计算机图形学课程读书报告

北京理工大学　计算机学院　07112005班　董若扬　1120202944

前言

这本书的内容占据了整个课程的大部分时间，前期学习理论知识消化吸收十分困难，在得知最后要写一份读书报告后，我就对照着书本内容按自己的学习思路整理，发现一步步地将理论知识掌握。最初觉得怎么迟迟不讲OpenGL，因为其他专业选修课，上来就是讲Android怎么开发，数据库怎么操作，不先进行理论知识的铺垫。但是当学完这些预备知识后，根据OpenGL红宝书和网络上丰富的学习资源，很快就熟练了OpenGL的代码，也能对每一步操作知其然更知其所以然。跟着网上几个教程做了几个demo，再把各种效果组合起来，再从整体进行顶层设计重构，最终顺利完成实践，而一切顺利的关键在于完成了理论知识的积累。不过鉴于课程学习时长有限，只有限地学习了乐学上所涉及的内容模块，两本书中还有大量丰富的图形学秘诀还有待将来进一步探索。

导论

导论让我了解了图形学的背景和历史。导论介绍了计算机图形学是一门研究如何用计算机表达、生成、处理和显示图形的学科。阅读导论一章，我们可以了解到计算机图形学广泛应用于计算机辅助设计、地理信息系统、计算机游戏、计算机动画、虚拟现实等领域。计算机图形是由计算机表示、生成、处理和显示的对象。就范围而言，计算机图形学包括客观世界中存在的所有物体甚至意识形态，如山、水、人等；就内容而言，计算机图形学不仅是物体的形状，而且是物体的材料、运动和其他特性。因此，计算机图形学就是将物体的坐标、纹理等属性存储在计算机中。存储在计算机中的一切都可以概括为“虚拟环境”。它通过发达的互联网与其他计算机相连，因此不同地方的人们可以看到“彼此”并相互交流。因此，虚拟现实已经成为一种共享的社会现实。

数学基础

计算机图形学中需要大量的数学知识储备，其中向量，矩阵等线性代数知识已经是老生常谈的内容了，外行人也知道绝对和它们脱不了关系。此章节中最硬核的内容为欧拉角和四元数。欧拉角这块内容我查阅了YouTube一个历史悠久的视频详细介绍了万向锁的问题，并由此介绍了四元数。关于四元数的内容我又在著名数学博主3Blue1Brown的长达40分钟的科普视频里详尽地学习了。最终参阅了几篇数学爱好者发表的文章，整理了自己的见解。在书写此章节最困难的不是表述自己的想法，而是如何书写漂亮整洁的公式。

向量

点与向量的关系

两点间距离：

$$|p_1 - p_2| = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2}$$

向量表示为两点间的差值：

$$v = p_2 - p_1$$

点可以表示为点与向量的和：

$$p_2 = p_1 + v$$

矢量加法：

$$v_1 = (x_1, y_1, z_1)$$
$$v_2 = (x_2, y_2, z_2)$$
$$v_1 + v_2 = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$

矢量伸缩：

$$\lambda v = (\lambda x, \lambda y, \lambda z)$$

若$|\lambda| > 0$，则新向量与原向量方向相同，否则相反。新向量的长度为：

$$|\lambda v| = |\lambda||v|$$

矢量的模：

空间的基由三个单位向量构成：

$$e_1 = (1, 0, 0)$$
$$e_2 = (0, 1, 0)$$
$$e_3 = (0, 0, 1)$$

任何向量(x, y, z)可以表示为这三个向量的线性组合：

$$(x, y, z) = xe_1 + ye_2 + ze_3$$

这三个向量通常被称为i, j, k,向量的模为：

$$|v| = \sqrt{x^2 + y^2 + z^2}$$

$|norm(v)| = 1$。

内积：

两个向量v_1，v_2的点乘表示为$v_1 \cdot v_2$

$$v_1 \cdot v_2 = x_1x_2 + y_1y_2 + z_1z_2$$

两个向量的夹角θ的余弦值有以下结论:

$$cos(\theta) = \frac{v_1 \cdot v_2}{|v_1| \times |v_2|}$$

叉乘：

两个向量的叉乘构成一个新的向量，记作$v_1 \times v_2$：

$$v_1 \times v_2 = (y_1z_2 - y_2z_1, x_2z_1 - x_1z_2, x_1y_2 - x_2y_1)$$

叉乘的性质：

$$|v_1 \times v_2| = |v_1||v_2|sin(\theta)$$

且$v_1 \times v_2$垂直于v_1和v_2构成的平面。

矩阵

平移矩阵

$$T(a,b,c)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\a&b&c&1\end{bmatrix}$$

伸缩矩阵

$$S(a,b,c)=\begin{bmatrix}a&0&0&0\\0&b&0&0\\0&0&c&0\\0&0&0&1\end{bmatrix}$$

旋转矩阵

绕Z轴

$$R(\theta)=\begin{bmatrix}cos\theta&sin\theta&0&0\\-sin\theta&cos\theta&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

绕Y轴

$$R(\theta)=\begin{bmatrix}cos\theta&0&-sin\theta&0\\0&1&0&0\\sin\theta&0&cos\theta&0\\0&0&0&1\end{bmatrix}$$

绕X轴

$$R(\theta)=\begin{bmatrix}1&0&0&0\\0&cos\theta&sin\theta&0\\0&-sin\theta&cos\theta&0\\0&0&0&1\end{bmatrix}$$

方向和角度

球坐标表示

设有一点$P(x,y,z)$，从原点到P的距离是$r=\sqrt{x^2+y^2+z^2}$。Q是P点在XY平面上的垂直投影。角β是X轴和OQ之间的夹角，角α是Z轴和OP之间的夹角。

根据假设，角$\triangle OPQ=\alpha$，那么有$OQ=rsin(\alpha)$。从而得出：

$$x = rsin\alpha cos\beta$$
$$y = rsin\alpha sin\beta$$
$$z = rcos\alpha$$

因此，由球坐标(r, α, β)，很容易使用这些公式计算出相应的笛卡尔坐标系(x, y, z)。逆关系也可以从下面的公式中求得：

$$\alpha = acos\frac{z}{r}$$
$$\beta = atan\frac{y}{x}$$

立体角

可以使用一组角度来定义单位球面上的一个点。所有方向矢量的集合可以用如下所示的二维空间来表示：

$$\Omega = \left\{(\theta, \phi)|0 \leq \phi \ < 2\pi, -\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}\right\}$$

假设球面上P点附近的区域为A，相应于A的立体角被定义为

$$\Gamma = \frac{A}{r^2}$$

这里r是球的半径，度量单位成为立体弧度。整个球体包含$4\pi r^2/r^2 = 4\pi$立体弧度。立体角的计算类似于平常的角度，即等于它所围的圆周长度与圆的半径之比。

微分立体角是一个“微分区域”对应的立体角。比如，在球体表面上一个面积趋于零的小面片就是一个微分区域。我们通常使用$d\omega$来表示微分区域。这里将要给出根据方向或球面上的点对应的角度θ和ϕ导出$d\omega$对应的公式。

假设p有球面坐标(θ, ϕ)，点p位于球面上高度为z圆心为C的圆周上。该圆半径为$rsin\theta$，因为沿着这个圆周的一个很小的水平扫略弧度一定是$d\phi$，因此有：

$$d\phi = \frac{dh}{rsin\theta}$$
$$\therefore dh = rd\phi sin\theta$$

同样地，如果考虑包p点的大圆周上所扫略的角度$d\theta$，则有：

$$d\theta = \frac{dv}{r}$$
$$\therefore dv = rd\theta$$

因为立体角是面积除以半径的平方，从上面的两个式子，求出微分立体角为：

$$d\omega = \frac{dh \times dv}{r^2} = sin\theta d\phi d\theta$$

平面保持变换

线段

计算机图形学中一种最基本的形状是连接两点的直线。一种最好的表示方法是它的参数化形式：给定在3D空间的两个点p_1和p_2，那么通过该两点的直线可以被如下所示的方程表示，t为任何实数。

$$p(t) = (1-t)p_1 + tp_2$$

重心组合

参数化直线表示是一种被称为重心组合的特殊情况，所谓重心组合指的是点的加权和，权值总和为1。设$p_1, p_2, \ldots, p_n$是一个点的序列，又$\alpha_1, \alpha_2, \ldots, \alpha_n$是一个实数序列，其和为1，那么

$$p = \sum_{i=1}^{n} \alpha_i p_i = \sum_{i=1}^{n} \alpha_i = 1$$

定义了一个重心组合。显然$n = 2$时它就是上面的直线方程。

仿射变换

仿射变换是一种保持重心组合的变换。设在3D空间中的p点是上面重心组合方程中的重心组合，假设f为从3D空间到3D空间的一个映射，$f(p)$仍然是这个空间中的一个点。那么f是仿射的，当且仅当 $f(p) = \sum_{i=1}^{n} \alpha_i f(p_i)$ 成立。

旋转变换

复数与2D旋转

如果有两个复数$z1 = a + bi$, $z2 = c + di$，可以使用分配律来计算它们的乘积:

$$z1z2 = (a+bi)(c+di) = ac + adi + bci + bdi2 = ac - bd + (bc + ad)i = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}$$

右侧的$\begin{bmatrix} c \\ d \end{bmatrix}$是向量形式表示的$z_2$，而左侧的$\begin{bmatrix} a & -b \\ b & a \end{bmatrix}$则是$z_1$的矩阵形式

那么$z_1 z_2$所表示的变换可以表示为

$$z_1 z_2 = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} c & -d \\ d & c \end{bmatrix} = \begin{bmatrix} ac - bd & -(bc + ad) \\ bc + ad & ac - bd \end{bmatrix}$$

$$z_1 = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} = \sqrt{a^2+b^2} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} ||z|| & 0 \\ 0 & ||z|| \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

此复数是旋转和缩放变换的复合，z_1与任意一个复数c相乘都会将c逆时针旋转$\theta = \arctan 2(b, a)$度，并将其缩放$|| z ||= \sqrt{a^2+b^2}$倍.

将复数与向量v相乘来进行旋转，旋转θ度之后的向量$v\prime$可以用等价的复数乘法来表示:

$$v' = zv = (\cos\theta + i\sin\theta)v$$

根据欧拉公式，还可以进行下一步的变形

$$\cos\theta + i\sin\theta = e^{i\theta}$$

如果定义$r = ||z||$，就得到了复数的极坐标形式：$z = re^{i\theta}$

类似的，向量v的旋转和缩放可以看成

$$v' = re^{i\theta}v$$

三维空间中的旋转

表示三维空间中旋转的方法有很多种。虽然使用欧拉角的旋转很常用，但是欧拉角的表示方法不仅会导致 Gimbal Lock，而且依赖于三个坐标轴的选定，使用四元数正是为了解决这个问题。轴角式旋转是旋转更加普遍的情况。

在轴角的表示方法中，一个旋转的定义需要使用到四个变量：旋转轴u的x, y, z坐标，以及一个旋转角 θ，也就是一共有四个自由度。这很明显是多于欧拉角的三个自由度的．实际上，任何三维中的旋转只需要三个自由度就可以定义了，为什么这里会多出一个自由度呢?

我的理解是，此处定义绕着一个向量u旋转，u的大小并不重要，就像标记地球上一处地点，只需要经度和纬度两个自由度即可。而为了消除模长这个多余的自由度，可以规定模长为1，这样空间中任意一个单位向量就唯一代表了这个方向。因此在计算旋转轴向量u前，必须先将它转为一个单位向量。

四元数

定义

设u_0是一个标量值，$u = (u_1, u_2, u_3)$是一个矢量，四元组(u_0, u_1, u_2, u_3)代表一个四元数.

定义为$u = u_0 + u$或者是$u = u_0 + iu_1 + ju_2 + ku_3$

性质

其中i, j, k具有以下性质：

$$
\begin{aligned}
i^2 = j^2 = k^2 = ijk = -1 \\
ij = k = -ji \\
jk = i = -kj \\
ki = j = -ik
\end{aligned}
$$

纯四元数是指它的标量部分$u_0 = 0$，显然纯四元数与在3D空间中的矢量存在一致关系。

加法

假设$v = (v_0, v_1, v_2, v_3)$是一个四元数，那么

$$
\begin{aligned}
u + v &= (u_0 + v_0) + i(u_1 + v_1) + j(u_2 + v_2) + k(u_3 + v_3) \\
&= (u_0 + v_0, u_1 + v_1, u_2 + v_2, u_3 + v_3)
\end{aligned}
$$

与标量的乘法

$$
c \cdot u = c(u_0 + iu_1 + ju_2 + ku_3) = (cu_0, cu_1, cu_2, cu_3)
$$

四元数乘法

$$
u \times v = [u_0 v_0 - (u \cdot v)] + (u \times v) + u_0 v + v_0 u
$$

假设u是个纯四元数，$u = iu_1 + ju_2 + ku_3$，那么

$$
u^2 = -(u \cdot u)
$$

四元数的逆

$$
u' = \frac{u^\star}{|u|^2}
$$

其中$u^\star$是四元数u的共轭。

四元数的旋转

假设有一个旋转序列，其中各个旋转分别为绕轴$s_i, i = 1, \ldots, n$，角度为$2\theta_i$，那么可以通过应用以下序列的四元数旋转算子得到：

$$
u_n u_{n-1} \ldots u_1 p u_1^\star \ldots u_{n-1}^\star u_n^\star
$$

这里$u_i = cos\theta_i + sin\theta_i$，这等价于求如下的四元数：

$$
u = u_n u_{n-1} \ldots u_1
$$

光照

我的观点，计算机图形学最核心和最有魅力的部分就在于光照。合适的光照提供的模拟可以极大地提升沉浸式的体验。光照问题中要重点关注的一个概念就是场景，所谓场景是一个图形对象的集合，是一个表现对象集合的数据结构。然而，场景必须要产生而不是幻觉中的场景，因此计算机图形学关心的是场景虚拟模型的构造。场景的光照问题是计算机图形学中的一个核心概念和实践问题。这个问题是一个在场景中模拟光照的问题，这个过程不仅要计算快速，而且要达到实时。计算速度非常快以至于人不能分辨真实和虚拟。

光

可见光是波长大约在$400nm - 700nm$范围的电磁辐射，波长引起对颜色的感知。而如果采用粒子理论，这种粒子叫做光子，每个光子携带的能量为E，该值正比于它的频率f:

$$E = hf$$

光子还有另一个不寻常的性质，它们彼此之间没有干涉现象发生——两条光线交叉彼此不互相影响。光在一个空间范围内如何与表明交互是出道舞台的一个实例，通常这与空间中移动粒子的分布有关。

光通量：用Φ来表示辐射能量或在体积V中的光通量。光通量是指单位时间内通过某个表明的能量（单位：瓦特）。能量正比于粒子流，因此光通量可以看成是单位时间内的光子能量。事实上，能量是正比于波长的所以要完全定义辐射能量，需要使用几号Φ_λ，即波长为λ的光波的辐射能量。从感知的角度看λ产生颜色的感知，而光通量引起亮度的感知。

对于一个个体的总光通量来说，首先是一种动态的平衡，场景中的某个部分不会变得忽明忽暗。其次是遵循能量守恒，因此有方程：

$$\text{发射} + \text{入射} = \text{直接流量} + \text{出射} + \text{吸收}$$

使用微分和积分的形式可表示为：

$$\Phi_e + \Phi_i = \Phi_s + \Phi_o + \Phi_a$$

简化假设

求解微分方程得到Φ（p，ω）实际上是无法达到的，一个满足实时性能的解和产生光照真实感的解是完全不同的，在这两种极端情形之间必须有一个折衷。因此需要一些简化假设。

波长独立性假设：通常假设不同波长的光波之间没有交互影响，通过估计一个点的波长分布，针对不同的波长独立求解，通过这些解的组合来获得具体的解。

时间不变性假设：假设能量分布方程不随着时间而改变。

真空中光传导假设：光传导的空间是真空的，图形对象之外没有其他物质存在。即吸收和出射只在表面的边界上发生，除了对象外没有其他物质能发射出光线，除了在对象的表面外没有散射或吸收现象发生。意味着不受阻碍的光线没有能量被吸收。

对象均质性假设：当光子撞击一个表面时，它们的能量有一部分会被吸收，还有一部分被反射出去。对于均质材料，如果我们考虑光的入射和出射方向，则它们之间的关系在对象的整个表面上处处都是一样的。

术语概念

光亮度

光亮度L定义为物体表面某个方向单位投影区域单位立体角的光通量。

设dA是表面区域，其法向为n，光从某一个与该法向夹角为θ的方向离开表面，经过的微分立体角为$d\omega$，如果离开微分区域的dA的光亮度为L，则对应的光通量为：

$$d^2\Phi = LdA\cos\theta d\omega$$

让dA和$d\omega$都非常非常小，$d^2\Phi$表示了在方向θ上的光通量。给出两个小面片dA和dB。假设r为它们之间的距离，从dA到dB的光亮度为L。设$\Phi(dA, dB)$为辐射能量的转移，则有：

$$\Phi(dA, dB) = LdA\cos\theta_A d\omega_B$$

由此可以得到：

- 光在相反方向上的光通量与原方向上光通量没有差别——交换光的方向时，光能量不发生改变。
- 光通量与距离的平方成反比，当两个区域间的距离变得更大时，光通量将减小，但光亮度不是这样的——衍射某条光线的光亮度是个常数，它与距离光源远近是没有关系的。

光强度

光强度I指单位立体角内的辐射能量（光通量）

$$d\Phi = Id\omega$$

与光亮度L关系：

$$I = LdA\cos\theta$$

辐射度

辐射度B是从一个表面的单位面积上流出的光通量，若辐射度B与离开区域dA的能量相关，那么光通量可以被计算成：

$$d\Phi = BdA$$

辐照度

辐照度E是到达一个表面的单位面积上的光通量，光通量可被计算为：

$$d\Phi = EdA$$

反射

假设一条光线射向表面的点p位置，其入射方向为ω_i。光能量将要反射的是个半球体，以在p点的切平面为底的一个半球。该半球包含了从p点可见的所有方向的集合，在这些方向上的光线将不会收到来自表面的任何部分的遮挡。有多少能量从反射方向ω_r离开表面？此时引入双向反射分布函数（$BRDF$）f（p，ω_i，ω_r），通过它把在点p方向为ω_r的反射光亮度和在点p处入射方向为ω_i的入射光联系在一起。则有：

$$L(p,\omega_r) = f(p,\omega_i,\omega_r)E(p,\omega_i)$$

对真实表面的函数f（p，ω_i，ω_r）精确描述是一件极端复杂的事，尤其当表面为非均质的时候，此时要对真实世界做进一步的抽象，主要采用两种理想化的材料属性：

- 镜面反射器：镜面反射器是像镜子一样的表面，它将入射光线反射出去，入射角等于反射角，且相应于ω_i的矢量和相应于ω_r的矢量于表面法向位于同一个平面。
- 漫反射器：漫反射器是一个粗糙的表面，它将入射光线的光亮度均匀地以p点为中心的半球范围内的所有方向散布，且能证明：

$$f(p,\omega_i,\omega_r) \propto \frac{1}{\pi}$$

光亮度方程

如果可以求出在所有点和所有方向上的$\phi(p,\omega)$，那么光照问题会完全得到解决，并且我们有关于这个函数的方程。其次我们给出了许多限制性假设，这是简化方程求解的重要条件。同时我们说明了该函数的目的不是在于$\phi(p,\omega)$,而是$L(p,\omega)$。在前面所有的简化假设的基础上，使用先前引进的各项，能够导出一个新方程，该方程使用光亮度而非光通量，更简单且更易理解——这就是光亮度方程，它给出了在表面上点处沿着给定方向ω的光亮度。光亮度一定是两个量的总和，第一个量是直接从该点发射的光亮度的大小，第二个量是从这个点反射的光亮度的大小。反射量的大小可以由所有射入这个表面p点的光线总和（总辐照度）乘以$BRDF$，因此有：

$$\text{光亮度} = \text{发射光亮度} + \text{总反射光亮度}$$

对任何入射方向ω_i其在方向ω上反射光亮度是辐照度乘以$BRDF$：

$$f(p,\omega_i,\omega)L(p,\omega_i)cos\theta_i d\omega_i$$

对于出射光亮度$L(p,\omega)$的光亮度方程是：

$$L(p,\omega) = L_e(p,\omega) + \int_\Omega f(p,\omega_i,\omega)L(p,\omega_i)cos\theta_i d\omega_i$$

$$= L_e(p_i,\omega) + \int_0^{2\pi}\int_0^{\frac{\pi}{2}} f(p,\omega_i,\omega)L(p,\omega_i)cos\theta_i sin\theta_i d\theta d\phi$$

这里$L_e(p,\omega)$是发射光亮度。

考虑点p处任意一条入射光线，依照方程需要沿着这条光线计算光亮度。因此沿着入射方向ω_i的反方向回溯直到击中另一个表面上的p\点，求出光亮度$L（p',\omega_i）$。但是为了计算需要再调用光亮度方程。也就是说相应于每一条入射光线（无穷多），都有这样的关系，说明光亮度如何沿着那条光线产生的。光亮度方程强调的是光照的全局效果，所看到的反射光依赖于入射光和材质的$BRDF$属性，而入射光又依赖于环境中其他表面产生的反射光。

光亮度方程的解

在计算机图形学应用中光亮度方程的重要性主要在于它暗含了所有可能的二维视图。如何抽象所需的信息来构造这样的图像。从光亮度方程中抽取出二维投影图像的过程叫做渲染。

- **视图独立解**：对场景的所有表面和尽可能多的方向上预计算$L（p,\omega）$的值，当需要一幅特定的图像时，对应于经过透镜的那些表面的方向的$（p,\omega）$被计算出来，其相应的$L（p,\omega）$就不再经过计算而是直接查表可得
- **视图依赖解**：光亮度方程的求解只求出形成图像所需要的光线集合。如果观察条件发生改变，那么被看到的光线计算光亮度的整个过程必须被重新执行。此处的视图

依赖解专门对一组（p，ω）计算L（p，ω）。

视图独立的方法的优势在于产生场景的一个图像所需的时间是个常数，独立于场景的复杂度和特定视图，所需的时间包括计算光线的方向和光线与透镜的相交时间，以及查对应L值的时间。

对光亮度方程求解方法可以被独立分成两种类型：

局部解：只考虑光源对对象的直接效果，不考虑对象间的反射，消除了光亮度方程中的递归操作。用来自光源的光线在入射方向上的求和计算替代积分计算。进一步简化，通常假设光源都是一些点，所以对于对象表面的每个点，都有唯一的一条光线代表来自一个特殊光源的入射光角度。

全局解：利用光亮度方程的递归性质，至少会有某些类型的对象间相互反射要被考虑在内。又分为"光线追踪法"，"辐照度法"，"蒙特卡洛法"求近似解。

简化方法

平淡明暗处理图形：

完全忽略积分项，意味着每个对象有一个预先定义好的光亮度值，从任何视点看去，对象都只是在二维平面的投影，显示自身的颜色。解中没有用到递归，甚至不受主要光源的影响。

光线追踪：

对光亮度方程进行简化，只允许点光源存在，同时$BRDF$只考虑镜面反射。例如在场景中找出表面上一点p，该点为光线的出发点，沿着光线从点p到点光源，并由此计算L_e（p，ω）。这个量也包括一个称为"环境光"的成分和一个称为"漫反射光"的成分。其中"环境光"是被假设用来表示来自间接光照的总背景光照，所谓"漫反射光"是表示表面的任意漫反射特性。光亮度方程余下的部分是递归地进行光线追踪，从p点开始沿着光线的反射方向跟踪光线直到碰到另一个表面，然后同样沿着光线递归地计算光亮度。这个递归过程连续进行，直到光亮度增量下降到某个预定的阈值以下。

蒙特卡洛路径追踪：

与光线跟踪相似，生成光亮度方程的一个估计解，该解包括镜面反射和漫反射。与递归光线不同，递归光线所沿着的特殊路径是取决于镜面反射方向，而此方法所生成的光线是沿着一个随机选取的方向。这样假设与表面相交为p，$BRDF$随机地旋转一条光线计算。这样从一个相交点到另一个相交点直到增量可以忽略不计。对每条主光线整个过程重复很多次，最后取结果的平均值。

实时图形

进一步简化光亮度方程，递归成分被完全去掉。光源仍然是点光源，只包括直接光照在内，积分由光源之和所取代，而且只计算到达表面上点p的局部贡献，没有关于光线与对象相交的繁重计算。

辐照度

让$BRDF$变成常数，不因为方向的改变而变化，并完全地除去方程中有关方向的那些项。用辐照度取代光亮度。假设所有表面都只是漫反射器，场景中的表面被分隔为很多小的表面单元，光亮度方程降为一系列线性方程。这个方程的解是与每个小表面单元相关的辐射度或者是每个小表面单元边界上点的辐照度。

蒙特卡洛光子追踪

尝试求得光亮度方程的一般统计解，从光源开始到场景内部对大量随机分布的光线进行跟踪。每条光线跟踪到与它相交的最近一个对象，进一步产生的光线基于对象的$BRDF$。将对象分割成小的表面单元，用来估计对象表面的一个连续密度函数，以便估计在表面上从任意位置处和任意方向上的光亮度。

颜色

色彩是计算机图形学不可或缺的一部分，如何通过代码模拟人眼在现实中看到的每一种色彩，对于展现计算机图形学的魅力十分重要。如果不能恰当地表现颜色，那么很多技术都会失去意义，例如光照。而仅仅能表现颜色还不够，人们还要尽可能地复现每一种颜色在每一台显示设备上在每个人眼中的感觉，由此衍生出许多颜色系统，被显示屏产业，印刷业等广泛应用。

颜色系统

计算机图形渲染中普遍使用来自实际显示器的RGB空间中的颜色。光源发射器所发出的光是用RGB三元组来表示的，表面反射函数也是用这些RGB值来表示光能量的，并用这些RGB值来设定像素。这种做法是不正确的，因为在不同的显示器上进行相同计算会产生不同的彩色图像，因为RGB系统是高度依赖设备的，对于一台显示器的RGB值在另一台显示器上可能形成不同的颜色。并且由于采用RGB值来度量光的能量，其不能很好地区分不同的颜色，因为区分颜色的两个方面，一是能量，依据波长分布，二是视觉系统如何作出反应。

当图形有意要仿真环境的光照效果时，要生成真实的图像，那么采用RGB系统就是完全不合适的，因为其为了计算上的方便牺牲了光照仿真的视觉效果。更恰当的方法是计算颜色的光谱亮度分布，使用$CIE-XYZ$颜色匹配函数将光谱转换成XYZ坐标。

视觉系统

光线经由瞳孔进入眼睛，孔径的尺寸是受虹膜控制的——对黑暗环境它变得比较宽、对明亮一些的环境则变得比较狭窄。光线的聚焦主要靠眼睛这个光学系统、它主要由虹膜、瞳孔、角膜所组成。角膜内充满晶状体，精细的调焦由透镜元成。透镜的厚度被睫状肌所控制。眼睛的后部有视网膜，它是由数以百万计的感光单元阵列所组成。在视网膜上视轴正对终点有一个小区域称为黄斑中心凹，在该黄斑区中感光单元特别密集，是视网膜上视觉最为敏锐的特殊区域。无论向什么地方看去，只有一处完全地聚焦任黄斑区上。之所以整个图像好像都是聚焦的，是因为我们的眼睛在不断地移动，使得不同的场景区域进入焦点。

在视网膜上：有两类感光单元——分别称为杆状细胞和圆锥细胞。大约有$130,000,000$个杆状细胞、$5,000,000-7,000,000$个圆锥细胞。杆状细胞分布任整个视网膜上，负责夜视，同时它也对运动高度敏感——尤其对外围的物体运动。圆锥细胞几乎完全集中任黄斑区及其附近：它的任务是保证视觉灵敏度和彩色视觉，只在白昼工作。

根据对不同波长光的反应将圆锥细胞分为三种类型。第一类对长光波（ 红色） 反应灵敏， 称之为L 型圆锥细抱； 第二类对中等波长的光波（ 绿色） 较为灵敏， 称之为M 型圆锥细胞， 第三类对短波长光波（ 蓝色） 比较灵敏， 称之为S 型圆锥细胞。

$$l = \int_{\Lambda} C(\lambda)L(\lambda)d\lambda$$
$$m = \int_{\Lambda} C(\lambda)M(\lambda)d\lambda$$
$$s = \int_{\Lambda} C(\lambda)S(\lambda)d\lambda$$

这里$\Lambda = [\lambda_a, \lambda_b]$。

这组LMS光谱反应函数将无穷颜色空间映射到三维颜色空间，该三维颜色空间用三元组(l, m, s)的形式来表示。

从上式可以看出，完全不同的颜色仍然能被映射为相问的三元组——虽然光谱分布是截然不可的，但它被感知为相问的颜色。如前面曾经提到过的，这被称作条件等色。如果将上式所表示的映射表示为$LMS(C) = (1, m, s)$,那么当下式成立时C_a和C_b是条件等色：

$$LMS(C_n) = LMS(C_b) = (l, m, s)$$

发射器系统

发射器系统通过混合具有不同光谱分布光的能量束来生成彩色的光，每种光的光谱能量分布为E_i （ λ），$i = 1$，2，3.那么发射器的光谱分布为：

$$C_E(\lambda) = \alpha_1 E_1(\lambda) + \alpha_2 E_2(\lambda) + \alpha_3 E_3(\lambda)$$

α_i 是光强度，E_i 构成基色，有一些可见颜色不能够由这种物理系统产生

国际照明协会定义了$CIE - RGB$基本光：

$$E_R(\lambda) = \delta(\lambda - \lambda_R), \lambda_R = 700nm$$
$$E_G(\lambda) = \delta(\lambda - \lambda_G), \lambda_G = 700nm$$
$$E_B(\lambda) = \delta(\lambda - \lambda_B), \lambda_B = 700nm$$

它们称为$CIE - RGB$原色。通过混合这些颜色，就可以生成所有可构造颜色集合中的任意一种了，也就是RGB空间。

$$C_{RGB}(\lambda) = \alpha_R E_R(\lambda) + \alpha_G E_G(\lambda) + \alpha_R B_B(\lambda)$$

CIE-RGB颜色匹配函数

使用从相对直接的实验中估计出来的函数的方法，这些函数称为颜色匹配函数，对颜色理论具有至关重要的意义。

考虑波长为λ_0的单色光，其光谱分布为$\delta(\lambda-\lambda_0)$。为了产生这种颜色，设光发射器所采用的基函数$E_i(\lambda)$的发光强度为$\gamma_i(\lambda_0)$。那么有：

$$\delta(\lambda-\lambda_0)\approx\sum_{i=1}^{3}\gamma_i(\lambda_0)E_i(\lambda)$$

结合前面的式子，经过推导，有以下结论：

$$\alpha_i=\int_{\Lambda}\gamma_i(\lambda)C(\lambda)d\lambda$$

换句话说，能够通过对光谱分布与γ_i函数的积做积分来得到基色$E_i(\lambda)$的光强水平(α_i)。因为它们给出了任意波长λ纯颜色相应的基颜色的光强度匹配，所以这些函数被称为颜色匹配函数。

颜色匹配函数可以通过简单的实验来估计。先产生一束波长为λ_0的纯参考颜色的光。同时用三种基光谱的光束叠加来产生一束光，调整这些光束的强度直到所叠加的光正好与参考光束相匹配为止。记录下三种基光谱的强度值$\gamma_1(\lambda_0)$、$\gamma_2(\lambda_0)$、$\gamma_3(\lambda_0)$。现在对可见光谱范围内的各种波长值$\lambda_1,\lambda_2,\ldots,\lambda_n$重复相同的实验。这样就得到了对三个函数的估计。

假如知道了来自表面$C(\lambda)$的颜色的光谱分布，希望在显示器上产生一种颜色，保证给人的感觉好像与观察者在现实活中看到的$C(\lambda)$一样。假设显示器采用$CIE-RGB$原色系统。那么使用上式计算光束光强度(α_i)，这里γ_i是RGB颜色匹函数。将这些光强度传递到显示器上，这可以被当作一种行动，根据前面的式子来产生$C(\lambda)$的条件等色。设$CIE-RGB$颜色匹配函数是：

$$\bar{r}=\gamma_1(\lambda)$$
$$\bar{g}=\gamma_2(\lambda)$$
$$\bar{b}=\gamma_3(\lambda)$$

那么：$C(\lambda)\approx\alpha_R E_R(\lambda)+\alpha_G E_G(\lambda)+\alpha_B E_B(\lambda)$，这里：

$$\alpha_R = \int_\Lambda \bar{r}(\lambda)C(\lambda)d\lambda$$

$$\alpha_G = \int_\Lambda \bar{g}(\lambda)C(\lambda)d\lambda$$

$$\alpha_B = \int_\Lambda \bar{b}(\lambda)C(\lambda)d\lambda$$

这样，就能通过数值积分求得RGB光强度值$(\alpha_R, \alpha_G, \alpha_B)$。

CIE-RGB色度空间

相应于每一个颜色C（λ），在$CIE-RGB$空间中都存在一个条件等色。这种光谱函数的无穷维空间到三维空间的映射显然是多对一的，$CIE-RGB$色度空间就是设想把每一个可见颜色转换到$3D$点上，且最终用二维面代替三维体。

颜色包含两种截然不同而独立的属性，一个是标量，称之为**亮度**，另一个是二维矢量，称之为**色度**。

$$\alpha_R l_R + \alpha_G l_G + \alpha_B l_B = L$$

CIE-XYZ色度空间

$CIE-RGB$色度图相当奇怪，其中可见颜色的很大部分不能通过发射器的$CIE-RGB$原色生成，在色度图的第一象限外面的颜色对应负的光强度，因此都不能用基色生成。

$CIE-XYZ$有三个基函数，分别是X（λ），Y（λ），Z（λ）。这种定义下X和Z具有零亮度，而Y的颜色匹配函数等于发光效率曲线V。

E是RGB系统，F为XYZ系统，那么可以通过矩阵A从RGB转到XYZ:

$$\begin{bmatrix} 0.489989 & 0.310008 & 0.2 \\ 0.176962 & 0.81240 & 0.010 \\ 0.0 & 0.01 & 0.99 \end{bmatrix}$$

也可以通过A^{-1}将$CIE-XYZ$颜色转为等价的$CIE-RGB$颜色:

$$\begin{bmatrix} 2.3647 & -0.89658 & -0.468083 \\ -0.515155 & 1.416409 & 0.088746 \\ 0.005203 & -0.014407 & 1.0092 \end{bmatrix}$$

RGB和XYZ颜色空间之间的转换

考虑一个显示RGB系统，它有三个原基色$R(\lambda)$、$G(\lambda)$、$B(\lambda)$。因为它们是实际颜色，会有在XYZ色度空间中的一个表示。假设关系如下：

$$
\begin{aligned}
R(\lambda) &= X_R X(\lambda) + Y_R Y(\lambda) + Z_R Z(\lambda) \\
G(\lambda) &= X_G X(\lambda) + Y_G Y(\lambda) + Z_G Z(\lambda) \\
B(\lambda) &= X_B X(\lambda) + Y_B Y(\lambda) + Z_B Z(\lambda)
\end{aligned}
$$

因此对应于这些颜色的二维空间XYZ色度坐标是：

$$
\begin{aligned}
\left(\frac{X_R}{X_R + Y_R + Z_R}, \frac{Y_R}{X_R + Y_R + Z_R} \right) &= (x_R, y_R) \\
\left(\frac{X_G}{X_G + Y_G + Z_G}, \frac{Y_G}{X_G + Y_G + Z_G} \right) &= (x_G, y_G) \\
\left(\frac{X_B}{X_B + Y_B + Z_B}, \frac{Y_B}{X_B + Y_B + Z_B} \right) &= (x_B, y_B)
\end{aligned}
$$

实际上，显示设备制造商公布二维空间的色度，因此上式的右边是已知的。因为上式的分母是未知的，相应于RGB的XYZ颜色可以写成：

$$
\begin{aligned}
C_R &= \alpha_R(x_R, y_R, z_R) \\
C_G &= \alpha_R(x_G, y_G, z_G) \\
C_B &= \alpha_R(x_B, y_B, z_B)
\end{aligned}
$$

现在寻找一个能完成从RGB到XYZ的转换矩阵A。特别地，RGB颜色$(1,0,0)$应该映射为C_R，$(0,1,0)$映射为C_G，且$(0,0,1)$映射为C_R。因此：

$$
\begin{aligned}
C_R &= (1,0,0)A \\
C_G &= (0,1,0)A \\
C_B &= (0,0,1)A
\end{aligned}
$$

所以有

$$
A = \begin{bmatrix}
\alpha_R x_R & \alpha_R y_R & \alpha_R z_R \\
\alpha_G x_G & \alpha_G y_G & \alpha_G z_G \\
\alpha_B x_B & \alpha_B y_B & \alpha_B z_B
\end{bmatrix}
$$

还需要确定$(\alpha_R, \alpha_G, \alpha_B)$。在$XYZ$系统中白色点是$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right)$。在$RGB$系统中白色点一般为$(1,1,1)$。那么有

$$
(1,1,1) = \frac{1}{3}(1,1,1)A
$$

重新整理得

$$\begin{bmatrix} x_R & x_G & x_B \\ y_R & y_G & y_B \\ z_R & z_G & z_B \end{bmatrix} \begin{bmatrix} \alpha_R \\ \alpha_G \\ \alpha_B \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

从中可以很容易地求得未知量。这个矩阵有两个目的：

- 给定在某一特定显示器上的一个颜色，我们希望能在另外一个显示器上重新得到。可以使用矩阵A将它转换到XYZ系统，然后再使用相应于第二个显示器的A^{-1}，将它从XYZ系统转换到第二个显示器的RGB上。
- 给定一个计算得出的XYZ颜色，能够使用A^{-1}转换它到一个特定显示器的RGB上。

坐标系统

不同的操作，不同的处理往往要在不同的坐标系下使用才会更加方便。就像实际物理问题，往往也是选取不同参考系进行研究。而最终要把不同的坐标系统统一起来，或者说让它们协同工作，互相配合，就需要不同坐标系统间的变换矩阵。不同的坐标系统有不同的空间名字。除了这些变换矩阵外，还有一种特殊的变换方式——投影，根据是否保留细节，还是符合透视原理的近大远小旋转正射投影和透视投影。

变换矩阵

为了将坐标从一个坐标系变换到另一个坐标系，需要用到几个变换矩阵，最重要的几个分别是模型$Model$、观察$View$、投影$Projection$三个矩阵。我们的顶点坐标起始于局部空间$Local\ Space$，在这里它称为局部坐标$Local\ Coordinate$，它在之后会变为世界坐标$World\ Coordinate$，观察坐标$View\ Coordinate$，裁剪坐标$Clip\ Coordinate$，并最后以屏幕坐标$Screen\ Coordinate$的形式结束。

1. 局部坐标是对象相对于局部原点的坐标，也是物体起始的坐标。
2. 下一步是将局部坐标变换为世界空间坐标，世界空间坐标是处于一个更大的空间范围的。这些坐标相对于世界的全局原点，它们会和其它物体一起相对于世界的原点进行摆放。
3. 接下来将世界坐标变换为观察空间坐标，使得每个坐标都是从摄像机或者说观察者的角度进行观察的。
4. 坐标到达观察空间之后，需要将其投影到裁剪坐标。裁剪坐标会被处理至-1.0到1.0的范围内，并判断哪些顶点将会出现在屏幕上。
5. 最后，将裁剪坐标变换为屏幕坐标，将使用一个叫做视口变换$Viewport\ Transform$的过程。视口变换将位于-1.0到1.0范围的坐标变换到由$glViewport$函数所定义的坐标范围内。最后变换出来的坐标将会送到光栅器，将其转化为片段。

之所以将顶点变换到各个不同的空间的原因是有些操作在特定的坐标系统中才有意义且更方便。例如，当需要对物体进行修改的时候，在局部空间中来操作会更说得通；如果要对一个物体做出一个相对于其它物体位置的操作时，在世界坐标系中来做这个才更说得通。

空间

局部空间

局部空间是指物体所在的坐标空间，即对象最开始所在的地方。比如创建了一个立方体。创建的立方体的原点有可能位于$(0, 0, 0)$，即便它有可能最后在程序中处于完全不同的位置。甚至有可能创建的所有模型都以$(0, 0, 0)$为初始位置。所以，模型的所有顶点都是在**局部**空间中：它们相对于物体来说都是局部的。

世界空间

我们想为每一个物体定义一个位置，从而能在更大的世界当中放置它们。世界空间中的坐标正如其名：是指顶点相对于世界的坐标。物体的坐标将会从局部变换到世界空间；该变换是由模型矩阵$Model\ Matrix$实现的。

模型矩阵是一种变换矩阵，它能通过对物体进行位移、缩放、旋转来将它置于它本应该在的位置或朝向。

观察空间

观察空间经常被人们称之$OpenGL$的摄像机$Camera$（所以有时也称为摄像机空间($Camera\ Space$)或视觉空间($Eye\ Space$)）。观察空间是将世界空间坐标转化为用户视野前方的坐标而产生的结果。因此观察空间就是从摄像机的视角所观察到的空间。而这通常是由一系列的位移和旋转的组合来完成，平移/旋转场景从而使得特定的对象被变换到摄像机的前方。这些组合在一起的变换通常存储在一个观察矩阵($View\ Matrix$)里，它被用来将世界坐标变换到观察空间。

裁剪空间

在一个顶点着色器运行的最后，$OpenGL$期望所有的坐标都能落在一个特定的范围内，且任何在这个范围之外的点都应该被裁剪掉。被裁剪掉的坐标就会被忽略，所以剩下的坐标就将变为屏幕上可见的片段。

为了将顶点坐标从观察变换到裁剪空间，需要定义一个投影矩阵($Projection\ Matrix$)，它指定了一个范围的坐标，比如在每个维度上的-1000到1000。投影矩阵接着会将在这个指定的范围内的坐标变换为标准化设备坐标的范围$(-1.0, 1.0)$。所有在范围外的坐标不会被映射到在-1.0到1.0的范围之间，所以会被裁剪掉。

投影

将特定范围内的坐标转化到标准化设备坐标系的过程（而且它很容易被映射到$2D$观察空间坐标）被称之为投影($Projection$)，因为使用投影矩阵能将$3D$坐标投影到很容易映射到$2D$的标准化设备坐标系中。

一旦所有顶点被变换到裁剪空间，最终的操作——透视除法($Perspective\ Division$)将会执行，在这个过程中我们将位置向量的x，y，z分量分别除以向量的齐次w分量；透视除法是将$4D$裁剪空间坐标变换为$3D$标准化设备坐标的过程。

正射投影

正射投影矩阵定义了一个类似立方体的平截头箱，它定义了一个裁剪空间，在这空间之外的顶点都会被裁剪掉。创建一个正射投影矩阵需要指定可见平截头体的宽、高和长度。在使用正射投影矩阵变换至裁剪空间之后处于这个平截头体内的所有坐标将不会被裁剪掉。

透视投影

透视投影矩阵将给定的平截头体范围映射到裁剪空间，除此之外还修改了每个顶点坐标的w值，从而使得离观察者越远的顶点坐标w分量越大。被变换到裁剪空间的坐标都会在$-w$到w的范围之间（任何大于这个范围的坐标都会被裁剪掉）。一旦坐标在裁剪空间内之后，透视除法就会被应用到裁剪空间坐标上：

$$out = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

组合

一个顶点坐标将会根据以下过程被变换到裁剪坐标：

$$V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$$

照相机

靠眼睛发射光线来完成对视锥体内物体的渲染和光线追踪是一件效率低下的事情，需要同时计算太多光线，其中很多光线完全是不必要的。这就衍生出使用照相机去“接收”光线的方法，每一条光线都来自于一个物体，这样处理效率就大大提高。为了使用照相机完成这些工作，就需要定义一个特殊的坐标系，让照相机在新的视图平面上工作。

观察坐标系

当说到摄像机/观察空间的时候，是在讨论以摄像机的视角作为场景原点时场景中所有的顶点坐标：观察矩阵把所有的世界坐标变换为相对于摄像机位置与方向的观察坐标。要定义一个摄像机，需要它在世界空间中的位置、观察的方向、一个指向它右侧的向量以及一个指向它上方的向量。

获取摄像机位置很简单。摄像机位置简单来说就是世界空间中一个指向摄像机位置的向量。用场景原点向量减去摄像机位置向量的结果就是摄像机的指向向量。

需要的另一个向量是一个**右向量**($Right\ Vector$)，它代表摄像机空间的x轴的正方向。为获取右向量我们需要先使用一个小技巧：先定义一个**上向量**($Up\ Vector$)。接下来把上向量和第二步得到的方向向量进行叉乘。两个向量叉乘的结果会同时垂直于两向量，因此会得到指向x轴正方向的那个向量。

已经有了x轴向量和z轴向量，获取一个指向摄像机的正y轴向量就相对简单了：把右向量和方向向量进行叉乘。

视图参照点（VRP）：这是WC中的一个点，它定义了新观察坐标系的原点。直观上可以把它看作是场景中的一个相关点，或者是描述“照相机”（即视平面和投影中心）的点。

视平面法向（VPN）：这是WC中的一个矢量，它的方向规定了观察坐标系的正Z轴方向。因此这个轴是经过VRP并与VPN平行的。它可以看成是抽象照相机所朝向的方向。

视图上方矢量（VUV）：WC中这个矢量定义了新坐标系的正Y轴方向。Y轴是将VUV向垂直于VPN并穿过VRP点的平面投影所得的。

观察坐标系的三个主要轴的名字又给了坐标系另一个名字——UVN系统。

LookAt 矩阵

使用矩阵的好处之一是如果你使用3个相互垂直（或非线性）的轴定义了一个坐标空间，你可以用这3个轴外加一个平移向量来创建一个矩阵，并且你可以用这个矩阵乘以任何向量来将其变换到那个坐标空间。

其中是右向量，是上向量，是方向向量是摄像机位置向量。注意，位置向量是相反的，因为最终希望把世界平移到与自身移动的相反方向。把这个$LookAt$矩阵作为观察矩阵可以很高效地把所有世界坐标变换到刚刚定义的观察空间。

欧拉角

欧拉角($Euler\ Angle$)是可以表示$3D$空间中任何旋转的3个值。一共有3种欧拉角：**俯仰角**($Pitch$)、**偏航角**(Yaw)和**滚转角**($Roll$)。

俯仰角是描述我们如何往上或往下看的角。偏航角表示我们往左和往右看的程度。滚转角代表我们如何翻滚摄像机。每个欧拉角都有一个值来表示，把三个角结合起来就能够计算3D空间中任何的旋转向量了。

场景构造

在许多大型模拟游戏中，我们会看到许许多多的3D对象，它们的纹理非常精细，它们确实与真实相混淆。但是这个三维的本质是什么？本章主要介绍立体几何。需要体会表示和渲染之间的区别——表示是一个数据集，一种描述对象的方法，它是一种客观形式；渲染是一种将现有数据与场景的当前情况相结合的技术，以便在透视图中映射对象。每个图形，甚至是一个实体，都是由平面多边形组成的——换句话说，它是由无数个面片单元组成的。

多边形与平面

平面方程

平面方程形式：

$$ax + by + cz = d$$

要求多边形的所有点位于相同平面是非常强的限制，所以在实际使用中大多数使用三角形。

光线与多边形相交

求解光线与多边形的交点，共分为两步：

1. 求解光线与多边形所在平面的相交点（除非光线与平面平行）
2. 将多边形和光线与平面的交点p投影到另一个平面（多边形不能与投影面垂直），因为投影后交点是否在多边形内部的关系不会改变，投影后判断光线与平面的交点是否在多边形内。

多面体

简单多面体有下列性质：

- 每条边正好连接两个顶点，且正好为两个面的公共边界。
- 每个顶点至少为三条边的交点。
- 除了沿着两个面的公共边之外，它们不再有别的相交
- 多面体的边数（E）、面数（F）和顶点数（V）总是满足欧拉公式：$V - E + F = 2$

顶点-面数据结构

把多面体的每个面——多边形独立存储，这样存储效率不高，每个顶点至少被存储3次，每条边至少被存储2次。

数据结构非常简单，但是能力非常有限。

翼边数据结构

整个多面体是通过一种骨来表示的，这个骨架包含了各种环结构（都是双向鲢表），分别有顶点环边坏和面环，除此之外，骨架还包含到其他骨架的几何系信息。在顶点环中每个节点包含一个$3D$点、到下一个和先前一个顶点的指针、以及到边坏的指针。顶点环保存了对象的几何信息。边环保存了对象的拓扑信息——边坏还保存有到下一条边和和先前边的指针，还有到所谓的"翼边"结构的指针，以及到它附近顶点和曲的指针。环包含到下一个面和到先前一个面的指针，也包含到边环的指针。

优点：可以方便、快速地获得以下信息：

1. 哪些面使用了这个顶点
2. 哪些边使用了这个顶点
3. 哪些面使用了这条边
4. 哪些边构成了这个面
5. 哪些面和这个面相邻

缺点：

1. 缺点是太零散：在查询哪些是多边形以及一个多边形的边数时，无法直接查询，需要通过非得循环找一圈才能得出结果。

场景层次结构

对象通过数据结构来定义它们的几何性质（顶点）、拓扑性质（边和面间的关系）以及材质属性。对象可以以各种不同的方式被操纵——平移到另一个位置，沿着轴旋转，伸缩变换，以及这些变换的组合。这些变换可以通过矩阵变换来实现。当然，对象不是孤立存在的，而是相互依赖的——在一个对象或是某一对象的一部分上的变换通常会对其他对象带来一定影响。可以通过建立一个数据结构来实现相互依赖的变换。

光照模型

想要通过微分和积分完美地展现光照效果，在一些大型项目中就行不通了。就像物理上研究问题，有时会对模型进行适当的简化。同理，在光照模型上，冯氏光照模型就是很好的简化模型，它在保证光照模拟效果的基础上大大减少了计算量，使得模拟更接近于实时，极大地提升了模拟效果。

漫反射和朗伯定律

漫反射发生在完全不光滑的表面，这里入射光的反射从任何角度看似乎都是一样的，其实这种表面遵从朗伯余弦定律。朗伯定律指出：对于漫反射体，表面任意一点处任意方向的反射光强度和光源入射角的余弦成正比。因此，漫反射表面那一点上的光亮度与观察角度无关。

冯氏光照模型(Phong Lighting Model)

现实世界的光照是极其复杂的，而且会受到诸多因素的影响，这是我们有限的计算能力所无法模拟的。因此$OpenGL$的光照使用的是简化的模型，对现实的情况进行近似，这样处理起来会更容易一些，而且看起来也差不多一样。这些光照模型都是基于我们对光的物理特性的理解。其中一个模型被称为冯氏光照模型($Phong\ Lighting\ Model$)。冯氏光照模型的主要结构由3个分量组成：**环境**($Ambient$)、**漫反射**($Diffuse$)和**镜面**($Specular$)光照。

- **环境光照(Ambient Lighting)**：即使在黑暗的情况下，世界上通常也仍然有一些光亮（月亮、远处的光），所以物体几乎永远不会是完全黑暗的。为了模拟这个，我们会使用一个环境光照常量，它永远会给物体一些颜色。
- **漫反射光照(Diffuse Lighting)**：模拟光源对物体的方向性影响(Directional Impact)。它是冯氏光照模型中视觉上最显著的分量。物体的某一部分越是正对着光源，它就会越亮。
- **镜面光照(Specular Lighting)**：模拟有光泽物体上面出现的亮点。镜面光照的颜色相比于物体的颜色会更倾向于光的颜色。

环境光照

光通常都不是来自于同一个光源，而是来自于周围分散的很多光源，即使它们可能并不是那么显而易见。光的一个属性是，它可以向很多方向发散并反弹，从而能够到达不是非常直接临近的点。所以，光能够在其它的表面上**反射**，对一个物体产生间接的影响。考虑到这种情况的算法叫做全局照明(Global Illumination)算法，但是这种算法既开销高昂又极其复杂。

使用一个简化的全局照明模型，即环境光照。使用一个很小的常量（光照）颜色，添加到物体片段的最终颜色中，这样子的话即便场景中没有直接的光源也能看起来存在有一些发散的光。

把环境光照添加到场景里非常简单。我们用光的颜色乘以一个很小的常量环境因子，再乘以物体的颜色，然后将最终结果作为片段的颜色。

漫反射光照

环境光照本身不能提供很好的结果，但是漫反射光照就能开始对物体产生显著的视觉影响了。漫反射光照使物体上与光线方向越接近的片段能从光源处获得更多的亮度。

光源所发出的光线落在物体的一个片段上，需要测量这个光线是以什么角度接触到这个片段的。如果光线垂直于物体表面，这束光对物体的影响会最大化。为了测量光线和片段的角度，使用一个叫做法向量(Normal Vector)的东西，它是垂直于片段表面的一个向量。

计算漫反射光照需要什么？

- 法向量：一个垂直于顶点表面的向量。
- 定向的光线：作为光源的位置与片段的位置之间向量差的方向向量。为了计算这个光线，我们需要光的位置向量和片段的位置向量。

法向量

法向量是一个垂直于顶点表面的（单位）向量。由于顶点本身并没有表面（它只是空间中一个独立的点），利用它周围的顶点来计算出这个顶点的表面。使用一个小技巧，使用叉乘对立方体所有的顶点计算法向量，但是由于$3D$立方体不是一个复杂的形状，所以我们可以简单地把法线数据手工添加到顶点数据中。

首先，法向量只是一个方向向量，不能表达空间中的特定位置。同时，法向量没有齐次坐标（顶点位置中的w分量）。这意味着，位移不应该影响到法向量。因此，如果我们打算把法向量乘以一个模型矩阵，我们就要从矩阵中移除位移部分，只选用模型矩阵左上角3×3的矩阵。对于法向量，只希望对它实施缩放和旋转变换。

其次，如果模型矩阵执行了不等比缩放，顶点的改变会导致法向量不再垂直于表面了。因此，不能用这样的模型矩阵来变换法向量。

等比缩放不会破坏法线，因为法线的方向没被改变，仅仅改变了法线的长度，而这很容易通过标准化来修复。每当应用一个不等比缩放时，法向量就不会再垂直于对应的表面了，这样光照就会被破坏。修复这个行为的诀窍是使用一个为法向量专门定制的模型矩阵。这个矩阵称之为法线矩阵($NormalMatrix$)，它使用了一些线性代数的操作来移除对法向量错误缩放的影响。

镜面光照

和漫反射光照一样，镜面光照也决定于光的方向向量和物体的法向量，但是它也决定于观察方向，例如用户是从什么方向看向这个片段的。镜面光照决定于表面的反射特性。如果把物体表面设想为一面镜子，那么镜面光照最强的地方就是看到表面上反射光的地方。

通过根据法向量翻折入射光的方向来计算反射向量。然后计算反射向量与观察方向的角度差，它们之间夹角越小，镜面光的作用就越大。由此产生的效果就是，看向在入射光在表面的反射方向时，会看到一点高光。

观察向量是计算镜面光照时需要的一个额外变量，可以使用观察者的世界空间位置和片段的位置来计算它。之后计算出镜面光照强度，用它乘以光源的颜色，并将它与环境光照和漫反射光照部分加和。

裁剪多边形

通过变换序列将对象局部坐标系中的一个点转换成在显示器上的投影。这样的一个投影是经过虚拟照相机获得的。这个现察管道的一个非常重要的方面先前并没有考虑到，那就是裁剪。在光线跟踪中这不是什么问题， 因为没有主光线会跑到视景体的外面，这个视景体区域由四个分别通过视平面一条边的平面和前后裁剪平面所包围。然而，在新的方法中，也就是当多边形投影到视平面上时，必须明确地执行对多边形的裁剪，以便只虑考虑位于视景体里面的对象或对象的一部分。

理论

多边形是一个点序列$[p_0, p_1, \ldots\ldots, p_{n-1}]$,$p_n = p_0$,相邻点用直线段相连，$p_{n-1}$与$p_0$首尾相连。$p_i$称为多边形的顶点，连接顶点间的直线段称为边。如果一个多边形除了公共顶点处处彼此不相交，那么它称为简单多边形。如果个多边形的每一个内角都小与180°，也就是说两条边在多边形里侧所夹的角度小于180°，那么它称为凸多边形。

为了维护基本类型的一致性，多边形的裁剪结果也应该是一个多边形（或者什么也没有）。

现在有能力渲染由多边形所构造的层次场景， 允许对于任何照相机参数设置这个过程比光线跟踪中的最初起点要快得多， 因为是将多边形投影到视平面上而不是执行数以百万计的光线-多边形相交测试。当然， 也失去了光线跟踪容易处理的对象间交又反射。可以为了速度牺牲真实感， 然而不能够牺牲掉光线跟踪能轻易处理的其他东西——可见性。如果在处理整个场景图时只是简单地对所遇到的每个多边形执行渲染的话， 结果将会是非常错误的。必须考虑到可见性关系， 就像通过当前照相机装置去"看"一样。找到一个方式排序多边形以便能根据可见性对它们进行正确的渲染——那些" 远处" 的多边形先于近处的多边形被渲染， 这样比较近的多边形就会覆盖远处的那些多边形。可以使用一个特别的数据结构" 二叉空间分割树" 完美实现。

Sutherland-Hodgman算法（二维）

给定一个一个凸多边形和一个凸裁剪区域，输入是以顺时针为顺序的多边形顶点的形式。

无限延伸裁剪边以创建边界，并使用此裁剪边剪切所有顶点。生成的新顶点列表以顺时针方式传递到裁剪多边形的下一条边，直到所有边都被使用。对于给定多边形的任何给定边与当前剪裁边 e，有四种可能的情况：

1. 两个顶点都在里面：只有第二个顶点被添加到输出列表；
2. 第一个顶点在外面，第二个在里面：与裁剪边界的交点和第二个顶点都被添加到输出列表中
3. 第一个顶点在里面，第二个在外面：只有与裁剪边的交点被添加到输出列表中；
4. 两个顶点都在外面：没有顶点添加到输出列表中。

在实现算法之前需要讨论两个子问题：

- 确定一个点是在裁剪多边形的内部还是外部。如果多边形的顶点按顺时针给出，则裁剪边右侧的所有点都在该多边形内。

  可以使用以下方法计算：

  给定一条线段始于(x_1, y_1)结束于(x_2, y_2)，计算
  $p = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$，

  如果$p < 0$，点在线段右边；

  如果$p = 0$，点在线段上；

  如果$p > 0$，点在线段左边.
- 计算待裁剪边(如上图三角形)与裁剪边(如上图正方形)的交点。

  可以使用以下公式计算它们的交点：

$$(P_x, P_y) = (\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)})$$

Weiler-Atherton算法——裁剪多边形

$Sutherland-Hodgeman$算法解决了裁剪窗口为凸多边形窗口的问题，但一些应用需要涉及任意多边形窗口（含凹多边形窗口）的裁剪。$Weiler-Atherton$多边形裁剪算法正是满足这种要求的算法。

算法描述

在算法中，裁剪窗口、被裁剪多边形可以是任意多边形：凸的、凹的（内角大于180°）、甚至是带有内环的（子区）。裁剪窗口和被裁剪多边形处于完全对等的地位，这里我们称：

1. 被裁剪多边形为主多边形，记为A；
2. 裁剪窗口为裁剪多边形，记为B。

主多边形A和裁剪多边形B的边界将整个二维平面分成了四个区域：

1. $A \cap B$（交：属于A且属于B）；

2. $A-B$（差：属于A不属于B）；
3. $B-A$（差：属于B不属于A）；
4. $A \cup B$(并：属于A或属于B，取反；即：不属于A且不属于B)。

内裁剪即通常意义上的裁剪，取图元位于窗口之内的部分，结果为$A \cap B$。外裁剪取图元位于窗口之外的部分，结果为$A-B$。

不难发现裁剪结果区域的边界由被裁剪多边形的部分边界和裁剪窗口的部分边界两部分构成，并且在交点处边界发生交替，即由被裁剪多边形的边界转至裁剪窗口的边界，或者反之。由于多边形构成一个封闭的区域，所以，如果被裁剪多边形和裁剪窗口有交点，则交点成对出现。这些交点分成两类：

1. 一类称"入"点，即被裁剪多边形由此点进入裁剪窗口。
2. 一类称"出"点，即被裁剪多边形由此点离开裁剪窗口。

算法思想

假设被裁剪多边形和裁剪窗口的顶点序列都按顺时针方向排列。当两个多边形相交时，交点必然成对出现，其中一个是从被裁剪多边形进入裁剪窗口的交点，称为"入点"，另一个是从被裁剪多边形离开裁剪窗口的交点，称为"出点"。

算法从被裁剪多边形的一个入点开始，碰到入点，沿着被裁剪多边形按顺时针方向搜集顶点序列；

而当遇到出点时，则沿着裁剪窗口按顺时针方向搜集顶点序列。

按上述规则，如此交替地沿着两个多边形的边线行进，直到回到起始点。这时，收集到的全部顶点序列就是裁剪所得的一个多边形。

由于可能存在分裂的多边形，因此算法要考虑：将搜集过的入点的入点记号删去，以免重复跟踪。将所有的入点搜集完毕后算法结束。

算法步骤

1. 顺时针输入被裁剪多边形顶点序列Ⅰ放入数组1中。
2. 顺时针输入裁剪窗口顶点序列Ⅱ放入数组2中。
3. 求出被裁剪多边形和裁剪窗口相交的所有交点，并给每个交点打上"入"、"出"标记。然后将交点按顺序插入序列Ⅰ得到新的顶点序列Ⅲ，并放入数组3中；同样也将交点按顺序插入序列Ⅱ得到新的顶点序列Ⅳ，放入数组4中；
4. 初始化输出数组Q，令数组Q为空。接着从数组3中寻找"入"点。如果"入"点没找到，程序结束。
5. 如果找到"入"点，则将"入"点放入S中暂存。
6. 将"入"点录入到输出数组Q中。并从数组3中将该"入"点的"入"点标记删去。

7. 沿数组3顺序取顶点：如果顶点不是“出点”，则将顶点录入到输出数组Q中，流程转第7步。否则，流程转第8步。
8. 沿数组4顺序取顶点：如果顶点不是“入点”，则将顶点录入到输出数组Q中，流程转第8步。否则，流程转第9步。
9. 如果顶点不等于起始点S，流程转第6步，继续跟踪数组3。否则，将数组Q输出；流程转第4步，寻找可能存在的分裂多边形。算法在第4步：满足“入”点没找到的条件时，算法结束。

算法特点

1. 裁剪窗口可以是矩形、任意凸多边形、任意凹多边形。
2. 可实现被裁剪多边形相对裁剪窗口的内裁或外裁，即保留窗口内的图形或保留窗口外的图形，因此在三维消隐中可以用来处理物体表面间的相互遮挡关系。
3. 裁剪思想新颖，方法简洁，裁剪一次完成，与裁剪窗口的边数无关。

算法小结

内裁算法，即保留裁剪窗口内的图形。而外裁算法（保留裁剪窗口外的图形）同内裁算法差不多。

外裁算法与内裁算法不同的是：

1. 从被裁剪多边形的一个“出点”开始，碰到出点，沿着被裁剪多边形按顺时针方向搜集顶点序列；
2. 而当遇到“入点”时，则沿着裁剪窗口按逆时针方向搜集顶点序列。

按上述规则，如此交替地沿着两个多边形的边线行进，直到回到起始点为止。这时，收集到的全部顶点序列就是裁剪所得的一个多边形。

由于可能存在分裂的多边形，因此算法要考虑：将搜集过的“出点”的出点记号删去，以免重复跟踪。将所有的出点搜集完毕后算法结束。

$Weiler-Atherton$算法的的设计思想很巧妙，裁剪是一次完成，不像$Sutherland-Hodgman$多边形裁剪算法，每次只对裁剪窗口的一条边界及其延长线进行裁剪，如裁剪窗口有n条边，则要调用n次$Sutherland-Hodgman$算法后才能最后得出裁剪结果。

但$Weiler-Atherton$算法的编程实现比$Sutherland-Hodgman$算法稍难，主要难在入、出点的查寻以及跨数组搜索上。

在三维中裁剪多边形

$Sutherland-Hodgman$算法推广到三维是很容易的，然而在这么做之前必须准确识别出三维裁剪边界。在规范观察空间中，视景体是一个规则金字塔由前裁剪平面和后裁剪平面切割所成的形状。在投影空间中视景体是一个立方体。

通常三维的裁剪算法与二维的裁剪算法有相同的结构，细微的变化是：

1. 裁剪边界是平面，共有六个这样的平面。
2. 线段或边与这些平面之间的相交计算显然是不同的

投影空间中的裁剪

投影空间中的视景体的定义：

$$-1 \leqslant x \leqslant 1$$
$$-1 \leqslant y \leqslant 1$$
$$0 \leqslant z \leqslant 1$$

$Sutherland-Hodgman$多边形裁剪算法是针对每个边界去裁剪整个多边形，然后将所得到的多边形传递给下一个边界。这需要两个操作：

1. 确定多边形和边界之间的关系：
   a. p_0在内部且p_1在内部
   b. p_0在内部且p_1在外部
   c. p_0在外部且p_1在内部
   d. p_0在外部且p_1在外部
2. 求出多边形边和边界之间的交。考虑x=1（右）裁剪平面，（1）中的四个条件将是：
   a. $x_0 \leqslant 1$ 和 $x_1 \leqslant 1$
   b. $x_0 \leqslant 1$ 和 $x_1 \geqslant 1$
   c. $x_0 \geqslant 1$ 和 $x_1 \leqslant 1$
   d. $x_0 \geqslant 1$ 和 $x_1 \geqslant 1$

对于其他几个平面的情况是相似的。

可见性确定

此前我们一直在讨论如何在对象特别多的情况下提升渲染效率。首先，只有落在视景体中的对象才是看得见的，此前已经讨论过使用裁剪来达到这一点。更困难的问题是对象间的可见性问题。从任何观点来看，都会存在一些对象遮挡另一些对象的情况。当移动头部时，对象遮挡的部分就会发生变化，同时会有新的对象进入视野。如果没有考虑可见性，将会导致场景渲染的严重错误。确定图像每个区域中每个对象的可见部分问题称为可见表面确定问题。

背面删除

背面删除可以使用在场景中由一组平面多边形表示，而所有对象都是封闭多面体的情况下。多面体的每个面要么是朝向照相机，要么是背向照相机的。这些背向照相机的面在进一步计算中不再考虑。很容易计算一个面是否背向照相机。

对于单个的凸对象，背面删除足以解决可见性问题。根据定义，对于一个凸的对象，当任意给定光线穿过它时，顶多会产生两个交点（当光线与对象的边或面相切时只有一个交点）。

很容易证明对于两点的情形，最靠近的一个总是前向的，而另一个则是背向的，由此可以测试多边形是否是背向的，若不是则可以以任意顺序渲染它。

列表优先权算法

列表优先权算法的核心思想是相对于COP点"由远及近"地对多边形排序。

排序的含义：假设环境中有n个多边形，我们想要一个序列$P_1 \ldots\ldots P_n$，保证任何多边形P_i不会遮挡$P_{i+1} \ldots\ldots P_n$中的任意一个多边形。直观上我们可以认为多边形P_i比$P_{i+1} \ldots\ldots P_n$离COP更远一点。

在投影空间排序

多边形一经转换到投影空间，求出每个多边形中心到COP的距离，然后在一维中根据这个距离进行排序。这就是所谓的z排序。

深度排序

假设有两个多边形P和Q，如果下列测试中任何一项成功的话，P可以先于Q渲染。

1. Q的Z范围完全位于P的Z范围之前。
2. Q的Y范围与P的Y范围不重叠。
3. Q的X范围与P的X范围不重叠。
4. P上所有点相对于平面Q来说与视点位于相反的两侧。
5. Q上所有点相对于平面P来说与视点位于相同的一侧。
6. P和Q在XY平面上的投影不重叠。

二叉空间分隔树

定义

BSP（$BinarySpacePartition$）表示二叉空间分割。使用这种方法可以使我们在运行时使用一个预先计算好的树来得到多边形从后向前的列表，它的复杂度为$O(n)$。它的基本思想是基于这样一个事实：任何平面都可以将空间分割成两个半空间。所有位于这个平面的一侧的点定义了一个半空间，位于另一侧的点定义了另一个半空间。此外，如果在任何半空间中有一个平面，它会进一步将此半空间分割为更小的两个子空间。可以使用多边形列表将这一过程一直进行下去，将子空间分割得越来越小，直到构造成一个二叉树。在这个树中，一个进行分割的多边形被存储在树的节点，所有位于子空间中的多边形都在相应的子树上。

构造

检查观察者的位置和位于树顶部的多边形之间的关系。很明显，当这个多边形将空间分割为两个部分时，观察者必须位于其中的一个半空间中。当然，位于同一半空间中的多边形要比另一半空间中的多边形离观察者更近。基于这样的事实，如果首先将较远处半空间中的多边形放置在最终的列表中，然后放置根多边形，再放置与观察者在同一半空间中的多边形，这样就很容易得到多边形从后向前的顺序了。我们对每一个子树都重复同样的过程，在每一个级别中选择相应的顺序，最终，就会得到正确的多边型的顺序。这一算法的一个优点就是无论观察者位于场景中的什么位置，无论观察者的朝向如何，它都可以很好的进行工作。如果预先为一个多边形模型计算一个BSP树，那么在运行时，只需要根据观察者的位置调用这个树，执行树遍历过程，就可以产生用于隐面消除的画家算法所需的从后向前的多边形顺序。

在这个算法中，在树的每一个节点处所作的判定，都依赖于观察者位于该节点多边形所产生的哪一个半空间中。利用所讨论过的事实，可以看到，如果将观察者位置的坐标代入给定多边形的平面方程式中，结果数值的符号如果是正号，就表示观察者位于该多边形的法向量所指向的半空间中，负号表示位于另一个半空间中，0值表示他位于这个多边形

所在的平面上。最后一种情况，对于遍历整个BSP树的意图来说，意味着半空间在屏幕上的投影不相交，并且可以在这一阶段的遍历过程中选择任何子树的顺序。

相同的，计算也要在预先计算一个BSP树时使用到。我们需要决定不同的多边形应该被放置在哪个子树中。从可实现性的观点出发，预先计算一个BSP树的过程可以被描述成下面的形式：对多边形集合，我们选择一个多边形。进一步计算该多边形的平面方程式。对剩余的多边形，我们用所说的方程式检查它们的所有顶点。如果所有顶点都是负值，那么多边形就放置在一个子树中；如果都是正值，那么多边形就放置在另一个子树中；如果结果有正有负，那么多边形就被分为两部分，分别放置在两个子树中。一旦我们将所有多边形分配到了正确的半空间中，就可以对子树进一步调用同样的方法来进行处理，直到当前的子表只包含一个多边形为止。

一个多边形被任何平面所分割的问题可以当作一个裁剪问题来处理。解决这一问题的算法只与我们在裁剪问题时所讨论的算法又很小的差别。唯一明显的差别就是在二进制搜索边缘裁剪过程中，将使用分割多边形的平面方程式来找到边的中点的位置。

问题

考查了树的创建和遍历之后，仍然有一些问题需要解决。当构建一个树时，我们可以选择剩余多边形中的任何一个来分割空间。选择不同的多边形会导致不同的树的结构。因此，就应该考虑选择哪个多边形有助于算法的效率。有些多边形会导致剩余多边形更多的分割。每一个多边形在通过渲染管道时都有一定的系统消耗，因此多边形越少，性能就越好。可以利用判据来选择有较少分割的多边形。

使用判据来平衡BSP树，并不需要每一级中的子树中的多边形的数量都相同，因为它不会影响运行时间。树的遍历总是假设至少每次都取一个多边形，因此平衡不会影响性能——仍然不得不每次取每一个多边形。另一方面，一个平衡的树可以以较少的迭代调用来执行遍历。可以将平衡作为第二判据来使用。总之，使用BSP树来进行从后向前排序的最大优点就是算法运行的复杂性较低 。这种方法也解决了多边形的多重交叠和多边形穿越问题。但是，通过使用一个预计算结构，已经失去了一定的灵活性。如果多边形的排列在运行时发生了改变， BSP树就必须发生相应的改变。由于计算量非常巨大，不能这样做。

应该注意，只有当多边形经历不同的变换设置时树才会受到影响。如果所有的多边形都使用同样的变换，那么分割仍然是正确的，树也将不会受到影响。这样，场景中的一个动态物体，它在世界中移动或者是旋转，仍然可以使用同样的BSP树。如果物体中的一部分相对于其他物体发生了移动，那就要使用其它的算法了。

结语

本学期选修的课程中，计算机图形学是最为硬核的一门——知识点多，内容复杂，全英文教学。课程难度很大程度来源于本课程对数学要求很高，要能熟练地掌握线性代数和微积分。其中四元数部分是令我最伤脑筋的，前前后后几个星期才弄明白其整个发展框架——从复数开始，到四元数的诞生，到四元数的应用。同时书中含有大量抽象的数学式子和很多从物理中借鉴的概念，起初只能从感性上感觉个大概。计算机图形学在历史上出现得较晚，使得基本是很基础的知识，相较于其他学科在历史上的时间，都算得上是很前沿的知识了。课程选用的两本教材刚好就是在图形学刚刚发展的几十年内出版的头号教材，其详尽的介绍了图形学很多概念的缘由，详细地读了一遍下来加以读书笔记的整理，真正地吸收了图形学思想的精髓。而且课本上习得的知识，很块就在OpenGL项目上得到展现，这种学习和实践相结合的方式，使得学习这门繁杂的课程增添不少乐趣。相较于现在很多开发者简单地调用现成的模型，使用现成的着色器，光照，照相机，使用成熟的框架开发，我觉得深入底层的学习十分有必要，这样才做到了知其然更知其所以然。从某种层次上说这就是科班学生需要掌握的必备技能。只有一切落实到了底层，才能在这个快速发展的行业里站稳脚跟，跟上时代的脚步。