中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

Lecture 22：期末复习

SSE315：计算机图形学
Computer Graphics

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn

重点复习部分

- **Lecture04：光栅化 (Rasterization)**
- **Lecture05：采样 (Sampling)**
- **Lecture06：空间变换**
- Lecture07：3D 空间变换
- Lecture09：深度和透明度
- **Lecture10：几何**
- Lecture11：网格和流形
- **Lecture13：几何查询**
- **Lecture14：几何查询加速**
- Lecture15：着色和阴影

Lecture04：光栅化
Rasterization

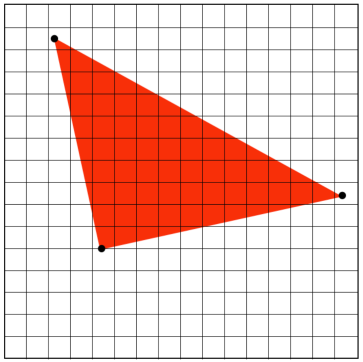
两种把图像“放到”屏幕上的技术

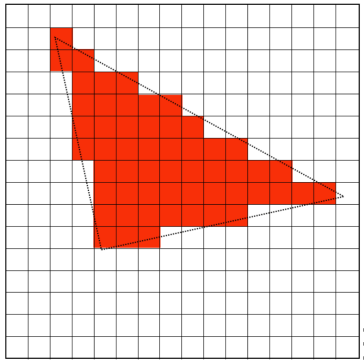
光栅化 Rasterization

- 对于每个图形基元 (primitive)，如三角形，点亮哪些像素
- 非常快（在 GPU 中数十亿个三角形每秒）
- 很难（但并非不可能）实现真实图片效果
- 非常适合 2D vector art, fonts, quick 3D preview

光线追踪 Ray tracing

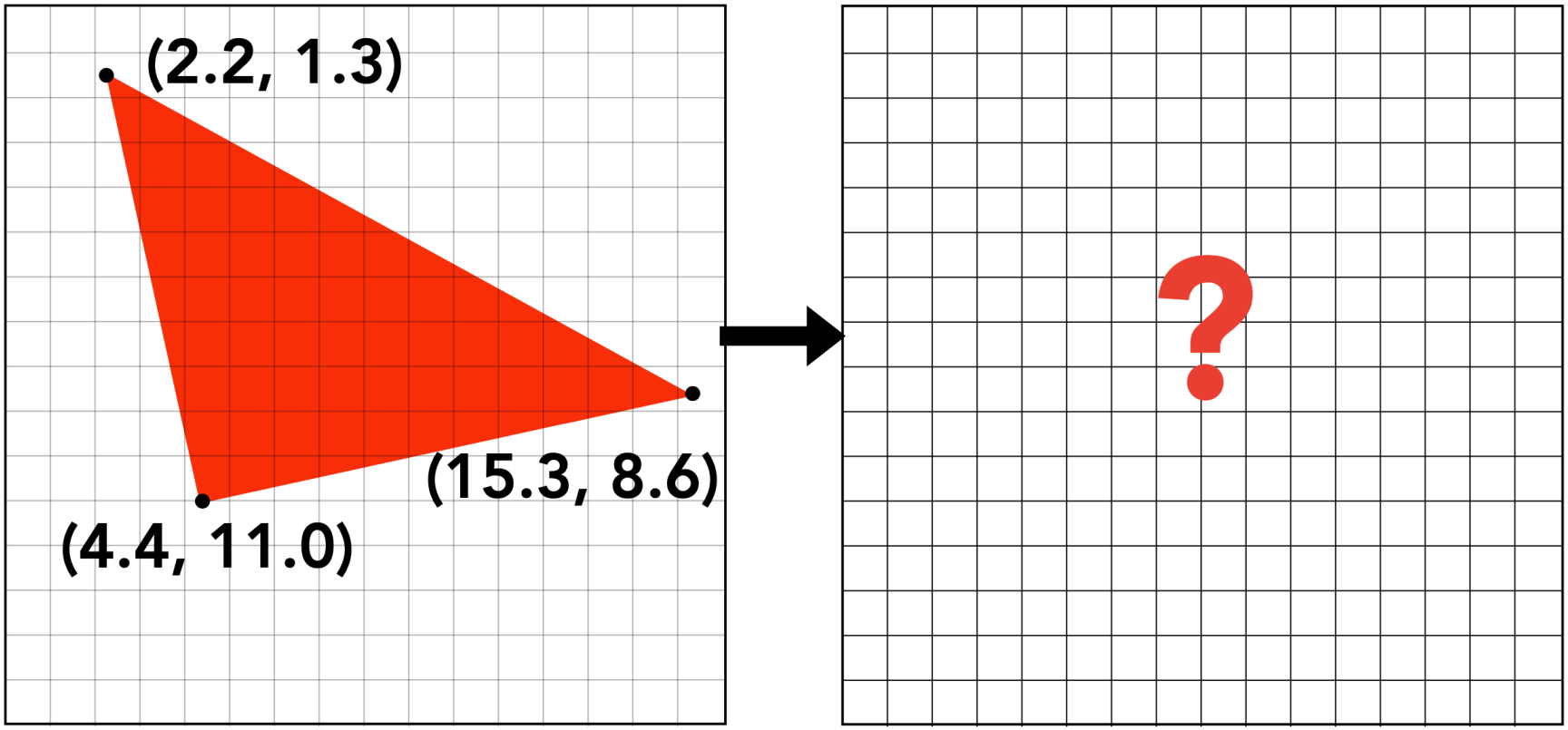
- 对于每个像素，我们可以看到哪些图形基元
- 很容易实现真实图片效果
- 一般来讲很慢





三角形呢?

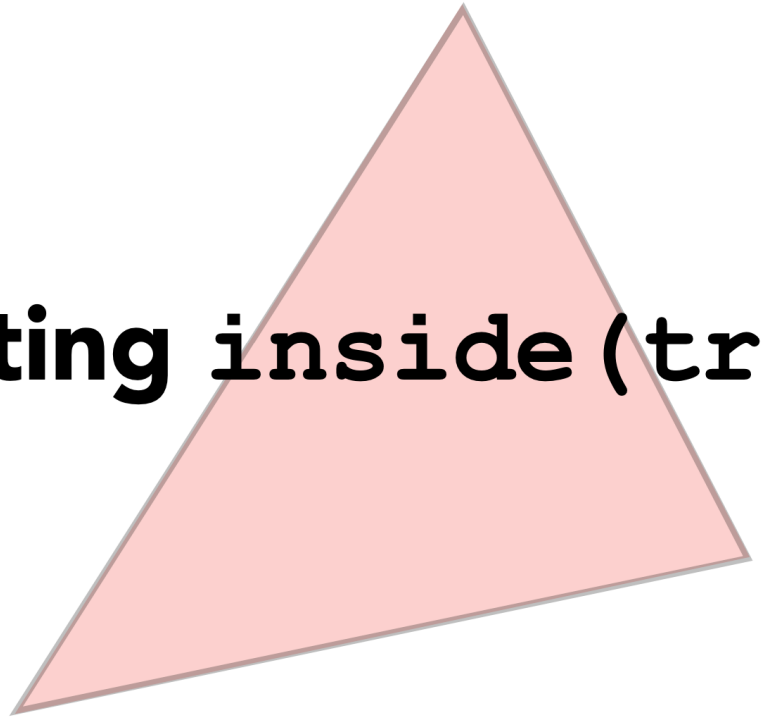
- 填充屏幕中哪些像素才能绘制三角形?



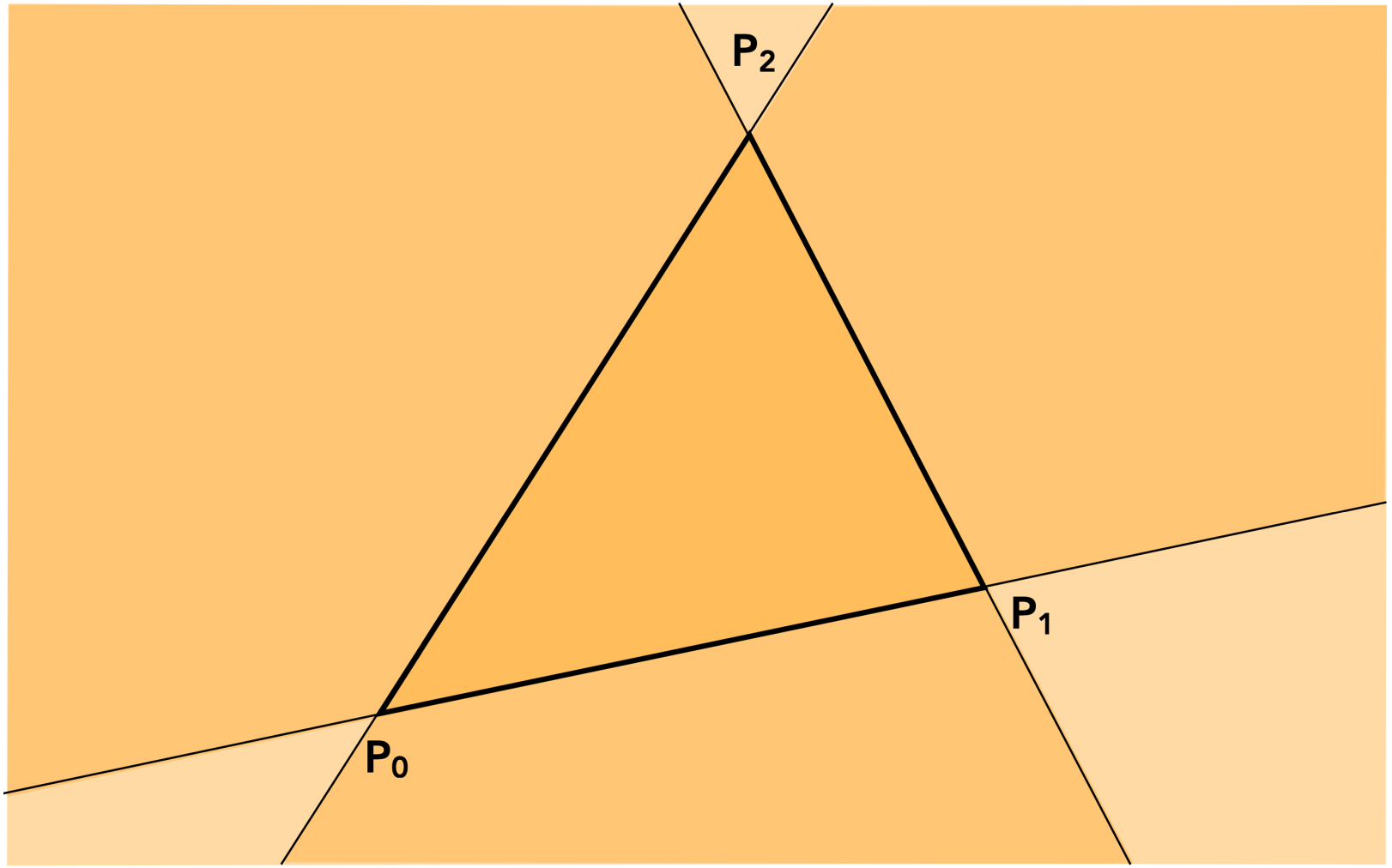
Input: position of triangle vertices projected on screen

Output: set of pixel values approximating triangle

Evaluating `inside(tri,x,y)`



三角形 = 三个半平面的交集

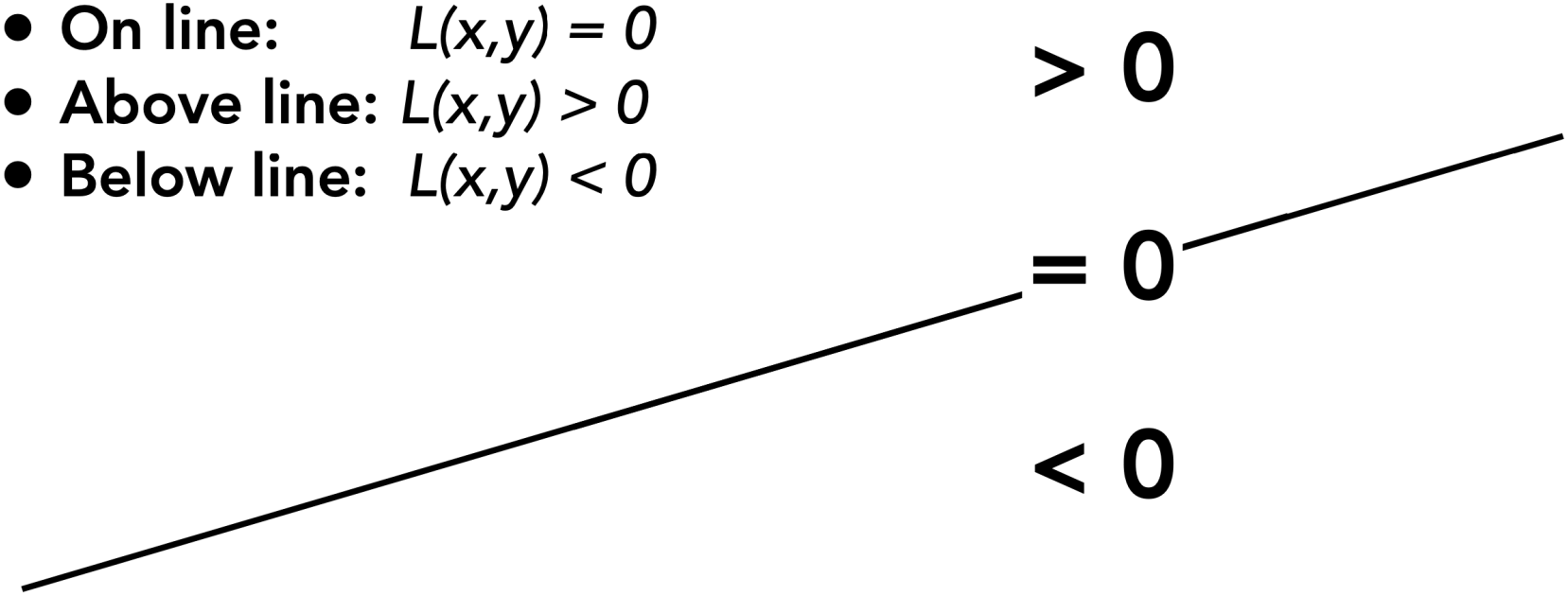


一条线定义一个半平面

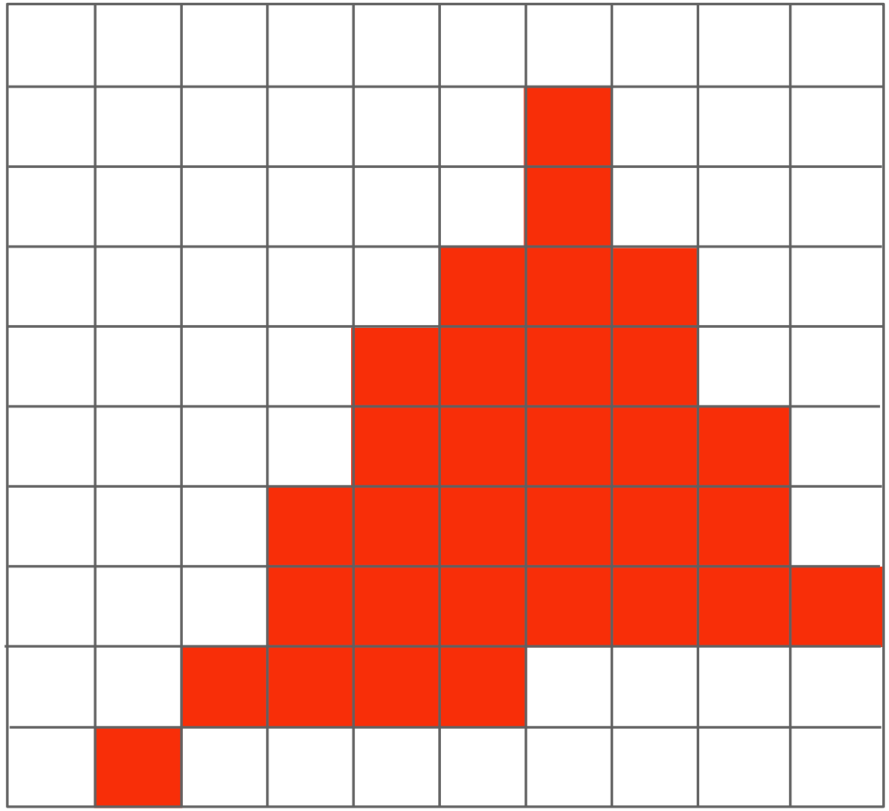
Implicit line equation

- $L(x,y) = Ax + By + C$

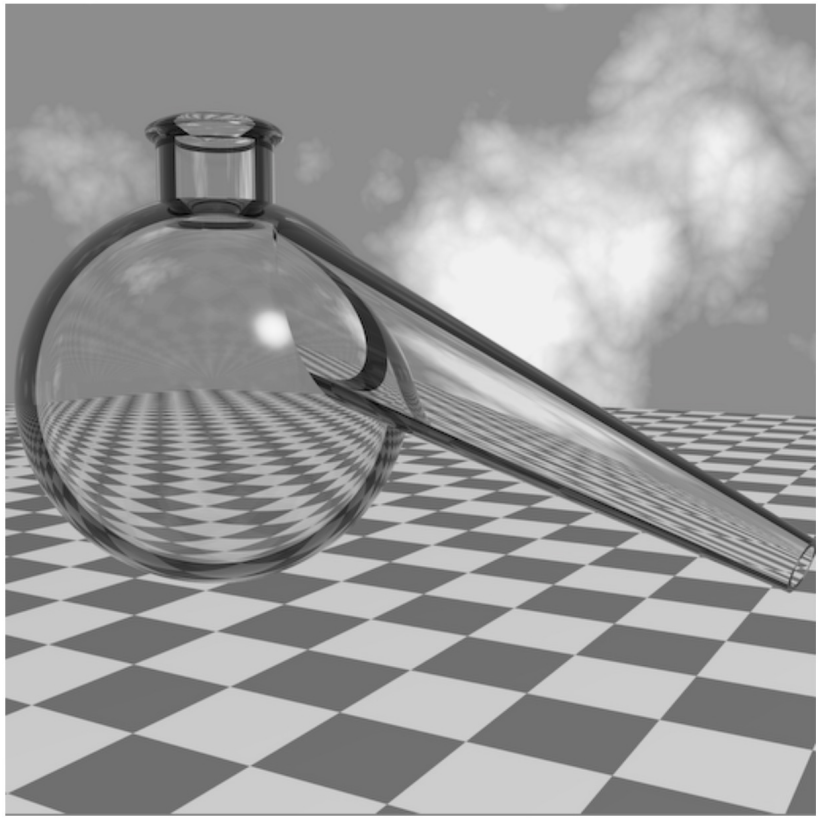
- **On line:** $L(x,y) = 0$
- **Above line:** $L(x,y) > 0$
- **Below line:** $L(x,y) < 0$

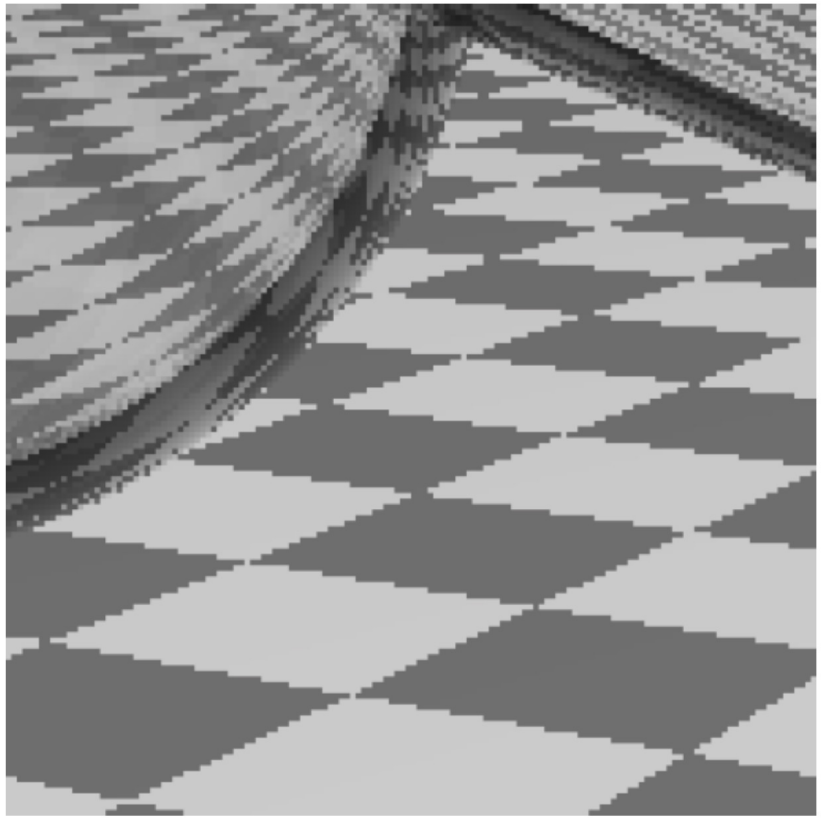


显示器将展现如下图片



Jaggies（楼梯样式）



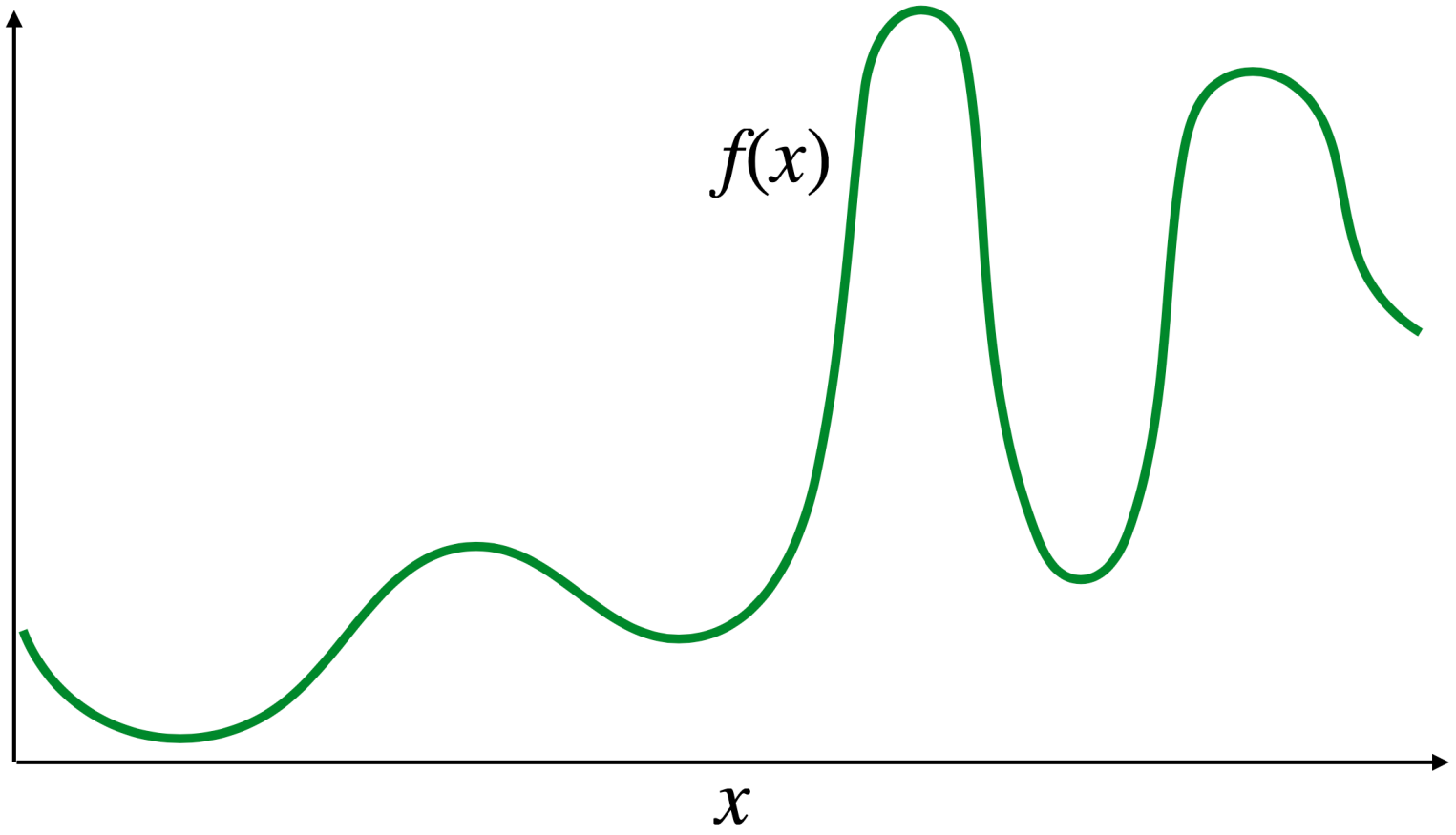


摩尔条纹 Moiré Patterns

Lecture05：采样
Sampling

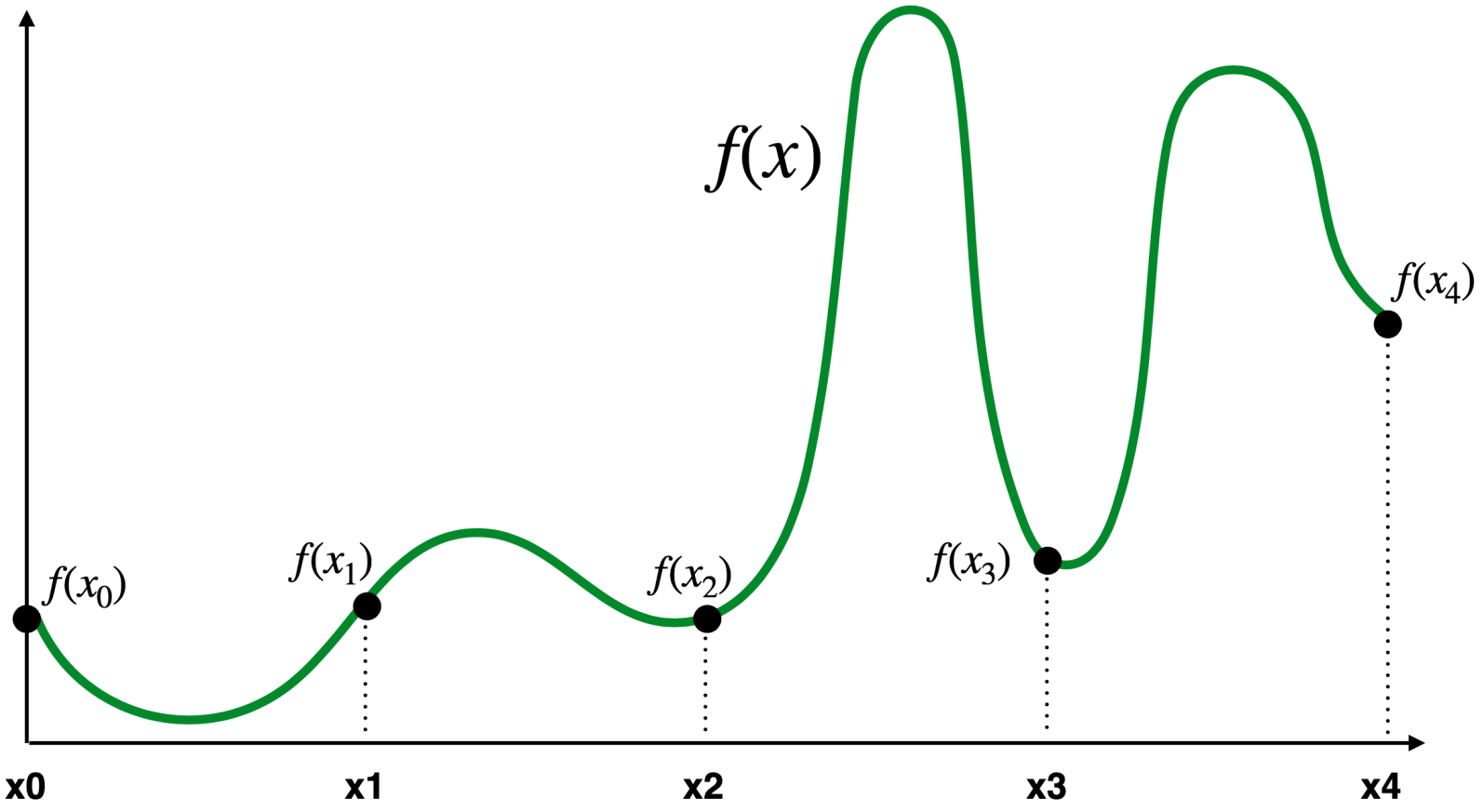
采样 101：采样 1D 的信号

- 采样 Sampling



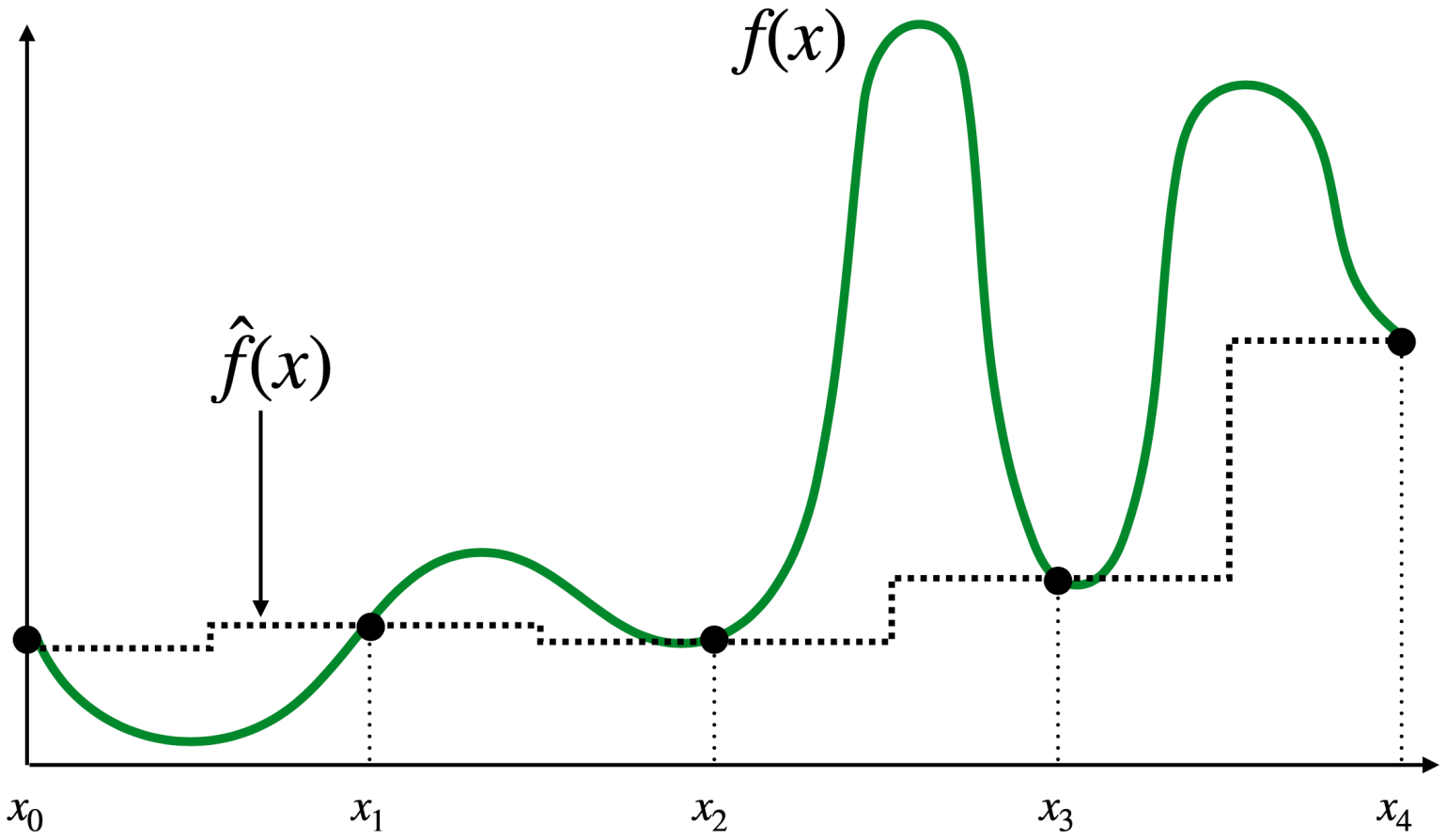
采样 = 对信号进行测量

- 以下为 $f(x)$ 的 5 个测量点（采样点）



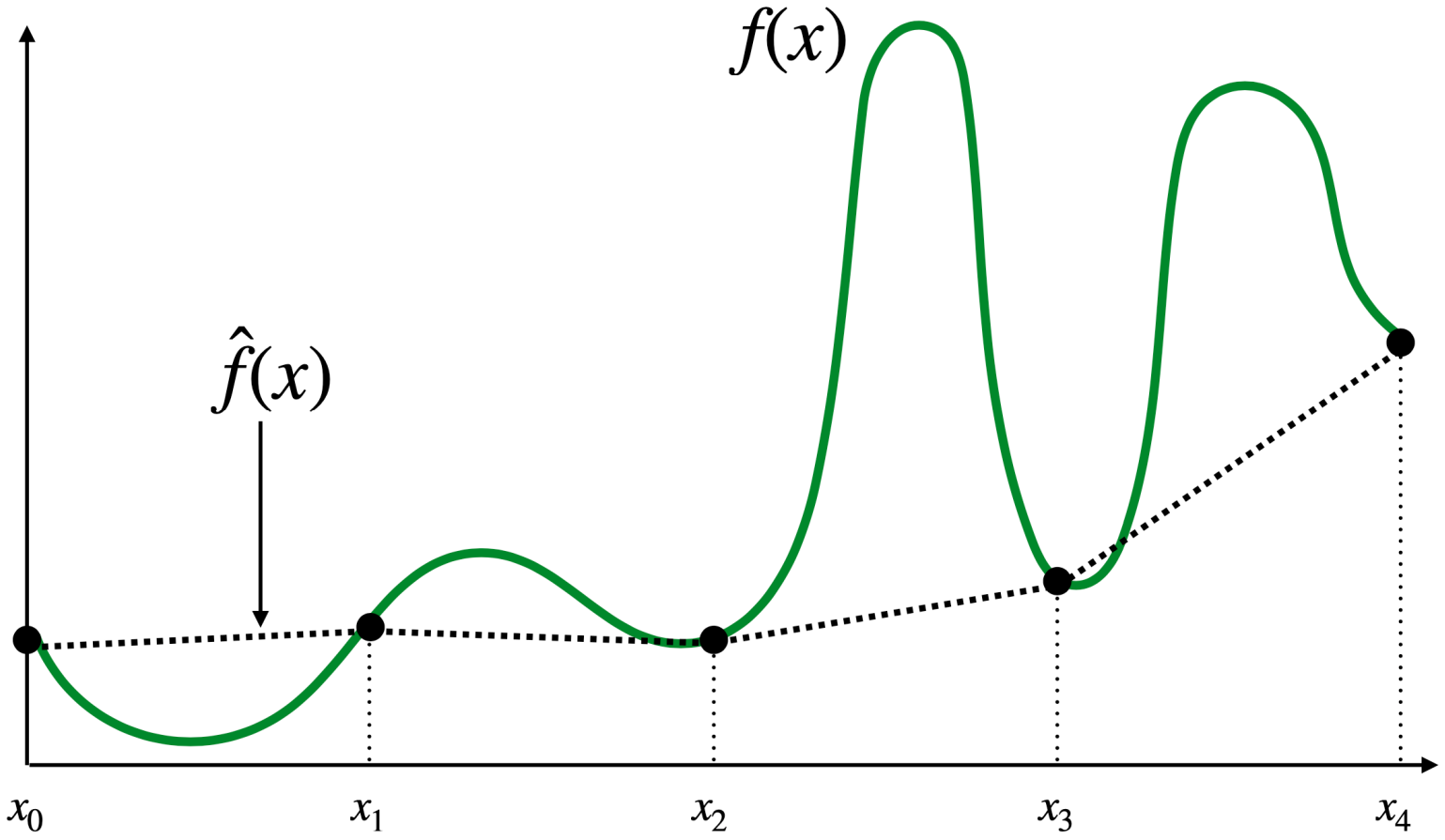
分段常数近似 Piecewise constant approximation

- $\hat{f}(x)$ = 最接近 x 的样本值



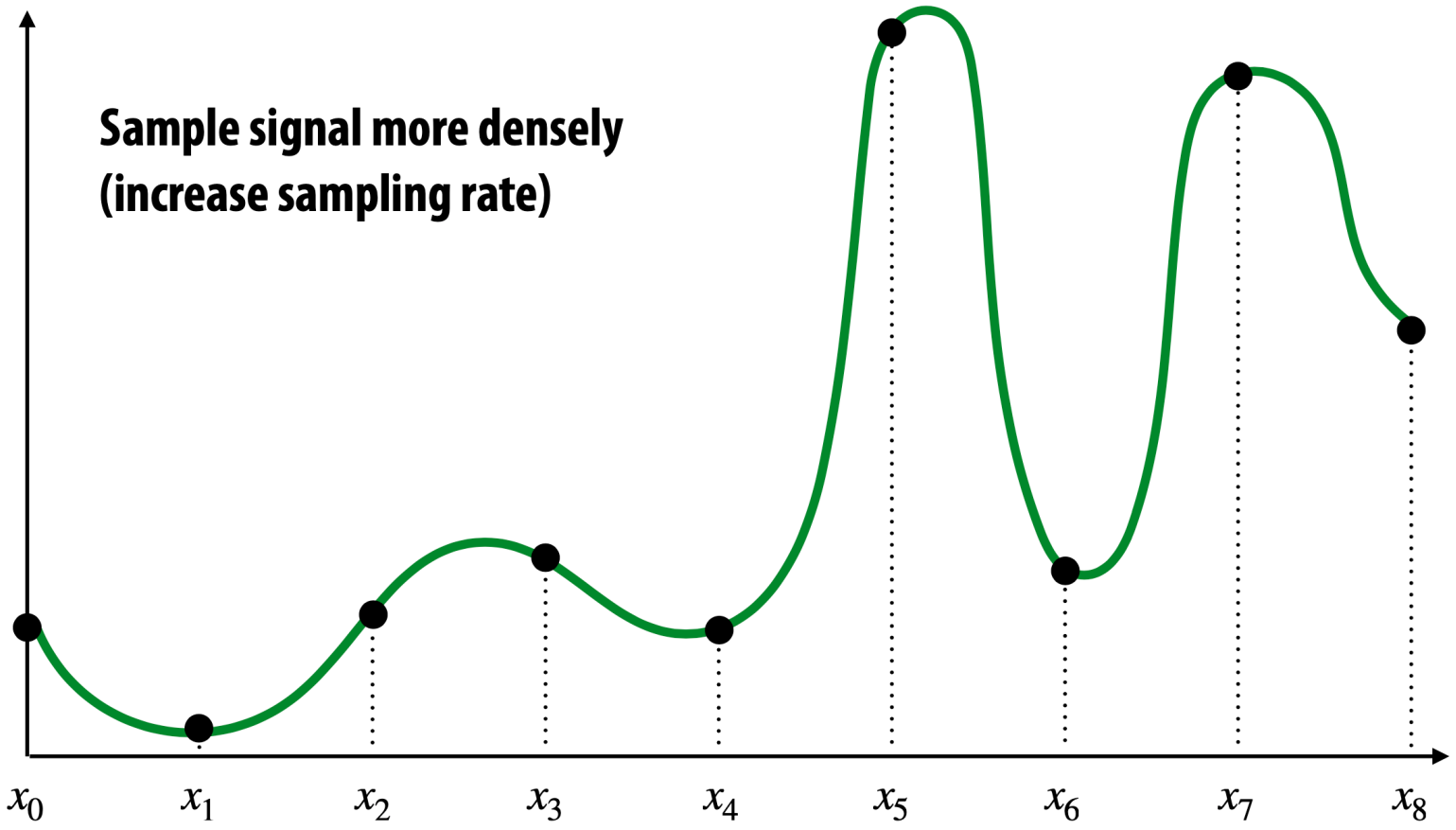
分段线性近似 Piecewise linear approximation

- $\hat{f}(x)$ = 两个连续样本值之间的线性插值



怎样更准确地表示信号?

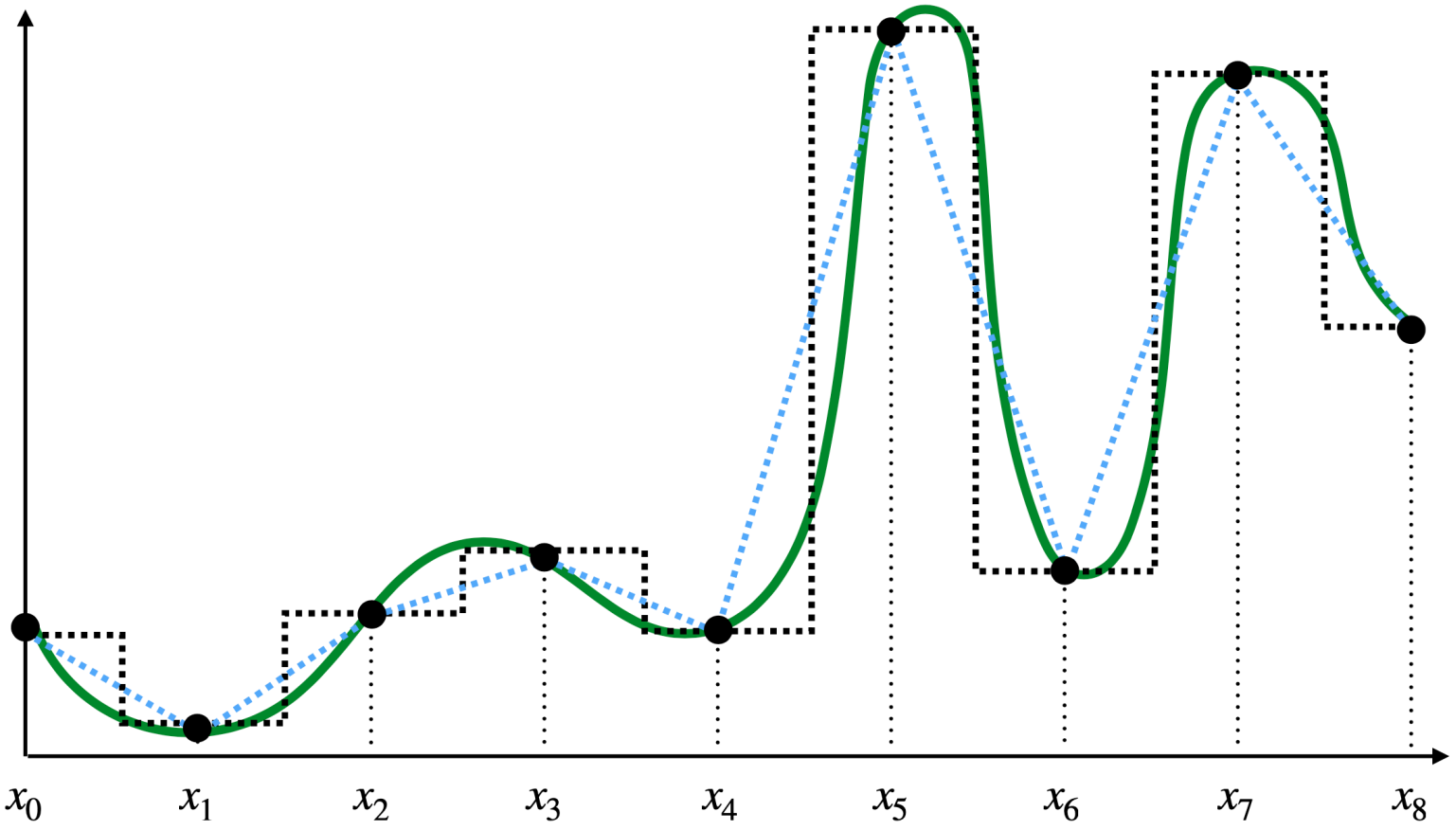
- 采样更多的点



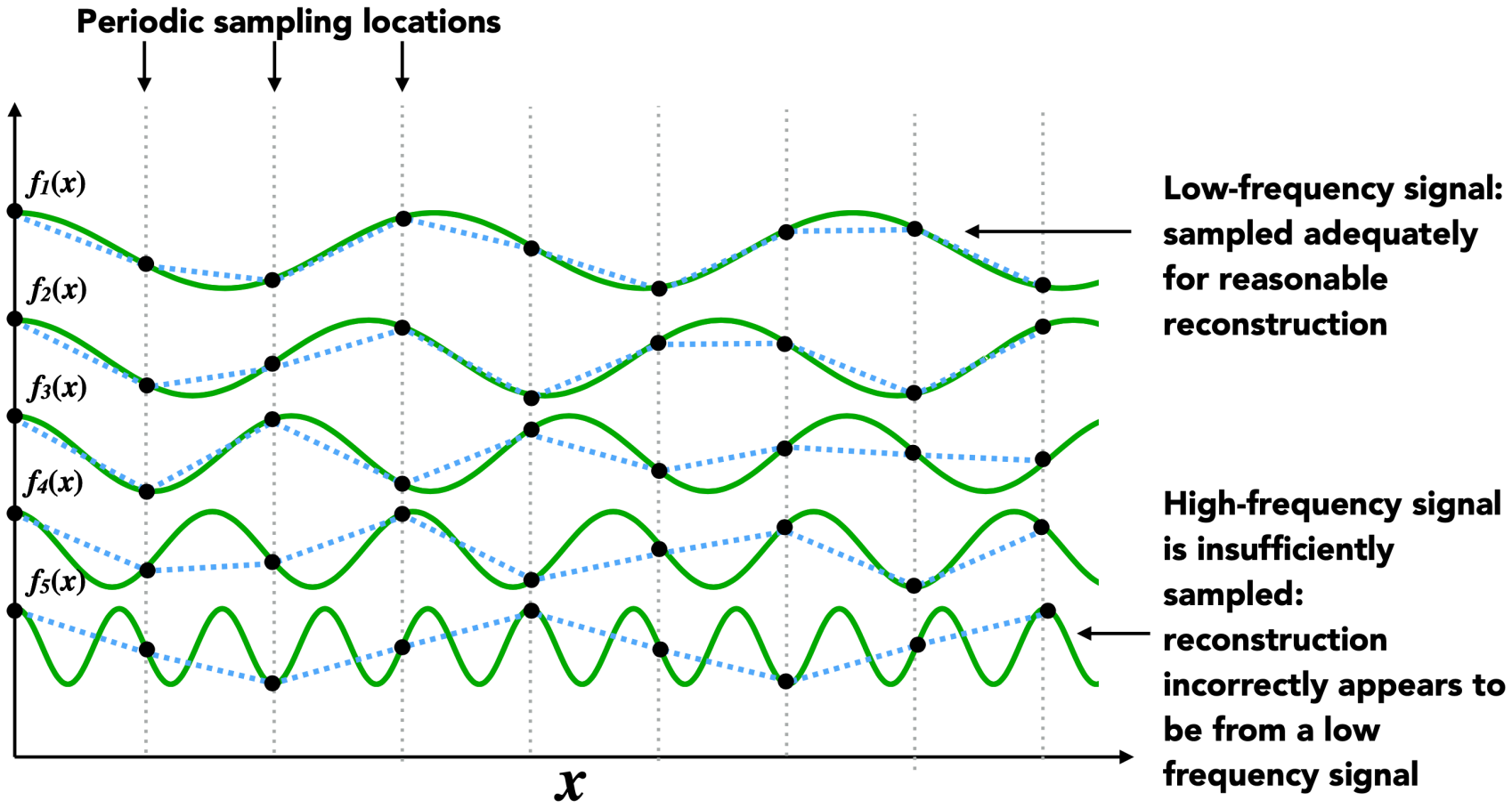
基于更密集采样点的重建

= reconstruction via nearest

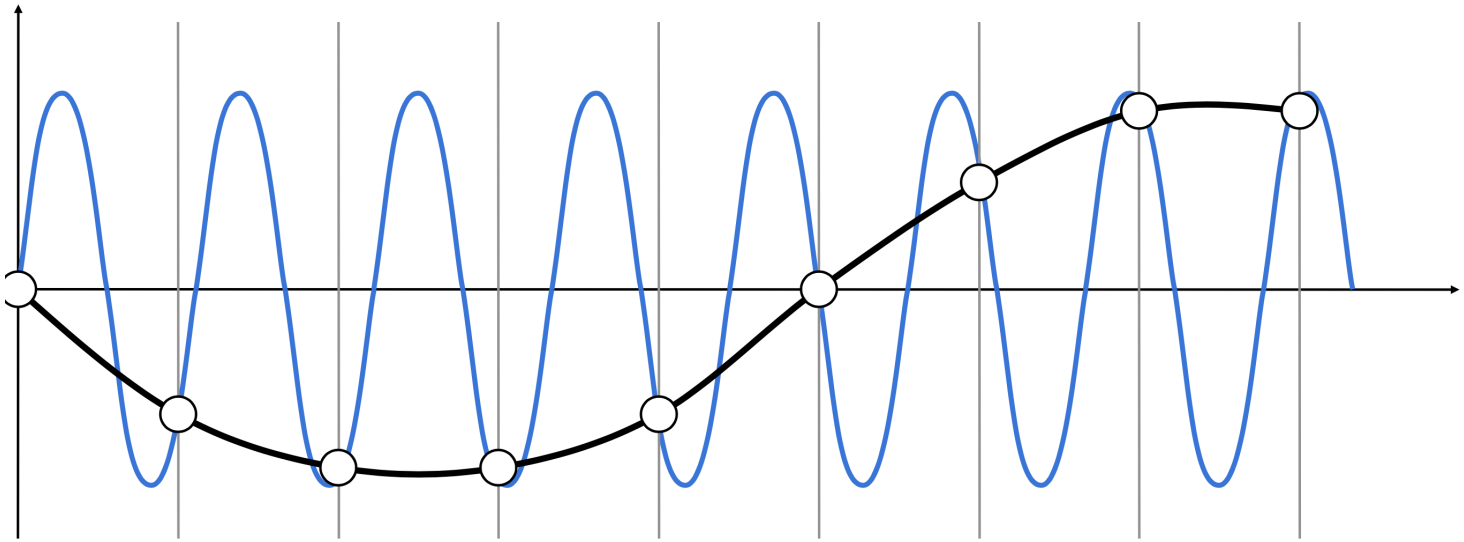
= reconstruction via linear interpolation



高频信号采样不足会导致走样

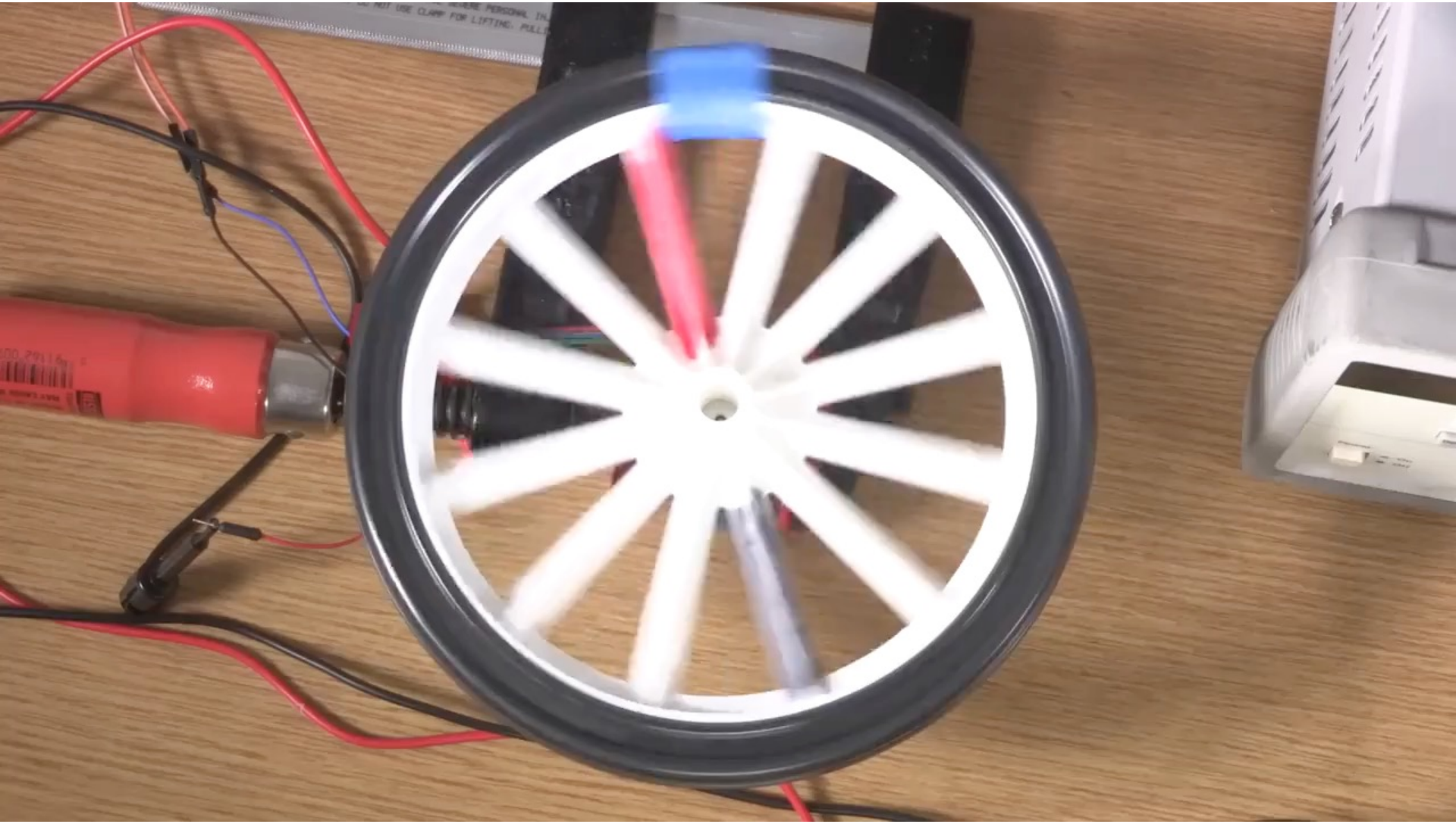


欠采样导致走样



- 高频信号采样不足：样本错误地看起来来自低频信号
- 在给定的采样率下无法区分的两个频率被称为**走样 aliasing**

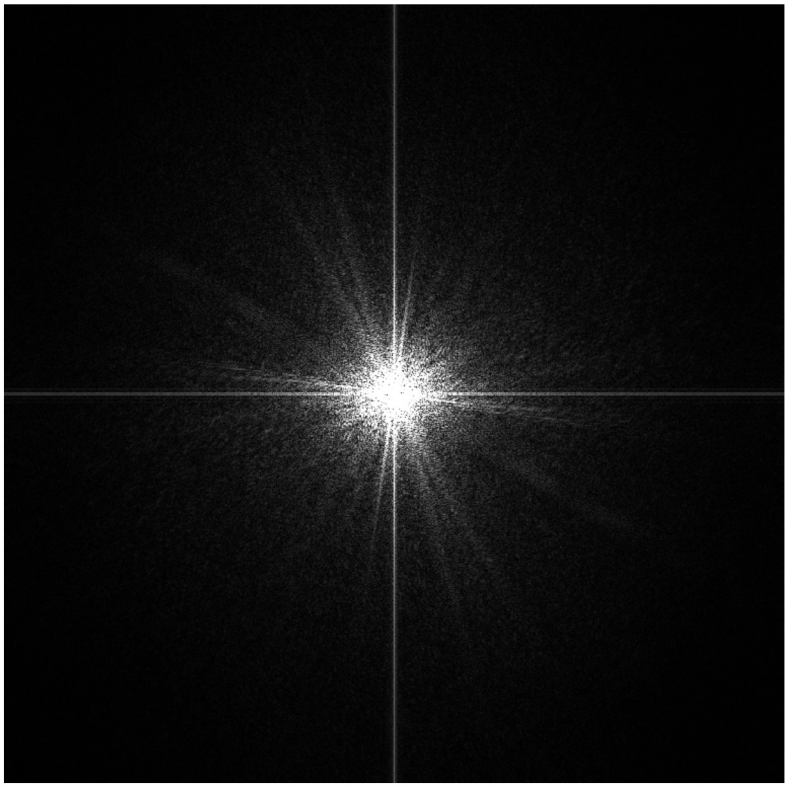
时间走样：车轮效果



对于快速旋转的车轮，相机的帧率（时间采样率）太低

可视化图像频率

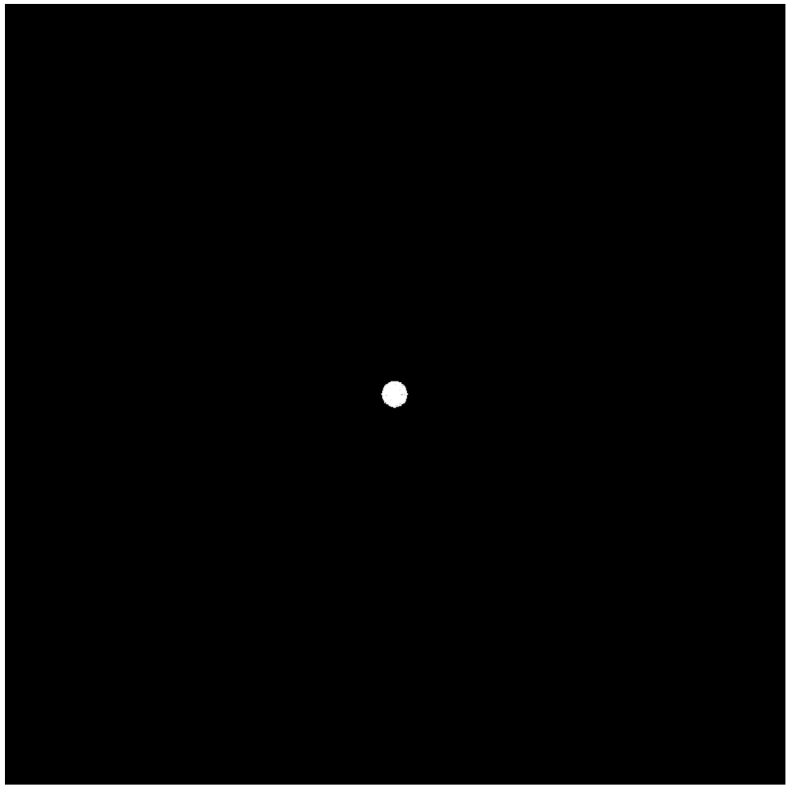
空域结果



频谱

过滤高频

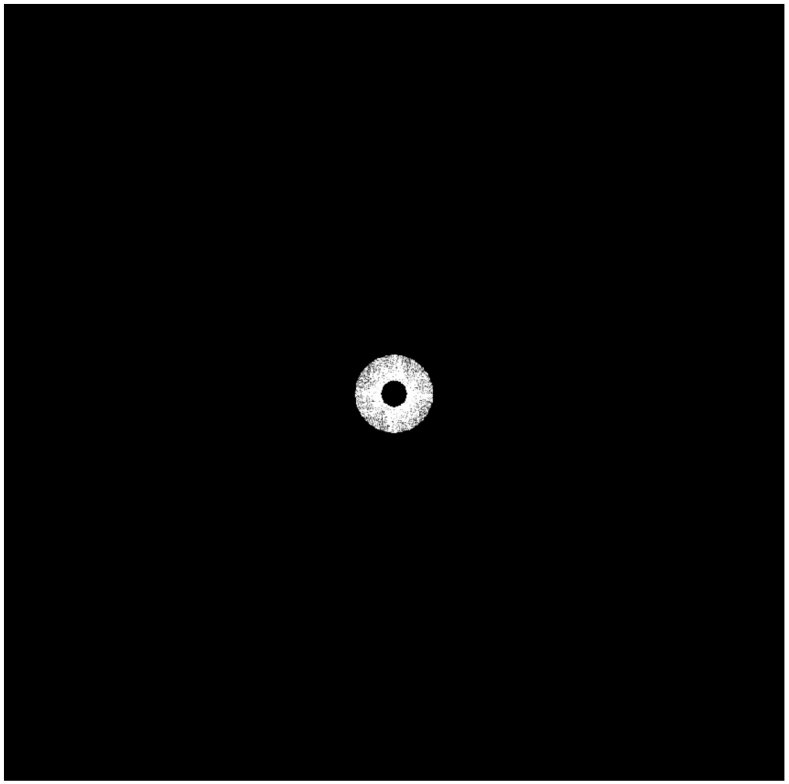
空域结果



频谱（低通滤波后）
高于阈值的频率的幅值为0

过滤低频和高频

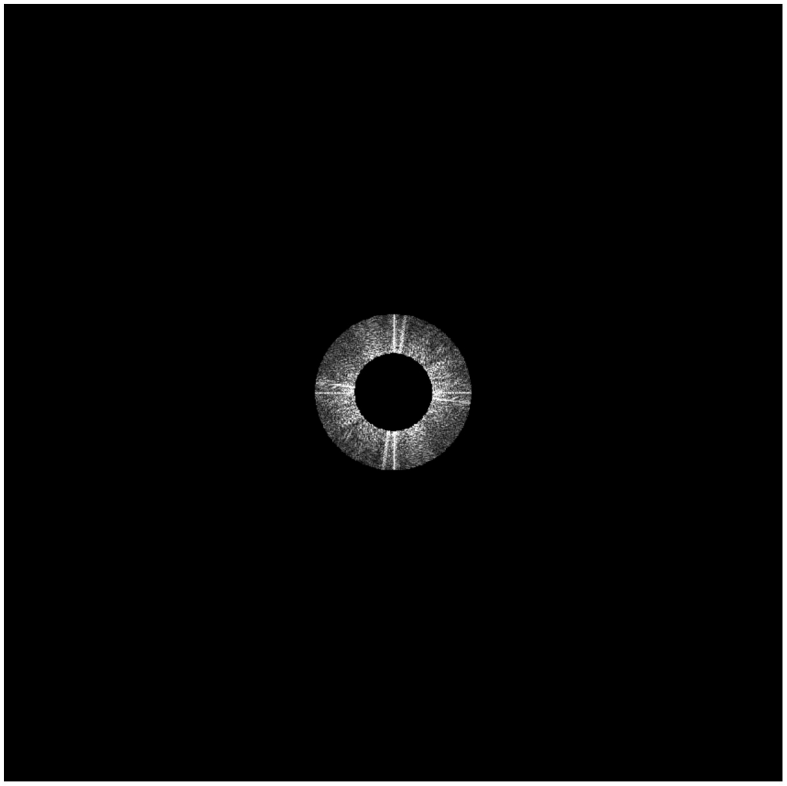
空域结果



频谱（带通滤波后）

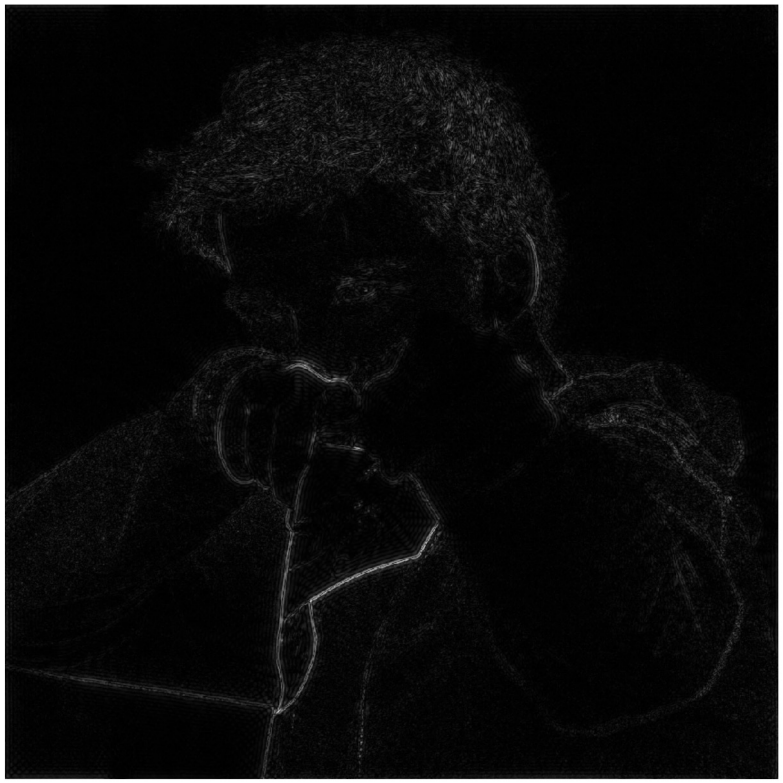
过滤低频和高频

空域结果

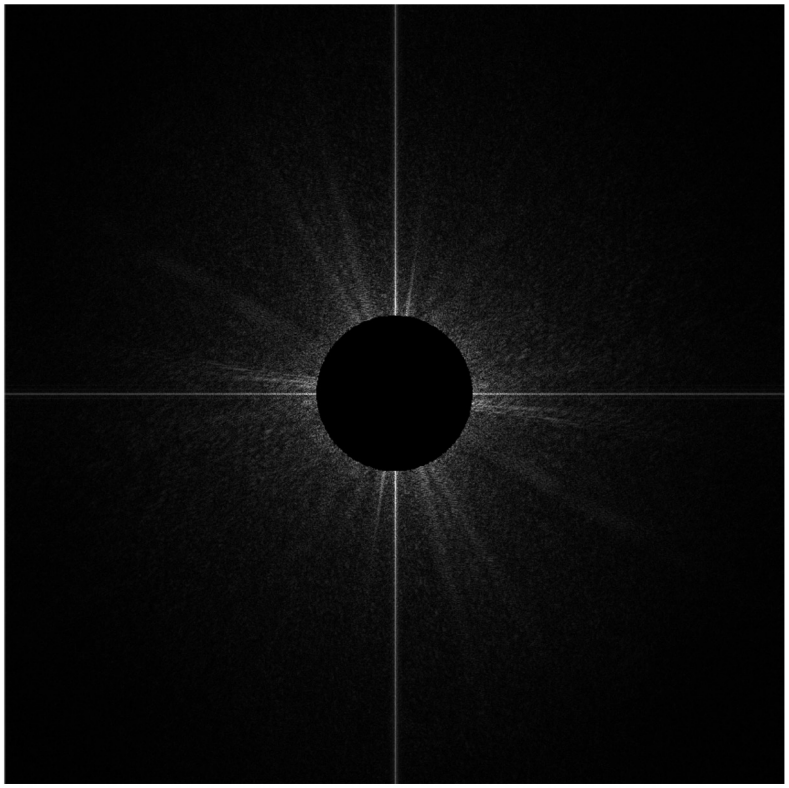


频谱（带通滤波后）

过滤低频

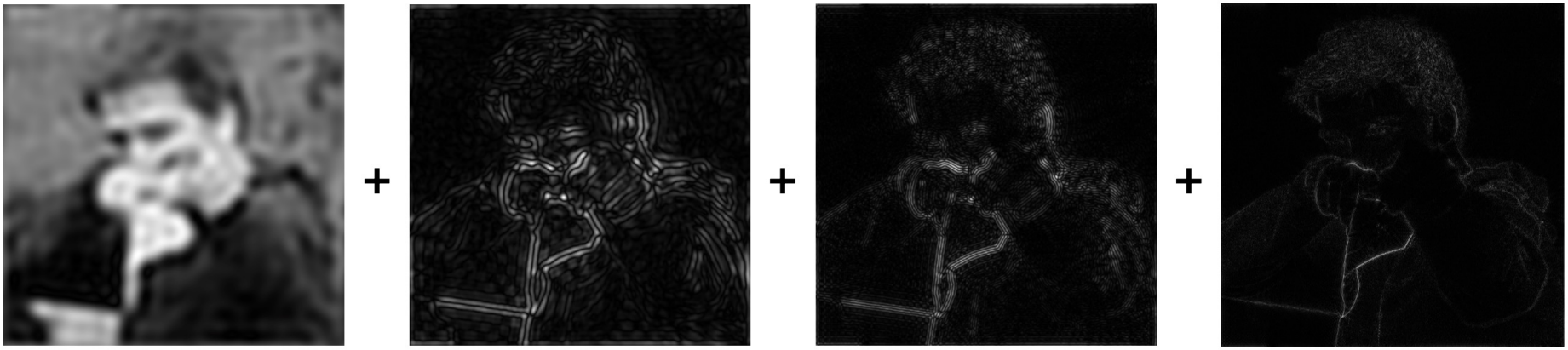


空域结果
(最锐利的边)



频谱（高通滤波后）
低于阈值的频率的幅值为0

图像分解为不同频率的部分



Filtering = Convolution (= Averaging)

尼奎斯特定理 Nyquist Theorem

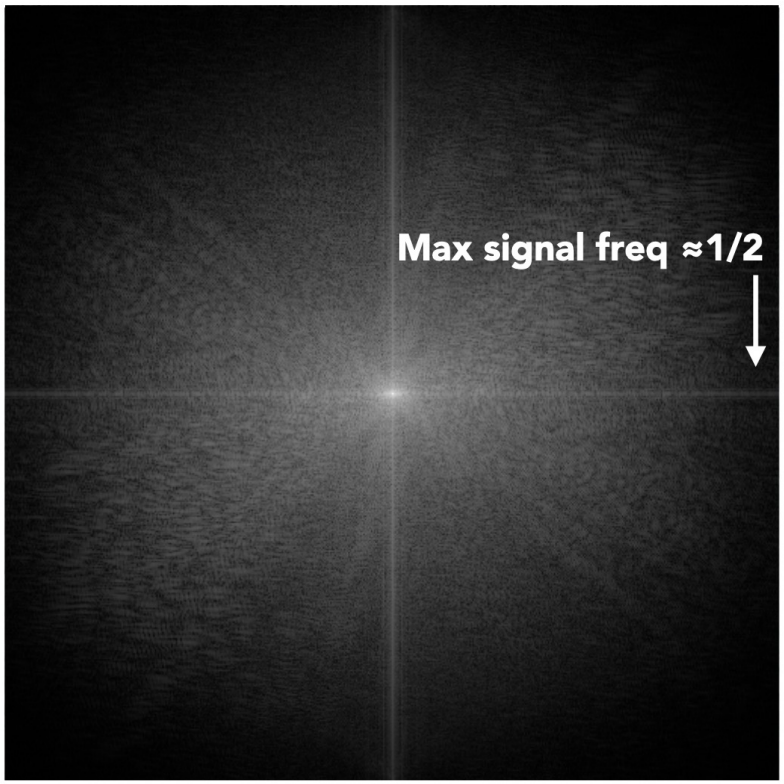
若信号的频率低于尼奎斯特频率（定义为采样频率的一半），则不会出现走样现象。

Theorem: We get no aliasing from frequencies in the signal that are less than the Nyquist frequency (which is defined as half the sampling frequency) *

* See proof: https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem

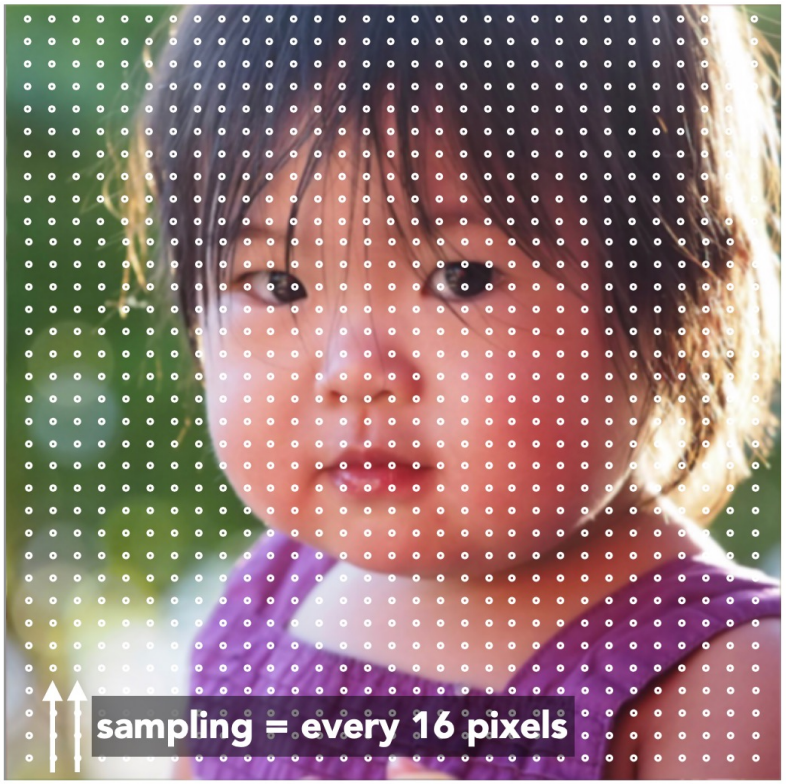
尼奎斯特频率：可视化例子

Spatial Domain

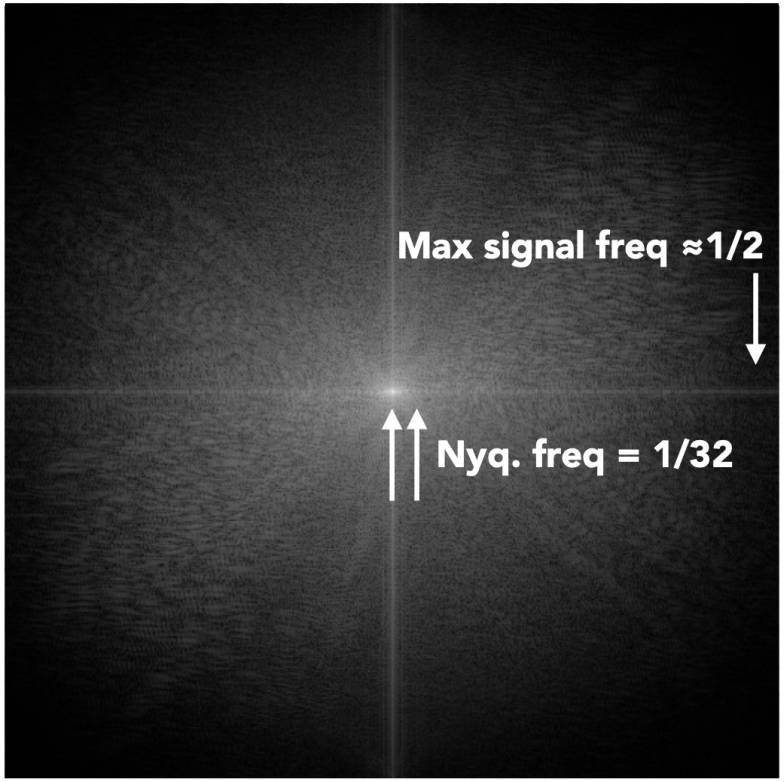


Frequency Domain

尼奎斯特频率：可视化例子

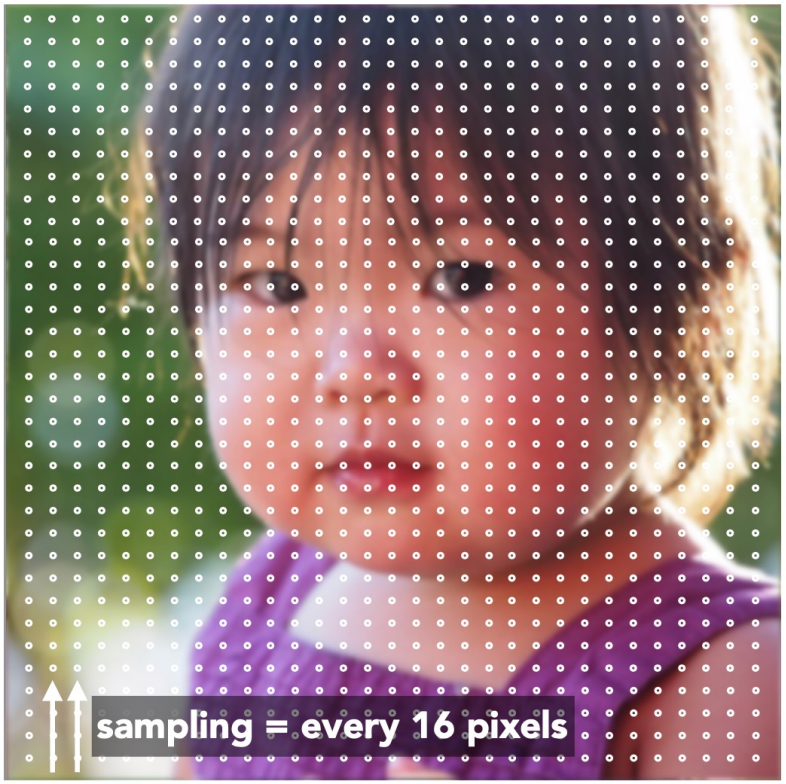


Spatial Domain

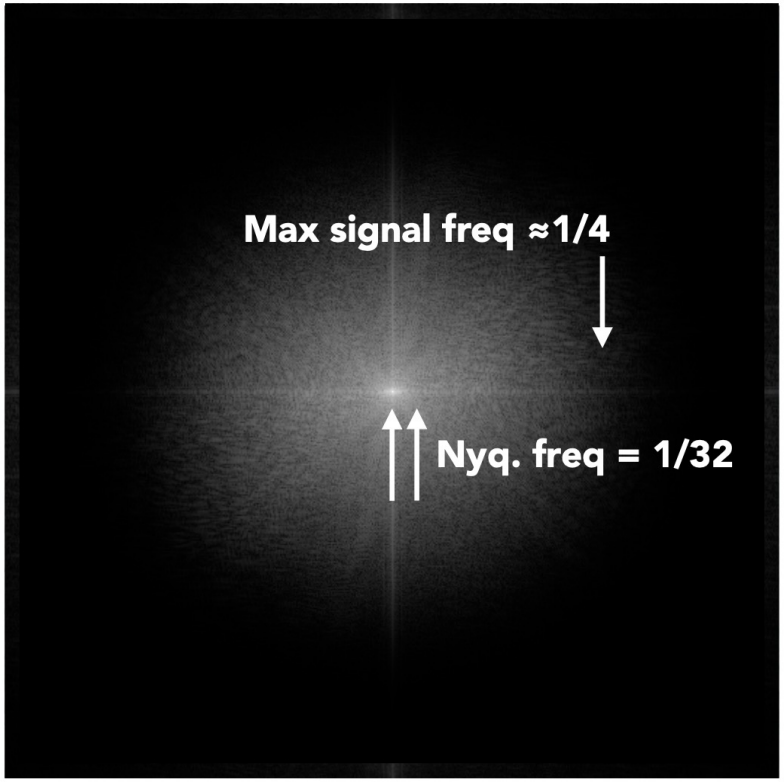


Frequency Domain

尼奎斯特频率：可视化例子

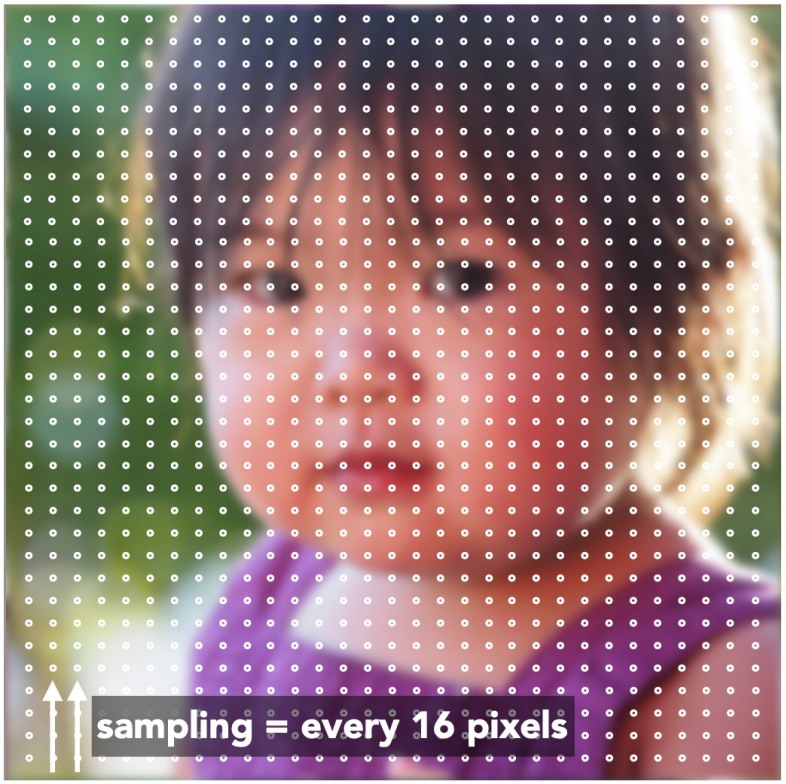


Spatial Domain

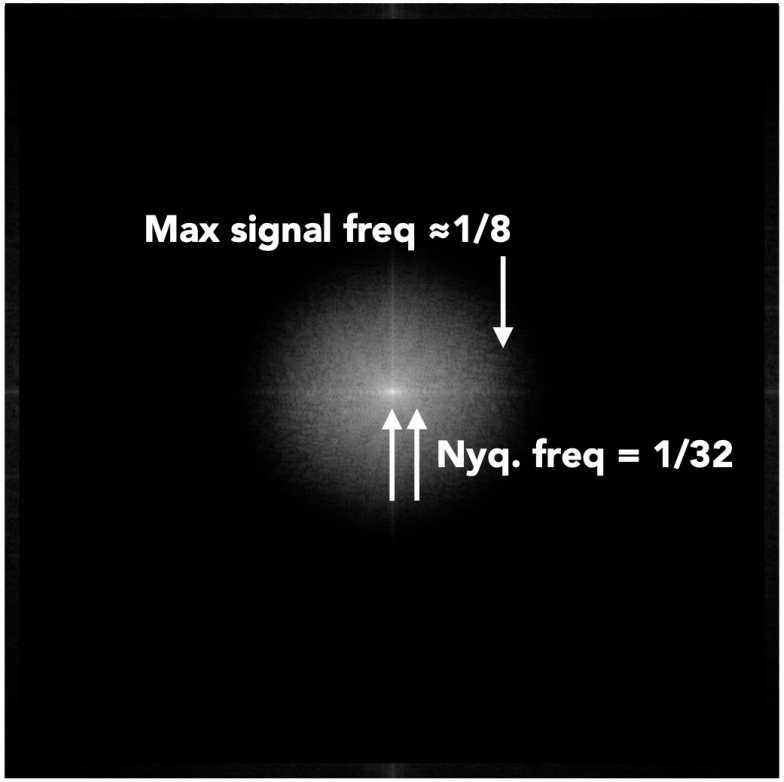


Frequency Domain

尼奎斯特频率：可视化例子

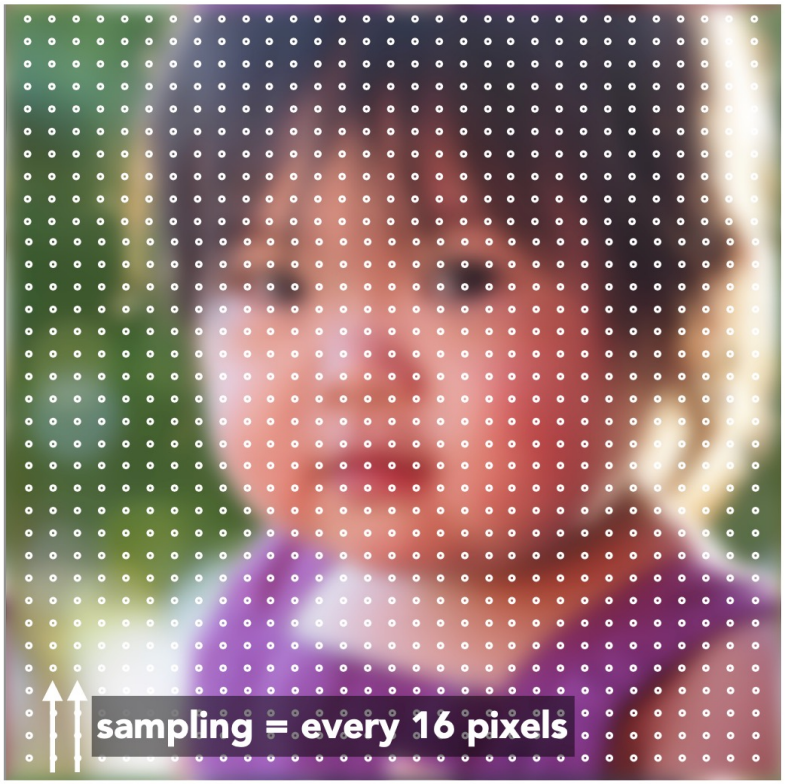


Spatial Domain

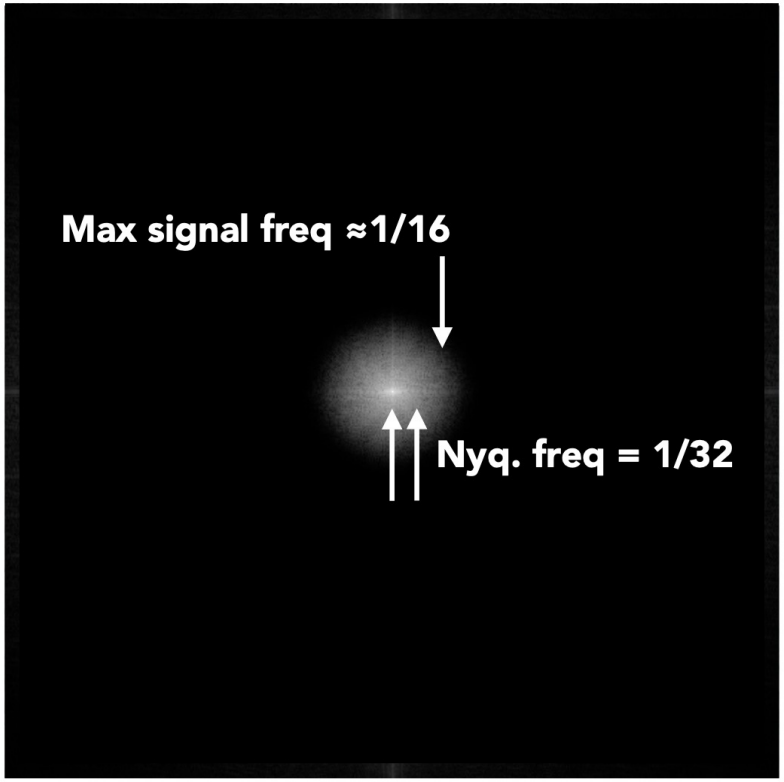


Frequency Domain

尼奎斯特频率：可视化例子

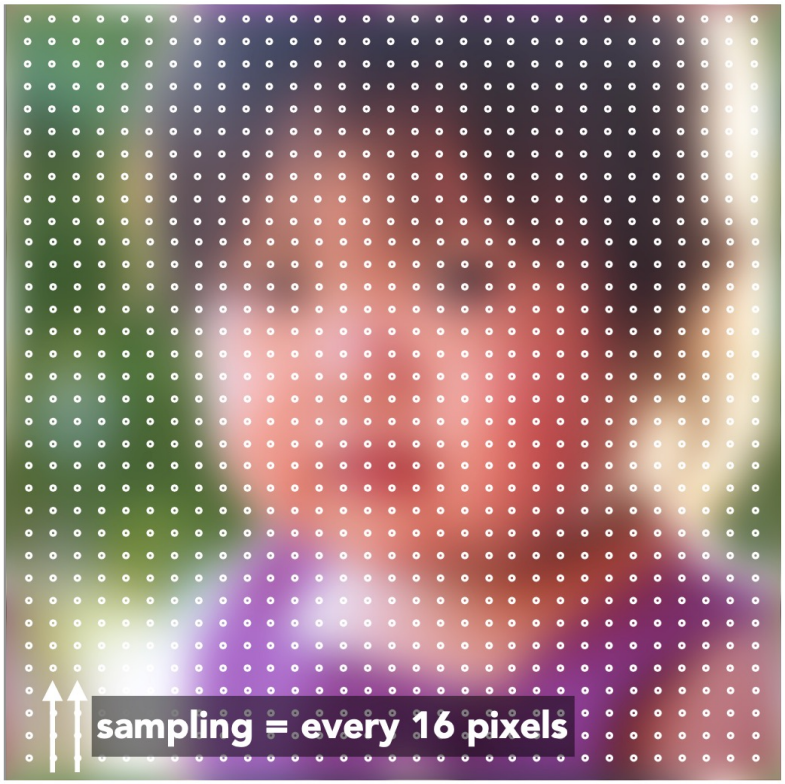


Spatial Domain

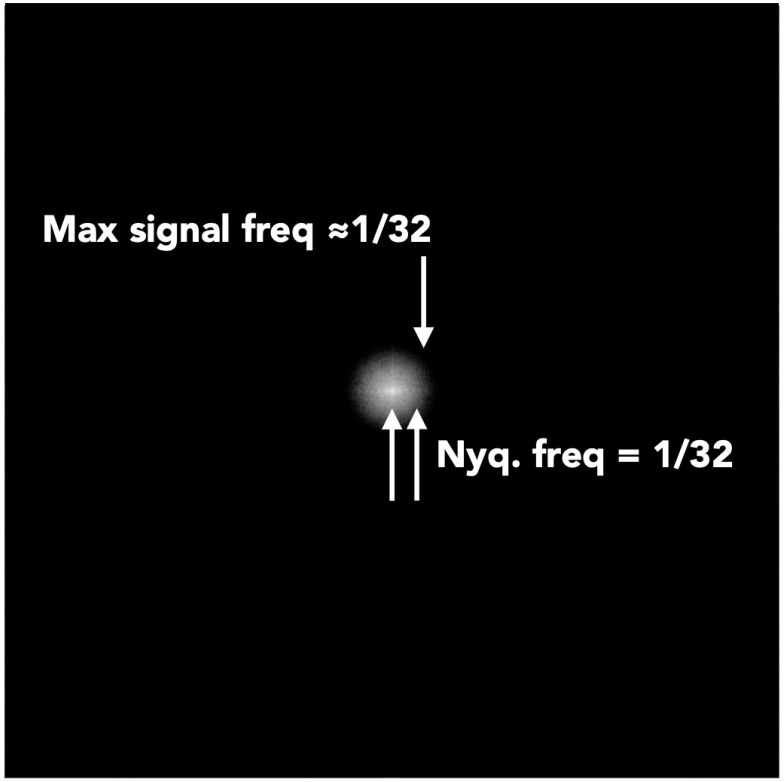


Frequency Domain

尼奎斯特频率：可视化例子

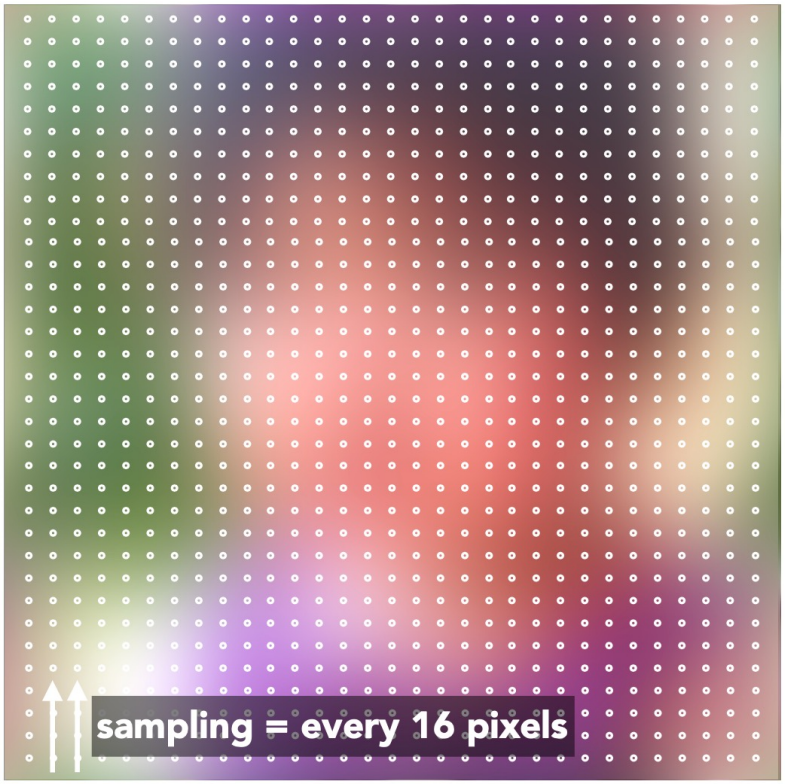


Spatial Domain

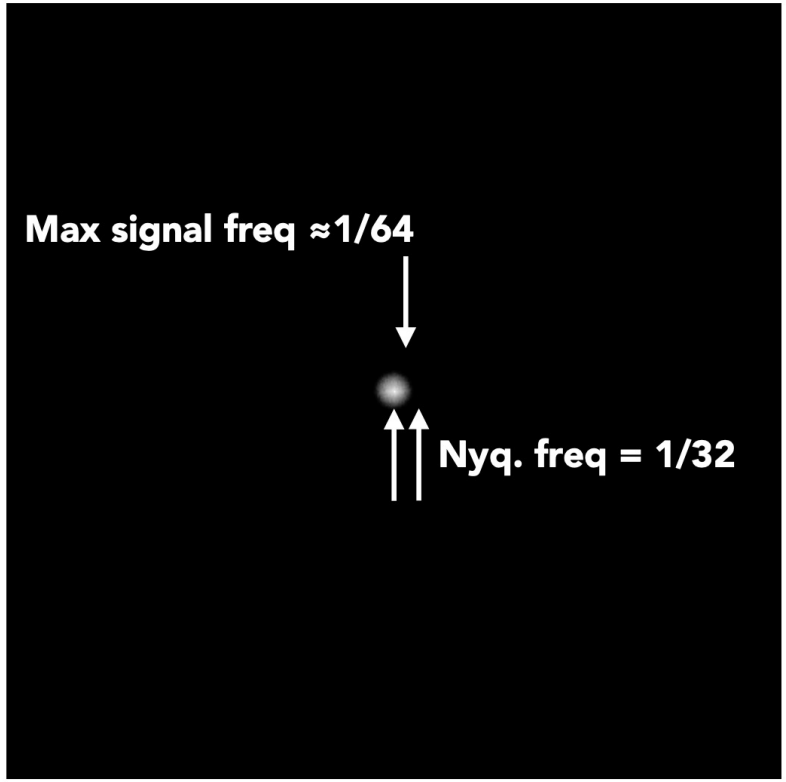


Frequency Domain

尼奎斯特频率：可视化例子



Spatial Domain



Frequency Domain

抗走样
Antialiasing

抗走样

Theorem: We get no aliasing from frequencies in the signal that are less than the Nyquist frequency (which is defined as half the sampling frequency)

因此，以信号中最高频率的两倍
进行采样将消除走样现象

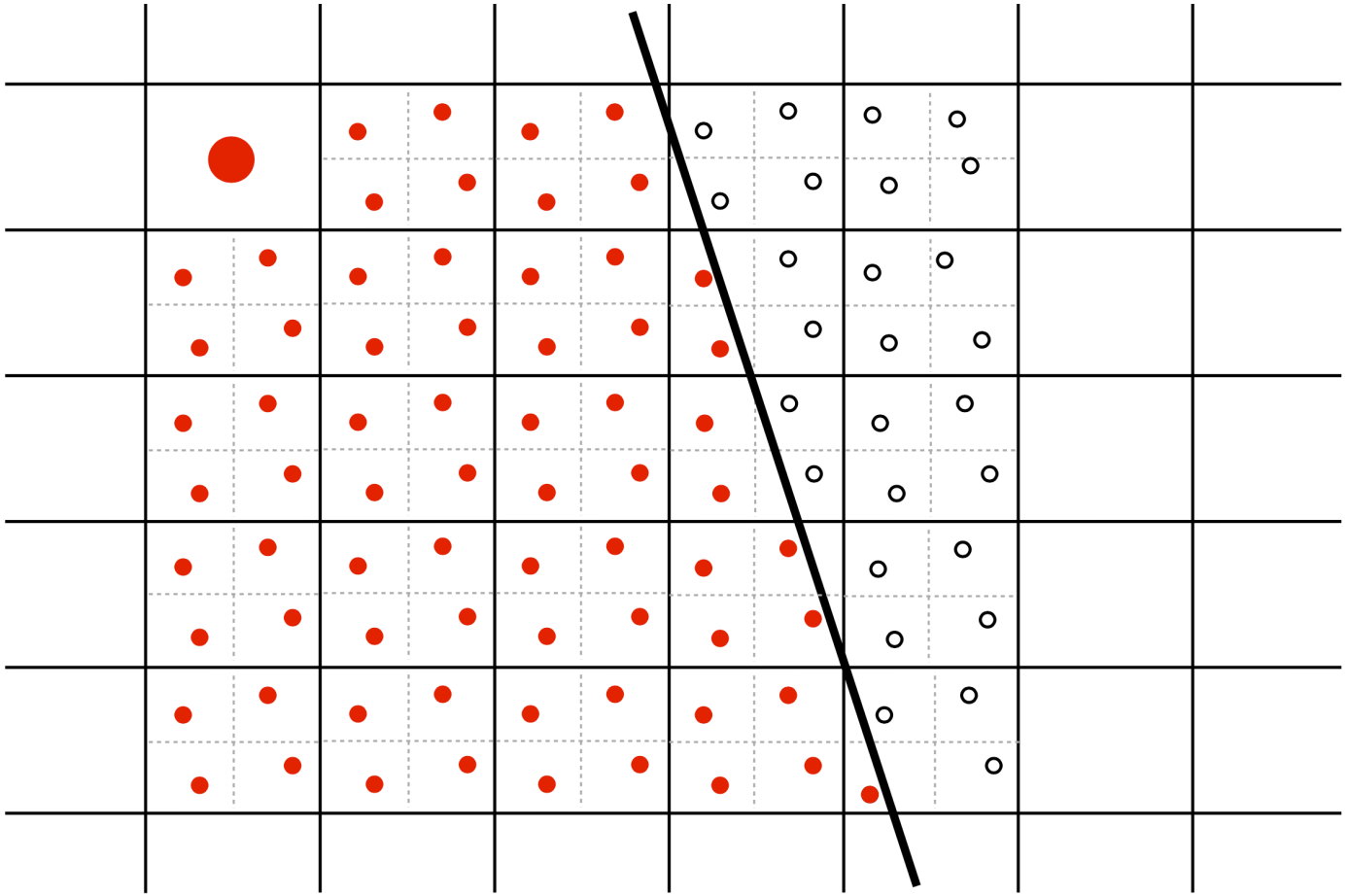
如何减少走样?

- 增加采样率（增加奈奎斯特频率）
  - 更高分辨率的显示器、传感器、帧缓冲区...
  - 但是：成本高昂，可能需要非常高的分辨率

- 抗走样
  - 在采样前去除（或降低）超过尼奎斯特频率的信号
  - 怎样在采样前滤除高频?

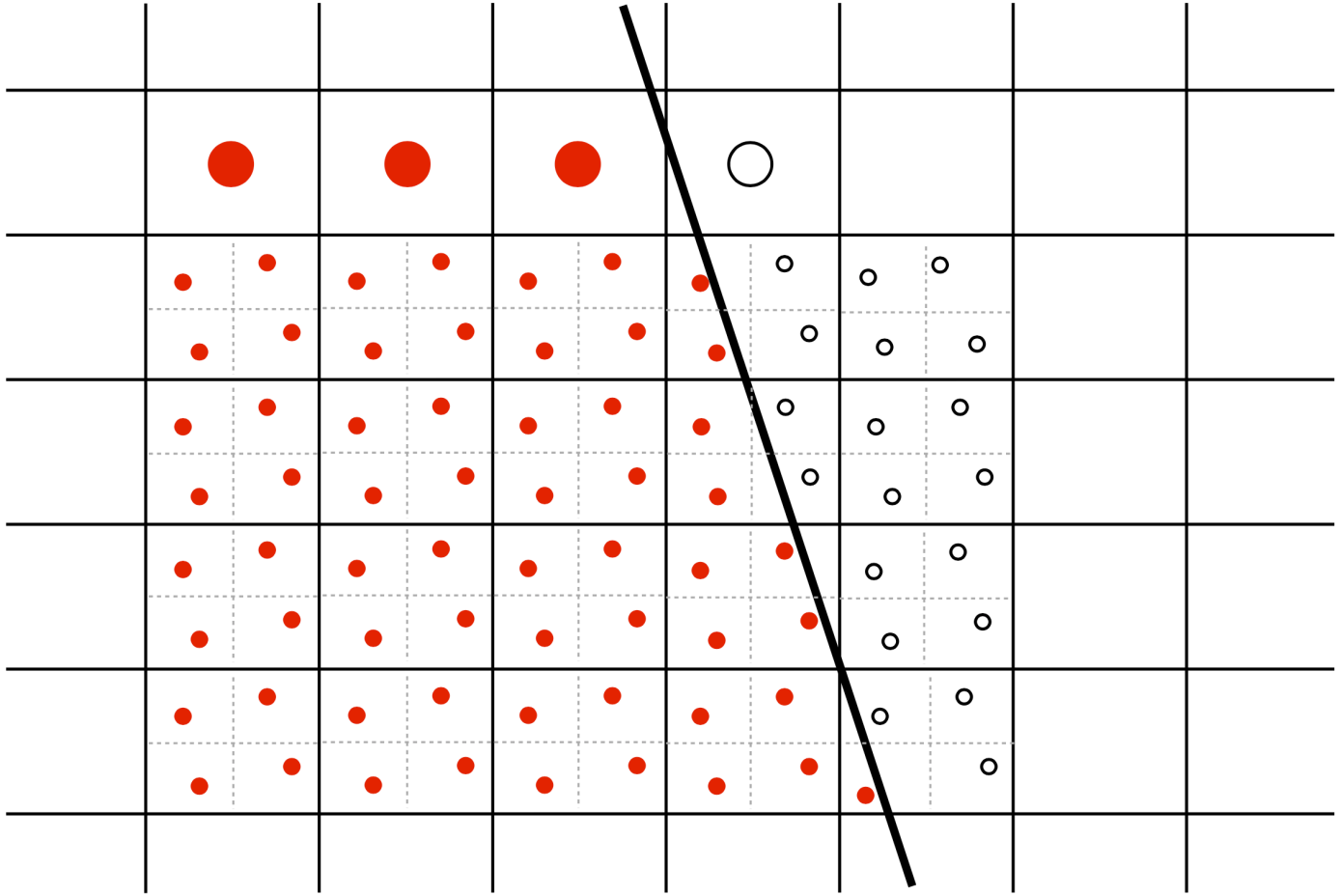
超采样

- 对每个像素里的 $n \times n$ 个样本取平均



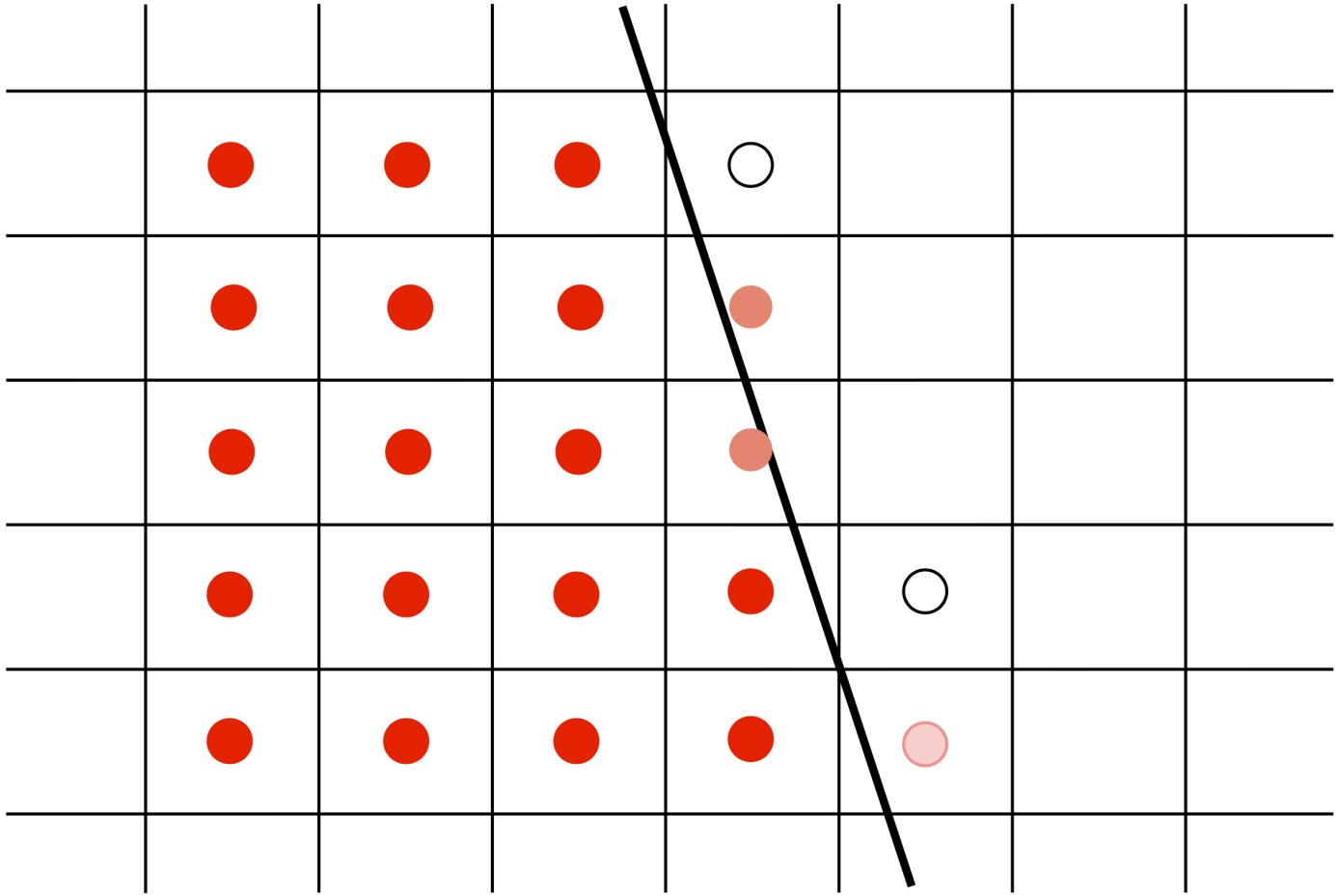
超采样

- 对每个像素里的 $n \times n$ 个样本取平均



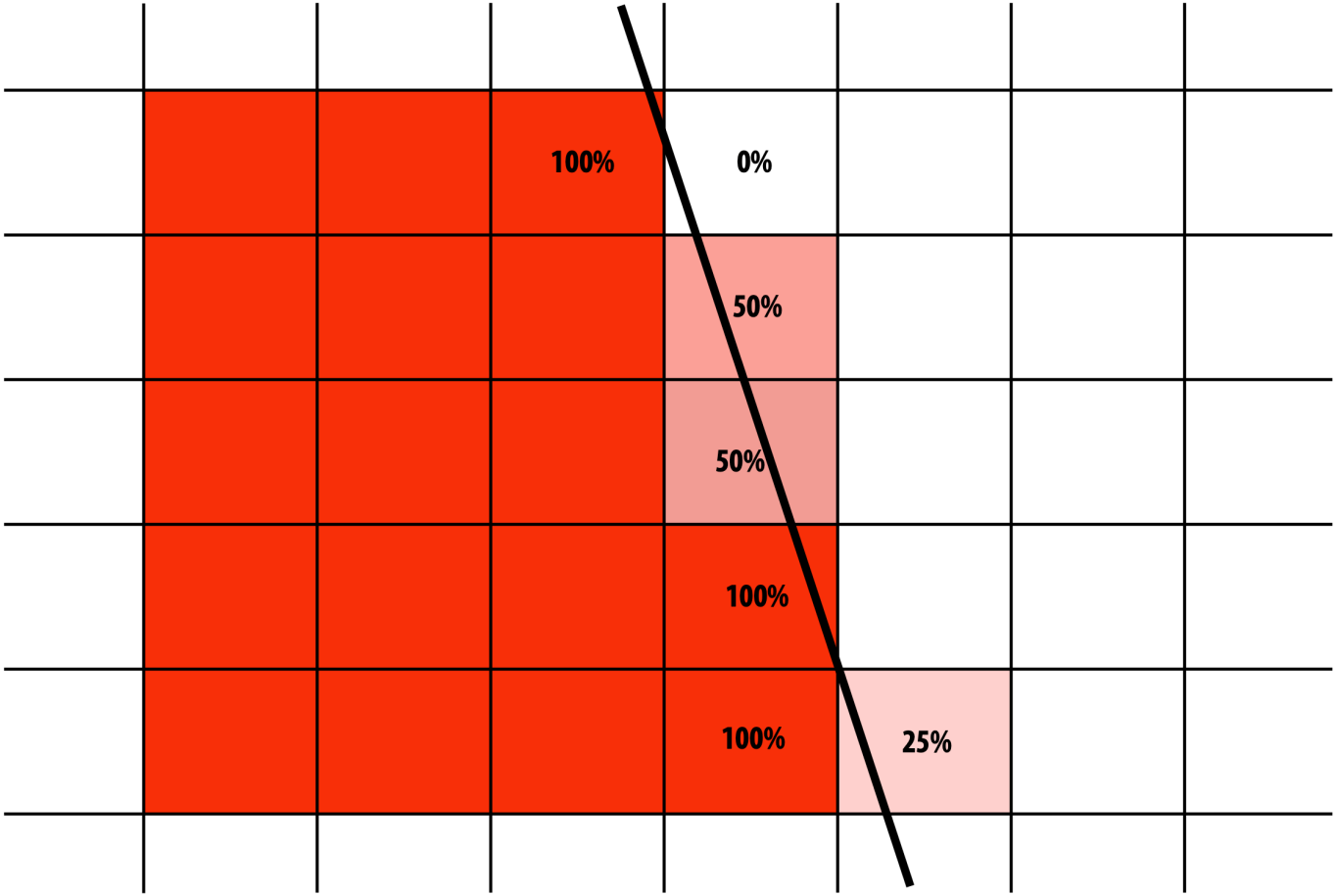
超采样

- 对每个像素里的 $n \times n$ 个样本取平均

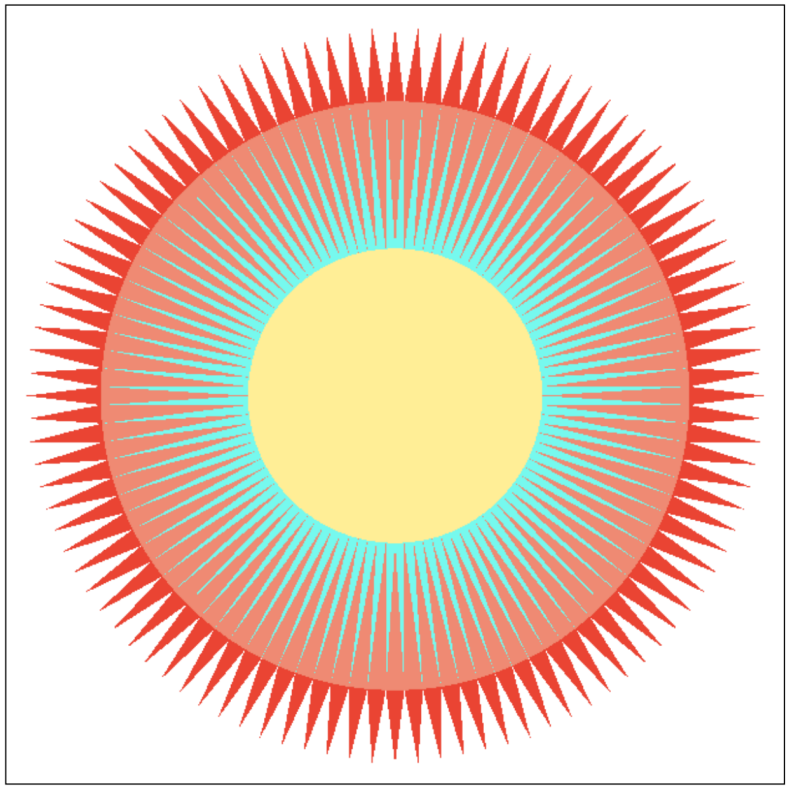


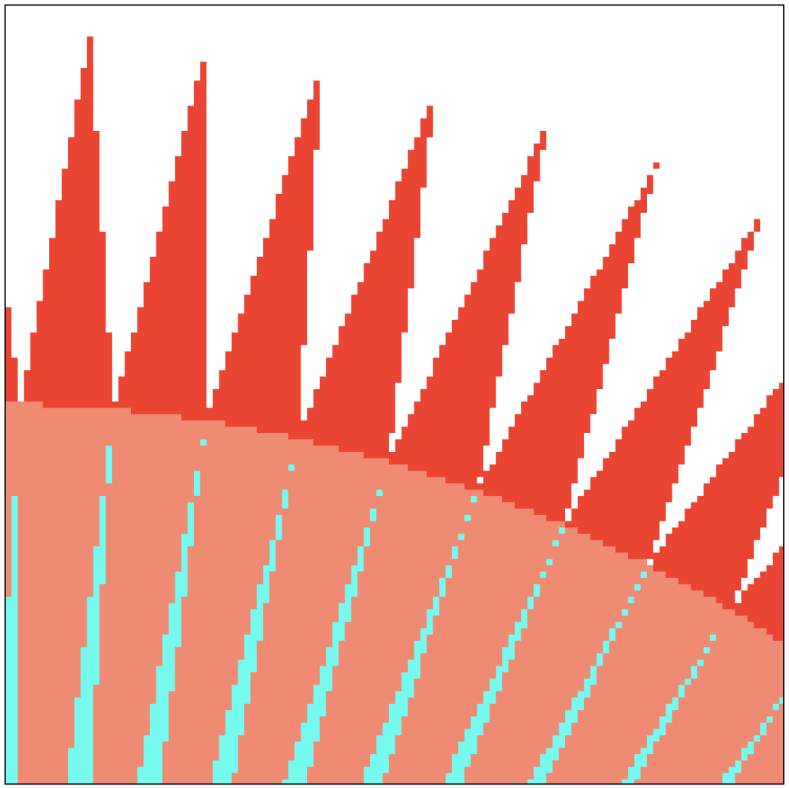
超采样的结果

口边进行了抗采样处理

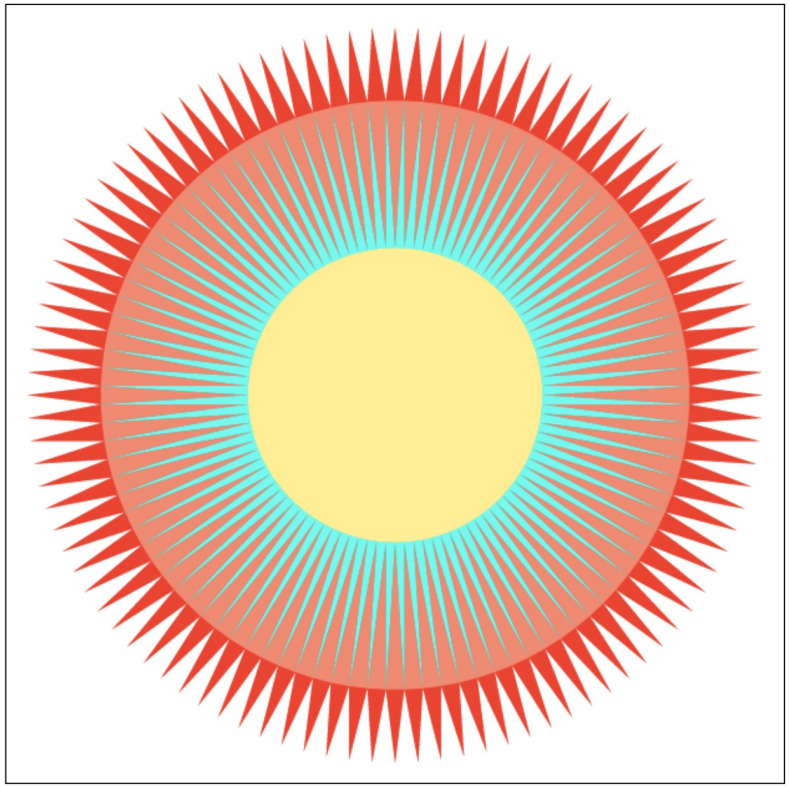


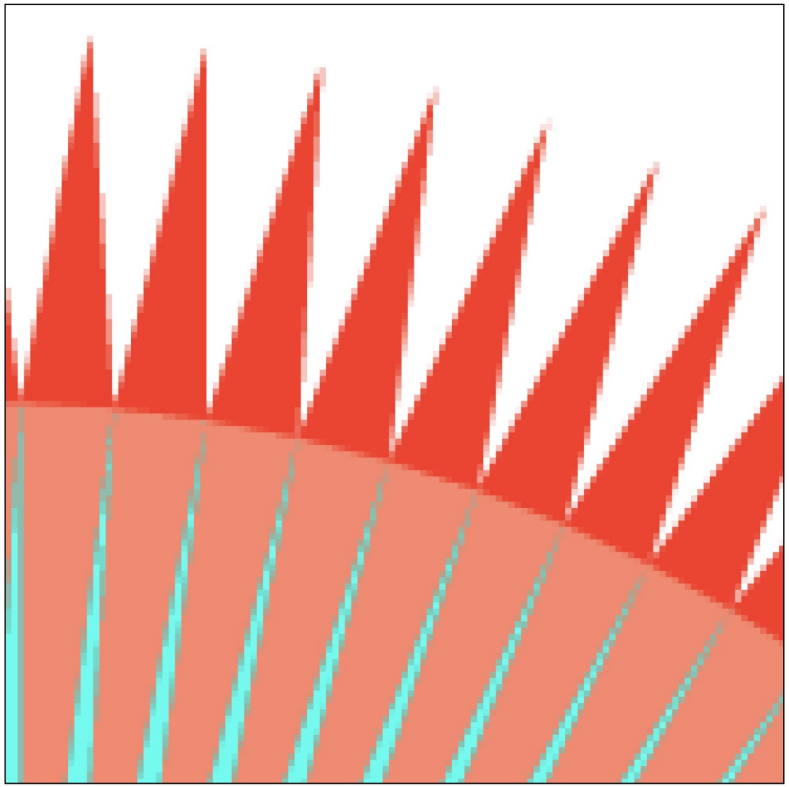
Point sampling



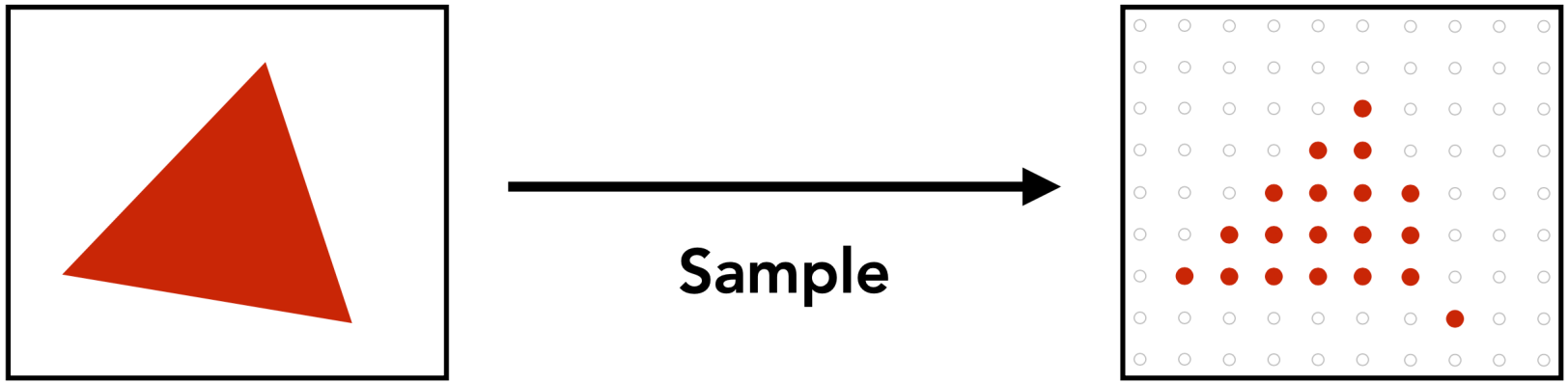


Antialiasing



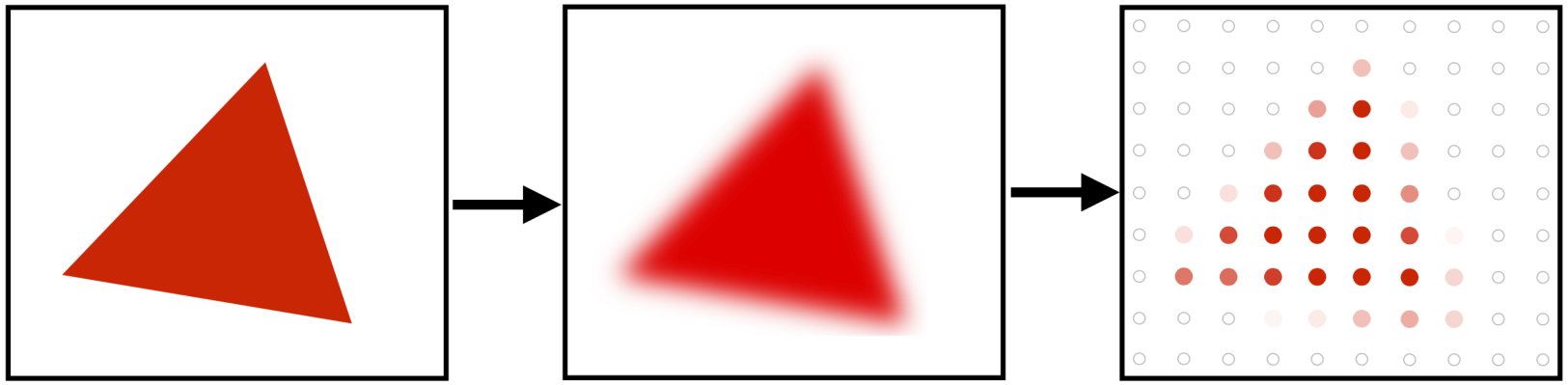


预过滤



注意光栅化三角形中的锯齿为纯红色或白色

预过滤

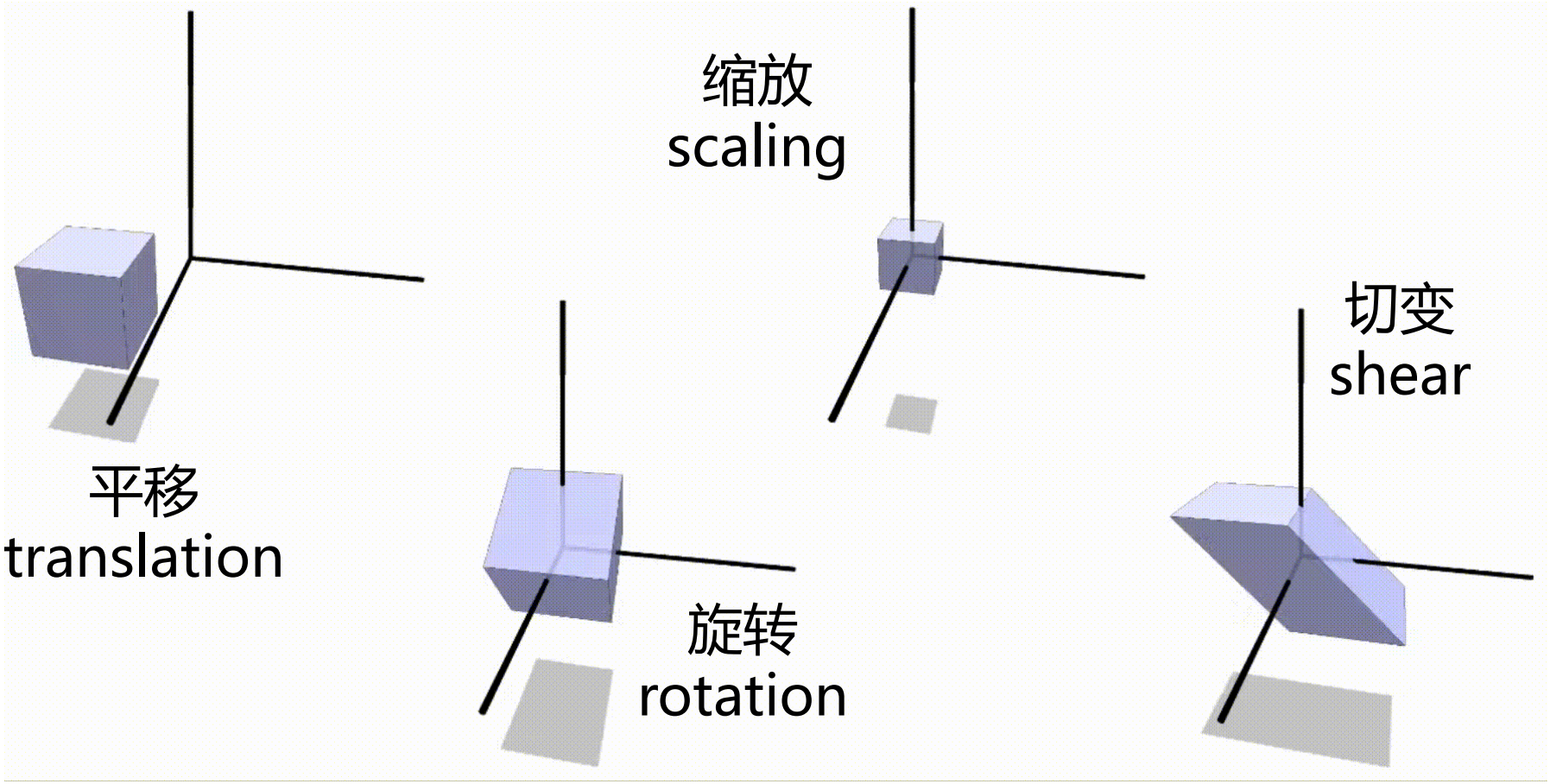


注意光栅化三角形中锯齿的值为红色与白色的中间值

Lecture06：空间变换
Spatial transformation

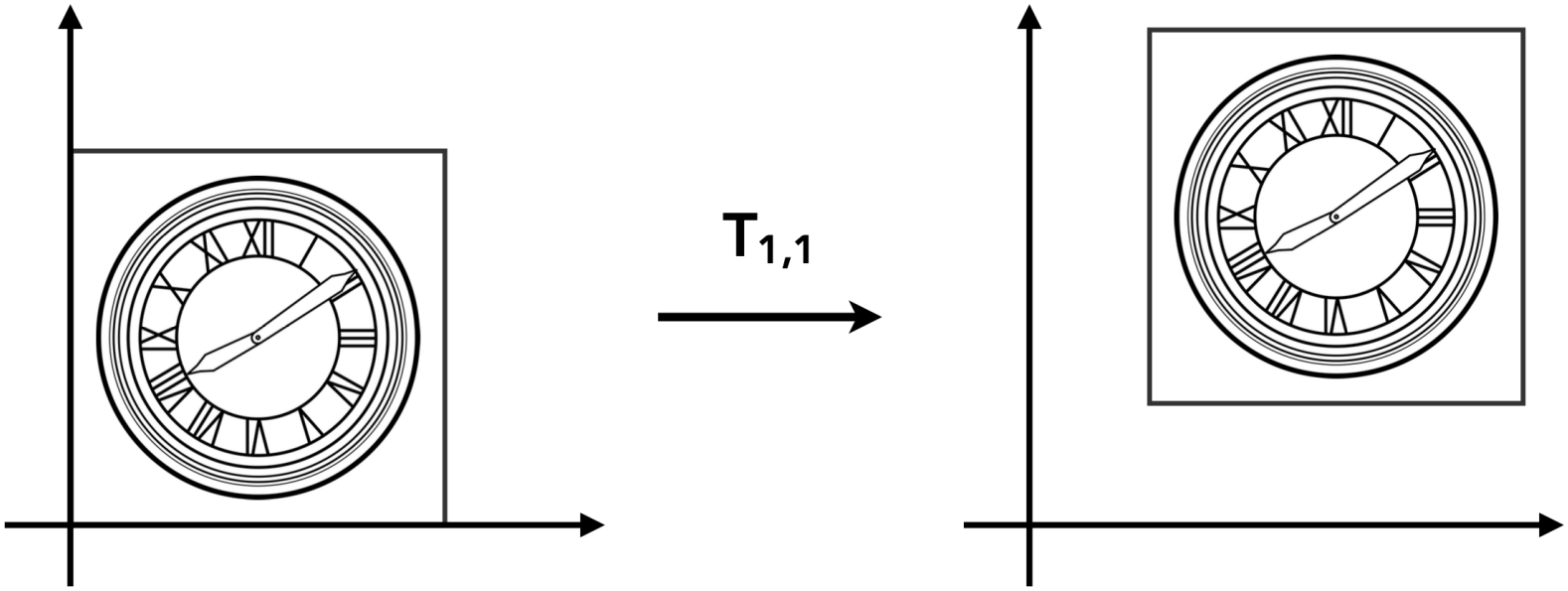
变换的种类

- 你能说出下列每种变换的名字吗?



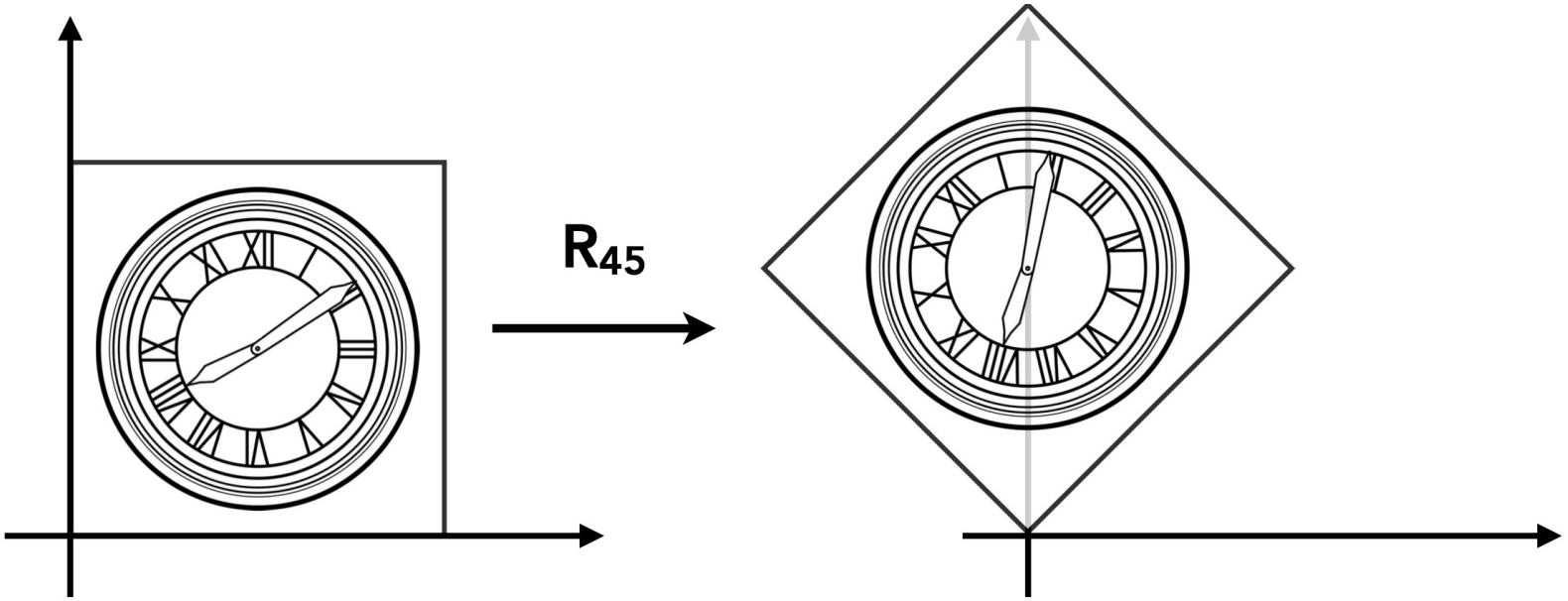
2D 视角的变换

口平移 Translation



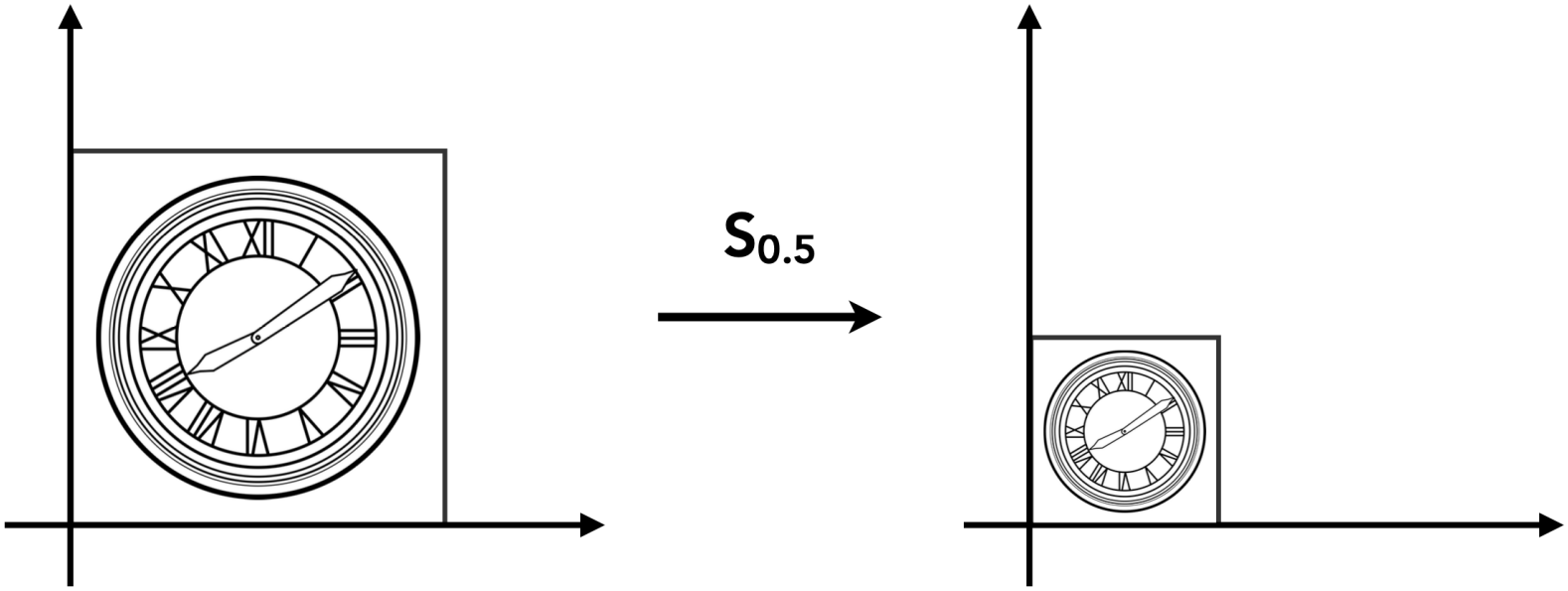
2D 视角的变换

口旋转 Rotation



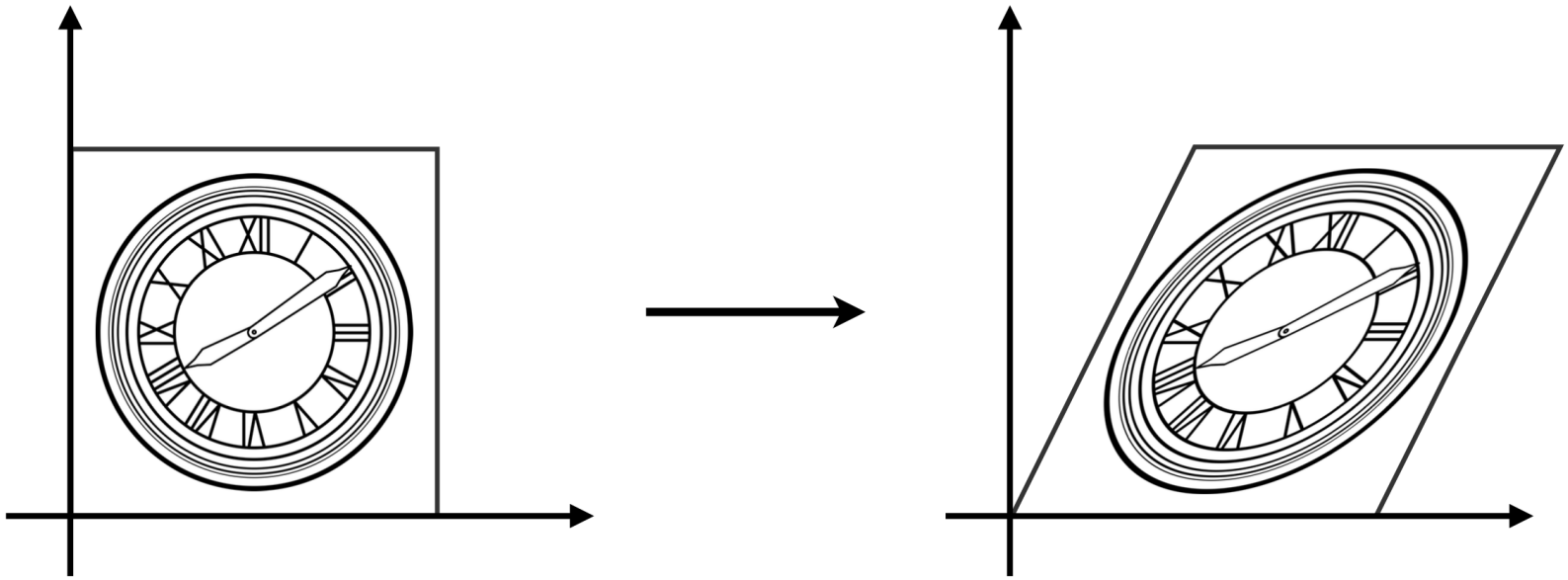
2D 视角的变换

- 缩放 Scaling



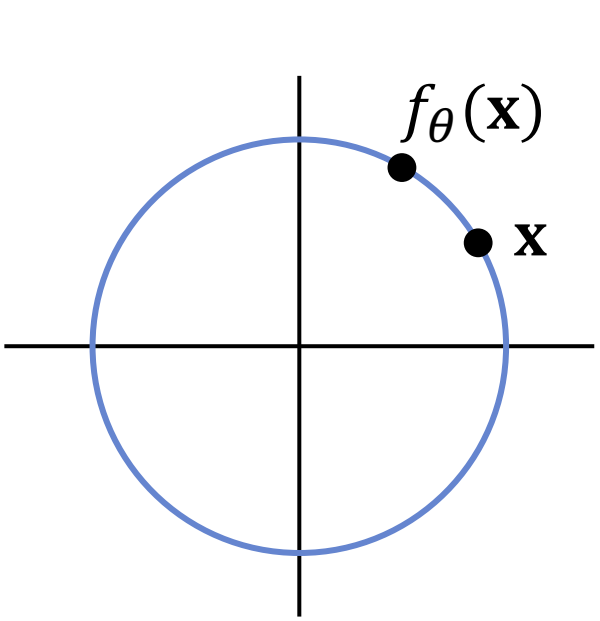
2D 视角的变换

- 切变 Shear

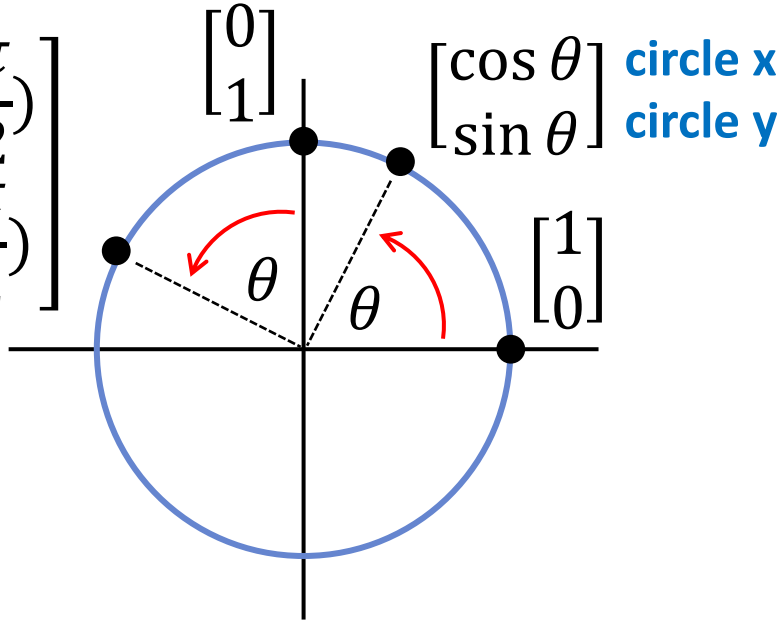


2D 旋转 – 矩阵表示

- 旋转保留了长度和原点，因此，角度为 θ 的 2D 旋转将点 $\mathbf{x}$ 映射到点 $f_\theta(\mathbf{x})$，且 $f_\theta(\mathbf{x})$ 处在半径为 $|\mathbf{x}|$ 的圆上



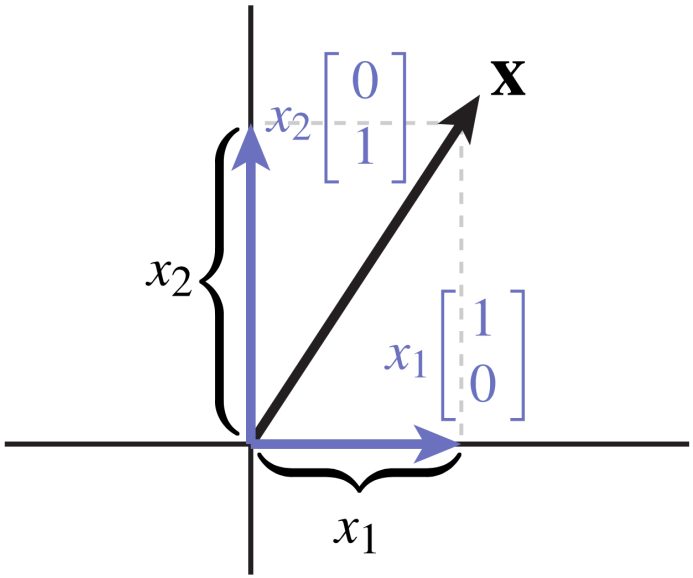
$$\begin{bmatrix} \cos(\theta + \frac{\pi}{2}) \\ \sin(\theta + \frac{\pi}{2}) \end{bmatrix} = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

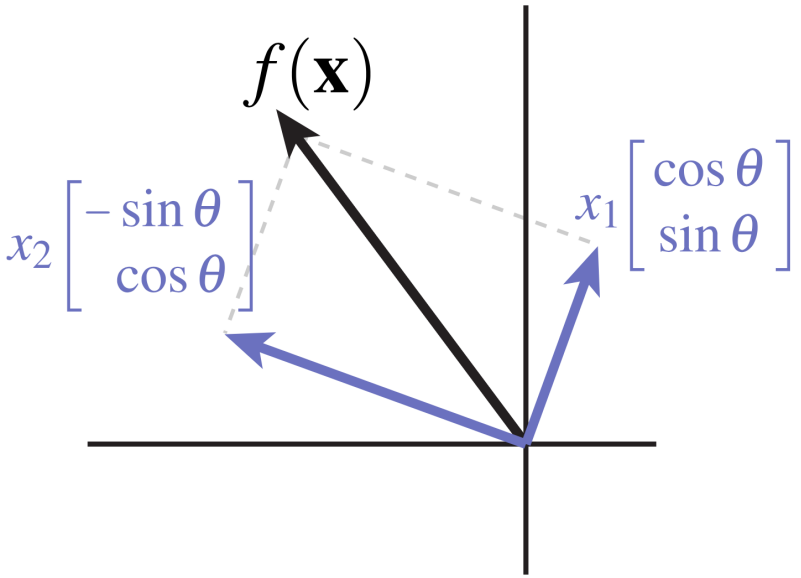


- 如果我们逆时针旋转 θ，$\mathbf{x} = (1, 0)$ 将映射到哪个点?
- $x = (0, 1)$ 呢?

更一般的向量 $x = (x_1, x_2)$ 呢?

2D 旋转 – 矩阵表示





$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$f(\mathbf{x}) = x_1 \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} + x_2 \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

口我们要如何用矩阵表示一个 2D 旋转函数 $f_\theta(x)$?

即用矩阵表示
一个线性映射

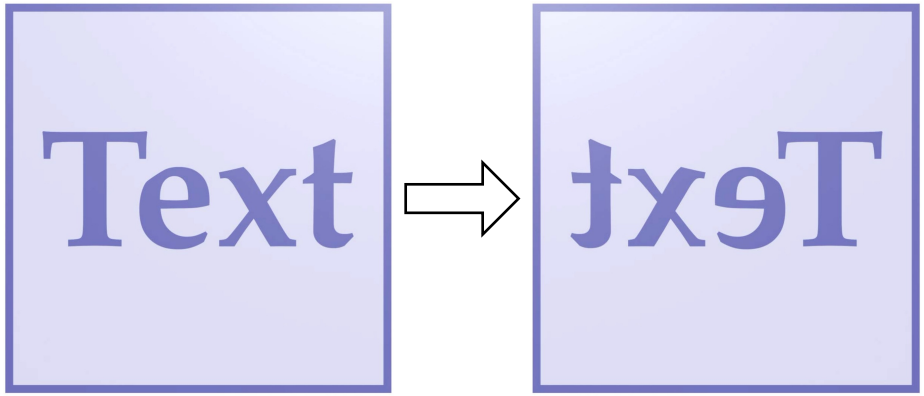
$$f_\theta(\mathbf{x}) = \begin{bmatrix} \cos\theta & -\sin(\theta) \\ \sin\theta & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

反射 Reflection

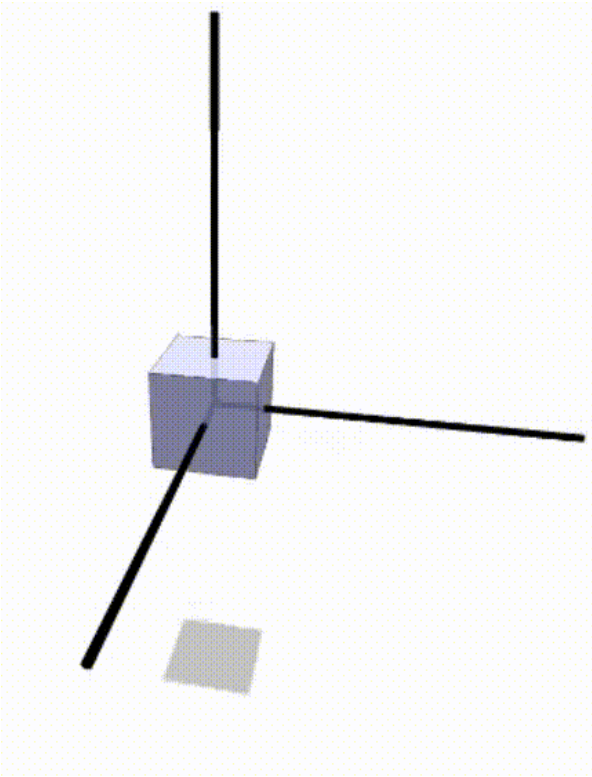
- Q：是否所有满足 $Q^T Q = I$ 的正交矩阵均描述了一个旋转?
- 旋转必须保持**原点 origin**、**距离 distance** 和**方位 orientation**
- 考虑如下的矩阵

$$Q = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Q^{\mathsf{T}} Q = \begin{bmatrix} (-1)^2 & 0 \\ 0 & 1 \end{bmatrix} = I$$

- Q：这个矩阵是否描述了一个旋转？如果不是，哪个量未能保持?
- A：No！它代表了一个沿着 y 轴的反射，因此未能保持方位



缩放 Scaling



- 每个向量 $\mathbf{u}$ 映射到标量倍数

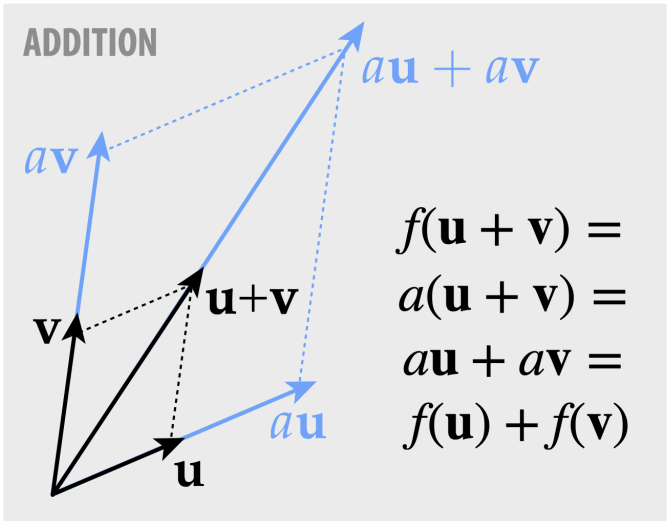
$$f(\mathbf{u}) = a\mathbf{u}, \quad a \in \mathbb{R}$$

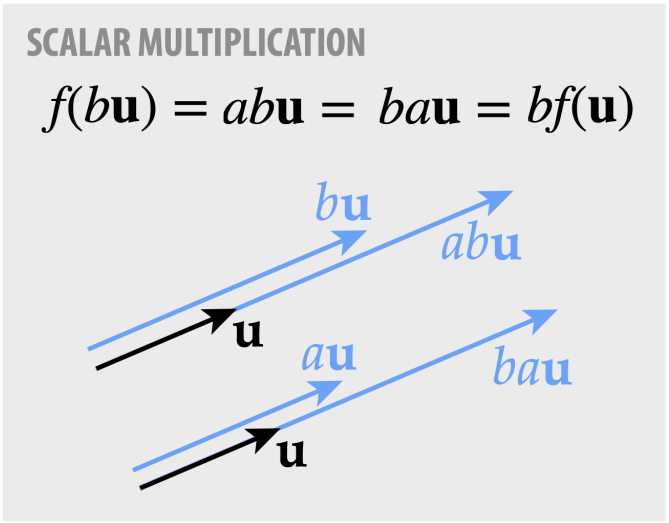
- 保持了所有向量的方向 direction*

$$\frac{\mathbf{u}}{|\mathbf{u}|} = \frac{a\mathbf{u}}{|a\mathbf{u}|}$$

*假定 $a \neq 0, \mathbf{u} \neq \mathbf{0}$

- Q：缩放是线性变换吗?
- A：是的





切变 Shear

- 切变将沿着方向 $\mathbf{u}$ 移动 $\mathbf{x}$，移动的大小由 $\mathbf{x}$ 沿着固定向量 $\mathbf{v}$ 的距离决定：

$$f_{\mathbf{u},\mathbf{v}}(\mathbf{x}) = \mathbf{x} + \langle \mathbf{v}, \mathbf{x} \rangle \mathbf{u}$$

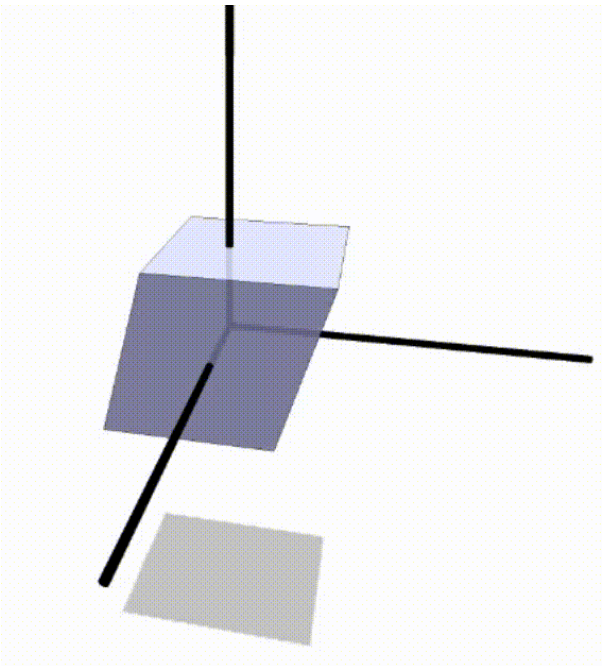
点积计算 $\mathbf{x}$ 沿着 $\mathbf{v}$ 的距离

- Q：此变换是否线性？
- A：Yes，可用如下矩阵表示

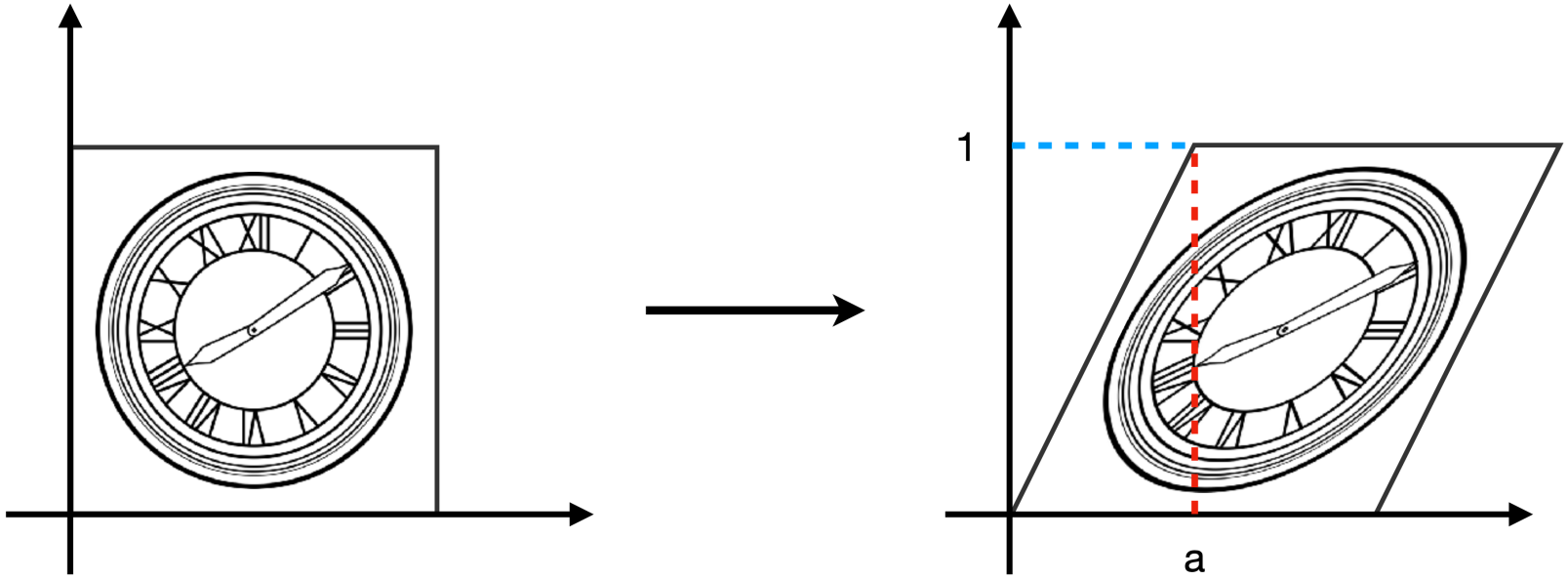
$$A_{\mathbf{u},\mathbf{v}} = I + \mathbf{u}\mathbf{v}^{\mathsf{T}}$$

- 例子

$$\mathbf{u} = (\cos(t), 0, 0)$$
$$\mathbf{v} = (0, 1, 0)$$
$$A_{\mathbf{u},\mathbf{v}} = \begin{bmatrix} 1 & \cos(t) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



倾斜 Shear – 2D



Hints:

Horizontal shift is 0 at y=0

Horizontal shift is a at y=1

Vertical shift is always 0

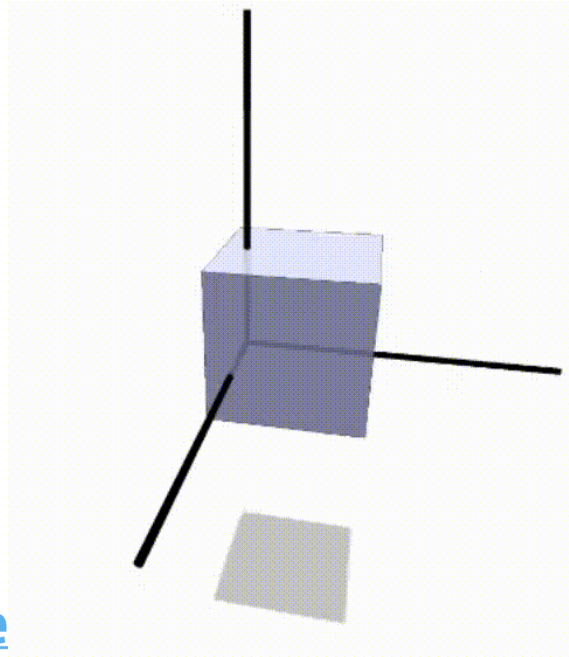
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$f_{\mathbf{u},\mathbf{v}}(\mathbf{x}) = \mathbf{x} + \langle \mathbf{v}, \mathbf{x} \rangle \mathbf{u} \qquad \begin{cases} \mathbf{u} = [1, 0] \\ \mathbf{v} = [0, a] \end{cases}$$

平移 Translations

- 到目前为止，我们忽略了一个基本的变换 – 平移
- 平移只是在给定点 $\mathbf{x}$ 上加一个偏移 $\mathbf{u}$

$$f_{\mathbf{u}}(\mathbf{x}) = \mathbf{x} + \mathbf{u}$$



- Q: 此变换是否线性?
- 让我们检查其是否符合定义

additivity

$$f_{\mathbf{u}}(\mathbf{x} + \mathbf{y}) = \mathbf{x} + \mathbf{y} + \mathbf{u}$$

$$f_{\mathbf{u}}(\mathbf{x}) + f_{\mathbf{u}}(\mathbf{y}) = \mathbf{x} + \mathbf{y} + 2\mathbf{u}$$

homogene

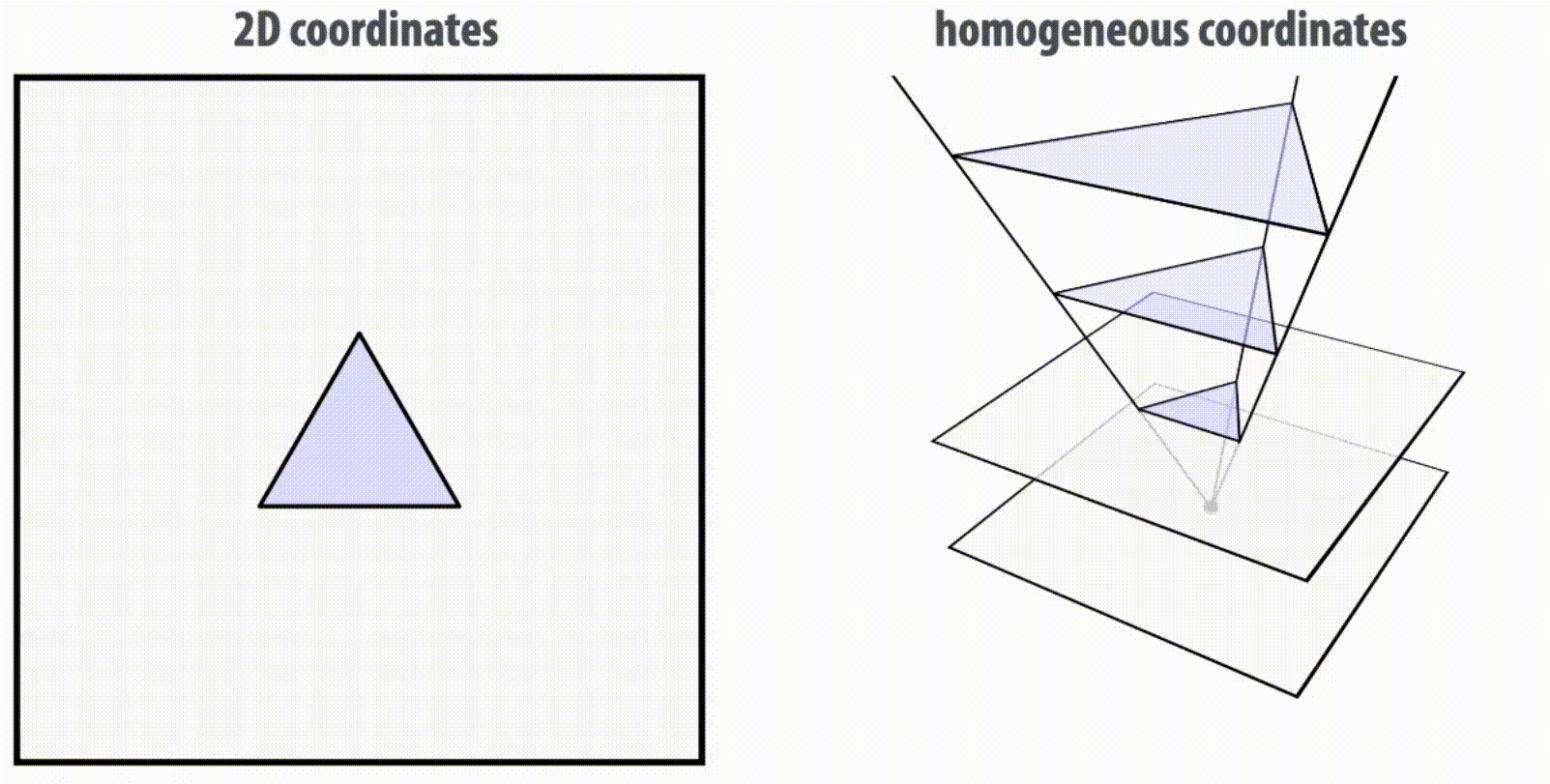
$$f_{\mathbf{u}}(a\mathbf{x}) = a\mathbf{x} + \mathbf{u}$$

$$af_{\mathbf{u}}(\mathbf{x}) = a\mathbf{x} + a\mathbf{u}$$

A：No，平移是仿射，而不是线性的

齐次坐标中的平移

- 让我们思考一下，如果我们对二维坐标 p 进行平移，齐次坐标 $\hat{p}$ 会发生什么



这个看起来像哪种变换？

齐次坐标中的 3D 变换

- 3D (或更高维度) 的情况类似，只需附加一个齐次坐标轴
- 3D 线性变换的矩阵表示只增加一个额外的单位行/列 (identity row/column)；平移同样是切变

point in 3D

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

rotate (x, y, z) around y by θ

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

scale x, y, z by a, b, c

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

shear (x, y) by z in (s, t) direction

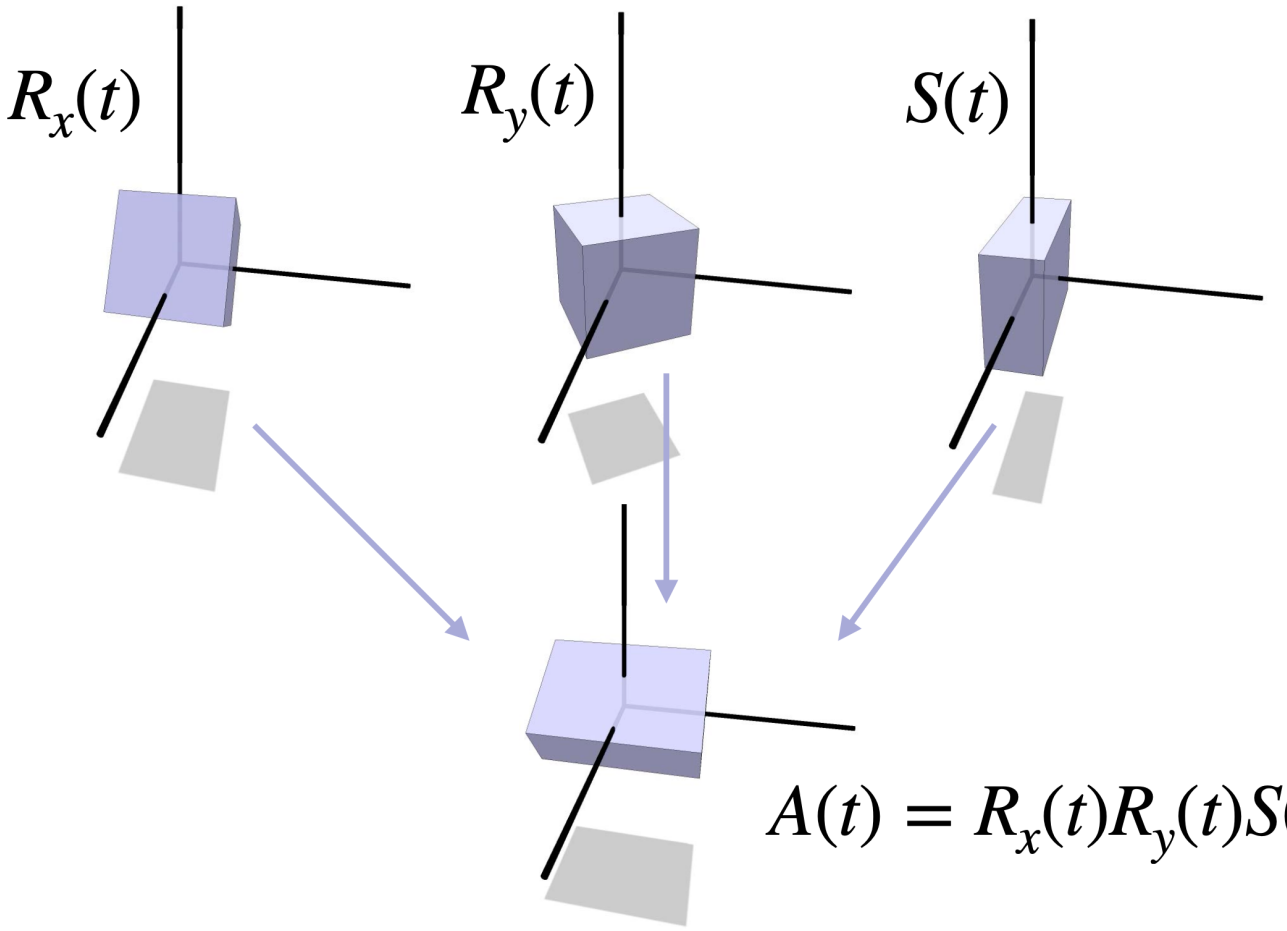
$$\begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & t & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translate (x, y, z) by (u, v, w)

$$\begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

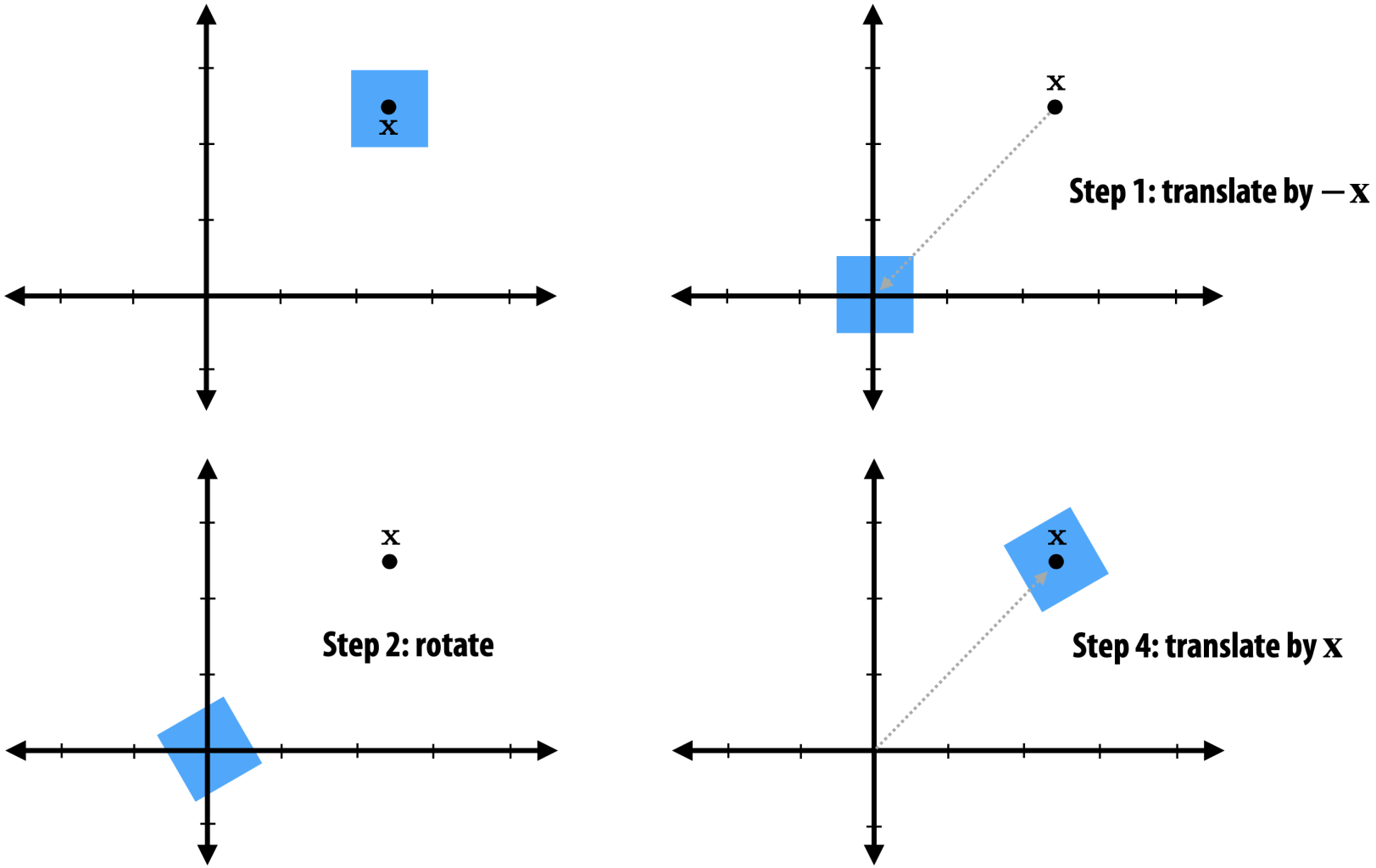
复合变换 Composite Transformations

- 根据这些基本变换 (旋转、反射、缩放、倾斜...)，我们现在可以通过**矩阵乘法**进行复合变换



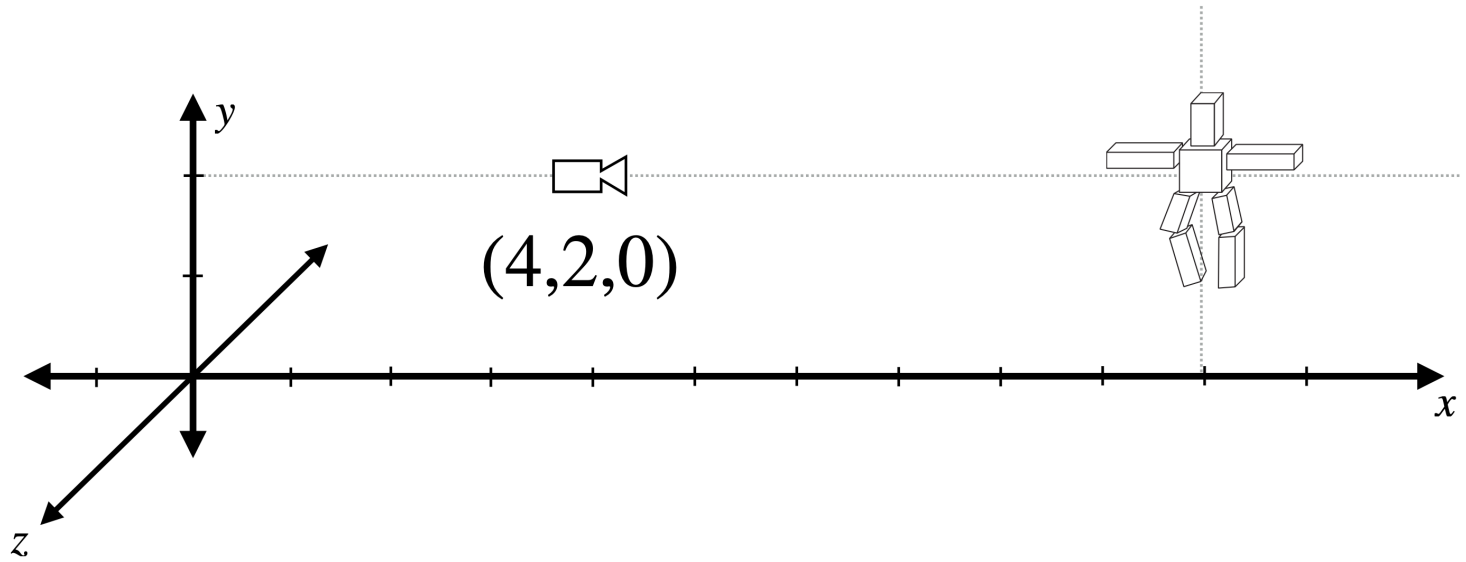
The last matrix gets applied first!

变换的顺序很重要


x
x
Step 1: translate by $-\mathbf{x}$
x
Step 2: rotate
x
Step 4: translate by $\mathbf{x}$

简单的相机变换

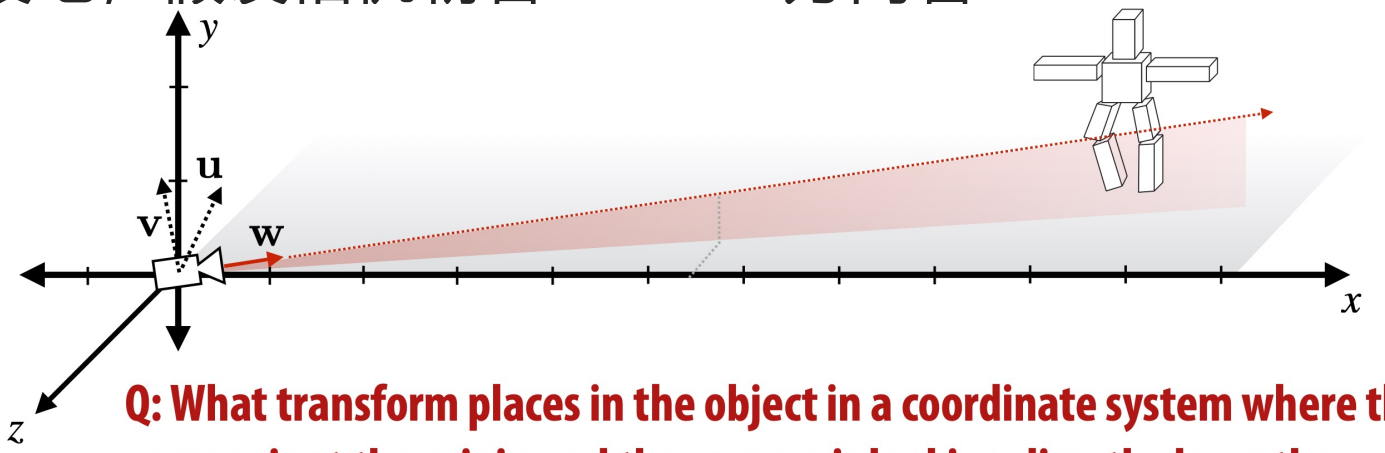
- 考虑一个位于(4, 2, 0)，面向 x 轴的相机，物体在世界坐标中给出



- Q：要应用什么 3D 变换，才能使物体处于相机在原点，且面向 $-z$ 轴的 (标准) 坐标系中？
  - 平移 (-4, -2, 0) 让物体处于相对相机的位置
  - 绕 y 逆时针旋转 π/2 使物体处在新坐标系中，在该坐标系中相机的视角方向与 $-z$ 轴对齐

简单的相机变换

- 更一般地，假设相机朝着 $w \in \mathbb{R}^3$ 方向看



Q: What transform places in the object in a coordinate system where the camera is at the origin and the camera is looking directly down the -z axis?

- 构建与 $\mathbf{w}$ 正交的向量 $\mathbf{u}, \mathbf{v}$
  - 比如，取一个大致朝上且垂直 $\mathbf{w}$ 的 $\mathbf{v}$，然后 $\mathbf{u} := \mathbf{w} \times \mathbf{v}$
- 其对应的矩阵为

$$R = \begin{bmatrix} u_x & v_x & -w_x \\ u_y & v_y & -w_y \\ u_z & v_z & -w_z \end{bmatrix}$$

R maps x-axis to $\mathbf{u}$, y-axis to $\mathbf{v}$, z-axis to $-\mathbf{w}$

Now invert. (How do we do that?)

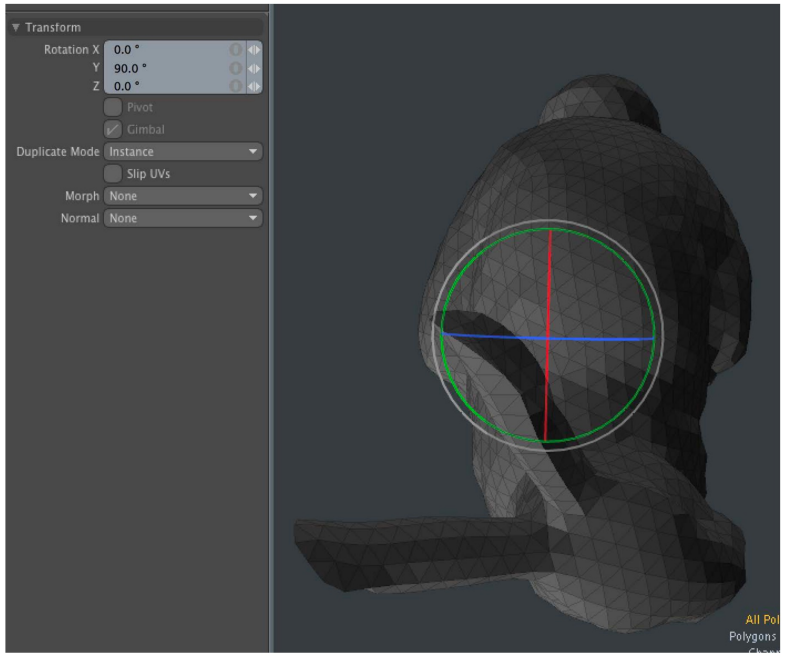
$$R^{-1} = R^{\mathsf{T}}$$

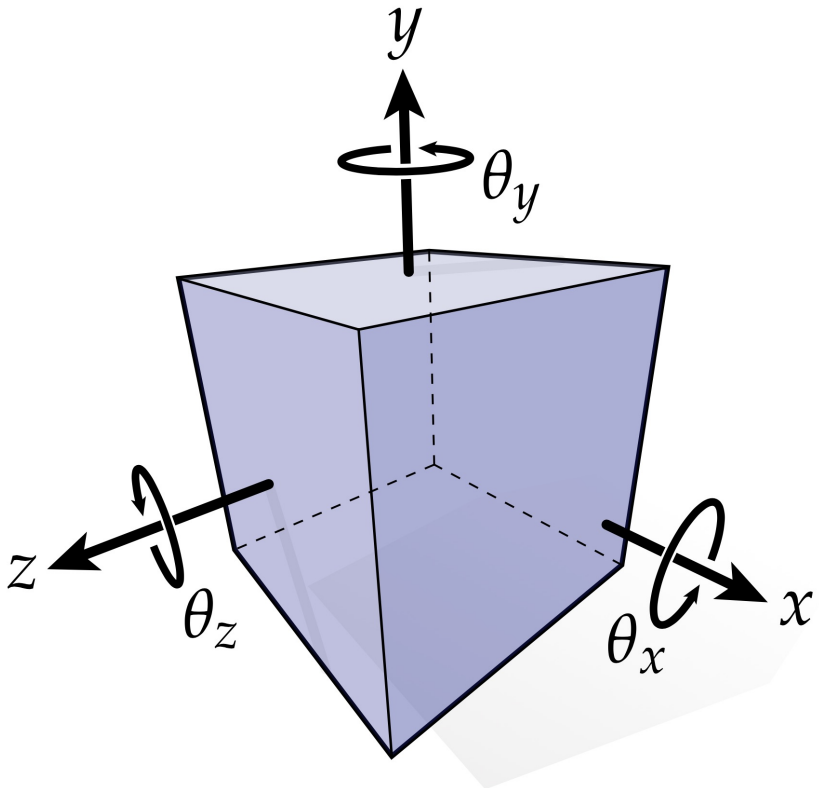
$$\begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ -w_x & -w_y & -w_z \end{bmatrix}$$

Lecture07： 3D 空间变换
3D Transformation

3D 中的旋转表示 – 欧拉角 Euler Angles

- 如何在 3D 中表示旋转?
- 一个简单的想法，能否直接将 2D 中的方法用到 3D 中的 x, y, z 轴?
- 我们称这种方案 (scheme) 为欧拉角，有什么优点?
- “Gimbal lock”万向节锁





Gimbal Lock

- 当使用欧拉角 $\theta_x, \theta_y, \theta_z$ 时，可能会出现无法绕其中一个轴旋转的情况
- 回顾一下绕三个轴的旋转矩阵

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 这些矩阵的乘积表示基于欧拉角的 3D 旋转

$$R_x R_y R_z = \begin{bmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z & \sin\theta_y \\ \cos\theta_z \sin\theta_x \sin\theta_y + \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z - \sin\theta_x \sin\theta_y \sin\theta_z & -\cos\theta_y \sin\theta_x \\ -\cos\theta_x \cos\theta_z \sin\theta_y + \sin\theta_x \sin\theta_z & \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_y \sin\theta_z & \cos\theta_x \cos\theta_y \end{bmatrix}$$

- 考虑一个特殊例子，$\theta_y = \pi/2$ (so $\cos\theta_y = 0, \sin\theta_y = 1$)

$$\Longrightarrow \begin{bmatrix} 0 & 0 & 1 \\ \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z - \sin\theta_x \sin\theta_z & 0 \\ -\cos\theta_x \cos\theta_z + \sin\theta_x \sin\theta_z & \cos\theta_z \sin\theta_x + \cos\theta_x \sin\theta_z & 0 \end{bmatrix}$$

Gimbal Lock 万向节锁

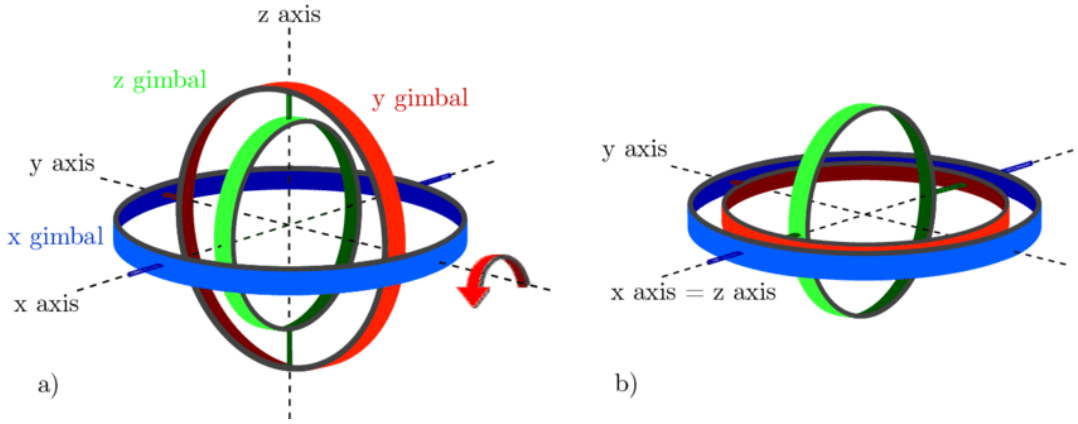
- 简化上一张幻灯片中的矩阵，我们得到

$$\begin{bmatrix} 0 & 0 & 1 \\ \sin(\theta_x+\theta_z) & \cos(\theta_x+\theta_z) & 0 \\ -\cos(\theta_x+\theta_z) & \sin(\theta_x+\theta_z) & 0 \end{bmatrix}$$

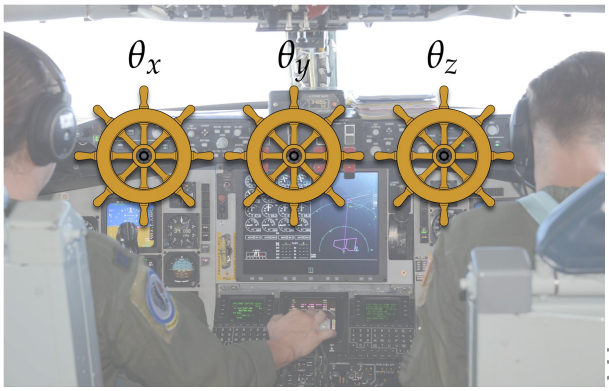
no matter how we adjust θ_x, θ_z, can only rotate in one plane!

Q: What does this matrix do?

- 我们被“锁定”在一个单一的旋转轴上



- 对飞机控制是个很糟糕的设计!

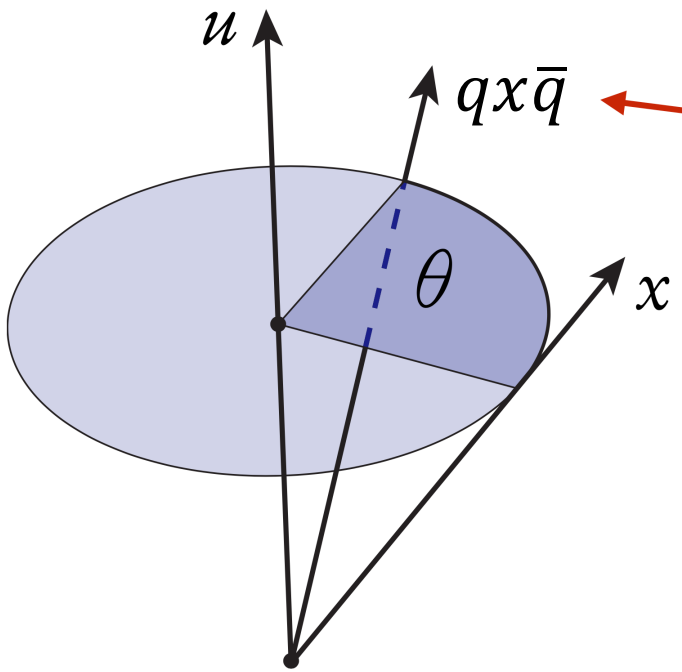


基于四元数的 3D 变换

- 四元数在图形学中的主要应用：3D 旋转 Rotations
- 考虑向量 x (pure imaginary) 和单位四元数 q

$$x \in \mathrm{Im}(\mathbb{H}) \qquad q = w + xi + yj + zk$$

$$q \in \mathbb{H}, \quad |q|^2 = 1 \qquad w^2 + x^2 + y^2 + z^2 = 1$$



always expresses *some* rotation

$\bar{q}$ 是 q 的共轭复数 (conjugate), 通过改变虚部的符号得到, i.e.,

$$\bar{q} = w - xi - yj - zk$$

Visualizing quaternions: https://eater.net/quaternions

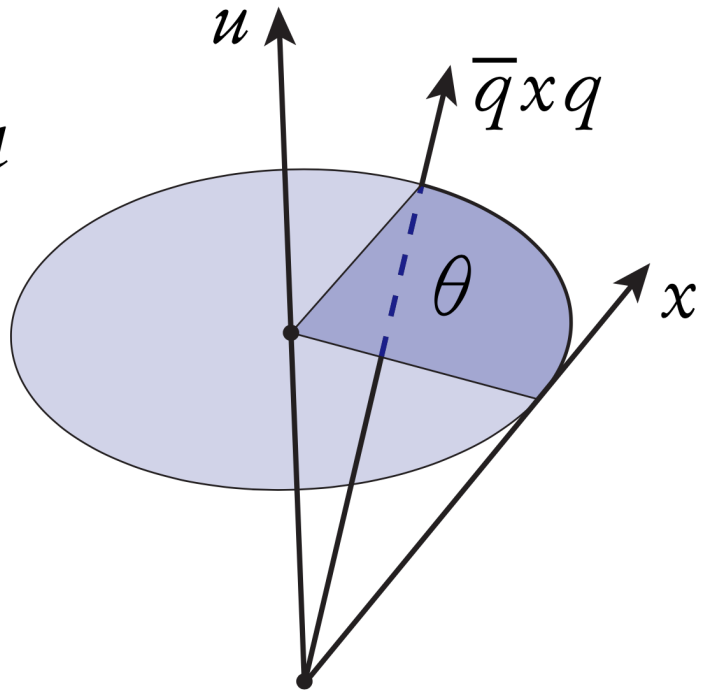
回顾从轴/角度旋转

□给定轴 $\mathbf{u}$，角度 θ，表示旋转的四元数 q 为

$$q = \cos(\theta/2) + \sin(\theta/2)u$$

Angle

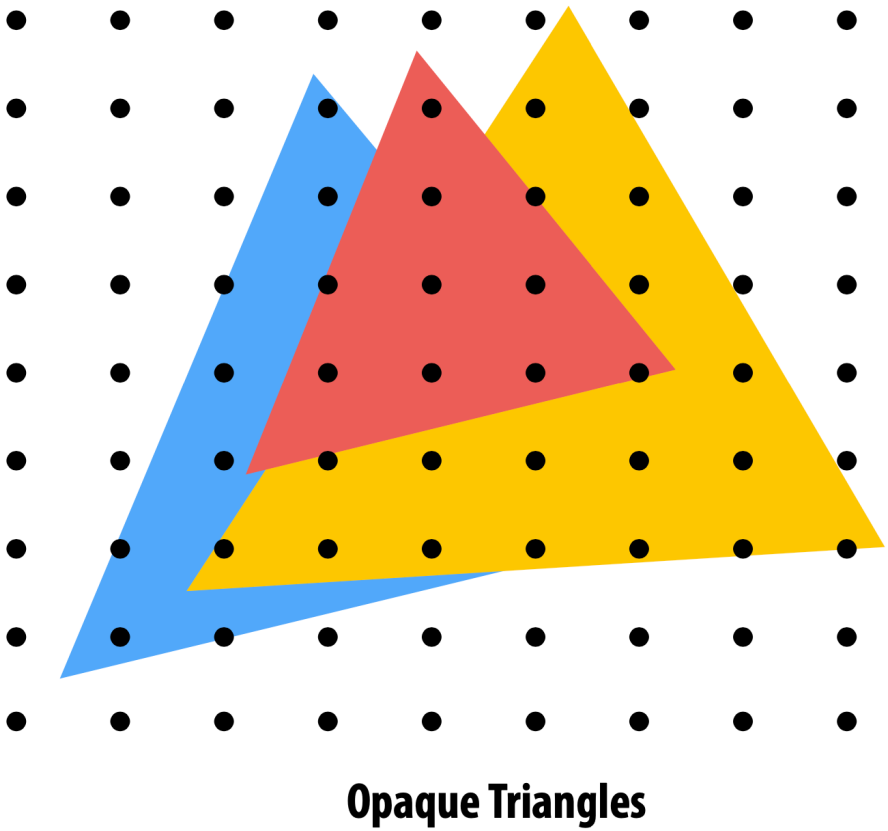
Unit vector



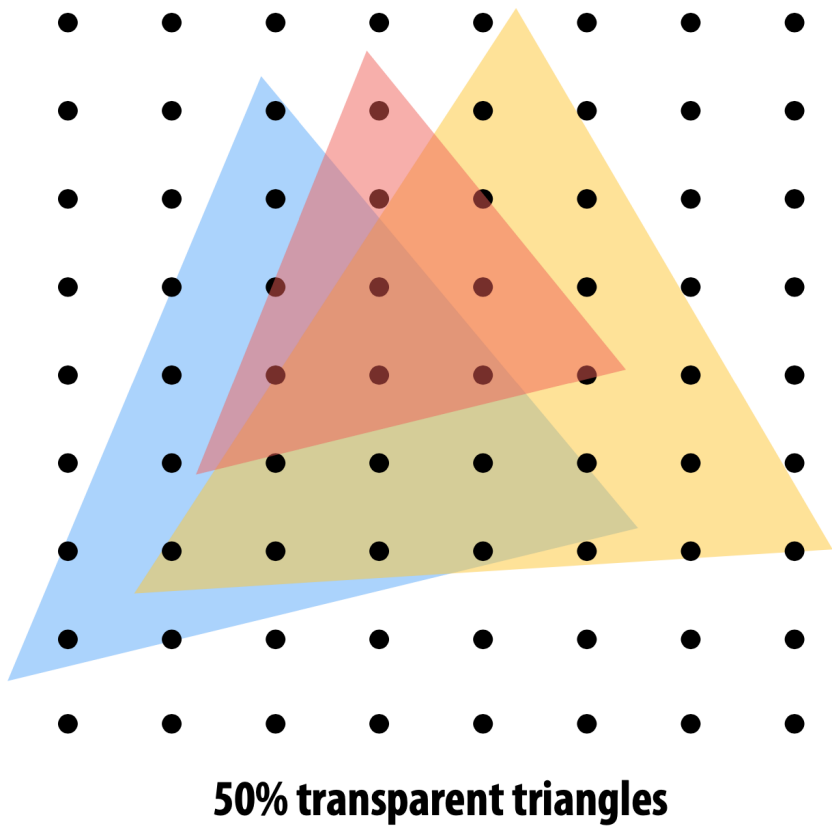
Lecture09：深度和透明度
Depth and Transparency

遮挡

- 在每个采样点上，哪个三角形可见?



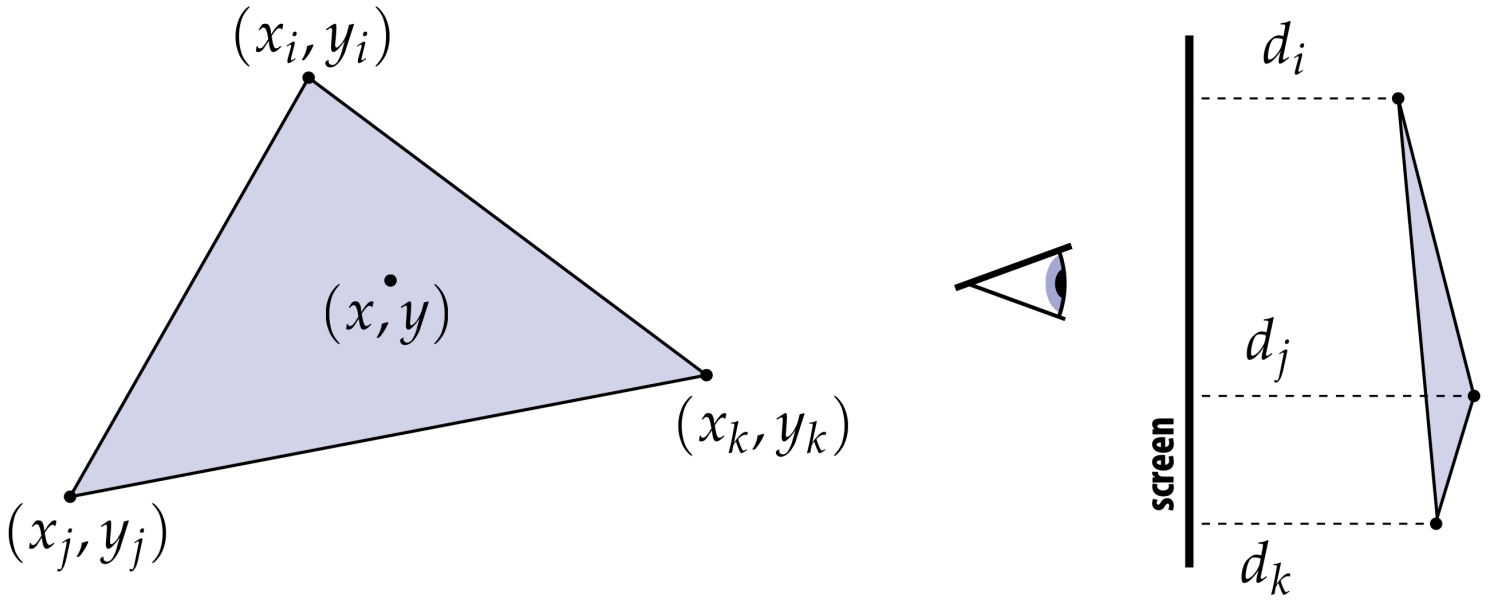
Opaque Triangles



50% transparent triangles

采样深度 Depth

- 假设我们有如下三角形：
  - 每个顶点的 2D 投影坐标为 (x_i, y_i)
  - 每个顶点的深度 (点到观察者的距离) 为 d_i

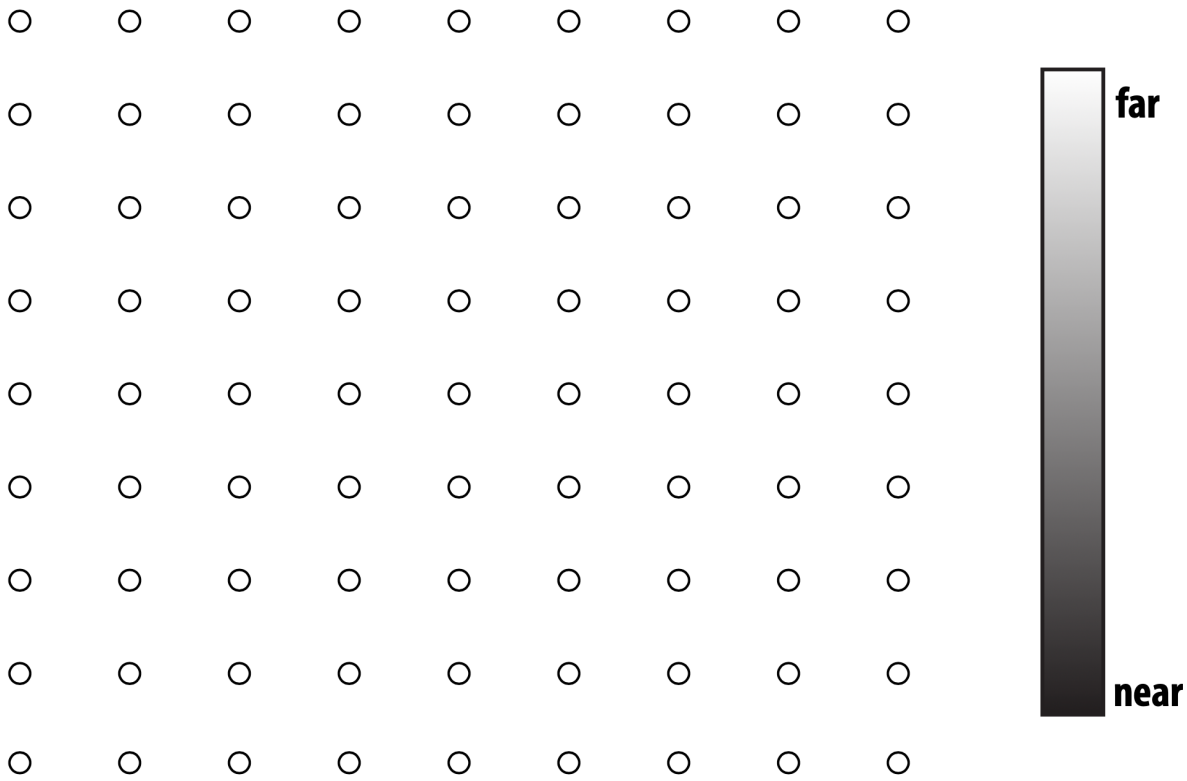


Q：如何计算任意给定点 (x, y) 的深度 d?

A： 使用重心坐标进行插值（与三角形其他线性变化的属性一样，如颜色）

深度缓冲 Depth-buffer (Z-buffer)

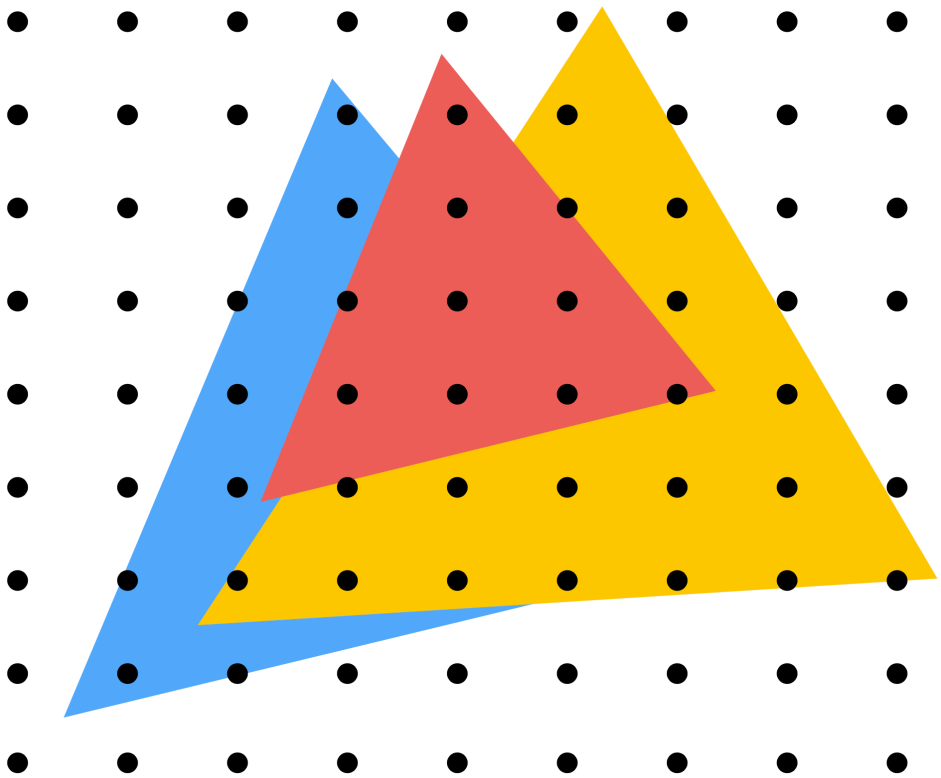
- 对于每个样本，深度缓冲存储到目前为止看到的最近三角形的深度



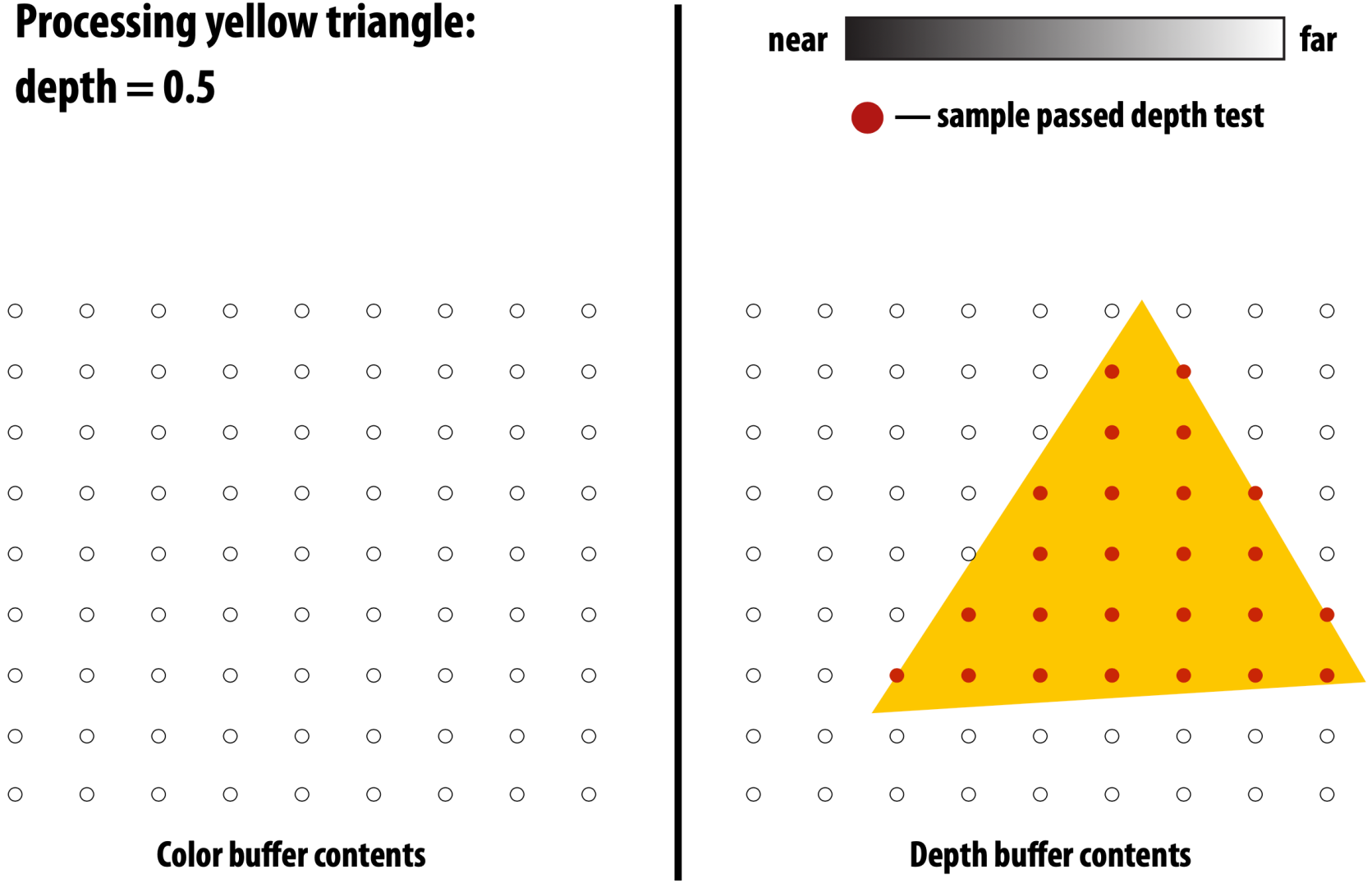
将所有深度缓冲值初始化为“无穷大” （最大值）

深度缓冲示例

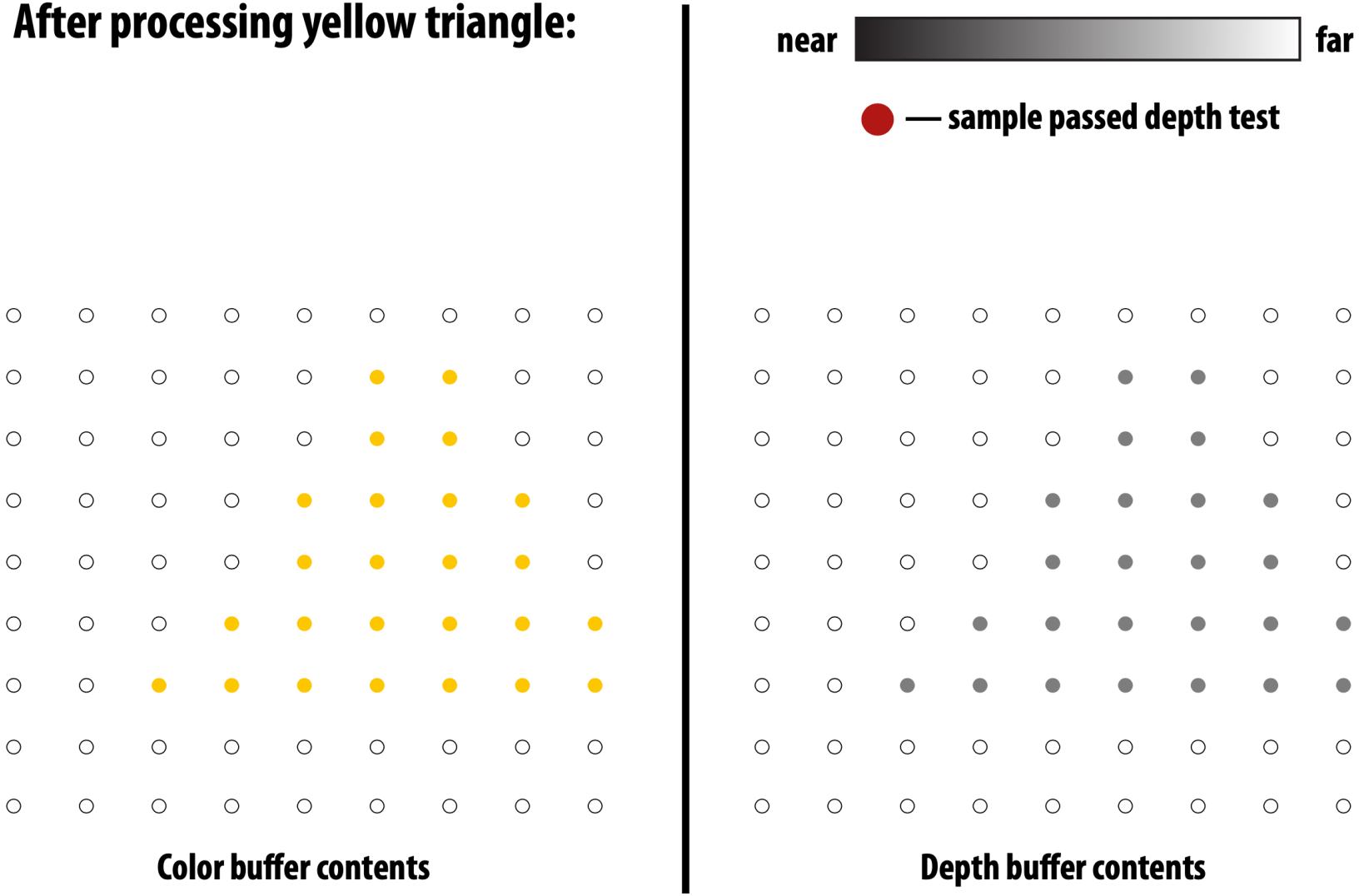
示例：渲染三个不透明三角形



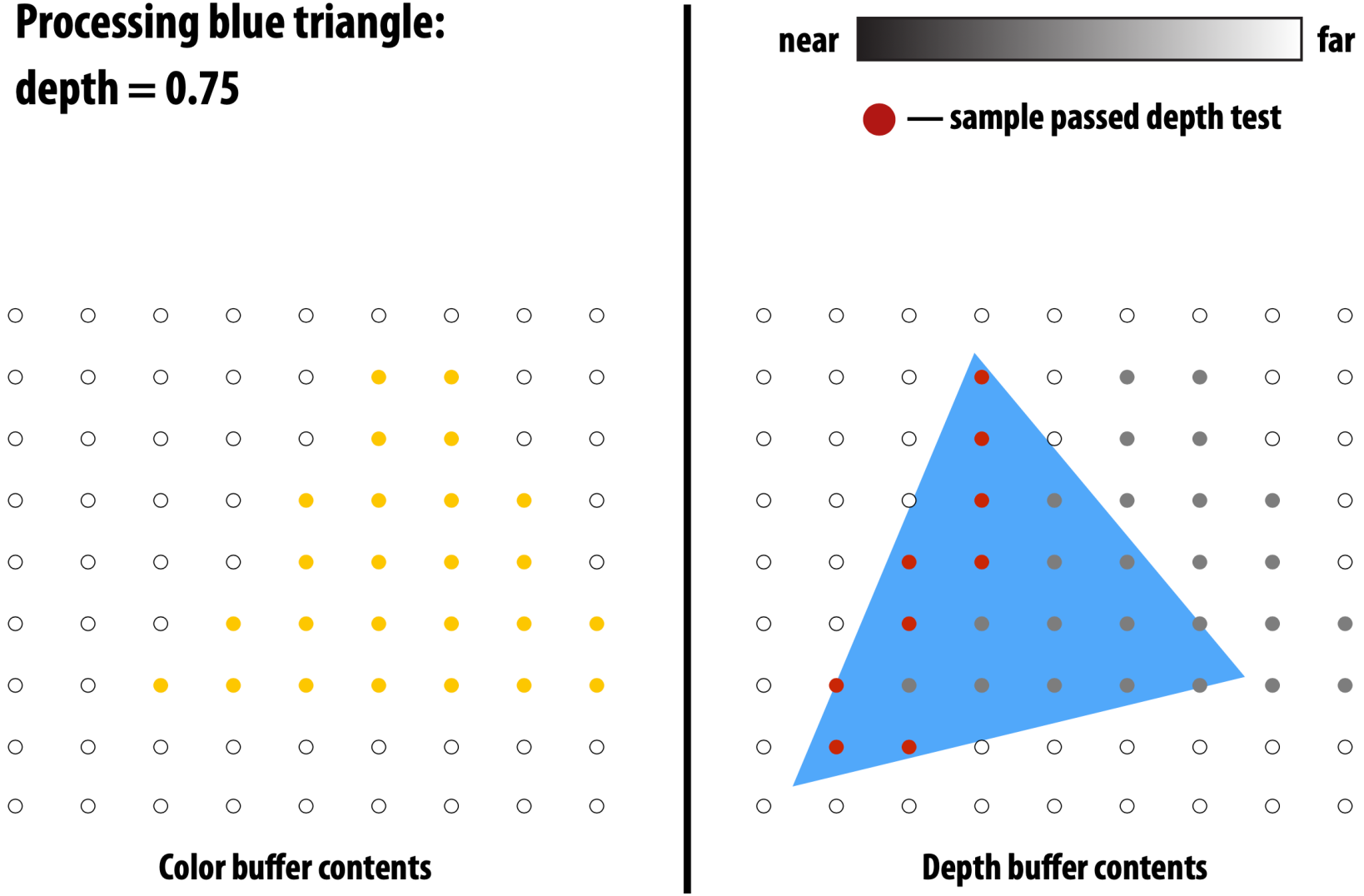
使用深度缓冲 (z-buffer) 进行遮挡



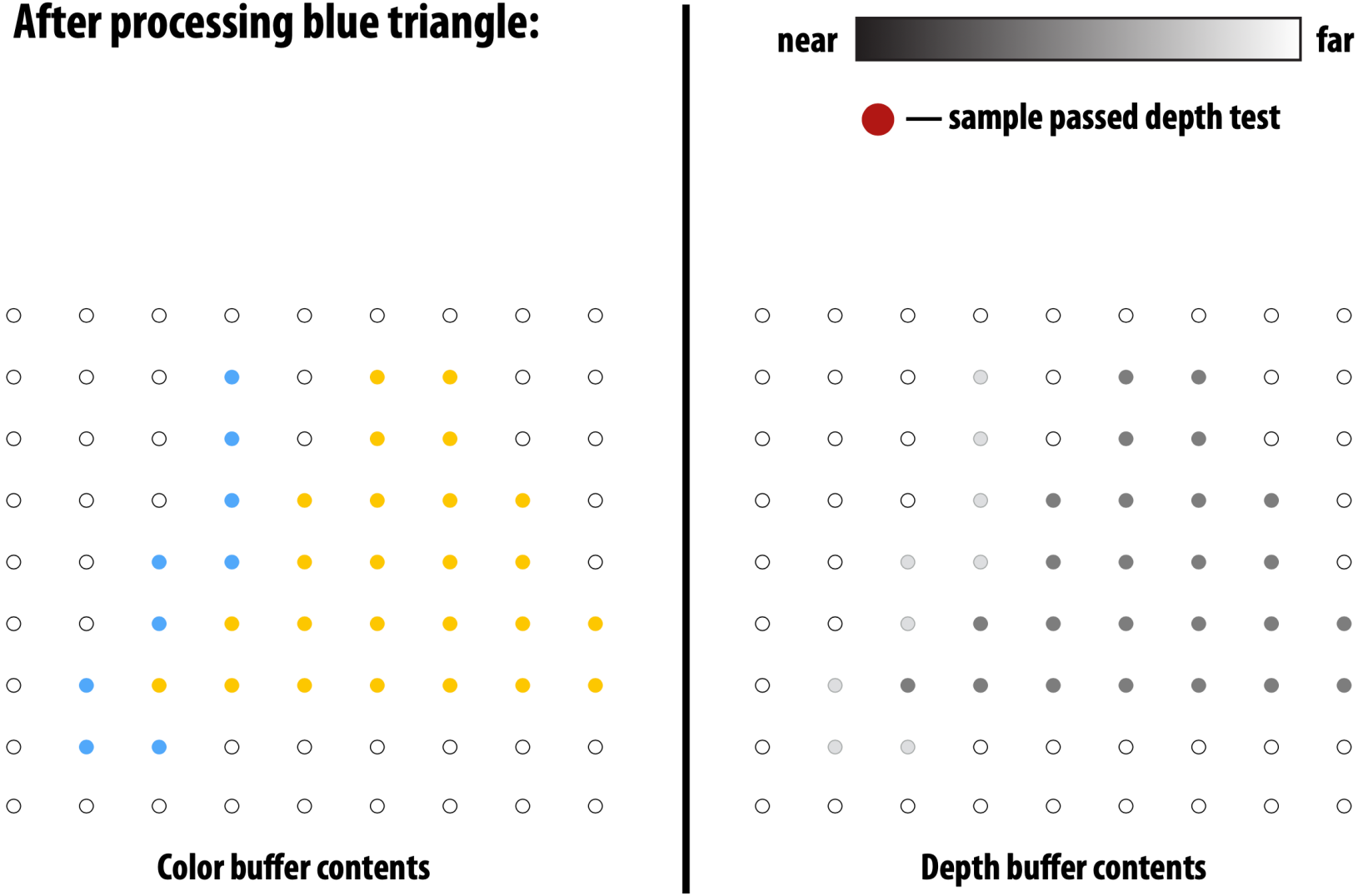
使用深度缓冲 (z-buffer) 进行遮挡



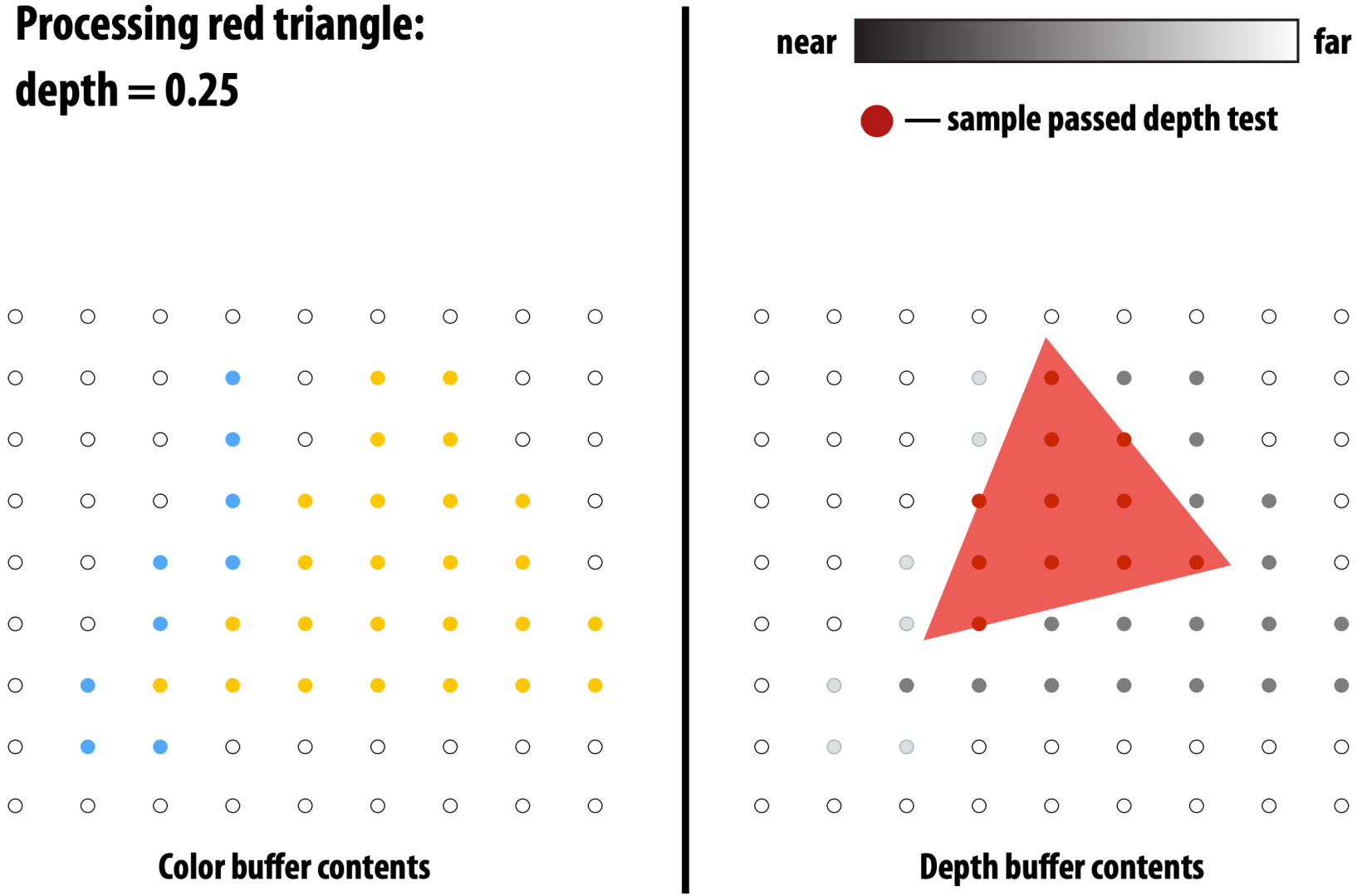
使用深度缓冲 (z-buffer) 进行遮挡



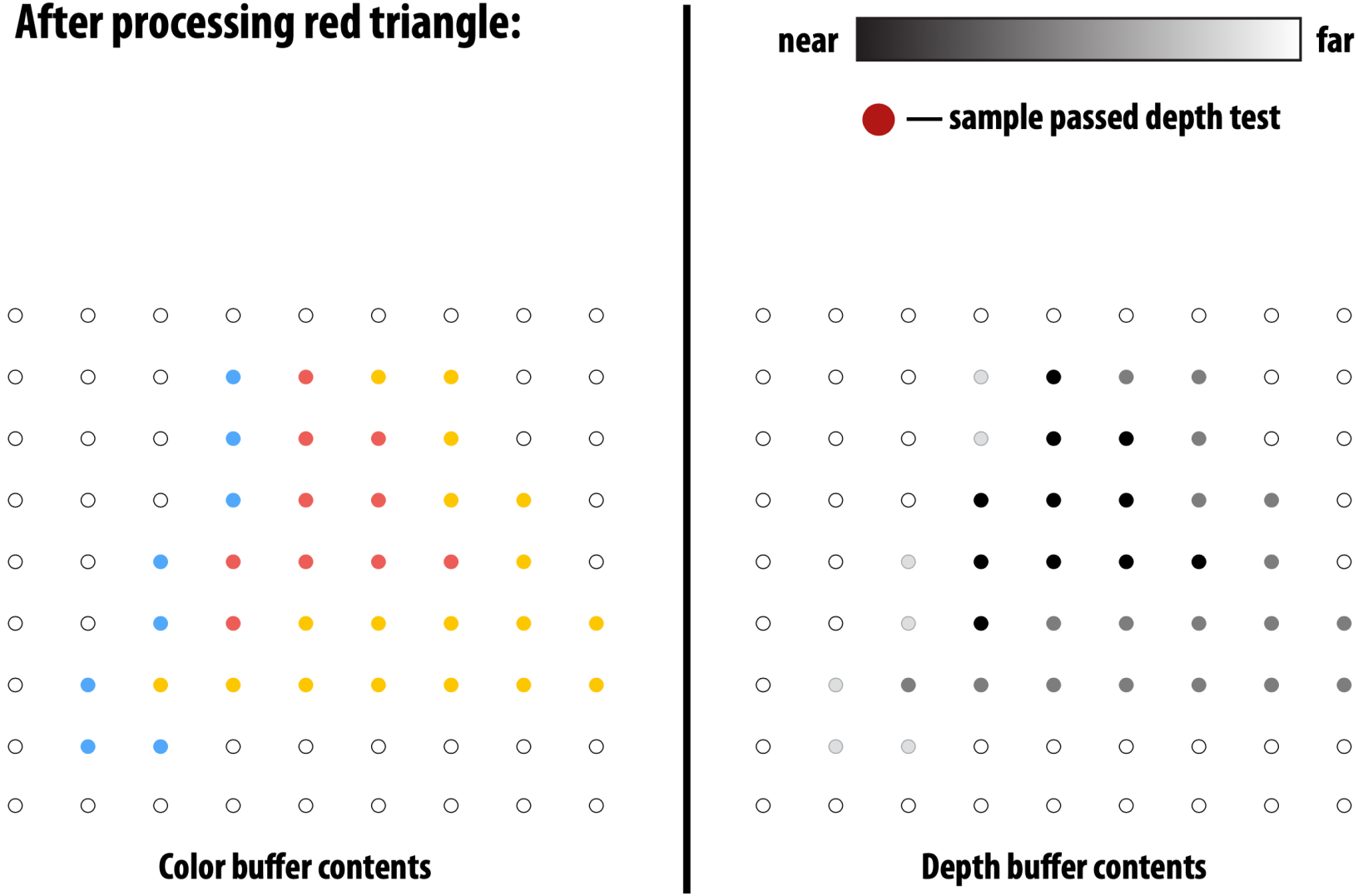
使用深度缓冲 (z-buffer) 进行遮挡



使用深度缓冲 (z-buffer) 进行遮挡



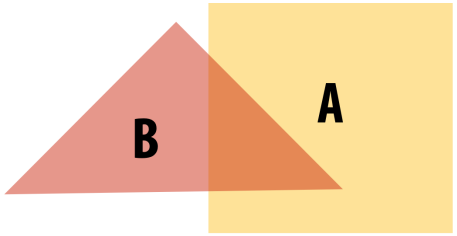
使用深度缓冲 (z-buffer) 进行遮挡



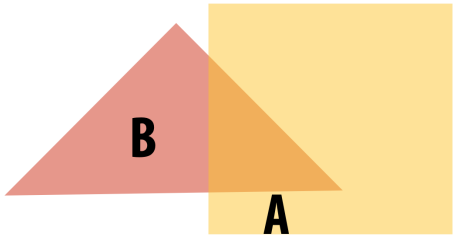
Does the order matter?

Over 算子 Operator

口将透明度为 α_A 的照片 A 与透明度为 α_B 的照片 B 混合



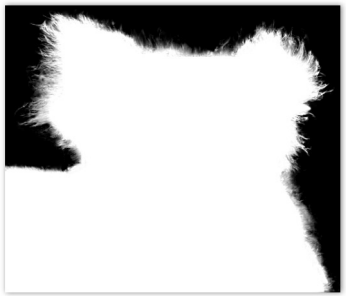
B over A



A over B

Notice: "over" is not commutative

$$A \text{ over } B \neq B \text{ over } A$$

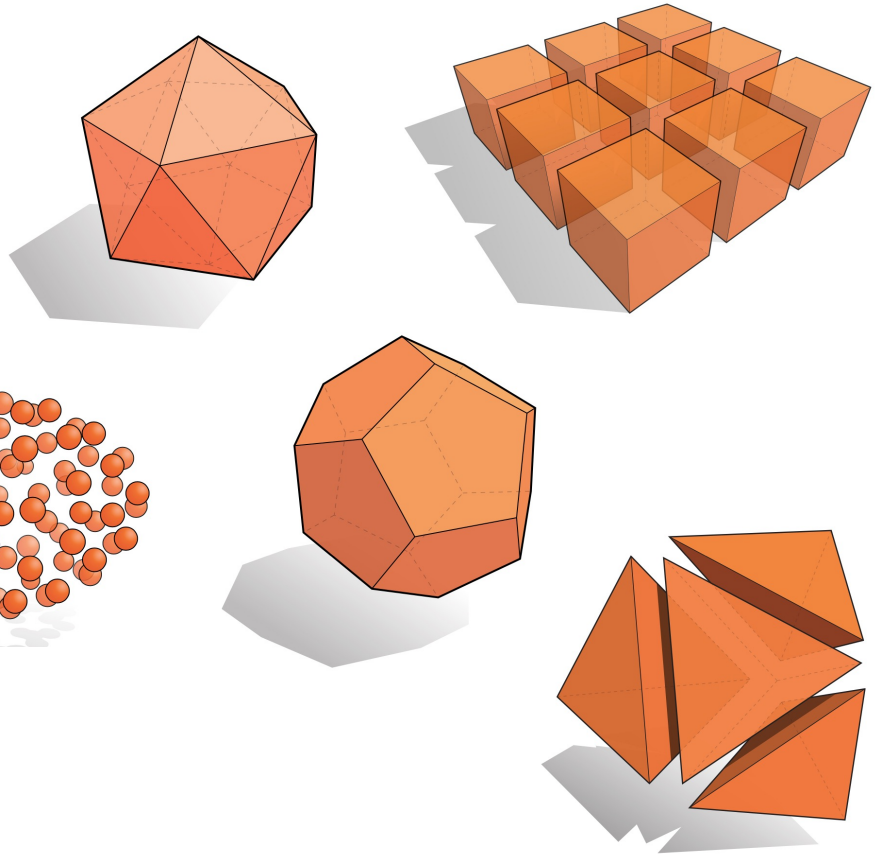


Koala over NYC

Lecture10：几何
Geometry

很多数字化编码几何图形的方式

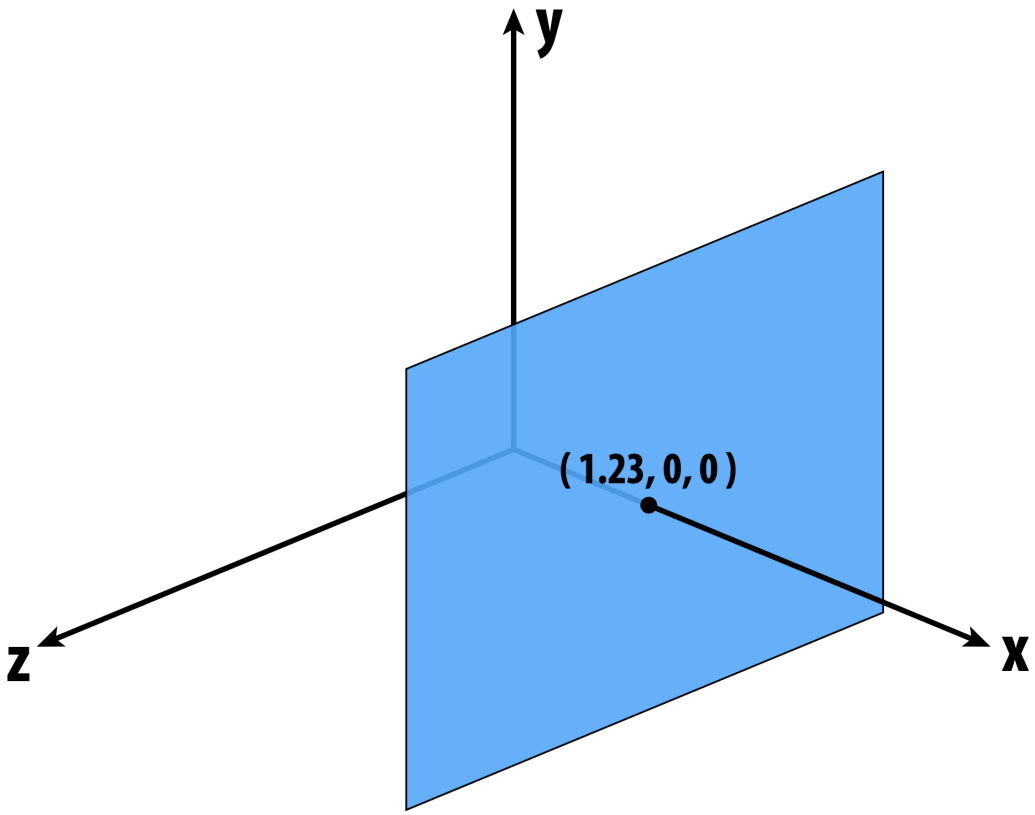
- **显式的 (explicit)**
  - 点云 (point cloud)
  - 多边形网格 (polygon mesh)
  - 细分 (subdivision)，NURBS
  - …
- **隐式的 (implicit)**
  - 水平集 (level set)
  - 代数曲面 (algebraic surface)
  - L-系统 (L-systems)
  - …



- 每种选择适合不同的任务/几何体类型
- 有时可能需要在不同表示方法之间来回转换

比如说，$f(x,y,z)=x-1.23$

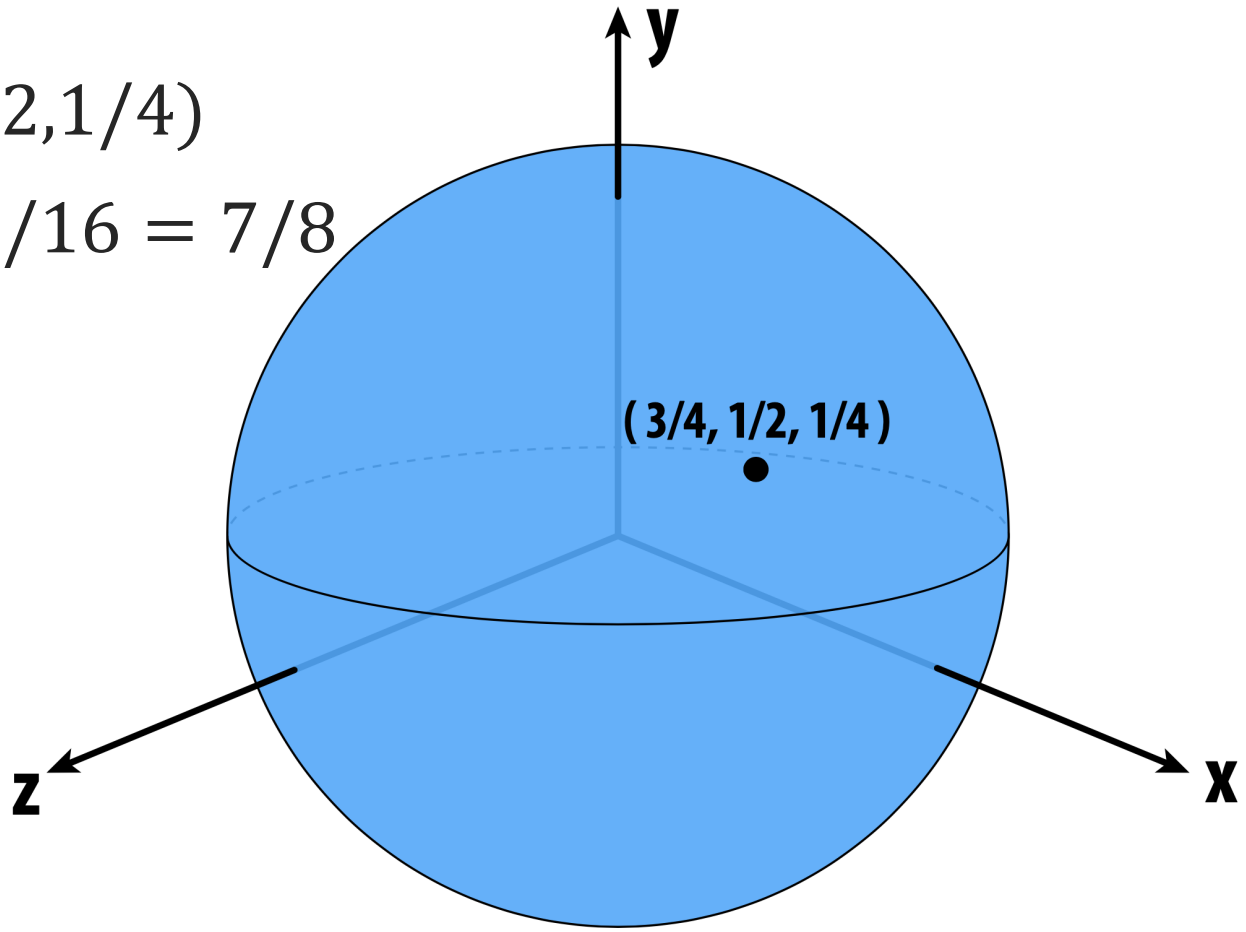
- 满足 $x=1.23$ 的点均在此平面上



- 观察：隐式曲面会使一些任务变得困难，比如采样

检查某点是否在单位球体内

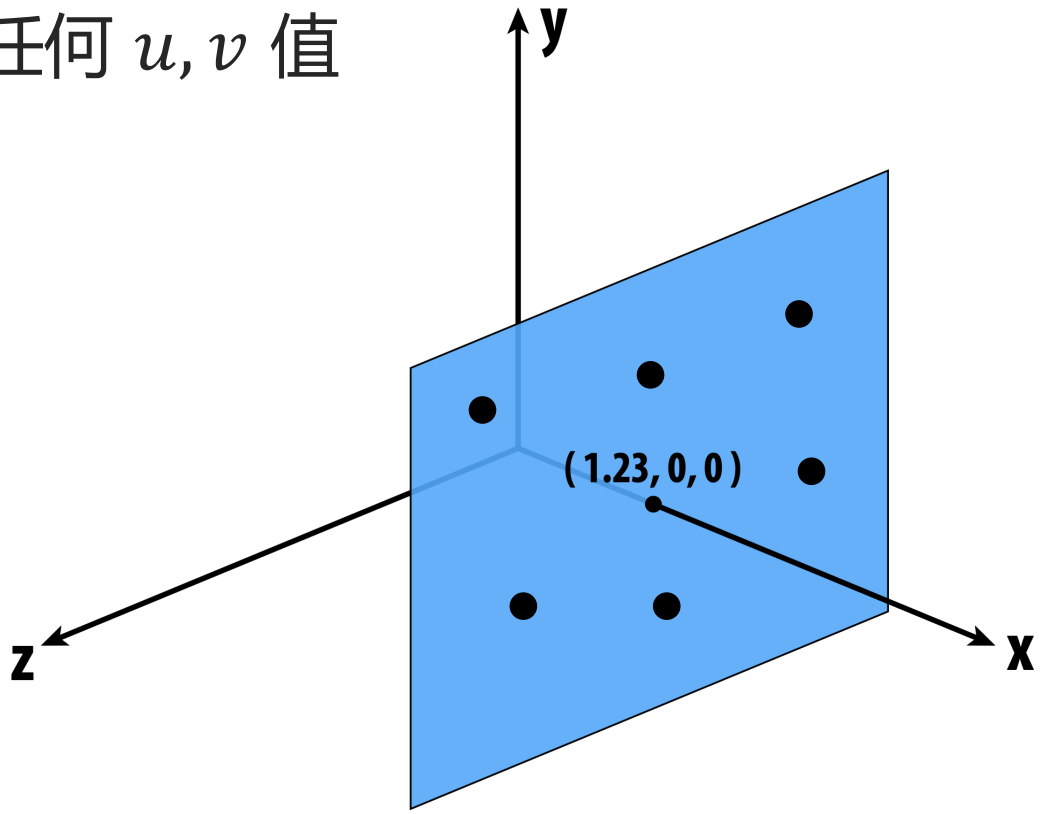
- 比如说点 $(3/4,1/2,1/4)$
- $9/16 + 4/16 + 1/16 = 7/8$
- $7/8 < 1$
- Yes.



- 隐式曲面让另一些任务变得简单，比如判断点是否在几何内部/外部

采样一个显式表面

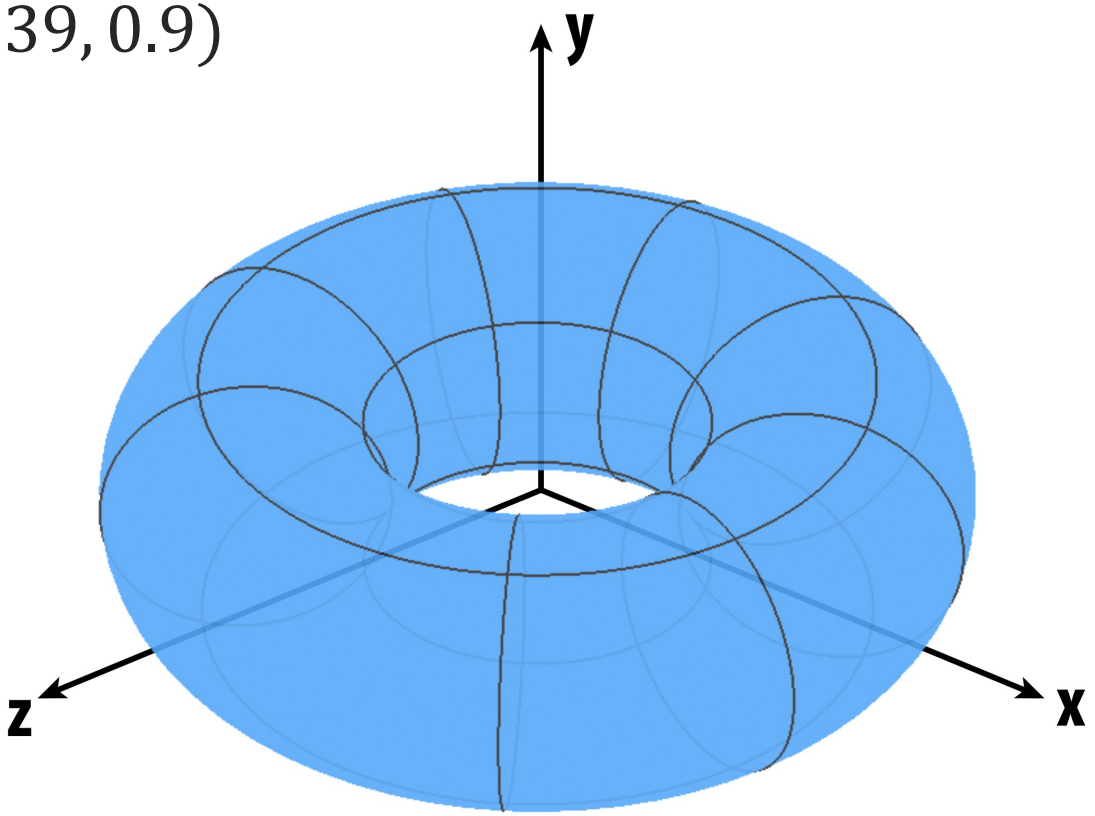
- 表面为：$f(u,v) = (1.23, u, v)$
- 只需要带入任何 u, v 值



- 显式表面让一些任务变得容易，比如采样

检查某点是否在环面内

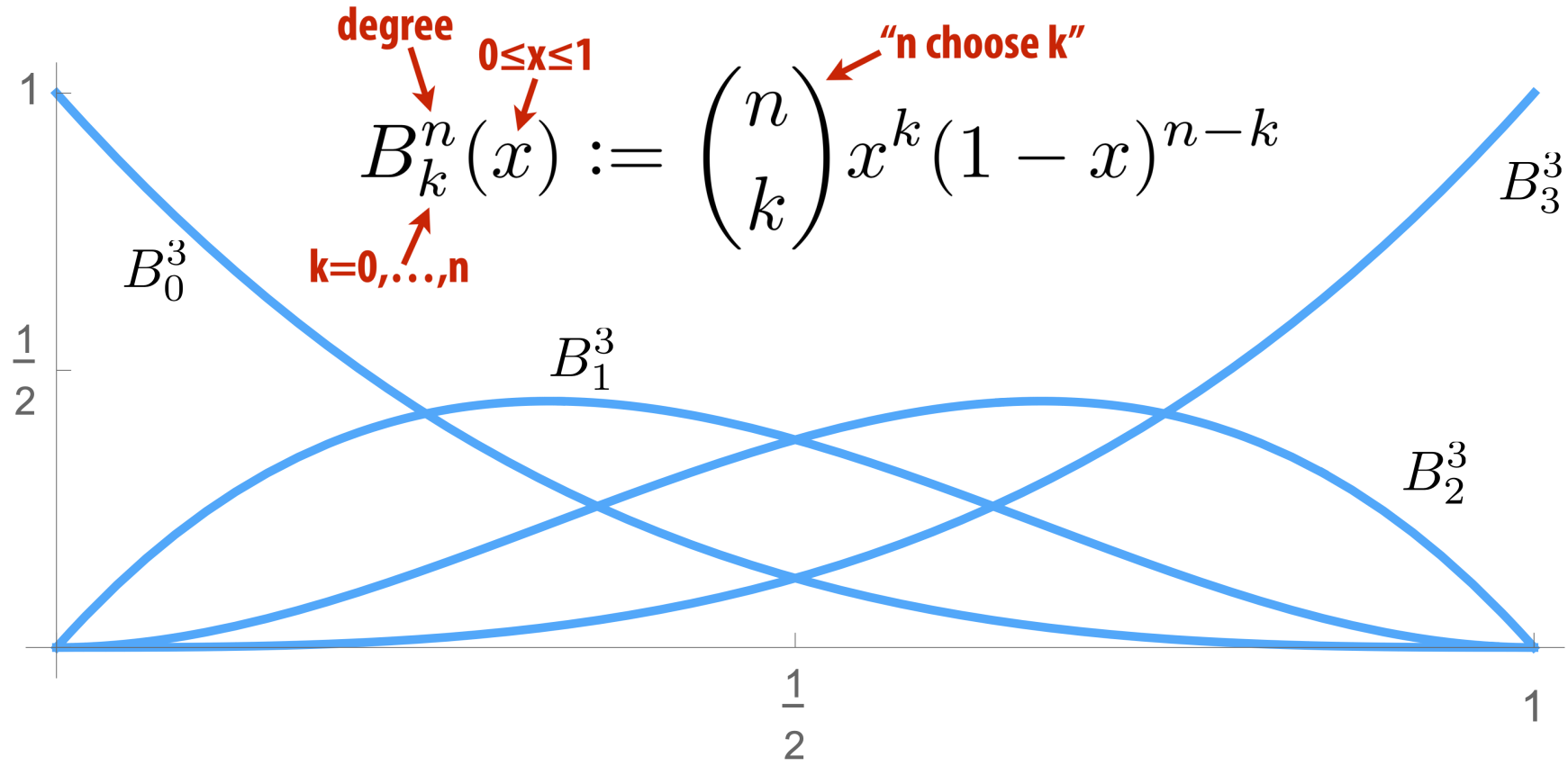
- 表面为 $f(u,v) = ((2+\cos u)\cos v, (2+\cos u)\sin v, \sin u)$
- 比如说点 $(1.96, -0.39, 0.9)$
- ...No



- 显式表面会使另一些任务变得困难，比如内部/外部测试

伯恩斯坦基 (Bernstein Basis)

- 线性插值本质上使用一阶多项式 (1st-order polynomials)
- 通过使用高阶多项式提供更大的灵活性
- 使用 Bernstein 基，而不是通常的基 $(1, x, x^2, x^3, \ldots)$



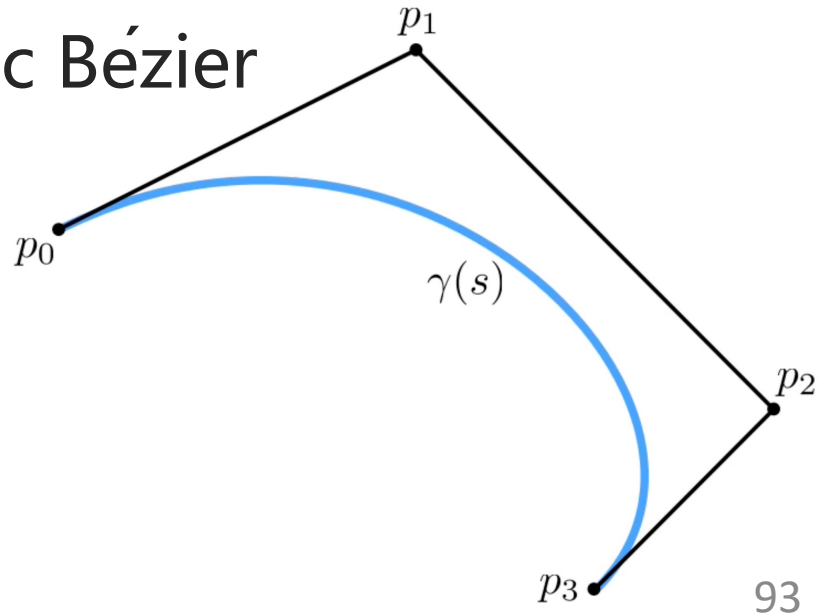
贝塞尔曲线 Bézier Curves (显式表达)

- 贝塞尔曲线是利用 Bernstein 基表示的曲线

$$\gamma(s) := \sum_{k=0}^{n} B_{n,k}(s) p_k$$

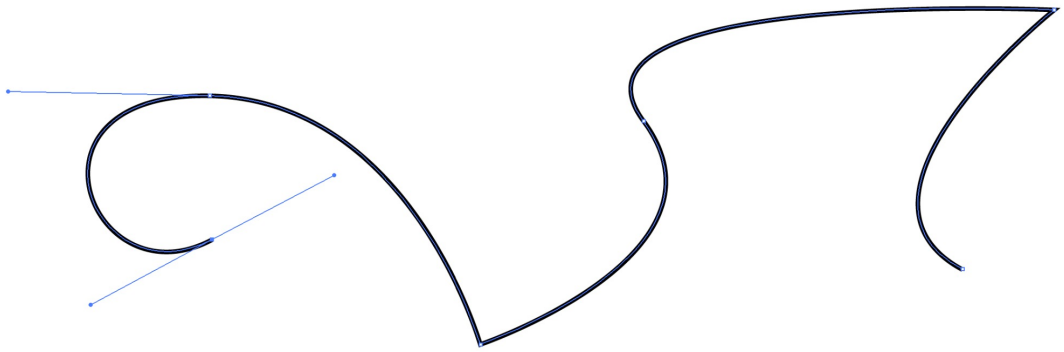
control points

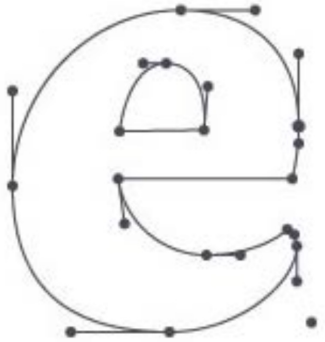
- $n = 1$，为一条线段
- $n = 3$，为立方贝塞尔曲线 cubic Bézier
- 重要特征：
  - 1. 穿过端点
  - 2. 与端段相切



分段贝塞尔曲线 (显式表达)

- 另一种想法：将许多低阶 Bézier 曲线拼接在一起
- 广泛使用在字体、SVG 等





- 分段 Bézier 曲线：

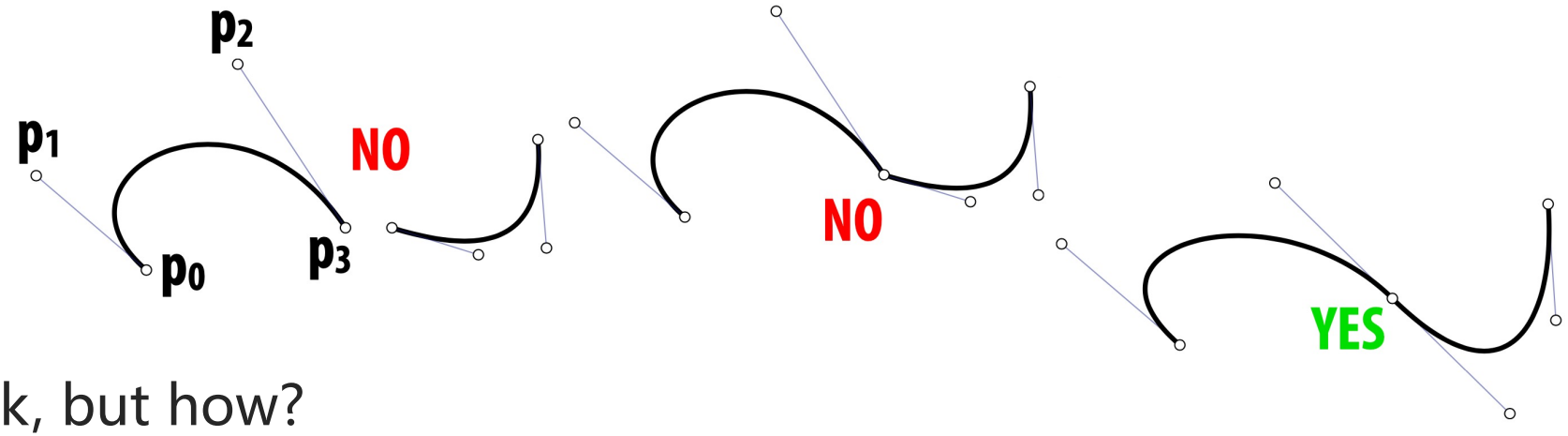
piecewise Bézier

$$\gamma(u) := \gamma_i\left(\frac{u-u_i}{u_{i+1}-u_i}\right), \qquad u_i \le u < u_{i+1}$$

single Bézier

贝塞尔曲线 – 切线连续性 (tangent continuity)

- 要获得**无缝自然**的曲线，需要点和切线对齐



- Ok, but how?
- 每条曲线是一个立方曲线

$$u^3 p_0 + 3u^2(1-u)p_1 + 3u(1-u)^2 p_2 + (1-u)^3 p_3$$

- 希望每个分段的端点相交
- 希望端点处的切线对齐
- **Q：有多少约束 constraints 与自由度 degrees of freedom?**
- **Q：二次贝塞尔曲线呢？线性贝塞尔曲线呢?**

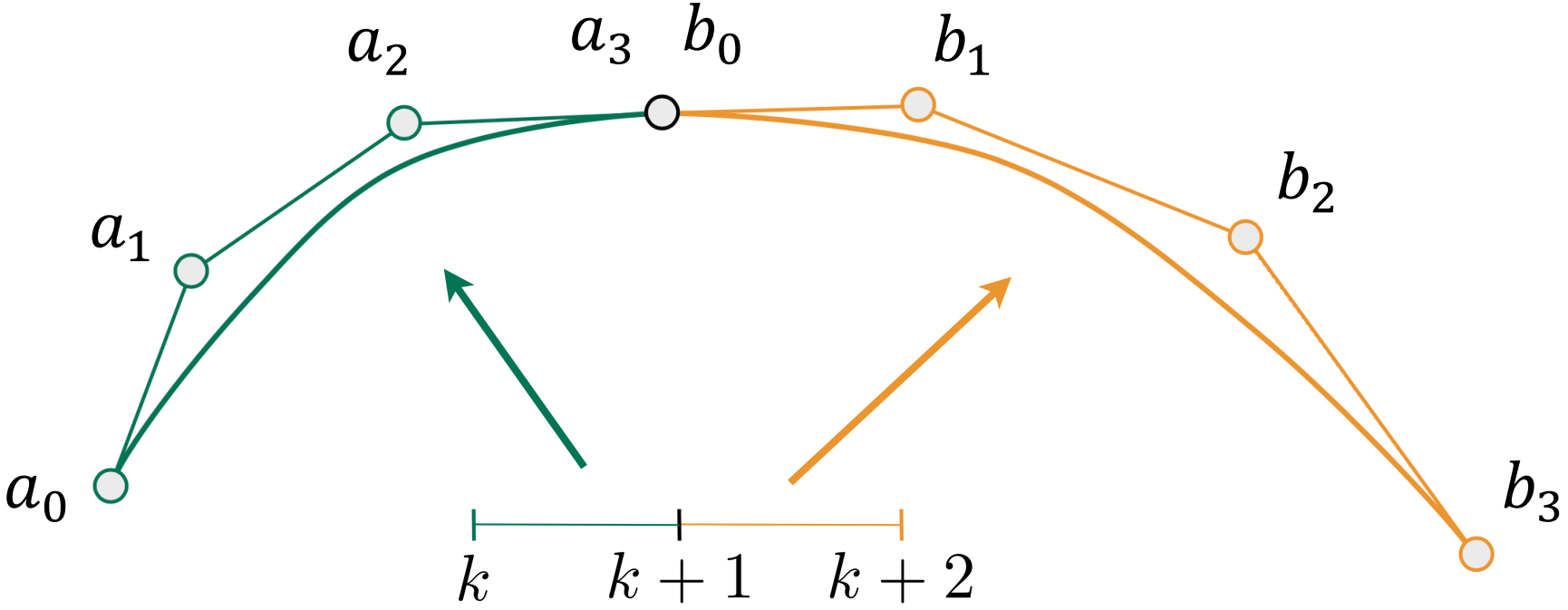
贝塞尔曲线 – 连续性 (Continuity)

- 两条贝塞尔曲线

$$\mathbf{a} : [k, k+1] \to \mathbb{R}^N$$

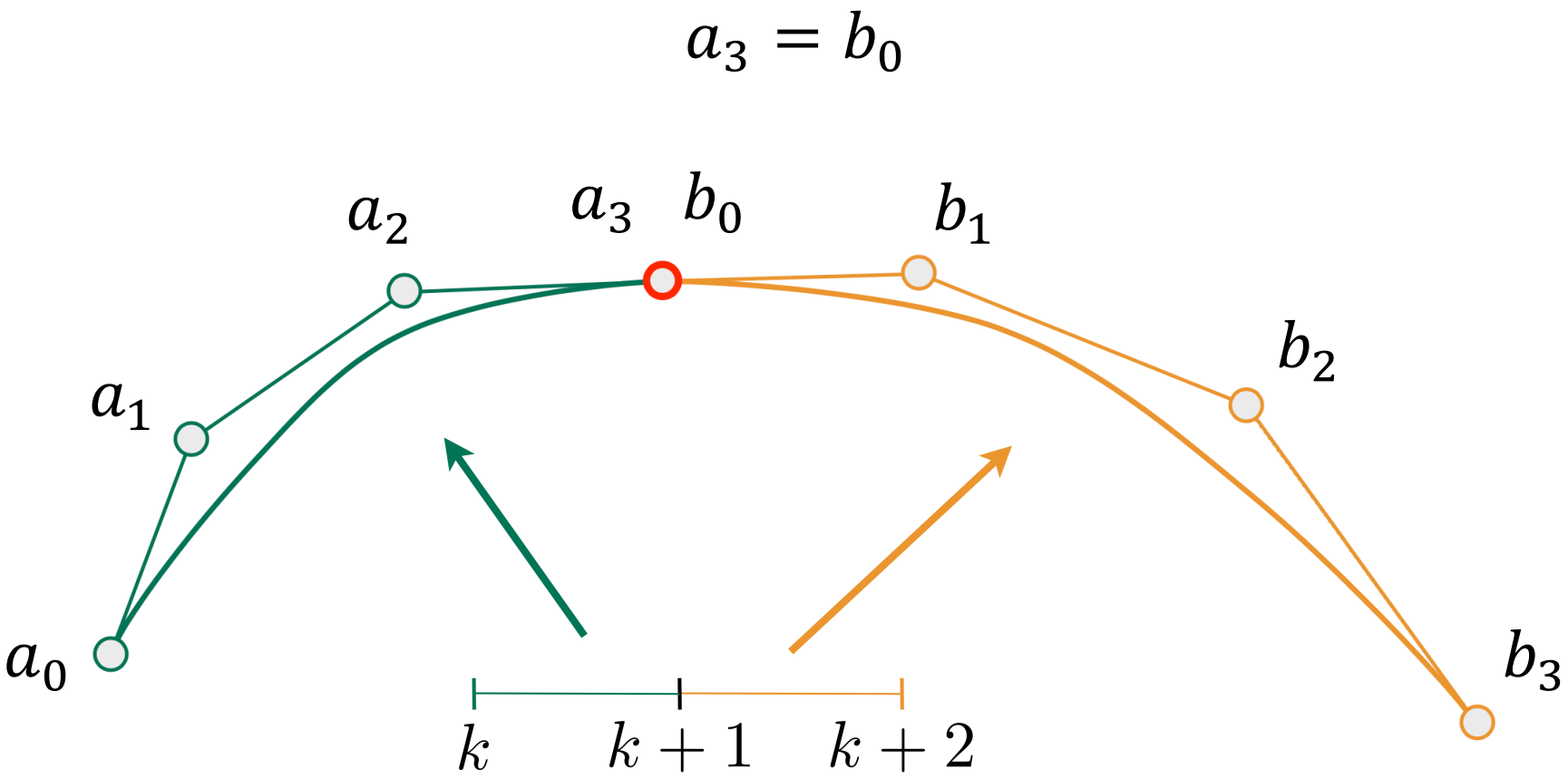
$$\mathbf{b} : [k+1, k+2] \to \mathbb{R}^N$$

Assuming integer partitions here, can generalize



贝塞尔曲线 – 连续性 (Continuity)

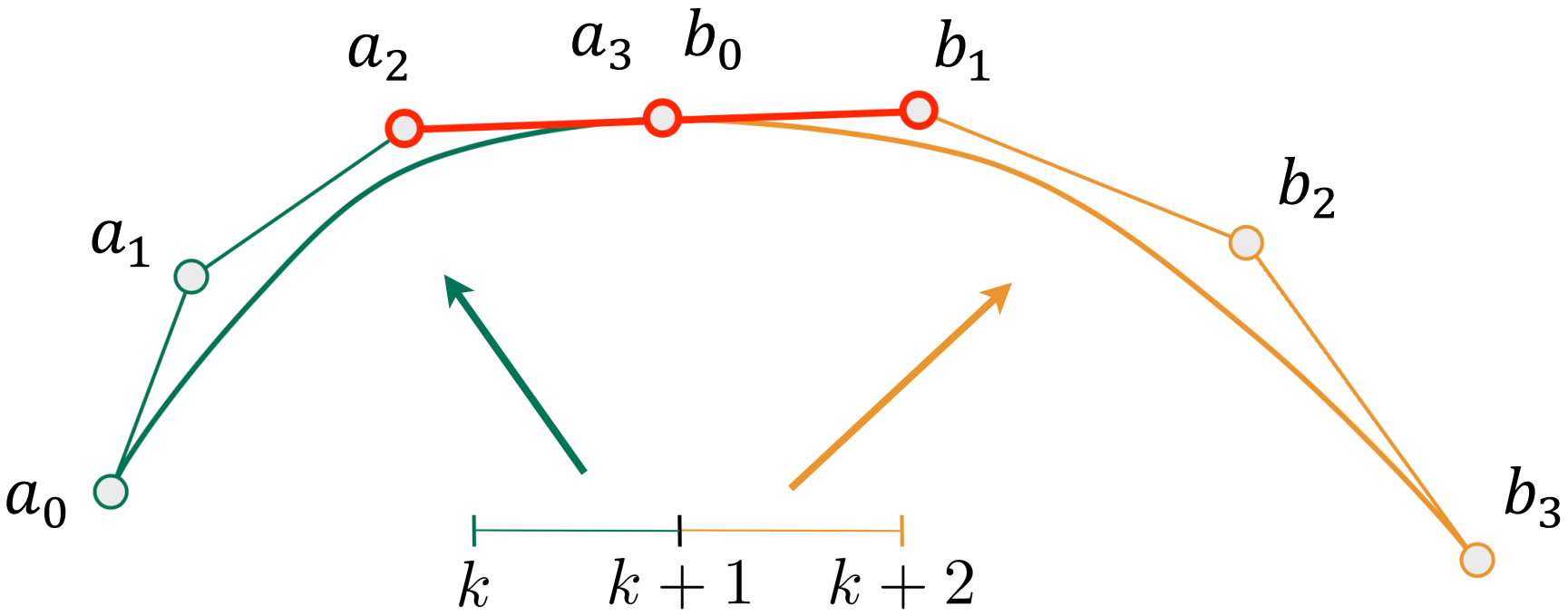
- C^0 连续性 (仅要求首位相连)



贝塞尔曲线 – 连续性 (Continuity)

- C^1 连续性 (仅要求首位相连)

$a_3 = b_0$ 且线段 (a_2, a_3) 和 (b_0, b_1) 在一条直线上

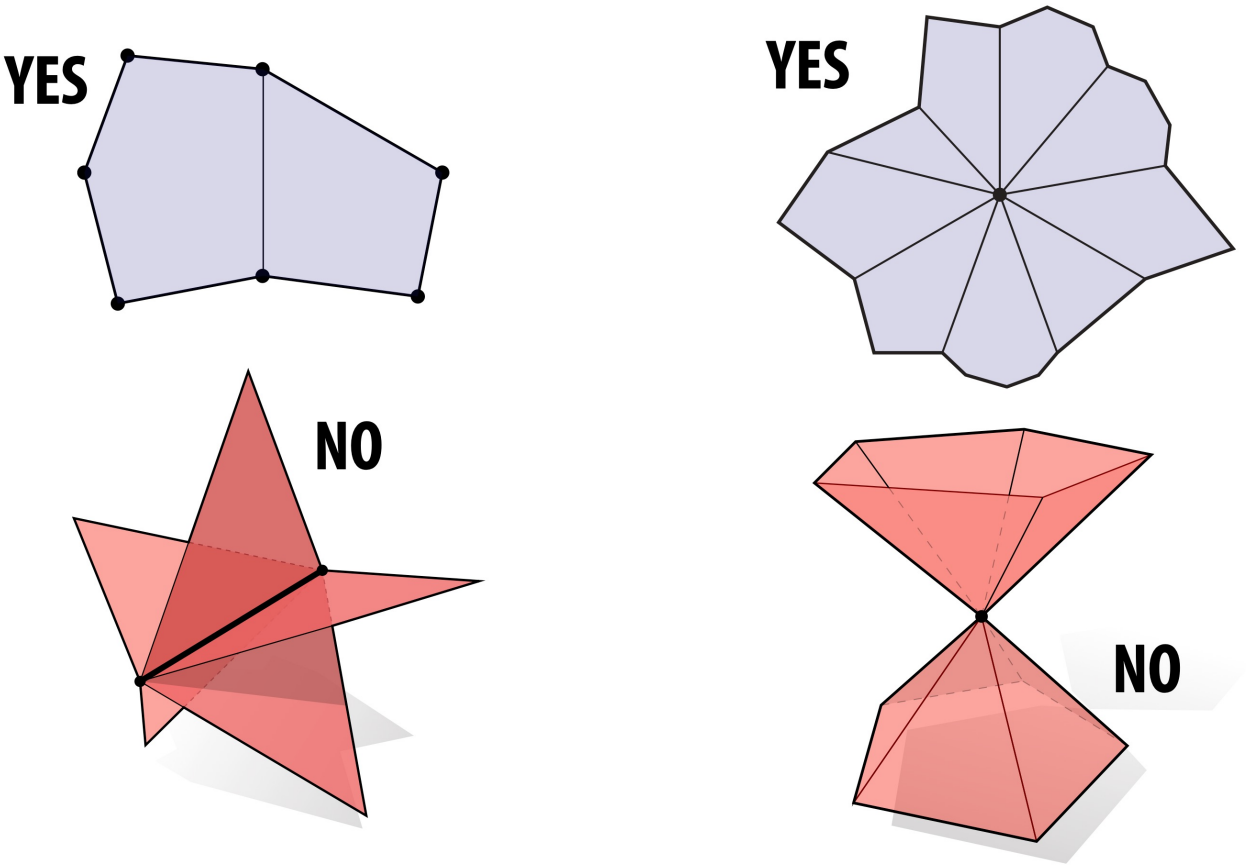


Lecture11：网格和流形
Mesh and manifold

流形多边形网格 (manifold polygon mesh)

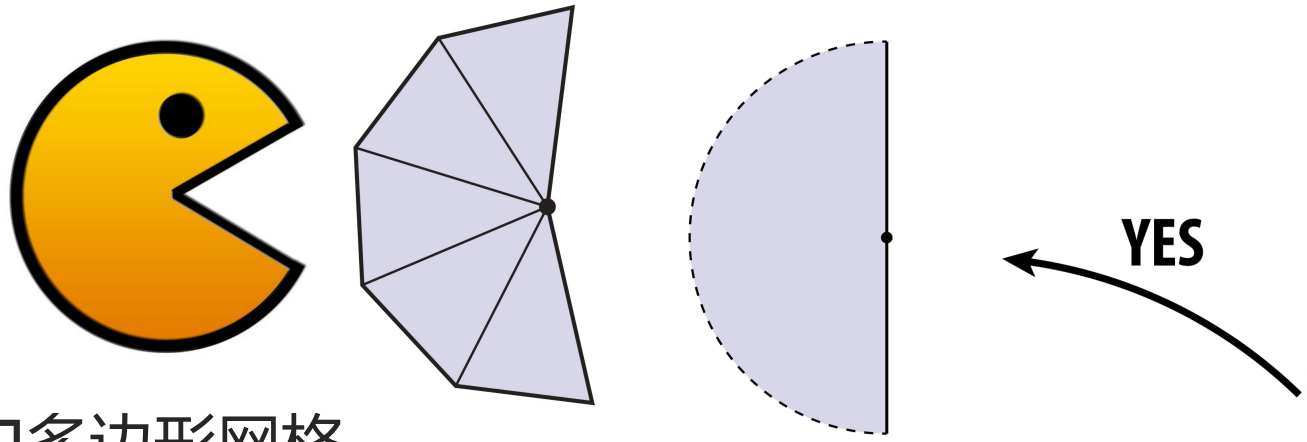
口判断多边形曲面是否为流形，只需检查两个简单的条件：

1. **每条“内部“边只包含在两个多边形中**
2. **每个“内部”点周围包含一圈循环的多边形**



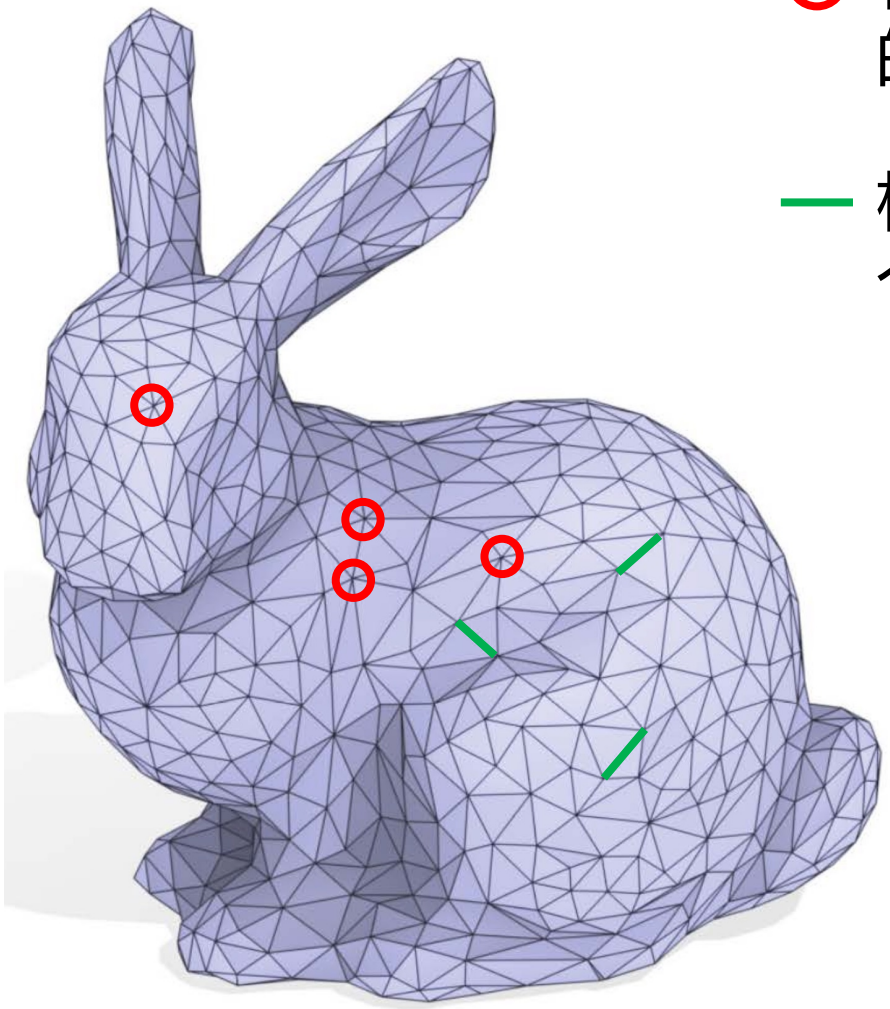
形状的边界 (boundary) 呢?

- 边界是曲面的“终点”
- 例如，裤子上的腰部和脚踝
- 在局部，边界看起来像半个磁盘
- 整体上看，每个边界都形成一个循环

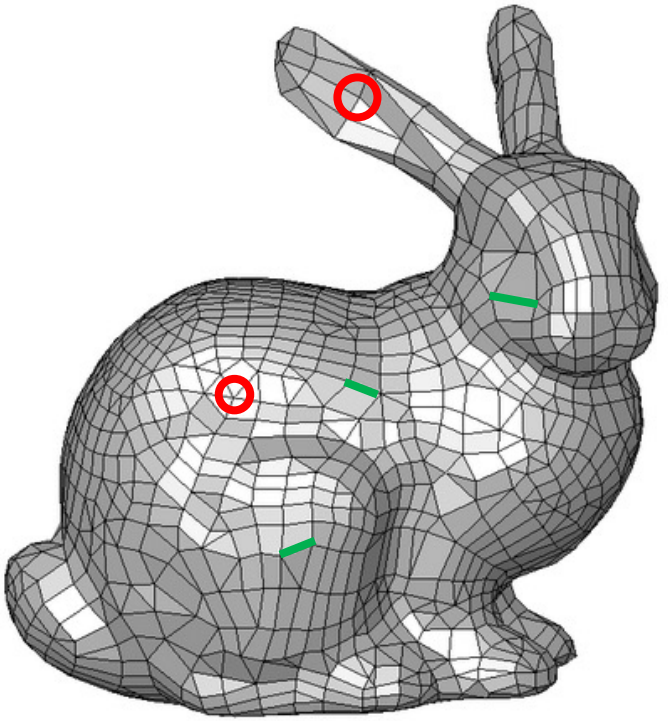


- 多边形网格
  - **每个“边界”边仅包含在一个多边形中**
  - **每个“边界”点周围包含成一组 (而不是一圈) 的多边形**

流形多边形网格



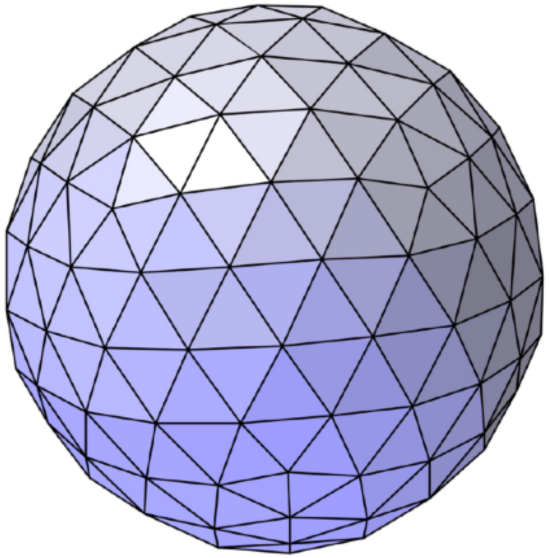
- 标识“内部”点，被与其相连的多边形环绕一圈
- 标识“内部”边，仅包含在两个多边形中



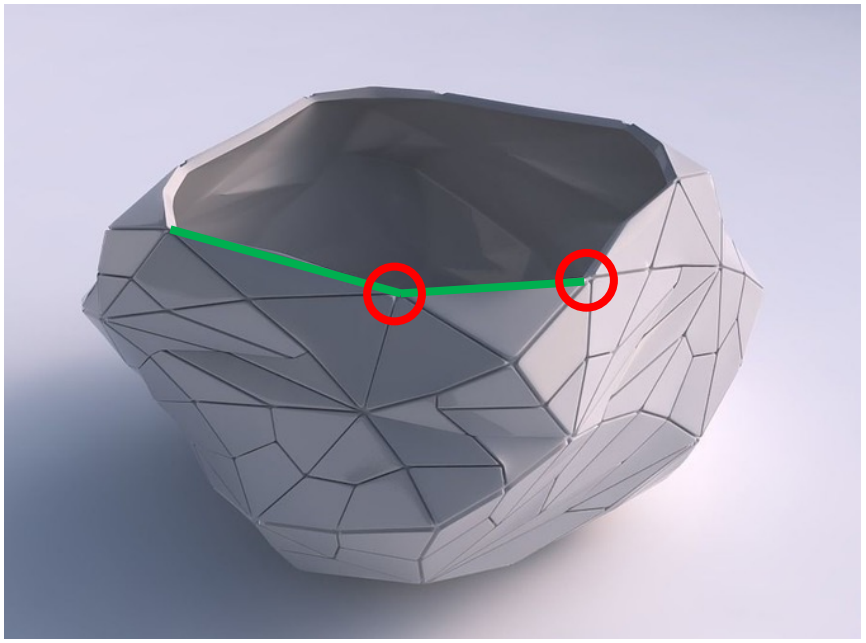
其他多边形网格也一样

流形多边形网格

- 并不是每个多边形网格都有边界



无边界

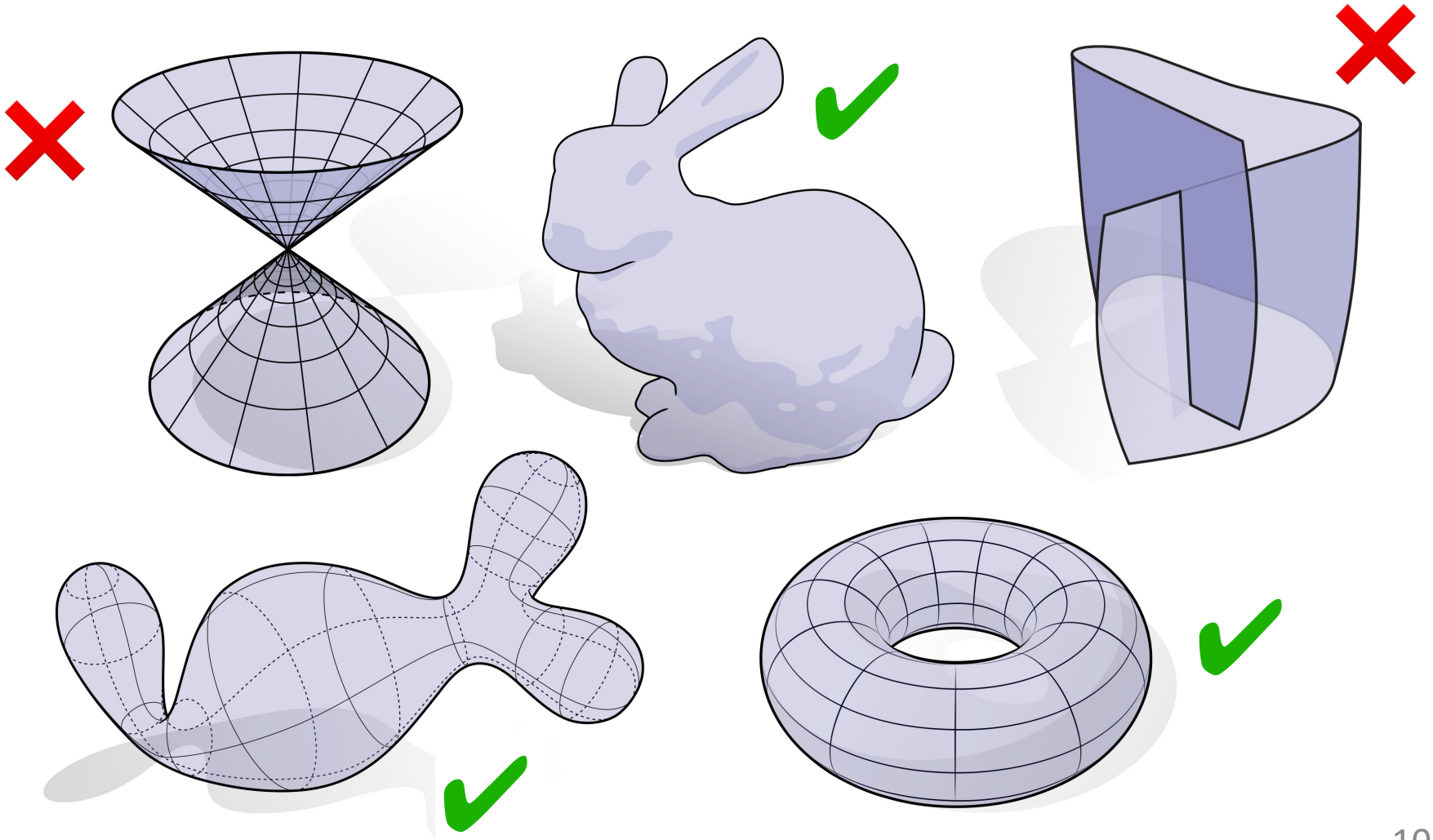


有边界

- 标识“边界”点，被与其相连的多边形环绕，但未构成一圈
- 标识“边界”边，仅包含在一个多边形中

流形与非流形 (manifold vs. nonmanifold)

- 下列哪些几何图形是流形的



邻接表 Adjacency list (Array-like)

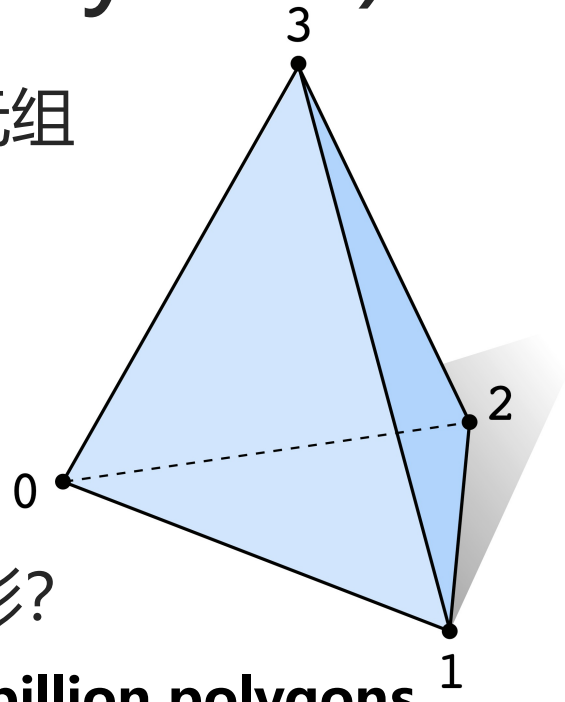
- 存储坐标的三元组 (x, y, z) 以及索引的三元组
- 比如四面体：

VERTICES

	x	y	z
0:	-1	-1	-1
1:	1	-1	1
2:	1	1	-1
3:	-1	1	1

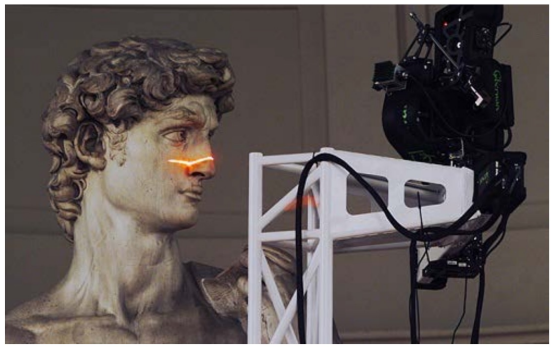
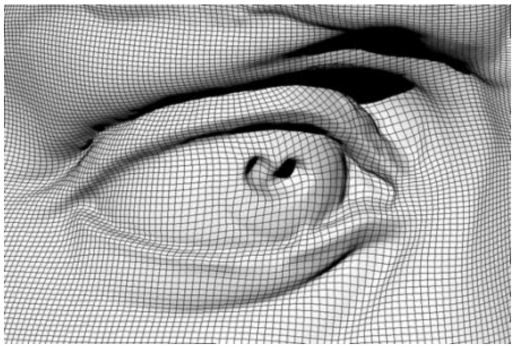
POLYGONS

	i	j	k
	0	2	1
	0	3	2
	3	0	1
	3	1	2



- Q：我们如何找到所有接触顶点 2 的多边形？
- 现在考虑一个更复杂的网格：

~1 billion polygons

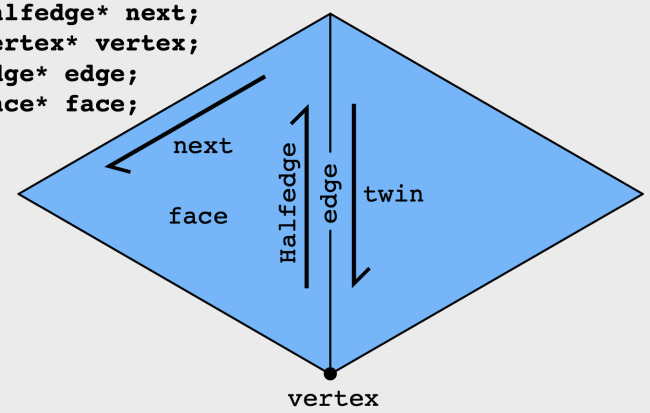




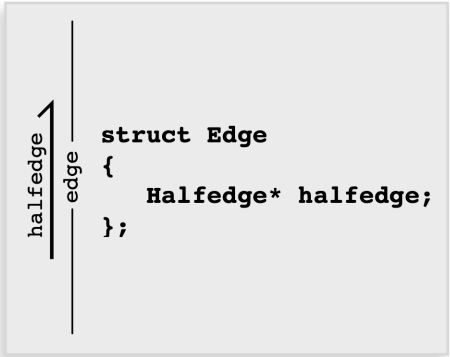
- 查找相邻多边形非常昂贵！

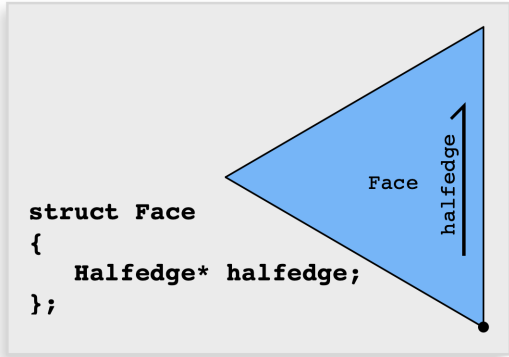
半边 Halfedge 数据结构 (类似链表)

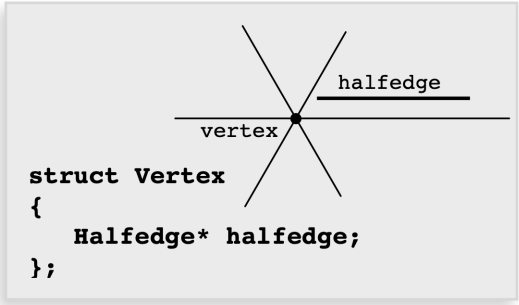
- 基本理念：存储一些邻居的信息
- 不需要详尽的清单；只需要几个关键指针 pointer (**就能让我们存储/计算所有网格信息**)
- 关键思想：两个半边缘充当网格元素之间的“粘合剂”

```
struct Halfedge
{
   Halfedge* twin;
   Halfedge* next;
   Vertex* vertex;
   Edge* edge;
   Face* face;
};
```







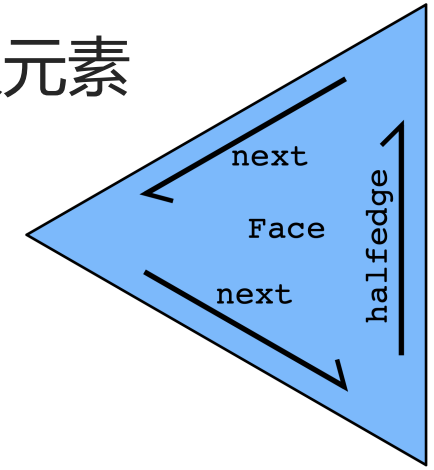


每个 halfedge 存储很多信息

- 每个顶点、边和面仅指向它的其中一个半边

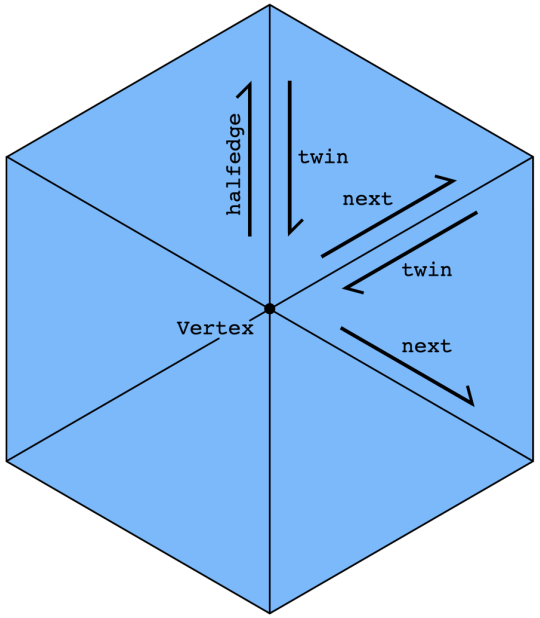
半边简化了网格遍历 mesh travesal

- 使用 twin 和 next 指针在网格中移动
- 使用 vertex、edge 和 face 指针来抓取元素
- **示例1：访问一个面的所有顶点**




```
Halfedge* h = f->halfedge;
do {
   h = h->next;
   // do something w/ h->vertex
}
while( h != f->halfedge );
```

- **示例2：访问一个顶点接触的所有邻居**




```
Halfedge* h = v->halfedge;
do {
   h = h->twin->next;
}
while( h != v->halfedge );
```

- **注意：只有当网格是流形时才有意义！**

半边连通性总是流形的

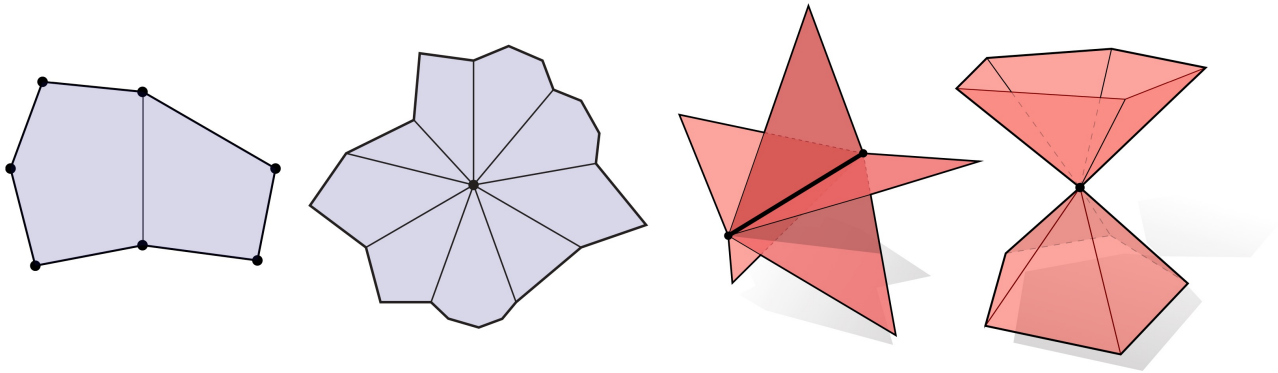
- 考虑一个简化的半边缘数据结构
- 仅需要以下条件，就能得到一个流形多边形网格

指向自己的指针

```
struct Halfedge {
   Halfedge *next, *twin;
};
```

```
twin->twin == this
twin != this
every he is someone's "next"
```

- **Keep following `next`, and you'll get faces**
- **Keep following `twin` and you'll get edges**
- **Keep following `next->twin` and you'll get vertices**

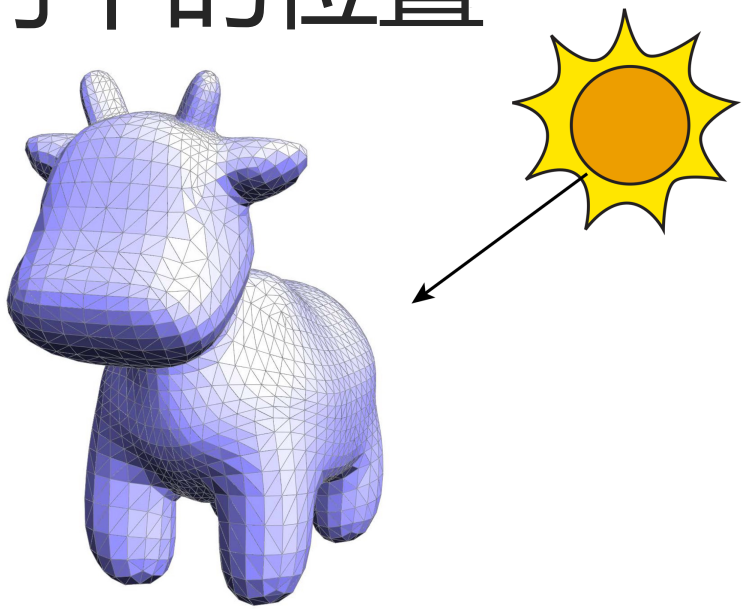


- 因此，从半边的构造来看，不能编码红色的非流形几何图形

Lecture13：光线-几何交互
Ray-geometry interaction

查找光线照射在几何中的位置

- (简单) 光线模型
  - 从一个光源发出的射线
  - 光线沿着既定方向与几何物体发生交叉
- 关注光线打在物体哪里



- 此类问题在 CG 中称为几何查询 (geometry queries)
  - 最近点查询 (Nearest point queries)
  - 距离测量 (Distance measures)
  - 交叉测试 (Intersection tests)
  - 包含性测试 (Containment tests)
  - 碰撞检测 (Collision detection)
  - …

光线方程 Ray equation

口光线可用如下方程表示

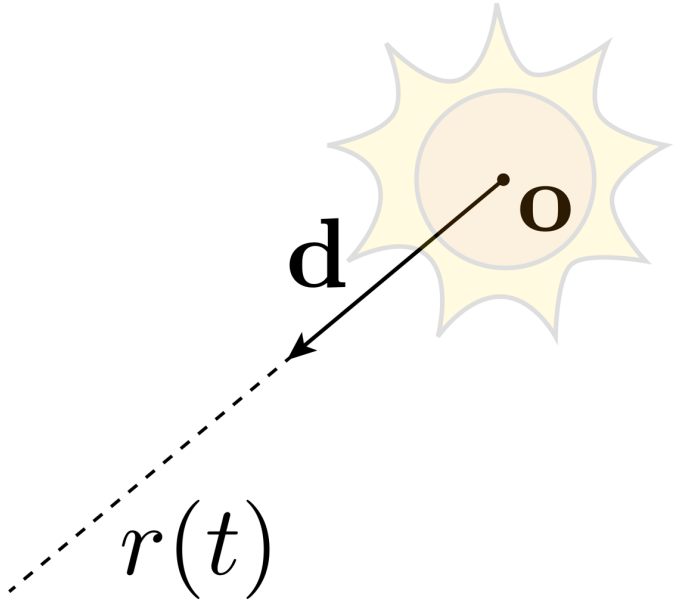
光线源点
origin

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

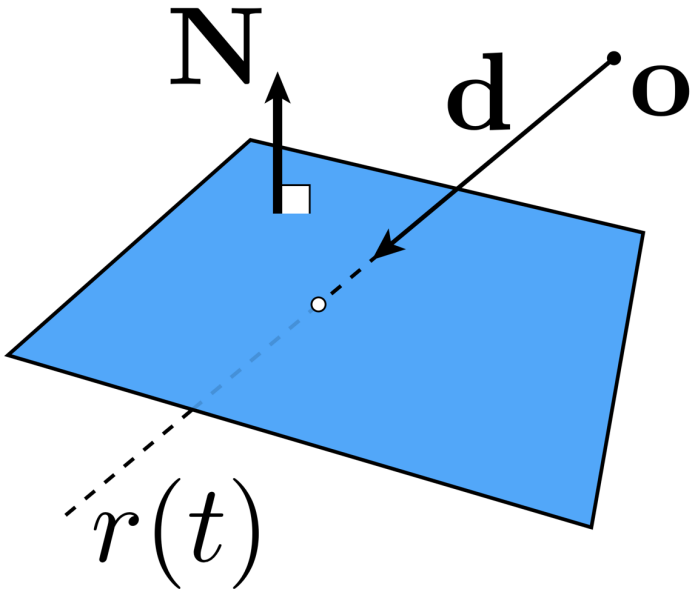
沿着光线在 t 时间上的点
point along ray at t

时间
time

单位方向
unit direction



光线与平面相交



- 假设我们有平面 $\mathbf{N}^{\mathrm{T}}\mathbf{x} = \mathrm{c}$
  - $\mathbf{N}$ 为单位法线
  - c 为偏移量 (offset)
- 我们如何找到平面与光线的交点?
- 算法：同样地，把点 x 替换为光线方程

$$\mathbf{N}^T\mathbf{r}(t) = c$$

- 求解 t:

$$\mathbf{N}^T(\mathbf{o} + t\mathbf{d}) = c \quad \Longrightarrow t = \frac{c - \mathbf{N}^T\mathbf{o}}{\mathbf{N}^T\mathbf{d}}$$

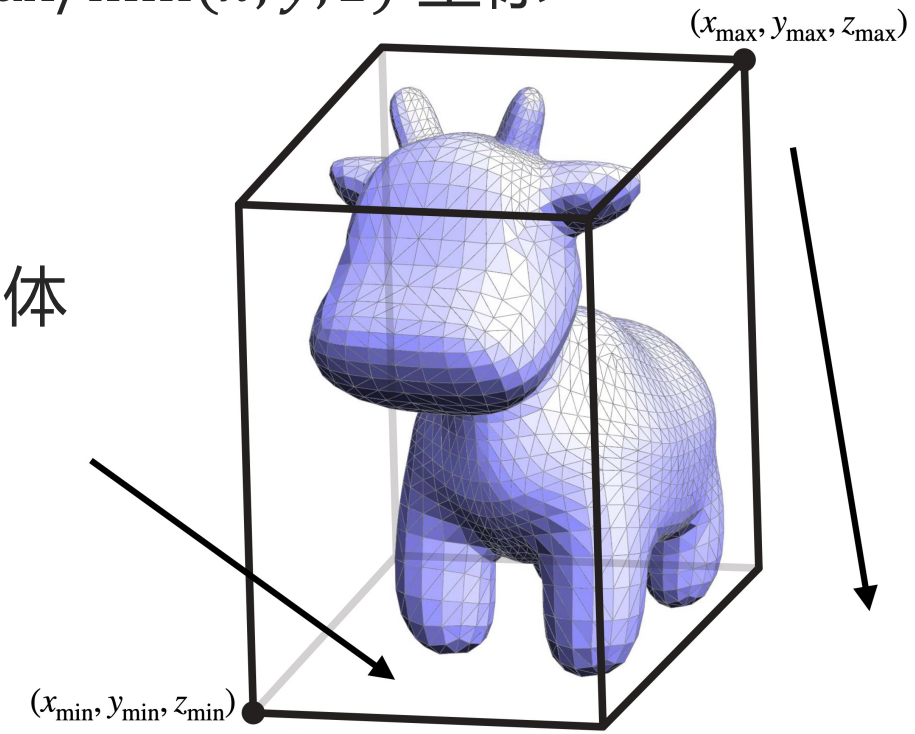
- 将 t 代入光线方程中 $\mathbf{r}(t) = \mathbf{o} + \frac{c - \mathbf{N}^T\mathbf{o}}{\mathbf{N}^T\mathbf{d}}\mathbf{d}$

Lecture14：几何交互加速
Ray-geometry interaction acceleration

包围盒

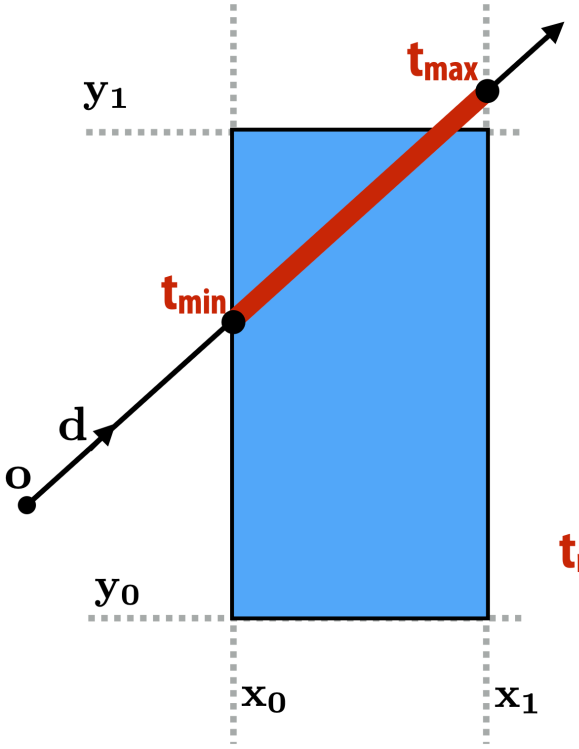
- 一种快速 (初步) 判断交叉的方法：用最小的立方体包围复杂的几何物体 (包含其所有图像基元)
  - Q：怎么构建包围盒?
  - A：循环所有的点，记录 $\max/\min(x, y, z)$ 坐标
  - Complexity: $O(N)$
- 检查光线是否与包围盒相交
  - 若没有，则一定没有击中物体
  - 若有，检查所有三角形找到相交的点 (也可能都不相交)

避免每一条光线都要检查所有图像基元!

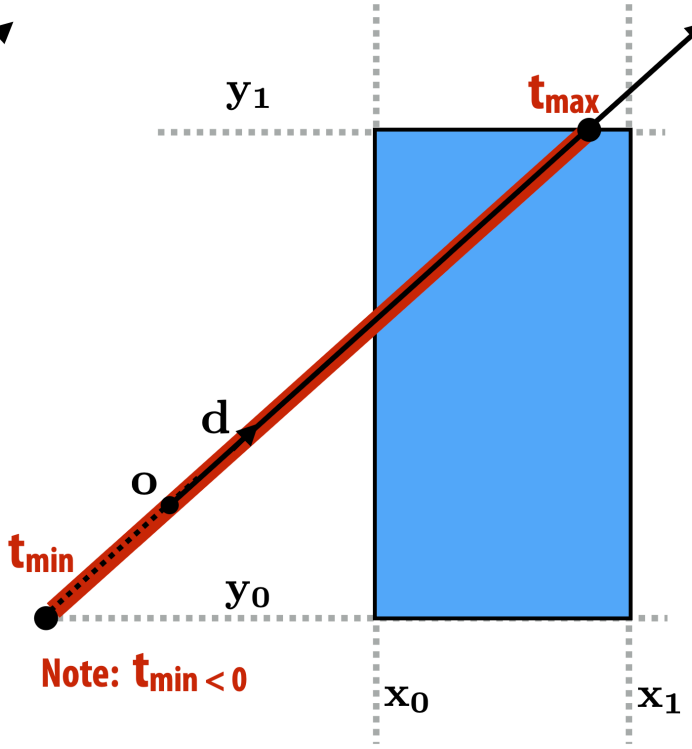


光线与轴对齐包围盒相交

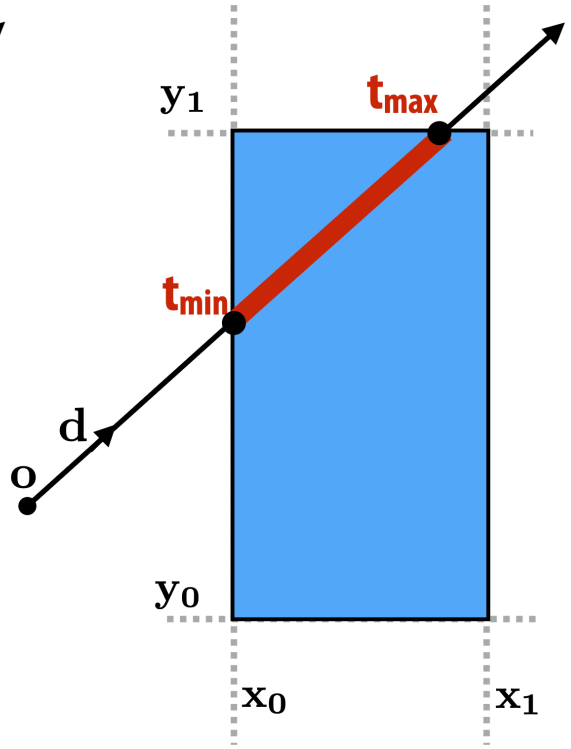
- 怎么判断光线与盒子相交?
- 2D 的例子 (3D 的情况也一样)
- 计算与所有平面对的交点，并取 t_{min}/t_{max} 的交集



Intersections with x planes

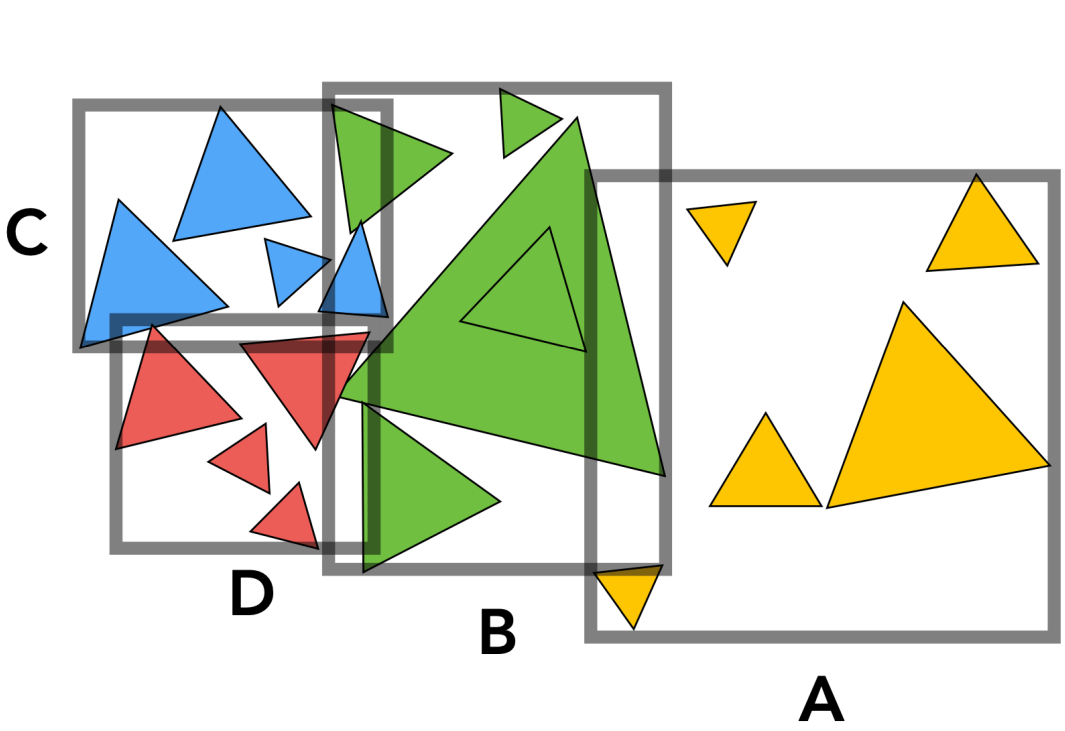


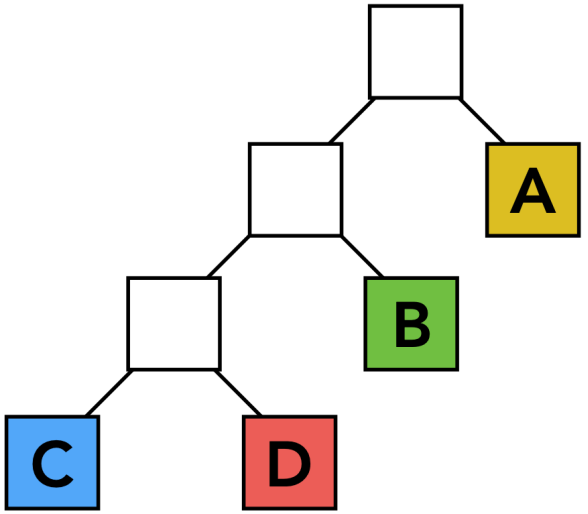
Intersections with y planes



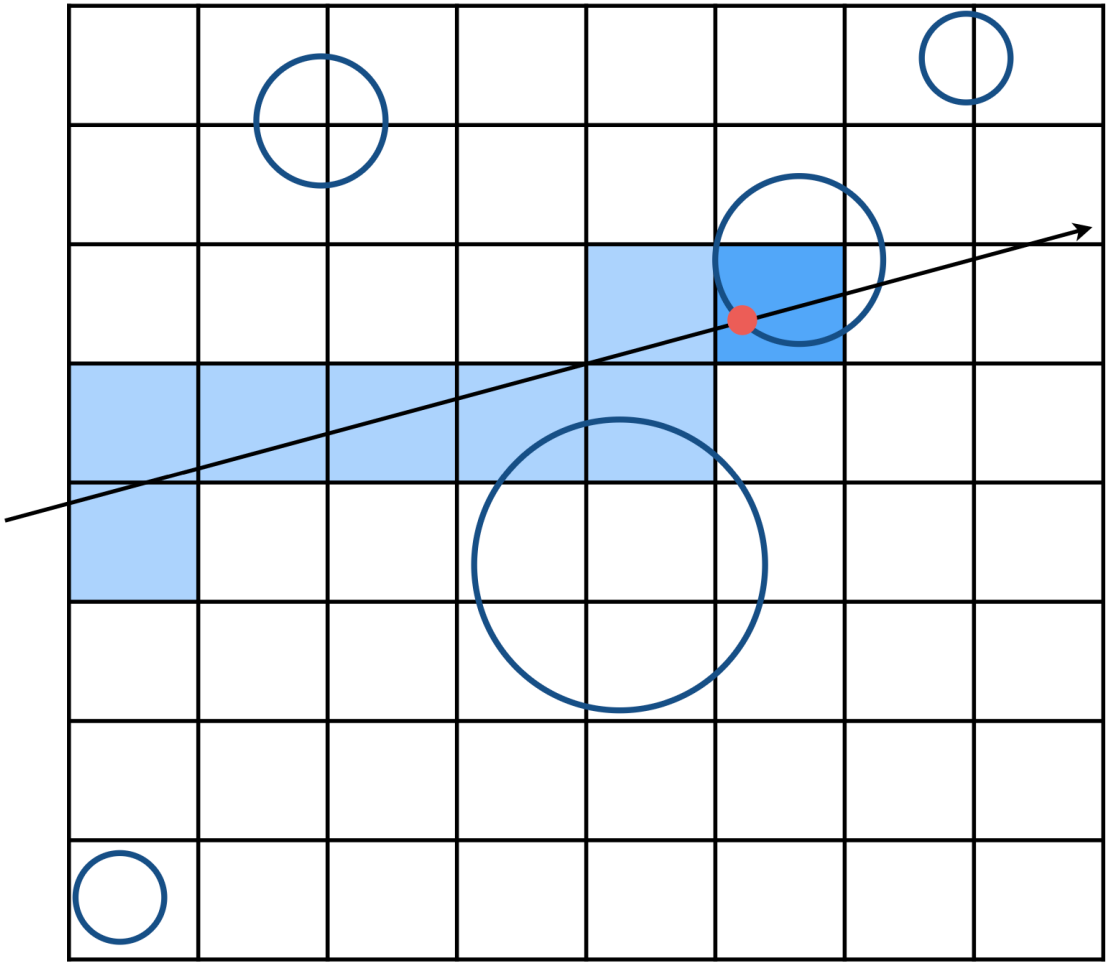
Final intersection result

层次包围盒





光线网格相交

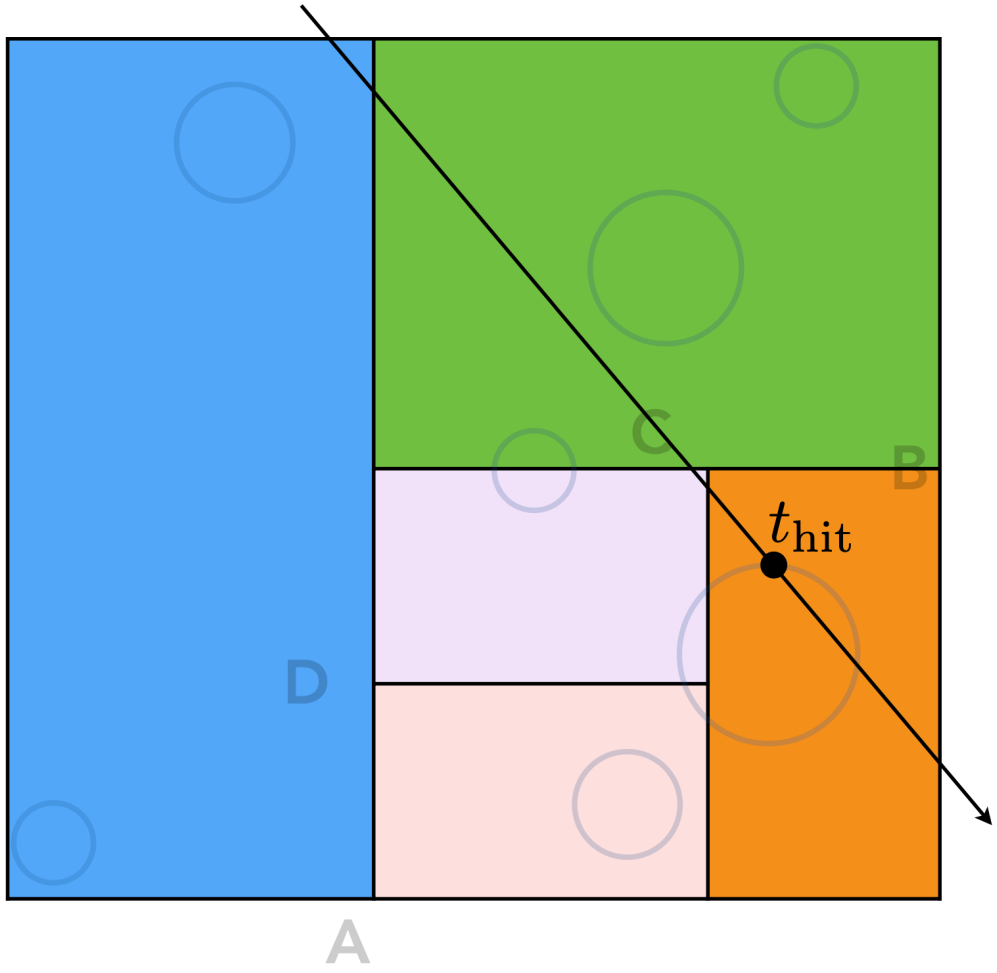


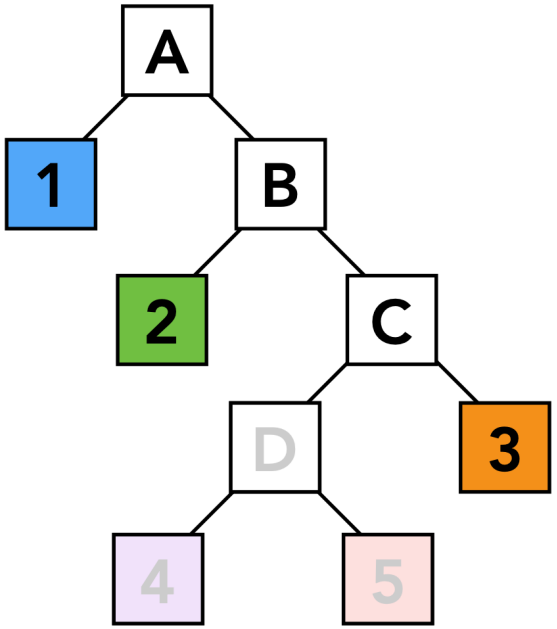
按光线的走向依次穿过网格内的格子

对于每一个格子

测试是否与格子中保存的图像基元相交

KD 树划分





找到与光线相交的图像基元！

不同加速结构的比较

结构	优点	缺点
均匀空间划分	1. 结构简单，易于实现 2. 访问速度快 3. 适合处理分布均匀的物体	1. 分布不均匀时，会有空格子，效率低下 2. 大对象需要跨越多个格子，增加计算复杂性
KD 树	1. 适合处理分布不均的场景 2. 查询效率高，特别是高维数据	1. 构建 KD 树较耗时 2. 动态场景效果不好，需要重新构建 KD 树
BVH 树	1. 适合处理任意分布的物体 2. 动态场景效果较好，更新成本低 3. BVH 与 GPU 并行架构匹配度高	1. 构建 BVH 树较耗时 2. 对于特定查询，如找最近物体，效率可能不如 KD 树

Lecture15：着色和阴影
Shading and shadow

光线感知

高光 (Specular highlights)
大部分光线沿着特定方向散射出去，形成高亮的点

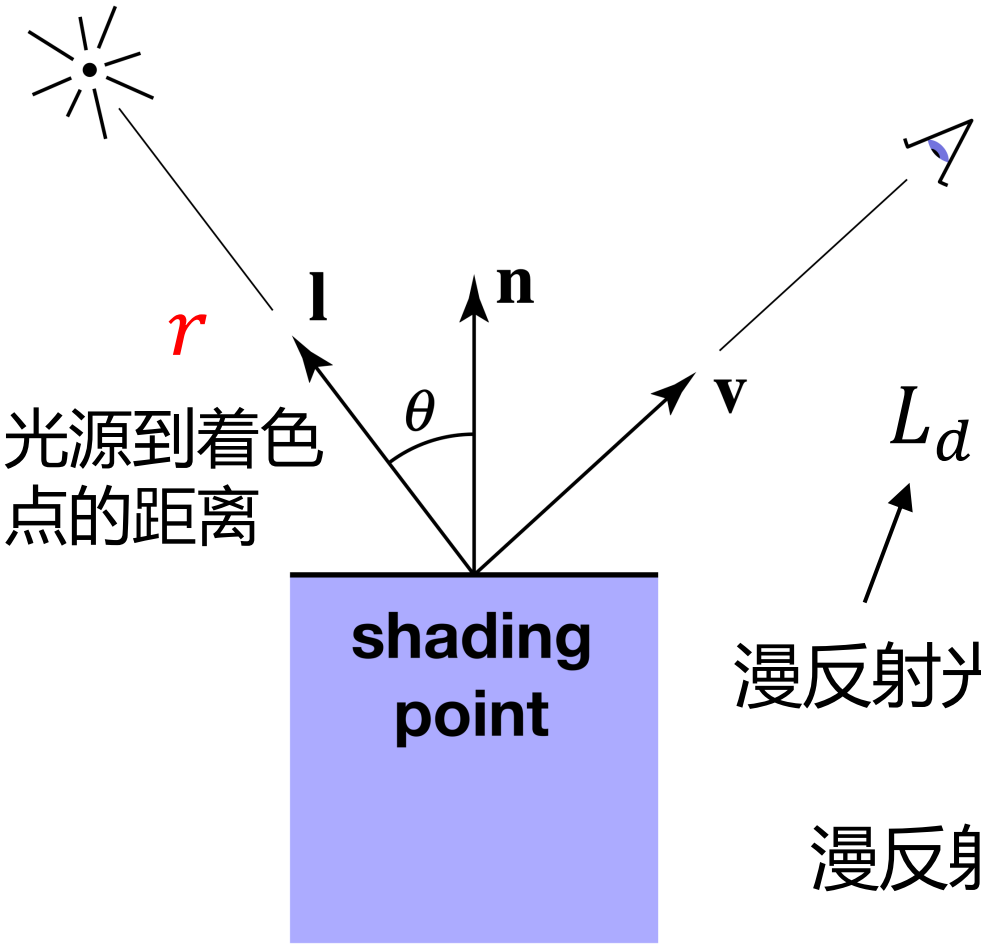
漫反射光 (Diffuse reflection)
光线向各个方向均匀地散射出去

环境光 (Ambient lighting)
光线从所有方向均匀地照射在物体上，可简化为固定光照

BP 反射模型同时考虑了这三种类型的光以生成真实的效果

漫反射着色

- 着色与观察方向无关



到达着色点的能量

$$L_d = k_d \left(\frac{I}{r^2}\right) \max(0, \mathbf{n} \cdot \mathbf{l})$$

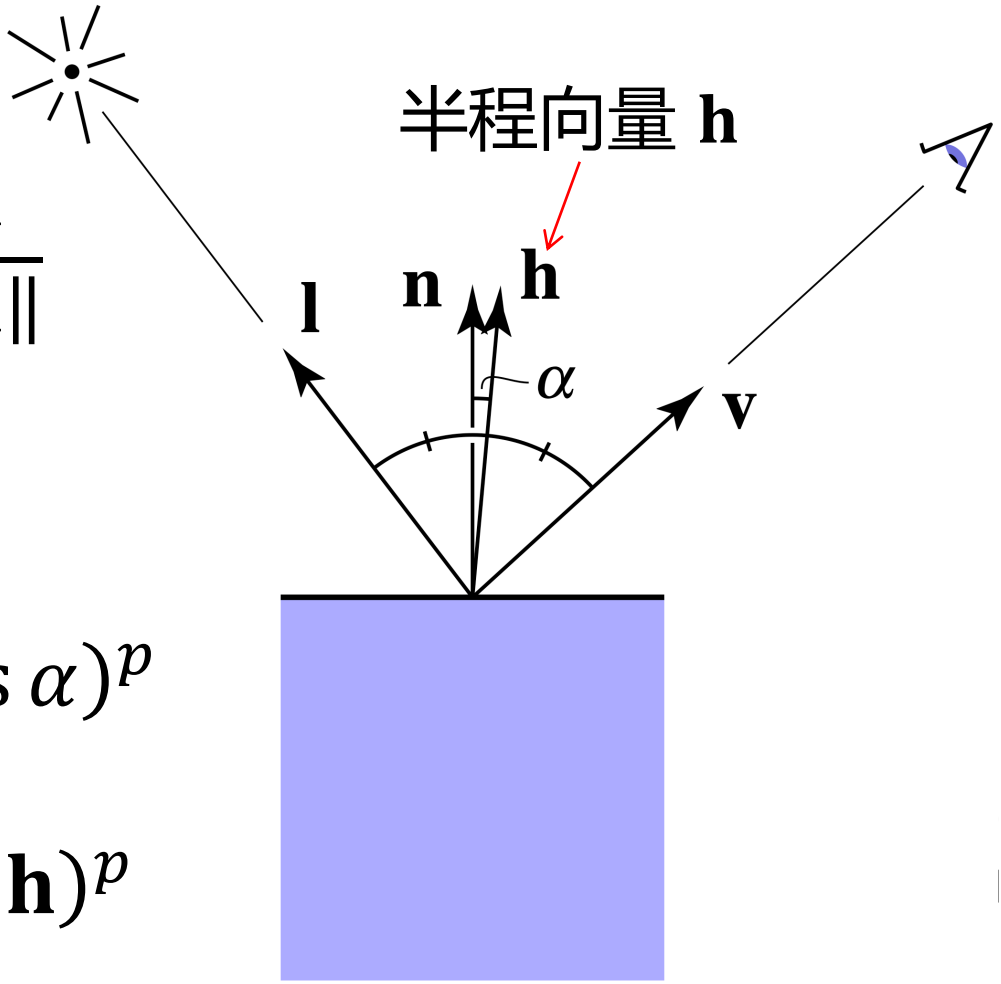
漫反射光

着色点接收的能量

漫反射系数 (决定亮度和颜色)

高光着色

- Blinn-Phong 模型对高光的处理
  - 观察角度与镜面反射角度接近 ⇔ 半程向量与法向量接近
- 用点积衡量向量接近程度



角平分线方向

$$\mathbf{h} = bisector(\mathbf{v},\mathbf{l}) = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$$

归一化

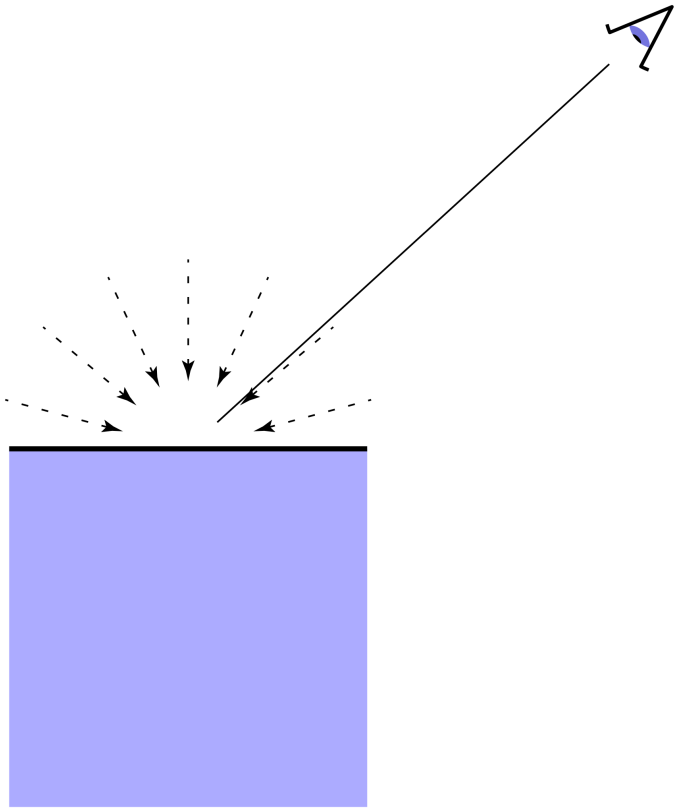
镜面反射光　　镜面反射光系数

$$L_s = k_s\left(\frac{I}{r^2}\right)\max(0,\cos\alpha)^p$$

$$= k_s\left(\frac{I}{r^2}\right)\max(0,\mathbf{n}\cdot\mathbf{h})^p$$

环境光着色

- 捕捉精确的环境光非常复杂，因此 Blinn-Phong 模型假定环境光着色不依赖于任何光线
- 增加一个固定的常数 I_a 以近似环境光 (呈现一点物体原本的颜色，而不是黑色)



$$L_a = k_a I_a$$

环境光常数

环境光

环境光系数

中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn