2.5 像素处理

这个阶段是所有先前阶段组合的结果，并且已经找到了在三角形或其他图元内被考虑的所有像素。像素处理阶段分为像素着色和合并，如图2.8右侧所示。像素处理是对图元内部的像素或样本执行逐像素或逐样本计算和操作的阶段。

2.5.1 像素着色

此处执行的任何逐像素着色计算，是使用内插着色数据作为输入的。最终结果是将一种或多种颜色传递到下一阶段。与通常由专用的，硬连线的芯片执行的三角形设置和遍历阶段不同，像素着色阶段由可编程GPU内核执行。为此，程序员为像素着色器（或在OpenGL中称为片元着色器）提供了一个程序，该程序可以包含任何所需的计算。这里可以使用多种技术，其中最重要的一种是纹理贴图。纹理在第6章中进行了更详细的论述。简单地说，纹理对象意味着将一个或多个图像“粘合”到该对象上，用于各种目的。图2.9描述了此过程的一个简单示例。图像可以是一维、二维或三维，其中二维图像最为常见。最简单的情况是，最终产品是每个片元的颜色值，这些值被传递到下一个子阶段。

图2.9. 没有纹理的龙模型显示在左上角。图像纹理中的片段“粘”在龙上，结果显示在左下角。

2.5.2 合并

每个像素的信息存储在颜色缓冲区中，它是一个矩形的颜色数组（每种颜色具有红色、绿色和蓝色分量）。合并阶段的职责是将像素着色阶段产生的片元颜色与当前存储在缓冲区中的颜色相结合。此阶段也称为ROP，表示“（管线）光栅操作”或“渲染输出单元”，具体取决于你访问的对象。与着色阶段不同，执行此阶段的GPU子单元通常不是完全可编程的。但是，它是高度可配置的，可以实现各种效果。

此阶段还负责解决可见性问题。这意味着当整个场景被渲染后，颜色缓冲区应该包含场景中从相机的角度可见的图元的颜色。对于大多数甚至所有图形硬件，这是通过z缓冲区（也称为深度缓冲区）算法完成的[238]。z缓冲区的大小和形状与颜色缓冲区相同，并且对于每个像素，它将z值存储到当前最接近的图元。这意味着当一个图元被渲染到某个像素时，该图元在该像素上的z值被计算并与同一像素的z缓冲区的内容进行比较。如果新的z值小于z缓冲区中的 z 值，则正在渲染的图元比之前在该像素处最靠近相机的图元更靠近相机。因此，该像素的z值和颜色将使用正在绘制的图元的z值和颜色进行更新。如果计算出的z值大于z缓冲区中的z值，则颜色缓冲区和z缓冲区保持不变。z缓冲区算法很简单，具有O(n)收敛性（其中n是正在渲染的图元数量），并且可以适用于为每个（相关）像素计算z值的任何绘图图元。另请注意，该算法允许以任何顺序呈现大多数图元，这是其流行的另一个原因。但是，z 缓冲区仅在屏幕上的每个点存储单个深度，因此它不能用于部分透明的图元。透明图元必须在所有不透明基元之后渲染，并以从后到前的顺序呈现，或使用单独的与顺序无关的算法（第5.5节）。透明度是基本z缓冲区算法的主要弱点之一。

我们已经提到颜色缓冲区用于存储颜色，而z缓冲区存储每个像素的z值。但是，还有其他通道和缓冲区可用于过滤和捕获片元信息。Alpha通道与颜色缓冲区相关联，并为每个像素存储相关的不透明度值（第5.5节）。在较旧的API中，alpha通道还用于通过alpha测试功能有选择地丢弃像素。如今，可以将丢弃操作插入到像素着色器程序中，并且可以使用任何类型的计算来触发丢弃。此类测试可用于确保完全透明的片段不会影响z缓冲区（第6.6节）。

模板缓冲区是一个离屏缓冲区，用于记录渲染图元的位置。它通常包含每像素 8 位。可以使用各种函数将图元渲染到模板缓冲区中，然后可以使用缓冲区的内容来控制渲染到颜色缓冲区和z缓冲区中。例如，假设一个实心圆已被绘制到模板缓冲区中。这可以与允许将后续图元渲染到仅存在圆圈的颜色缓冲区中的运算符结合使用。模板缓冲区可以成为生成某些特殊效果的强大工具。管线末端的所有这些功能都称为光栅操作 (ROP) 或混合操作。可以将当前在颜色缓冲区中的颜色与三角形内正在处理的像素的颜色混合。这可以启用诸如透明度或颜色样本累积等效果。如前所述，混合操作通常可以使用API进行配置，而不是完全可编程的。但是，某些API支持光栅顺序视图，也称为像素着色器排序，可实现可编程混合功能。

帧缓冲区通常由系统上的所有缓冲区组成。

当图元到达并通过光栅化阶段时，从相机的角度上看，这些可见的图元将会显示在屏幕上。屏幕显示颜色缓冲区的内容。为了避免让人类观察者在被光栅化并发送到屏幕时看到图元，使用了双缓冲。这意味着场景的渲染发生在屏幕外的后台缓冲区中。在后台缓冲区中渲染场景后，后台缓冲区的内容将与之前显示在屏幕上的前台缓冲区的内容交换。交换通常发生在垂直重描期间，这是安全的时候。

有关不同缓冲区和缓冲方法的更多信息，请参阅第5.4.2、23.6和23.7节。