第5章 真实感图形

CHAPTER 5

真实感图形的研究目标是让计算机生成如同照片般真实的图像。为了在计算机中生成真实感图形，首先需要构造场景的几何模型。

对于光源，除了给出其方位外，还应给出其发出的光线的颜色和强度等；对于物体，除外形描述外，还应指定它们各自的材料属性以确定物体表面的颜色、物体的透明度。

一般来说，用计算机生成真实感图形需要经过以下4个步骤：

(1) 建立三维场景的几何模型，确定景物表面的光照属性。

(2) 对三维场景进行取景变换，并将其投影到二维平面上。

(3) 采用隐藏面消除算法剔除当前视点处不可见的场景表面。

(4) 对显示屏幕上的每一像素，根据光照明模型，计算在该像素内可见的场景表面的光亮度。

本章将重点阐述光照明模型及物体表面光亮度的计算方法。

5.1 光照明模型

光照明模型考虑物体表面上每一个点所代表的微小面元受到来自光源或周围环境光线的照射而产生的反射或透射光亮度。物体表面上某一特定点的光照明效果与光源、观察点位置、物体表面局部几何形状、表面朝向及材料属性有关。

本节将从实用的观点出发，重点介绍真实感图形中常见的4类光照明模型，即泛光(Ambient Reflection)模型、Lambert漫反射(Diffuse Reflection)模型、Phong镜面反射(Specular Reflection)模型、Whitted整体光照明(Global Illumination)模型。

5.1.1 泛光模型

在现实世界中，物体表面由于受到光源的照射而呈现出不同的光照效果。在这些光源中，除了直接的光源(如太阳、电灯等)外，还包括来自周围环境的反射光(如来自地面、墙壁的反射光)等间接光源。

泛光模型是试图刻画周围环境反射光对物体表面照明贡献的最简单的光照明模型。它假定环境反射光沿任何方向对任何物体表面入射的光亮度都是相等的，即为各向同性的泛光，并采用一个常量来近似表示它。泛光模型可由以下简单公式予以描述：

$$I_{\text{env}} = K_a I_a \tag{5.1}$$

其中，I_{env} 为物体表面对泛光的反射光亮度；I_a 为泛射光的入射光亮度；K_a 为物体表面对泛光的反射率。

在现实生活中，泛光中包含各种不同波长的光。由于计算机图形学所采用的图形显示器仅取红、绿、蓝 3 个分量来合成所有不同的颜色，因此在具体计算物体表面的泛光光亮度时，将它分解为红、绿、蓝分量，亦即：

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} K_{aR} \cdot R_a \\ K_{aG} \cdot G_a \\ K_{aB} \cdot B_a \end{pmatrix} \tag{5.2}$$

其中，R_a、G_a、B_a 为入射到物体表面泛光的红、绿、蓝分量；K_{aR}、K_{aG}、K_{aB} 为物体表面分别对泛光红、绿、蓝分量的反射率，其取值范围为 0～1。

图 5.1 所示的是犹他茶壶的泛光照明效果。由于仅考虑了泛光反射光亮度，真实感显示的三维茶壶看起来如同是用单一颜色填充的二维多边形物体。

图 5.1　犹他茶壶的泛光照明效果

5.1.2　Lambert 漫反射模型

现实世界中直接光源发出的光线只能沿一定方向照射到物体表面上。也就是说，直接光源对物体表面的照射是有方向性的。

入射到物体表面的光线一部分被物体吸收，另一部分经物体透射或被表面反射出去。物体表面的反射光可分为漫反射光和镜面反射光，其中漫反射光是物体表面对入射光线朝各个方向的均匀反射，如图 5.2 所示，其大小只与入射光的光亮度和入射方向有关，而与漫反射光的反射方向无关。纯漫射表面只产生漫反射。自然界的大部分表面如地面、房屋、树木、花草均可认为是纯漫射面。

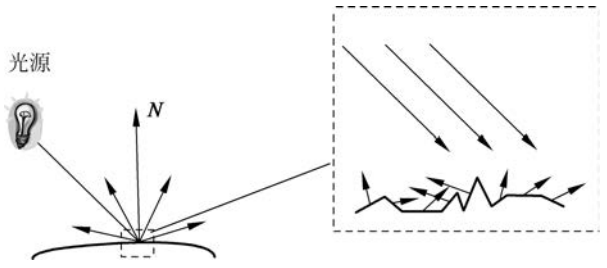


图 5.2　均匀反射的光线

Lambert 最早总结出上述规律。他指出，漫反射光亮度和光源入射角（入射光线和表面法向量的夹角）的余弦成正比，如图 5.3 所示。漫反射光亮度计算公式如下：

$$I_d = K_d I_e \cos\alpha \tag{5.3}$$

其中，I_d 为表面漫反射光的光亮度；K_d 为物体表面的漫反射率；I_e 为发自光源的入射光的光亮度；α 为光源入射角。

同样，在具体计算时，K_d 和 I_e 将分别由其红、绿、蓝分量代替。下面看如何计算 $\cos\alpha$。

如图 5.3 所示，设点 A 是景物表面上需要计算漫反射光亮度的一个采样点，$\boldsymbol{N}$ 是点 A 处物体表面的法向量，$\boldsymbol{L}$ 表示光源的入射方向矢量。已知两个矢量的点积公式为：

$$(\boldsymbol{N},\boldsymbol{L}) = |\boldsymbol{N}| \cdot |\boldsymbol{L}| \cos\alpha \tag{5.4}$$

其中，$|\boldsymbol{N}|$ 和 $|\boldsymbol{L}|$ 是矢量的长度，于是：

$$\cos\alpha = \frac{(\boldsymbol{N},\boldsymbol{L})}{|\boldsymbol{N}| \cdot |\boldsymbol{L}|} = \frac{(\boldsymbol{N},\boldsymbol{L})}{\sqrt{(\boldsymbol{N},\boldsymbol{N})}\ \sqrt{(\boldsymbol{L},\boldsymbol{L})}} \tag{5.5}$$

式(5.5)中比较复杂的是开方运算。Lalonde P 给出了一个快速计算平方根的方法并给出了 C 语言源程序。

从式(5.3)不难看出，当入射光线垂直于物体表面时，光源入射角 $\alpha=0°$，此时，$\cos\alpha=1$，漫反射光亮度达到最大值。随着光源入射角 α 的增大，漫反射光亮度逐渐减小。当 $\alpha=90°$ 时，$\cos\alpha=0$，此时漫反射光亮度达到最小值 0。实际应用 Lambert 模型时，有可能出现光源入射角 α 位于 90°～180°的情况，此时，$\cos\alpha$ 取负值，需要人为地将漫反射光亮度的值取为 0。

合并式(5.1)和式(5.3)，一个综合了泛光和表面漫反射分量的光照明模型可表达为：

$$I = K_a I_a + K_d I_e \cos\alpha \tag{5.6}$$

其中，I_a 是环境泛光入射光亮度，一般取值范围为 $0.02I_e$～$0.2I_e$；I 为景物表面的反射光亮度。上式常称为 Lambert 光照模型。图 5.4 是采用上述光照明模型生成的真实感图形。

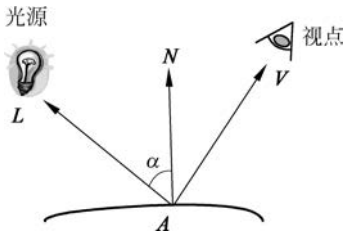


图 5.3　Lambert 漫反射示意图

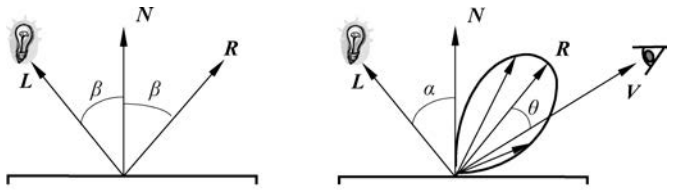
图 5.4　Lambert 光照模型的光照明效果

5.1.3　Phong 镜面反射模型

现实世界有许多物体表面很光滑，如玻璃器皿、家具、汽车车身等，当人们从某些视线方向观察这些表面时，往往会看到表面上呈现出特别亮的区域，即所谓的高光(Highlight)，如图 5.5 所示。事实上，光滑表面除了有漫反射光以外，还存在镜面反射光。

图 5.5　镜面高光

镜面反射光是一种朝向一定方向的反射光，它遵从光的反射定律，即反射光和入射光对称地分布于物体表面法线方向的两侧，反射角等于入射角，如图 5.6(a)所示。对于非理想镜面，通常认为其表面由许多朝向沿表面宏观法向附近随机分布的微平面构成，其镜面反射光分布于物体表面镜面反射方向的周围，如图 5.6(b)所示。



(a) 理想镜面反射　　(b) 非理想镜面反射

图 5.6　镜面反射示意图

可以采用余弦函数的幂次来模拟镜面反射光：

$$I_s = k_s I_e \cos^n \theta \tag{5.7}$$

其中：

I_s 为物体表面镜面反射光亮度。

I_e 为发自光源的入射光的光亮度。

θ 为镜面反射方向和视线方向的夹角。

k_s 表示表面的镜面反射率，对于绝大多数表面材料，镜面反射光的光谱组成及其分布曲线与入射光基本相似，因此 k_s 可简单地取为一个标量系数。

n 为镜面反射光的会聚指数，或称为高光指数，它是一个正实数，其取值取决于表面材料的属性和表面的光滑程度。一般为从一到数百不等。对于光滑的表面，其镜面反射光的会聚程度较高，此时可将 n 值取得大一些；而对于光滑度较低的表面，其镜面反射光呈发散状态，此时可将 n 值取得小一点。

从式(5.7)不难看出，镜面反射光亮度不仅取决于物体表面的法线方向，而且依赖于光源和观察者的相对位置。只有当观察者位于比较合适的方位时，才可以观察到物体表面某些区域呈现的高光。当观察者方位改变时，高光区域的位置也会随之移动甚至消失。

式(5.7)中需要计算 $\cos\theta$ 的值，这涉及镜面反射方向 $\boldsymbol{R}$ 和视线方向 $\boldsymbol{V}$ 两个矢量。当用户指定观察者的位置后，$\boldsymbol{V}$ 的计算是非常直接的。下面看如何计算 $\boldsymbol{R}$。

如图 5.7 所示，不妨假设光线方向 $\boldsymbol{L}$、表面法向 $\boldsymbol{N}$ 和视线方向 $\boldsymbol{V}$ 均为单位矢量，入射角为 β，于是有：

$$\boldsymbol{L} + \boldsymbol{R} = 2\boldsymbol{S} \tag{5.8}$$

$$\boldsymbol{S} = |\boldsymbol{L}| \cos\beta \boldsymbol{N} = \cos\beta \boldsymbol{N} = (\boldsymbol{L}, \boldsymbol{N})\boldsymbol{N} \tag{5.9}$$

综合上面两式可得：

$$\boldsymbol{R} = 2(\boldsymbol{L}, \boldsymbol{N})\boldsymbol{N} - \boldsymbol{L} \tag{5.10}$$

一旦 $\boldsymbol{R}$ 和 $\boldsymbol{V}$ 确定，即可参考式(5.5)计算 $\cos\theta$。但因为式(5.10)的计算涉及矢量点乘，在实际应用中，经常采用 $\cos\gamma$ 来取代 $\cos\theta$，其中 γ 是表面法向 $\boldsymbol{N}$ 和角平分矢量 $\boldsymbol{H} = (\boldsymbol{L} + \boldsymbol{V})/2$ 的夹角，如图 5.8 所示。$\boldsymbol{H}$ 是将入射光线反射到视线方向的虚拟镜面的法向量。此时，镜面反射光亮度公式可改写为：

$$I_s = k_s I_e \cos^n \gamma \tag{5.11}$$

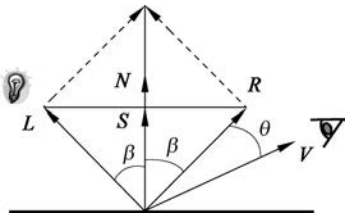


图 5.7　镜面反射方向的计算

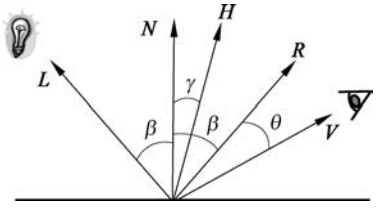


图 5.8　虚拟镜面法向量 $\boldsymbol{H}$

显然，$\cos^n\gamma$ 也能描述射向观察者的镜面反射光的分布函数，但其计算量比 $\cos^n\theta$ 要小很多。

对于一般反射面来说，其表面反射光中既有漫反射分量，也有镜面反射分量，还包含泛光反射分量。综合考虑 3 方面的贡献和场景中多光源照明的情形，一个较完整的光照模型可表述如下：

$$I = K_a I_a + \sum_{i=1}^{m} I_i (K_d \cos\alpha + k_s \cos^n\gamma) \tag{5.12}$$

分别将 $\cos\alpha$ 和 $\cos\gamma$ 写成矢量积形式，得：

$$I = K_a I_a + \sum_{i=1}^{m} I_i (K_d (\boldsymbol{N}, \boldsymbol{L}) + k_s (\boldsymbol{N}, \boldsymbol{H})^n) \tag{5.13}$$

其中，m 为光源的个数。上式由 B. T. Phong 在 1973 年提出，在计算机图形学中称为 Phong 镜面反射模型。图 5.9 中所示的是采用 Phong 镜面反射模型的绘制效果。

图 5.9 Phong 镜面反射模型的光照明效果

5.1.4 Whitted 整体光照明模型

Lambert 光照模型和 Phong 镜面反射模型仅考虑了从光源直接发出的光线对物体表面光亮度的贡献，而没有考虑光线在物体之间的相互反射（如在镜子中可以看到其他物体）和透射（如可以看到透明物体后面的其他物体），因此它们被称为局部光照明模型。显然，仅用局部光照明模型生成的图形在真实感上是有缺陷的。

为模拟现实世界中景物表面之间的镜面反射和透射现象，T. Whitted 假设从某一观察方向 $\boldsymbol{V}$ 所观察到的物体表面某点 P 的光亮度的贡献来自 3 个方面：其一，由光源直接照射引起的反射光亮度 I_c；其二，沿 $\boldsymbol{V}$ 的镜面反射方向 $\boldsymbol{r}$ 入射的环境光 I_s 在表面产生的镜面反射光；其三，沿 $\boldsymbol{V}$ 的规则透射方向 $\boldsymbol{t}$ 入射的环境光 I_t 通过透射在表面上产生的规则透射光，如图 5.10 所示。

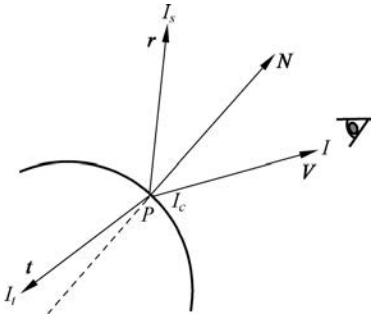


图 5.10 镜面反射和规则透射

Whitted 基于上述假设建立了 Whitted 整体光照明模型：

$$I = I_c + k_s I_s + k_t I_t \tag{5.14}$$

其中：

I_c 为由光源直接照射在表面上引起的反射光亮度，可直接采用 Phong 镜面反射模型计算。

I_s 为沿 $\boldsymbol{V}$ 的镜面反射方向 $\boldsymbol{r}$ 入射到表面上的环境光在表面上产生的镜面反射光。

I_t 为沿 $\boldsymbol{V}$ 的规则透射方向 $\boldsymbol{t}$ 入射到表面上的环境光通过透射在表面上产生的规则透射光。

k_s 为表面的镜面反射率。

k_t 为表面的透射率。

为利用 Whitted 模型计算物体表面某一点(例如,图 5.10 中的点 P)的光亮度,除了利用 Phong 镜面反射模型计算出 I_c 之外,还必须计算出 I_s、I_t 的值。

I_s、I_t 的计算涉及一个复杂的过程。以图 5.10 为例,为计算点 P 处 I_s 的值,可假设观察者位于 P 点位置,沿 $\boldsymbol{r}$ 方向对场景进行观察。因此,必须首先求出 $\boldsymbol{r}$,并利用 Whitted 模型计算场景沿 $\boldsymbol{r}$ 方向投射到位于点 P 处的观察者眼中的光亮度,这导致了一个递归的过程。I_t 的计算也类似。

下面将在 5.3 节详细介绍 Whitted 模型的求解方法——光线跟踪(Ray Tracing)。

5.1.5 光照明模型的进一步完善

从物体表面上的某一点朝某一特定方向反射的光线的光亮度可由一个与入射光线方向、物体表面法向、观察方向等有关的函数来描述。通常这个函数是各向异性的,即如果绕该点表面法向转动该物体的话,所观察到的光亮度是有变化的。最常见的一个例子是丝绸,它的颜色将随着其绒面方向的不同而不同。从物理的角度来说,这个反射函数是一个与光的波长相关的光谱函数。因此,精确的反射函数模型的计算非常困难。上面介绍的几个光照明模型只是对光照明现象的简单模拟,能够给人们提供视觉上可以接受的效果而已。

光照明模型一直是计算机图形学研究的重点,其主要目标是更深入地理解物体表面结构及其与光线之间的相互作用机理,从物理上更准确、更精细地模拟光照明现象。在计算机图形学近半个世纪的实践中,研究人员提出了许多光照明模型,它们大致可以分为 3 类:以 Phong 模型为代表的基于经验的简单光照明模型、以 Blinn 模型和 Cook-Torrence 模型为代表的基于物理的光照明模型,以及双向反射分布函数(Bi-Directional Reflectance Distribution Function,BRDF)模型。

BRDF 是指一个微小面元上,特定反射光的辐照度与入射光的辐照度的比值,是一个无维度的标量。由于它是一个关于入射光入射角度(θ_i,Φ_i)和反射光出射角度(θ_o,Φ_o)的四维函数,因此有可能基于正交基函数构建其拟合表示。球面调和函数、小波、Zernike 多项式均可用作表示 BRDF 的正交基,而 Kautz 等则将 BRDF 表示为可分函数(Separable Function)之和。从理论上讲,这些表示均能精确地表示任意 BRDF,但实际上却往往需要数十、数百乃至数千个系数才能获得较为理想的逼近效果。Matusik 等提出了一个数据驱动的反射模型,他们将每一获取的 BDRF 视为取自所有可能的 BRDF 构成的空间的一个高维向量,并应用线性(子空间)与非线性(流形)降维工具以找出能表征测量数据的更低维表示。这样,用户可定义少量(15～30 个)具有感知意义的参数方向,实现对降

图 5.11 采用数据驱动的反射模型绘制的茶壶

维 BRDF 空间的导航。图 5.11 是 Matusik 等采用这种方法绘制的带有油污指纹的不锈钢茶壶。从本质上说，BDRF 仅主要刻画物体表面的朝向细节，考虑到纹理映射是刻画物体表面空间细节的主要方法，Mcallister 在其博士学位论文中，结合纹理映射和 BDRF 方法，提出了六维的空间 BRDF 方法。

虽然采用 BRDF 模型绘制的物体具有很强的真实感，但与基于经验的简单光照明模型及基于物理的光照明模型仅需数个系数相比，BRDF 模型所需的系数明显偏多，加之 BRDF 模型需要存储大量的相关采样数据以及需要较大的计算量，因此目前 BRDF 模型仍未得到广泛应用。

5.2 多边形物体的明暗处理

在计算机图形学中，场景中的许多物体都采用多边形表示。对于理想漫射体，最简单的方法是先依据 Lambert 模型按每一个(可见的)多边形的法向量计算出一个颜色值，然后将该颜色值赋给该多边形在屏幕上的投影所覆盖的全体像素。

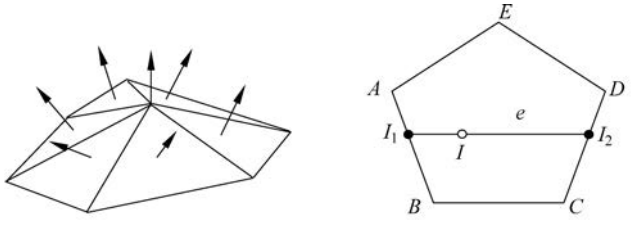
图 5.12 Flat 明暗处理图例

上述绘制方法称为 Flat 明暗处理(Shading)。该方法处理简单，但景物表面上相邻的多边形之间颜色差异较大，存在马赫带(Mach Bands)效应(光亮度变化不连续的边界处呈现亮带或黑带)，如图 5.12 所示。为消除这种相邻多边形之间颜色不连续的现象，可以通过插值方法来处理。

5.2.1 Gouraud 明暗处理

Gouraud 明暗处理又称光亮度插值明暗处理。首先，为多边形物体的每一个顶点赋一个法向量。顶点处的法向量可通过计算所有共享该顶点的多边形的法向量的平均值得到，如图 5.13(a)所示。然后，利用 Phong 模型计算每一顶点处的光亮度。多边形内部各点处的光亮度值则通过对多边形顶点处的光亮度的双线性插值得到。

如图 5.13(b)所示，假设多边形 $ABCDE$ 每一顶点处的光亮度已计算好，e 是经过点 I 的扫描线，它与多边形相交于点 I_1 和点 I_2。于是，首先利用点 A、点 B 的光亮度线性插值得到点 I_1 处的光亮度；然后利用点 D、点 C 处的光亮度线性插值得到点 I_2 处的光亮度；最后利用点 I_1 和点 I_2 处的光亮度线性插值得到点 I 处的光亮度。



(a) 计算多边形顶点处的法向量　　(b) 计算多边形内一点的光亮度

图 5.13 Gouraud 明暗处理中的多边形顶点法向量计算及光亮度计算

Gouraud 明暗处理简单快速，所生成的图形在真实感上较 Flat 明暗处理有了较大的提高，如图 5.14 所示，但马赫带效应依然存在，这是由 Gouraud 明暗处理中线性插值的本质决定的。为消除马赫带效应，可以采用高次插值，但这势必带来计算量的问题。Gouraud 明暗处理的另一缺陷是不能正确模拟高光。

图 5.14　Gouraud 明暗处理图例

5.2.2　Phong 明暗处理

Phong 明暗处理(Phong Shading)又称法向插值明暗处理。与 Gouraud 明暗处理直接插值顶点处的光亮度不同，Phong 明暗处理插值顶点处的法向。如图 5.15 所示，顶点 P_A、P_B 处的法向可取围绕它们的多边形的法向量的平均值。位于 P_A、P_B 连线上的点 P_1、P_2 和 P_3 处的法向可以通过线性插值顶点 P_A、P_B 的法向得到。然后利用 Phong 模型分别计算其光亮度。对于多边形内部的点，其法向可采用类似于 Gouraud 明暗处理中的双线性插值方法处理。

仍以图 5.13(b)为例。假设多边形 A、B、C、D、E 每个顶点处的法向量已计算好，首先对点 A、点 B 处的法向量做线性插值得到点 I_1 处的法向量，对点 D、点 C 处的法向量做线性插值得到点 I_2 处的法向量；然后，取点 I_1 和点 I_2 处的法向量的线性插值得到点 I 处的法向量；最后，根据计算所得到的法向量，利用 Phong 模型计算点 I 处的光亮度。

Phong 明暗处理由于对法向进行插值，故不仅能较好地模拟高光，而且相邻多边形之间的光亮度过渡也比 Gouraud 明暗处理更自然，但其计算量比 Gouraud 明暗处理要大得多。图 5.16 是用 Phong 明暗处理生成的真实感图形。

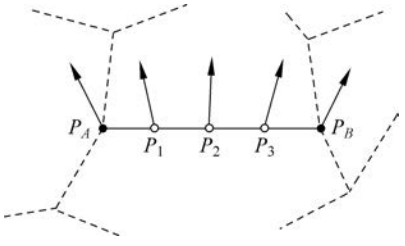


图 5.15　Phong 明暗处理中法向的线性插值

图 5.16　Phong 明暗处理图例

5.3　光线跟踪算法

光线跟踪是迄今为止最为成功的生成真实感图形的算法之一。光线跟踪不仅算法简单，而且与前面介绍的明暗处理方法相比，它所生成的图形要真实得多。图 5.17 是 2008 年 Tran G. Glasses 采用光线跟踪算法生成的图例，请注意图中玻璃的镜面反射效果、透明效果。

5.3.1　基本原理

光线跟踪技术的前身是1968年Appel提出的光线投射(Ray Casting)法。光线投射法生成真实感图形的步骤如下：首先从视点出发通过屏幕上的每一像素的中心向场景发出一条光线，并求出该条光线与场景中物体的全部交点；然后将各交点沿着光线投射方向排序，获得离视点最近的交点；最后依据局部光照明模型计算该交点处的光亮度，并将所得光亮度值赋给该像素。当所有屏幕像素都处理完毕时，即得到一幅真实感图形，如图5.18所示。

图5.17　光线跟踪算法生成的图例

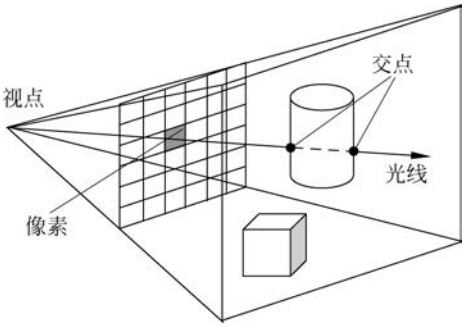


图5.18　光线投射原理

1980年，Whitted为求解他提出的整体光照明模型，对光线投射算法进行了扩展使其能处理镜面反射和规则透射。其原理如下：

首先，从视点出发向屏幕上的每一像素发出一条光线，求出光线与场景所有交点中离视点最近的交点 P，并依据局部光照明模型计算该交点处的颜色值 I_c，同时在交点处沿着该光线的镜面反射方向和透射方向各衍生出一条光线(若点 P 所在的表面非镜面或不透明，则无须衍生出相应的光线)；然后，分别对衍生出的光线递归地执行前两个步骤，以计算来自镜面反射方向和透射方向周围环境对点 P 光亮度的贡献 I_s 和 I_t；最后，依据Whitted光照明模型即可计算出点 P 处的光亮度，并将计算出的光亮度赋给该像素。当所有屏幕像素都处理完毕时，即可得到一幅真实感图形。

上述光线衍生及其在环境景物之间传递的过程就是光线跟踪过程，它是对真实世界光照明过程的逆过程的一种近似。光线跟踪算法逆向跟踪从光源发出的光经由物体之间的多次反射和折射后投射到物体表面，最终进入人眼的过程，如图5.19所示。光线跟踪过程一般可用一棵称为光线树的二叉树来表示，如图5.20所示。

在算法中，光线跟踪的递归过程并非无休止地进行下去，一般采用下面条件之一作为光线跟踪递归过程终止的条件。

(1) 光线与环境中的任何物体均不相交，或交于纯漫射面。

(2) 被跟踪的光线返回的光亮度值对像素颜色的贡献很小。

(3) 已递归到给定深度。

5.3.2　光线跟踪算法的伪语言描述

下面给出光线跟踪算法的伪语言描述，注意前面提到的递归的3个出口是如何应用到光线跟踪子函数RayTrace()中的。

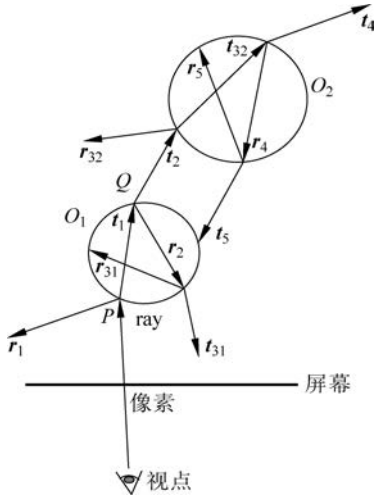


图 5.19 光线衍生过程

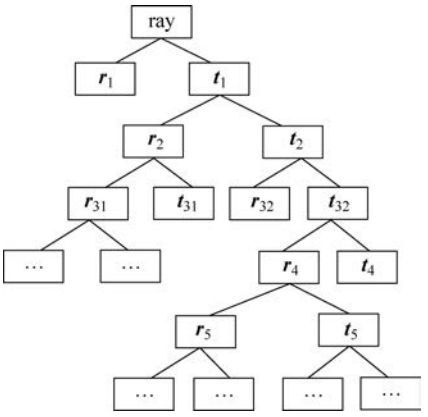


图 5.20 图 5.19 中光线衍生过程的光线树表示

```
main ()                              //说明: 主函数
{
    for(需要计算颜色的每一像素 pixel) {
        确定通过视点 V 和像素 pixel 的光线 R;
        depth = 0;                   // 说明: 递归深度
        ratio = 1.0;                 //说明: 当前光线的衰减系数,1.0 表示无衰减
        // 说明: color 是经计算后返回的颜色值
        RayTrace(R, ratio, depth, color);
        置当前像素 pixel 的颜色为 color;
    }
} // 说明: 主函数 main()结束

RayTrace(R, ratio, depth, color)  //说明: 光线跟踪子函数
{
    if(ratio < THRESHOLD) {
        置 color 为黑色;
        return;
    }
    if(depth > MAXDEPTH) {
        置 color 为黑色;
        return;
    }
    光线 R 与场景中的所有物体求交。若存在交点,则找出离 R 起始点最近的交点 P。
    if(交点不存在) {
        置 color 为黑色;
        return;
    }
    用局部光照明模型计算交点 P 处的颜色值,并将其存入 local_color;
    if(交点 P 所在的表面为光滑镜面) {
        计算反射光线 Rr;
        RayTrace(Rr, ks * ratio, depth + 1, reflected_color);
    }
    if(交点 P 所在的表面为透明表面) {
        计算反射光线 Rt;
```

```
        RayTrace(Rt, kt * ratio, depth + 1, transmitted_color);
    }
    依照 Whitted 模型合成最终的颜色值,即:
    color = local_color + ks * reflected_color + kt * transmitted_color;
    return;
} // 说明: 光线跟踪子函数 RayTrace()结束
```

5.3.3　阴影计算

阴影(Shadow)指的是场景中未被光源直接照射到或仅被部分光源直接照射到的区域。阴影区域分为本影和半影。本影是场景中完全未受到光源直接照射的区域,而半影指的是场景中仅受部分光源直接照射的区域。阴影生成的方法很多,常用的有影域多边形法、曲面细节多边形法、z 缓冲器法等。

在光线跟踪算法中,要测试场景中某点 P 是否被某一点光源 L 直接照射,只需从 P 出发向光源 L 发射一条阴影测试光线 $\boldsymbol{R}$ 即可。若 $\boldsymbol{R}$ 在到达 L 的途中与场景中的物体不相交,则点 P 受光源 L 直接照射;反之,点 P 被位于它与光源 L 之间的某一物体所遮挡,若遮挡物为不透明体,则点 P 位于光源的阴影之中,如图 5.21 所示。

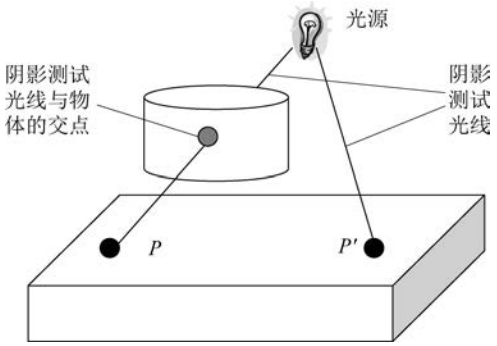


图 5.21　阴影测试

假设场景中有 m 个点光源(对于场景中的线光源或面光源,可将其离散后用多个点光源近似表示),则包含阴影计算的 Phong 模型可调整为:

$$I(P)=K_a I_a+\sum_{i=1}^{m}(f_i(P)I_i(k_{d_i}(\boldsymbol{N},\boldsymbol{L})+k_s(\boldsymbol{N},\boldsymbol{H})^n)) \tag{5.15}$$

其中,P 为待计算颜色的点,而:

$$f_i(P)=\begin{cases}0 & \text{若 } P \text{ 未受光源 } i \text{ 直接照射}\\ 1 & \text{若 } P \text{ 受光源 } i \text{ 直接照射}\end{cases} \tag{5.16}$$

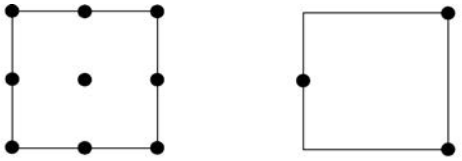
5.3.4　反走样

由于计算机屏幕采用方形(或矩形)显示,而光线跟踪算法本质上是对画面的点采样,因此容易导致图形走样。从理论上说,走样现象不可能完全消除,但可以采用一些称为反走样(Anti-Aliasing)的技术来减轻图形走样对画面质量的影响。

通常情况下,光线跟踪程序仅通过屏幕上每一像素的中心点发射一条主光线,经光线跟踪返回一个光亮度值作为该像素的显示值。如果对一像素的不同子区域发射多条光线(例

如，均匀发射出 9 条光线，如图 5.22(a)所示，沿这些光线分别进行光线跟踪可返回多个光亮度值，将这些光亮度值的平均值作为该像素的显示光亮度，可在很大程度上提高该像素所显示颜色的准确性。上述方法称为超采样(Supersampling)，是光线跟踪算法中一种常用的反走样处理手法。

自适应超采样方法是对上述超采样方法的改进。自适应超采样方法首先发出 3 条经过挑选的光线，如图 5.22(b)所示，经光线跟踪得到 3 个光亮度值。假如这 3 个光亮度值非常接近，那么从其他 6 个位置发出的光线所返回的光亮度值极有可能和这 3 个光亮度值非常接近。因此，只需取这 3 个光亮度值的平均作为该像素的显示光亮度即可。反之，如果返回的 3 个光亮度值差异较大，则将该像素进一步剖分成更小的区域，发出更多采样光线，再取这些采样光线所返回的光亮度值的平均值赋给该像素。事实上，对整个屏幕而言，在绝大多数的像素上只需发出 3 条光线即可满足要求。



(a) 利用9条光线的超采样　(b) 从3条光线起始的自适应超采样

图 5.22　超采样和自适应超采样反走样方法

在超采样方法中，由于每像素上采样光线的数目和位置分布固定，不仅计算量大大增加，而且会在画面上留下人工处理的痕迹。进一步解决这一问题的方法是采用随机光线跟踪(Stochastic Ray Tracing)和统计光线跟踪(Statistical Ray Tracing)。

5.3.5　加速技术

光线跟踪算法涉及光线与场景中物体的大量求交运算。以生成一幅分辨率为 1280×1024 像素的真实感图形为例，需要发射 1280×1024 条主光线。假设每条主光线通过反射和折射平均衍生出 5 条光线，再假设场景中包含 4 个点光源(在每一被跟踪光线与场景的交点处需要发出 4 条阴影探测光线)和 100 000 个独立的物体表面，则生成这样的一幅图像共需要进行 1280×1024×5×4×100 000 次直线与各类物体表面的求交运算。如果还需要进行反走样处理，则计算量会成倍增加。因此，如何对光线跟踪算法进行加速，是关系到算法是否实用的关键。

实际上，真正与某一被跟踪的光线相交的场景表面数量极其有限，如能简便地判定光线与场景中景物表面的相对位置关系，避免光线与实际不交的景物表面的求交运算，将显著提高光线跟踪算法的效率。

一种典型的光线跟踪加速算法是包围盒技术，即将场景中的所有表面按其空间位置关系分层次组织成树状结构，其中树的根节点表示整个场景，中间节点则分别表示场景中空间位置较为接近的一组表面，叶节点即为单个景物表面。每一个节点中的表面或表面片集合都用一个形状简单的包围盒包裹起来。若光线与包围盒不相交，则必定不与其中所含的景物面片相交；只有当光线与包围盒相交时，才进行光线与其中所含的景物面片的求交运算。显然，采用包裹相对紧密的包围盒有助于提高包围盒测试的准确性。常用的包围盒体有长

方体、平行 $2n$ 面体等，如图 5.23 所示。

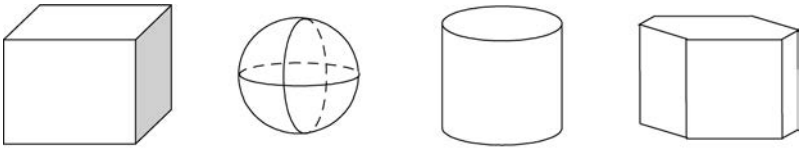


图 5.23　常用的包围盒体

另一种光线跟踪加速算法是景物空间分割技术，即将景物空间分割成一个个小的空间单元。被跟踪的光线仅与它所穿过的空间单元中所含的物体表面进行求交测试。典型算法包括基于空间均匀剖分的三维 DDA 算法、基于空间二叉分割(BSP)的算法和基于空间自适应剖分的八叉树算法。这类算法的优势在于可充分利用相邻空间单元的空间连贯性，使光线快速跨越一个个空单元，从而迅速到达非空单元，求得光线与景物的第一个交点。

5.3.6　光线跟踪实例程序

光线跟踪算法经过近 30 年的发展，已成为高度成熟的真实感图形绘制算法，在计算机动画、工程设计等领域已得到非常成功的应用。目前市场上著名的三维动画软件，如 Maya、Softimage、3ds Max 等，其真实感绘制模块均无一例外地采用了光线跟踪算法。除了商品化绘制引擎(如 Mental Ray)以外，一些组织也致力于研发真实感绘制程序，如由 POV-小组(POV-Team^{TM})开发的 POV-Ray(the Persistence of Vision Ray-Tracer)就是一款开放源代码的、基于光线跟踪的真实感绘制软件包，其前身是由 Buck 和 Collins 编写的 DKBTrace 2.12。在用 POV-Ray 生成真实感图形时，用户需要首先建立场景描述文件，然后调用光线跟踪绘制程序。幸运的是，虽然 POV-Ray 是一个开源软件包，但它提供了较为完整的关于如何建立场景文件及使用光线跟踪程序的文档。此外，它还提供了许多预定好的场景、三维形状以及材质，供用户学习以及在用户自己建立的场景文件中调用。图 5.24 是采用 POV-Ray 绘制的犹他茶壶，本书的配套资源中提供了相关的场景文件。

本书的配套资源中还提供了一个简明的光线跟踪绘制框架程序，其中涵盖了简单的场景造型、主要光线及阴影光线的发射及接收、光线与球体的求交、简单反走样等光线跟踪算法要素。图 5.25 为该框架程序生成的真实感图形。需要指出的是，该程序框架并未涉及取景变换、反射及透射光线计算等要素，建议读者结合 5.3.2 节，将这些光线跟踪要素补充到该框架中。

图 5.24　POV-Ray 绘制的犹他茶壶

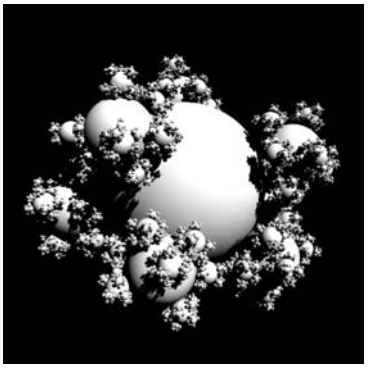


图 5.25　光线跟踪绘制框架程序生成的真实感图形

5.4 纹理映射

计算机图形学中的纹理(Texture)一词通常是指物体的表面细节。如图 5.26 所示的砖墙、草地和天空,整个场景仅包含 8 个多边形,其中砖墙采用长方块表示,而草地和天空均各由一个四边形表示。由于分别采用了二维图像作为纹理,因此画面真实感大大增强。

(a) 场景多边形表示　　(b) 纹理映射效果

图 5.26　纹理示例:砖墙、草地和天空

1974 年,Catmull 首先采用二维图像来定义物体表面的颜色(漫反射系数),这种纹理被称为颜色纹理;而 Blinn 则于 1978 年提出了在光照模型计算中适当扰动物体表面的法向生成表面凹凸纹理的方法。上述两类纹理至今仍然是经常使用的纹理类型。利用纹理映射技术还可以改变物体的其他表面属性,如镜面反射属性、透明度和折射率等。纹理也可以直接在三维空间中定义。

5.4.1 颜色纹理

根据 5.1.2 节给出的光照模型式(5.3),在计算景物表面任一点 P 的光亮度时,首先必须确定该点处的入射光线矢量 $\boldsymbol{L}$、该点处表面的法向量 $\boldsymbol{N}$ 以及表面的颜色(漫反射率 K_d)。显然,很容易根据点 P 的空间位置以及该点所在表面的方程计算 $\boldsymbol{L}$ 和 $\boldsymbol{N}$,但确立表面上任一点的颜色却并不简单。自然界和生活中的表面通常具有丰富的颜色细节,如大理石表面和木质家具表面呈现清晰的自然纹理,罐头和商品包装盒的表面印有各种装饰性纹理,房间墙上贴的字画也可认为是一种附着在表面的人工纹理。在上述情形中,表面上各点的颜色依据纹理图案呈现有序的分布。

确定表面上的颜色纹理通常有两种方法:一种是依据预先建立的表面的纹理模型,计算表面上各点的颜色值,如表面上规则的几何图案即可通过计算机建模,确定表面上各点应取的颜色值;另一种是建立表面上的每一点和一已知图像上的点的对应关系,取图像上相应点的颜色值作为表面上各点的颜色值,这种方法在计算机图形学中称为纹理映射(Texture Mapping)。

为了使映射在景物表面的颜色纹理不因摄像机的方位或景物在空间位置的改变而漂移,常采用景物表面的参数化表示来确立表面的纹理映射坐标,设景物表面的参数化表示为 $f(u,v)$,而纹理图像可表示为 $T(s,t)$,建立景物表面参数空间(u,v)和纹理图像参数空间(s,t)之间的一一对应关系,即可实现纹理图像在景物表面的映射。

5.4.2　几何纹理

颜色纹理描述了光滑表面上各点处的颜色分布。自然界还存在另一类景物表面，如橘子、树干、粗纹布料、混凝土墙面等，其表面凹凸不平，具有丰富的几何细节。自然也可以将上述表面的照片经数字化后作为颜色纹理映射到相应的几何表面，以增加真实感，但任何照片都是在给定视点、视线方向和光照条件下景物表面的图像，无法表达在改变视点和光照条件下景物表面几何细节所呈现出的不同的光照效果。

Blinn 提出，可以在不改变物体宏观几何的前提下，用凹凸映射(Bump Mapping)技术模拟出物体表面粗糙的、褶皱的、凹凸不平的光照效果，其思想是在应用光照明模型计算景物表面光亮度时，对景物表面法向进行微小的扰动。

假设物体表面 $\boldsymbol{S}$ 由参数方程 $\boldsymbol{S}=\boldsymbol{S}(u,v)$ 表示，于是，对于 $\boldsymbol{S}$ 上的任意一点 (u,v)，其法向可由其沿 u、v 方向的偏导数 $\boldsymbol{S}_u$、$\boldsymbol{S}_v$ 的叉积 $\boldsymbol{n}=\boldsymbol{S}_u\times\boldsymbol{S}_v$ 定义。

Blinn 通过沿着表面 $\boldsymbol{S}$ 的法线方向叠加一个微小的扰动量 $P(u,v)$ 定义了一张新的表面 $\boldsymbol{S}'$，即：

$$\boldsymbol{S}'=\boldsymbol{S}(u,v)+P(u,v)\frac{\boldsymbol{n}}{|\boldsymbol{n}|} \tag{5.17}$$

新表面的法向可用 $\boldsymbol{n}'=\boldsymbol{S}'_u\times\boldsymbol{S}'_v$ 计算。在计算表面 $\boldsymbol{S}$ 的光亮度时，Blinn 使用新表面的法向量 $\boldsymbol{n}'$ 取代原光滑表面的法向量 $\boldsymbol{n}$，生成物体表面的凹凸效果。具体地，$\boldsymbol{n}'$ 可写为：

$$\boldsymbol{n}'=\boldsymbol{n}+\frac{P_u(\boldsymbol{n}\times\boldsymbol{S}_v)}{|\boldsymbol{n}|}+\frac{P_v(\boldsymbol{S}_u\times\boldsymbol{n})}{|\boldsymbol{n}|} \tag{5.18}$$

其中，右端第一项为原光滑表面 $\boldsymbol{S}$ 的法向，而第二、三项则为对表面 $\boldsymbol{S}$ 原法向的扰动向量。扰动函数 $P(u,v)$ 既可解析定义，也可通过查找表(如二维图像)的形式定义。采用不同的扰动函数将获得不同的效果。利用较平滑的扰动函数将生成比较规则的凹凸特征，而利用随机度较高的扰动函数将生成较粗糙的表面效果。图 5.27 所示为球体表面分别以规则图案和不规则噪声图案为纹理的凹凸映射效果，注意球体表面虽然看似有凹凸，但其轮廓仍是光滑的。图 5.28 比较了颜色纹理效果与凹凸纹理效果。从图中不难看出，对于类似于花岗岩等表面有微小凹陷和凸起的物体，结合颜色纹理和凹凸纹理，可以获得较为真实的模拟效果。

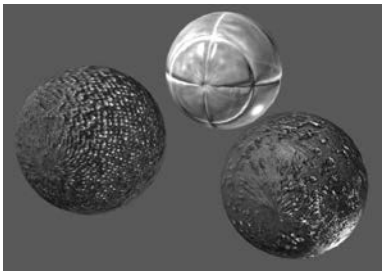


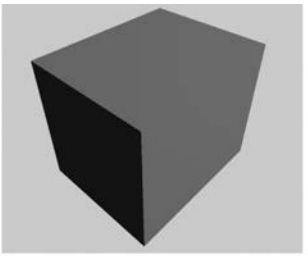
图 5.27　凹凸纹理示例

5.4.3　纹理反走样

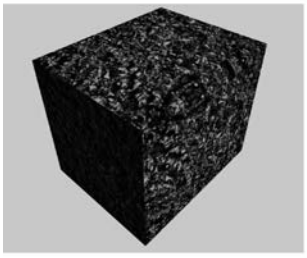
无论是颜色纹理还是几何纹理都是一种表面细节，若采用常规点采样方式绘制画面，极易引起纹理走样，如图 5.29 所示。从本质上说，纹理走样是由于对景物表面欠采样引起的，即屏幕上两相邻像素所对应的景物表面的采样点被映射到纹理空间不相邻的两像素，从而导致纹理空间某些信息丢失，如图 5.30 所示。

目前已提出了多种纹理反走样技术。Catmull 提出的纹理反走样算法基于前置滤波方法。该算法首先确定屏幕像素 P 上可见的景物表面区域 A，然后将该区域直接映射到纹理

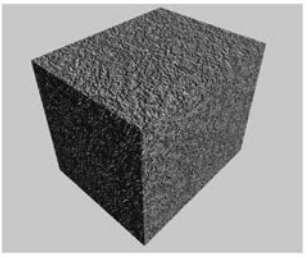
(a) 花岗岩纹理图像



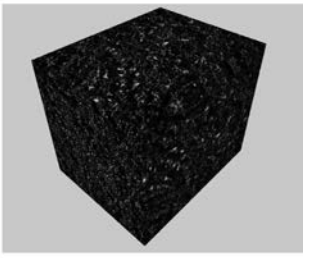
(b) 花岗岩几何造型(长方体)



(c) 颜色纹理



(d) 凹凸纹理



(e) 颜色纹理结合凹凸纹理

图 5.28　颜色纹理与凹凸纹理

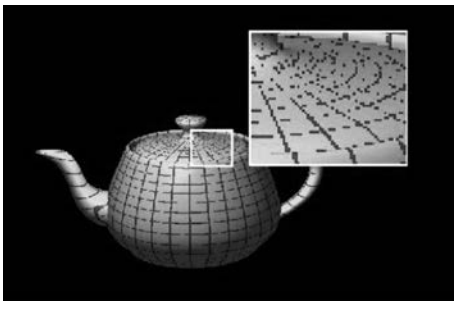


图 5.29　纹理走样

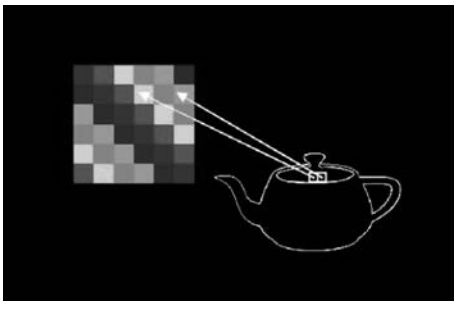


图 5.30　欠采样

空间区域 T，并取区域 T 内的所有纹理像素颜色值的平均作为景物表面区域 A 的平均纹理颜色。最后代入光照明模型，计算出像素 P 应显示的光亮度值。而 Crow 则采用超采样方法，他将屏幕像素 P 的 4 个角点分别映射到纹理空间，得到 4 个纹理像素值，然后将这 4 个纹理像素值取平均作为像素 P 所对应的可见表面区域的纹理颜色，如图 5.31 所示。

然而，上述纹理反走样方法均需频繁计算屏幕像素至景物表面、景物表面至纹理图像的空间映射关系，并计算一个屏幕像素在纹理图像上的映射区域所覆盖的多个纹理像素的平均颜色值，因而难以实时实现。

Mipmap 方法通过预先计算并存贮原始纹理图像的一组多分辨率版本，能显著地节省纹理反走样的计算量，是目前应用广泛的纹理反走样算法之一。Mipmap 算法从原始的纹理图像出发，首先生成一个分辨率为原始图像 1/4 的新的纹理图像版本，新版本中的每一个像素值取原始图像中相对应的 4 个像素颜色值的平均值，如图 5.32(a)所示；然后类似地基于所得到的新纹理图像版本生成一个更低分辨率的、尺寸更小的纹理图像版本；这一过程一直持续到最后生成的纹理图像仅包含一个像素为止，从而得到一个由不同分辨率图像构

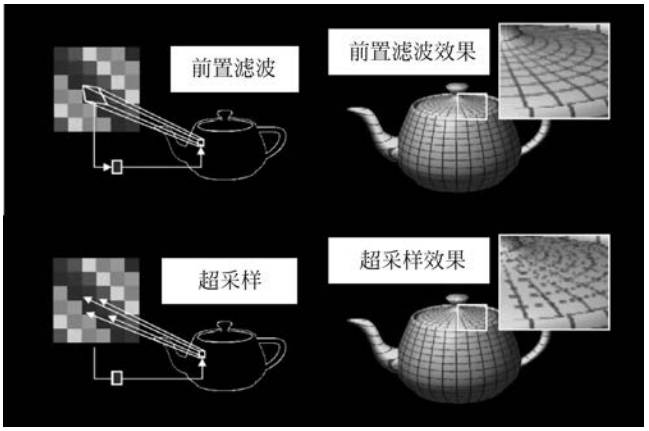
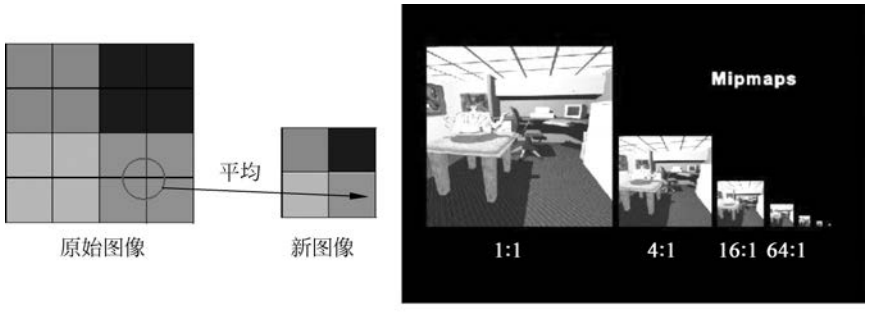


图 5.31　前置滤波与超采样纹理反走样

成的纹理图像序列，各级图像与原始图像的压缩率分别为 1∶1、4∶1、16∶1、64∶1、256∶1 等，如图 5.32(b)所示。



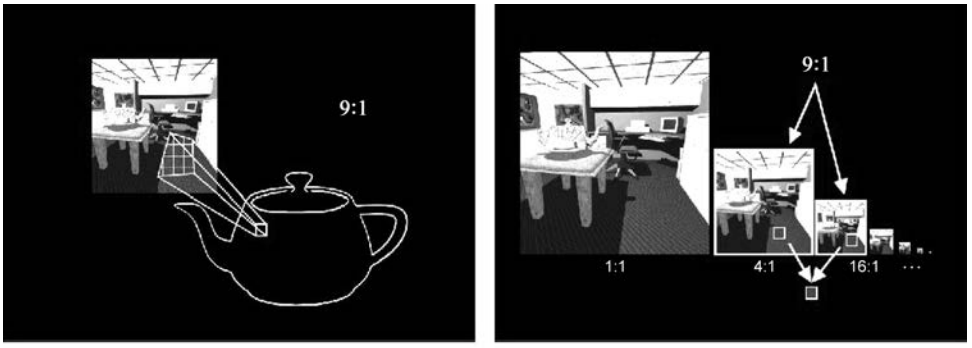
(a) 新图像中像素值的计算　　(b) Mipmap 图像示例

图 5.32　Mipmap 算法中不同精度的纹理图像

在纹理映射阶段，屏幕上的每一像素内的可见表面区域被映射到原始纹理图像上的一块区域。Mipmap 算法估计该区域所覆盖的原始纹理图像中像素的个数，并以此作为选取适当分辨率的纹理图像版本的一种测度。如在图 5.33(a)中，屏幕上当前像素所对应的可见表面区域映射到原始纹理图像上大约覆盖了 9 像素。因此，对于当前屏幕像素，其理想的纹理映射图像的分辨率应为原始纹理图像分辨率的 1/9。查找后发现 Mipmap 算法的数据结构中并没有提供这一分辨率的纹理图像压缩版本。

为计算所需映射的纹理颜色值，Mipmap 算法从预先构造的纹理图像序列中找出其压缩率最接近当前纹理像素与屏幕像素比率的两个纹理图像，算法随即在相邻分辨率的两个纹理图像上查取(或计算)当前屏幕像素映射点的纹理颜色值，并根据两个纹理图像对原始图像的压缩率在所得到的两个纹理颜色值间取加权平均值，作为当前屏幕像素可见表面区域的颜色值，如图 5.33(b)所示。

图 5.34 的比较说明了 Mipmap 纹理反走样算法的效果，其中(a)～(d)4 幅图像均采用 512×324 的分辨率绘制。(a)在绘制时未进行纹理反走样处理；(b)在绘制时采用了 Mipmap 反走样技术，可以看出图形质量比(a)要好；(c)在绘制时采用了一个像素取 9 个采



(a) 屏幕当前像素区域映射为9个纹理像素　　(b) 采用Mipmap方法进行纹理映射

图 5.33　Mipmap 算法中颜色值的计算

样点的超采样算法进行反走样处理；而(d)在 9∶1 超采样的基础上进一步采取 Mipmap 反走样技术进行绘制，可以看出，(d)是所有 4 帧图像中质量最好的。

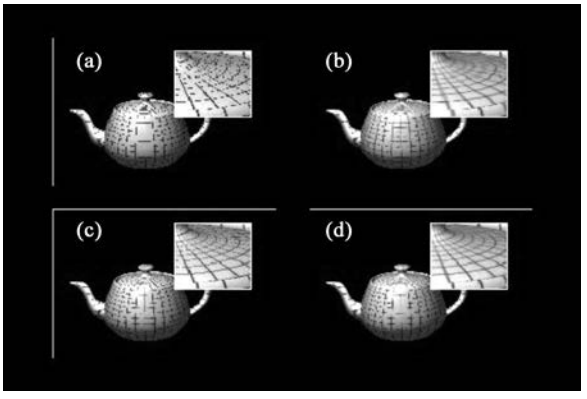


图 5.34　Mipmap 反走样效果比较

5.4.4　纹理映射实例程序

纹理映射技术在 20 世纪 90 年代得到了快速发展，现在即便是普通 PC 机所配备的显卡也能够支持纹理映射功能。利用 OpenGL 实现纹理映射一般包括 3 个主要步骤。

(1) 生成纹理数据：纹理数据一般以原始 RGB 图像数据的形式存储于计算机内存，它既可以是从硬盘读入的数字图像(如 BMP、PNG、TGA、TIFF 等格式的图像)，也可以是由前续绘制过程生成的纹理。

(2) 将纹理数据载入纹理内存：在将纹理数据载入纹理内存之前，首先需要进行一些相关设置，以便 OpenGL 知道如何处理纹理数据。以下伪代码给出了将纹理数据载入纹理内存的一个基本框架：

```
void loadTextures ()
{
    glBindTexture (…, id); /* 将纹理数据载入纹理内存时首先要调用的函数是 glBindTexture()。该函数告诉 OpenGL 当前的工作纹理是编号为 id 的纹理。一个纹理的编号是一个无符号整数，供程序员存取该纹理时使用。例如，调用 glBindTexture(GL_TEXTURE_2D, 2)将使编号为 2 的纹理成为工作纹理 */
```

```
    glPixelStorei (…); /* 该函数告诉 OpenGL 即将载入的纹理数据是如何排列的。例如,调用
glPixelStorei (GL_UNPACK_ALIGNMENT,1)时,纹理数据按字节顺序排列,即在每一像素中,红、绿、蓝的
数值各占一字节 */
    glTexParameteri (…); /* 该函数设置与纹理映射方式(如在某一方向上是否重复)相关的参
数,有关这些参数的意义已超出本书的范围 */
    glTexEnvf (…); /* 该函数设置纹理环境变量,它告诉 OpenGL 纹理数据将如何作用于物体几
何表面。例如,调用 glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE)将纹理环境参数
的符号名设置为 GL_TEXTURE_ENV_MODE,并将相应的纹理函数的符号名设置为 GL_MODULATE,即允许将
诸如光照等效果应用于纹理。假如不想让光照等效果影响纹理数据,而只想让纹理原样呈现在物体
几何表面上,可将纹理函数的符号名设置为 GL_DECAL */
    glTexImage2D (…); /* 该函数用于指定二维纹理图像数据,并把纹理数据载入纹理内存。例
如,调用 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight, 0, GL_RGB, GL_
UNSIGNED_BYTE, imageData)把由红、绿、蓝三分量组成的、宽度和高度分别为 imageWidth 和
imageHeight 的二维纹理图像数据 imageData 载入纹理内存 */
    /* 重复上述操作以载入其他纹理 */
    …
}
```

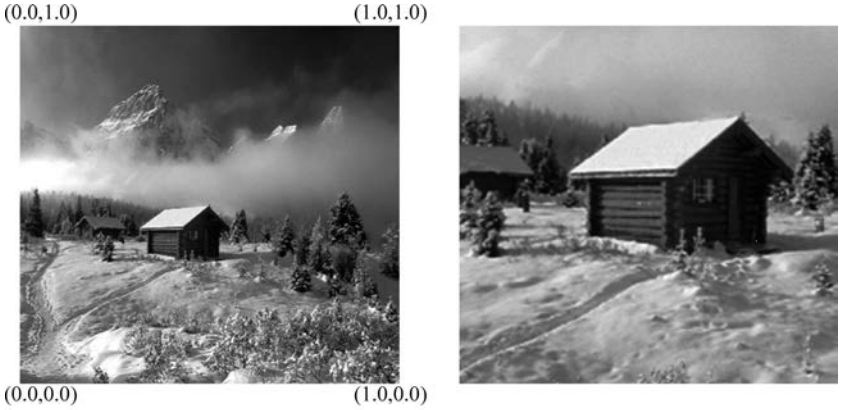
(3) 将纹理数据映射到物体表面：一旦纹理数据载入纹理内存,即可将其映射到物体表面。以下伪代码给出了将纹理数据映射到物体表面的一个基本框架：

```
void renderTextureObjects()
{
    glBindTexture (…,id);        /* 指定编号为 id 的纹理为当前工作纹理 */
    glBegin (…);
        glTexCoord2f (…);        /* 指定顶点的纹理坐标 */
        glVertex3f (…);          /* 指定顶点的位置 */
        /* 重复 glTexCoordf(), glVertex3f()设置其他顶点的纹理坐标和几何位置 */
    glEnd ();
    /* 重复上述操作将纹理数据映射到其他物体表面 */
}
```

从上述绘制框架不难看出,在将纹理映射到物体表面时,首先需要在调用 glBegin()/glEnd()之前(而不能在 glBegin()/glEnd()之间)调用 glBindTexture(),通过纹理编号来绑定纹理,其次需要在指定物体的位置之前指定其相应的纹理坐标(也称 uv 坐标)。在 OpenGL 的纹理坐标系统中,左下角、右下角、右上角和左上角的纹理坐标分别为(0.0,0.0)、(1.0,0.0)、(1.0,1.0)和(0.0,1.0),如图 5.35(a)所示。通过为待绘制的物体表面的每一个顶点指定一个纹理坐标,可决定图像纹理中的哪一块区域将映射到物体表面上,图 5.35(b)是以下代码片断所对应的纹理映射结果。

```
glBegin (GL_QUADS);
    glTexCoord2f (0.2, 0.2); glVertex3f (0.0, 0.0, 0.0);
    glTexCoord2f (0.6, 0.2); glVertex3f (10.0, 0.0, 0.0);
    glTexCoord2f (0.6, 0.6); glVertex3f (10.0, 10.0, 0.0);
    glTexCoord2f (0.2, 0.6); glVertex3f (0.0, 10.0, 0.0);
glEnd ();
```

值得指出的是,首先需要激活纹理映射功能才能获得纹理映射效果,这可在进行 OpenGL 基本设置时通过调用 glEnable(GL_TEXTURE_2D)来实现。有关 OpenGL 实现纹理映射的具体细节,可参考 Miller N 的综述及本书配套资源中的二维纹理映射实例程序。



(a) 纹理坐标系统　　(b) 纹理映射结果

图 5.35　纹理坐标系统及纹理映射结果

5.5　辐射度方法

光线跟踪算法通过跟踪光线在镜面和透明面之间的传播路径很好地模拟了场景中存在的镜面反射和规则透射现象。然而，现实世界中漫反射表面之间同样存在光能传递，例如，相距很近的不同颜色的漫反射表面间会出现颜色辉映现象，如图 5.36 所示。显然，要对场景中众多的漫射表面朝不同方向发出的漫射光线逐一跟踪是十分困难的。

图 5.36　炼钢炉前(真实场景)

5.5.1　辐射度方法简介

注意光是一种辐射能，在一个封闭的环境中，场景中的光能经过表面之间的反射和透射，最终达到平衡状态。场景中各表面的光亮度实际上是场景中光能分布的反映，基于此，Goral 和 Nishita 等分别独立地提出了计算场景中光能分布的辐射度方法。

辐射度为单位时间内从物体单位表面积向外辐射的光能，它既包含物体作为光源自身向外发出的能量，也包含该物体表面接受来自周围场景表面传递给它的能量后，再次反射出去的部分能量。

为求解场景中物体表面的辐射度，需要将场景中每一物体的表面分解为互不重叠的小面片(Patch)$P_i(i=1,2,\cdots,n)$，每一小面片的辐射度 B_i 和漫反射率 ρ_i 均假定为常数。记面片 P_i 的面积为 A_i，其自身拥有的辐射度为 E_i，显然，对于面片 P_i：

$$B_i = E_i + \rho_i H_i \tag{5.19}$$

其中，H_i 为周围场景面片入射到面片 P_i 单位面积上的光能，不难发现，H_i 是周围面片辐射度的函数。设面片 P_j 为一周围场景面片，由辐射度定义，面片 P_j(其面积为 A_j)向外辐射的总光能为 B_jA_j，其中一部分到达面片 P_i。设到达面片 P_i 的光能占面片 P_j 向外辐射

的总光能的比例为 F_{ji}，则从周围环境面片入射到面片 P_i 的总光能为 $\sum_{j=1}^{n} F_{ji}B_jA_j$，入射到面片 P_i 单位面积上的光能 H_i 则为：

$$H_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} F_{ji}B_jA_j/A_i \tag{5.20}$$

F_{ji} 称为面片 P_j 到面片 P_i 的形状因子(form factor)。对于由纯漫射表面组成的封闭环境，F_{ji} 仅取决于面片 P_i、P_j 的面积、朝向以及它们在空间的相对位置。形状因子具有如下交换关系：$A_jF_{ji}=A_iF_{ij}$，因此，式(5.20)可转换为：

$$H_i = \sum_{j=1}^{n} F_{ij}B_j \tag{5.21}$$

代入式(5.19)，得：

$$B_i = E_i + \rho_i \sum_{\substack{j=1 \\ j\neq i}}^{n} F_{ij}B_j \quad (i=1,2,\cdots,n) \tag{5.22}$$

其矩阵形式如下：

$$\begin{pmatrix} 1 & -\rho_1F_{12} & \cdots & -\rho_1F_{1n} \\ -\rho_2F_{21} & 1 & \cdots & -\rho_2F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_nF_{n1} & -\rho_nF_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \tag{5.23}$$

式(5.23)即为理想漫射环境的辐射度系统方程。求解该系统方程即可得到场景中每一面片的光能辐射度。

不难发现，计算和确定场景面片之间的形状因子是求解系统辐射度方程的关键。如前所述，对于纯漫射环境，形状因子 F_{ji} 是一个取决于面片面积、朝向及其空间相对位置的纯几何量，如图 5.37 所示。由于它的解析表达式较为复杂，常采用半立方体方法、光线采样方法计算。

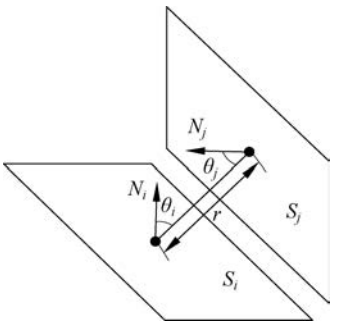


图 5.37　形状因子计算示意图

理论上说，辐射度系统方程可采用任何一种线性方程组的求解算法来求解。但是，在实际应用中，为满足每一小面片的辐射度和漫反射率均为常数的假设，常常需要对场景中的物体表面进行细致的分割，对于一个复杂场景，其辐射度系统方程中矩阵的规模可能非常庞大。虽然可采用高斯-赛德尔等迭代方法求解辐射度系统方程，但其系数矩阵的存储量和计算量使得它们在求解复杂场景的辐射度时无能为力。因此，稍后提出的许多光能辐射度改进算法，例如逐步求精辐射度算法和小波辐射度算法中，均巧妙地避开了方程组的直接求解过程。

辐射度算法是继光线跟踪算法之后真实感图形绘制的又一重要算法。与光线跟踪算法相比，辐射度算法具有两个明显的优点：其一，对于景物表面漫射光的模拟更精确，且对于给定场景，其光能分布只需计算一次，这使得它更适用于需要实时绘制的场合，如场景漫游、三维游戏等；其二，很容易处理面光源，因此能自然地生成软影。然而，经典辐射度算法只

适用于理想漫反射场景，这使它很难模拟镜面反射和透射现象，而这正是光线跟踪算法的长处。只有将两者有机地结合起来才有利于实现对现实世界更精确、更真实的模拟。图 5.38 是采用辐射度算法生成的真实感图形。

图 5.38　采用辐射度算法生成的真实感图形

5.5.2　辐射度方法实例程序

辐射度方法起源于热能工程领域，在计算机图形学领域得到了进一步发展。与光线跟踪算法类似，目前辐射度方法已经成熟，其中主要算法可分为基于矩阵的辐射度算法、逐步求精辐射度算法和小波辐射度算法。Willmott 与 Heckbert 在一份技术报告中，针对这 3 类算法在处理诸如奇异性、遮挡、镜面反射以及场景复杂度等方面的能力进行了较为全面的比较，并发布了用于该技术报告的辐射度系统的源代码。本书下载资源提供了一个基于命令行的辐射度绘制程序以及一个基于 OpenGL 的实时辐射度场景观察程序的执行文件供读者学习、参考。

5.6　实时绘制技术

实时绘制指的是利用计算机快速生成三维场景的真实感图形。它与图形硬件的发展和人们对人机交互的需求密不可分。

图像绘制的速度采用帧频来衡量。帧频的单位为帧/秒。当帧频为 1 时，用户能明显感觉到屏幕上画面之间的跳跃；当帧频为 6 时，用户能产生交互的感觉；当帧频为 15 时，用户能流畅地进行人机交互；而当帧频达到 72 或以上时，显示速度上的差异人眼已经难以区分了。

实时绘制技术并不仅仅关注屏幕上画面生成的速度，图像的真实感也是实时绘制技术所追求的重要目标，实时绘制的关键是如何充分发挥图形硬件和图形算法各自的长处，在绘制速度和图形质量之间取得某种平衡。在介绍当前各种实时绘制技术之前，下面首先简单介绍三维图形绘制流水线的基本结构。

5.6.1　图形绘制流水线与图形 API

三维图形绘制由一系列顺序的过程组成，这些过程已可部分或全部采用硬件实现，它们连接在一起称为图形绘制流水线。一般情况下，三维图形绘制流水线分为以下 5 个阶段。

(1) 场景描述。具体包括三维场景的几何描述(造型)、物体及相机运动描述、物体可见性检查等。

(2) 取景变换。将三维场景中的物体从景物空间变换到屏幕空间。具体包括物体的平移/旋转/比例变换、世界坐标系到视点坐标系的变换、透视变换、背面剔除(也可在后面的屏幕空间完成)、光源设置、视域裁剪、物体数据变换为屏幕空间的基本体素(如点、线、多边形)等。

(3) 扫描处理。包括背面剔除(背面剔除也可在视点坐标系完成)、斜率(Delta 增量)计算、三角形扫描线转换等。

(4) 绘制/光栅化。对屏幕基本体素进行光栅化,并将结果存入帧缓存。此阶段主要执行顶点级和像素级操作,如光亮度计算、纹理表查找、深度测试、Alpha 透明度测试、反走样处理等。

(5) 屏幕显示。将帧缓存中的内容显示在监视器的屏幕上。

除第一阶段外,三维图形绘制流水线的其余 4 个主要阶段通常都由图形应用程序接口(API)直接管理,如 OpenGL、微软的 Direct3D、Pixar 的 RenderMan 等,由 API 驱动硬件执行相应的操作。有些高性能的图形 API,如 SGI OpenGL Performer,甚至能支持图形绘制流程的每一步。事实上,图形 API 的作用是将图形硬件执行的功能抽象为应用程序,借助图形 API,应用程序开发人员只需直接调用 API 编写代码,应用程序便能运行于支持该 API 的所有硬件平台上,如图 5.39 所示。第 9 章中将简单介绍几个常用的图形 API。

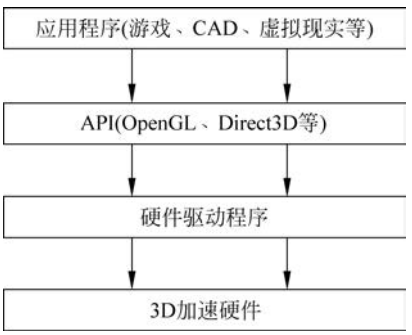


图 5.39　3D API

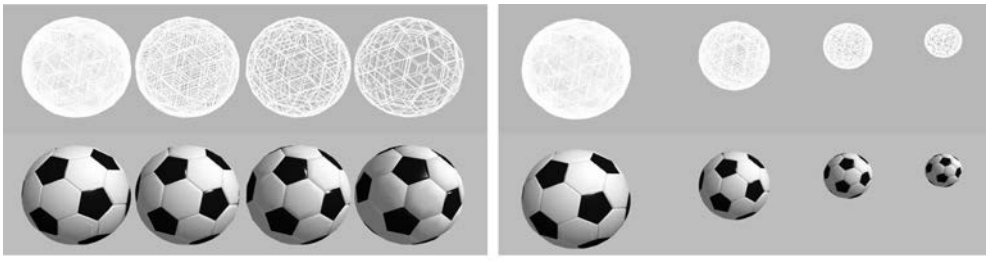
5.6.2　常用的实时绘制技术

实时绘制技术是一种限时计算技术,即要求计算机在尽可能保持画面真实感的前提下,在给定的时间内完成绘制任务。由于实际应用所涉及的场景中景物面片的数量已远远超出了目前图形硬件的处理能力,因此需要在不牺牲或少量牺牲场景视觉细节的前提下,尽可能减少参与当前帧画面绘制的场景面片的数量。下面对目前实时绘制中常用的几种技术进行简单介绍。

1. 细节层次技术

细节层次(LOD)模型是以不同精度刻画物体不同程度细节的一组模型。细节层次技术的基本思想是根据物体在画面上的视觉重要性选取适当的细节层次绘制该物体。评价物体在当前画面上的视觉重要性的要素很多,包括物体在画面上投影区域的大小、物体是否是

用户关注的中心、物体是否处于高速运动状态等。在图 5.40(a)中，按从左到右的顺序显示了足球从高精度到低精度的 4 个细节层次表示的绘制结果，显然，人们很容易观察出不同细节层次之间的差别；而如果将上述 4 个细节层次表示的足球依精度的高低对应离视点的距离进行显示，将很难察觉其中的差别，如图 5.40(b)所示。上述足球的 4 个细节层次模型中分别包含 4348、2425、1326 和 705 个三角形。



(a) 足球从低到高4个不同精度层次表示　　(b) 当足球从近到远时取不同精度层次表示进行绘制

图 5.40　足球的 LOD 表示

自动生成物体的细节层次表示是细节层次技术的重要研究内容。一般情况下，物体的细节层次表示可由网格简化算法来完成。网格简化算法的思想是基于一定的误差度量准则(如点面之间的距离、多边形之间的夹角、曲面的曲率等)，通过一些简单的几何元素删除操作(如顶点/边/三角形删除、边/三角形折叠、平面/顶点合并等)删除复杂几何中的一些相对于物体外观而言次要的几何元素，或重新对物体进行较低精度的采样，以生成物体的较低精度的细节层次表示。网格简化算法有的能保持物体的拓扑结构不变，有的则不能。图 5.41 是飞机模型的 3 个细节层次，分别包含 4629、2002、483 个三角形。从图中可以看出，虽然最低精度的模型中的三角形数目仅为最高精度的模型中的三角形数目的近 1/10，但仍较好地保持了原始物体的外形结构，显然，当飞机离视点较远时，采用该细节层次既能显著节省绘制时间，又能保持较好的视觉真实性。

图 5.41　飞机模型的 3 个细节层次

2. 网格优化压缩技术

从图形绘制流水线不难看出，场景绘制的速度受到场景中三角形数目的制约，在绘制一个三角形时，必须将其全部 3 个顶点的信息传送到图形硬件。注意到三角形每一个顶点均为多个三角形共享，为避免同一顶点的信息重复传送，大多数图形 API 均采用三角形带(Triangle Strips)和三角形扇(Triangle Fans)等复合三角形结构进行传输，如图 5.42 所示，以充分利用图形硬件的有限带宽，提高绘制效率。

在三角形带结构中，相邻的两个三角形之间共享一条公共边，整个三角形带被表达为一

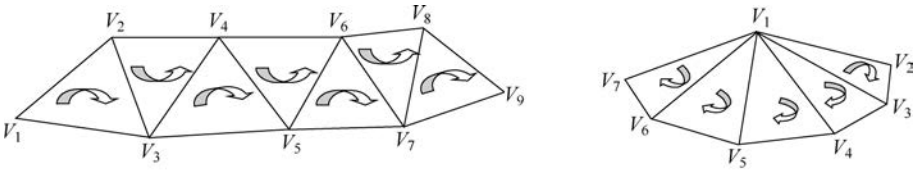


图 5.42　三角形带与三角形扇

个顶点序列 $V_1V_2V_3\cdots V_n$ ($n\geqslant 3$)，共定义了($n-2$)个三角形，其中第 i 个三角形由顶点 $V_iV_{i+1}V_{i+2}$ 定义；而在三角形扇中，一系列三角形共享同一个起始顶点，整个三角形扇同样可表示为一个顶点序列 $V_1V_2V_3\cdots V_n$ ($n\geqslant 3$)，它包含($n-2$)个三角形，其中第 i 个三角形由顶点 $V_1V_{i+1}V_{i+2}$ 定义。显然，采用类似于三角形带或三角形扇这样的复合结构，将把处理与传输 m 个三角形的代价从 $3m$ 个顶点降到($m+2$)个顶点。

已有不少算法能有效地将一个由三角形网格表示的物体转换成由三角形带、三角形扇或其混合表示的物体。通常，一个复杂的物体往往需要表示为很多三角形带、三角形扇或三角形本身的组合。例如，图 5.43 中的模型被离散成 1770 个独立的三角形，通常情况下需要由 1770×3＝5310 个顶点描述。如果利用三角形带和三角形扇的复合结构，则可用 2176 个顶点将其表示为 203 个图形对象(包括三角形带、三角形扇、三角形)。这使得图形硬件需要处理的图形对象数目降低到原数目的 1/9，而需要处理与传输的顶点个数降低到原数目的 2/5。

图 5.43　由三角形带、三角形扇、三角形混合表示的物体

3. 遮挡剔除技术

遮挡剔除指的是在对场景进行取景变换之前剔除掉场景中对于当前视点被遮挡的某些物体的整体或局部，从而加速场景的绘制。如图 5.44(a)所示的模型，显然在绘制该画面时，只需将表示该模型外壳的几何信息传送到图形硬件即可，而其内部的许多零部件，如图 5.44(b)所示，则可以事先被剔除掉，无须传送给图形硬件。

图 5.44　遮挡剔除示例

为实现遮挡剔除，首先需要对物体进行可见性检查。物体的可见性检查一般在场景数据组织阶段由计算机 CPU 计算完成，不涉及图形硬件。例如，可以事先对场景所在空间进行网格剖分，针对每个网格计算当视点位置落在该网格内部时沿每个观察方向场景的可能

可见面集(Potentially Visible Set,PVS)并予以保存；在对场景进行绘制时，当视点位于某一网格内部时，只需将事先计算好的该网格内沿当前观察方向可能可见的面片传送到图形硬件即可。

4. 基于图像的绘制技术

基于图像的绘制(Image-Based Rendering,IBR)是一种新的绘制方法。不同于传统的基于几何模型的绘制方法，基于图像的绘制方法，以待绘制的场景的一系列二维图像作为输入，通过将它们重新整合来生成在新的视点和新的视线方向上的场景画面。

基于图像的绘制技术的前身是环境映射(Environment Map)技术。环境映射提供了在一给定视点处沿所有视线方向观察场景的完整记录。记录环境映射的表面可以是包围该视点的立方体表面、球面或圆柱面。

QuickTime VR 系统是一个基于图像的场景漫游系统，它主要基于圆柱形环境映射。为实现对整个场景的漫游，需要预先在场景中设置一系列采样点作为固定的视点，在每个采样点处建立环境映射，并在各采样点(热点)之间建立链接关系，用户通过单击这些热点即可从不同的视点位置观察整个场景。建立圆柱环境映射的方法是采用全景相机拍摄得到场景的全景图，如图 5.45 所示，或者采用普通相机先从多视角拍摄多幅图像，再利用图像处理工具(如 Adobe PhotoShop CS3)将其拼合成全景图，并将其映射到圆柱表面上。QuickTime VR 新增加的 Cubic VR 功能允许用户实现全方位漫游。

图 5.45　全景相机拍摄的全景图

基于图像的绘制技术由于采用现实世界真实场景的照片(或计算机绘制生成的图像)作为输入，因此无须使用复杂的场景几何造型即可实现对场景的浏览、漫游，其绘制计算量与所需绘制的场景的几何复杂度无关，仅与所需绘制的画面分辨率有关，从而使对高度复杂的场景进行实时绘制成为可能。但另一方面，该项技术仅适用于静态场景，用户无法与场景中的景物进行实时交互；而且，其绘制质量在很大程度上取决于原始图像的采样数目和相应的插值方法。当输入的图像采样过于稀疏时，基于图像的绘制算法将不可避免地产生错误。至于需要从哪些空间方位拍摄多少幅照片，才能构成对场景的完整采样，至今仍然是一个未解决的问题。

基于图像的绘制技术仍在不断的发展和完善之中。其代表性工作包括：基于视图插值的方法、基于全景函数(Plenoptic Function)的方法、基于分层表示的方法等。

5.6.3　实时光线跟踪

谈到光线跟踪，人们往往会联想到高度真实感图形、巨大的计算量，而较难将其与实时

图形联系在一起。然而，随着计算机计算速度的快速提升、图形硬件处理能力的高速发展以及计算机网络技术的成熟，实时光线跟踪已经成为可能。与传统的基于图形绘制流水线的光栅化算法相比，光线跟踪在灵活性、连贯性、并行性、遮挡剔除、着色效率与质量等方面均有较大的优势。因此，在可预见的将来，三维游戏、虚拟现实、产品原型评价、电子商务等既需要有较高的视觉真实感，又对图形实时绘制有较高要求的应用领域将从中获益。

为使光线跟踪达到实时，除了对传统的加速技术（光线与体素快速求交方法、光线穿越策略、空间与层次加速结构、采样策略等）继续改进外，更多地需要结合计算机图形技术的新发展，探索场景数据的高效存储与管理机制、并行计算策略、基于新图形硬件（如 GPU）的快速计算方法，乃至开发全新的光线跟踪图形硬件。

OpenRT 是一个实时光线跟踪 API，源自德国 Saarland 大学的实时光线跟踪项目。作为一个实时光线跟踪引擎，OpenRT 采用与 OpenGL 类似的语法，具有类似于离线光线跟踪器（如 RenderMan、POV-Ray）的绘制能力，支持着色（Shader）程序，具有并行绘制功能，且支持动态场景。图 5.46 所示是 OpenRT 应用于工业设计的一个示例，其中（a）为 Mercedes C-Class 车模型，包含 320 000 个 Bézier 面片，采用 OpenRT 直接绘制（未进行三角化）；（b）为将该模型置入一个由 200 000 个三角形及一个高动态范围环境映照所构成的场景中，在绘制时采用了光线跟踪着色器（如玻璃、汽车油漆）；（c）的车内饰在绘制时采用了一个支持双向纹理功能（Bidirectional Texture Function）的着色器，其中的样本取自真实环境；在（d）中，所有这些效果无缝地结合在一起了。

(a) Mercedes C-Class车模型　(b) 将车模型置入一个环境映照图中

(c) 车内饰采用支持双向纹理功能的Shader绘制　(d) 上述效果的集成

图 5.46　OpenRT 应用示例

5.7 非真实感图形绘制技术

在真实感图形朝着更高的真实感、更贴近现实世界的方向发展的同时，人们发现不同艺术形式、不同表现风格的非真实感图形往往更有利于传递特定的信息，表达人类的思想和情感，反映个性化的审美追求。事实上，照相机只是近代出现的用于捕获现实世界影像的工具。在照相机发明之前的长达几千年的时间里，人们往往借助素描、绘画等艺术形式来刻画丰富多彩的现实世界。

自 20 世纪 90 年代中期开始，非真实感图形绘制(Non-Photorealistic Rendering，NPR)逐渐成为计算机图形的研究热点之一。非真实感图形绘制又称风格绘制(Stylistic Rendering)，顾名思义，它指的是利用计算机生成不具有照片般真实感，而具有手绘风格的图形的技术。

非真实感图形绘制的主要特点如下。

(1) 非真实感图形应该表现出艺术特质。其一，对所需绘制的场景或物体加以抽象，去除不必要表现的冗余信息；其二，以艺术家的眼光，而不仅仅是从编程者的角度来表现现实世界。

(2) 非真实感图形应该像人类艺术作品一样，具有不同的风格、品位，也包含类似的缺陷或不完美之处。

(3) 非真实感图形是真实感图形的有效补充。它既不应该被理解为真实感图形的对立面，也不应该仅仅局限于目前有限的几类绘制风格。

非真实感图形技术主要分为素描(Sketching)、卡通绘制(Cartoon Rendering)、美术绘制(Painterly Rendering)等。

5.7.1 素描

素描的创作过程看起来似乎漫不经心，但对大多数人而言，素描仍是一种富有表现力的信息交流方式，它已渗透到现实生活的许多方面：朋友间聊天兴致所至时会在纸上随意勾画几笔；专业美术工作者或艺术家进行时尚设计时会将他们的灵感用素描画记录下来；在工程、医学、建筑等应用中，技术人员也常采用技术插图来刻画技术细节，如图 5.47 所示。因此，如何利用计算机的帮助，快速完成素描引起了不少研究者的兴趣。

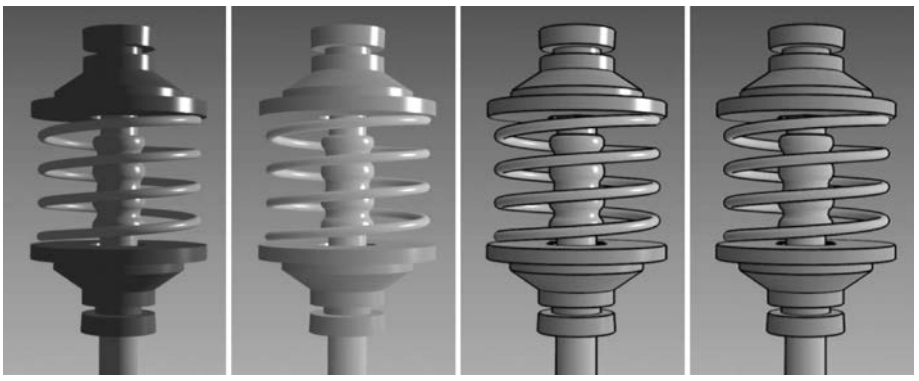


图 5.47 技术插图

体现素描风格的绘制算法大体上可分为轮廓绘制(Silhouette Rendering)算法和画影线(Hatching)两类。前一类在绘制风格上主要强调勾画物体的轮廓等重要特征,如图 5.48 所示;而后者则试图模仿铅笔或墨笔的笔触,并表达物体表面的明暗、阴影等光照效果,如图 5.49 所示。

图 5.48　轮廓绘制

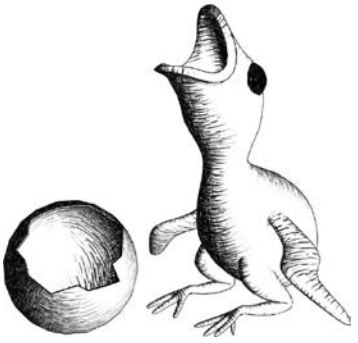


图 5.49　画影线

5.7.2　卡通绘制

卡通目前已成为娱乐、教育中的一类重要信息传递方式。与纯文字或照片相比,卡通的简单性(对现实世界的简化)和通用性(仅使用单纯性的图形描述,从而消除了语言的障碍)使其能以非常有限的空间传递大量的信息。事实上,目前计算机已经在传统的卡通动画产业中扮演着重要角色,卡通风格绘制在强调物体轮廓的同时,利用明暗处理模型(通常为 Flat 明暗处理),并结合物体材料颜色、阴影颜色等刻画物体外形、光效等。随着图形加速卡的快速发展,实时卡通绘制已经成为可能。通过图形硬件加速功能,结合轮廓提取、纹理映射等技术,可直接从三维几何模型实时生成卡通风格图形,如图 5.50 所示。

5.7.3　美术绘制

对传统的绘画风格,如油画、水彩画、蜡笔画等的模拟是非真实感图形绘制的重要研究内容之一。传统的绘画作品往往由大量独立的笔画(Strokes)构成。正是生成这些笔画的技术和所采用的工具的不同决定了绘画作品的风格。事实上,区分一幅绘画作品属于哪种艺术流派的重要准则之一就是观察其中笔画的特性。基于上述思想,Haeberli 在生成图形时,采用了短笔刷(Brush Strokes)取代逐个像素绘制的方法,创造出了类似印象派绘画的效果。这一思想后来被用于生成其他一系列美术风格的图形,如图 5.51 所示。

图 5.50　卡通绘制

图 5.51　美术绘制——静物

按照产生图形的不同方式，美术绘制方法主要分为基于物理的方法和自动绘画方法两类。在基于物理的方法中，计算机根据用户指定的笔画，基于仿真传统媒介的物理模型，得到特定的绘画效果。例如，Curtis 等在建立水、颜料、纸张之间相互作用的物理模型的基础上，生成了较好的水彩画效果。而自动绘画方法则首先由用户提供三维几何模型（或二维图像）和绘画参数，计算机自动生成所有的笔画并最终完成非真实感图形的绘制过程。

习题

1. 局部光照明模型与整体光照明有何不同？

2. 比较 Gouraud 明暗处理和 Phong 明暗处理的差异。

3. 以球面（采用三角形网格表示）为例，结合图 5.13 所示的方法计算每个顶点的法向量，并将计算结果与精确的球面法向进行比较。

4. 试推导 Gouraud 明暗处理中多边形内部点的光亮度值计算的双线性插值公式。

5. 以球面（采用三角形网格表示）为例，结合扫描线消隐算法，分别采用表 5.1 描述的方法对球面着色，并比较着色效果。

表 5.1 绘制的明暗处理方式与光照明模型

方法编号	明暗处理方式	光照明模型
1	Flat 明暗处理	泛光模型
2		Lambert 漫反射模型
3		Phong 模型
4	Gouraud 明暗处理	泛光模型
5		Lambert 漫反射模型
6		Phong 模型
7	Phong 明暗处理	泛光模型
8		Lambert 漫反射模型
9		Phong 模型

6. 简述光线跟踪算法的原理。

7. 如图 5.52 所示，P 为入射光线 $\boldsymbol{L}$ 和物体的交点，$\boldsymbol{N}$ 为点 P 处的物体表面法向，$\boldsymbol{R}_r$ 为镜面反射光线的方向，θ_i 为 $\boldsymbol{L}$ 与 $\boldsymbol{N}$ 的夹角，θ_r 为 $\boldsymbol{R}_r$ 与 $\boldsymbol{N}$ 的夹角。由光线反射定律知道，$\theta_i=\theta_r$。不妨假设 $\boldsymbol{L}$、$\boldsymbol{N}$ 均为单位矢量，试求出光线反射方向 $\boldsymbol{R}_r$（假设也为单位矢量）。

8. Snell 定律表明：位于折射率为 η_1 的介质 1 中、与表面法向 $\boldsymbol{N}$ 的夹角为 θ_1 的入射光线 $\boldsymbol{L}$，在进入折射率为 η_2 的介质 2 后，将产生折射，其折射方向 $\boldsymbol{R}_t$ 与 $\boldsymbol{N}$ 的夹角为 θ_2，如图 5.53 所示。假设已知 $\boldsymbol{L}$、$\boldsymbol{N}$、η_1、η_2，试基于 Snell 定律计算光线折射方向 $\boldsymbol{R}_t$。

9. 光线跟踪算法中一般采用直线的参数方程来表示光线。假设光线 $\boldsymbol{R}$ 的起始点为 $\boldsymbol{P}$，方向为 $\boldsymbol{D}$（不妨假设 $\boldsymbol{D}$ 为单位矢量），则光线的参数方程可表示为 $\boldsymbol{R}(t)=\boldsymbol{P}+t\boldsymbol{D}$。假设 V_1、V_2、V_3 是三角形的顶点，试给出光线 $\boldsymbol{R}$ 与 $\Delta V_1V_2V_3$ 的求交算法（提示：算法可分两步：①计算光线 $\boldsymbol{R}$ 与 $\Delta V_1V_2V_3$ 所在平面的交点 P_i；②判别交点 P_i 是否在 $\Delta V_1V_2V_3$ 内部或边界上）。

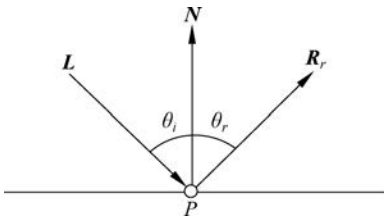


图 5.52　光线的入射与反射

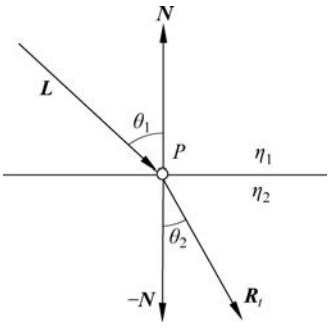


图 5.53　光线的入射与折射

10. 如图 5.54 所示，$\boldsymbol{R}$ 是从点 P 出发、方向为 $\boldsymbol{D}$(单位矢量)的光线。球的中心为 O，半径为 r。$OE \perp PE$，P_i 是光线与球的交点(如果存在交点)，$|PE|=a$，$|OE|=b$，$|PO|=c$，$|EP_i|=d$。

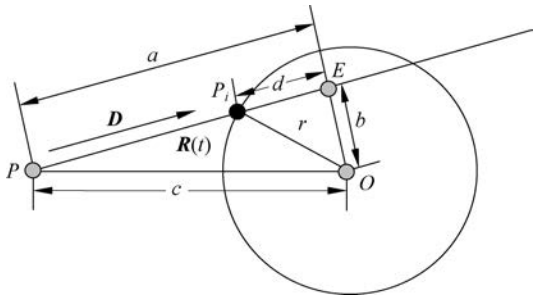


图 5.54　光线与球面求交

记 $d\_sqr=r^2-(|PO|^2-a^2)$，试：

(1) 证明当 $d\_sqr<0$ 时，光线 $\boldsymbol{R}$ 与球 O 无交点。

(2) 求出当 $d\_sqr\geqslant 0$ 时，光线 $\boldsymbol{R}$ 与球 O 的交点 P_i。

11. 下载、安装、学习 POV-Ray。

12. 学习本书下载资源中的光线跟踪绘制程序，结合本章 5.3.2 节光线跟踪算法的伪代码，实现镜面及透明效果。

13. 什么是纹理映射？颜色纹理与几何纹理的区别是什么？

14. 简述 Mipmap 纹理反走样方法的原理。

15. 学习本书下载资源的二维纹理映射实例程序，体会利用 OpenGL 实现纹理映射的主要步骤。

16. 简述辐射度方法的原理，并推导理想漫射环境的辐射度系统方程。

17. 用示意图画出图形绘制流水线的结构。

18. 常用的实时绘制技术有哪些？它们分别从哪些角度实现图形的加速绘制？

19. 简述非真实感图形的主要特点。

20. 就“非真实感图形不应被理解为真实感图形的对立面，也不应该仅仅局限于目前有限的几类绘制风格”谈谈你的看法。