《Real Time Rendering》第二章 图形渲染管线

http://www.cnblogs.com/alonecat06/category/417541.html

《Real Time Rendering》第二章 图形渲染管线

这一章将介绍的是被认为是实时图形学核心部件的图形渲染管线，亦可简称为管线。管线的主要功能是生成或渲染二维图像、三维物体、光源、着色方程式、纹理等。渲染管线是实时渲染的底层实现。管线的作用如图2.1所示。图片中的对象所处位置及外形由其几何数据所处环境的特性以及摄像机的位置共同决定对象的外表是受到材质属性、光源、纹理及着色模形所影响。

渲染管线进行渲染的不同阶段会在下面被解析和讨论。这将聚焦于各阶段所负责的功能而非其实现。实现的细节要么留在最后一章，要么是程序员不能控制的元件。举个例子，对于一个正使用直线的人来说，他最在意的是这条线的特性，例如其顶点数据格式、颜色、线型类型以及所谓深度提示是否可用，而非该线是使用Bresenham直线绘制算法或symmetric double-step算法绘制。这些渲染阶段通常是实现在不可编程的硬件上，这使得对其实现进行优化和改进是不可能的。一些书例如Rogers的深入地介绍了基本的绘制和填充算法的细节，而我们对底层硬件、图像生成的算法和代码控制是有限的。

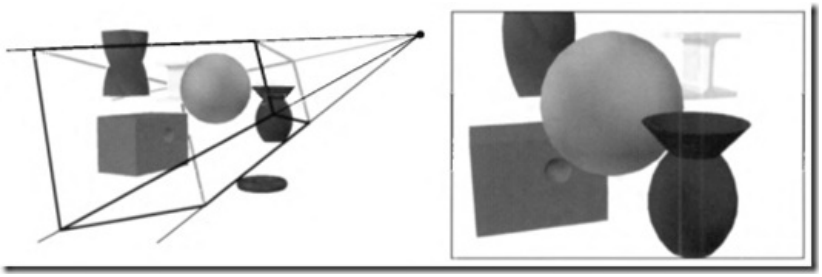

图2.1 在左图，一虚拟摄像机放置在棱椎的顶部（即四条线交汇处）。只有可视体内部的图元会被渲染。因为图像是以透视画法渲染的（像这里的情况），可视体成一平截头体，即一切去顶端的四棱椎。右图展示了摄像机的所见。注意到红色的圆环体因为其位置在可视平截头体外，故未被渲染同样的是左图的蓝色螺旋柱体亦被平截头体顶平面裁剪掉。

2.1 架构

在现实世界，管线这个概念以各种不同方式出现，从工厂的组装流水线到运送滑雪者的上山吊车。它同样出现在图形渲染领域中一条管线由多个阶段、步骤组成。例如油汽管线，在第一阶段的石油不能在第二阶段石油进入第三阶段前进入第二阶段，这预示着管线的速度取决于最慢的阶段，无论其它阶段有多快。

理论上，一个非管线系统若分成几条管线阶段，可给予n的加速因子。在运行效率上的提升是应用管线的主要原因。例如一雪橇吊车只有一张椅子是低效率的；增加更多的椅子可成比例地加速运送滑雪者上山。管线阶段是在汽车组装流水线上，方向盘的装配阶段需要3分钟，而其它阶段需要两分钟，则能达到的最佳汽车组装速率为3分钟一辆；其它阶段必须闲置1分钟去等待方向盘组装阶段的完成。针对这条独特的管线，方向盘组装阶段成为其瓶颈，这是因为它决定了整条管线的生产效率。

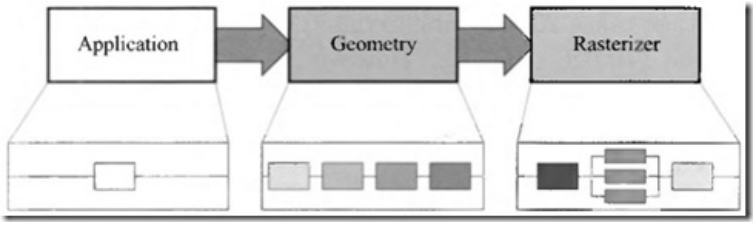

图2.2 渲染管线的基本构成由3个阶段组成：应用程序，几何及光栅。每一个阶段本身都有可能是一条管线，就如图所示的几何阶段；或者一个部分并行的阶段，就如图所示的光珊阶段。在图中，应用程序阶段是单一的过程，但亦有可能是管线化或并行化。

在实时计算机图形中，亦存在同样的管线设计。如图2.2，实时渲染管线可粗略分为三个概念性的阶段——应用程序，几何及光珊。这个结构是渲染管线的核心机制，应用在实时计算机图形学应用程序中。因此接下来的章节将对其作基础的精要的论述。每一个阶段通常本身也是一条管线，这意味着它由一些子阶段组成。我们对概念性的阶段（应用程序，几何和光栅），功能性阶段及管线阶段进行区分。功能性的阶段规定了运行某个任务，但并未指定任务在管线中如何执行。而管线阶段则是与其它管线阶段并行执行。为了高性能的需要，管线阶段可能并行执行。例如几何阶段可分为5个功能阶段，但它在图形系统的实现是取决其在管线阶段的划分。一个给定的实现可能将两个功能阶段联合在一个管线阶段中实现，同时一个时间消耗比较大的功能阶段，亦可分割为一系列管线阶段，甚至对其并行化。

最慢的管线阶段决定了渲染速度、图像的更新速率。这个对其速度可以以帧率(fps)表达，这是每秒渲染图像的数目。它亦可以以赫兹表示，即一秒的多少分之一的记号，是更新的频率。用于应用程序生成图片的时间通常是各种各样的，取决于每帧图片计算的复杂度。帧率所表达的要么是某一帧的速率，要么是一段时间内的平均速率。赫兹是用硬件方面，例如显示，这是一个固定的速率。因为这涉及管线，将其处理所有需渲染数据所用时间简单相加是不足够的。当然，这是管线构造所导致的结果，它让各阶段可并行运行。如果我们能定位出瓶颈，即管线中最慢的阶段所在，并测量出它处理数据所需时间，那样的话，我们可能测出渲染速度。假设，瓶颈阶段需要20微秒去运行；渲染速度为1/0.020即50Hz。但这仅当输出设备能以这个速度更新时才成立；否则的话，真正的输出帧率将更慢。在管线的背景下，吞吐量比渲染速度更能说明一切。

例子：渲染速度。假设我们设备的最大更新频率为60Hz，并且渲染管线的瓶颈已找到，该阶段需时62.5微秒运行。渲染速度的计算如下。首先，忽略输出设备，得到最大渲染速度为1/0.0625=16fps。接着，调整输出设备的频率：60Hz意味着渲染速度可以是60Hx,60/2=30Hz,60/3=20Hz,60/4=15Hz,60/5=12Hz等5种速度。这意味着我们可以预期渲染速度为15Hz，这是因为输出设备所能达到的最大持续输出帧率是少于16fps

正如名字所暗示的，应用程序阶段是由应用程序驱动并且其由运行在通用CPU的软件代码所实现。这些中央处理器通常包含多个内核并且可并行运行多个线程。这使得中央处理器能有效处理应用程序阶段各种大相径庭的任务。这些传统上运行于 中央处理器的任务包括碰撞检测，全局加速算法，动画，物理模拟等各种，这取决于应用程序的类型。接下来的是几何阶段，处理几何变换，投影等。这个阶段计算绘

制什么，如何绘制及在那里绘制。几何阶段通常运行在图形处理器（GPU）中。这类处理器色含多个可编程内核和一些固定操作的硬件。最后在光栅阶段应用前阶段产生的数据绘制（渲染）图片，并且进行逐象素的计算。整个光栅阶段都在GPU上运行。这些阶段及其内部管线将在接下来的三章被讨论。更多的关于图形处理器处理这些阶段的细节见第三章。

2.2 应用程序阶段

因为应用程序阶段运行于CPU上，开发者可以对其全面掌控。因些，开发者可完全决定其实现并在之后对其修改以提高运行效率。在这里的改变亦能影响后面阶段的运行效率。但如一些应用程序阶的算法和设定可能减少将被渲染的多边形。

在应用程序阶段的最后步骤，将被渲染的几何体会输入到几何阶段。这些几何体都是绘制图元，例如点、线和三角形等最后将输出到屏幕（或者被输出设备所用）。这是应用程序阶段中最重要的任务。

这阶段的实现是以软件为基础，这导致到不像几何和光珊阶段那样，被析分为多个子阶段。但是为了提升运行效率，该阶段经常并行运行在多个处理器核心上。在中央处理器设计方面说，这叫超标量体系结构，因为它能够在同一阶段同一时间内运行数个运算步骤。15.5节展示各种应用多核处理器的方法。

一通常在这阶段实现的运算步骤是碰撞检查。当一关于两物体的碰撞检测出来后，反应将会被生成并发送到碰撞的对象和力反馈设备上，应用程序阶段亦是处理包括键 盘、鼠标、头盔等设备输入的地方。跟据不同的输入，作出不同的反应。该阶段的其它步骤包括纹理动画或者一些不运行在其它阶段的计算。一些加速算法，例如层次化视图平截体裁剪（见第十四章）亦是在此处实现。

2.3 几何阶段

几何阶段负责了大多数逐多边形和逐顶点的操作。该阶段可进一步分割成下面的功能阶段：模型和视图的变换，顶点着色，投影，裁剪及屏幕映射（图2.3）应注意的是，功能阶段可能与管线阶段相等，亦可不同。在某些场合下，一系列连续功能阶段组合成一个单独的管线阶段（它与其它管线阶段并行运行），在另外一些情况下，一个功能阶段也可能被分割成多个更小的管线阶段。

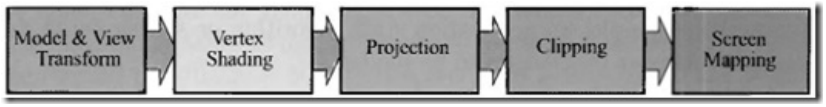

图2.3 几何阶段细分一条管线的数个功能阶段

例如，考虑一极端情况，整个渲染管线可能在单一一个处理器，以软件实现，这样的话你可以说整个管线由一个管线阶段组成。当然，这就是在单独的图形加速芯片和显卡出现前去生成图形的方法。而另一个极端是，所有的功能阶段都细分为更细小的管线阶段，并且每个管线阶可运行在指定的处理器核心元素。

2.3.1 模型及视图变换

在成为屏幕上的图象前，模型会被变换到多个空间或坐标系。在最初，模型处于它所拥有的模型空间，简单的说是它未进行变换。每个模型都可与模型变换相关联，以对其定位和定向。一个模型可与多个模型变换相关联。这使得在没有复制基本的几何数据的情况下，可以有多个不同位置、朝向和大小的模型拷贝（或者说是实例）。

模型上的顶点和法何量会被模型变换所变换。对象的坐标系称为模型坐标系，当应用了模型变换后，模型处于世界坐标系或世界空间。世界空间是唯一的，当模型完成其相应的模型变换后，所有的模型处于同一空间。

像前面所说的，只有摄像机（或观察者）能看见的模型才会被渲染。摄像机处于世界空间，有方向及位置。为了进行投影和裁剪，摄像机和所有的模型会进行视图变换。进行视图变换的目的是将摄像机放置于坐标原点，并使其方向朝 Z轴负向，且Y轴指向上，X轴指向右。视图变换后，实际位置和方向取决于应用程序接口（API）的实现。

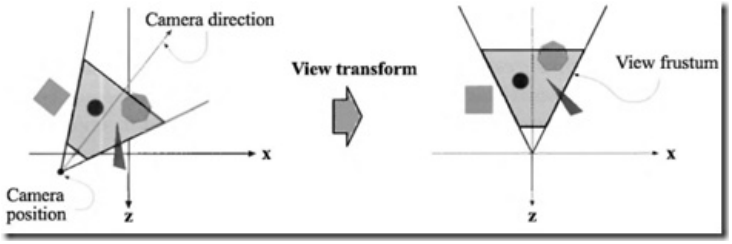

图2.4 在左图中，摄像机的位置及朝向如用户所设想。视图变换重定位摄像机到原点，朝向沿Z轴负方向，如右图所示。这个做法使得裁剪和投影更简单和快速。浅灰色的区域是可视体。因为可视体是一平截头体，可假定其为透视的，相同的方法可用于其它任何一种投影。

因此这个空间所描述的被叫作摄像机空间，或者更普遍的说，叫视角空间。图2.4展示了一个视图变换影响摄像机和模型的例子。所有的模型变换和视图变换都由4×4矩形阵实现，这将是第四章的主题。

2.3.2 顶点着色

为了实现逼真的场景，仅渲染对象的形状和位置是不足够的，它们的外观亦需要模拟。这些描述包括第每对象材质，以及光源照射对象产生的特效。模拟材质和光源的方法有很多种，包括从最简单的颜色到精细的物理特性描述。

决定光和材质特效的操作称为着色。它包括了计算不同点的着色方程。典型的某些这类的计算运行在几何阶段的模型顶点数组上，另外一些则运行在逐象素的光珊化阶段。各类的材质数据，例如点所在位置，法向量，颜色或其它着色方程需用到的数值信息，可储存在每个顶点中。顶点着色的结果（这可能是颜色，向量，纹理坐标或其它种类的着色数据）会被送进光珊化阶段去插值。

着色阶段通常被认为是发生在世界空间。在实践中，有时则将相应的实体（例如摄像机和光源）变换到其它空间（例如模型或视觉空间）并在那运行计算更为方便。因为如果所有包含在着色计算中的对象均被变换到同一空间，则光源、摄像机和模型的相对关系是保留的。

关于着色更深入的讨论将贯穿这本书，将在第三至五章中详述。

2.3.3 投影

着色完成后，渲染系统开始进行投影，即将可视体转换为一位于（-1，-1，-1）到（1，1，1）单位立方体。这个立方体叫做规则观察体。有两种常用的投影方法，即正投影（亦称为平行投影）和透视投影。如图2.5

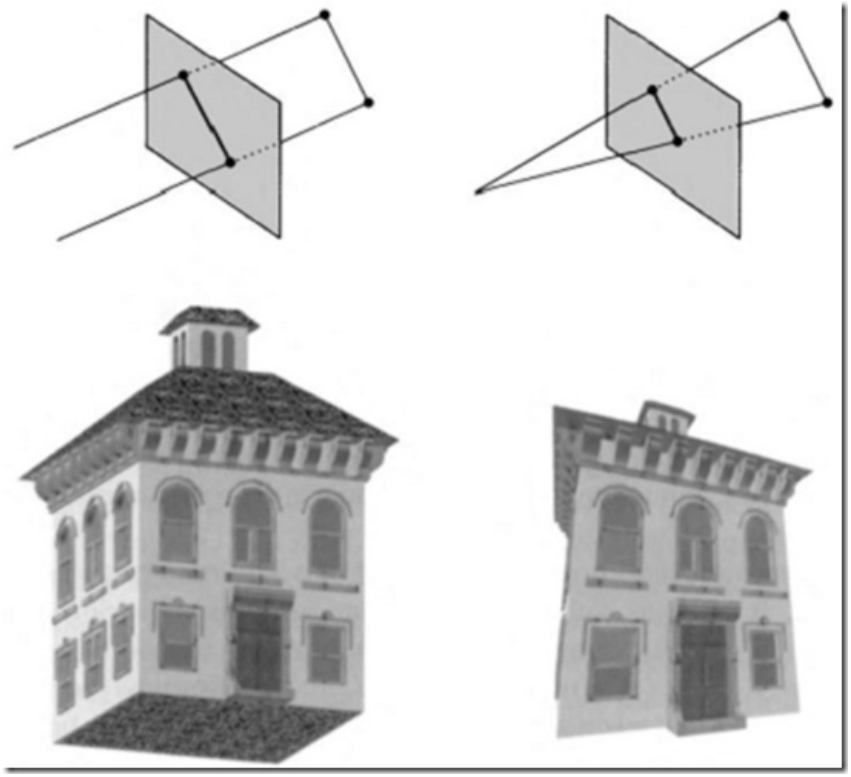

图2.5 左边是正投影或者叫平行投影；右图是透视投影。

正投影的可视体为则的正方体，投影将可视体转为单位立方体。正投影主要的特性是变换后平行线依然保持平衡。这个变换是位置变换和比例变换的组合。

而透视投影有些复杂。在这种投影中，越远离摄像机的物体，它在投影后看起来越小。更进一步来说，平行线将在地平线会聚。透视投影的变换是模拟人类感知物体的方式。在几何方面说，可视体是平截头体，即一截去头部的以矩形为基底的四棱锥。这个平截头体也被变换到单位立方体中。正投影和透视投影的变换都可由一4×4的矩阵构建，而在进行任一种变换后，模型被认为处于规格化设备坐标系。

虽然这些矩阵变换是从一个可视体变换到另一个，但它们仍被称为投影，因为在完成显示后，Z坐标不会再保存在图片中。通过这样的方法模型从三维空间投影到二维上。

2.3.4 裁剪

只有完全或部分处于可视体中的图元才会被传递到光栅阶段，该阶段将在屏幕绘制这些图元。完全处于可视体的图元会直接传递到下一个阶段。而在可视体外的图元则因其无需被渲染而不会传递到下一阶段。部分处于可视体内的图元则需要裁剪。例如一个顶点在可视体外而另一个在可视体内的线段将被可视体裁剪，所以在外面的顶点将被一位于线段与可视体相交的顶点所代替。使用投影矩阵意味着图元与单位立方体相裁剪。在裁剪之前进行视图变换和投影变换的好处是使得裁剪问题比较一致；图元总可被单位立方体裁剪。裁剪的过程如图2.6所描述。作为六个裁剪平面的补充，用户可定义额外的裁剪平面去剔除对象。在646页的图14.1通过一幅图片展示了这种叫分割的呈现方式。不像前面的那些运行在可编程处理单元的几何阶段，裁剪阶段（以及接下来的屏幕映射阶段）通常是运行在固定操作的硬件上。

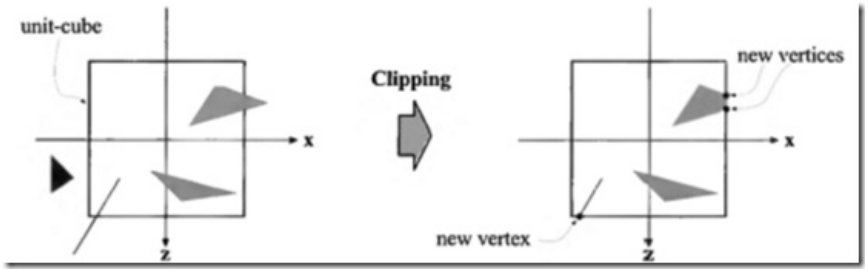

图2.6 进行投影变换后，只有在单位立方体内的图元（相应地处于可视体内的图元）才需要后续的处理。因些，在单位立方体外的图元将被丢弃而完全在内的图元将被保留。与单位立方体相交的图元会被单位立方体裁剪，新的顶点会生成而旧的会被丢弃。

2.3.5 屏幕映射

只有（被裁剪过的）在可视体里面的的图元才会被传递到屏幕映射阶段，在进入这阶段时坐标仍然为三维的。图元的X、Y坐标被变换到屏幕从标系中。屏幕坐标系与Z坐标一并为窗口坐标系。假设场景应渲染在一个最小角（X1Y1）最大角在（X2Y2）且X1＜X2和Y1＜Y2的窗体中。那么屏幕映射是首先进行平移随后进行缩放操作。Z坐标不受映射的影响。新的X，Y坐标被称为屏幕坐标。这些新的X、Y坐标与Z坐标（-1≤Z≤1）一道进入光栅阶段。图2.7描述了屏幕映射的过程。

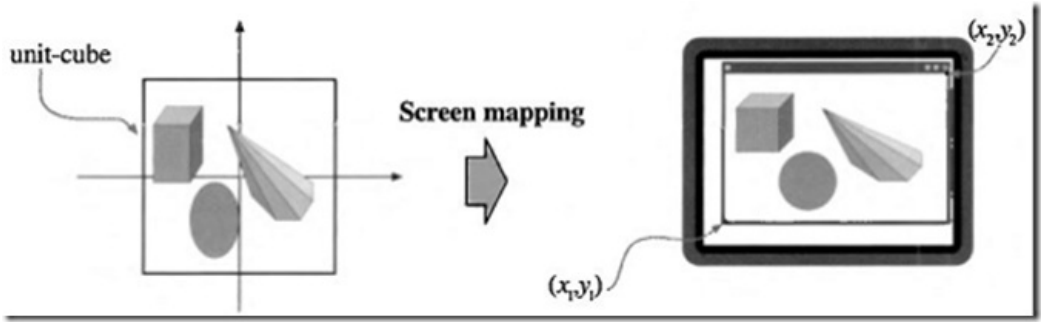

图2.7 图元在投影变换后处于单位立方体内，而屏幕映射负责找屏幕上对应的坐标

一个困惑是整型和浮点型的点值如何与像素坐标（或纹理坐标）要关联。DirecttX9和其之前版本的坐标系中，0.0是像素的中央，这意味着一片范围为[0，9]的像素其所覆盖的跨距为[-0.5，9.5]。Heckbert[520]给出了一个理论上更一致的策略。给出一水平的像素队列并放在笛卡尔坐标中。最左边的像素是浮点坐标的0.0。OpenGL一直以来都使用这种策略，而DirectX10及其后继版本也使用这种方式。在中央的像素为0.5。所以一系列范围为[0，9]的像素覆盖的范围为[0.0，10.0]。转换公式如下：

d =floor(c)　(2.1)

c = d+0.5　(2.2)

d是散化（整数）的像素索引，而c是连续的（浮点）像素值。

虽然所有的API的像素位置值均是由左到右递增，但在关于零点像素位于最顶或最底边缘的问题目上OpenGL与DirectX是不相一致的。OpenGL更倾向于笛卡尔坐标系，将左下角设定为数值最低的点，而DirectX有时定义左上角为这个点，这依赖于周边环境。每个做法都有其背后的逻辑，关于它们的不同并没有正确的回答。例如在OpenGL中，（0，0）位于图片的左下角而在DirectX中位于图片的左上角。DirectX采用如此做法的原因是屏幕上的很多的现象均从上到下的：微软的窗体使用这种坐标系，我们阅读的顺序，多种图片格式储存缓冲数据的方式。这个不同点的存在对跨图形API是很重要。

2.4 光栅阶段

得到已变换及投影后的顶点及与之相关联的着色数据（所有均来自几何阶段），光珊阶段的目标是计算并设置像素的颜色。这个过程叫光珊化或扫描变换，即从二维顶点所处的屏幕空间（所有顶点都包含Z值即深度值，及各种与相关的着色信息）到屏幕上的像素的转换。

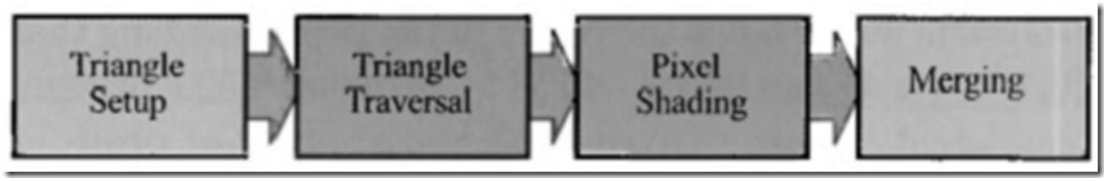

图2.8 光珊阶段细分为功能阶段的管线。

与几何阶段相似，该阶段细分为几个功能阶段：三角形设定，三角形遍历，色素着色和融合。

2.4.1 三角形建立

在这个阶段微分及其它关于三角形表面的数据会被计算。这些数据用于扫描转换及几何阶段产生的各种着色数据的插值。这个过程运行在专门为其设计的硬件上。

2.4.2 三角形遍历

在这个阶段像素将被检查三解形是否覆盖其中心，而对于与三角形部分重合的像素，其重合部分将生成片段（fragment）。找到哪些采样点或像素在三角形中的过程通常叫三角形遍历或扫描转换。每个三角形片段的属性均由三个三角形顶点的数据插值而生成（详见第五章）。这些属性包括片段的深度，以及来自几何阶段的着色数据。Akeley和Jermoluk [7]和Rigers[1077]提供了三角形遍历方面的信息。

2.4.3 像素着色

所有逐像素的着色计算都在这阶段进行，使用插值得来的着色数据作为输入。最终的结果为一或种将被传送到下一阶段的颜色。不像三角形建立和遍历那样通常运行在专用的，电路半导体的阶段，像素着色阶段在可编程GPU内核上运行，大量的技术可以在这里使用，其中最重要的技术之一为纹理贴图，关于纹理贴图的详细内容详见第六章。简单地说，纹理贴图即将一图像贴在物体上。图2.9描述了这一过程。图象可为一维、二维、三维的，而二维图像更为普遍。

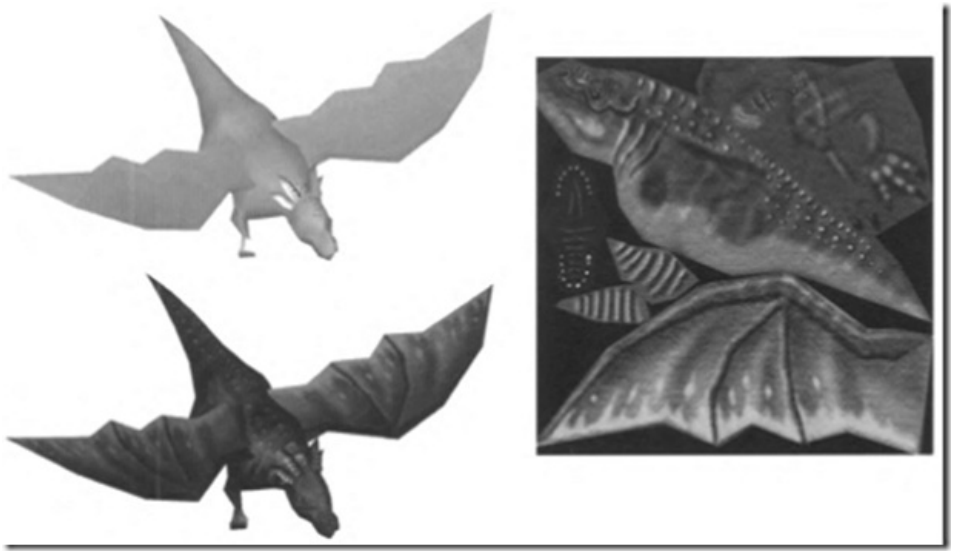

图2.9 左上角为一没有纹理贴图的飞龙模型。左下角为一贴上图像纹理的飞龙。

2.4.4 融合

每个像素的信息都储存在颜色缓冲器中，这是一个颜色的矩阵列（每种颜色包含红、绿和蓝成分）。融合阶段的任务是合成现储存于缓冲器的由着色阶段产生片段的颜色。不像其它着色阶段，典型的运行该阶段的GPU子单元并非完全可编程的。但它是高度可配置支持多种特效的。

这个阶段也负责可见性的计算。这意味着当整个屏幕被渲染时，颜色缓冲器应包含从摄像机角度上可见的场景的图元的颜色。对于图形硬件，这些工作由Z缓存（亦叫深度缓存）算法去完成。Z缓存与颜色缓存的大小，格式相同，而对于每个像素，它储存从摄像机到与摄像机最靠近的图元的Z值。这意味着当图元被渲染到某个像素时，其图元的Z值会被计算并与Z缓存中相同像素的Z值相比较，如果新的Z值小于Z缓存中的值，那么将被渲染到这像素的图元离摄像机比之前的图元更近。那么那个像素的Z值和颜色会被即将绘制的图元所更新。如果计算得出的Z值大于Z缓存中的值时，颜色缓存和Z缓存将被保留。Z缓存算法十分简单，其收敛性为O(n)（n指将被渲染的图元的数目），而且它对于所有可计算出每个（相关的）像素Z值的图元都是有效的。并且注意到这个算法允许按任意顺序绘制图元，这也是其流行的原因。但半透明的图元并不能按任意顺序绘制。它们必须在所有不透明图元绘制后再进行绘制，并且以由后到前的顺序（见5.7节）。这是Z

缓存算法的主要不足之处。
之前我们说过颜色缓存储存每个像素点颜色而深度缓存储存相应的Z值。然而还有另外一些管道和缓存可以过滤和捕获片段的信息。alpha管道与颜色缓存相关并且储存每个像素对应的透明值（见5.7节）。可选的alpha测试可在深度测试执行前在传入片段上运行。片段的alpha值与参考值作某些特定的测试（如等于，大于等），如果片断未能通过测试，它将不再进行进一步的处理。这种测试经常用于不影响深度缓存的全透明片段（见6.6节）。
模版缓存则是用于记录所呈现图元位置的离屏缓存。每个像素通常与占用8个位。图元可使用各种方法渲染到模版缓存中，而缓存中的内容可以控制颜色缓存和Z缓存的渲染。举个例子，假设一填充的圆形被绘制在模版缓存中。这可以与操作符联合，这个操作符允许将后续的图元仅在圆形所出现之处进行绘制。模版是一个强大的特效生成工具。所有这些功能在管线的尽头叫做光栅操作（ROP）或混合操作。
帧缓存大体上组成了系统中的所有缓存，但有时用于指长颜色缓存和Z缓存组成的集合。在1990年，Haeberli和Akeley [474]提出了对帧缓存的另一种补充，叫累积缓存。在这种缓存中，图像可通过一系列操作符进行累积。例如为了生成动态模糊一系列展示物体运动的图像可被积累和平均。其它可生成的特效包括景深，反锯齿，软阴影等。
当图元通过光栅阶段处理，那些从摄像机视觉为可见的对象将被显示到屏幕上。屏幕上显示颜色缓存的内容。为了避免人们看到图元在进行光栅化并传送到屏幕，双缓存技术被使用。这意味着场景的渲染离屏进行，在后台缓存中。一旦在后台缓存中完成场景的渲染，后台缓中的内容将与之前展示在屏幕的前台缓存内容交换。交换发生在垂直回扫时，这时候执行这个操作是安全的。
对更多关于不同缓存及缓冲方法的信息，参见5.6.2节及18.1节
2.5 穿越管线
点、线和三角形是渲染图元，它们共同组成了模形或对象。想像一下一个交互的计算机辅助设计系统，用户正在检查一手机的设计。在此我们将跟随这个模形穿越整个图形管线，管线是由三个阶段组成：应用程序、几何和光栅。场景是以透视画法绘制在屏幕上。在这个简单的例子中，手机模型引入了直线段（用于展示边缘部分）和三角形（用于展示表面）。一部分的三角形以二维图像纹理贴图去表现键盘和屏幕。在这个例子中，着色计算全部在几何阶段，除了发生在光珊阶段的纹理贴图。
应用程序阶段
CAD应用程序允许用户选择和移动模型的一部分。例如，用户可能选择手机的顶部并移动鼠标翻开手机。应用阶段必须将鼠标的移动转换为相应的旋转矩阵，然后务必确认渲染时那阵矩阵正确地应用于手机翻盖上。另一个例子：播放一个摄像机沿预定路径运动从不角度去展示手机的动画。摄像机的参数，例如位置和视觉方向，必须由应用程序阶段更新。对于被渲染的每一帧，应用程序阶段将摄像机位置，光，和模型的图元填入管线的下一个主要阶段——几何阶段。
几何阶段
在应用程序阶段，视图矩阵已被计算好，与此同时的还有关于每个对像位置和朝向的模型矩阵。对于每个传递到几何阶段的对像，这两个矩阵通相乘为一个矩阵。几何阶段对象的顶点、法何量被这个连接的矩阵转换到视觉空间。然后使用材质和光源等属性计算顶点的着色。投影被执行，将对像转换到代表视觉可见的单元立方体空间。所有在单元立方体之外的图元将被丢弃。所有与单元立方体相交的图元将被裁剪以获得一组完全处于单元立方体的图元。然后将顶点映射到在屏幕的窗口上。在所有的逐多边形操作执行完后，结果数据被传递到光栅阶段——管线的最后一个主要阶段。
光栅阶段
在这个阶段，所有图元光珊化，转换为窗体上的像素。每个对像上每条可见的线和三角面都通过光珊器进入屏幕空间，准备转换。那些与纹理相关联的三角面会应用上纹理（图片）后再行绘制。可见性问题通过Z缓存算法解决，随同的还有可选的alpha测试和模版测试。所有对像依次处理，而最后的图像显示在屏幕上。
结论
这样管线是API和图形硬件十年来以实时渲染应用程序为目标进行演化的结果。需要注意的是这个进化不仅在渲染管线上，线下渲染管线也以另一种路径发展。电影产品的渲染通常使用微多边形管线。学术研究和预测渲染（predictive rendering）应用，例如architectural pre-visualization使用到光线跟踪渲染器（见9.8.2节）
在多年以前，应用开发者使用这里所描述的过程的唯一方法是使用图形API所定义的固定功能管线。以固定功能管线去命名的原因是图形硬件以一些不能灵活编程的单元实现。而管线的各部分可设置不同的状态。Z缓存测试可以开启或关闭，但无法通过程序控制不同阶段中各功能的应用顺序。最新近（很可能是最后）的固定功能机器例子是任天堂的Wii。可编程GPU使得它可以准确地决定管线中各种子阶段进行什么运算。当关于固定功能管线的研究引入一些基本原理后，最新的发展则以可编程GPU为目标。在这本书的第三版这种可编程能力默认假定存在，因为它是利用GPU的现代方法。
进一步阅读及资源
Blinn的《A trip down the graphics pipeline》是一本关于软件渲染旧书，但它是一学习实现渲染管线实现巧妙方法的好资源。对于固定功能管线，古老（但仍时常更新）的《OpenGL编程指南》（又叫红宝书）提供了关于固定功能管线及其使用的相应算法的透彻的描述。我们书本的网页http://www.realtime rendering.com，给出了各种渲染引擎的链接。