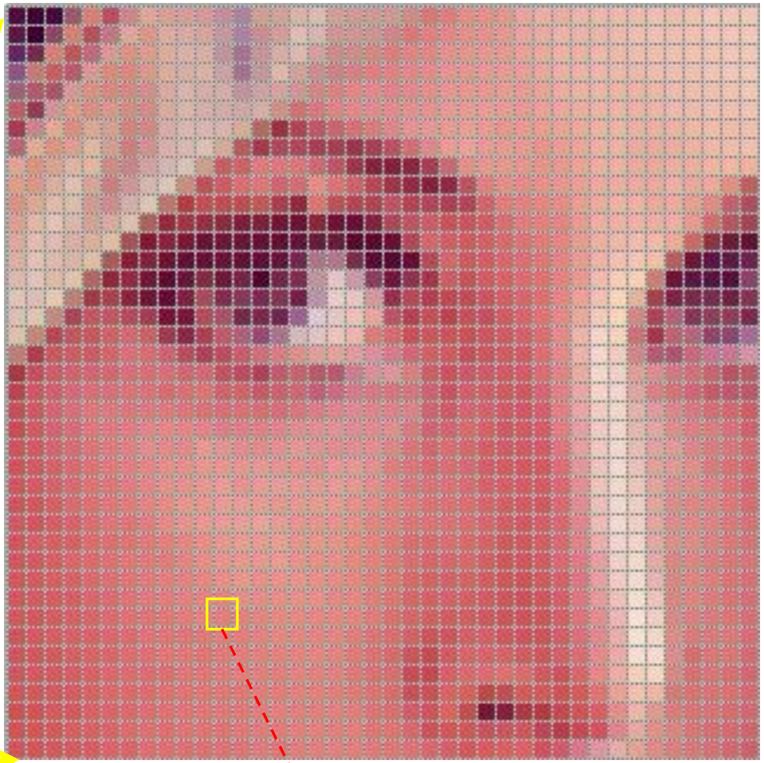
第二章 传统图形学基础

2.2 图元绘制

- 2.2.1 数字图像及光栅显示器
- 2.2.2 2D图形
- 2.2.3 2D图形的光栅化
  - 2.2.3.1 线段的光栅化
  - 2.2.3.2 多边形的光栅化

2.2.1 2D数字图像及光栅显示器

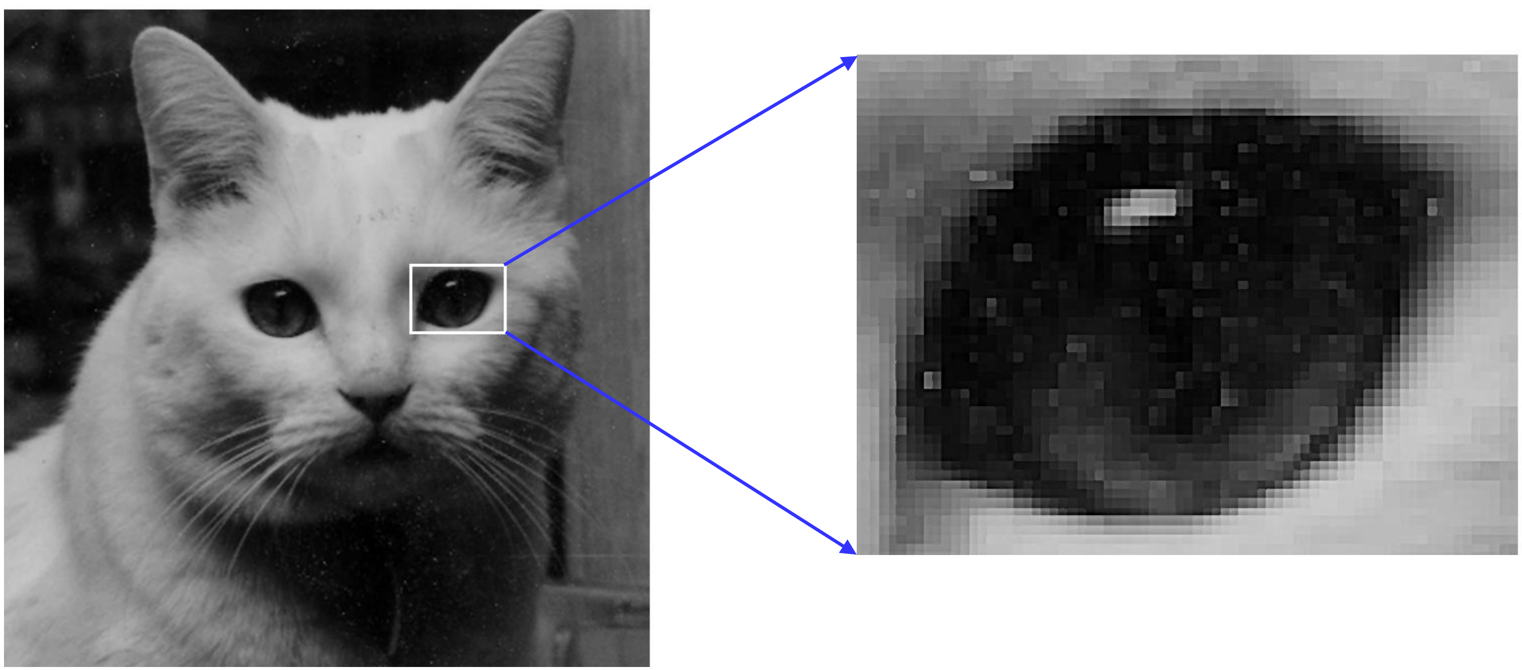
2D图像(image)



0.6 R + 0.3 G + 0.1 B

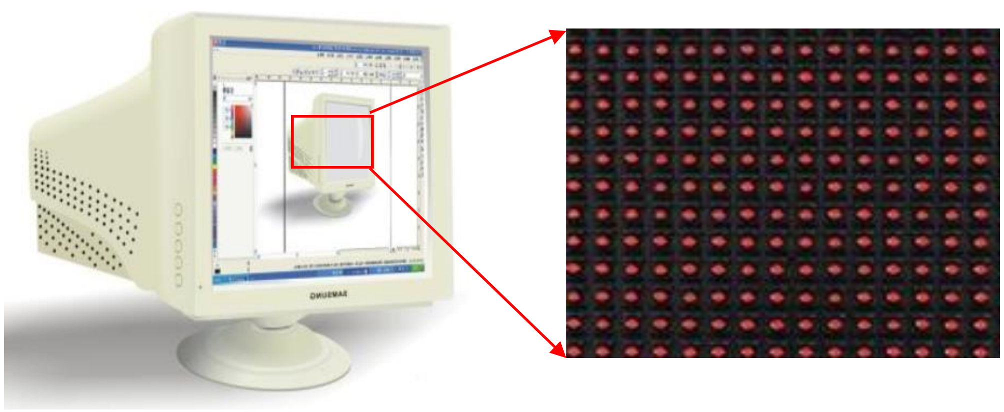
像素与光栅

- 像素（pixels）： 图像的基本单元
- 光栅（raster）： 像素的阵列



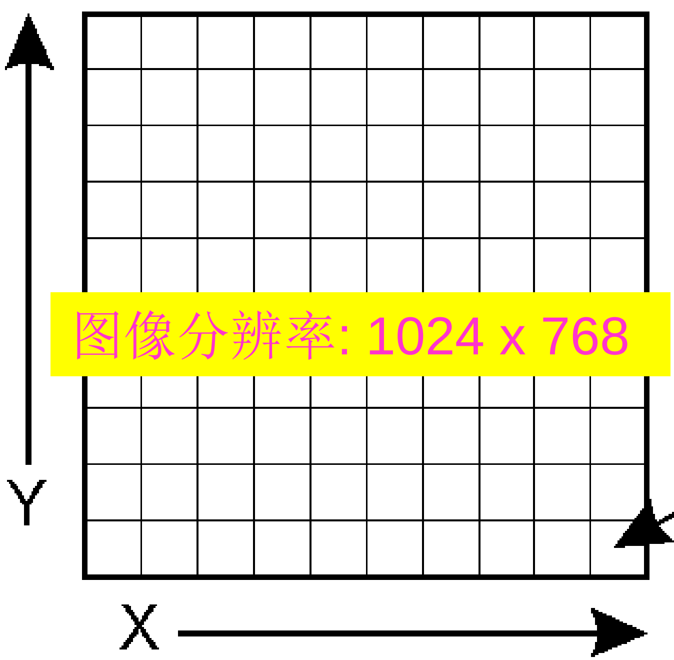
图像显示器

- 1946年，美国宾夕法尼亚大学，世界上第一台电子计算机问世
- 1959 年，麻省理工学院林肯实验室，第一台阴极射线显像管（CRT）显示器
  - 光栅显示器（rasterized display）

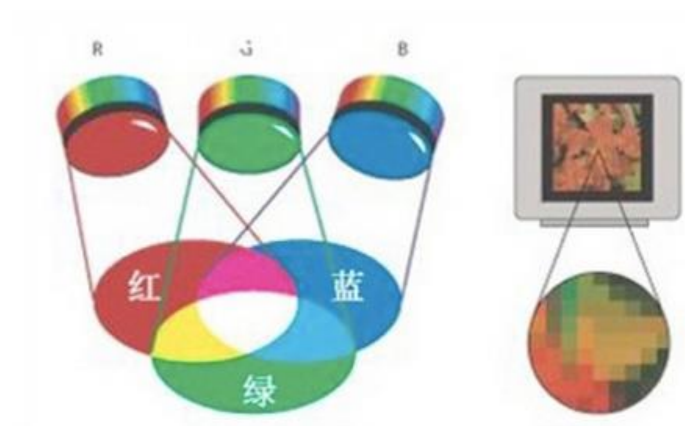


光栅显示屏/数字图像的数学模型（栅格图、点阵图）

- 像素构成的矩阵：连续空间的离散采样
  - 矩阵的每个元素称为片元 (fragment)
  - 片元赋予了颜色后称为像素 (pixel)



pixel **0.6 R + 0.3 G + 0.1 B**

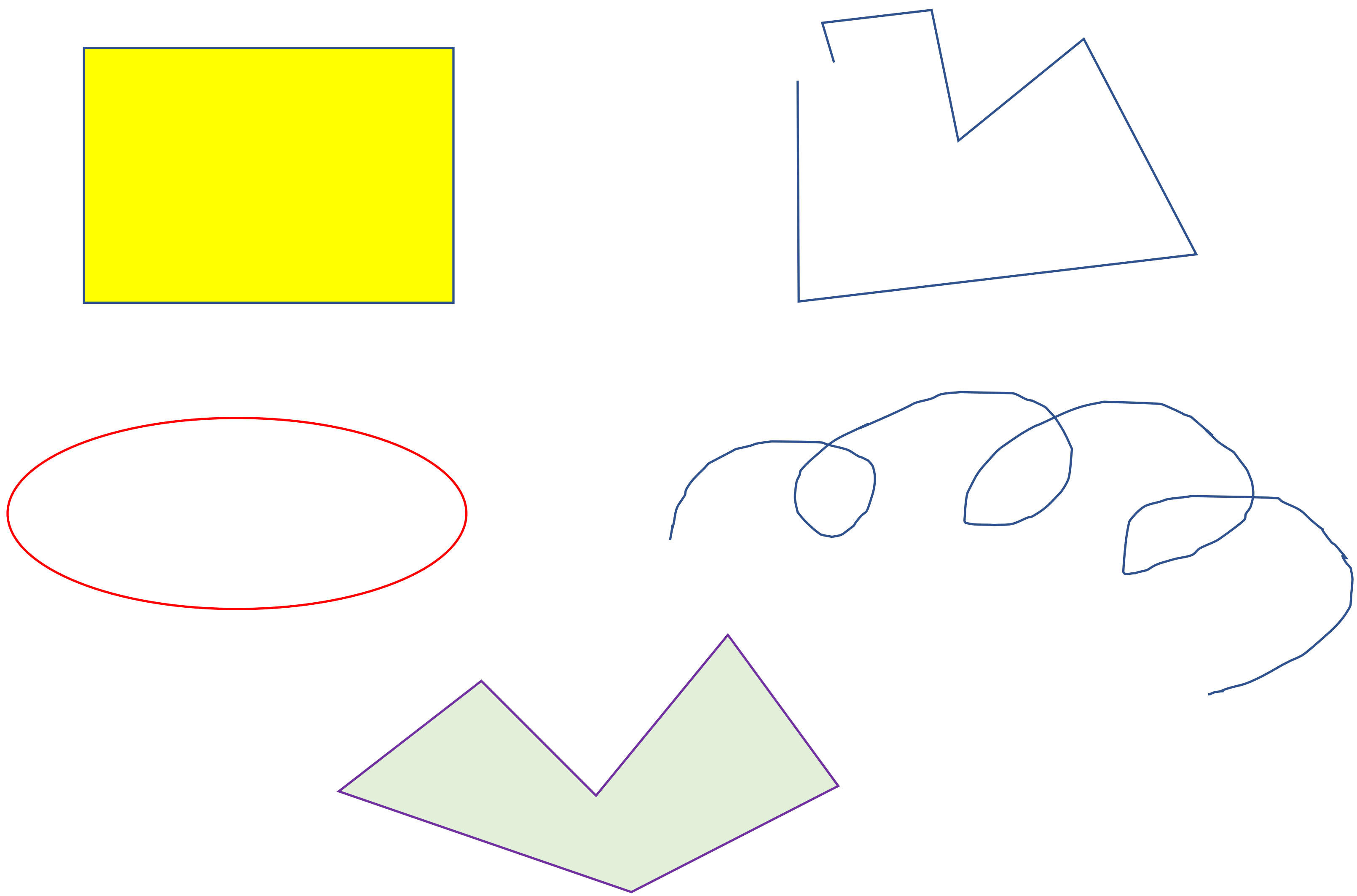


现有显示设备基本都是光栅显示器

低分辨率

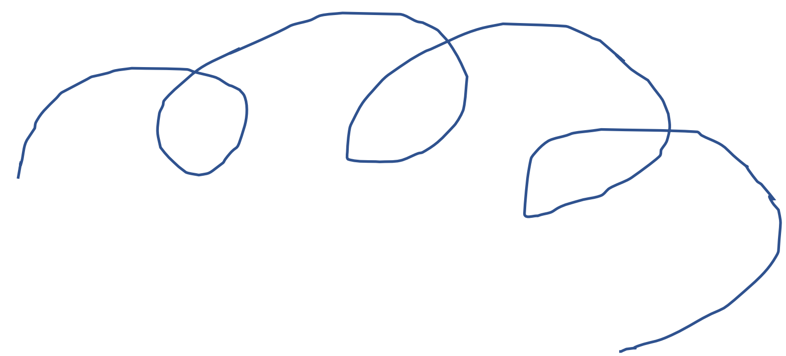
2.2.2 2D图形

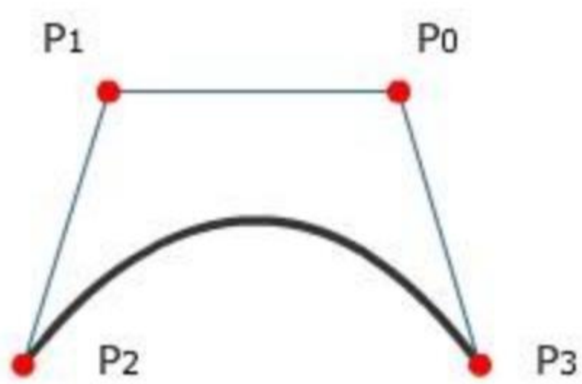
2D图形

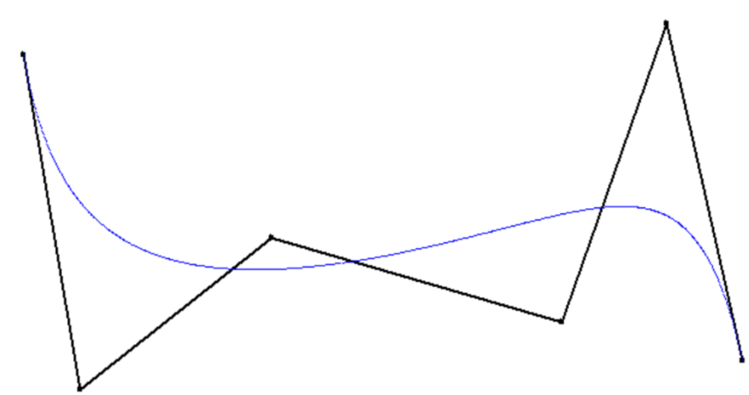


2D图形

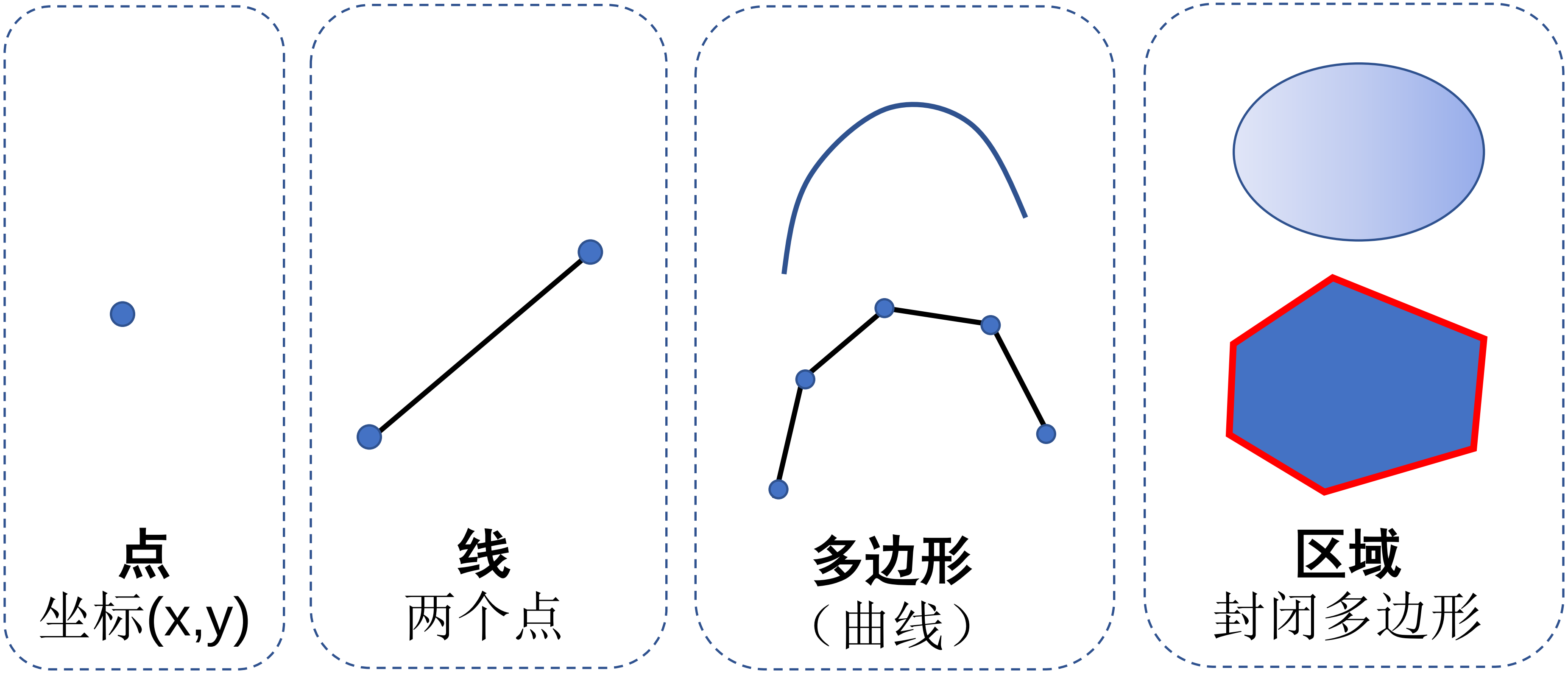
- 有数学坐标表达的，由点、线、区域等要素所构成的几何对象





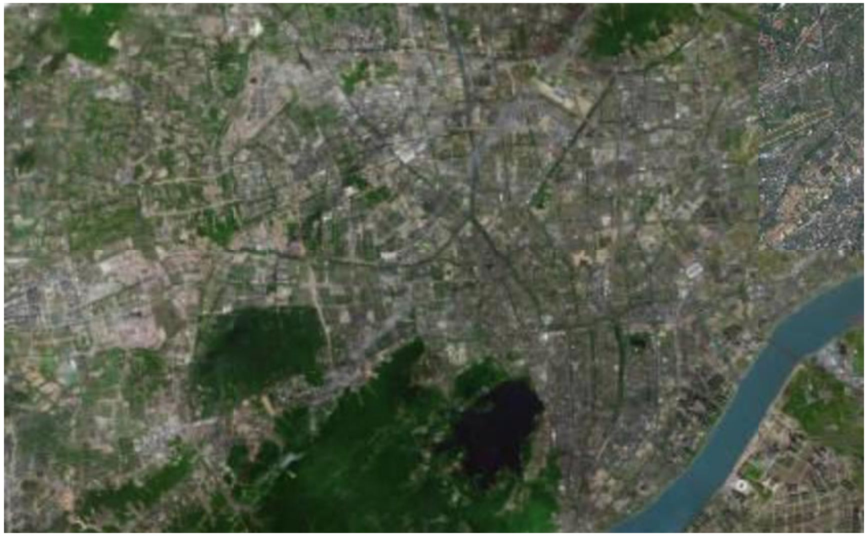


2D图形（矢量）：
具有数学表达的几何对象（点、线、面）



图形对象的属性：大小、线宽、类型、填充、颜色…

举个例子：地图



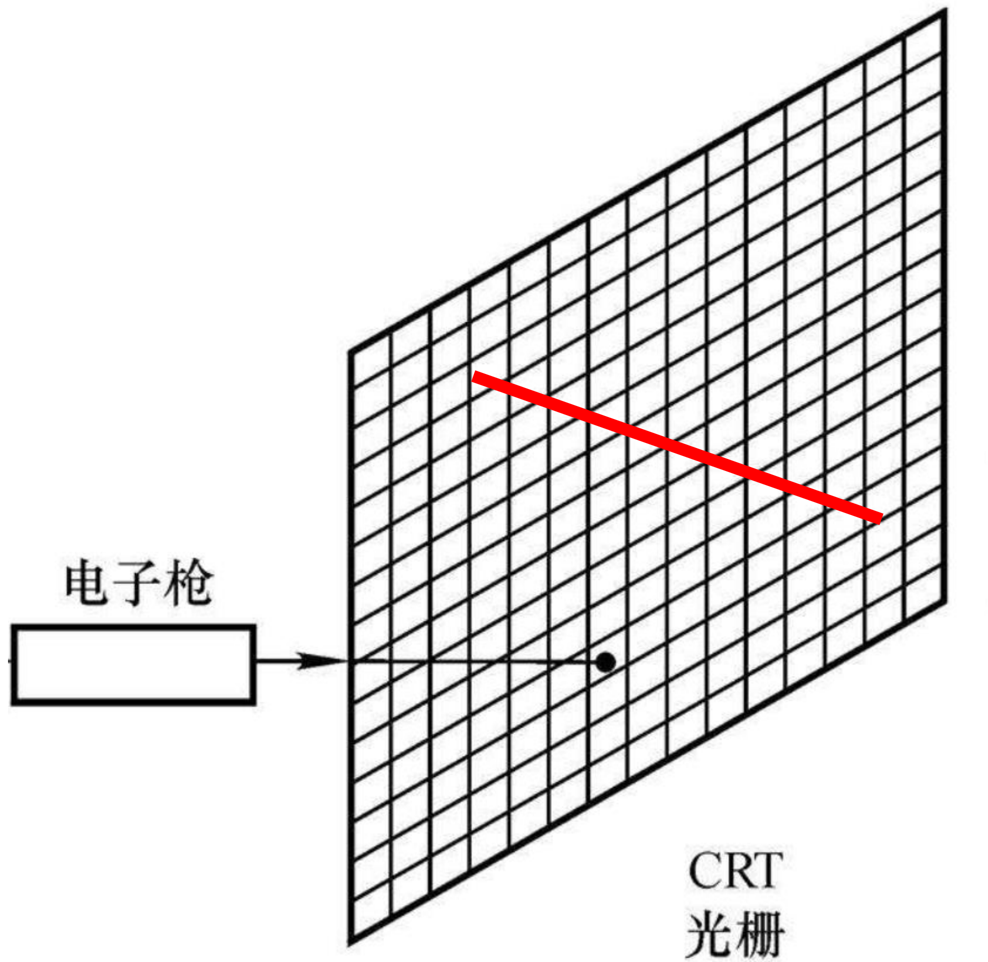
高空影像

矢量图形（路网、地块）

影像+路网

适量地图为什么能无限放大？

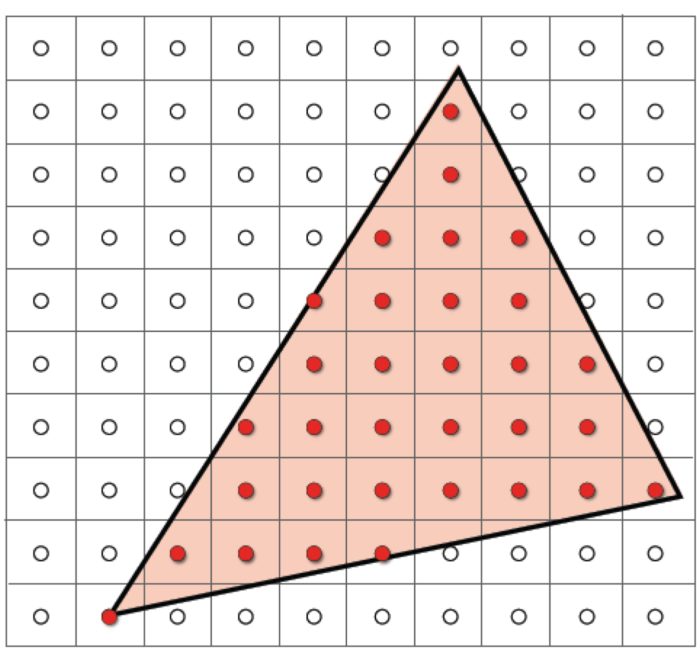
问题：如何在光栅显示器上显示几何图形？

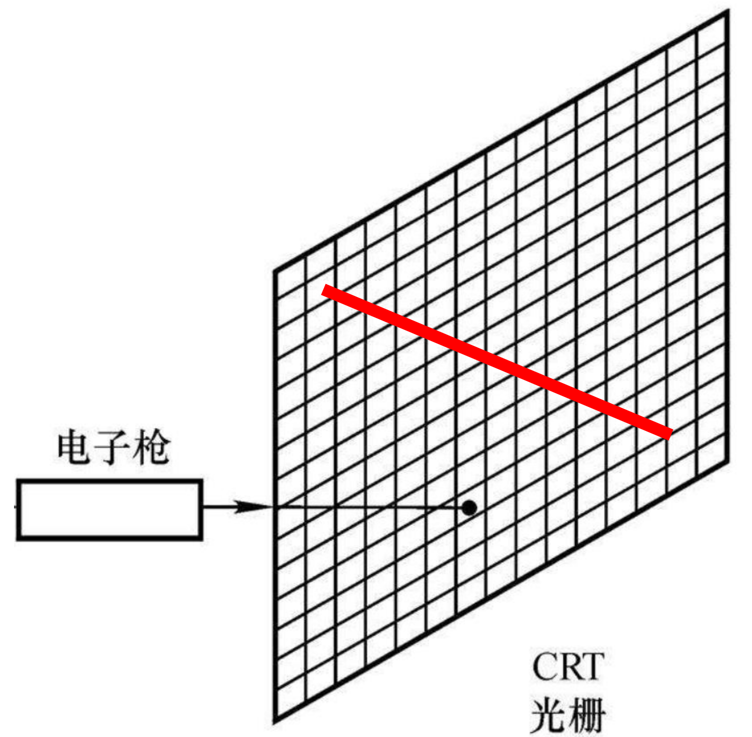


2.2.3. 2D图形的光栅化

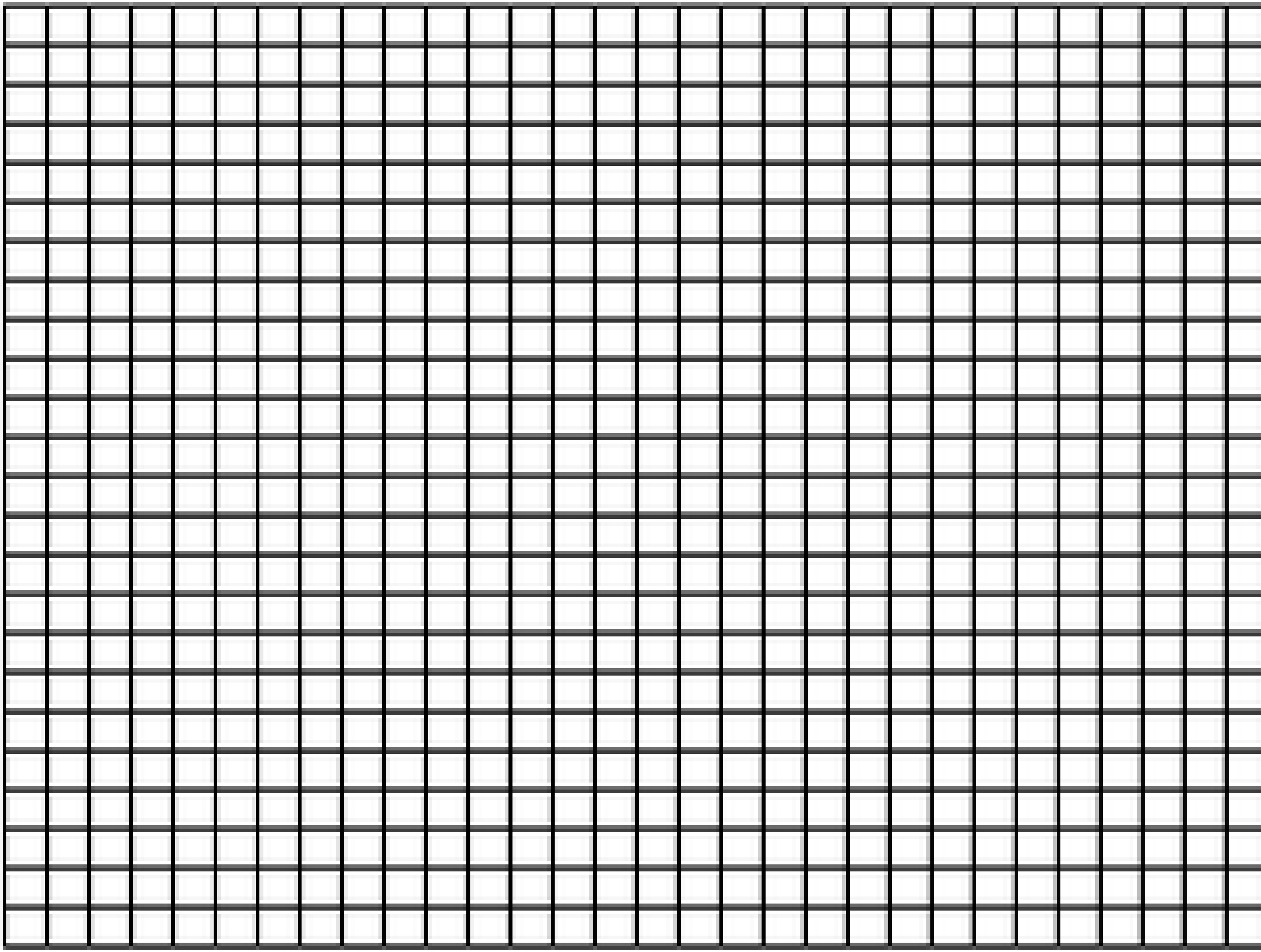
渲染(rendering)：2D图形呈现在光栅屏幕

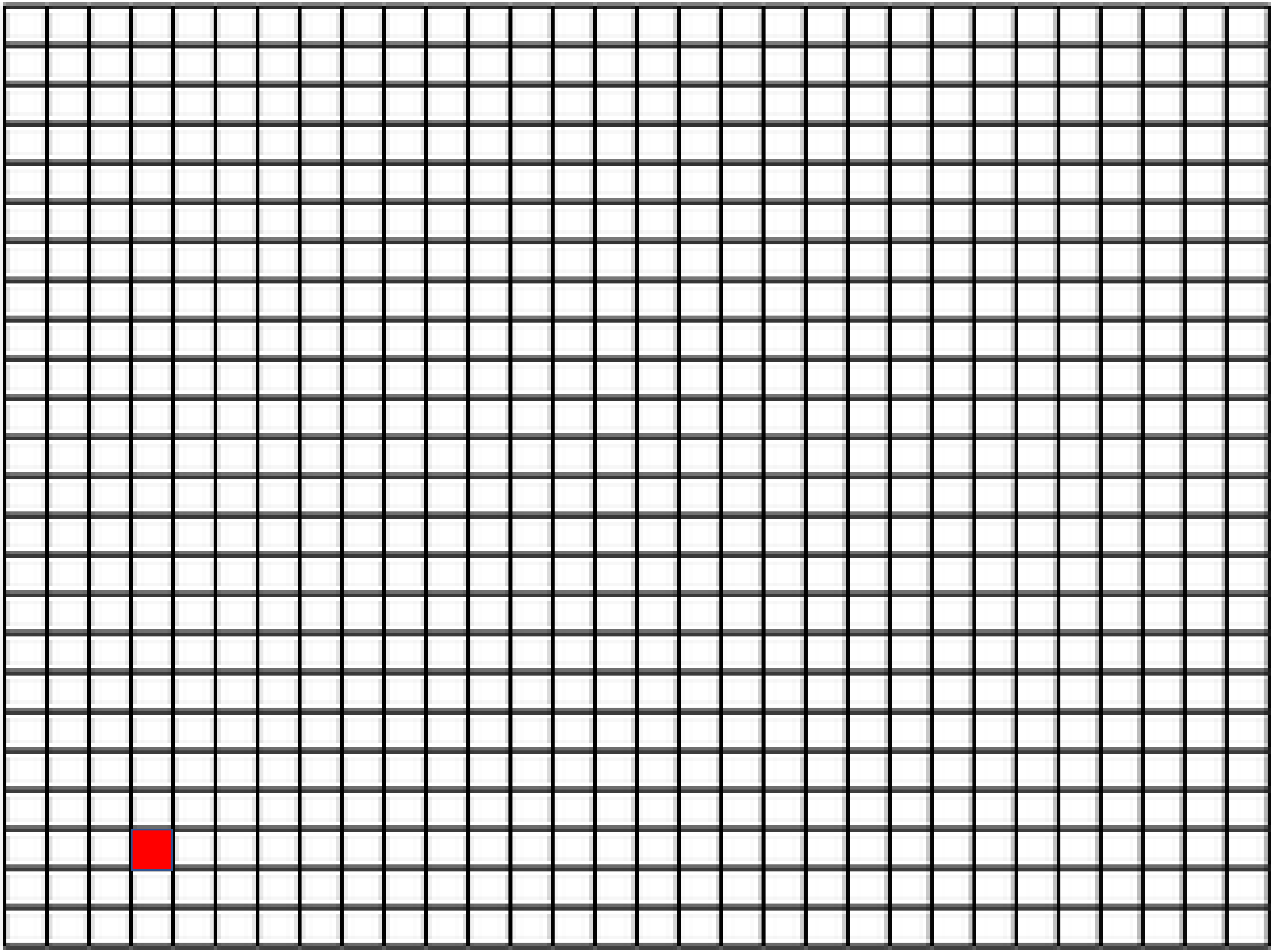
- 光栅化(rasterization)：将几何图形转化为光栅像素表达
  - 本质的数学问题是什么？

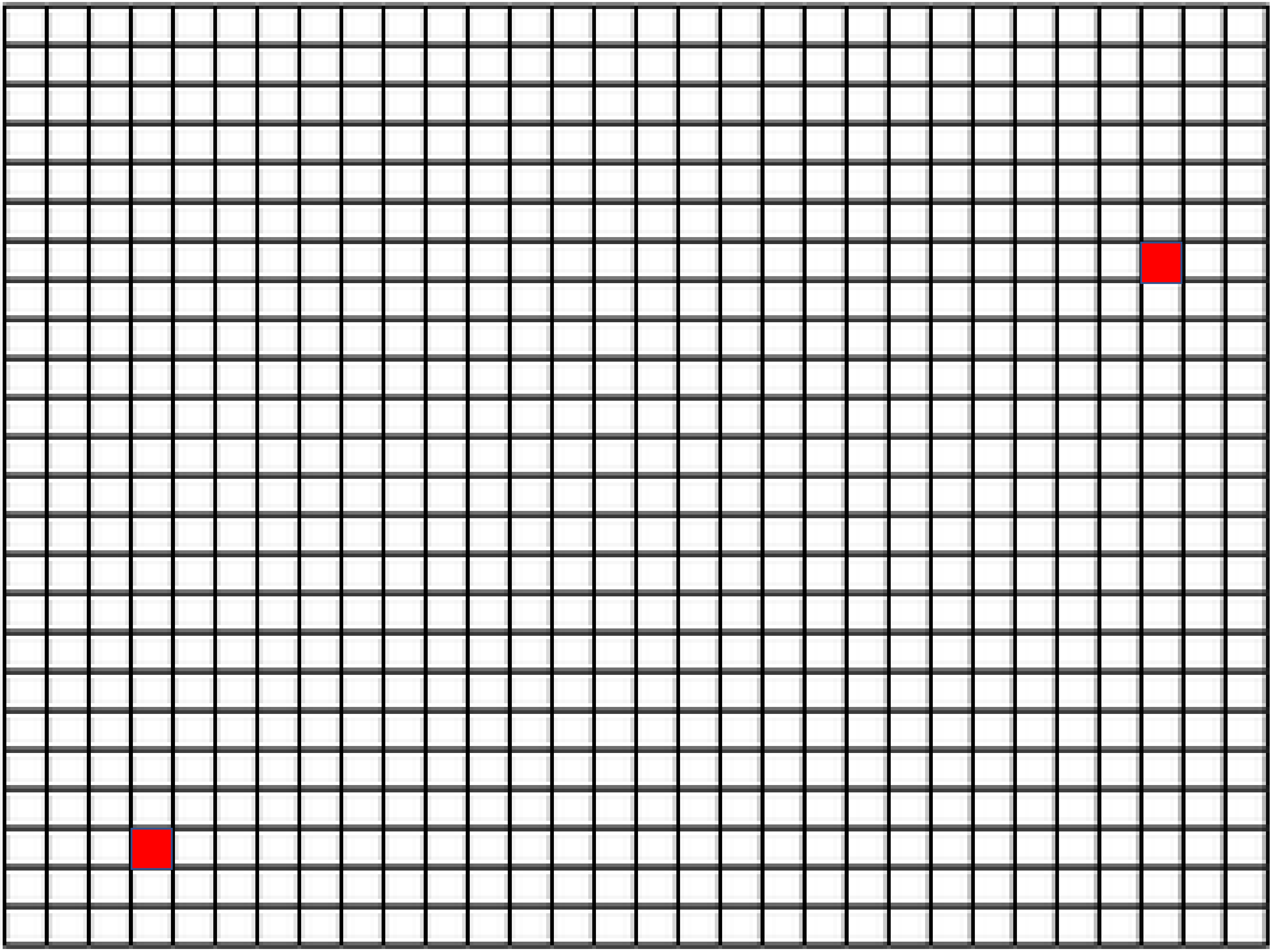


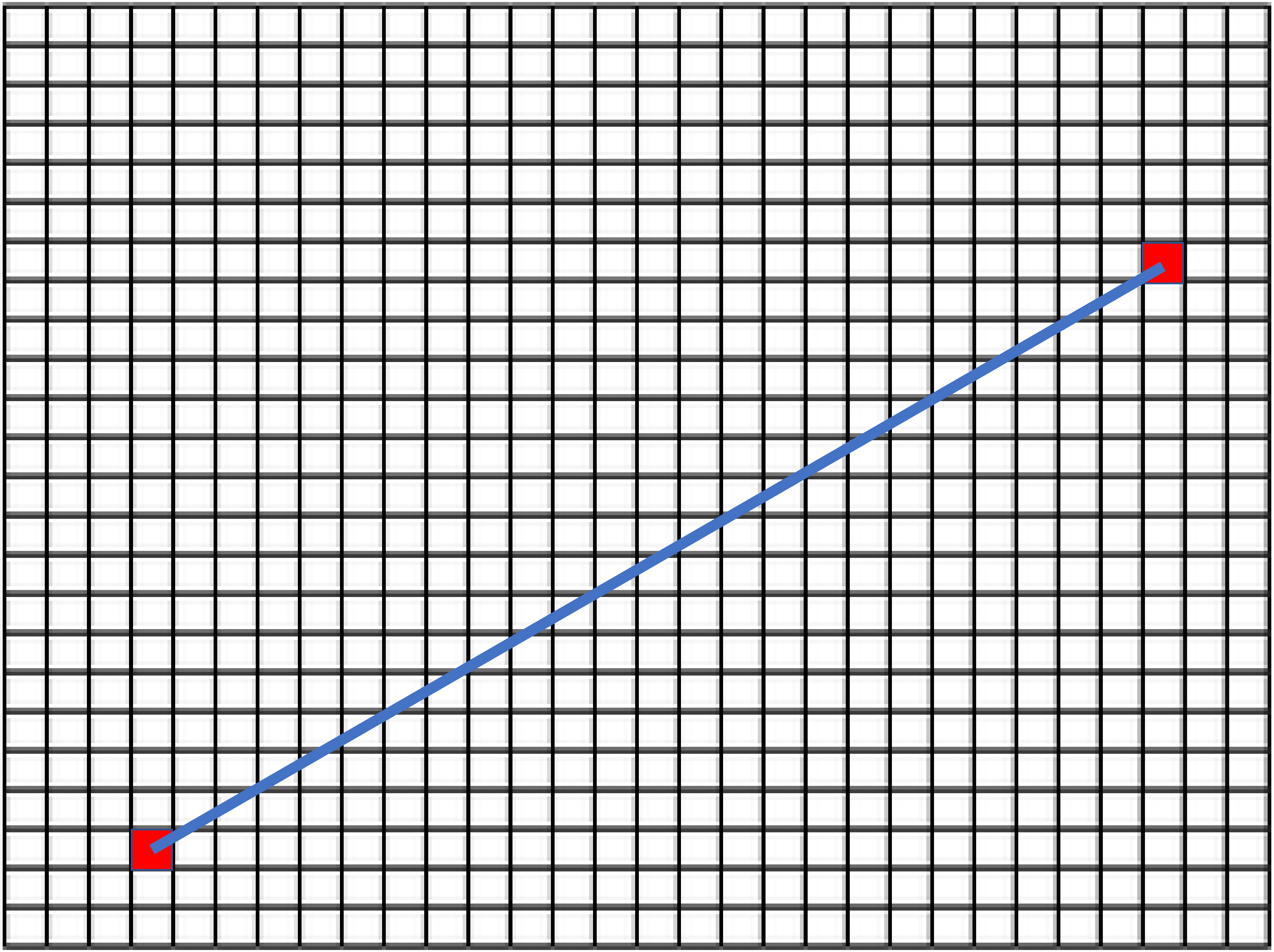


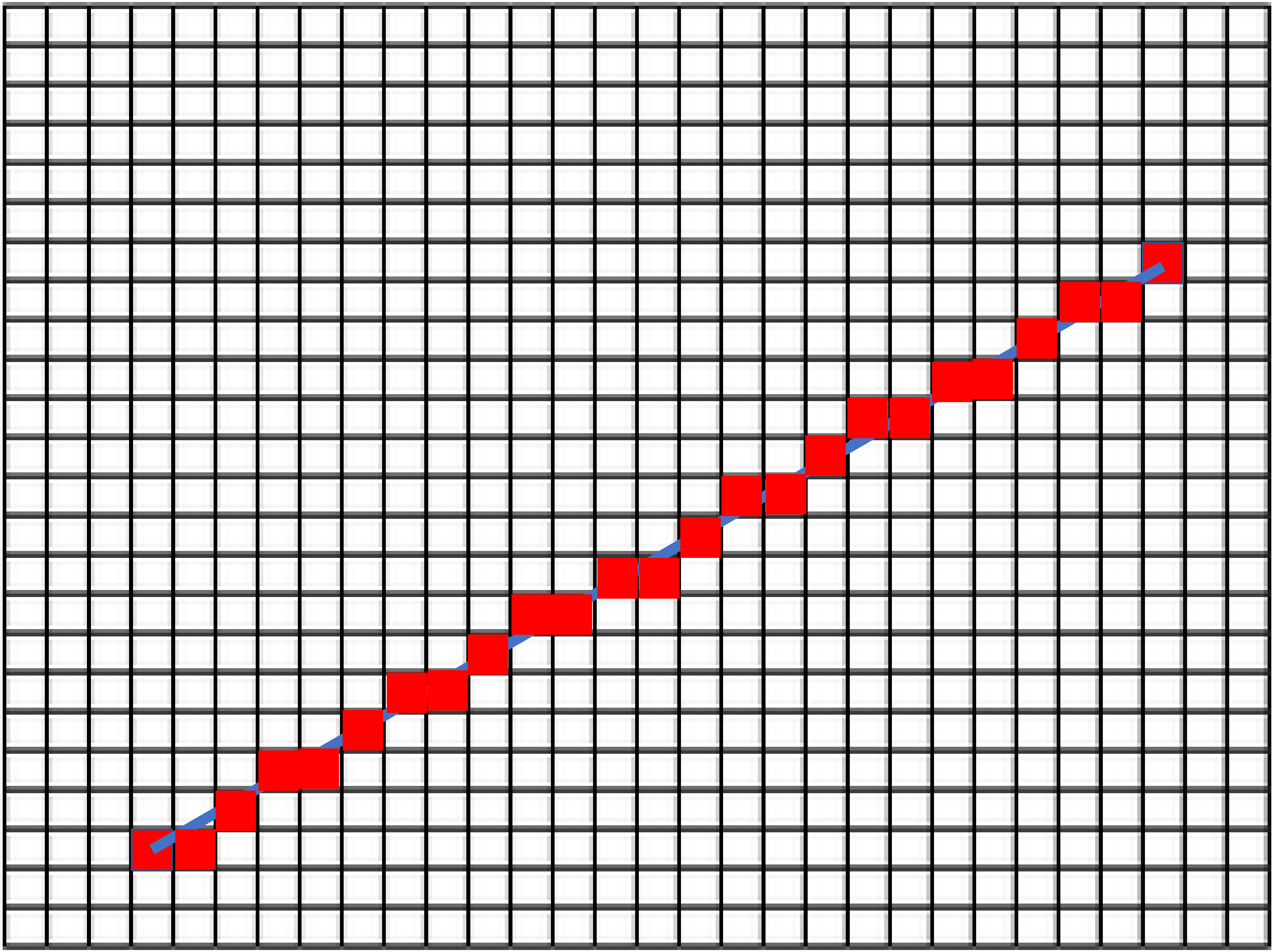
考虑：
如何画一条线段？



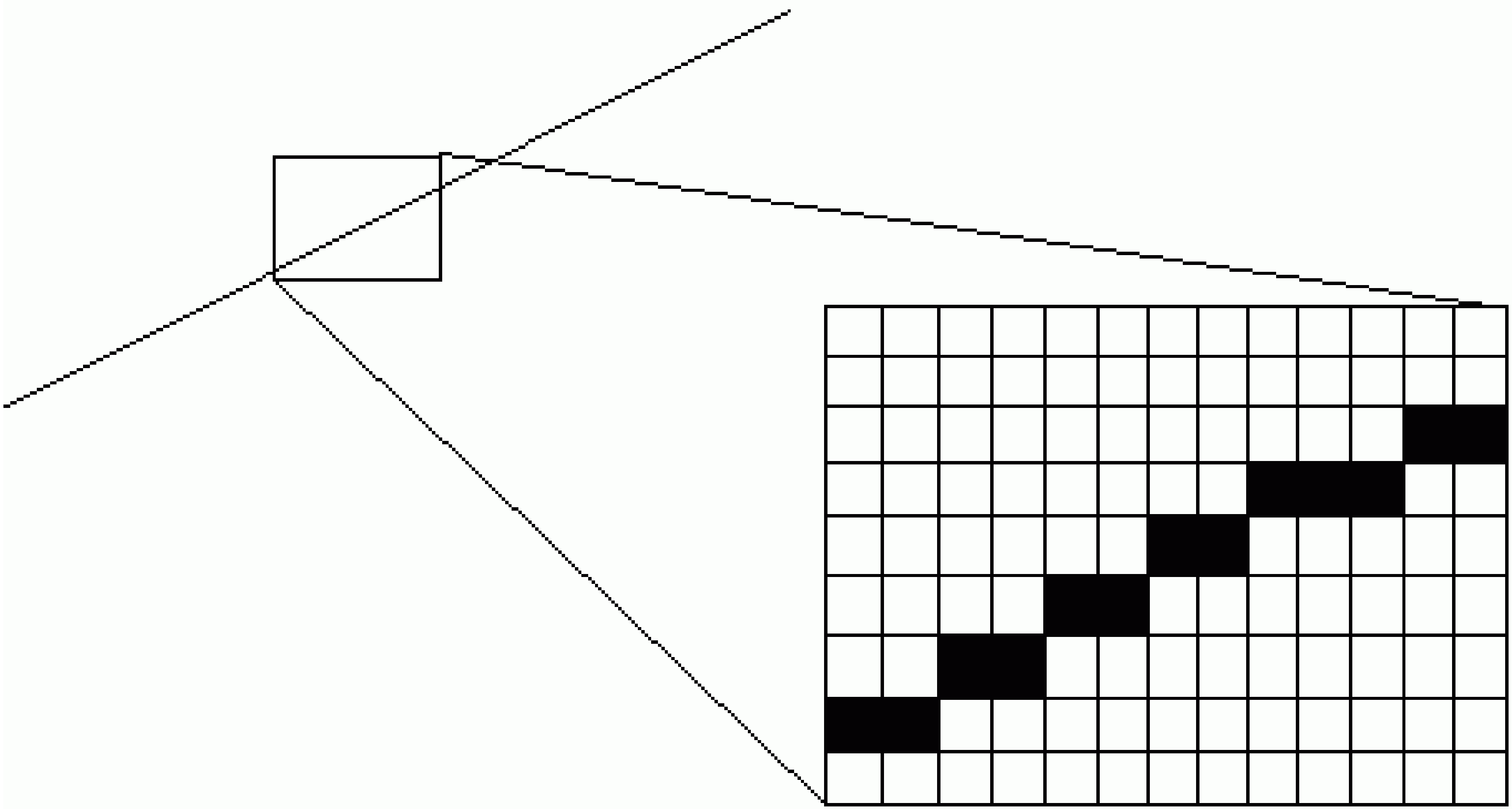








直线在栅格设备上的表现形式



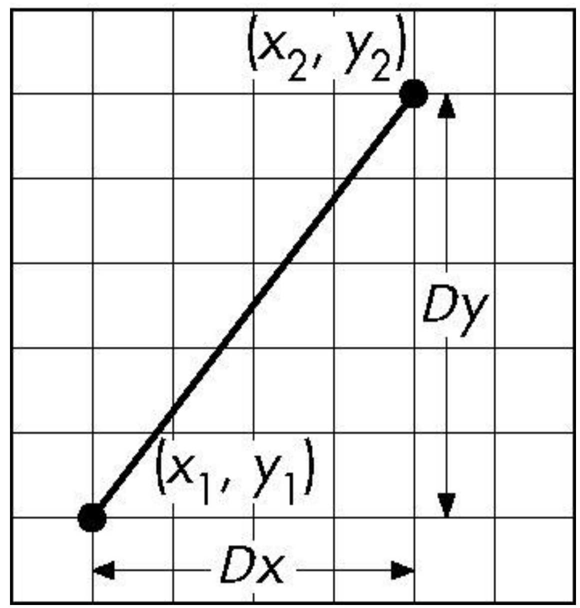
2.2.3.1 线段的光栅化

线段的光栅化

- 首先考虑端点坐标为整数的线段

    m = Dy/Dx

y = mx + h



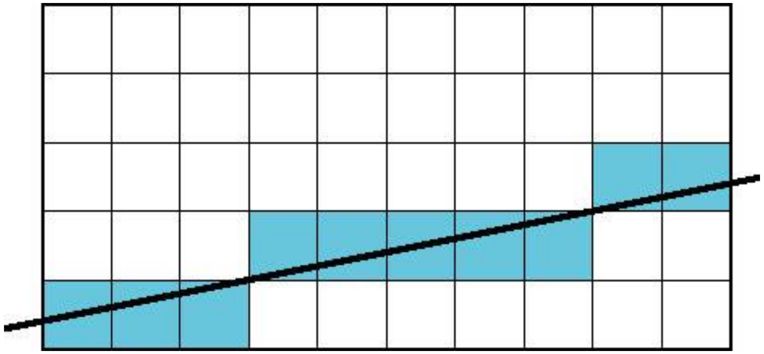
(1) DDA算法

- DDA: Digital Differential Analyzer 数字微分分析法
  - 直线 $y = mx + h$满足微分方程

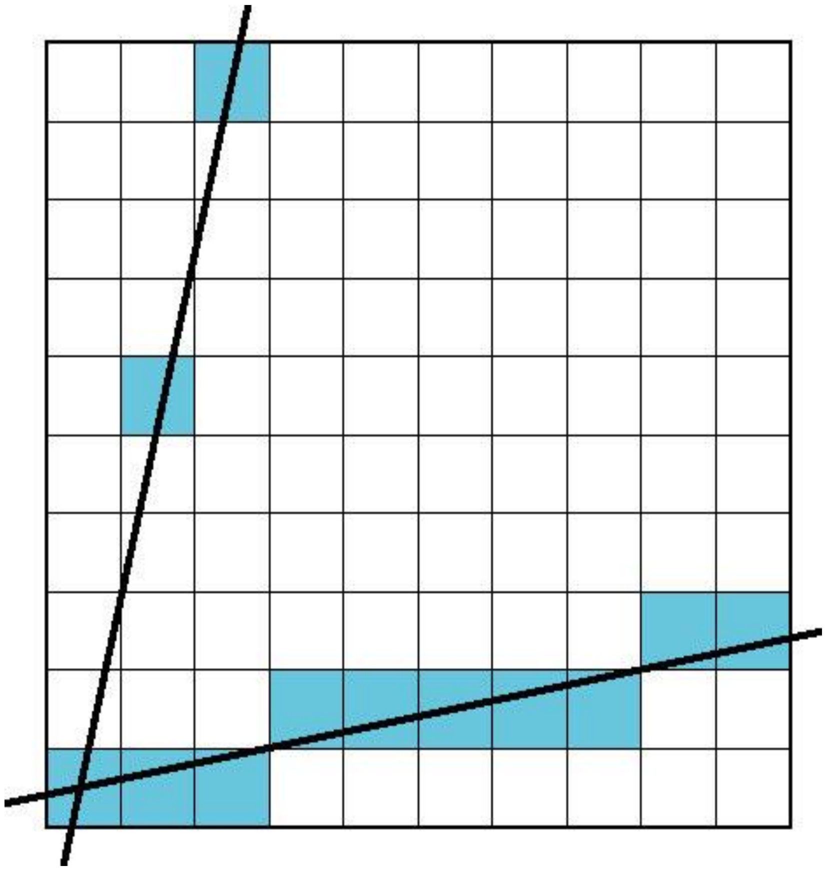
  $dy/dx = m = Dy/Dx = (y_2 - y_1)/(x_2 - x_1)$
- 沿扫描线 $Dx = 1$

```
for(x = x1; x<=x2; x++)
{
  y+=m;
  write_pixel(x, round(y));
}
```

- **对于每个x画出最接近的整数y**



问题：斜率大的直线



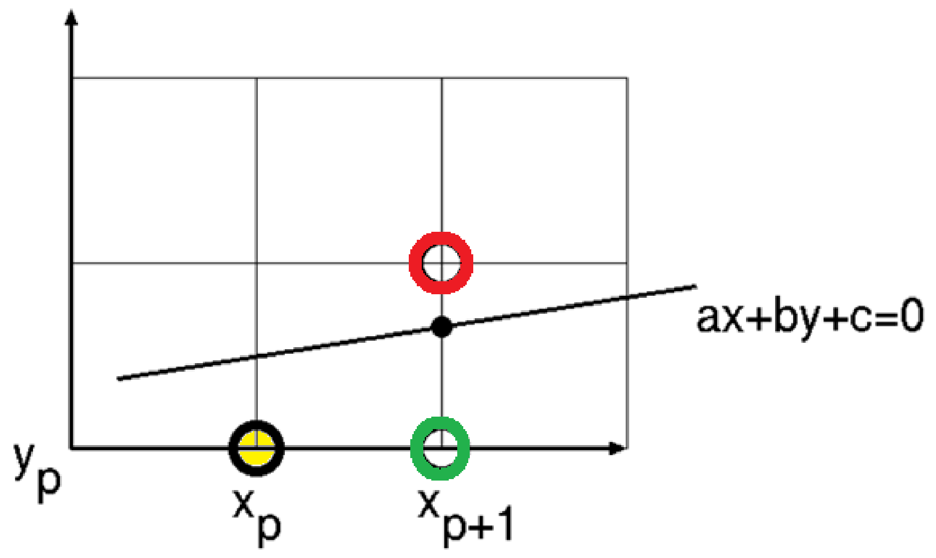
利用对称性，交换x与y的角色

(2) Bresenham算法（midpoint算法）

- DDA算法中每一步需要一次浮点加法
- 在Bresenham算法中可以不出现任何浮点运算
- 只考虑$0 \le m \le 1$的情形
  - 其它情形利用对称性处理
- 假设像素中心在半整数处
- 如果从一个已被确定激活的像素出发，那么下一像素的可能位置只会有两种可能

可能的下一个像素

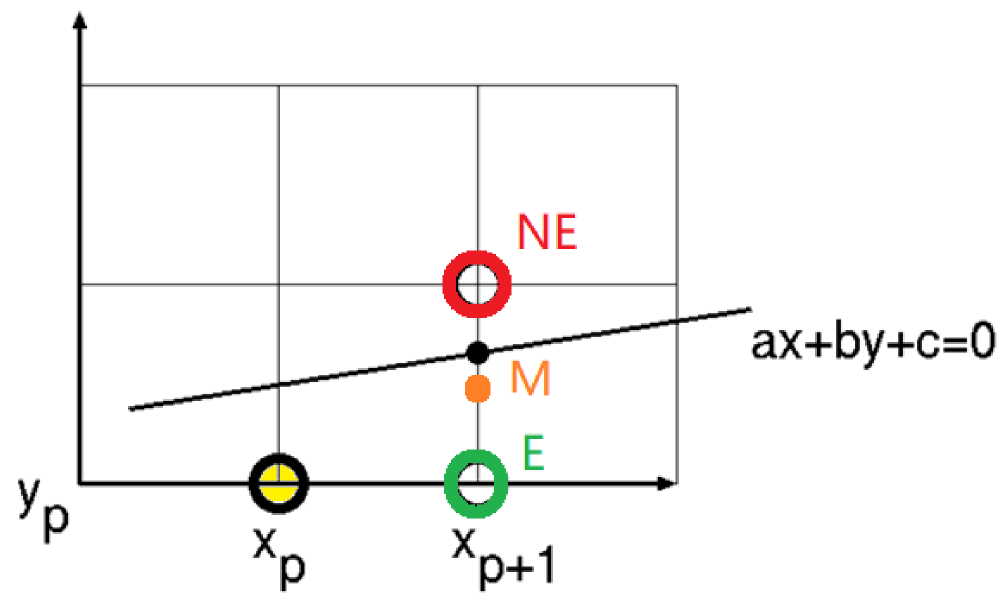
当 0<m<1时，
假设（x_p,y_p）是当前被点亮的像素，
则可能的下一个像素为(x_{p+1}, y_p) 或者（x_{p+1}, y_p+1）



算法概述

- 假设（x_p,y_p）是被点亮的像素
- 检查直线与$x = x_{p+1}$的交点
- 根据交点的位置来判断下一步选 E：(x_{p+1}, y_p)

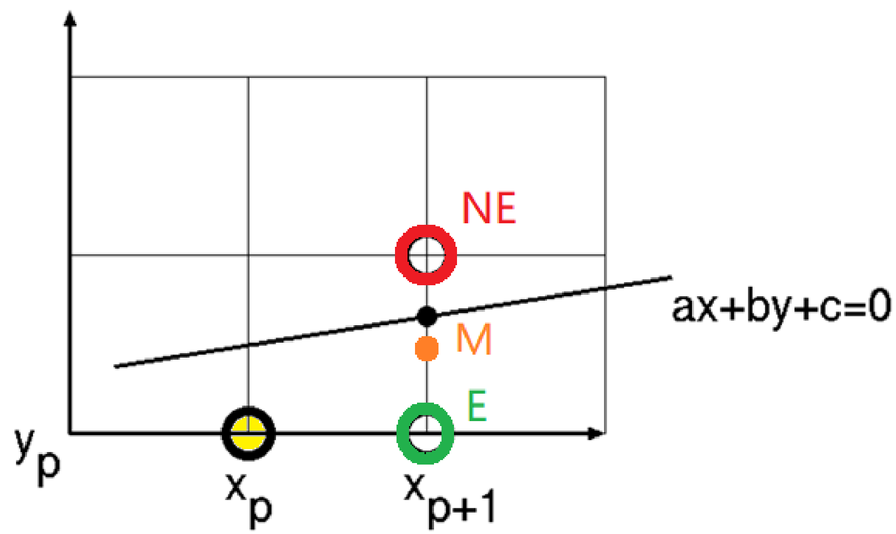
或者 NE：(x_{p+1}, y_p+1)



判断的核心

- 检查直线与$x = x_{p+1}$的交点
- 判断交点离哪个点最近
- 等价于：

判断两个候选点的中点M在直线之上还是之下



判断的核心

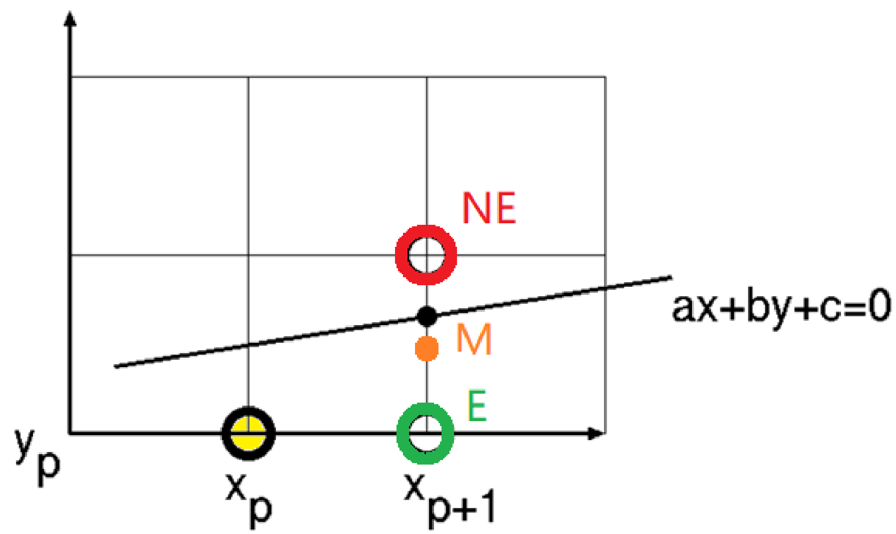
- 假设目标是绘制点(x1,y1),(x2,y2)之间的线段
- 则斜率为dy/dx， 其中dx = x2-x1, dy = y2 – y1
- 则该线段可以表示为 $y = mx + n$ $y = \frac{dy}{dx}x + n$

则 $F(x,y) = ax + by + c = 0$

$$F(x,y) = dy.x - dx.y + dx.n = 0$$

如何判断点和直线的相对位置?

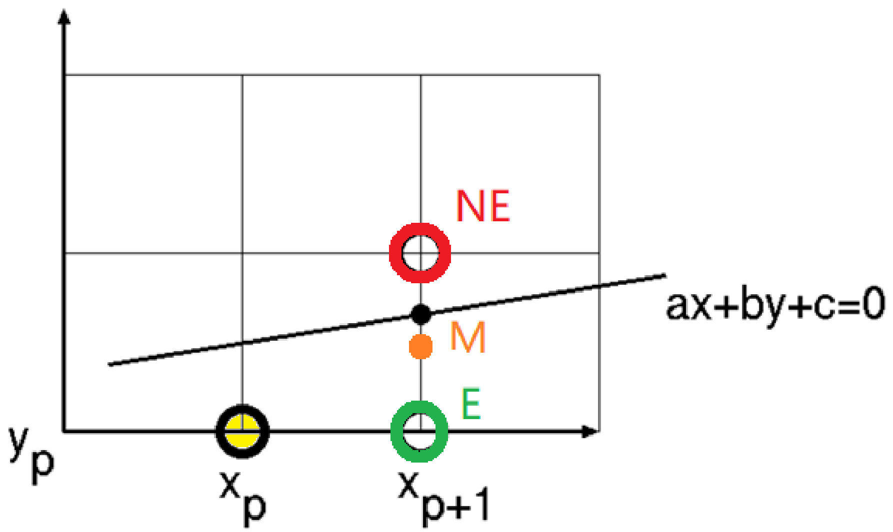
判断 F(x,y) > 0 还是 <0



决策变量

将M点带入F

$$d = F(x_p + 1, y_p + \frac{1}{2})$$



更新决策变量

根据上一步选E还是NE，我们可以计算下一步的决策变量：

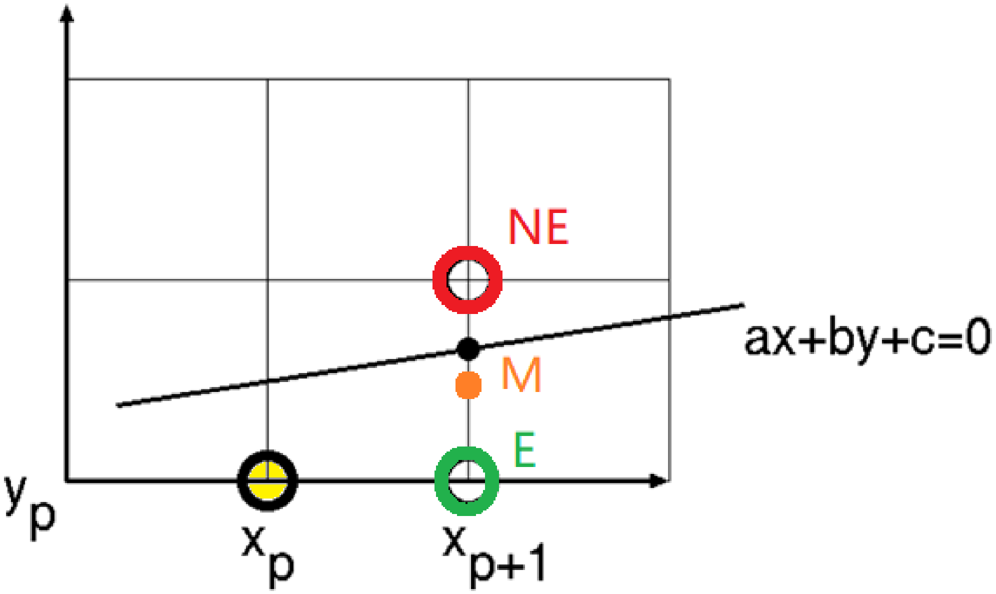
如果选E:

$$d_{new} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$

$$d_{old} = F(x_p + 1, y_p + \frac{1}{2})$$

$$= a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

$$d_{new} = d_{old} + a$$
$$= d_{old} + dy$$



更新决策变量

根据上一步选E还是NE，我们可以计算下一步的决策变量：

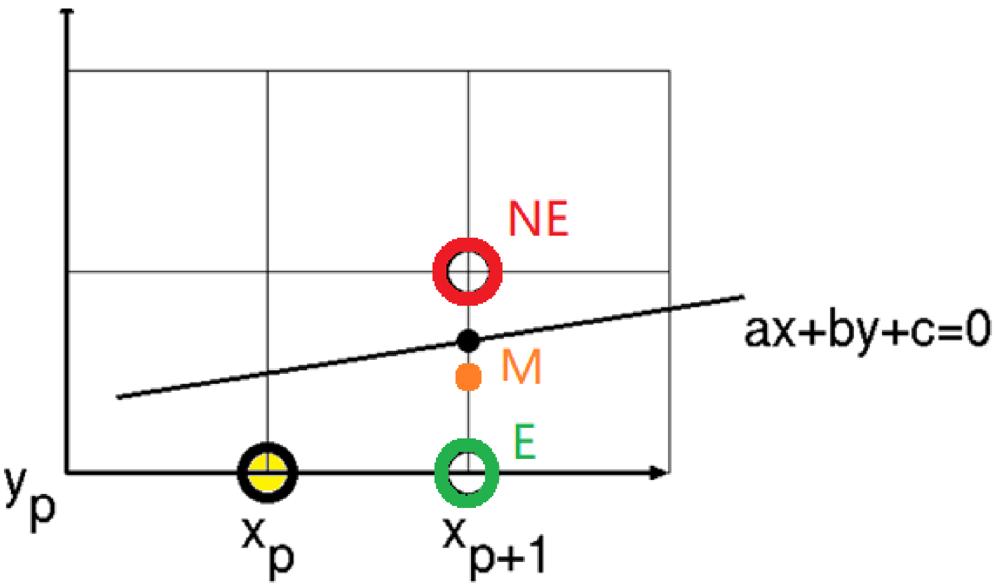
如果选NE:

$$d_{new} = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

$$d_{old} = F(x_p + 1, y_p + \frac{1}{2})$$

$$= a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

$$\begin{aligned} d_{new} &= d_{old} + a + b \\ &= d_{old} + dy - dx \end{aligned}$$



决策变量的初始值

$$
\begin{aligned}
d_{start} &= F(x_1+1, y_1+\frac{1}{2}) = a(x_1+1)+b(y_1+\frac{1}{2})+c \\
&= ax_1+by_1+c+a+\frac{b}{2} \\
&= F(x_1, y_1)+a+\frac{b}{2}
\end{aligned}
$$

因为*F(x1,y1)= 0,* 所以

$$d_{start} = dy - dx/2$$

上式乘以2以避免浮点数的计算

（我们仅需要考虑决策变量的符号）

决策变量总结:

- 决策变量的新表达式 $d = F(x,y) = 2(ax+by+c) = 0$
- 初始值 $d_{start} = 2dy - dx$
- 如果选E作为下一个像素点 $d_{new} = d_{old} + 2dy$
- 如果选NE作为下一个像素点 $d_{new} = d_{old} + 2dy - 2dx$

- 对每个x值，只需要进行整数加法以及测试
- 可以在图形芯片上用单个指令实现

Midpoint algorithm

```
void MidpointLine(int
   x1,y1,x2,y2)
{
int dx=x2-x1;
int dy=y2-y1;
int d=2*dy-dx;
int increE=2*dy;
int incrNE=2*(dy-dx);
x=x1;
y=y1;
WritePixel(x,y);
```

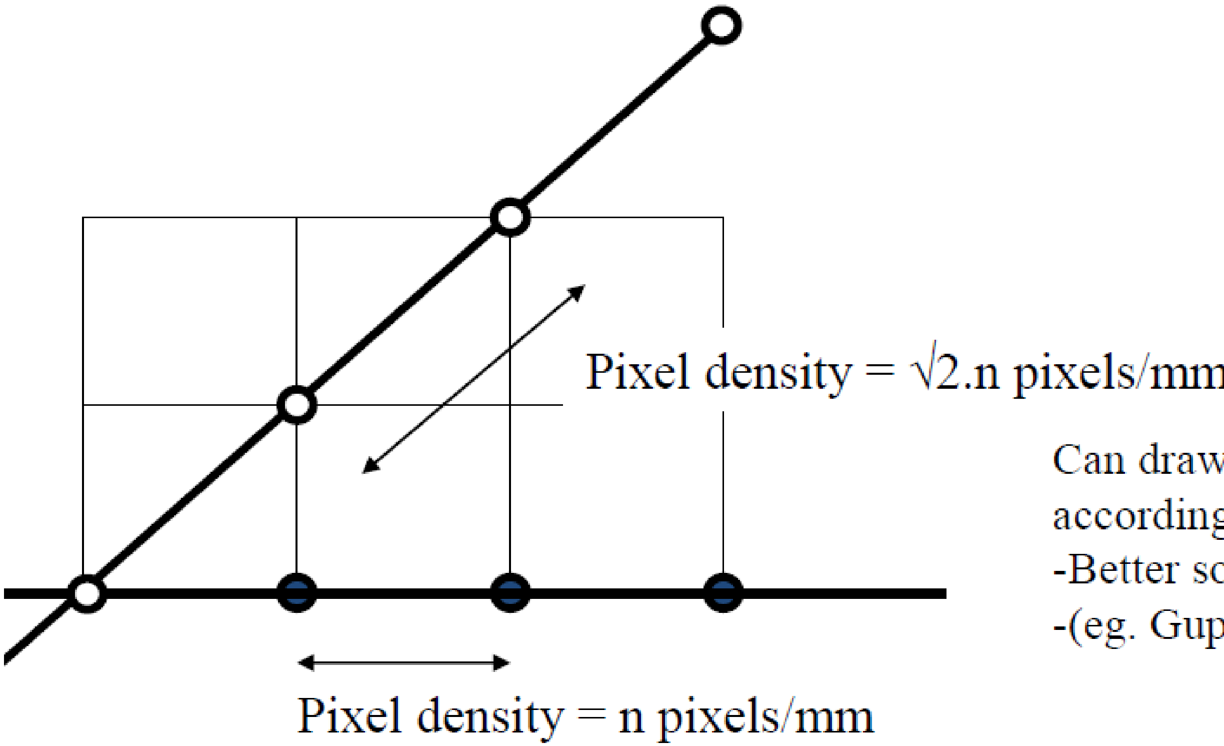
```
while (x < x2) {
        if (d<= 0) {
                d+=incrE;
                x++
        } else {
                d+=incrNE;
                x++;
                y++;
        }
        WritePixel(x,y);
}
}
```

拓展：

What if the slope is not between 0 and 1?

- Use symmetry
- For example,
  - For $m > 1$, switch x and y, draw the line, and switch back x and y
  - For $m < 0$, negate the line, draw the line, and negate again

Midpoint方法有什么缺点？



Can draw lines in darker colours according to line direction.

-Better solution : antialiasing !

-(eg. Gupta-Sproull algorithm)

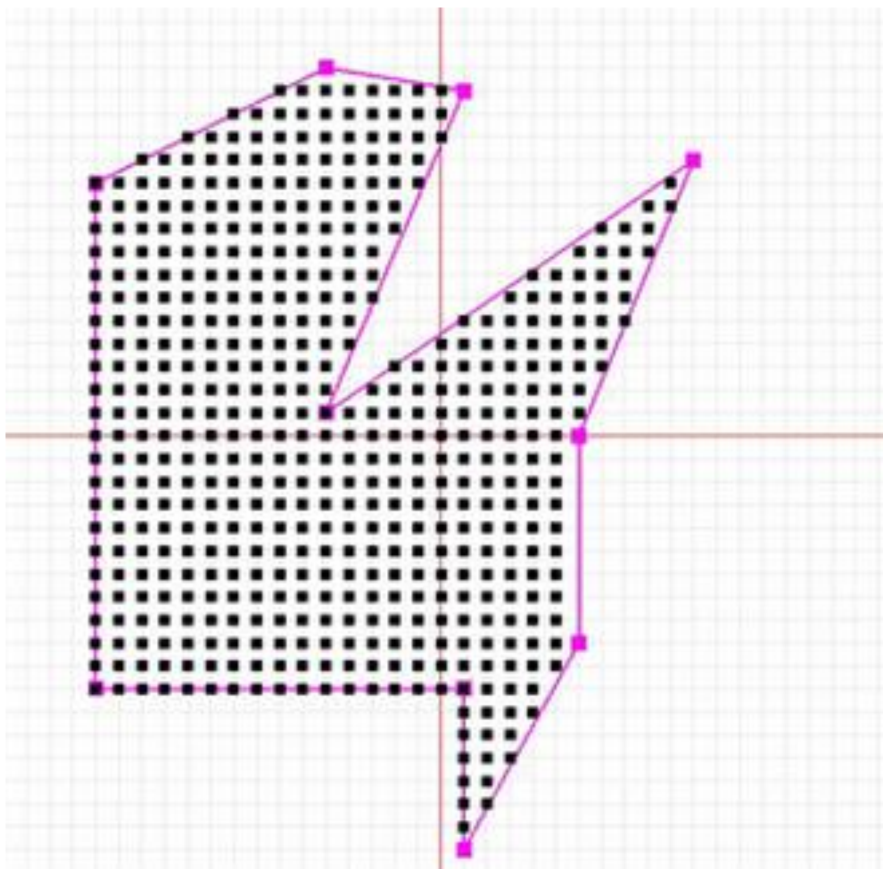
硬件实现

- 线段的光栅化已成熟，并已由硬件实现
- GDI:
  - Moveto(x,y)
  - Lineto(x,y)

2.2.3.2 多边形区域的光栅化

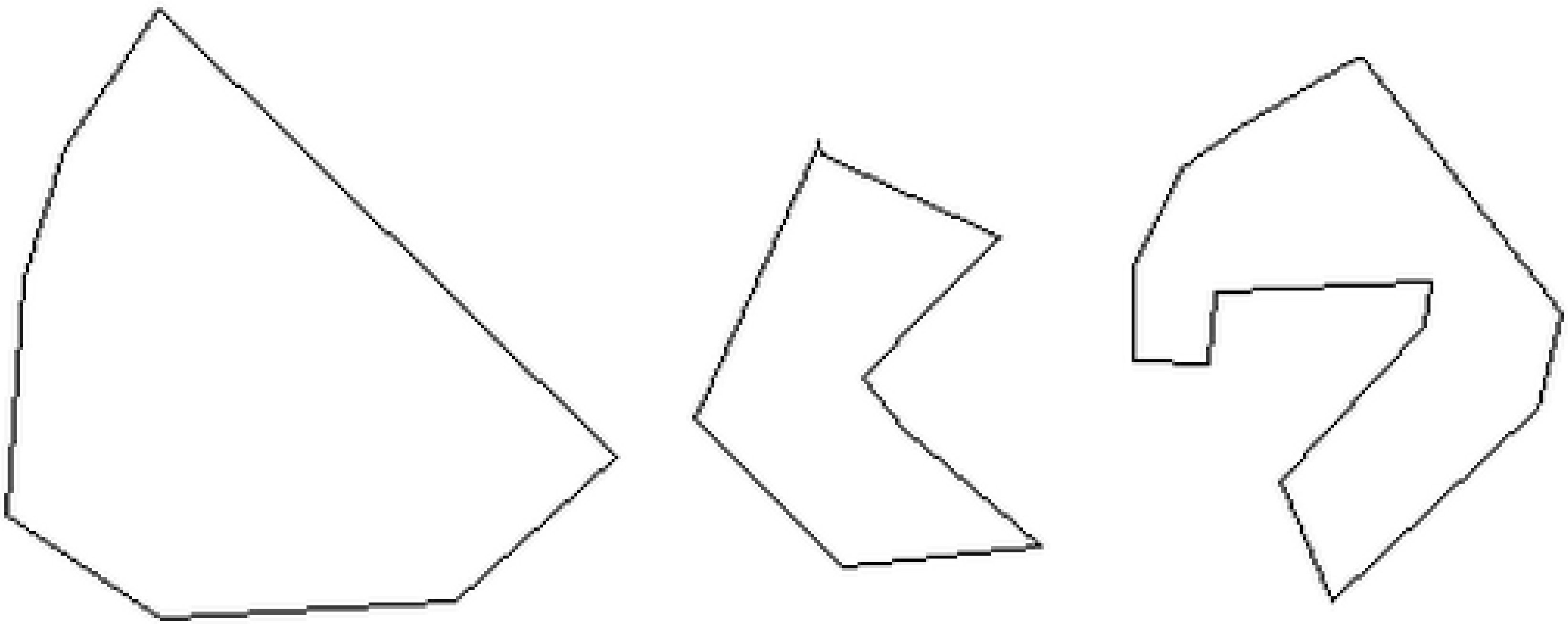
多边形区域的光栅化

- 多边形区域的填充：区域内部的像素点



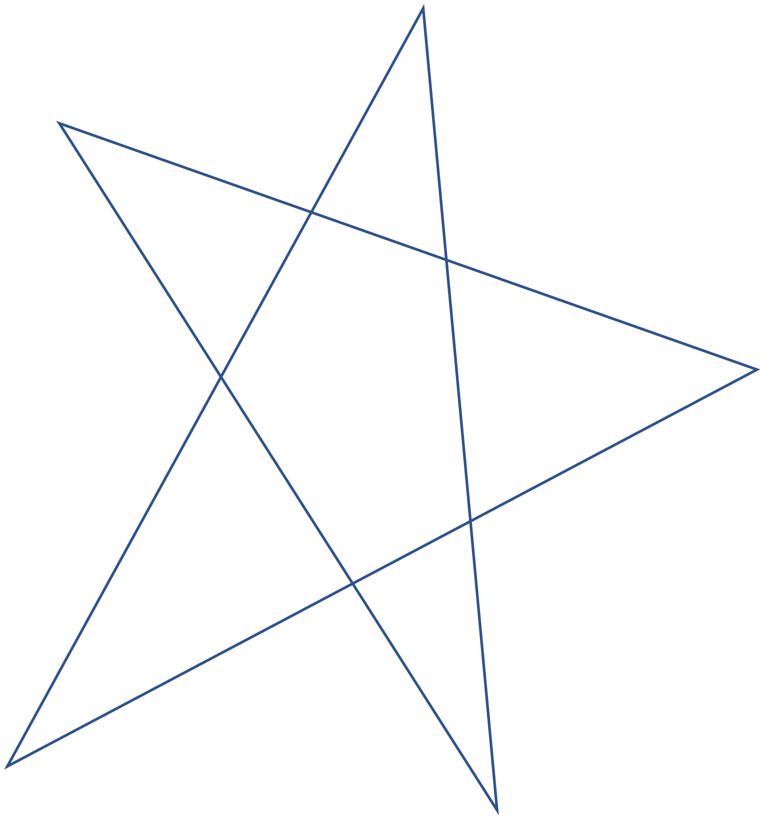
内外检测

- 如何决定封闭多边形的内外?



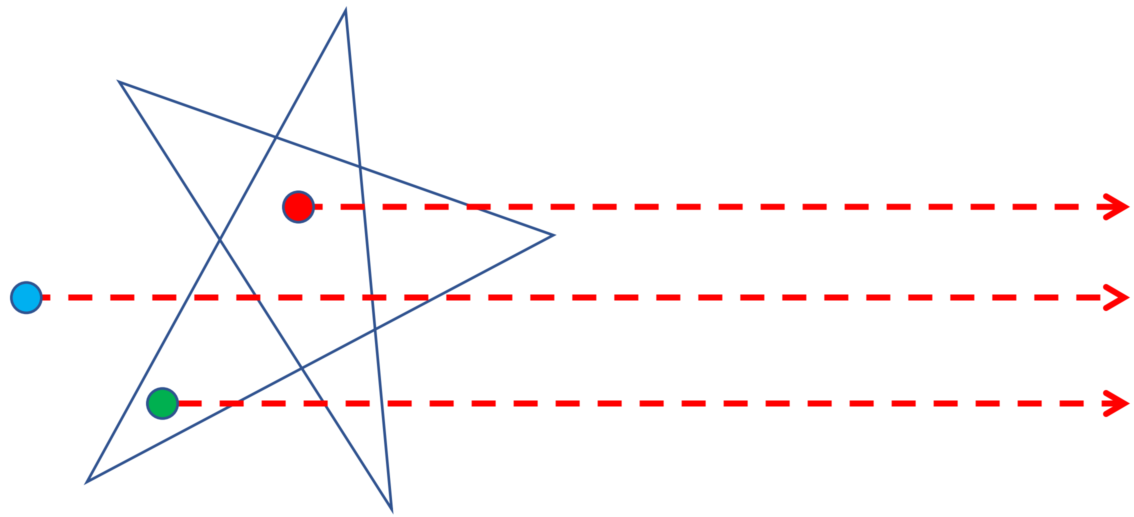
内外检测

- 如何决定封闭多边形的内外?



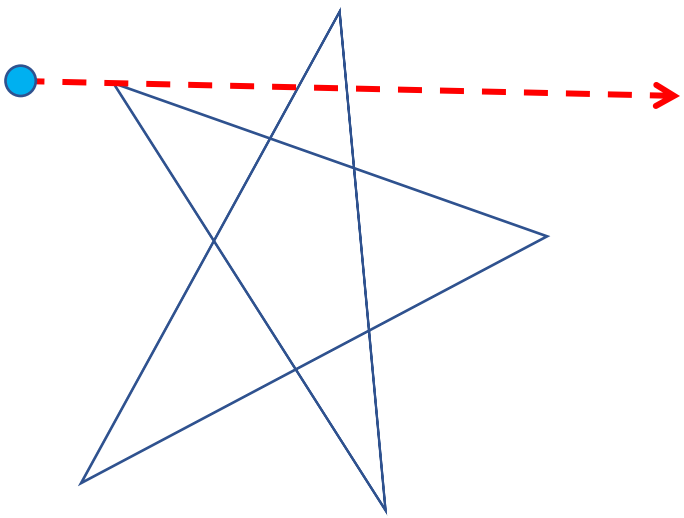
内外检测：奇偶检测法

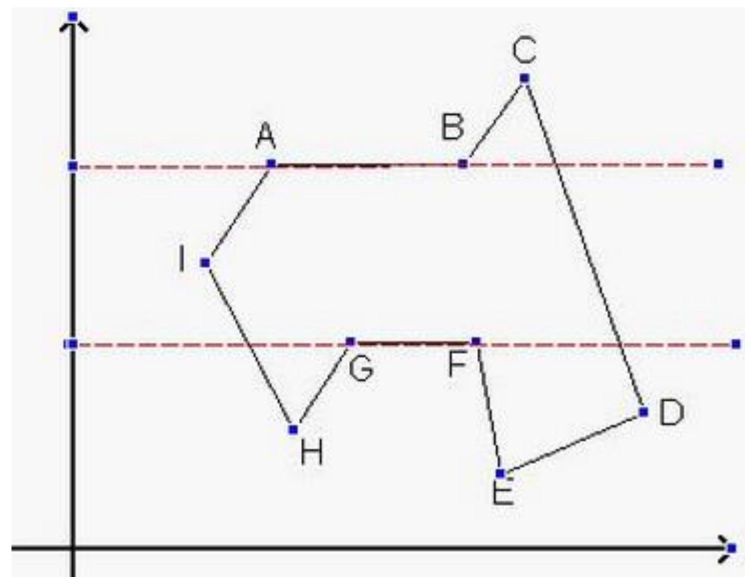
- 也称为射线法
  - 从一点p引射线，如果与多边形边界交点数为偶数，则p在多边形外，否则在多边形内部



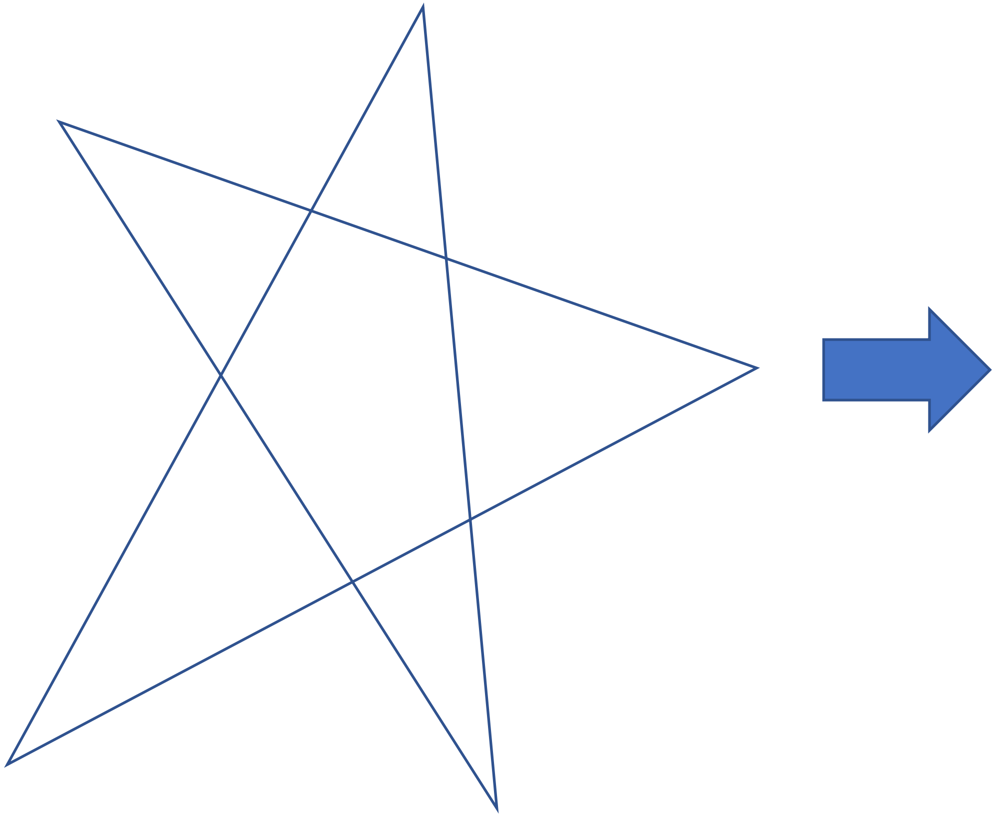
内外检测：奇偶检测法

- 奇异情形

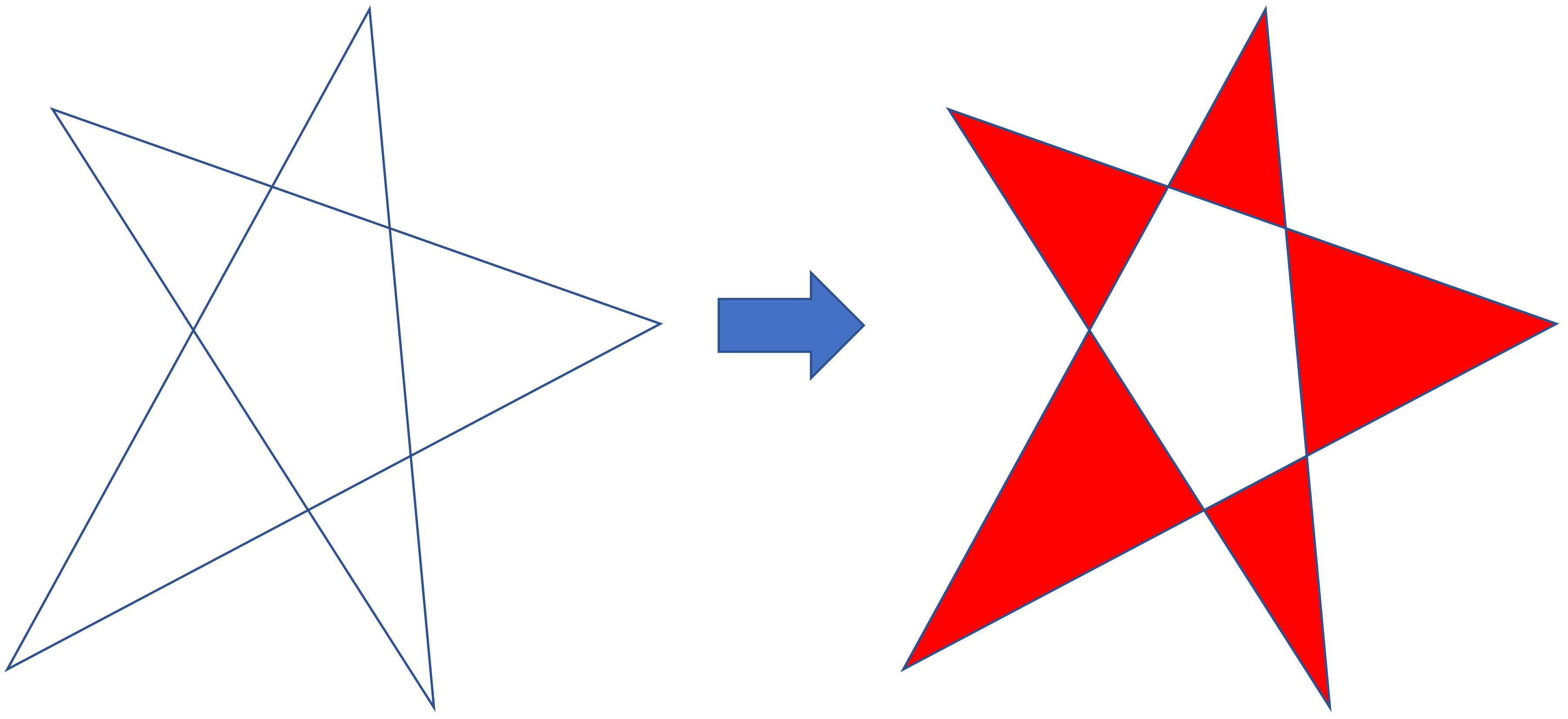




内外检测：奇偶检测法

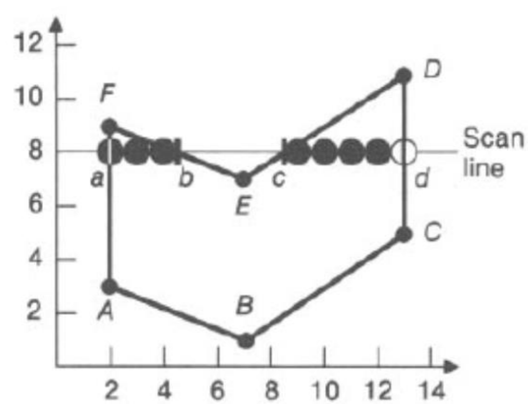


内外检测：奇偶检测法



扫描线转化算法

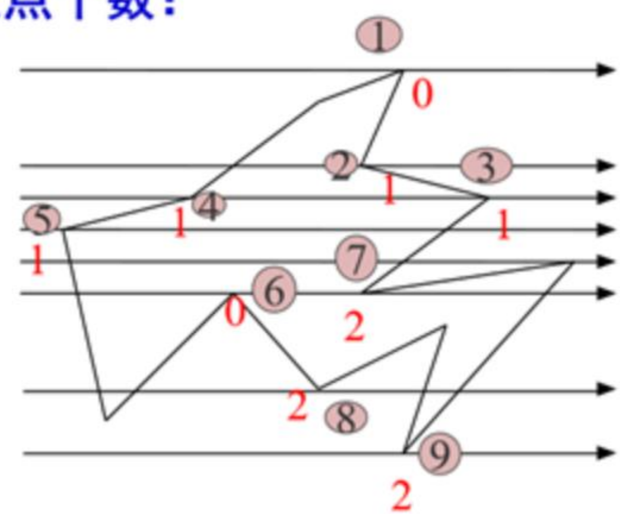
- 传统算法：沿着扫描线填充多边形
- 针对每一条扫描线:
  - 找到扫描线与所有边的交点
  - 根据交点的x值，对交点进行排序
  - 将交点进行配对，每一对交点之间进行填充



交点取舍

当扫描线与多边形顶点相交时，交点如何取舍?
顶点相关的两条边分别落在扫描线的两边，只取1，同边0或2。

举例计算交点个数:



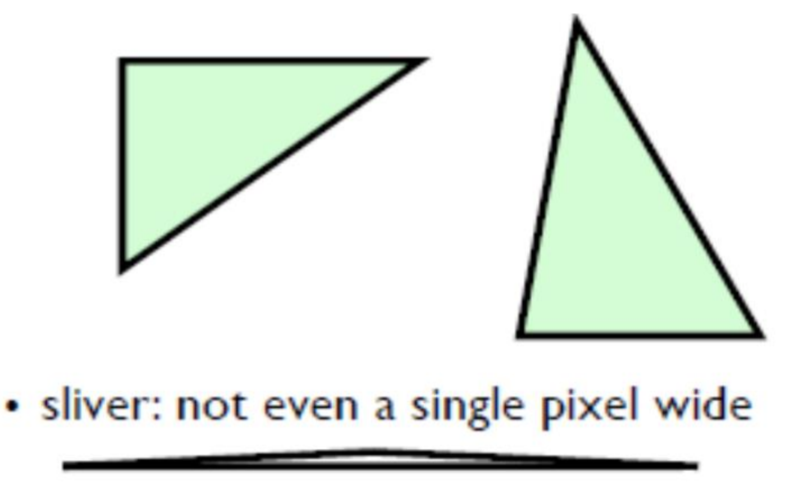
硬件实现

- 多边形区域的光栅化已成熟，并已由硬件实现
- GDI:
  - Polygon(…)
  - FillRgn(…)
  - CreateBrush(…)

扫描线转化算法的优缺点

- 简单、可处理凹多边形

- 顺序操作，比较难以高效并行
- 有例外情况需要特殊处理

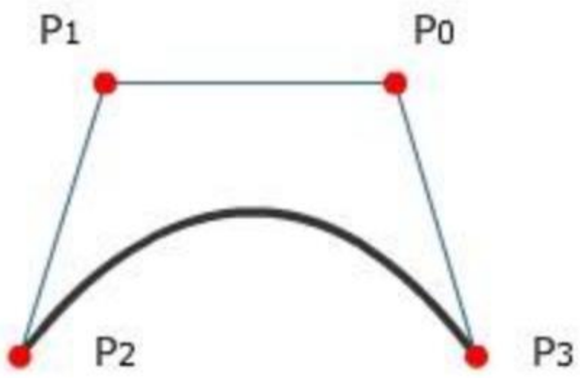


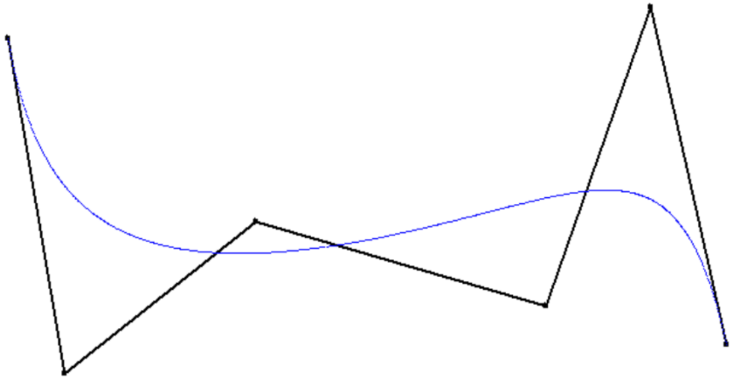
小结

- 2.2.1 数字图像及光栅显示器
- 2.2.2 2D图形
- 2.2.3 2D图形的光栅化
  - 2.2.3.1 线段的光栅化
  - 2.2.3.2 多边形的光栅化

3.3 光滑曲线的光栅化

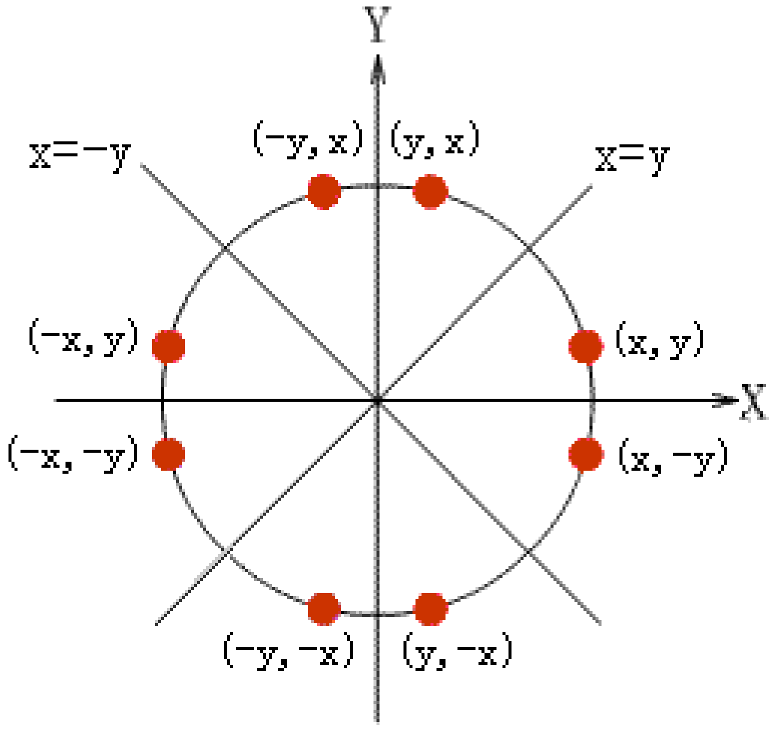
光滑曲线

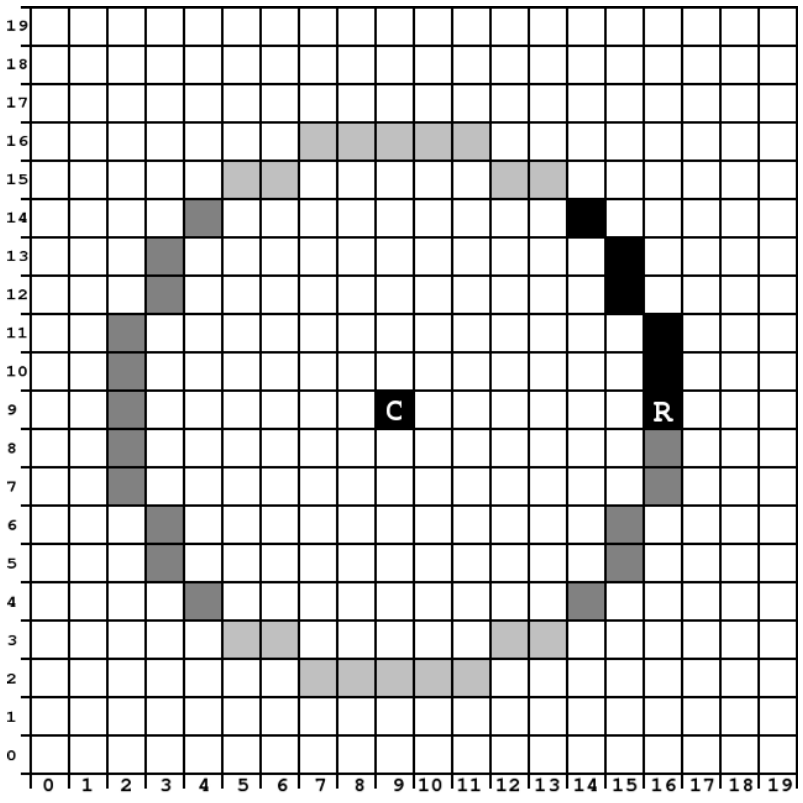




3.3.1 圆的光栅化算法

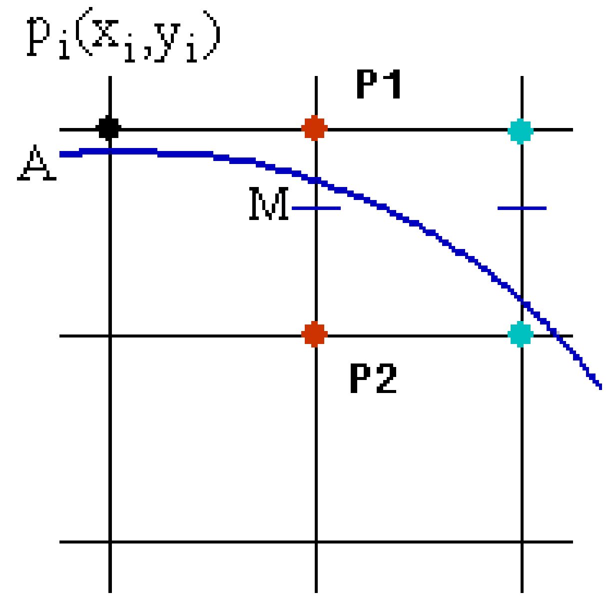
- 特殊算法：圆的方程及其八分对称性

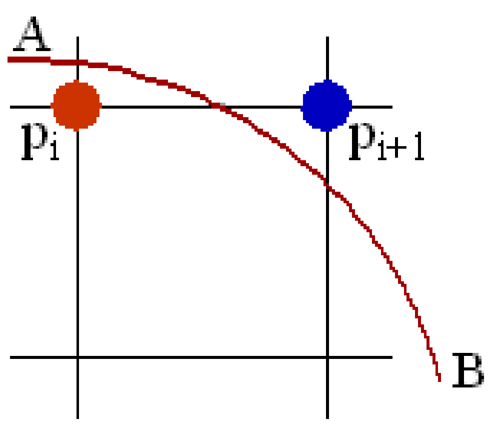




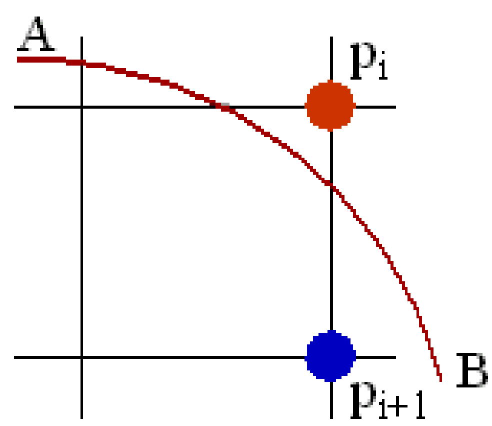
圆的光栅化算法

- 中点法
  - Bresenham算法
- 正负判定法

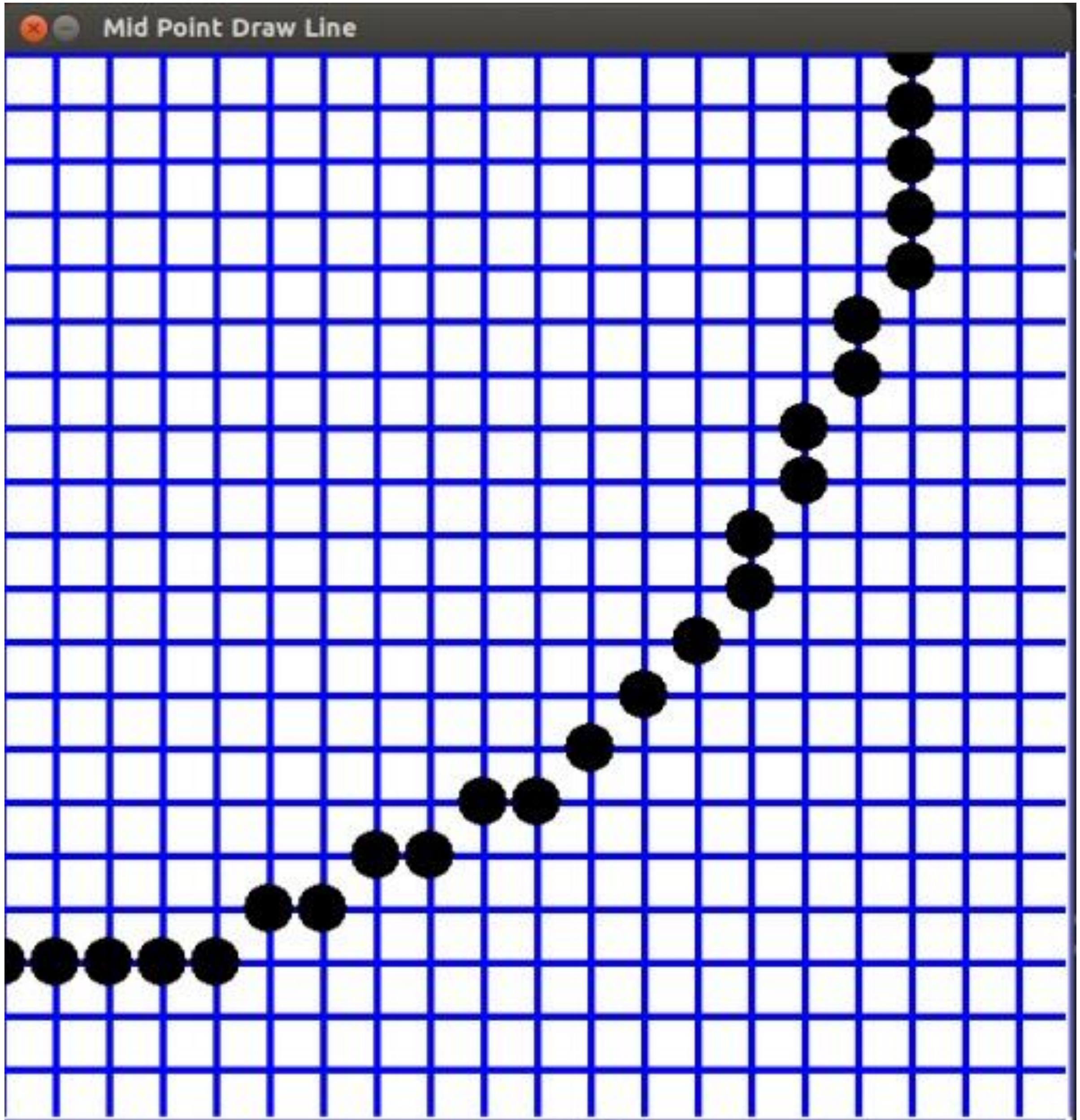




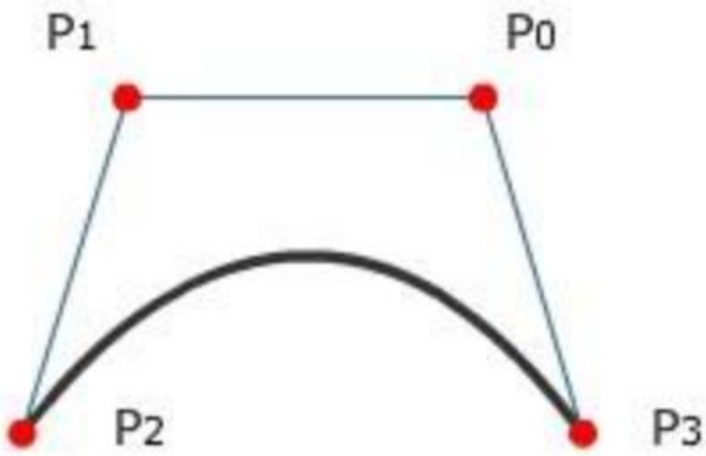
(a) $F(x_i,y_i)\le 0$，向右走

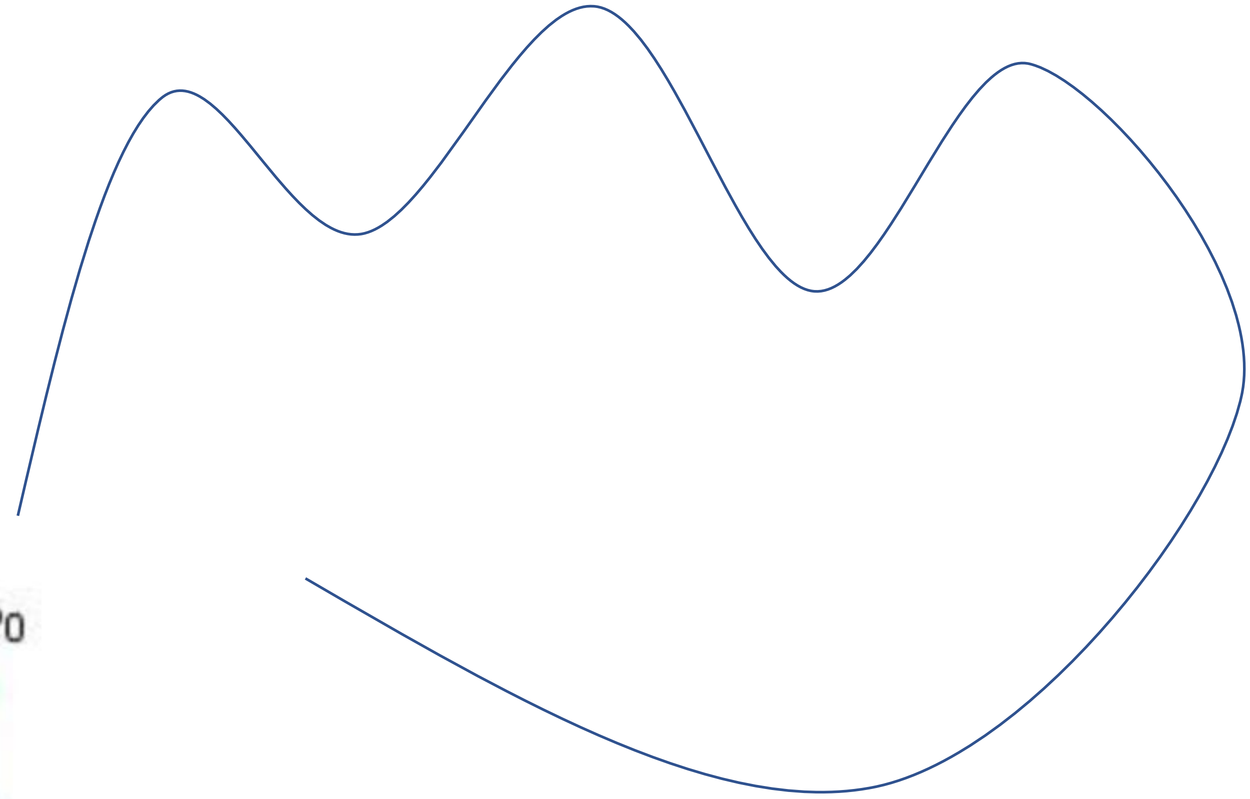


(b) $F(x_i,y_i)>0$，向下走


Mid Point Draw Line

一般曲线的情况呢?

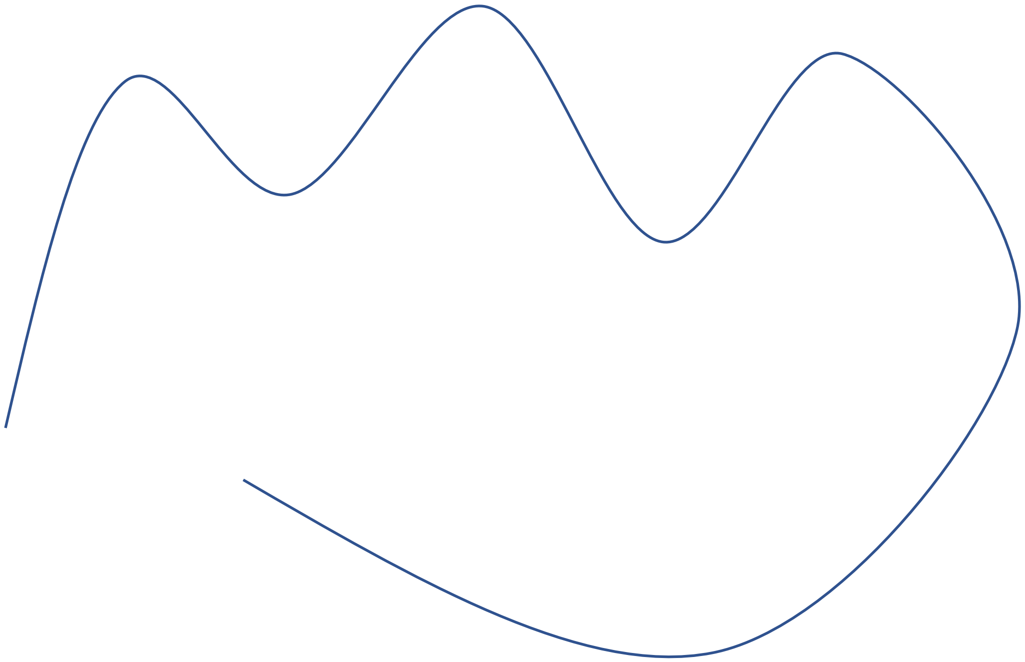




数学表达

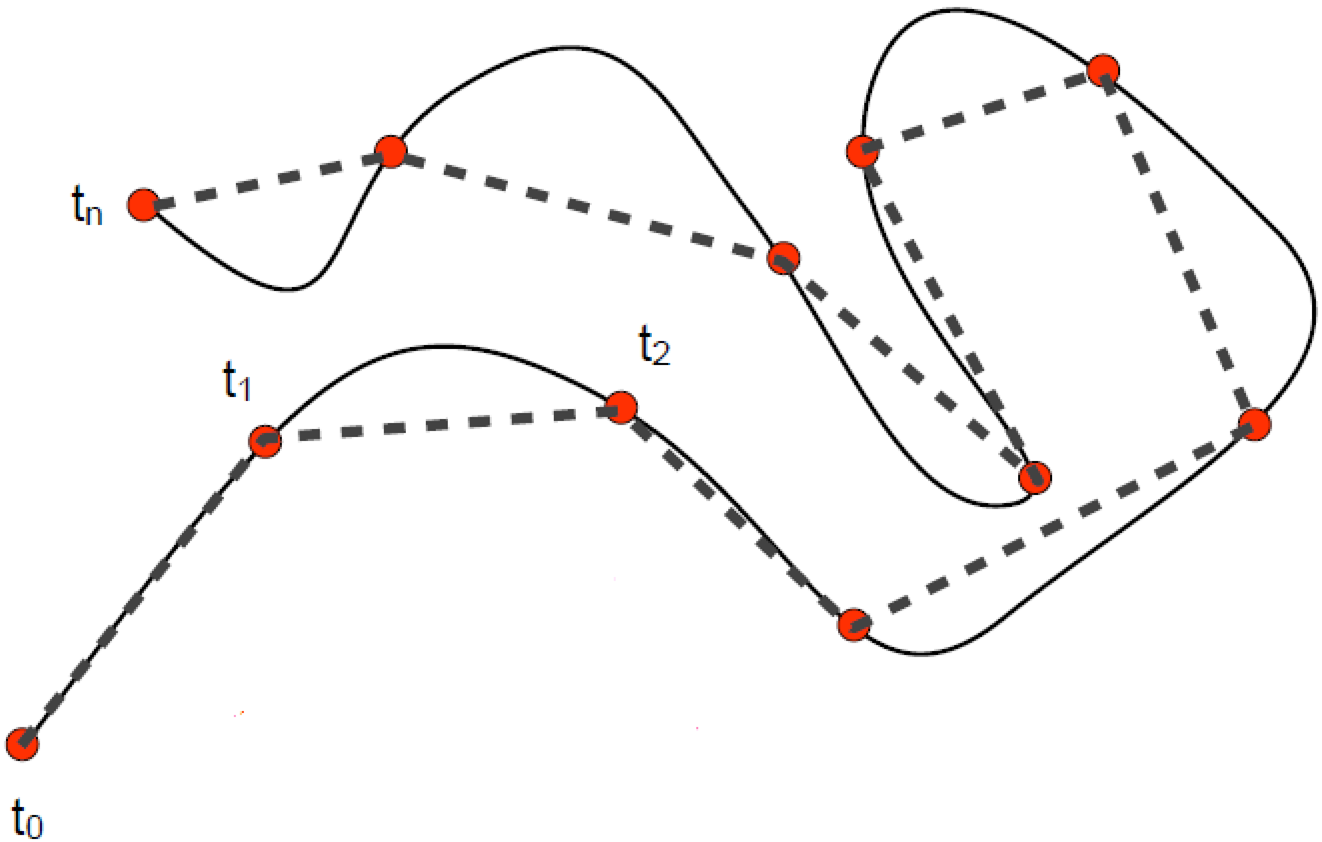
- 函数表达

- 参数表达

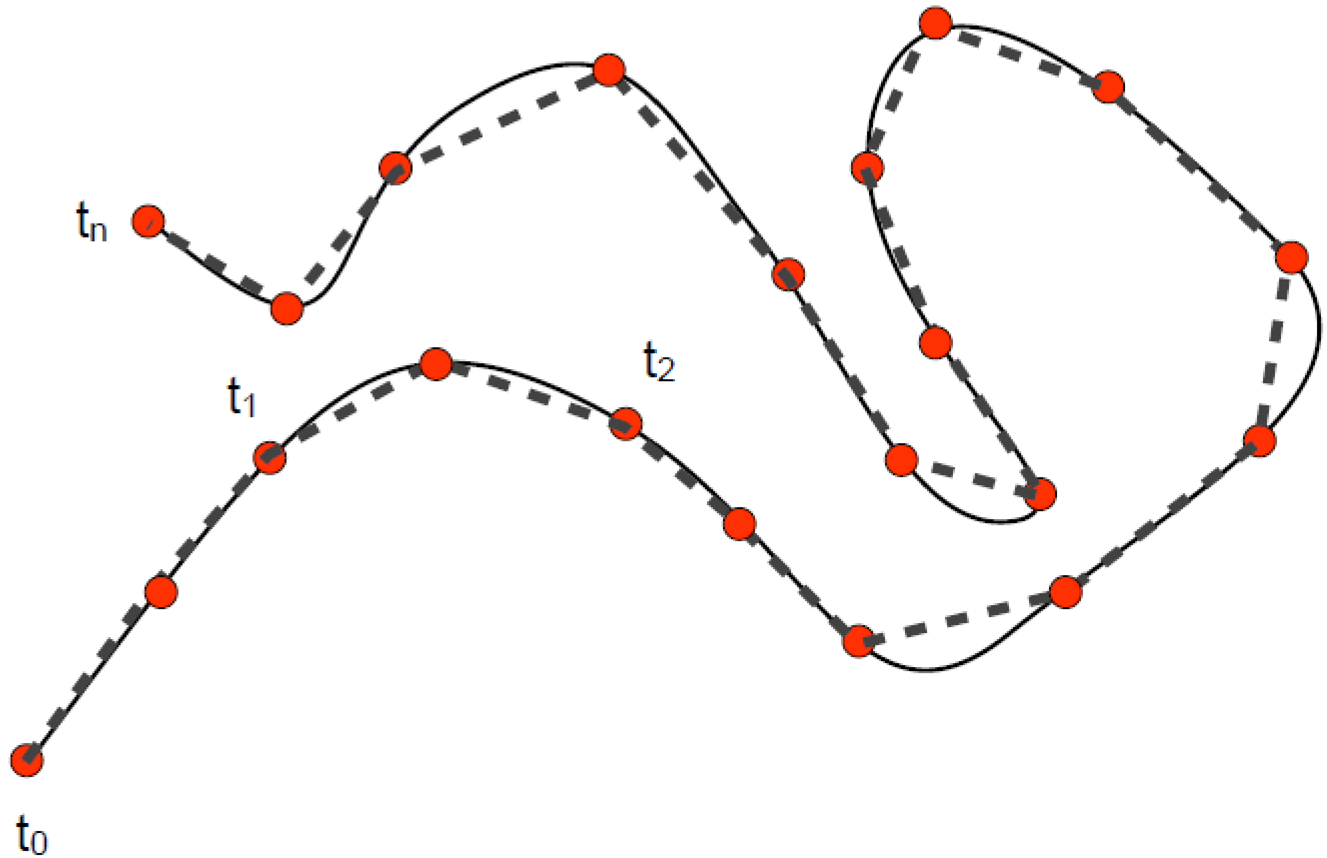


3.3.2 参数曲线的离散：采样

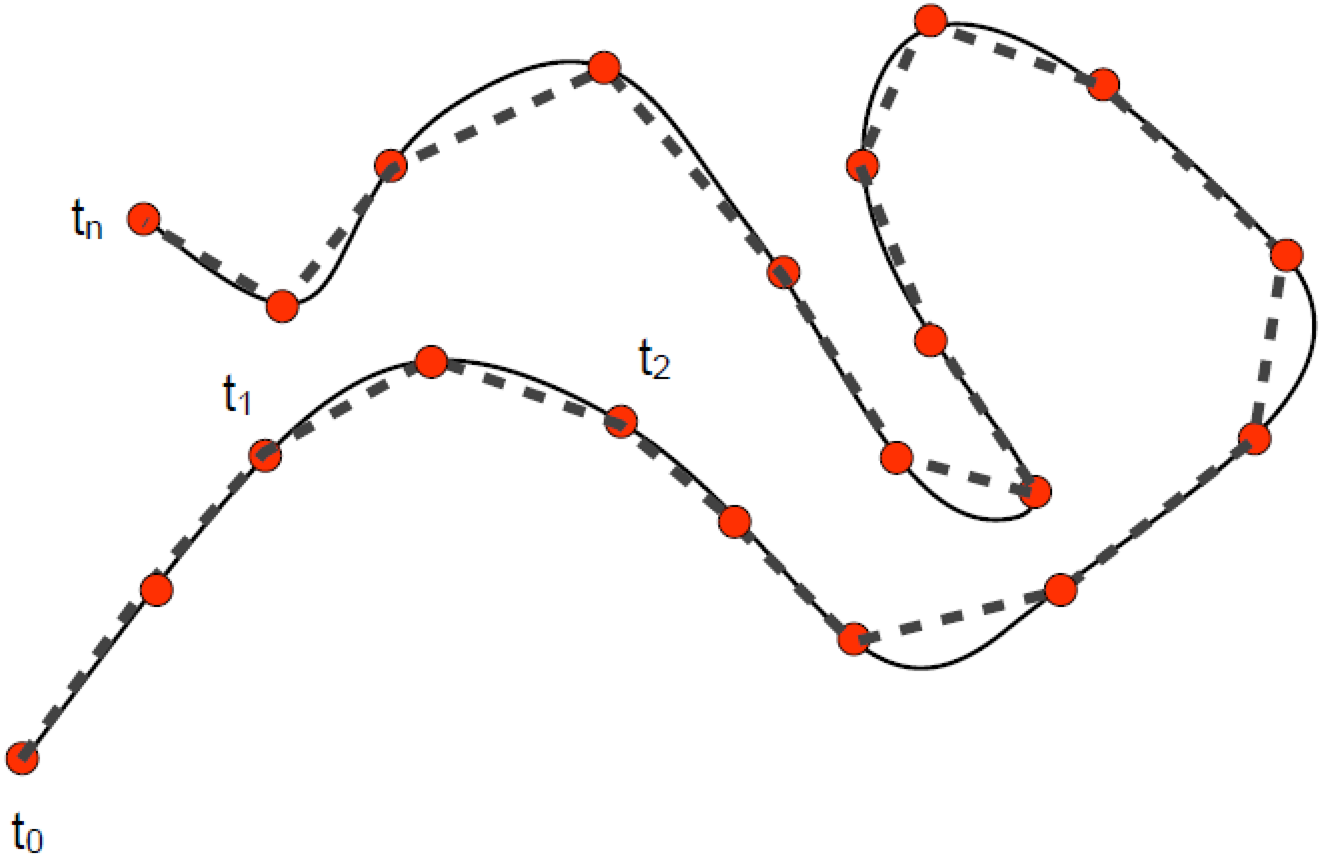
- 分段线性逼近：多边形



曲线的离散：采样



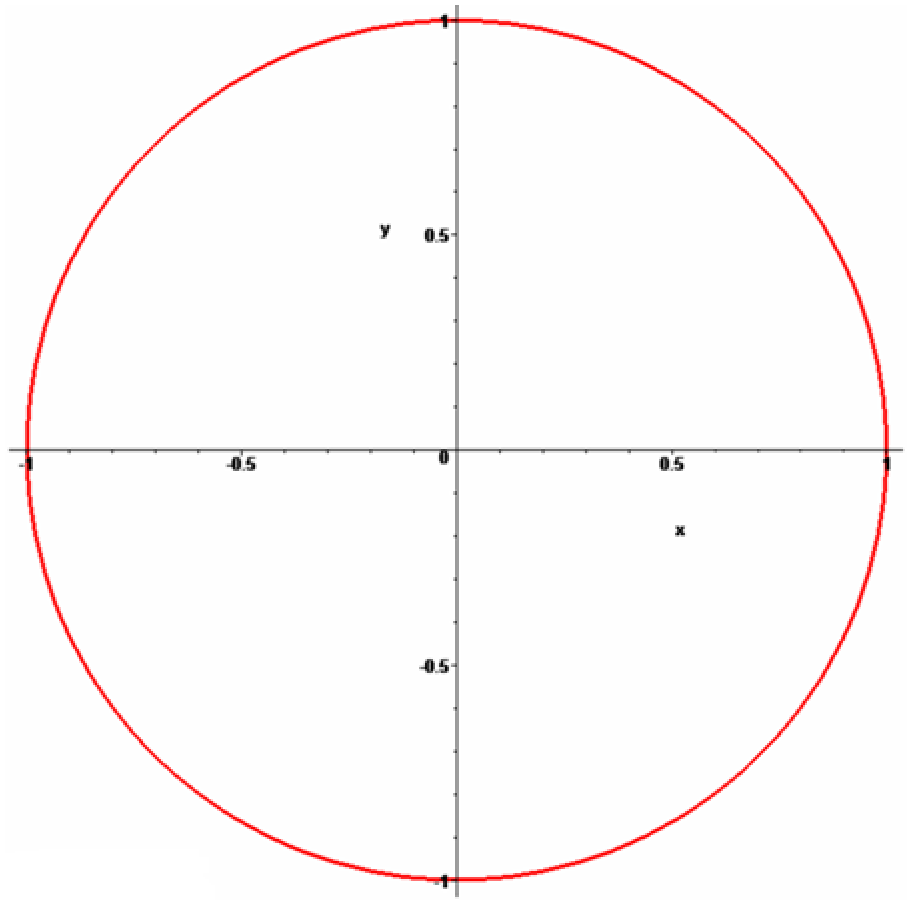
问题：如何采样逼近？



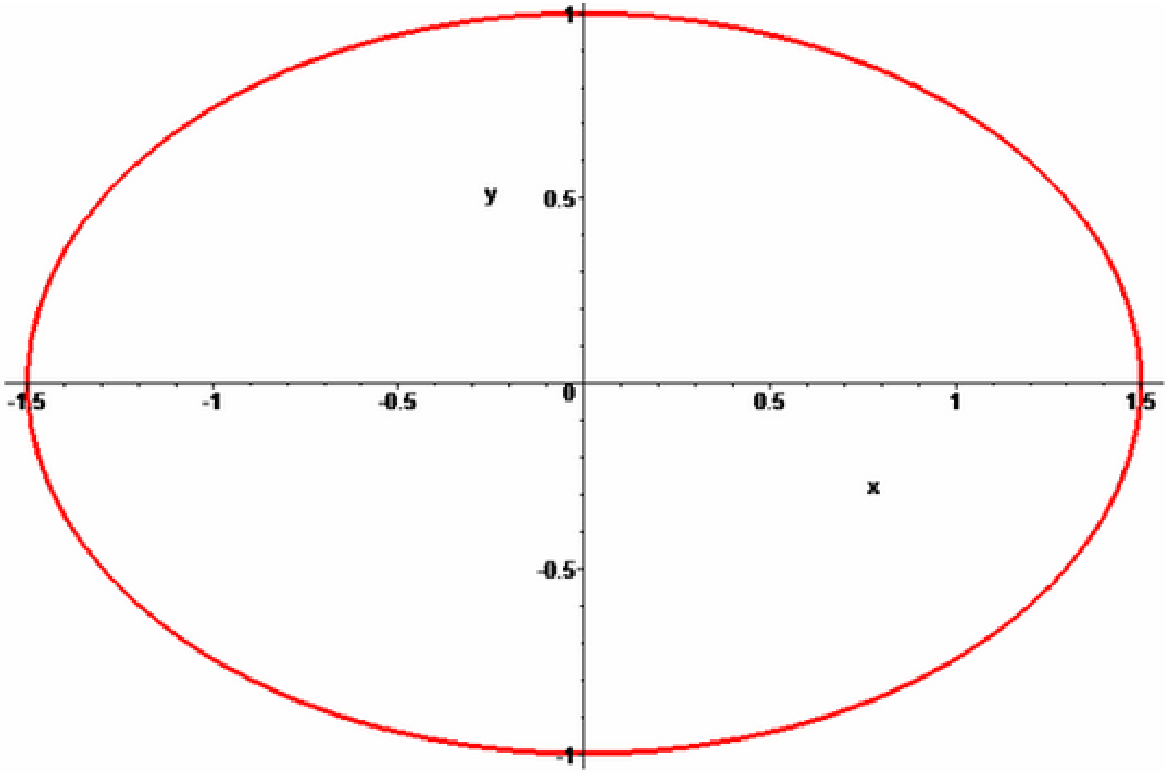
3.3.3 隐函数表达的曲线呢？

$$f(x, y) = 0$$

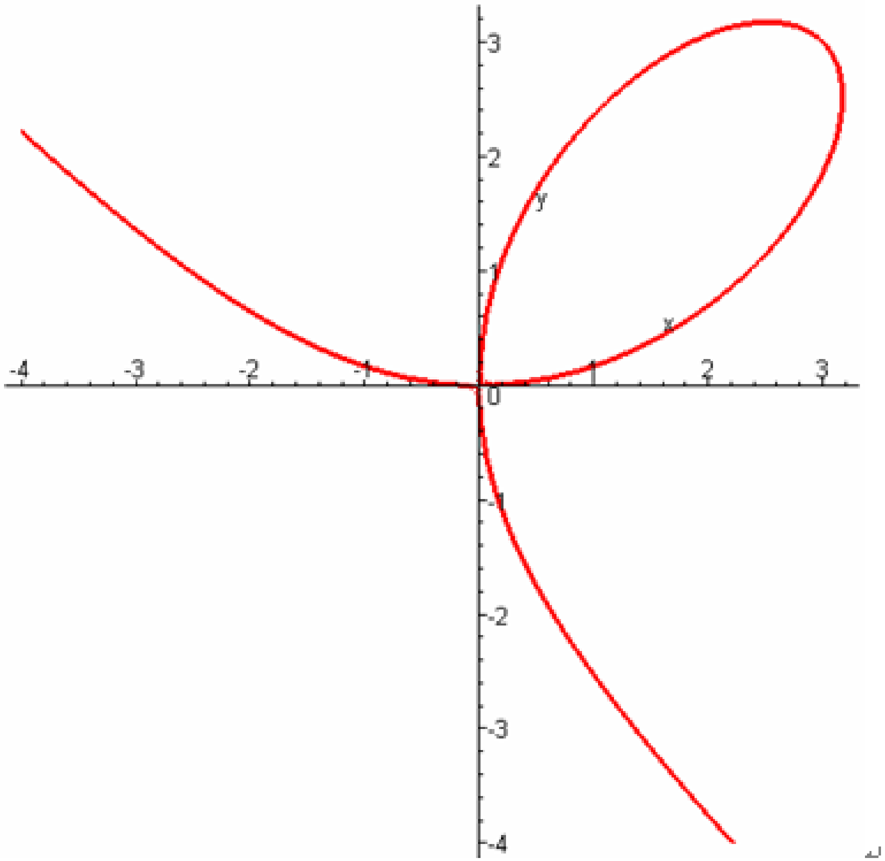
$$x^2 + y^2 = a^2$$



$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

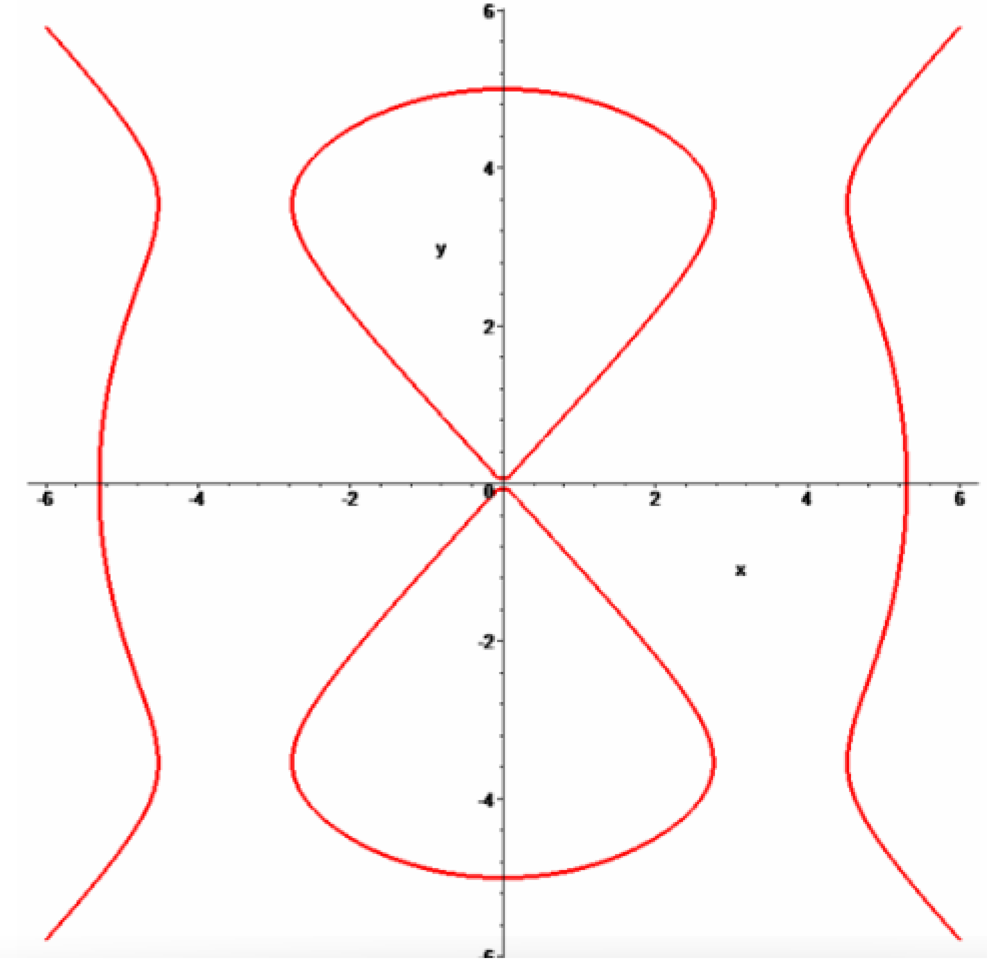


$$x^3 + y^3 = 6xy$$

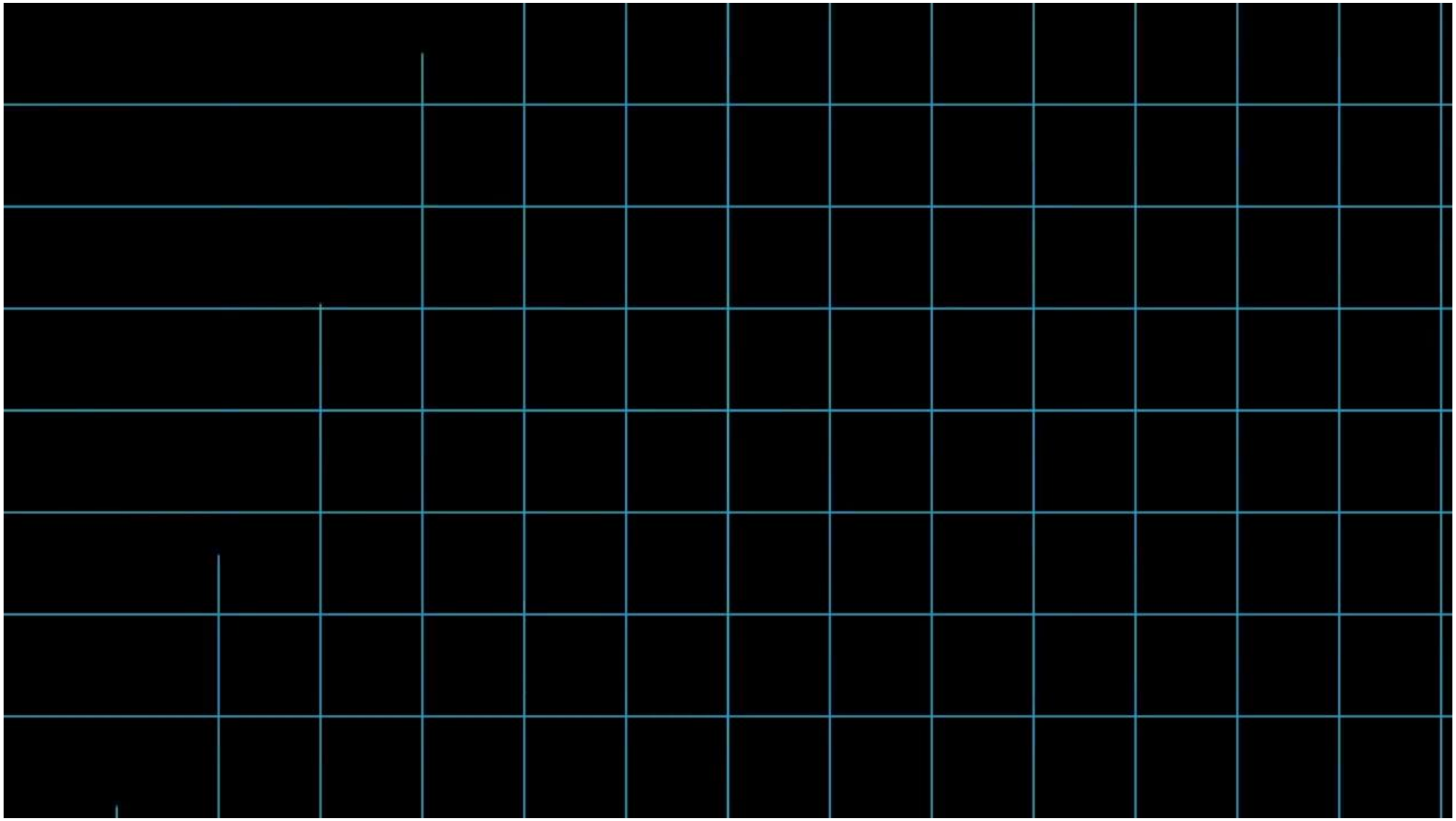


http://matkcy.github.io/MA1104-implicitplot.html

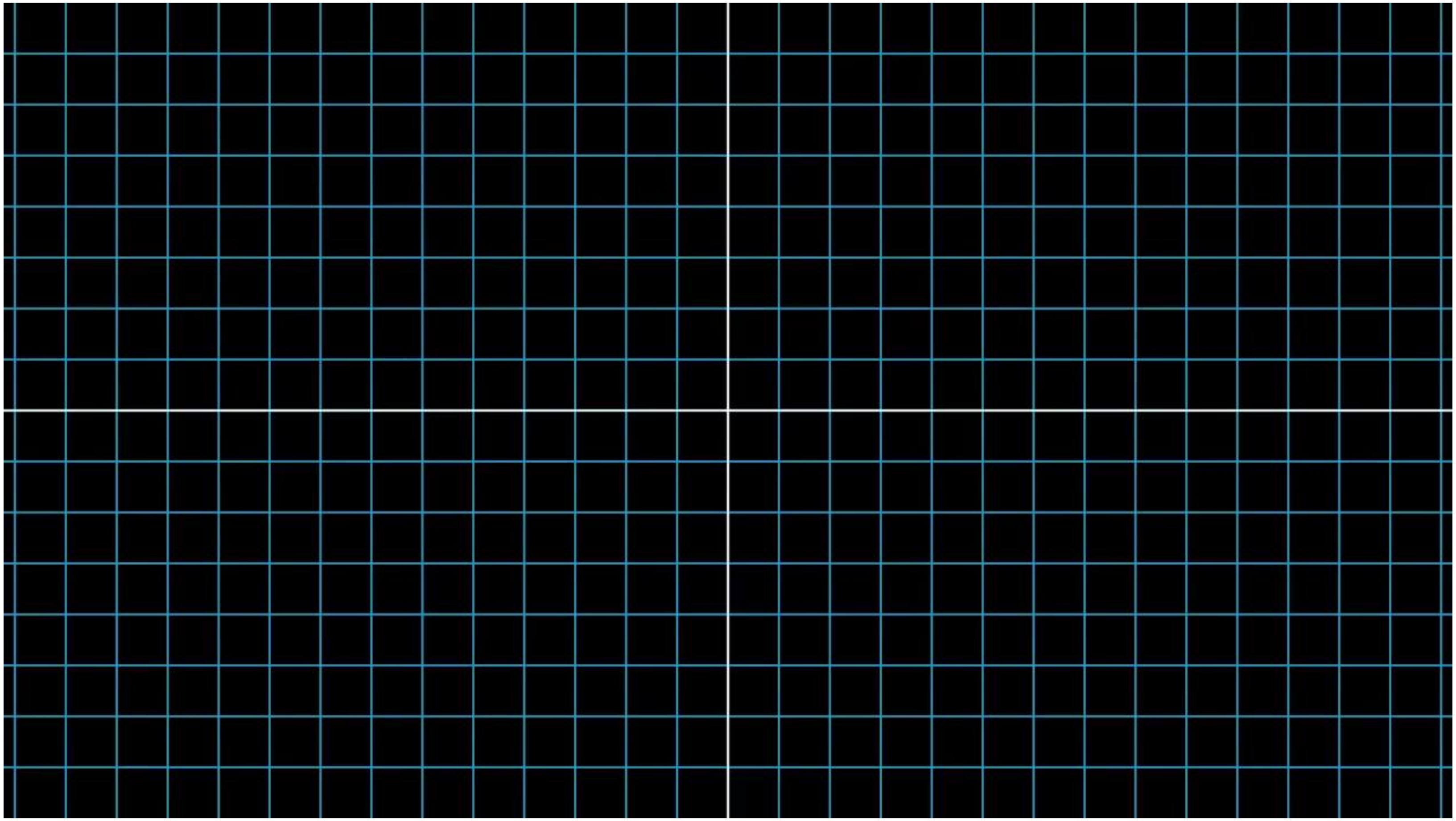
$$y^2(y^2 - a^2) = x^2(x^2 - b^2)$$



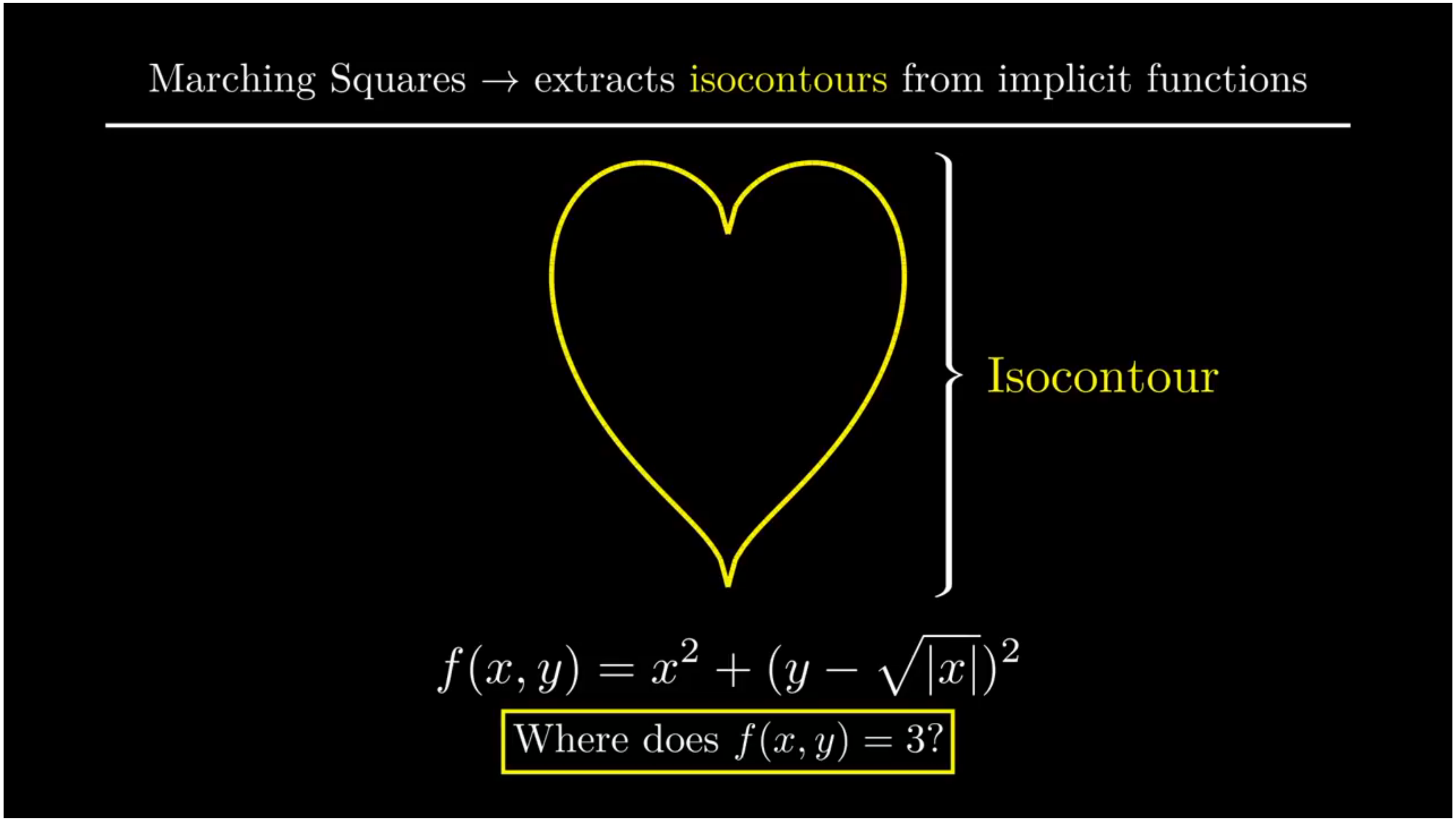
采样



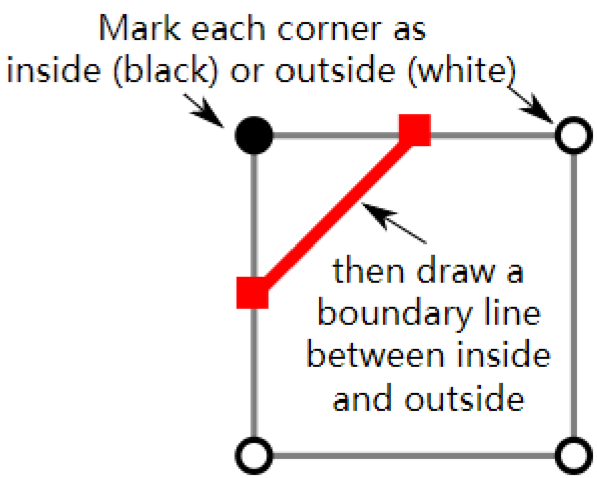
绘制边界

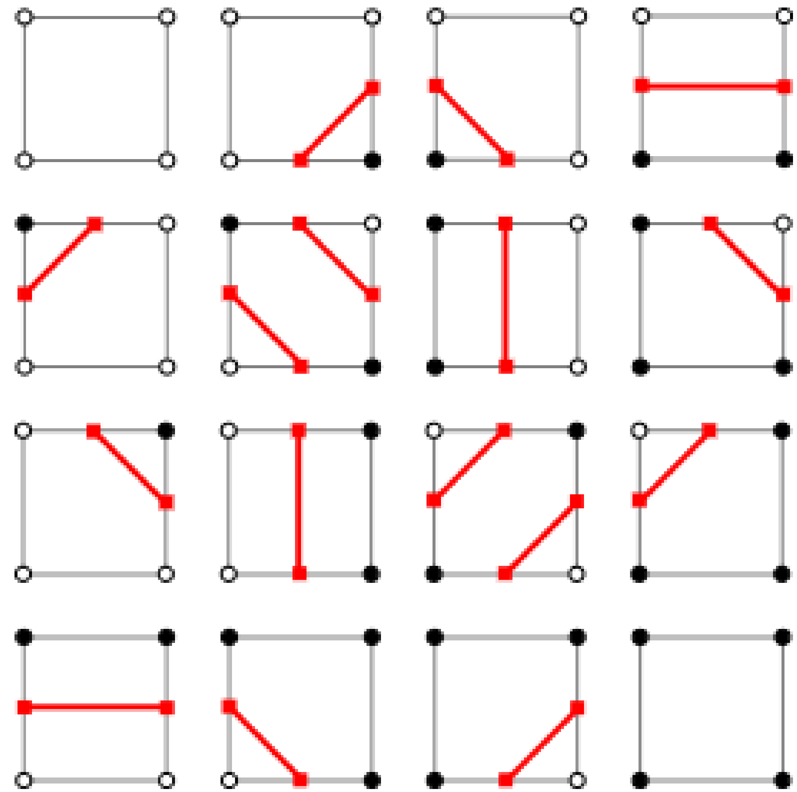


Marching square

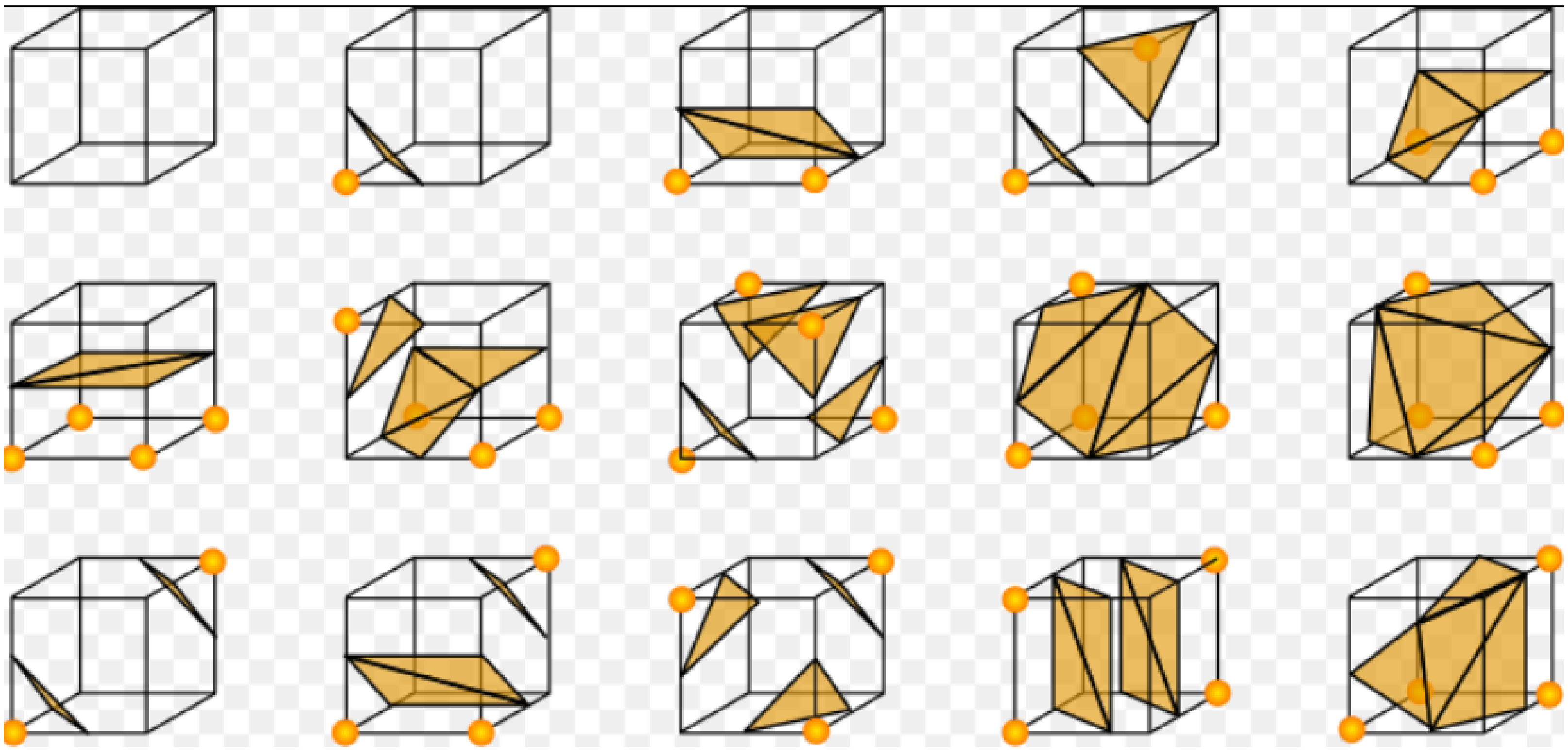


Marching square

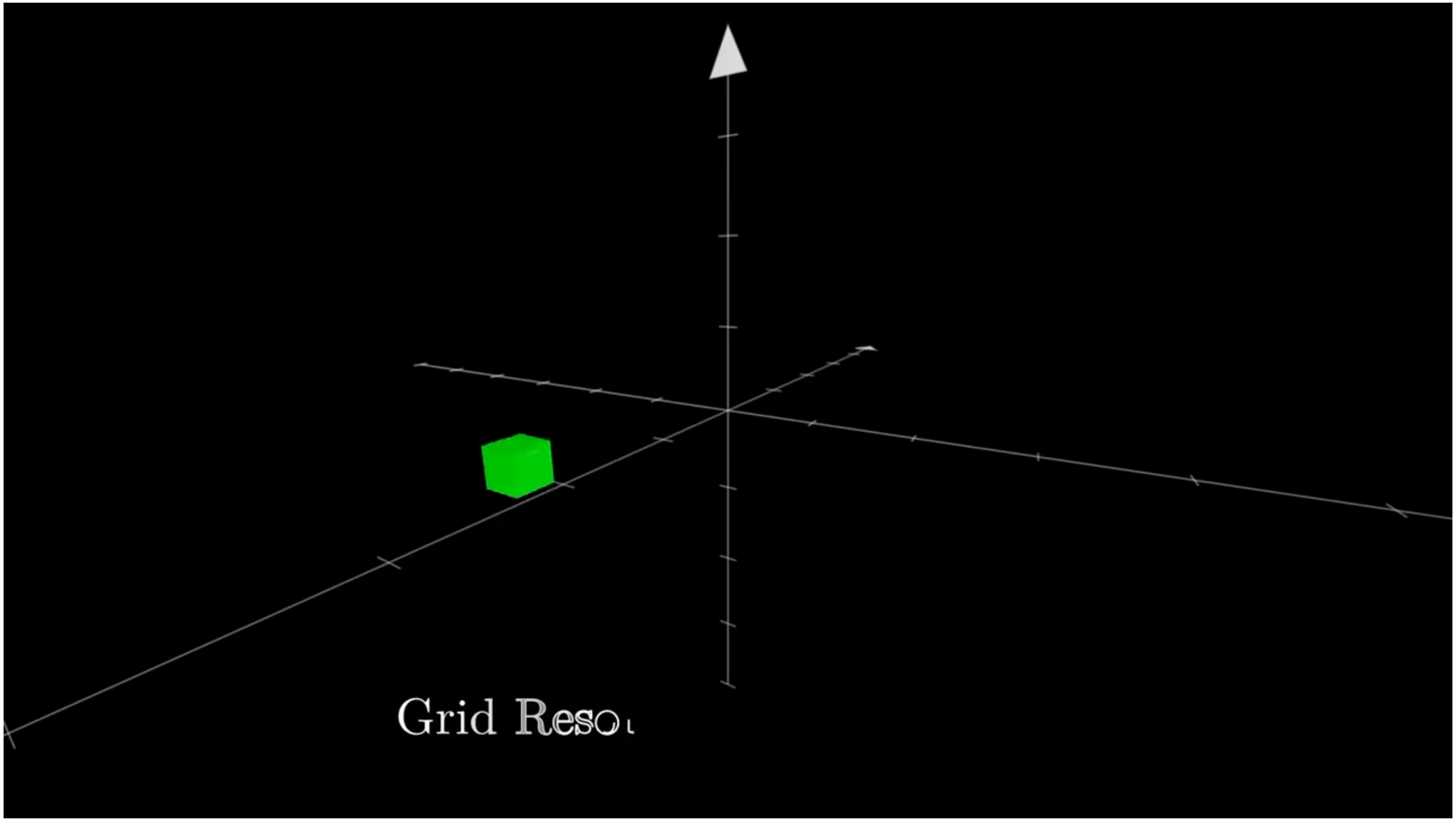




Marching cube



Marching cube



Marching cube