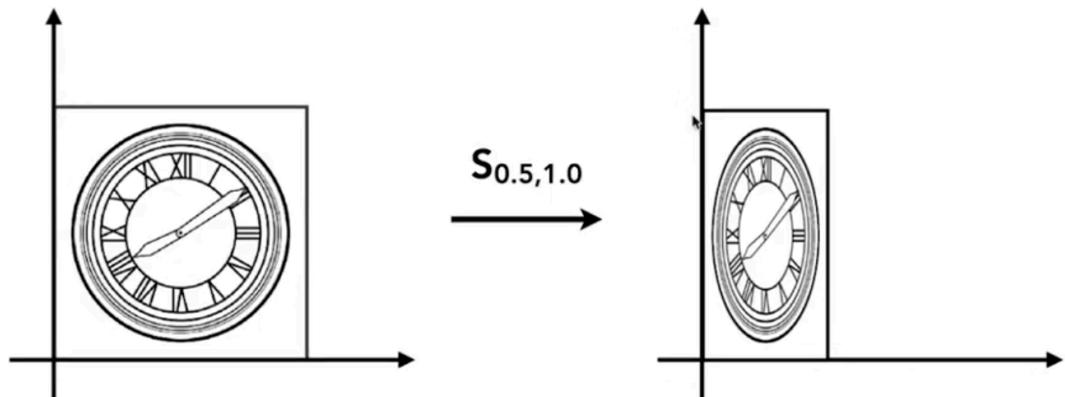


# Transformation

Viewing : 3D to 2D projection

## Scale

### Scale (Non-Uniform)



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



# Shear

GAMER: Lingqi Yan

## Shear Matrix

Hints:

- Horizontal shift is 0 at  $y=0$
- Horizontal shift is  $a$  at  $y=1$
- Vertical shift is always 0

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

16 Lingqi Yan, UC Santa Barbara

# Rotate

默认  $(0, 0)$  CCW

GAMER: Lingqi Yan

## Rotation Matrix

$$\mathbf{R}_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

18 Lingqi Yan, UC Santa Barbara

变换可写成矩阵乘积数据的形式 ==> 线性变换

## Linear Transforms = Matrices

(of the same dimension)

$$x' = a x + b y$$

$$y' = c x + d y$$

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

$$\mathbf{x}' = \mathbf{M} \mathbf{x}$$

如果需要以统一的方法表示平移

- Translation cannot be represented in matrix form

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

则应该引入齐次化过程。

点的变换添加一个1，向量的变换添加一个0.

(在射线法中，计算射线的方向变换) (计算法向量时，法向量的变换)

## Add a third coordinate (w-coordinate)

- 2D point =  $(x, y, 1)^T$
- 2D vector =  $(x, y, 0)^T$

## Matrix representation of translations

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

Valid operation if w-coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector
- point + vector = point
- point + point = ??

仿射变换都可以写成齐次坐标的形式

# Affine Transformations

Affine map = linear map + translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Using homogenous coordinates:

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



27

Lingqi Yan, UC Santa Barbara



## 2D Transformations

Scale

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

Rotation

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

Translation

$$\mathbf{T}(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$



28

Lingqi Yan, UC Santa Barbara



仅在2D下的仿射变换，矩阵最后一行为001（可以在计算时作为常量引入，而不用特意存储）

# 矩阵的结合律：变换合成与分解

Sequence of affine transforms  $A_1, A_2, A_3, \dots$

- Compose by matrix multiplication
- Very important for performance!

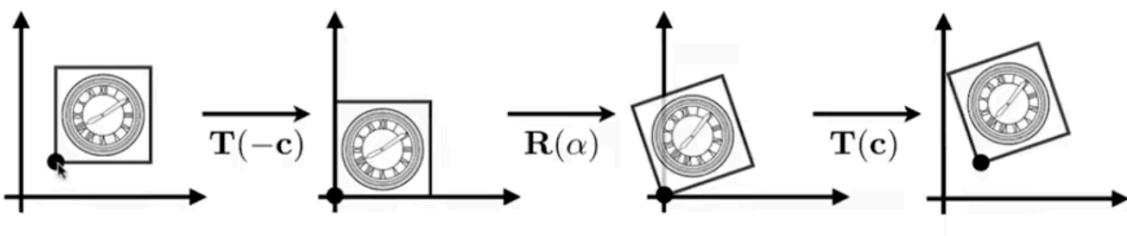
$$A_n(\dots A_2(A_1(\mathbf{x}))) = \mathbf{A}_n \cdots \mathbf{A}_2 \cdot \mathbf{A}_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Pre-multiply  $n$  matrices to obtain a single matrix representing combined transform

对于非原点的点的变换操作：先移至原点，变换后再移回去

How to rotate around a given point  $c$ ?

1. Translate center to origin
2. Rotate
3. Translate back



Matrix representation?

$$\mathbf{T}(\mathbf{c}) \cdot \mathbf{R}(\alpha) \cdot \mathbf{T}(-\mathbf{c})$$

旋转负角度 是旋转该角度下的旋转矩阵的逆矩阵

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

$$\boxed{R_{-\theta} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}} = R_\theta^T$$

$$R_{-\theta} = R_\theta^{-1} \text{ (by definition)}$$



在旋转中，旋转矩阵的逆等于旋转矩阵的转置 ==> 正交矩阵

### 三维变换

## 3D Transformations

Use homogeneous coordinates again:

- 3D point =  $(x, y, z, 1)^T$
- 3D vector =  $(x, y, z, 0)^T$

In general,  $(x, y, z, w)$  ( $w \neq 0$ ) is the 3D point:

$$(x/w, y/w, z/w)$$

# 3D Transformations

Use  $4 \times 4$  matrices for affine transformations

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

3D的旋转 (右手坐标系)

(cosa 在1的右下, 以此类推)

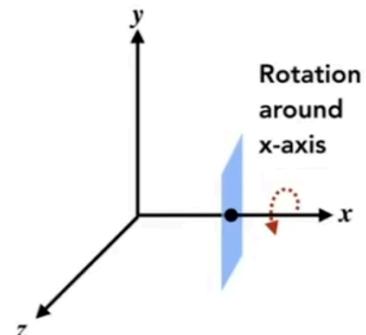
## 3D Transformations

Rotation around x-, y-, or z-axis

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

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

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



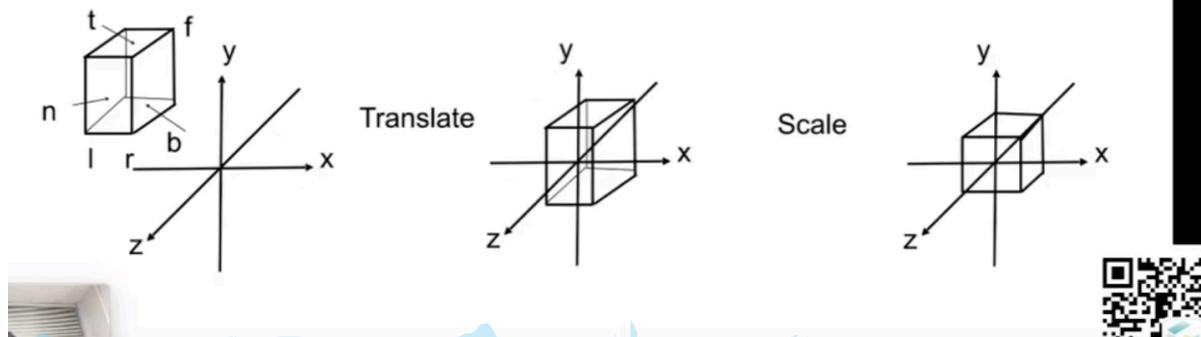
正交变换投影:

描述一个长方体，将其中心移动到原点，再将边长拉扯至立方体 $[-1, 1]^3$ 上。

# Orthographic Projection

- In general

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



透视投影：相似三角形原理，处于远近平面中间的点，投影至近平面所需要的xy上的变换。

# Perspective Projection

- In order to find a transformation
  - Find the relationship between transformed points  $(x', y', z')$  and the original points  $(x, y, z)$

$$y' = \frac{n}{z}y \quad x' = \frac{n}{z}x \text{ (similar to } y')$$

- In homogeneous coordinates,

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



## 光栅化

将标准平面坐标系中的像素映射到屏幕上：视口变换

$z$ 方向不变，即屏幕平面始终朝向相机反向

# Canonical Cube to Screen

- Irrelevant to z
- Transform in xy plane:  $[-1, 1]^2$  to  $[0, \text{width}] \times [0, \text{height}]$
- Viewport transform matrix:

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

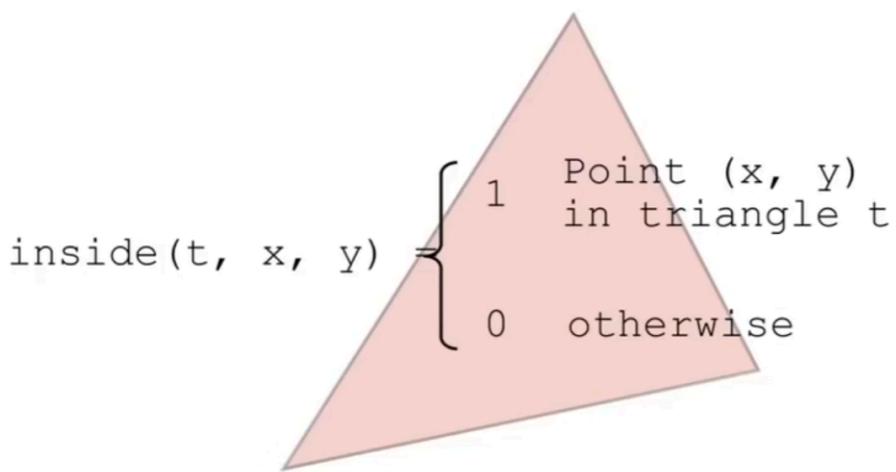
三角形是最基础的多边形。

三角形内部一定是平面的。

通过向量的叉积定义一个点是否在三角形内。

采样法：使用一个连续的函数，在一个离散的集合中询问该集合元素对应的函数值，得出的值的集合。（将函数离散化的过程）

如：屏幕空间中，不同像素的中心是否在需要渲染的三角形内



Aliasing 走样 锯齿

所以需要反走样，抗锯齿

滤波：去掉特定频率的信息

对于一个特定的图像，变化大的频率高，变化小的频率低。

**【高通低通带通】**

卷积：对于某个窗口的特定区域的数，进行一系列的规则运算后得到的数写回该区域中（如一维卷积中，对一个数组做大小为三的卷积）

# 卷积核... Convolution

卷起来  
滑动窗口协议, 以  
交叉相

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

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

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

Result		3								
--------	--	---	--	--	--	--	--	--	--	--

AVIN 滑动窗口协议  
Convolution  
kernel 卷积核  
加权平均  
10年的脑血栓清通了

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							
--------	--	---	---	--	--	--	--	--	--	--

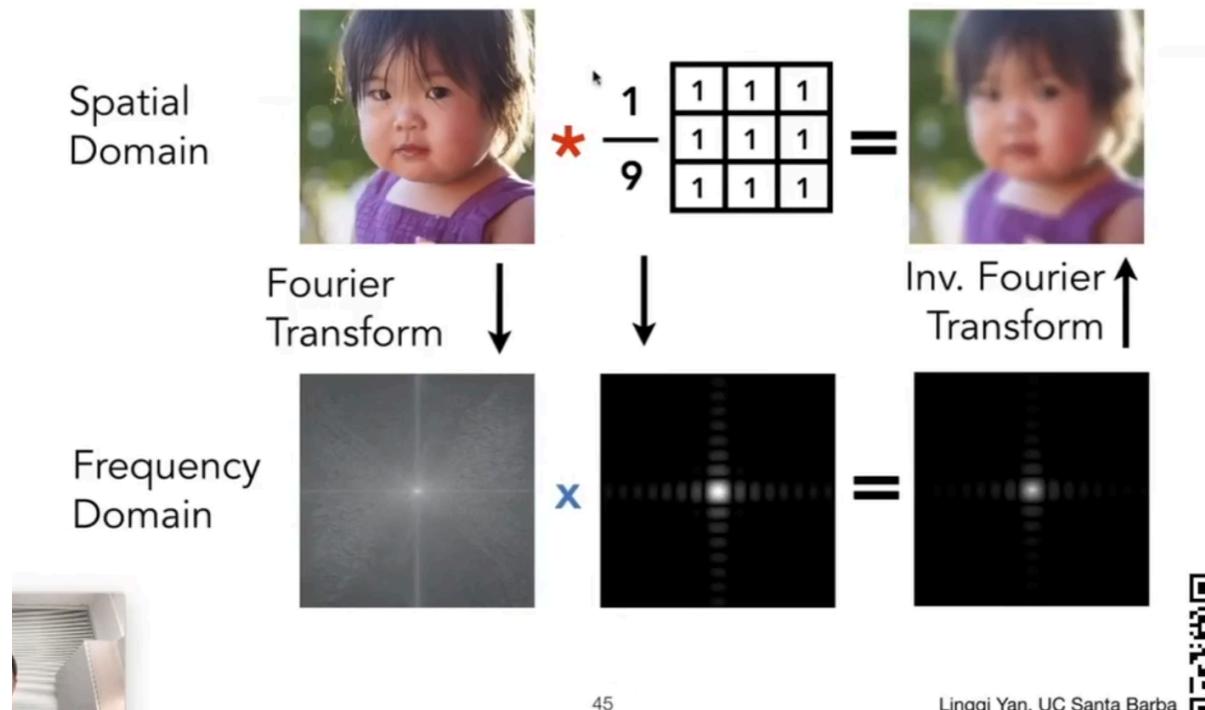
做模糊的两种方案：

对原图像取平均卷积

对原图进行傅里叶变换后，乘以卷积网络的傅里叶变换值，再逆变换回去

(时域的卷积 = 频域的乘积)

## Convolution Theorem



45

Lingqi Yan, UC Santa Barbara

box filter 盒子滤波器 (也是低通滤波器) (模糊)

盒子越大 频域图 (白色区) 越小 , 可以这么理解: 因为  
边界大 (多) 了。

也可以这么理解: 对某块区域做滤波, 滤波区越大, 滤后平均的范围就越大, 就只能留下更低的频率, 所以越模糊; 而盒子越小, (如用一个像素去滤波, 等于没滤)

采样就是在重复频域上的内容

用冲击函数对一块连续区域的函数进行采样, 得到一系  
列的点。

在频域上解释, 就是将一块区域复制粘贴到频域中。

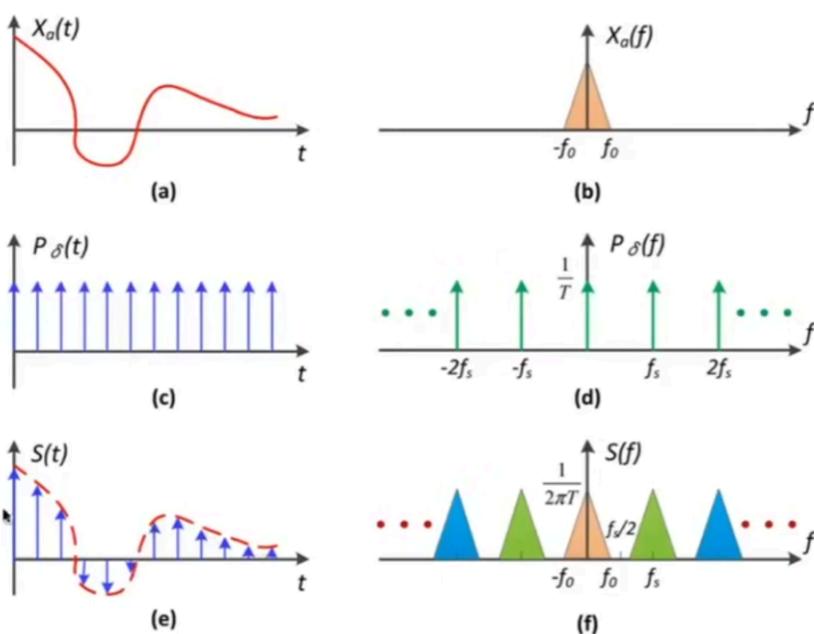
# Sampling = Repeating Frequency Contents

苏有朋老师讲得好啊

记忆回路

冲激

梳妆函数

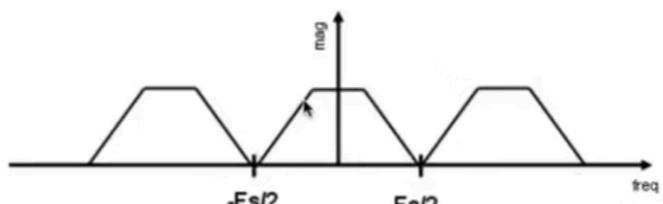


[https://www.researchgate.net/figure/The-evolution-of-sampling-theorem-a-The-time-domain-of-the-band-limited-signal-and-b\\_fig5\\_301556095](https://www.researchgate.net/figure/The-evolution-of-sampling-theorem-a-The-time-domain-of-the-band-limited-signal-and-b_fig5_301556095)

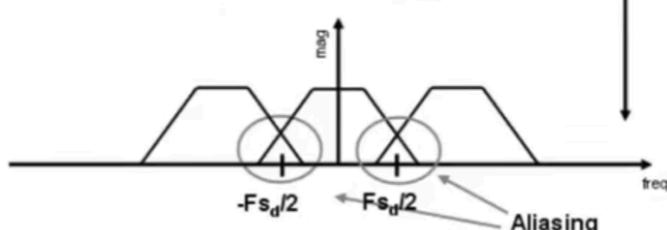
采样率不足时会造成频谱混叠

# Aliasing = Mixed Frequency Contents

Dense sampling:



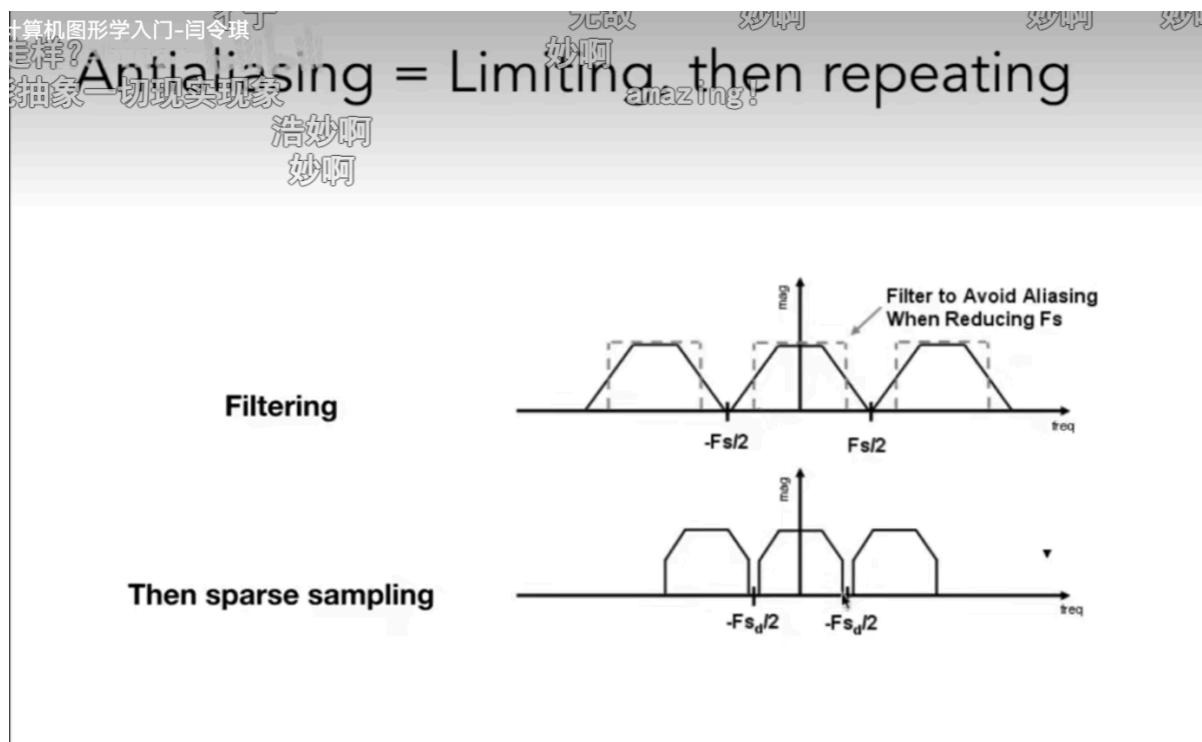
Sparse sampling:



反走样：先做模糊，再做采样

先进行低通滤波，将高频信息去掉，再进行原来的频率的采样（采样频率不变时）

另一种方案是提高分辨率，即提高采样频率

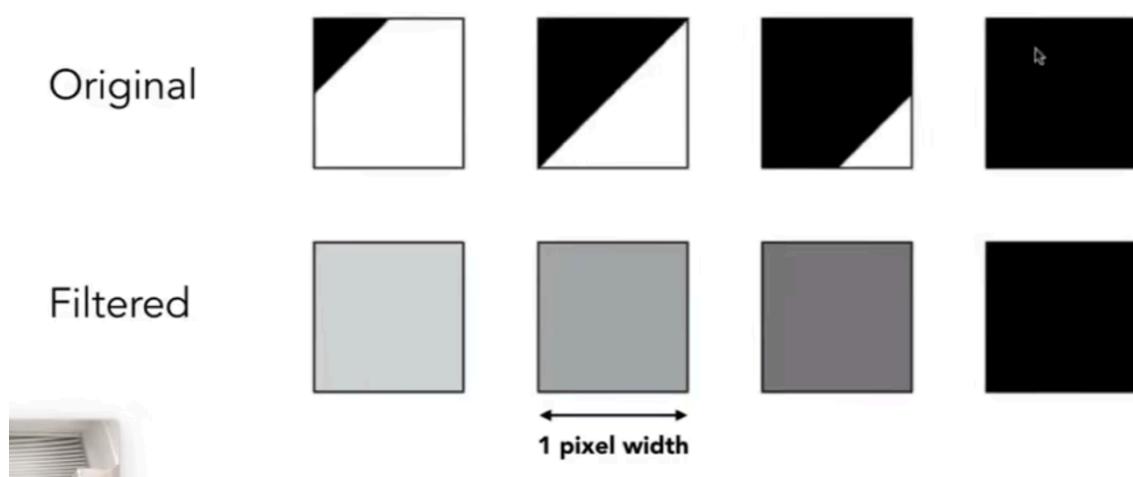


用一定大小的低通滤波器（即对图像进行卷积。）

用一个像素的box filter进行滤波，对于边界的情况可虑成灰值

## Antialiasing by Computing Average Pixel Value 和我想的

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.

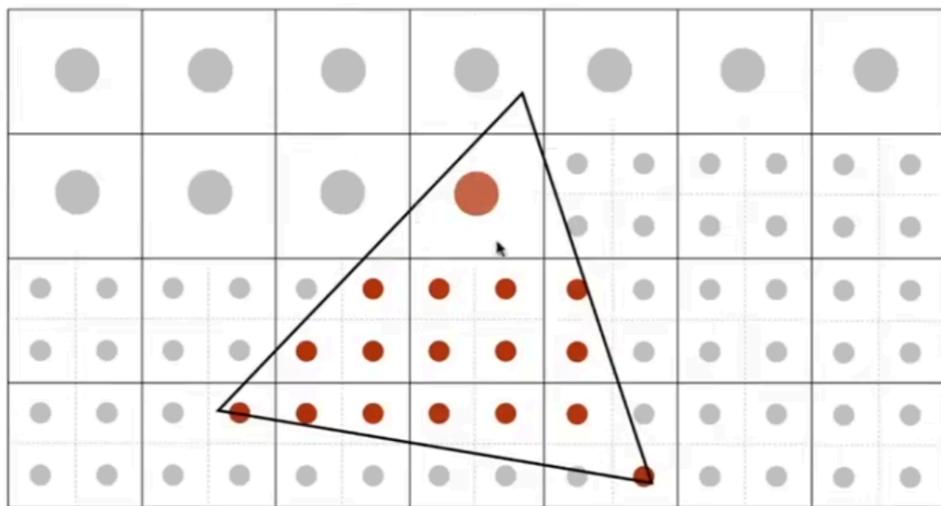


## 抗锯齿效果

用多像素box filter进行时，求覆盖率 MSAA

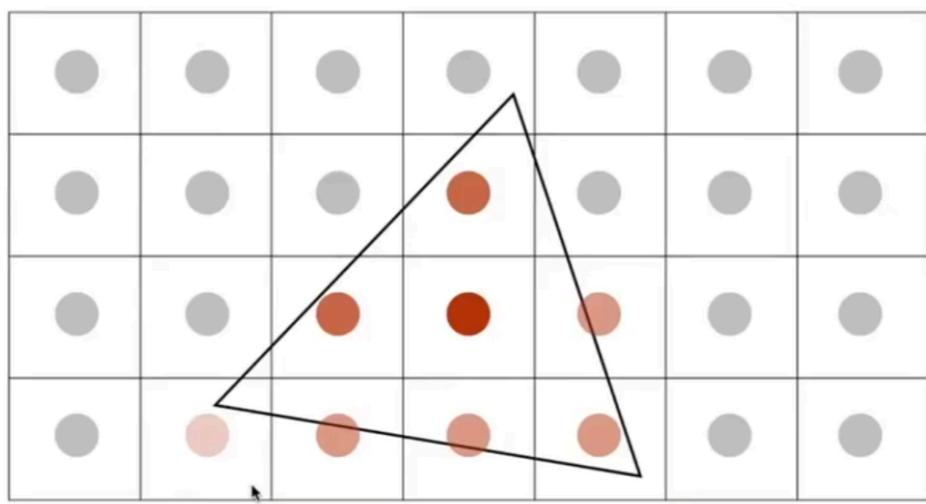
(模糊操作)

Average the NxN samples “inside” each pixel.



一个颜色用了更高的采样密度  
Supersampling: Step 2  
八叉树 颜色减弱吗  
每个像素里本身不就应该是颜色一样的嘛？

Average the NxN samples “inside” each pixel.



This is the corresponding signal emitted by the display

				75%			
		100%	100%	50%			
	25%	50%	50%	50%			

再进行采样

可查 抖动分布采样 分时抗锯齿

MSAA

FXAA (图像后期处理，得到锯齿的图，找到边界，并替换边界 (边缘检测) )

TAA (时间相关，相邻两帧的点，复用上一帧的点进行的操作)

超分辨率

都是解决样本不足的问题

DLSS

# Antialiasing Today

No free lunch!

- What's the cost of MSAA?

Milestones (personal idea)

- FXAA (Fast Approximate AA)
- TAA (Temporal AA)

Super resolution / super sampling

- From low resolution to high resolution
- Essentially still "not enough samples" problem
- DLSS (Deep Learning Super Sampling)



## 可见度和遮挡问题

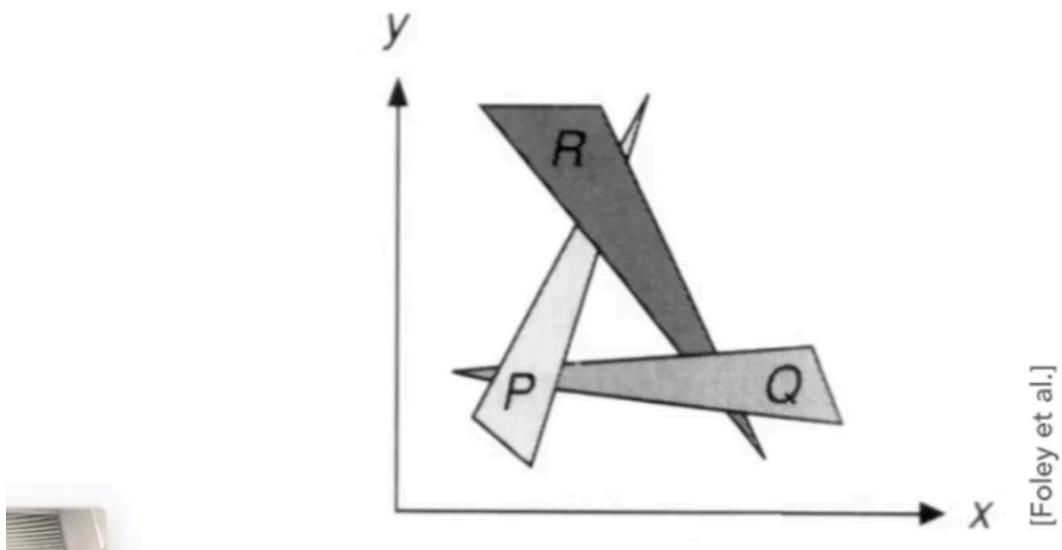
画家算法：排序，然后由远及近进行绘制。

会存在循环覆盖问题

# Painter's Algorithm 苹果应用商店图标

Requires sorting in depth ( $O(n \log n)$  for  $n$  triangles)

Can have unresolvable depth order



[Foley et al.]

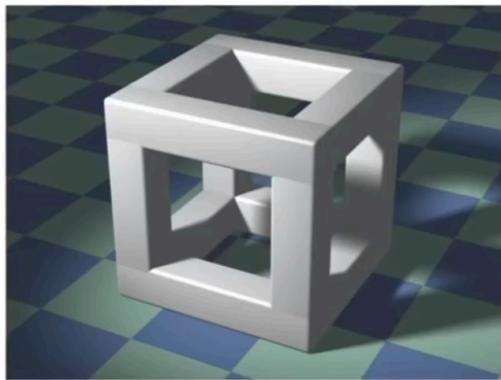
深度缓冲

Z-Buffer

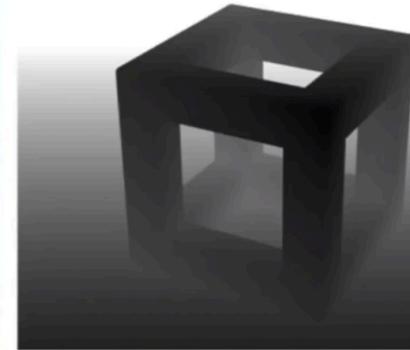
对每个像素存一个深度缓存。

在渲染时，同时生成深度缓存。同步生成。

Image source: Dominic Alves, flickr.



Rendering



Depth / Z buffer



先认为所有像素的深度无限远，然后对任意三角形覆盖的所有像素与深度缓存中对应的值进行比较，取小更新，并在渲染区绘制。

就像找数组最大值时的cur\_max

米奥啊

## Z-Buffer Algorithm

秒啊 因为很简单。。

Initialize depth buffer to  $\infty$ 

During rasterization:

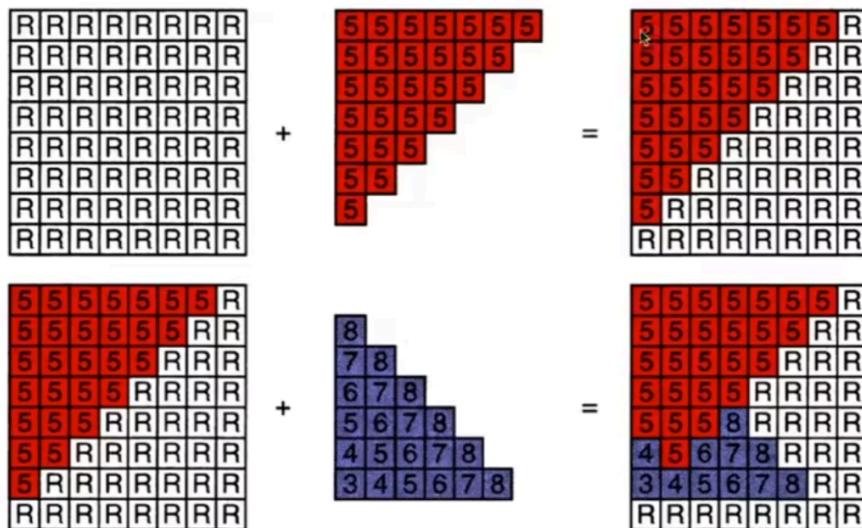
```

for (each triangle T)
    for (each sample (x,y,z) in T)
        if (z < zbuffer[x,y])           // closest sample so far
            framebuffer[x,y] = rgb;      // update color
            zbuffer[x,y] = z;           // update depth
        else
            ;                         // do nothing, this sample is occluded
    
```



## Z-Buffer Algorithm

应该是通过三角形三个顶点记录的深度信息线性插值得到的



## Z-Buffer Complexity

### Complexity

- $O(n)$  for  $n$  triangles (assuming constant coverage)
- How is it possible to sort  $n$  triangles in linear time?

深度缓存算法跟顺序无关

对MSAA的结果做深度缓存。

Z-Buffer对透明物体处理不了，需要进行特殊处理。

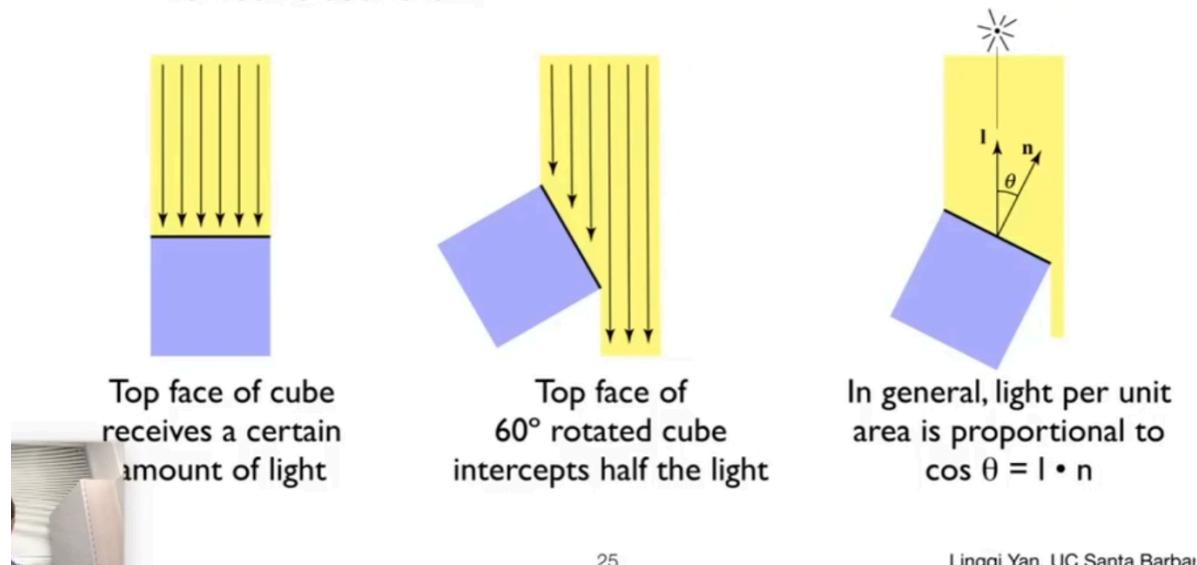
### 着色 Shading

漫反射：接受到的能量 跟 法线方向与光线方向的夹角余弦有关

# Diffuse Reflection

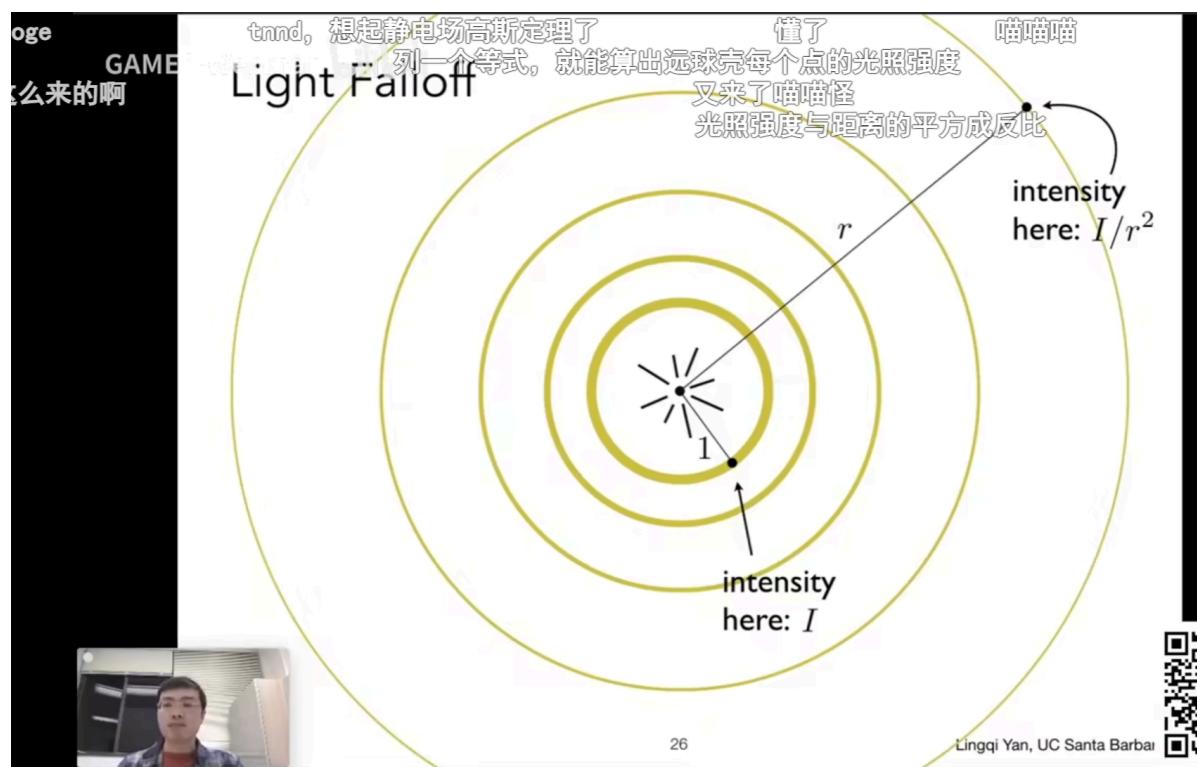
- But how much light (energy) is received?

- Lambert's cosine law



25

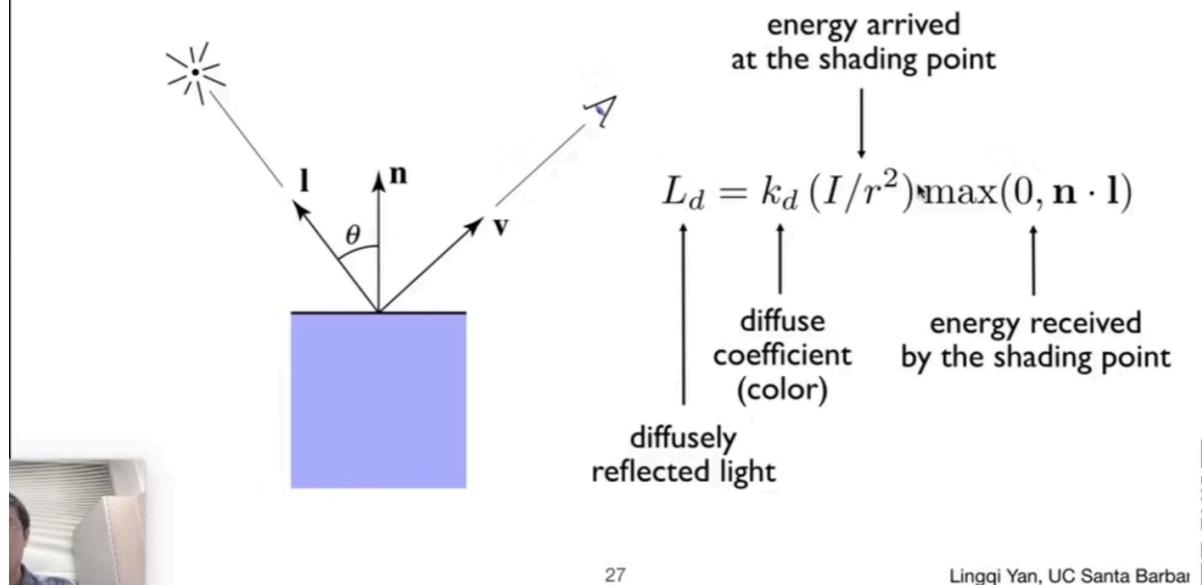
光强与距离相关，在同距离处的光的能量相同，但由于周长原因，强度越来越小。



最终漫反射的公式如下：

# Lambertian (Diffuse) Shading

Shading **independent** of view direction



27

Lingqi Yan, UC Santa Barbara

$k_d$  表示为一个 Color，跟物体贴图颜色相关，就可以定义漫反射的颜色。

漫反射跟观察角度无关，与物体面的方向相关。

高光

# Specular Term (Blinn-Phong)

妙

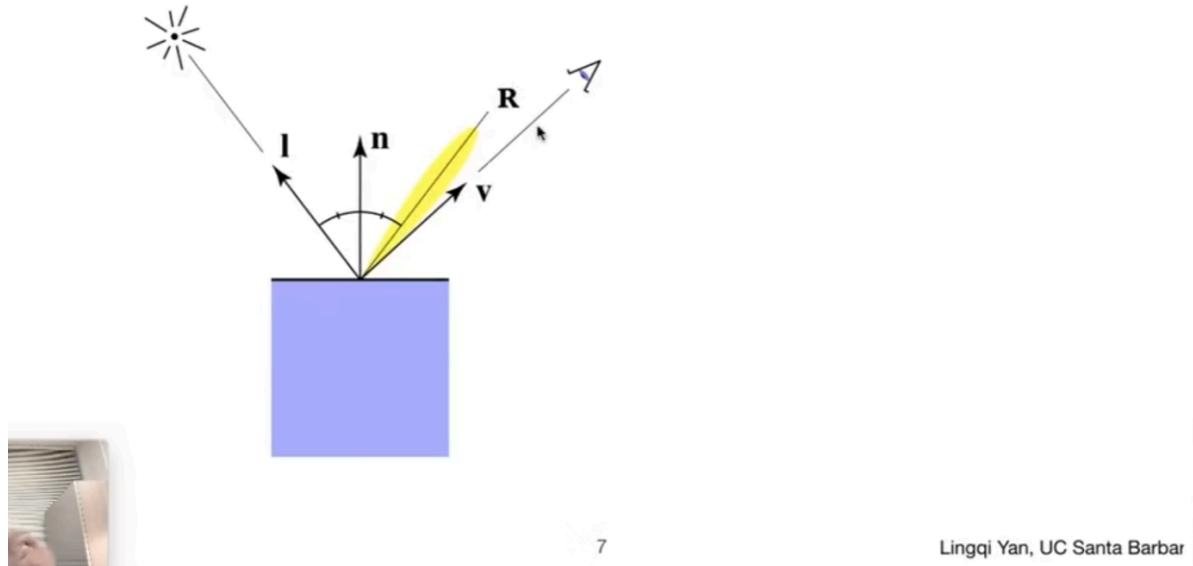
妙啊

妙啊

妙啊

Intensity **depends** on view direction

- Bright near mirror reflection direction



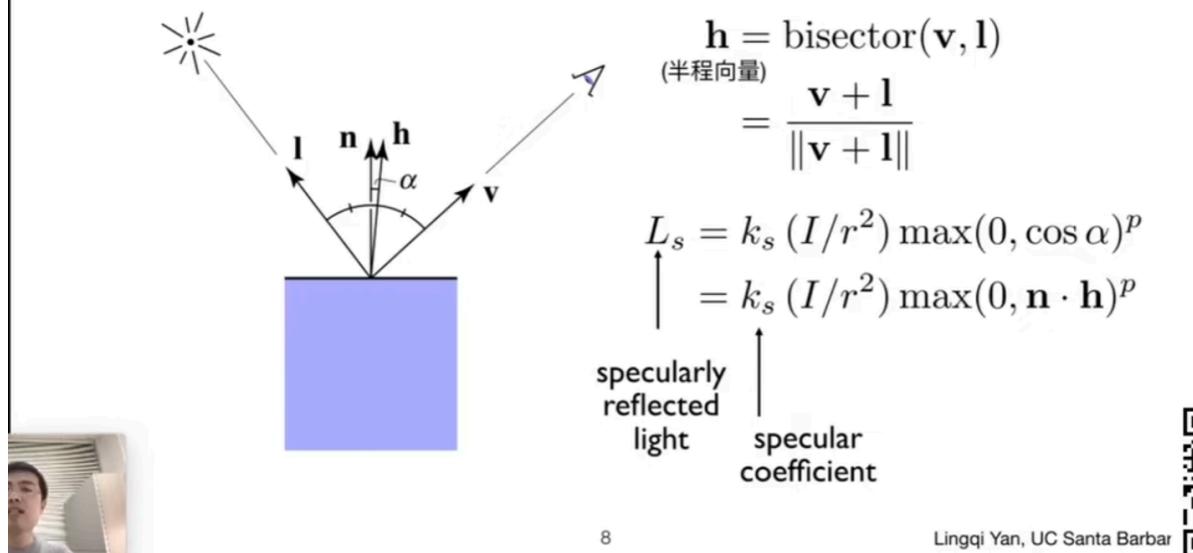
高光项



## Specular Term (Blinn-Phong)

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

- Measure "near" by dot product of unit vectors



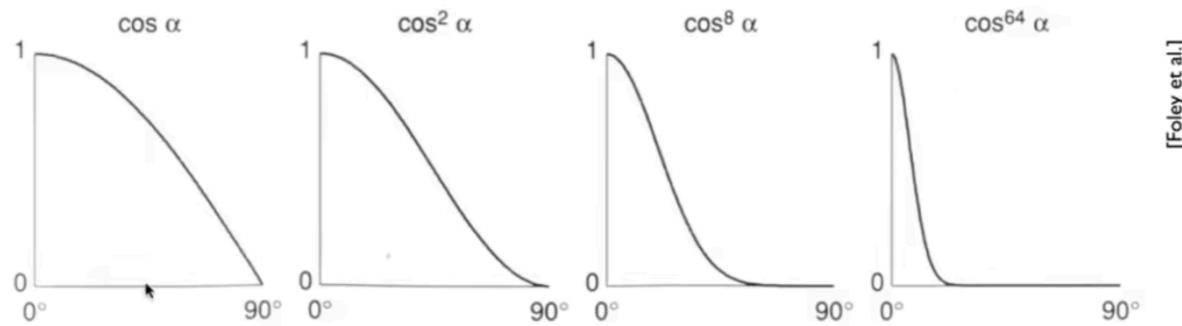
当v和R接近时， n和h一定接近。

使用点乘来判断

镜面反射系数一般为白色，用夹角余弦的指数来生成高光（高光性质）。

## Cosine Power Plots

Increasing  $p$  narrows the reflection lobe



[Foley et al.]

## 环境光

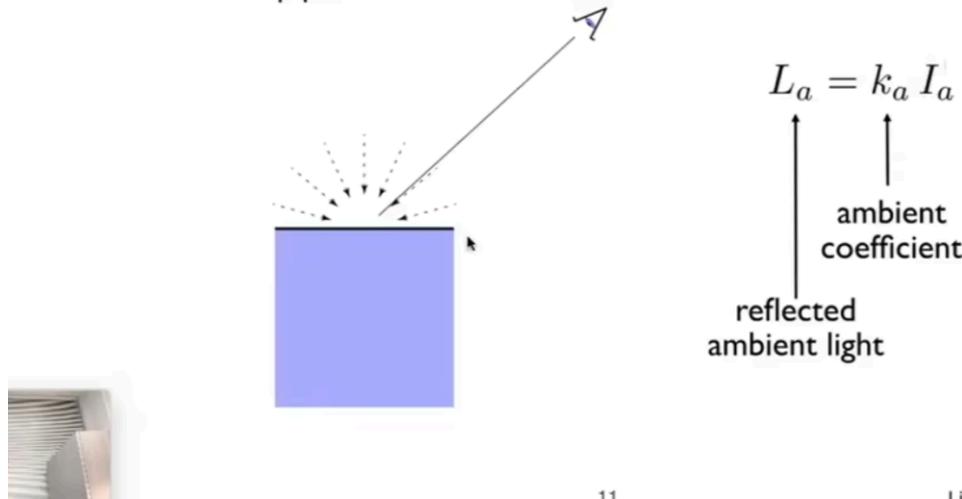
该模型下，从环境射进摄像机的值，保证所有地方都能看到，是一个常数。

更细致的，需要考虑全局光照。

# Ambient Term

Shading that does not depend on anything

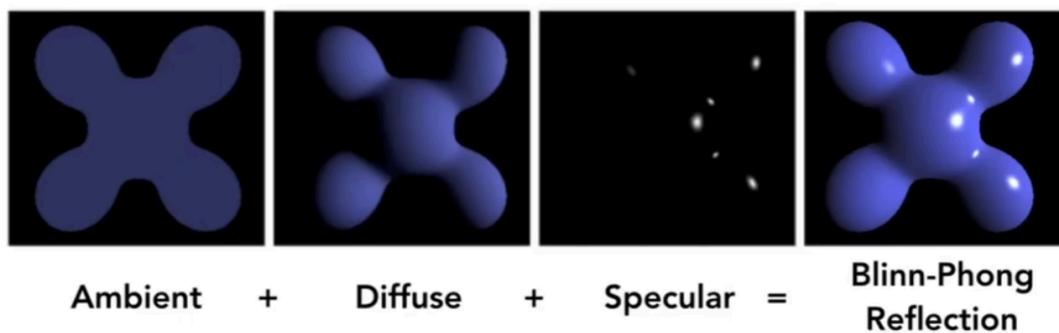
- Add constant color to account for disregarded illumination and fill in black shadows
- This is approximate / fake!



11

Lingqi Yan, UC Santa Barbara

ader的ambient常数项写成零了！  
太暗了！真太牛逼了！



$$\begin{aligned}
 L &= L_a + L_d + L_s \\
 &= k_a I_a + k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p
 \end{aligned}$$



12

Lingqi Yan, UC Santa Barbara



bling-feng 模型是经验模型。

## 着色细分

Flat shading 使用每个三角面同向法，三角形各顶点颜色一致，内部无颜色变化

计算机图形学入门-闫令琪

### Shade each triangle (flat shading)

Flat shading

- Triangle face is flat — one normal vector
- Not good for smooth surfaces



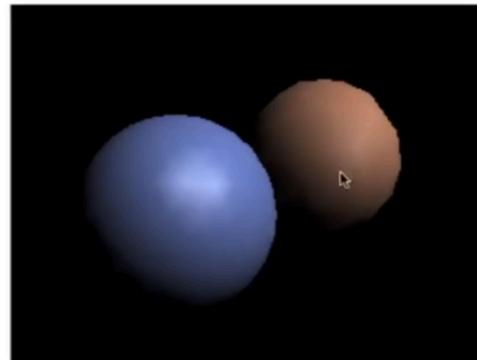
## Gouraud Shading

三角形每个顶点有颜色，在内部对这个三角形的颜色做插值。

## Shade each vertex (Gouraud shading)

### Gouraud shading

- Interpolate colors from vertices across triangle
- Each vertex has a normal vector (how?)



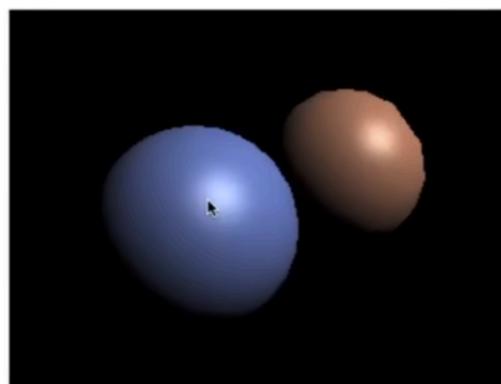
### Phong shading

逐像素着色

## Shade each pixel (Phong shading)

### Phong shading

- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel
- Not the Blinn-Phong Reflectance Model



逐顶点的法线：顶点的法线为它所被共面的法线的平均  
(简单平均、加权平均都可)。

## Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry

- e.g. consider a sphere

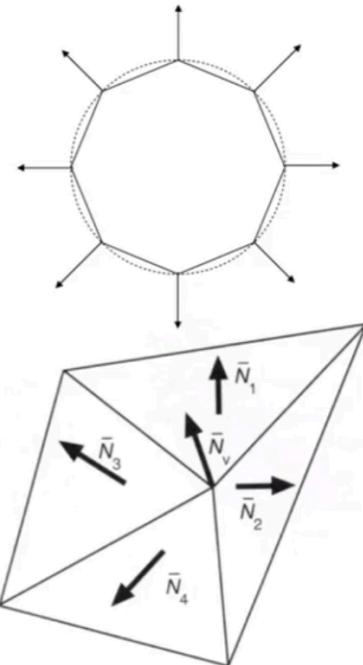
Otherwise have to infer vertex normals from triangle faces

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

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



20



Lingqi Yan, UC Santa Barbara



逐像素的法线：(与重心坐标相关，从一个顶点到另一个顶点的法线方向平均变化)

///=====

着色器练习

Inigo Quilez



Procedurally modeled, 800 line shader.  
<http://shadertoy.com/view/Id3Gz2>

====

## 纹理映射

对于漫反射中的kd。可以通过漫反射映射出去。

三维物体的表面是二维的，可以与一个图有一一对应的关系。

Surfaces are 2D

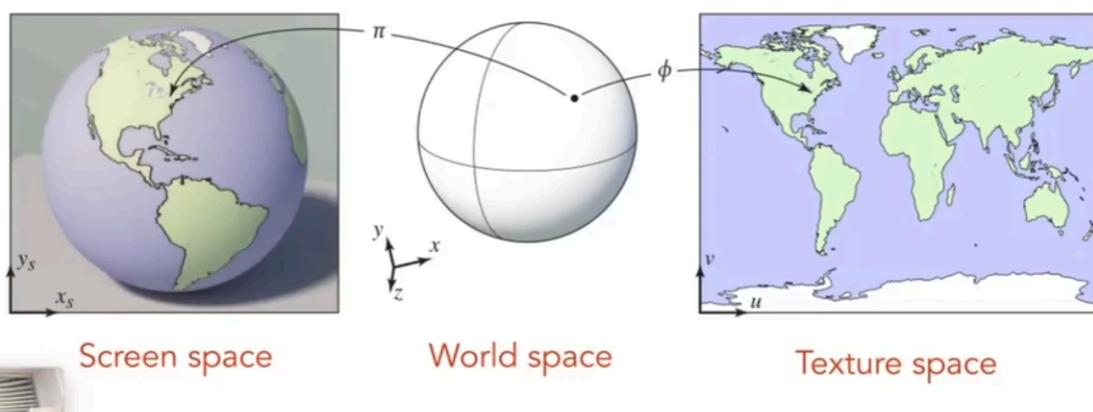
标注UV的是不是比较合理解这一部分0.0

微分几何

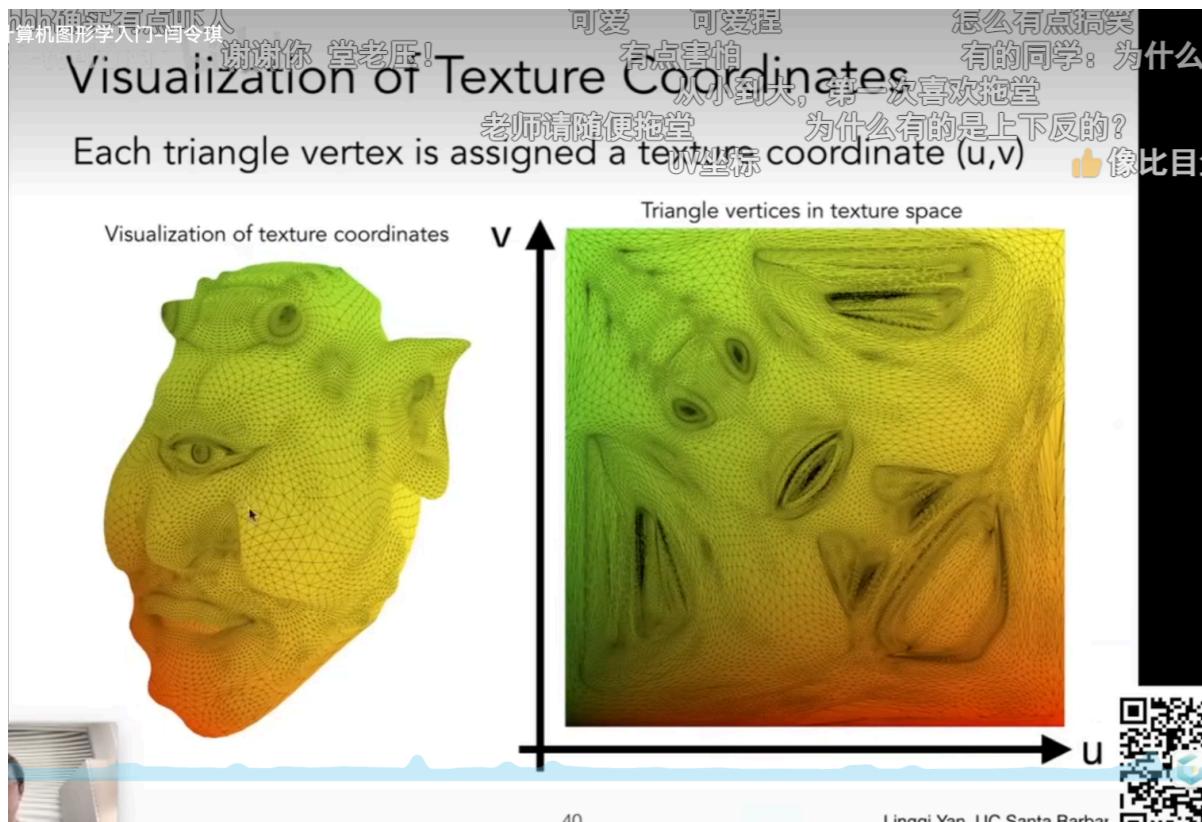
微分流形

Surface lives in 3D world space

Every 3D surface point also has a place where it goes in the 2D image (**texture**).



纹理坐标：u-v, 范围[0,1]



在三角形内部进行插值

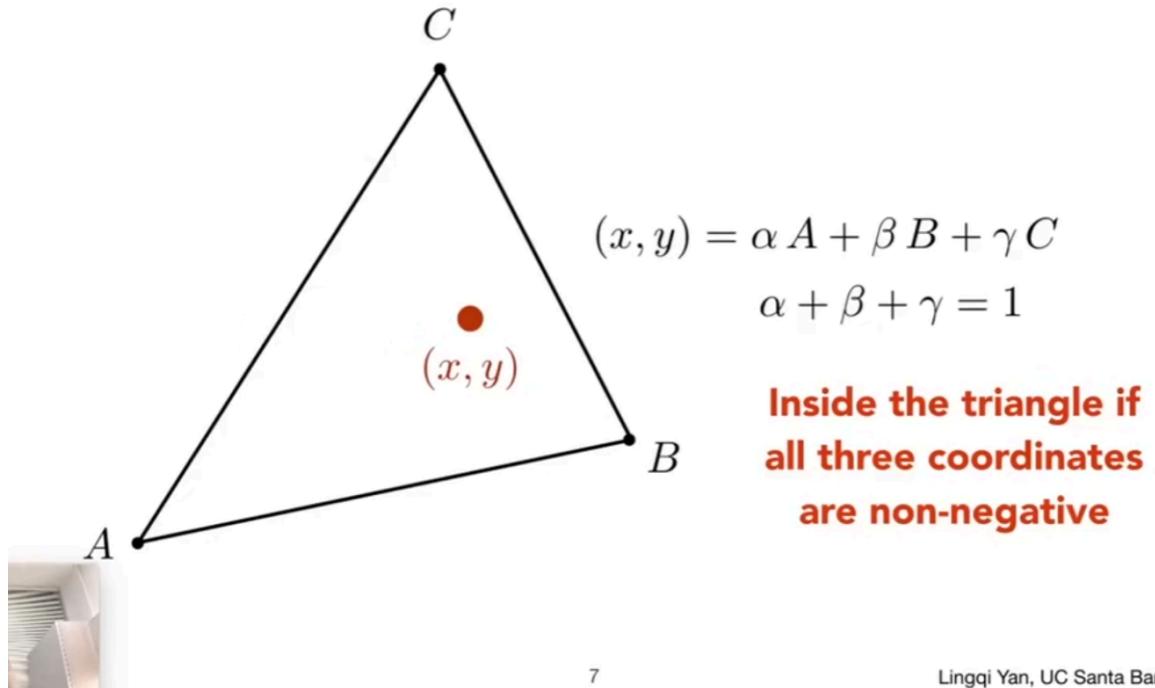
引入重心坐标

在知道顶点的情况下，知道三角形内任意一点的值。

点在三角形内，必须要 $\alpha$ 等都非负。

# Barycentric Coordinates

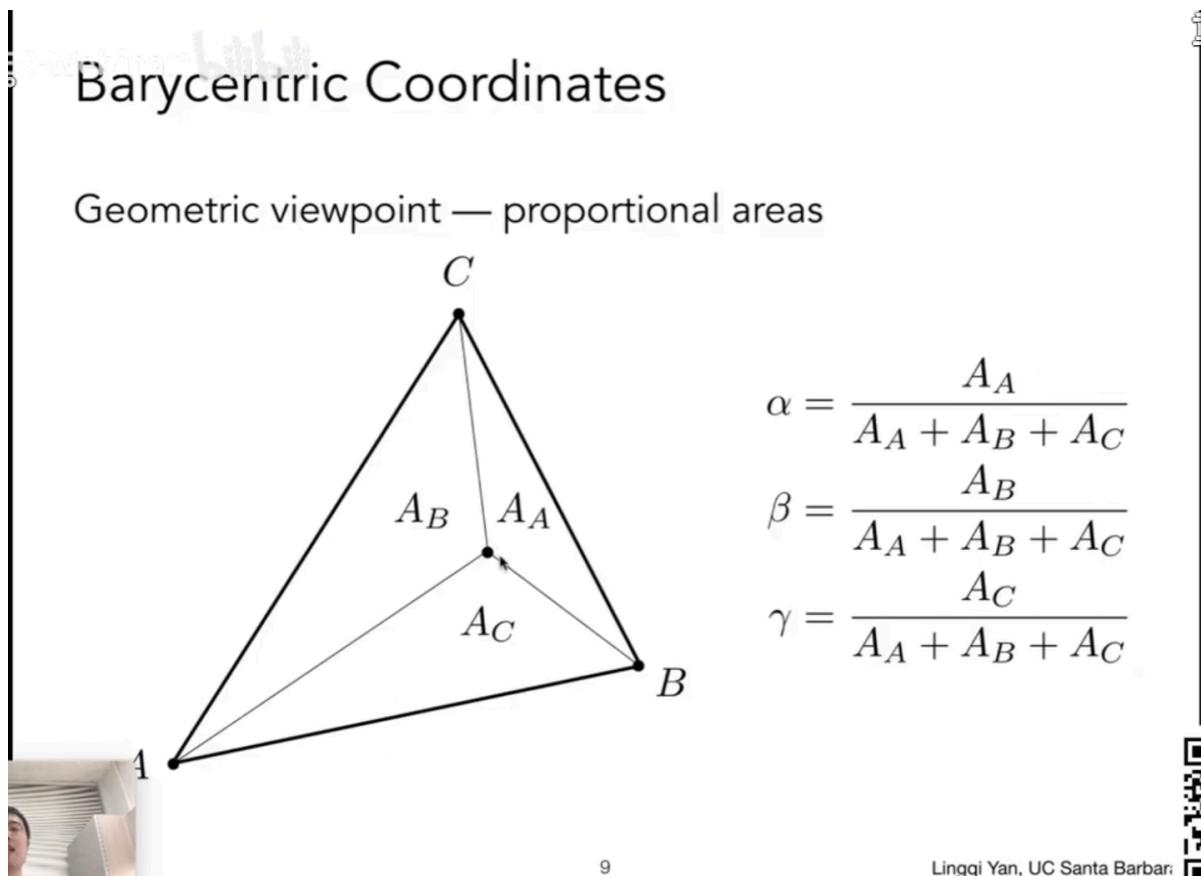
A coordinate system for triangles  $(\alpha, \beta, \gamma)$



7

Lingqi Yan, UC Santa Barbara

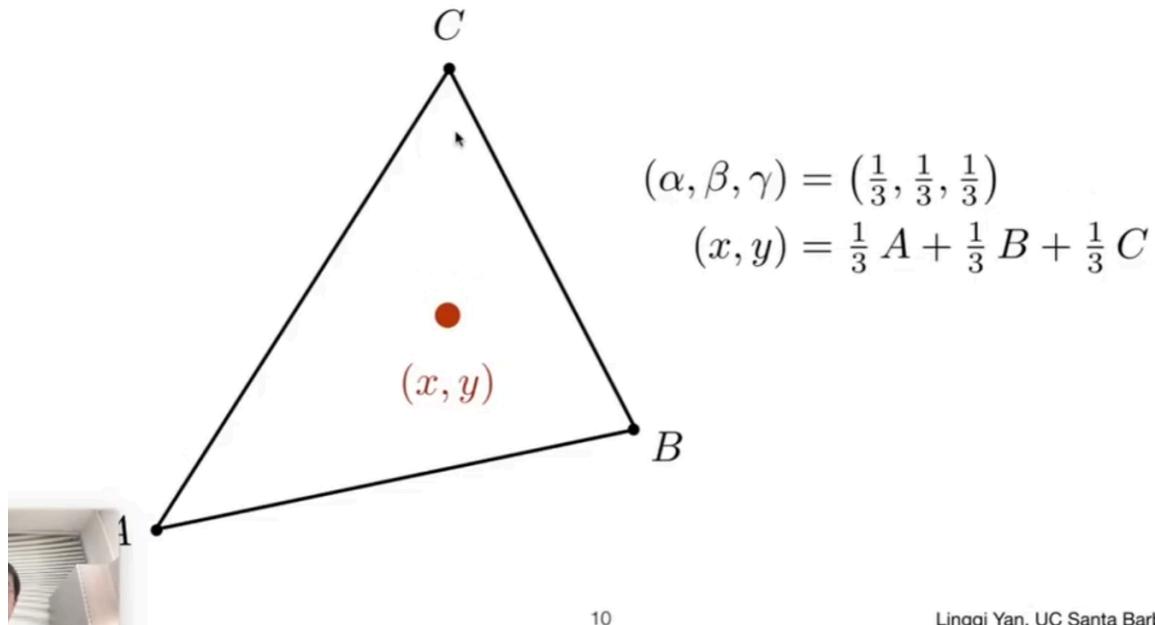
可从面积比求出任意点坐标的重心坐标系数



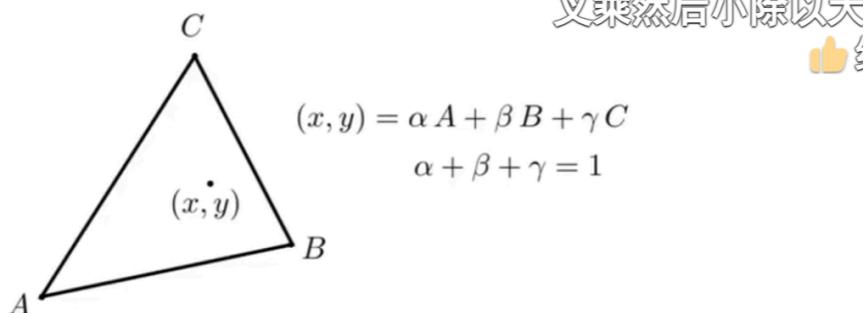
9

Lingqi Yan, UC Santa Barbara

What's the barycentric coordinate of the centroid?



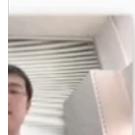
Barycentric Coordinates: 公式啊 刚刚去恶补了一下二维向量叉乘，回来了  
这个写个函数封装一下就行了，只传递顶点坐  
叉乘然后小除以大



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

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

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



重心坐标不具有投影不变性，所以应先做三维插值，再投影。

# Simple Texture Mapping: Diffuse Color

for each rasterized screen sample  $(x, y)$ :

$(u, v) = \text{evaluate texture coordinate at } (x, y)$

`texcolor = texture.sample(u, v);`

set sample's color to `texcolor`;

Usually a pixel's center

Using barycentric  
coordinates!

Usually the diffuse albedo  $K_d$

(recall the Blinn-Phong reflectance model)



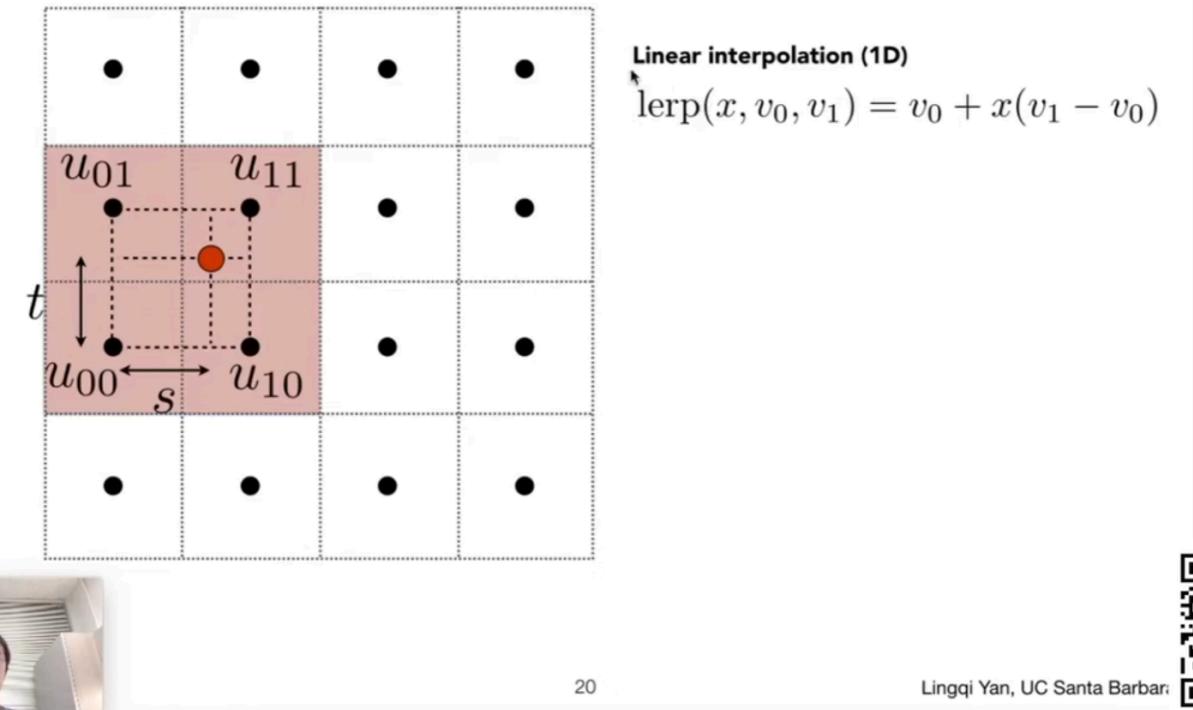
## 需要解决的问题

纹理放大：当纹理过小，像素映射到纹理图的一个非整数区域时，可能会出现多块物体表面像素映射同一个纹理像素的情况。

## 双线性插值

针对映射的像素位置选定需要纳入插值的纹理像素区

# Bilinear interpolation

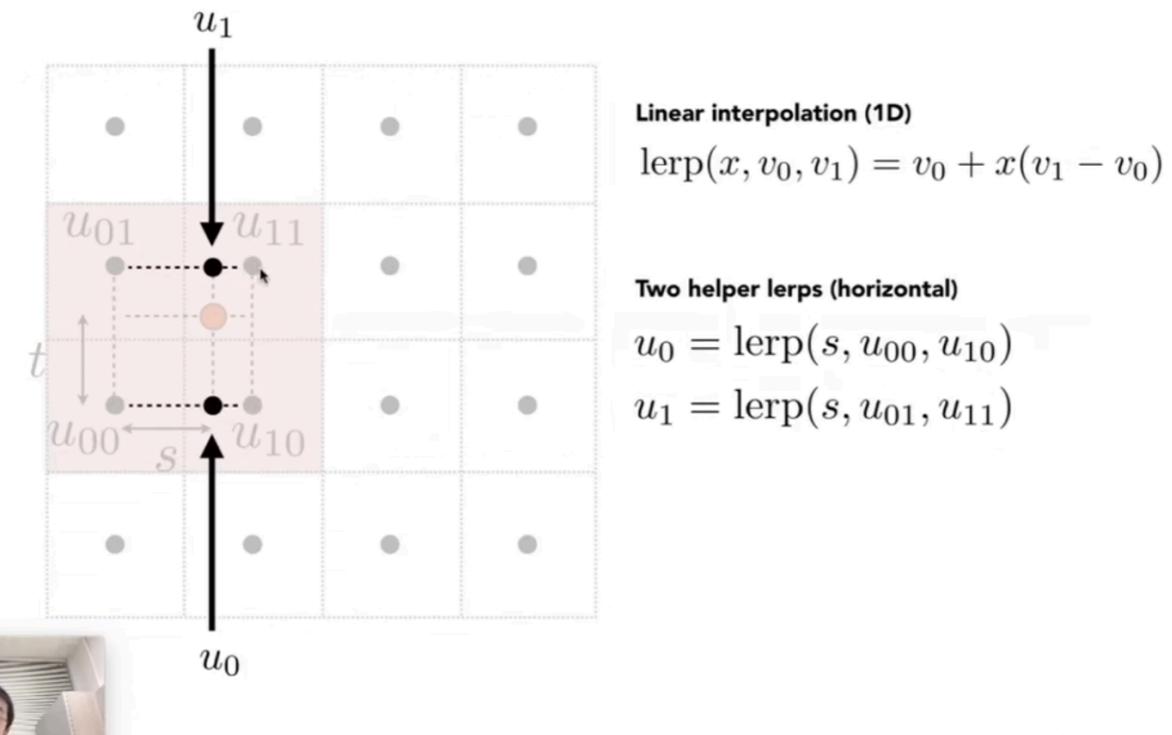


20

Lingqi Yan, UC Santa Barbara

在水平上进行插值

# Bilinear interpolation

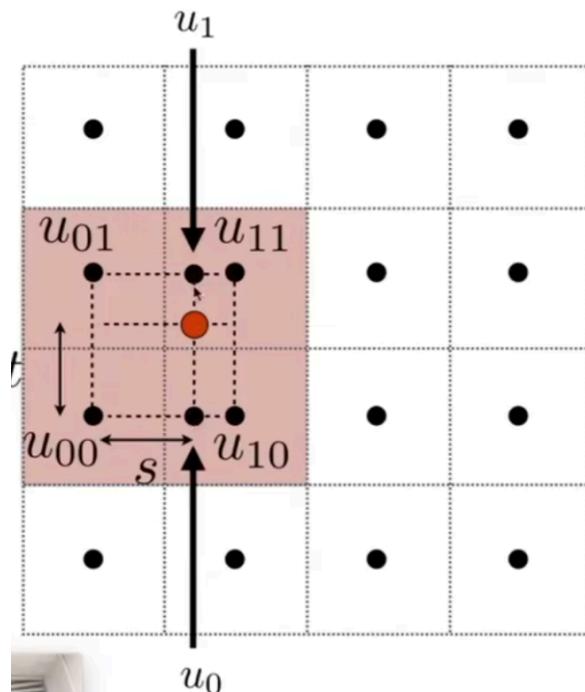


再在竖直上进行插值

# Bilinear interpolation

这么细的地方，处理要这么复杂呀

抗锯齿一个点在不在三角形



Linear interpolation (1D)

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

Two helper lerps

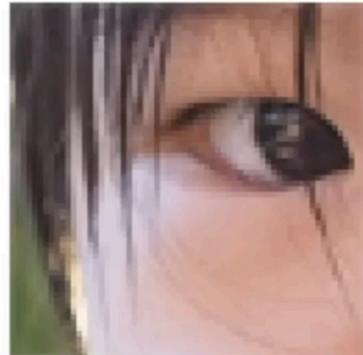
$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

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

Final vertical lerp, to get result:

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

最邻近-双线性 (周围4个) -立方插值 (周围16个)



Nearest



Bilinear

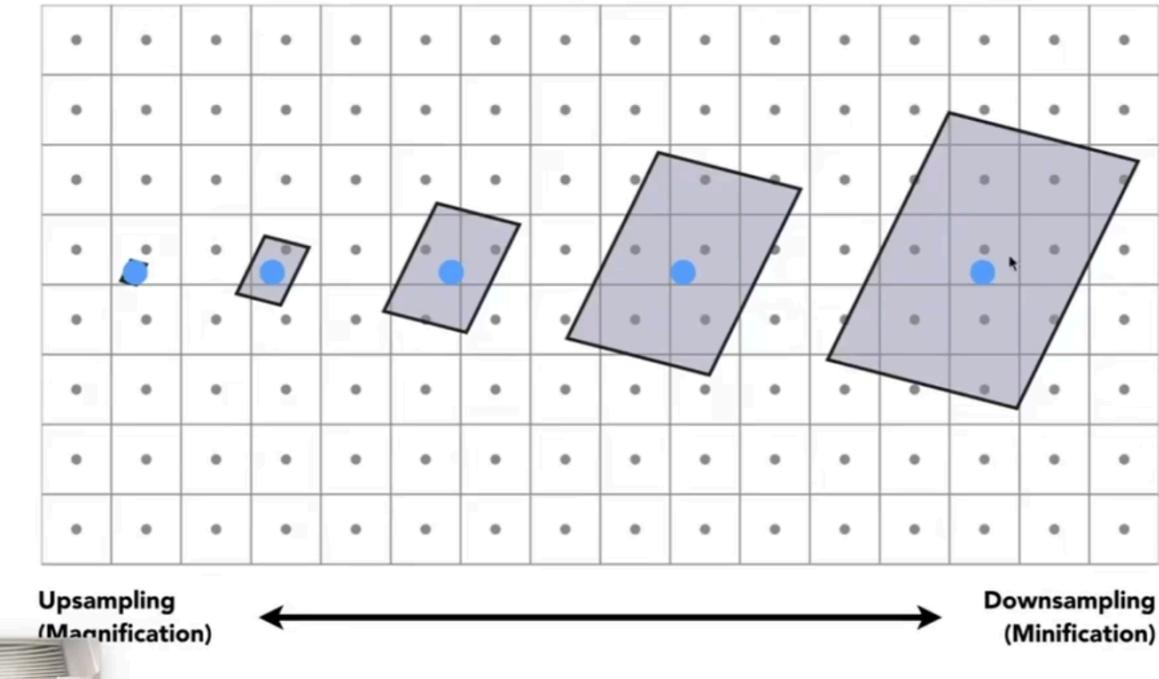


Bicubic

纹理过大时，远处出现摩尔纹，近处出现锯齿 ==> 出现了走样

像素覆盖的纹理区域：

# Screen Pixel “Footprint” in Texture



点查询问题和范围查询问题

一种好方法是直接对该区域进行均值。

Mipmap 也称图像金字塔

## Mipmap

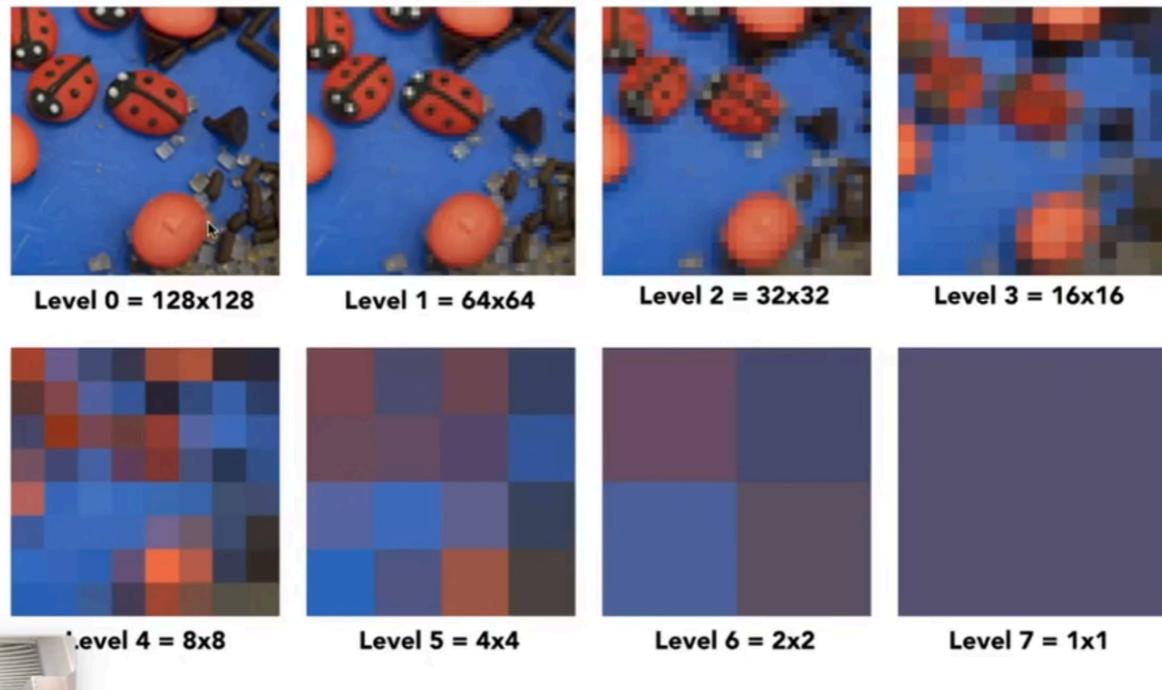
Allowing (*fast, approx., square*) range queries

近似，正方形的查询。

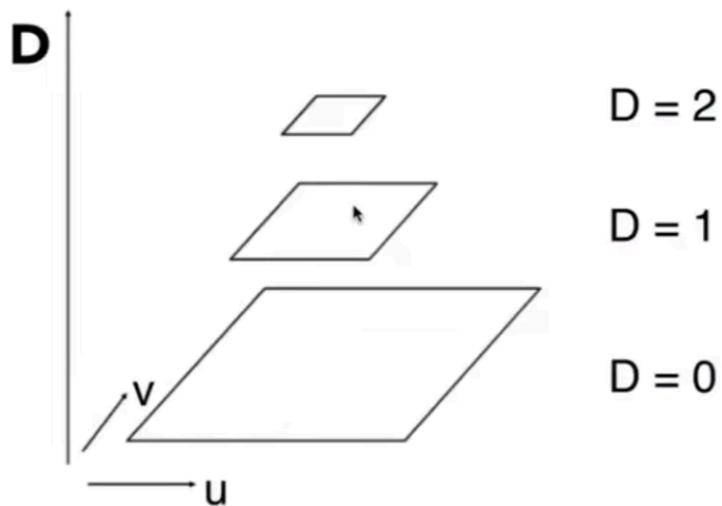
先生成一系列的纹理层组

# Mipmap (L. Williams 83)

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



## Mipmap (L. Williams 83)



"Mip hierarchy"

level = D

What is the storage overhead of a mipmap?

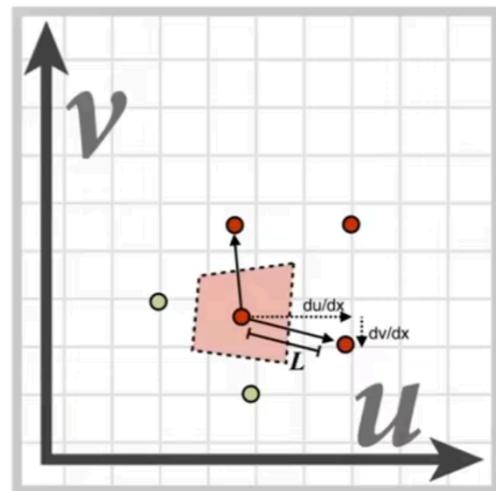
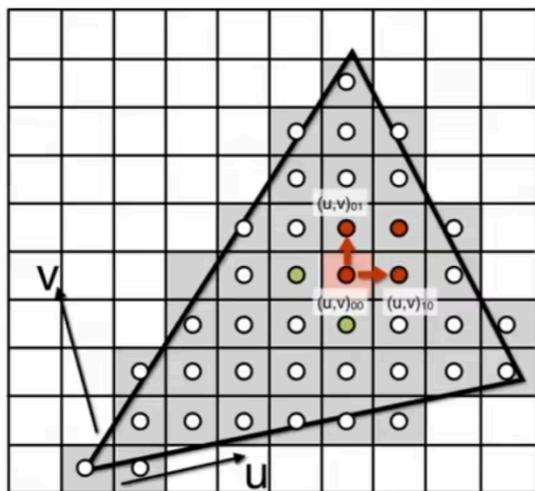


引入的存储量多三分之一。  $(1+1/4+1/16) \dots = 4/3$

对屏幕像素所在区域映射到纹理上的对应区域：

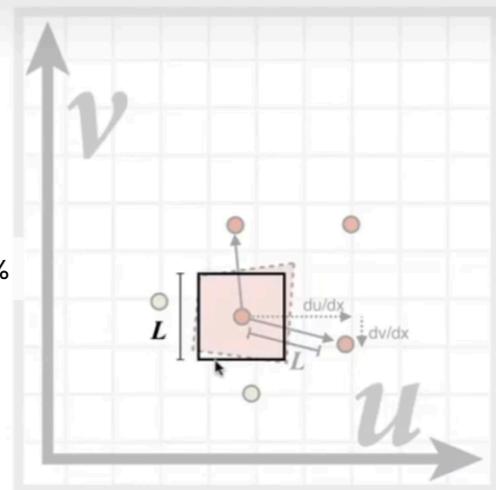
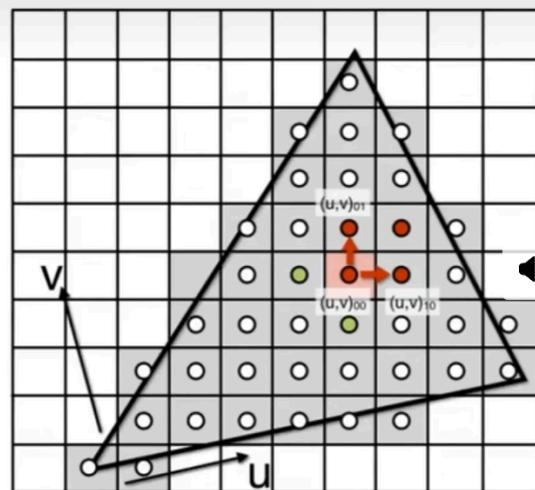
## 这里的 $du$ 、 $dv$ 和 $L$ 分别是什么？Level D

分辨率不一样



$$D = \log_2 L \quad L = \max \left( \sqrt{\left( \frac{du}{dx} \right)^2 + \left( \frac{dv}{dx} \right)^2}, \sqrt{\left( \frac{du}{dy} \right)^2 + \left( \frac{dv}{dy} \right)^2} \right)$$

88%



$$D = \log_2 L \quad L = \max \left( \sqrt{\left( \frac{du}{dx} \right)^2 + \left( \frac{dv}{dx} \right)^2}, \sqrt{\left( \frac{du}{dy} \right)^2 + \left( \frac{dv}{dy} \right)^2} \right)$$

求出该区域在第 $\log_2 L$ 层能变为1个像素大小。

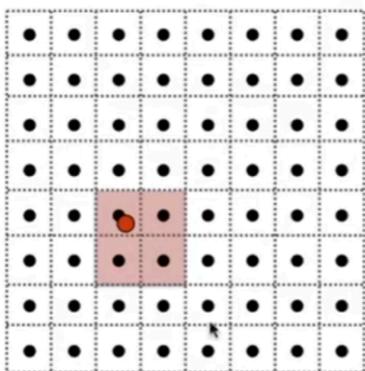
为了让各层之间连续，应该对层与层之间做一次三线性插值。（先双线性插值，然后在层与层间插值）

# Trilinear interpolation

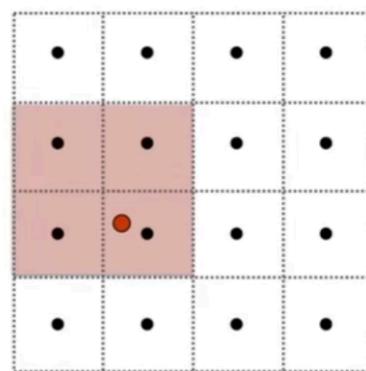
线性插值

十层饼了

什么立体插值，格子  
三线性插值可以用b



Mipmap Level D



Mipmap Level D+1

Bilinear result

Bilinear result

Linear interpolation based on continuous D value



但是，通过三线性插值形成的mipmap会存在远处糊的情况

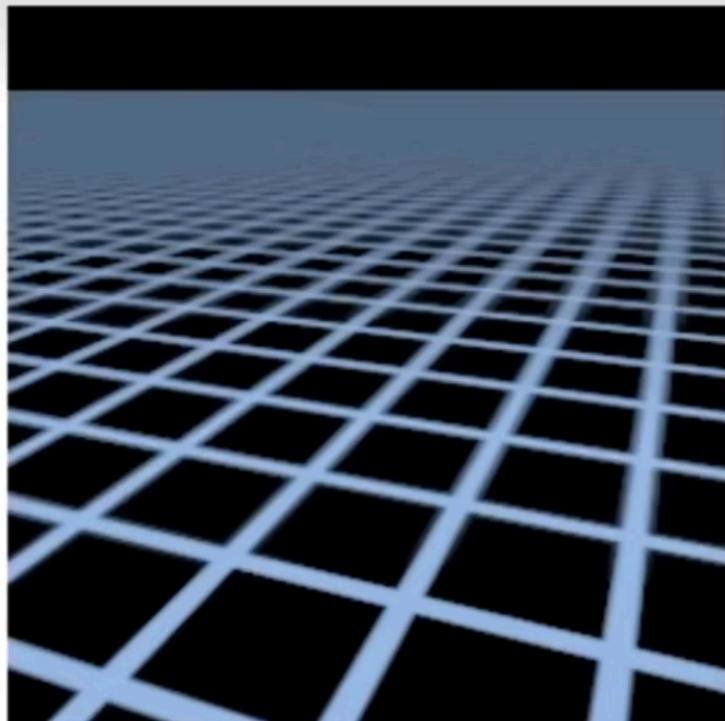
计算机图形学入门-闫令琪

必应文库

## Mipmap Limitations

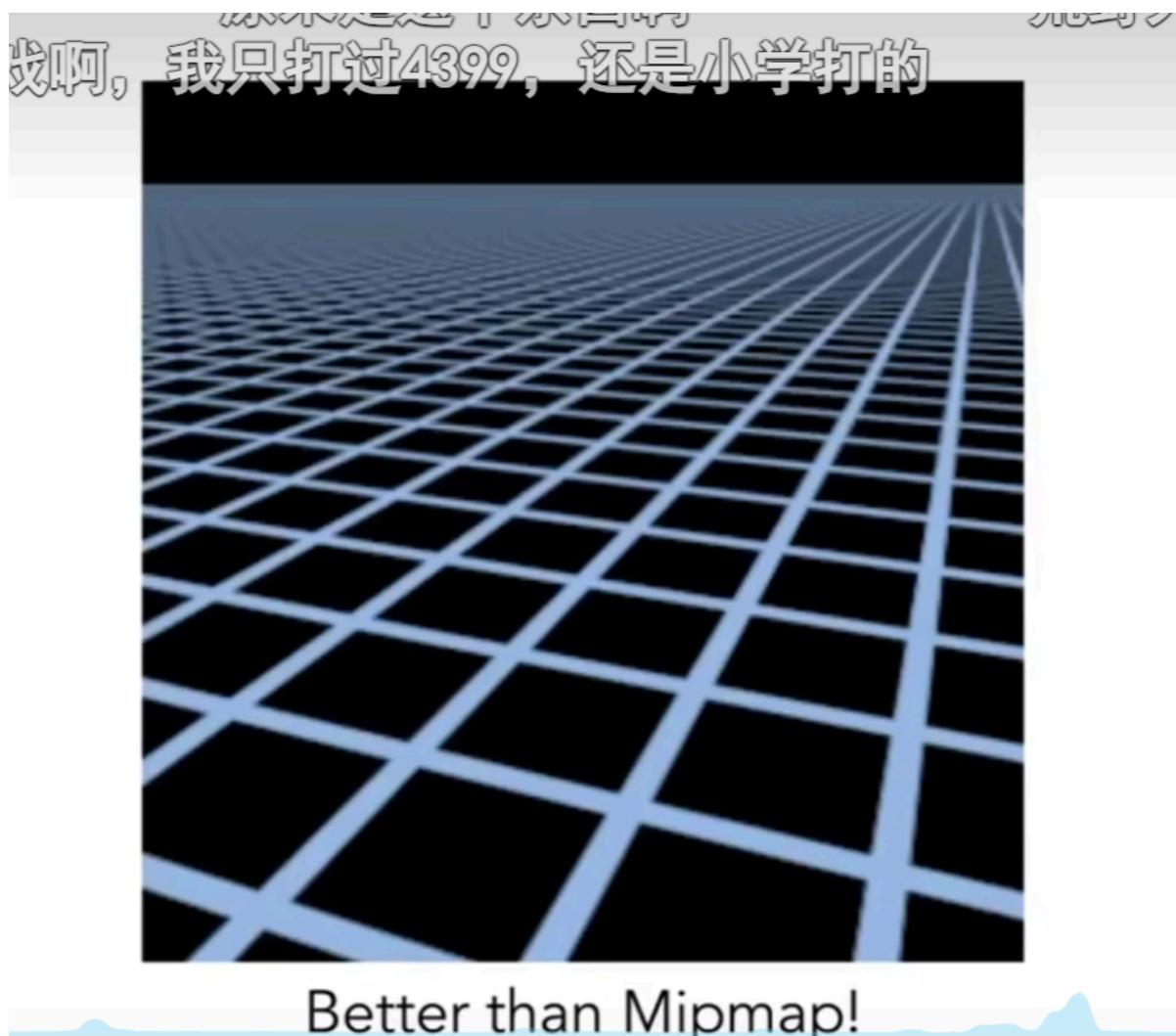
Overblur

Why?



Mipmap trilinear sampling

这时候，可以用各向异性过滤来解决

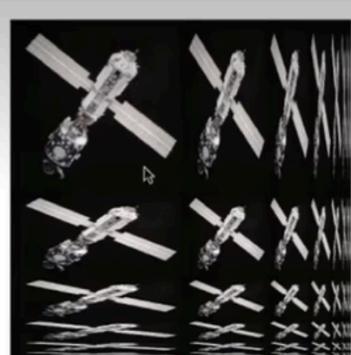


计算机图形学入门-闫令琪

## Anisotropic Filtering

Ripmaps and summed area tables

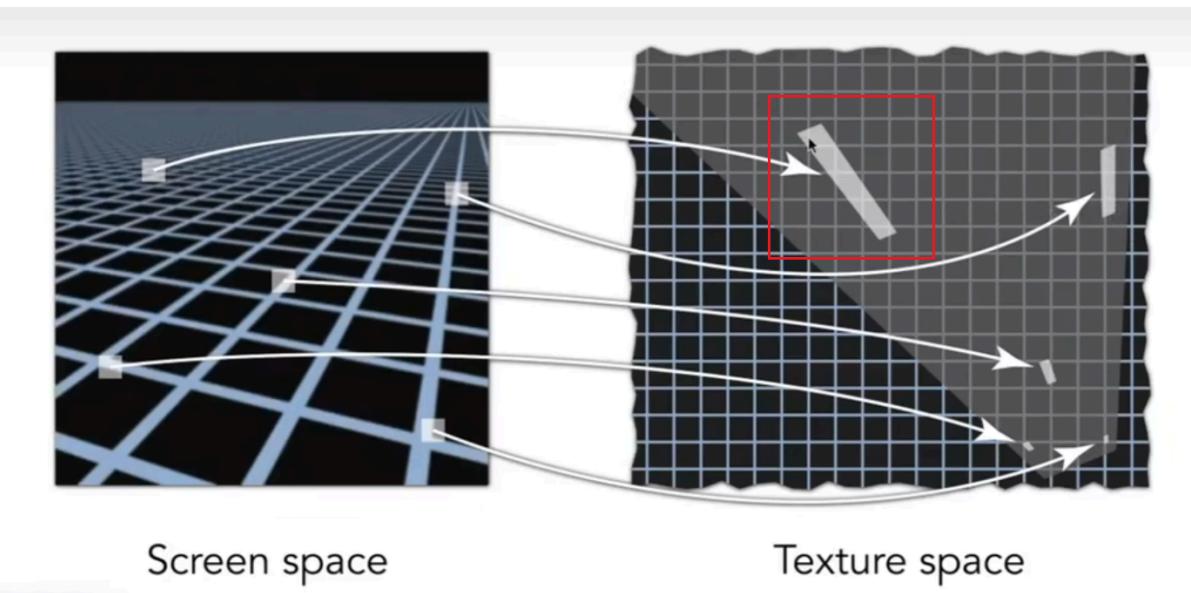
- Can look up **axis-aligned rectangular zones**
- Diagonal footprints still a problem



Wikipedia

对矩形区域进行映射为正方形

但对于斜着的区域不太行

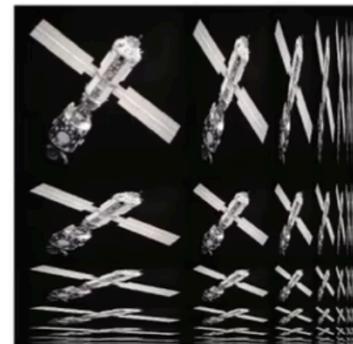


## 使用EWA

### Anisotropic Filtering

Ripmaps and summed area tables

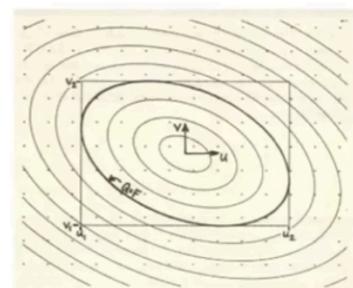
- Can look up axis-aligned rectangular zones
- Diagonal footprints still a problem



Wikipedia

EWA filtering

- Use multiple lookups
- Weighted average
- Mipmap hierarchy still helps
- Can handle irregular footprints



Greene & Heckbert '86

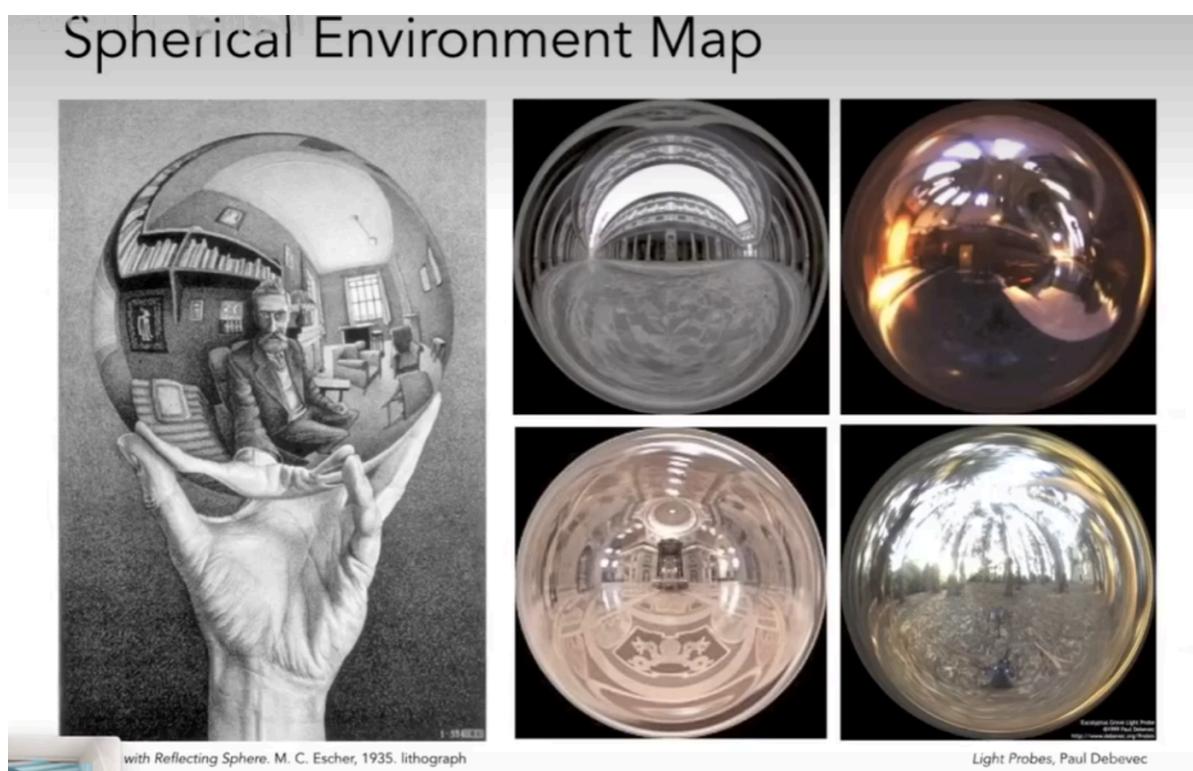
//=====  
=====

# Geometry

纹理可以用来表示环境光

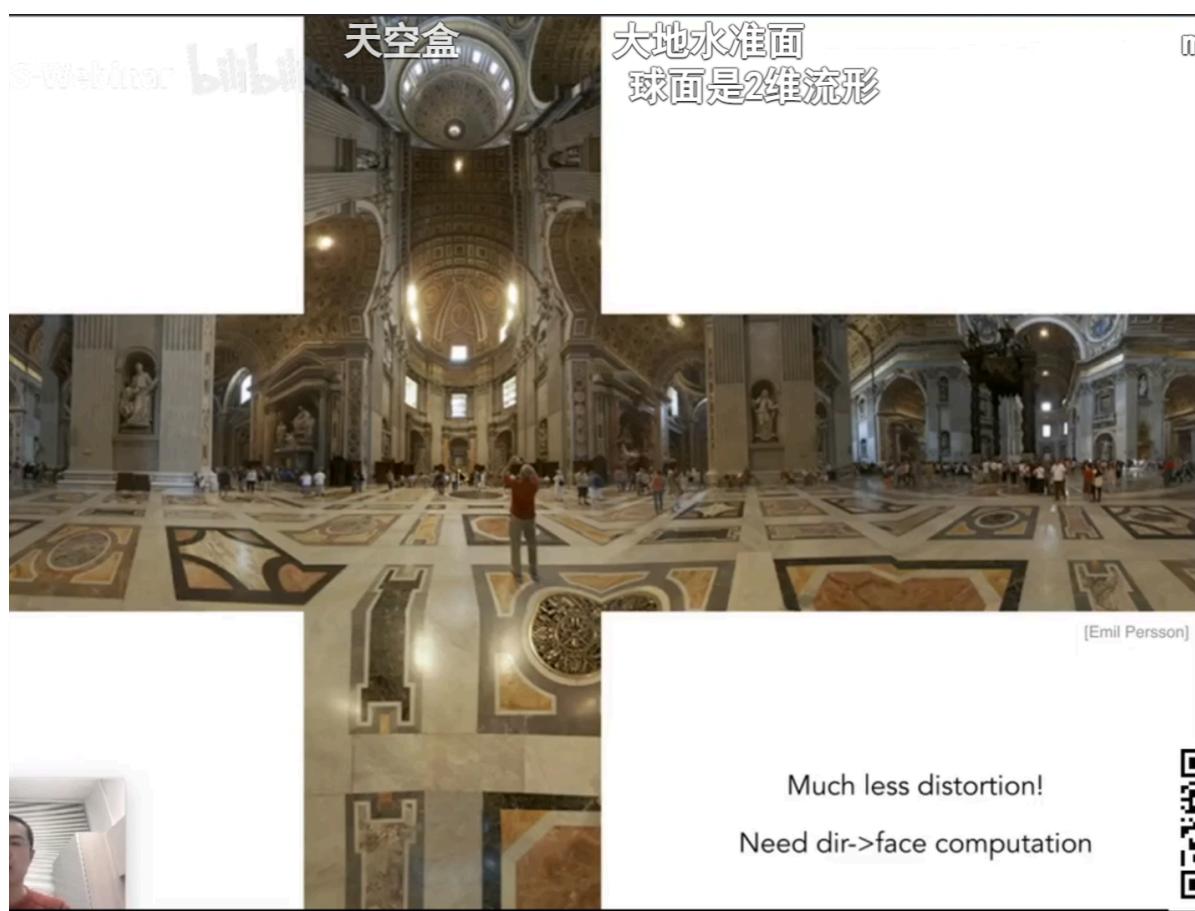
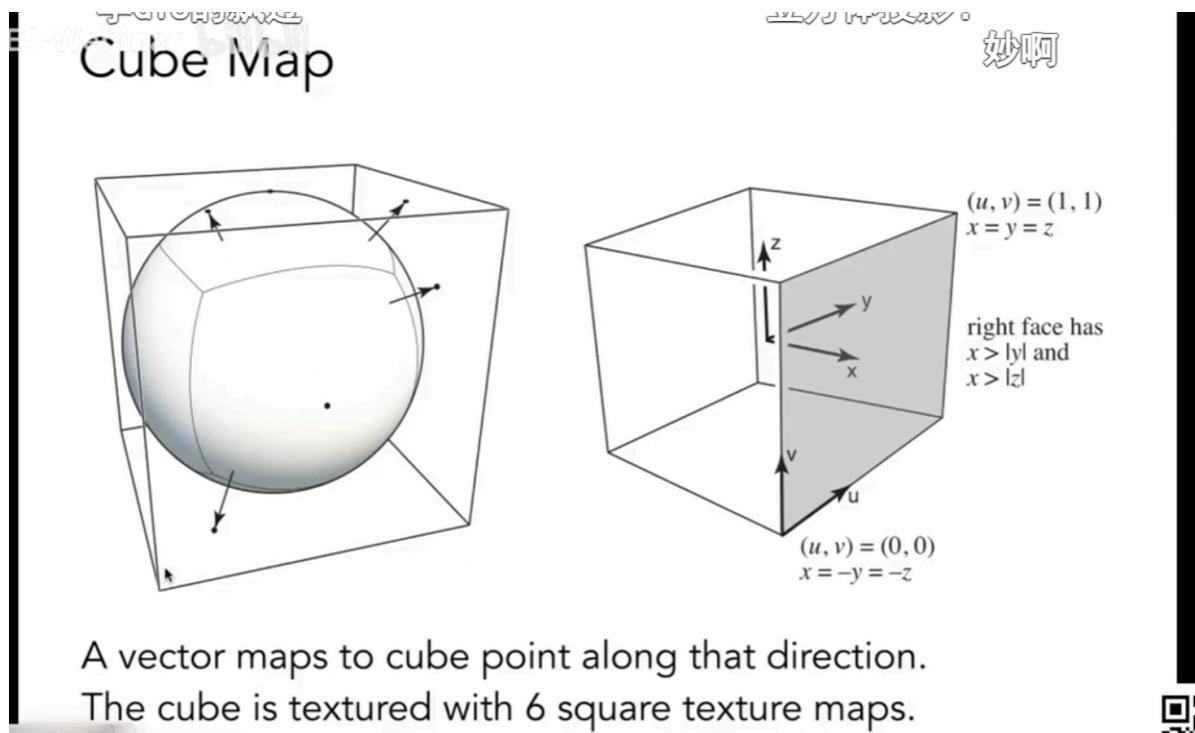
环境光都来自无穷远处，只记录方向即可。

环境光可记录在一个球面上，并进行展开



会存在一个扭曲问题

使用Cube map 球的包围盒

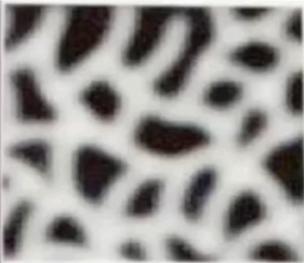


纹理还能做凹凸贴图，可以模拟出法线 法线贴图

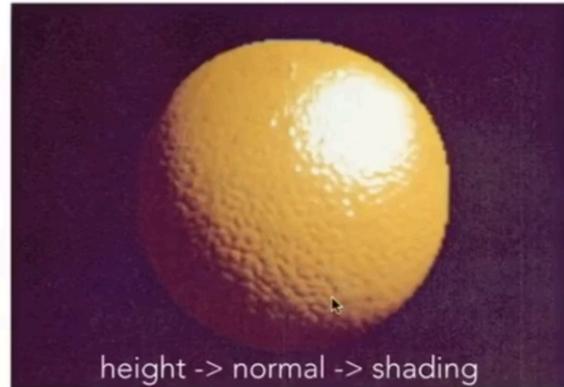
## 贴图, TBN变换 Textures can affect shading!

- Textures doesn't have to only represent colors

- What if it stores the height / normal?
- Bump / normal mapping
- **Fake** the detailed geometry



Relative height to the underlying surface

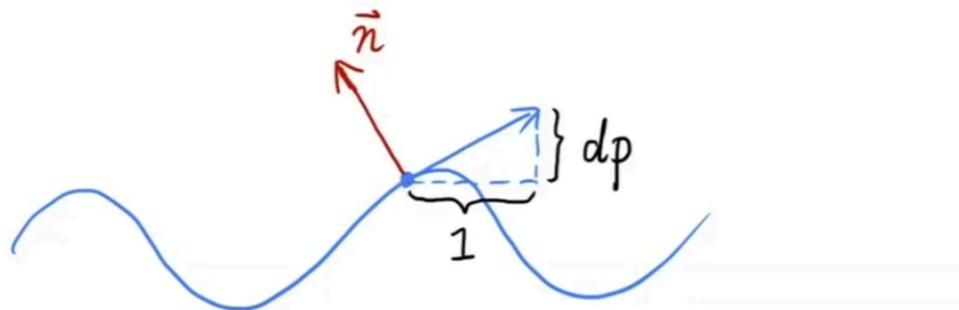


height -> normal -> shading

不改变几何信息，对法线进行扰动，定义任何一个像素的及其邻近位置的高度，通过高度差来重新计算法线。  
(相对高度移动)

## How to perturb the normal (in flatland)

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



用差分方法得切线，再逆时针旋转90°（XY对换后对Y求负）得法线。

### 三维情况

是呀，为啥不叉乘  
How to perturb the normal (in 3D)  
切线方向和副切线方向的叉乘 妙啊  
没看懂？

- 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  $n = (-dp/du, -dp/dv, 1).normalized()$

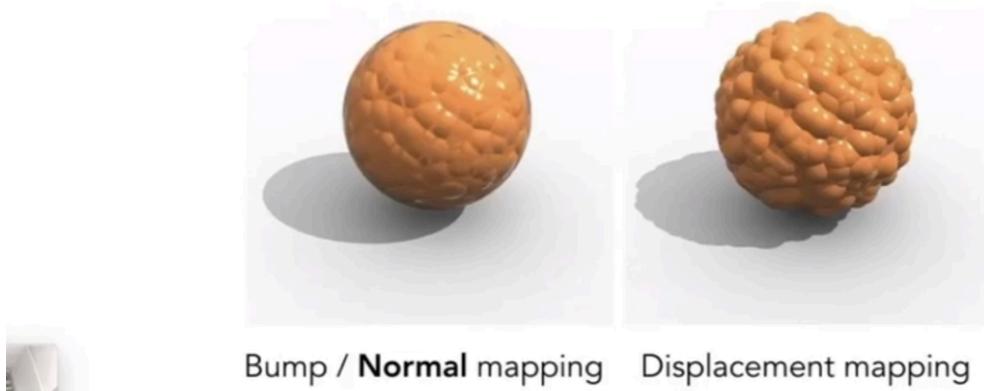
凹凸贴图在边缘和阴影处存在穿帮情况

另一种方式：位移贴图

移动了顶点的位置。但需要模型能完整反应纹理的变化。

# Textures can affect shading!

- **Displacement mapping** — a more advanced approach
  - Uses the same texture as in bumping mapping
  - Actually **moves the vertices**



引入动态曲面细分