

第一讲 向量与线性代数

向量

基本概念

长度、方向、不随平移而改变

单位向量，用来表示方向

向量求和（几何上三角形、平行四边形法则，代数上坐标相加即可）

点乘

向量->数

$$a \cdot b = \|a\| \|b\| \cos\theta = x_a x_b + y_a y_b + z_a z_b$$

用处

1计算余弦，得到夹角

2计算投影，把原向量分解成两个垂直的向量

3分析两个向量有多接近

4分析两个向量前与后的信息（b在a的前方还是后方，方向基本一致还是相反）

叉乘

$$\|a \times b\| = \|a\| \|b\| \sin\theta$$

$$a \times b = -b \times a$$

$$a \times b = \begin{pmatrix} y_a z_b - y_b z_a \\ z_a x_b - x_a z_b \\ x_a y_b - y_a x_b \end{pmatrix}$$

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

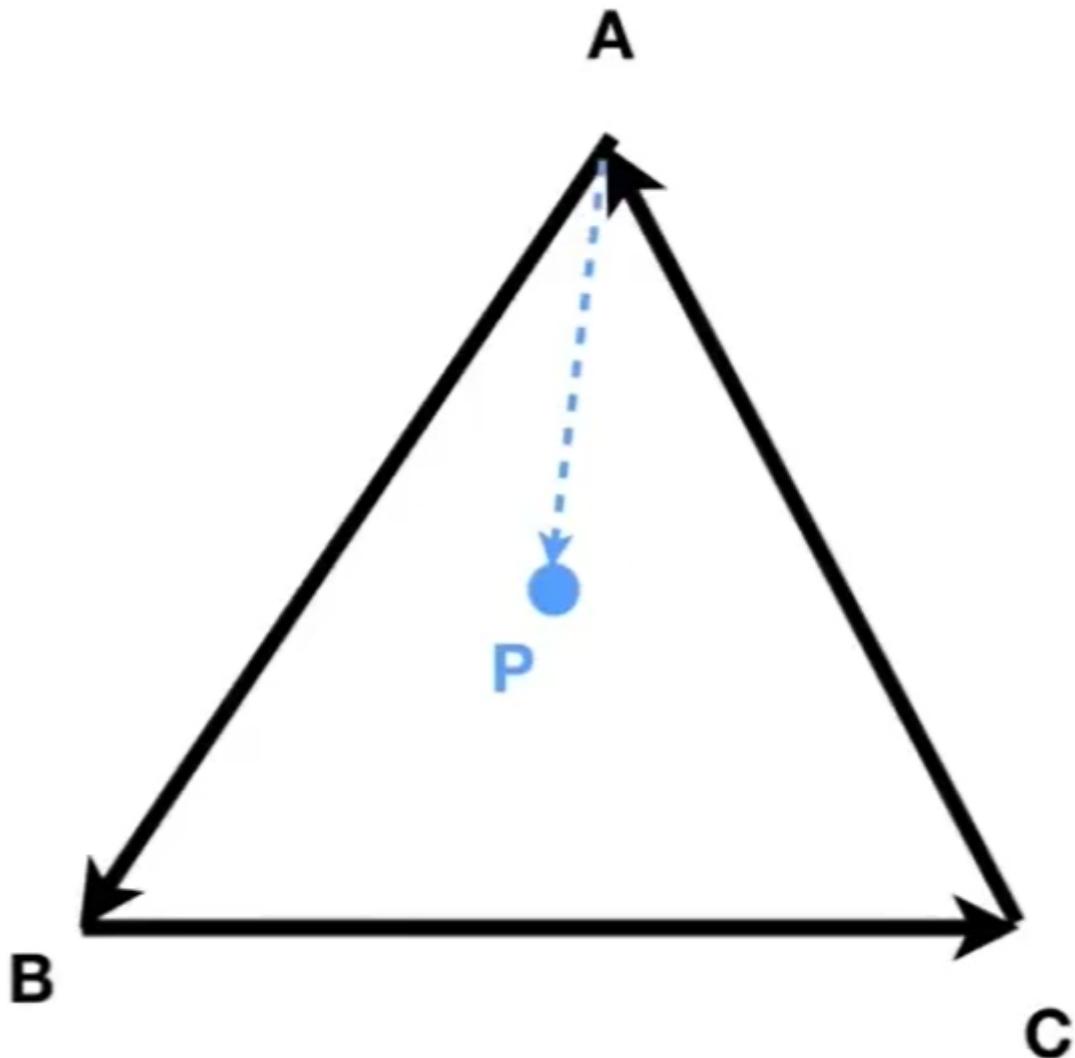
方向和a, b都垂直（右手定则）

用处

1建立三维空间的一个直角坐标系

2判断左和右，结果为正，b在a左侧；反之，b在a的右侧

3判断内和外，点p是否在三角形中



计算 $AB \times AP, BC \times BP, CA \times CP$,假如同号，说明在内侧（顺时针、逆时针无所谓）

标准正交坐标系

把向量分解到三个方向上

矩阵

乘积

$$(m \times n)(n \times p) = (m \times p)$$

性质

1无交换律

2结合律 $A(BC) = (AB)C$

3分配律

转置

1行列互换

$$2(AB)^T = B^T A^T$$

$$3a \cdot b = a^T b$$

单位矩阵

$$AA^{-1} = A^{-1}A = I$$

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

第二讲 二维变换

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

缩放

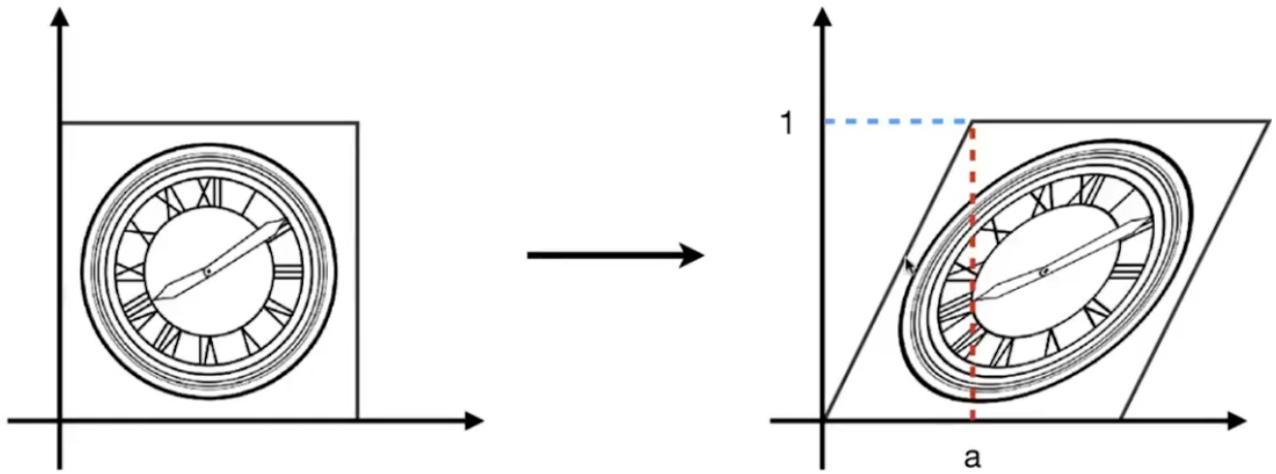
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

对称/反射

$$y\text{轴对称} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

切变

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



旋转

默认逆时针，以原点为中心

利用特殊点求解

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

特殊性质：旋转矩阵是正交矩阵， $R^{-1} = R^T$

齐次坐标

平移操作不太好写，所以引出齐次坐标

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

一般操作

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

二维点 $(x, y, 1)^T$

二维向量 $(x, y, 0)^T$

向量+向量=向量

点-点=向量

点+向量=向量

点+点=这两个点的中点

因为

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \text{ 是二维点 } \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}$$

逆变换

M^{-1}

矩阵组合

将变换矩阵相乘，用一个矩阵实现一系列操作

矩阵分解

比如，绕任意点C旋转，先把图平移到原点，再旋转一个角度，在平移回去

第三讲 变换2.0

三维变换

三维点 $(x, y, z, 1)^T$

三维向量 $(x, y, z, 0)^T$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

先线性变换，再平移

绕轴旋转

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

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

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

一般旋转

罗德里格斯旋转方程

假设绕n轴（默认过原点），旋转 α 角

旋转方程为，假如不过原点，先平移->再旋转->平移

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n} \mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

观测变换

视图/摄像机变换

相机定义

1 位置(position) \vec{e}

2 目视方向(look_at/gaze direction) \hat{g}

3 向上方向(up direction) \hat{t}

让物体和相机一起变换

默认相机永远在1原点 2 向上方向为Y 3 目视-Z

相机做了一些变换，则将他变换 M_{view} 回默认位置，这时别的物体再和相机一起做 M_{view} 变换

转移方法

$$M_{view} = R_{view} T_{view}$$

先平移e到原点

Translate e to origin

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

旋转g到-Z，旋转t到Y，X就自然对上了（但写起来很麻烦）

所以反过来考虑，先把X到($g \times t$)，再把Y到t，Z到-g，原来的变换就是这个变换的逆变换。

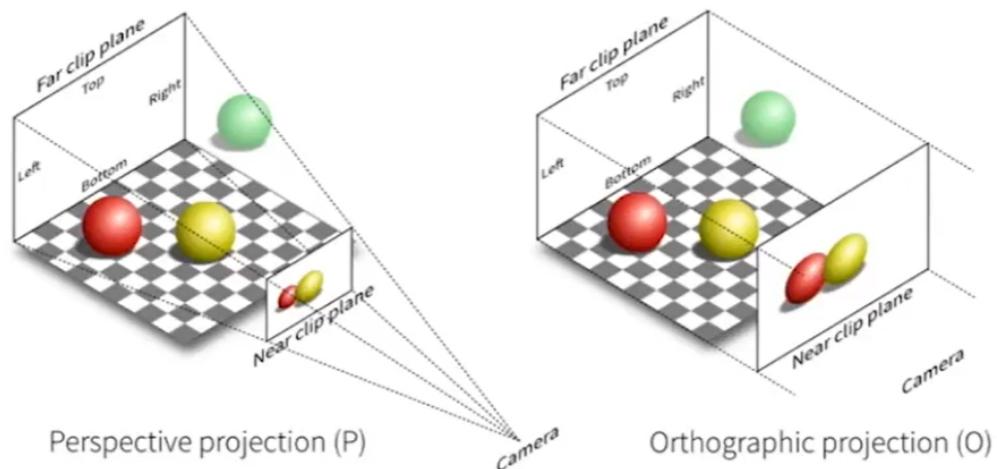
- Consider its **inverse** rotation: X to ($g \times t$)，Y to t, Z to -g

$$R_{view}^{-1} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g} \times \hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g} \times \hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{WHY?} \quad R_{view} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & y_{\hat{g} \times \hat{t}} & z_{\hat{g} \times \hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

为什么求转置就可以得到逆呢？因为旋转矩阵是正交矩阵，转置==逆

投影变换

- Perspective projection vs. orthographic projection



正交投影（没有近大远小）

认为摄像机在无限远

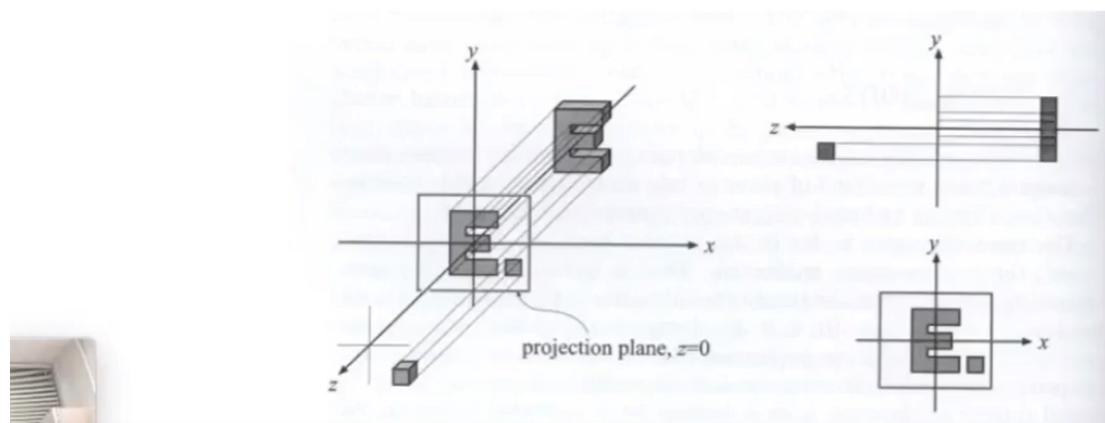
一种简单的投影方式

把物体的Z坐标去掉，就可以得到得到在 XOY 上的投影了

再进行平移和缩放，让投影落在 $[-1, 1]^2$ 中（方便后面的计算）

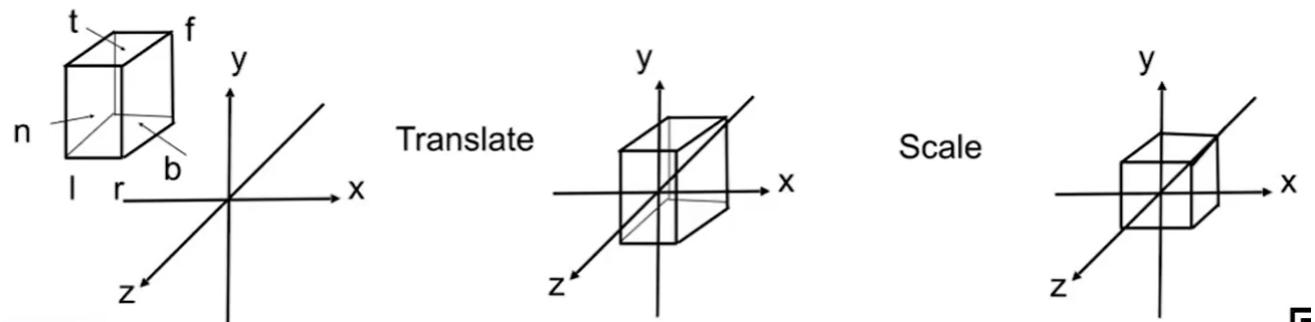
- A simple way of understanding

- Camera located at origin, looking at $-Z$, up at Y (looks familiar?)
- Drop Z coordinate
- Translate and scale the resulting rectangle to $[-1, 1]^2$



一般方法

- We want to map a cuboid $[l, r] \times [b, t] \times [f, n]$ to the “canonical (正则、规范、标准)” cube $[-1, 1]^3$



1 定义立方体的左l、右r、下b、上t、远f、近n ($l < r, b < t, f < n$ (因为是在-z方向上的))
opengl使用左手系也是这个原因

2 转换为标准立方体

先平移，再缩放

• Transformation matrix?

- Translate (**center** to origin) **first**, then scale (length/width/height to **2**)

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

透视投影 (存在近大远小)

性质

平行线会相交与一点

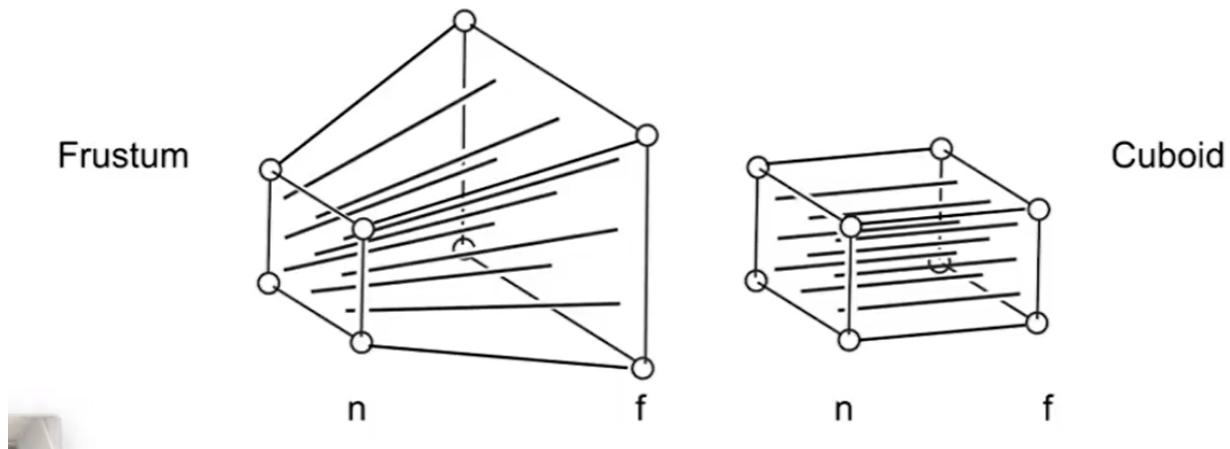
先验知识

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \text{ 是三维点} \quad \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix} \quad (w \neq 0)$$

步骤

- How to do perspective projection

- First “squish” the frustum into a cuboid ($n \rightarrow n, f \rightarrow f$) ($M_{\text{persp} \rightarrow \text{ortho}}$)
- Do orthographic projection (M_{ortho} , already known!)

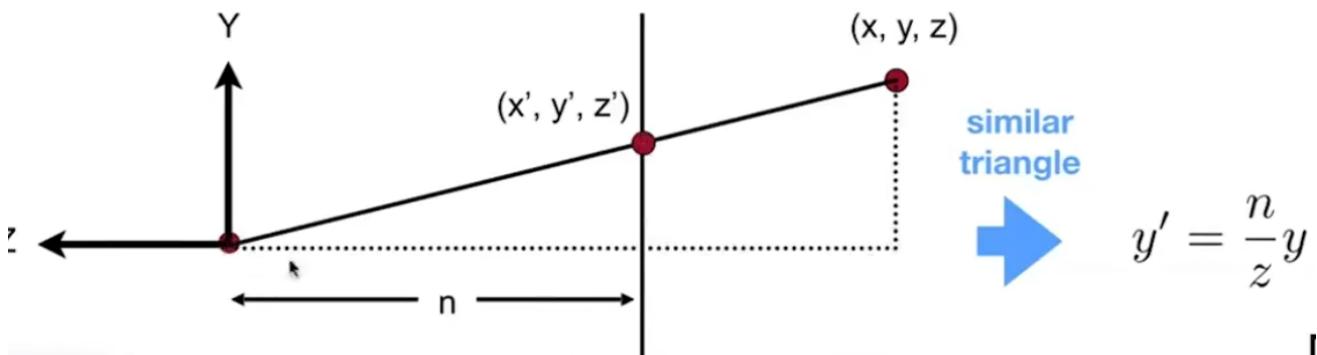


1 把椎体‘挤压’成长方体

近平面（n面）不变

远面（f面）Z值不变，远面中心不变

- Recall the key idea: Find the relationship between transformed points (x', y', z') and the original points (x, y, z)



相似三角形求 $y' = \frac{n}{z}y, x' = \frac{n}{z}x$

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \underset{\text{mult. by } z}{=} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

得到挤压后的点，可以算出一部分变换矩阵

$$M_{persp \rightarrow ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

根据近平面 (n面) 上的点不变，则点 (x,y,n,1) 不变，可以得到下面的推导，得到第一个方程 $An + B = n^2$

$$M_{persp \rightarrow ortho}^{(4 \times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix} \xrightarrow{\text{replace } z \text{ with } n} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} == \begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix}$$

- So the third row must be of the form (0 0 A B)



$$(0 \quad 0 \quad A \quad B) \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \text{n}^2 \text{ has nothing to do with x and y}$$

根据远面z值不变，可以得到第二个方程， $Af + B = f^2$

Any point's z on the far plane will not change



$$\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} == \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} \xrightarrow{\text{ }} Af + B = f^2$$

得到 $A = n + f, B = -nf$, 矩阵完成，挤压完成。

2做正交投影

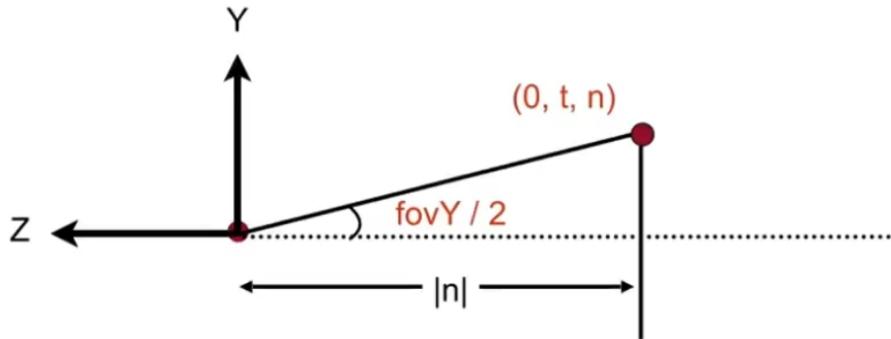
3得到投影

第四讲 光栅化（三角形）

需要定义的东西

1 画面的长宽比 aspect

2 垂直可视角度 (vertical field of view,fovY)



$$\tan \frac{fovY}{2} = \frac{t}{|n|}$$
$$aspect = \frac{r}{t}$$

MVP之后做什么？

M, 模型变换，即摆放模型

V, 视图变换，即摆放相机

P, 投影，得到一个 $[-1, 1]^3$ 的标准立方体

把立方体放在屏幕上

屏幕

定义

1 二维数组，由像素 (pixel) 组成

2 一种典型的光栅 (raster==德语的屏幕, rasterize==把东西画在屏幕上) 成像设备

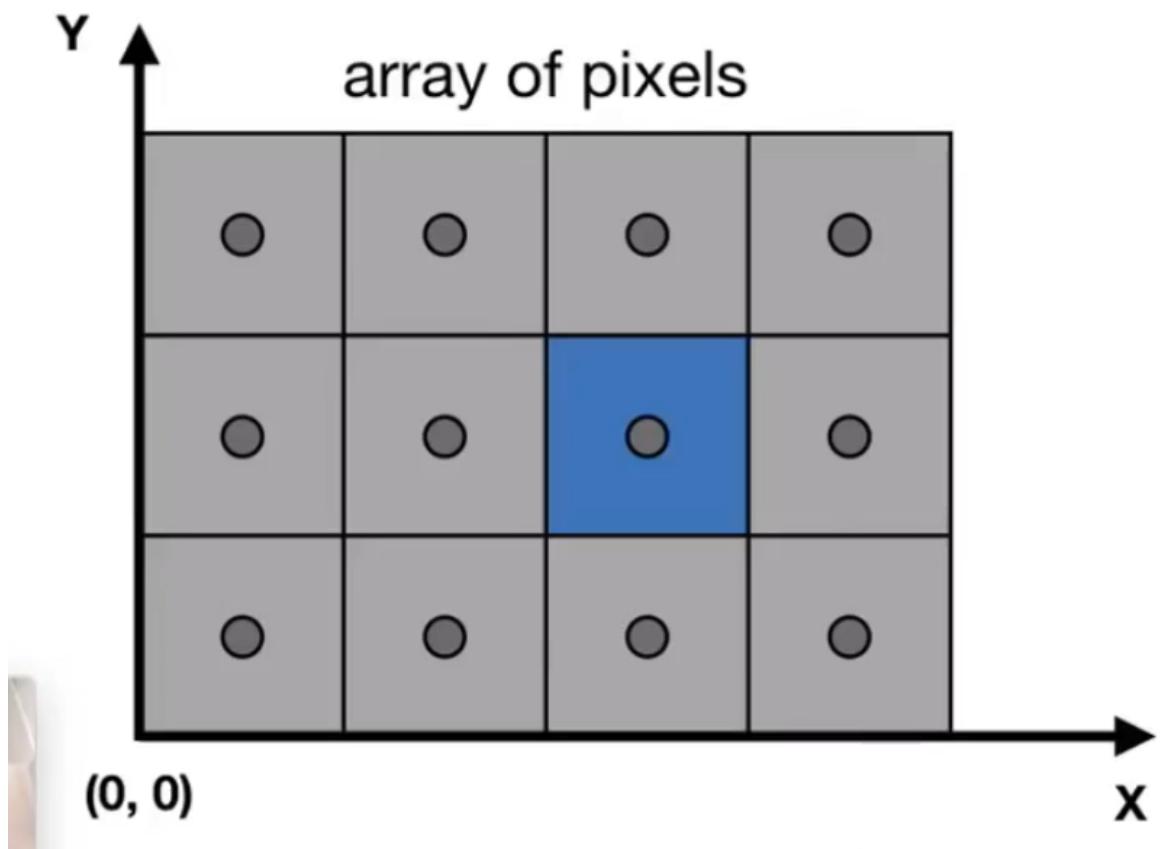
定义屏幕空间

1 像素坐标为 (x, y) , x, y 整数，从 $(0, 0)$ 到 $(width - 1, height - 1)$, 像素中心在 $(x + 0.5, y + 0.5)$

2 屏幕覆盖范围 $(0, 0)$ 到 $(width, height)$

Defining the screen space

- Slightly different from the “tiger book”



让立方体在屏幕空间中显示

使用如下方程即可（视口变换）

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

将三角形光栅化成像素

为什么是三角形？

1 三角形是最基础的多边形
2 其他的多边形都可以拆解为三角形
3 一定在一个平面内
4 方便定义
5 方便内部根据三个顶点渐变

步骤

1 判断像素的中心点是否在三角形内

(1) 采样 (sampling)

对每一个点进行判断（也可以以三角形的上下左右界做一个长方形，只判断这里面的点），计算函数值（比如在三角形内为1，不在为0）

判断函数实现：

做叉积，判断是否同号，正好在边上，可以算也可以不算

2 在内的让像素点亮起

第五讲 光栅化 (反走样)

反走样anti-aliasing (一种简单的抗锯齿MSAA multi sampling anti aliasing)

走样的类型

1 锯齿
2 摩尔纹
3 车轮效应...

走样的原因

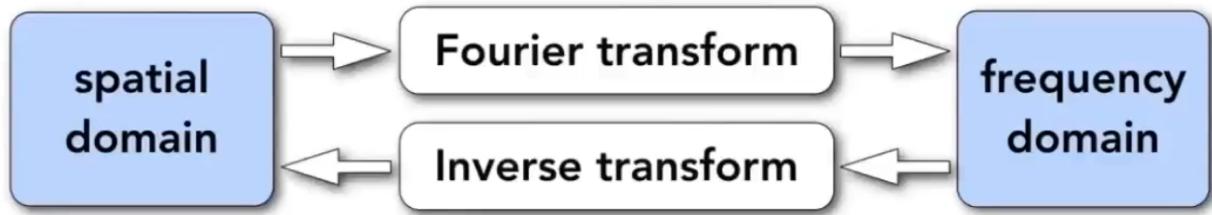
信号太快，但采样比较慢

解释

先验知识：1 傅里叶级数展开，任何周期函数都可以用sin, cos和一个常数的线性组和表示

2 傅里叶变换，将一个信号分解成许多频率的段

$$f(x) \quad F(\omega) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i \omega x} dx \quad F(\omega)$$



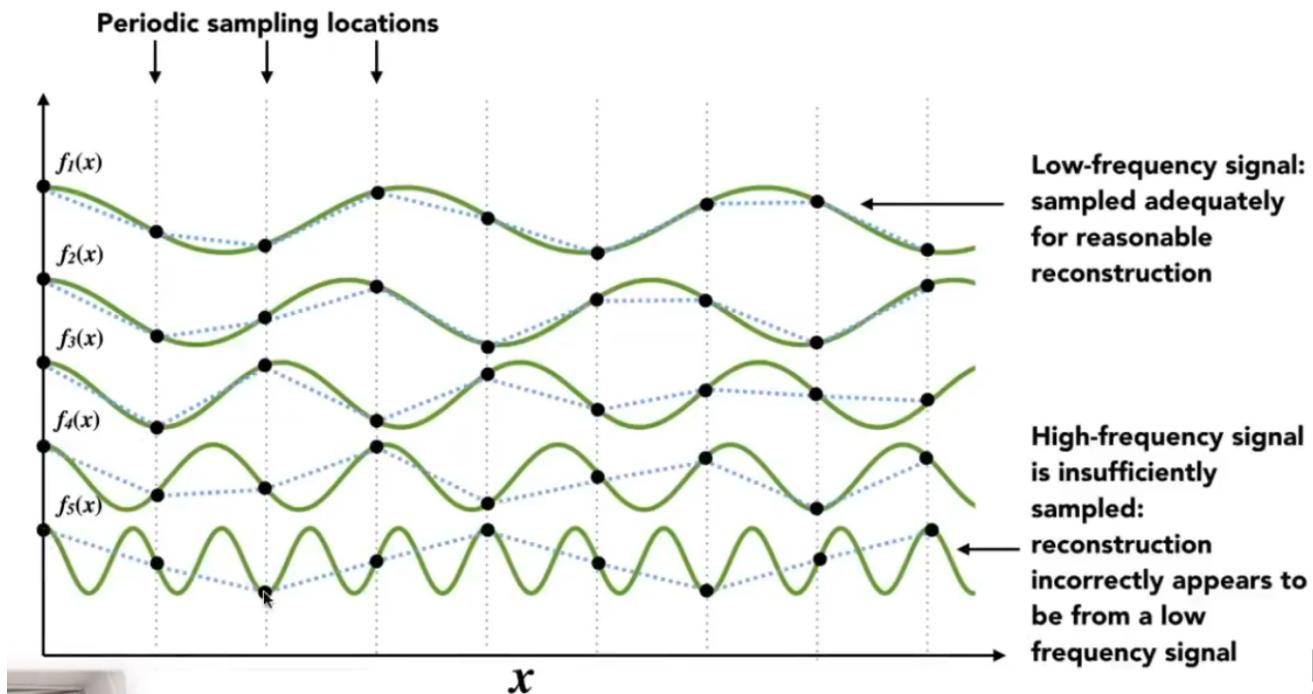
$$f(x) = \int_{-\infty}^{\infty} F(\omega)e^{2\pi i \omega x} d\omega$$

Recall $e^{ix} = \cos x + i \sin x$



不难发现，频率越高，采样结果越差

Higher Frequencies Need Faster Sampling



高频率信息往往在图像中的边界（或者说是一些轮廓），因为这些边界处会发生很大的偏差，所以频率很高

反走样方法

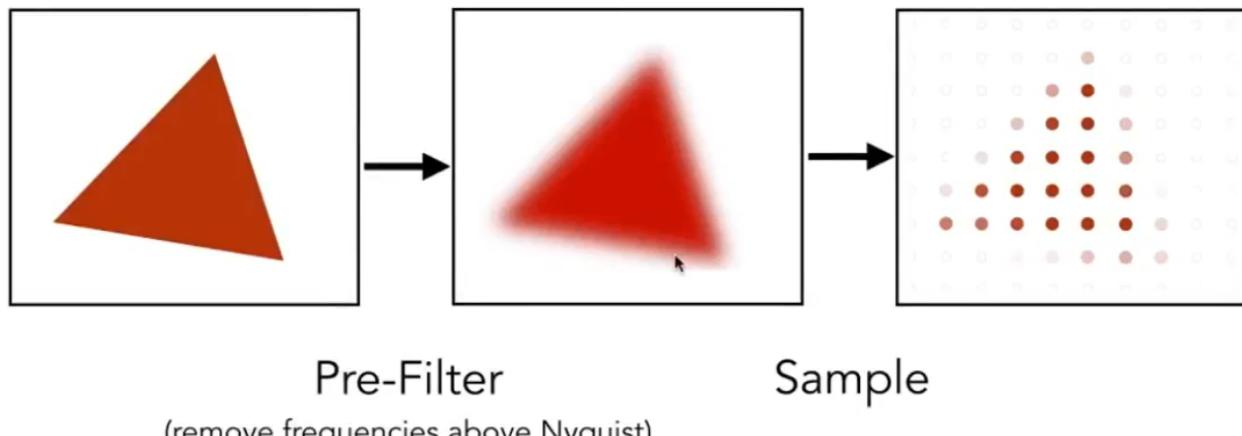
方法一

增加采样率，买一块高分辨率屏幕，但并不实用

方法二

先得到一个模糊（滤波filtering，去掉一些频率）的三角形，在判断像素点是否在模糊的三角形内（步骤不能反）

Antialiased Sampling



Note antialiased edges in rasterized triangle
where pixel values take intermediate values

滤波=卷积（=平均）

卷积可以简单理解为，把一个信号和它周围的信号取一个加权平均，来代替这个信号，就做到了模糊

Convolution

Signal

1	3	5	3	7	1	3	8	6	4
---	---	---	---	---	---	---	---	---	---

Filter

1/4	1/2	1/4
-----	-----	-----

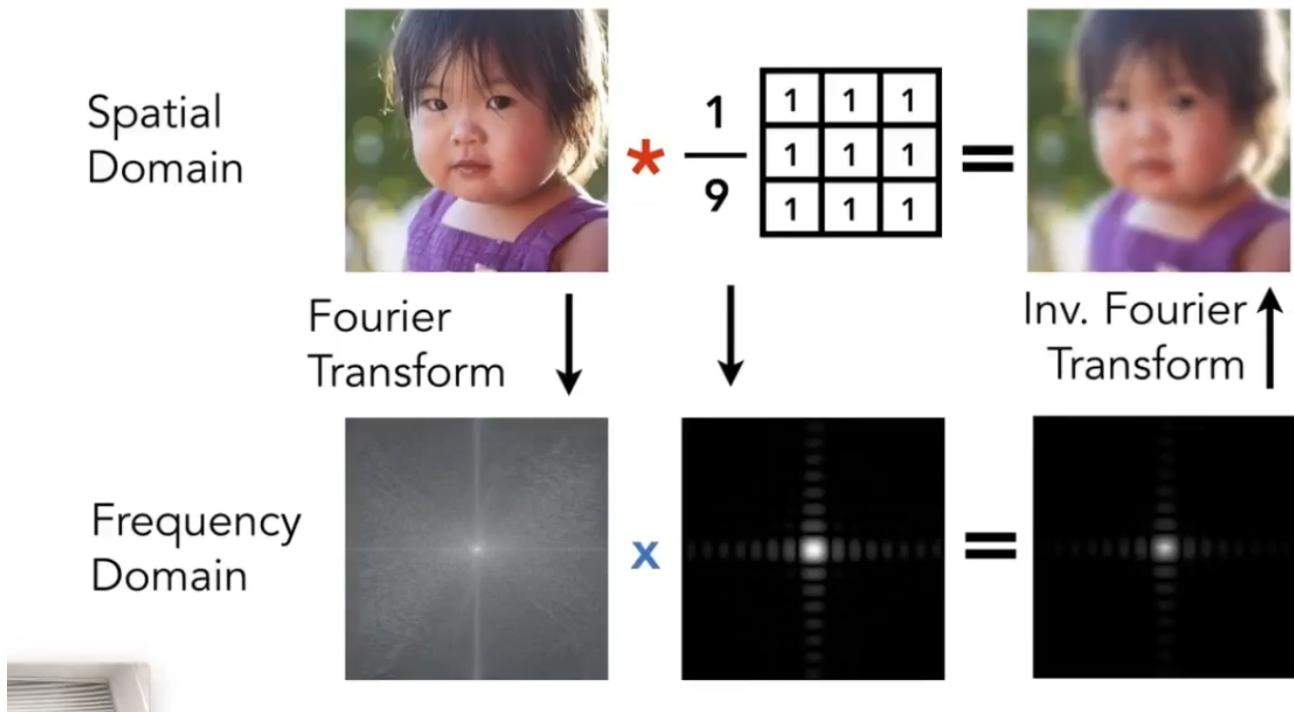
$$3 \times (1/4) + 5 \times (1/2) + 3 \times (1/4) = 4$$

Result

	3	4							
--	---	---	--	--	--	--	--	--	--

并且有，时域上的卷积=频域上的乘积，频域上的卷积=时域上的乘积

Convolution Theorem



滤波器（低通滤波，即只让低频通过），即那个 3×3 的矩阵

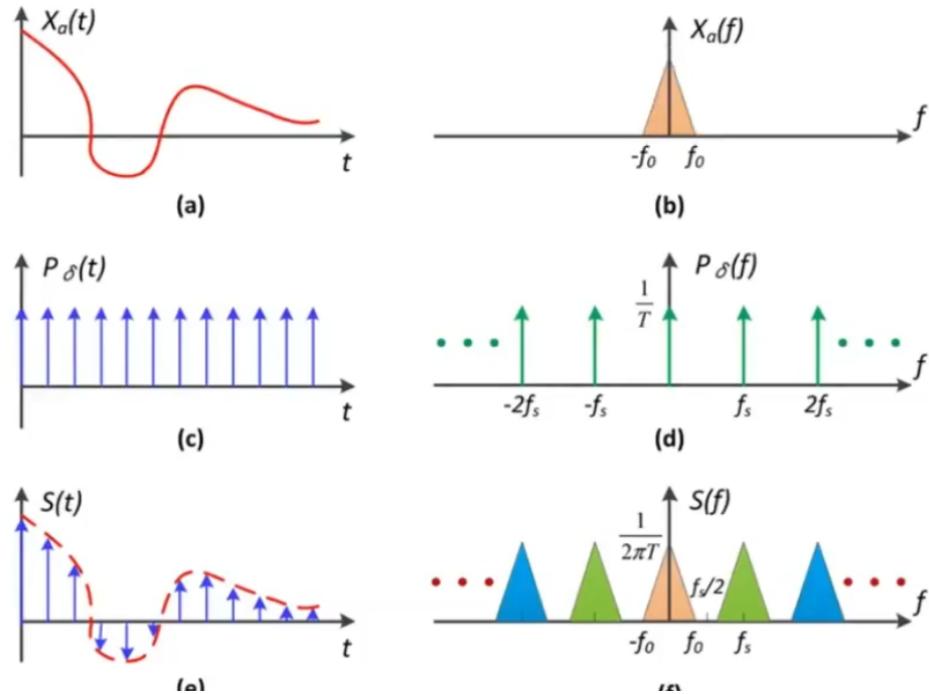
滤波器越大，频域上覆盖的越小（因为模糊的范围变大了，假如大到比图片还大，那就都一个颜色了）

滤波器越小，频域上覆盖的越大（假如只有一个，那就没有过滤了）

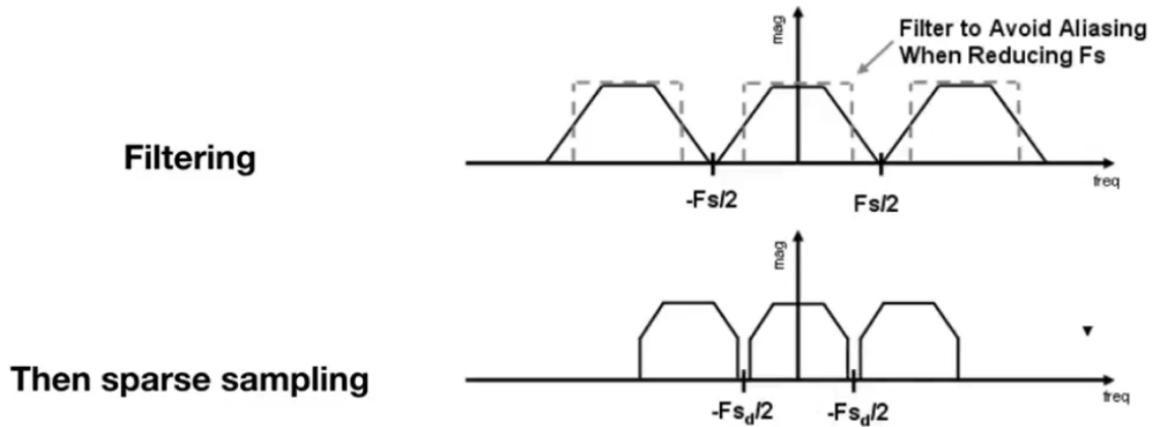
采样（在频域上）=重复原始信号的频谱

$a*c=e$, 所以 b 卷积 $d=f$

Sampling = Repeating Frequency Contents



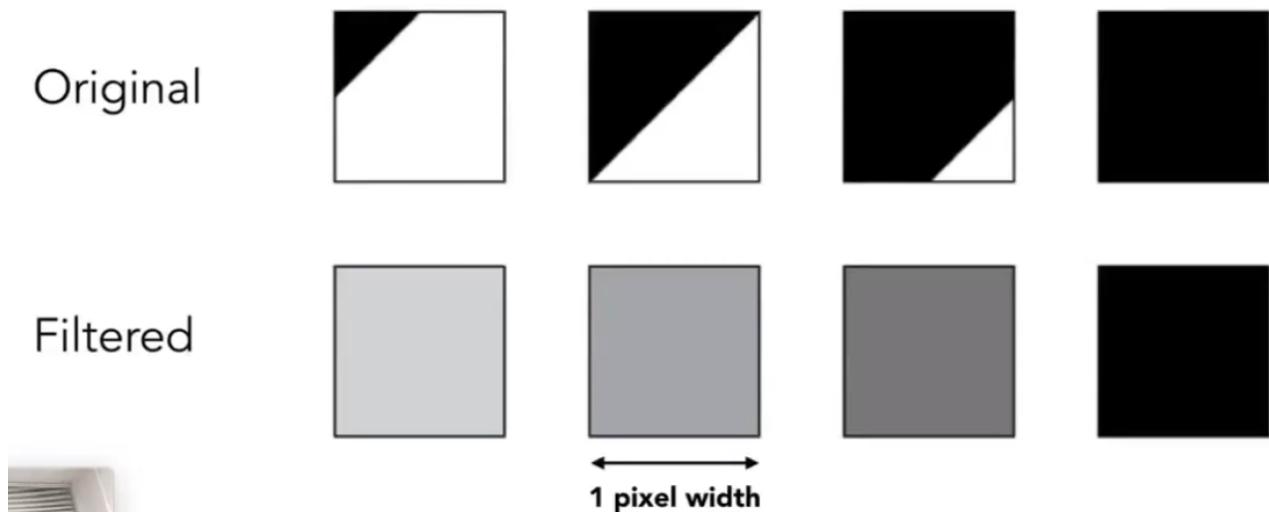
采样率低的时候，重复频谱会重叠，造成走样，所以当我们过滤掉高频的信号，就可以防止堆叠（如下图，假如不过滤高频，下方就会重叠了）



具体操作

1 对每一个像素做一个卷积（做一个平均），对每一个像素的覆盖面积（像素值）求一个平均

In rasterizing one triangle, the average value inside a pixel area of $f(x,y) = \text{inside}(\text{triangle},x,y)$ is equal to the area of the pixel covered by the triangle.



但这个计算比较麻烦，所以我们采用超级采样（supersampling）

把这个像素分解成 4×4 个像素，判断每一个小点是否在三角形中，再平均，就可以得到近似值，这个平均值就是最终的像素颜色。

可以理解为后一步的采用，在这一步模糊的同时其实已经做了，

2 然后对每一个像素采样

显然，这种方法带来了很大的计算量

其他的方法，FXAA（先得到一个有锯齿的图，再把锯齿去除），TAA（复用上一帧的画面）...

第六讲 着色 (shading)

可见性、遮挡

油画家算法

先画最远的，再画最近的

由于要排序，所以复杂度是 $O(n \log n)$ 的

z-buffering 深度缓存算法

算法思想：始终维护最小的z值

每一个三角形的计算是常数级别的，所以复杂度为O(n)

需要存储的信息

1 frame buffer存储颜色的值

2 depth buffer存储深度的值

为了简便，z值为正，z小则近，z大则远

着色

定义

1 shading point (着色点，物体表面的一个点)

2 法线 n

3 观测方向 v

4 光照方向 l

5 表面参数 (颜色，亮度)

6 不考虑影子，只考虑着色本身

7 光的强度 $I = I/r^2$

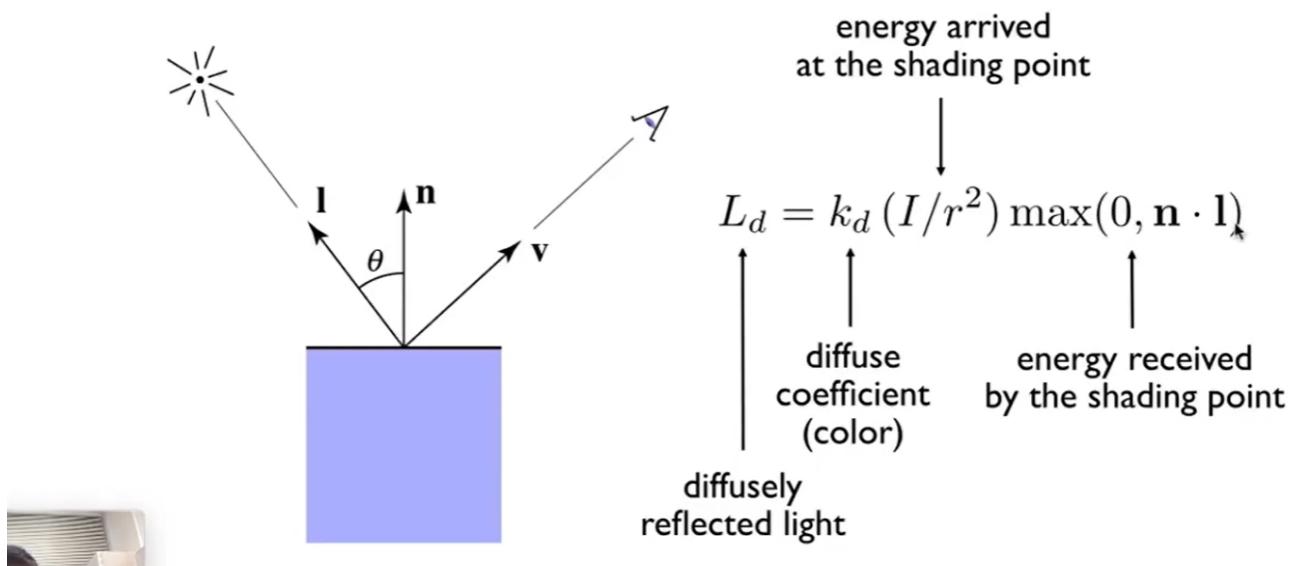
Blin-Phong反射模型

步骤

1 漫反射Lambertian (diffuse) shading

计算漫反射后光的能量，max是因为假如cos为负数，则没有光射过来， $n \cdot l$ 都是单位长度，所以直接得到cos

Shading **independent** of view direction



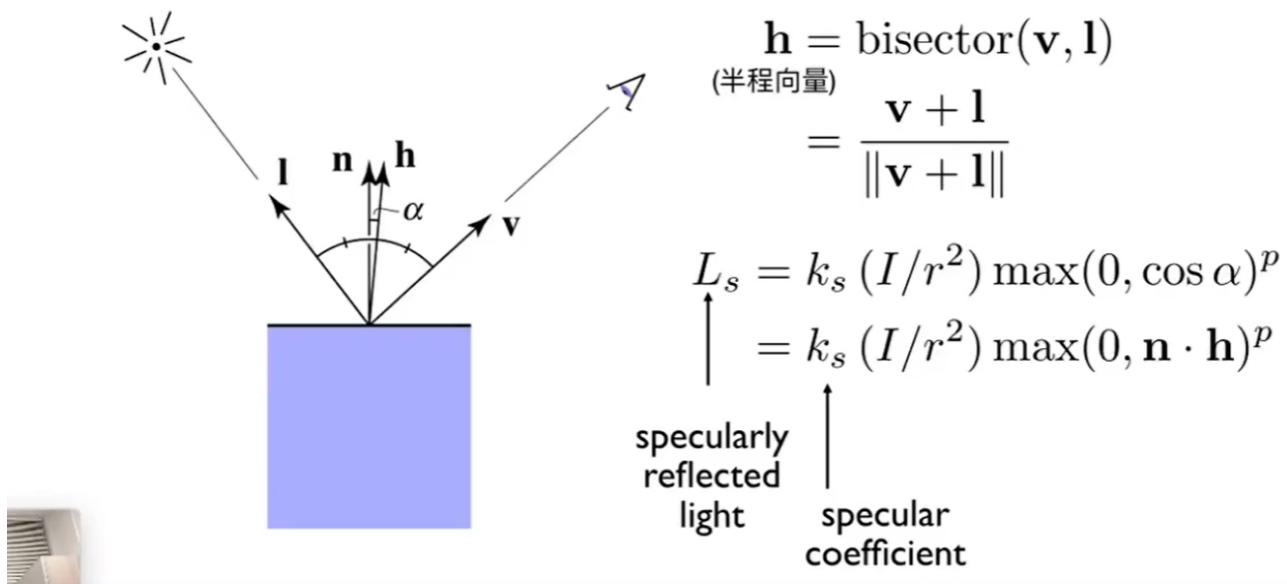
2 高光

h 为 l 和 v 的角平分线， n 和 h 越接近，则 v 和 l 的反射角越接近（blin-Phong的改进，这样比算反射角和 v 的夹角（Phong反射模型）好算）

p （100~200）的次方是因为 $\cos\theta$ 的值比较大，直接用会导致高光过大

V close to mirror direction \Leftrightarrow **half vector near normal**

- Measure “near” by dot product of unit vectors



3 环境光照 (不被光源直接照射，但是会被环境中的漫反射照亮)

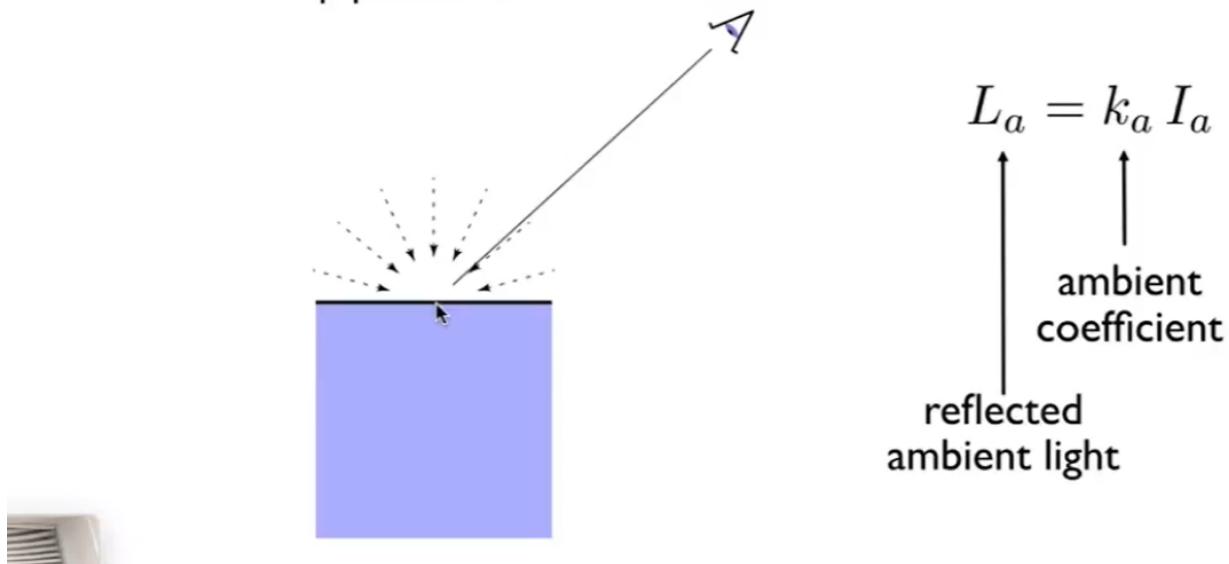
从各个方向来，所以和 v , n 都没有关系，是一个常数

实际上只是个假设，并不是真实的

Ambient Term

Shading that does not depend on anything

- Add constant color to account for disregarded illumination and fill in black shadows
- This is approximate / fake!



着色频率

flat shading

着色每一个三角形 (flat shading)，认为一个三角形是一个shading point，两条边做叉积得出法线（对于平滑平面效果不好，三角形内部没有颜色过渡）

Gouraud shading

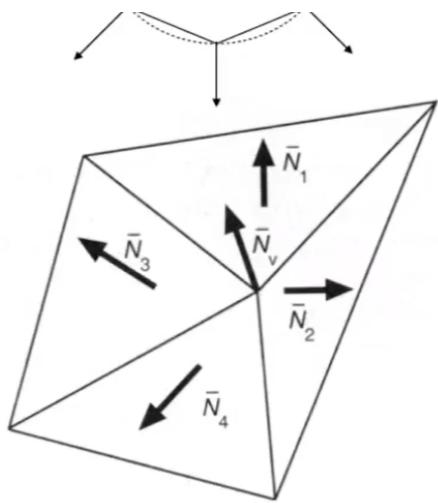
对于每一个顶点求法线着色 (Gouraud shading)

每一个顶点都是周围许多三角形共用的一个顶点，那么这个顶点的法线，就是这些面的法线的平均（加权平均，按三角形面积）

Otherwise have to infer vertex normals from triangle faces

- Simple scheme: **average surrounding face normals**

$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$



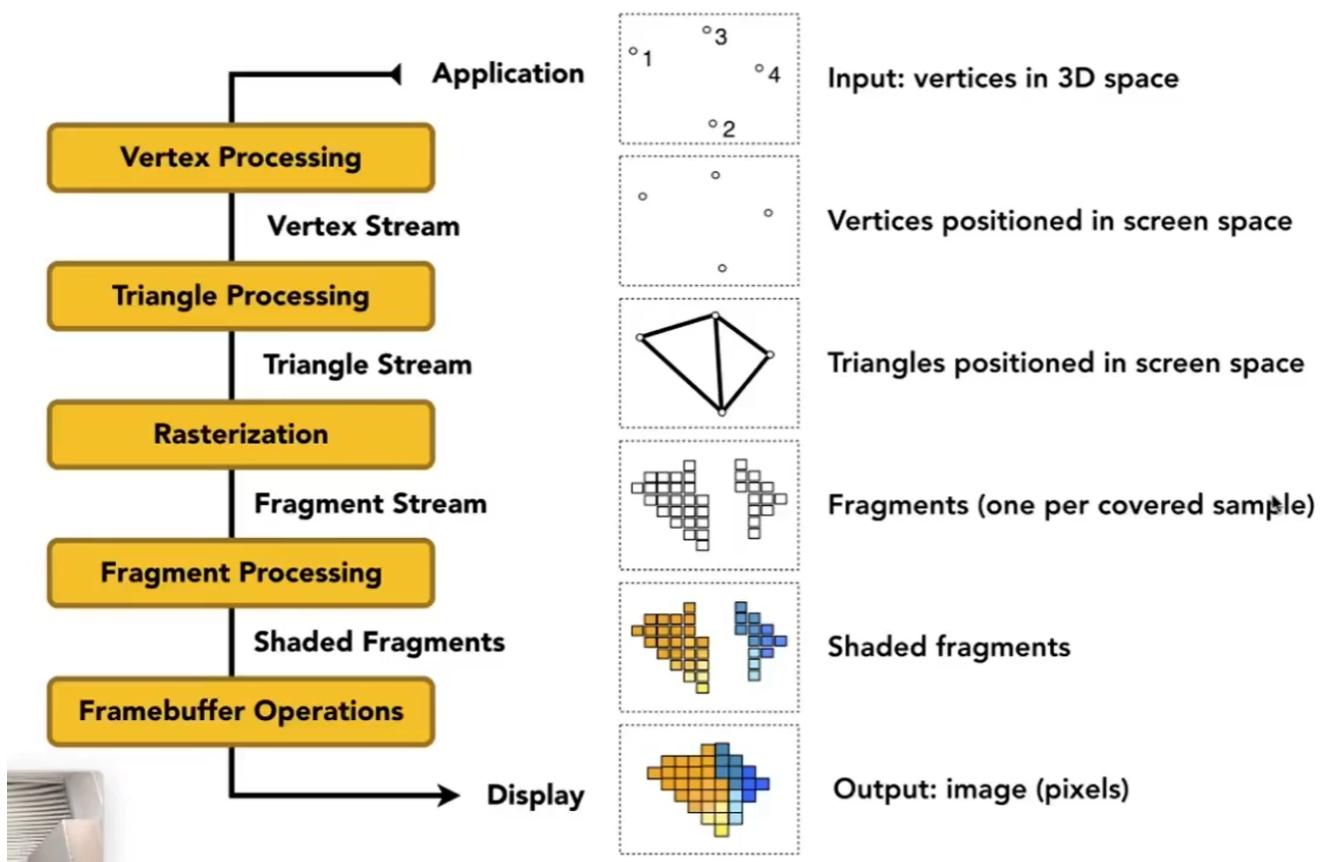
Phong shading

对三角形每一个顶点做法线，对于每一个像素点插值、计算、着色（不是Blin-Phong 反射模型），

图形管线 (实时渲染管线)

变换->投影->光栅化（是否在三角形内，是否可见）->着色

Graphics Pipeline



shader

设计顶点/像素如何着色的代码

vertex shader

pixel/fragment shader

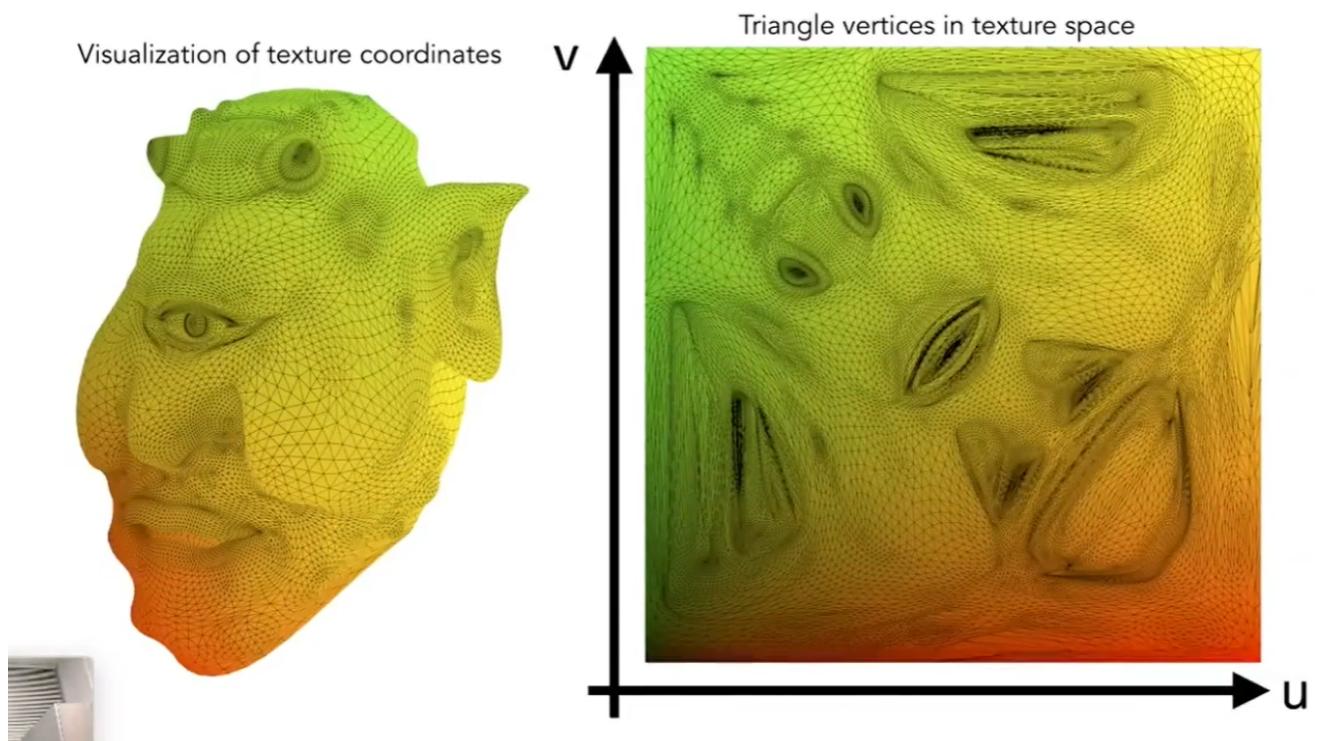
纹理映射

定义

纹理就是三维物体上展开的一个小的二维平面

每一个纹理我们定义一个坐标， u, v 的范围 $(0,1)$

Each triangle vertex is assigned a texture coordinate (u,v)



插值--重心坐标

目的

做三角形内部的插值，在三角形内部平滑过渡

插什么值

纹理坐标、颜色、法线

定义

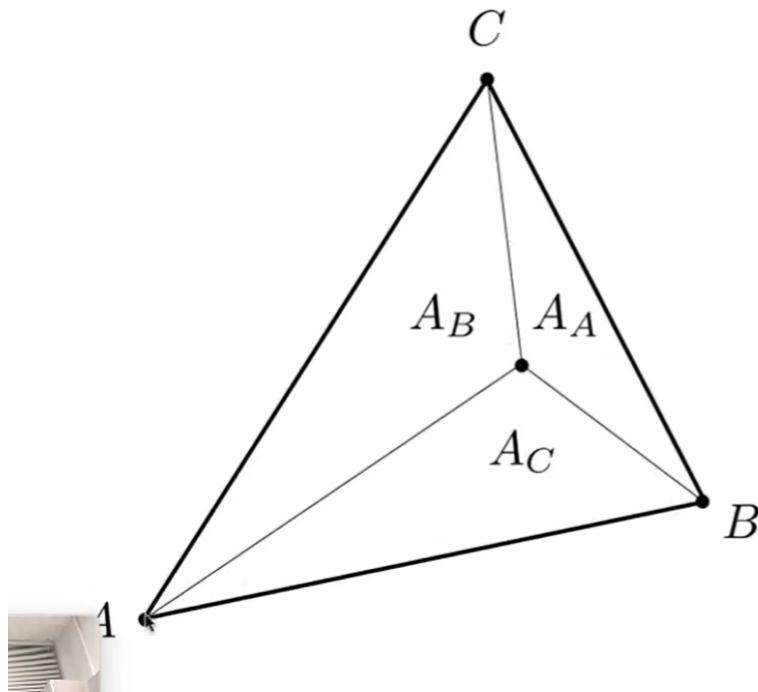
对于一个三角形内部的点 $(x, y) = \alpha A + \beta B + \gamma C, \alpha + \beta + \gamma = 1, \alpha, \beta, \gamma \geq 0$, 可以这么表示, 那么我们可以用 (α, β, γ) 表示这个点, 即这个点的重心坐标

重心

$(x, y) = \frac{1}{3}\alpha + \frac{1}{3}\beta + \frac{1}{3}\gamma$, 把三角形分成三个等面积的三角形

也可以用面积来表示

Geometric viewpoint — proportional areas



$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$
$$\beta = \frac{A_B}{A_A + A_B + A_C}$$
$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

插值方法

用这三个系数去和三角形三个顶点的属性做线性组合, $V = \alpha V_A + \beta V_B + \gamma V_C$, V 可以是坐标、纹理坐标、颜色、法线、深度、材质属性...

坏处

投影后会改变, 所以三维的需要先插值, 再投影

应用纹理

方法

使用纹理坐标，应用到对应的采样坐标上即可

带来的问题

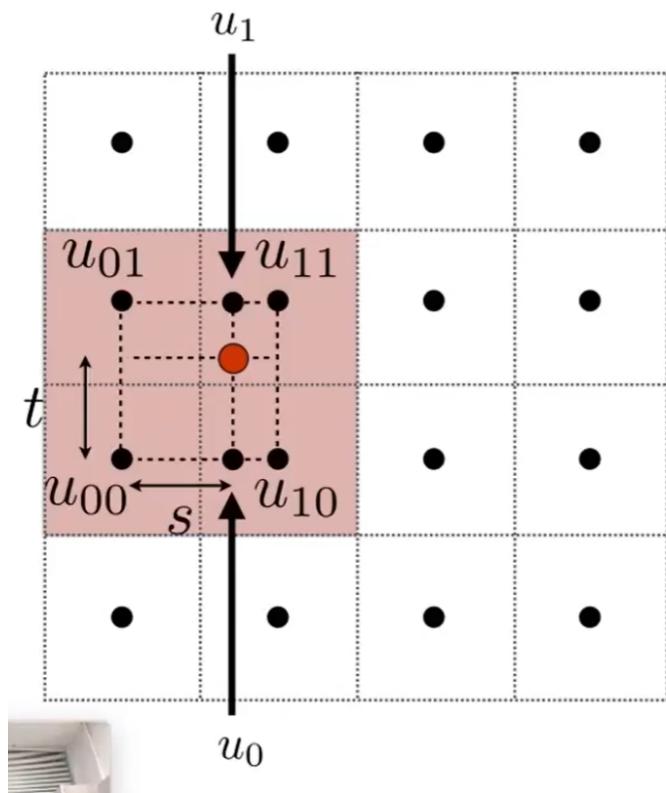
假如纹理太小了，显示出来的效果会不清晰（Bilinear双线性插值）

解决办法：Bilinear双线性插值（质量一般）

对于一个非整型的坐标，如何给他一些值，让过渡更平滑呢？

我们参考它附近的四个点，做两次线性插值，根据s, t的大小决定值的大小更接近哪个像素

Bilinear interpolation



Linear interpolation (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Two helper lerps

$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

$$u_1 = \text{lerp}(s, u_{01}, u_{11})$$

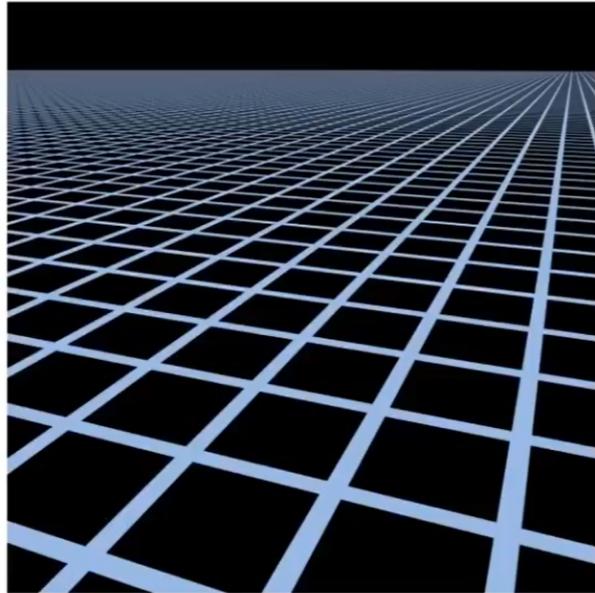
Final vertical lerp, to get result:

$$f(x, y) = \text{lerp}(t, u_0, u_1)$$

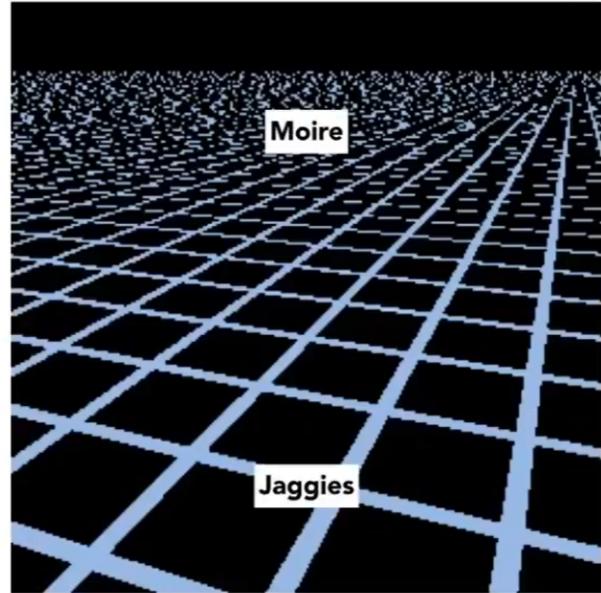
办法二：Bicubic（取周围十六个，做三次的插值），运算量大

纹理太大了会走样（mipmap、各向异性过滤）

近处一个像素覆盖的纹理较小，远处一个像素会覆盖大量的纹理



Reference



Point sampled

超采样，可以解决问题，但是开销太大了

解决方法： **mipmap**（快速，近似，正方形的范围查询）

1每次将边长/2，最终存储量为 $1 + \frac{1}{4} + \frac{1}{16} + \dots = \frac{4}{3}$ ，存储量只多了 $\frac{1}{3}$

"Mip" comes from the Latin "multum in parvo", meaning a multitude in a small space



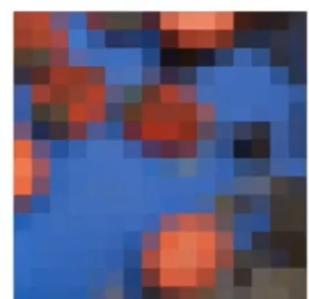
Level 0 = 128x128



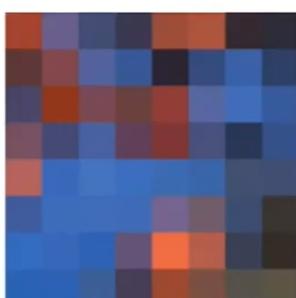
Level 1 = 64x64



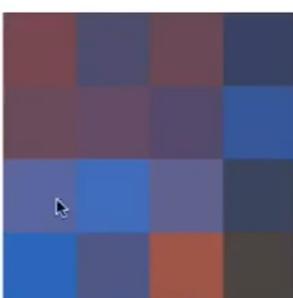
Level 2 = 32x32



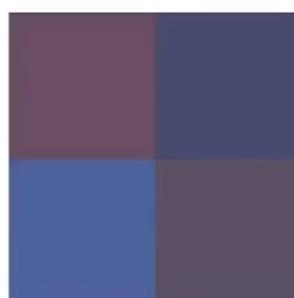
Level 3 = 16x16



Level 4 = 8x8



Level 5 = 4x4



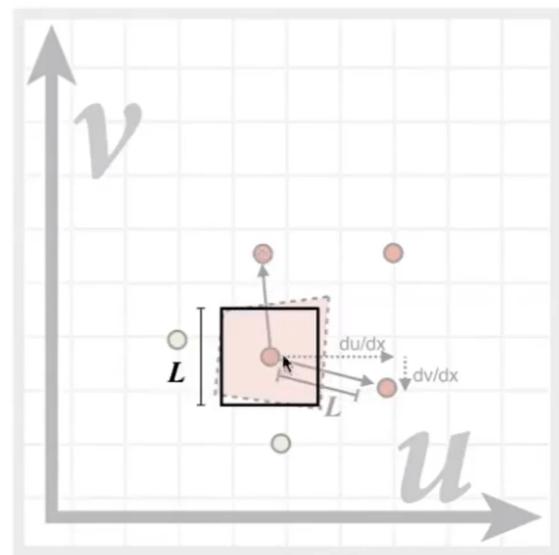
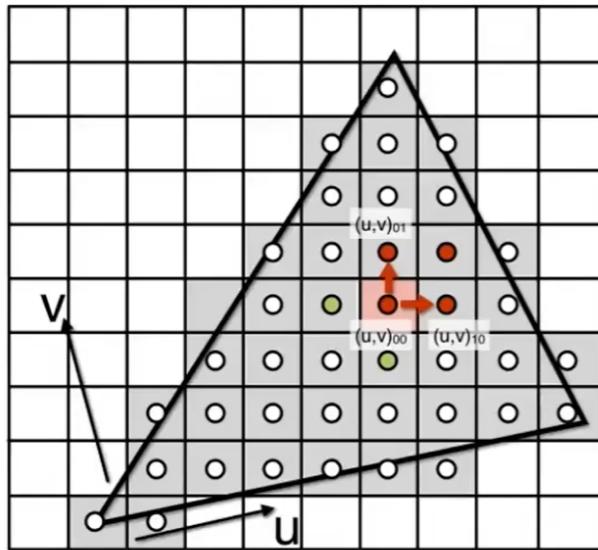
Level 6 = 2x2



Level 7 = 1x1

2找到某个像素点对应的纹理范围（近似）

方法是找到周围三个点（如图所示）对应的纹理坐标，确定一个L（一般是最长的那个距离，如图），再以像素点对应的纹理坐标为中心，边长为L的正方形为所得范围

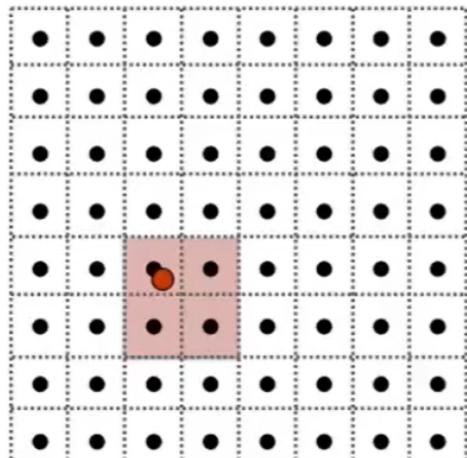


3查询这个范围的平均值

根据范围的大小，直接去查那张图（第 $D = \log_2 L$ 张图）的位置的值就好了，但是得到的结果比较离散

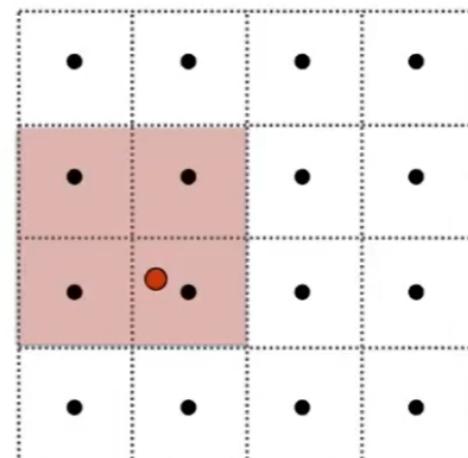
可以对层与层之间做三线性插值，这样就可以得到连续的值了

分别在D、D+1层做双线性插值，再对他们两个做线性插值



Mipmap Level D

Bilinear result



Mipmap Level D+1

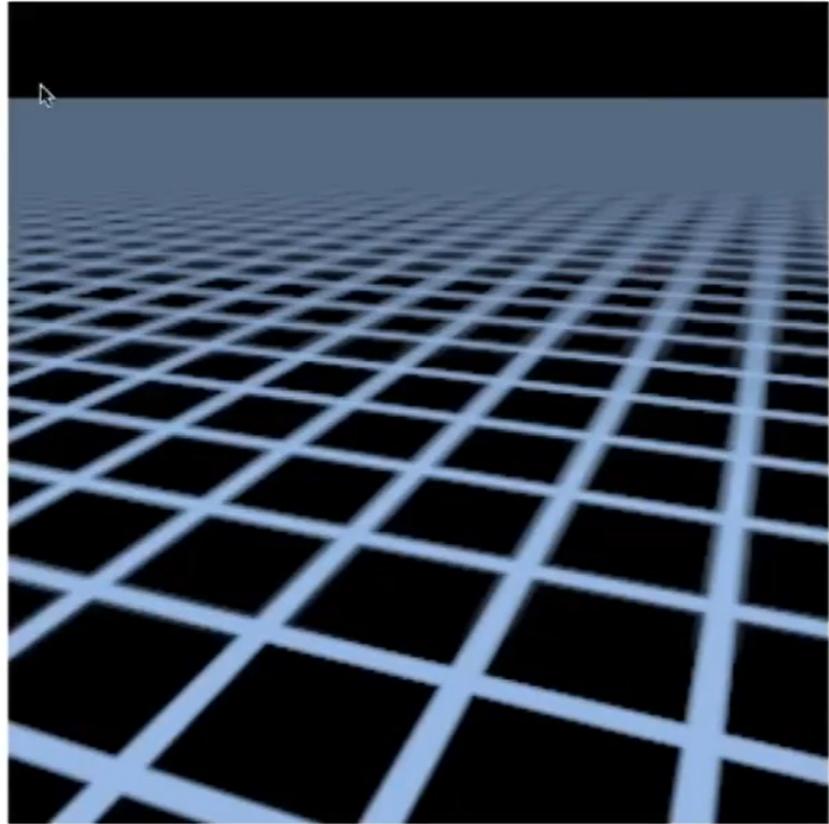
Bilinear result

Linear interpolation based on continuous D value

4 结果会比较糊

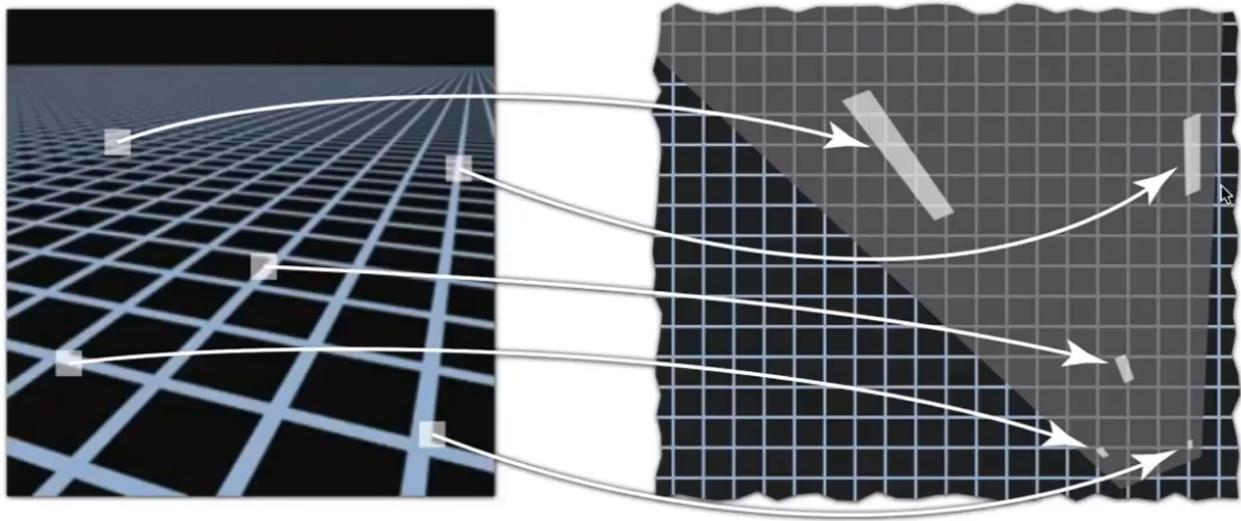
Overblur

Why?



Mipmap trilinear sampling

因为取得对应的材质范围是正方形，而一般情况很可能不是正方形，而是不规则的



Screen space

Texture space

解决办法，各项异性过滤

mipmap里存储的都是边长都除2的图片，这里我们也存储只缩小长/宽的图片ripmap，

这样就可以解决一部分长方形的范围了，代价为开销为原来的3倍

但是比较奇怪的，比如最上面的斜长条形，依然不能解决

解决办法有EWA过滤，开销就更大了

具体应用

环境光照 environment map

用纹理反映环境光（球形环境光，可是展开物体会扭曲）



Light from the environment



Rendering with the environment

[Blinn & Newell 1976]

凹凸贴图bump mapping

在不把几何形体变复杂的情况下，定义一个复杂纹理，来显示出凹凸的效果

对于一些映射来的纹理，定义一个假的法向量，从而欺骗shader，画上一些较暗的部分，达到凹凸的效果

步骤

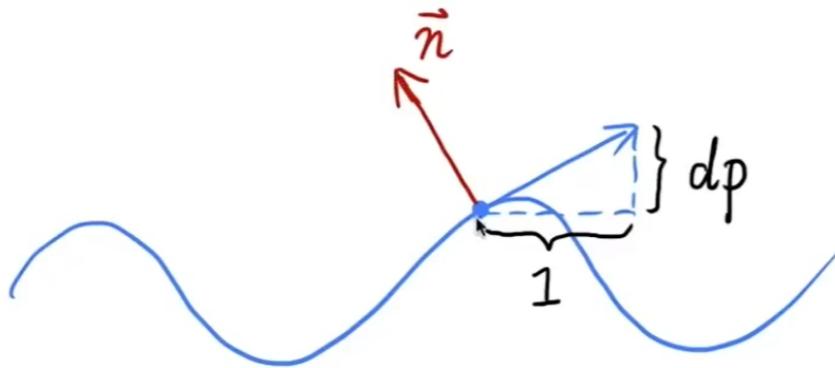
1 不给原来的每个三角形增加额外的细节

2 给每个像素增加扰动（height shift）

3 怎么修改法向量？

对于二维

- Original surface normal $n(p) = (0, 1)$
- Derivative at p is $dp = c * [h(p+1) - h(p)]$
- Perturbed normal is then $n(p) = (-dp, 1).normalized()$



对于三维，分别计算u方向和v方向的切线

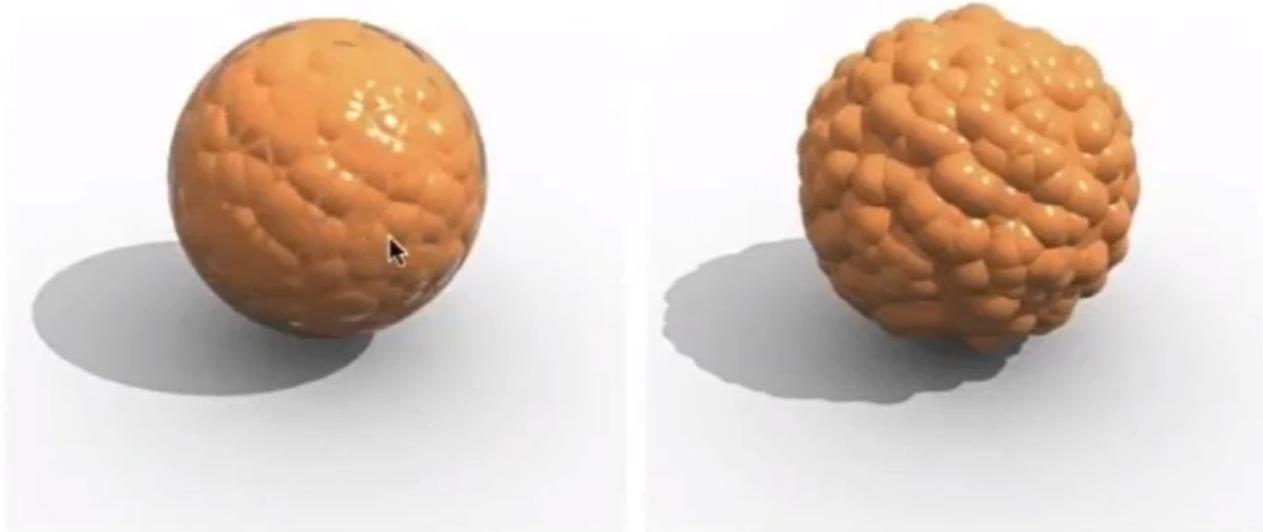
对于一般情况， p 法向量可能不朝向 $(0,0,1)$,所以需要定义一个局部的坐标系，使 p 的法向量始终朝向 $(0,0,1)$

- Original surface normal $n(p) = (0, 0, 1)$
- Derivatives at p are
 - $dp/du = c1 * [h(u+1) - h(u)]$
 - $dp/dv = c2 * [h(v+1) - h(v)]$
- Perturbed normal is $n = (-dp/du, -dp/dv, 1).normalized()$

位移贴图 displacement mapping

把顶点位置移动了，要求三角形够细

凹凸贴图只是改变法线方向，欺骗人的眼睛，实际的几何形体没有变，所以在边缘和阴影处会露馅



Bump / **Normal** mapping

Displacement mapping

三维纹理

第七讲 几何

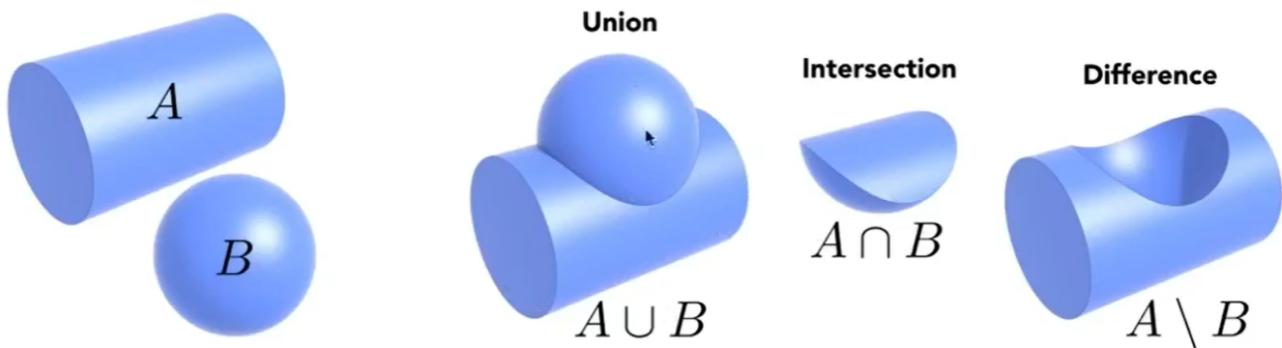
隐式表示 implicit

定义方法

1 满足某种关系的点的集合，比如圆 $x^2 + y^2 + z^2 = 1$, 或者其他的方程 $f(x, y, z) = 1$

2 通过布尔运算组合几何形体

Combine implicit geometry via Boolean operations

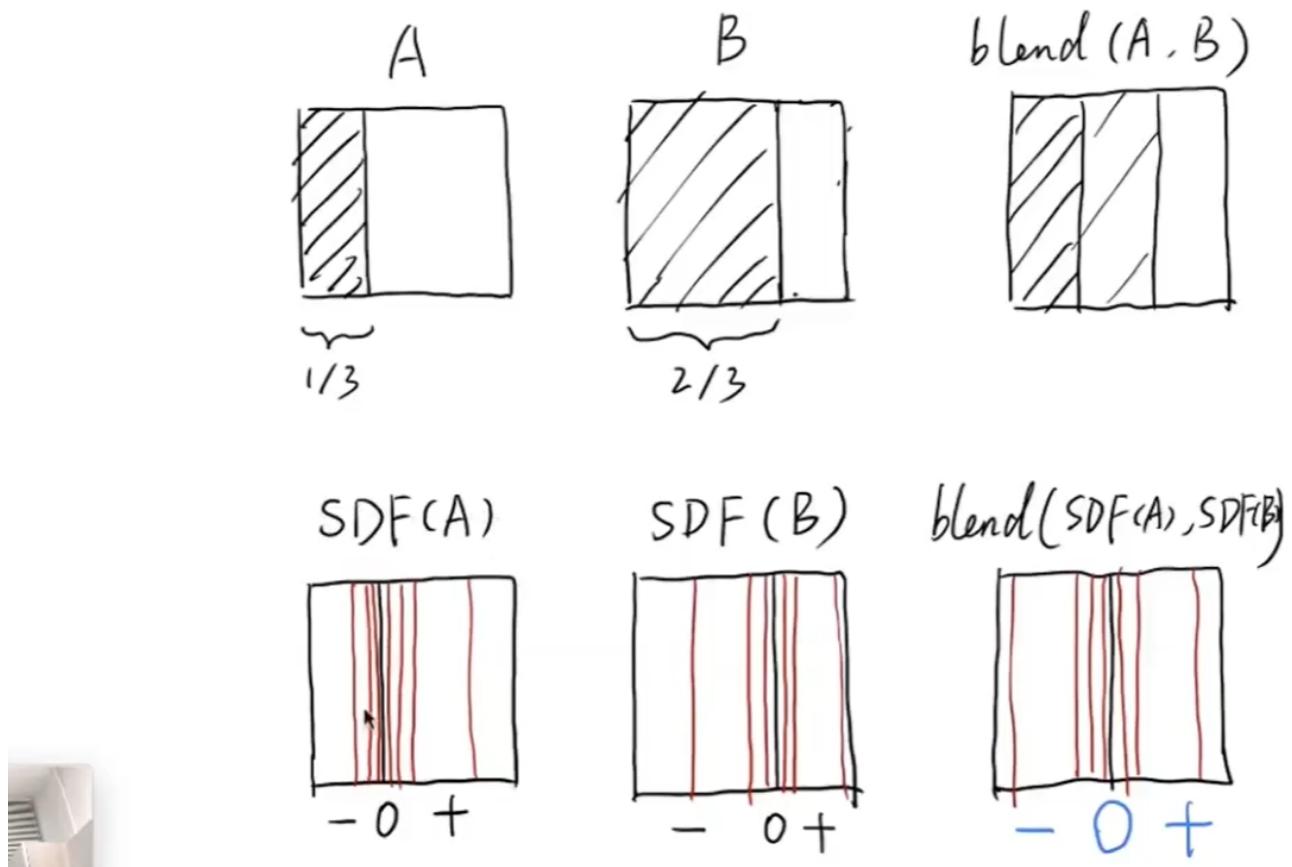


3 距离函数

比如，等高线

表示点到边界的最短距离

An Example: Blending a moving boundary



4分形表示

有点像递归

Fractals (implicit)

Exhibit self-similarity, detail at all scales
“Language” for describing natural phenomena
Hard to control shape!

Three images illustrating fractal patterns:
1. A blue and orange fractal coastline, showing a complex, self-similar pattern.
2. A close-up of a green Romanesco broccoli, showing its characteristic fractal-like structure.
3. A satellite view of a river delta, showing a complex, branching network that exhibits fractal properties at different scales.

问题

采样的时候比较困难，比较难找到所有的点，也很难看出他是什么样的。

好处

方便判断点是否在面上

显式表示 explicit

方法

1 点云，用表面上的点来表示，一系列点的集合

表示简单，但需要特别多的点

2 多边形面（polygon mesh）用的最多

通过存储多边形（通常是三角形或四边形）来表示面

方便处理、采样等。

比点云会带来更复杂的数据结构

举例 .obj文件

左上为点，右上为法线，左下为纹理坐标，右下为连接方法，格式为 点的索引/纹理坐标索引/法线索引

Commonly used in Graphics research

Just a text file that specifies vertices, normals, texture coordinates **and their connectivities**

```
1 # This is a comment
2
3 v 1.000000 -1.000000 -1.000000
4 v 1.000000 -1.000000 1.000000
5 v -1.000000 -1.000000 1.000000
6 v -1.000000 -1.000000 -1.000000
7 v 1.000000 1.000000 -1.000000
8 v 0.999999 1.000000 1.000001
9 v -1.000000 1.000000 1.000000
10 v -1.000000 1.000000 -1.000000
11
12 vt 0.748573 0.750412
13 vt 0.749279 0.501284
14 vt 0.999110 0.501077
15 vt 0.999455 0.750380
16 vt 0.250471 0.500702
17 vt 0.249682 0.749677
18 vt 0.001085 0.750380
19 vt 0.001517 0.499994
20 vt 0.499422 0.500239
21 vt 0.500149 0.750166
22 vt 0.748355 0.998230
23 vt 0.500193 0.998728
24 vt 0.498993 0.250415
25 vt 0.748953 0.250920
26
```

```
vn 0.000000 0.000000 -1.000000
vn -1.000000 -0.000000 -0.000000
vn -0.000000 -0.000000 1.000000
vn -0.000001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
f 5/1/6 6/10/6 2/9/6
f 5/1/7 8/11/7 6/10/7
f 8/11/7 7/12/7 6/10/7
f 1/2/8 2/9/8 3/13/8
f 1/2/8 3/13/8 4/14/8
47
```

3 参数映射

$$f(u, v) = ((2 + \cos u) \cos v, (2 + \cos u) \sin v, \sin u)$$

What points lie on this surface?

Just plug in (u, v) values!

问题

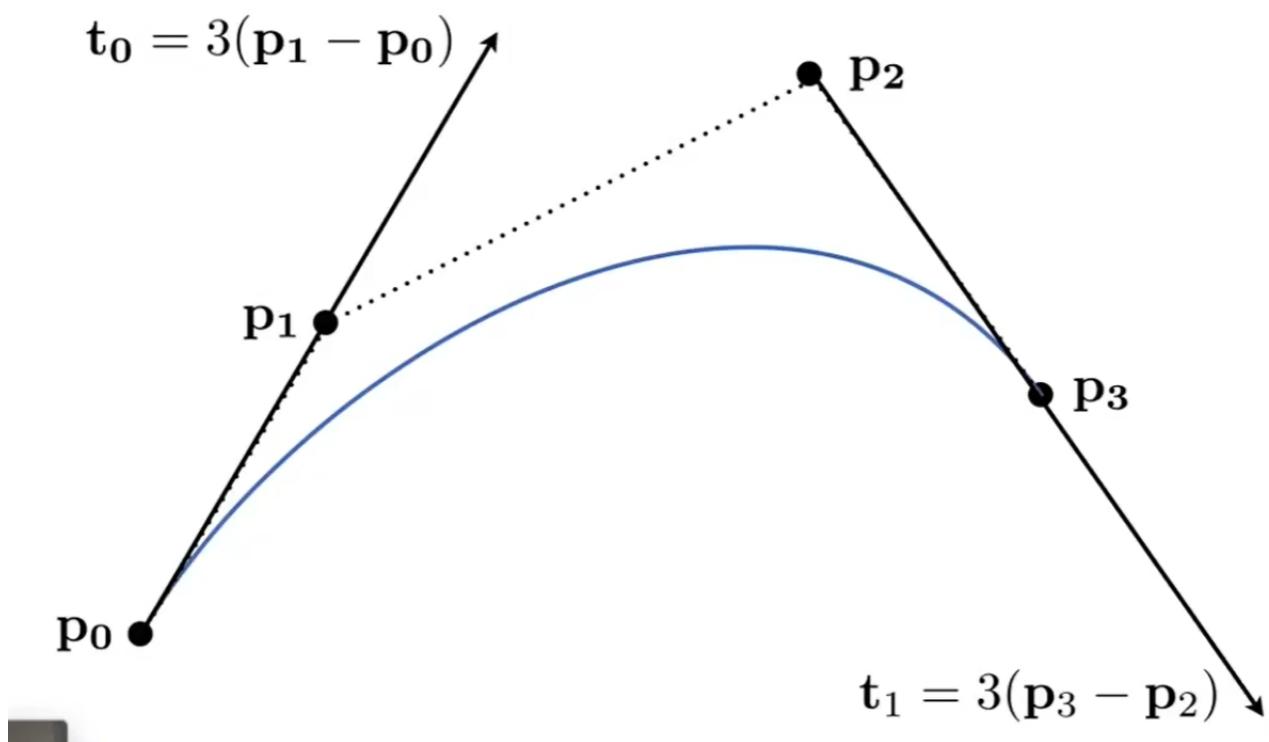
不好判断点相对面在哪

贝塞尔曲线 (Bezier curves)

定义

用一些控制点定义曲线

起点为 p_0 , 终点为 p_3 , 一开始的曲线与 $p_0 p_1$ 相切, 后来的曲线和 $p_2 p_3$ 相切, 除了起始点, 其他点不一定要经过



绘制方法

de Casteljau Algorithm

考虑只有三个点的情况 (quadratic Bezier)

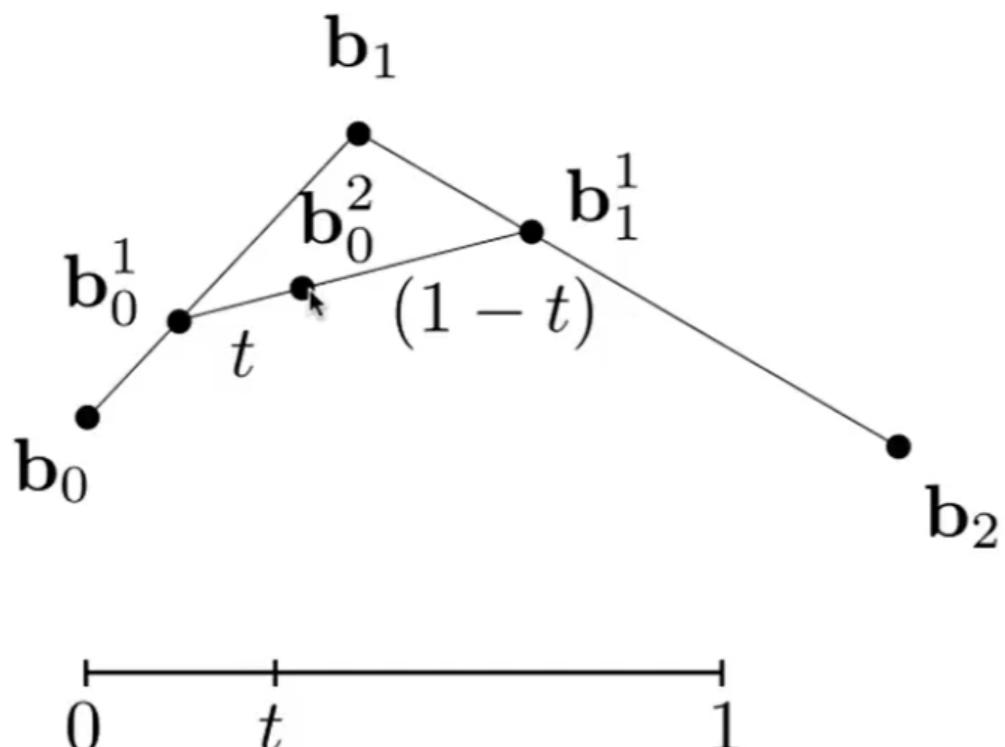
假设从 b_0 到 b_1 , b_1 到 b_2 需要一个单位时间, 初始点出时间设为 0, 终点处为 1, 那我们把 0 到 1 上对应的曲线的点画出来即可

比如说要找 $t = \frac{1}{3}$ 时的点

那么先找到 $\frac{1}{3}b_0 b_1$ 处的点, 和 $\frac{1}{3}b_1 b_2$ 处的点, 把两个点连起来, 得到的直线的 $\frac{1}{3}$ 处为所得

四个点的话, 就是一个递归的过程

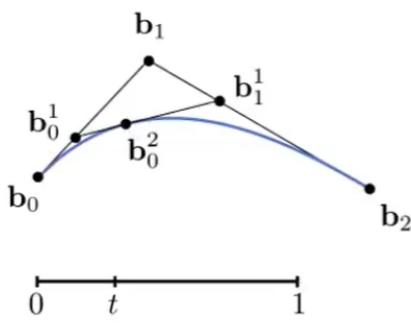
Repeat recursively



代数公式

三个点

Example: quadratic Bézier curve from three points



$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0 + t\mathbf{b}_1$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1 + t\mathbf{b}_2$$

$$\mathbf{b}_0^2(t) = (1-t)\mathbf{b}_0^1 + t\mathbf{b}_1^1$$

$$\mathbf{b}_0^2(t) = (1-t)^2\mathbf{b}_0 + 2t(1-t)\mathbf{b}_1 + t^2\mathbf{b}_2,$$

n个点

Bernstein form of a Bézier curve of order n:

系数 $B_i^n(t) = C_n^i t^i (1-t)^{n-i}$ 为伯恩斯坦多项式

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

好处

1方便做仿射变换，想找仿射变换后的曲线，可以把控制点变换后，再计算曲线即可

仿射变换指的是二维坐标到二维坐标的变换，所以投影变换不行

2 凸包性质

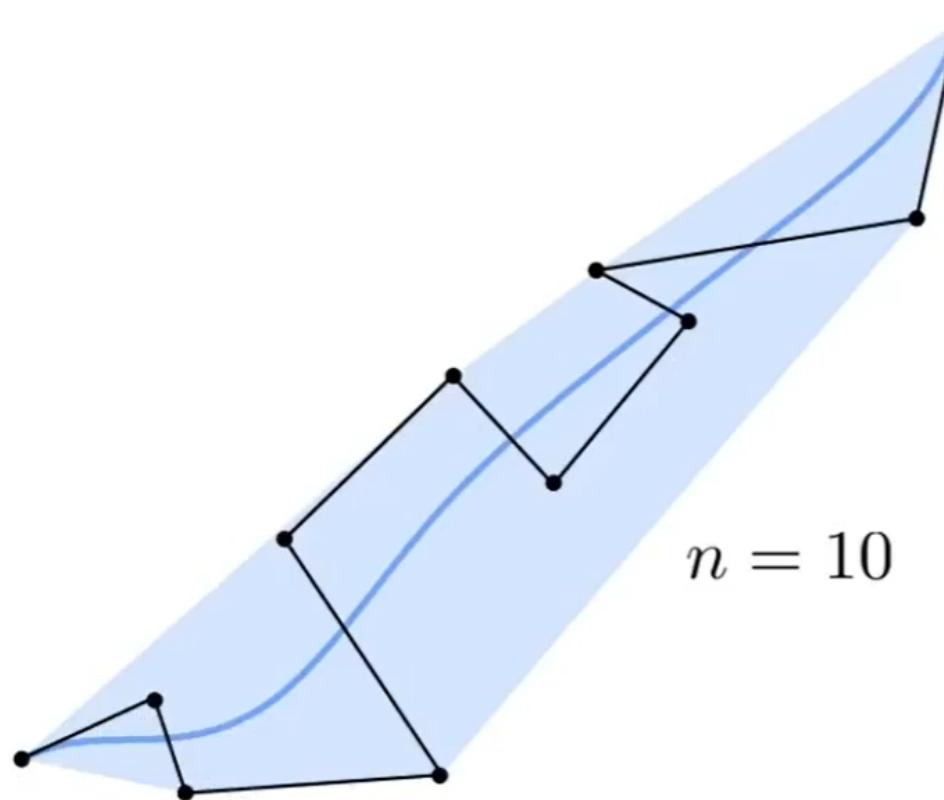
贝赛尔曲线一定在控制点的凸包内

凸包：包围平面上所有点的凸多边形

逐段贝赛尔曲线 (piecewise Bezier)

高阶的贝赛尔曲线可能会变得平滑

Higher-Order Bézier Curves?



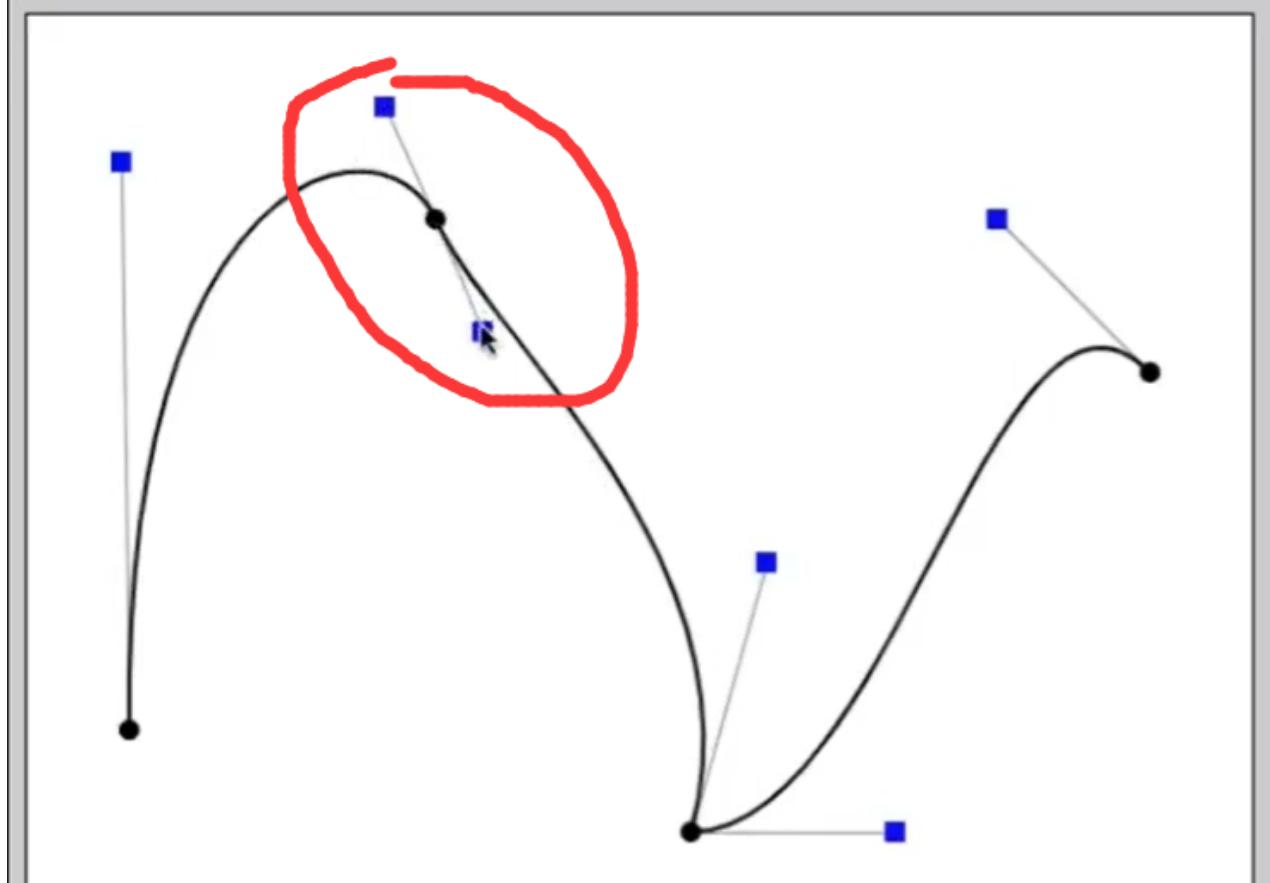
因此人们往往会每四个控制点画一个贝赛尔曲线

如何保证曲线是光滑的呢？

当某一个点即为前一个曲线的终点，也为下个曲线的起点，形成两个切线要共线且方向正好相反且大小也要一样（导数连续）

Lock Control Point Pairs

Hide Controls



c_0 连续, $a_n = b_0$, a为前一段的点, b为后一段的点

c_1 连续, $a_n = b_0 = \frac{1}{2}(a_{n-1} + b_1)$

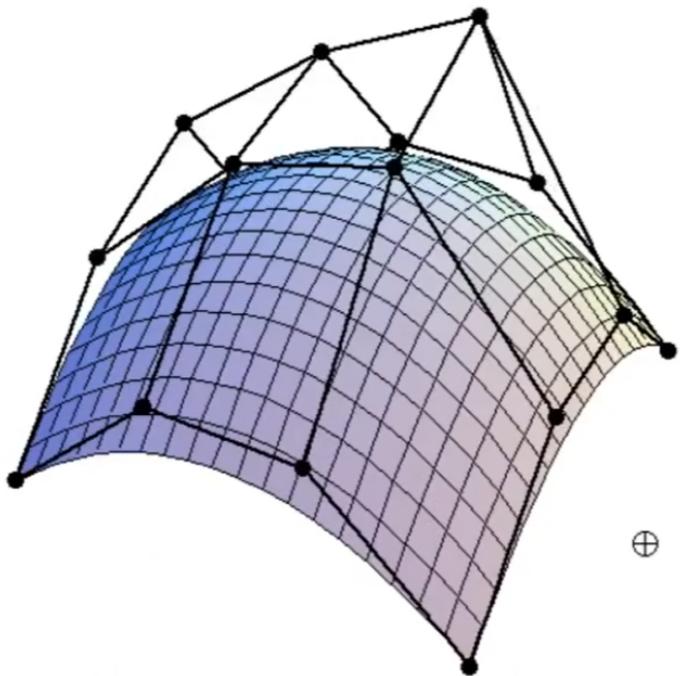
贝塞尔曲面

有点像二次插值

先找一些点, 画成一些曲线, 再用曲线上各自的点, 在做一系列的贝塞尔曲线, 得到曲面

Bicubic Bézier Surface Patch

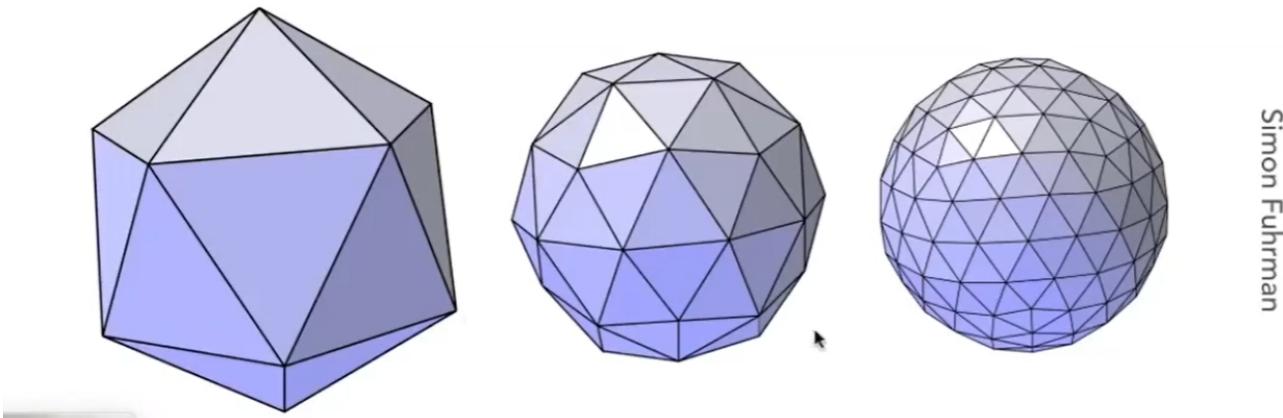
按 ESC 即可退出全屏模式



Bezier surface and 4×4 array of control points

曲面细分mesh subdivision

使用更多的三角形，使原来的形体有所变化，模型变得更加光滑



Loop Subdivision (Loop是人名，不是循环) 只能用于三角形面

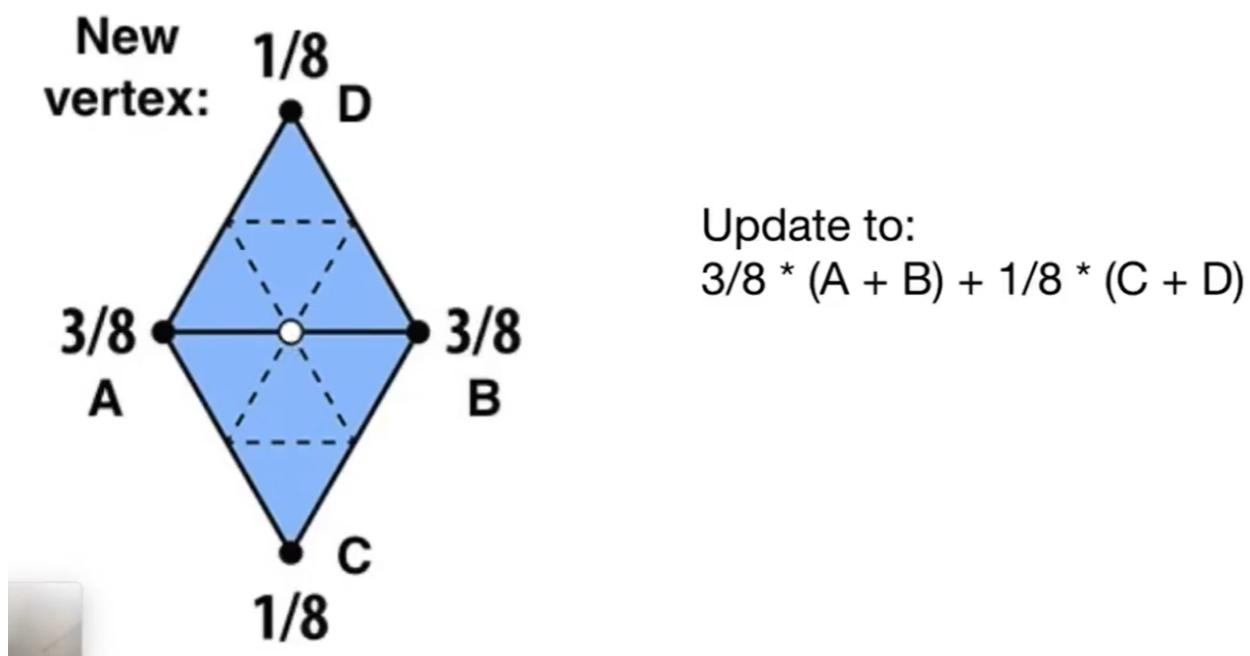
步骤

1 把一个三角形分成四个三角形

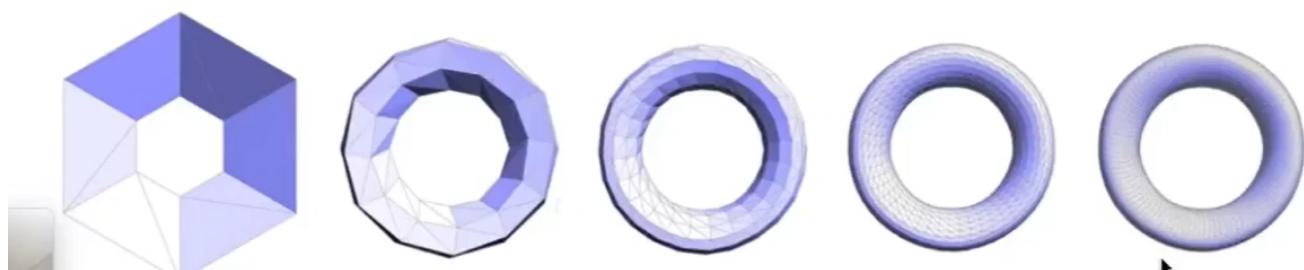
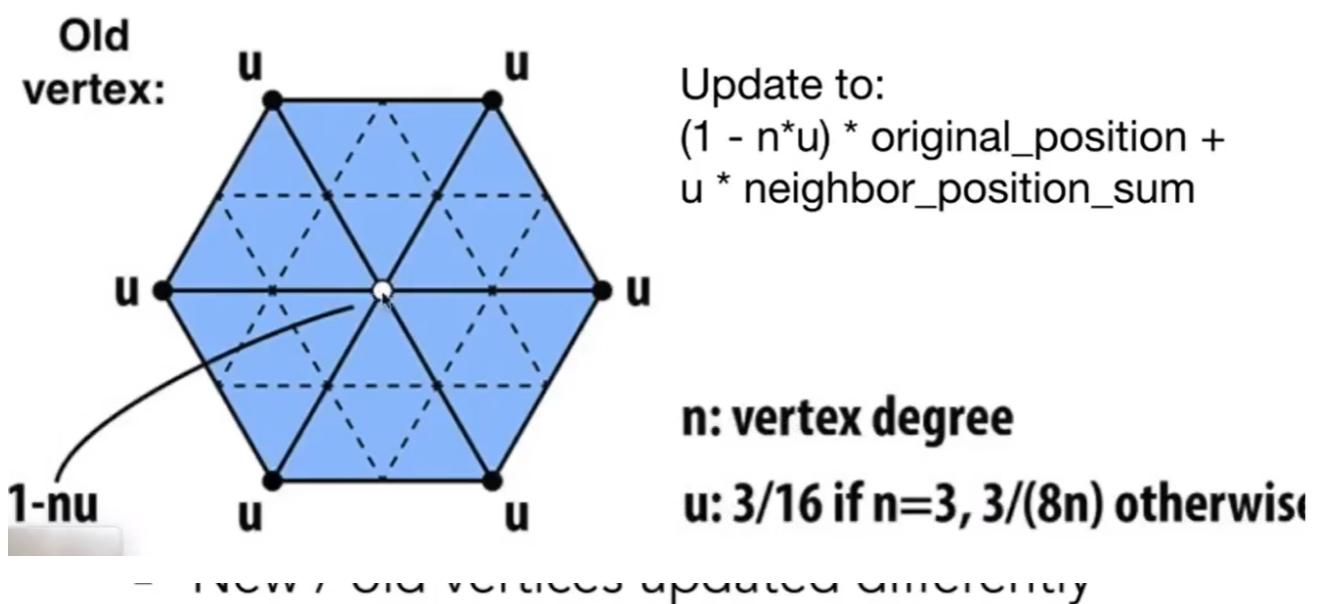
2 调整三角形位置，先区分新的顶点和旧的顶点，再根据权重来调整点的位置，新旧不一样

举例

对于新的顶点，往往是在两个三角形共享的边上的，比如图中的白点，做如下更新



对于旧的顶点， n 为顶点的度，一部分保留原有位置信息，再加上周围的点

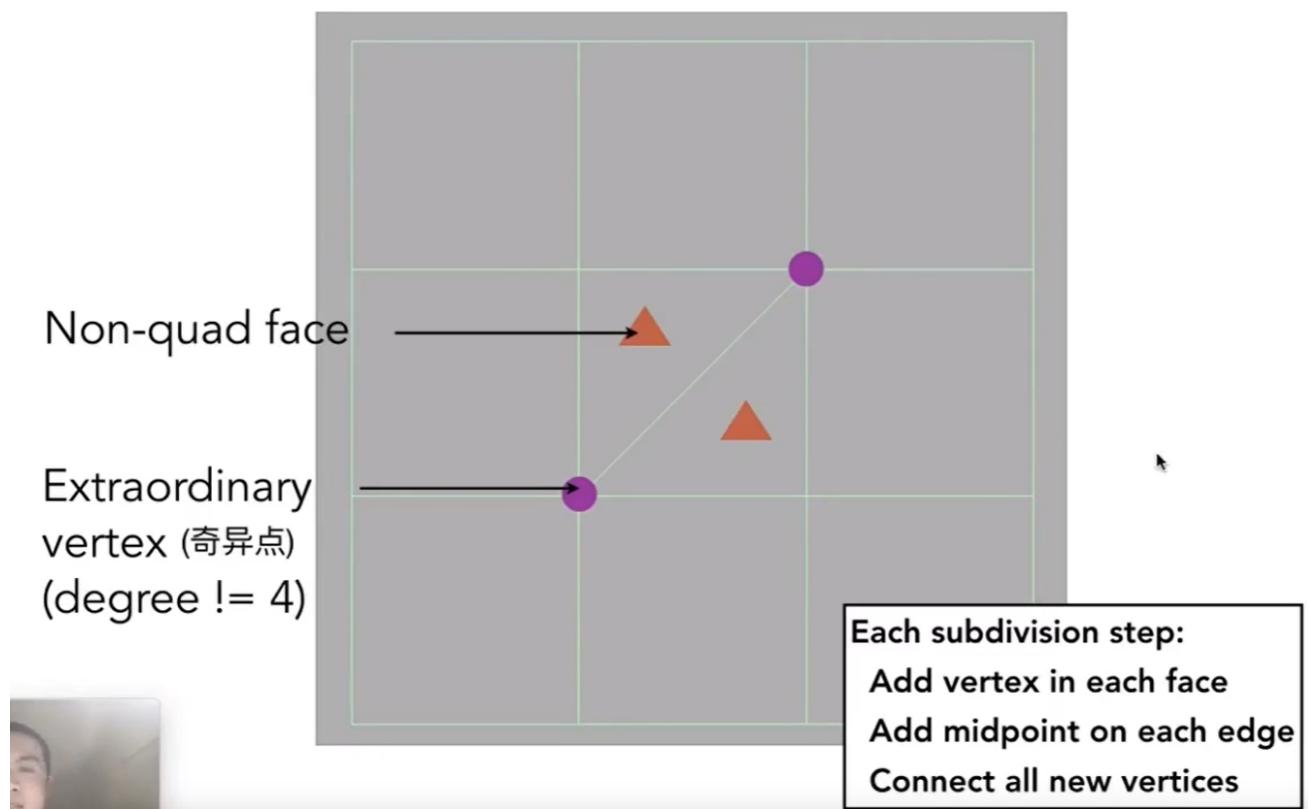


Catmull-Clark Subdivision用于各个面

定义一些概念

1 noun-quad face 非四边形面

2 extraordinary vertex 奇异点, 读不为4的点



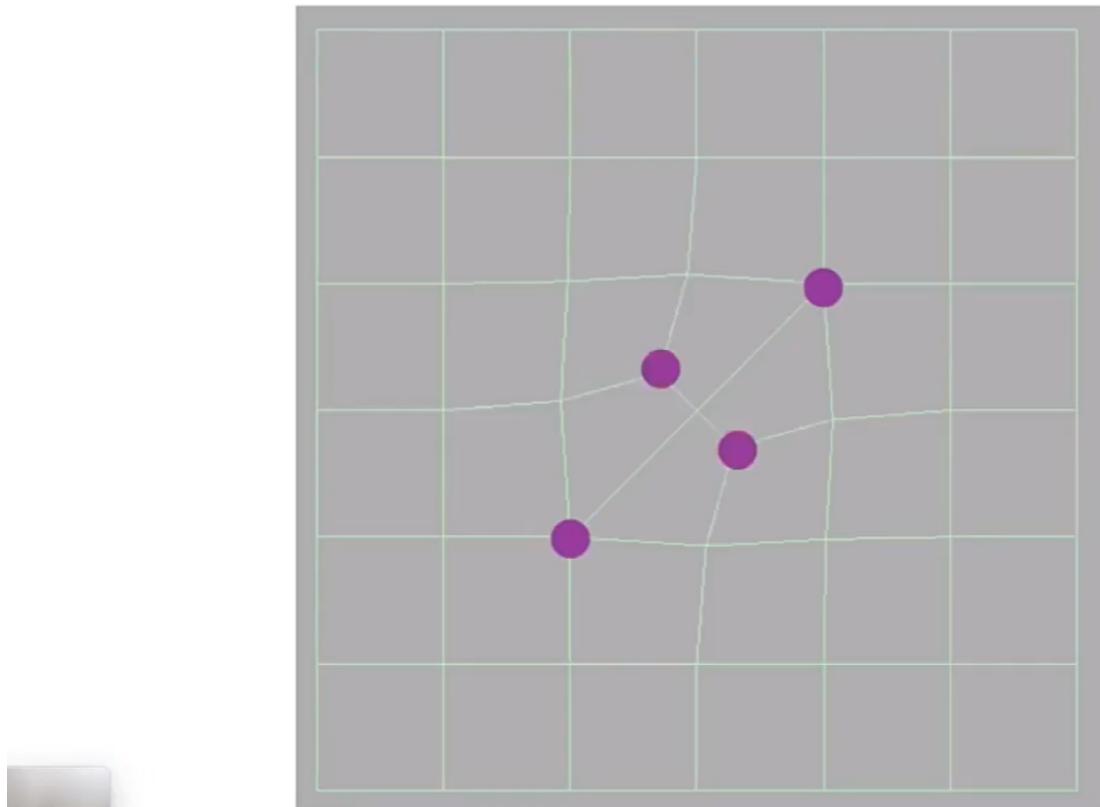
步骤

1 每个面中取一点（可以是重心，也可以是其他的）

2 每条边都取中点

3 把新的点都连起来（一个面的内部的连起来）

Catmull-Clark Subdivision (General Mesh)



可以发现一次细分后

奇异点增加了2（即原来非四边形个数）

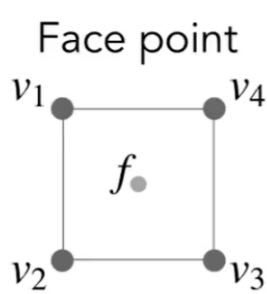
度数上，原本的奇异点度数不变，新的点的度数为那个非四边形的边数

非四边形面个数为0，都消失了

相当于原来的非四边形都转换成奇异点了，在做一次细分，奇异点数不会增加

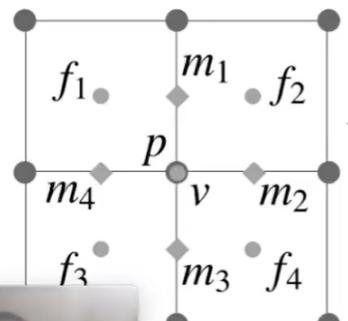
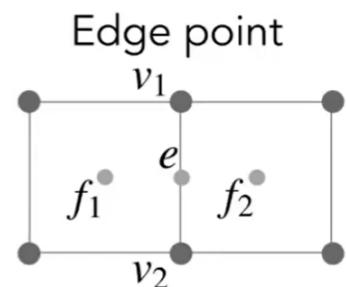
4 把点分成三类，（1）新的点中，在面中间的点（2）新的点中，在边中间的点（3）旧的点，分别更新

FYI: Catmull-Clark Vertex Update Rules (Quad Mesh)



$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$



$$v = \frac{f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p}{16}$$

m midpoint of edge
 p old "vertex point"



20

Lingqi Yan, UC Santa Barbara

曲面简化 mesh simplification

为了提升性能，减少三角形数量，简化模型



方法 边坍缩 edge collapsing

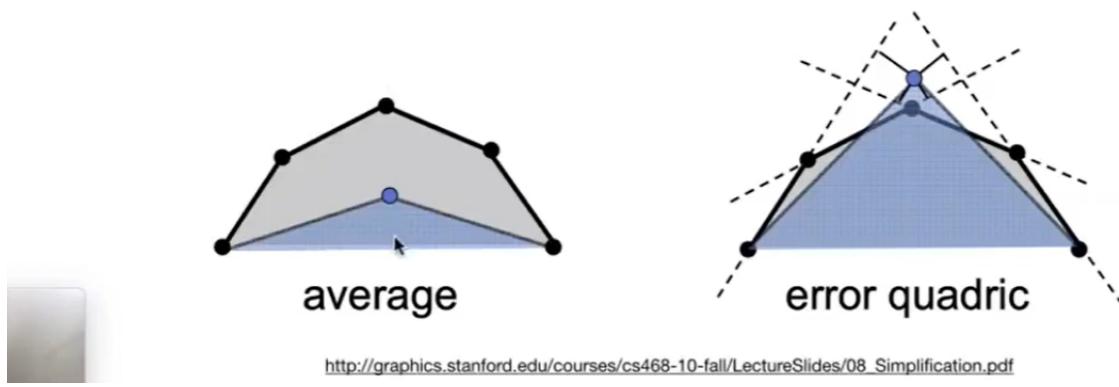
二次误差度量 Quadric Error Metrics

如何让下图的蓝色三角形可以描述灰色五边形的轮廓？

假如第三个点取五个点或者上面三个点的平均，会显得有些扁

二次误差就是找一点，这个点和原来的五个点的距离的平方和最小

- Quadric error: new vertex should minimize its **sum of square distance** (L2 distance) to previously related triangle planes!



步骤

贪心算法

1 对模型中所有的边，都假设如果坍缩这个边，计算会带来多少的误差

2 从小到大排序

3 从小的开始一次坍缩

问题

1 坍缩一条边，会影响其他的边，其他的边又得再算一次

所以我们需要有一个数据结构，可以得到最小值，之后还要更新一些边，所以需要堆

第八讲 光线追踪

应用

离线的画面渲染，比如动画片

定义

光线：

1 沿直线传播（只是假设）

2 光和光之间不会碰撞（只是假设）

3 光是从光源出发，最后到达人眼（可逆的，光追事实上也是这么做的）

光栅化方法形成阴影shadow mapping (不是光追)

解释

假如摄像机，可以看到这点，但从光源处看不到这个点，那么这个点在阴影处

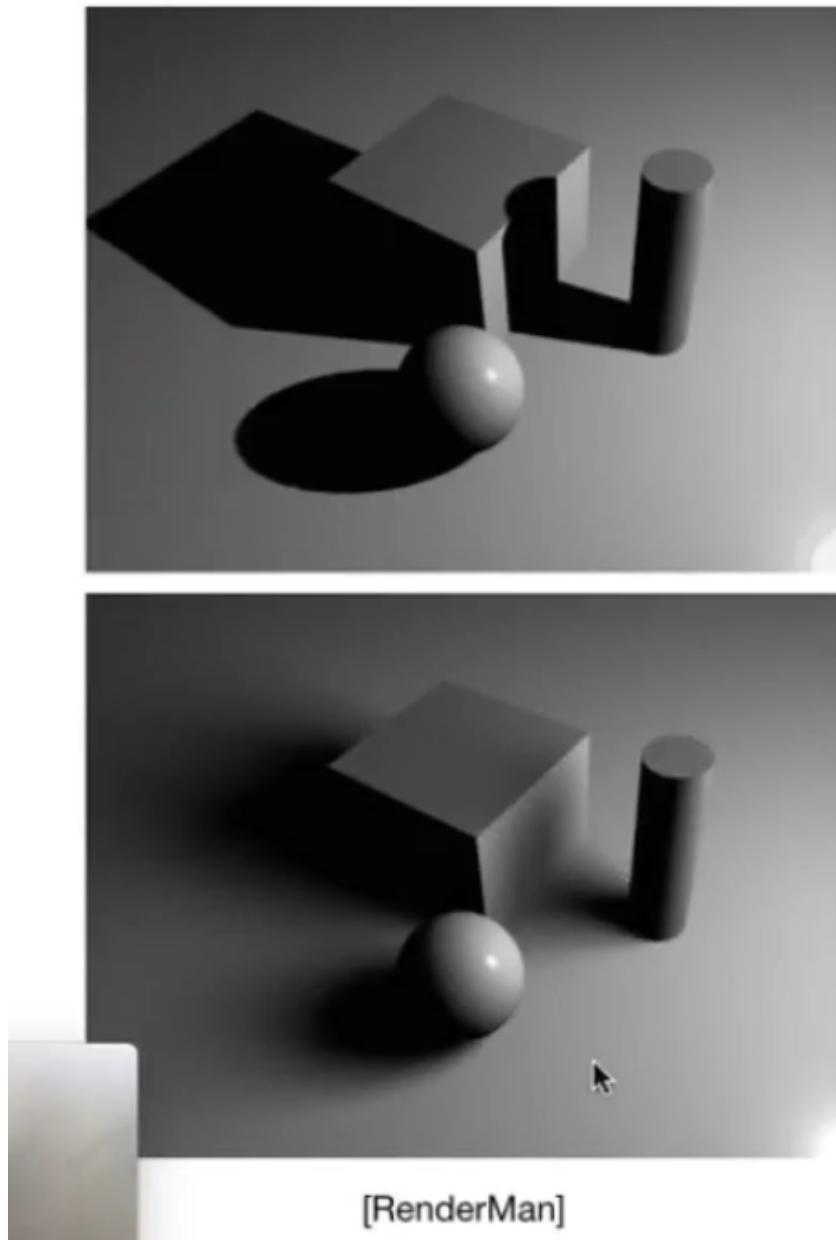
操作

- 1 从光源处看场景，对场景做投影，记录看到的点的深度
- 2 从相机处看场景，计算这个点在光源处看时的位置，和实际到光源的深度
- 3 一致的话，就可以从光源看到，深度大的话就看不到，被物体挡住了

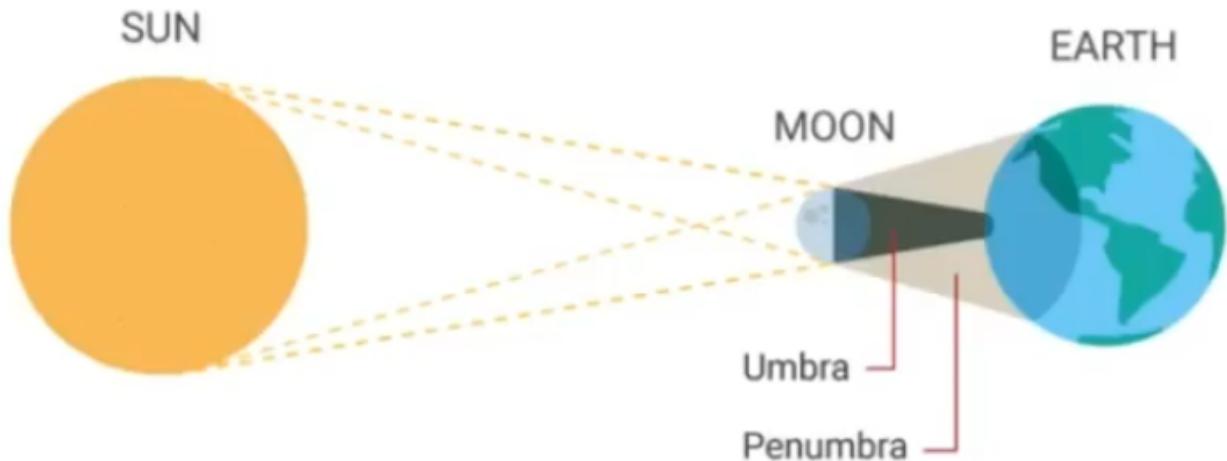
问题

- 1 浮点数计算相等很困难
- 2 阴影信息走样
- 3 当光线反射不止一次时很麻烦
- 4 磨砂表面的反射
- 5 理论上只能做硬阴影（边缘非常锐利）

- Hard shadows vs. soft shadows



光源有一定大小会产生软阴影，图中的penumbra（半影就是一个例子）



....

应用

- 1 早期动画
- 2 很多3d游戏（塞尔达、马里奥奥德赛）
- 3 实时画面运用的多

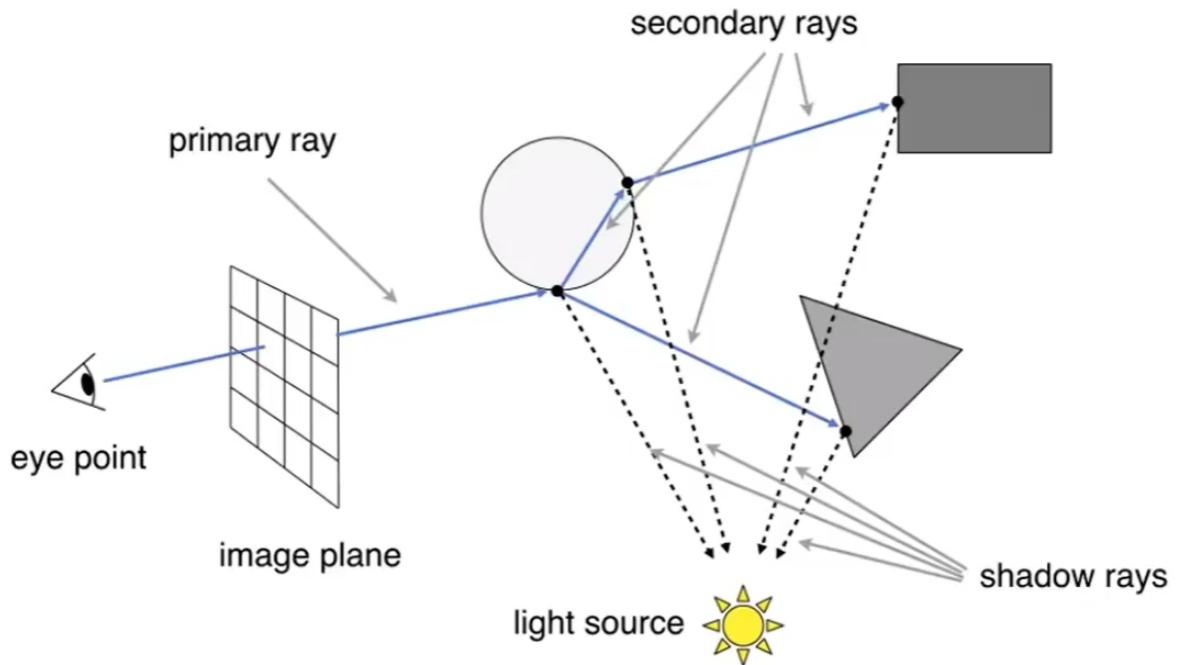
Whitted-Style Ray Tracing (递归算法)

方法

模拟光线弹射

- 1 设定primary ray，即人眼出去的第一道光线
- 2 将碰到的点和光源相连
- 3 再将primary折射、反射，和别的物体交的点和光源相连
- 4 判断这些点是否可见（和光源中间是否有遮挡）
- 5 计算光线传播中的能量损失，在第一个点处加权平均可以得到这个点的光照情况，就可以着色了

Recursive Ray Tracing



实际操作

光线： 定义好起点的射线，由一个点 o 和方向向量 d 来表示

光线上任意一点： $r(t) = o + td, -\infty \leq t < \infty$

1 求像素对应的场景坐标

要缩放这些坐标，把他们还原到光栅化之前的位置

2 求交点

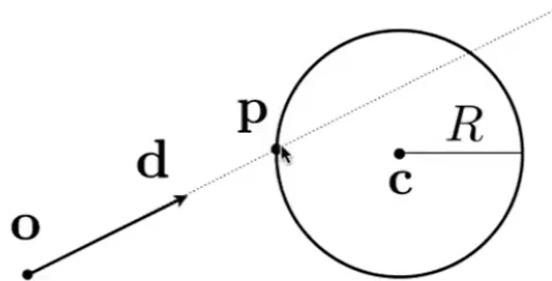
和球（点即在球上，又在线上）

Ray: $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}, 0 \leq t < \infty$

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

What is an intersection?

The intersection \mathbf{p} must satisfy both ray equation and sphere equation



Solve for intersection:

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

解满足 $t > 0$, 且等式有解, 且相交或相切

和一般隐式表示的物体求交

解方程即可, 可以用来判断点是否在物体内, 假如点在物体内, 可以得到奇数个交点, 点在物体外, 会得到偶数个交点

Ray: $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}, 0 \leq t < \infty$

General implicit surface: $\mathbf{p} : f(\mathbf{p}) = 0$

Substitute ray equation: $f(\mathbf{o} + t \mathbf{d}) = 0$

和三角形求交

对于每一个三角形来说:

方法一

可以求三角形所在平面和光线的交点, 再判断交点在不在三角形内

首先要定义这个平面, 需要一个法线和平面上的一个点

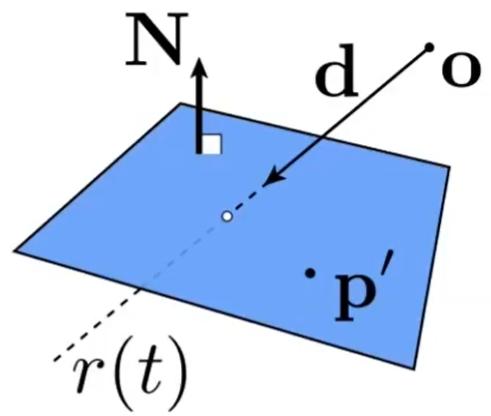
Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}, \quad 0 \leq t < \infty$$

Plane equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$$

Solve for intersection



Set $\mathbf{p} = \mathbf{r}(t)$ and solve for t

$$(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = (\mathbf{o} + t \mathbf{d} - \mathbf{p}') \cdot \mathbf{N} = 0$$

$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$

Check: $0 \leq t < \infty$



方法二Moller Trumbore算法

假如光线在三角形内有交点，则这个点一定可以写成重心坐标形式，解出来即可

$b_1 \geq 0, b_2 \geq 0, t \geq 0, b_1 + b_2 \leq 1$

则满足条件

Möller Trumbore Algorithm

A faster approach, giving barycentric coordinate directly

Derivation in the discussion section!

$$\vec{O} + t\vec{D} = (1 - b_1 - b_2)\vec{P}_0 + b_1\vec{P}_1 + b_2\vec{P}_2$$

Where:

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{S}_1 \cdot \vec{E}_1} \begin{bmatrix} \vec{S}_2 \cdot \vec{E}_2 \\ \vec{S}_1 \cdot \vec{S} \\ \vec{S}_2 \cdot \vec{D} \end{bmatrix}$$
$$\vec{E}_1 = \vec{P}_1 - \vec{P}_0$$
$$\vec{E}_2 = \vec{P}_2 - \vec{P}_0$$
$$\vec{S} = \vec{O} - \vec{P}_0$$

Cost = (1 div, 27 mul, 17 add)

$$\vec{S}_1 = \vec{D} \times \vec{E}_2$$

$$\vec{S}_2 = \vec{S} \times \vec{E}_1$$

Recall: How to determine if the “intersection” is inside the triangle?

Hint:
(1-b1-b2), b1, b2 are barycentric coordinates!

问题

假如用光线和物体上每个三角形求交，可以得到结果，但很慢

加速方法

包围盒 (把物体完全包围的几何体)

原理

假如光线不能进入包围盒，则不可能到达里面的物体上

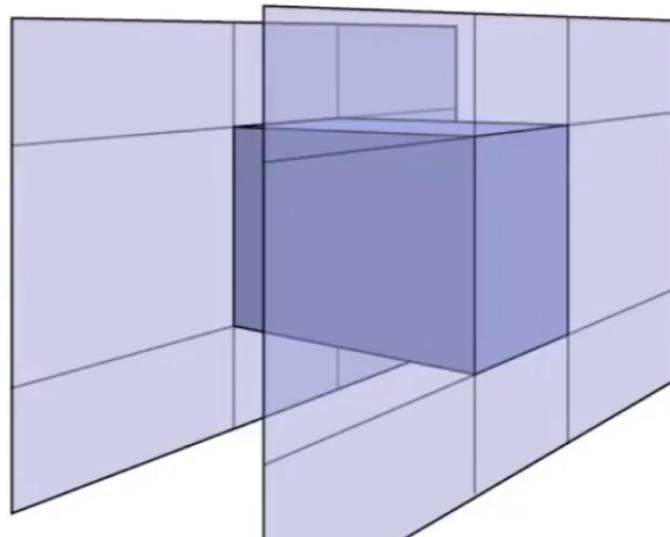
操作

通常使用轴对齐包围盒AABB

Specifically:

We often use an
Axis-Aligned
Bounding Box (AABB)
(轴对齐包围盒)

i.e. any side of the BB
is along either x, y, or z
axis



定义一个包围盒

用两个点pMin,pMax

如何判定和包围合的是否交：

先考虑二维：

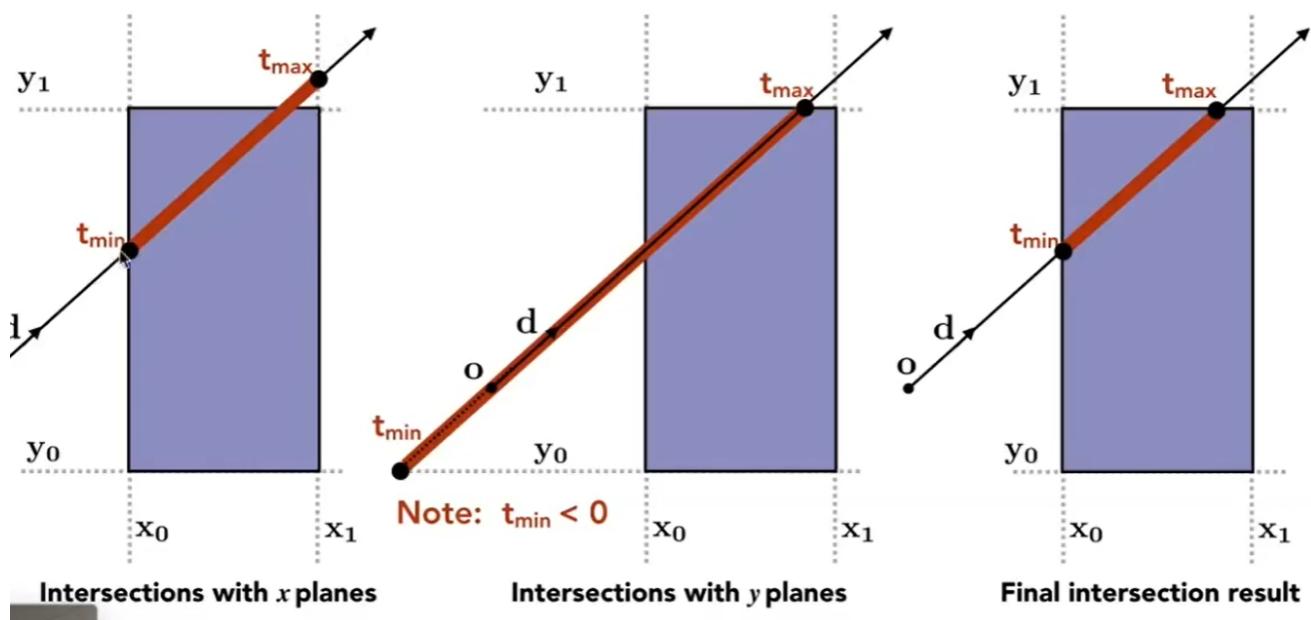
求出和 $x = x_0, x_1$ 的交点时的t，分别为 t_{min}, t_{max}

求出 $y = y_0, y_1$ 的交点时的t，分别为 t_{min}, t_{max}

假如两个区间有重合即 $t_{enter} = \max\{t_{min}\}, t_{exit} = \min\{t_{max}\}, t_{enter} < t_{exit}$ ，则光线经过这个正方形

假如 $t_{exit} < 0$ 说明盒子在光源后面，则不存在交点

假如 $t_{exit} \geq 0, t_{enter} < 0$ ，光源在盒子里面，肯定有交点

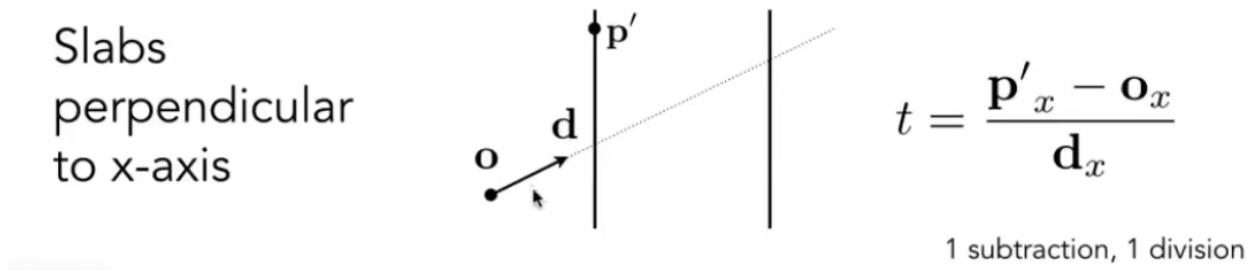


三维的情况和二维类似，只是求和对面的交点时的t，再求交

假如两个区间有重合即 $t_{enter} = \max\{t_{min}\}, t_{exit} = \min\{t_{max}\}, t_{enter} < t_{exit}$ 且 $t_{exit} \geq 0$ ，则光线经过这个正方形

这时求交就不需要严格地求和平面的交点了，可以找一个面，只关心x轴方向上的分量

比如求 $t_{minx} = (pMin.x - o.x)/d.x$



均匀网格 uniform grids

由于格子求交，比物体求交快

- (1) 把整个场景分成一个个的小格子
- (2) 判断光线是否和物体所在的盒子相交
- (3) 再去判断是否和物体是否有交点

在均匀分布的场景中比较适用

但在复杂场景中就不适用了

空间划分 spatial partitions (KD-Tree)

- (1) 八叉树（用的不多）

把空间切两刀，分成八块，给定一个标准（比如这个子空间里已经不和任何物体交了），让某些子空间继续划分，某些停止划分

- (2) **KD-Tree**（保留了二叉树的性质，不像八叉树纬度高的时候分叉太多）

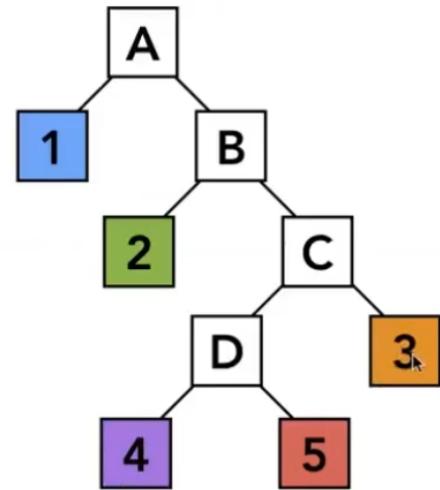
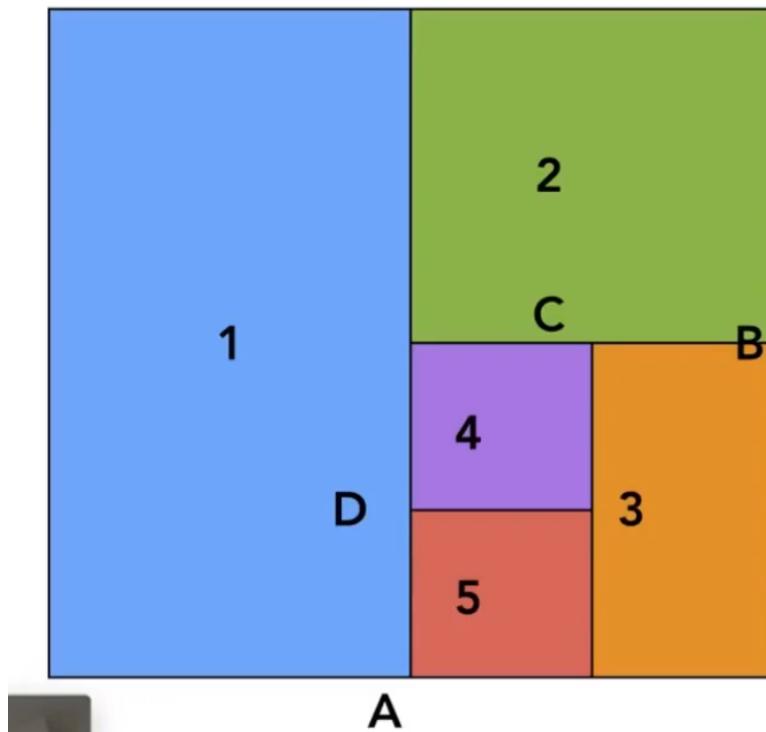
步骤

每次只砍一刀

二维情况，第一层水平划分，下一层垂直划分，交替进行

三维情况，第一层沿x轴划分，下一层沿y轴划分，下一层沿z轴划分，交替进行

KD-Tree Pre-Processing



Note: also subdivide
nodes 1 and 2, etc.

存储的数据

中间节点

1 沿哪个轴切的

2 切的位置

3 子节点的指针

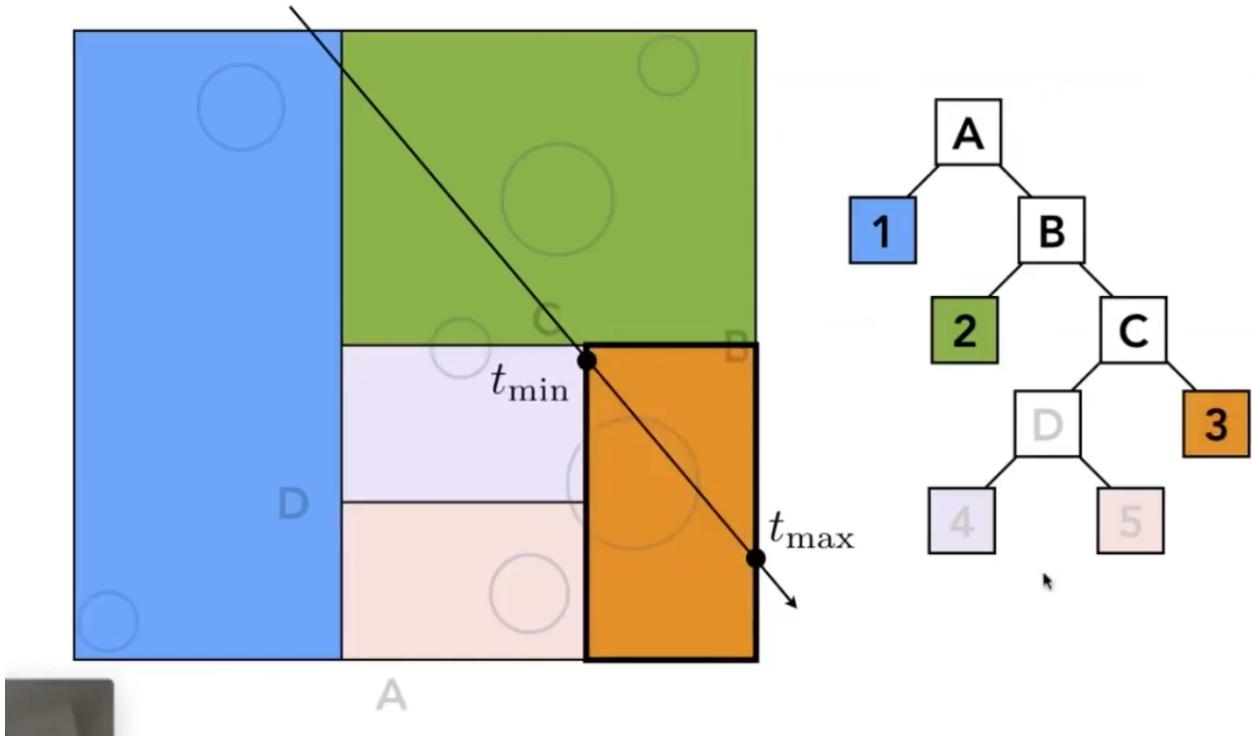
4 中间节点不存储物体

叶子节点

1 一些物体

具体步骤

分别判断和这些叶子节点的盒子是否有交点，有的话在和内部的物体求交



难点

1 怎么判断划分出的格子，和实际物体相交，因此该方法最近几年用的不多

2 一个物体可能在多个格子里，很麻烦

(3) BSP-Tree

每次选择一个方向砍一刀，与kd-tree不同于kd-tree总沿水平/垂直的方向砍一刀

但是计算太复杂，维度高时太复杂

物体划分 object partitions(BVH)

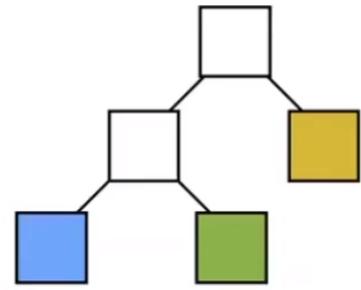
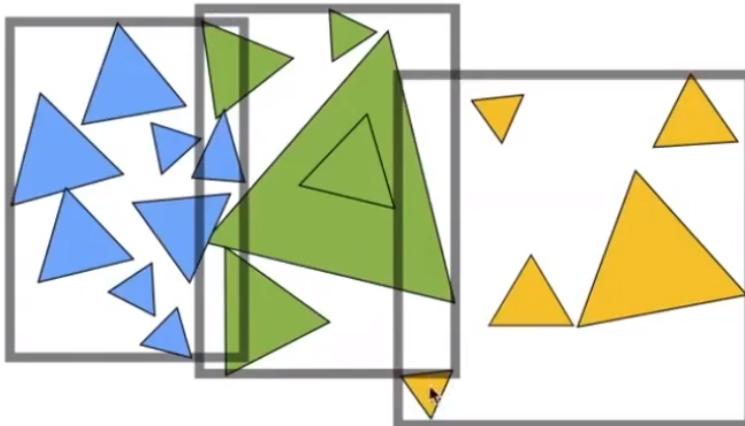
Bounding Volume Hierarchy (BVH) 使用广泛

基本方法

1 用盒子把场景包围

2 把三角形分成两部分，再分别求包围盒，然后继续划分，直到达到某个标准，比如每个盒子里只有5个三角形（保证一个物体只在一个盒子里），当然也要尽可能让包围盒的重叠少一些

3 把实际物体记录在叶子节点里



实际操作

1 如何划分节点？

- (1) 选择最长的轴来切，比如x的跨度比较大，那就竖直x轴切分
- (2) 取中间的物体来划分（快速选择算法O(n)，找无序数中第*i*大的数），树更加平衡

2 递归找出距离眼睛最小的交点，当父节点已经不相交就可以剪枝了

数据结构

中间节点

- (1) 包围盒 (2) 子节点指针

叶子节点

- (1) 包围盒 (2) 物体列表

辐射度量学radiometry

学习原因

1 比如光的强度（light intensity）为10，10代表什么呢？

2 whitted style 光线追踪不真实

3 为了精确描述光照

概念

Radiant energy (辐射能) : 电磁辐射的能量，用符号Q[J=Joule]表示

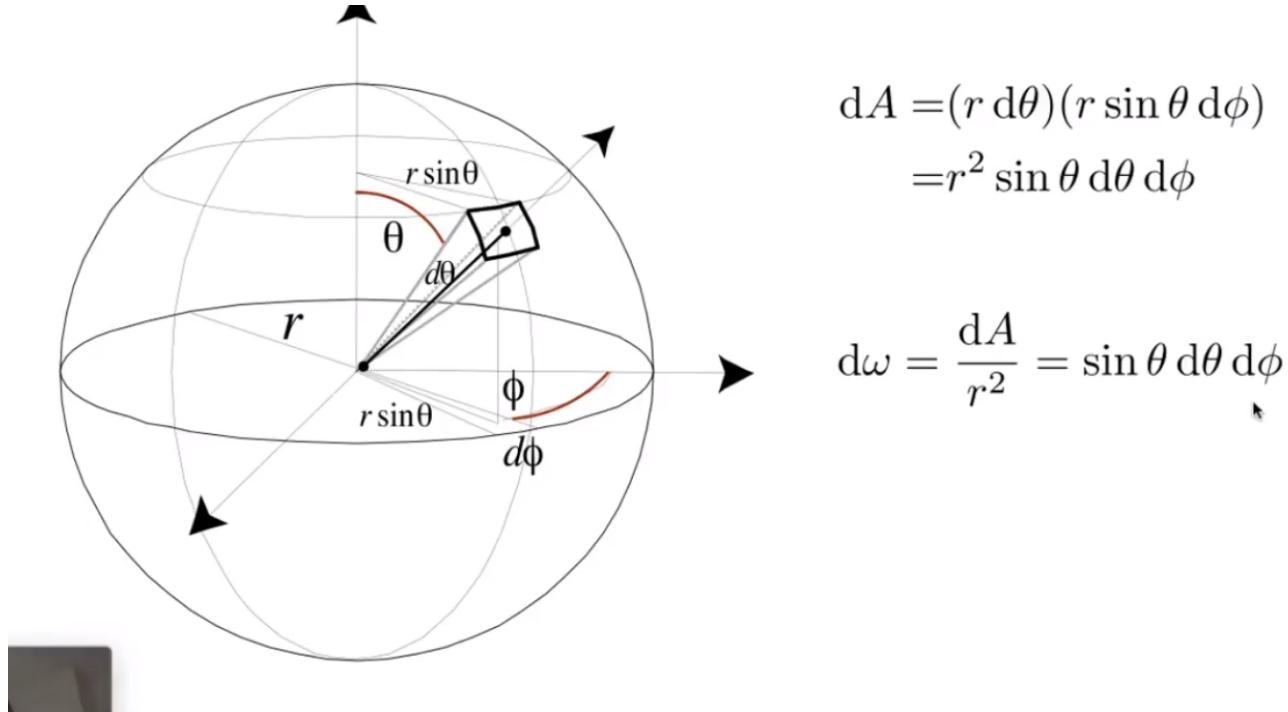
Radiant flux (辐射通量) : 单位时间的能量，功率 $\Phi = \frac{dQ}{dt}$ [W = Watt] [lm = lumen]* 瓦特/流明

Radiant Intensity (辐射强度) : 单位立体角的能量, $I(\omega) = \frac{d\Phi}{d\omega} [\frac{W}{sr}] [\frac{lm}{sr}] = cd$

solid angles (立体角) : 平面上的角度为 $\theta = \frac{l}{r} = \frac{\text{弧长}}{\text{半径}}$

立体角为 $\Omega = \frac{A}{r^2} = \frac{\text{球上一块的面积}}{\text{半径平方}}$, 整个球立体角为 4π

differential solid angles (单位(微分)立体角) : $d\omega = \frac{dA}{r^2} = \sin\theta d\theta d\phi$



Irradiance (辐照度) : 单位面积的能量(面和光线垂直, 即投影面积上),

$$E(X) = \frac{d\Phi(X)}{dA} [\frac{W}{m^2}] [\frac{lm}{m^2}] = lux$$

radiance (光谱辐射) : 单位立体角并在单位面积内的能量 $L(p, \omega) = \frac{d^2\Phi(p, \omega)}{d\omega dA \cos\theta}$

当理解为单位立体角内的Irradiance, 方便理解某个面接受的能量

Incident Radiance

Incident radiance is the irradiance per unit solid angle arriving at the surface.



$$L(p, \omega) = \frac{dE(p)}{d\omega \cos \theta}$$

i.e. it is the light arriving at the surface along a given ray (point on surface and incident direction).

当理解为单位面积内的Intensity，方便理解发出去的能量

Exiting Radiance

Exiting surface radiance is the intensity per unit projected area leaving the surface.



$$L(p, \omega) = \frac{dI(p, \omega)}{dA \cos \theta}$$

e.g. for an area light it is the light emitted along a given ray (point on surface and exit direction).

Irradiance可以理解为从dA面积上接受的能量

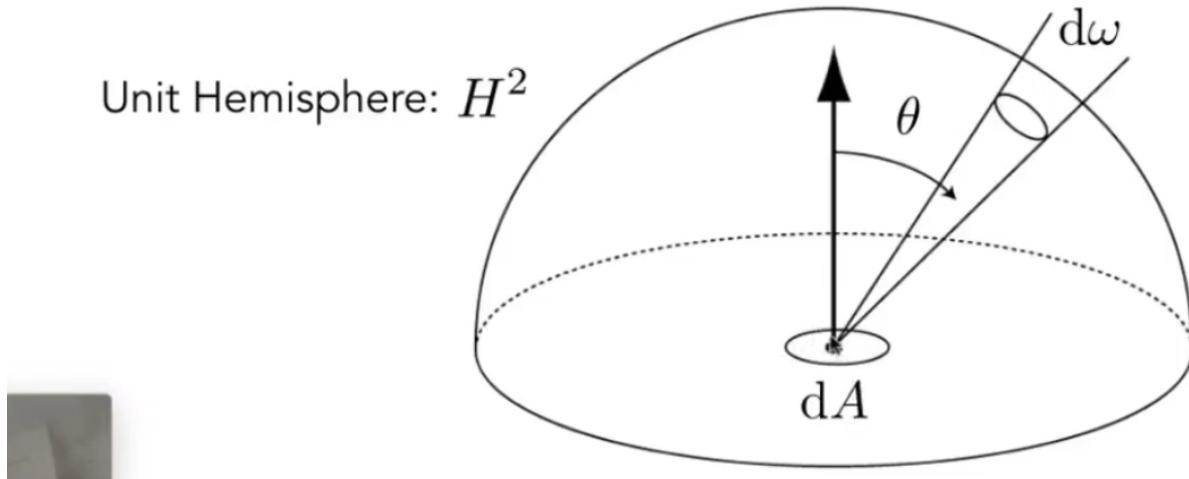
Radiance可以理解为面积dA从方向dω接受的能量

Irradiance: total power received by area dA

Radiance: power received by area dA from "direction" $d\omega$

$$dE(p, \omega) = L_i(p, \omega) \cos \theta d\omega$$

$$E(p) = \int_{H^2} L_i(p, \omega) \cos \theta d\omega$$



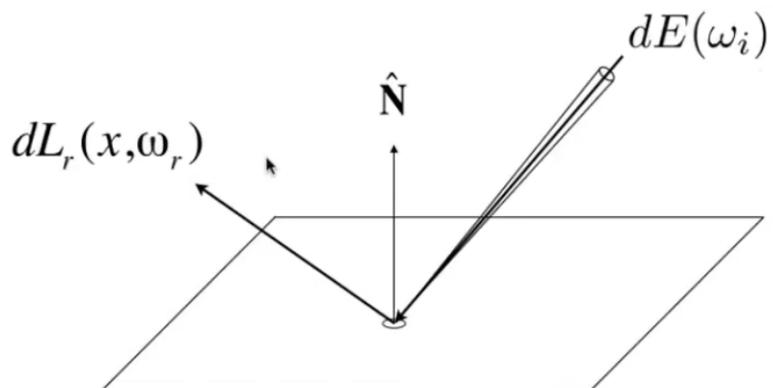
BRDF (bidirectional reflectance distribution function 双向反射传播函数)

如何理解反射

某一点接受各个方向来的Irradiance（能量），再把它发射到四面八方（分配到各个立体角上）

Radiance from direction ω_i turns into the power E that dA receives

Then power E will become the radiance to any other direction ω_o .



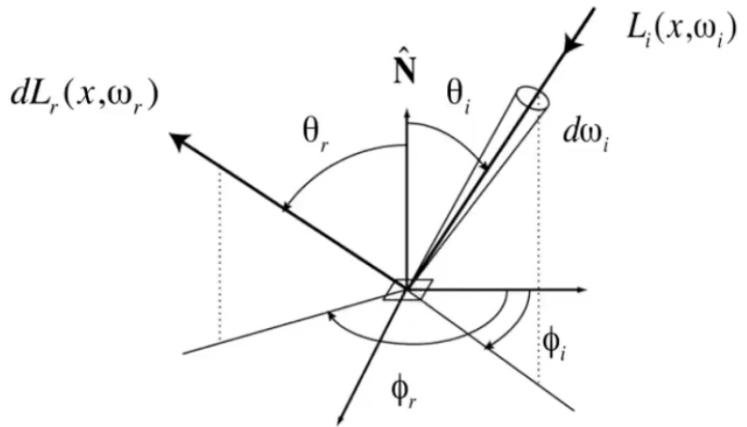
Differential irradiance incoming: $dE(\omega_i) = L(\omega_i) \cos \theta_i d\omega_i$

Differential radiance exiting (due to $dE(\omega_i)$): $dL_r(\omega_r)$

功能

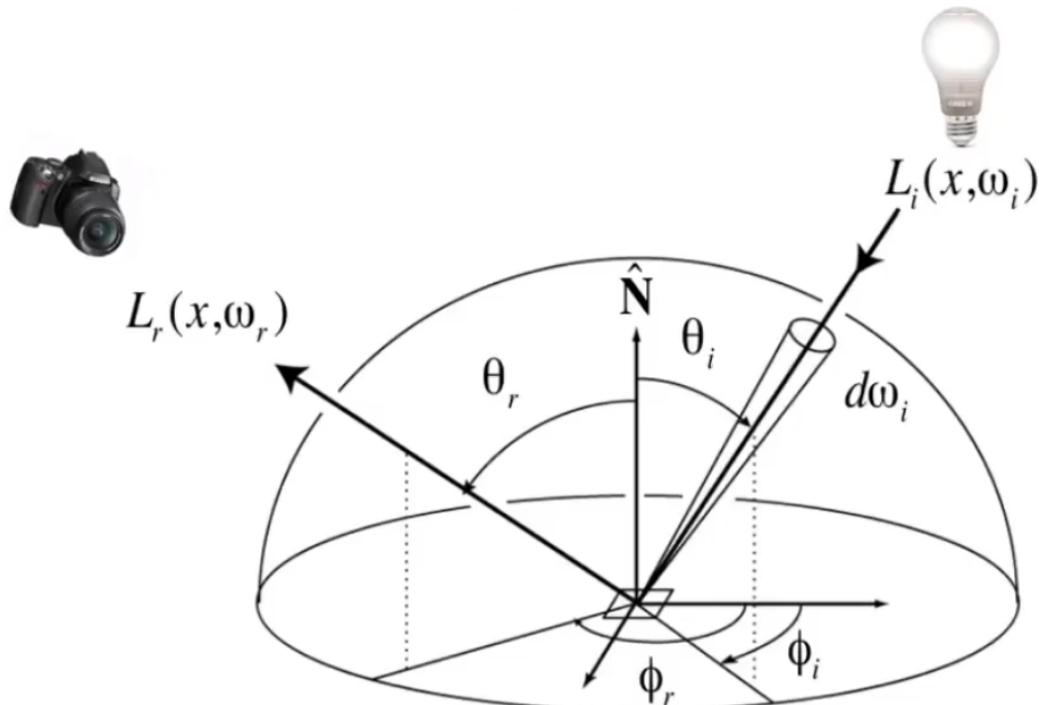
计算从dA发射到各个角度的能量，定义如何分配这些能量

The Bidirectional Reflectance Distribution Function (BRDF)
represents how much light is reflected into each outgoing direction ω_r
from each incoming direction



$$f_r(\omega_i \rightarrow \omega_r) = \frac{dL_r(\omega_r)}{dE_i(\omega_i)} = \frac{dL_r(\omega_r)}{L_i(\omega_i) \cos \theta_i d\omega_i} \left[\frac{1}{sr} \right]$$

把所有入射的方向都计算一次，加起来（积分）就可以得到结果



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

困难

不只是光源可以到达点p，其他物体反射的光也可以到达，需要递归求解，比较麻烦

渲染方程 rendering equation

假如物体本身会发光，那么BRDF需要加一个自身的光

假如有许多点光源，加起来就好，面光源积分即可

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

其他物体反射光，就把其他物体当成光源，递归定义

$$L_r(x, \omega_r) = L_e(x, \omega_r) + \int_{\Omega} L_r(x, -\omega_i) [f(x, \omega_i, \omega_r) \cos \theta_i] d\omega_i$$

Reflected Light (Output Image)	Emission	Reflected Light	BRDF	Cosine of Incident angle
UNKNOWN	KNOWN	UNKNOWN	KNOWN	KNOWN

Is a Fredholm Integral Equation of second kind
[extensively studied numerically] with canonical form

$$I(u) = e(u) + \int I(v) [K(u, v)] dv$$

Kernel of equation

简化形式

$$l(u) = e(u) + \int l(v) [K(u, v)] dv$$

Kernel of equation
Light Transport Operator

$$L = E + KL$$

把L分解，写成展开形式

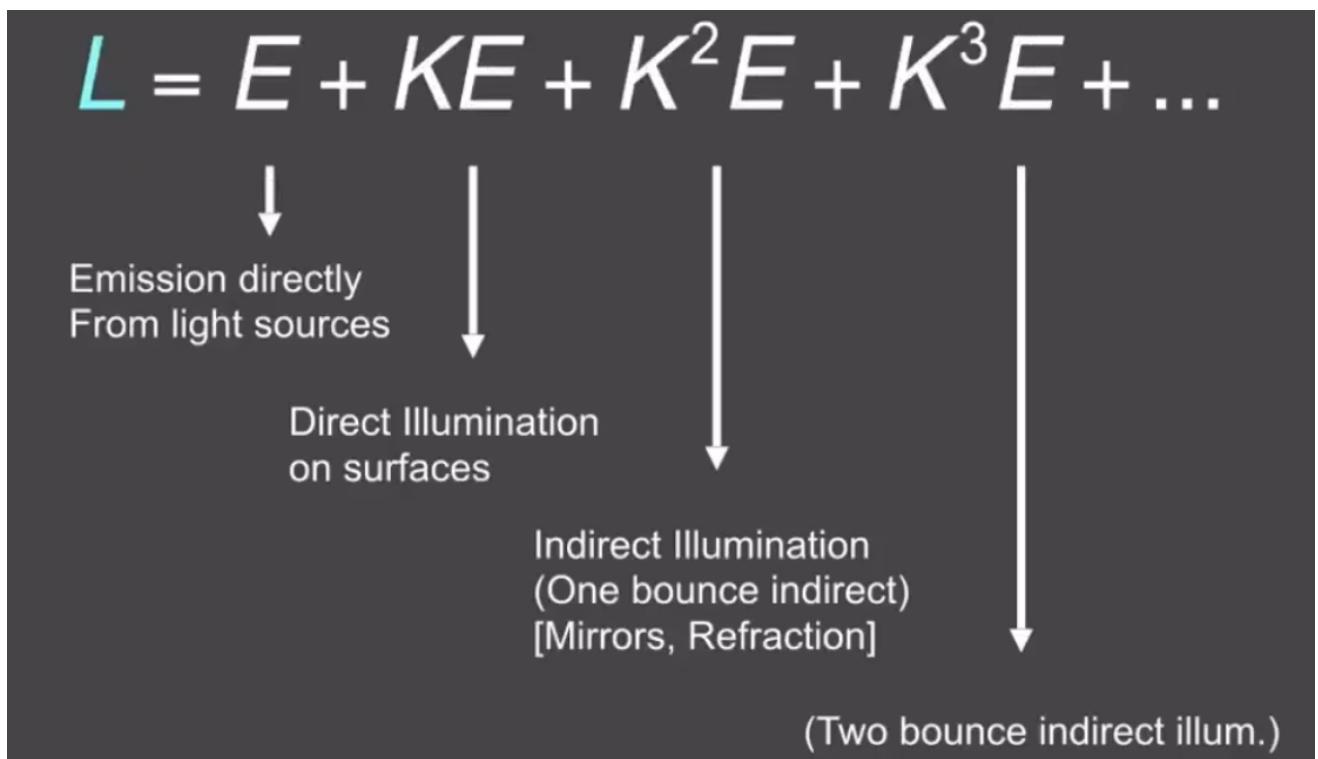
- Approximate set of all paths of light in scene

$$\begin{aligned} L &= E + KL \\ IL - KL &= E \\ (I - K)L &= E \\ L &= (I - K)^{-1}E \end{aligned}$$

Binomial Theorem

$$\begin{aligned} L &= (I + K + K^2 + K^3 + \dots)E \\ L &= E + KE + K^2E + K^3E + \dots \end{aligned}$$

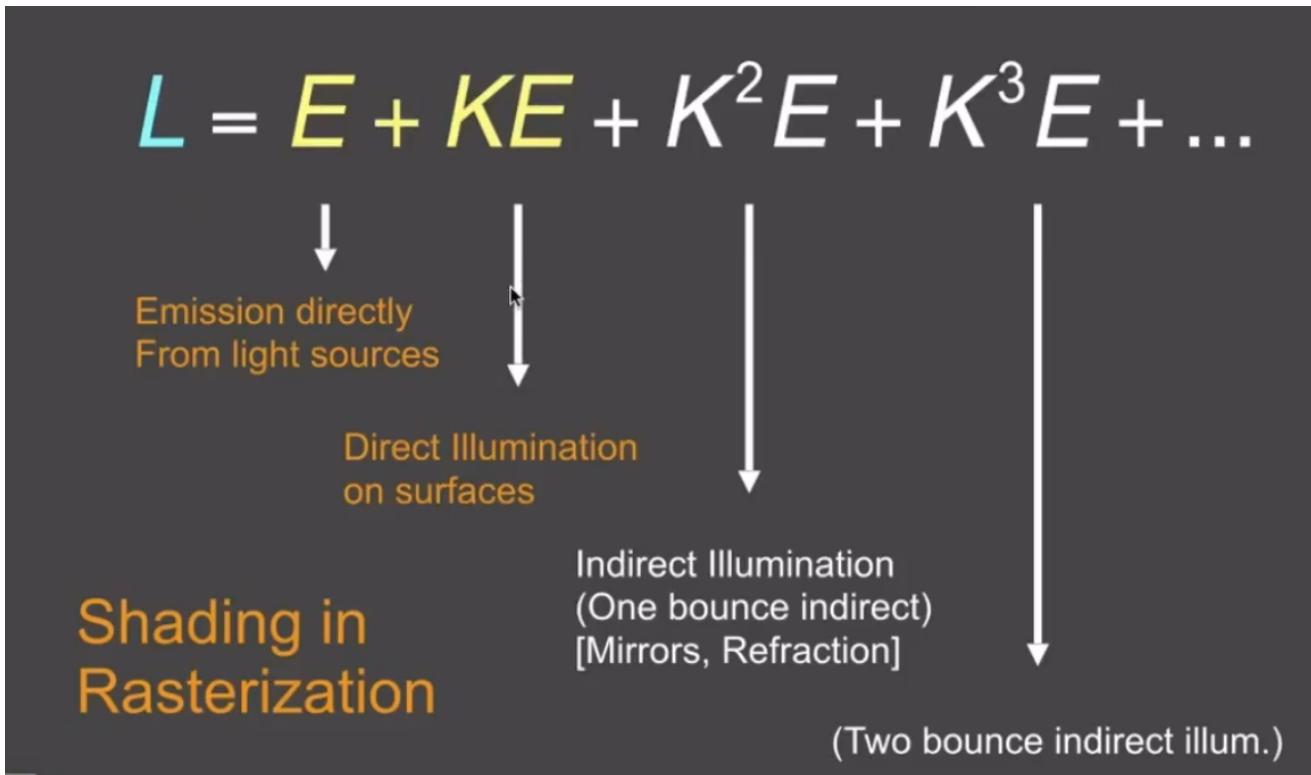
分解成自身的光，弹射一次的光（直接光照），弹射两次的光，弹射三次的光（间接光照）...



全局光照

光栅化只能告诉我们前两项，后面做起来比较麻烦，把后面也做了就是全局光照

最后会收敛到某一个亮度



蒙特卡洛路径追踪

概率论回顾

随机变量x

$$\text{期望 } E(X) = \sum_{i=0}^n x_i p_i = \int_{-\infty}^{+\infty} x f(x) dx$$

$$Y = h(x)$$

$$E(Y) = \int_{-\infty}^{+\infty} h(x) f(x) dx$$

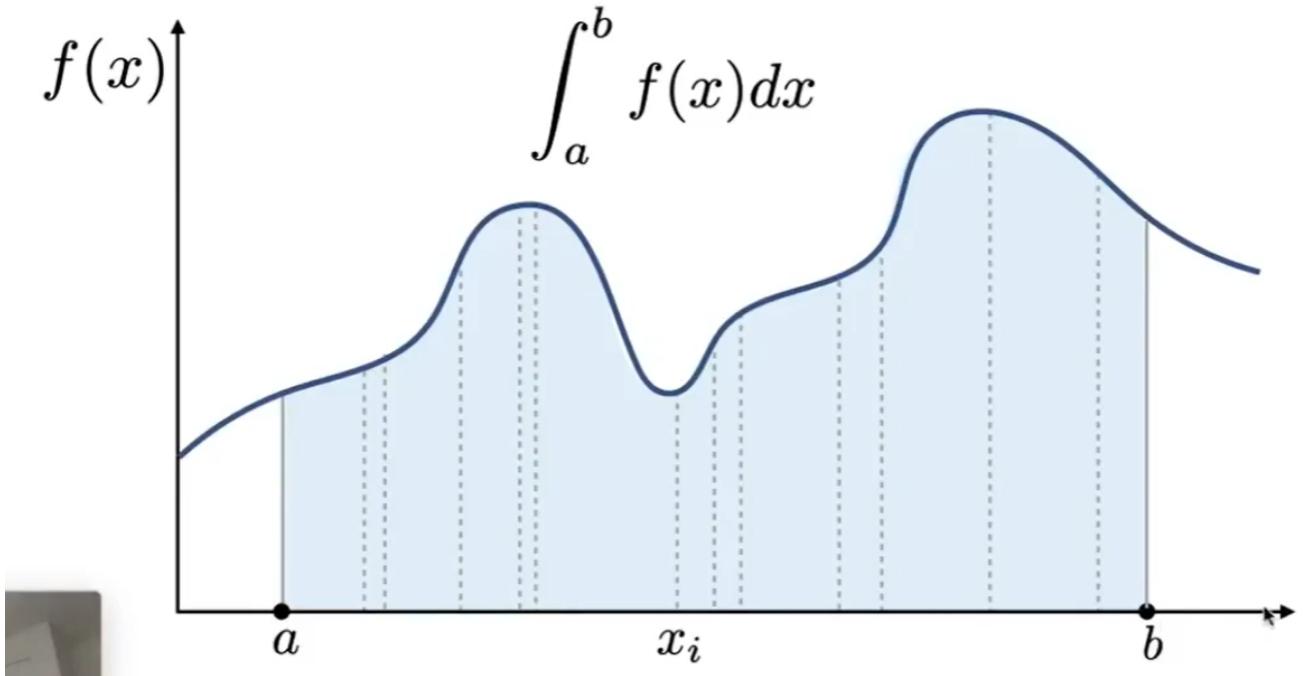
概率密度f

$$\text{概率分布 } F(x) = \int_{-\infty}^{+\infty} f(x) dx$$

蒙特卡洛积分

用处

计算一个复杂函数（不好求不定积分，不好积）的定积分



方法

在积分域内不断取 $f(x)$, 假设整个区域内为长度为 $b-a$, 高度为 $f(x)$ 的长方形, 重复多次取平均值

假如均匀采样, 即 X_i 服从均匀分布

Definite integral

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

Uniform random variable

$$X_i \sim p(x) = \frac{1}{b-a}$$

Basic Monte Carlo estimator

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

对于一般采样

$$\int f(x) dx = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)} \quad X_i \sim p(x)$$

要求

1 样本越多，偏差越小

2 积分域在X上

路径追踪 (对于每个点只考虑一根光线)

whitted style 的问题

(1) 光线到漫反射就停止，开始着色了

(2) 漫反射后的光线没有显示出来

使用渲染方程的困难点和解决办法

如何计算定积分的值--蒙特卡洛积分

只考虑直接光照，确定 $f(x)$ （积分号后的一串），确定 $p(x)$ （ $1/2\pi$ ，因为只考虑半个球面）

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

Monte Carlo integration: $\int_a^b f(x) dx \approx \frac{1}{N} \sum_{k=1}^N \frac{f(X_k)}{p(X_k)}$ $X_k \sim p(x)$

What's our "f(x)"? $L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i)$

What's our pdf? $p(\omega_i) = 1/2\pi$

(assume uniformly sampling the hemisphere)

$$\begin{aligned} L_o(p, \omega_o) &= \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i \\ &\approx \frac{1}{N} \sum_{i=1}^N \frac{L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i)}{p(\omega_i)} \end{aligned}$$

伪代码

```

shade(p, wo)
    Randomly choose N directions wi~pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
    Return Lo

```

加上别的点（间接光照）

```

shade(p, wo)
    Randomly choose N directions wi~pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
        Else If ray r hit an object at q
            Lo += (1 / N) * shade(q, -wi) * f_r * cosine
            / pdf(wi)
    Return Lo

```

但依然有问题

1 光线数量会爆炸，只有n=1不会爆炸。

这就是路径追踪，找到了视点和光源的一条路径，但噪声很大

```

shade(p, wo)

    Randomly choose ONE direction wi~pdf(w)
    Trace a ray r(p, wi)
    If ray r hit the light
        Return L_i * f_r * cosine / pdf(wi)
    Else If ray r hit an object at q
        Return shade(q, -wi) * f_r * cosine / pdf(wi)

```

从像素点找N个样本中的位置，做出N条光线，求出他们的光照，取平均，也是一种蒙特卡洛积分

```

ray_generation(camPos, pixel)

    Uniformly choose N sample positions within the pixel
    pixel_radiance = 0.0
    For each sample in the pixel
        Shoot a ray r(camPos, cam_to_sample)
        If ray r hit the scene at p
            pixel_radiance += 1 / N * shade(p, sample_to_cam)
    Return pixel_radiance

```

2 递归的停止条件没加

不能简单的设定一个弹射次数，能量会损失

解决方法：俄罗斯轮盘赌

设定一个概率P，让递归停止

假如在概率P内，将渲染结果放大为: L_o/p

否则结束，返回0

原因，这是一个0-1分布，期望为 $L_0/P * P + 0 = L_0$

伪代码

```

shade(p, wo)
    Manually specify a probability P_RR
    Randomly select ksi in a uniform dist. in [0, 1]
    If (ksi > P_RR) return 0.0;

    Randomly choose ONE direction wi-pdf(w)
    Trace a ray r(p, wi)
    If ray r hit the light
        Return L_i * f_r * cosine / pdf(wi) / P_RR
    Else If ray r hit an object at q
        Return shade(q, -wi) * f_r * cosine / pdf(wi) / P_RR

```

3 效率不高，假如每个像素取的样本小，得到的效果不好

原因：由于采样均匀，能不能遇到光源很靠运气，有很多光源浪费了

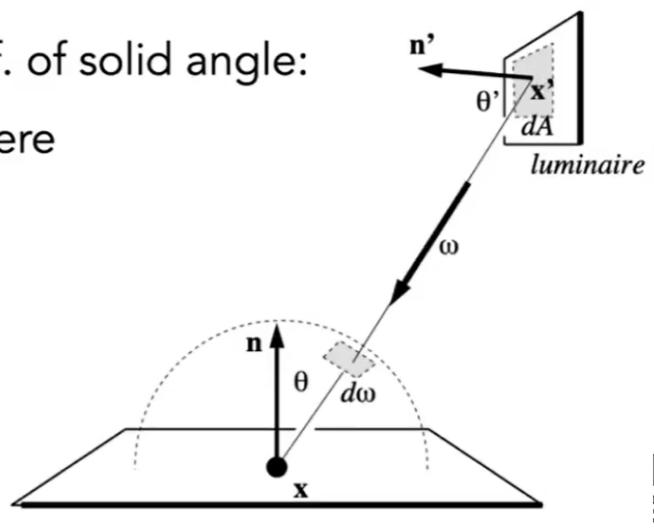
解决方法：在光源上取一块面积dA积分、采样

将dA投影在单位圆上，来得到对应的 $d\omega$

Easy! Recall the alternative def. of solid angle:
Projected area on the unit sphere

$$d\omega = \frac{dA \cos \theta'}{\|x' - x\|^2}$$

(Note: θ' , not θ)



重写渲染方程，变量替换

$$\begin{aligned}
 L_o(x, \omega_o) &= \int_{\Omega^+} L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \cos \theta d\omega_i \\
 &= \int_A L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \frac{\cos \theta \cos \theta'}{\|x' - x\|^2} dA
 \end{aligned}$$

4 没有判断光源是否可以达到点p

最终方法

1 光源直接贡献（判断光源是否可以达到点p）

2 间接光照贡献

伪代码

```

shade(p, wo)

    # Contribution from the light source.

    Uniformly sample the light at x' (pdf_light = 1 / A)
    L_dir = L_i * f_r * cos θ * cos θ' / |x' - p|^2 / pdf_light
    ↴
    # Contribution from other reflectors.

    L_indir = 0.0

    Test Russian Roulette with probability P_RR
    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos θ / pdf_hemi / P_RR
    ↴
    Return L_dir + L_indir
    ↴

```

结果与真实情况几乎一模一样

Yes, almost 100% correct, a.k.a. **PHOTO-REALISTIC**



Photo

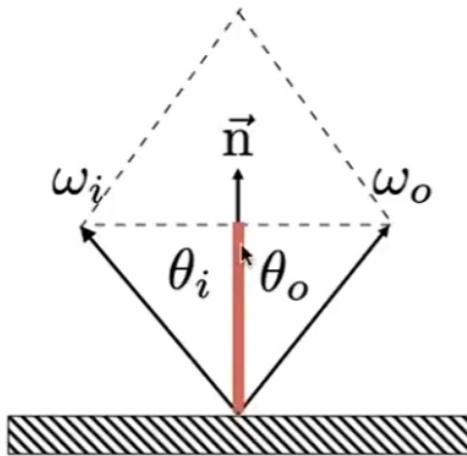


Path traced:
global illumination

第九讲 材质与外观

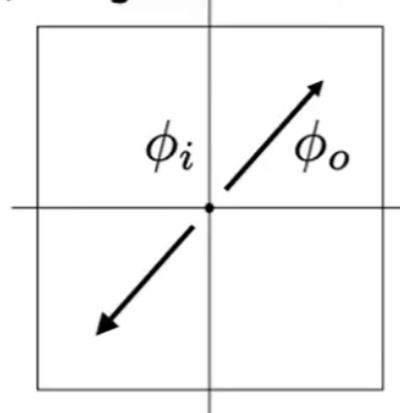
材质==BRDF(决定了光形成的方式，就是作业中的 f_r)

反射



$$\theta = \theta_o = \theta_i$$

**Top-down view
(looking down on surface)**



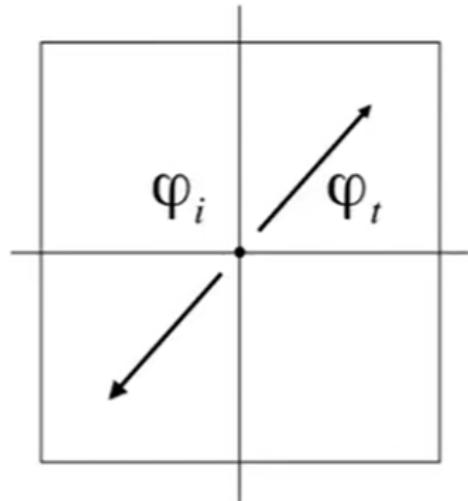
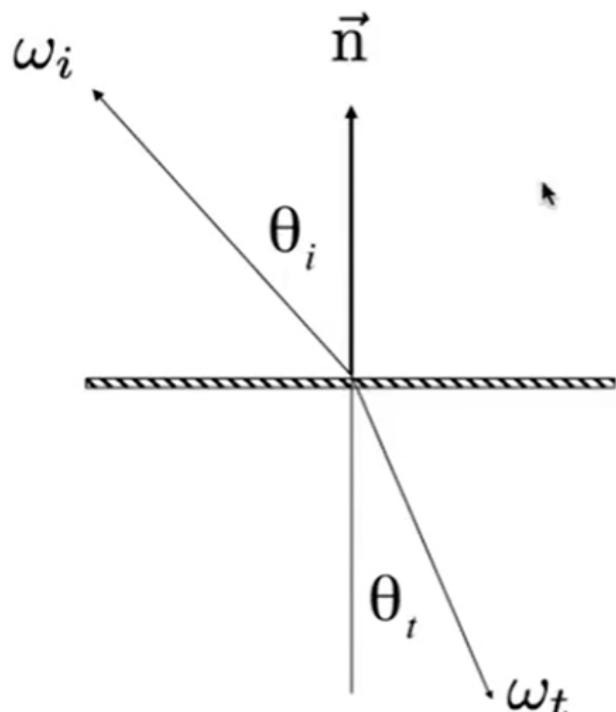
$$\phi_o = (\phi_i + \pi) \bmod 2\pi$$

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

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

折射

Transmitted angle depends on
index of refraction (IOR) for incident ray
index of refraction (IOR) for exiting ray

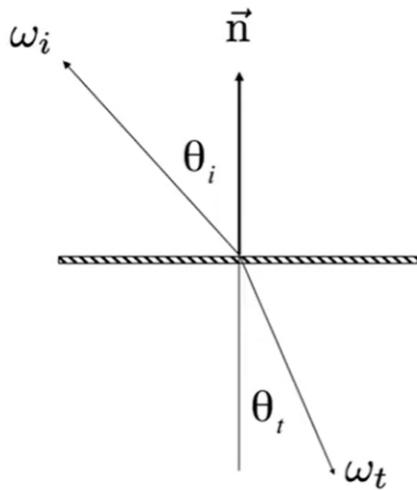


$$\varphi_t = \varphi_i \pm \pi$$


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

全反射产生条件：当光从密的地方进入稀疏的地方

Law of Refraction



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

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

Total internal reflection:

$$1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos^2 \theta_i) < 0$$

When light is moving from a more optically dense

medium to a less optically dense medium: $\frac{\eta_i}{\eta_t} > 1$

Light incident on boundary from large enough angle will
not exit medium.

菲涅尔项

现象

垂直向下看的时候，几乎看不到反射；水平看过去比较明显

Reflectance depends on incident angle (and polarization of light)



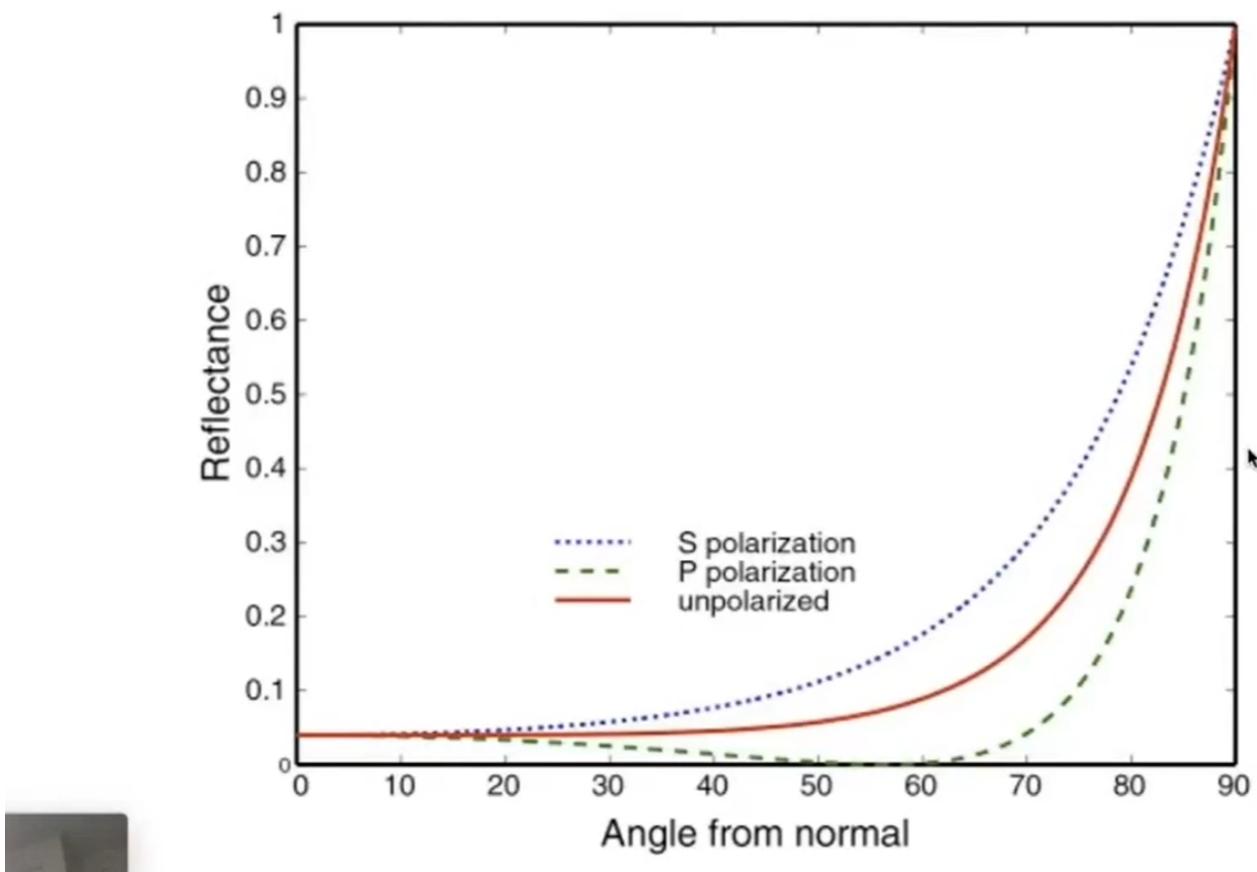
This example: reflectance increases with grazing angle

[Lafortune et al. 1997]

光和物体平行，几乎所有能量都被反射；垂直的时候大部分折射

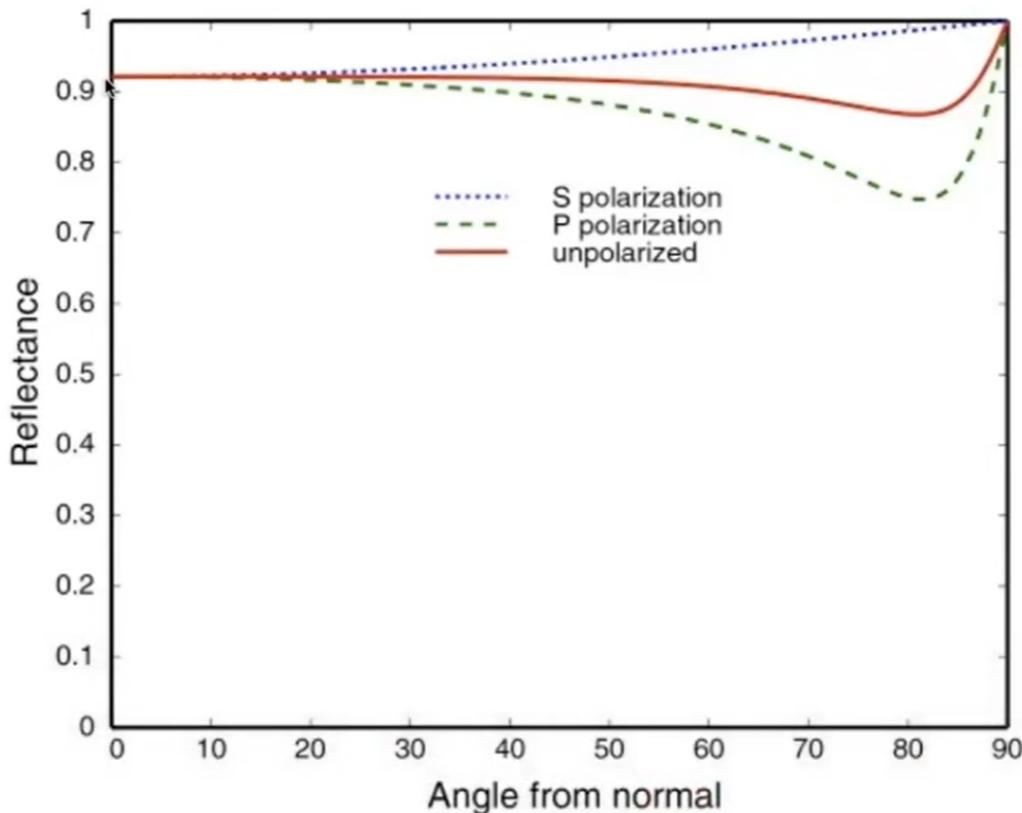
绝缘体的菲涅尔项

Fresnel Term (Dielectric, $\eta=1.5$)



导体的菲涅尔项，比如铜制的镜子，垂直的时候反射的光依然很大

Fresnel Term (Conductor)



计算

复杂版

$$R_s = \left| \frac{n_1 \cos \theta_i - n_2 \cos \theta_t}{n_1 \cos \theta_i + n_2 \cos \theta_t} \right|^2 = \left| \frac{n_1 \cos \theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}}{n_1 \cos \theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}} \right|^2,$$
$$R_p = \left| \frac{n_1 \cos \theta_t - n_2 \cos \theta_i}{n_1 \cos \theta_t + n_2 \cos \theta_i} \right|^2 = \left| \frac{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2} - n_2 \cos \theta_i}{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2} + n_2 \cos \theta_i} \right|^2.$$
$$R_{\text{eff}} = \frac{1}{2} (R_s + R_p).$$

简单版 Schlick's approximation

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

微表面模型

概念

假设物体表面粗糙

- 1 从远处看是平且粗糙的，材质外观
- 2 从近处看是凹凸不平的，每个小的微表面是镜面，不同的几何

微表面的BRDF

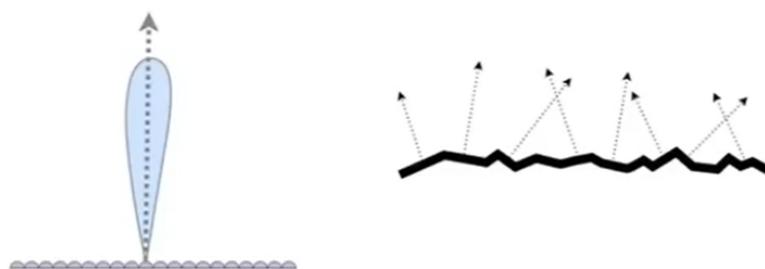
根据微表面的法线分布分布

大致朝前，是雾面的

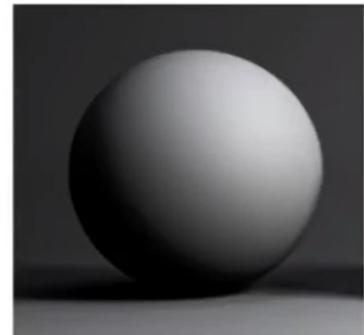
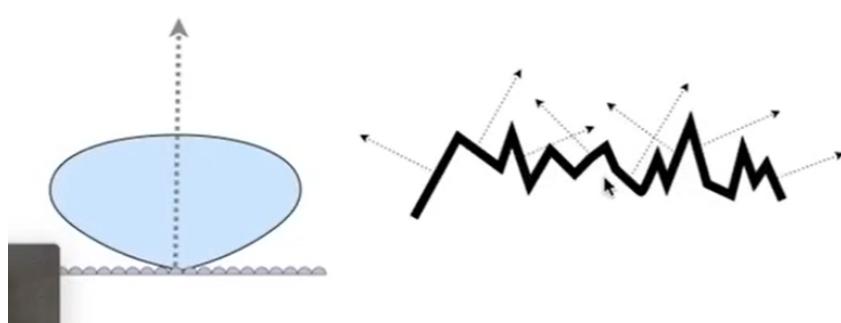
朝四面八方，是漫反射

- Key: the **distribution** of microfacets' normals

- Concentrated <=> glossy



- Spread <=> diffuse



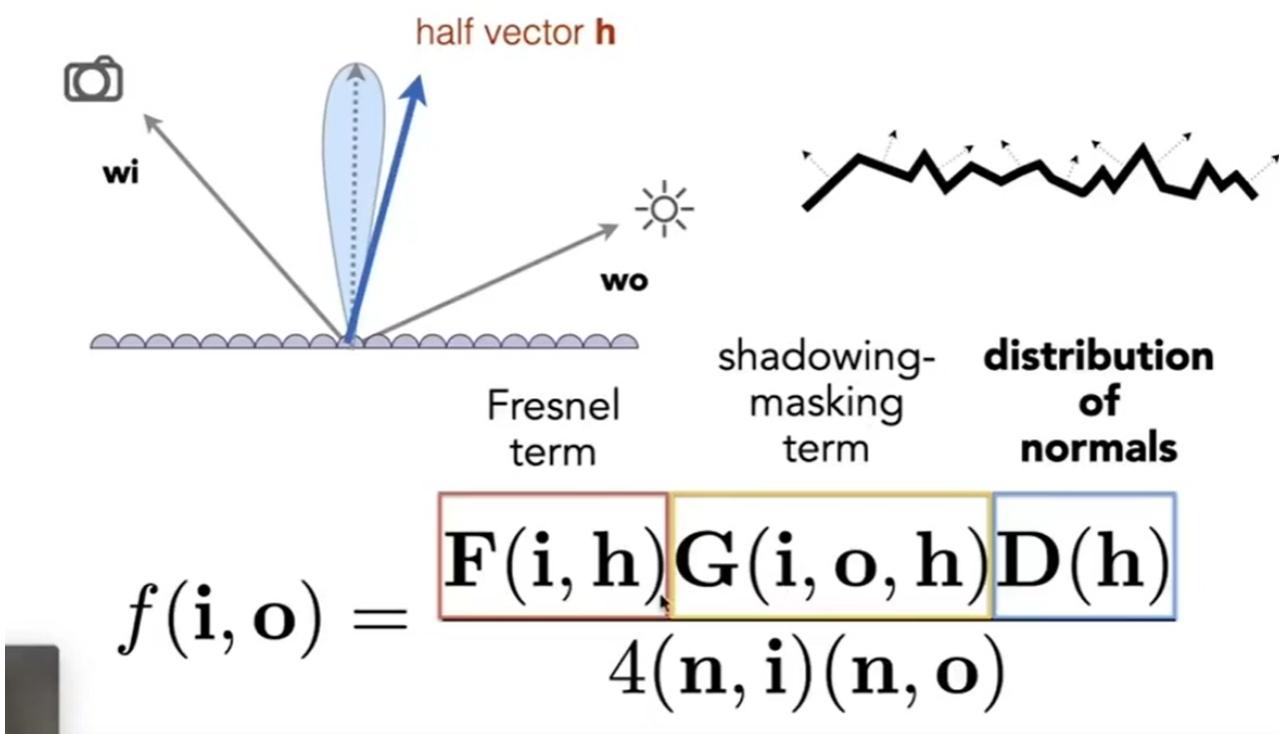
有多少光从 w_i 反射到 w_o ？

F: 菲涅尔项

D(h): w_i 和 w_o 的角平分线出大概分布多少法线

G: 当光几乎平行地经过表面，前面的微表面会遮挡后面的微表面，用该项来修正

- What kind of microfacets reflect w_i to w_o ?
(hint: microfacets are mirrors)

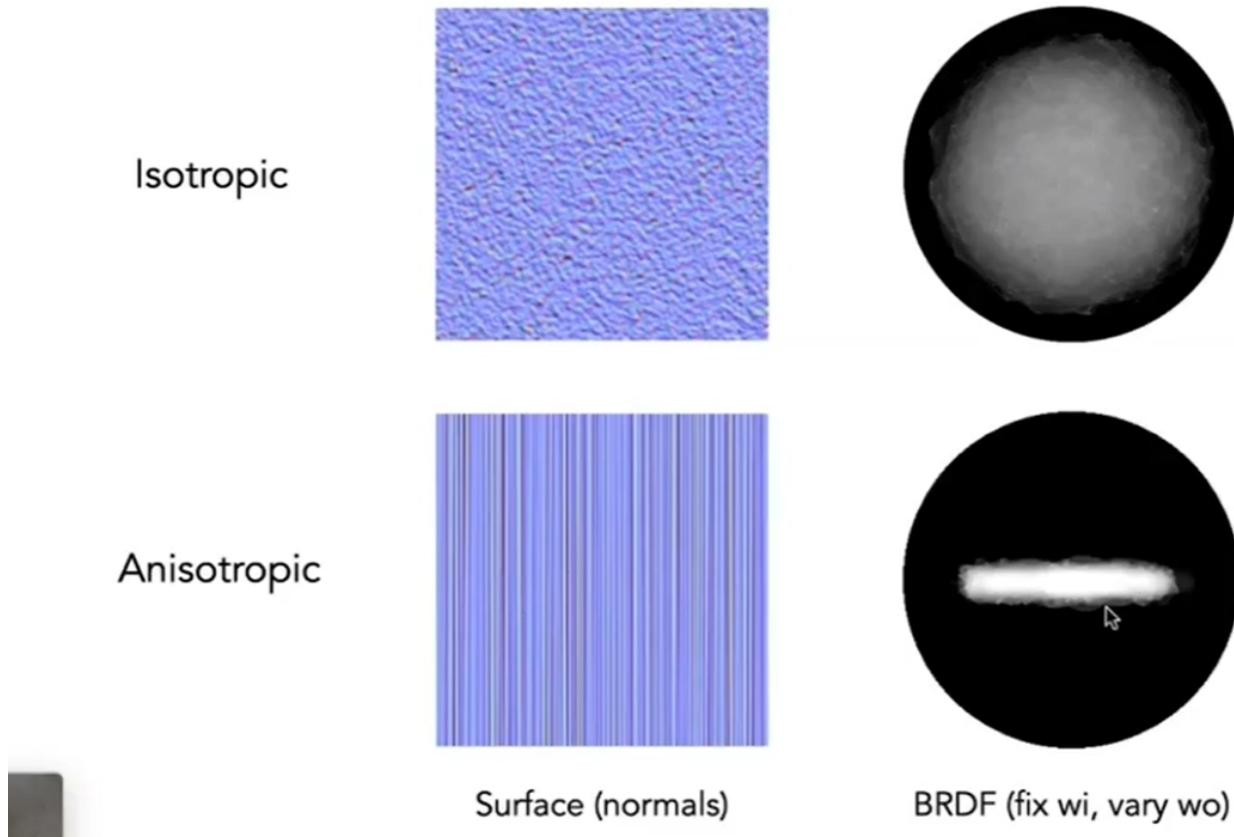


各向同性/各向异性材质

各向同性: 微表面各方向法线分布一致

各向异性: 微表面各方向法线分布有方向性

- Key: directionality of underlying surface



BRDF总结

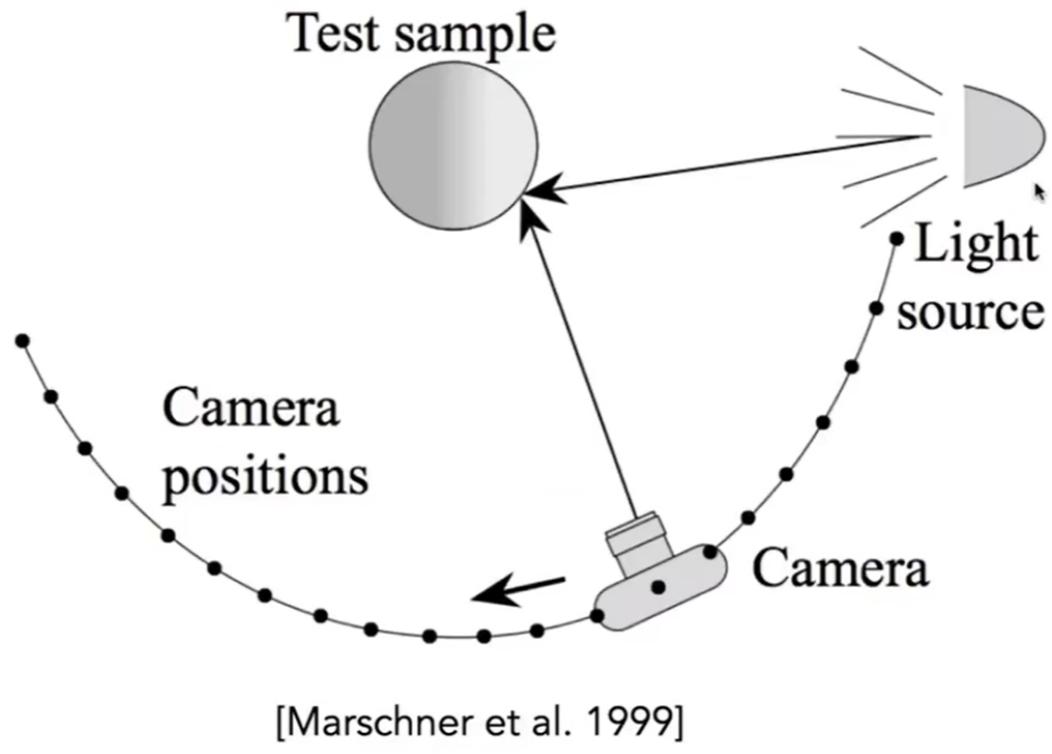
性质

- 1 非负性, $f_r > 0$
- 2 线性, 本身可以分成几块, 再加起来
- 3 可逆性, 交换入射和出射方向, f_r 不变
- 4 能量守恒
- 5 各项同性 (数据只需要存储三维)、各向异性

测量

从四面八方打光, 并从四面八方测量

Image-Based BRDF Measurement



第十讲 渲染前沿技术

光线传播

概念

- 1 无偏估计：估计的期望是正确的
- 2 有偏估计：估计的期望是不正确的

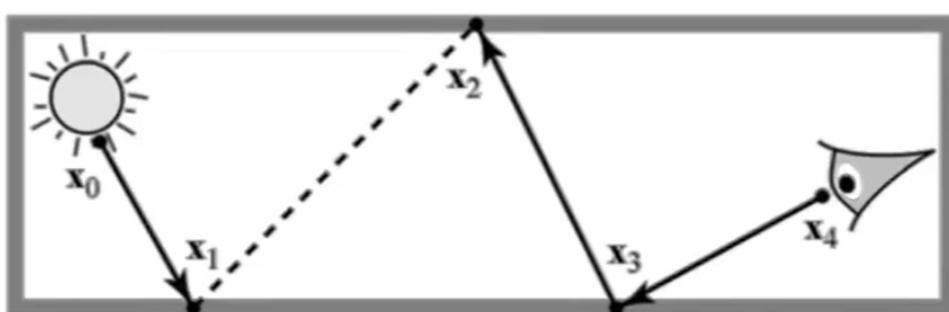
双向路径追踪 (bidirection path tracing BDPT)

无偏估计

从光源生成一部分路径，从相机生成一部分，中间连起来

- BDPT

- Traces sub-paths from both the camera and the light
- Connects the end points from both sub-paths



[Veach 1997]

Metropolis(人名) light transport (MLT)

无偏估计

好处：适合复杂的困难的光路传播

坏处：不知道什么时候光线会收敛

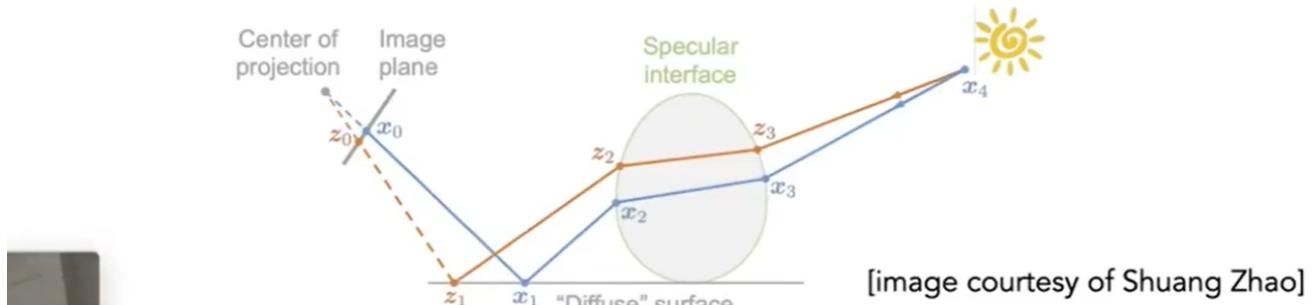
局部性比较强，导致有的地方收敛了，有的地方没有，导致抖动比较大

马尔科夫链的蒙特卡洛方法

根据一条采样的光线，在它周围生成其他的采样

局部效果比较好

- A Markov Chain Monte Carlo (MCMC) application
 - Jumping from the current sample to the next with some PDF
- Very good at **locally** exploring difficult light paths
- Key idea
 - Locally perturb an existing path to get a new path



光子映射 (photon mapping)

有偏方法

优势

光的聚焦所显示的图案 (caustics) 和处理specular-diffuse-specular (SDS)

实现方法 (其中一种)

1 从光源出发，直到光子打到漫反射 (diffuse) 表面后停止

2 从相机出发，直到打到漫反射 (diffuse) 表面后停止

3 计算局部的密度估计，由于光子分布越集中的地方越亮，所以对于每一个观察点我们取周围最近的N个光子，计算N个光子所占的面积A, N/A就是密度

4 N大效果好，但是会糊；步骤1多发射一些光子，效果会更好，有偏但一致

模糊==有偏

样本足够多就不模糊==一致

Vertex connection and merging

结合了BDPT和photon mapping

实时辐射度方法 (Instant Radiosity)

认为已经照亮的地方就是光源，用这些光源去渲染别的点

问题：1 缝隙会出现一些不该出现的发光点

2 镜面物体做不了

外观建模

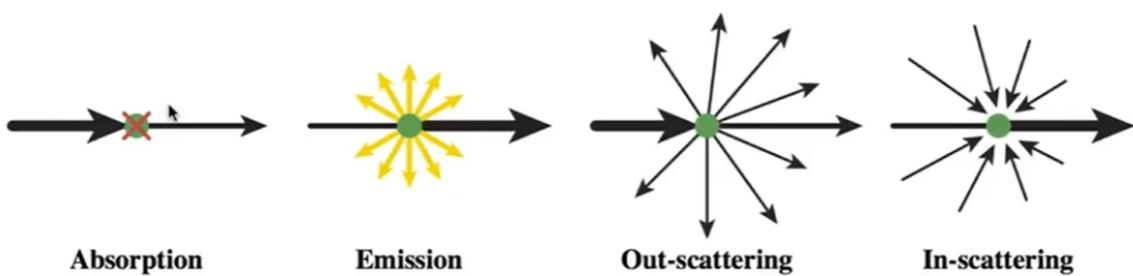
非表面模型

散射介质

雾、云

光线穿过这些介质的时候，会被吸收也会被散射

- At any point as light travels through a participating medium, it can be (partially) absorbed and scattered.



[Wojciech Jarosz]

怎么散射？

由**Phase function**决定（类似BRDF）

应用：游戏、动画场景

头发

头发上的高光有白色的也有有色的

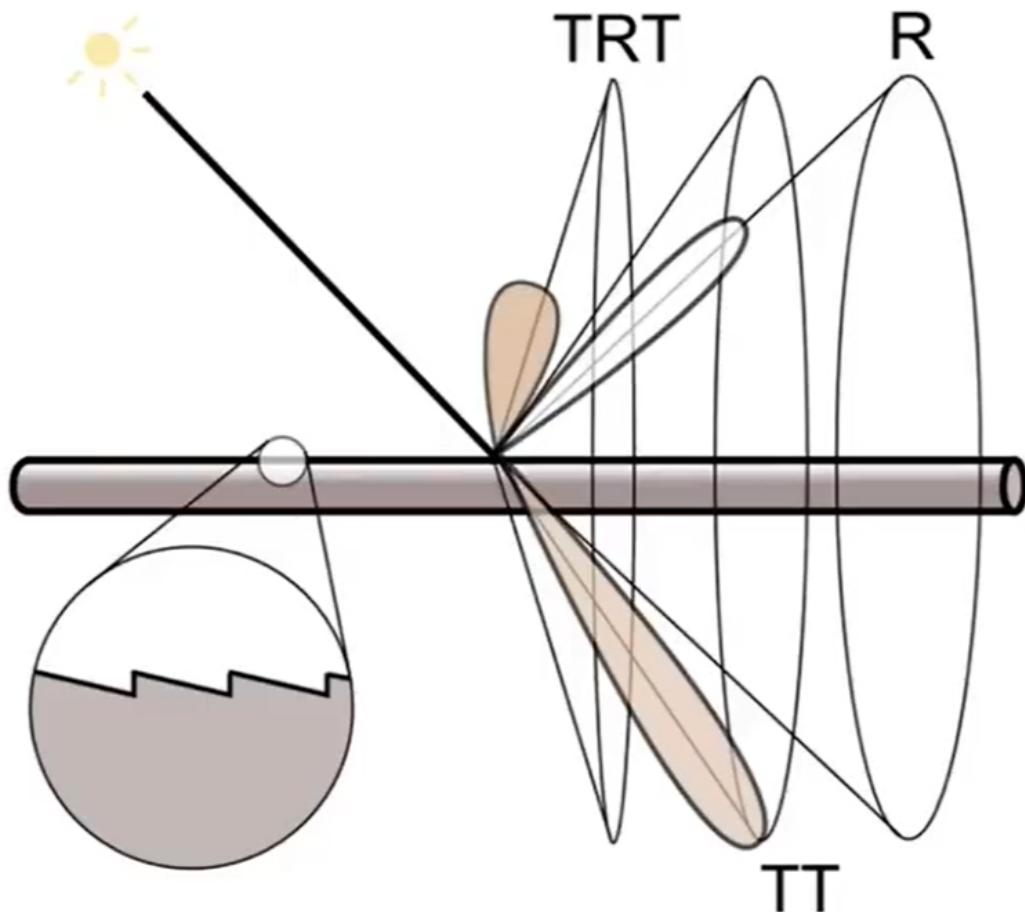
1 kajiya-kay 模型

把打到头发上的光分为漫反射和高光，效果很差

2 Marschner 模型 用的比较多

有的光反射reflection，有的光穿进去再出来T (transmission) T，有的光穿进去在内壁反射再出来TRT

把头发的一小段当做一个圆柱体，有外表面和内里组成，内里的色素多就越黑



人和动物的毛发都由三层组成：表面、中间、髓质（光遇到它会散射），但是动物的髓质是很大的，所以人和动物的毛发不能一概而论

使用 双层圆柱模型

颗粒材质 (granular material)

表面模型

半透明 (translucent) 材质

translucent 光即会穿过也会在里面散射

比如：玉石、水母

使用**BSSRDF**，光由一个点进入物体，并会从其他点出来

用两个光源来近似

布料

一系列缠绕的纤维

纤维缠绕成股，股再缠绕成线

解决办法：

1把线当成体积，计算的时候分成很小的体积，计算量很大

2每个纤维都渲染

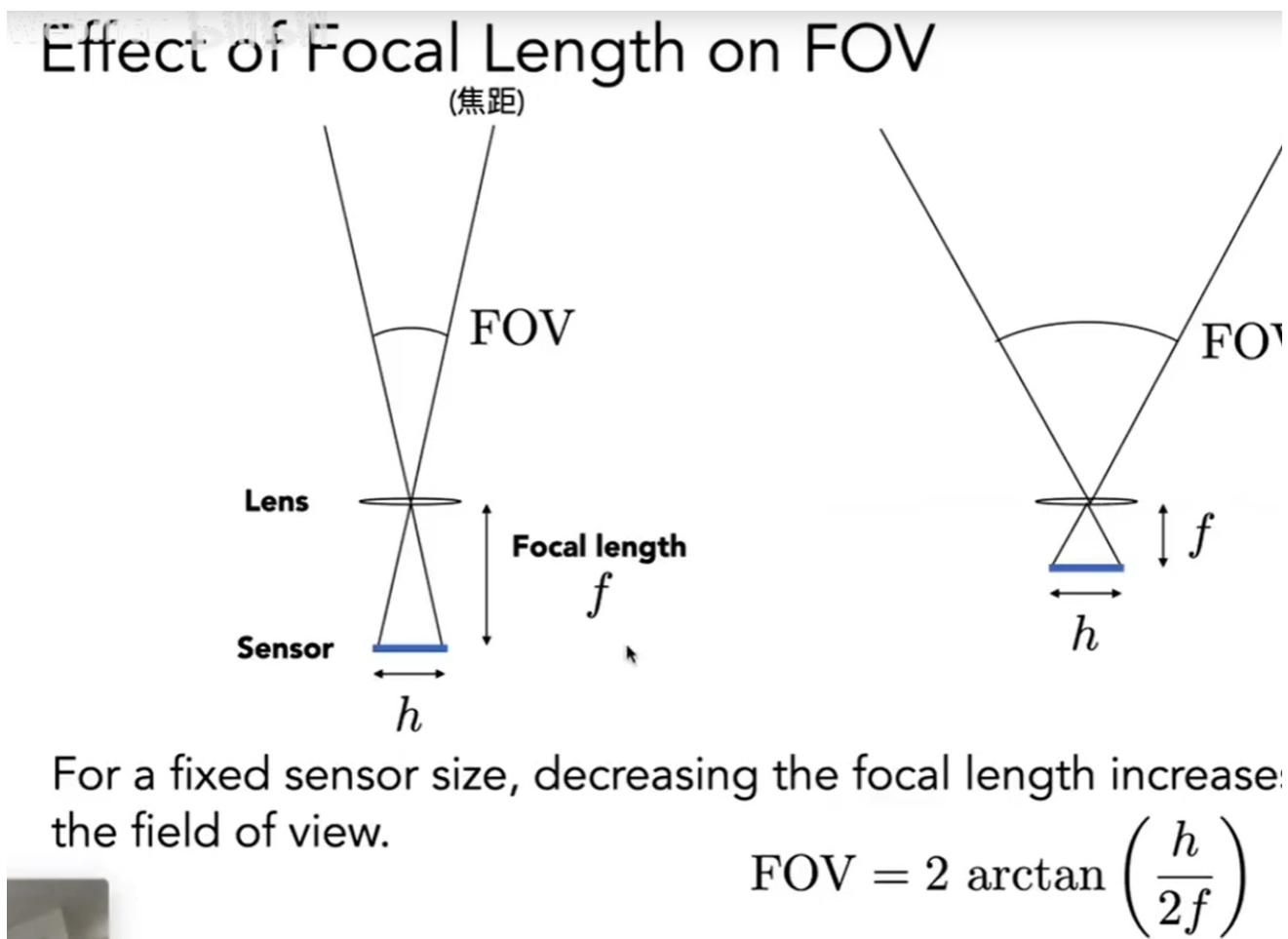
第十一讲 相机、透镜、光场

相机

传感器上每一个点记录了当时的Irradiance

视场 (field of view fov)

焦距越小，视场越大（广角）



定义焦距指的是对于35mm格式的胶片为基准

曝光 (exposure)

$\text{exposure} = \text{time} * \text{irradiance}$

快门留的时间越长，进入的光越多，越亮

IOS (gain)

简单的在照片上线性提高曝光度，噪声也会被放大

光圈

使用FN来表示，只关心N是多少，F只是个标识

快门

控制光能进来多久

快门时间越长，运动模糊越明显

运动模糊不一定是坏事

透镜

假设薄透镜可以改变焦距

使用透镜组可以实现

焦距、物距、像距

Gauss' Ray Tracing Construction

$$\frac{h_o}{z_o - f} = \frac{h_i}{f}$$

$$\frac{h_o}{f} = \frac{h_i}{z_i - f}$$

$$\frac{h_o}{h_i} = \frac{z_o - f}{f}$$

$$\frac{h_o}{h_i} = \frac{f}{z_i - f}$$

$$\frac{z_o - f}{f} = \frac{f}{z_i - f}$$

Object / image heights
factor out - applies to all rays

$$(z_o - f)(z_i - f) = f^2$$

$$z_o z_i - (z_o + z_i)f + f^2 = f^2$$

$$z_o z_i = (z_o + z_i)f$$

$$\frac{1}{f} = \frac{1}{z_i} + \frac{1}{z_o}$$



•

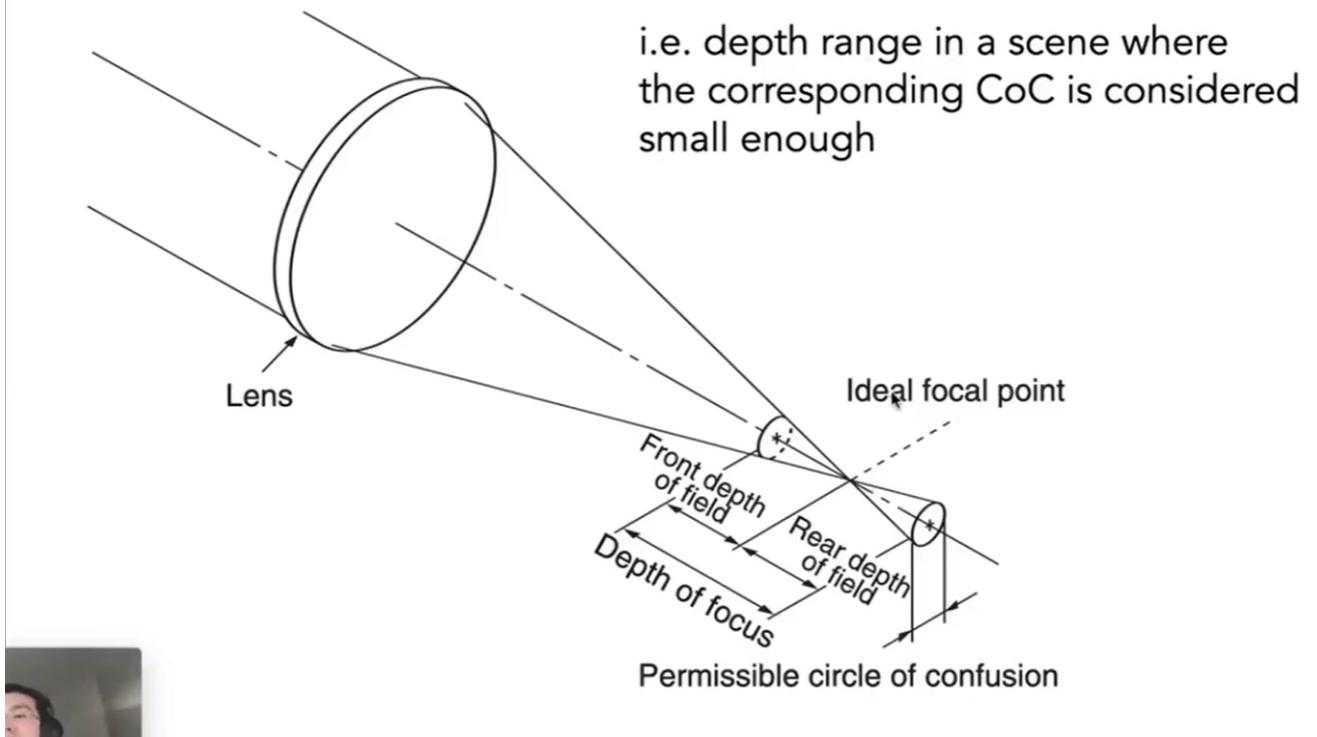
远处的物体模糊的原因，成像平面不在相距

光圈越大，越模糊

景深 depth of field

成像在理想成像点的一定范围内，我们认为是锐利的

这个范围我们称为景深

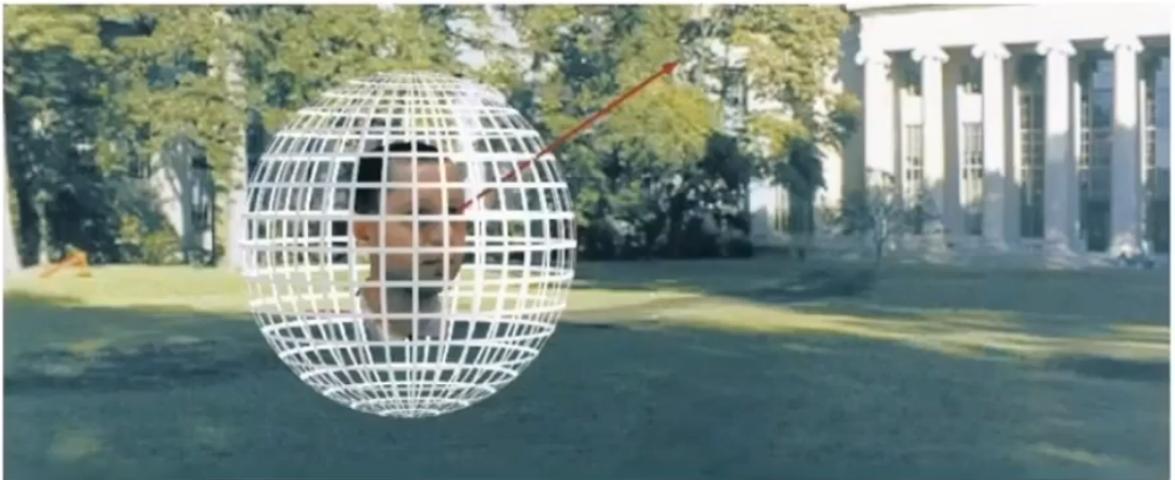


光场 (lumigraph/light field)

全光函数

彩色照片，两个角度一个波长（表示颜色）

Color snapshot



$$P(\theta, \phi, \lambda)$$

is intensity of light

- Seen from a single view point
- At a single time
- As a function of wavelength



电影，加个时间

A movie



$$P(\theta, \phi, \lambda, t)$$

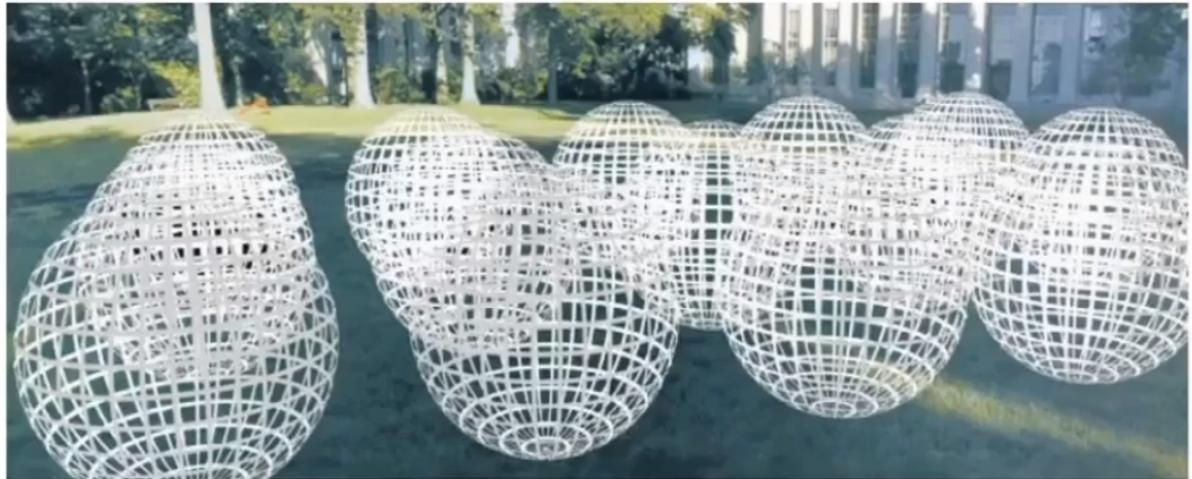
is intensity of light

- Seen from a single view point
- Over time
- As a function of wavelength



全息电影 (vr) , 加个位置坐标, 在任何时间、任何位置、任何角度看到的颜色

Holographic movie



$$P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$$

is intensity of light

- Seen from ANY viewpoint
- Over time
- As a function of wavelength

由光路的可逆性，看到物体可以理解为物体的包围盒上任意一点向各个的地方发射的光的情况

光场：物体（物体的包围盒）任意一点向各个的方向的光的强度，因此有了光场就可以知道从任意角度看向物体能看到什么

Only need plenoptic surface

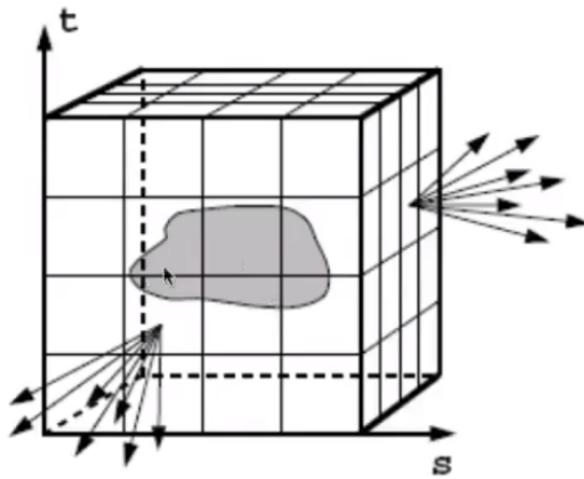
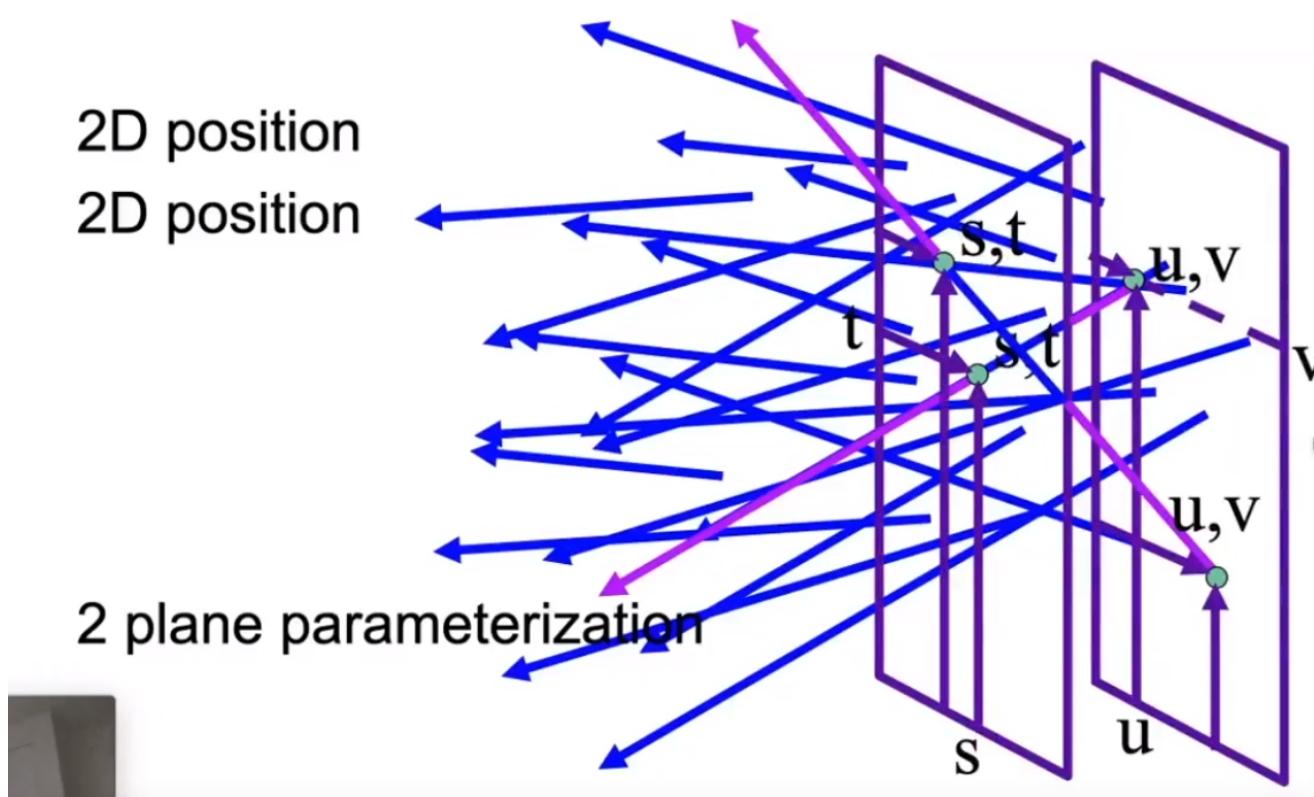


Figure 1: The surface of a cube holds all the radiance information due to the enclosed object.

光场定义（参数化）

定义两个平面，光分别在这两个平面上的位置 (u, v) (s, t)

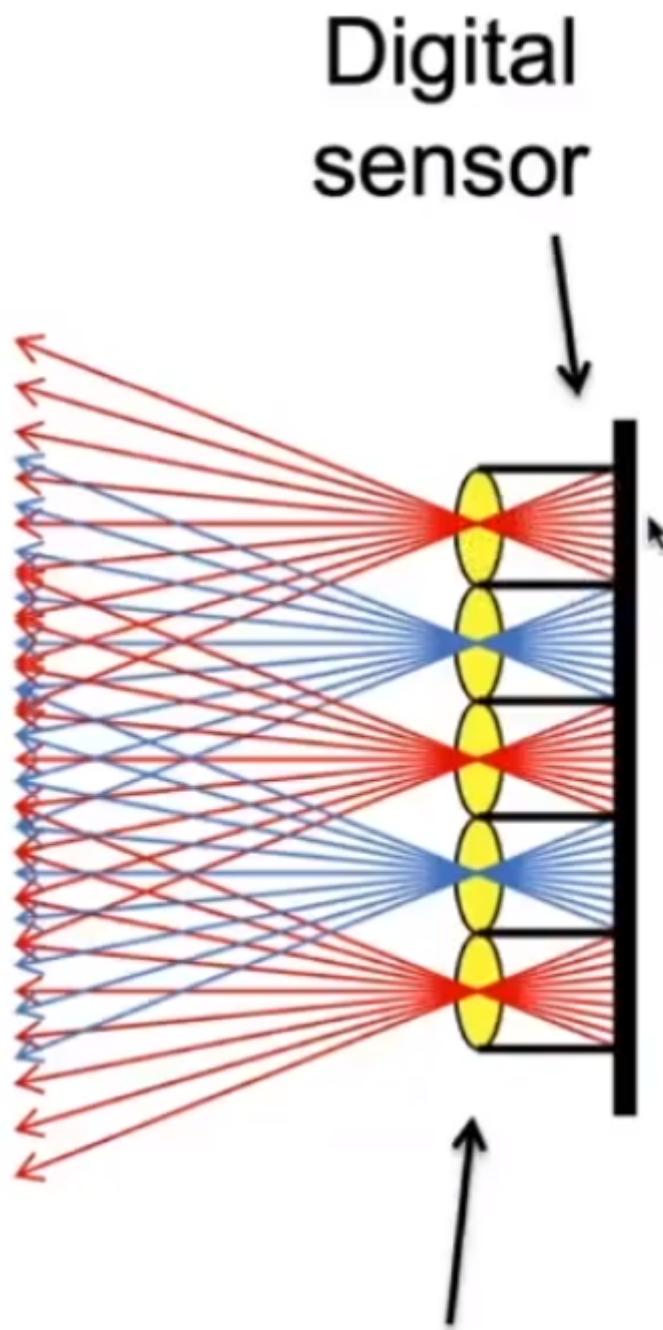
Lumigraph - Organization



光场照相机

先拍，再聚焦

在原本的感光元件处放置微透镜，让光中的不同颜色分开，分别记录在后面的感光元件上
(irradiance->radiance)



**Microlens
array (MLA)**

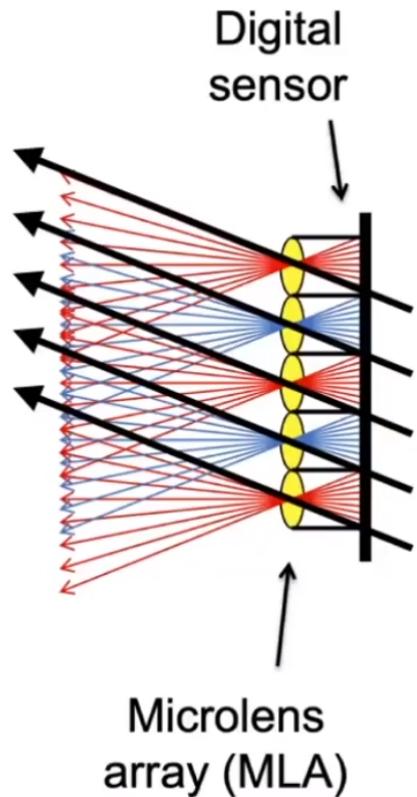
如何得到普通照片，比如对于每一个微透镜都取最下面的光线，就得到了相机从下面拍的照片，都取中间的光线，就得到了从中间拍的照片，就像在移动相机

相当于相机记录了整个光场

Light Field Camera

How to get a “regular” photo from the light field photo?

- A simple case — always choose the pixel at the bottom of each block
- Then the central ones & the top ones
- Essentially “moving the camera around”



第十二讲 颜色和感知

颜色

颜色是人感知的结果

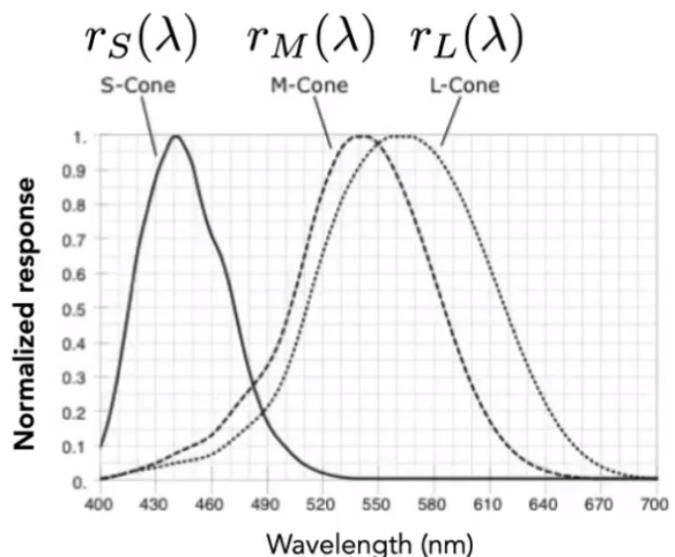
人眼视网膜感应部位 (s m l) 对不同波长的感知，让人看到不同的颜色，但个体之间有很大差异

Now we have three detectors (S, M, L cone cells), each with a different spectral response curve

$$S = \int r_S(\lambda) s(\lambda) d\lambda$$

$$M = \int r_M(\lambda) s(\lambda) d\lambda$$

$$L = \int r_L(\lambda) s(\lambda) d\lambda$$



加色系统

计算机表示颜色的方式

Additive Color

- Given a set of primary lights, each with its own spectral distribution (e.g. R,G,B display pixels):

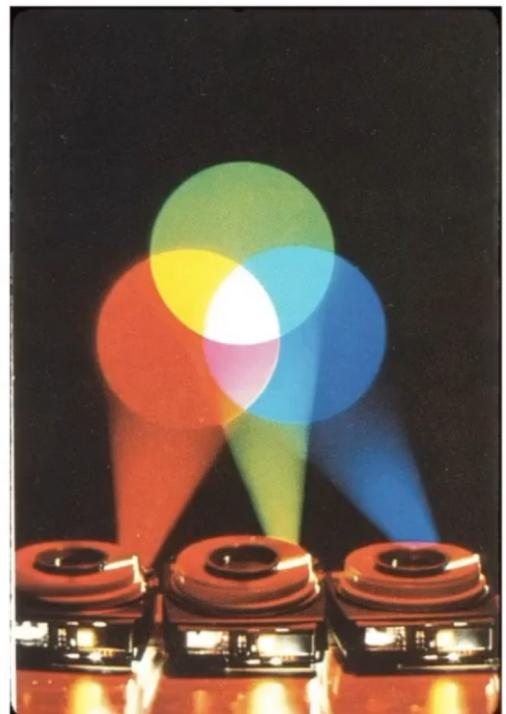
$$s_R(\lambda), s_G(\lambda), s_B(\lambda)$$

- Adjust the brightness of these lights and add them together:

$$R s_R(\lambda) + G s_G(\lambda) + B s_B(\lambda)$$

- The color is now described by the scalar values:

$$R, G, B$$



标准颜色空间

sRGB

把X, Y, Z归一化，只需要记录两个值

Separating Luminance, Chromaticity

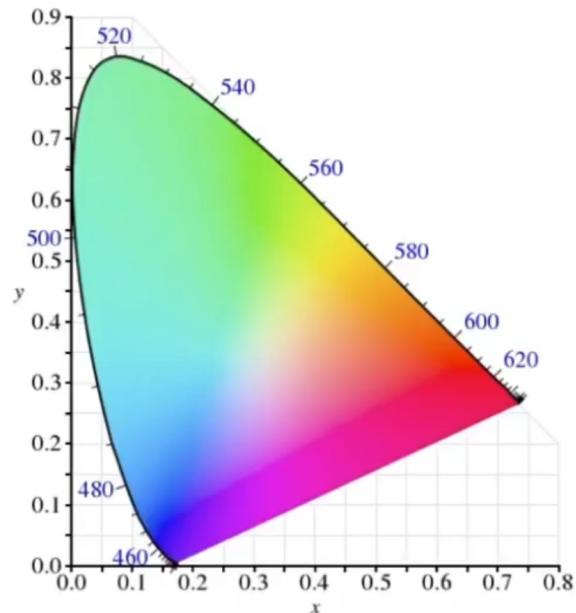
Luminance: Y

Chromaticity: x, y, z , defined as
(色度)

$$x = \frac{X}{X + Y + Z}$$

$$y = \frac{Y}{X + Y + Z}$$

$$z = \frac{Z}{X + Y + Z}$$



- since $x + y + z = 1$, we only need to record two of the three
- usually choose x and y , leading to (x, y) coords at a specific brightness Y

色域

不同颜色空间表示颜色的范围

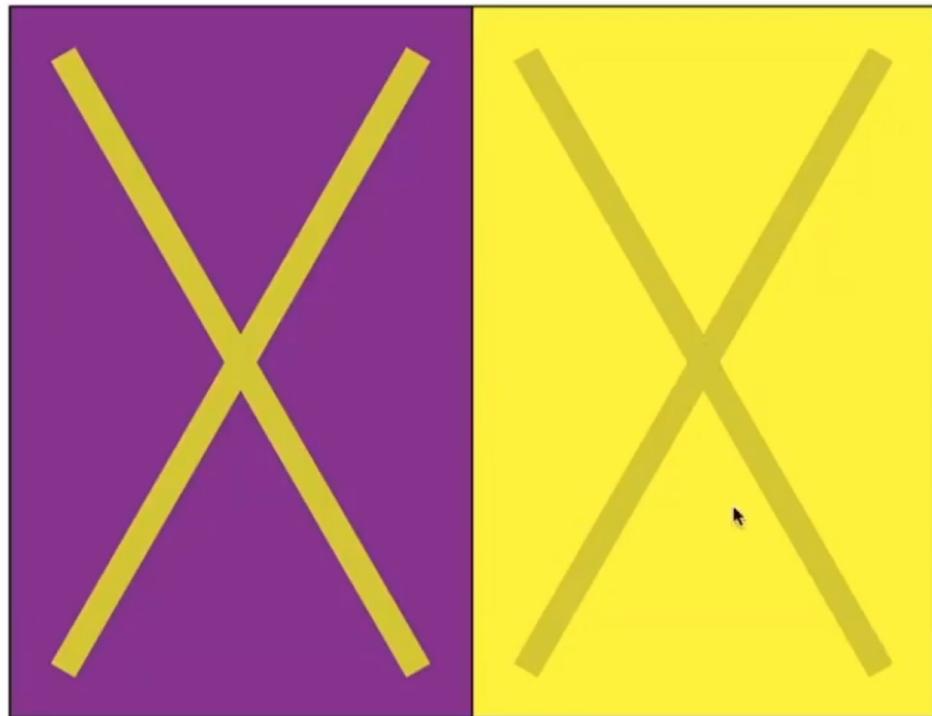
互补色

黑白、红绿、黄蓝

颜色本身是相对的

其实两个颜色是一样的

Everything is Relative



减色系统

在打印中用的很多

黑色可以调出来，但是黑色用的多且便宜，所以加了个黑色

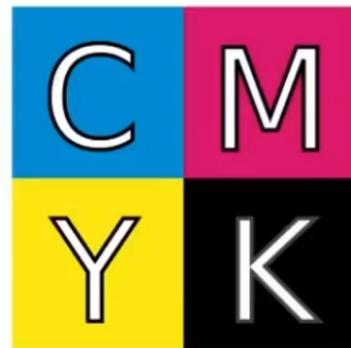
CMYK: A Subtractive Color Space

Subtractive color model

- The more you mix, the darker it will be

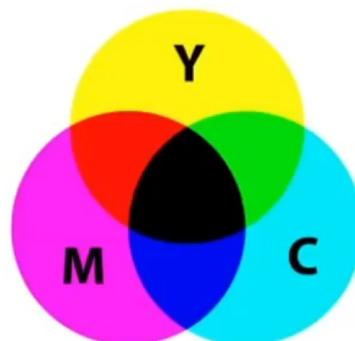
Cyan, Magenta, Yellow, and Key

Widely used in printing



Question:

- If mixing C, M and Y gives K, why do you need K?



第十三讲 动画/模拟

关键帧动画 keyframe animation

给出关键帧，关键帧之间插值

物理模拟 physical simulation

根据牛顿定律 $F = ma$

建立物体间的相互作用力

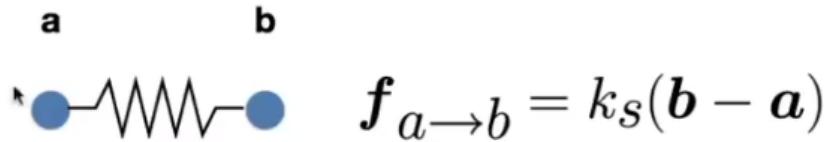
质点弹簧系统 (mass spring rope)

一系列由质点和弹簧组成的系统

胡克定律

A Simple Spring

Idealized spring



$$\mathbf{f}_{a \rightarrow b} = k_s(\mathbf{b} - \mathbf{a})$$

$$\mathbf{f}_{b \rightarrow a} = -\mathbf{f}_{a \rightarrow b}$$

Force pulls points together

Strength proportional to displacement (Hooke's Law)

k_s is a spring coefficient: stiffness

一般情况，弹簧长度l

Non-Zero Length Spring

Spring with non-zero rest length



$$\mathbf{f}_{a \rightarrow b} = k_s \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} (\|\mathbf{b} - \mathbf{a}\| - l)$$

Rest length

假如不加入摩擦力，会一直弹下去，b点指的是速度

Introducing Energy Loss

Simple motion damping

$$\frac{f}{\dot{b}} \quad f = -k_d \dot{b}$$

- Behaves like viscous drag on motion
- Slows down motion in the direction of velocity
- k_d is a damping coefficient

但只能描述外部的力，弹簧内部的损失没法表示

Damp only the internal, spring-driven motion

$$f_b = -k_d \frac{\dot{b} - \dot{a}}{\|b - a\|} (\dot{b} - \dot{a}) \cdot \frac{b - a}{\|b - a\|}$$

Relative velocity of b,
assuming a is static (vector)

Damping force applied on b

Relative velocity projected to the direction from a to b (scalar)

Direction from a to b

- Viscous drag only on change in spring length
 - Won't slow group motion for the spring system (e.g. global translation or rotation of the group)
- Note: This is only one specific type of damping

粒子系统

1由大量粒子组成

2计算内部（粒子和粒子之间的，引力斥力摩擦力空气阻力碰撞）力和外部力

3更新粒子位置和速度，有一些消失

4渲染

适合表现一些魔法的特效

运动学forward kinematics

定义一些关节、骨骼，让他们的旋转

逆运动学

给你动作，来计算关节、骨骼是怎么旋转的

问题：可能无解或多个解

Rigging

给物体添加造型

设置一些控制点，控制物体

动作捕捉 motion capture

优势：

贴近真实

得到结果快

劣势：

准备起来复杂

可能不符合艺术需求，需要调整

单粒子模拟single particle simulation

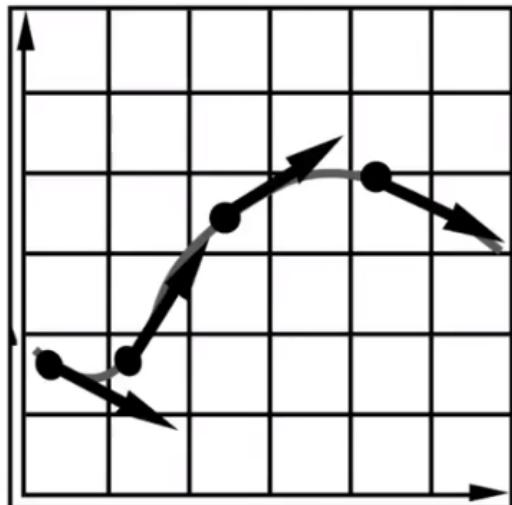
常微分方程

Computing position of particle over time requires solving a first-order ordinary differential equation:

$$\frac{dx}{dt} = \dot{x} = v(x, t)$$

"First-order" refers to the first derivative being taken.

"Ordinary" means no "partial" derivatives, i.e. x is just a function of t



Witkin and Baraff

欧拉方法 (显式、前向)

定义

x 点为速度，根据上一帧位置/速度，更新下一帧的位置/速度

Δt 越小，效果越好

Euler's Method

Euler's Method (a.k.a. Forward Euler, Explicit Euler)

- Simple iterative method
- Commonly used
- Very inaccurate
- Most often goes **unstable**

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t \dot{\boldsymbol{x}}^t$$

$$\dot{\boldsymbol{x}}^{t+\Delta t} = \dot{\boldsymbol{x}}^t + \Delta t \ddot{\boldsymbol{x}}^t$$

问题

但结果很不稳定，比如下图的螺旋形

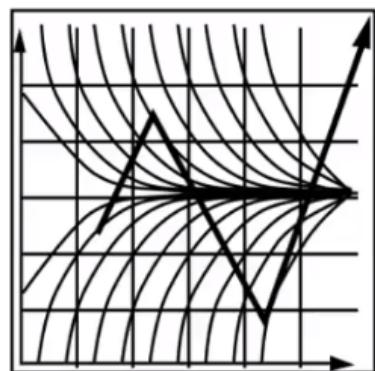
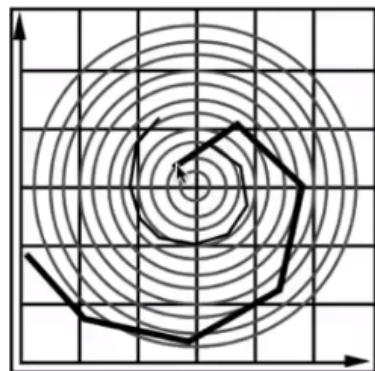
Instability of the Euler Method

The Euler method (explicit / forward)

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t \mathbf{v}(\mathbf{x}, t)$$

Two key problems:

- Inaccuracies increase as time step Δt increases
- Instability is a common, serious problem that can cause simulation to diverge



解决办法

1 中点法，用两次欧拉方法，第二次是对中点用

2 自适应步长，计算中点位置，假如两次位置变化不大，或者步长已经很短了，就停止

欧拉方法（隐式、后向）

定义

用下一帧更新速度、位置

Implicit Euler Method

Implicit methods

- Informally called backward methods
- Use derivatives in the future, for the current step

$$\begin{aligned}\mathbf{x}^{t+\Delta t} &= \mathbf{x}^t + \Delta t \dot{\mathbf{x}}^{t+\Delta t} \\ \dot{\mathbf{x}}^{t+\Delta t} &= \dot{\mathbf{x}}^t + \Delta t \ddot{\mathbf{x}}^{t+\Delta t}\end{aligned}$$

刚体模拟 rigid body simulation

Simple case

- Similar to simulating a particle
- Just consider a bit more properties

$$\frac{d}{dt} \begin{pmatrix} \mathbf{X} \\ \theta \\ \dot{\mathbf{X}} \\ \omega \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{X}} \\ \omega \\ \mathbf{F}/M \\ \Gamma/I \end{pmatrix}$$

\mathbf{X} : positions
 θ : rotation angle
 ω : angular velocity
 \mathbf{F} : forces
 Γ : torque
 I : momentum of inertia

流体模拟 fluid simulation

1 假设流体是由大量的刚体小球组成的

2 假设水不可压缩

3 当某一部分的密度不对时，需要做一些修正（梯度下降）

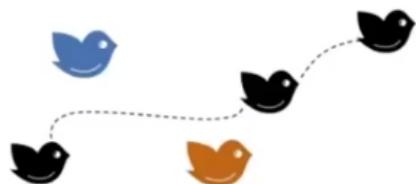
欧拉视角和拉格朗日视角

拉格朗日视角：每一个粒子都做对即可（质点法）

欧拉视角：对一个网格每一帧穿过的物体（网格法）

TWO DIFFERENT VIEWS TO SIMULATING LARGE CONVECTIONS OF MATTERS

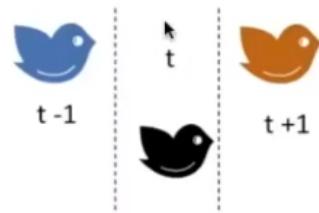
(“质点法”) LAGRANGIAN APPROACH



Photographer follows the same bird through out its course.

(“网格法”) EULERIAN APPROACH

The photographer is stationary and can take picture of all the birds passing through one frame only (say at time 't').



<https://www.youtube.com/watch?v=iDlzLkic1pY>