图形学

线性代数

点乘

结果是标量，一个数

$$\vec{a}\cdot\vec{b}=||\vec{a}||||\vec{b}||cos\theta$$

满足交换律、分配律和结合律

计算点乘结果：

$$\vec{a}\cdot\vec{b}=\begin{bmatrix}x_a\\y_a\end{bmatrix}\cdot\begin{bmatrix}x_b\\y_b\end{bmatrix}=x_ax_b+y_ay_b$$

点乘作用：

- 计算两个向量的余弦夹角
- 计算一个向量在另一个向量上的投影（还可以分解成两垂直的向量）
- 判断两个方向向量的有多接近
- 判断两个方向向量是否同向（都向前点乘为正，一前一后点乘为负，垂直为0）

叉乘

叉乘的结果是向量，其方向垂直于相乘的两向量，满足右手螺旋定则

$$||\vec{a}\times\vec{b}||=||\vec{a}||||\vec{b}||sin\phi$$

变换公式：

$$\vec{a}\times\vec{b}=-\vec{b}\times\vec{a}$$
$$\vec{a}\times\vec{a}=\vec{0}$$
$$\vec{a}\times(\vec{b}+\vec{c})=\vec{a}\times\vec{b}+\vec{a}\times\vec{c}$$
$$\vec{a}\times(k\vec{b})=k(\vec{a}\times\vec{b})$$

NOTE: 没有交换律

叉乘作用：

- 计算出垂直于平面的方向
- 判断两个向量的左右
- 判断是否在图形的内部

矩阵相乘

矩阵A(m,n)×B(n,p) = C(m,p)

变换公式：

$$(AB)C=A(BC)$$
$$A(B+C)=AB+AC$$
$$(A+B)C=AC+BC$$

NOTE: 没有交换律
单位矩阵就是I

$$(AB)^T=B^TA^T$$
$$AA^{-1}=A^{-1}A=I$$

$$(AB)^{-1} = B^{-1}A^{-1}$$

矩阵乘向量

矩阵在左边，列向量在右边，相当于把列向量当作一列的矩阵

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}$$

向量的点乘和叉乘都可以写成矩阵相乘的形式

点乘：

$$\vec{a} \cdot \vec{b} = \vec{a}^T \vec{b} = \begin{bmatrix} x_a & y_a & z_a \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = x_a x_b + y_a y_b + z_a z_b$$

叉乘：

$$\vec{a} \times \vec{b} = A^* b = \begin{bmatrix} 0 & -z_a & y_a \\ z_a & 0 & -x_a \\ -y_a & x_a & 0 \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}$$

变换（Transform）

线性变换（$\vec{a}^* = M\vec{a}$）：

- 缩放（scale）变换：乘上对角阵，对角的系数就是每一维的放缩系数
- 镜像（reflection）变换：乘上对角阵，对角需要镜像的维度上，其系数是-1
- 切变（shear）变换： 当y=0是水平偏移是0，y=1时水平偏移是1

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- 旋转（rotation）变换，以原点为中心，逆时针旋转θ度

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

一个矩阵的逆等于其转置，那就是正交矩阵。所以旋转变换的矩阵是正交矩阵。

然而，平移（translation）变换却不属于线性变换，平移一个向量无法通过和矩阵相乘得到结果

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

齐次坐标（homogeneous coordinates）的引入，就是不希望平移是一种特例
对于二维点，可以表示成$(x, y, 1)^T$
对于二维的向量，可以表示成$(x, y, 0)^T$（平移并不会影响向量的值）
那么平移变换也可以表示成矩阵和列相乘

$$\begin{bmatrix} x^* \\ y^* \\ w^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

那么新增的一维，可以验证出：

- 向量＋向量=向量
- 点-点 = 向量
- 点＋向量=点
- 点+点=中点

仿射变换=线性变换+平移变换
可用以下公式表示二维的仿射变换：

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

三维旋转

对于某个点，分别将其沿着x轴、y轴、z轴旋转的话，其对应的变换矩阵如下：

- 沿x轴旋转时，x的坐标不变

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

- 沿y轴旋转时，y的坐标不变（但要注意，x叉乘z的方向是y的反方向，所以这里符号和其他不一样）

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

- 沿z轴旋转时，z的坐标不变

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

通用一点，如果沿着某个过原点的向量n，旋转α角有如下公式

$$R(n,\alpha) = Icos\alpha + (1 - cos(\alpha))\vec{n}\vec{n}^T + sin(\alpha) \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}$$

对于旋转而言，两个旋转矩阵相加并不是两个旋转角度相加做旋转的结果，四元数就是解决这个问题的

视图变换（View/Camera Transformation）

得到一个画面分4步：

1. 模型变换（模型位置、角度、尺度调整）
2. 视图变换（摄像机位置调整到原点，朝向-Z方向）
3. 投影变换（透视投影+正交投影，将物体放缩到x、y、z的-1，1）
4. 视口变换（将-1，1上的物体变换到二维的屏幕空间）

观测（Viewing）变换=视图（View）变换+投影（Projection）变换

定义一个相机：

1. Position（相机位置）
2. Look-at / gaze direction（相机朝向）
3. Up direction（画面为上的方向）

当把相机挪到原点，方向朝着-Z，然后场景中所有目标都随着移动和旋转，相机看到的东西不变，但是方便了一系列操作。

现在有一个position为$\vec{e}$，朝向$\vec{g}$，画面上方向为$\vec{t}$的相机，如果要将其移动到原点，方向朝-Z，画面上方向朝Y。则需要如下：

1. 将position $\vec{e}$移到原点，则平移矩阵为：

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. 将朝向$\vec{g}$旋转到-Z
3. 将画面上方向$\vec{t}$旋转到Y，自然$\vec{g} \times \vec{t}$就是X的方向

如果将任意轴旋转到x、y、z轴上，求其旋转矩阵是比较困难的，但如果反过来，x、y、z轴的向量旋转到指定方向，则比较简单

$$R_{view}^{-1} = \begin{bmatrix} x_{gt} & x_t & x_{-g} & 0 \\ y_{gt} & y_t & y_{-g} & 0 \\ z_{gt} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

而旋转矩阵是正交矩阵，旋转矩阵的逆就是它的转置，故

$$R_{view} = \begin{bmatrix} x_{gt} & y_{gt} & z_{gt} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & 0 & 0 \\ 0 & 0 & z_{-g} & 1 \end{bmatrix}$$

投影变换（Projection transformation）

- 正交投影（Orthographic projection），不会近大远小。
  - 假设相机离的无限远，那么近处平面和远处的平面就会越来越接近，此时物体的前后，不会影响它显示的大小。
- 透视投影（Perspective projection），会近大远小
  - 摄像机是一个点，以摄像机为顶点，向外连出一个空间中的四棱锥，希望将四棱锥中某个区域到另一个区域之间（frustum）的所有东西显示出来

经过投影变换后的所有物体都会在处于x、y、z都是[-1, 1]的立方体内

正交投影

正交投影变换
假设有一立方体，其左右在x轴上的值分别为l,r，其上下在y轴上的值分别为
t,b，其浅深在z轴上的值分别为n,f。因为摄像机是朝向-Z方向，所以n>f。

1. 先将立方体的中心移动到原点
2. 然后将立方体的三个轴放缩到[-1, 1]

那么此时的变换矩阵为

$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

做完正交变换之后的物体，在绝对比例上有变换，当然后续还会做视口变换。

透视投影

用的最广泛的投影
透视投影变换要怎么做：

1. 先将从近平面延申出来的棱台，“挤压”回一个长方体
2. 做正交投影变换

挤压这一个过程怎么做：
假设从原点到近平面距离为n，到远平面距离为f，那么这两个平面之间的点(x,y,z)，经过"挤压"后，就变成$(\frac{n}{z}x, \frac{n}{z}y, unknown)$
对于近平面上的点，变换后xyz都不变，对于远平面上的点，z坐标轴不变；而近平面和远平面之间的点，z轴坐标无法确定，这些点“挤压”前后是这样的

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} ==> \begin{bmatrix} \frac{n}{z}x \\ \frac{n}{z}y \\ unknown \\ 1 \end{bmatrix}$$

乘上z后可以把分母去掉

$$\begin{bmatrix} nx \\ ny \\ unknown \\ z \end{bmatrix}$$

那么我们可以得到这个挤压的变换矩阵应该是：

$$\begin{bmatrix} nx \\ ny \\ unknown \\ z \end{bmatrix} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

然后结合一些特殊点

1. 近平面上的所有点，“挤压”前后不变化

$$\begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} ==> \begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} == \begin{bmatrix} nx \\ ny \\ n^2 \\ n \end{bmatrix}$$

   可以看见变换前后点的第三行，变换后的结果只有n，没有x、y，那么可以推出变换矩阵第三行应该为：
   $\begin{bmatrix} 0 & 0 & A & B \end{bmatrix}$
   故有$An + B = n^2$
2. 远平面上的中点，“挤压”前后也是所有坐标不变化

$$\begin{bmatrix} 0 \\ 0 \\ f \\ 1 \end{bmatrix} ==> \begin{bmatrix} 0 \\ 0 \\ f \\ 1 \end{bmatrix} == \begin{bmatrix} 0 \\ 0 \\ f^2 \\ f \end{bmatrix}$$

   故有$Af + B = f^2$
   联立求解，最终有：
   A = n + f
   b = -nf

所以挤压过程的变换是这样的：

$$\begin{bmatrix} nx \\ ny \\ nz + fz - nf \\ z \end{bmatrix} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

定义一个视锥需要

1. 垂直可视角度
2. 宽高比

光栅化

什么是一个屏幕：

1. 像素的数组
2. 分辨率是这个数组的维度
3. 光栅成像设备

光栅化（rasterization）：把三维空间的几何形体显示在屏幕上

假设我们定义一个屏幕的左下角位于(0,0)坐标，往上是y轴，往右是x轴，假设屏幕宽度为width，高度为height。
如何将投影后x、y、z处于[-1, 1]之间的物体变换到屏幕上？（视口变换）

1. 忽略z轴
2. 只要将物体先放缩然后移动到屏幕中间

那么它的变换矩阵是：

$$M_{viewport} = \begin{bmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

当视口变换完成后，我们需要确认每个三角面内的着色像素
为什么用三角形：

1. 最基础的多边形，任何形状都可以用三角形描述
2. 三点可确定一个平面

如何确认处于某个三角面内的像素：

1. 找出这个三角形的最大最小x、y，那么就可以找到一个正方形（Bounding Box）
2. 只需要遍历这个Box内的所有像素，判断每个像素的中心是否在这个三角形内（至于如何判断一个点是否在三角形内，就可以用每条边和对应角和点的连线去做叉乘，即可，只要三次叉乘的结果都是同正或同负，那么就在三角形内）

为什么会锯齿（比如：实际画的是一个三角形，但是这个三角形只能由一个个正方形的小像素组成）：

1. 像素本身有一定大小
2. 采样频率又不够高

抗锯齿（反走样，Antializasing）

走样的原本定义：
使用同样的采样方法，采样两种不同的频率，而无法区分采样结果属于什么频率。
方法：

- 先做低通滤波（平滑操作），再用像素中心采样，采样到什么颜色就什么颜色
- 为什么可行：
  - 对于图像而言：采样相当于在时（空间）域上取离散的点，相当于在频域上将其频域信息（频谱）复制粘贴很多份。而当走样时，在频域上会显示出这很多份信息（频谱）的重叠。而先做低通滤波，会率掉高频（频谱范围减少），那么当走样发生时，频域上的重叠就会有所减少。
- 具体怎么操作：
  - 模糊操作（平滑操作），每个像素有多少比例被三角形覆盖，那么这个像素的值就设为多少
    - 这一步可用到MSAA（在一个像素里，设置更多的采样点，比如4*4，然后看最终被三角形在这个像素内覆盖率多少个点，如x个点，那么就有x/16被三角形覆盖）
- 抗锯齿方案：
  - MSAA
  - FXAA(Fast Approximate AA)，纯后期图像处理，先得到有锯齿的图像，使用图像匹配的方法找到锯齿边界然后换掉。
  - TAA（Temporal AA），在时间上复用上帧结果的信息（静止物体可行）

深度缓存

有多个三角形要绘制出来的时候，应该如何确认三角形的绘制顺序，一个简单的思路就是采用（油）画家算法，先画远处三角形再画近处，但是一个问题就是，可能会存在空间中的三角形互相交叠，无法通过排序来找出正确的绘制顺序。

深度缓存可以解决绘制顺序的问题，深度缓存的做法是维持一个深度缓存（depth buffer）矩阵，记录着当前所有像素的最浅的深度（算法一开始会设成无限大），还会维持一个当前着色的帧缓存（frame buffer）矩阵。然后遍历每个要绘制的空间三角形，遍历每个三角形内部的像素，如果该像素的深度值要更浅，那么更新depth buffer对应像素的最浅深度，并且，更新frame buffer对应像素的值。

着色（Shading）

在图形学中，对于不同物体应用不同材质的过程叫着色

Blinn-Phong Reflectance Model

- 高光（Specular Term）
- 漫反射（Diffuse Term）
- 环境光（Ambient Term）

漫反射：光线从一个方向射进来后，从四面八方反射出去，从任意角度看都一样

- 点光源，可以理解为圆形的能量不断向外扩张，那么假设点光源能量强度为I，距离点光源距离为r的点，其到达的能量为I/r^2
- 漫反射出来的光的大小，和着色点位置入射光i法线n夹角θ的余弦成正比

那么漫反射的光应该为

$$L_d = k_d(I/r^2) \times max(0, cos(\theta))$$
$$cos(\theta) = \vec{n} \cdot \vec{i}$$

$\vec{n}$，$\vec{i}$为单位向量。k_d是一个系数，决定接收到的能量以多少比例反射出来，r、g、b可以各有一个系数来控制这个点的颜色。

高光：光线从一个方向射进来后，几乎以镜面反射的方向反射出去

通过判断法线和（入射角和出射角）角平分线（半程向量）是否足够接近，即可判断是否高光

$$L_s = k_s(I/r^2) \times max(0, cos(\alpha))^p$$
$$cos(\alpha) = \vec{n} \cdot \vec{h}$$

$\vec{n}$，$\vec{h}$分别是法线的单位向量和半程向量的单位向量，$cos(\alpha)$可以判断两个向量是否接近，但是cos这个函数会衰减的比较慢，加上指数p可以更好的，倍增这个值，当不处于高光角度时，锐减这个值。

环境光：

大胆假设环境光是常数：

$$L_a = k_a I_a$$

但是实际要精确计算需要用到环境光照的知识

着色频率（Shading Frequencies）

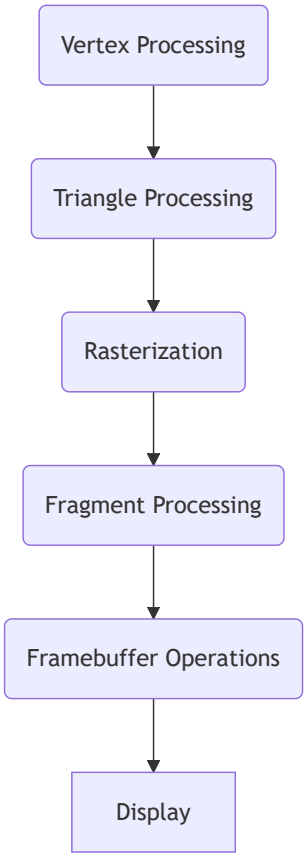
- 每个三角面着色（flat shading）
- 每个顶点着色（gouraud shading）
- 每个像素着色（phong shading）

不一定逐三角面着色效果就很差，如果模型本来就有足够多的面，那么用相对简单的着色模型flat shading，也有很好的效果。

如何求顶点的法线：

1. 找到所有和该顶点相关联的面
2. 求出这些面的法线，然后求平均，即为顶点法线

图形（实时渲染）管线（Graphics Real-time Rendering）Pipeline



输入一堆空间上的点，然后依次经过如下处理：

1. 顶点处理，将空间上的点从三维空间投影到屏幕空间
2. 三角形处理，将屏幕空间上的点，组装成三角形
3. 光栅化，将三角形离散化，用一个个离散的像素去覆盖三角形
4. 着色，使用深度缓存技术取得每个像素的最小的深度和颜色。
5. 帧缓冲区处理

shader：实时渲染管线中的可编程部分，控制顶点或像素的着色，在硬件上编程的语言。
如果写的是顶点着色，那么就是顶点着色器（vertex shader）
如果写的是像素着色，那么就是像素着色器（fragment/pixel color）

纹理映射（Texture Mapping）

纹理就是一张图，可以用于定义任何一个点的不同属性（包括反射系数等）
将纹理拉伸、压缩、撕开或者其他变换后蒙在在一个三维物体的表明就是纹理映射。
主要是通过建立纹理上每个三角形和模型每个三角面的映射实现。

纹理的横纵坐标分别是uv

不同的材质就是不同的着色方法

- 低分辨率的纹理直接贴到高分辨的画面中会存在问题，比如会很明显看到一个个格子，相当于最近邻插值。可改用双线性插值或bicubic解决。
- 高分辨率的纹理直接贴到低分辨率的画面中也会存在问题，会导致锯齿或摩尔纹的出现，采用类似MSAA的超采样方法可以解决这种问题，就是成本过高，而如果有方法可以立刻求出某一个块区域的纹理的平均值（范围查询）作为画面该像素的纹理，就可以高效解决这个问题。
    - 而Mipmap的方法就可以实现范围查询，只需额外1/3空间，迅速，近似，但只允许正方形范围的查询。原理大概是对纹理建立图像金字塔（如原先是128*128，那么通过平均池化，得到64*64，32*32，...，再到1*1），将像素坐标转为纹理uv坐标，得到覆盖的纹理区域，通过查找对应相邻两层的金字塔来插值（三线性插值）即可迅速得到区域近似平均值。

        - 但是Mipmap会导致过分模糊（OverBlur）的问题，一是因为插值本来就是近似，二是只允许正方形范围查询，而当一个像素映射到纹理中的区域，往往不是正方形。
    - 各向异性过滤（Anisotropic Filetering），需额外3倍空间，除了对正方形，还能对长方形进行范围查询。但是对于斜着跨了几个uv个点的区域还是无能为力
    - EWA filtering，用椭圆来做

三维的纹理

三维物体空间中任何一点都有值，不止在表面。实际上并没有一个纹理的图，而是定义了在三维空间中的一个噪声函数。可以用来描述空间中任何一点的值，通过将噪声的值经过各种处理，就可以表示当前的颜色。

重心坐标（Barycentric coordinates）

重心坐标可以解决三角形在三个顶点上的值插值到三角形内部像素的问题。

三角形ABC所在平面内的一个点（x,y）的坐标，都可以通过三个顶点的线性组合得到

$$(x,y) = \alpha A + \beta B + \gamma C$$
$$\alpha + \beta + \gamma = 1$$

如果要求点在三角形内部，那还必然满足α, β, γ非负
如果将这个点分别连上A、B、C，就可以在三角形内部画出三个三角形，如果把顶点A、B、C对面的三角形面积定义为S_A, S_B, S_C。那么有：

$$\alpha = \frac{S_A}{S_A + S_B + S_C}$$
$$\beta = \frac{S_B}{S_A + S_B + S_C}$$
$$\gamma = \frac{S_C}{S_A + S_B + S_C}$$

如果三个小三角形的面积相等，那么这个点就是三角形重心。

应该在三维空间中使用重心坐标插值了，再投影到二维空间，因为3D->2D的空间中，重心坐标会发生变化。

纹理的应用

环境光

纹理可以用来表述环境光，环境光默认只受方向影响，忽略位置。
环境光的信息可以被一个光滑的球面（spherical map）所反映。如果将球面的纹理铺开，则会存在拉伸变形。如果将球面的信息，放到立方体（cube map）上，则可以在展开时避免变形。

凹凸贴图（Bump Mapping）

纹理还可以影响着色，可以使用将相对高度信息存在纹理上（法线贴图），改变法线方向，从而影响着色结果，使得简单的模型，可以拥有更复杂的凹凸纹理。
如何求法线呢?

1. 先求切线，切线就是函数的求导，求梯度，可用相邻值作差再除相邻距离。求u，v两个方向的偏导，得到两个切线
    - 假设原本法线是n(p) = (0,0,1)
    - 得到p点的u，v偏导为
        - dp/du = c1 * (h(u+1) - h(u))
        - dp/dv = c2 * (h(v+1) - h(v))，c1，c2是一个作为系数调整
    - 那么新法线是 n = (-dp/du, -dp/dv, 1)
2. 将切线旋转九十度得到法线
    - normal = (-dp/du, -dp/dv, 1).nomoralized()

3. 上面是假设法线永远向上，但其实实际不是，不过，我们可以定义一个局部相对坐标，让其法线就是向上，最后进行坐标转换即可。

位移贴图（Displacement mapping）

位移贴图也是将相对高度信息存在纹理上，但是真的给这些顶点做一个位置上的移动（存在几何上的变化），而不是直接修改法线。但是他要求几何模型的三角面要足够多，因为他无法修改三角形内部。

环境光遮蔽（Ambient occlusion）

在纹理上算好环境光遮蔽（阴影），0~1，可以用于最终乘上去。

几何（Geometry）

- 隐式的几何（Implicit Geometry），不会给出点的具体位置，只会给这些点满足的关系。
  - 种类：
    - 公式表示，如（球面的隐式表示，$x^2+y^2+z^2=1$，更通用的可以表示成：$f(x,y,z)=0$）
    - CSG（constructive solid geometry），通过简单几何体的运算（取交集，并集）表示
    - 距离函数（distance functions），不直接描述几何的表面，而是描述任何一个点到这个表面的最近距离（正的话，点在外面，否则在里面），在做blend的时候好用，距离为0即为表面。水平集（level sets）可以用来表示距离函数
    - 分型（Fractals），如果几何的部分和整体是类似的（递归）。
  - 易于判断某个点在物体内，还是在物体外，还是在面上
  - 但难以找出在所有在满足该条件的点
- 显式的几何（Explicit Geometry）
  - 种类：
    - 点云（Point clouds），用足够多的点（x，y，z的数组）去表示表面，当需要表示复杂的模型时，就需要特别多的点。
    - 多边形网格（Polygon mesh），比如用多个三角形去拟合球面，然后给出这些三角形的位置即可
    - 参数映射，比如给出（u，v）可以有函数映射到（x，y，z）
  - 易于找到在这个面上所有的点，全遍历一次即可
  - 但难以判断某个点在不在面上，在面内还是外

曲线

贝塞尔曲线（Bezier Curves）

通过de Casteljau算法，可以根据给定一些控制点可以画出一条贝塞尔曲线。比如给出A、B、C三个点，那么起点就是A，终点就是C，曲线必过这两点，假设用t表示某个时刻（进度），t在0到1之间，那么只要表示好t时刻的所在的点，就能表示好整个曲线。而t时刻的位置，先在AB之间取点d1 = AB * t，再在BC之间取点d2 = BC * t，再在d1d2之间取d3 = d1d2 * t，即为t时刻的位置。

假设有三个点b_0, b_1, b_2，那么有中间点的公式：

$$b_{0,1}(t)=(1-t)b_0+tb_1$$
$$b_{1,1}(t)=(1-t)b_1+tb_2$$

$b_{i,j}$为第i次计算中的第j个点

$$b_{0,2}(t)=(1-t)b_{0,1}+tb_{1,1}$$

最终有

$$b_{0,2}(t)=(1-t)^2b_0+2t(1-t)b_1+t^2b_2$$

每一项前的系数是伯恩斯坦多项式：

$$B_i^n=\left(\begin{array}{c}n\\i\end{array}\right)t^i(1-t)^{n-i}$$

贝塞尔曲线的特点：

1. 过起点和终点
2. 对于四个控制点的情况（3次），起点的切线是三倍的b1-b0，终点的切线是3倍的b3-b2
3. 经过仿射变换前后的贝塞尔曲线不变
4. 凸包性质，曲线不会超过控制点形成的凸包

用超多控制点来控制贝塞尔曲线并不合适，可以每次四个控制点，然后首尾相接，得到分段的贝塞尔曲线（piecewise bezier curve）

样条曲线（Splines Curve）

曲线通过几个给定点，由这几个给定的点控制的曲线
b样条曲线，拥有局部性

曲面

在4*4的坐标上，先水平四行上的每4个点各自做贝塞尔曲线，如果水平上的时间是u，那么可以得到u时刻的四个点，再把四个点在竖直方向上做贝塞尔曲线，也可以得到v时刻的一个点，那么就相当于建立了（u，v）到（x，y，z）的映射，所以是一种显式的几何表示

三角形面的操作

- 网格的细分（Mesh Subdivision）
- 网格的简化（Mesh Simplification）
- 网格的正则化（Mesh Regularization）

网格的细分

细分：

1. 引入更多三角形
2. 让这些三角形的位置发生变化，使得模型更加光滑

Loop Subdivision（仅用于三角形网格，跟循环没有关系，人名是loop）

1. 先将大三角形按上，中，左下，右下划分为4个三角形，原三角形边的中间产生了新的顶点
2. 某个新顶点肯定被相邻的两原三角形共享，新顶点的位置，应该等于3/8左右两原顶点的和，加上1/8上下对着的两原顶点的和。以此来更新新顶点的位置
3. 而老顶点的位置，则是相邻老顶点和自己的加权平均

Catmull-Clark Subdivision（通用，有三角形四边形都可以）

1. 所有边上都取一个中点，在所有面内也取一个中点，每一个多边形内，都将这些点连起来
2. 顶点更新

特点：

1. 经过一次处理后，原网格所有非四边形面都会变成四边形面，原网格每个非四边形都会产生一个奇异点（度不为四）

曲面简化

边坍缩（edge collapsing）

至于要坍缩哪些边，可以用二次误差度量（quadric error metrics）决定
总的算法大致是：

1. 使用优先队列去装，每条边坍缩后可能产生的误差
2. 每次取误差最小的边坍缩，然后更新受影响的边的误差

阴影

光栅化或光线追踪可以生成阴影

着色解决的是一个局部的行为，只考虑着色点，考虑光源和摄像机，完全不考虑其他物体或者同一个物体的其他部分对该点的影响，解决不了阴影问题。

阴影映射（Shadow mapping）

本质是一种图像空间的算法

- 生成阴影这一步并不需要场景的几何信息
- 会产生走样

一个点如果不在阴影里，那么这个点必要可以在光源角度看到，和能被摄像机看到

点光源下如何生成阴影：

1. 先假设在光源角度有摄像机朝向场景，做光栅化，记录每个像素的深度（shadow map，这个map分辨率太低可能会导致走样）
2. 再用真正摄像机的位置去看场景，做光栅化，遍历每个像素，如果这个像素对应的在3D场景中的位置投影回光源摄像机屏幕中，在光源摄像机中的深度并不是先前记录的深度，则这点处于阴影位置

可能会有不少误差，误差来源：

1. 浮点数判断本身有精度问题
2. 一个像素映射回3D场景中，会存在一个像素代表多个点的问题
   而且只能实现硬阴影（只能单一点光源）

硬阴影和软阴影

- 硬阴影：要么在阴影里，要么不在（非0即1）
- 软阴影：阴影是过渡的，从有阴影过渡到没阴影
  - 性质：越靠近物体根部，阴影越硬
  - 当光源存在一定体积大小（或多个光源）时，就会导致存在全影（光源被完全挡住）、半影（光源被部分挡住），这就导致了全影部分更硬，半影部分更软

光线追踪（Ray Tracing）

光栅化无法很好地解决以下问题（全局效果）：

- 软阴影
- glossy reflection（类似镜面反射，但又不完全，比镜面反射粗糙）
- 间接光照（Indirect illumination）

光栅化更快，适用于实时场景
光线追踪比较准确，但相对较慢，更多用于离线

在光线追踪技术中，作如下假设：

1. 光沿直线传播
2. 光线之间不会碰撞
3. 光线可逆，即光源发出光射沿着某一路线向眼睛，同样眼睛沿着该路线感知光源

递归（Whitted-Style）光线追踪

1. 从眼睛发出射线（到第一个碰撞点之间叫primary ray），穿过成像平面的每个像素
2. 这个射线如果和场景中的某个光滑物体碰撞，那么可能会发生反射和折射（反射和折射的都叫secondary rays），如果是发生漫反射，就不会继续反射折射。
3. 能反射和折射出去的射线又会继续反射和折射下去，每次反射和折射都会有能量衰减
4. 将射线的所有碰撞点和光源相连（这个线叫做shadow rays），所有碰撞点都会计算着色，最终贡献到同一个像素

求交点

射线的数学定义：$r(t) = \vec{o} + t\vec{d}, 0 < t < \infty$，o是起点，d是方向向量。

- 如果曲面使用隐式表示，隐函数表示，比如f(p) = 0，那么只要求解f(o + td) = 0即可
- 如果是显式表示，需要判断三角形和射线的交点：
  i. 先判断射线和三角形所在平面的交点
  ii. 再判断交点是否再三角形内
- 交点也可以直接用Moller Trumbore（MT）算法解，这个算法有一条等式，等号左边表示射线，右边是三角形的重心坐标表示三角形内某一点，即$\vec{o}+t\vec{d}=(1-b_1-b_2)\vec{p_1}+b_1\vec{p_1}+b_2\vec{p_2}$，解该方程便求解，只要解的结果，t是正的，三角形每个点前的系数也是非负，即在三角形内。

平面数学定义：$(p-p^*)\cdot\vec{N}=0$，p^*是平面上的给定一点，$\vec{N}$是垂直于平面的向量（法线），p是平面上的任意一点。该式最终也可写为：$ax+by+cz+d=0$
既然平面可以写成函数，将射线表示带入，便可求出交点

空间中有一个点，和一个封闭的物体，从点发出一条射线，如果和物体有偶数个交点，必定在物体外部，如果有奇数个交点，必在物体内部

光线和三角面快速求交

一个模型有很多个三角面，如果判断模型和光线的交点，需要判断很多个三角形和光线的交点，再找出最近的交点的话，速度会十分慢

包围盒（bounding volumes）

给模型建立一个简单的包围盒，先判断光线过不过包围盒，如果不过可以直接免去很多判断

最常用的是轴对齐包围盒（Axis-Aligned Bounding Box，AABB），可以理解成，用三对平面框住的盒子，这三对平面都是分别垂直于x，y，z轴，这样可以减轻运算的负担（比如，一根射线穿过垂直与x轴的平面(如x=1)，只需看这个射线在x轴上的分量，什么时候过x=1即可）。

- 那么给定一条射线，对于每一对平面，都可以求出t_{min}，t_{max}，离射线起点近的是t_{min}，远的是t_{max}。然后，进入盒体时间即为$t_{enter}=max(t_{min,1}，t_{min,2}，t_{min,3})$，离开盒体时间为$t_{exit}=min(t_{max,1}，t_{max,2}，t_{max,3})$
- 那么通过判断，$t_{enter}<t_{exit}$可以得到该射线穿没穿盒子，但这种情况其实考虑不全，因为，射线的反方向穿过盒子，上述判断依然成立，故应该增加条件$t_{exit}>=0$

当判断到光线与包围盒相交时，如何更进一步找到包围盒内部，与物体的实际交点：

- 均匀格子（Uniform grids），在包围盒里，再分成一个个均匀的格子，标识这一个个均匀的格子是否有物体。那么光线进入物体后，先根据所在的格子，是否有物体，若没有物体继续找下一个格子，直到找到有物体的格子并与物体发生相交，或者直接光线射出去。
  - 均匀格子，适用于长得比较均匀的物体，若物体形状不均匀，则可能有大量的空白格子，要判断，但这些空白格子内又没有物体。
- 空间划分（Spatial partitions）
  - 八叉树（Oct-Tree）,分别按垂直于x、y、z轴的的平面切分包围盒，就可以得到8块，每一小块又可以以类似的方法切成8块，然后可以设置一些规则停止切分，比如如果切完之后有超过7块是没有物体的那么就不切。
    - 八叉树缺点：跟维度相关，二维是四叉，三维八叉，随维度指数增加
  - KD-Tree，轮流按垂直于x，y，z轴的平面划分，划分时尽量贴着某个内部形状的面
  - BSP-Tree，也是每次用一个平面划分，但是平面不必横平竖直（垂直于x，y，z轴）
- 物体划分（Object Partitions）

KD-Tree的节点要存储的：

1. 当前是沿哪个轴划分
2. 划分的实际位置
3. 非叶子节点应有两个子节点存储划分的两半
4. 实际的 物体（数据）存在叶子节点处

KD-Tree判断流程：

1. 树应该在做光追前，建立好
2. 如果光线和包围盒无交点，则跳过，如果有交点，则判断根节点的两个子节点的格子是否有交点，如果和格子有交点，则递归去找子节点的两个子子节点，以此类推
3. 最终找到叶子节点，并和叶子节点中的物体判断求交

但是KD-Tree有两个问题：

1. 物体可能会存在于多个叶子节点里（物体同时于多个格子相交）
2. 三角形与盒子的求交有困难

按物体划分可以解决该问题。
物体划分（Object Partitions）
物体划分的结构称为：Bounding Volume Hierarchy（BVH）
就是每次使用两个包围盒来框住物体，尽可能将物体分为两组，这两个盒可能重合，但是一个物体只能属于一个框。停止划分可以设定一个条件，比如包围盒内的物体已经少于n个。

BVH里怎么划分物体呢：

- 按当前包围盒中，边更长的来切，按物体重心坐标属于哪边来分
- 按重心坐标沿某一轴来排序（当然实际不用排序，可以使用快速选择算法来找中间值），找出中间物体来划分

KD-Tree：先分成两半，再考虑子节点和物体的相交情况
BVH：把物体分为两堆，再求出各自包围盒
两种方法，只是在划分包围盒阶段（建树）有区别，在最终光线判断相交物体时并无区别。

辐射度量学（Basic radiometry）

需要精确地衡量光的能量单位，光的属性

Radiant Intensity：每单位立体角的辐射功率
Solid Angles（立体角）：A/r^2，A是立体角形成圆锥，在球上截取的面积，值的范围$[0, 4\pi]$

辐射能量Q，单位焦耳(J)
辐射通量（功率）$\Phi = \frac{dQ}{dt}$，单位瓦特（W），或者流明（lm，lumen），每单位时间的能量
辐射强度$I(\omega) = \frac{d\Phi}{d\omega}$，每单位立体角的功率

如果给出立体角a，那么就可以求出，在该方向上的强度 $I = a/4\pi$

微分立体角：
在球面坐标系下，用一个θ（向上方向偏离的角度），和一个ϕ（绕着向上的轴旋转的角度）就可以描述一个方向。那么偏离一点点的$d\theta$和$d\phi$，形成的立体角就叫做微分立体角。
微分立体角在球上的面积：

$$dA = (rd\theta)(rsin\theta d\phi) = r^2 sin\theta d\theta d\phi$$

微分立体角：

$$d\omega = \frac{dA}{r^2} = sin\theta d\theta d\phi$$

Irradiance：每单位面积入射的功率，单位W/m^2，$lm/m^2 = lux$，入射的功率只算直射部分（算垂直于表面的分量）

$$E(x) = \frac{d\Phi(x)}{dA}$$

那么在距离点光源（功率为Φ）为r的点上，其Irradiance为：

$$E = \frac{\Phi}{4\pi r^2}$$

Radiance：单位面积向单位立体角投射（接收）的功率，单位$W/srm^2, cd/m^2 = lm/srm^2 = nit$

$$L(p, \omega) = \frac{d^2\Phi(p, \omega)}{d\omega dAcos\theta}$$

θ为表面法线和立体角（ω）的夹角

所以Irradiance相当于Radiance按立体角积一个半球的分

双向反射分布函数（Bidirectional Reflectance Distribution Function，BRDF）
描述一个光以某一角度射到平面上，触发反射，那么在某一个角度接收的功率应该是多少

假设光以立体角ω_i的角度射入单位面积的平面，radiance为$L_i(\omega_i)$，那么在ω_r的角度收到的功率比例为

$$f_r(\omega_i->\omega_r)=\frac{dL_r(\omega_r)}{dE_i(\omega_i)}=\frac{dL_r(\omega_r)}{L_i(\omega_i)cos\theta_i d\omega_i}\frac{1}{sr}$$

那么最终着色点的显示，应该就是将所有入射到着色点的光，又反射到特定出射方向（观测方向）的总和，即反射方程：

$$L_r(p,\omega_r)=\int_{H^2}f_r(p,\omega_i->\omega_r)L_i(p,\omega_i)cos\theta_i d\omega_i$$

渲染方程就是反射方程加上自发光的内容：

$$L_o(p,\omega_o)=L_e(p,\omega_o)+L_r(p,\omega_r)$$

如果把场景中所有的光定义为L，直射的光定义为E，反射的操作是K，那么有

$$L=E+KL$$

最终经过变换和泰勒展开有：

$$L=E+KE+K^2E+K^3E+...$$

直接光打到眼睛+直接光照+间接光照+更多的间接光照+...

光栅化只能做到渲染光直接打到眼睛和直接光照，间接光照实现困难
全局光照是直接光照+间接光照的集合

蒙特卡洛积分（Monte Carlo Integration）

对定积分的数值求解：

- 黎曼积分：求函数f(x)在(a,b)之间的定积分，那么可以将区间分成有限的n份，在每份中取函数在中间的值和(b-a)/n相乘相加。
- 蒙特卡洛积分：求函数f(x)在(a,b)之间的定积分，可以在(a,b)之间先不断随机取x_i，求得多个$f(x_i)$的平均值，最终乘上(b-a)。

在概率论里，一个概率密度函数为p(x)，那么求期望应该是

$$E[X]=\int xp(x)dx$$

如果有Y = f(x)，那么Y的期望为

$$E[Y]=\int f(x)p(x)dx$$

假设有定积分：

$$\int_a^b f(x)dx$$

有X_i在a和b之间，对X_i的采样遵循p(x)的概率密度函数
那么蒙特卡洛积分可以估算该定积分为：

$$F_N=\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$$

如果X_i均匀采样，那么p(x) = 1/(b-a)，那么有：

$$F_N = \frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$$

路径追踪（Path Tracing）

Whitted-Style Ray Tracing在漫反射之后就停住了，不会真正的把光线反射到四面八方，路径追踪就是解决这个问题。

回到渲染方程：

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_{\Omega^+} L_i(p,\omega_i) f_r(p,\omega_i,\omega_o)(n\cdot\omega_i)d\omega_i$$

法线$\vec{n}$和$\vec{\omega_i}$点乘可得到$cos\theta$
然后，利用蒙特卡洛积分方法，渲染方程可写成：

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \frac{1}{N}\sum_{i=1}^{N} \frac{L_i(p,\omega_i) f_r(p,\omega_i,\omega_o)(n\cdot\omega_i)}{p(\omega_i)}$$

均匀采样时，$p(\omega_i) = 1/2\pi$

算法思路：

1. 遍历每个像素，在每个像素内随机生成N个点，在观测点与这N个点之间生成射线。
2. 遍历射线与场景相交的每个点p，计算这些点的着色，并取平均，作为该像素的颜色。每个点的着色过程如下
    i. 着色是递归进行的，要计算p点的着色，先计算随机采样一根入射光线，来计算着色。如果入射的光线是光源发出的，那么不用继续递归，如果入射的光线是另一个着色点反射的，那么递归计算该着色点的着色。
        a. 这里只随机采样一个入射光线无疑会有很大误差，但如果随机采样t根光线，再递归到下一个着色点时又会采样t根，就会指数爆炸，所以不如再同一个像素发射N个射线，最终平均N个射线的结果。
        b. 还有另一个问题是，为了避免无限递归下去（光线反射无限次），采用俄罗斯轮盘赌来限制多次反射，即设置r的概率，决定是否采样一根入射光线的结果L_i，那么返回结果就是$r * L_i + (1-r) * 0.0$，这样的期望与光线反射无数次是相等的，但可以避免无限反射的计算。
        c. 最后一个可以优化的点是，N有可能，在取很大的值的时候，才有可能得到更优的结果，特别是光源比较小时（着色时随机采样1根入射光线难以探测到光源），那么应该对光源做特别处理，我们入射光线的随机采样不应该是均匀采样，应该是对光源的采样，光源的面积A是可以转换为入射的立体角的，那么对立体角的定积分，就可以变成对A的定积分，然后求出光源的贡献（直接光照）。至于其他物体的反射（间接光照），则照旧即可。
        d. 路径追踪不好求点光源的贡献，不过可以把很小面积的光源当作是一个点光源

材质（Material）

材质就是BRDF，会定义漫反射系数，会定义光是以怎么一种方式反射的

漫反射系数

根据渲染方程，忽略自发光项后有，假设入射的光线是均匀的：

$$\begin{aligned} L_o(w_o) &= \int_{H^2} f_r L_i(\omega_i) cos\theta_i d\omega_i \\ &= f_r L_i \int_{H^2} cos\theta_i d\omega_i \\ &= \pi f_r L_i \end{aligned}$$

根据能量守恒有，$L_o = L_i$，则有

$$f_r = \frac{1}{\pi}$$

此时的BRDF就是常数$\frac{1}{\pi}$不吸收能量

如果定义一个反射率ρ，那么就可以代表不同颜色的BRDF，不同颜色的漫反射系数

$$f_r = \frac{\rho}{\pi}$$

反射（Reflection）

入射向量、反射向量和法线之间的关系：

$$\vec{\omega_o} + \vec{\omega_i} = 2cos\theta\vec{n} = 2(\vec{\omega_i} \cdot \vec{n})\vec{n}$$

在方位角上（沿着法线的逆方向看）

$$\phi_o = (\phi_i + \pi) mod 2\pi$$

折射（Refraction）

根据斯内尔定律（折射定律，Snell's Law）
假设入射角是θ_i，折射角是θ_t，入射介质的折射率为η_i，出射介质是η_t

$$\eta_i sin\theta_i = \eta_t sin\theta_t$$

可推：

$$\begin{aligned} cos\theta_t &= \sqrt{1 - sin^2\theta_t} \\ &= \sqrt{1 - (\frac{\eta_i}{\eta_t})^2 sin^2\theta_i} \\ &= \sqrt{1 - (\frac{\eta_i}{\eta_t})^2 (1 - cos^2\theta_i)} \end{aligned}$$

当$\frac{\eta_i}{\eta_t} > 1$时，即当入射介质的折射率大于折射介质的折射有可能不发生折射，这个现象就是全反射现象，比如从水下看水上就可能发生，能看到来自水上的光的角度会比较小。

方位角同样是相差180°。

表示折射属性的是BTDF，BSTF = BTDF + BRDF。但其实一般也不分那么细，用BRDF更多。

菲涅尔项（Fresnel Term）

菲涅尔项可以解释有多少能量发生了反射，有多少能量发生了折射
当入射角很大时，会有更多的能量反射
当入射角很小时，会有更多的能量折射

对于绝缘体上述现象会很明显，对于导体（金属）的话，无论什么角度，都反射很多。

对于菲涅尔项，有Schlick近似，可求出光的反射率：

$$\begin{aligned} R(\theta) &= R_0 + (1 - R_0)(1 - cos\theta)^5 \\ R_0 &= (\frac{\eta_i - \eta_o}{\eta_i + \eta_o})^2 \end{aligned}$$

微表面材质（Micorfacet Material）

只要距离观测物体足够远，就可以忽略观测物体的表面的微小物体，只得到他们最终形成的整体作用，

微表面模型假设：

1. 从远处看，看到的面是平坦而且粗糙的
2. 从近处看，是凹凸不平的，可以看作是无数个小平面的镜面反射
    i. 如果这些微表面的法线的朝向相对一致，那么就是glossy的（较光滑）
    ii. 如果这些法线朝向各自差很远，那么就是更加diffuse（粗糙）
    iii. 上面两点说明，可以用法线的分布来表示一个表面粗糙与否

微表面模型会认为，BRDF由三项组成：

$$f(i,o)=\frac{F(i,h)G(i,o,h)D(h)}{4(n,i)(n,o)}，\ h是半程向量$$

- 菲涅尔项（F），表示有多少比例能量被反射
- 几何项（Shadowing-masking，G），对表面互相遮挡的修正（当光以几乎和平面平行的角度射入，会被粗糙表面挡住不少，这种入射方向叫grazing angle）
- 法线分布项（D），描述当前这个表面，有多少微表面的法线接近h（只有当法线接近半程向量，才能发生反射）

材质分类

- 各向同性材质（Isotropic），
  - 从微表面来解释，其微表面不存在方向性，或者方向性很弱
  - 从BRDF来解释，反射不依赖于方位角，只跟观测和入射光之间方位角之差相关
- 各向异性材质（Anisotropic）
  - 从微表面来解释，其微表面存在明确方向性，法线分布也具有明确方向
  - 从BRDF来解释，反射依赖于方位角

BRDF的特点

1. 非负
2. 线性（总的BRDF可以由各项线性组合，比如在Bling-phong里，有镜面反射+漫反射+环境光）
3. 可逆性（交换入射和出射的角色，得到的结果一样）
4. 能量守恒

BRDF怎么测

1. 枚举所有的出射方向，把光源放上去
2. 枚举所有的入射方向，把相机放上去，测得radiance