点只能用作几何着色器的输入数据，却并不会被绘制出来。

5.5.2.7 控制点面片列表

将顶点数据解释为N个控制点的面片列表，此图元常用于渲染流水线的曲面细分阶段。（此环节为可选阶段）

5.5.3 索引

四边形的顶点列表如下：

Vertex v[4] = {v0,v1,v2,v3} UINT indexList[6] = {0,1,2, 0,2,3}

八边形的索引列表：

UINT indexList[24] = {

0,1,2,

0,2,3,

0,3,4,

0,4,5,

0,5,6,

0,6,7,

0,7,8,

0,8,1,

}

处理完顶点列表中那些独立的顶点之后，显卡就能通过索引列表把顶点组合成一系列三角形。

5.6 顶点着色器阶段

待图元被装配完毕后，其顶点会被送入顶点着色器阶段。

我们可以把顶点着色器看作是一种输入与输出数据皆为单个顶点的函数。

for(UINT i = 0; i < numVertices; i++)

outputVertex[i] = VertexShader(inputVertex[i])

其中的顶点着色器函数就是我们要实现的那一部分，因为在这一阶段中对顶点的操作实际是由GPU来执行的，所以速度很快。

在顶点着色器中，不但可以访问输入的顶点数据，也能够访问纹理和其他存于显存中的数据。

5.6.1 局部空间和世界空间

局部坐标系通常是一种以目标中心为原点，并且坐标轴与该物体对齐的简易便用坐标系。

将局部坐标系内的坐标转换到全局场景坐标系中的过程叫做世界变换，所使用的的变换矩阵名为世界矩阵。

方法一：根据物体的位置与朝向，指定其局部空间坐标系的原点和诸坐标轴相对于全局场景坐标系的坐标，再运用坐标变换即可。

方法二：定义一系列的变换组合W，即W=SRT。

首先，缩放矩阵S将物体缩放到世界空间，其次旋转矩阵R用来定义局部空间相对于世界空间的朝向，最后平移矩阵T定义的是局部空间的原点相对于世界空间的位置。

矩阵W=SRT中的行向量则分别存储的是局部空间的x轴y轴z轴以及原点相对于全局空间的齐次坐标。

5.6.2 观察空间

我们为该摄像机赋予一个局部坐标系，也被称作观察空间，视觉空间或摄像机空间。

在此坐标系中，该虚拟摄像机位于原点并沿z轴的正方向观察，x轴指向摄像机的右侧，y轴则指向摄像机的上方。

由世界空间至观察空间的坐标变换称为取景变换，此变换所用的矩阵称为观察矩阵。

世界坐标系和观察坐标系通常只有位置和朝向这两点差异，所以由观察空间到世界空间的变换可以直接表示为W=RT。

W=RT矩阵内容的推导见书本。

只要给定摄像机的位置，观察目标点以及世界空间中“向上”方向的向量，我们就能构建出对应的摄像机局部坐标系，并推导出相应的观察矩阵。

XMMATRIX XM\_CALLCONV XMMatrixLookAtLH(

FXMVECTOR EyePosition,//虚拟摄像机位置

FXMVECTOR FocusPosition,//观察目标点

FXMVECTOR UpDirection//世界空间中向上方向的向量

)

一般来说，世界空间中的y轴方向与虚拟摄像机向上向量的方向相同。

5.6.3 投影和齐次裁剪空间

除了摄像机在世界空间中的位置和朝向，还有另一个关键组成要素：即摄像机可观察到的空间体积。此范围可用一个四棱锥截取的平截头体来表示。

下一个任务是将平截头体内的3D几何体投影到一个2D投影窗口之中。

我们将由顶点到观察点的连线称为顶点的投影线。继而可以定义出：将3D顶点v变换至其投影线与2D投影平面交点v'的透视投影变换。

5.6.3.1 定义平截头体

在观察空间中，我们可以通过近平面n，远平面f，垂直视场角a以及纵横比r这四个参数来定义一个：以原点作为投影中心，并沿z轴方向进行观察的平截头体。

投影窗口实质上即为观察空间中场景的2D图像。由于该图像终将被映射到后台缓冲区中，因此，我们希望令投影窗口与后台缓冲区两者的纵横比保持一致。如若投影窗口与后台缓冲区的纵横比不一致，那么映射过程中，就需要对投影窗口再将投影窗口进行不等比缩放，继而导致图像出现拉伸变形。

可以通过垂直视场角a和纵横比来确定水平视场角β。

5.6.3.2 投影顶点

我们可以求出给定点（x,y,z）在投影平面z=d中的投影。

5.6.3.3 规则化设备坐标

若点(x,y,z)位于平截头体之中，当且仅当：

-1 <= x' / r <= 1

-1 < = y' <= 1

n <= z <= f

以NDC坐标来表示的x轴和y轴上的投影坐标：

x' = x / rztan(a/2)

y' = y / ztan(a/2)

在NDC坐标中，投影窗口的高和宽都为2，所以它的大小是固定的。

5.6.3.4 用矩阵来表示投影公式

为了保证变换的一致性，我们将用矩阵来表示投影变换。然而，由于上述变换的非线性特征，所以并不存在与之对应的矩阵表示。我们的方法是将其一分为二，即将其分为线性与非线性两个处理部分。非线性部分要进行除以z的计算过程。

我们还将对z坐标进行归一化处理，这意味着在执行非线性部分除以z的计算时，我们却无最初的z坐标可用。也就是说，我们一定要再此变换之前保存早先传入的初始z坐标。

为了做到这一点，我们要利用齐次坐标将输入的z坐标复制到输出的w坐标。

此投影矩阵公式见书本。

5.6.3.5 归一化深度值

为了实现深度缓冲算法，我们仍需保留3D深度信息。深度坐标也要被映射到归一化区间[0,1]以内。

因此，我们必须构建一个保序函数g(z)，用来把z坐标从区间[n,f]映射到区间[0,1]。

推导g(z)的公式见书本。

透视投影矩阵公式见书本。

在顶点乘以透视矩阵之后但还未进行透视除法之前，几何体会处于所谓的齐次裁剪空间或投影空间之中。待完成透视除法之后，便是用规格化设备坐标（NDC）来表示几何体了。

5.6.3.6 XMMatrixPerspectiveFovLH函数

可以使用XMMatrixPerspectiveFovLH函数来构建透视投影矩阵

XMMATRIX XM\_CALLCONV XMMatrixPerspectiveFovLH(

float FovAngleY,//弧度制表示的垂直视场角

float Aspect,//纵横比 宽度/高度

float NearZ,//近平面距离

float FarZ

)

5.7 曲面细分阶段

曲面细分阶段是利用镶嵌化处理技术对网格中的三角形进行细分，以此来增加物体表面上的三角形数量。

再将这些新增的三角形偏移到适当的位置，使网格表现出更加细腻的细节。

使用曲面细分的优点：

1，实现一种细节层次(LOD)机制，使离虚拟摄像机较近的三角形经镶嵌化处理得到更加丰富的细节，对距摄像机较远的三角形不进行任何更改；

2，我们在内存中仅维护简单的低模网格，再根据需求为它动态地增添额外的三角形，以此节省内存资源；

3，我们可以在处理动画和物理模拟时采用简单的低模网格，而仅在渲染的过程中使用经镶嵌化处理的高模网格。

曲面细分是一个可选的渲染阶段，可在用户需要之时才开启。

5.8 几何着色器阶段

几何着色器是一个可选渲染阶段。

几何着色器接手的输入应当是完整的图元。几何着色器的主要优点是可以创建或销毁几何体。比如说，我们可以利用几何着色器将输入的图元拓展为一个或多个其他图元，抑或根据某些条件而选择不输出某些图元。

顶点着色器与之相比，则不能创建顶点。几何着色器的常见拿手好戏是将一个点或一条线扩展为一个四边形。

几何着色器能够为后续的绘制操作，而将顶点数据流输出至显存中的某个缓冲区之内。

5.9 裁剪

完全位于视椎体之外的几何体需要被丢弃，而处于平截头体交界以外的几何体部分也一定要接受被裁剪(clip)操作。因此，只有平截头体之内的物体对象才会最终保留下来。

裁剪操作是由硬件来负责的，所以我们也就不再赘述其具体的实现细节。

5.10 光栅化阶段

光栅化阶段的主要任务是为投影至屏幕上的3D三角形计算出对应的像素颜色。

5.10.1 视口变换

当裁剪操作完成之后，硬件会通过透视除法将物体从齐次裁剪空间变换为规格化设备坐标(NDC)。

一旦物体的顶点位于NDC空间内，构成2D图像的2D顶点x,y坐标就会被变换到后台缓冲区中称为视口的矩形里。

待此变换完成后，这些x,y坐标都将以像素为单位表示。

5.10.2 背面剔除

每个三角形都有两个面，我们采用以下约定来对它们进行区分。

如果组成三角形的顶点顺序为v0,v1,v2，那么，我们通过下述方法来计算此三角形的法线n：

e0 = v1 - v0； e1 = v2 - v0; n = e0 x e1 / |e0 x e1|

法向量由正面射出，另一面则为背面。

如果观察者看到的是三角形的正面，我们就称此三角形是正面朝向的，如果观察者看到的是三角形的背面，则称为背面朝向的。

根据观察者的视角看去，顶点绕序为顺时针方向的三角形为正面朝向。

令构建每个物体所用都是法线指向外侧的三角形，那么摄像机将看不到实体对象中背面朝向的三角形，这是因为正面朝向的三角形会把背面朝向的三角形遮挡起来。

背面朝向的三角形都被正面朝向的三角形所遮挡，所以绘制它们是没有意义的。

背面剔除就是用于将背面朝向的三角形从渲染流水线中除去的处理流程。这种操作将待处理的三角形重量削减一半。

我们可以在观察者察觉不到情况下，从后面的处理流程中丢弃掉背面朝向的三角形。

5.10.3 顶点属性插值

我们要通过指定顶点来定义三角形。除了位置信息以外，我们还能给顶点附加颜色，法向量和纹理坐标等其他属性。

经过视口变换之后，我们需要为求取三角形内诸像素所附的属性而进行插值运算。

而且，除了上述顶点属性，还需对顶点的深度值进行内插，继而得到每个像素参与实现深度缓冲算法的深度值。

为了得到屏幕空间中各个顶点的插值属性，往往要通过一种名为透视校正插值的方法。

从本质上来说，插值法即利用三角形顶点的属性值计算出其内部像素的属性值。

我们无须考虑透视校正插值法处理像素属性的具体数学细节，因为硬件会自动完成相应的处理。

5.11 像素着色器阶段

我们编写的像素着色器是一种由GPU来执行的程序。它会针对每一个像素片段（片元）进行处理（即每处理一个像素就要执行一次像素着色器），并根据顶点的插值属性作为输入来计算出对应的像素颜色。

5.12 输出合并阶段

通过像素着色器生成的像素片段会被移送至渲染流水线的输出合并阶段。

在此阶段中，一些像素片段可能会被丢弃（例如，未通过深度缓冲区测试或模板缓冲区测试的像素片段）。而后，剩下的像素片段将会被写入后台缓冲区中。

此技术可令当前处理的像素与后台缓冲区中的对应像素相融合，而不仅是对后者进行完全的覆写。

一些透明这样的效果，也是由混合技术来实现的。