图形学笔记

重点内容

- 第一~三章：conceptions
- 第四章：transformation
  - 平面解析几何 & 空间解析几何
  - 向量 & 矩阵
  - 平移、缩放、旋转、反射、错切变换，齐次坐标变换，逆变换和组合变换
- 第五章：viewing
  - Modeling和viewing变换
  - OpenGL用LookAt实现观察的方法
  - 投影(2个)
  - 视见体的定义(View Volume)
  - 视口变换和不变形处理方法(viewport)
- 第六章：lighting and shading
  - 局部光照模型
  - 光与材质
  - 光源属性
  - 着色方式(Gouraud和Phong)
- 第七章：texture
  - OBJ格式
  - maping方式
  - 透明物的混合映射
  - OpenGL中mipmaping和 billboard的概念
- 第八章：randering
  - 线段裁剪(2)
  - 多边形裁剪
  - 直线扫描转换(2)
  - 三角形扫描转换
  - 4个颜色模型

第四章：transformation

点是否在三角形内部（平面）

$$Q \in \triangle P_0P_1P_2\text{（逆时针）} \iff \begin{cases} N \cdot ((P_1 - P_0) \times (Q - P_0)) > 0 \\ N \cdot ((P_2 - P_1) \times (Q - P_1)) > 0 \\ N \cdot ((P_0 - P_2) \times (Q - P_2)) > 0 \end{cases}$$

如何判断一个多面体是一个凸多面体？

（fake）用点到面的距离判断：对于所有面，如果其他顶点都在该面的同一边，则为凸多面体，否则不是。

坐标系变换

$$单位向量：i, j, k => u, v, n$$
$$P' = MP$$
$$M = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{bmatrix}$$

齐次坐标变换

$$(x, y, z, w) \iff (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$$

以下变换均为 $P' = MP$，即左乘后为结果

坐标系变换

$$M = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ Q_x & Q_y & Q_z & 1 \end{bmatrix}$$

平移

$$T(d_x, d_y, d_z)P = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{bmatrix} = P'$$

放缩

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

旋转 - 绕轴

1. 绕 X 轴逆时针旋转：

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

2. 绕 Y 轴逆时针旋转：

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

3. 绕 Z 轴逆时针旋转：

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

旋转 - uvn

- u： 新 x 轴（右向量）
- v： 新 y 轴（上向量）
- n： 新 z 轴（前向量）

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad V_{uvn} = RV_{xyz}$$

$$R' = R^T = R^{-1} = \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad V_{xyz} = R'V_{uvn}$$

模型变换

将模型从自己的坐标系放置于场景坐标系中

$$M = T_{位移}R_{旋转}S_{放缩}$$

错切

$$沿x轴 \quad M = \begin{bmatrix} 1 & \cot\theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

投影

透视投影

简单

- eyes —— COP（Center of projection），位于原点。
- 投影平面位于 $z = d$
- 位于物体的 (x, y, z) 将投影至 (x_p, y_p, z_p)
- $$\begin{cases} x_p = \frac{x}{z/d} \\ y_p = \frac{y}{z/d} \\ z_p = \frac{z}{z/d} = d \end{cases} \qquad \begin{cases} h = z/d \\ x_h = h \cdot x_p = x \\ y_h = h \cdot y_p = y \\ z_h = h \cdot z_p = z \end{cases}$$

- $$\text{投影矩阵}M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$ $$M_p \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} \overset{\text{齐次坐标}}{=} \begin{bmatrix} x_p \\ y_p \\ d \\ 1 \end{bmatrix}$$

通常

- eyes —— COP（Center of projection），位于$(x_{prp}, y_{prp}, z_{prp})$。
- 投影平面位于 $z = z_{vp}$
- 位于物体的 (x, y, z) 将投影至 (x_p, y_p, z_{vp})
- 方程通过参数方程表示，慢慢推

平行投影

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

第五章：Viewing

lookat

- $LookAt(eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z)$
  - eye：相机位置
  - at：观察点
  - up：上方
- $Look(eye_x, eye_y, eye_z, n_x, n_y, n_z, v_{up_x}, v_{up_y}, v_{up_z})$
  - eye：相机位置
  - n：前方
  - up：上方
- 推导 uvn
  - n：新 z 轴（前向量） $V_{PN} = at - eye \quad n = V_{PN}/|V_{PN}|$
  - u：新 x 轴（左向量）$u = up \times n/|up \times n|$
  - v：新 y 轴（上向量）$v = n \times u/|n \times u|$
- 观察矩阵
  - $eye = (e_x, e_y, e_z)$
  - $$T = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- $$M = RT = \begin{bmatrix} u_x & u_y & u_z & -u \cdot eye \\ v_x & v_y & v_z & -v \cdot eye \\ n_x & n_y & n_z & -n \cdot eye \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- $$P' = MP$$
  $$P\text{为世界坐标系}$$
  $$P'\text{为观察坐标系}$$

视见体正则化

- 将所有投影的视见体归一至 NDC 空间 $(-1,-1,-1)-(1,1,1)$
- 统一施行裁减、投影

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

正交投影

$$(left, bottom, -near) \to (-1,-1,1)(right, top, -far) \to (1,1,-1)$$

1. 平移中心至原点
2. 放缩边长为 2
3. $$M = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{near+far}{near-far} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

透视投影

1. 取投影面 $d=-1, x=\pm1, y=\pm1$
2. 将 near 投影至 $z=-1$ ，将 far 投影至 $z=1$
3. $$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$
   $$\begin{cases} \alpha = \frac{near+far}{near-far} \\ \beta = \frac{2 \cdot near \cdot far}{near-far} \end{cases} \qquad PPT\text{版本（老师备注书上有错，存疑）}$$

视口 viewport

视口变换

- NDC -> window
- window -> viewport
- 都是平移放缩平移，$P' = T_2 S T_1 P$
- 不变形处理方法：
    - 将视见体的长宽比保持与 viewport 一致 （from 网络）
    - 改变window尺寸，使得 $S_x = S_y$

阴影

- 平行光源：$a(x+\alpha d_x)+b(y+\alpha d_y)+c(z+\alpha d_z)+d=0$
- 点光源 （光源平移至原点）：$a(\alpha x)+b(\alpha y)+c(\alpha z)+d=0$

第六章：Lighting and Shading

局部光照

- 仅仅关注物体和光源的相互关系
- Phong 光照模型
    - 环境光 (ambient) + 漫反射 (diffuse) + 高光 (specular)
    - 环境光：$A = L_0 + \sum_{Light}(L_A * C_A)$
        - L_0：场景环境光
        - L_A：环境光
        - C_A：材质环境光
    - 漫反射： $D = \sum_{Light} L_D * C_D * (L \cdot N)$
        - L：光向量（反射点指向光源的单位向量）
        - N：法向量
    - 高光：$S = \sum_{Light} L_S * C_S * (V \cdot R)^K$
        - 观察向量：反射点指向眼睛的单位向量
        - 反射向量：$R = L - 2N(N \cdot L)$
        - K：材质相关系数
- Blinn-Phong
    - 优化了相机和光照同侧时，点乘为负数，光照突然消失的现象
    - 取 $H = \frac{L+V}{|L+V|}$ ，有 $2\angle NH = \angle VR$
    - $S = \sum_{Light} L_S * C_S * (N \cdot H)^K$

光源类型

- 点光源
- 平行光源
- 聚光灯
- 环境光源

光衰减

- 点光源
  - 物理理论：$A_f = \frac{1}{d^2}$
  - 实践：$A_f = \frac{1}{a+bd+cd^2}$
    - a, b, c 为实践总结的参数
    - d 为距离
- 平行光源： $A_f = 1$

着色 Shading

平面着色 Flat (Constant) shading

每个三角形，只会选择一个点进行计算，整个三角形都采用一个结果。

高效，但是在平面相交的地方，存在颜色突变

Gouraud (Smooth) Shading

- 逐顶点着色 (per-vertex shading)
- 对三角形的每个顶点进行着色计算
- 每个顶点的法线，可能是三角形的法线，也可能是相邻三角形法线的平均值
- 通过对顶点插值计算面的颜色
- 两点之间插值：$Color(C) = \frac{|BC|}{|AB|} \times Color(A) + \frac{|AC|}{|AB|} \times Color(B)$
- 三角形插值：先插边，再用边上的两点插面（通常选取面内所在行交出的两点）
- 优化计算：

$$\begin{cases} I_{P1} = (1-\alpha_1) \times I_A + \alpha_1 \times I_B \\ I_{P2} = (1-\alpha_2) \times I_A + \alpha_2 \times I_B \end{cases} \implies \begin{aligned} I_{P2} &= I_{P1} + (\alpha_2 - \alpha_1)(I_B - I_A) \\ &= I_{P1} + \Delta I_{BA} \times \Delta\alpha \end{aligned}$$

Phong Shading

- 逐片元着色 (per-pixel shading)
- 对点的法向量进行插值，再对每个点计算光照
- Gouraud 需要将表面切割成非常小的块以获得满意的结果。
- Phong 不需要过度切割，但需要更多的计算。效果比 Gouraud 好

全局光照

- 光线追踪 Ray Tracing
- 辐射度算法 Radiosity

第七章 texture

OBJ格式

数据格式	意思
v x y z	顶点坐标 (x, y, z)
vt u v [w]	纹理坐标 $u, v \in [0, 1]$
vn x y z	顶点法向量
f v1[/vt1][/vn1] v2[/vt2][/vn2] v3[/vt3][/vn3] ...	面元

贴图方法 Mapping Method

- 纹理贴图 Texture Mapping
    - 使用图像填充表面
- 环境贴图 Environment Mapping
    - 将周围环境保存为贴图
    - 可以模拟高反射的表面
- 法向贴图 Bump maps
    - 在渲染过程中改变法向量
    - 使用像素着色器（fragment shader）独立处理每个像素

贴图坐标

- 参数坐标：用于对曲线和曲面建模
- 纹理坐标 $(0, 0) - (1, 1)$：用于映射的图像中的点
- 对象或世界坐标：从概念上讲，映射发生的位置
- 窗口坐标（像素）：最终图像的位置

正向映射

- 将纹理映射至物体上
- 理论方式，需要计算的像素点过多

$$\begin{cases} x = f_x(s, t) \\ y = f_y(s, t) \\ z = f_z(s, t) \end{cases} \qquad \begin{aligned} s, t \quad & \text{为纹理坐标} \\ x, y, z \quad & \text{为世界坐标} \end{aligned}$$

反向映射

- 给定像素、物体上的点，求解其纹理坐标
- 稍微现实点，但所需映射函数很难找到

$$\begin{cases} s = f_s(x, y, z) \\ t = f_t(x, y, z) \end{cases} \qquad \begin{aligned} s, t \quad & \text{为纹理坐标} \\ x, y, z \quad & \text{为世界坐标} \end{aligned}$$

两步映射

1. 把纹理映射到一个简单的三维中间表面上，如球面、圆柱体或者立方体表面
2. 把带有映射为例的中间表面映射到我们需要绘制的对象表面上

圆柱体映射

$$\begin{cases} x = r\cos u \\ y = v/h \\ z = -r\sin u \end{cases} \qquad \begin{matrix} s \to u = 0..2\pi \\ t \to v = 0..h \\ r\text{为固定值} \end{matrix}$$

球面映射

- 以与圆柱体类似的方式，但必须决定放置失真的位置（墨卡托投影）
- 通常用于环境贴图

$$\begin{cases} x = r\sin\alpha\cos\theta \\ y = r\cos\alpha \\ z = -r\sin\alpha\sin\theta \end{cases} \qquad \begin{matrix} \alpha = 0..\pi \\ \theta = 0..2\pi \\ r\text{为固定值} \end{matrix}$$

立方体映射

- 通过简单的正交投影易于使用
- 也是通常用于环境贴图

旋转体映射

- 取旋转的曲线的参数方程

$$\begin{gathered} (x, y, z) = f(t) \quad t \in [0, 1] \\ s \to \theta = 0..2\pi \\ t \to t = 0..1 \end{gathered}$$

混合映射

- 使用RGBA，标记颜色的透明度
- 1 完全不透明 0 完全透明
- 使用 Color Buffer 储存颜色属性，供混合不同颜色
- 多个面相交时，分割成小面，以供正常渲染

Texture Mapping in OpenGL

- wrapping Mode
  - 超出0.0~1.0范围的纹理坐标如何处理
  - 重复，镜像重复，延长，留空
- Texture Filtering
  - 将整数的像素坐标转换为浮点型纹理坐标，而后的像素取值
  - 邻近点：取最近的像素点
  - 双线性：取上下左右四个像素点插值
  - 三线性：解决浮点数mipmap等级，在双线性基础上对相邻mipmap取平均值
  - 各项异性
- mipmaping
  - 预先渲染一系列2倍数缩小的纹理序列
  - 纹理采样时依据物体的大小选择图像
  - 减少纹理采样时的失真
- billboard
  - 3D 世界中的 2D 元素，例如血条

    - 平面图像，通常有些部分为透明
    - 始终朝向相机
- texture depth
    - 在一个表面上有多个贴图的时候，容易出现混杂
    - OpenGL提供了参数设定贴图深度，保证正常渲染

第八章 randering

线段裁剪

Cohen-Sutherland

基础原理

- 依据线段顶点与四周直线的关系，判断是否保留

顶点关系	线段情况	操作
都在四条直线内	全在内部	全部保留
同在某条直线外部	全在外部	全部舍弃
有一个点都在四条直线内部	部分在内部	裁剪
既不都在内部，也不同在某条外部	可能部分在内部（斜挎角落）	裁剪再判断

编码加速 Outcodes

- 对每个顶点进行 01 编码：TBRL
- 直线内部为0，外部为1
- 原理转换
    - 线段完全保留 $\begin{aligned}&\Longleftrightarrow code_1 == 0 \quad \&\& \quad code_2 == 0\\&\Longleftrightarrow code_1|code_2 == 0\end{aligned}$
    - 线段完全放弃 $\Longleftrightarrow \quad code_1\&code_2 \neq 0$
    - 其他：进行裁剪再判断
- 不同的位值，代表需要在此裁剪

劣势

- 在大多数应用程序中，剪切窗口相对于整个数据的大小来说很小
- 大多数线段在窗口的一侧或多侧之外，需要根据它们的编码来消除
- 需要执行多个步骤来缩短的线段，效率低下

3D 下的裁剪算法

- 使用 6 位的编码
- 使用平面对线段进行裁剪

梁友栋-Barsky

- 使用参数方程裁剪线段

流程

- 取直线参数方程

$$\begin{cases} x = x_1 + t * (x_2 - x_1) \\ y = y_1 + t * (y_2 - y_1) \end{cases} \quad 0 \le t \le 1$$

- 定义始边、终边。用以切割线段
  - 始边：靠近s1的窗边界线
  - 终边：靠近s2的窗边界线

$$\begin{cases} x_2 - x_1 \ge 0 & \iff x_L\text{为始边，}x_R\text{为终边} \\ x_2 - x_1 < 0 & \iff x_R\text{为始边，}x_L\text{为终边} \end{cases}$$
$$\begin{cases} y_2 - y_1 \ge 0 & \iff y_B\text{为始边，}y_T\text{为终边} \\ y_2 - y_1 < 0 & \iff y_B\text{为终边，}x_T\text{为始边} \end{cases}$$

- 参数化裁剪线段
  - 始边参数为 t_1', t_1''，则 $t_1 = \max\{0, t_1', t_1''\}$
  - 终边参数为 t_2', t_2''，则 $t_2 = \min\{1, t_2', t_2''\}$
  - 当 $t_1 < t_2$ 时，裁剪所得直线为 $t \in [t_1, t_2]$
  - 当 $t_1 > t_2$ 时，直线不可见
- 始边、终边参数求解

$$\begin{cases} x_L \le (x_2 - x_1) * t + x_1 \le x_R \\ y_B \le (y_2 - y_1) * t + y_1 \le y_T \end{cases} \Rightarrow \quad t * p_k \le q_k \quad k = 1, 2, 3, 4$$

$$\begin{aligned} p_1 &= -(x_2 - x_1) & q_1 &= x_1 - x_L \\ p_2 &= +(x_2 - x_1) & q_2 &= x_R - x_1 \\ p_3 &= -(y_2 - y_1) & q_3 &= y_1 - y_B \\ p_4 &= +(y_2 - y_1) & q_4 &= y_T - y_1 \end{aligned}$$

  - 由之前始终边分类得
    - 若 $p_k < 0$，则 t_k 为始边的参数
    - 若 $p_k > 0$，则 t_k 为终边的参数
    - 若 $p_k = 0$，$q_k < 0 \iff$ 线段不可见

优点

- 和 $Cohen - Sutherland$ 一样容易判断接受和拒绝
- 通过参数方程，只用判断一次，不用递归判断
- 容易拓展至 3D

多边形裁剪

Sutherland-Hodgman

- 用四个窗边界线依次裁剪多边形的所有边

扫描转化

- 将几何图形的解析表示转化为像素点的点阵表示
- 依据给定的顶点，算出哪些像素位于图形上
- 算出的 Fragment 具有依据多边形算出的：位置、颜色和纹理坐标等属性

直线扫描转化

DDA Algorithm

- 原理
  - 定义线段的顶点为 (x_1, y_1) 和 (x_2, y_2) （$x_1 < x_2$）
  - 对于 $x \in [x_1, x_2]$，取最靠近线段的像素点
- 逐像素增量计算（避免乘法）

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + k \end{cases}$$

- 当 $|k| > 1$ 时（ y轴增量较大），选择递增 y
- 当 $x_1 > x_2$ 时，既可以交换，也可以选择采取递减

Bresenham Algorithm

- 考虑 $0 < |k| \le 1 \;\&\; x_1 < x_2$
- 通过计算与 $(x_{i+1}, f(x_{i+1}))$ 的偏移距离判断 $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i) \; or \; (x_i + 1, y_i + 1)$

$$\begin{aligned} d_1 &= f(x_{i+1}) - y_i \\ &= -y_i + k(x_i + 1) + b \end{aligned}$$

$$\begin{aligned} d_2 &= y_i + 1 - f(x_{i+1}) \\ &= y_i + 1 - k(x_i + 1) - b \end{aligned}$$

$$d_1 - d_2 = 2k(x_i + 1) - 2y_i + 2b - 1$$

$$\begin{cases} d_1 - d_2 > 0 & \text{选择}(x_i + 1, y_i + 1) \\ d_1 - d_2 < 0 & \text{选择}(x_i + 1, y_i) \\ d_1 - d_2 = 0 & \text{选择任意一点} \end{cases}$$

$$\text{令} \begin{cases} \Delta_x = x_2 - x_1 > 0 \\ \Delta_y = y_2 - y_1 > 0 \end{cases}$$

$$\begin{aligned} p_i &= \Delta_x * (d_1 - d_2) \quad \text{（省去除法运算）} \\ &= \Delta_x(2\Delta_y/\Delta_x(x_i + 1) - 2y_i + 2b - 1) \\ &= 2\Delta_y x_i - 2\Delta_x y_i + \Delta_x(2b - 1) + 2\Delta_y \end{aligned}$$

$$\begin{aligned} p_{i+1} &= 2\Delta_y x_{i+1} - 2\Delta_x y_{i+1} + \Delta_x(2b - 1) + 2\Delta_y \\ &= p_i + 2\Delta_y(x_{i+1} - x_i) - 2\Delta_x(y_{i+1} - y_i) \\ &= p_i + 2\Delta_y - 2\Delta_x(y_{i+1} - y_i) \end{aligned}$$

$$\begin{aligned} p_1 &= 2\Delta_y x_1 - 2\Delta_x y_1 + \Delta_x(2b - 1) + 2\Delta_y \\ &= 2\Delta_y x_1 - 2\Delta_x((\Delta_y/\Delta_x)x_1 + b) + \Delta_x(2b - 1) + 2\Delta_y \\ &= 2\Delta_y x_1 - 2\Delta_y x_1 - 2\Delta_x b + \Delta_x(2b - 1) + 2\Delta_y \\ &= 2\Delta_y - \Delta_x \end{aligned}$$

- p_i 的符号与 $d_1 - d_2$ 相同

$$p_i > 0 \begin{cases} y_{y+1} = y_i + 1 \\ p_{i+1} = p + i + 2\Delta_y - 2\Delta_x \end{cases}$$

$$p_i < 0 \begin{cases} y_{y+1} = y_i \\ p_{i+1} = p + i + 2\Delta_y \end{cases}$$

三角形扫描转换

- 理论：找出与多边形相交的扫描线，计算重叠段并设置像素
- 重排序：计算所有边与扫描线的交点，重排序（或者使用类似桶的数据结构）得到每一个扫描线的焦点
- 如果恰好有两条边与一条扫描线相交，我们需要确定扫描线上两条边像素之间的所有像素的颜色。这些像素一起称为一个 Fragment

三角形详细计算

- 计算扫描线与三个边的交点，再依据 X 坐标排序
- 顶点的焦交点：如果相连两条边位于扫描线同一侧则算两个交点，否则算一个

隐藏表面消除

- Painter’s Algorithm
  - 从远至进一层一层画，最终将画像完整
- z-Buffer Algorithm
  - 开辟深度缓存 Depth Buffer ，记录像素点的深度数据
  - 渲染、叠加时只保存最近的像素

抗锯齿 Antialiasing

- 用离散像素对直线的近似会产生锯齿，最直观的就是“画图”中的直线。
- 一个简单的抗锯齿方法，是对颜色采取比例覆盖的方法

$$Color_{new} = \alpha \times Color_{current} + (1-\alpha) \times Color_{existing}$$

多边形混叠 Polygon Aliasing

- 混叠问题对于多边形也很严重，会导致忽略较小（相较于像素）的多边形
- 解决方案： 像素的颜色取决于多个多边形的颜色，同样采取比例覆盖

颜色模型

人眼

人也中含有两种主要的细胞参与视觉

- 杆细胞：感受光强（luminance ）和亮度（brightness）
- 锥细胞：感受色度（chroma）和颜色（color）
- 对红绿蓝敏感

从人的主观角度，颜色包含三个要素：

1. 色调 hue/chrome
2. 饱和度 saturation
3. 明亮度 luminance

RGB - 正方体

- 三个分量 R、G、B
- 亮度 $Luminance = 0.30 * Red + 0.59 * Green + 0.11 * Blue$
- 如果具有色弱、色盲，也可以通过亮度区分颜色

CMYK

- C：Cyan = 青色
- M：Magenta = 品红色
- Y：Yellow = 黄色
- K：blacK=黑色
- 是 RGB 的补色，用于反射模型，而非 RGB 的发射模型

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

HSV - 圆锥

- Hue（色调、色相）
- Saturation（饱和度、色彩纯净度）
- Value（明度）越低越黑

HLS - 纺锤

- hue（色相）
- saturation（饱和度）
- lightness（亮度）越高越白

第九章 Modeling and Hierarchy

场景树

- 依据场景中的物体的从属关系，建出一棵树
- 如果允许单个物体被多次利用，则可以建出一个 DAG