中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

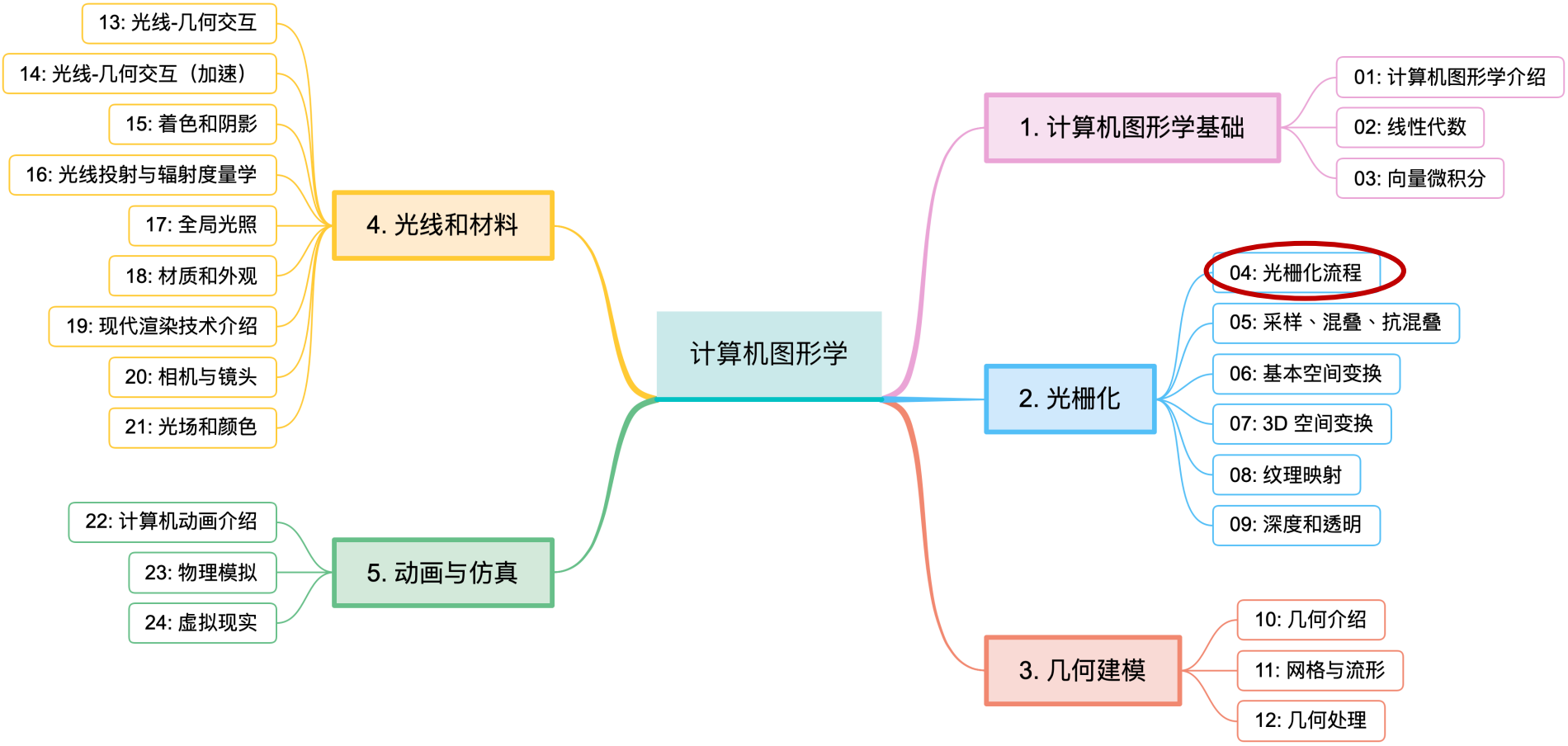
Lecture 04：光栅化

SSE315：计算机图形学
Computer Graphics

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn

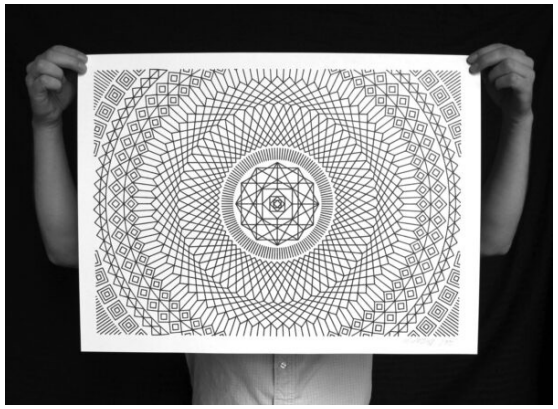

计算机图形学
1. 计算机图形学基础
01: 计算机图形学介绍
02: 线性代数
03: 向量微积分
2. 光栅化
04: 光栅化流程
05: 采样、混叠、抗混叠
06: 基本空间变换
07: 3D 空间变换
08: 纹理映射
09: 深度和透明
3. 几何建模
10: 几何介绍
11: 网格与流形
12: 几何处理
4. 光线和材料
13: 光线-几何交互
14: 光线-几何交互（加速）
15: 着色和阴影
16: 光线投射与辐射度量学
17: 全局光照
18: 材质和外观
19: 现代渲染技术介绍
20: 相机与镜头
21: 光场和颜色
5. 动画与仿真
22: 计算机动画介绍
23: 物理模拟
24: 虚拟现实

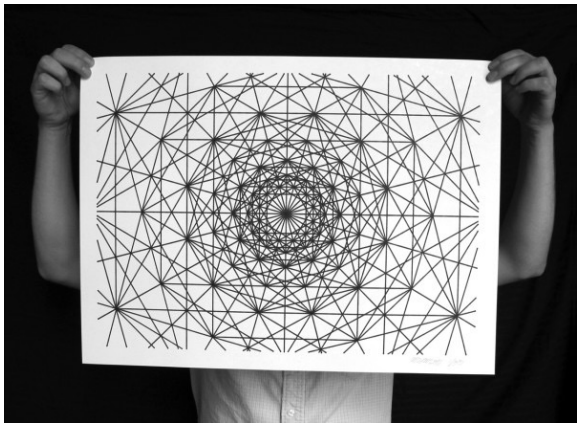
Today’s topics

- 常见的绘图机
- 光栅化流程
- 画一个三角形

绘图机
Drawing Machines

数控夏普 CNC Sharpie 绘图机



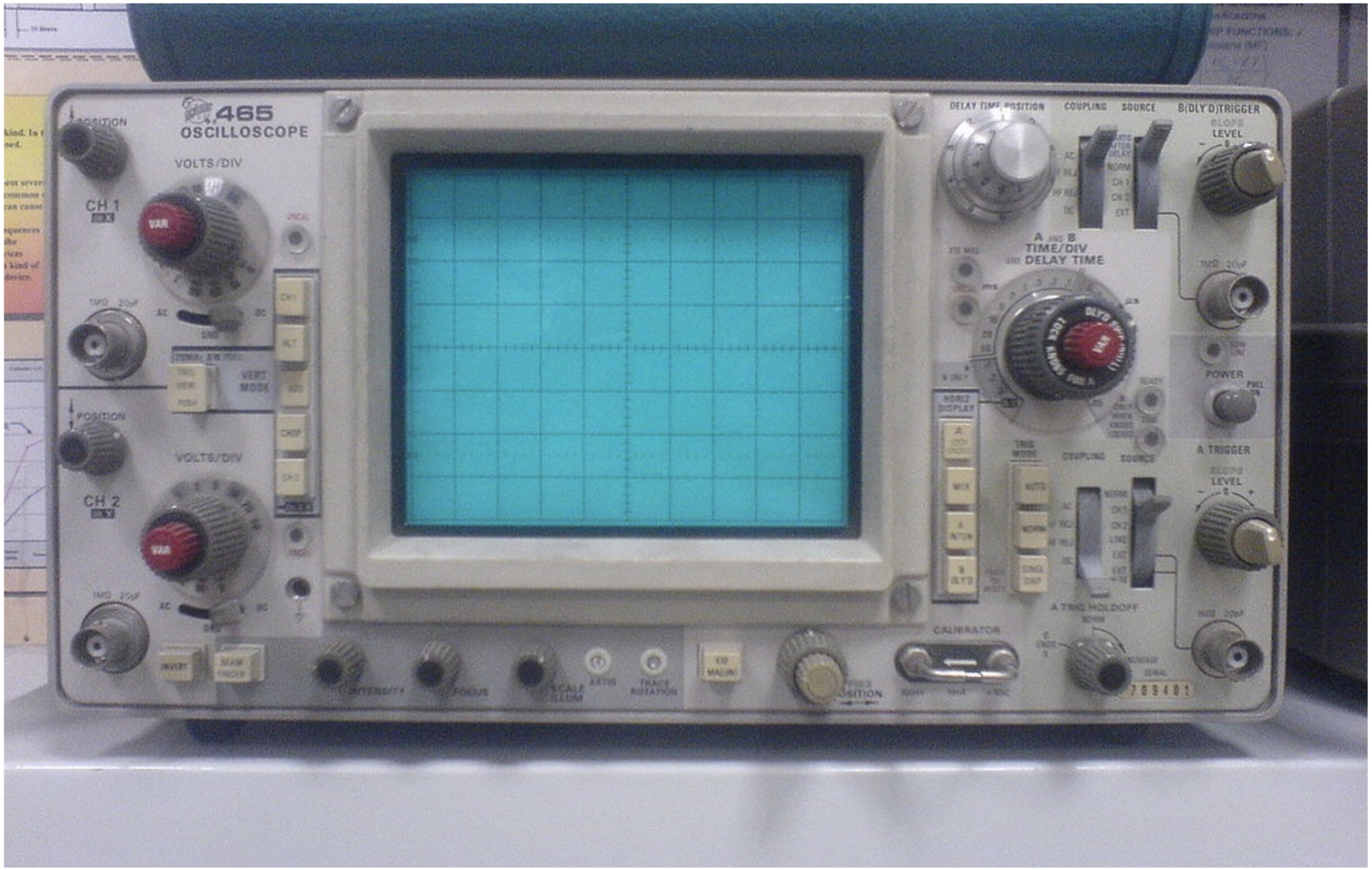


Aaron drawing machine

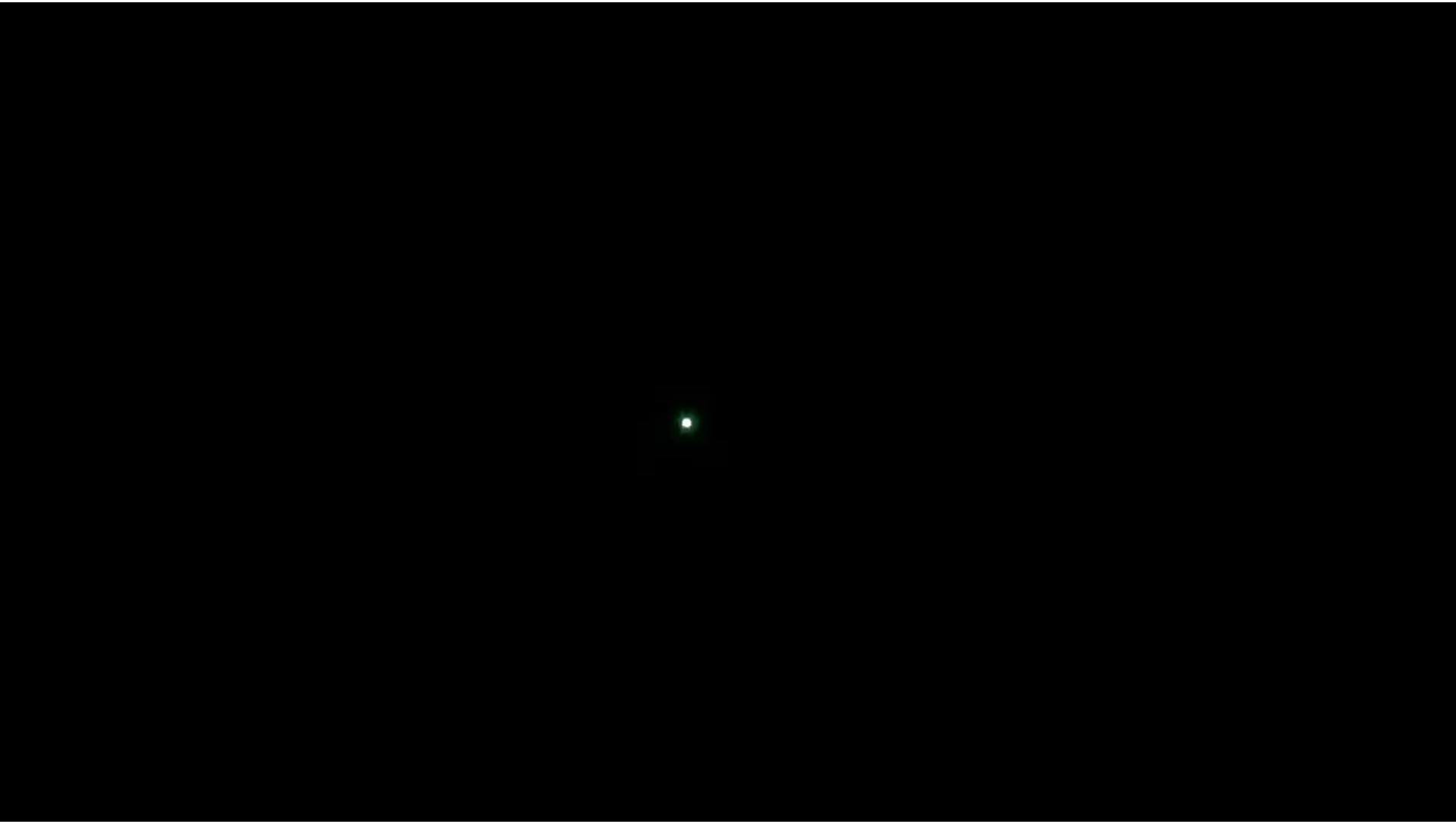
激光切割机 Laser cutters

不同的光栅显示设备
Different Raster Displays

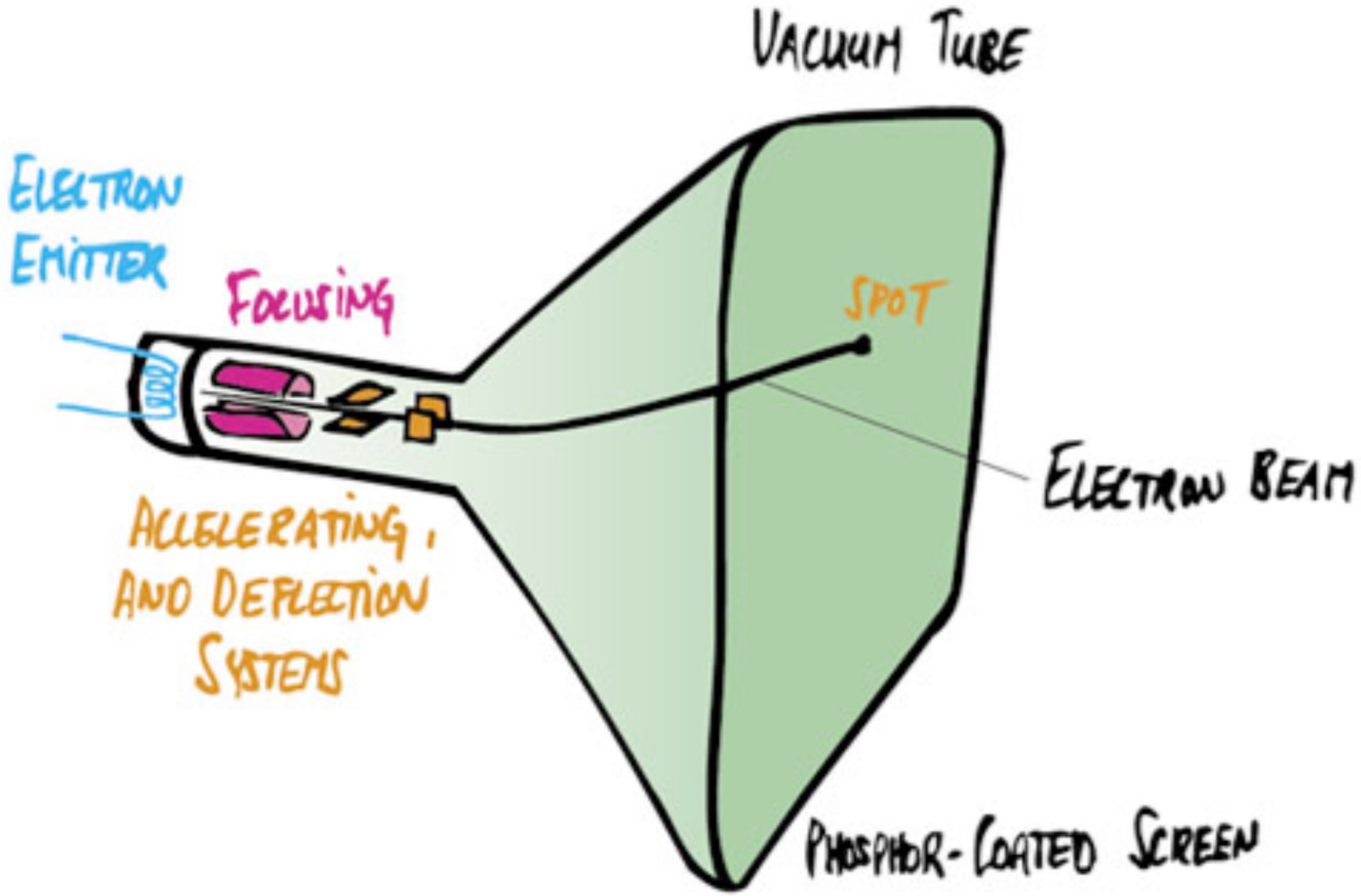
示波器 Oscilloscope



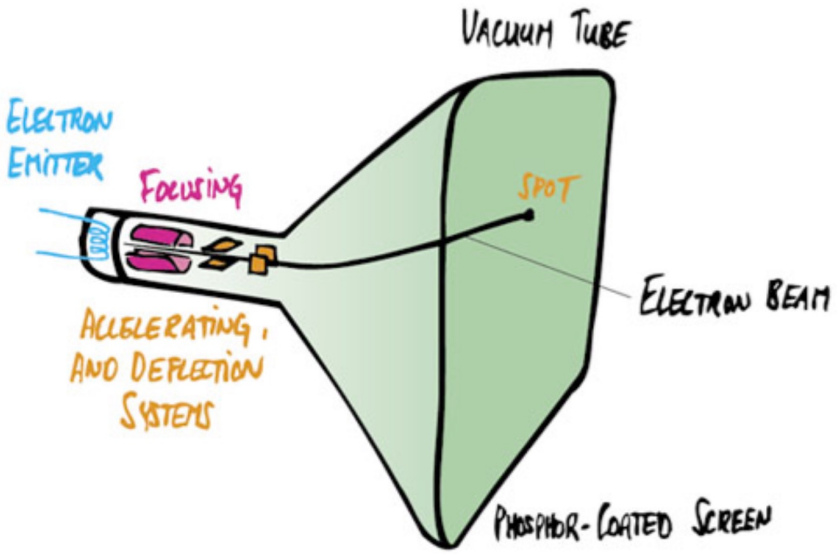
示波器艺术



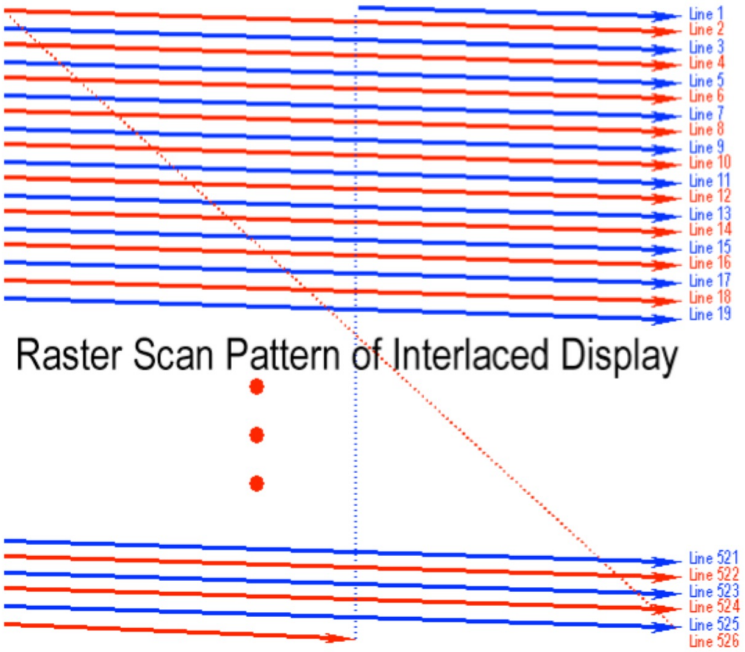
阴极射线管 Cathode Ray Tube



Television - Raster Display CRT

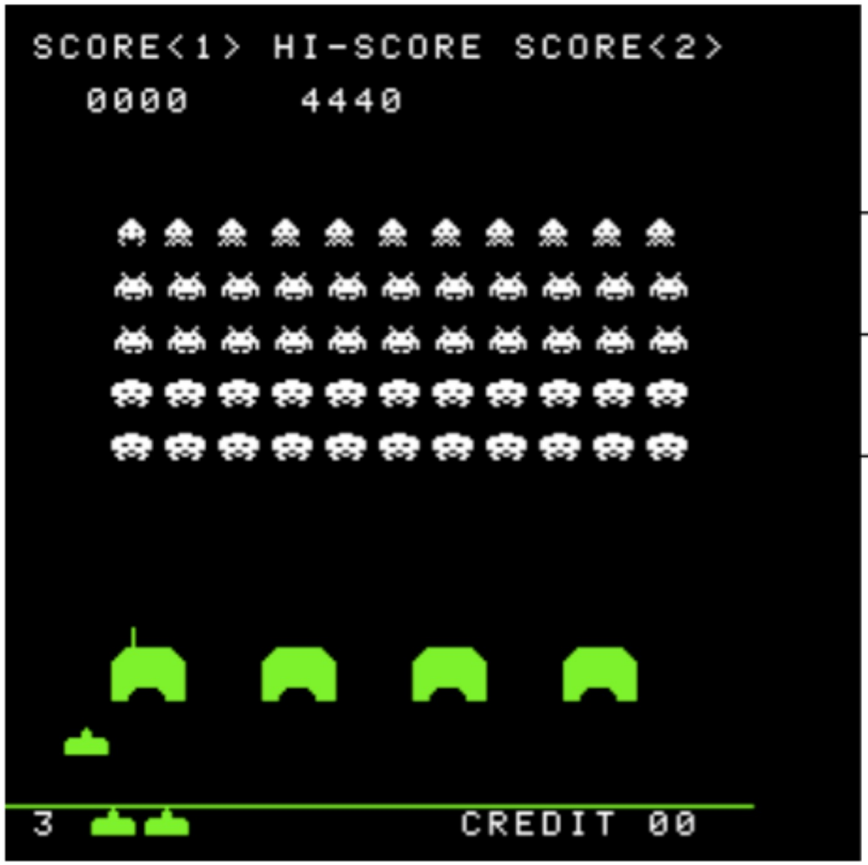


Cathode Ray Tube



Raster Scan (modulate intensity)

帧缓冲：光栅显示的内存



DAC =
Digital to Analog Convertors

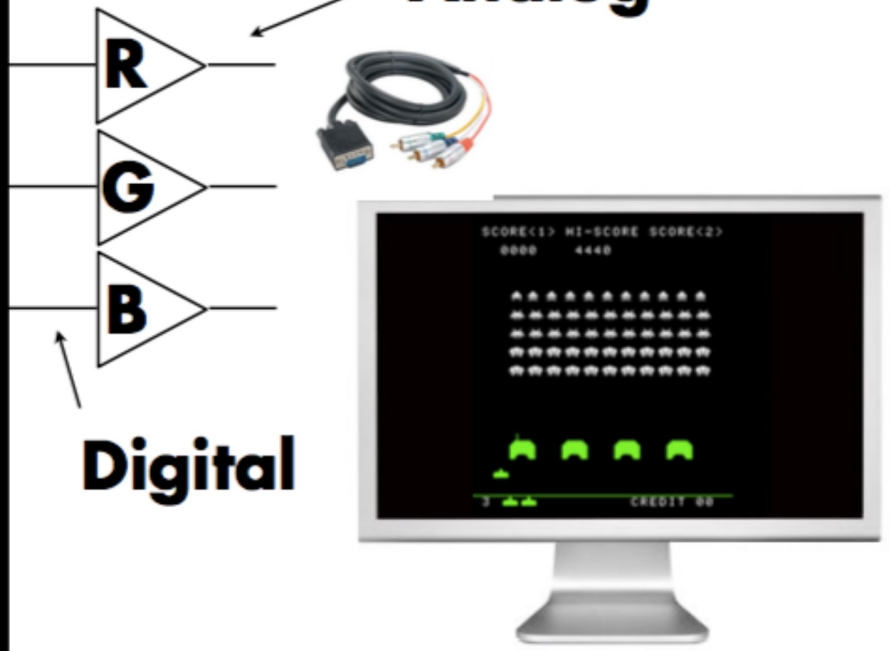


Image = 2D array of colors

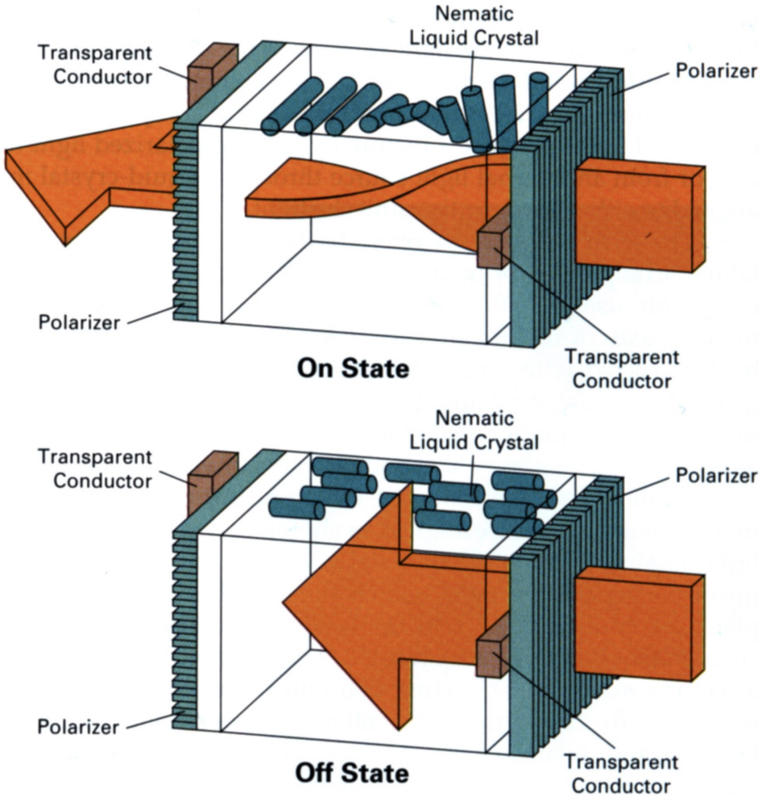
平板显示器 Flat Panel Displays

Low-Res LCD Display

Color LCD, OLED, ...

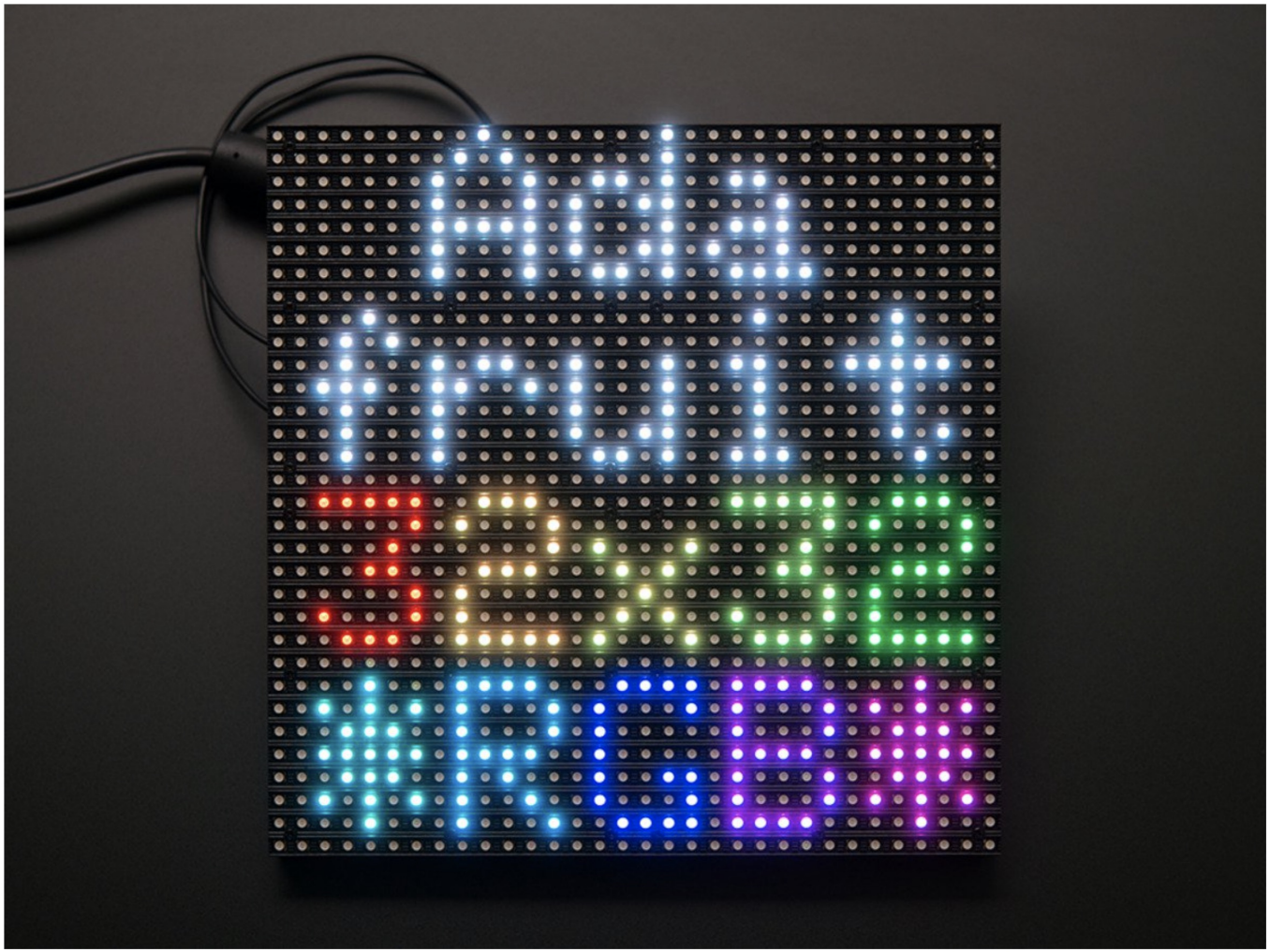
液晶显示器 Liquid Crystal Display LCD

- 背光系统：LCD需要一个背光源（通常是LED灯或冷阴极荧光灯）来发出光线
- 液晶层：这些光线穿过一个含有液晶的层面。这些液晶可以在电压的作用下改变排列，从而改变光线的传播方向
- 控制像素：每个像素都有一个电晶体管（TFT）来控制，它决定了液晶的电压，从而控制液晶的排列和光线的传播方向
- 偏振器：光线通过第二个偏振器后，光线的亮度就被改变了。如果光线的传播方向和偏振器相匹配，我们就可以看到亮的像素。如果不匹配，像素就是暗的



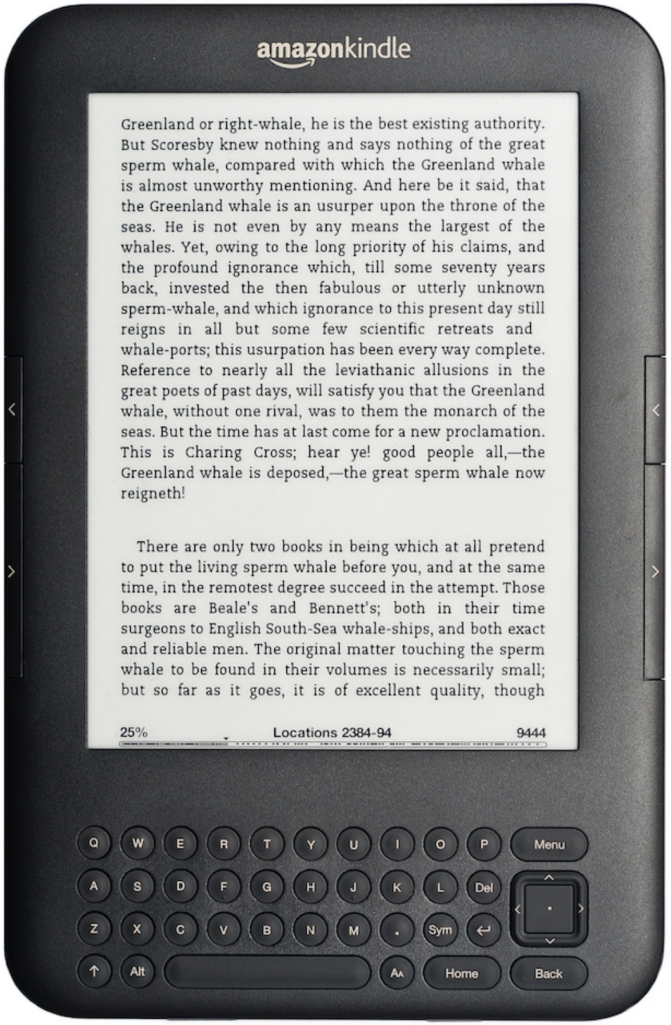
LCD 的优点是体积小、重量轻、功耗低、辐射少、视角宽

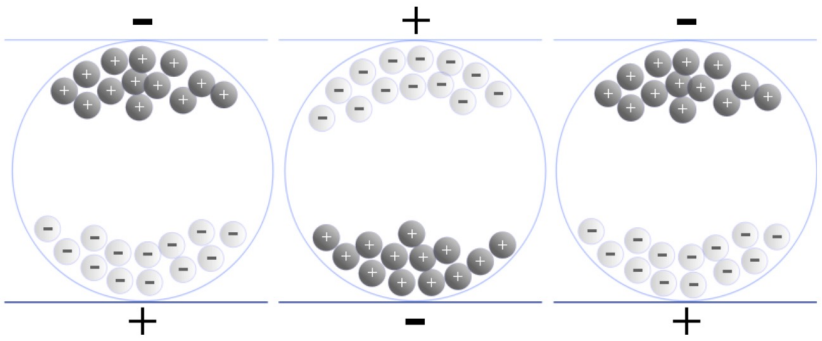
LED 阵列显示器



发光二极管 (Light emitting diode) 阵列

电泳（电子墨水）显示器



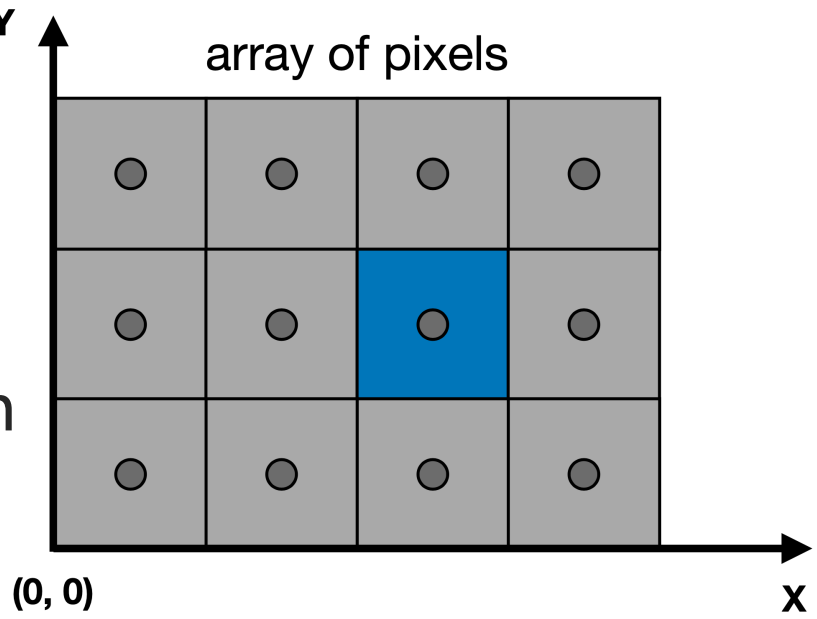


[Wikimedia Commons —Senarclens]

刷新率比较慢

光栅化
Rasterization

屏幕 Screen



- 屏幕是什么?
  - 一个（二维）像素数组
  - 数字的大小: 分辨率 Resolution
  - 一种典型的光栅显示器
- Raster == 德语中的 screen
  - Rasterize == drawing onto the screen
- Pixel (short for "picture element")
  - 颜色是 (red, green, blue) 三色的混合
  - 我们暂时把像素当作一个颜色均匀的小方块

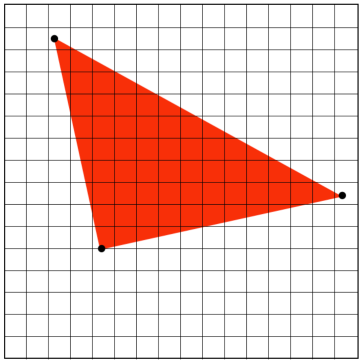
两种把图像“放到”屏幕上的技术

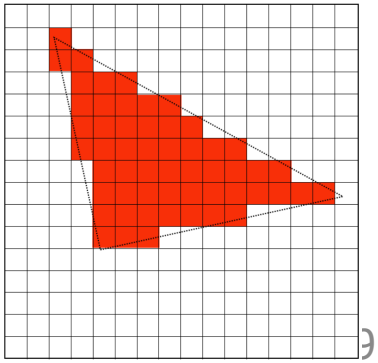
光栅化 Rasterization

- 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
- 随着课程进行我们将介绍更多

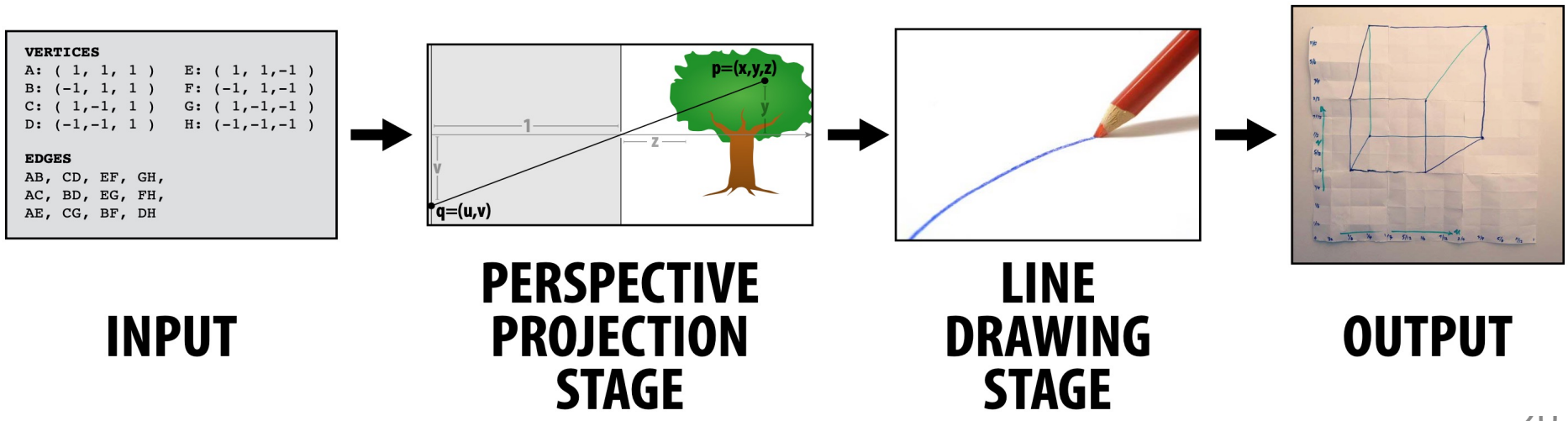




3D 图像生成的流程（Pipeline）

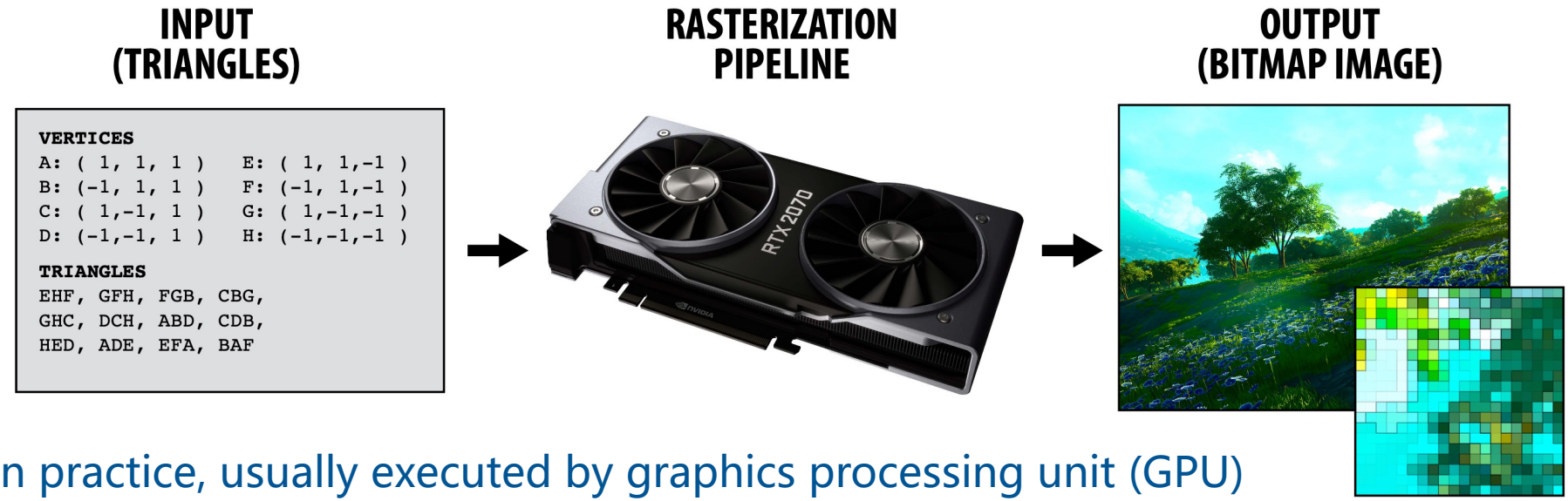
- 从一个 pipeline 的角度讨论图像生成：
  - INPUTS —— 我们想画什么图像？
  - STAGES —— 输入的转换序列 → 输出
  - OUTPUTS —— 最终图像

- 比如，我们第一节课画立方体的 pipeline



光栅化流程 Rasterization pipeline

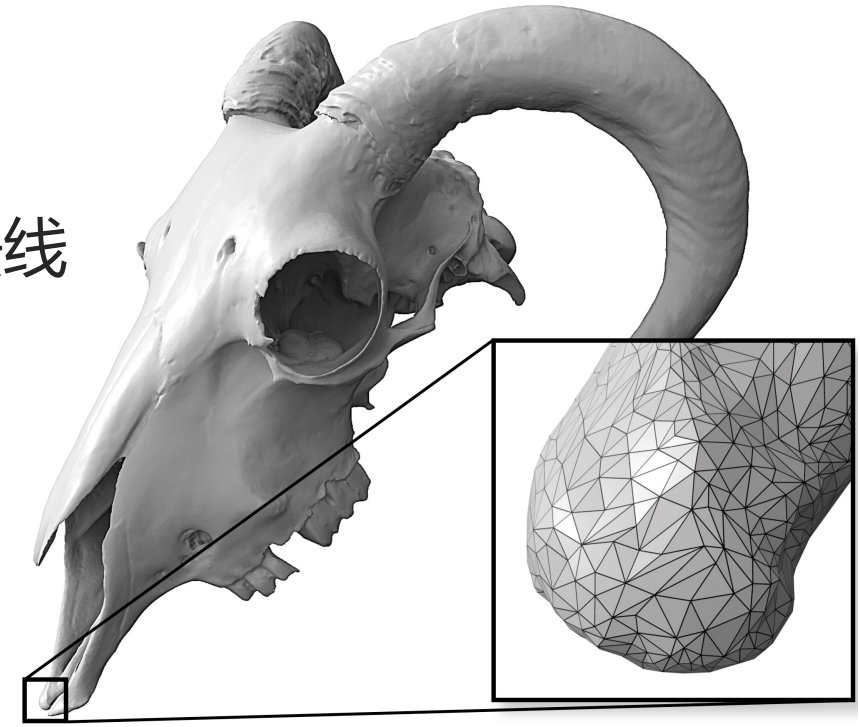
- 基于光栅化的现代实时图像生成
  - Input：图形基元 - 基本都是**点 (points)** 和**三角形 (triangles)**
    - 可能有其他属性（例如颜色）
  - Output：图像（可能有深度、透明度等）
- 我们的目标：理解其中不同阶段的细节*

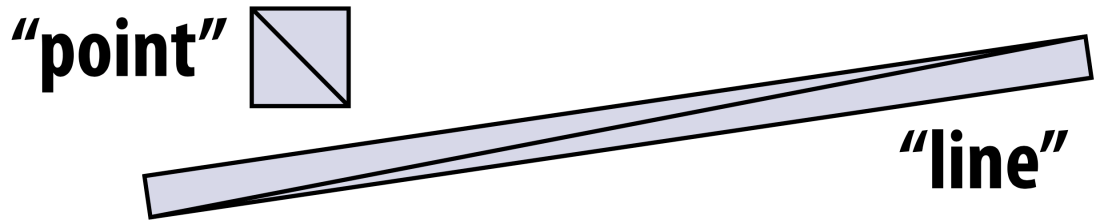


*In practice, usually executed by graphics processing unit (GPU)

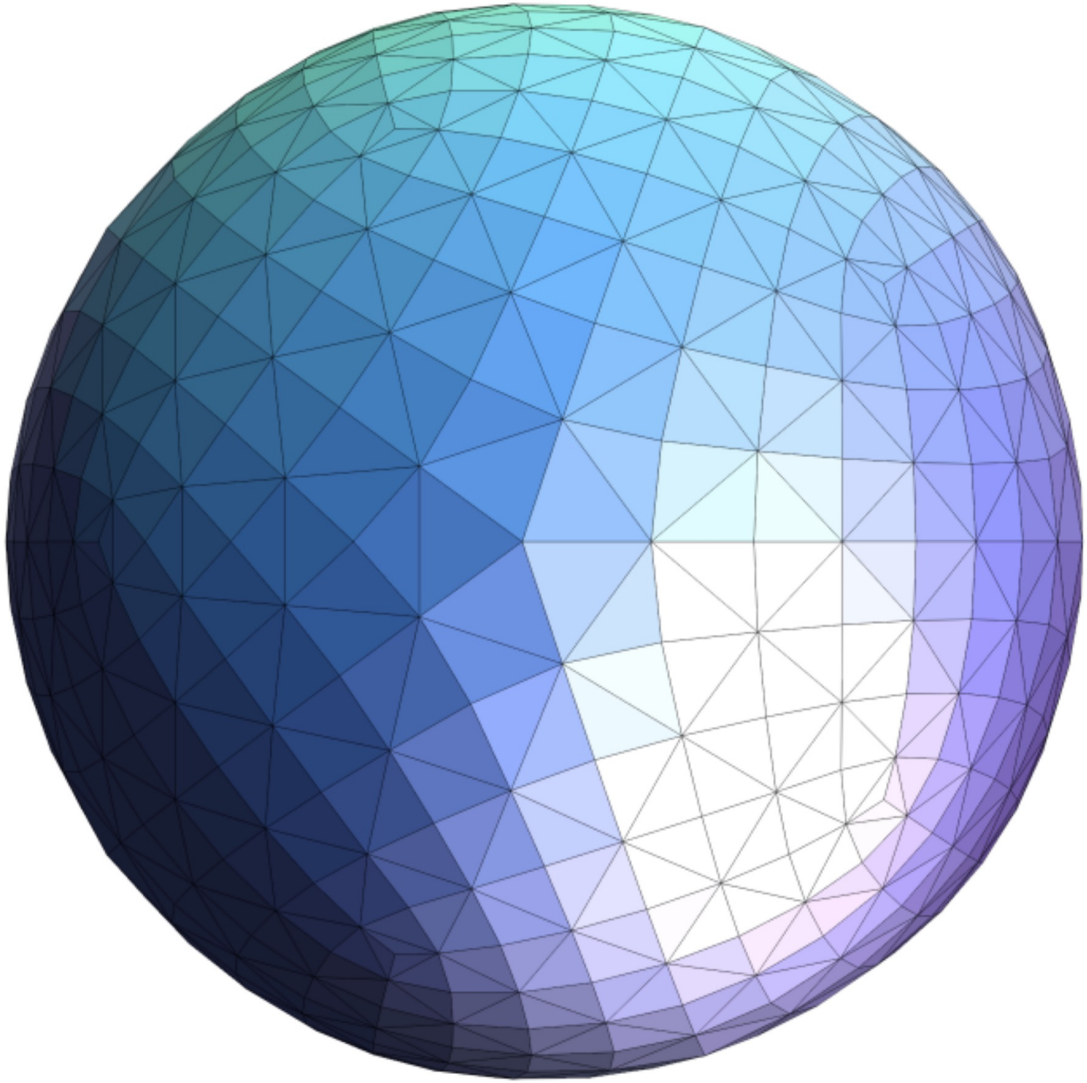
Why triangles?

- 光栅化管道将所有的图形基元转换成三角形
- 为什么选择三角形?
  - 可以近似任何形状
  - 总是平面的，有定义明确的法线
  - 定义良好的三角形顶点插值方法（重心插值法）
- 关键原因：一旦所有东西都被简化为三角形，就可以专注于设计一个深度优化的绘制流程

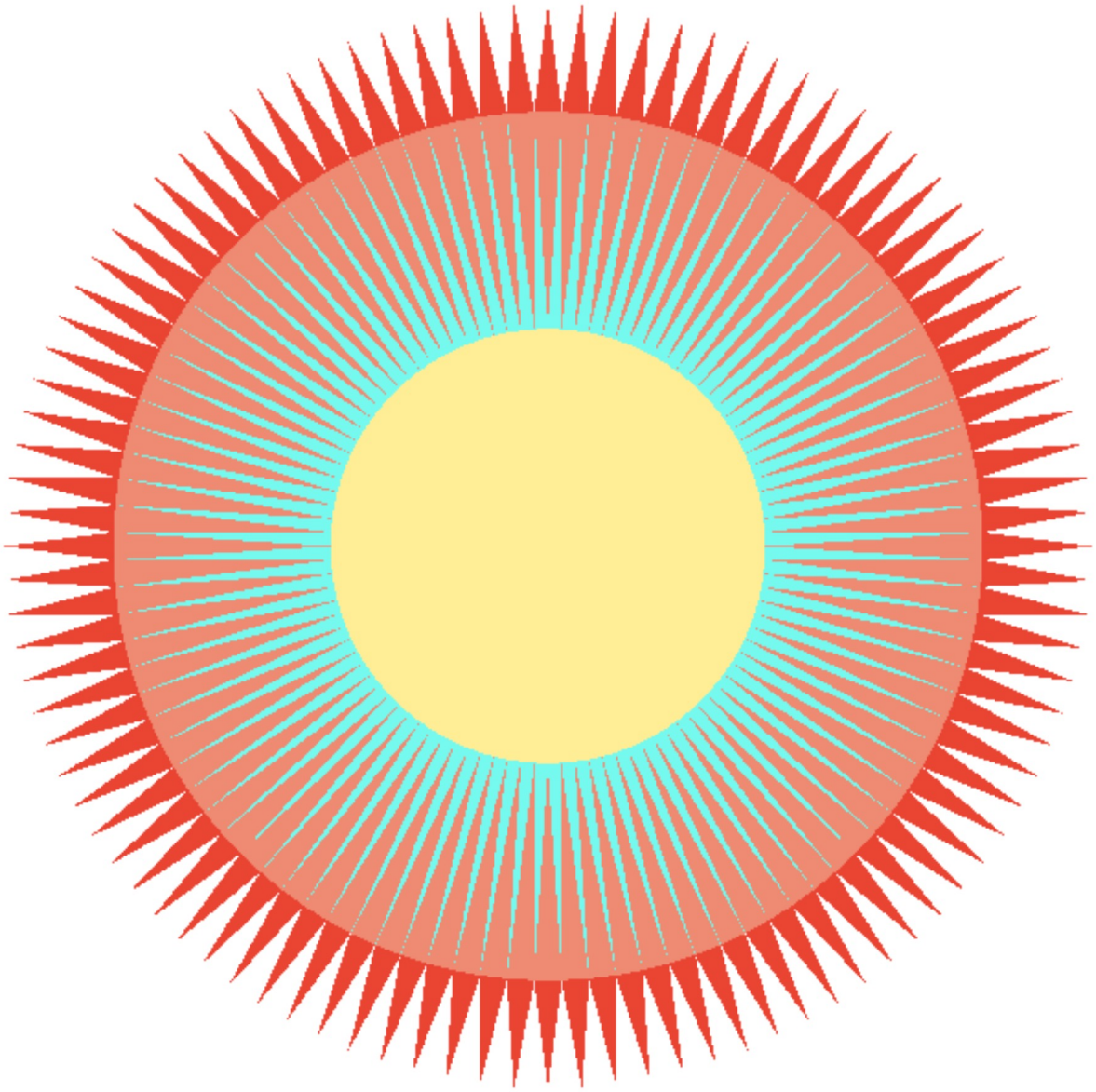




三角形网格 Triangle meshes



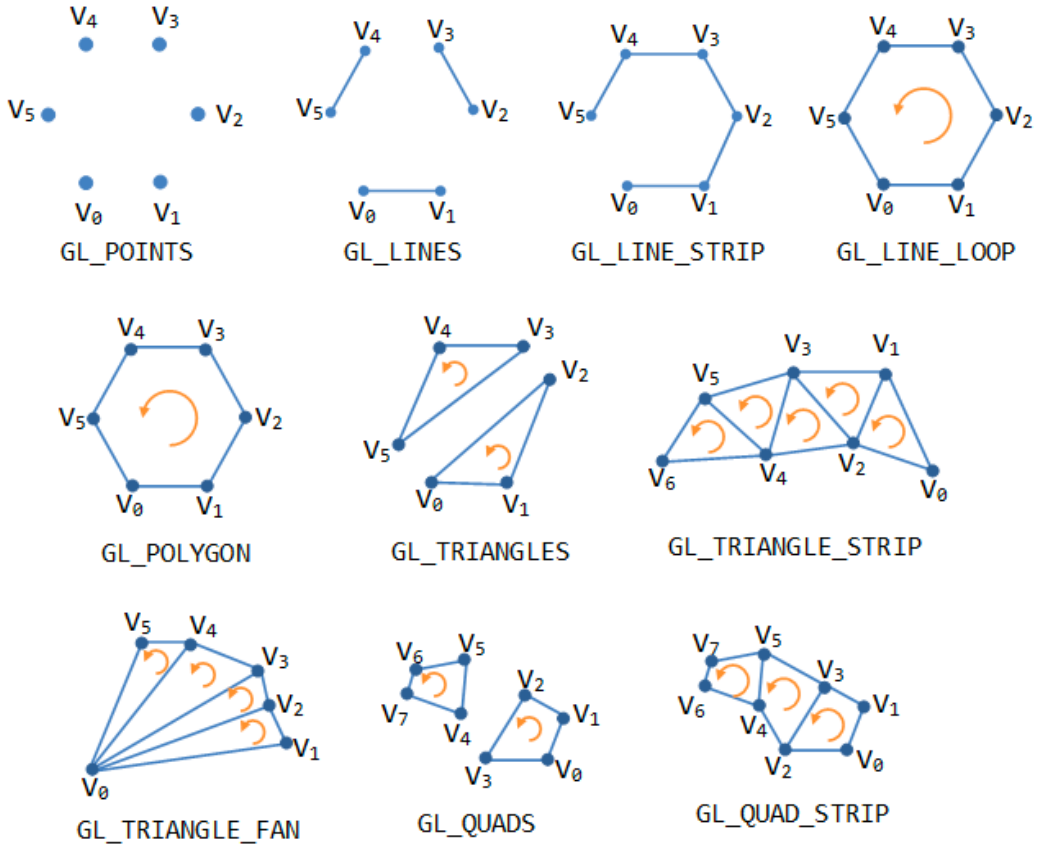
三角形网格 Triangle meshes



多边形网格 Polygon meshes

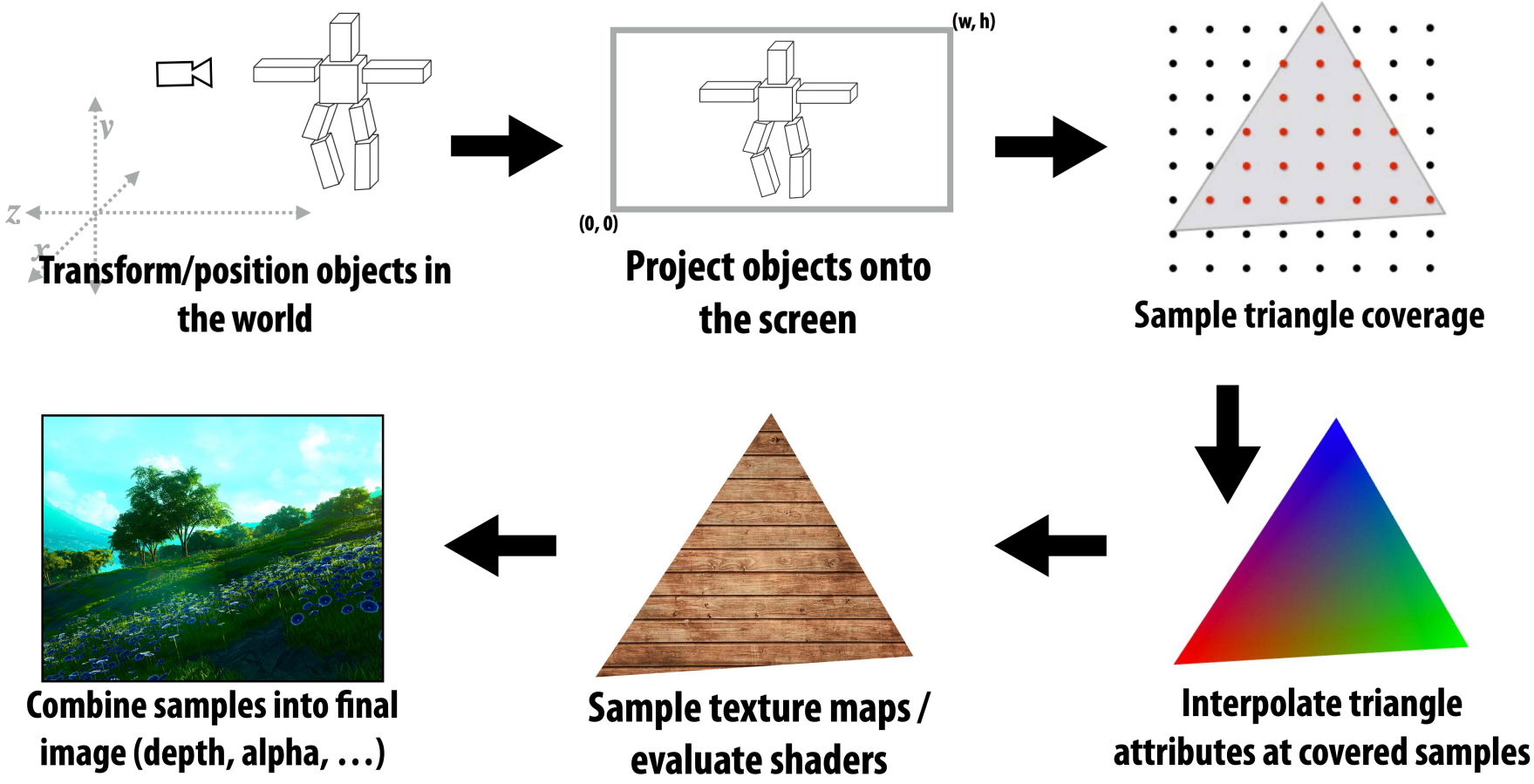
图形基元 Primitives

- 在 OpenGL 中，一个对象由三角形、四边形、线段和点等几何基元组成。一个基元由一个或多个顶点组成



OpenGL Primitives

光栅化流程中的步骤

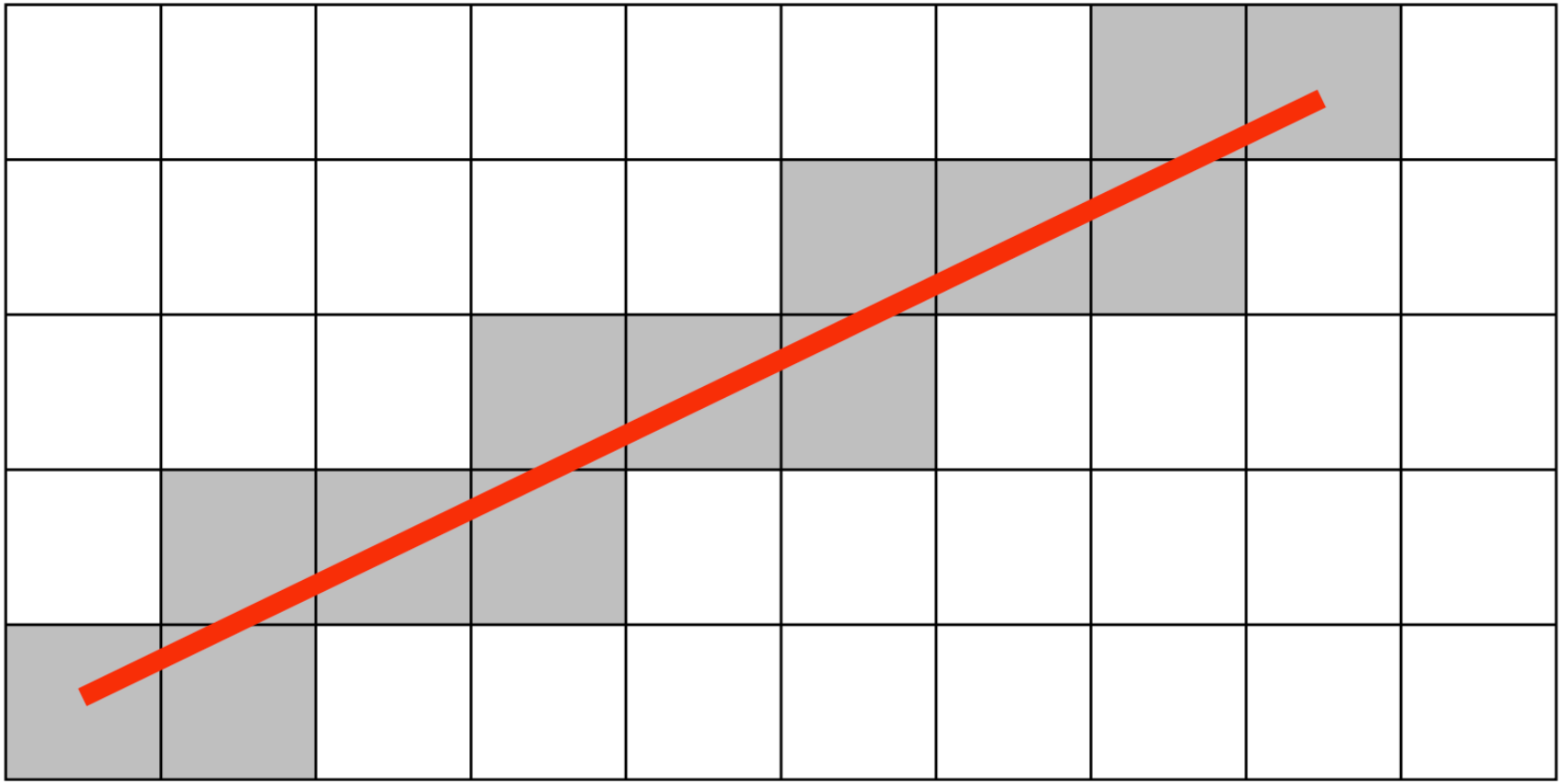


- “真实世界”的标准光栅化流程 (OpenGL/Direct3D)

在屏幕上画一个三角形

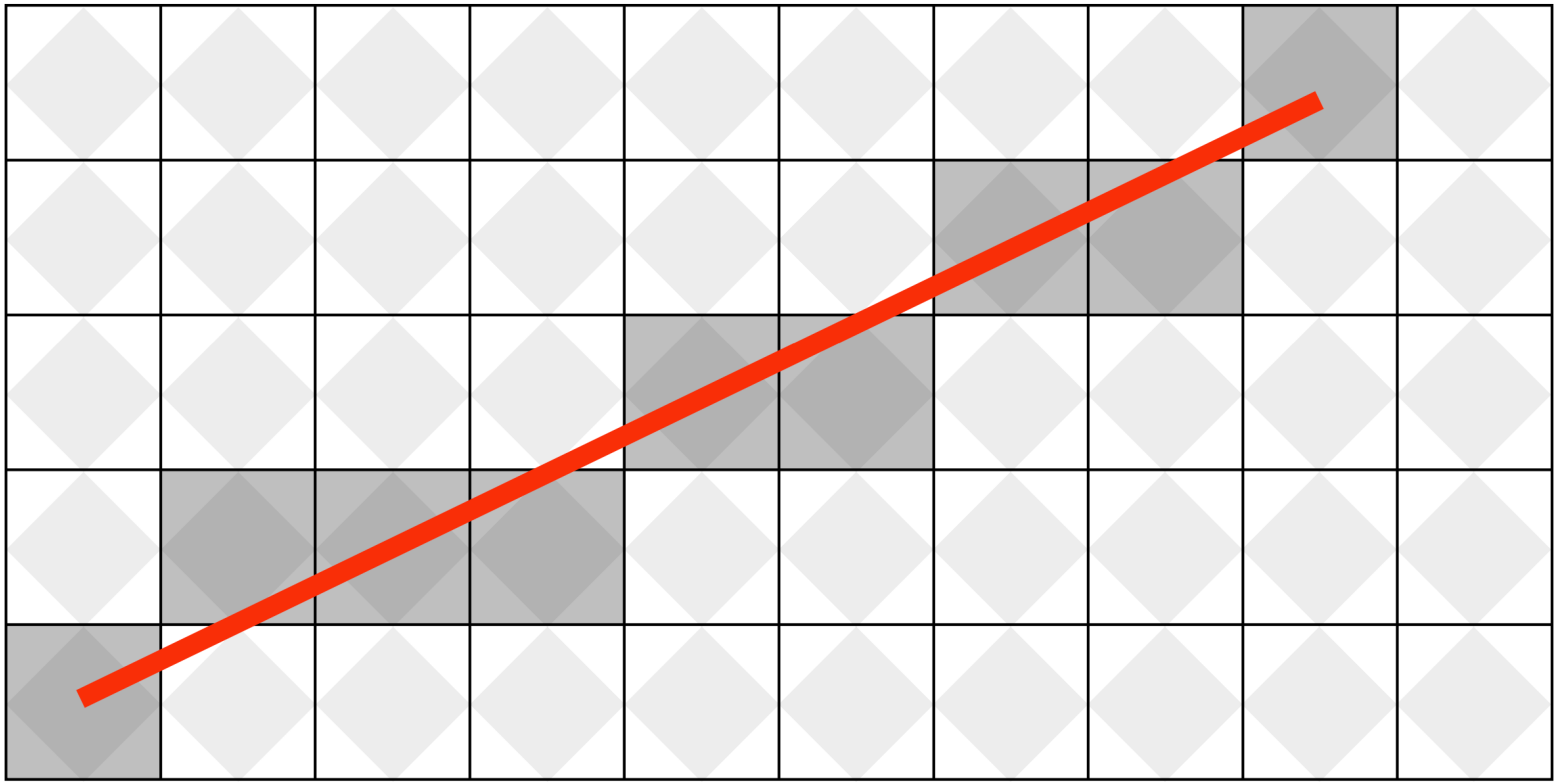
我们应该填充哪些像素来绘制直线?

点亮所有被线条穿过的像素?



我们应该填充哪些像素来绘制直线?

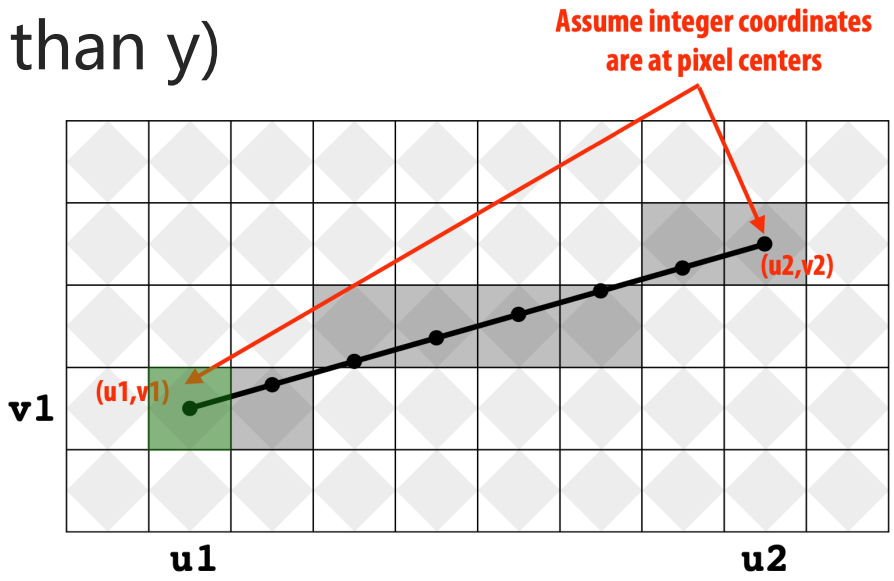
钻石规则 Diamond rule (现代 GPU 使用):
如果线穿过响应的“钻石”，就点亮像素



增量线光栅化

- 我们用两个点表示一条线：**(u1, v1), (u2, v2)**
- 线的斜率 **s=(v2-v1)/(u2-u1)**
- 考虑一个具体的例子
  - u1<u2, v1<v2 (line points toward upper-right)
  - 0<s<1 (more change in x than y)

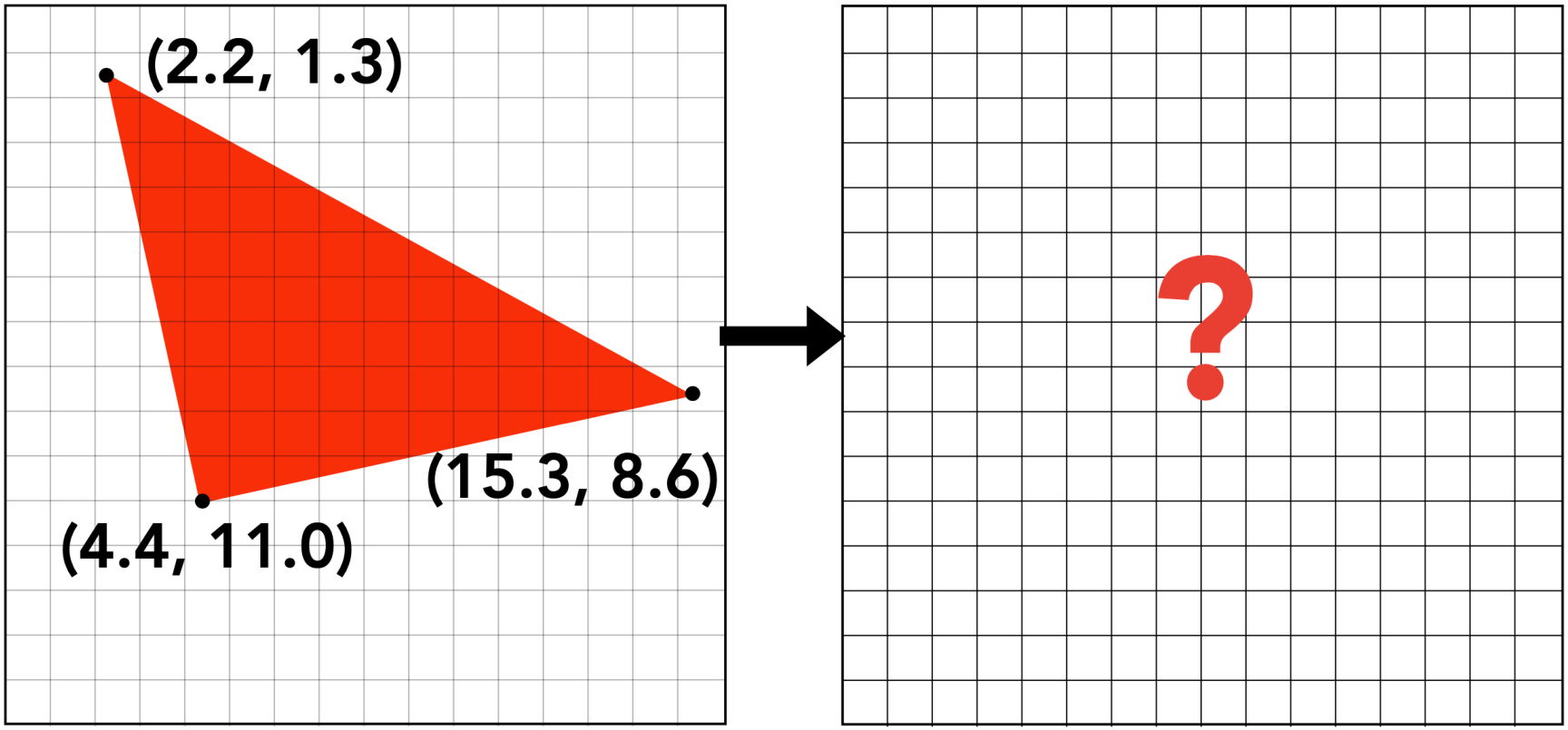
```
v = v1;
for(u=u1; u<=u2; u++)
{
    v += s;
    draw(u, round(v))
}
```



易于实现，但不是当代软件/硬件画线的唯一方式

三角形呢?

口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

我们从一个简单的方法开始：采样

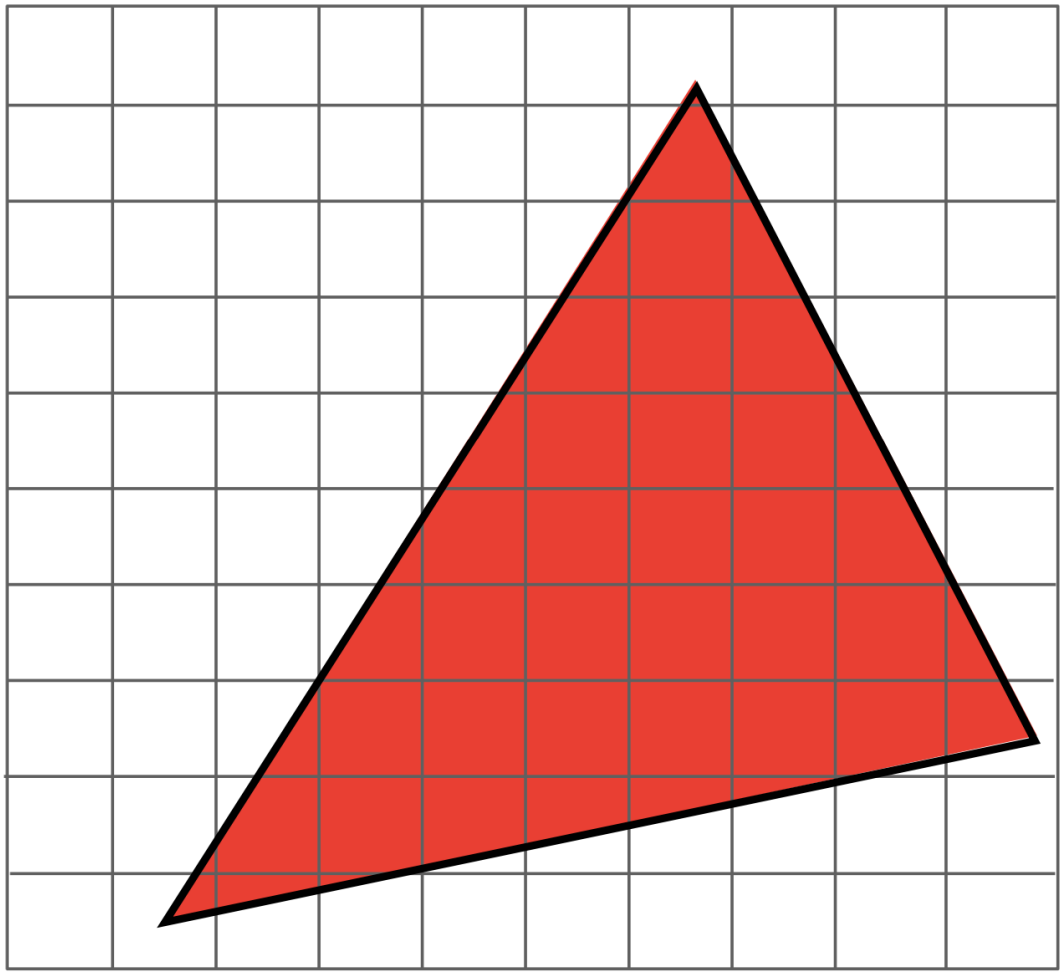
采样函数

- 测量一个函数在某点的值就是采样 Sampling
- 我们可以通过**周期采样 (periodic sampling)** 来**离散化 (discritize)** 函数

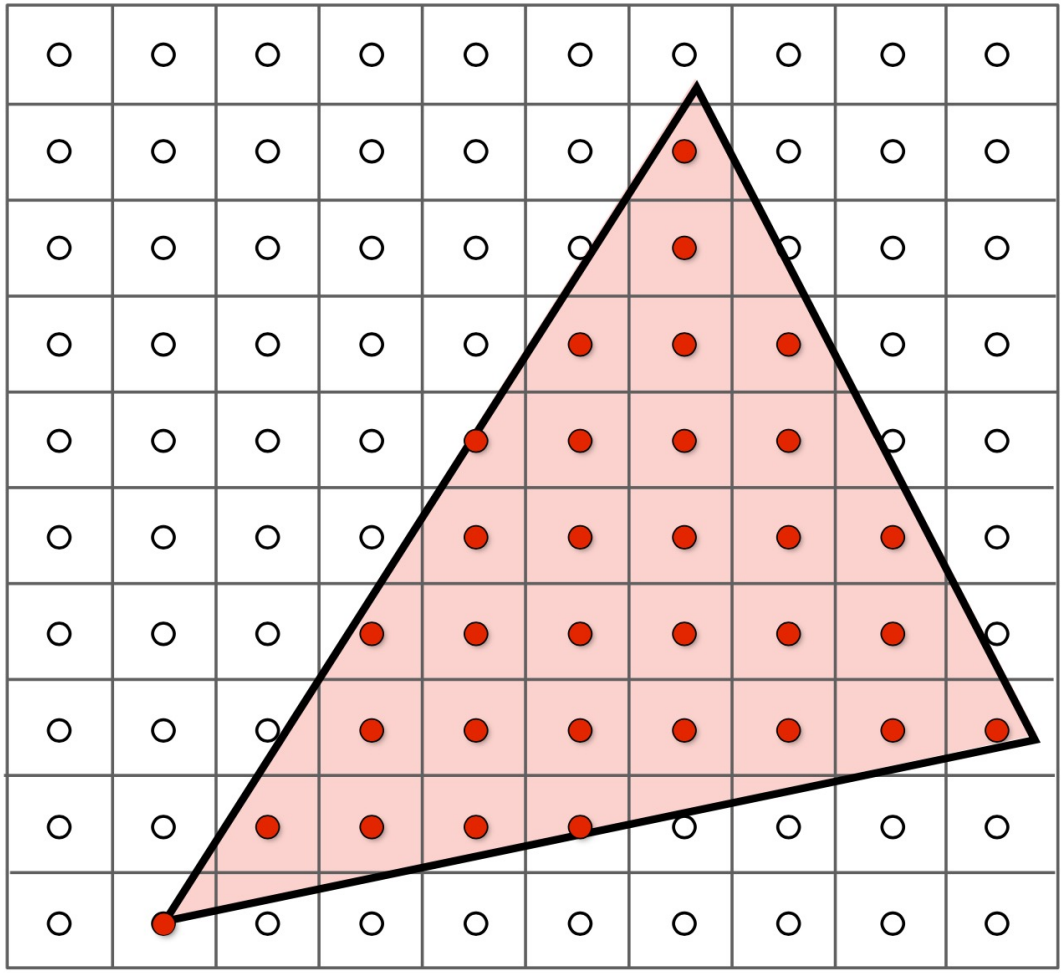
```
for( int x = 0; x < xmax; x++ )
    output[x] = f(x);
```

- 采样是图形的核心思想
  - 我们可以采样时间 (1D)、面积 (2D)、和体积 (3D)...
- 我们将对 n 维函数，甚至无穷维函数进行采样

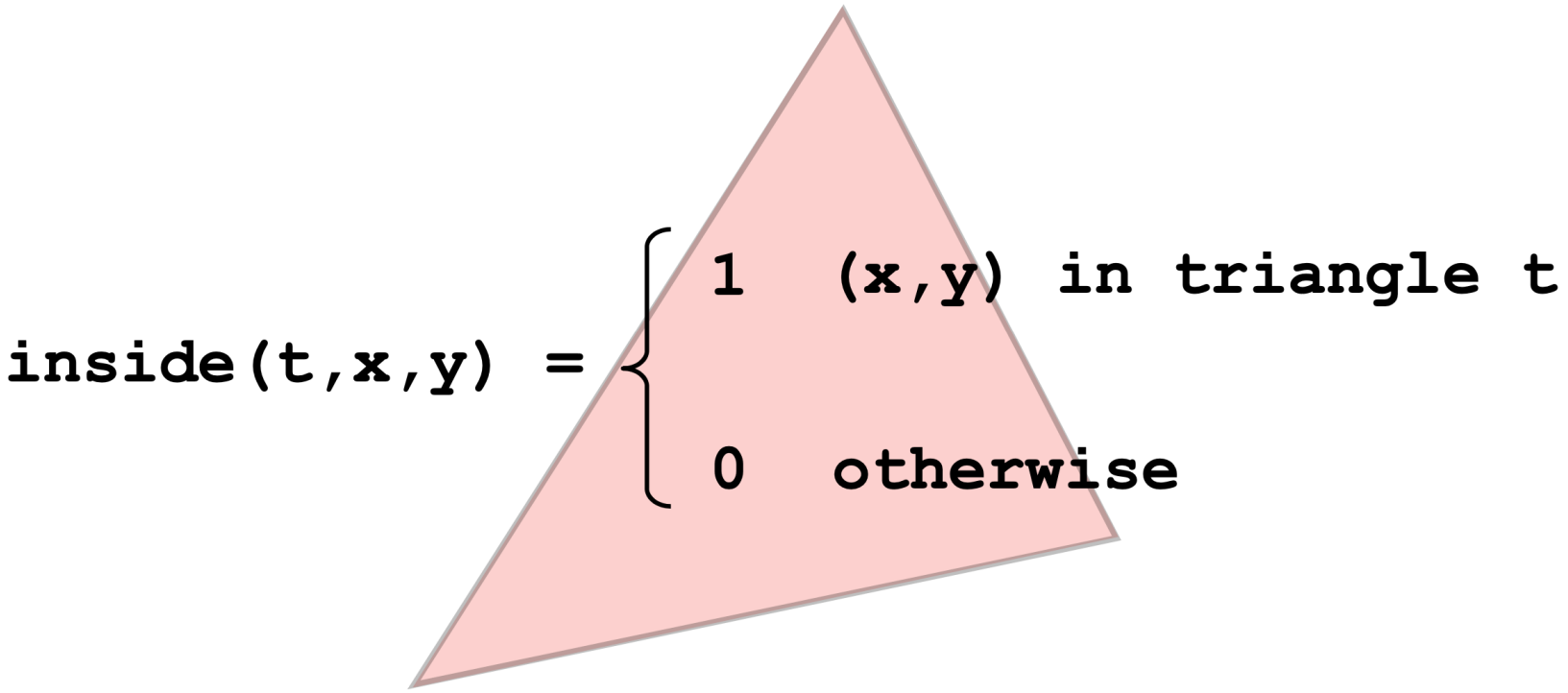
以 2D 采样的形式进行光栅化



若像素中心在三角形内，则采样



定义一个函数：inside(tri, x, y)



x, y: not necessarily integers

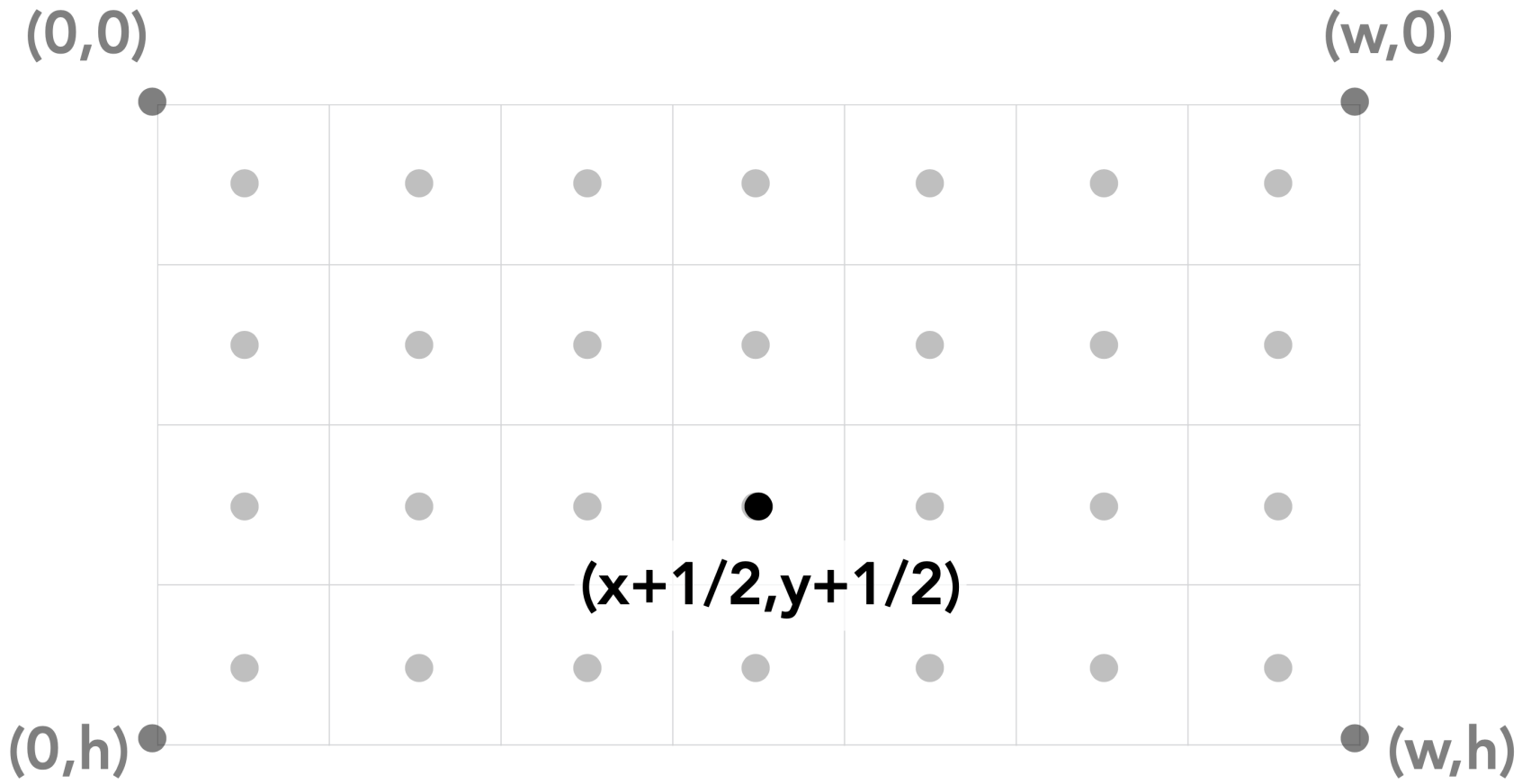
光栅化 = 采样一个 2D 指示函数

Rasterize triangle `tri` by sampling the function

`f(x,y) = inside(tri,x,y)`

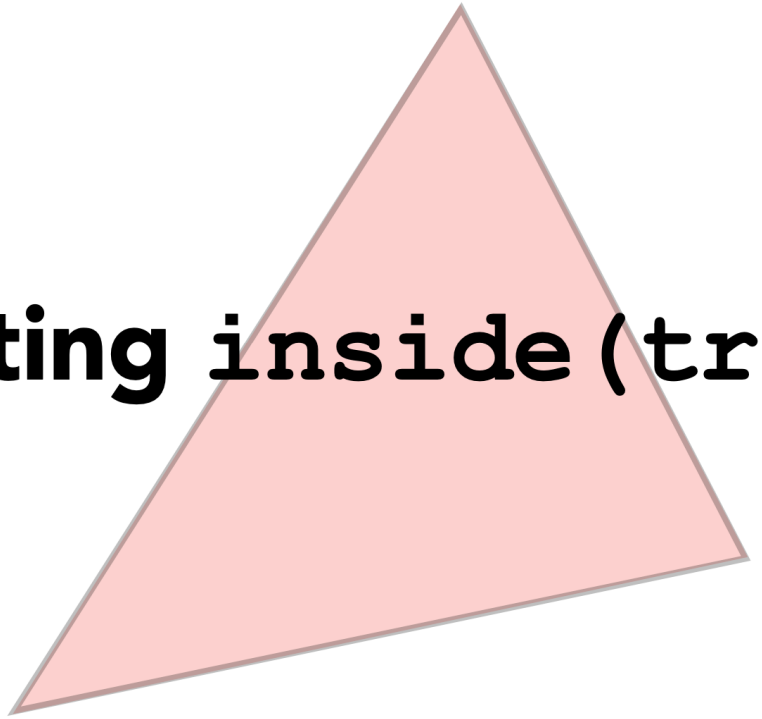
```
for( int x = 0; x < xmax; x++ )
  for( int y = 0; y < ymax; y++ )
    Image[x][y] = f(x + 0.5, y + 0.5);
```

实现细节：采样点的位置

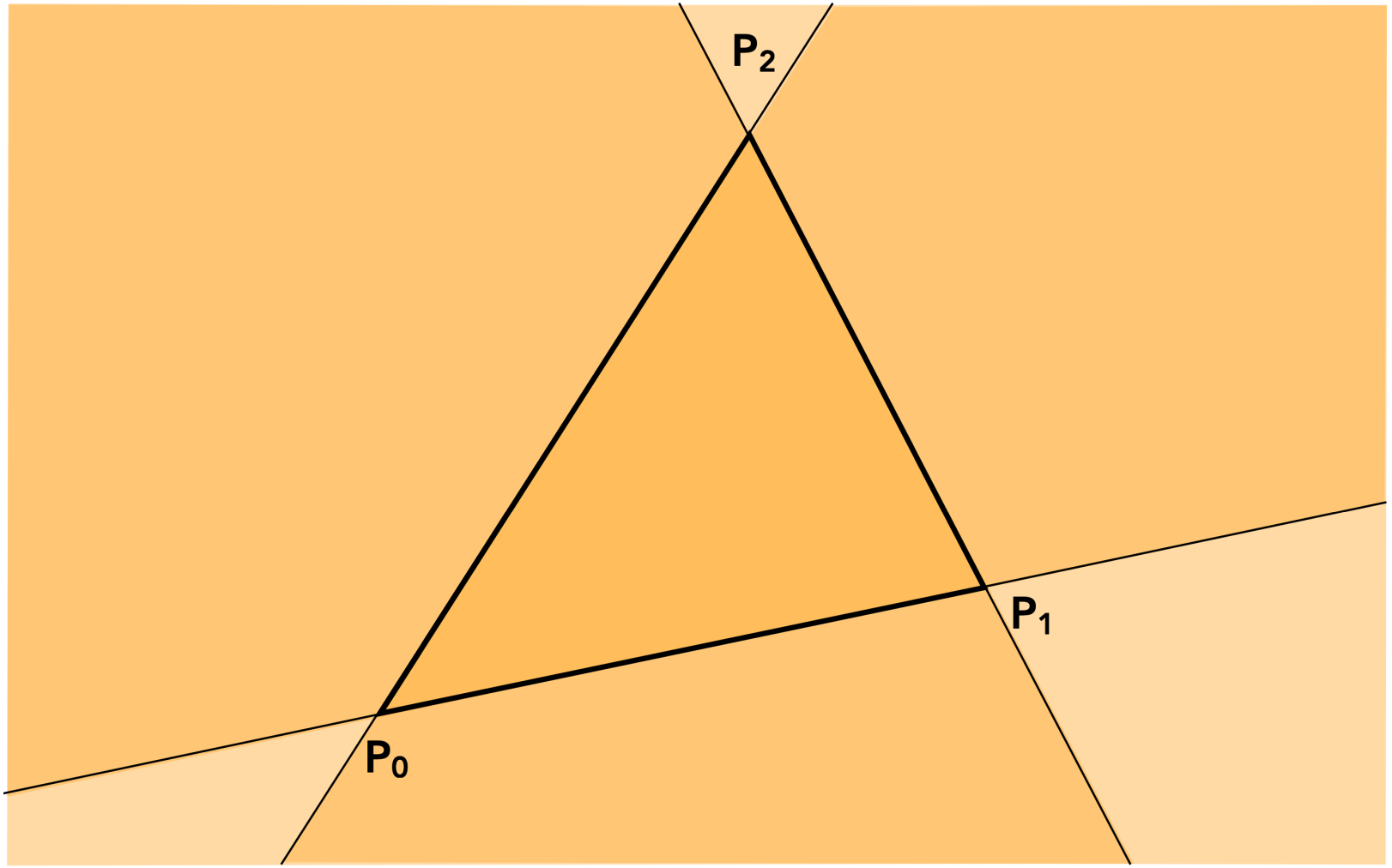


Sample location for pixel (x,y)

Evaluating `inside(tri,x,y)`



三角形 = 三个半平面的交集

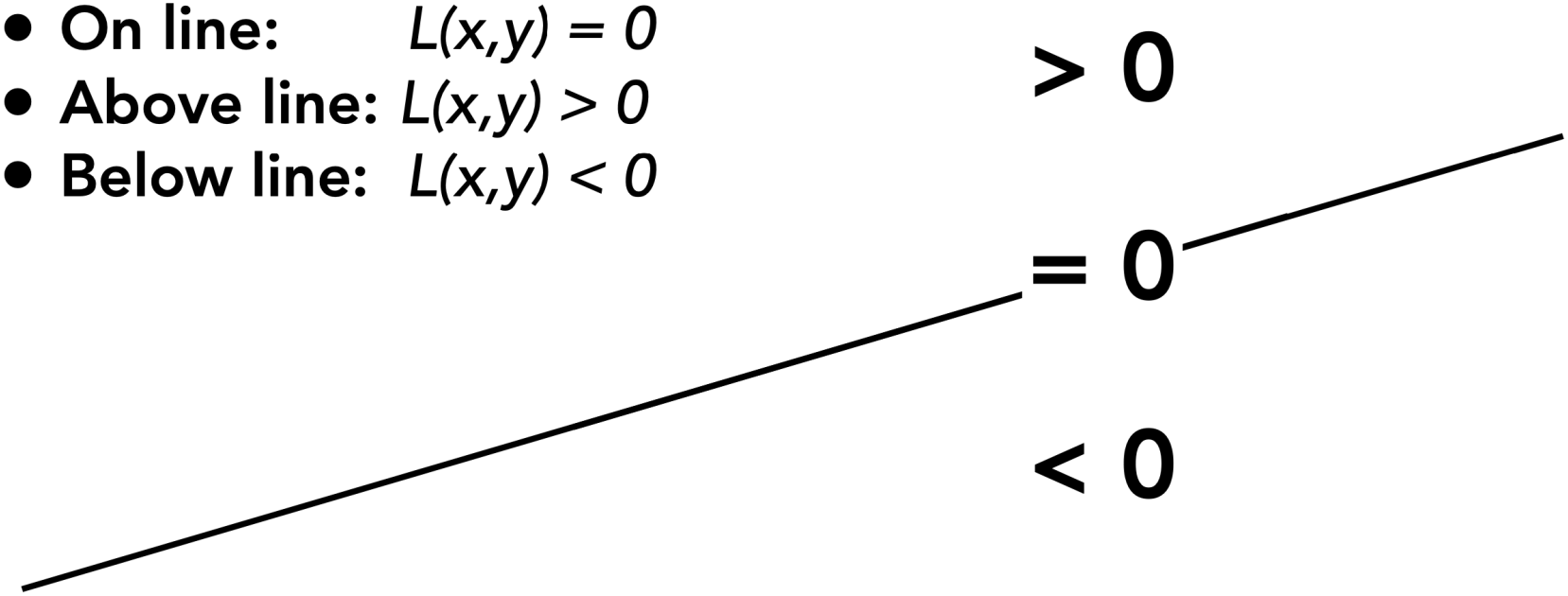


一条线定义一个半平面

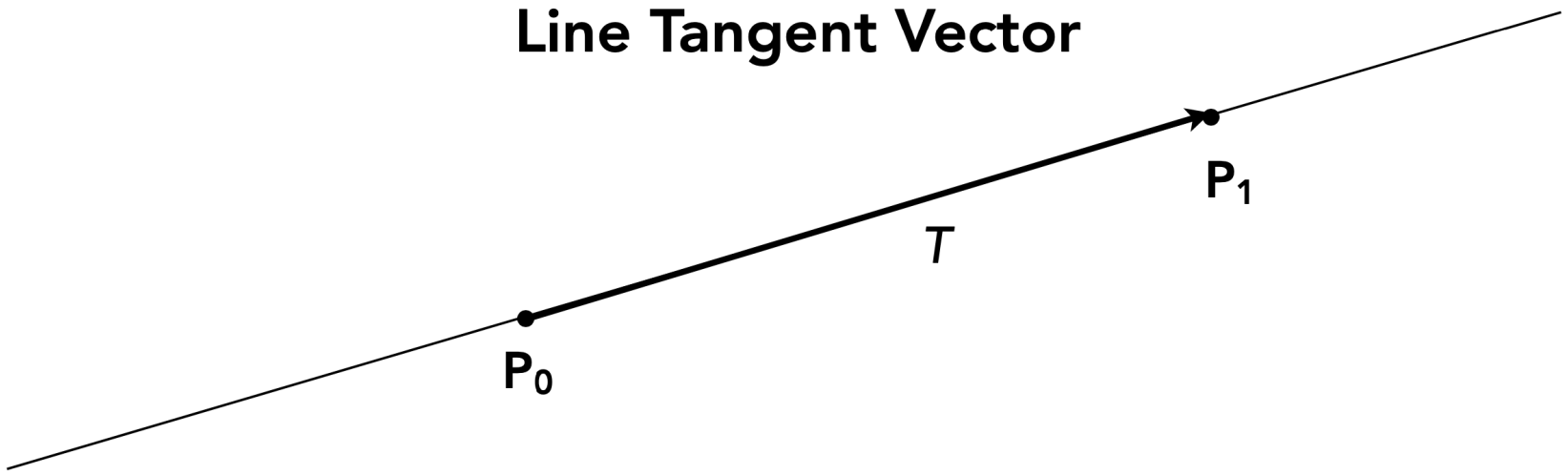
Implicit line equation

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

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

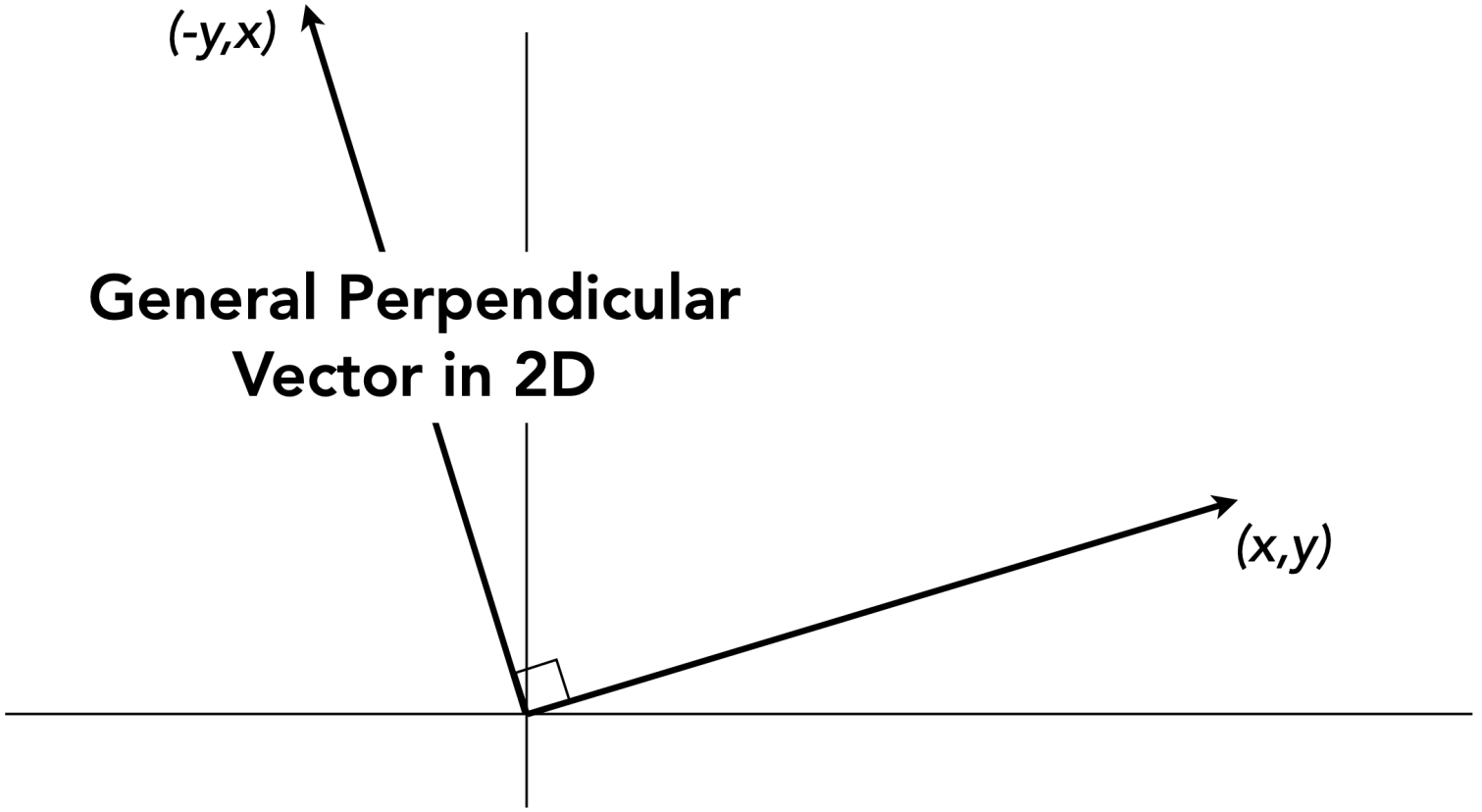


直线方程推导



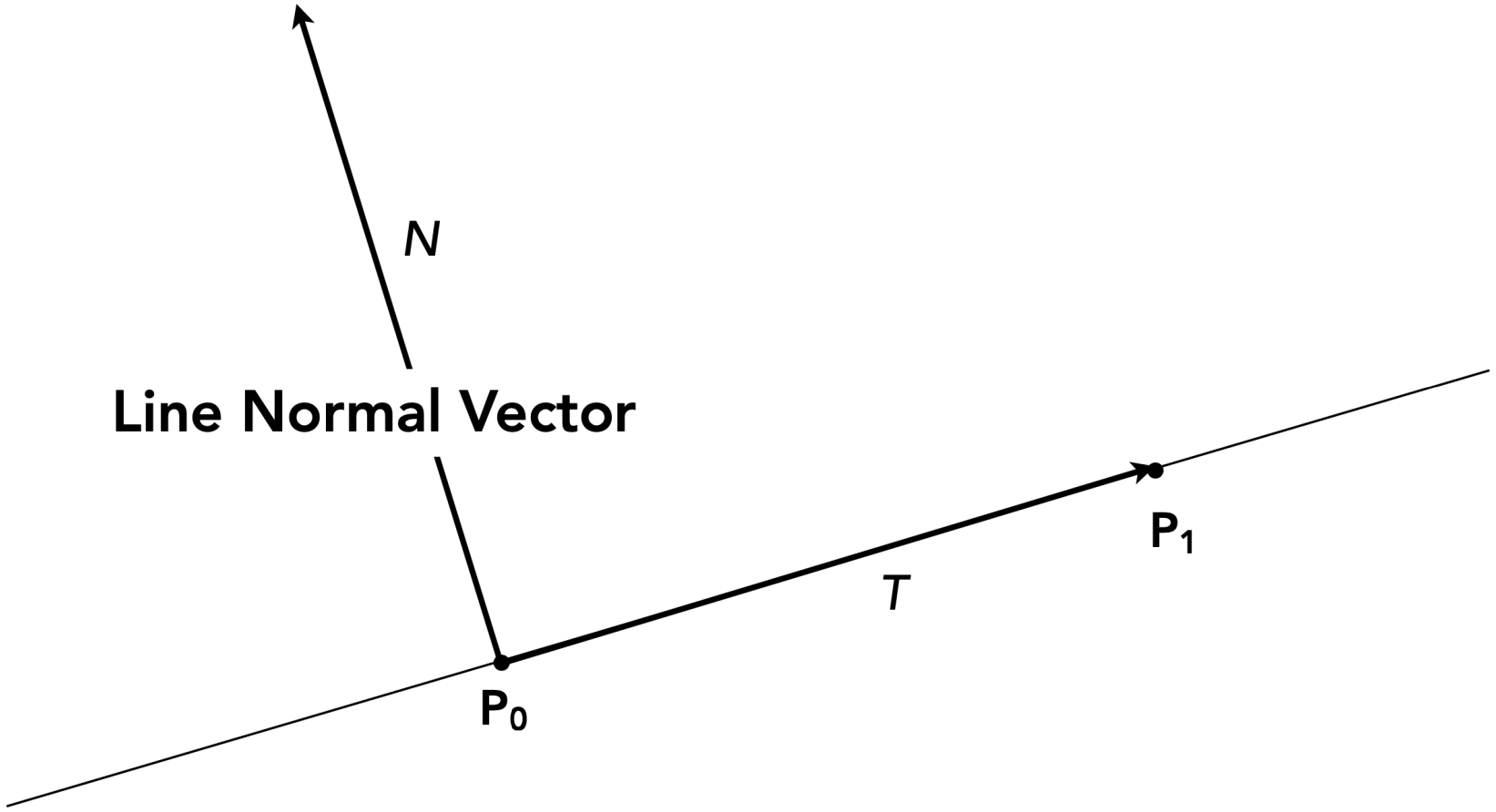
$$T = P_1 - P_0 = (x_1 - x_0, y_1 - y_0)$$

直线方程推导



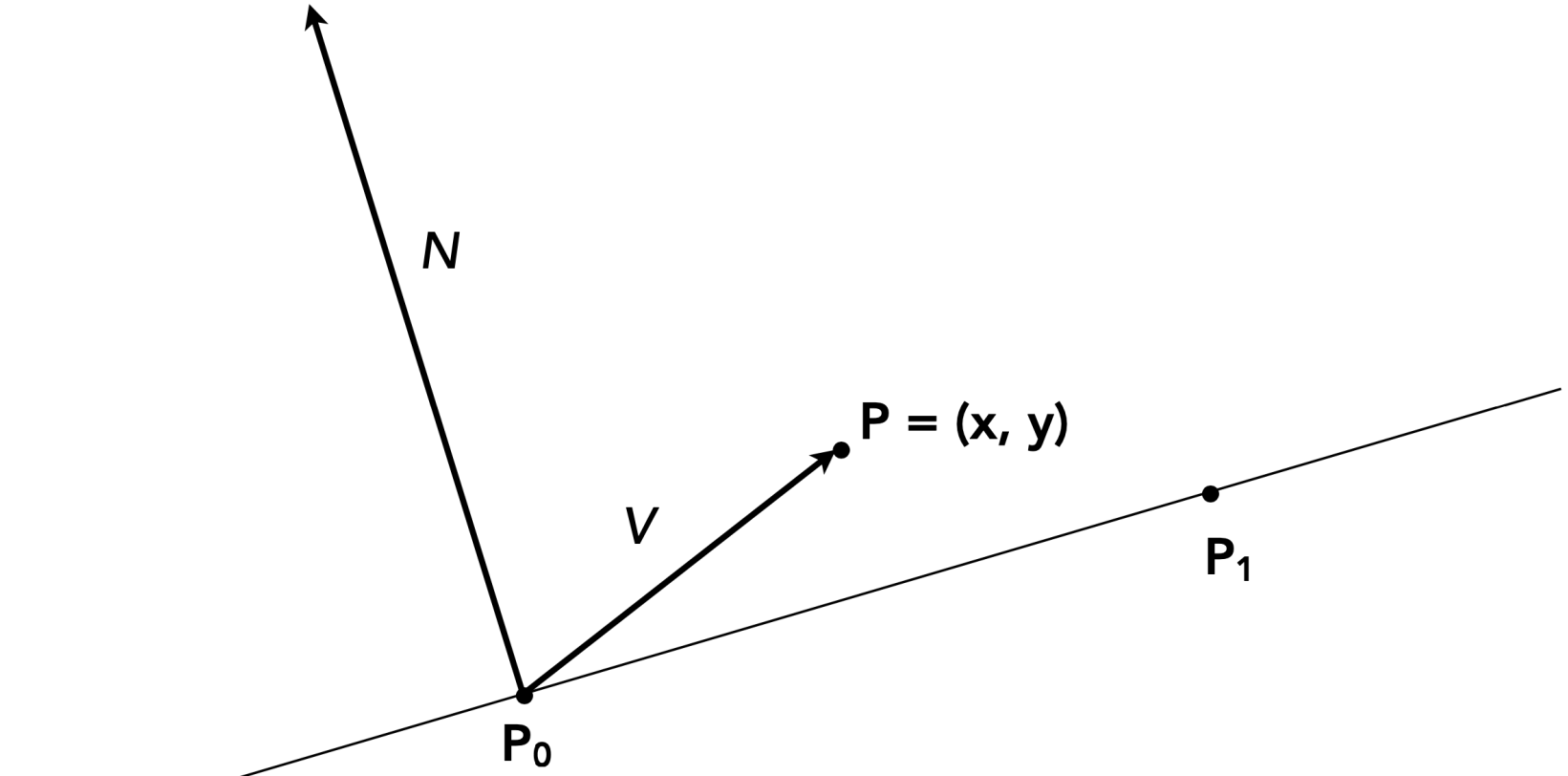
$$\mathrm{Perp}(x, y) = (-y, x)$$

直线方程推导



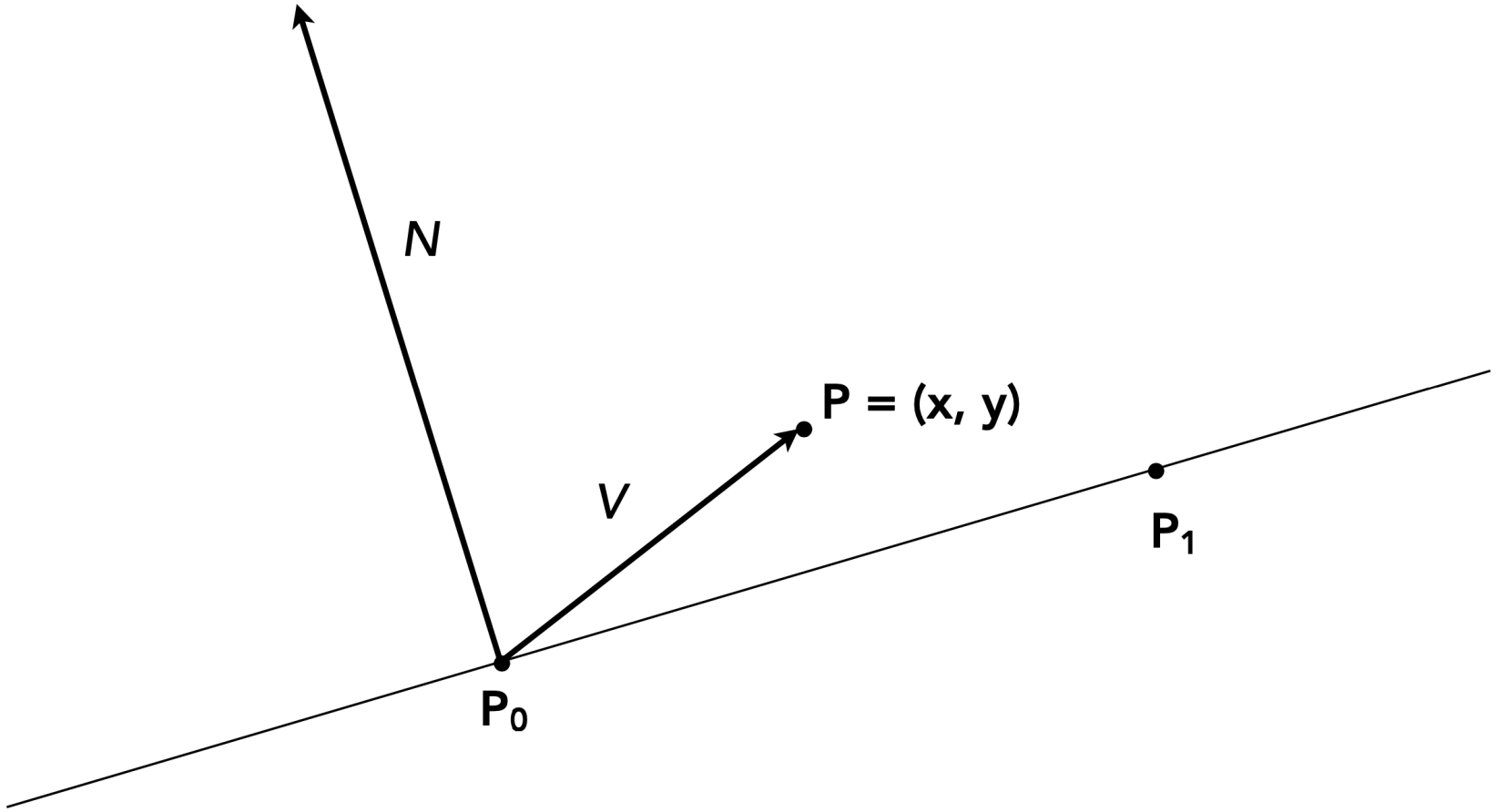
$$N = \mathrm{Perp}(T) = (-(y_1 - y_0), x_1 - x_0)$$

直线方程推导



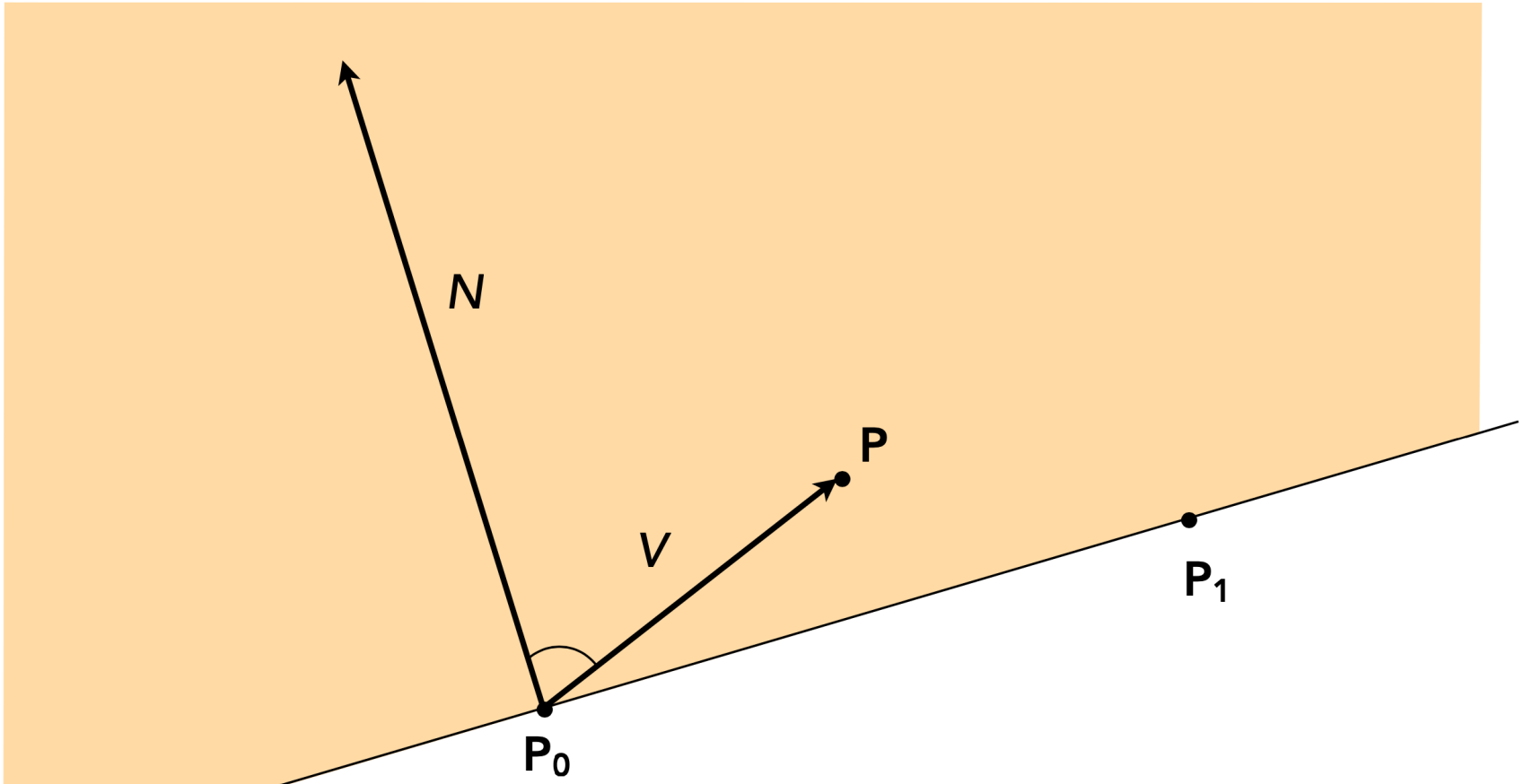
$$V = P - P_0 = (x - x_0, y - y_0)$$

直线方程



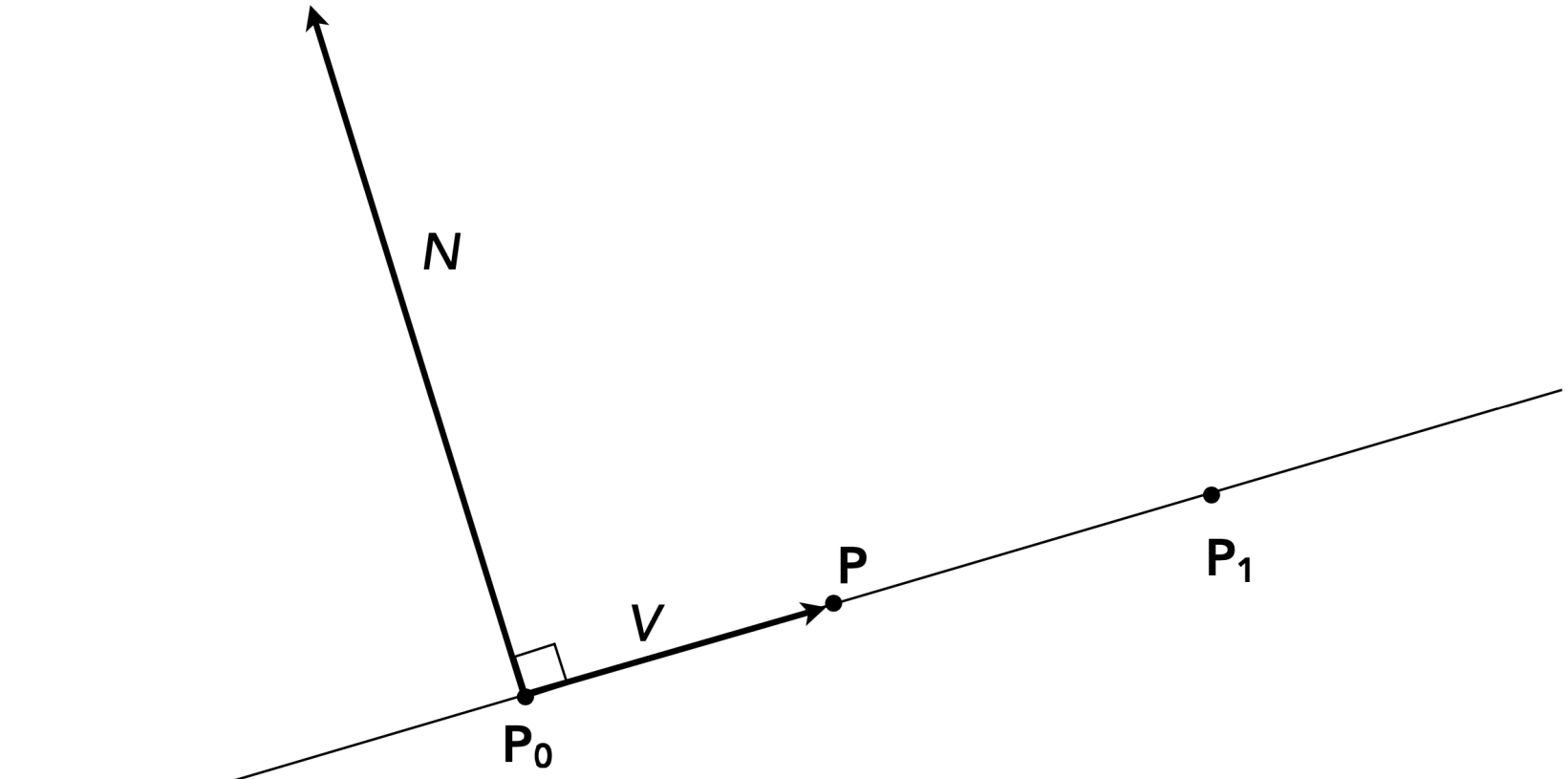
$$L(x,y) = V \cdot N = -(x - x_0)(y_1 - y_0) + (y - y_0)(x_1 - x_0)$$

直线方程测试



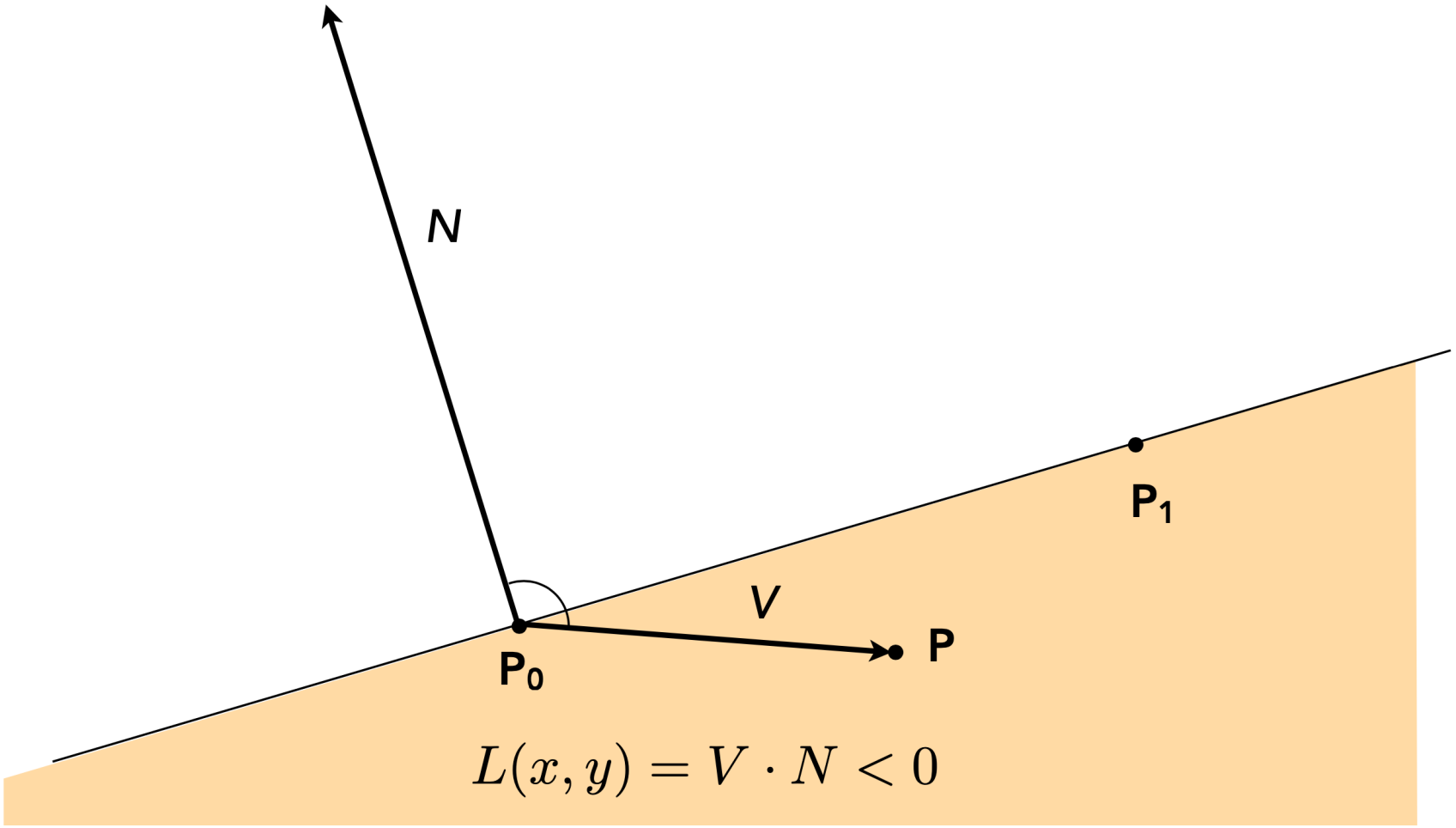
$$L(x,y) = V \cdot N > 0$$

直线方程测试



$$L(x,y) = V \cdot N = 0$$

直线方程测试



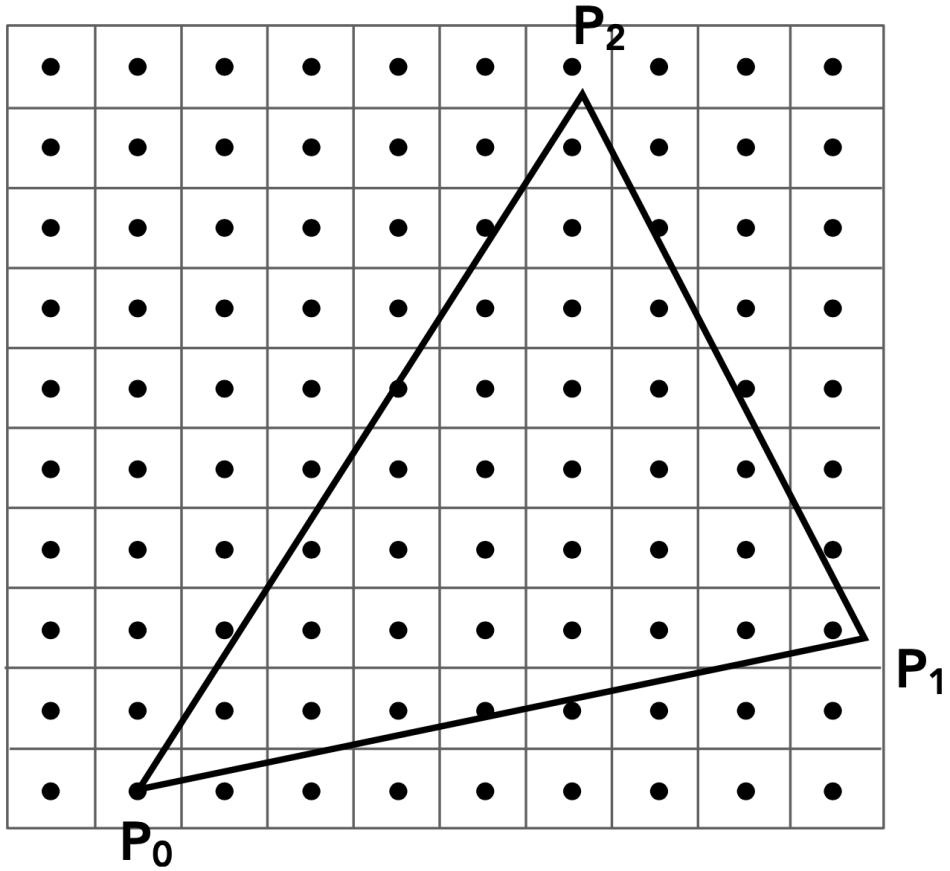
三角形内点测试：三线测试

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$L_i(x, y) = -(x - X_i)\, dY_i + (y - Y_i)\, dX_i$
$= A_i x + B_i y + C_i$

$L_i(x, y) = 0$: point on edge
< 0 : outside edge
> 0 : inside edge



Compute line equations from pairs of vertices

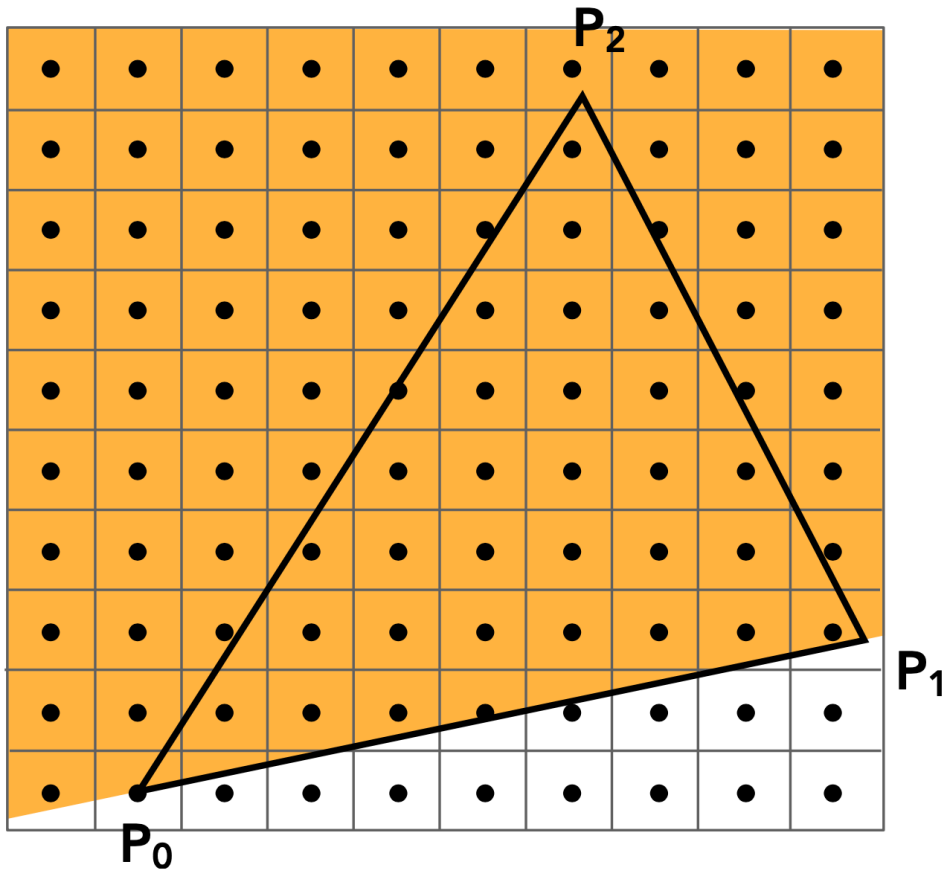
三角形内点测试：三线测试

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$L_i(x, y) = -(x - X_i)\, dY_i + (y - Y_i)\, dX_i$
$= A_i x + B_i y + C_i$

$L_i(x, y) = 0$: point on edge
< 0 : outside edge
> 0 : inside edge



$L_0(x, y) > 0$

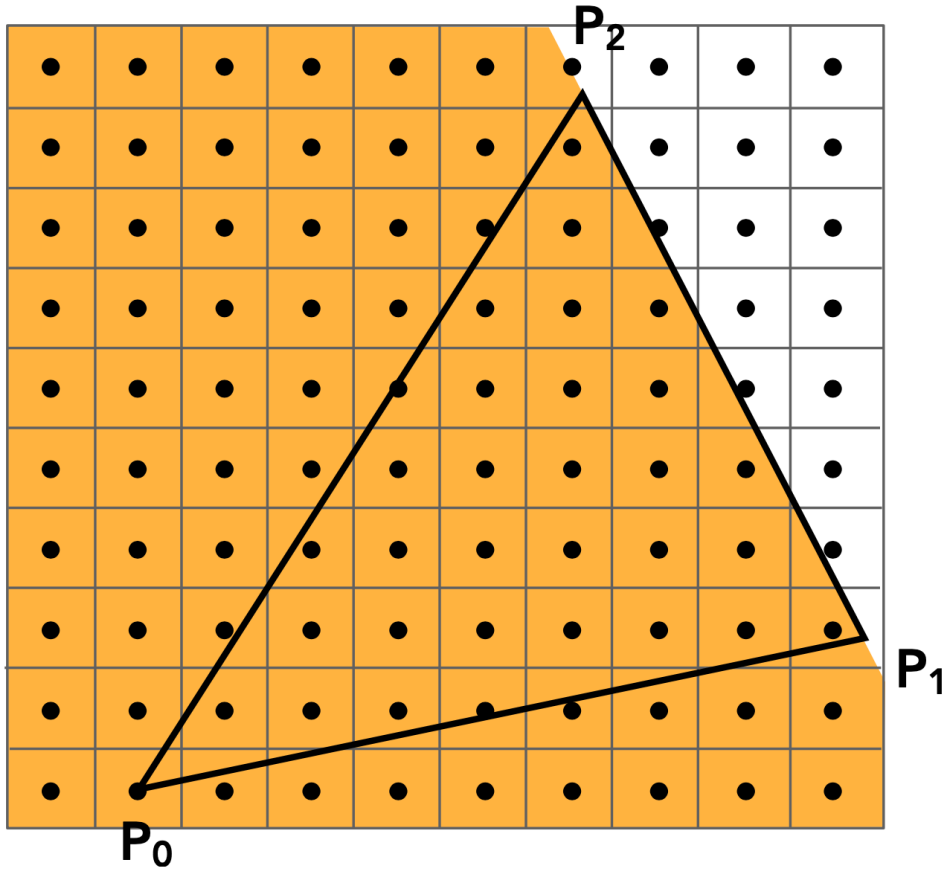
三角形内点测试：三线测试

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$L_i(x, y) = -(x - X_i)\, dY_i + (y - Y_i)\, dX_i$
$= A_i x + B_i y + C_i$

$L_i(x, y) = 0$: point on edge
< 0 : outside edge
> 0 : inside edge



$L_1(x, y) > 0$

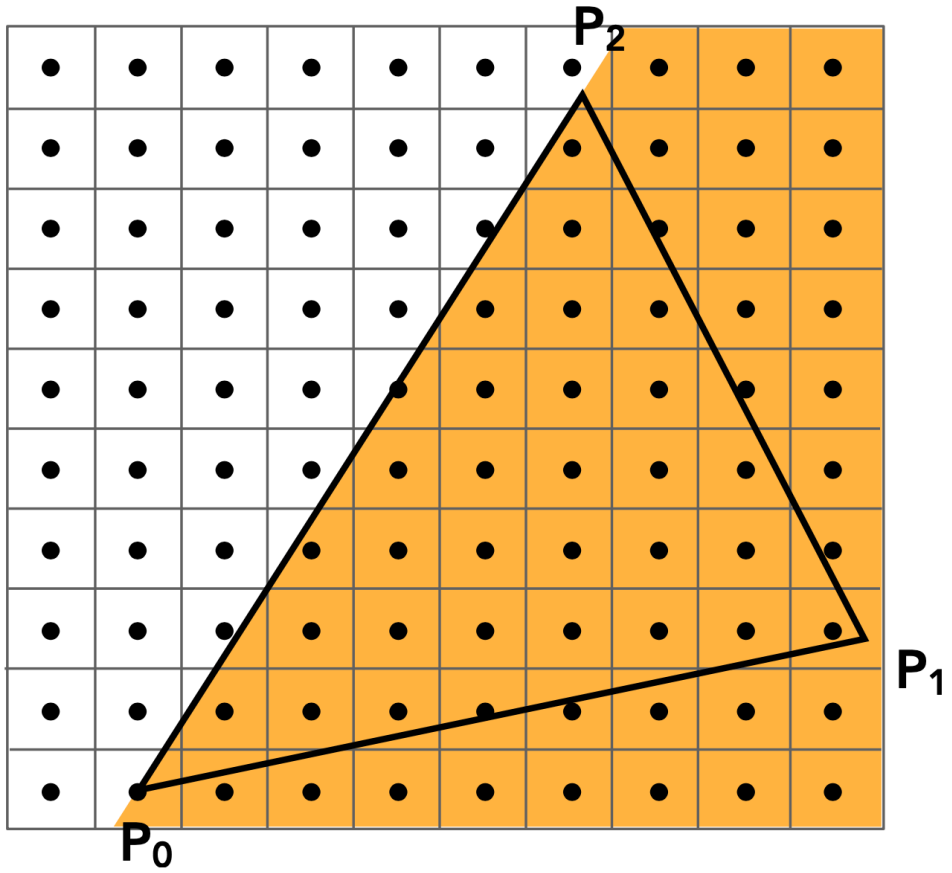
三角形内点测试：三线测试

$P_i = (X_i, Y_i)$

$dX_i = X_{i+1} - X_i$
$dY_i = Y_{i+1} - Y_i$

$L_i(x, y) = -(x - X_i)\, dY_i + (y - Y_i)\, dX_i$
$= A_i x + B_i y + C_i$

$L_i(x, y) = 0$: point on edge
< 0 : outside edge
> 0 : inside edge



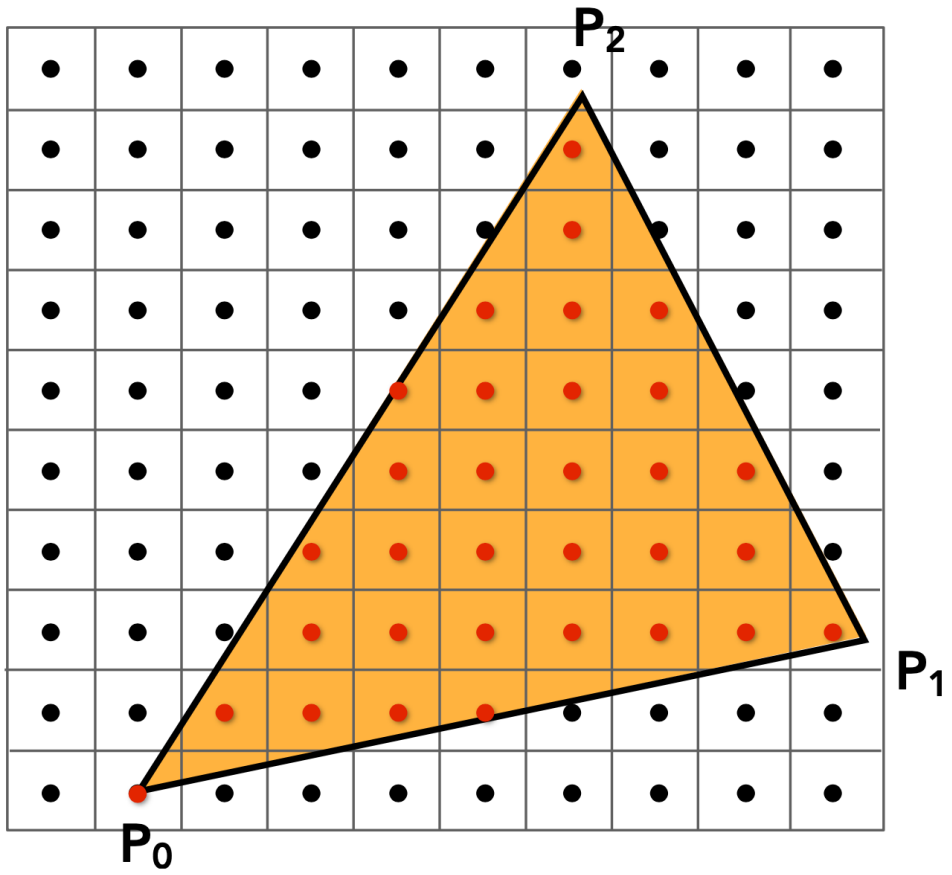
$L_2(x, y) > 0$

三角形内点测试：三线测试

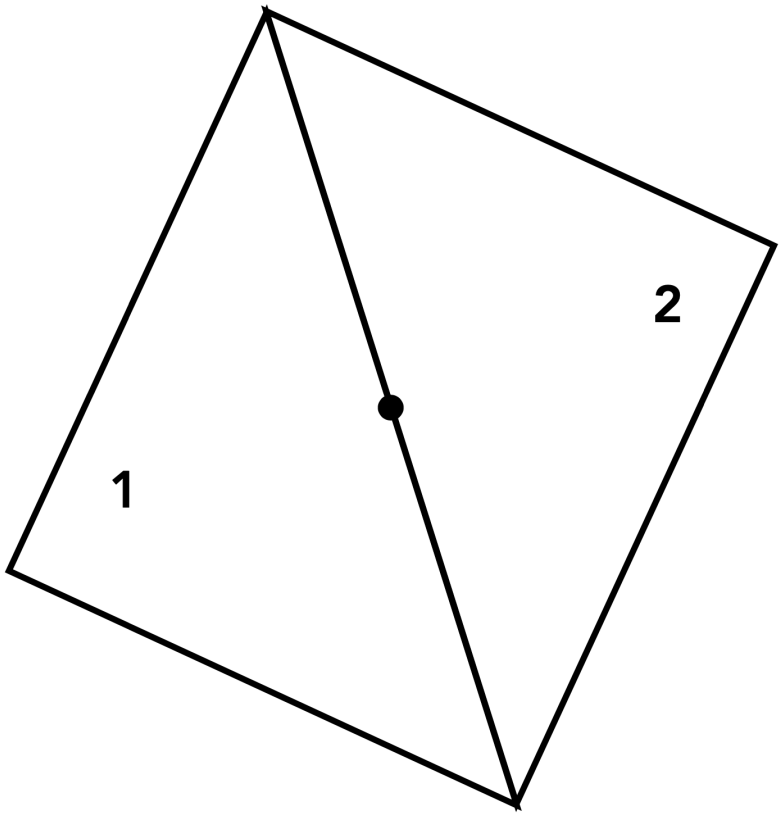
Sample point $s = (sx, sy)$ is inside the triangle if it is inside all three lines.

$$
\begin{aligned}
&inside(sx, sy) = \\
&\quad L_0(sx, sy) > 0 \;\&\& \\
&\quad L_1(sx, sy) > 0 \;\&\& \\
&\quad L_2(sx, sy) > 0;
\end{aligned}
$$

Note: actual implementation of $inside(sx,sy)$ involves ≤ checks based on edge rules

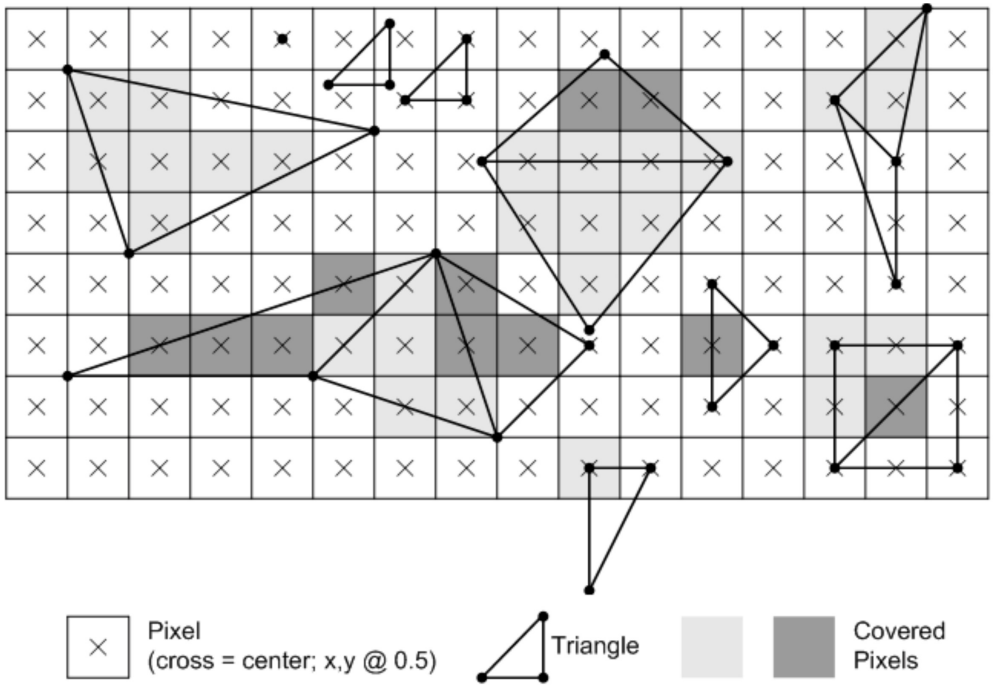


边上的采样点



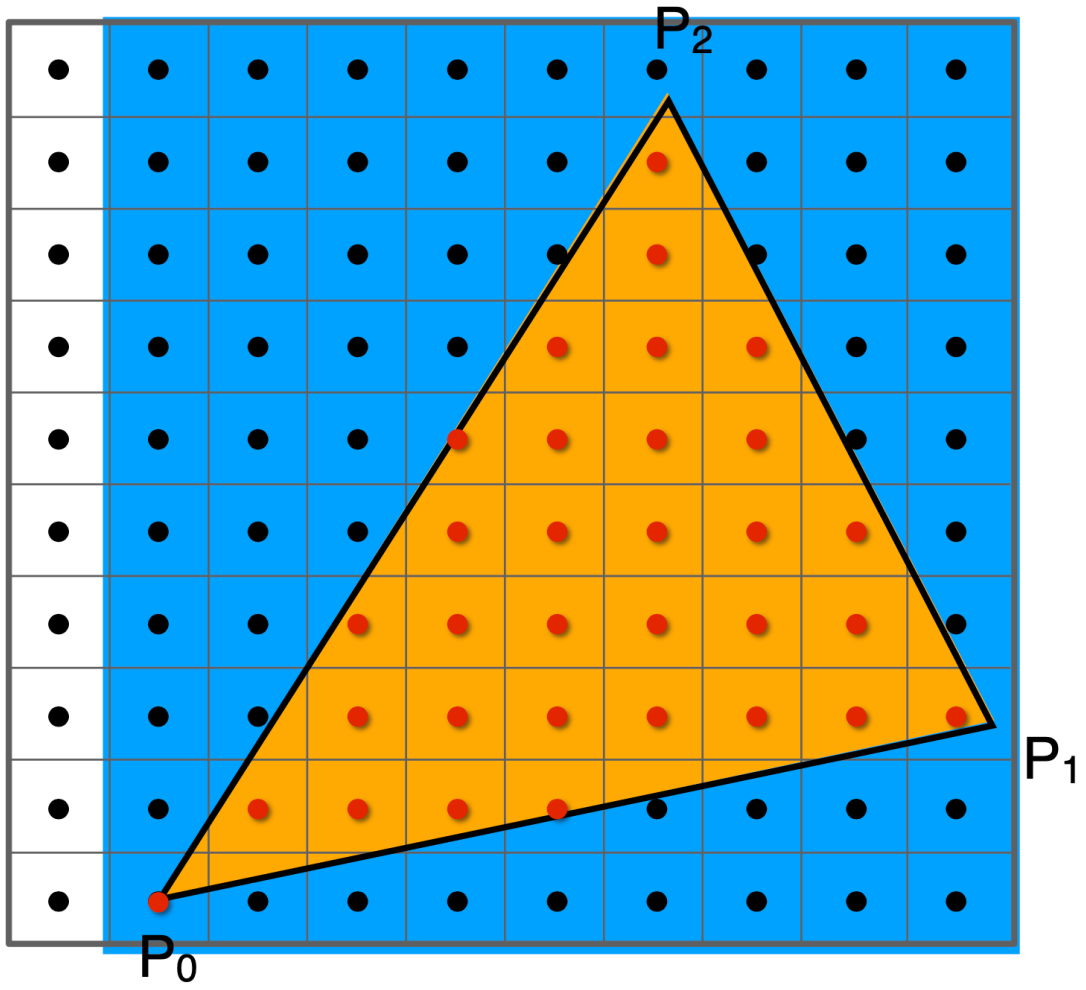
OpenGL/Direct3D 边缘规则

- 当采样点落在边上时，如果边是“顶边”或“左边”，则该采样被分类为三角形内
  - **顶边 (top edge)**: 高于所有其他边的水平边
  - **左边 (left edge)**：非水平且位于三角形左侧的边（三角形可以有一个或两个左边）



如何加速三角形的绘制过程?

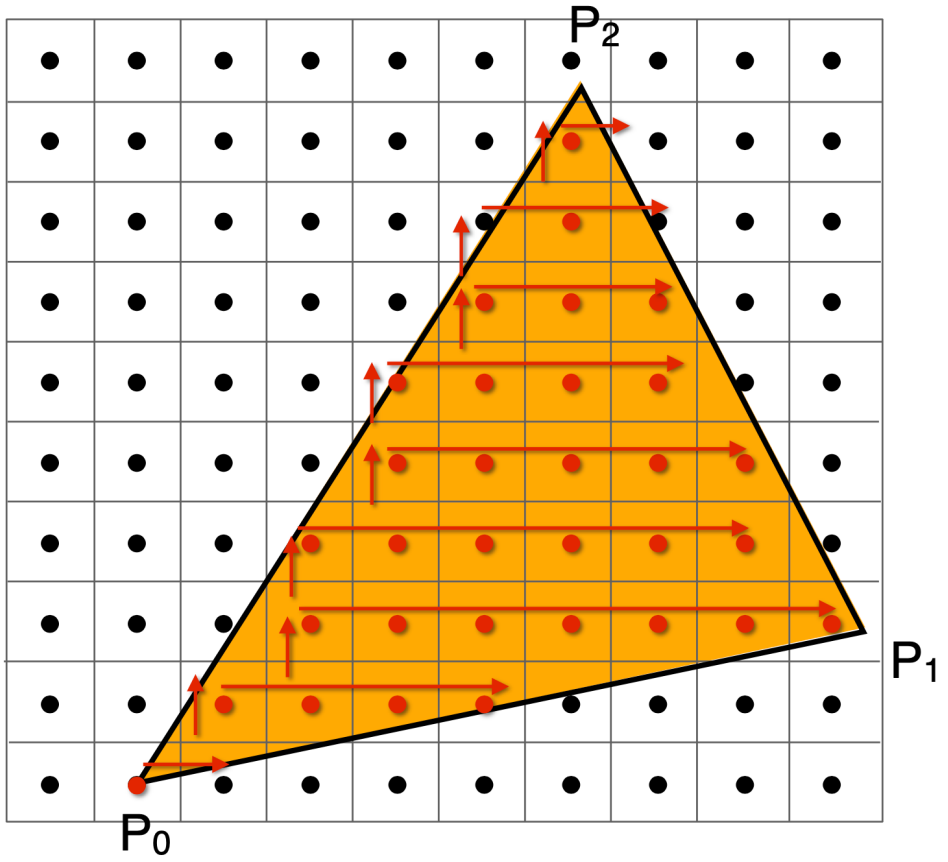
包围盒加速三角形遍历



Always efficient?

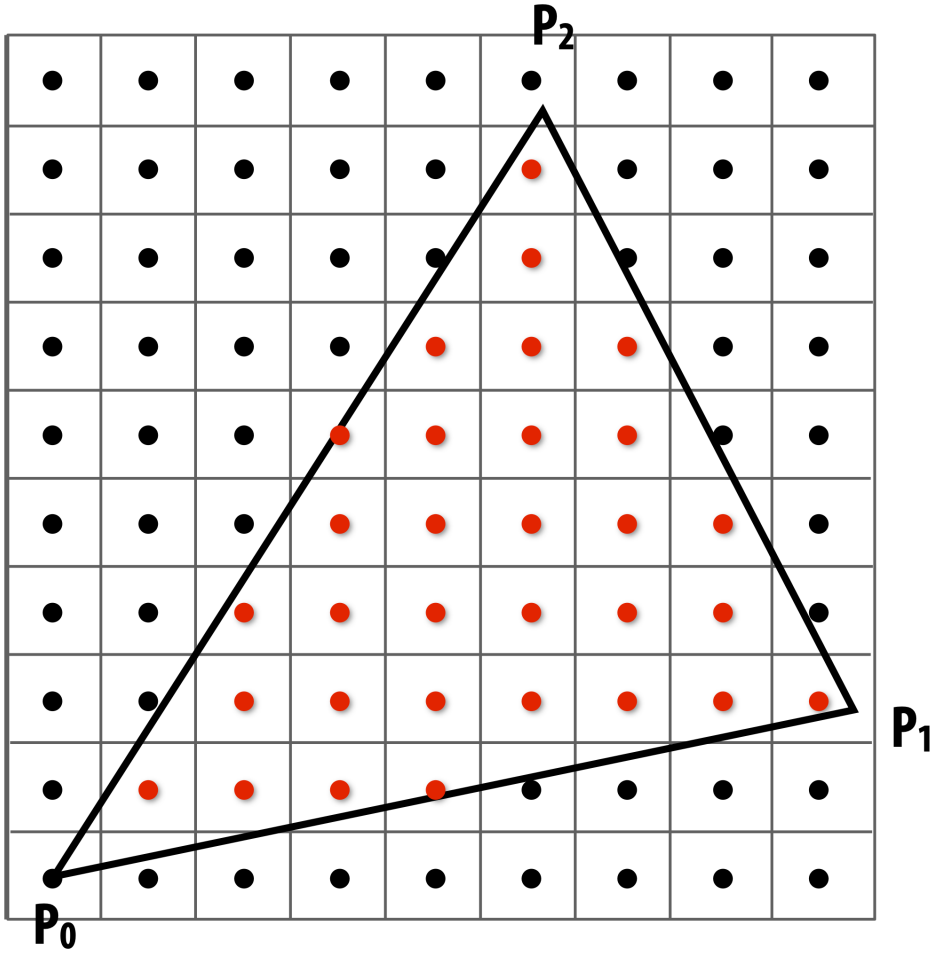
增量三角形遍历

- 通过“增量式”技术提高半平面检查的计算效率
- 增量式方法以一种与数据在内存中的排列方式相一致的模式访问内存，这可以大大提高渲染的速度
  - backtrack
  - zigzag
  - Hilbert/Morton curves



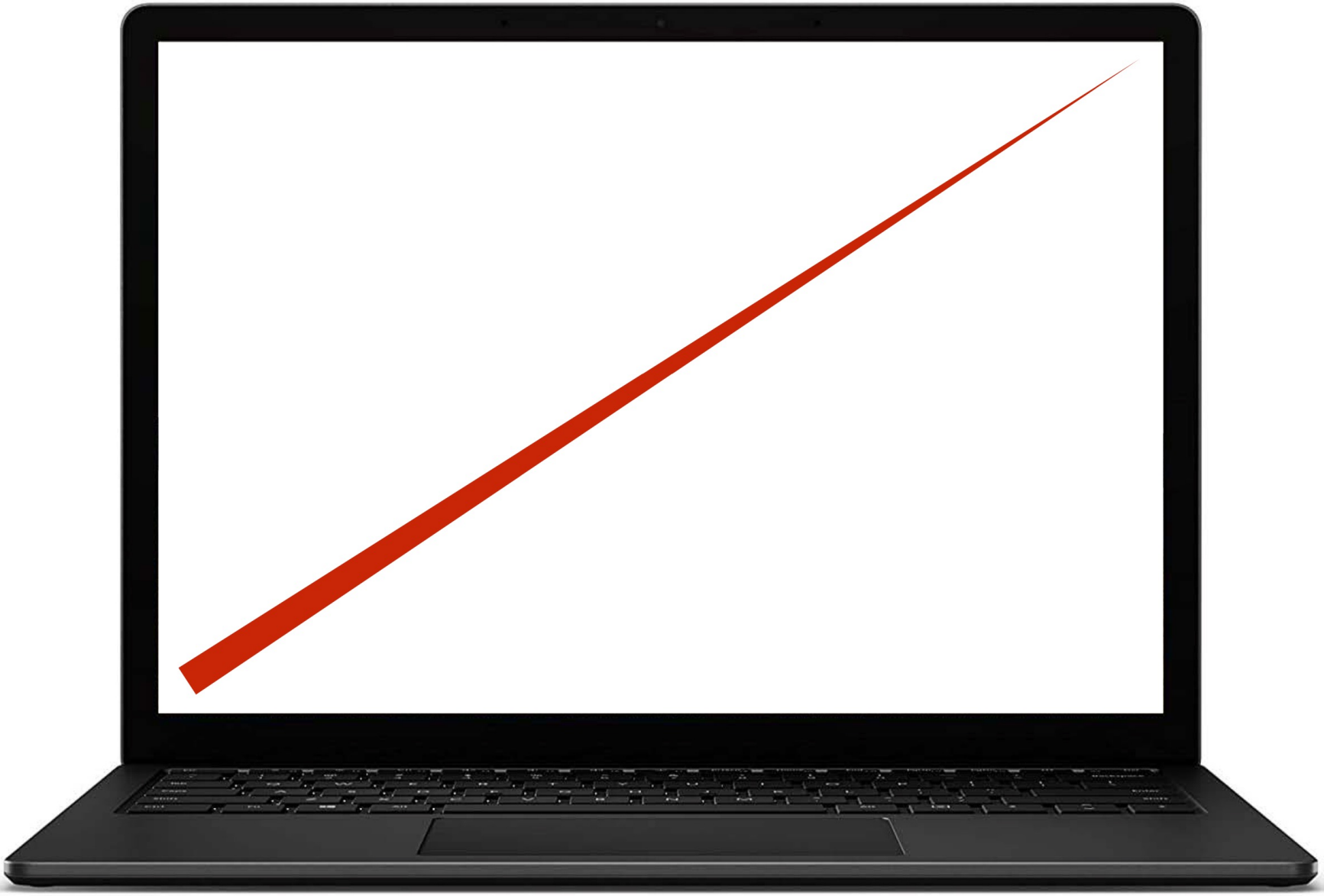
并行覆盖检测

- 增量式遍历是串行的；现代硬件支持高度并行计算
- 替代方案：并行测试三角形“包围盒”内的所有样本点
- 大量的并行可以抵消额外的测试（大多数三角形覆盖许多样本，特别是超采样时）
- 现代图形处理单元（GPU）具有专用硬件，可以高效地进行 point-in-triangle 测试



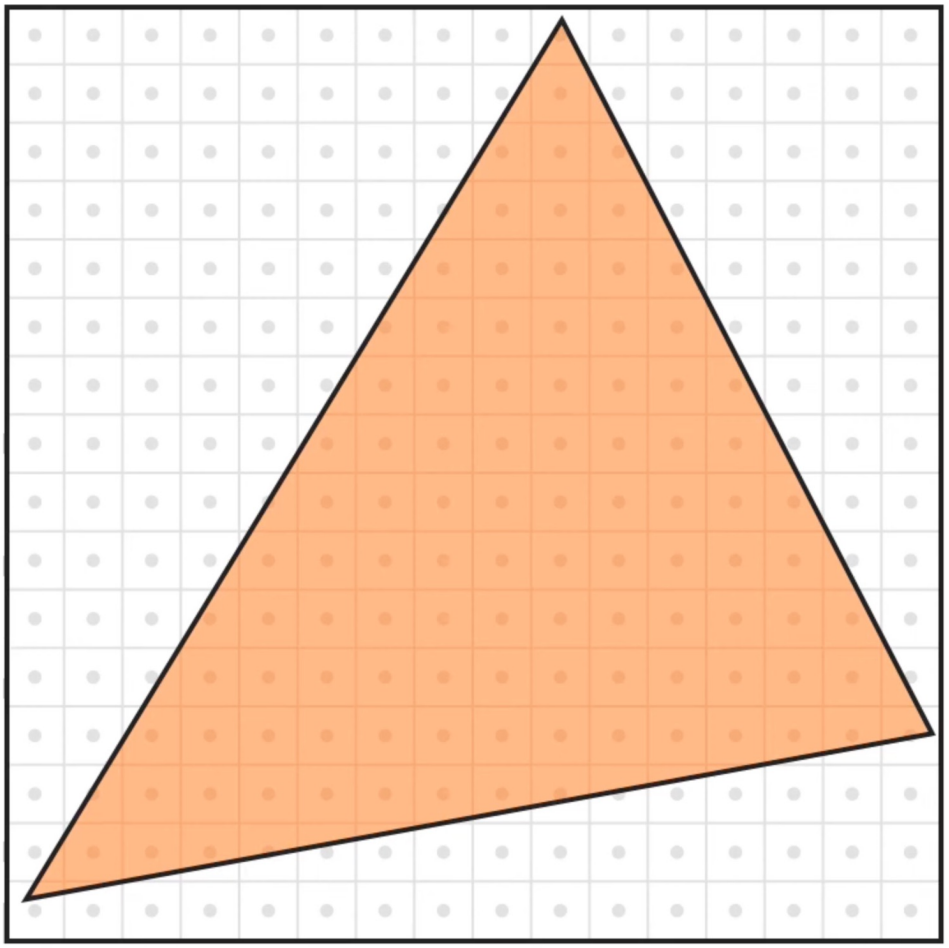
Q：在什么样的场景中，初级的并行策略仍然非常低效？

Naïve approach can be (very) wasteful…



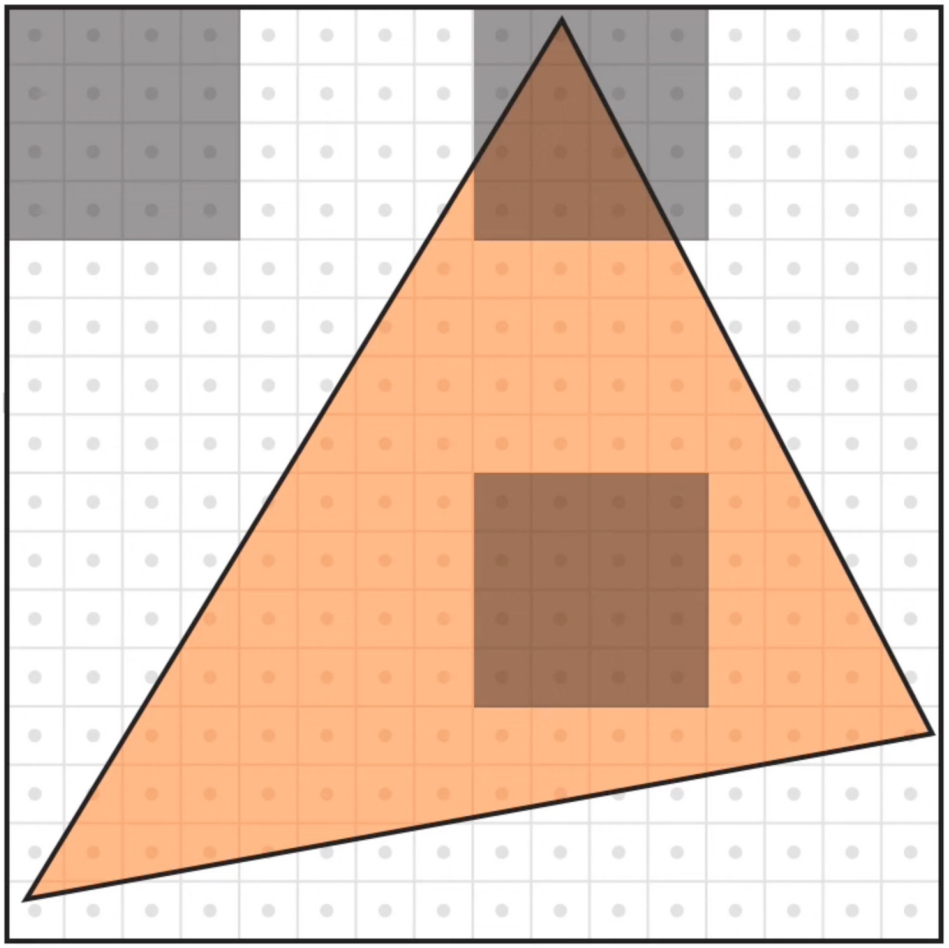
混合方法：平铺三角形遍历

- 理念：从粗到细



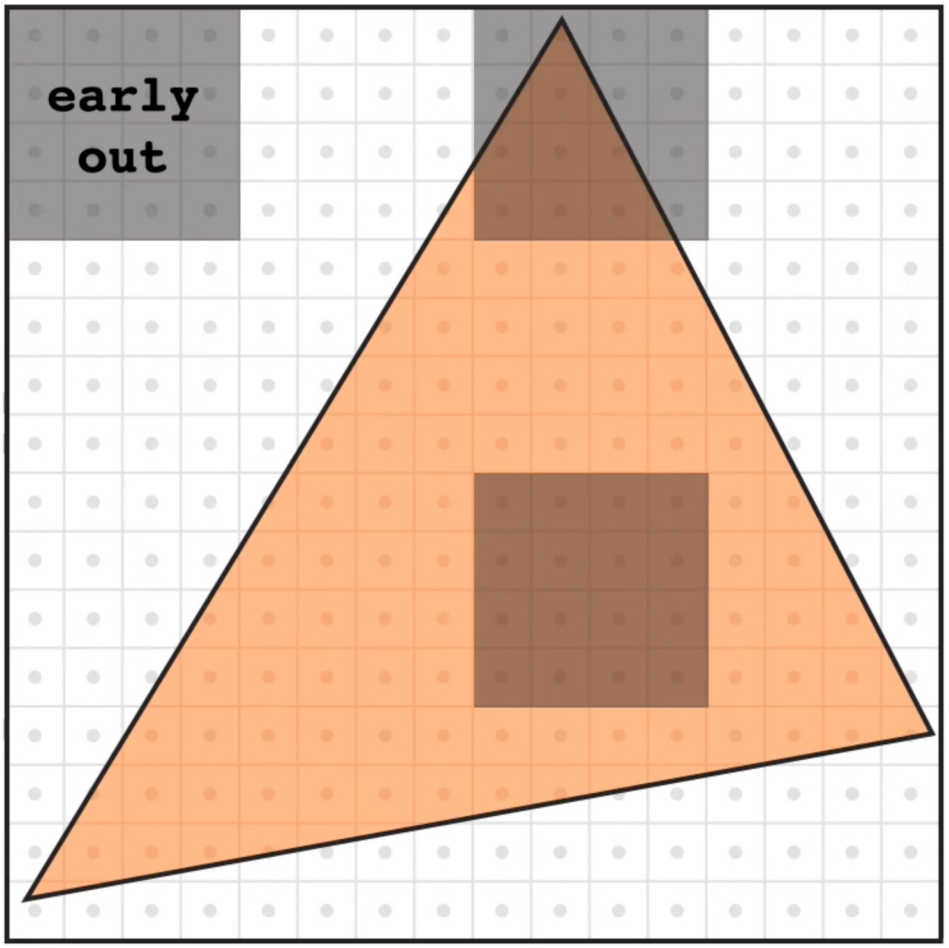
混合方法：平铺三角形遍历

- 理念：从粗到细
- 首先，检查块是否与三角形相交



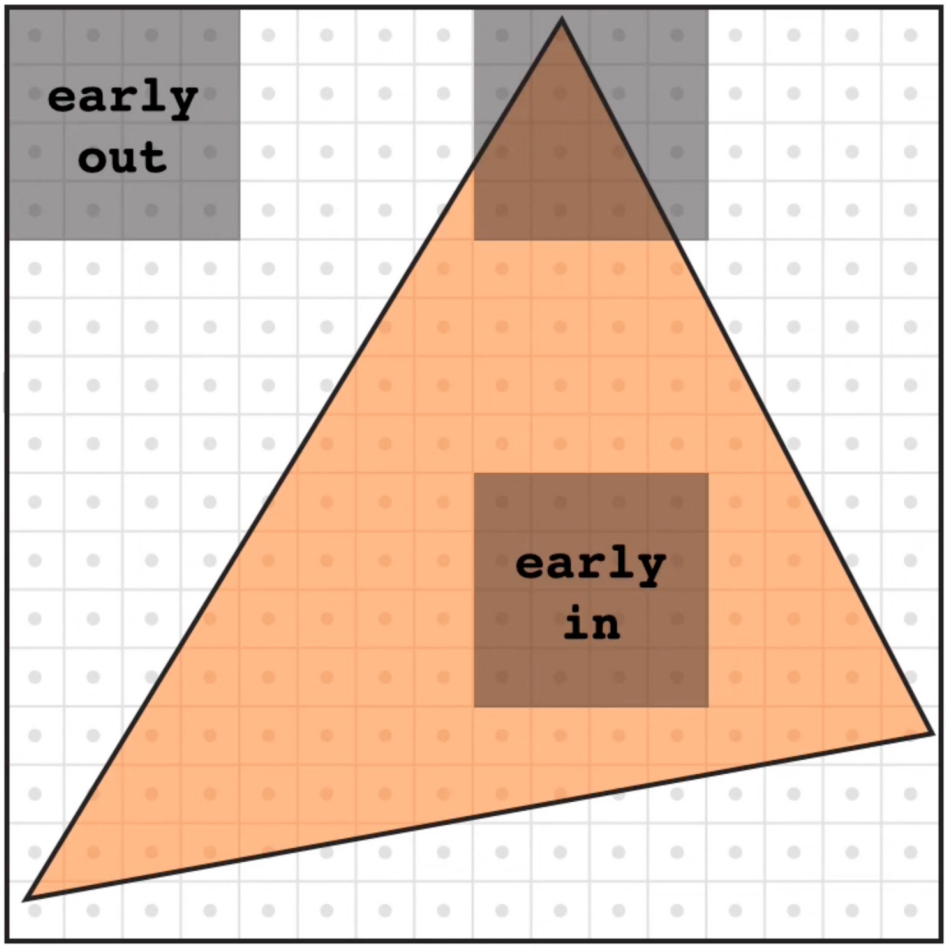
混合方法：平铺三角形遍历

- 理念：从粗到细
- 首先，检查块是否与三角形相交
- 如果没有，完全跳过此块（“提前退出 early out”）



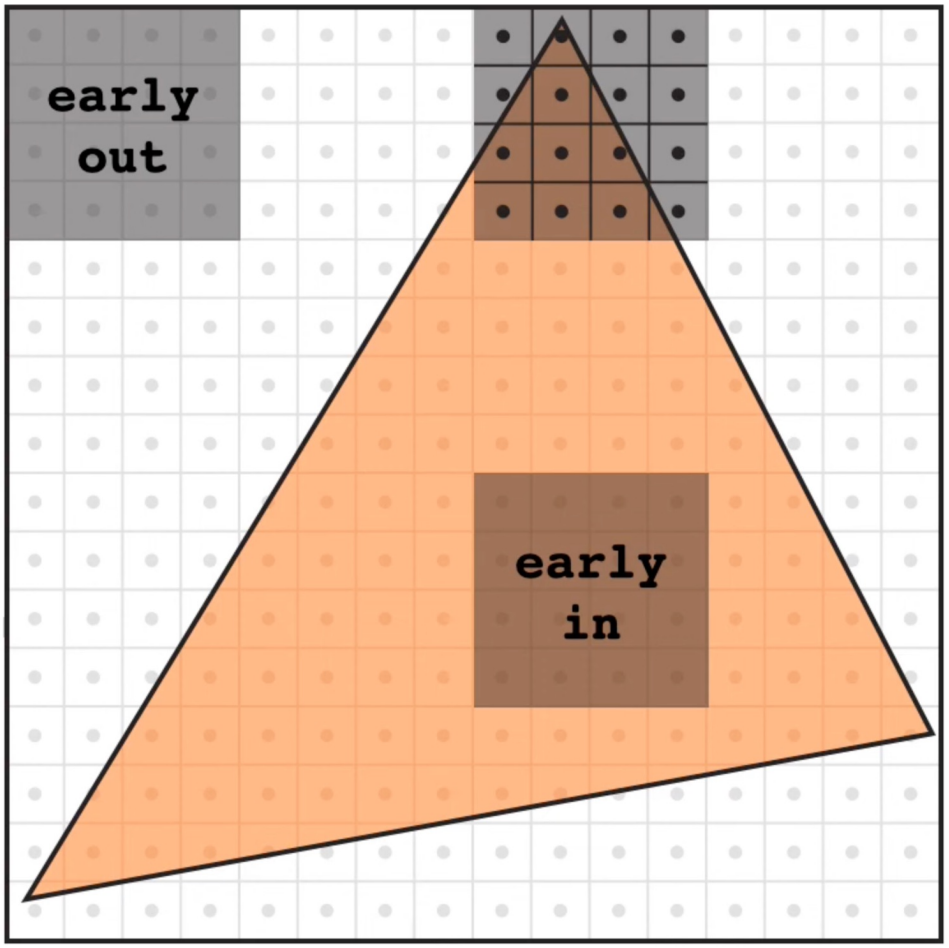
混合方法：平铺三角形遍历

- 理念：从粗到细
- 首先，检查块是否与三角形相交
- 如果没有，完全跳过此块（“提前退出 early out”）
- 如果块包含在三角形内，则块内所有点都被覆盖了（“早期进入 early in”）

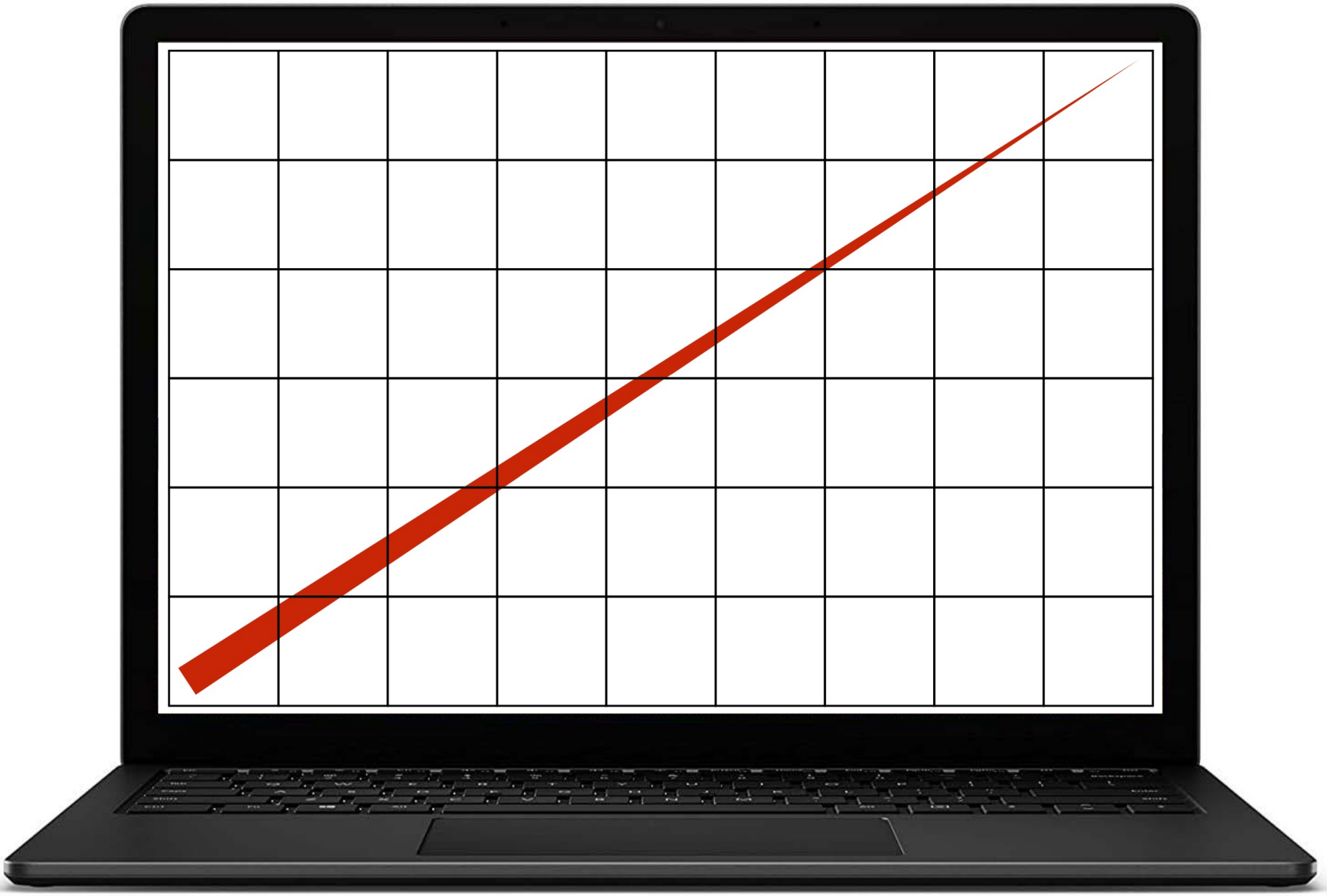


混合方法：平铺三角形遍历

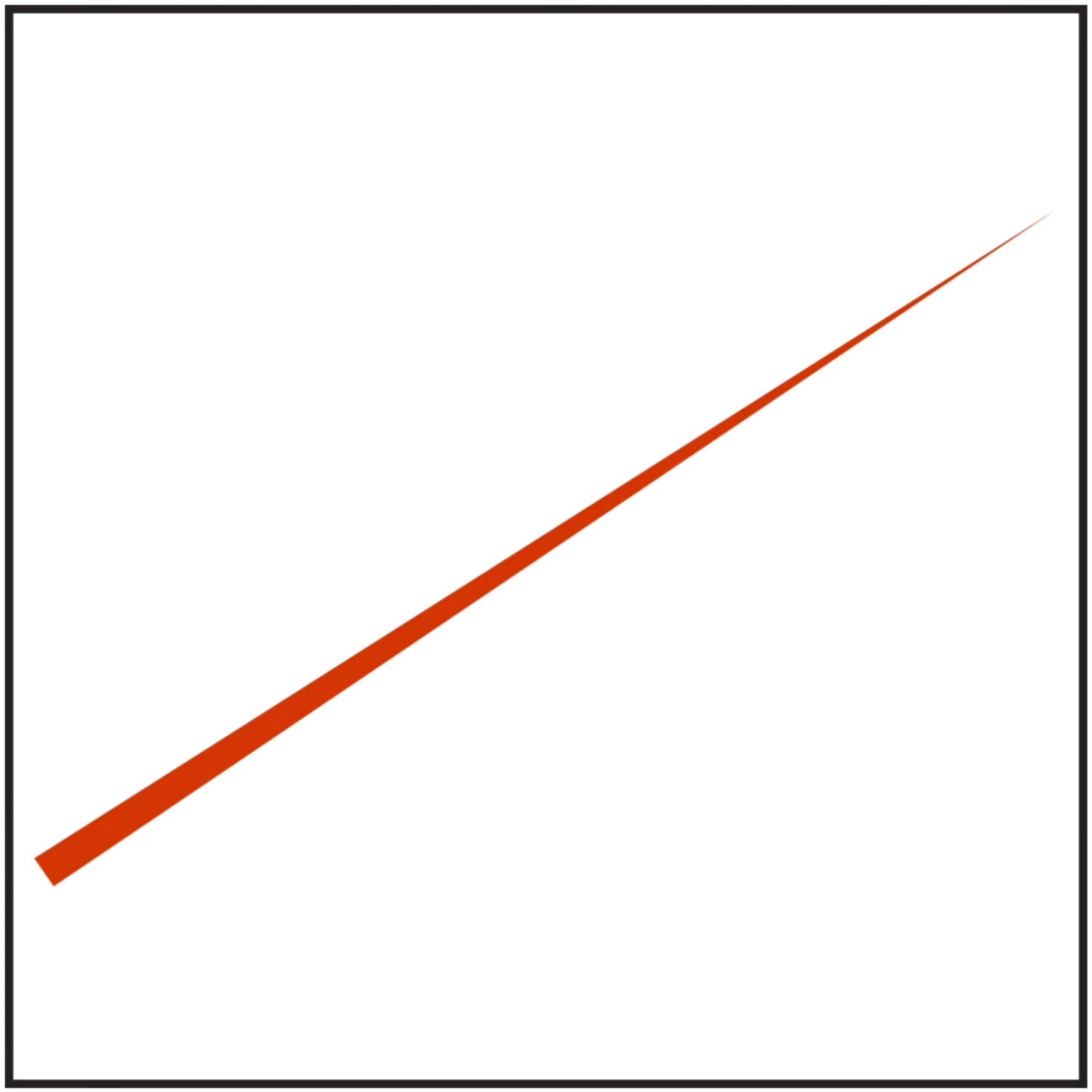
- 理念：从粗到细
- 首先，检查块是否与三角形相交
- 如果没有，完全跳过此块（“提前退出 early out”）
- 如果块包含在三角形内，则块内所有点都被覆盖了（“早期进入 early in”）
- 否则，并行测试块中的每个采样点



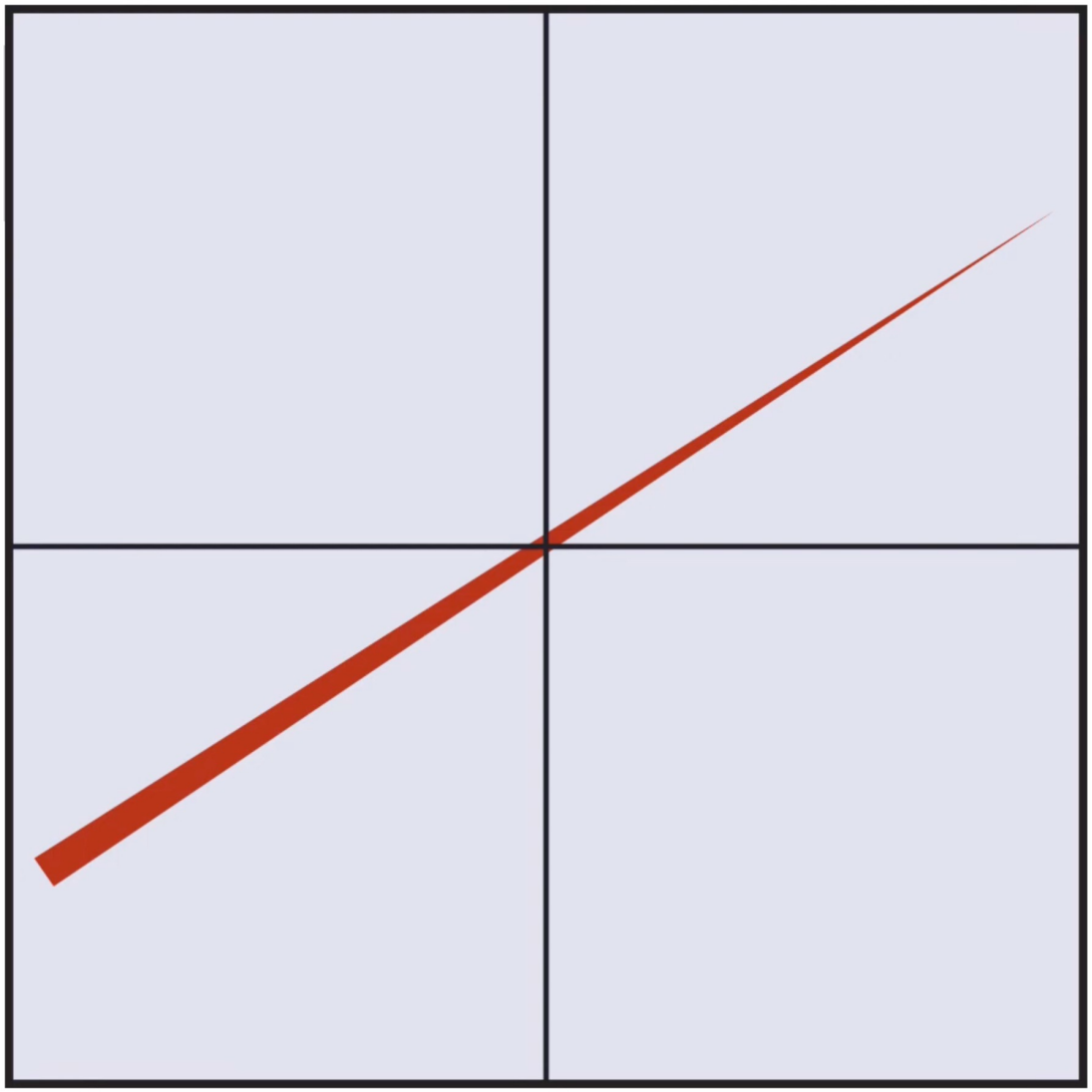
Can we do even better for this example?



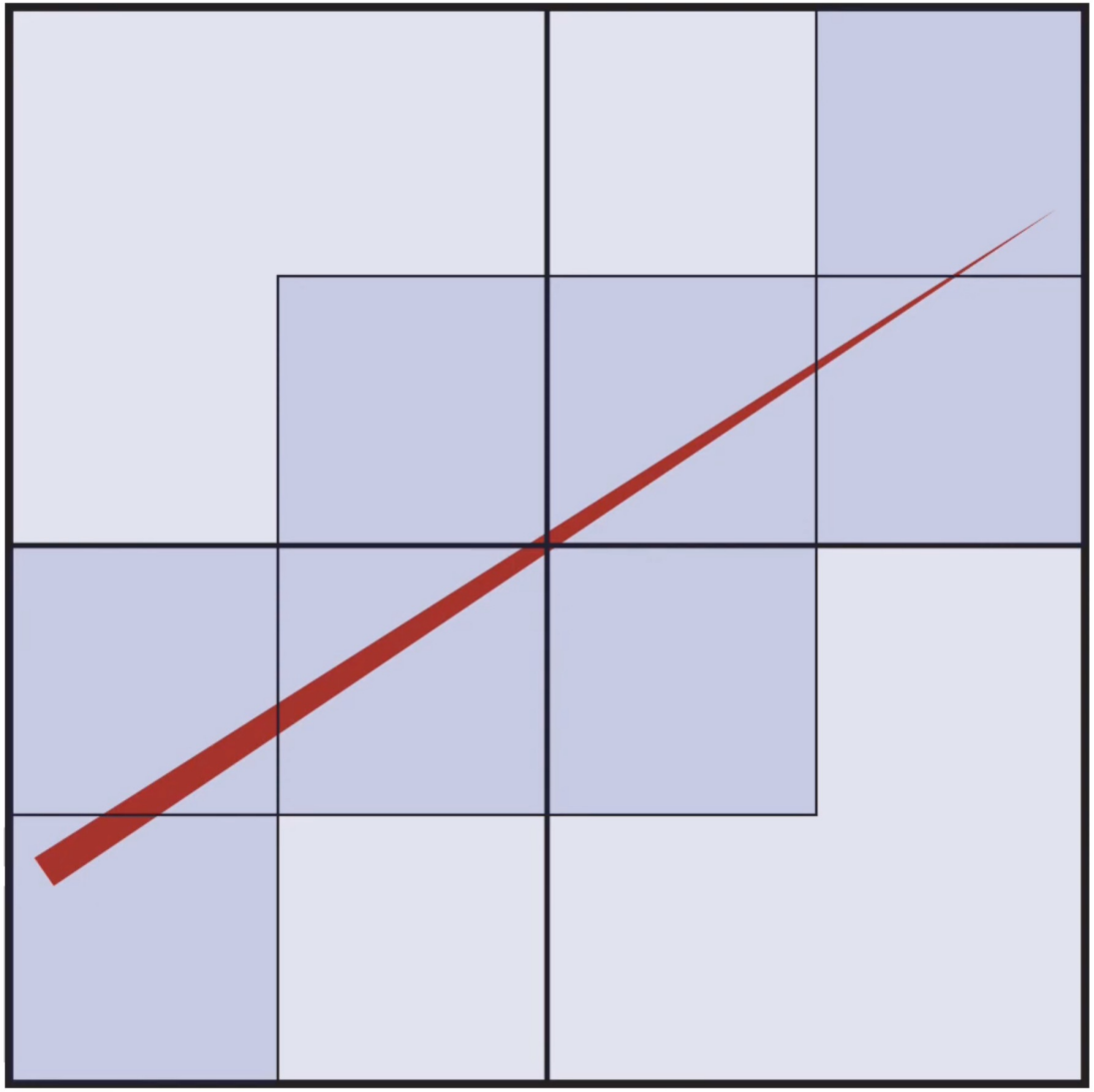
分层策略 Hierarchical strategies



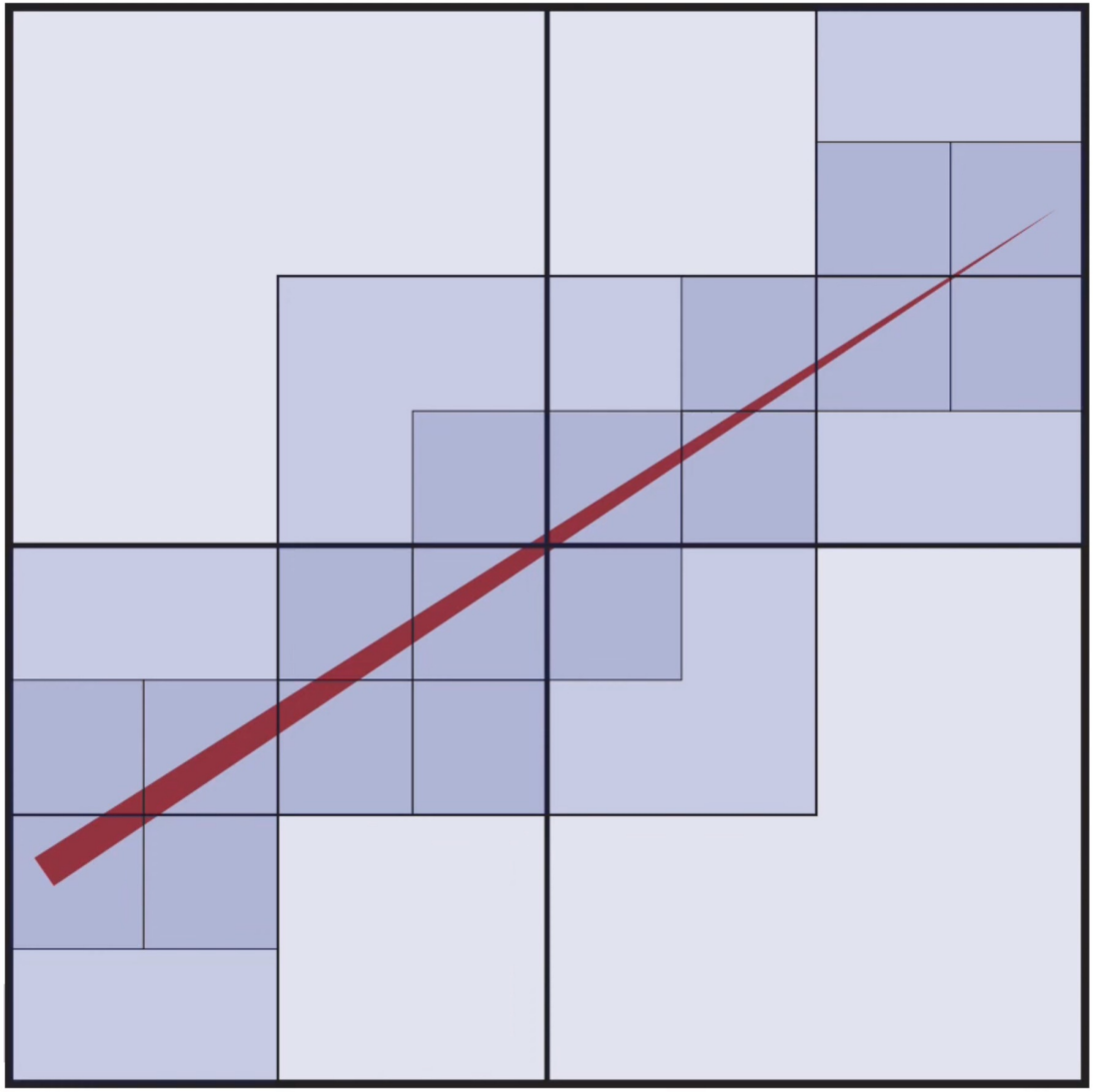
分层策略 Hierarchical strategies



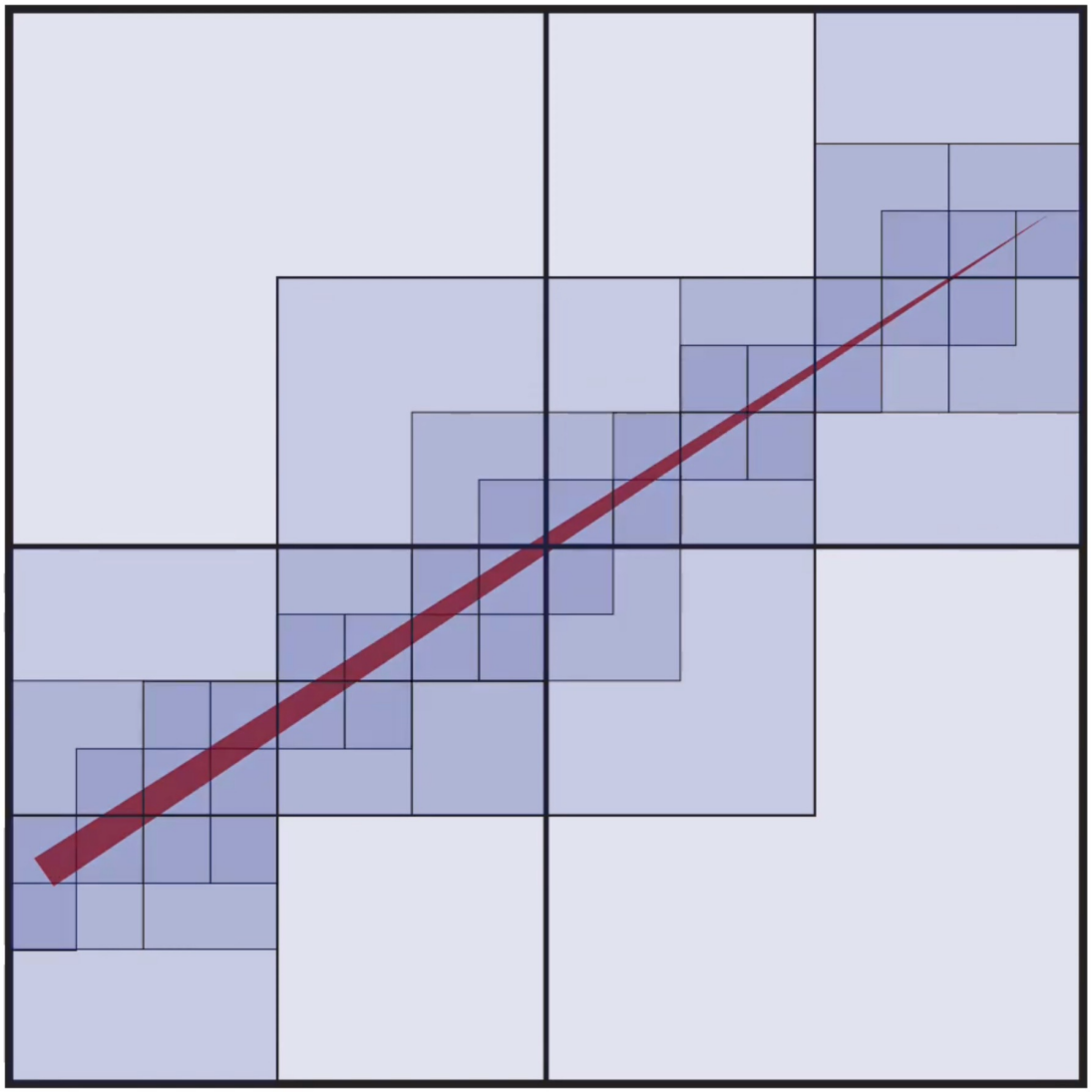
分层策略 Hierarchical strategies



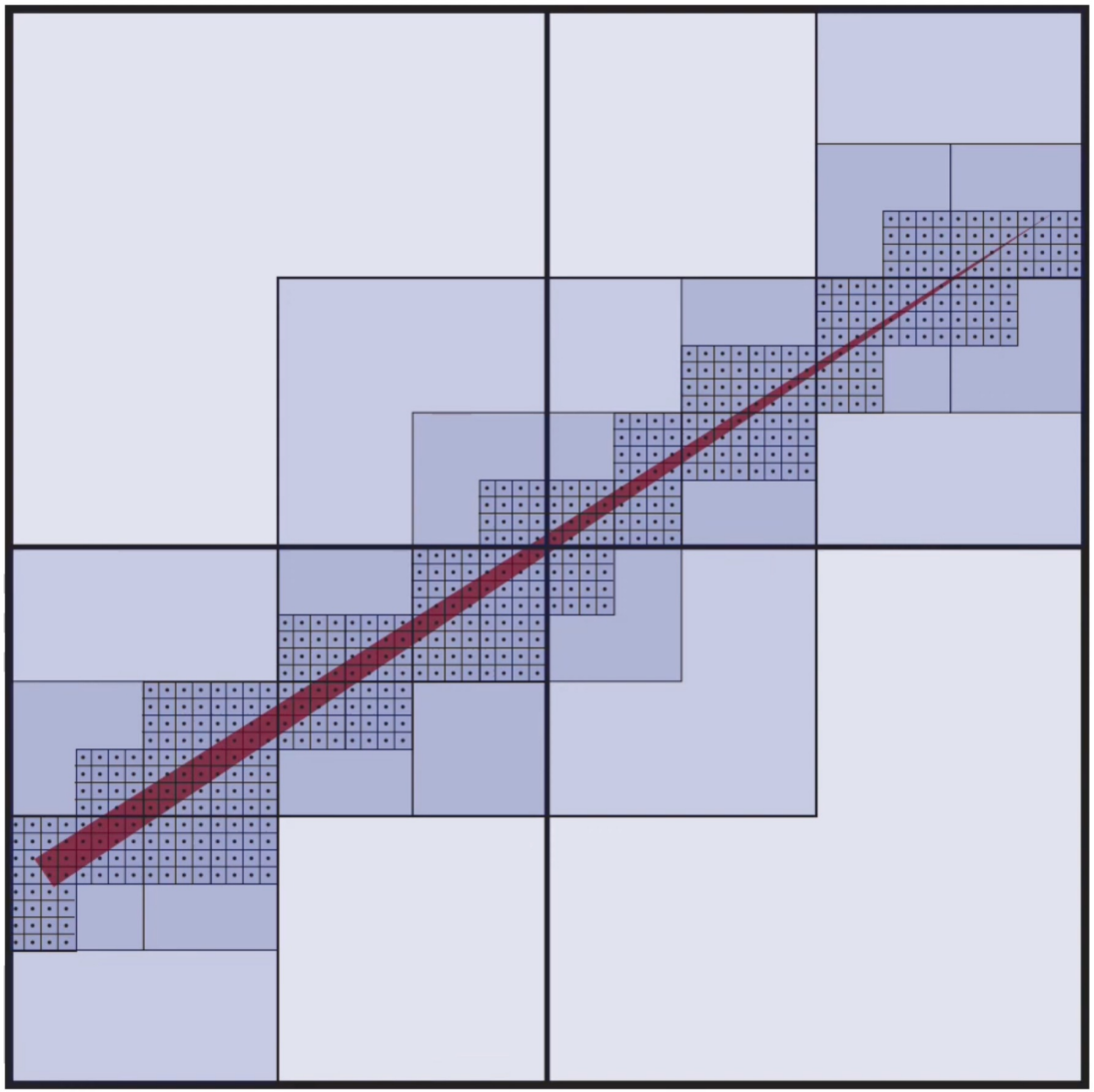
分层策略 Hierarchical strategies



分层策略 Hierarchical strategies



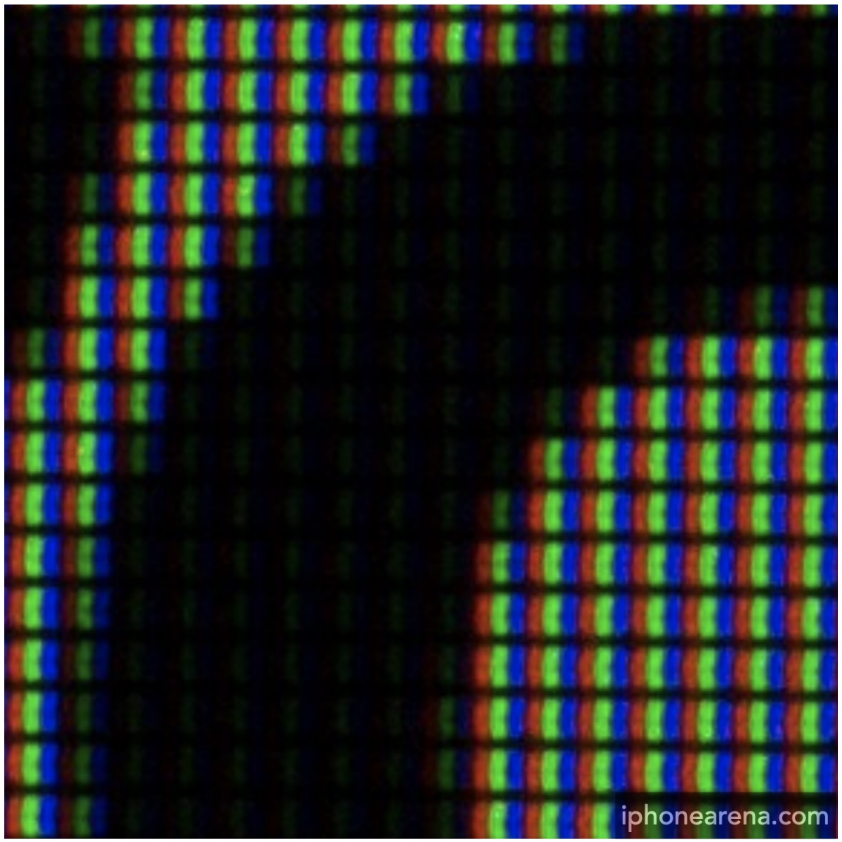
分层策略 Hierarchical strategies



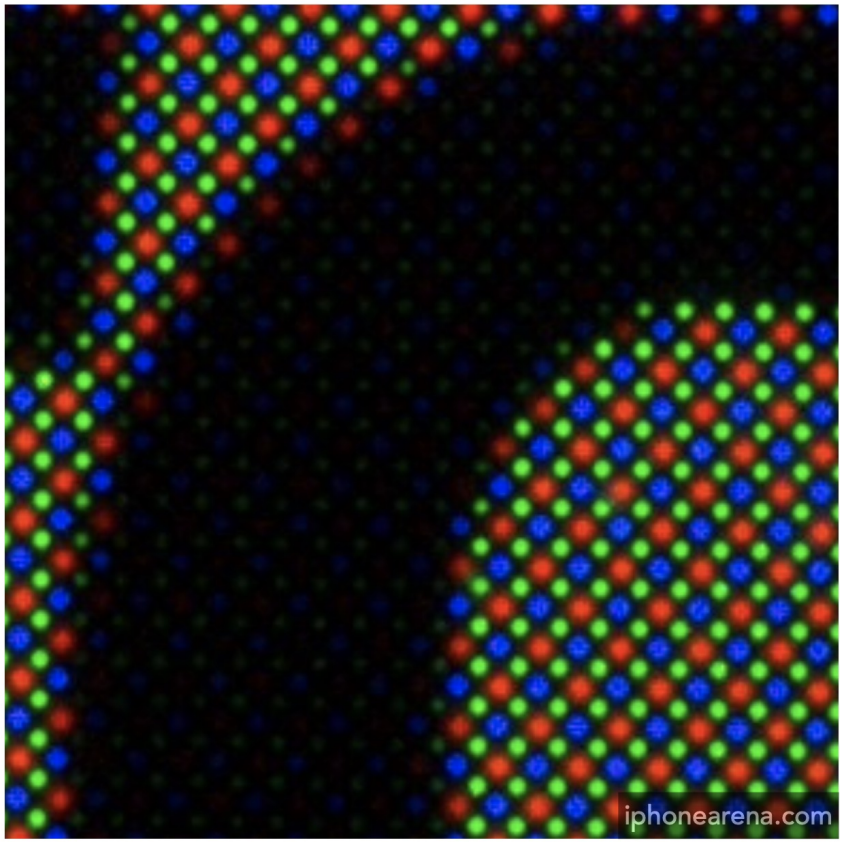
Q: Better way to find finest blocks? A: Maybe: incremental traversal!

显示器上的信号重构

LCD 屏幕像素特写



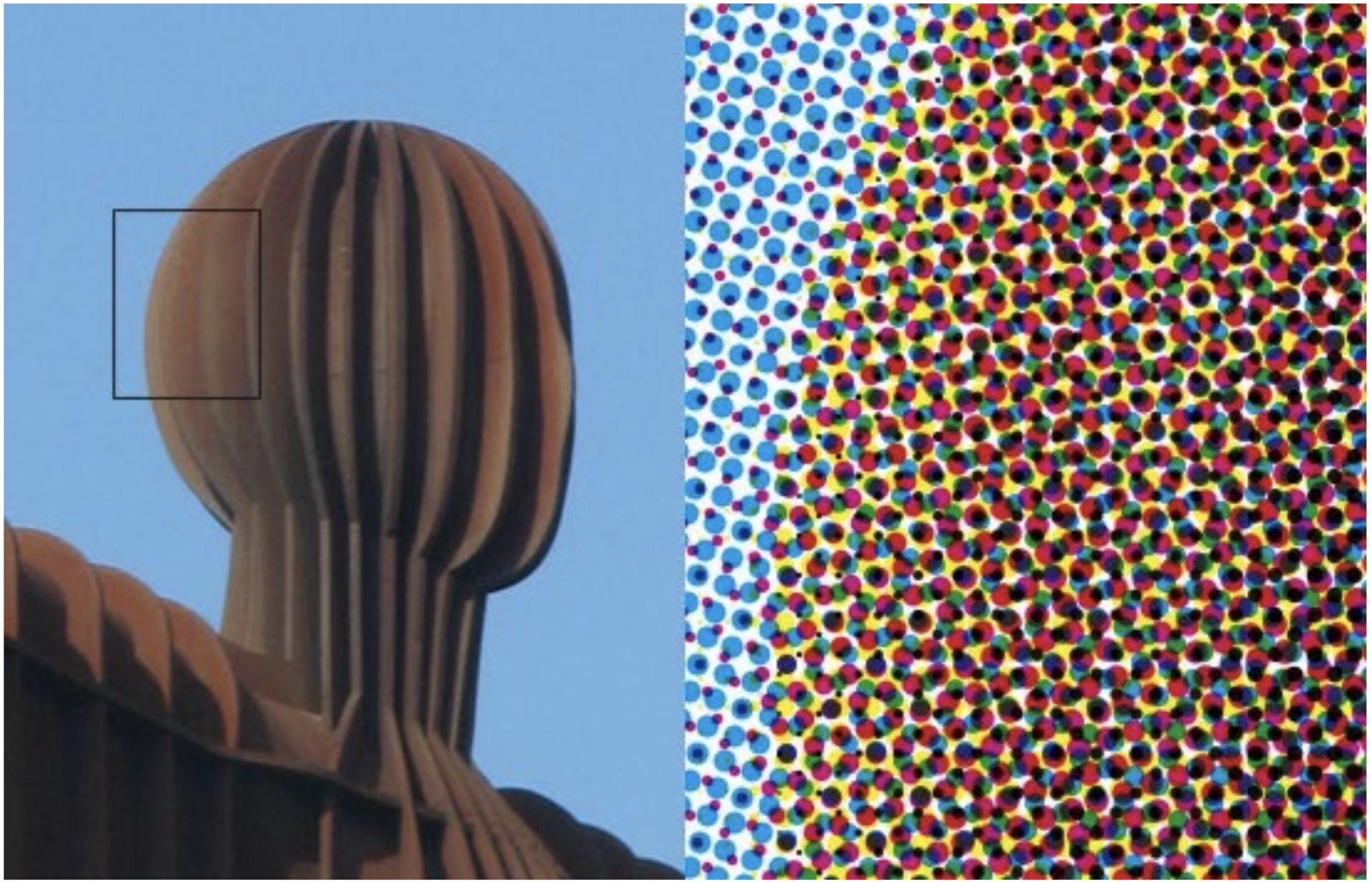
iPhone 6S



Galaxy S5

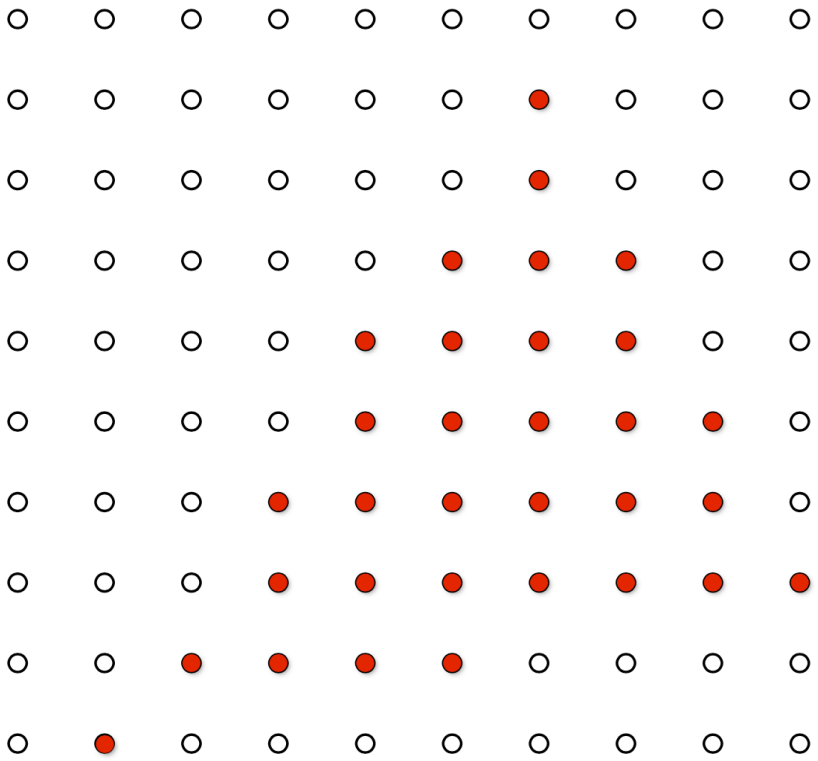
Notice R,G,B pixel geometry! But in this class, we will assume a colored square full-color pixel.

其他显示方法呢?

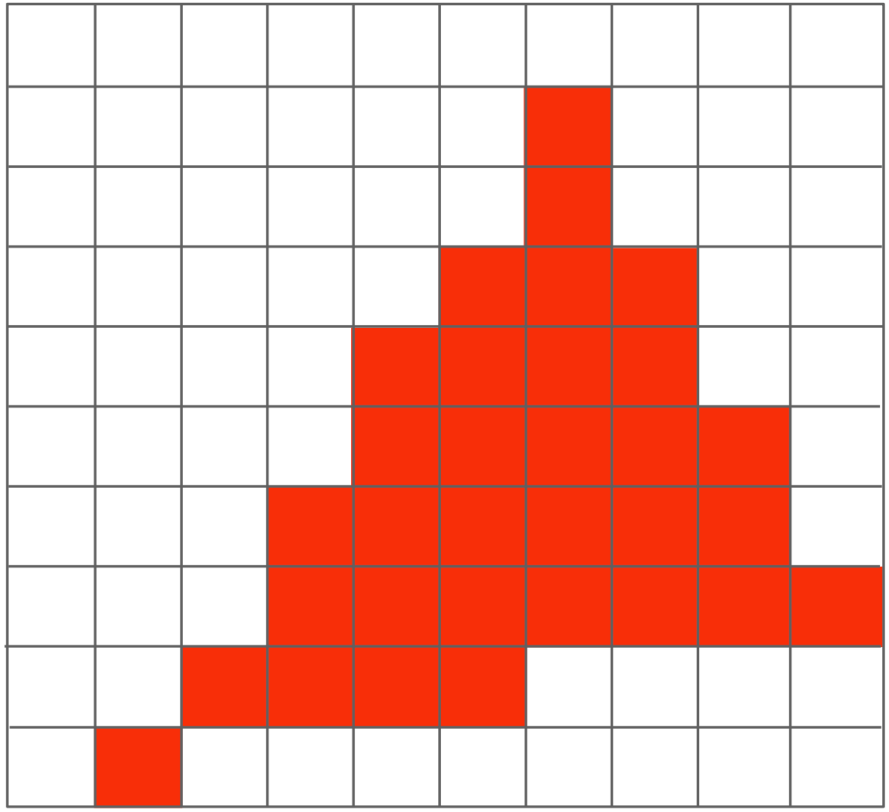


Color print: observe half-tone pattern

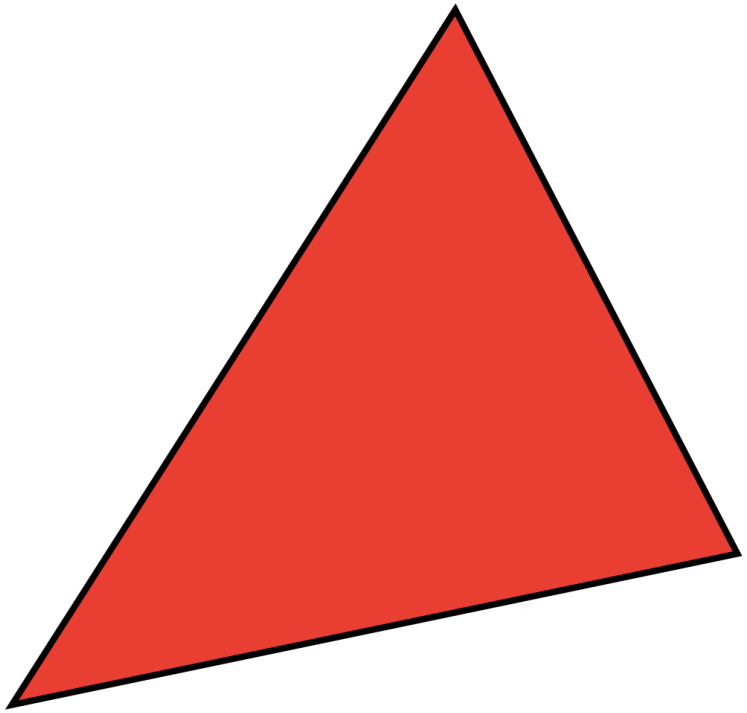
如果我们向显示器发送如下采样信号



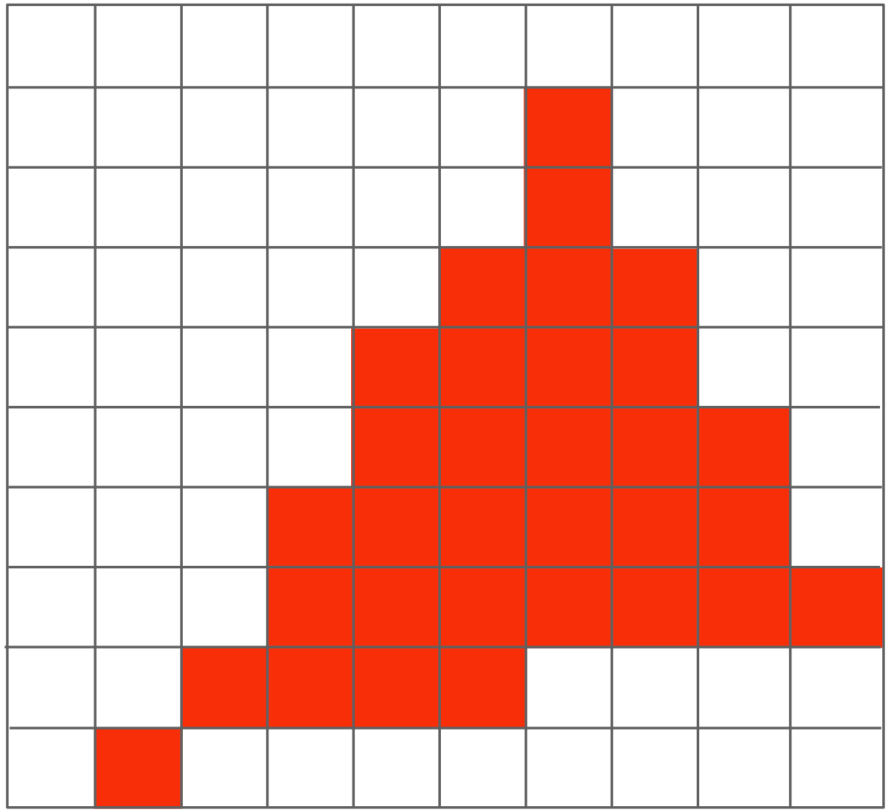
显示器将展现如下图片



而我们想要的是如下的图片

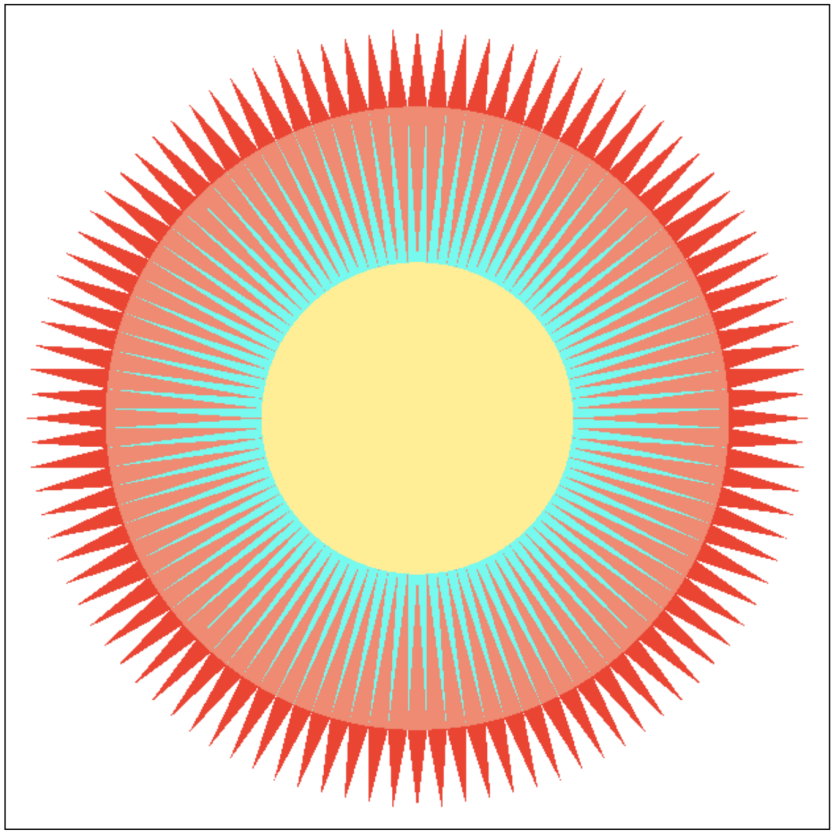


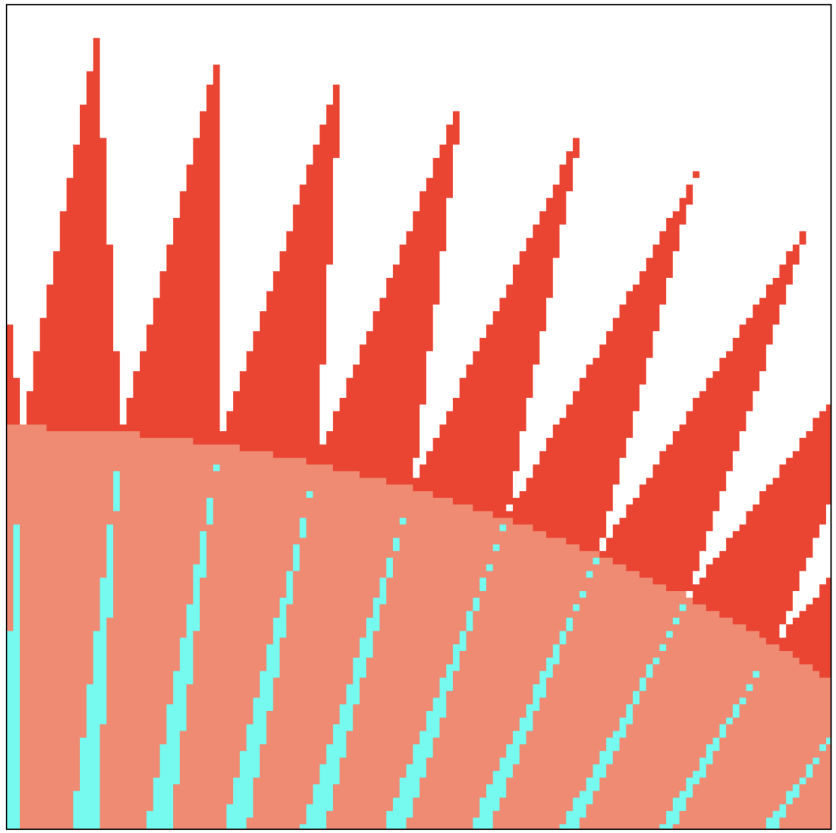
所以，这幅图有什么问题?



锯齿 Jaggies!

锯齿 Jaggies





我们能否做的更好?

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