# 1 光栅显示扫描系统

# 2 光栅图形学

基本图形的扫描转换(连续的图形转化为离散的点)
多边形的扫描转换和区域填充
裁剪
反走样(解决逼近误差)
投影(三维图形投影到二维)
消隐(消除不可见物体)

# 3 图形的扫描转换和区域填充

## 3.1 直线的扫描转换

要考虑灰度亮度颜色效率等问题
常用算法:

### 3.1.1 数值微分法(DDA)

对于y=kx+b，每次取点(x,round(x))，然后可以减少乘法，$y_{i+1}-y_i=\Delta y=k$
对于斜率小于1的沿x方向步进，否则沿y方向步进

### 3.1.2 中点划线法

主要优化DDA中的取整
确定某个点可能的两个位置重点和直线位置关系
需要判别式d=F(x,y)=ax+by+c
根据d与0的关系判断位置
同样使用增量计算进行优化
$a=y_0-y_1, b=x_0-x_1$
$d_0=F(x_0+1,y_0+0.5)=F(x_0,y_0)+a+0.5b=a+0.5b$
乘以2变成2d=2a+b把所有数变成整数
然后$\Delta_1=2a, \Delta_2=2a+2b$

### 3.1.3 Bresenham算法

通过误差项的符号进行计算
k=dy/dx,e=$d_0-0.5, e_{i+1}=e_i+k$
$e_i<0$ →右方
$e_i>0$ →右上方

## 3.2 圆弧的扫描转换算法

圆的特性：八对称性
整个圆只需要八分之一圆弧加对称性即可

### 3.2.1 DDA

计算复杂

### 3.2.2 中点画圆法

构造判别式$F(x,y)=x^2+y^2-R^2$
$d=F(M)=F(x_p+1)^2(y_p-0.5)^2-R^2$

### 3.2.3 Bresenham算法

和直线类似

### 3.2.4 生成圆弧的正负法

一个点在圆弧内，下一个点在圆弧外，判别式的符号不断变化

### 3.2.5 圆的多边形逼近法

当多边形边数足够多的时足够逼近圆
对于n边形，内角$\alpha=\frac{2\pi}{n}$
逼近误差$\delta=R-Rcos\frac{\alpha}{2}=R-Rcos\frac{\pi}{n}$

## 3.3 多边形扫描转换

线框显示→面着色
多边形
外环:逆时针
内环:顺时针

### 3.3.1 逐点判断法

逐个像素判断
效率过低

### 3.3.2 射线法

有某个点发出射线，根据交点数量来判定在内部还是外部
但是和顶点相交需要特殊考虑(向其他方向发出射线，让他不过顶点)

### 3.3.3 累计角度法

向各个顶点发出射线，而边，无论是外环还是内环都会是有方向的，所以，可以决定角度角度的方向，逆时针或者顺时针，分别对应正负角度，求和，外部和为0，内部和为正负$\frac{\pi}{2}$
效率比较低，割裂了相邻像素间的关系

### 3.3.4 扫描线算法

求交:扫描线与多边形的交点，但是需要注意扫描线与多边形定点交点的取舍
排序:将交点进行排序
配对:确定内部与外部
填色

#### 数据结构

活性边表(AET)：把当前扫描线相交的边成为活性边，将它与扫描线的交点x坐标按递增顺序存放在链表中，保留了直线的连贯性和扫描线的连贯性
链表节点内容：当前扫描线与边的交点的x坐标
$\Delta x$：从当前扫描线到下一条线间x增量 $y_{max}$：改边所交最高扫描线号
新边表(NET)：存放该扫描线第一次出现的边，若谋变的较低端点为$y_{min}$，则该边就放在扫描线$y_{min}$的新边表中
新边表每个节点的数据：x坐标–斜率$\frac{dx}{dy}$–$y_{max}$

#### 算法过程

初始化新边表NET:
对于每条扫描线进行处理
while(扫描线未处理完)
{
把新边表NET[y]中的边界点用插入排序法插入AET表，使之按x坐标递增
便利AET表，把配对焦点区间像素改变颜色
y=y+1
便利AET表，把$y_{max} = y$的节点从AET中删除，并把$y_{max} \geq y$节点的x值递增$\Delta x$
}

## 3.4 边缘填充法

对于每条扫描线和个多边形边的交点，将该扫描线上交点右方的所有像素颜色取补，对多边形的每条边作此处理，与边顺序无关
缺点：一个点可能被访问多次，计算量大，不适合图像填充
优化：增加栅栏(与扫描线垂直的直线，通常过一丁点且把多边形分为左右两半)，把交点和栅栏之间的像素取补

## 3.5 边界标志法

对多边形的每条边进行直线扫描转换，也就是对多边形的边界所在像素打上标记
然后采用扫描线算法类似算法给每个像素打上位置标记

## 3.6 区域填充算法

4向连通区域vs8向连通区域

### 3.6.1 递归算法

填充每个点的4/8向连通点

### 3.6.2 扫描线算法

填充种子点所在扫描线位于给定区域的一个区段
确定与这一区段相连通的上下两条扫描线是哪个位于给定与区内的区段，保存下来，留作种子
重复上述步骤

# 4 裁剪

## 4.1 直线段裁剪

### 4.1.1 直接求交法

思想:
两个端点都在窗口之内→直线在窗口内
两个端点同时位于窗口的左侧，右侧，上侧，下侧时→直线在窗口外
然后判断有效交点

Table 1: 区域划分

| | | |
|---|---|---|
| 4 | 3 | 2 |
| 5 | 0 | 1 |
| 6 | 7 | 8 |

#### 4.1.2 Cohen-Sutherland裁剪

快速判断是否完全在边界外
使用编码：$C_tC_bC_rC_l$

$$C_t = \begin{cases} 1, y > y_r \\ 0, other \end{cases} \quad C_b = \begin{cases} 1, y < y_b \\ 0, other \end{cases} \quad C_r = \begin{cases} 1, x > x_r \\ 0, other \end{cases} \quad C_l = \begin{cases} 1, x < x_l \\ 0, other \end{cases} \tag{1}$$

算法过程：
$P_1P_2$在窗口外$\Leftrightarrow$ $code1$ & $code2 \neq 0$
$P_1P_2$在窗口内$\Leftrightarrow$ $code1 = 0$ && $code2 = 0$
$P_1$在窗口内，交换$P_1P_2$
用$P_1P_2$和窗口边缘的交点取代$P_1$，算法继续

#### 4.1.3 中点分割裁剪算法

优势：适合硬件实现
二分查找，按照上面的编码方法进行检查，最终收敛于交点或者子段足够小

#### 4.1.4 Nicholl-Lee-Nicholl算法

主要消除C-S算法中的多次求交的情况
对2D平面的划分更为详细
内部：0
角域：2,4,6,8
边域：1,3,5,7
算法：首先通过一定变换(交换，对称等)使$P_1$落在0,5,6区域
然后通过$P_1$做引到四个顶点的射线，把区域分为4部分，通过判断$P_2$在哪个区域来判定$P_1P_2$与窗口那些边相交
特点：效率较高，仅适合二维矩形

#### 4.1.5 梁友栋-Barsky算法

首先排除与窗口某一条边向平行的线段
线段与其延长线和裁剪窗口的四条边所在直线有4个交点
根据始点$P_1$到终点$P_2$的方向记录4个交点，A,B,C,D，取A,B中离$P_2$近的点，取C,D中离$P_1$近的点

#### 4.1.6 参数化算法(Lyrus-Beck算法)

对于凸多边形区域R和直线段$P_1P_2$
$P(t) = (P_2 - P_1) * t + P_1$
P(t)早凸多边形内的充要条件是，对于凸多边形边界上任意一点A和该点内法向N，都有：
$N \cdot (P(t) - A) > 0$
当凸多边形是矩形窗口且矩形的边域坐标轴平行时，算法退化为Liang-Barsky算法

### 4.2 多边形裁剪

需要保证裁剪后仍然闭合

#### 4.2.1 Sutherland-Hodgeman算法

多边形用顶点序列表示
逐次多边形裁剪
每次用窗口的一条边裁剪多边形，裁剪结果传递给下一条边
判断可见性
对于输入线段SP(S为起点，P为终点)
1.SP均不可见，输出0个顶点
2.SP均可见，输出P
3.S可见，P不可见，SP与边界交点为I，输出I
4.S不可见，P可见，SP与边界交点为I，输出I，P
缺陷：连续两个输出点中间的所有点不可见，而这连个点不在窗口同一边缘

#### 4.2.2 Weiler-Atherton算法

首先约定顶点序列的顺序：外环逆时针，内环顺时针
裁剪结果有主多边形的部分边界和裁剪多边形的部分边界共同组成
入点：主多边形边界由此进入裁剪多边形区域内部
出点：离开裁剪多边形区域进入主多边形边界内部
算法过程：
1.建立主多边形和裁剪多边形的顶点表
2.求交点，归类(出点还是入点)，按序插入顶点表，在两个表的响应顶点间建立双向指针
3.裁剪
3.1如果还有未跟踪的点，把它作为一个起点，建立新的裁剪结果多边形顶点表，插入该点
3.2如果该点为为入点，在主多边形顶点表内跟踪，否则在裁剪多边形顶点表内跟踪
3.3如果跟踪到的是多边形顶点，将其加入结果顶点表，继续，知道遇到新的交点，重复3.2-3.3，直至回到起点

# 5 字符

## 5.1 字库

分为矢量型和点阵型良两种
矢量型存储量小，美观，变换方便，需要光栅化后才能显示
点阵型存储量大，易于显示

# 6 图形变换与投影

## 6.1 图形变换

### 6.1.1 平移

$$P' = P + T = \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

### 6.1.2 旋转

逆时针为正，旋转$\theta$

$$P' = R \cdot P = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

### 6.1.3 缩放

$$P' = S \cdot P = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

## 6.2 齐次坐标

$$\begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix} x_h = h \cdot x, y_h = h \cdot y, h \neq 0$$

### 6.2.1 平移

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T(t_x, t_y) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

### 6.2.2 旋转

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

### 6.2.3 比例变换

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = S(s_x, s_y) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

### 6.2.4 错切边换

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & b & 0 \\ d & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

沿x方向错切：d=0
沿y方向错切：b=0
沿x,y方向错切：b≠0且d≠0

### 6.2.5 仿射变换

$$A_f = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

## 6.3 二维图形的显示流程

坐标系Coordinate System
世界坐标系WCS
用户坐标系UCS
局部坐标系LCS
造型坐标系MCS
屏幕坐标系SCS
设备坐标系DCS

### 6.3.1 窗口到市区的变换

窗口：左下角$(x_{min}, y_{min})$，长宽$E_x, E_y$
视口：左下角$(u_{min}, v_{min})$，长宽$E_u, E_v$
转换矩阵

$$M_{wv} = T(u_{min}, v_{min}) S(\frac{E_u}{E_x}, \frac{E_v}{E_y}) T(-x_{min}, -y_{min})$$

## 6.4 三维几何变换

### 6.4.1 三维齐次坐标

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \Rightarrow \begin{pmatrix} x_h \\ y_h \\ z_h \\ h \end{pmatrix}, x_h = h \cdot x, y_h = h \cdot y, z_h = h \cdot z, h \neq 0$$

### 6.4.2 三维变换的一般形式

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 6.4.3 三维坐标的之间的变换

P在原坐标系坐标

$$P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

新的坐标系原点

$$C' = \begin{bmatrix} c'_x \\ c'_y \\ c'_z \end{bmatrix}$$

新的坐标系单位向量

$$X' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} Y' = \begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \end{bmatrix} Z' = \begin{bmatrix} z'_1 \\ z'_2 \\ z'_3 \end{bmatrix}$$

新坐标系

$$P \to P' : P' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = R \cdot T \cdot P = \begin{bmatrix} x'_1 & x'_2 & x'_3 & 0 \\ y'_1 & y'_2 & y'_3 & 0 \\ z'_1 & z'_2 & z'_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' \to P : P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = C' + (p'_x X' + p'_y Y' + p'_z Z')$$

## 6.5 投影变换

投影变换：三维物体转换为二维图形表示的过程

### 6.5.1 投影分类

平行投影：正平行投影，斜平行投影
透视投影：一点透视，二点透视，三点透视(主要由投影平面和xyz轴的平行关系决定，平行的越多，，灭点越少)
三视图是平行投影
灭点：不平行于投影平面的平行线经过透视投影后收敛于一点
主灭点：平行于坐标轴的平行线的灭点

### 6.5.2 观察坐标系

世界坐标系⇒观察坐标系
观察坐标系的远点在世界坐标系的位置$(O_{vx}, O_{vy}, O_{vz})$
观察坐标系的三个坐标轴向量$(X_{vx}, X_{vy}, X_{vz}), (Y_{vx}, Y_{vy}, Y_{vz}), (Z_{vx}, Z_{vy}), Z_{vz}$
变换矩阵

$$M_{W\to V} = R \cdot T$$

$$R = \begin{bmatrix} X_{vx} & X_{vy} & X_{vz} & 0 \\ Y_{vx} & Y_{vy} & Y_{vz} & 0 \\ Z_{vx} & Z_{vy} & Z_{vz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T = \begin{bmatrix} 1 & 0 & 0 & -O_{vx} \\ 0 & 1 & 0 & -O_{vy} \\ 0 & 0 & 1 & -O_{vz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## 6.6 三维裁剪

合二维裁剪类似，可以通过参数方程，也可以通过判别式

## 6.7 透视投影变换

投影平面$Z_V = 0$，投影中心(0,0,d)，待投影点$P(x_P, y_p, z_p)$，求投影点$Q(x_Q, y_Q, z_Q)$
透视投影变换矩阵

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix}$$

$$Q = M_{per} \cdot P$$

## 6.8 平行投影变换

投影平面$Z_V = 0$，投影方向(0,0,-1)，待投影点$P(x_P, y_p, z_p)$，求投影点$Q(x_Q, y_Q, z_Q)$
透视投影变换矩阵

$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q = M_{ort} \cdot P$$

$$\lim_{d \to \infty} M_{per} = M_{ort}$$

## 6.9 反走样

用离散量表示连续梁引起的失真现象称为走样
用于减少或消除失真效果的技术成为反走样

### 6.9.1 提高分辨率

锯齿仍然存在，数量更多，宽度更小
只能减轻，不能消除锯齿

### 6.9.2 超采样

以更高的分辨率进行采样，最终进行一定的平均

### 6.9.3 区域采样

通过每个像素与目标图形的相交区域面积确定该像素的亮度
为简化计算可以采用离散的方法
存在的问题：计算比较复杂，并且中心和边缘对亮度的贡献不应该一样

### 6.9.4 加权区域采样

解决了上面贡献度不同的问题
使用圆锥滤波器进行积分

### 6.9.5 随机采样

均匀采样会引起干涉纹理(Moire Pattern)
所以采用随机采样，但是随机采样会导致一定的模糊

#### 6.9.6 抖动采样

加上某种限定的随机采样

# 7 消隐

为了真实地反映不同物体之间或物体的不同部分之间由于遮挡而造成的某些物体或部分的不可见现象，须把不可见部分从图中消除，称为消除隐藏线和消除隐藏面，简称消隐，也称为可见面检测。
投影变换失去了深度信息，往往导致二义性

## 7.1 三维物体的表示

构造实体表示(CSG)：不唯一但是适合交互，由一些基本的体的运算表示
边界表示(B-rep)：体-面-环(一个面由一个外环和多个内环组成)-边-点

## 7.2 提高消隐算法效率的常用方法

### 7.2.1 排序

通过遮挡关系决定计算顺序

### 7.2.2 连贯性

观察的物体或视口内的图像缓慢变化
物体连贯性
深度连贯性
区域连贯性
扫描线连贯性

### 7.2.3 包围盒技术

包围盒：包围目标的简单形状，避免盲目求交
AABB(Axis Aligned Bounding Box)

### 7.2.4 后向面剔除

面法向V
$V \cdot N > 0$ ⇒后向面
$V \cdot N < 0$ ⇒前向面

#### 7.2.5 空间分割技术

## 7.3 线消隐算法

输入:面表和线表
简介:假设E为面F的一条边，需判别F外的内一个面和E的遮挡关系(使用栈来存储需要消隐的线段，因为线段可能被截成两段)

### 7.3.1 多边形对直线段的遮挡判断算法

1.线段两端点和视点在多边形面同侧，转7
2.判断线段投影和多边形投影有无交点，无交点说明未被遮挡，转7
3.求直线和相应无穷平面的交点。如果无交点，说明线面平行，转4，否则，如果交点在线段内部，线段分为两端，与视点同侧未被遮挡，异侧可能会被遮挡，转4，如果交点在线段外部，说明线段完全在平面之后，但是是不是在多边形之后还无法判断，转4
4.求所剩线段的投影与多边形投影的所有交点，并根据交点在元直线参数方程中的参数值求出Z值即深度，若无交点，转5
5.以上所求得的各交点将线段的投影分为若干段，取第一段中点
6.判断该重点是否在多边形投影内，剩下的可以依次交替取值进行判断
7.结束

### 7.3.2 加速方法

1.剔除后向面，减少线面求交运算
2.包围盒测试，简化线线求交，线面求交
3.空间分割，减少求交

## 7.4 面消隐算法

### 7.4.1 画家算法

思路：先涂上背景色，然后从后往前按顺序绘制各个面
关键：如何将场景中的物体按照深度排序
深度重叠测试：如果$Z_{max}(P) < Z_{min}(Q)$，那么Q深度比P大，P肯定不能遮挡Q
投影重叠测试：P和Q在oxy平面上投影的包围盒在x/y方向上不相交;P各顶点在Q远离/靠近视点一侧
无法处理的情况：多边形相互穿透，循环重叠

### 7.4.2 Z-Buffer算法

帧缓存来存放每个像素的颜色值(初值为背景色)
深度缓存来存放每个像素的深度值(初值取z的最小值)
过程：对每个面上的每个点的值计算它在帧缓冲器的相应位置，然后比较他的

深度和深度缓存中存储值的大小，如果在前面，替换帧缓冲器的值
优点：与物体遮挡关系无关，有利于硬件实现，简单
缺点：空间占用高，计算过多没有利用物体的相关性和连续性

### 7.4.3 解决空间占用问题

1.区域划分
2.扫描线Z-Buffer算法
3.使用改进算法，更改循环的顺序
原算法：
for 每个多边形
for 多边形内每个像素
计算投影位置，比较深度，修改Z-Buffer
改进算法：
for 每个像素
for 每个多边形
像素是否在多边形内
计算深度，和一个深度缓存变量进行比较
关键点：
1.如何判断像素是否在投影多边形内/包含性检测 射线法：向某方向引一条射线，通过交点数进行判断
若射线正好经过多边形的顶点，则采用“左开右闭”的原则来实现，即：当射线与某条边的顶点相交时，若边在射线的左侧，交点有效，计数；若边在射线的右侧， 交点无效，不计数
累计角度法：弧长累计角度为$2\pi$，点在多边形内部，累计角度为0，点在多边形外部
射线法加速：只计交点数目不真正求交点
累计角度发：以顶点符号变化(象限之间的变换)为基础进行弧长累加，不真正计算角度
2.如何计算深度：对于ax+by+cz+d=0，$depth = -\frac{ai+bj+d}{c}(c \neq 0)$，如果c=0，引一条射线，交点的最后一个

### 7.4.4 扫描线Z-Buffer算法

目的：解决Z-Buffer算法中没有利用到图形的相关性与连续性的问题
思想：处理当前扫描线是，开一个数组作为Z-Buffer
找出与当前扫描线相关的多边形及其中相关的边对
对每一个边对之间的小区间上的各像素，采用增量法计算式深度，并与Z-Buffer上的值进行补缴，找出各像素处可见平面，写入帧缓存
x增量：对于边px+qy+r=0，x=$-\frac{py+r}{p}$，$x_{j+1} = x_j - \frac{q}{p}$，$\Delta x = -\frac{q}{p}$
z增量：多边形龟缩在平面方程为xz+by+cz+d=0，z=$-\frac{ax+by+d}{c}$，$z_{i+1} = z_i - \frac{a}{c}$，$\Delta z_x = -\frac{a}{c}$，$\Delta z_y = -\frac{b}{c}$
数据结构：
多边形表:存储多边形

活性多边形表APT:存储与当前扫描线相交的多边形
边表ET：每个多边形存储一个边表
活化边对表AET:与当前扫描线相交的边对(需要进行配对)
算法流程:
建多边形Y表，对每一个多边形根据顶点最小的y值，将多边形置入多边形表。
初始化APT，AET
对于每条扫描线
1.初始化帧缓存CB，深度缓存ZB
2.将对应扫描线的多边形加入APT中
3.对新加入的多边形，生成边表
4.对于APT中的每个多边形，扫描其中的边，进行配对并加入AET中
5.对于AET的每条边，对于$x_l < x < x_r$中的每个像素($x_l, x_r$分别是配对的边和扫描线交点中较大和较小的x坐标)，通过增量计算深度，并与ZB进行比较，决定是否更新CB
6.删除APT中多边形顶点最大y坐标在这条扫描线上的多边形，并删除相应ET
7.对于AET的每一个边对，删除最大y坐标在扫描线上的边，如果一个边对中只删除了一条，那么需要重新配对
8.更新$x_l, x_r, z_l$
仍然存在的问题:
每个多边形覆盖的每个像素均要进行计算
多个多边形覆盖的区域需要多次计算

### 7.4.5 区间扫描线算法

思想:把当前扫描线与各多边形在投影平面的投影的交点进行排序后，使扫描线分为若干子区间。只要在区间任一点处找出在该处z值最大的一个面，这个区间上的每一个象素就用这个面的颜色来显示。
优势:一次性计算一段，不必逐像素计算
活化边表AET中结点是边，不是边对
如何知道每个区间有几个相关多边形
为每个多边形增加一个标志位，每遇到一条该多边形的边，进行取反
这样就可以知道每一段上被哪些多边形覆盖，然后可以取每一段的中点，计算它在每个覆盖多边形上的深度，可以得到深度最小的多边形
**面相互贯穿时需要将其切分**

### 7.4.6 区域子分割算法

思想:把物体投影到屏幕区域上，然后递归分割区域，直到区域内目标足够简单，可以显示，或者已经分割到象素为止。
区域与多边形的简单覆盖关系有四种：内含、相交、包围和分离。
算法步骤:
1.对所有多边形判断与该区域的位置关系
2.如果都是分离，置为背景色
3.如果只有一个多边形与其相交，或该多边形完全在区域内，先给区域着背景

色，然后是再用扫描线算法填充多边形的颜色(只和一个多边形有联系)
4.有一个多边形包围了该区域，或者该区域与多个多边形相交但是离视点最近的多边形包围了该区域，着上离视点最近的多边形的颜色(简而言之，有一个多边形包围了该区域，并且没有一个更近的多边形把它遮住)
5.对于不属于上述情况的，将区域一分为四，分别进行处理，如果只剩下最后一像素，取该区域颜色最靠近视点的多边形颜色，或与该窗口相交的多边形颜色的平均值
这是一个分而治之的递归问题，需要提高效率:
1.空间分割，不在同一空间的多边形和区域不需要进行判断
2.对所有多边形按其顶点z坐标最大值排序，组成多边形列表，随着区域分割，不断修改多边形列表，找到离视点最近的多边形(粗略排序，和画家算法类似)
3.使用包围盒技术，减少计算量

#### 7.4.7 光线投射算法

基本思想:
和人眼接受物体光线类似，只不过把视点当成了光源，最后检测光线落在了哪里，就着哪一种颜色
算法流程:
对于屏幕的每个像素，计算视点到改像素的射线
对于每个物体，和该射线求交
如果有交点，排序，取离视点最近的交点的颜色
如果没有交点，使用背景色

# 8 颜色视觉

颜色三个特性:
色调-主波长
饱和度-纯度:白光所占比例
亮度-明度
颜色纺锤体

## 8.1 CIE色度图

### 8.1.1 CIE-RGB系统

c=rG+gG+bB
根据RGB三原色匹配任意颜色的光谱–三刺激值线
存在的问题：曲线的一部分三刺激值为负数

#### 8.1.2 CIE-XYZ系统

c=xX+yY+zZ
任何颜色都能由标准三原色混合匹配

#### 8.1.3 色度图

色度空间：用三原色的单位向量定义三维颜色空间(和笛卡尔坐标系类似)
色度图，色度平面
用于唯一的表示三刺激空间中的所有颜色值
常见颜色模型：RGB/CMY/HSV
画家配色方法

# 9 光照明模型

光照明模型/明暗效应模型(illumination model, shading model, lighting model)：
根据光学物理的有关定律，计算景物表面上任何一点投向观察者眼中的光亮度的大小和色彩组成
局部光照明模型：只考虑物体与光源之间的关系
整体光照明模型：考虑反射光折射光等等
能量关系：能量守恒
$I_i = I_d + I_s + I_t + I_y$
$I_i$ :入射光强
$I_d$ :漫反射光强
$I_s$ :镜面反射光强
$I_t$ :透射光强
$I_y$ :吸收光强

## 9.1 Phong光照明模型

阴影： 加强场景的层次感
自身阴影：背向光源(后向面)
投影阴影：不透明景物的遮挡
本影： 半影：

# 10 OPENGL

## 10.1 如何绘制立方体

顶点，面，三角形
局部坐标系
贴图
观察者

遮挡
光照(阴影、高光)
显示

## 10.2 OPENGL库

核心库：GL
功能库：GLU

## 10.3 OPENGL功能

物体描述(建模)
变换(几何变换、投影变换、视口变换)
颜色模式
像素处理(像素、位图、字体、图形处理、贴图、抗锯齿)

## 10.4 GLM

矩阵变换
**四元数**

## 10.5 几何变换

观察变换
建立观察坐标系(u,v,w)

CIE-RGB系统 CIE-XYZ系统