

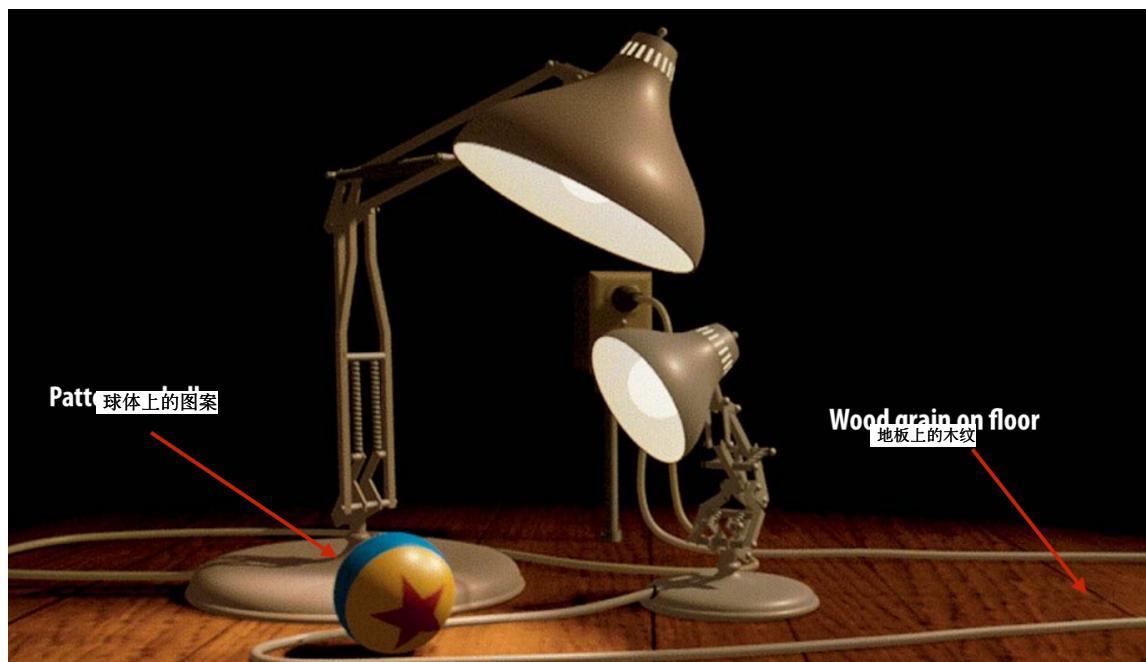
# 讲义四 - 纹理映射

CS248a 2023 Winter

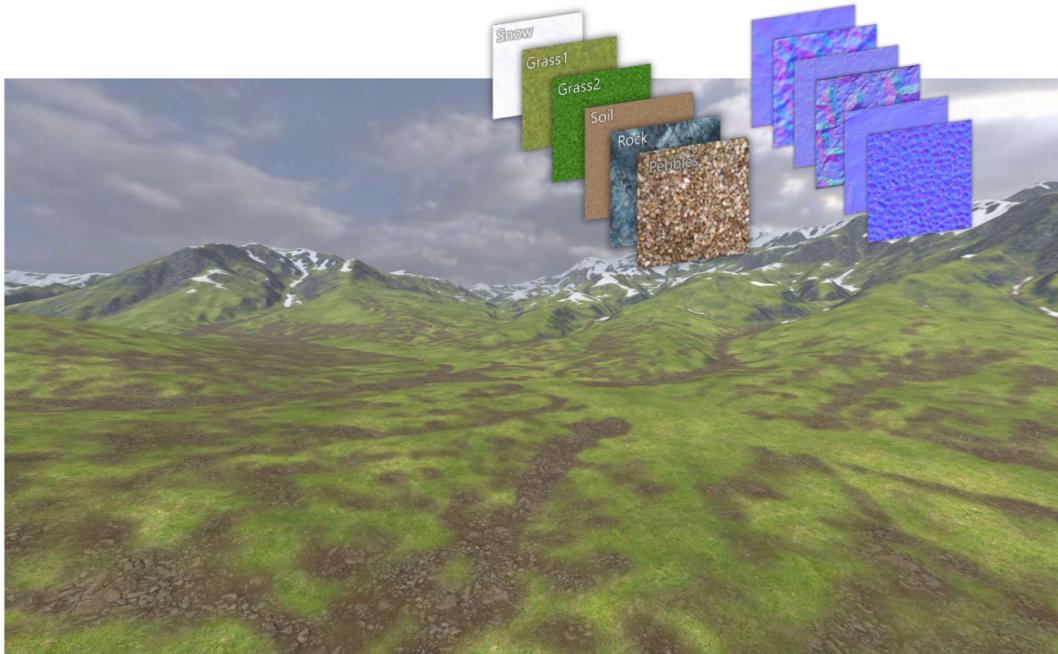


## 纹理映射有很多用途

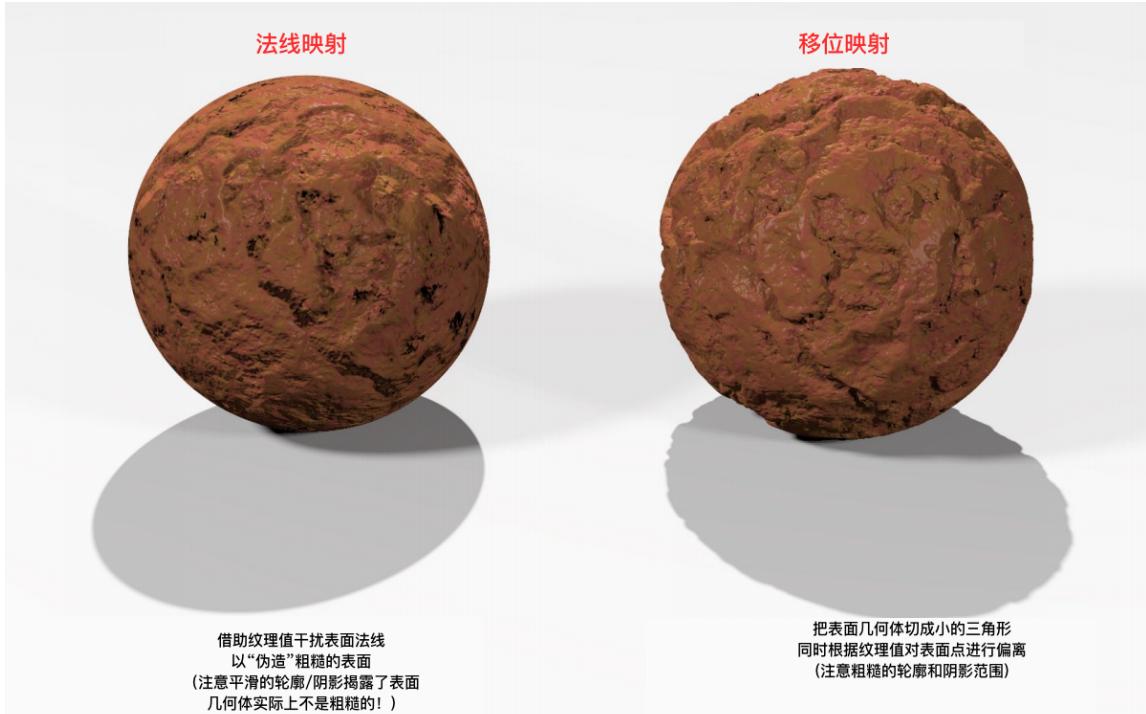
定义表面反射中的变化



## 分层材料



## 法线和移位映射



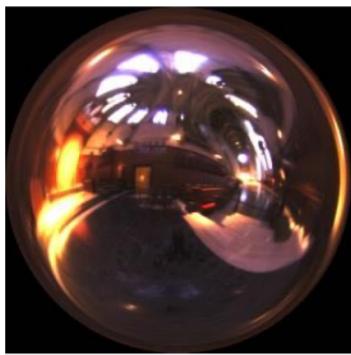
## 表达预算算的光照和阴影



原始模型

使用环境光遮挡的模型

提取的环境光遮挡纹理贴图



格雷斯大教堂环境纹理贴图



用于渲染的环境纹理贴图

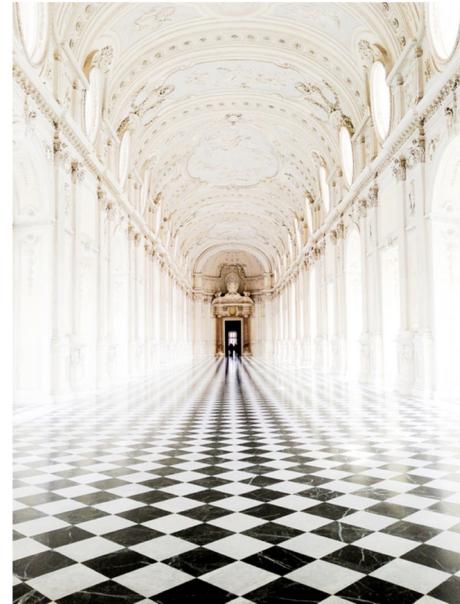
## 透视和纹理

### ■ 以前:

- 变换（如何操纵空间中的原始（基本）图形 primitive）
- 光栅化（如何将原始（基本）图形转换为着色的像素）

### ■ 现在:

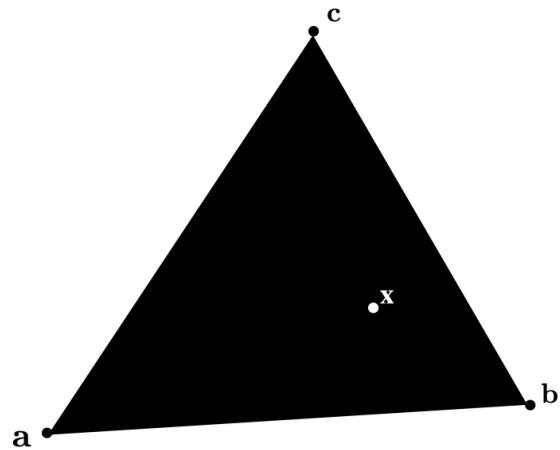
- 观察两种思路在一起产生冲突的地方
- 透视变换
- 讨论怎样在基础图形上映射纹理以获得更多细节
- ...以及在纹理映射中透视在怎样的情形会遭遇挑战



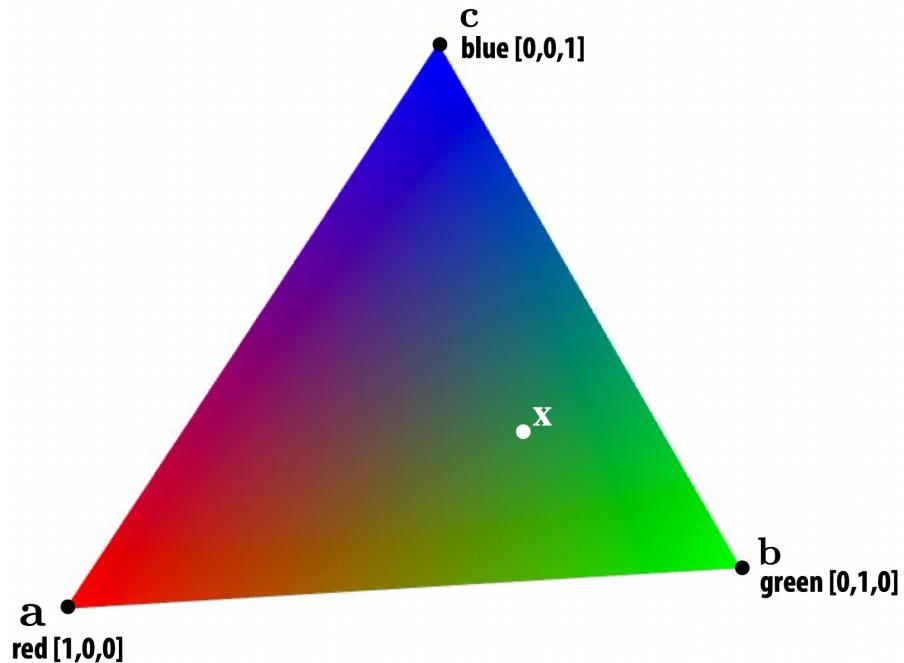
为什么难以渲染  
这样一张图片？

## 回忆一下讲义2中的 $coverage(x, y)$ 函数

在讲义2中，我们讨论了，在给定三角形顶点的2D位置的情形下，如何采样覆盖率。



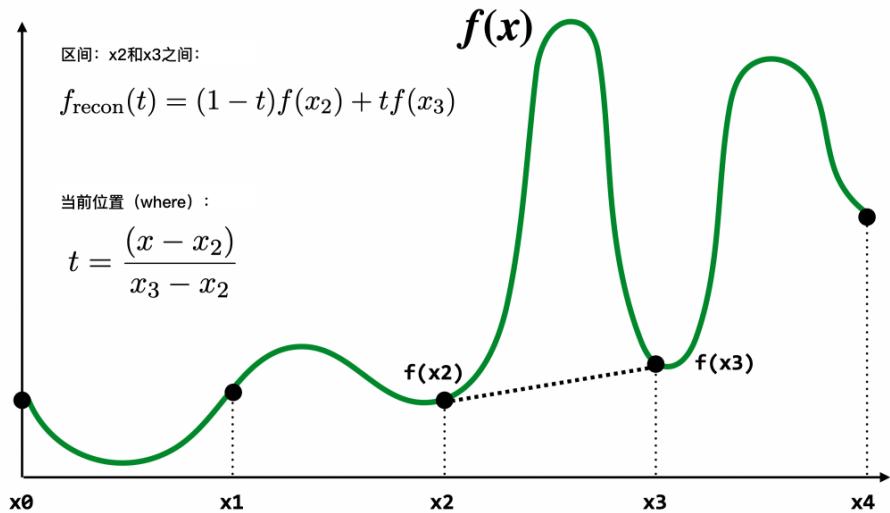
思考采样色彩函数  $color(x, y)$



给定点  $a, b, c$  处的色彩，在  $X$  点定点的色彩是什么？

## 回顾：1D中的操作

$f_{recon}(x)$  = 在相距x的两个最近的样本值之间线性插值



思考三角形上的相似行为

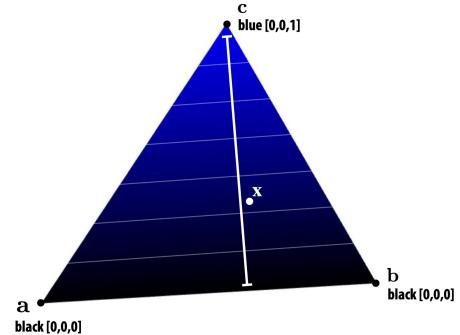
色彩依赖于从 $b - a$ 的距离

$$color = (1 - t)[0\ 0\ 0] + t[0\ 0\ 1]$$

$$t = \frac{\text{distance from } x \text{ to } b-a^1}{\text{distance from } c \text{ to } b-a^2}$$

---

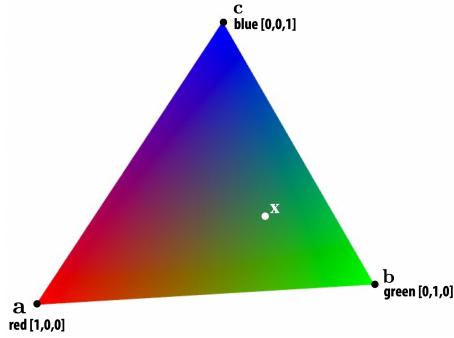
<sup>1</sup> 从x到b-a的距离  
<sup>2</sup> 从c到b-a的距离



2D环境中，我们如何在3个值之间插值？

## 三角形上数值的线性插值

对于三角形中的点（远点定为a）， $b - a$ 和 $c - a$ 构成了非正交基



$$\begin{aligned}x &= a + \beta(b - a) + \gamma(c - a) \\&= (1 - \beta - \gamma)a + \beta b + \gamma c \\&= \alpha a + \beta b + \gamma c\end{aligned}$$

$$\alpha + \beta + \gamma = 1$$

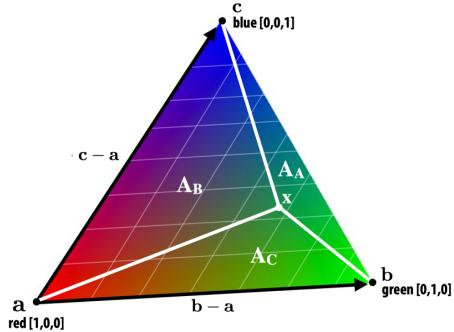
点X处的色彩值是3个三角形顶点处色彩值的线性组合

$$x_{color} = \alpha a_{color} + \beta b_{color} + \gamma c_{color}$$

给出点 $a, b, c$ 处的色彩，在点 $X$ 处三角形的色彩是什么？

## 重心坐标为面积的比率

有向面积 (signed area) 的比率:



$$\alpha = A_A/A$$

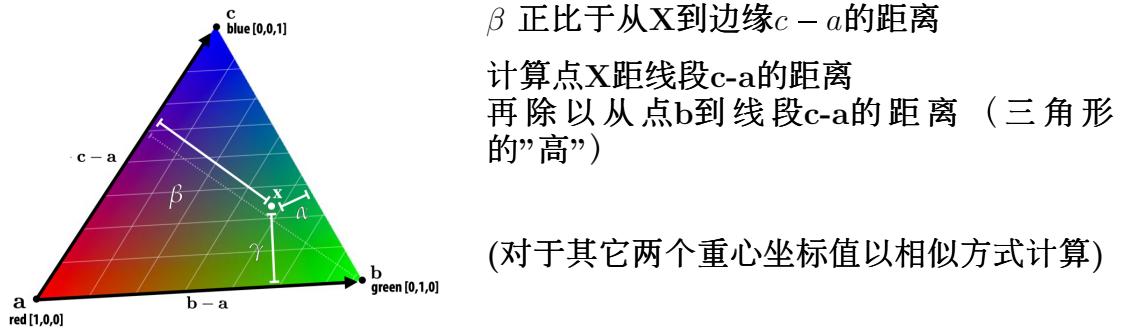
$$\beta = A_B/A$$

$$\gamma = A_C/A$$

为什么坐标值的和必须为1?

为什么坐标值必须在0和1之间?

## 重心坐标也可作为伸缩的距离比率



在三角形上，可否以这种方式线性插值（在顶点上定义的）任何值

此处：线性插值了位置( $X, Y, Z$ )，色彩( $r, g, b$ )，以及额外的值( $u, v$ )

$$\mathbf{C} = (x_2, y_2, z_2, r_2, g_2, b_2, u_2, v_2)$$

顶点是绿色的，所以( $r_2, g_2, b_2$ ) = (0,1,0)

$$\mathbf{A} = (x_0, y_0, z_0, r_0, g_0, b_0, u_0, v_0)$$

顶点是黑色的，所以( $r_0, g_0, b_0$ ) = (0,0,0)

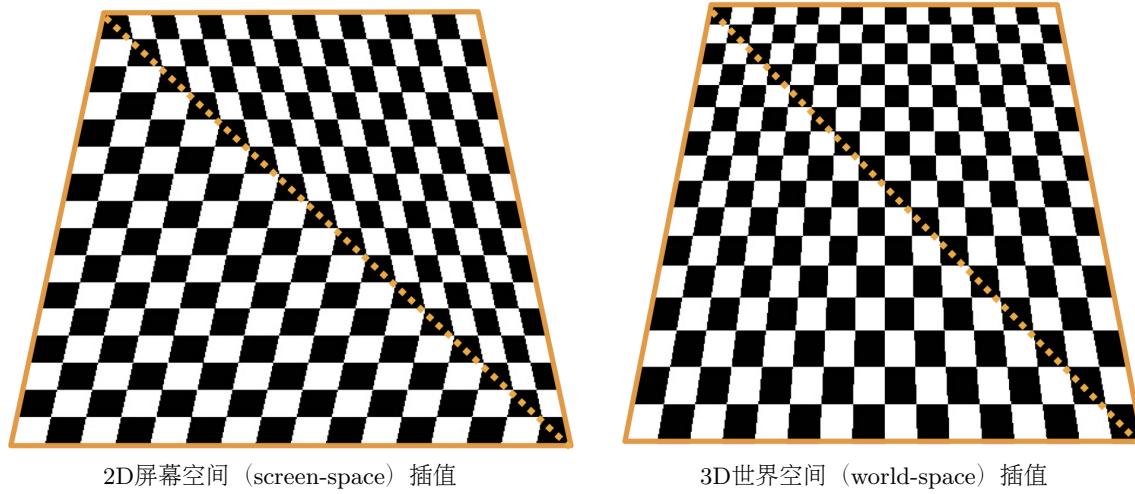
$$\mathbf{B} = (x_1, y_1, z_1, r_1, g_1, b_1, u_1, v_1)$$

顶点是红色的，所以( $r_1, g_1, b_1$ ) = (1,0,0)

不是很快...透视的非正确插值

## 透视的正确插值

这里是一个（由两个三角形所构成的）平面，以俯视角度，按照透视方式所渲染



## 透视正确的插值

假设三角形属性横跨整个三角形表面线性变化

位于三角形上3D点 $P = [xyz]^T$ 的属性被给出如下：

$$f(x, y, z) = ax + by + cz$$

透视投射 $P$ ,获得2D齐次 (homogeneous) 坐标表达如下：

$$\begin{bmatrix} x_{2D-H} \\ y_{2D-H} \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

在2D-H中的点 $P$ 的投射      扔掉 $z$ 以移动到2D-H  
3D-H坐标中点 $P$ 的透视投射

\*注意：使用一个更通用的透视投射矩阵，仅仅会改变 $x_{2d}$ 和 $y_{2d}$ 的系数。（仍然持有齐次属性-比如 $f/w$ 还是齐次的）

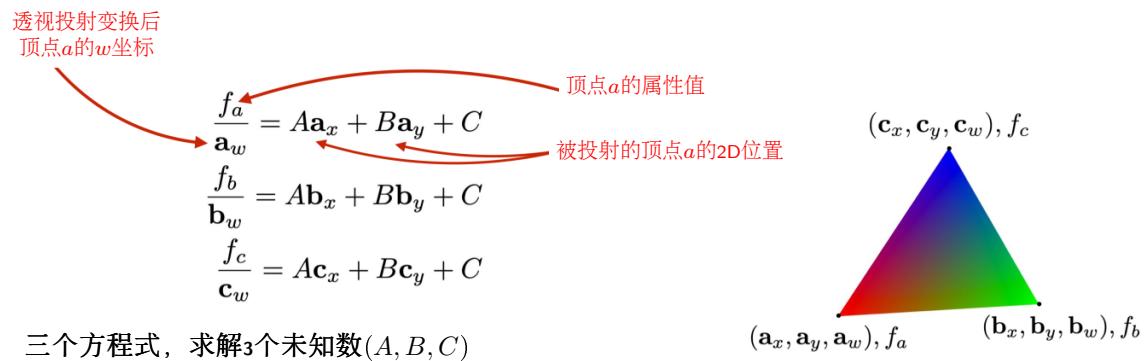
随后代入本页幻灯片顶部的等式  $f \dots$

$$\begin{aligned} f(x_{2D-H}, y_{2D-H}) &= ax_{2D-H} + by_{2D-H} + cw \\ \frac{f(x_{2D-H}, y_{2D-H})}{w} &= \frac{a}{w}x_{2D-H} + \frac{b}{w}y_{2D-H} + c \\ \frac{f(x(2D), y(2D))}{w} &= \frac{a}{w}x(2D) + \frac{b}{w}y(2D) + c \end{aligned}$$

最终...  $\frac{f}{w}$  为 2D 屏幕坐标  $[x_{2D}, y_{2D}]^T$  的仿射函数。

## 表面属性的直接评估

针对任意表面属性 (将在三角形顶点处的值定义为:  $f_a, f_b, f_c$ )



这个计算被实现为先于采样的每三角形“设置”，正如你通过评估覆盖率计算边缘等式

## 高效的透视正确插值

设置:

给定  $f_a, f_b, f_c$  和  $w_a, w_b, w_c \dots$ , 针对方程  $f/w(x, y) = Ax + By + C$  计算  
同时计算求解  $1/w(x, y)$  的方程式

在每个覆盖的样本  $(x, y)$  评估表面属性  $f(x, y)$

评估  $1/w(x, y)$  (来源于针对  $1/w$  预计算的方程式)  
交换  $1/w(x, y)$  位置获得  $w(x, y)$

针对每个三角形属性:

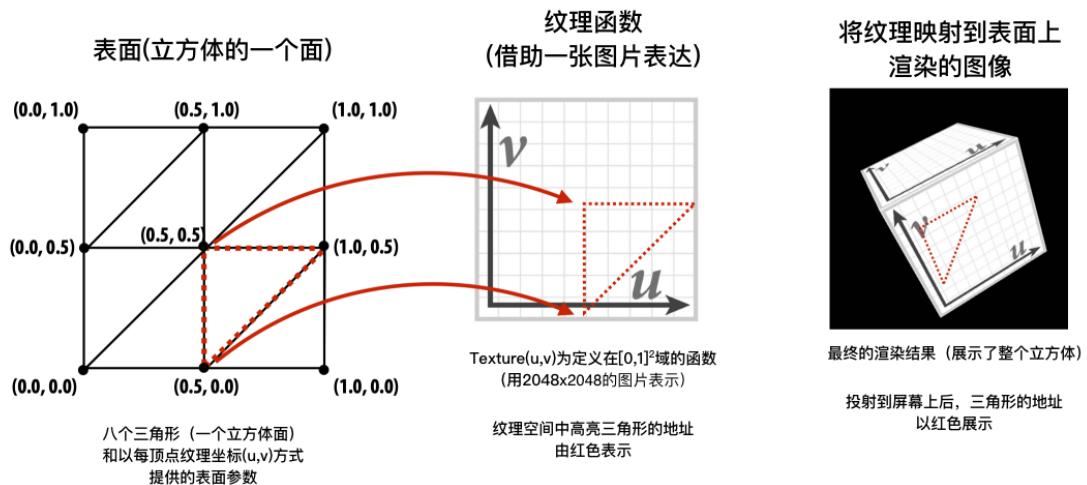
评估  $f/w(x, y)$  (来源于针对  $f/w$  预计算的方程式)  
用  $w(x, y)$  乘以  $f/w(x, y)$  获得  $f(x, y)$

针对任何跨三角形表面线性变化的属性  $f$  都有效:

例如: 色彩、深度、纹理坐标

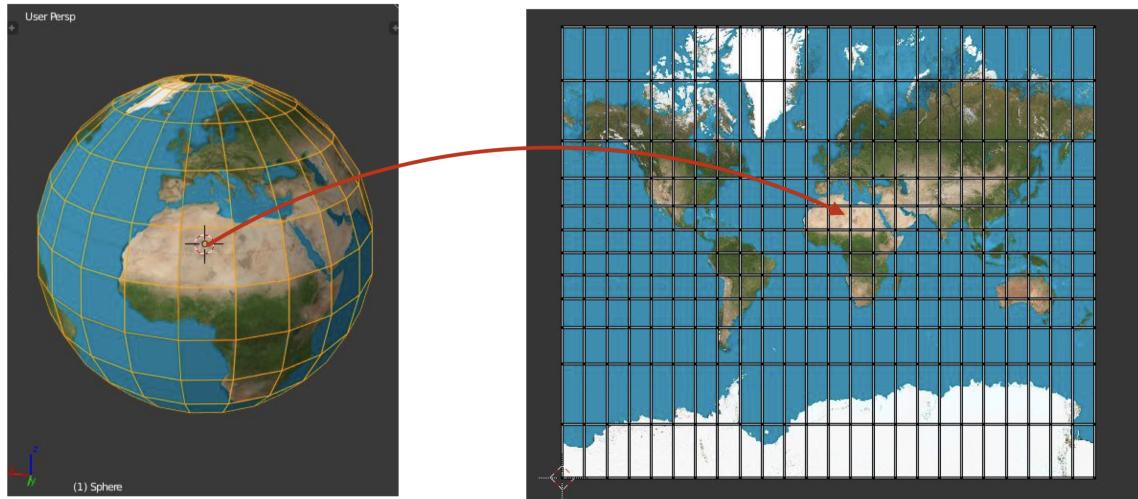
## 纹理坐标

“纹理坐标”定义了从表面坐标（例如，三角形上的点）到纹理图像域中的坐标映射



今天我们会假设表面到纹理的空间映射被提供作为顶点属性。  
(不讨论生成表面纹理参数的方法)

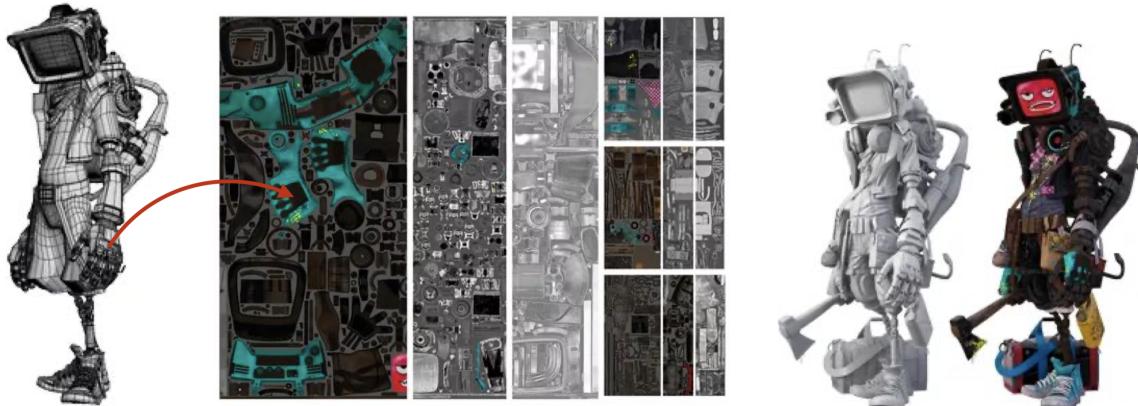
从表面点到纹理空间有很多不同映射



样例：球体上的mercator<sup>3</sup>项目

<https://blender.stackexchange.com/questions/3315/how-to-get-perfect-uv-sphere-mercator-projection>

纹理“地图”

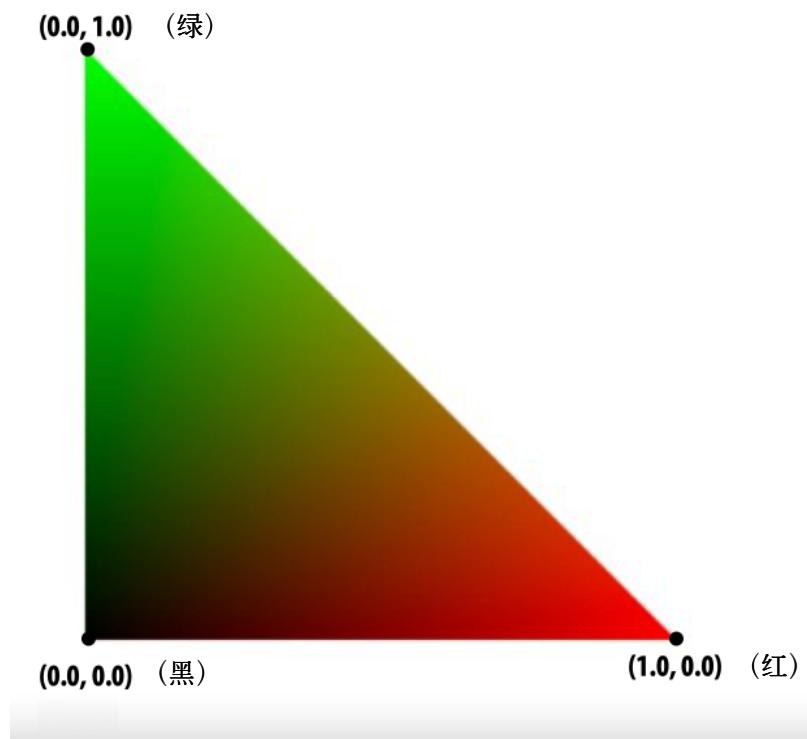


<https://www.creativebloq.com/3d/how-create-killer-3d-robot-21410645>

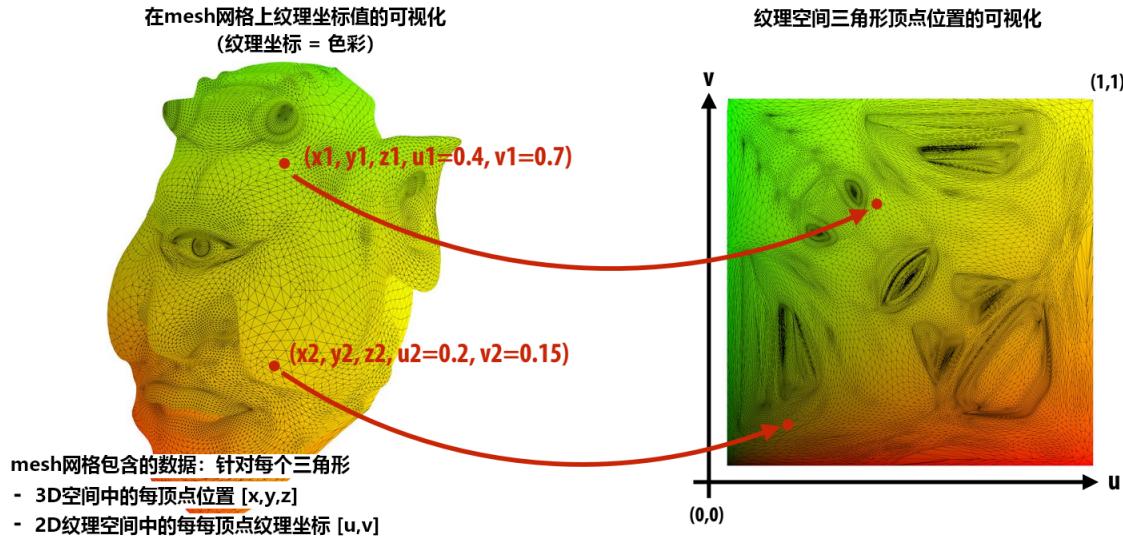
<sup>3</sup>德国地图学家-墨卡托,<https://www.britannica.com/science/cartography>

## 纹理坐标的可视化

在三角形上纹理坐标被线性插值

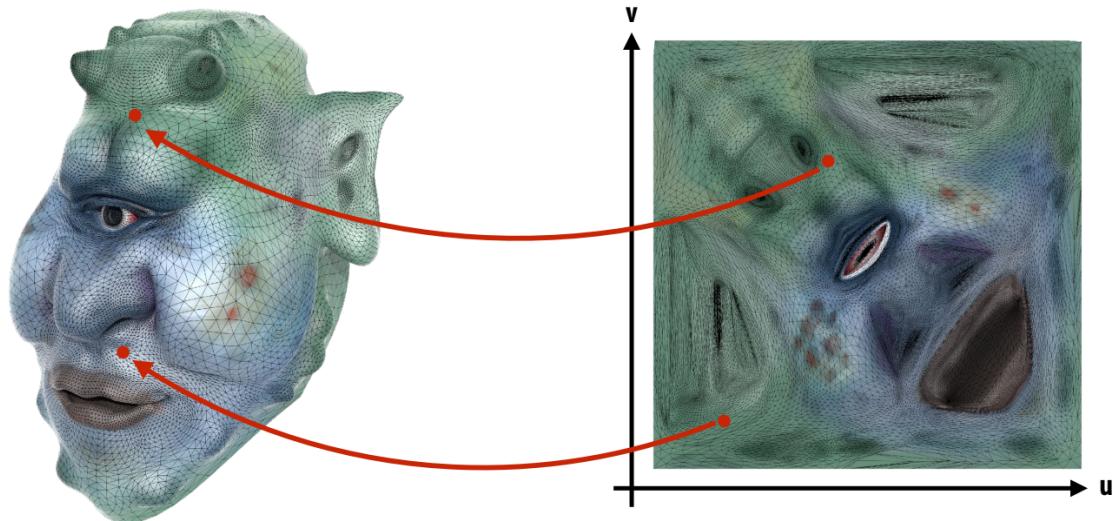


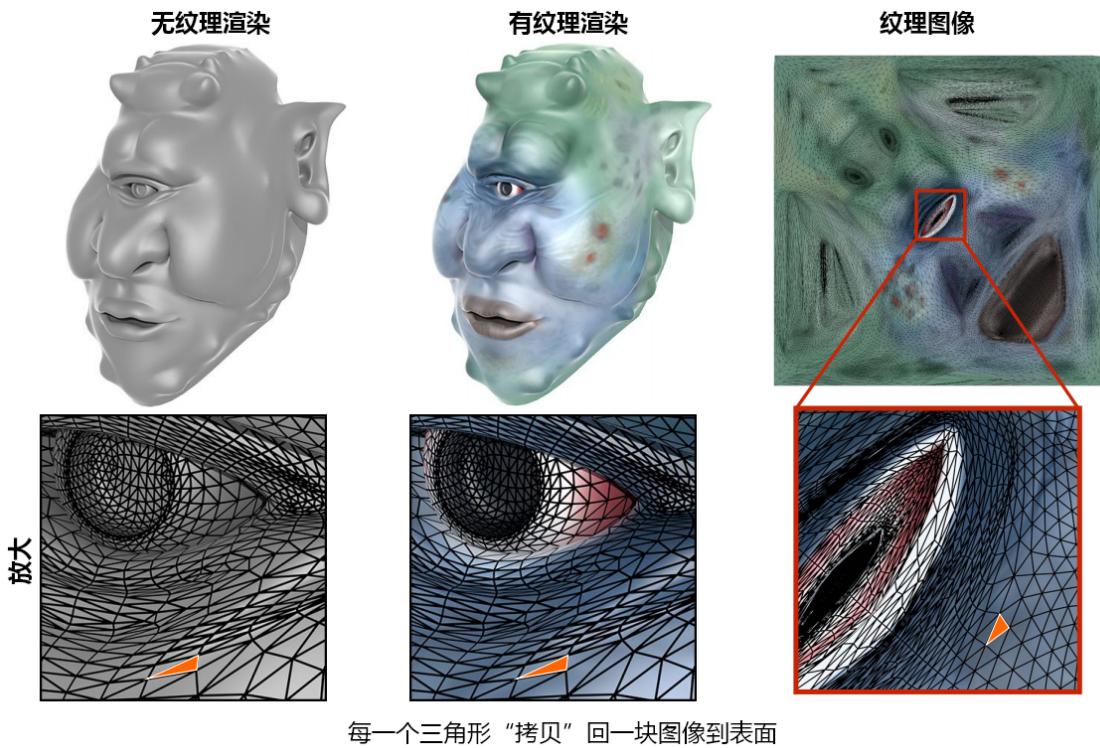
纹理坐标被提供到三角形顶点上  
(正如3D位置被提供到三角形顶点上)



## 纹理映射增添细节

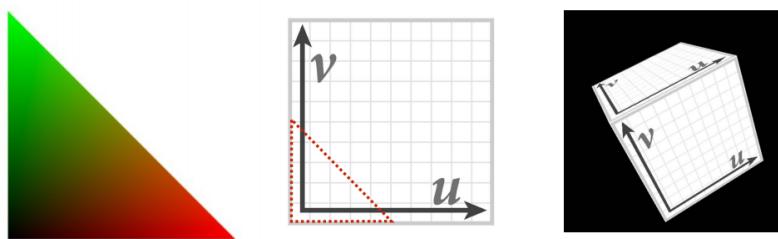
纹理坐标空间中，在指定位置采样纹理映射图以决定表面上对应点的表面色彩。





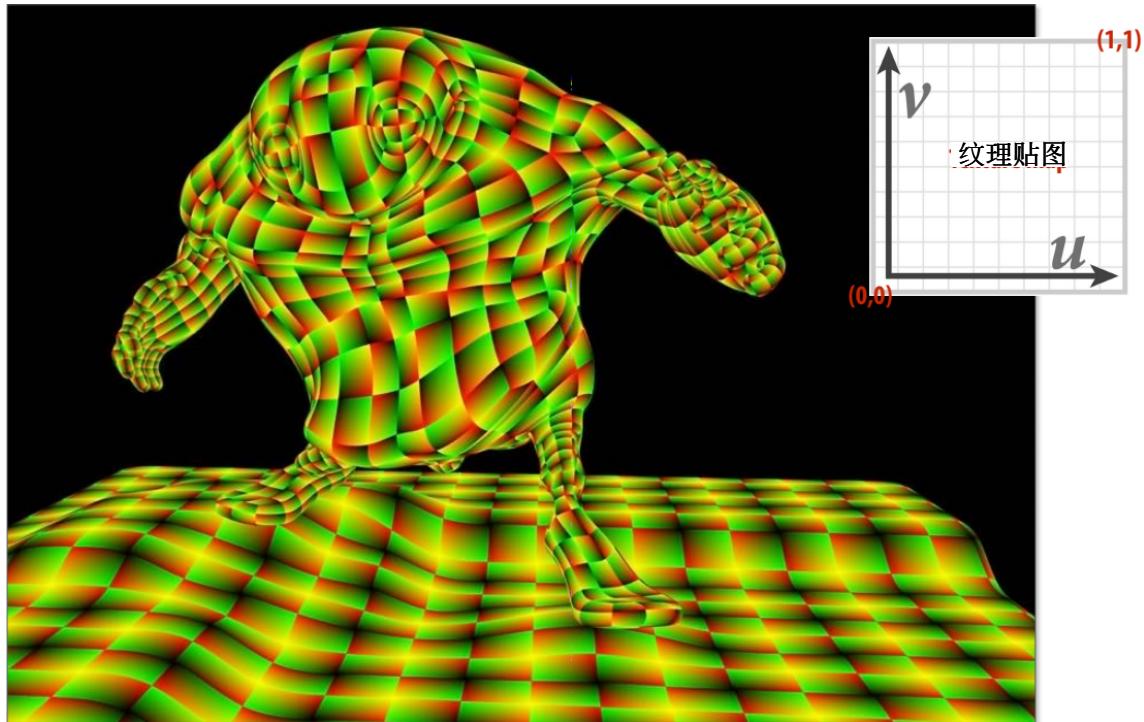
## 纹理采样101

- 用于映射纹理到表面的最基本算法：
  - 针对每个纹理采样地址  $(X, Y)$
  - 跨三角形插值  $U$  和  $V$  坐标，获得在  $(X, Y)$  处的值
  - 在由  $(U, V)$  指定的地址采样（评估）纹理
  - 把表面点的色彩设置为被采样的纹理值



## 纹理坐标可视化

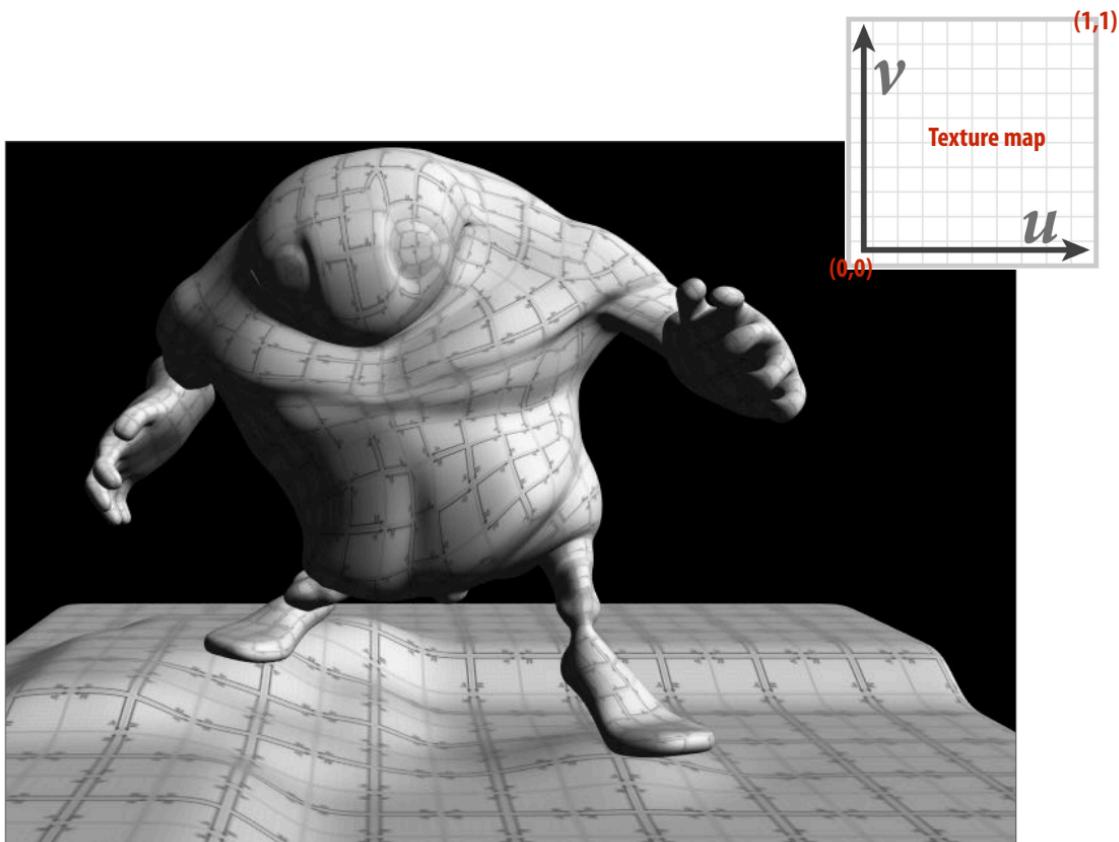
定义从表面点到纹理域中点( $uv$ )的映射



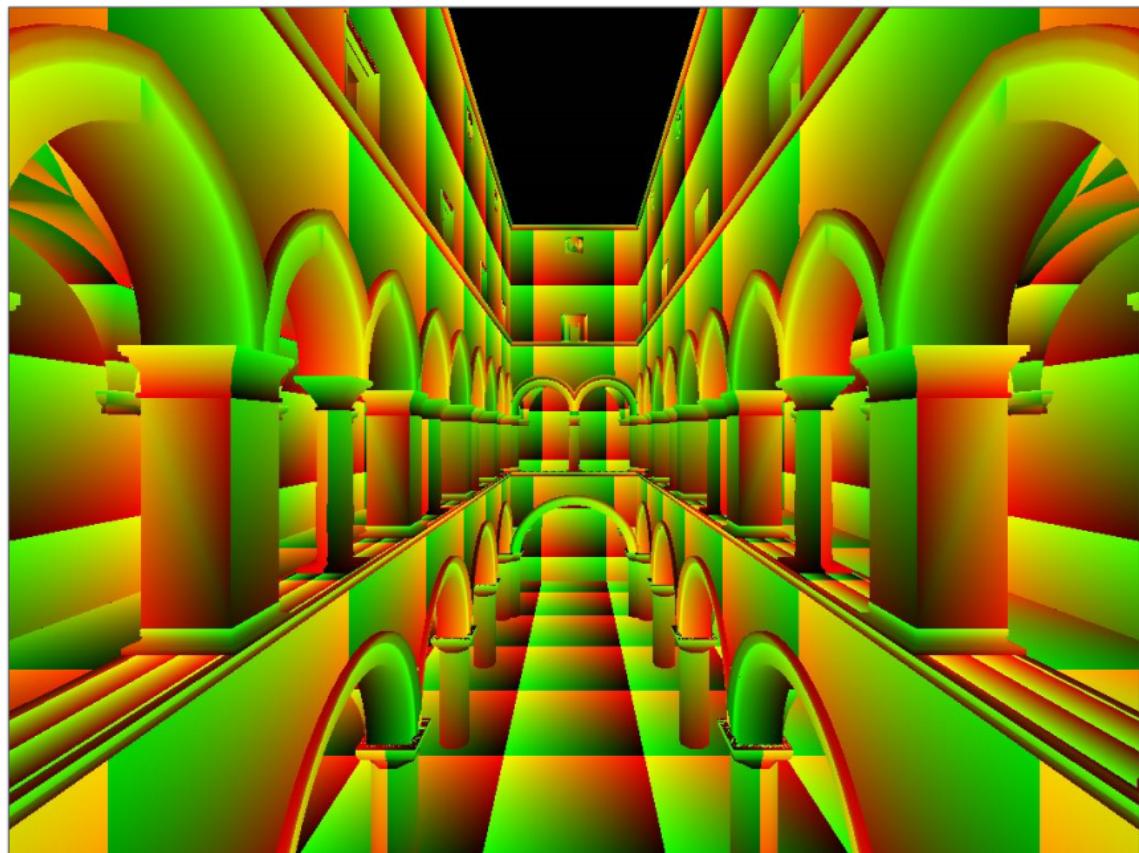
红色通道= $u$ , 绿色通道= $v$

所以 $(u, v) = (0, 0)$ 是黑色,  $(u, v) = (1, 1)$ 是黄色

## 渲染结果

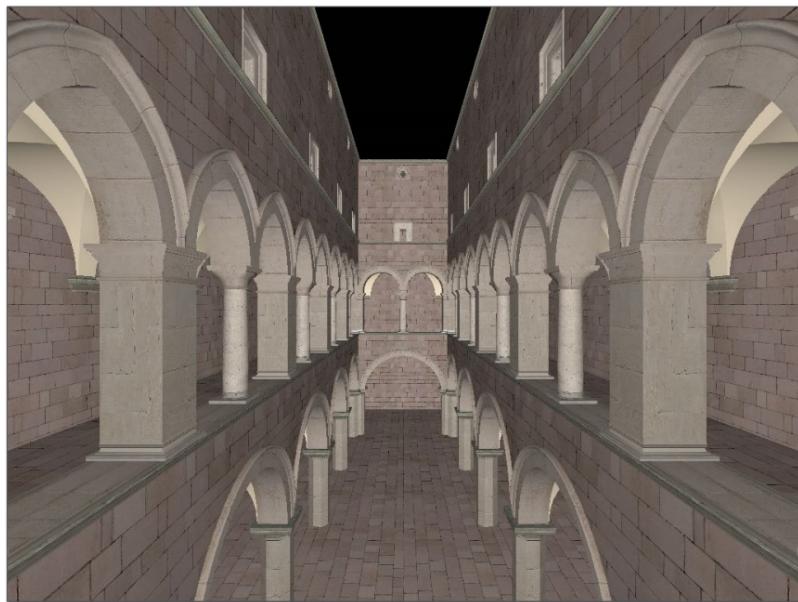


## 纹理坐标的可视化表达



要注意纹理坐标在表面上重复

例子：纹理化的场景



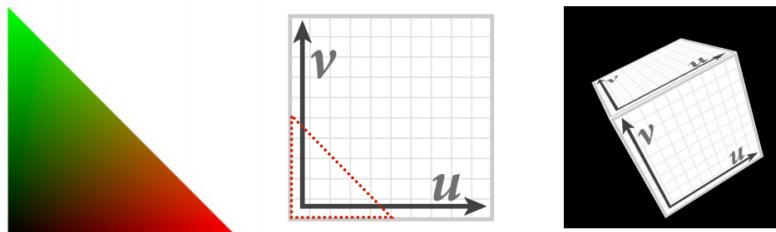
例子：前面场景中使用的纹理



## 纹理映射：基本算法

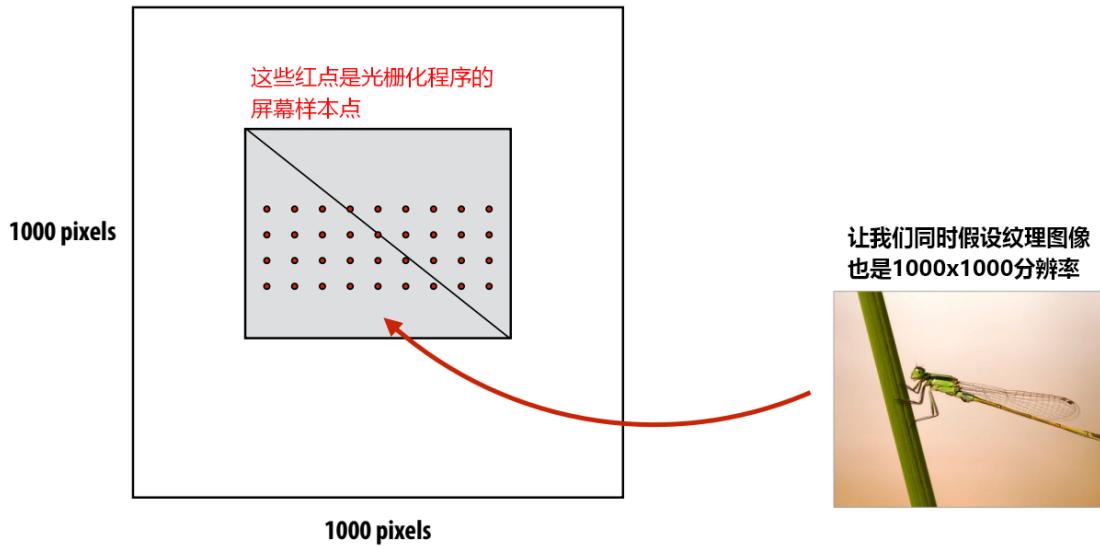
■ 用于映射纹理到表面的最基本算法：

- 针对每个纹理采样地址  $(X, Y)$ 
  - 跨三角形插值  $U$  和  $V$  坐标，获得在  $(X, Y)$  处的值
  - 在由  $(U, V)$  指定的地址采样（评估）纹理
  - 把表面点的色彩设置为被采样的纹理值

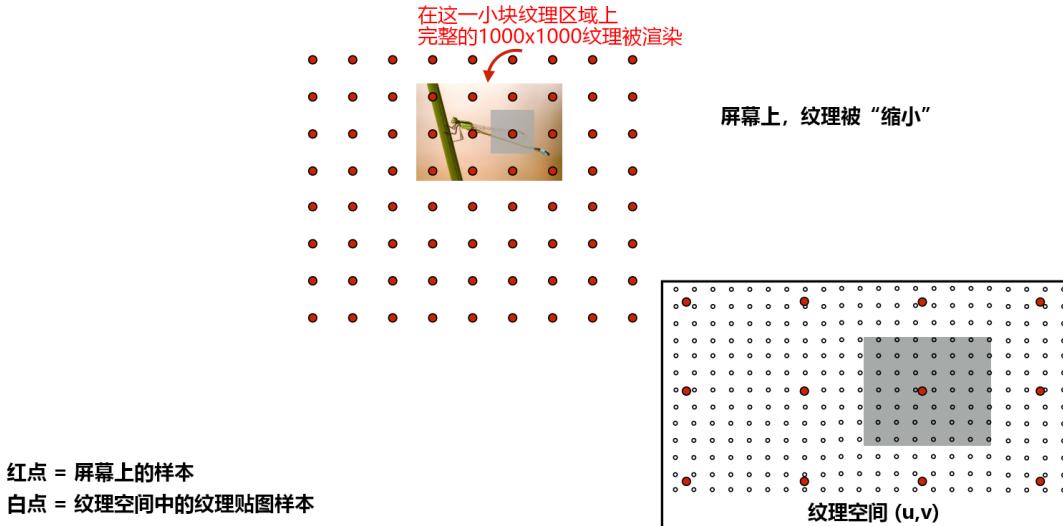


## 想法的实验验证

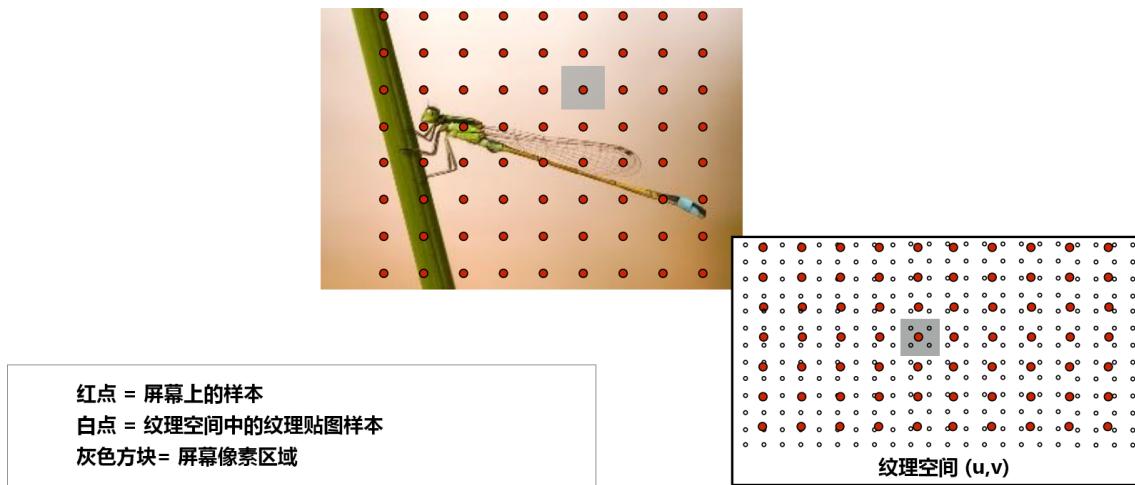
假设渲染一个四边形纹理贴图到一个  $1000 \times 1000$  像素的输出图像上



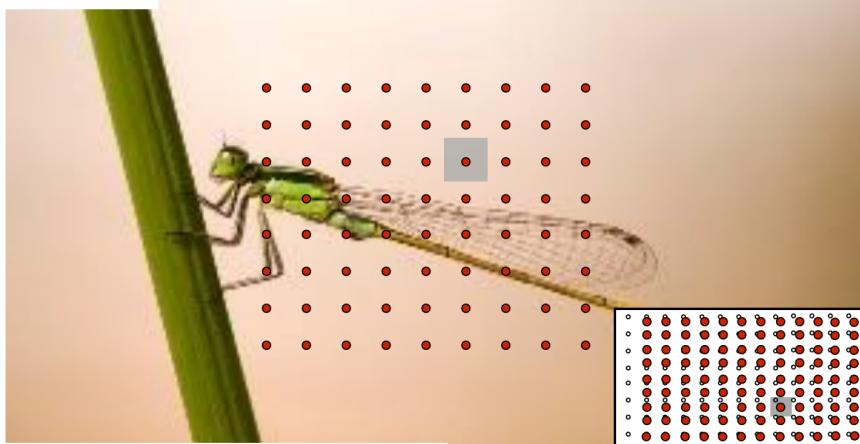
## 在屏幕上的采样率 VS 纹理内的采样率：目标被放大



## 放大中...(Zooming in)



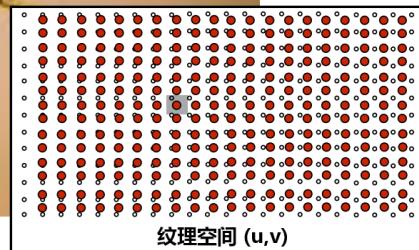
放大中...(Zooming in)



红点 = 屏幕上样本

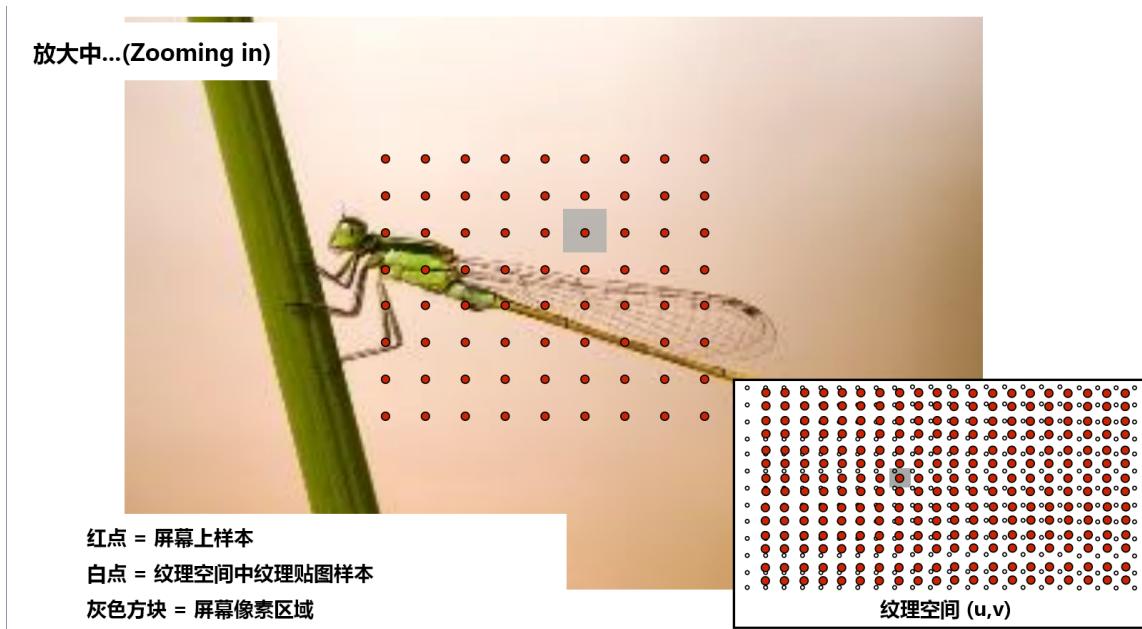
白点 = 纹理空间中纹理贴图样本

灰色方块 = 屏幕像素区域

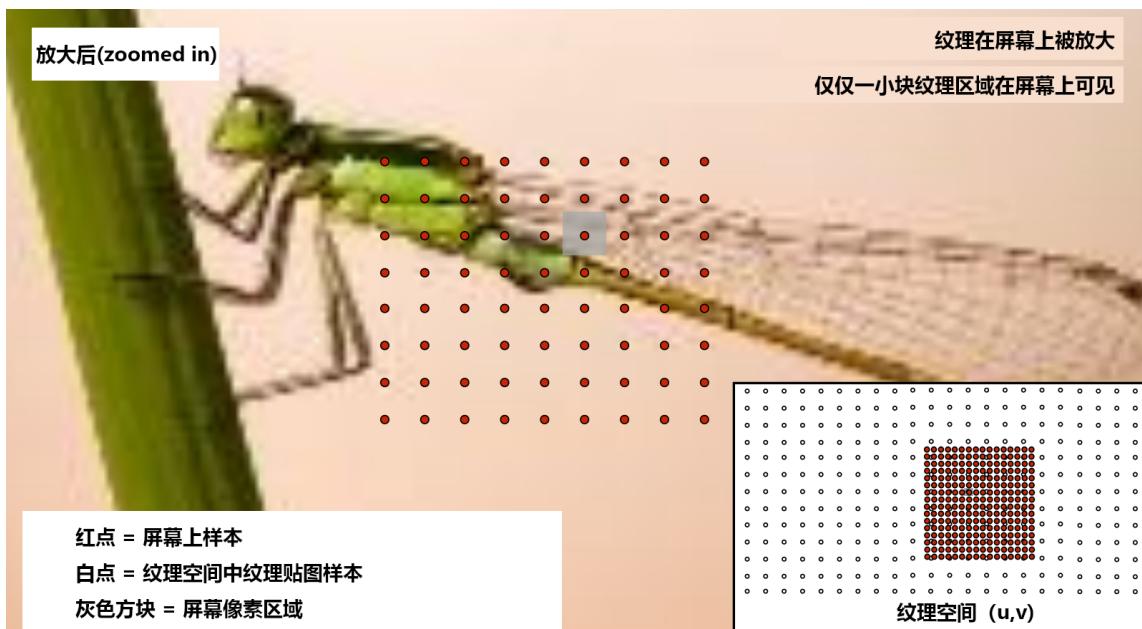


纹理空间 ( $u, v$ )

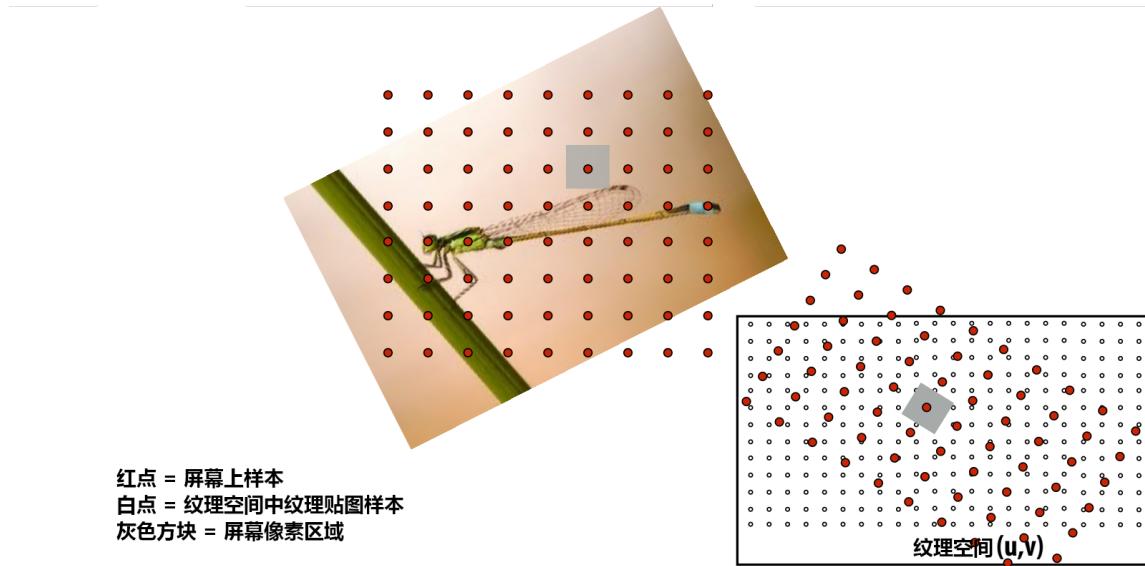
t



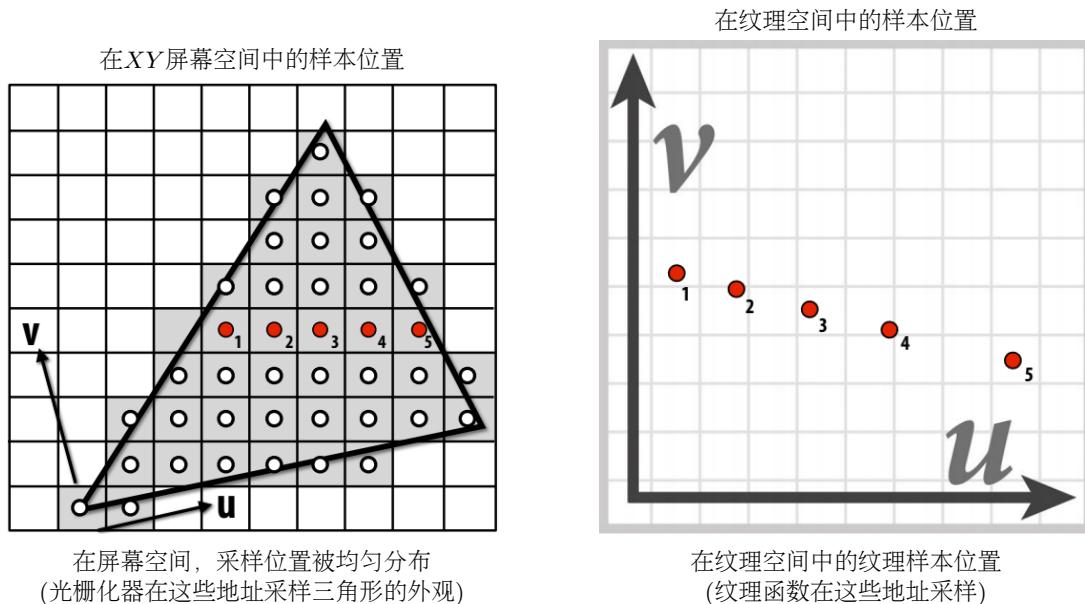
t



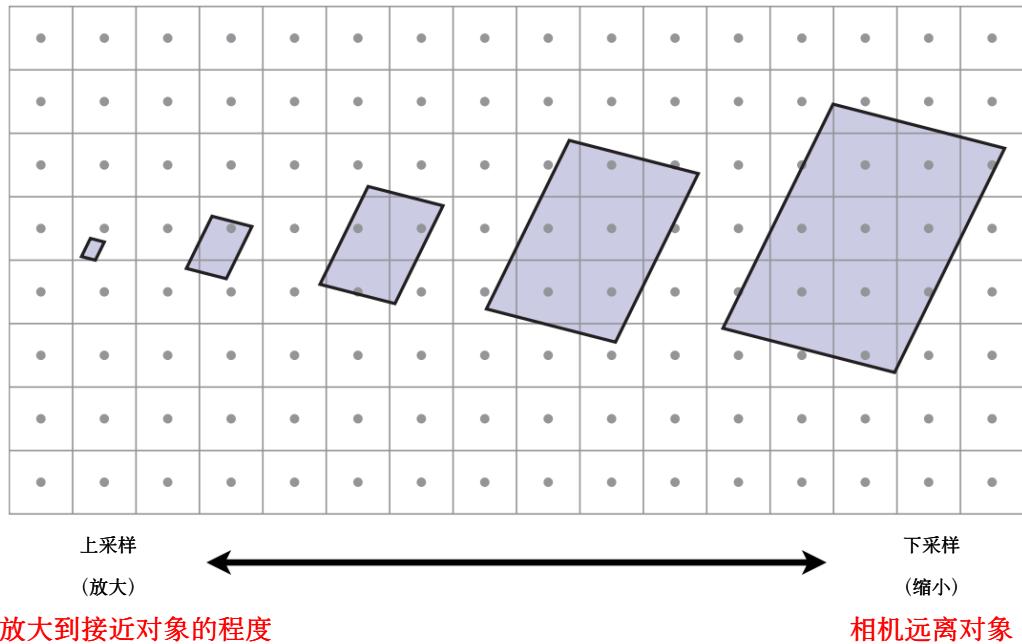
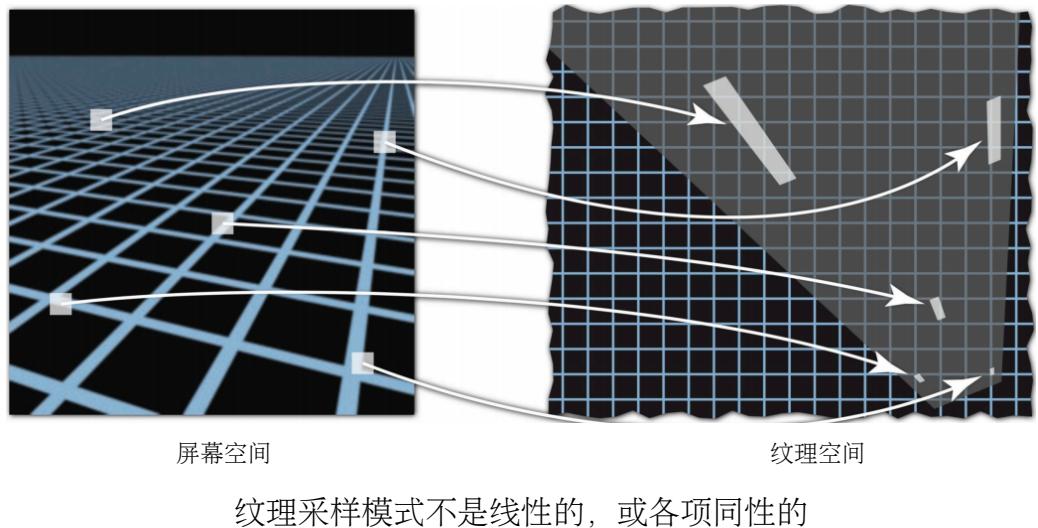
## 屏幕上的采样率对比纹理内的：目标旋转



在屏幕上以相等距离区隔的样本  $\neq$  在纹理空间中也以相等距离区隔样本



## 纹理空间中的屏幕像素足迹

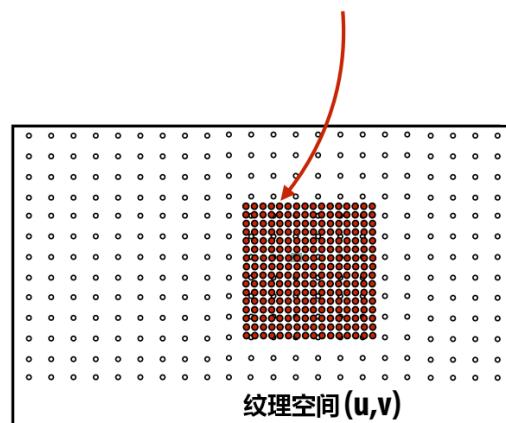


## 屏幕像素区域 VS 纹理像素区域

- 在最优的观看尺寸上:
  - 在像素采样率和纹理像素采样率之间1:1映射
  - 在屏幕和纹理解析度上有依赖
- 当像素区域大于纹理像素区域（纹理缩小化）
  - 思考：镜头远远离开目标物
  - 多个纹理像素样本对应一个像素样本
- 当像素区域小于纹理像素区域（纹理放大化）
  - 思考：镜头移近到目标物上
  - 每纹理像素样本对应多个像素样本

# 纹理放大化

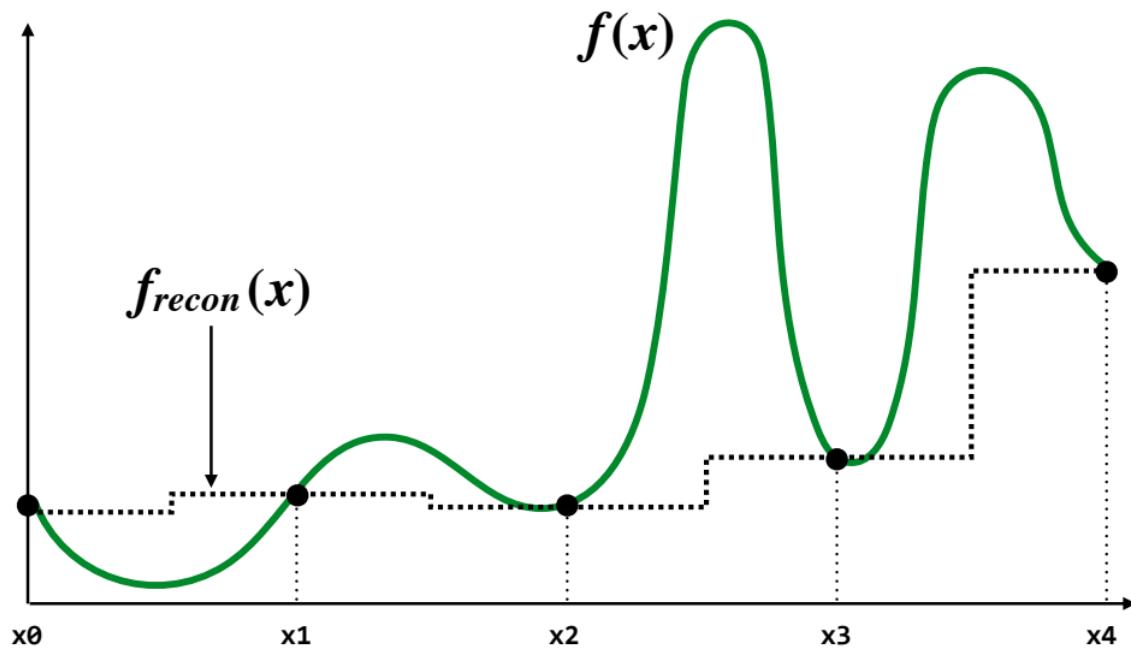
在这些红点处纹理的色彩是什么？



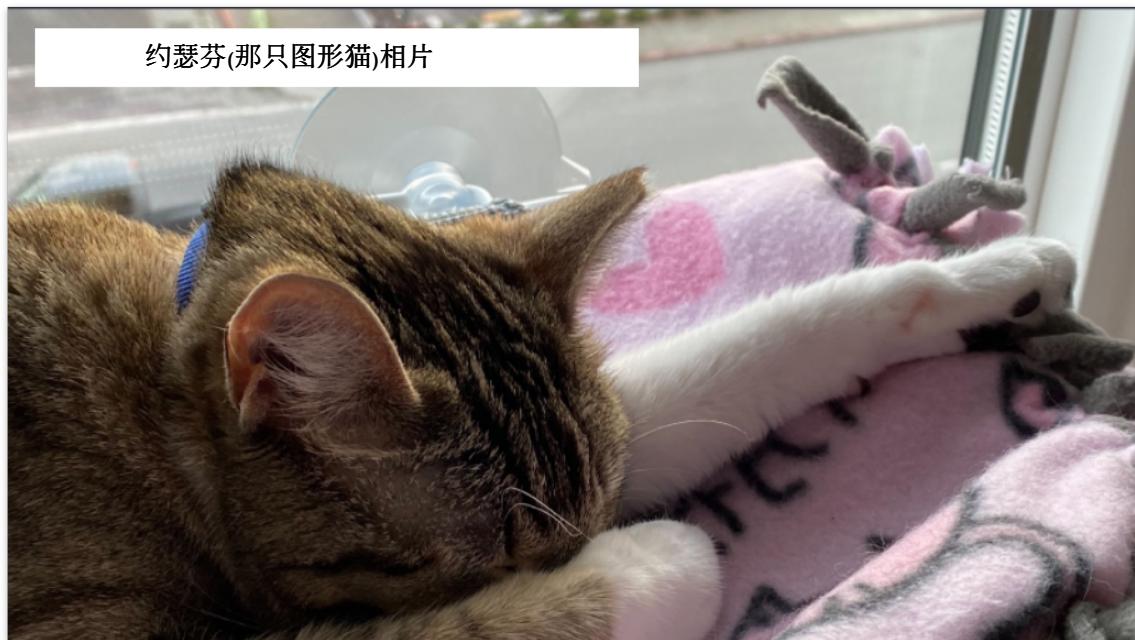
## 回顾：逐段的常量近似

$f_{\text{recon}}(x)$  = 最接近 $x$ 的样本值

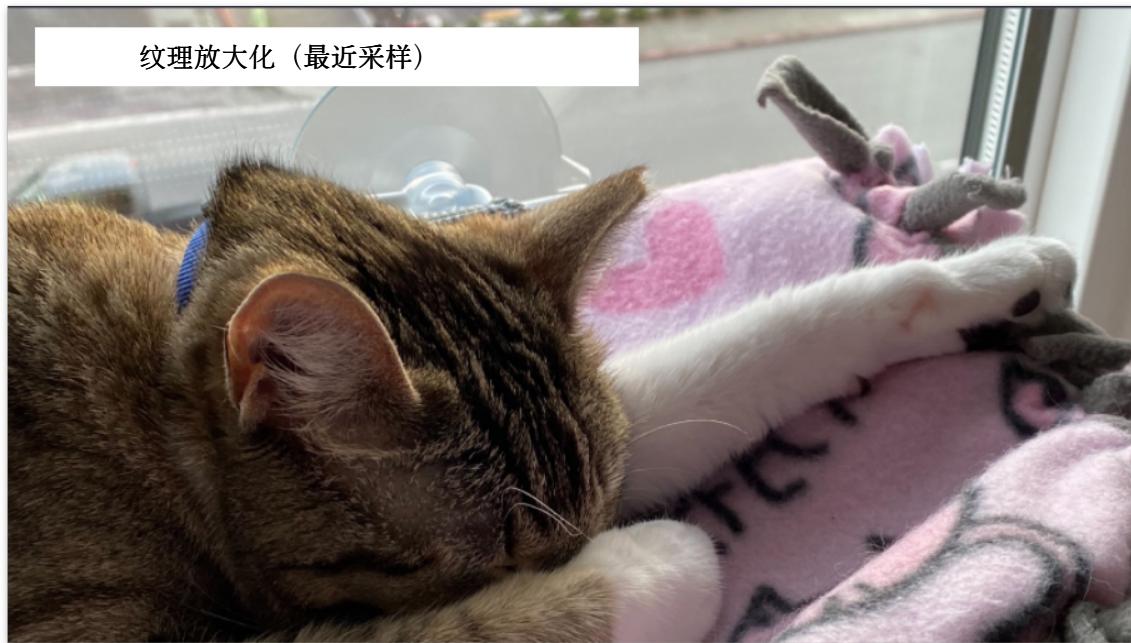
$f_{\text{recon}}(x)$ 近似出 $f(x)$



约瑟芬(那只图形猫)相片



纹理放大化(最近采样)



二



三



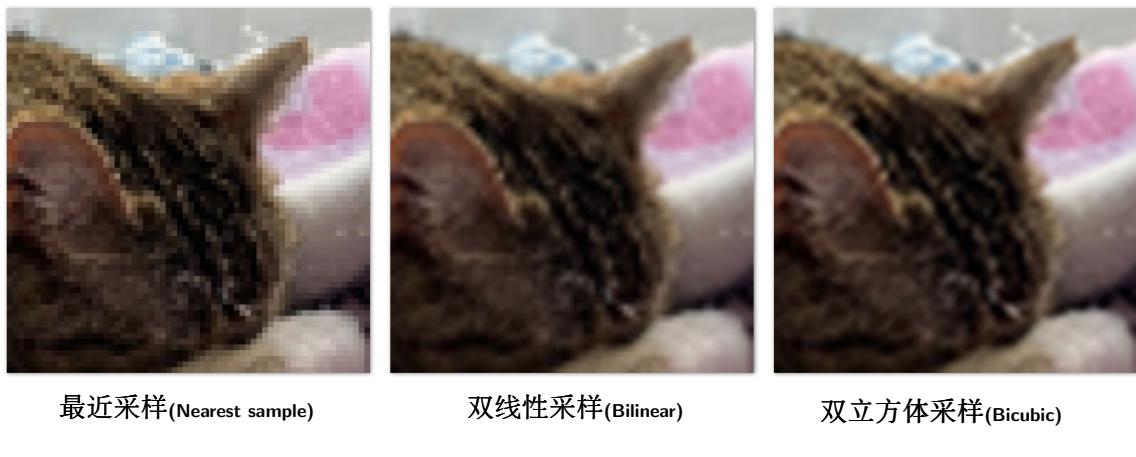
34

## 四



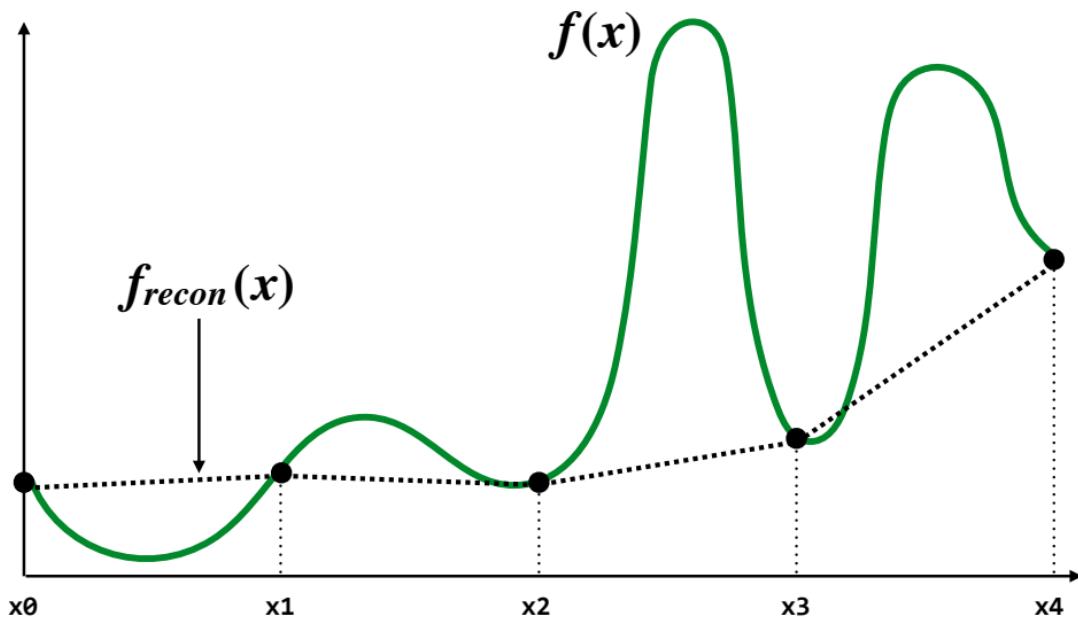
### 纹理放大化

- 通常不想要这种情形 | 这意味着我们没有足够的纹理分辨率
- 放大涉及到在纹理贴图中各种值的差值（下方：3种不同的kernel函数的差值）



## 回顾：逐段的线性近似

$f_{recon}(x)$  = 在两个最接近 $x$ 的样本之间的线性差值



纹理放大化（双线性采样）



二



三

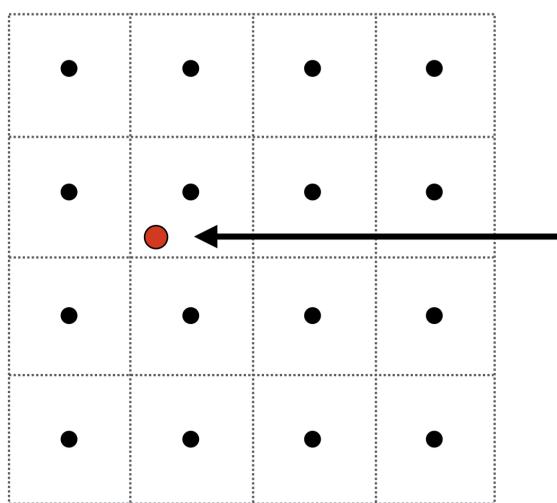


四



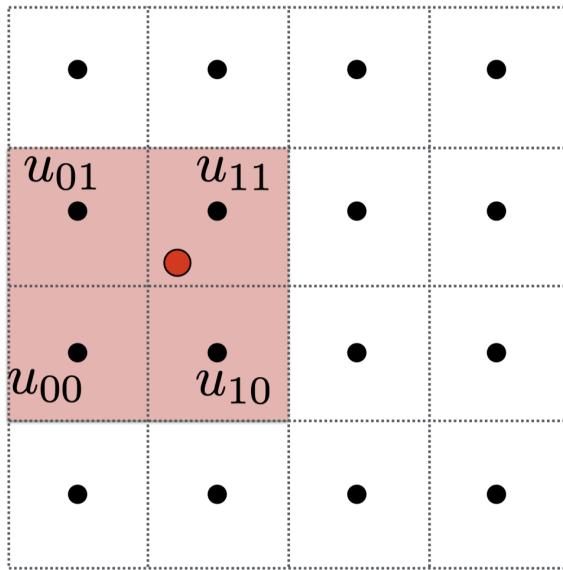
双线性插值

一



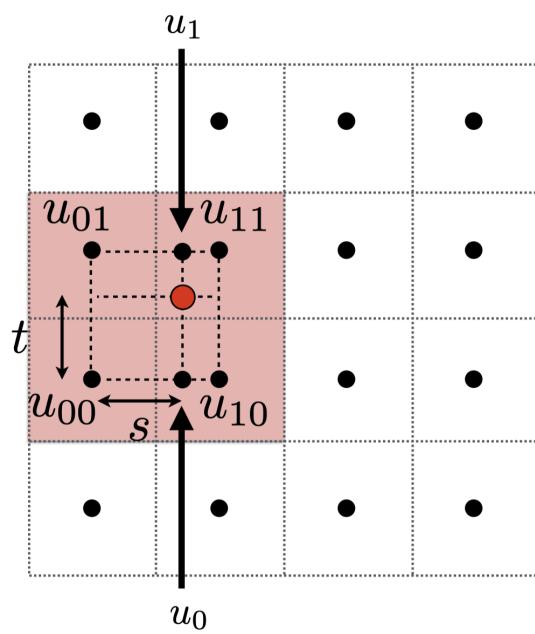
在红点处想要采样纹理值  $f(x, y)$   
黑点指示出了样本地址

二



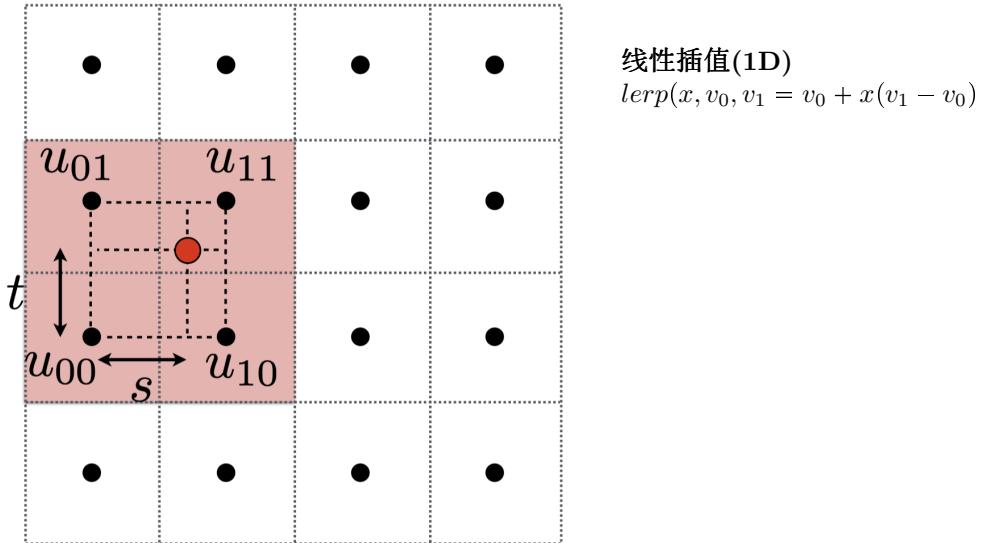
取四个最近样本地址，  
使用色块所标记的纹理值

三

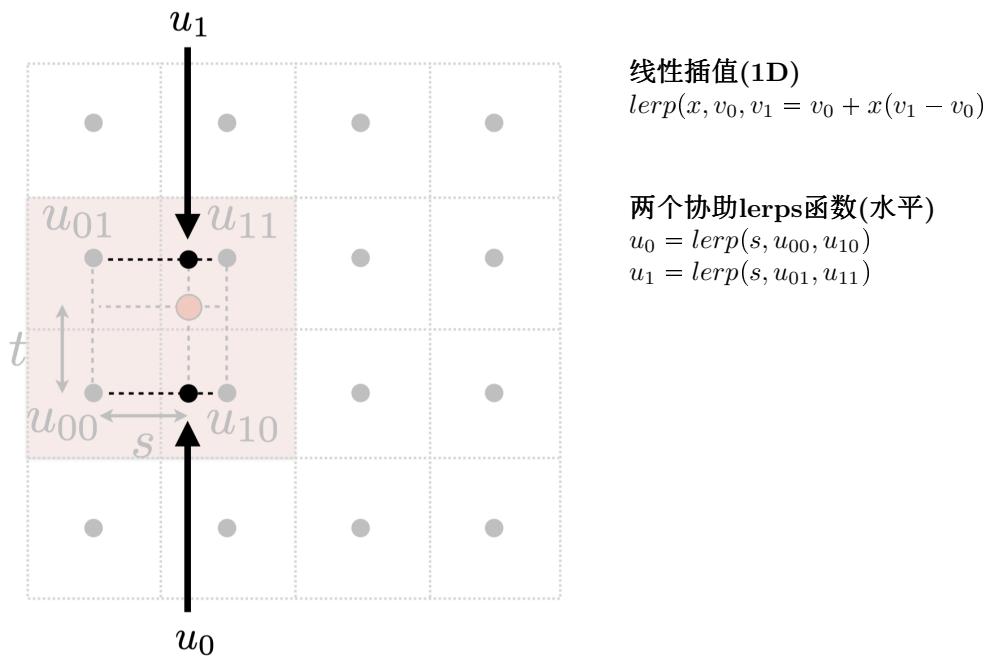


同时获取小数偏离， 显示为 $(s, t)$

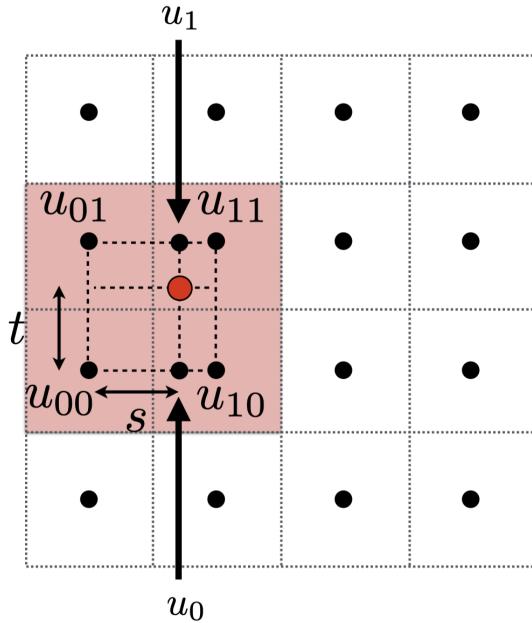
四



五



## 六



线性插值(1D)

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

两个协助lerps函数(水平)

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

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

最后垂直方向插值，以获取最终结果

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

## 纹理缩小化

目前为止，希望你已经认识到：

应用纹理就是一种采样的形式！

$$t(u, v)$$

## 约瑟芬缩小化

假设纹理贴图是9x9

同时被应用到一个  
跨3x3屏幕区域的方块

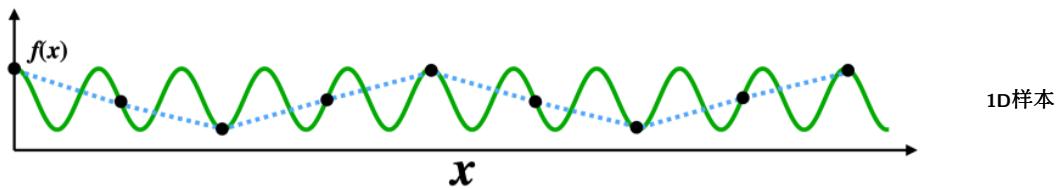


红点=需要渲染的样本  
白点=纹理贴图中的样本

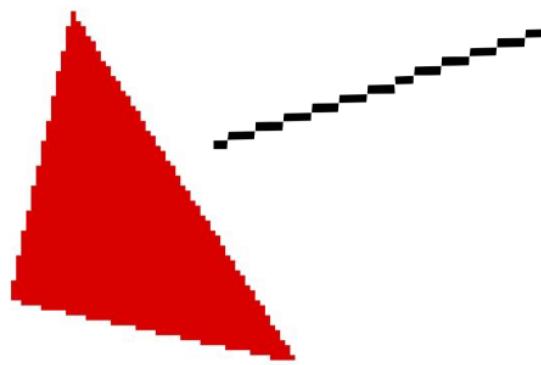
当纹理被缩小，纹理贴图被稀疏采样！

## 回顾：锯齿化

过低采样一个高频信号可能导致锯齿化



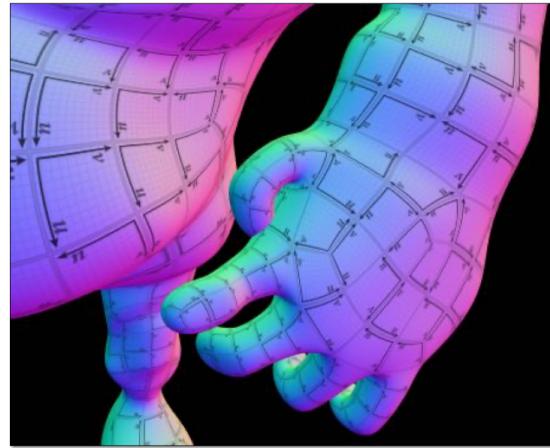
2D样本：  
云纹图案，锯齿



## 由于过低采样导致的锯齿化

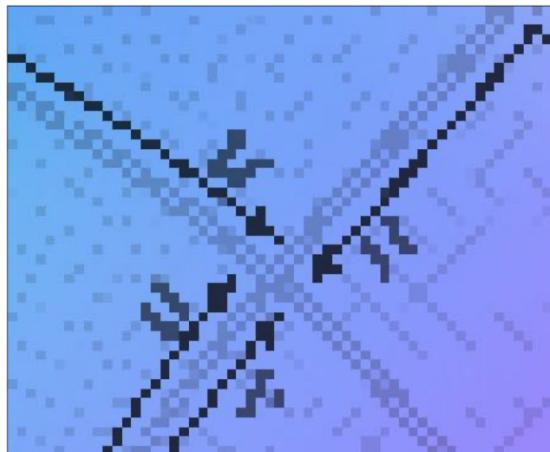


每像素一个纹理样本（锯齿化！）



抗锯齿纹理采样

## 由于过低采样导致的锯齿化(放大)

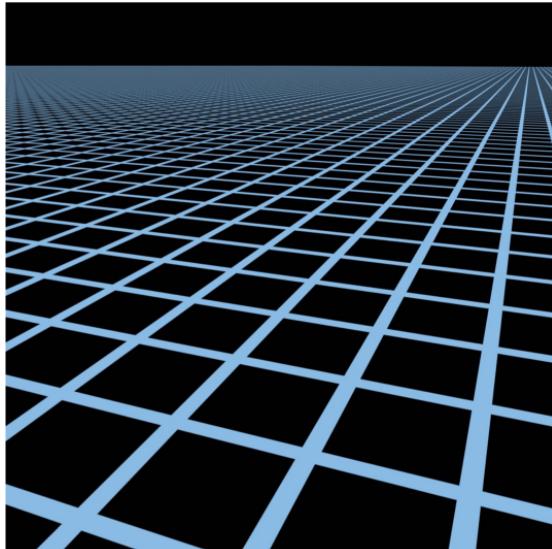


每像素一个纹理样本  
(锯齿化!)

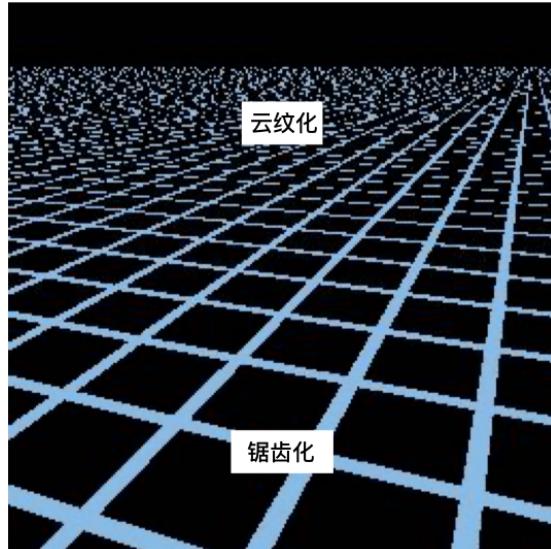


抗锯齿纹理采样

另一个例子



抗锯齿效果

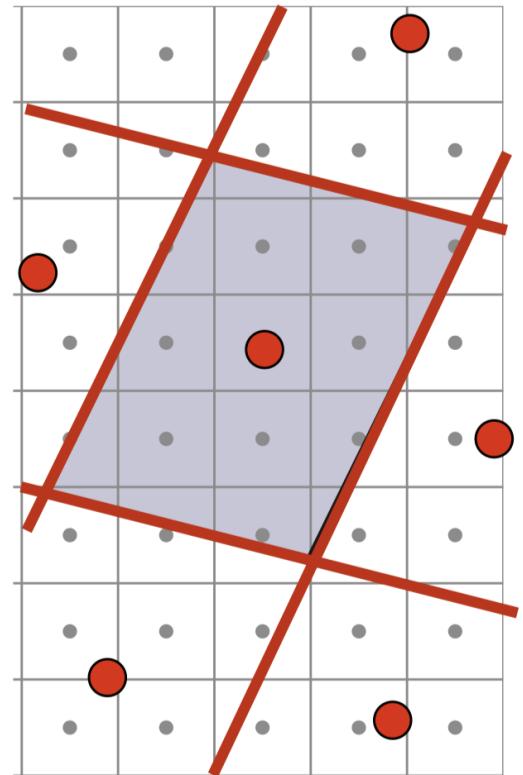


渲染图像: 256x256像素

## 纹理缩小化-不好处理的情形

### ■ 挑战:

- 很多纹理像素对于输出图像像素的色彩有贡献 (仅仅采样其中之一可能导致锯齿)
- 像素足迹的形状可能是复杂的



着色器区域=像素存在区域

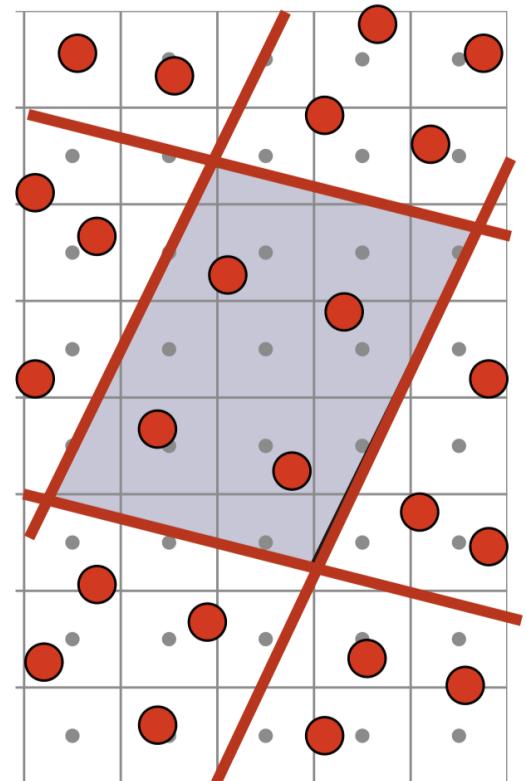
红线=屏幕像素范围

红点=用于临近像素的所有纹理空间  
样本点

■ 已经知道的一种解决方案：超采样

- 每像素平均很多纹理样本，借助像素区域尺寸大小的过滤器，可能近似出卷积化的纹理贴图结果
- 有没有问题？

可选解决方案：  
从纹理中移除高频以减轻锯齿化！



着色器区域=像素存在区域

红线=屏幕像素范围

红点=用于临近像素的所有纹理空间  
样本点

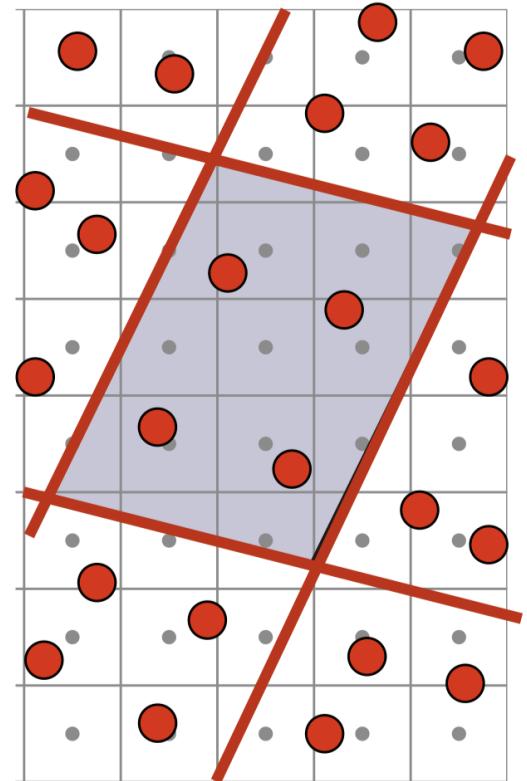
■ 挑战:

- 很多纹理像素对于输出图像像素的色彩有贡献 (只采样其中之一可能导致锯齿)
- 像素足迹的形状可能是复杂的

■ 已经知道的一种解决方案: 超采样

- 每像素平均很多纹理样本, 借助像素区域尺寸大小的过滤器, 可能近似出卷积化的纹理贴图结果
- 有没有问题?

可选解决方案:  
从纹理中移除高频以减轻锯齿化!

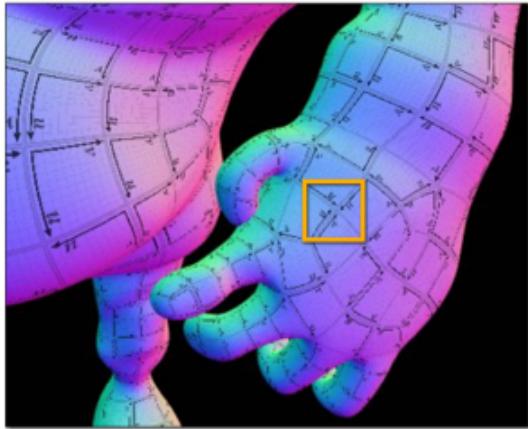


着色器区域=像素存在区域

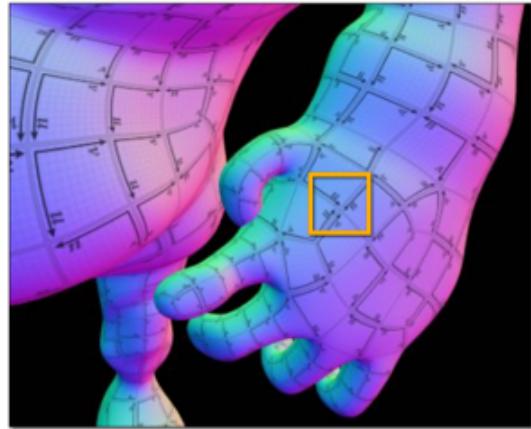
红线=屏幕像素范围

红点=用于临近像素的所有纹理空间  
样本点

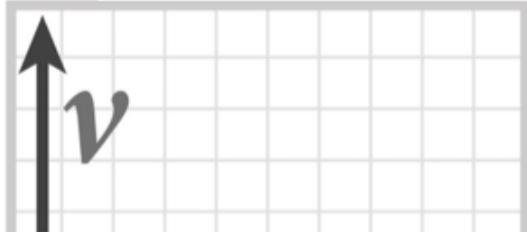
## 预过滤(pre-filtering)纹理贴图减轻锯齿化

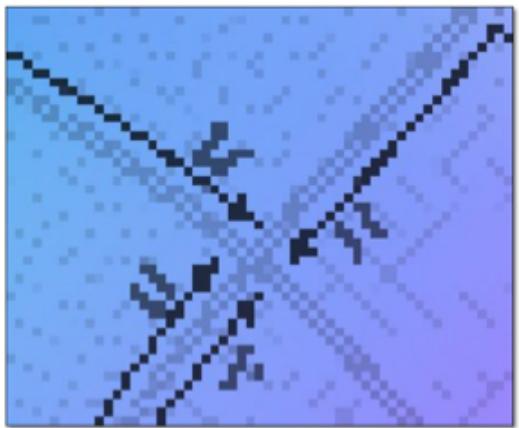


每像素一个纹理样本  
(锯齿化! )



预过滤的纹理贴图  
(高频移除)





非纹理数据的预先过滤  
(导致图像出现锯齿)



预先过滤的纹理贴图  
(高频移除)



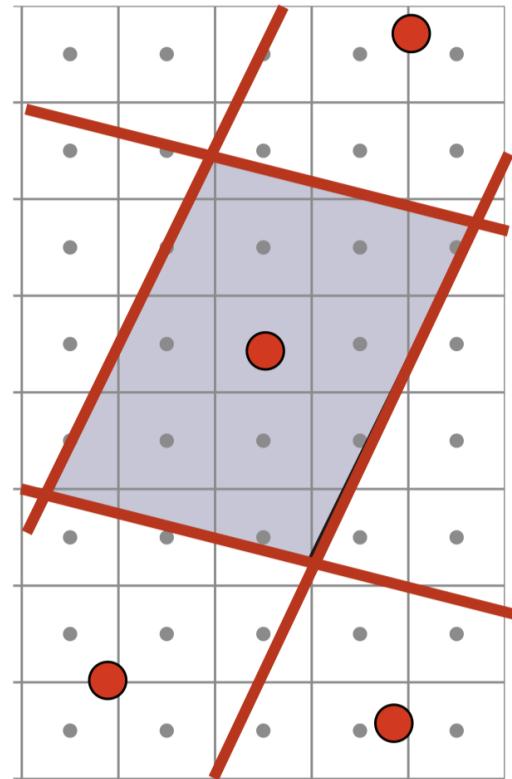
## 但是我们应该预先过滤多少呢？

- 预过滤的总量依赖于目标离开的有多远：

- 轻微的缩小：图像像素极限  
放大：图像像素跨越了纹理的大片区域

- 思路：

- 低通过滤器和低采样纹理文件，同时存储依次更低的解析分辨率
- 针对每个样本，借助纹理文件，其分辨率近似屏幕采样率



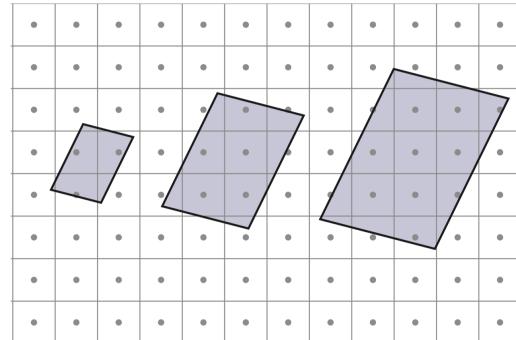
着色器区域=像素存在区域

红线=屏幕像素范围

红点=用于临近像素的所有纹理空间  
样本点

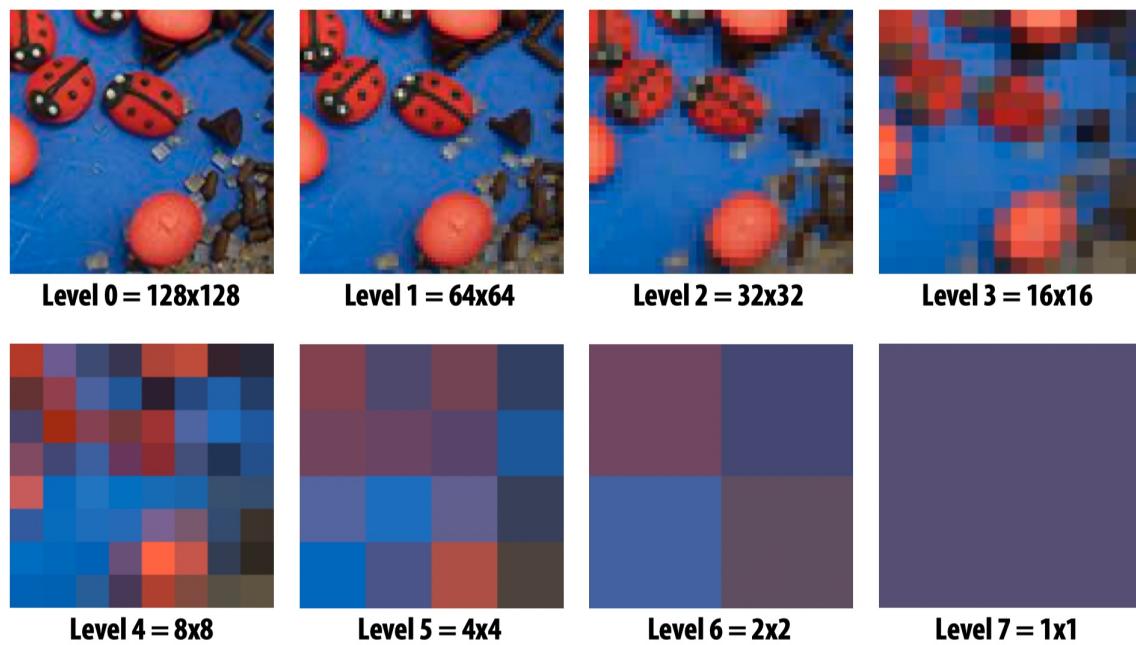
## 但是我们应该预先过滤多少呢?

- 预过滤的总量依赖于目标离开的有多远:
  - 轻微的缩小: 图像像素极限
  - 放大: 图像像素跨越了纹理的大片区域
- 思路:
  - 低通过滤器和低采样纹理文件, 同时存储依次更低的解析分辨率
  - 当采样纹理时, 借助纹理文件, 其预过滤总量匹配想要的采样率

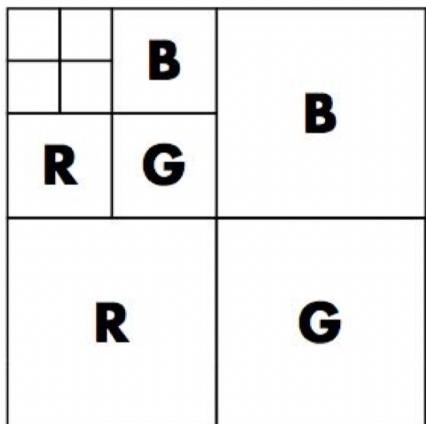


## 渐进纹理贴图-Mipmap(L.Williams 83)

每个渐进纹理贴图级别被低采样（低通过滤）之前的版本

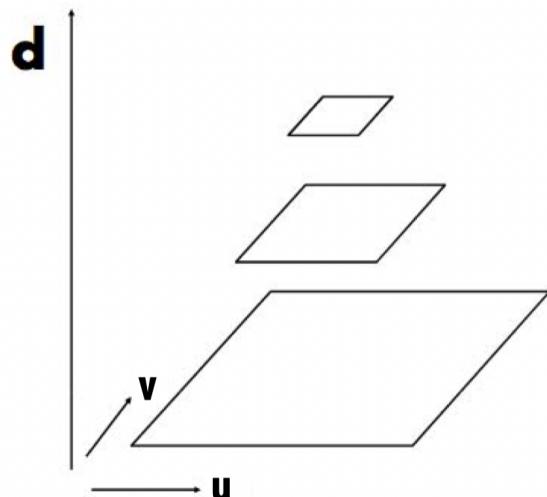


”Mip”来自于拉丁文”multum in parvo”,意指在小空间中有很多(相似的东西)(a great deal in small space)。



W William 最初所建议的 ~~red~~

~~min - max - linear~~  
渐进式映射(mip-map)布局



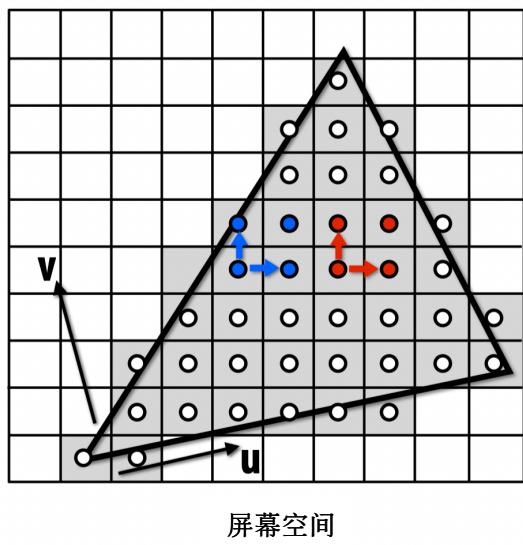
“渐进式层级”

**level = d**

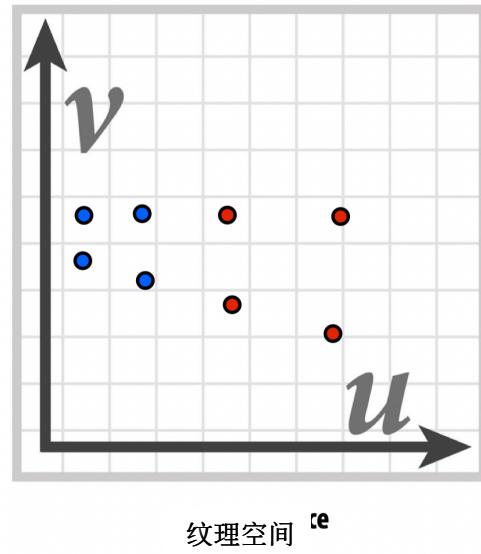
渐进式纹理贴图的存储开销是什么？

## 计算渐进式映射的级别

计算临近屏幕样本坐标值之间的差异

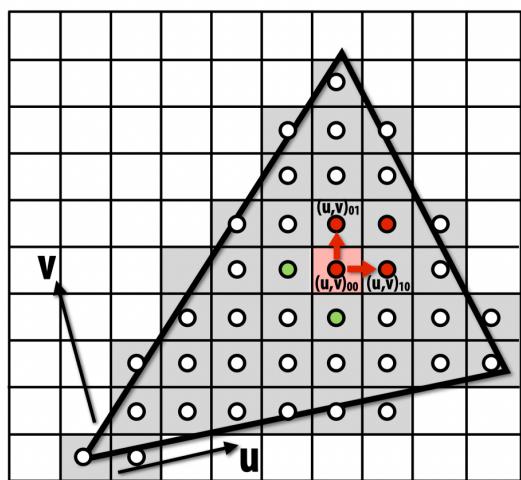


屏幕空间

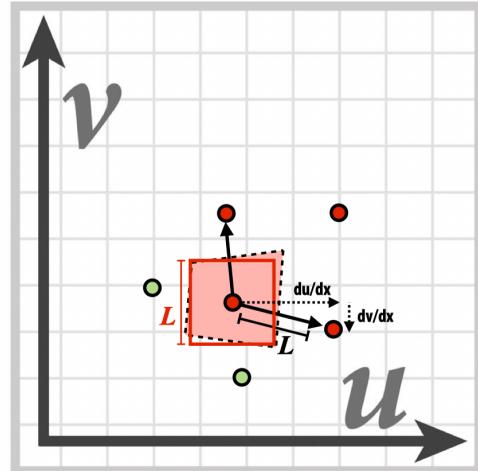


纹理空间

计算临近屏幕样本坐标值之间的差异



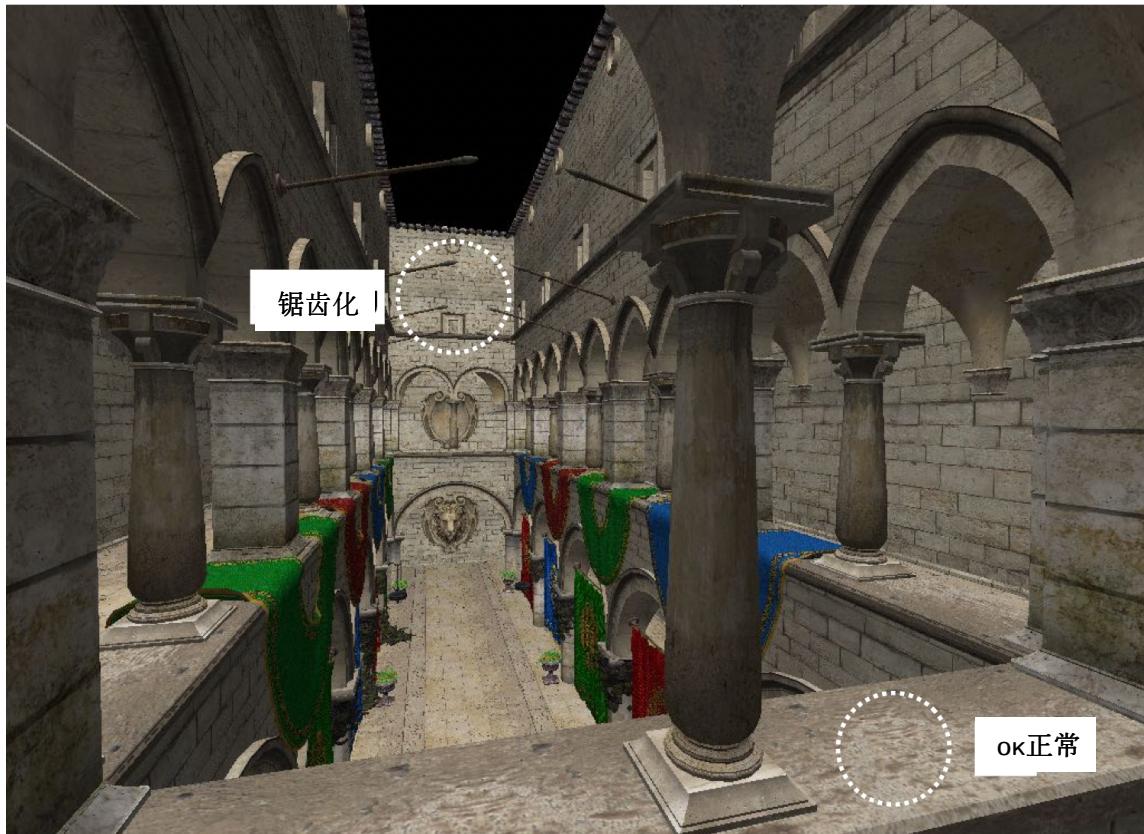
$$\begin{aligned} \frac{du}{dx} &= u_{10} - u_{00} & \frac{dv}{dx} &= v_{10} - v_{00} \\ \frac{du}{dy} &= u_{01} - u_{00} & \frac{dv}{dy} &= v_{01} - v_{00} \end{aligned}$$



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

*mip-map d = log<sub>2</sub> L*

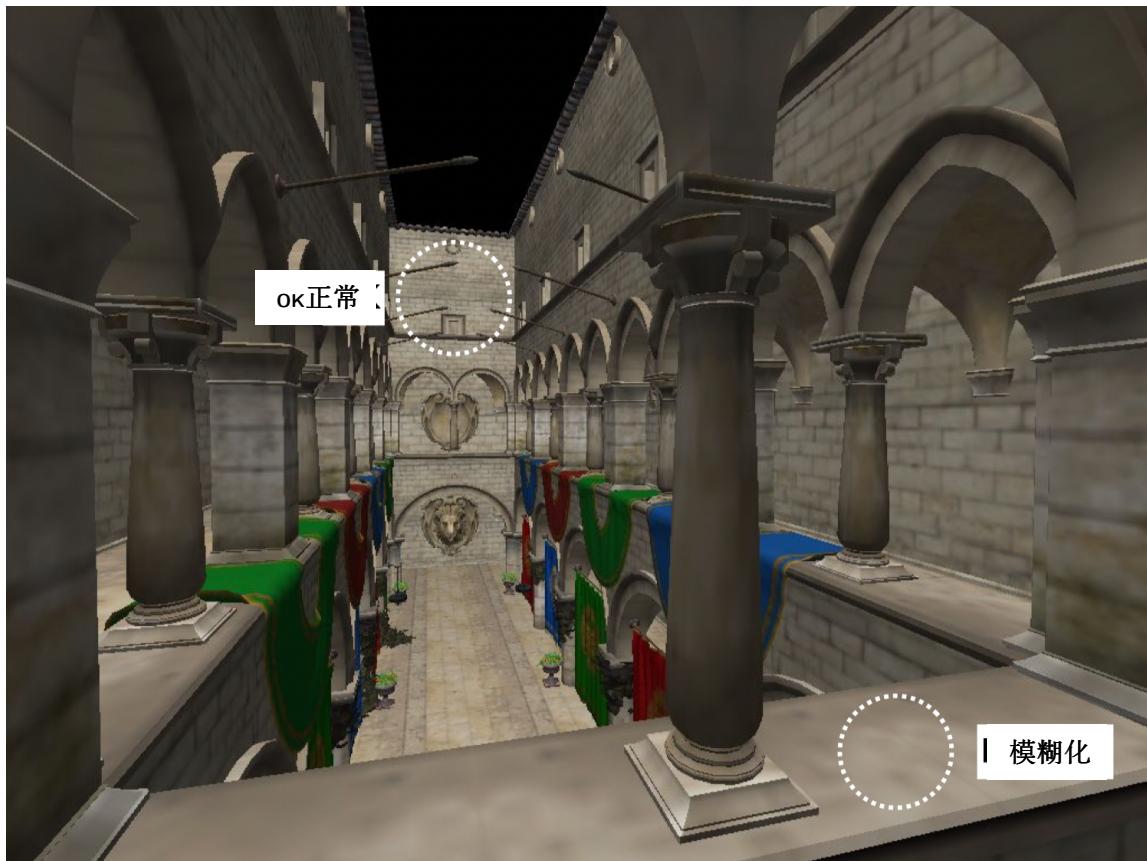
在级别0的双线性再采样



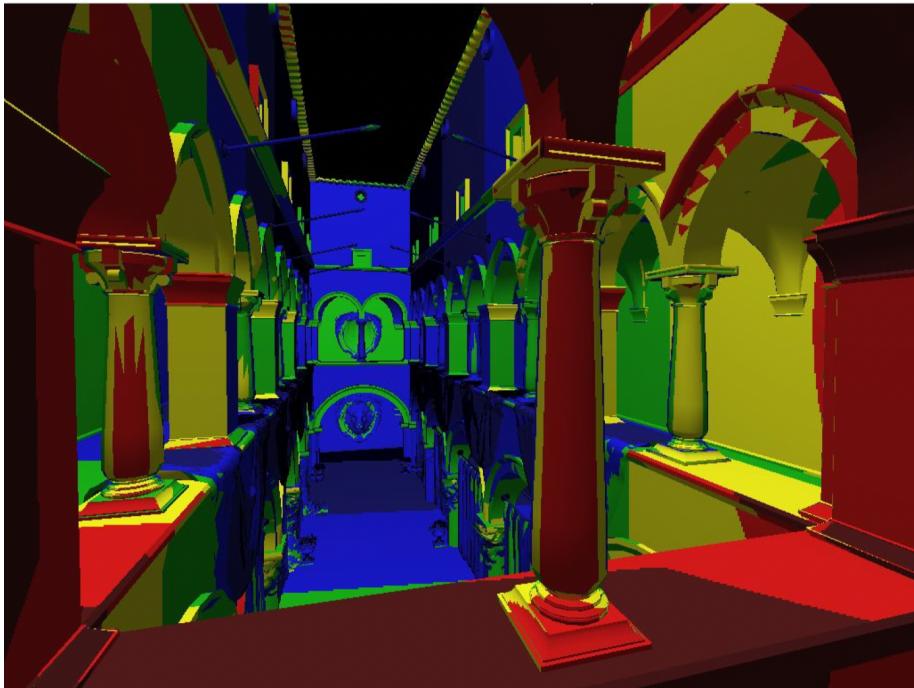
在级别2的双线性再采样



在级别4的双线性再采样



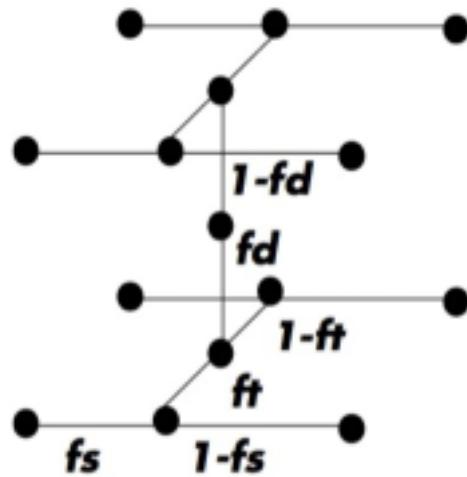
## 渐进映射级别的可视化



仅仅双线性过滤:  $d$ 贴近最近级别

## “三线性”过滤

线性插值从两个临近的渐进映射级别的双线性插值结果  
(在两个不同预过滤级别间平滑过渡)



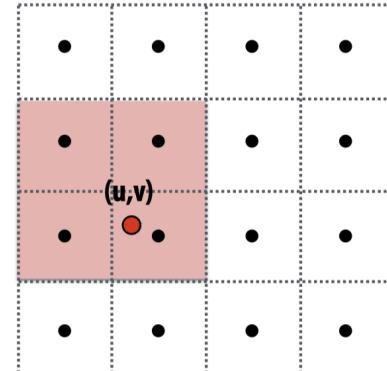
$$\text{lerp}(t, \mathbf{v}_1, \mathbf{v}_2) = \mathbf{v}_1 + t(\mathbf{v}_2 - \mathbf{v}_1)$$

双线性再采样:

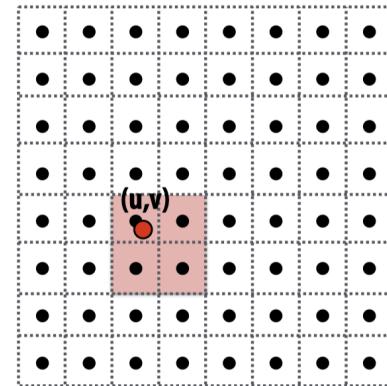
四个纹理像素读取, 3个lerps操作(3个乘法 + 6个加法)

三线性再采样:

八个纹理像素读取, 7个lerps操作(7个乘法 + 14个加法)

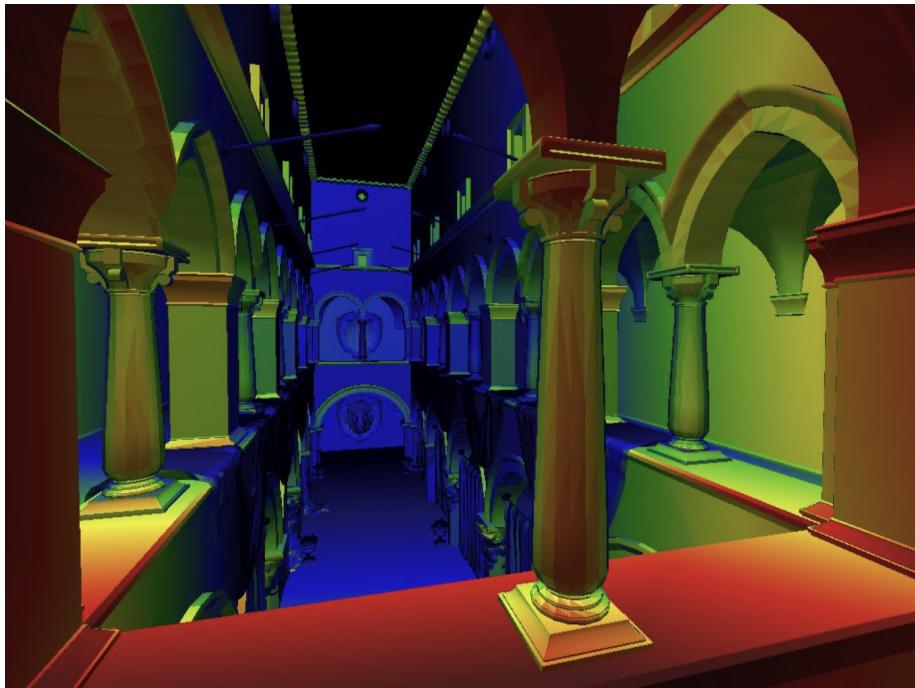


渐进式映射纹理像素: 级别  $d+1$



渐进式映射纹理像素: 级别  $d$

## 渐进映射级别的可视化

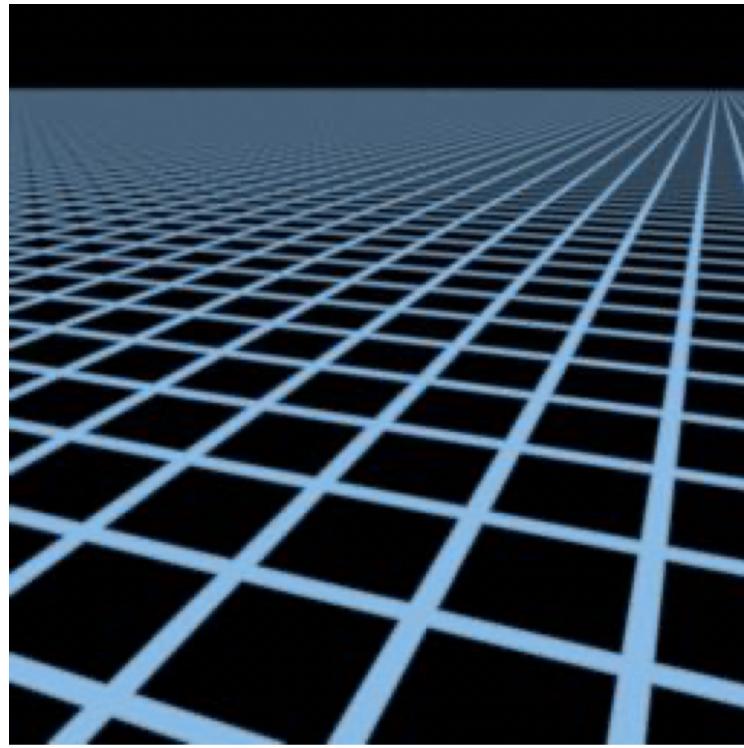


(三线性过滤: 连续级别值 $d$ 的可视化)

## 双线性VS三线性过滤的代价

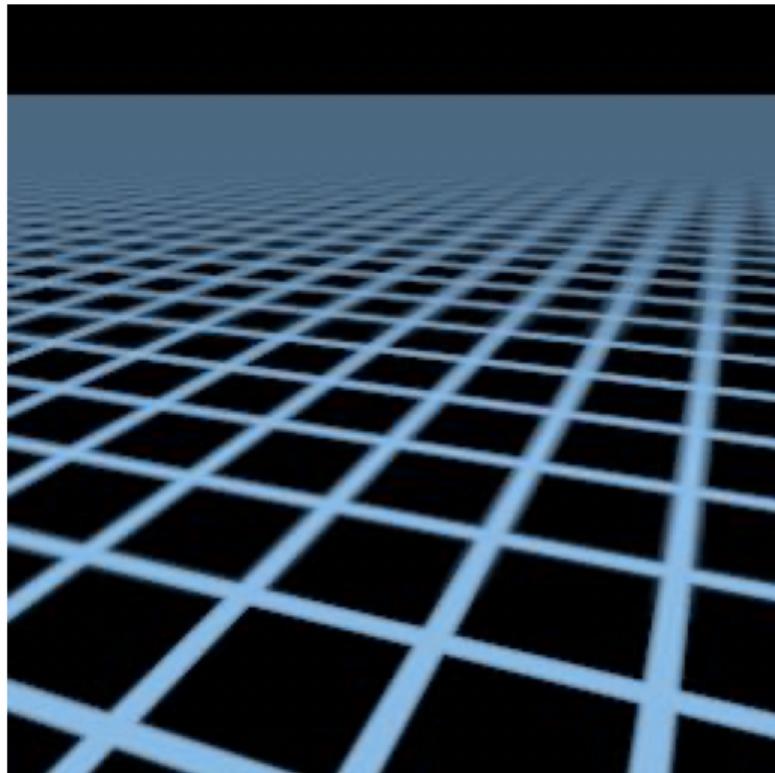
- 双线性再采样:
  - 四个纹理像素读取
  - 3个lerps操作(3个乘法 + 6个加法)
- 三线性再采样:
  - 八个纹理像素读取
  - 7个lerps操作(7个乘法 + 14个加法)

样例：渐进式映射极限



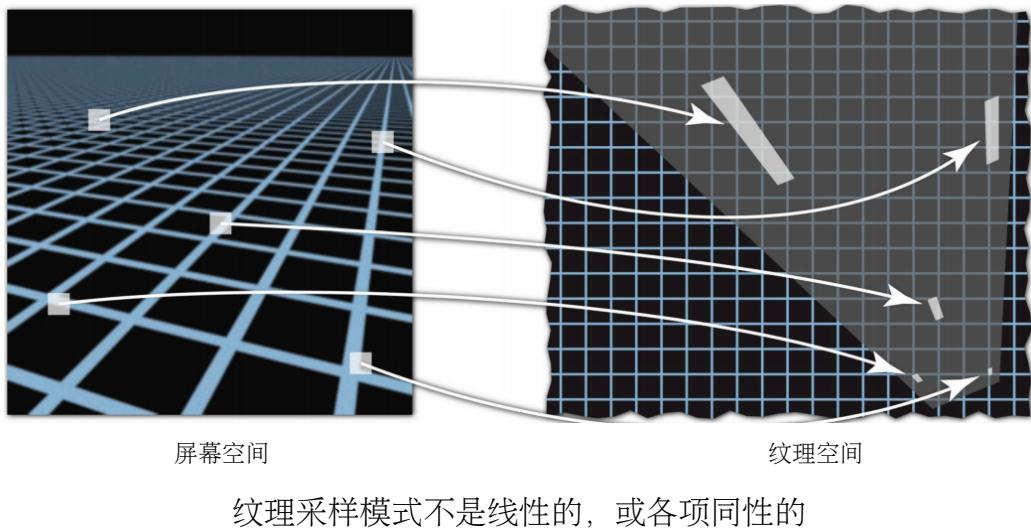
超采样：每像素点采样512纹理样本  
(想要的结果)

过于模糊,为啥?



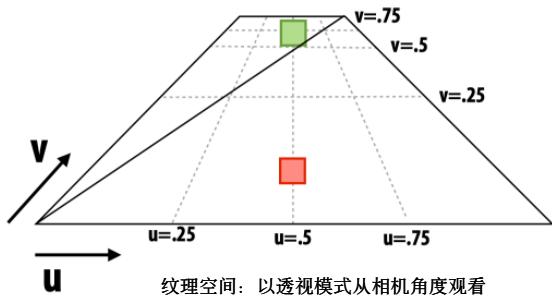
渐进式映射三线性采样

## 纹理空间中的屏幕像素足迹

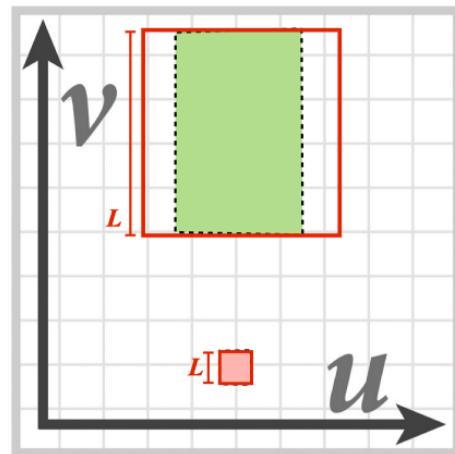
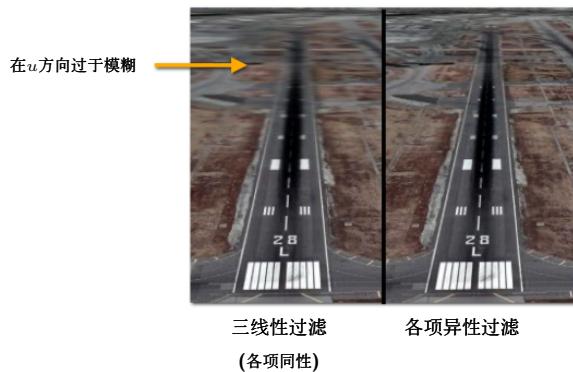


在纹理空间中像素区域可能不会映射到各项同性区域

正确过滤要求各项异性技术来过滤足迹

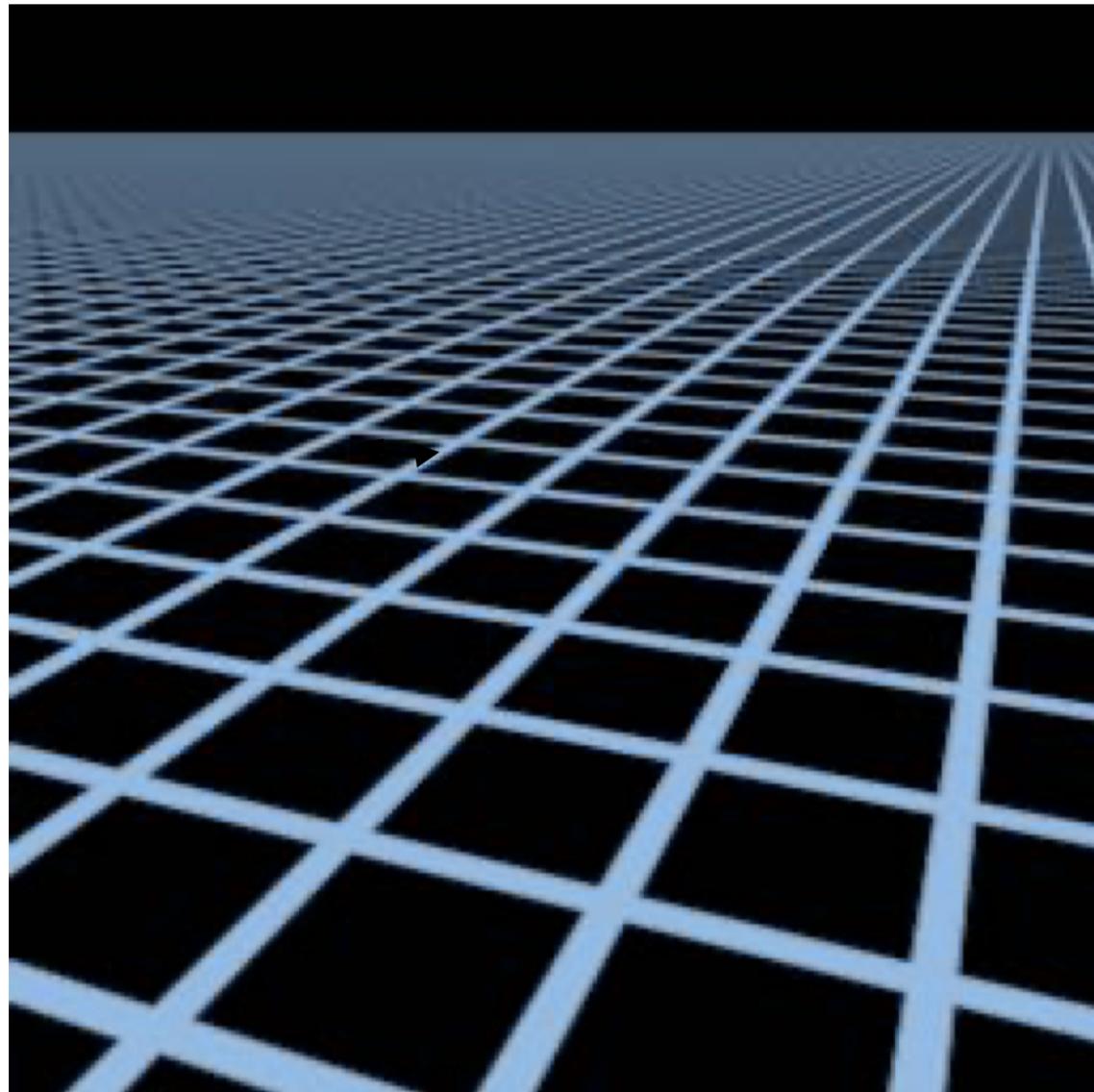


纹理空间：以透视模式从相机角度观看



(当代的各项异性纹理过滤解决方案合  
并多渐进式映射样本以近似在任意纹理  
空间区域上的纹理值的积分)

## 各项异性过滤



椭圆加权平均(EWA)过滤  
(借助在渐进式映射中的多次查找来近似出过滤区域)

## 总结：借助渐进式映射的纹理过滤

- 小存储开销 (33%):

- 渐进式映射纹理贴图为原始纹理贴图的 $\frac{4}{3}$ 尺寸大小
- 针对每个各项同性过滤的采样操作:
  - 常量过滤操作代价（无关渐进式映射级别）
  - 常量纹理像素访问数量（无关渐进式映射级别）
- 借助预过滤，而不是超采样技术抗锯齿
  - 回忆一下：当采样覆盖度时，我们使用超采样来定位锯齿问题
- 双线性三线性过滤是各项同性的，因而抗锯齿时将会“过于模糊”
  - 各项异性纹理过滤技术，以更高的计算量和内存带宽为代价，提供了更高的图像质量  
(实践：多渐进式映射采样)

## 完整纹理采样操作

1. 从屏幕样本 $x, y$ 计算 $u$ 和 $v$ （经由属性方程式的评估）
2. 从临近屏幕样本计算 $\frac{du}{dx}, \frac{du}{dy}, \frac{dv}{dx}, \frac{dv}{dy}$ 微分
3. 计算渐进式映射级别 $d$
4. 转换标准化的 $[0, 1]$ 域的纹理坐标 $(u, v)$ 到 $[W, H]$ 域的纹理坐标 $(U, V)$
5. 计算要过滤窗口中的必要像素
6. 从内存中加载必要的纹理像素（针对三线性过滤需要八个纹理像素）
7. 根据 $(U, V, d)$ 执行三线性插值

总结：纹理采样操作不只是图片像素查找！  
这会涉及大量数学操作。

由于这个缘故，现代的各类GPU都有专门的固定功能硬件  
用于支持执行纹理采样操作。

## 总结：纹理映射

- 纹理：用于为表面添加在几何体中没有捕获的可视细节
- 纹理坐标：定义三角形表面点（物体坐标空间）到纹理坐标空间中点之间的映射
- 纹理映射为采样操作且易于产生锯齿
  - 解决方案：预计算和存储多个纹理图片的再采样版本（每个使用不同数量的低通过滤操作以移除数量不断增加的高频细节）
  - 在渲染中：基于纹理空间中临近屏幕样本间的距离来动态选择多少低通过滤操作是必要的
    - 目标是在纹理中尽可能多的保留高频内容（细节），同时避免锯齿化

## 致谢

感谢Ren Ng, Pat Hanrahan, 和Keenan Crane提供的幻灯材料。