计算图形学期末复习

- 成像系统
- 物理成像系统
  *人类视觉系统
  *针孔照相机
- 虚拟照相机系成像系统
  *成像模型
  *全局和局部照明

- 物理图像——图片
  *矢量图/光栅图

- 成像四要素
  *Objects（对象，几何形状）
  *Viewer（观察者）
  *Light source（光源）
  *materials（材质）

- 三色理论
  *使用三种基色或原色来近似所感知到的所有颜色
  *计算图图形学中所谓的三原色不一定与人眼所感知的三种值恰好完全匹配

- 加色系统以及减色系统
  *加色
  ——通过把三原色的值加在一起形成一种颜色
  ——原色为 RGB
  ——基于自身发光的颜色系统
  *减色
  ——通过在白光中过滤掉青色，品红色和黄色形成最终的颜色
  ——基于反光的颜色系统

- 针孔照相机
  *(x,y,z)点的投影(xp, yp, -d)
  xp = -d*x/z
  yp = -d*y/z
  *视域或视角：胶片上能成完整像的最大物体所张的角度
  θ=2 tan-1(h/2d)
  *无穷景深：不论远近都会清晰成像
  *缺点：
  ——针孔太小——》单条光线——》用透镜替换小孔
  ——视角不能调节——》使用焦距可调的透镜》胶片平面与小孔距离可调

- 虚拟照相机
  - 投影线：对象上一点到透镜中心的连线

• 投影中心(COP)：透镜中心
• 投影平面：移至透镜前的虚拟成像平面
• 对象上一点的像位于投影线与投影平面的交点
• 图像大小是有限的
- 场景中位于视角/视域体中对象在胶片上成像
• 在投影平面内设置一个裁剪矩形(clipping rectangle)或裁剪窗口(clipping window)
• 给定投影中心的位置、投影平面的位置和方向，还有裁剪窗口的大小，就能确定哪些对象会在图像中出现。

- 优势
    • 把对象、观察者和光源区分开
    • 二维图形是三维图形的一个特殊情形
    • 可以得到简单的软件 API (Application programming interface)
    - 指定对象、光源、照相机、材料属性
    - 由 API 的实现确定最终的图像
    • 可以得到快速的硬件实现
    • 主流 API， 如 OpenGL 和 Direct3D 都是基于该模型

- 全局光照与局部光照
  *全局光照不能对每个对象独立地计算它的颜色与阴影
  *全局光照模型光线跟踪、光子映射
  *局部光照模型——Phong 光照模型——交互式或实时图形渲染
  *光线跟踪方法的不足之处
    •原则上它可以得到全局光照效果，例如阴影和多重反射，但是光线跟踪速度很慢，
    •不适用于交互系统的需要
    •借助 GPU 实现的光线跟踪接近于实时

- Pipeline Architeture
  *主要步骤
  -顶点处理
  -裁剪和图元装配
  -光栅化
  -片段处理

- 顶点处理
  *图元的类型和顶点集定义了场景的几何
  *坐标的每个变换相当于一次矩阵惩罚
  *顶点处理器也计算顶点的颜色

- 投影 Projection
    • 把三维观察者位置与三维对象结合在一起确定二维图像的构成
    - 透视投影：所有投影线交于投影中心
    - 平行投影：投影线平行，投影中心在无穷远，用投影方向表示

- 在顶点处理步中，对各个顶点的处理是相互独立的

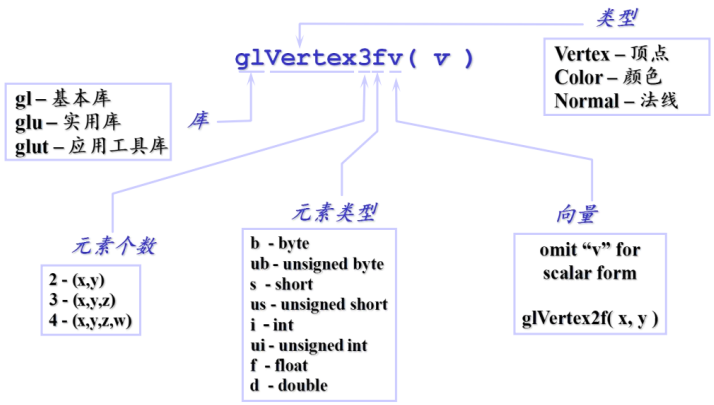
- 图元装配
  *在进行裁剪的光栅化处理之前，顶点必须组装成集合对象

- 裁剪
  不在视景体中的对象要从场景中裁剪掉

- 光栅化
  - 如果一个对象不被裁掉，那么在帧缓冲区中相应的像素就必须被赋予颜色
  - 光栅化程序为每个图元生成一组片段
  - 片段是“潜在的像素”
  - 在帧缓冲区中有一个位置
  - 具有颜色和深度属性
  - 光栅化程序在对象上对顶点的属性进行插值得到片段的属性

- 片段处理
  - 对片段进行处理，以确定帧缓冲区中相应像素的颜色
  - 颜色可以由纹理映射确定，也可以由顶点颜色插值得到
  - 片段可能被离照相机更近的其它片段挡住
    - 隐藏面消除

- OpenGL API



- OpenGL 程序示例

```
#include <GL/glut.h>

int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("简单示例“);
        glutDisplayFunc(display);
        init();
        glutMainLoop();
}
```

glut.h包含了gl.h和glu.h

窗口属性

设置回调函数

设置初始状态

一切就绪，进入消息循环

- 矩阵与向量
- 向量
  *向量的归一化

$$\frac{\boldsymbol{a}}{|\boldsymbol{a}|}=\frac{(a_1,a_2,a_3)^T}{\sqrt{a_1^2+a_2^2+a_3^2}}$$

- 矩阵
  -矩阵与向量的乘法
  -投影观点

$$\begin{bmatrix}|\\ \mathbf{y}\\ |\end{bmatrix}=\begin{bmatrix}-\ \mathbf{r}_1\ -\\ -\ \mathbf{r}_2\ -\\ -\ \mathbf{r}_3\ -\end{bmatrix}\begin{bmatrix}|\\ \mathbf{x}\\ |\end{bmatrix}$$

  -加权观点

$$\begin{bmatrix}|\\ \mathbf{y}\\ |\end{bmatrix}=\begin{bmatrix}| & | & |\\ \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3\\ | & | & |\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$$

$$\mathbf{y}=x_1\mathbf{c}_1+x_2\mathbf{c}_2+x_3\mathbf{c}_3.$$

- 模型变换
  -平移、旋转、缩放
  -绕任意轴旋转
  -复杂变换
- 相机变换（视点变换）
- 平移

*平移由平移向量 d 确定

-三个自由度

-P'=P+d

•应用在某个标架中的齐次坐标表示

$\mathbf{p}=[\,x\ y\ z\ 1]^T$

$\mathbf{p}'=[x'\ y'\ z'\ 1]^T$

$\mathbf{d}=[d_x\ d_y\ d_z\ 0]^T$

注意：这里是四维的齐次坐标，用的是点=点+向量

第四维度为1表示点，为0表示方向（向量）

因此 $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ 或者

$x'=x+d_x$

$y'=y+d_y$

$z'=z+d_z$

15

- 平移矩阵

•可以用在齐次坐标中一个4x4的矩阵**T**表示平移：$\mathbf{p}'=\mathbf{Tp}$ 其中

$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

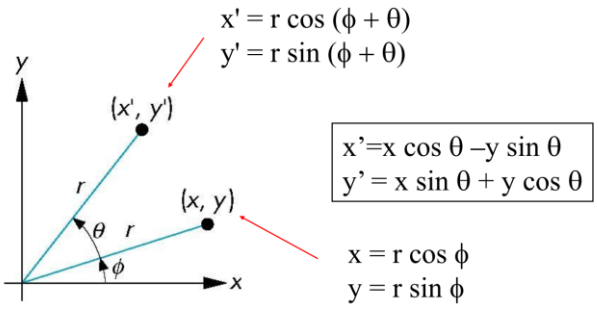
请自行动手乘一下。

图形学中大量采用这种形式，这种形式更容易实现，因为所有的仿射变换（旋转和平移）都可以用这种形式统一表示，矩阵乘法可以复合在一起

- 二维旋转

考虑绕原点旋转θ度

- 半径保持不变，角度增加了θ



- 三维旋转
- 绕 z 轴旋转

在三维空间中绕 z 轴旋转， 点的 z 坐标不变

- 等价于在 z=常数的平面上进行二维旋转

- 其齐次坐标表示为

p’=Rz(q)p

*旋转矩阵

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

- x 轴与 y 轴

  - 对于绕x轴的旋转， x坐标不变

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

  - 对于绕y轴的旋转， y坐标不变

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

- 缩放

  • 沿每个坐标轴伸展或收缩(原点为不动点)

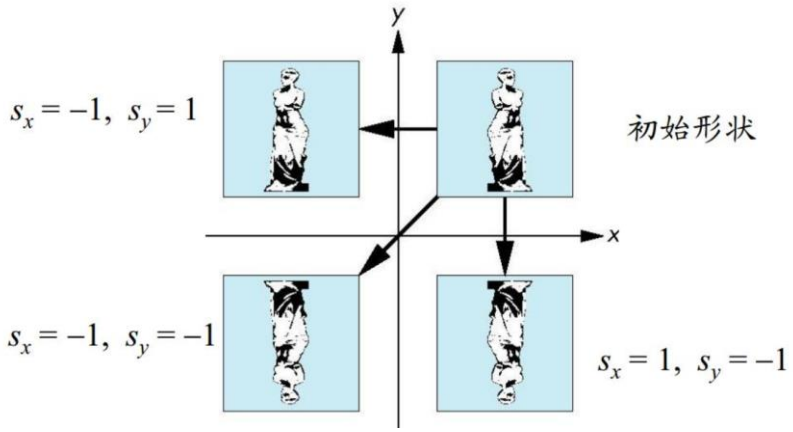
$$x' = s_x x$$
$$y' = s_y y$$
$$z' = s_z z$$

$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 反射

- 特殊的缩放
  - 缩放系数为负数

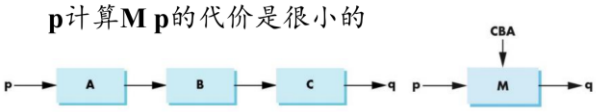


- 逆变换

  虽然可以直接计算矩阵的逆，但根据几何意义可以给出各种变换的逆
  - 平移: $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
  - 旋转: $\mathbf{R}^{-1}(q) = \mathbf{R}(-q)$
    - 对所有旋转矩阵成立
    - 注意 $\cos(-q) = \cos(q)$, $\sin(-q) = -\sin(q)$

      $\mathbf{R}^{-1}(q) = \mathbf{R}^{T}(q)$
  - 缩放: $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

- 变换的复合
  - 可以通过把旋转、平移与缩放矩阵相乘从而形成任意的仿射变换
  - 由于对许多顶点应用同样的变换，因此构造矩阵$\mathbf{M} = \mathbf{C}\,\mathbf{B}\,\mathbf{A}$的代价相比于对许多顶点$\mathbf{p}$计算$\mathbf{M}\,\mathbf{p}$的代价是很小的



  - 难点在于如何根据应用程序的要求构造出满足要求的变换矩阵
  - 注意在右边的矩阵是首先被应用的矩阵
  - 从数学的角度来说，下述表示是等价的

    $\mathbf{p}' = \mathbf{ABCp} = \mathbf{A}(\mathbf{B}(\mathbf{Cp}))$
  - 变换的顺序是不可交换的
    - 矩阵乘法不满足交换律

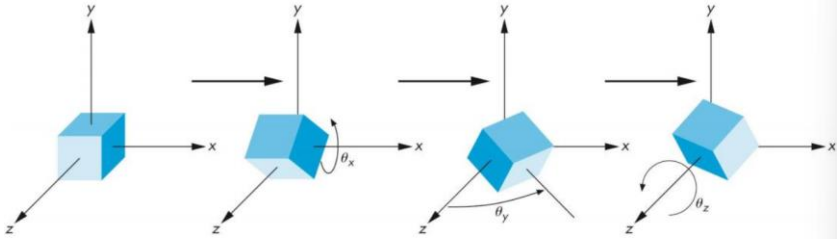
- 绕原点的一般旋转
  - 绕过原点任一轴旋转θ角可以分解为绕x, y, z轴旋转的复合

$$\mathbf{R}(\theta) = \mathbf{R}_z(\theta_z)\,\mathbf{R}_y(\theta_y)\,\mathbf{R}_x(\theta_x)$$

$\theta_x\,\theta_y\,\theta_z$被称为欧拉角（Euler angles）

  - 注意：因为矩阵乘法不具有交换性，因此调换z,y,x的顺序将导致不同的旋转效果。调换顺序之后，如果为了得到原来的旋转效果，则旋转角度应相应改变。

- 绕任意轴的旋转



  - 策略：先经过两次旋转使旋转轴**r**与z轴对齐，然后绕z轴旋转角度θ

$$\mathbf{R} = \mathbf{R}_x(-\theta_x)\mathbf{R}_y(-\theta_y)\mathbf{R}_z(\theta)\,\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

  - 绕任意轴

$$\boldsymbol{r} = (r_x, r_y, r_z)^T$$

$\boldsymbol{r}$为归一化的旋转轴

$$\mathbf{R} = \begin{pmatrix} \cos\phi + (1-\cos\phi)r_x^2 & (1-\cos\phi)r_x r_y - r_z\sin\phi & (1-\cos\phi)r_x r_z + r_y\sin\phi \\ (1-\cos\phi)r_x r_y + r_z\sin\phi & \cos\phi + (1-\cos\phi)r_y^2 & (1-\cos\phi)r_y r_z - r_x\sin\phi \\ (1-\cos\phi)r_x r_z - r_y\sin\phi & (1-\cos\phi)r_y r_z + r_x\sin\phi & \cos\phi + (1-\cos\phi)r_z^2 \end{pmatrix}$$

扩展到齐次坐标？

$$\mathbf{M} = \mathbf{T}(\mathbf{p}_0)\,\mathbf{RT}(-\mathbf{p}_0)$$

- 模型视图变换

- 相机变换（视点变换）

  - the eye position **e**,
  - the gaze direction **g**,
  - the view-up vector **t**.

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$

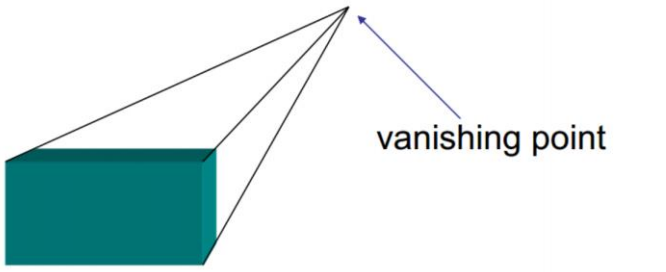
$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|},$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 灭点

  - 对象上（不平行于投影面）的平行线在投影后交于一个灭点(vanishing point)
  - 手工绘制简单透视投影时要利用这些灭点



注意区分三点透视、两点透视、单点透视

- 优缺点

  - 同样大小的对象，离视点越远，投影结果就越小(diminuition)
    - 看起来更自然
  - 直线上等距的几点投影后不一定等距—非均匀缩短(nonuniform foreshortening)
    - 借助透视投影图测量尺寸较平行投影困难
  - 只有在平行于投影面的平面上角度被保持
  - 相对于平行投影而言，更难用手工进行绘制（但对计算机而言，没有增加更多的困难）
  - 主要应用在动画等真实感图形领域

L5（2）正交变换、透视变换自行观看
熟悉视图设置等

- 明暗着色

  —多边形网格模型、隐藏面消除、模拟光照、光滑明暗处理、镜面光、阴影、纹理映射、光线跟踪、辐射度方法

  -光源类型：点光源、聚光灯、无穷远光源、环境光

- Phong 光照模型

  模型/向量

  • 可以快速计算的局部光照模型

  • 有三类分量

  – 漫反射光

  – 镜面反射光

  – 环境光

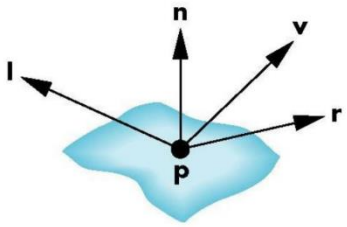
  • 使用四个向量

  – 入射光方向**l**

  – 视点方向**v**

  – 法向**n**

  – 理想反射方向**r**



  -光源模型、反射系数、理想反射、朗伯表面

  明暗处理的具体实现自行了解

- 光线跟踪

  -空间加速结构、均匀分割网格、八叉树、KD Tree、BVH

  -抗锯齿——随机采样

  -Whitted-Sytle 方法的不足

  - Whitted-Style 光线跟踪
    - 对每个点， 计算局部光照明
    - 如果是镜面或者折射面， 则继续递归计算
    - 如果不是镜面或者折射面， 则仅包含局部光照明
  - 定性、 不准确
  - 没有反映出完整的全局光照明效果， 尤其是当物体表面不是镜面或者折射面时

- 辐射度学

  -辐照度和辐出度

  -辐射强度

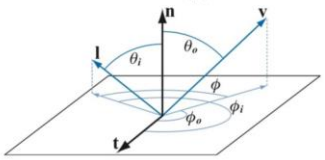
  -（理解到光衰减和距离的关系即可）

- 入射光与反射光

  -BRDF 双向反射分布函数

  • 定义：

$$f(l,v)=\frac{dL_o(v)}{dE(l)}$$



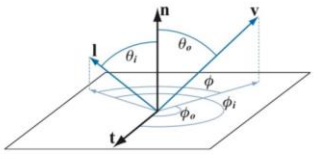
  • 如何理解：入射光l打到点p的一个微小面元，形成辐照度E, E按照一定比例朝v方向反射，把这个比值定义为入射方向l，出射方向v的BRDF值。

  • 经过推导，朗伯表面的BRDF=c/π, c是[0,1]间的常数。

-反射方程

• 反射方程：

$$L_o(v) = \int_\Omega f(l,\ v)L_i(l)cos\theta_i d\omega_i$$



-渲染方程

$$L_o(v) = L_e + \int_\Omega f(l,\ v)L_i(l)cos\theta_i d\omega_i$$

-路径跟踪

$$L_o(v) = L_e + \int_\Omega f(l,\ v)L_i(l)cos\theta_i d\omega_i$$

自发光　　BRDF　　入射辐射率
考虑递归

*需要进行半球面积分，多次折射之后引起指数灾难

-蒙卡洛特模拟

- 路径跟踪伪代码示例

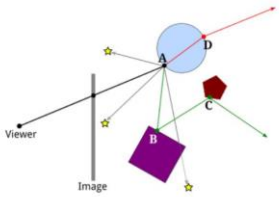
```
void Render(Image finalImage, count numSamples) {
  foreach (pixel in finalImage) {
    foreach (i in numSamples) {
      Ray r = camera.generateRay(pixel);
      pixel.color += TracePath(r, 0);
    }
    pixel.color /= numSamples;  // Average samples.
  }
}
```

```
Color TracePath(Ray ray, count depth) {
  if (depth >= MaxDepth) {
    return Black;  // Bounced enough times.
  }

  ray.FindNearestObject();
  if (ray.hitSomething == false) {
    return Black;  // Nothing was hit.
  }

  Material material = ray.thingHit->material;
  Color emittance = material.emittance;

  // Pick a random direction from here and keep going.
  Ray newRay;
  newRay.origin = ray.pointWhereObjWasHit;

  // This is NOT a cosine-weighted distribution!
  newRay.direction = RandomUnitVectorInHemisphereOf(ray.normalWhereObjWasHit);
```


```
Color TracePath(Ray ray, count depth) {
  if (depth >= MaxDepth) {
    return Black;  // Bounced enough times.
  }

  ray.FindNearestObject();
  if (ray.hitSomething == false) {
    return Black;  // Nothing was hit.
  }

  Material material = ray.thingHit->material;
  Color emittance = material.emittance;

  // Pick a random direction from here and keep going.
  Ray newRay;
  newRay.origin = ray.pointWhereObjWasHit;

  // This is NOT a cosine-weighted distribution!
  newRay.direction = RandomUnitVectorInHemisphereOf(ray.normalWhereObjWasHit);
```


蒙特卡洛采样解决指数爆炸问题

- 光线跟踪算法
  - 光线跟踪算法框架
    - Viewing (Primary) Ray
    - Secondary Ray
    - Shadow Ray



for each pixel **do**
  compute viewing ray
  if (ray hits an object with $t \in [0, \infty)$) **then**
    Compute $\mathbf{n}$
    Evaluate shading model and set pixel to that color
  else
    set pixel color to background color

- 光线与场景求交
  - 光线与球面求交
    - 光线方程：

$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$

    - 球面方程：

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0$$

    - 带入求解：

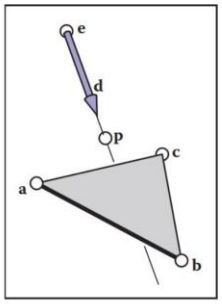
$$(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - R^2 = 0.$$

  - 光线与球面求交

$$(\mathbf{d} \cdot \mathbf{d})t^2 + 2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2 = 0$$

$$t = \frac{-\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}) \pm \sqrt{(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))^2 - (\mathbf{d} \cdot \mathbf{d})((\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2)}}{(\mathbf{d} \cdot \mathbf{d})}.$$

  - 光线与三角形求交



$$\begin{aligned} x_e + tx_d &= x_a + \beta(x_b - x_a) + \gamma(x_c - x_a), \\ y_e + ty_d &= y_a + \beta(y_b - y_a) + \gamma(y_c - y_a), \\ z_e + tz_d &= z_a + \beta(z_b - z_a) + \gamma(z_c - z_a). \end{aligned}$$

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

• 光线与三角形求交

$$\beta = \frac{\begin{vmatrix} x_a - x_e & x_a - x_c & x_d \\ y_a - y_e & y_a - y_c & y_d \\ z_a - z_e & z_a - z_c & z_d \end{vmatrix}}{|\mathbf{A}|},$$

$$\gamma = \frac{\begin{vmatrix} x_a - x_b & x_a - x_e & x_d \\ y_a - y_b & y_a - y_e & y_d \\ z_a - z_b & z_a - z_e & z_d \end{vmatrix}}{|\mathbf{A}|},$$

$$\mathbf{A} = \begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix}$$

$$t = \frac{\begin{vmatrix} x_a - x_b & x_a - x_c & x_a - x_e \\ y_a - y_b & y_a - y_c & y_a - y_e \\ z_a - z_b & z_a - z_c & z_a - z_e \end{vmatrix}}{|\mathbf{A}|},$$

- 像素着色

*Whitted 着色模型

$$I_\lambda = L_{a\lambda}k_a + \sum_{lights} f_{att}L_{p\lambda}[k_d(\vec{n}\cdot\vec{l}) + k_s(\vec{r}\cdot\vec{v})^n] + k_r I_{r\lambda} + k_t I_{t\lambda}$$

递归反射项　　递归折射项

- 前三部分为直接光照
- 后两部分为间接光照

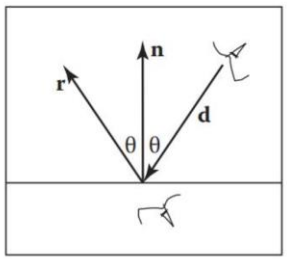
-阴影

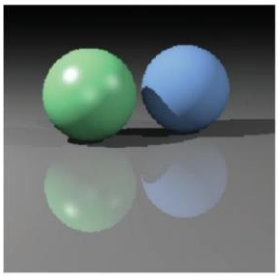
function raycolor(ray $\mathbf{e} + t\mathbf{d}$, real t_0, real t_1)
hit-record rec, srec
if (scene→hit($\mathbf{e} + t\mathbf{d}$, t_0, t_1, rec)) **then**
  $\mathbf{p} = \mathbf{e} + (\text{rec}.t)\,\mathbf{d}$
  color $c = \text{rec}.k_a\ I_a$
  if (not scene→hit($\mathbf{p} + s\mathbf{l}$, ϵ, ∞, srec)) **then**
    vector3 $\mathbf{h} = \text{normalized}(\text{normalized}(\mathbf{l}) + \text{normalized}(-\mathbf{d}))$
    $c = c + \text{rec}.k_d\ I \max(0, \text{rec}.\mathbf{n}\cdot\mathbf{l}) + (\text{rec}.k_s)\ I\ (\text{rec}.\mathbf{n}\cdot\mathbf{h})^{\text{rec}.p}$
  return c
else
  return background-color

-反射

$\mathbf{r} = \mathbf{d} - 2(\mathbf{d}\cdot\mathbf{n})\mathbf{n}$

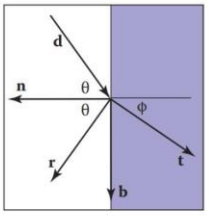
color $c = c + k_m$raycolor$(\mathbf{p} + s\mathbf{r}, \epsilon, \infty)$





-折射

• Snell's Law



$$n \sin\theta = n_t \sin\phi.$$

$$\mathbf{t} = \frac{n(\mathbf{d} + \mathbf{n}\cos\theta))}{n_t} - \mathbf{n}\cos\phi$$
$$= \frac{n(\mathbf{d} - \mathbf{n}(\mathbf{d}\cdot\mathbf{n}))}{n_t} - \mathbf{n}\sqrt{1 - \frac{n^2(1-(\mathbf{d}\cdot\mathbf{n})^2)}{n_t^2}}$$

• 根号内小于零？

- 几何的表示
  *隐式
  *显式
  *参数表示

- 贝塞尔曲线
  - 贝塞尔曲线（Bézier Curve）是一段 n 次多项式曲线，是构造自由曲线曲面的重要和基本方法之一。
  - 它具有许多优点，诸如保凸性，凸包性，曲线形状不依赖于坐标系的选择，人机交互手段灵活等。由于它能满足几何造型对曲线曲面的要求，因此在理论和应用上均得到了极大的重视和发展。

- **控制点控制曲线的基本形状**
- **要求曲线通过第 0 个和最后一个控制点**
- **曲线起始点和终止点处的切线与对应控制点切线一致**

线性贝塞尔曲线

• 给定点P0、P1两个控制点，线性贝塞尔曲线只是一条两点之间的直线。这条线由下式给出：

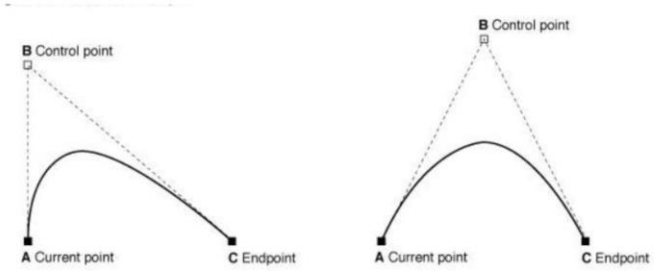
$$\mathbf{B}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , t \in [0,1]$$

• 等同于线性插值

二次方贝塞尔曲线

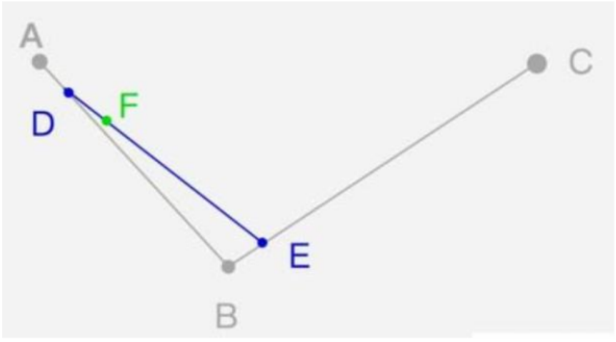
- 二次方贝塞尔曲线的路径由给定点P0、P1、P2三个控制点的函数B（t）得到：

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \ , t \in [0,1]$$



二次方贝塞尔曲线

- 二次方贝塞尔曲线与线性贝塞尔曲线的关系：
- 如下图：
  - D=(1-t)A+tB, E=(1-t)B+tC
  - F=(1-t)D+tE



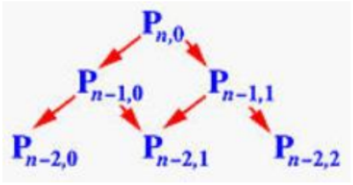
以此类推

de Casteljau算法

- 对于n+1个控制点，可生成n次贝塞尔曲线
- 采用递归算法

$$P_0^n = (1-t)P_0^{n-1} + tP_1^{n-1} \qquad t \in [0,1]$$

$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1,2,\cdots,n,\ i = 0,1,\cdots,n-k \end{cases}$$



点云

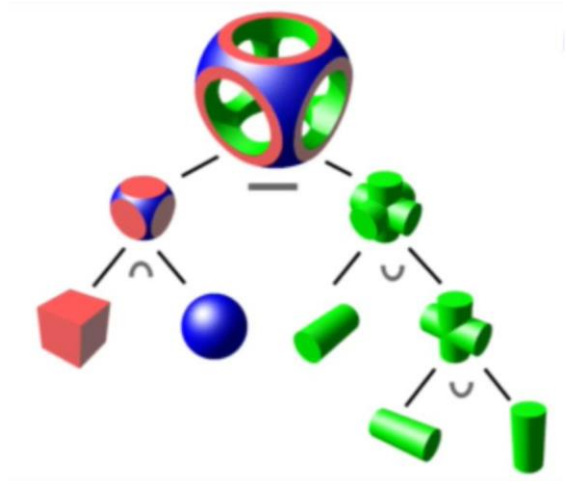
*点云的注册：将多个点云拼接成完整的点云

- 求最优的刚体变换， 使得两个点云可以重合

CSG

就是把复杂物体看出简单物体的加减

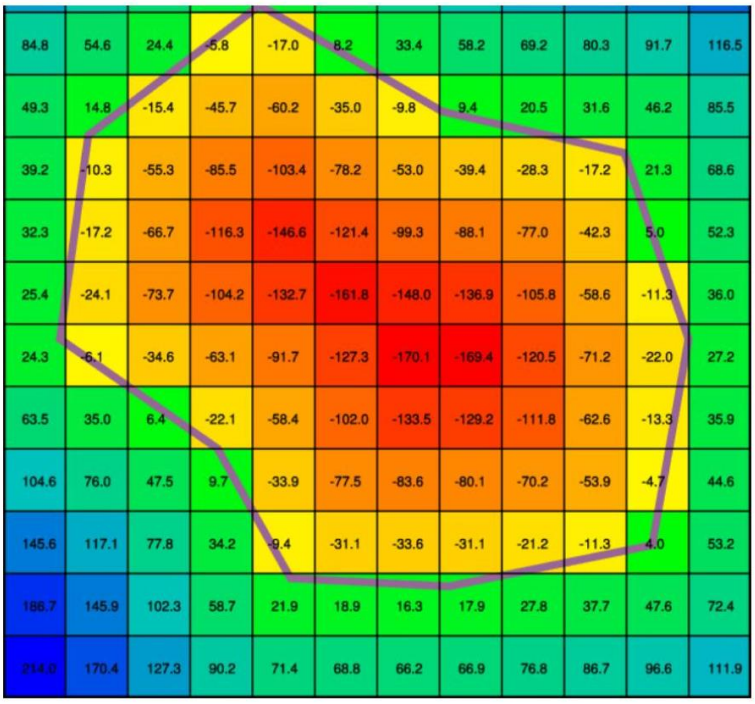
- Constructive Solid Geometry
- 体表示+布尔运算



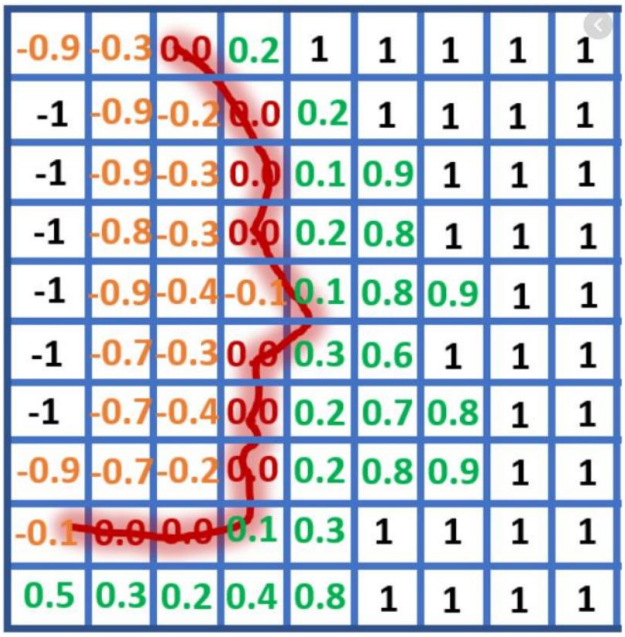
符号距离场

很简单，看图

- 符号距离场 (Signed Distance Field)



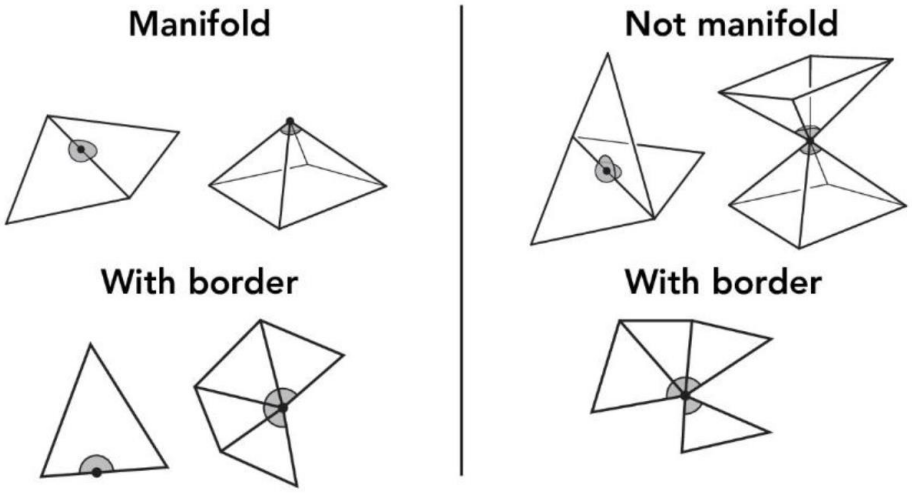
- 截断符号距离场 (Truncated Signed Distance Field)



了解点云和符号距离场的互相转换就行

- 二维流形

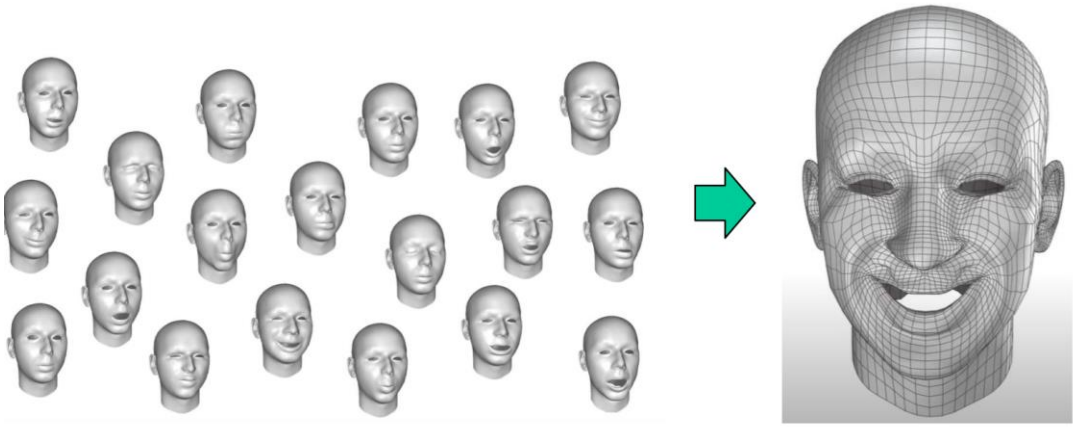
- 二维流形：一个表面，当用一个小球去切割时，得到一个二维的圆盘。



- 用小球去截取任意部位得到不割裂的圆盘

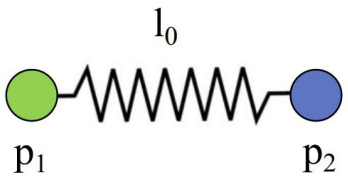
- 动画
- 关键帧插值
  *线性插值
  *非线性插值

- blendshape
  - 通过线性组合得到新的人脸表情
  - F=F0+w1*(F1-F0)+w2*(F2-F0)+…



- 仿真与模拟

- 质点弹簧系统-柔性物质的模拟



$$f_1 = -\left[k_s(|p_1 - p_2| - l_0) + k_d\left(\frac{(v_1 - v_2)\cdot(p_1 - p_2)}{|p_1 - p_2|}\right)\right]\cdot\frac{(p_1 - p_2)}{|p_1 - p_2|}$$

$$f_2 = -f_1$$