计算机图形学B

7. 隐藏面消除

消隐——表现空间遮挡关系，提供真实感。隐藏线隐藏面 v.s. 可见面可见线

表面模型——面消隐；线框模型——线消隐

按照处理单元分类——以像素为单元，以物体为单元；按照算法实现思路分类——基于区域（from-region），基于视点（from-point）

以像素为处理单元的消隐算法——遍历每个像素，确定距视点的最近的物体，用该物体的颜色显示像素

以物体为处理单元的消隐算法——遍历每个物体，将该物体与其他物体比较，确定可见部分，显示物体表面的可见部分

基于区域的消隐算法——适合于实现设计并划分好的固定场景。预先计算出特定区域内可能看见的所有物体，它们组成一个潜在可视集合PVS，程序运行时根据视点所处位置进行PVS索引

基于视点的消隐算法——适合于大规模场景。根据当前视点，实时地计算PVS

提高消隐算法效率的方法

利用连贯性——物体、面、区域、扫描线、深度等均具有连贯性

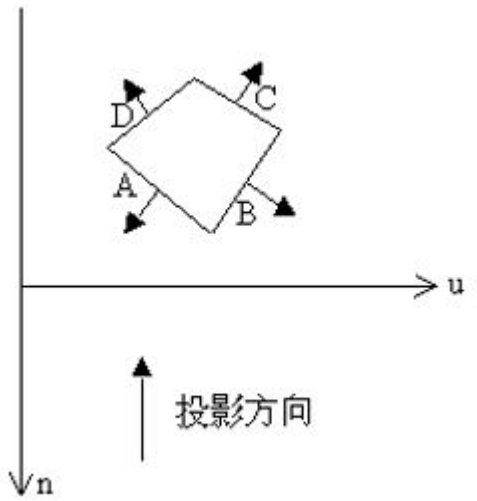
将透视投影变换为平行投影——平行投影下的消隐更简单

包围盒技术——利用简单形体保卫目标物体，避免盲目地求交

KD-Tree——加速碰撞测试，利用层次化结构缩小碰撞测试的范围；加速静态物体拾取操作；加速可见性算法测试。

KD-Tree高效建树策略——保持平衡，控制树高，内存单元循环利用动态回收。

背面剔除——后向面不可见，直接剔除。面的法向与投影方向点乘大于0说明是背面



消隐的基本问题——排序

整体排序——画家算法

点排序——Z-Buffer算法，光线投射算法

区间排序——扫描线算法

区域排序——区域子分算法

画家算法——建立深度优先级表。对场景中的多边形按照深度排序，形成深度优先级表， 按照从远到近的顺序显示多边形。

```
1．将所有多边形存入一个线性表L中
2．如果L中仅有一个多边形，则直接绘制；否则根据多边形的最小深度zmin预排序，记表首为P，记L-{P}中的任意一个为Q
3．判断P和Q的关系
        若对所有Q，zmax(P)<zmin(Q)，则P距离观察点最远，并且不会遮挡其他多边形，那么绘制P，L=L-{P}，返回2
        若存在某个Q，使得zmax(P)>zmin(Q)，则进行4的进一步的判断，若都能通过判断，则绘制P，L=L-{P}，返回2；否则若已经交换过的P和Q再次交换，说明无法正确排序，返回报错
4．判断P与Q的遮挡关系：
        判断投影平面上P'和Q'的包围盒不相交，说明P不会遮挡Q，排序无误
        否则，判断P的所有顶点位于Q的平面的不可见一侧（矢量点乘判符号），说明P不会遮挡Q，排序无误
        否则，判断Q的所有顶点位于P的平面的可见一侧（矢量点乘判符号），说明P不会遮挡Q，排序无误
        否则，对投影P'和Q'求交，若有交，则在相交区域内取点判断深度，如果P的深度小，说明排序无误；否则交换P和Q，重新判断新的P（原来的Q）与Q的关系
```

画家算法的难点在于空间多边形的排序算法计算量大，并且对于循环遮挡的多边形和交叉的多边形很难处理。

Z-Buffer算法——初始化时将Z缓冲器的每个单元深度置为最小，将帧缓冲器的每个单元颜色置为背景色；对每个多边形中的每个像素，计算其在投影区域内的深度d，如果d>Z缓冲器中的值，则Z缓冲器该单元深度置为d，帧缓冲器该单元颜色置为多边形颜色值

优点：算法简单稳定，便于硬件加速，不需要整个场景的集合数据；缺点：需要Z缓冲器，计算深度的次数=多边形个数*多边形平均占据的像素个数

扫描线Z-Buffer算法——对于Z-Buffer算法的改进算法

改进一：将窗口分割为扫描线，对于每一个多边形，求出多边形在投影平面上的投影与当前扫描线的相交区间，之后计算该区间内的像素即可

改进二：采用多边形分类表，活化多边形表来避免盲目求交

改进三：采用活化边表避免盲目求交

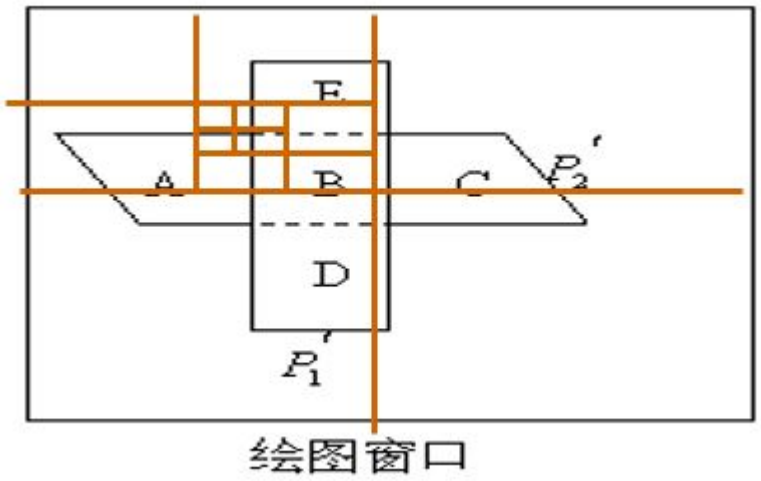
改进四：利用连贯性计算深度，插值法

扫描线消隐算法——不需要Z-Buffer；在一条扫描线上，以区间为单位确定多边形的可见性

数据结构——多边形分类表，活化多边形表，边的分类表，活化边表

```
1．建立多边形分类表，建立每一个多边形的边的分类表
2．Z-Buufer初始化，帧缓存初始化，建立活化边表
3．遍历扫描线，对活化边表中的每个区间，计算覆盖该区间的所有多边形的深度，记最前面的多边形为P，用P的颜色显示这一个区间
```

区域子分算法——分割窗口直至窗口足够简单；利用区域的连贯性提高排序效率。



窗口足够简单——窗口与多边形分离，窗口包含多边形，窗口与多边形相交，窗口包围多边形。

光线投射算法——沿投影线，遇到的第一个物体即为最靠前的物体。难点在于光线与物体表面求交。

```
1．对于每一个像素点，从视点发出经过该像素点的投影线
2．将投影线与多边形求交
3．若有交点，则取最近的交点的多边形的颜色显示这一像素；否则用背景色显示
```

AABB快速计算方法——用两个顶点即可确定包围盒与视域体的相交情况。相对于检测平面**π**，正侧点p的有向距离最大，负侧点n的有向距离最小。若p在**π**的负方向，那么AABB完全在视域体之外；否则检测n，若n在**π**的负方向，则AABB与视域体相交，否则完全包含在其内部。

8. 纹理贴图和颜色模型

纹理贴图——用图像、函数或其他数据源来改变表面在每一处的外观的过程。可以节省大量造型工作量，节省内存，加快绘制。

几何细节纹理贴图，镜面高光纹理贴图，凹凸纹理...

广义纹理贴图——一种有效的改变物体表面属性的方法，表面属性包括颜色，高光，凹凸，反射，透明度等。

投影函数——将三维空间点转化为纹理坐标

on the fly（即时）投影：将纹理坐标记录在顶点上，可以实时绘制，也可以实时计算。优点：纹理坐标不需要送到图形卡，节约传输带宽。

纹理坐标——(u, v, w)形式，w表示投影方向的深度；(s, t, r, q)形式，q为齐次坐标，可以用于表示聚光灯效果。

对应函数——将参数空间的值转化为纹理空间的位置

第一种对应函数——用API选择一部分纹理（子纹理）进行纹理映射

第二种对应函数——利用矩阵变换进行纹理映射。

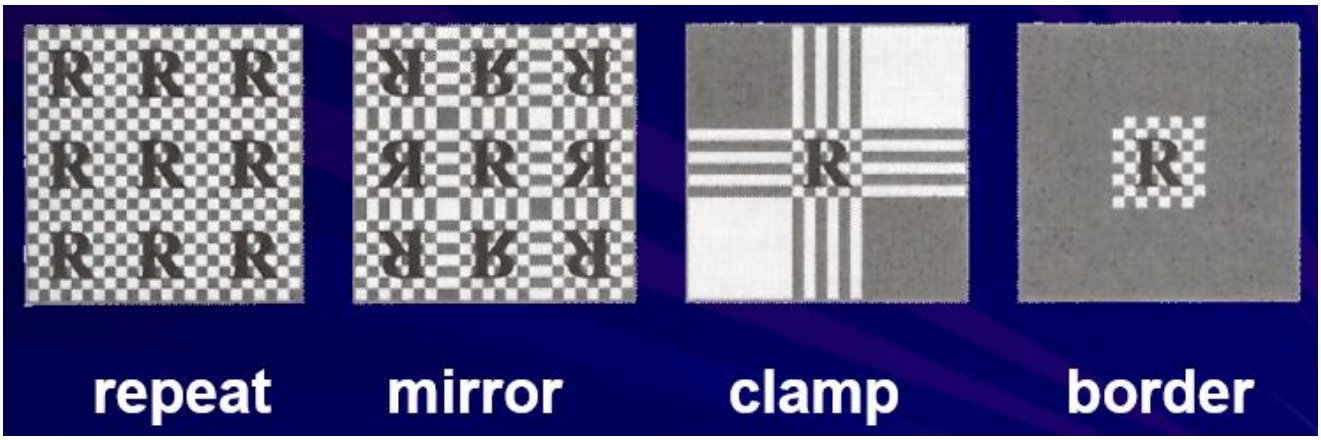
超出[0,1)纹理空间的解决方法（OpenGL中称为wrapping mode）

repeat / tile：纹理图像在表面重复

mirror：纹理图像在表面重复且每隔一副翻转一次，保证边界连续

clamp-to-edge：把[0,1)范围之外阶段，截断到[0,1)内半个纹素

clamp-to-border：[0,1)之外用单独定义的边界颜色或把纹理的边作为边界，截断到[0,1)外的半个纹素



三维纹理——避免二维纹理中可能出现的畸变和接缝问题。

合成方法：噪声函数。由于计算噪声函数的代价较大，通常是预先计算好空间格点的值，再进行插值。

纹理贴合方式——Combine Function / Texture Blending Operations

Replace：将原来表面颜色替换为纹理颜色

Decal：印花，若纹理包含Alpha值，则用它与表面颜色进行混合

Modulate：将表面颜色与纹理颜色相乘

纹理图像大小——在硬件图形加速卡中，纹理图大小通常为$2^m \times 2^n$的纹素

纹理放大——投影得到的像素数目比原始纹理图大，需要把纹理图像放大

最近邻域滤波——每个像素选择最近的纹素。产生锯齿，效果不好。

双线性插值——每个像素选择最近的四个纹素进行双线性插值。质量好。

纹理缩小——投影得到的像素数目比原始纹理图小，需要把纹理图像缩小

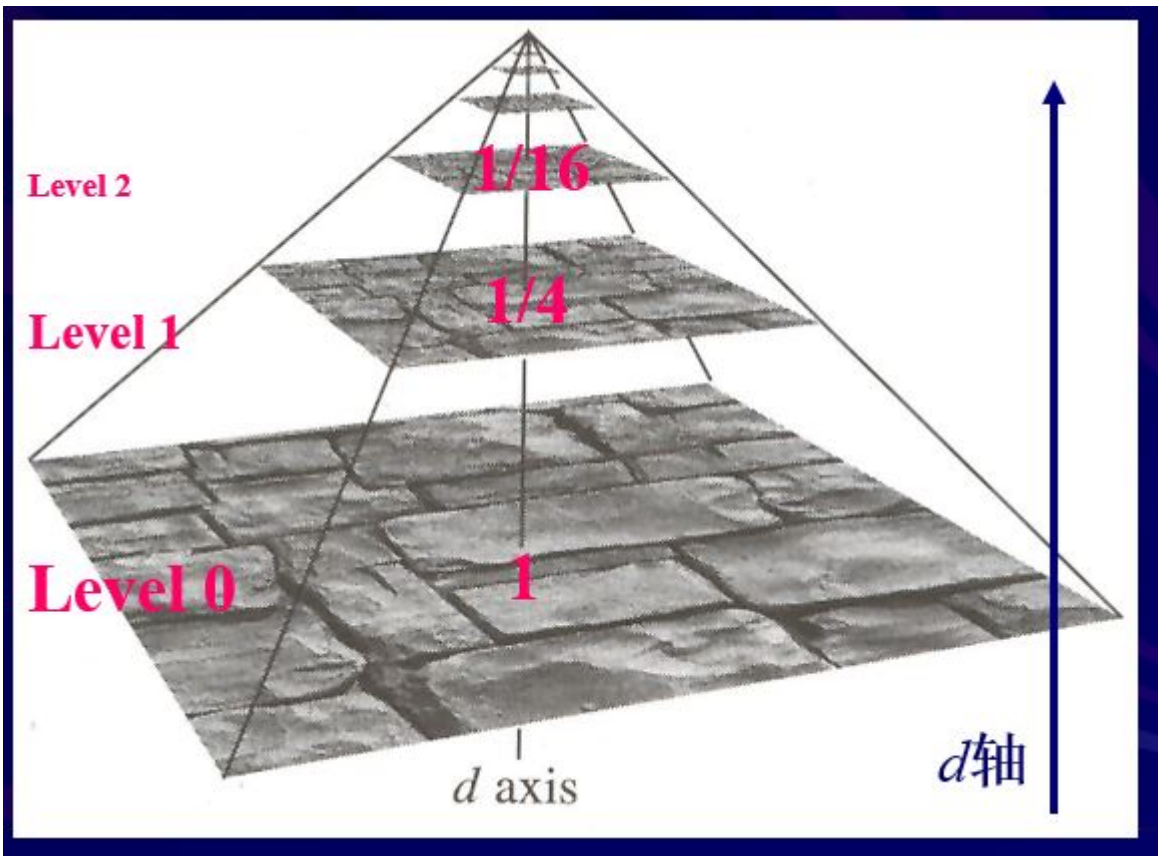
由于纹理图像缩小，多个纹素覆盖一个像素，为了得到正确的像素颜色，应该综合考虑所有纹素。

最近邻域法——选择像素中心的可见纹素，但会引起严重的走样

双线性插值——效果比最近邻域法烧好，但也会引起严重的走样

纹理反走样算法——对纹理预处理，建立多个纹素覆盖单个像素的快速逼近计算的数据结构。目的是一个采样点可以检索出一个或多个纹素的效果。

Mipmapping——最受欢迎的纹理反走样方法。在绘制之前，把原始纹理图像重复滤波，生成多幅小的纹理图像。常用的滤波方法是2x2 mean pooling，下采样直至子纹理的任意维只有一个纹素。



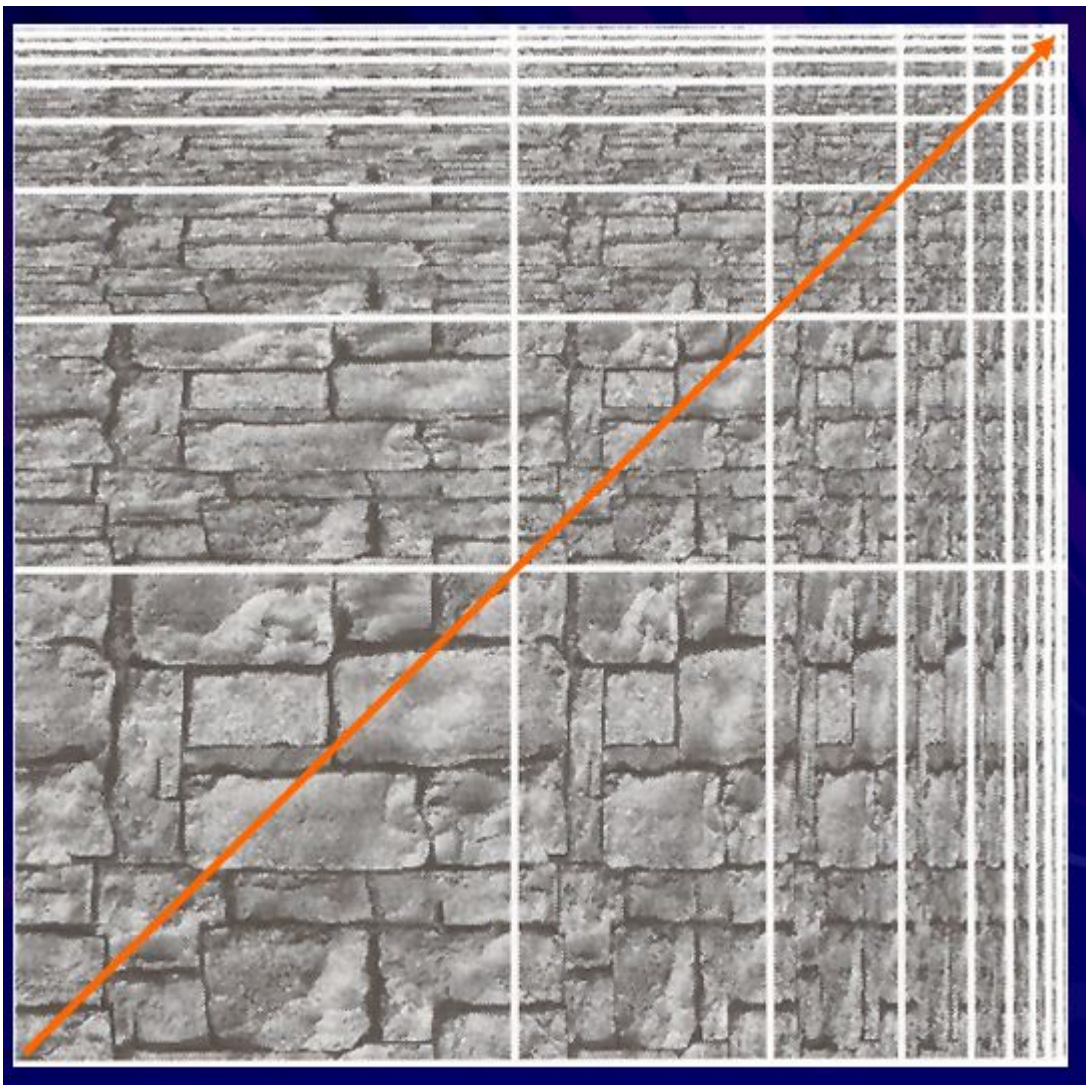
构造高质量MIipmap的重要元素：好的滤波（通常采用Mean-pooling，但Gaussian，Lanczos等更好）；Gamma矫正（将纹理图像的亮度恢复到原图像）

检索Mipmap结构——三线性插值法。首先需要找出大致多少纹素影响到某个像素，需要计算出d（层次细节），d的意义类似于“纹理层”，但其可以不是整数，可以形象地理解为在金字塔结构中到底层的距离。首先对纹理层$\lfloor d \rfloor$和$\lceil d \rceil$进行采样；之后用(u, v)在这两个纹理层上进行双线性插值，最后用$d - \lfloor d \rfloor$对这两个采样值进行线性插值。

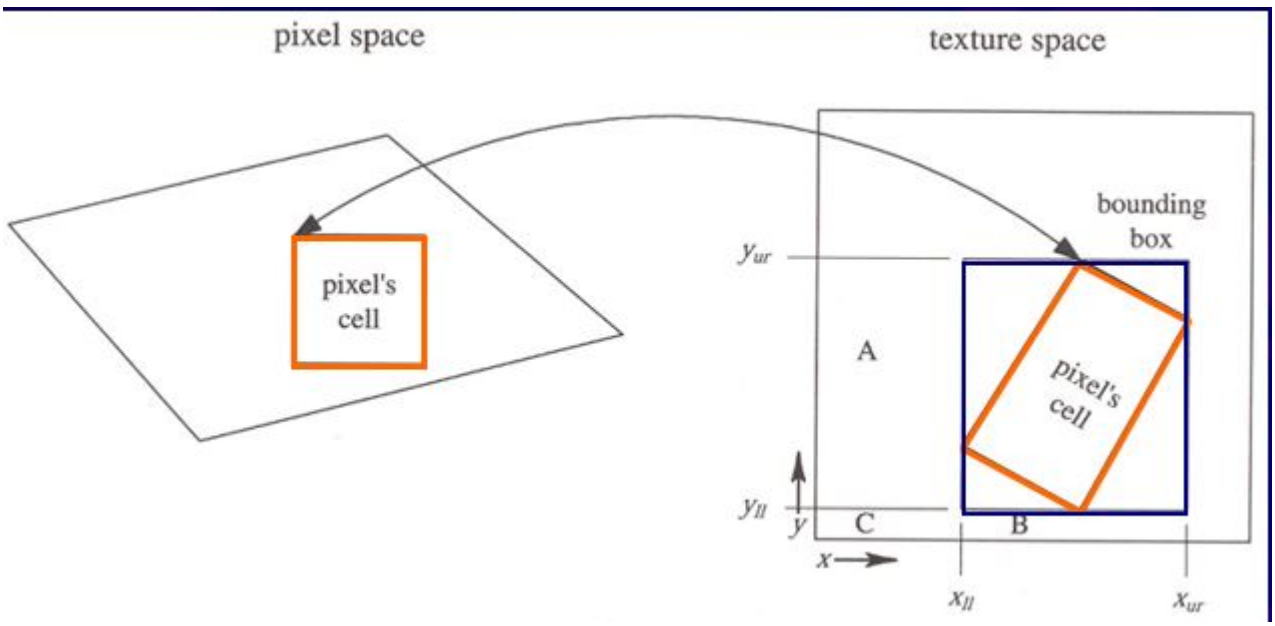
Mipmap的LOD Bias参数（Lecel of Detail Bias）——作为偏移量加在d上，用来影响纹理的精锐度。如果在金字塔上沿着d坐标上移，那么纹理看起来更模糊；如果下移，就会锐化。根据情况选择LOD Bias参数。

Mipmap的缺陷——过于模糊现象，如果像素在一个方向覆盖很多纹素，而另一个方向覆盖很少的纹素，但Mipmap中不能得到长方形纹理区域，正方形区域会使得结果过于模糊。

Ripmapping——HP公司提出的解决过于模糊的方法，允许长方形下采样。Ripmap子纹理中对角线就是Mipmap子纹理。在插值是需要使用四个子纹理来进行。



求和面积表——首先建立与纹理大小相同的数组，但包含的颜色的位数精度更高；数组的每个位置计算(0,0)带该位置的矩形区域内纹素的和和该位置的纹素；纹理贴图时，像素单元在纹理空间的投影是由一矩形构成的，通过读取求和面积表来决定该矩形的平均颜色，并作为像素单元的纹理颜色。

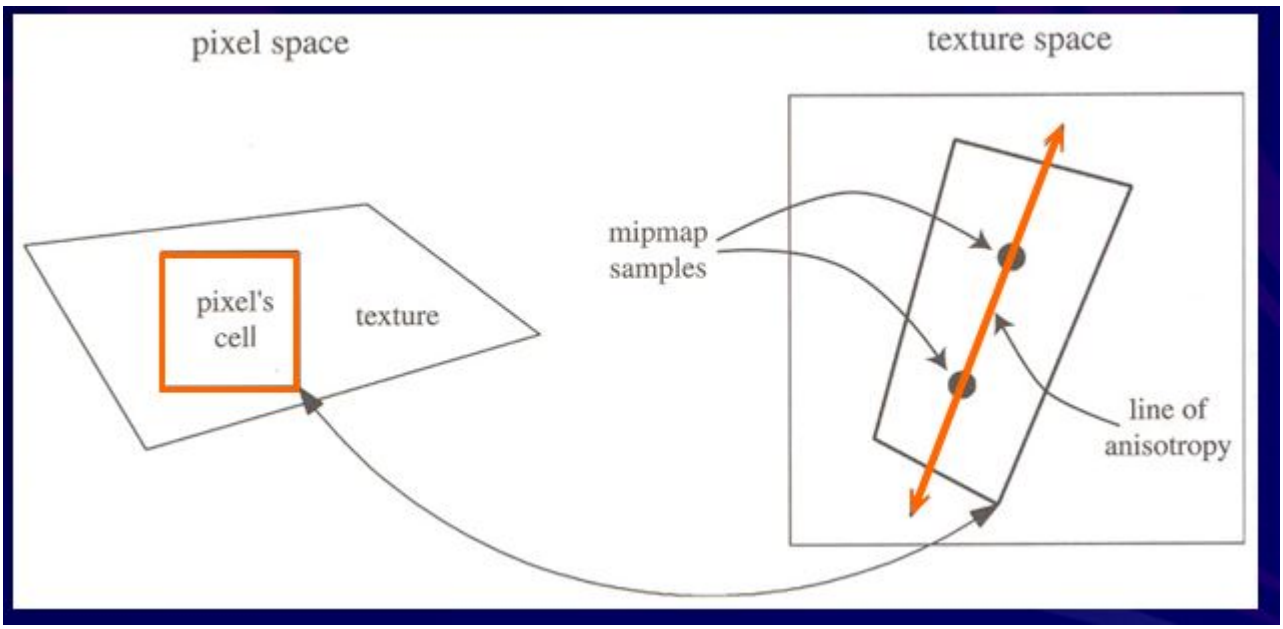


计算时用包围盒来代替真实的像素覆盖的纹素。

$$c = \frac{s[x_{ur,y_{ur}}] - s[x_{ur}, y_{ll}] - s[x_{ll}, y_{ur}] + s[x_{ll}, y_{ll}]}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}$$

无约束各向异性滤波——Ripmap和求和面积表都是各向异性滤波，能够检索非正方形区域的纹素值，对于接近水平和垂直方向最有效，但这两种方法都很耗费内存，Mipmap只需要1/3倍，Ripmap需要原纹理的3倍，求和面积表至少需要1倍。

由于纹理内存比较昂贵，不能浪费，对于Ripmap有如下改进：充分利用已有的Mipmap硬件，将像素单元在纹理空间投影的四边形采样多次，然后加权平均，（不用Ripmap那样的一个矩形去覆盖，而是用多个正方形来覆盖目标四边形，）四边形的短边用来确定d（Mipmap中用的是长边），长边用来创建一条各项异性直线，并确定采样次数。



优点：允许各向异性直线朝向任何方向，实时性好，除了Mipmap的纹理内存，不需要额外的纹理内存。

纹理高速缓存（Texture Cache）

纹理内存使用的一般原则——纹理尽量小，尽量使用相同纹理的多边形成组

纹理缓存的LRU策略和预取策略，加速纹理存取和上载。

Clipmap方法——将整个图像数据库看作一个Mipmap，在任何特定的视域只有一小部分低层次的Mipmap是需要的，因而根据视域对Mipmap进行裁减即可

纹理压缩——减小所需的纹理内存，增加有效Cache的大小，减少带宽需求

多重纹理绘制——实际操作中，光照方程不是以此计算的，而是在不同的pass计算，每一个pass都是对前一个pass的修改，这种计数被称作多重绘制技术。在多重绘制时，每重绘制计算光照方程的一部分，并采用帧缓存存储中间接过，再通过图像合成计数，就可以得到任意复杂的光照方程。

多重纹理贴图——减少绘制pass，允许复杂的shading model

纹理动画——逐帧变化的视频或者静态纹理坐标线性变换，得到的动态的纹理。

纹理贴图的方法——Alpha混合，基于环境映照的反射，采用凹凸纹理的粗糙表面模拟

Alpha Mapping——用来模拟印花效果或剪纸效果。在测试和替换Z-Buffer的值之前，先测试alpha的值，如果alpha为0，该像素就不需要做任何事情。

交叉树——纹理树没有厚度，若视点旋转则会出现缺陷，因此将纹理树复制并旋转90度，类似“玩具圣诞树”，得到一课交叉树（Cross Tree）

Light Mapping——静态光源，物体表面的漫反射在任何角度看起来都是一样的，因此可以计算出漫反射的光照纹理，再贴在物体上。

由于Light Map一般低分辨率，储存空间小，并且可以动态计算，所以一般将光照纹理单独分开，方便进行修改产生动态光源的效果。

Gloss Mapping光泽贴图——改变物体表面镜面高光分量贡献的纹理（改变物体表面光泽）。Gloss Map只是调制了高光的亮度，并没有改变会聚指数。

$$\text{光照方程：} o = t_{diff} \otimes i_{diff} + t_{gloss} i_{spec}$$

t_{diff}：漫反射颜色纹理的RGB值

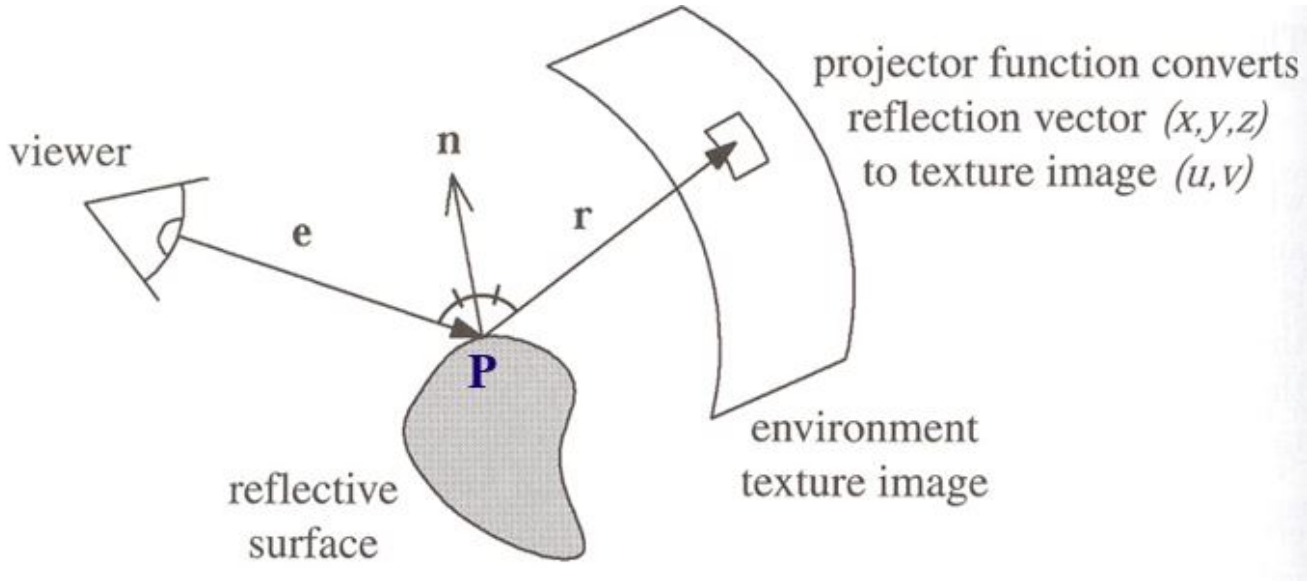
i_{diff}：插值得到的漫反射光亮度值

i_{spec}：插值得到的镜面高光光亮度值

t_{gloss}：$glossmap$值

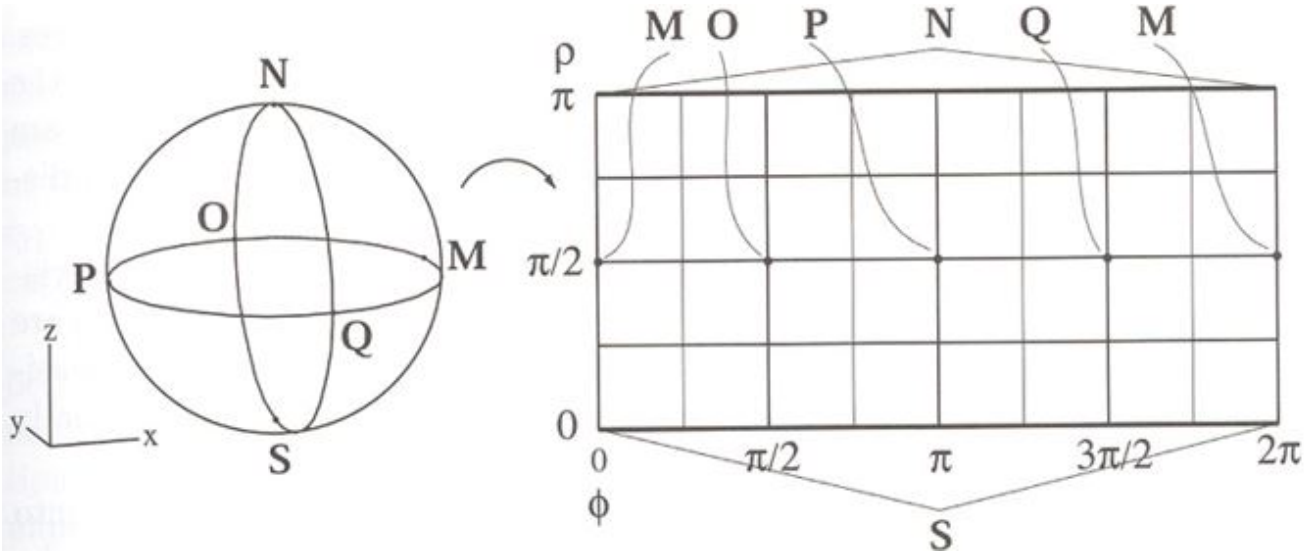
Environment Mapping环境映照（EM）——在曲面上产生近似反射效果的简单有效的方法。

EM与RT的区别——EM方法从视点到反射体某点P的一根光线e开始，根据P点的位置和法向n，计算反射光线r；但EM不会真正计算反射光线的最近交点，而是用该反射光线作为索引，得到EM中相应的环境纹理。



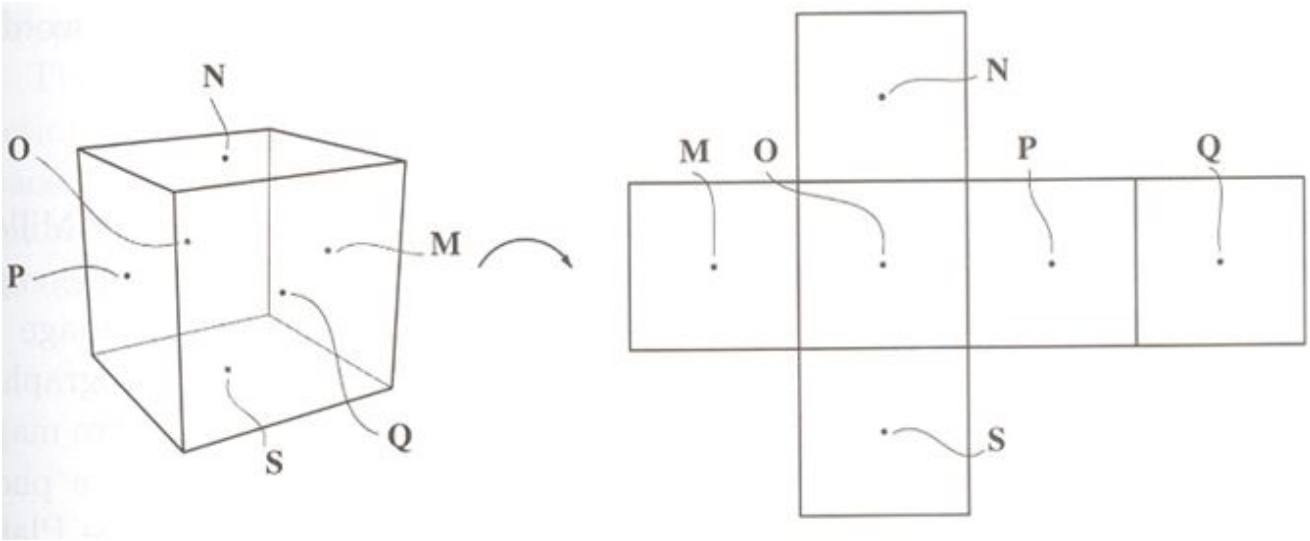
1. 载入或生成一幅表示环境的二维图像EM
2. 对于包含反射物体的每一个像素，计算相应点的表面法向
3. 根据视线向量和法向，计算反射向量
4. 用反射向量计算EM索引，表示该反射方向的物体
5. 用EM相应纹素来计算当前的像素颜色

Blinn-Newell方法——第一个EM方法。对于每个像素，首先计算反射向量，然后将该向量变换到球面坐标(ρ, ϕ)（经纬度），$\rho = \arccos(-r_z)$，$phi = \arctan(r_y, r_x)$；再将球面展开为矩形（EM），得到相应纹素。



优点：简单易于实现；缺点：在$\phi = 0$出存在边界，纹理在球的两极会汇聚成一点。由于边界问题和两极问题，会发生不可避免的错误，因此Blinn方法应用有限。

立方体EM方法——速度快且实用性强，在图形硬件中最受欢迎。将摄像机置于立方体中心，之后将环境投影到立方体的面上作为EM；实用中，场景需要绘制6次，摄像机必须正交看立方体的面。



EM面的选取——反射向量中绝对值最大的分量决定了选择哪个面

纹理坐标的计算——余下的两个坐标除以最大绝对值分量（得到[-1,1]之间的值），再变换到[0,1)上用于计算纹理坐标。

优点：没有奇异情况，与视点无关可以用于任何视线方向；缺点：Per-Pixel计算的，如果两个顶点反射在不同的面上，则需要正确插值。

Sphere Mapping——第一个商用图形卡支持的EM方法。通过正投影观察一个纯反射球面的外形。

真实环境球面图——对一个发亮的球面拍照，得到的圆形图像（也成为光探测器Light Probe）

虚拟场景的球面图——RT生成，或将立方体环境映射变形

抛物面映射——.....

环境映照的缺陷——对于平坦表面的效果较差，因为平坦表面反射光线变化很小，EM中只有一小块纹理被映射到很大的表面上，纹素过分放大；若采用正投影，情况更糟，反射光线相同，表面颜色只有一种。这种情况下需要使用其他平面反射方法。

Diffuse，Gloss和Reflection Mapping结合的光照模型——一个纹理四个通道，其中RGB作为漫反射颜色纹理，alpha作为光泽贴图强度

$$o = t_{diff} \otimes i_{diff} + t_{gloss} r_{spec}$$

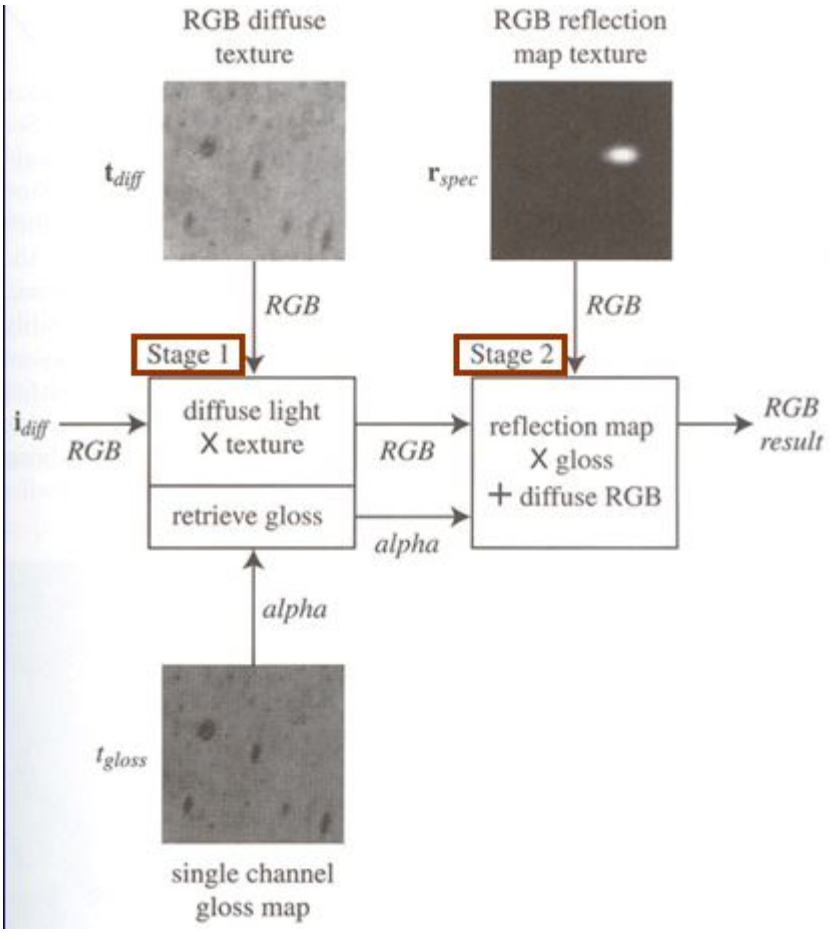
t_{diff}：漫反射颜色纹理的RGB值

i_{diff}：插值得到的漫反射光亮度值

r_{spec}：反射映射的RGB镜面高光值

t_{gloss}：单色光泽贴图，用来调制高光强度

根据公式，计算光照模型有两步：第一步将插值得到的漫反射光亮度和物体的RGB纹理相乘；第二步将光泽贴图与反射镜面高光相乘。两步结果相加就得到最终结果。



Bump Mapping——凹凸纹理贴图，用于模拟不平的表面的方法。基本想法是用纹理去修改物体表面的光照模型中的法向，而不是颜色；这一过程中物体几何法向没有变化，仅仅是光照法向改变了，相当于“欺骗了眼睛”，在物理意义上没有等价的操作。

Offset Vector Bump Map（偏移矢量凹凸纹理）——每个店储存两个带符号的值b_u和b_v，对应于法向在图像u和v的偏移量。

Height-Field Bump Map（高度场凹凸纹理）——灰度纹理值代表了高度，白色表示高，黑色表示矮。用相邻列的差得到u向斜率，相邻行的差得到v向斜率。

Bump Map优点：Per-Pixel凹凸纹理，花费地，几何细节丰富；缺陷：侧影轮廓处还是光滑的轮廓线而没有凹凸感，凹凸处不会投下自身的阴影。

Emboss Bump Mapping——浮雕凹凸纹理贴图，首先拷贝高度场图像，稍作偏移，再与原图相减。

```
1．将高度场用作漫反射单色纹理，绘制曲面
2．将所有定点坐标沿光源方向偏移
3．把高度场用作漫反射单色纹理，再次绘制曲面，并与第一次绘制结果相减，得到浮雕效果
4．用漫反射照明、Gouraud-shading再次绘制曲面，不应用高度场，将得到的图像与3的结果相加
```

缺点：只能适用于漫反射，不能用于镜面高光；光源在表面上时没有偏移量，凹凸纹理小时；需要偏移一个纹素，不允许Mipmap技术的使用

Dot3 Bump Mapping——点积凹凸纹理贴图。将真实的表面法向量(x,y,z)储存在normal map（法向图）中，每个8bit分量都映射到[-1, 1]上。对于每个定点，计算光源效果，首先将光源位置变换到表面的切空间，然后沿表面对光矢量进行插值；换言之，插值不是用的颜色和深度，而是光矢量。

Normal Map的优点：可以用来计算漫反射，也可以用来计算凹凸上的镜面高光。

EM Bump Mapping——让光泽表面产生凹凸效果。用Bump Map对于EM坐标进行扰动，使得反射矢量摇摆，从而使得反射表面发生变形。

优点：只需要两个纹理（一个Bump Map一个EM）而与光照无关，若与平面反射相结合，EMBM可以反射动态变化的场景；缺点：控制曲面效果很困难，缺少包含整个世界场景的环境图。

光学特性

主波长——产生颜色光的波长，对应于视觉感知的色调

光亮度——即表示光能大小又表示色彩组成的物理量，与周围环境无关

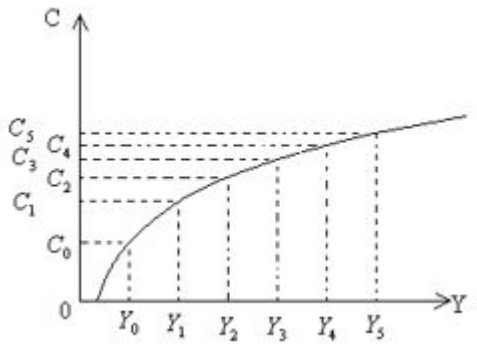
明度——视觉系统感知到的亮度

Webe定律：明度C与亮度Y，明度差dC为常数是，dY/Y也是常数。

$$\text{存在}\boldsymbol{a},\ \text{使得}:\ \boldsymbol{a}\cdot\frac{dY}{Y}=dC$$
$$\Rightarrow C=\boldsymbol{a}\log Y+\boldsymbol{c},\ \boldsymbol{c}\text{为常数}$$

单色模型（选择灰度/亮度）

亮度均匀分布——明度分布不均匀，灰度看起来分布均匀，就要求明度分布均匀。

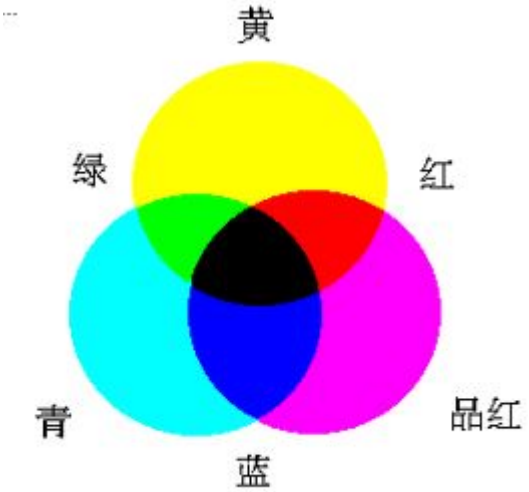


明度均匀分布——解出灰度$\boldsymbol{Y_i}=\boldsymbol{Y_0}^{\frac{n-i}{n}}$，给定n和$\boldsymbol{Y_0}$，就能得到合理分布的灰度序列。

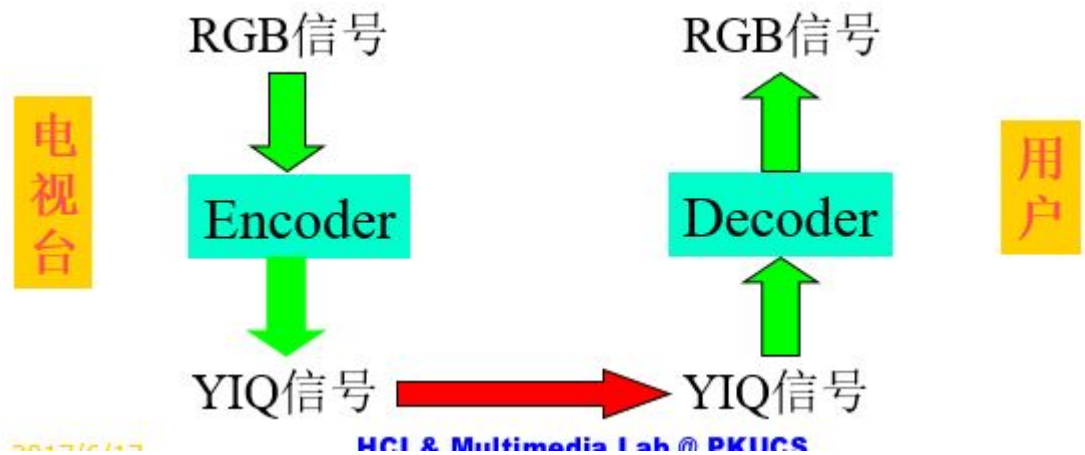
面向硬件的颜色模型

RGB模型——颜色=R*r+G*g+B*b，加色系统。用于彩色CRT。

CMY模型——青（Cyan）品红（Magenta）黄（Yellow），减色模型。用于硬拷贝设备。



YIQ模型——Y亮度，I,Q色差。用于电视传播系统。该模型节约带宽并且可以兼容黑白电视。



面向用户的颜色模型——HSV模型。Hue色彩，Saturation饱和度，Value明度。用于用户调色板

9. 真实感图形绘制

真实感图形绘制的步骤

数学方法建立三维场景的集合描述。通常是由三维造型系统完成。

将三维几何描述转为二维透视图。通过场景的透视变换来完成。

隐藏面消除，确定所有可见面，将视野外和被遮挡的不可见面消去。

计算可见面的颜色。基于光学物理的光照明模型进行计算，确定投影画面上的每一个像素的颜色。

简单光照明模型——仅考虑反射光；反射被分为理想漫反射和镜面反射；认为物体表面光滑；忽略光源颜色和形状（白色点光源）

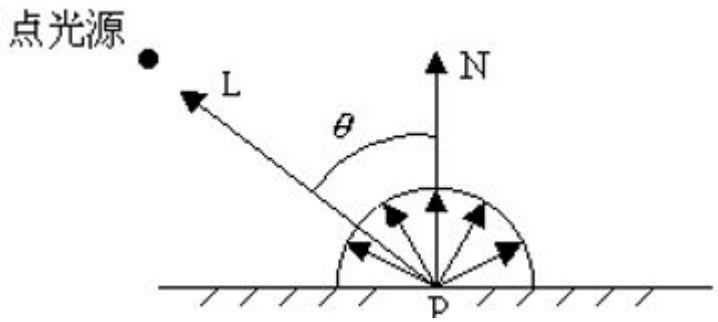
光源发出的光为直接光；物体对直接光的反射为直接反射；否则为间接光和间接反射。

环境光——光线在场景中经过复杂传播后，形成的弥漫于整个空间的光线

$$I_e = K_a I_a,\ I_a\text{环境光亮度},\ K_a\text{环境光反射系数},\ I_e\text{物体表面呈现的亮度}$$

漫反射——粗糙的、无光泽物体表面对电光源发出的光的反射

$$I_d = I_p K_d \cos\theta,\ \theta \in [0, \frac{\pi}{2}],\ I_p\text{点光源亮度},\ K_d\text{漫反射系数},\ \theta\text{入射角}$$

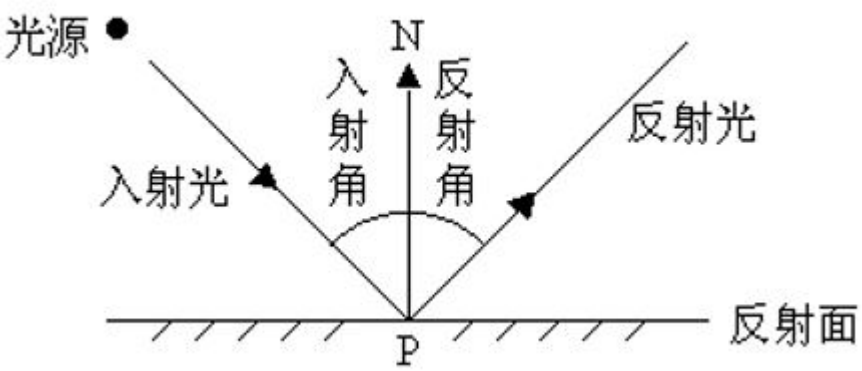


漫反射与环境光结合起来得到$I = I_e + I_d = I_a K_a + I_p K_d (L \cdot N)$

镜面反射——光滑物体（比如金属）表面对光的反射

高光——入射光在光滑物体表面形成的特别亮的区域

理想镜面反射

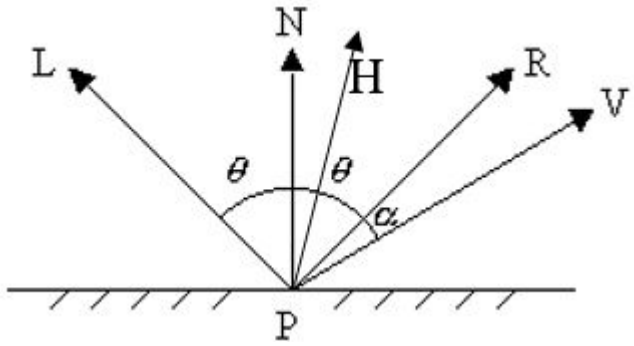


Phong模型——非理想镜面反射

$$I_s = I_p K_s \cos^n \alpha,\ I_s = I_p K_s (V \cdot R)^n$$

n镜面反射指数，V视点方向，R的计算用$H \cdot N$代替，夹角为α的一半

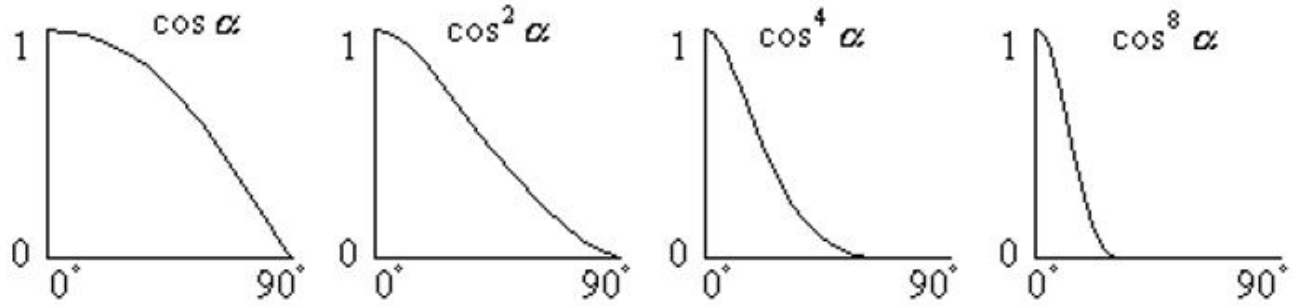
$$H = \frac{(L+V)}{2}$$



将环境光、漫反射和镜面反射结合，得到下式。对下式的分析，可以发现平行投影的好处，幂次项会趋于0，简化计算。

$$I = I_e + I_d + I_s = I_a K_a + I_p [K_d (L \cdot N) + K_s (H \cdot N)^n]$$

- **平行投影的好处，简化计算**



光的衰减——光在从光源到物体表面的过程中发生衰减

衰减函数：$f(d) = \min\{\frac{1}{c_0 + c_1 d + c_2 d^2}, 1\}$

光照明方程：$I = I_a K_a + f(d) I_p [K_d (L \cdot N) + K_s (N \cdot H)^n]$

产生彩色——首先选择合适的颜色模型（比如RGB），之后为每一种基色建立光照明方程

$$I_\lambda = I_{a\lambda} K_{a\lambda} + f(d) I_{p\lambda} [K_{d\lambda} (L \cdot N) + K_{s\lambda} (H \cdot N)^n]$$

$$\lambda = R, G, B$$

采用多光源的照明方程——m个光源

$$I_\lambda = I_{a\lambda}K_{a\lambda} + \sum_{i=1}^{m} f(d_i)I_{p_i\lambda}[K_{d\lambda}(L_i \cdot N) + K_{s\lambda}(H \cdot N_i)^n]$$

多边形绘制方法——均匀着色，光滑着色，Gouraud着色，Phong着色

插值着色法的问题：透视变形，依赖方向，公共定点处颜色不连续，定点方向不一定有代表性

均匀着色——去多边形上的一点，利用光照明方程计算颜色，之后用该颜色填充整个多边形。适用于光源在无穷远处，视点在无穷远处，多边形是物体表面的精确表示的情况。

优点：计算速度很快；缺点：相邻多边形颜色过渡不光滑

光滑着色——插值法

Gouraud着色——颜色插值法

1. 计算多边形单位法矢量
2. 计算多边形顶点的单位法矢量
3. 利用光照明方程计算定点颜色
4. 对多边形定点颜色进行双线性插值，得到内部各个点的颜色

Phong着色——法向插值法

1. 计算多边形单位法矢量
2. 计算多边形顶点的单位法矢量
3. 对多边形顶点法矢量双线性插值，获得内部各点的法矢量
4. 利用光照明方程计算各点颜色

阴影——光源不能直接照射的区域，对于光源来说的不可见面。考虑阴影的光照明方程如下

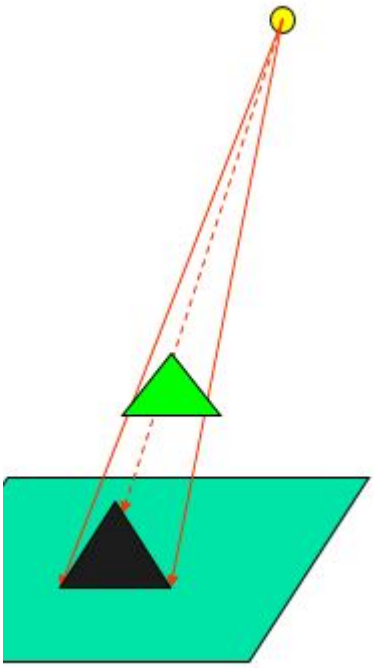
$$I_\lambda = I_{a\lambda}K_{a\lambda} + \sum_{i=1}^{m} S_i f(d_i)I_{p_i\lambda}[K_{d\lambda}(L_i \cdot N) + K_{s\lambda}(H \cdot N_i)^n]$$

$$S_i = 0(\text{处于第}i\text{个光源的阴影中})\ or\ 1(else)$$

Z-Buffer算法——将所有景物变换到光源坐标系中，利用Z-buffer算法进行消隐，得到的所有平面都是被照亮的面。

优点：算法简单；缺点：每个光源一个Z-Buffer，开销大

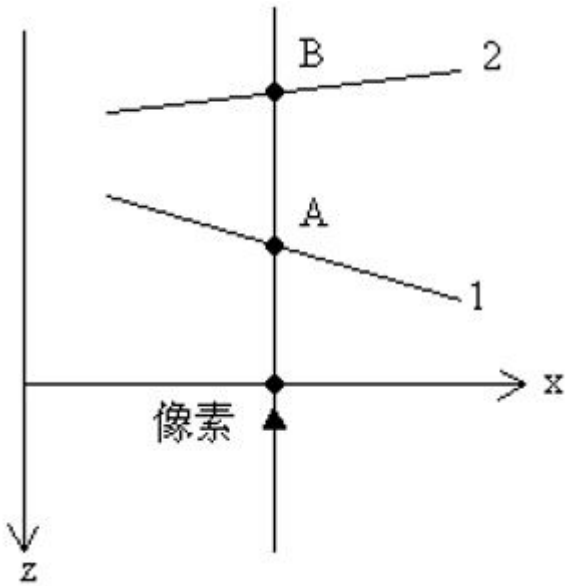
阴影细节多边形——在景物空间中利用裁剪算法求出被光源直接照射的多边形或其一部分，之后将这些多边形作为表面细节贴在物体表面上



光线追踪算法——从可见点P向光源发出测试光线，如果该光线在到达光源之前与其他物体相交，则P点位于阴影之中。

简单透明——不考虑折射现象的简单透明

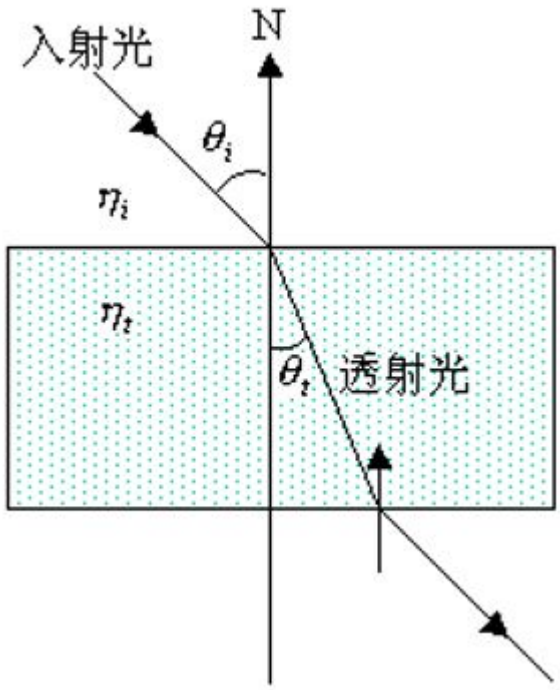
插值透明：$I_\lambda = (1 - K_{t_1})I_{\lambda_1} + K_{t_2}I_{\lambda_2}$，$K_{t_i}$为透射系数



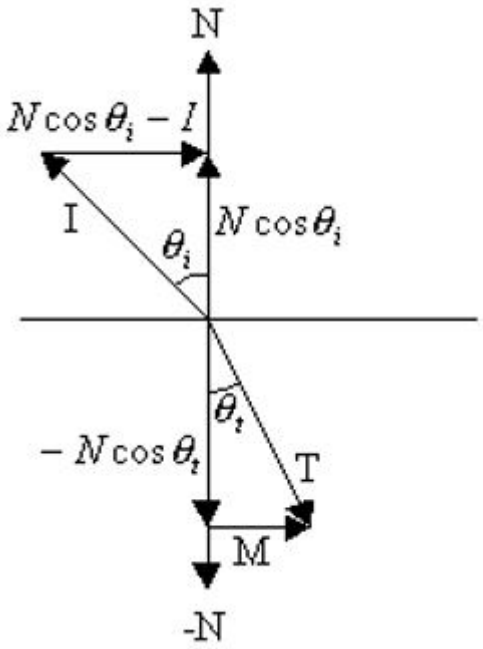
过滤透明：$I_\lambda = I_{\lambda_1} + K_{t_2}C_{t_\lambda}I_{\lambda_2}$，透射系数越大颜色透射得越多，$C_{t_\lambda}$不同的颜色系数不同

考虑折射的透明

折射定律：$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i}$



投射适量的计算：$T = (\eta \cos\theta_i - \cos\theta_t)N - \eta I$



整体光照明模型——光线追踪（RT），辐射度。需要考虑周围环境的光对物体表面的影响，以及物体表面的细节纹理。

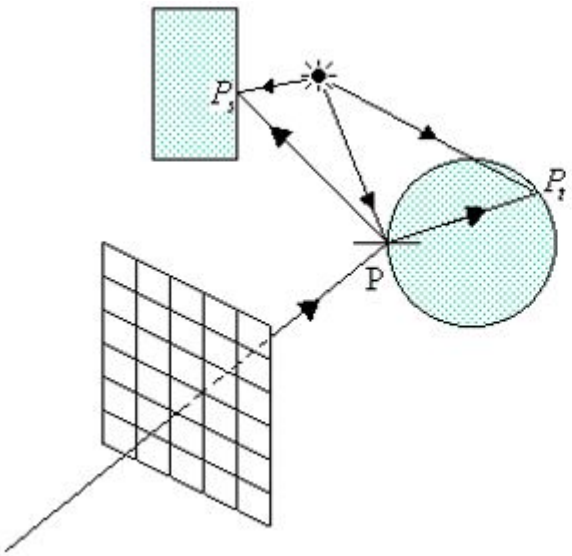
物体表面的入射光——光源直接照射，其他物体的反射光，投射光。在局部光照模型中，只有直接照射被考虑了。

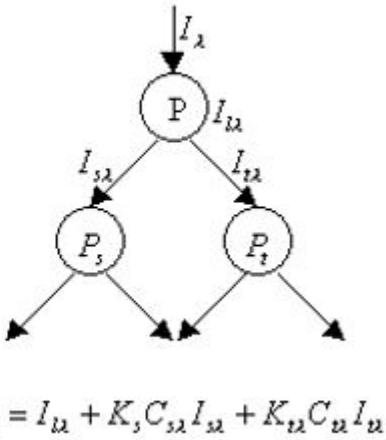
$$I_\lambda = I_{l_\lambda} + K_s C_{s_\lambda} I_{s_\lambda} + K_t C_{t_\lambda} I_{t_\lambda}$$

I_{s_λ}为镜面反射方向其他物体反射或折射的光亮度

I_{t_λ}为投射方向其他物体反射或折射来的光

光线追踪法（RT）——光线传播的逆过程（人眼→物体表面→物体表面→...→光源）





$$I_\lambda = I_{l\lambda} + K_s C_{s\lambda} I_{s\lambda} + K_{t\lambda} C_{t\lambda} I_{t\lambda}$$

关键问题：光线与物体表面求交，阴影测试线S，逢交点考察一次反射折射

终止条件：光线没有碰到任何物体，光线碰到了背景，光线在多次反射折射后产生衰减至小于某个阈值，反射或折射次数（追踪深度）大于一个定值

```
RayTracing(start, direction, weight, color):
1. 测试weight是否小于阈值MinWeight，若小于则终止，颜色为黑色
2. 计算光线与所有物体的交点中距离start最近的点
3. 如果没有交点，则终止，颜色为黑色
4. 否则在交点处利用局部光照模型计算光强度Ilocal
5. 计算反射方向R，向R进行追踪，RayTracing(最近的交点,R,weight*Wr,Ir)
6. 计算折射方向T，向T进行追踪，RayTracing(最近的交点,T,weight*Wt,It)
7. color = Ilocal+Ks*Ir+Kt*It，返回
```

加速技术：基本的RT，每条射线都需要和所有物体求交，之后再排序才能确定可见点，这一方法效率很低。加速技术包括提高求交速度，减少求交次数，减少光线条数，采用广义光线和并行计算等。

优点：较好的模拟了光在光滑物体表面的反射和折射现象；缺点：无法模拟光在景物之间漫反射引起的彩色渗透现象（多重漫反射效果）

辐射度方法——将场景看作封闭的能量系统，能量经过多重反射达到平衡状态，根据能量守恒定律计算表面能量即可。与视点无关，问题在于如何提高算法效率

定义

立体角：作半径为r的球面，用立体角的边界在球面上所截棉结除以半径平方来表示立体角大小

$$d\omega = \frac{dS}{r^2}$$

光通量：单位时间内通过面元的光能量，dF

光强度：某方向上单位立体角内的光通量。表征了物体在某一方向上的发光情况

$$J = \frac{dF}{d\omega}$$

光亮度：发光面元单位面积上某方向辐射的光能

$$I = \frac{J}{dS_i \cos i} = \frac{dF}{dS_i \cos i d\omega}$$

假设——封闭系统中，景物表面均为理想漫反射表面，景物表面上每一点向周围各个方向辐射的光亮度相同，因而表面各点的光亮度只与位置有关而与辐射方向无关。

$$dp = I\cos\theta d\omega,\ dp\text{为表面某点单位面积上朝某辐射方向发出的光通量}$$

$$B = \int_{\Omega} dp = \int_{\Omega} I\cos\theta d\omega,\ B\text{为该点总辐射度，}\Omega\text{为该点法线方向的半球面空间}$$

$$B = \int_{\Omega} I\cos\theta d\omega = I\int_0^{2\pi}\int_0^{\frac{\pi}{2}} \cos\theta\sin\theta d\theta d\Phi = I\pi$$

通过上式发现$B = I\pi$，因而可以通过求解整个场景的辐射度来计算各个点的光亮度

辐射度求解方法

$$B(x)dA(x) = E(x)dA(x) + \rho(x)H(x)$$

$B(x)$为某微面元的辐射度

$\rho(x)$为表面在该面元处的反射率

$H(x)$为环境入射到面元出的光能

$E(x)$为表面在面元处的自身辐射度，若表面为漫射光源，$E(x) > 0$，否则$E(x) = 0$

$dA(x)$为面元的面积