2018-2019年度第二学期 00106501

计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

第三节　光栅化算法

光栅化（rasterization）

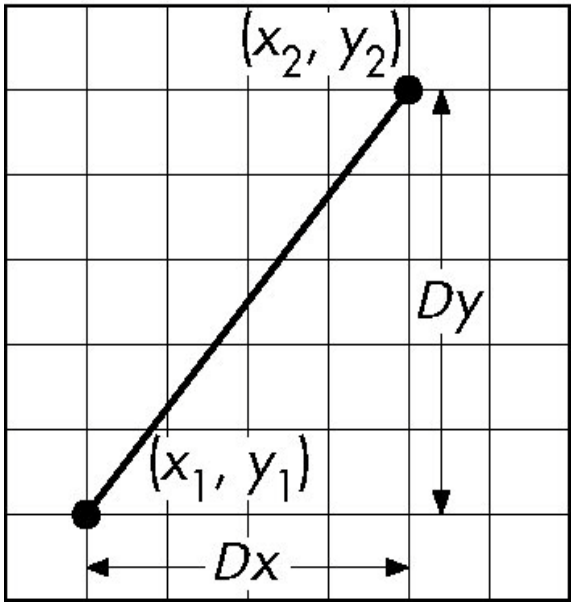
- 亦称为扫描转化（scan conversion）
  - 把用一组顶点定义的对象内部的像素用相应的亮度激活
    - 线段
    - 多边形：扫描转化 = 填充
- 明暗处理的结果由对象的颜色、纹理、以及明暗处理模型确定
- 本节只考虑如何从顶点出发，确定恰当的像素表示几何对象

直线段的扫描转化

- 首先考虑在窗口坐标系中端点坐标为整数的线段
- 假设系统实现中有一个write_pixel函数

m = Dy/Dx

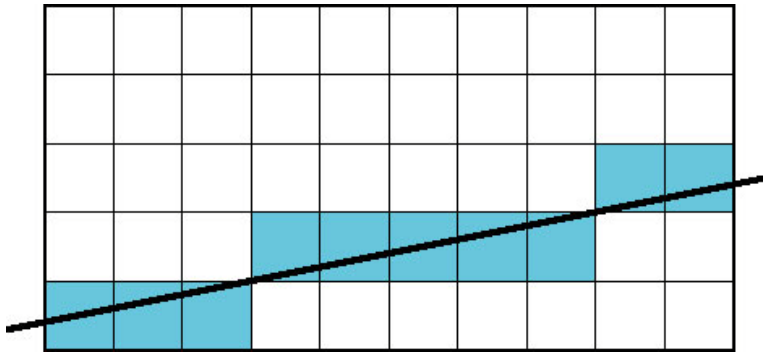
y = mx + h



DDA算法

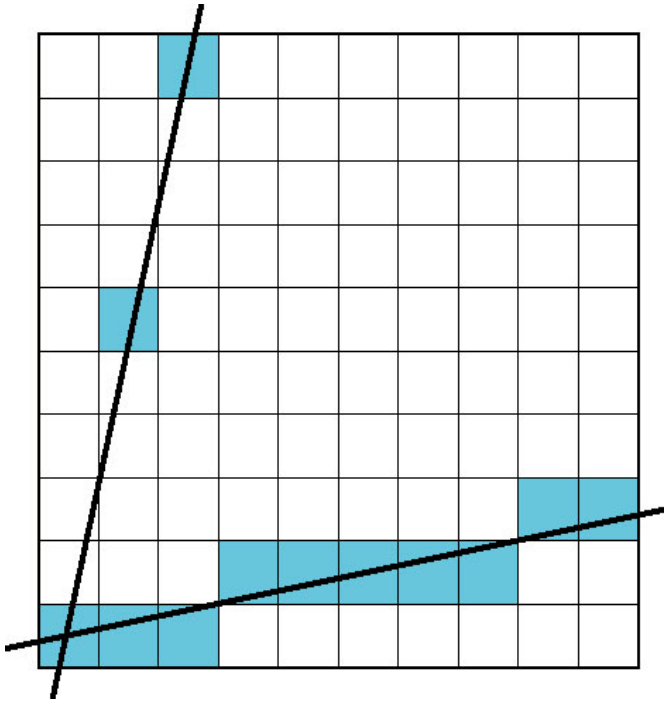
- DDA: Digital Differential Analyzer 数字微分分析法
  - DDA 过去是一个机械设备，用于求解微分方程的数值解
  - 直线 $y = mx + h$ 满足微分方程 $dy/dx = m = Dy/Dx = (y_2 - y_1)/(x_2 - x_1)$
- 沿扫描线 Dx = 1

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



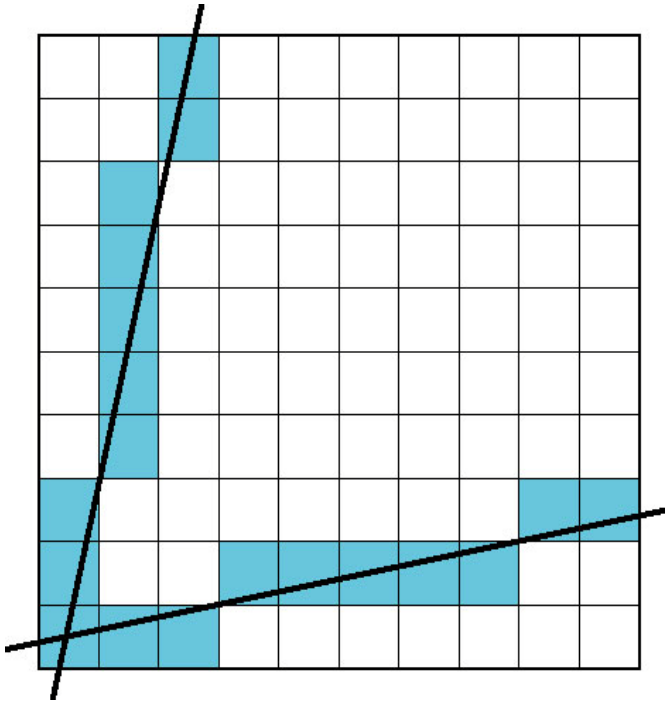
问题

- 对于每个x画出最接近的整数y
  - 对于斜率大的直线有问题



利用对称性

- 只对$0 \le |m| \le 1$的直线应用上述算法
- 对于$|m| > 1$的直线，交换x与y的角色
  - 对于每个y，找出最接近的整数x



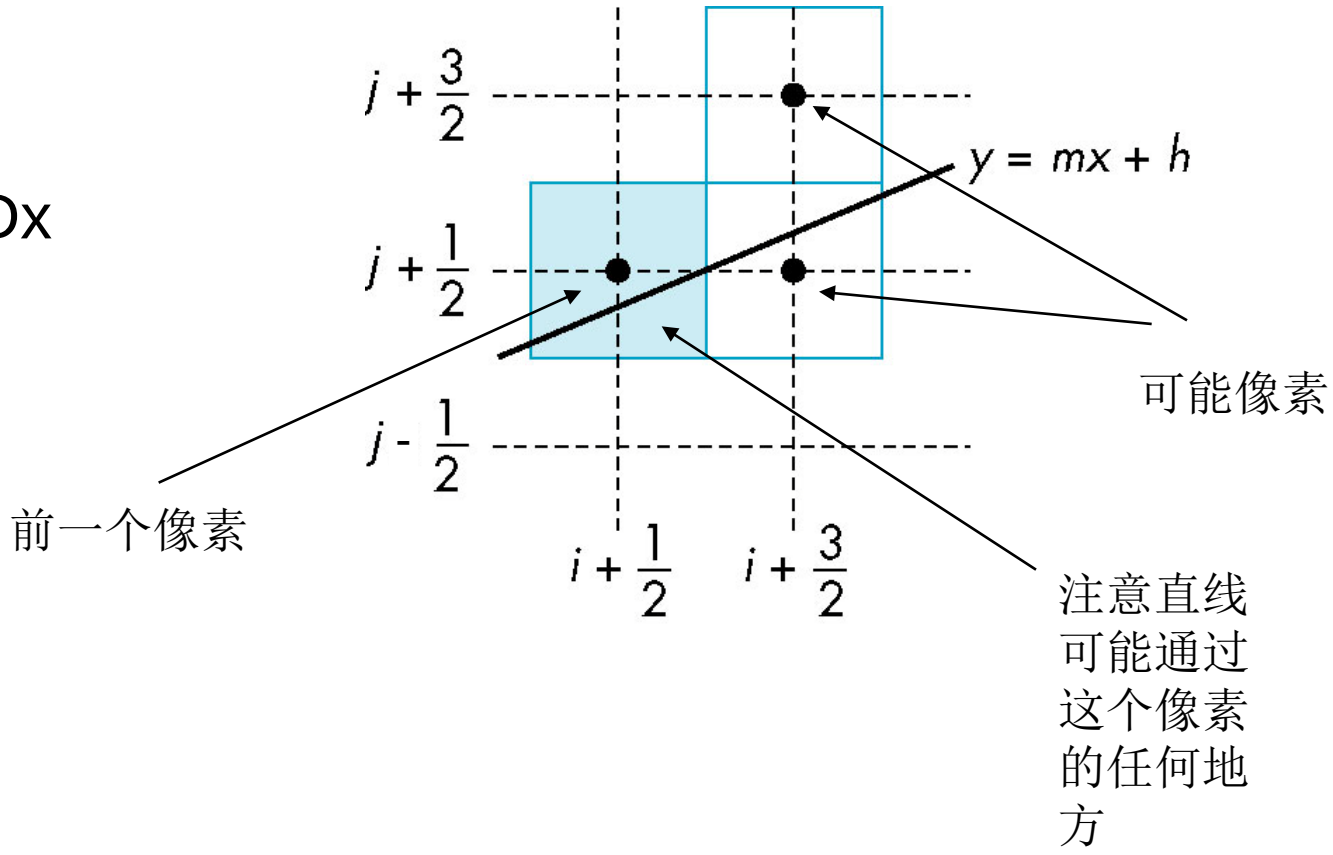
Bresenham算法

- 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

可能像素

$0 \le m \le 1$

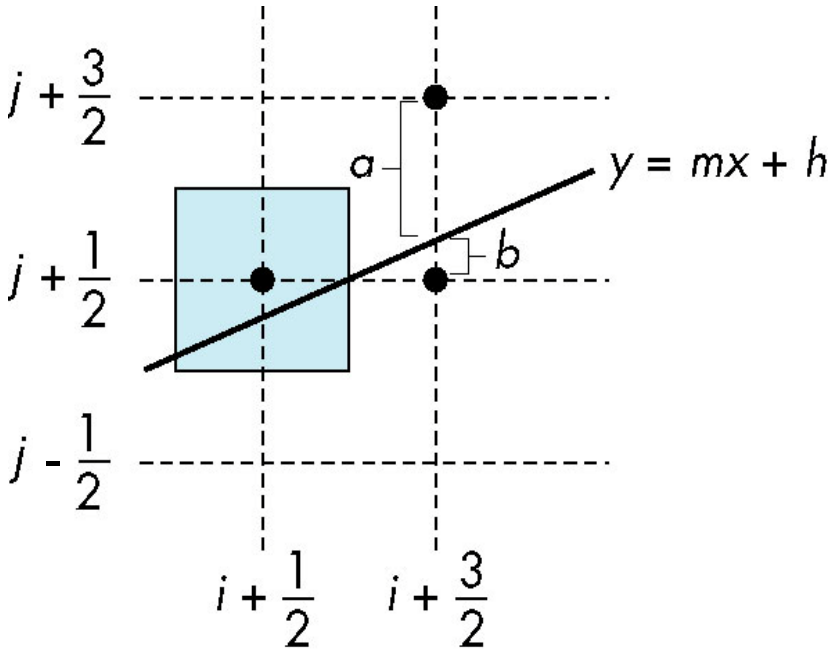
$m = Dy/Dx$



决策变量

d = Dx (a – b)

d为整数
d < 0 采用上像素
d > 0 采用下像素



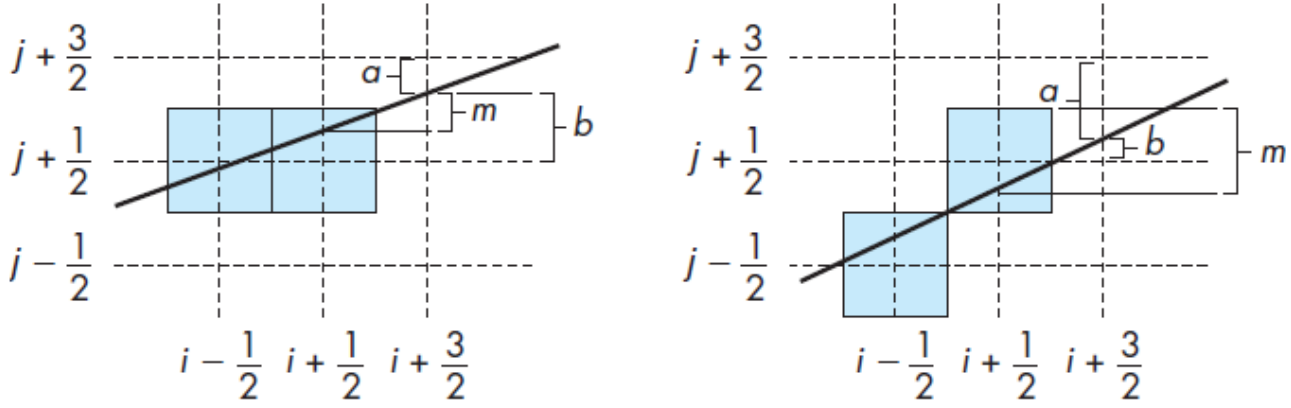
增量形式

- 如果基于第k步的决策变量d_k表示第k+1步的决策变量，可以使算法更有效：

  $d_{k+1} = d_k - 2\,Dy$, 若 $d_k > 0$;

  $d_{k+1} = d_k - 2(Dy - Dx)$, 否则

- 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

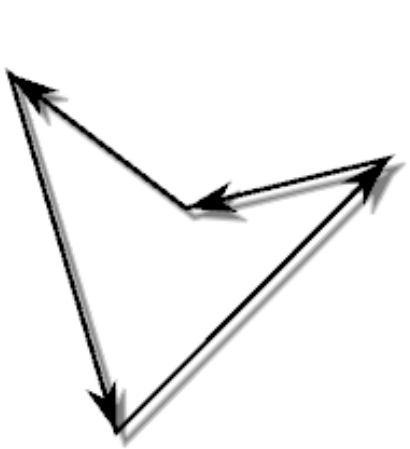


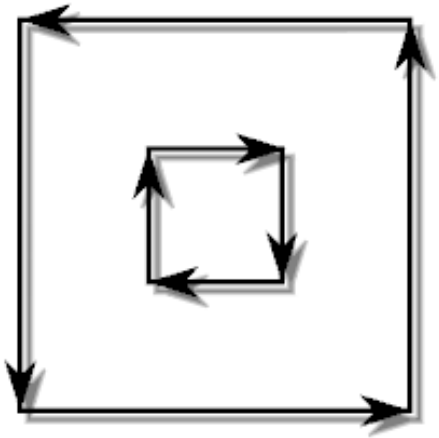
多边形的扫描转换

- 就是多边形的填充
- 第一个光栅系统
  - 可以显示被填充的多边形
  - 当时无法实时给多边形内部每个点着以不同颜色
- 直线的光栅化算法就是Bresenham算法，而多边形的填充算法有许多种
  - 具体选择与系统的实现框架有关

内外检测

- 如何区分内部与外部？
  - 对于凸多边形，很容易做到
  - 对于非简单多边形，就非常困难
  - 可以采用奇偶检测的方法
    - 统计与边界的交点数
  - 环绕次数（winding number）





奇偶检测

- odd-even test，也称为射线法
  - 比较常用
  - 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
  - 如果交点为顶点，需要特别处理
  - 通常射线就是扫描线

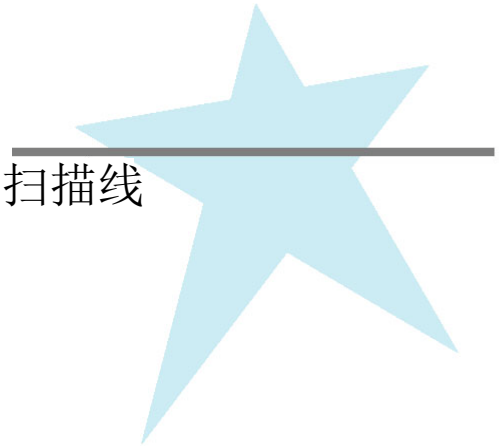
环绕数

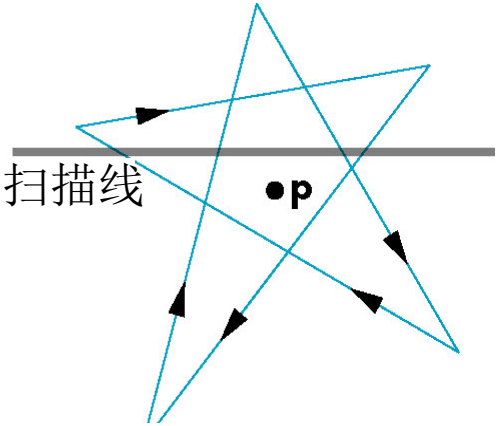
- 首先根据多边形的边界建立一条环路
  - 从一个顶点出发，依次遍历各边，最后回到该顶点
- 一点的环绕数计算

  被多边形各边按上述环路遍历所环绕的次数

  逆时针方向为正，

  顺时针方向为负
- 内外检测
  - 如果环绕数不等于0
    就是内部



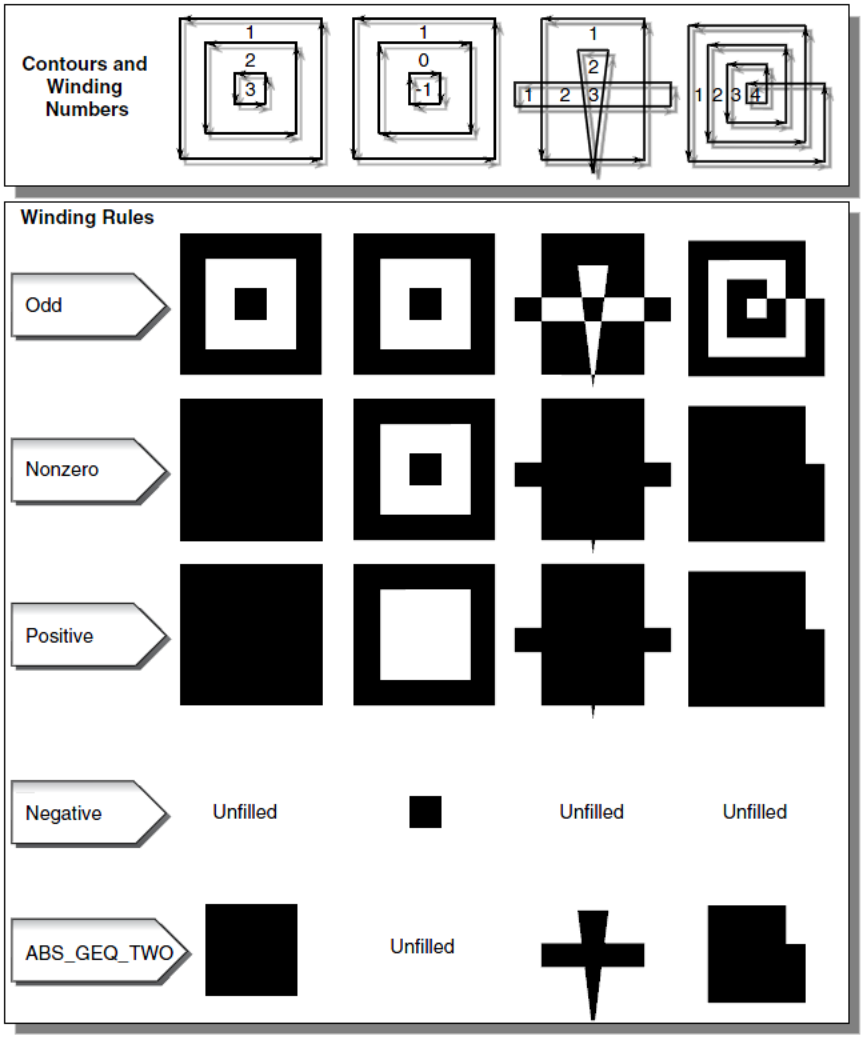


OpenGL与凹多边形

- OpenGL只保证正确填充凸多边形
- 实际应用时由用户保证这条约定被遵守，或者用其它软件把给定多边形剖分为凸多边形
  - 一般结果就是三角形的集合
  - 好的剖分算法应当不生成过长或过细的三角形
  - GLU具有功能实现这种剖分

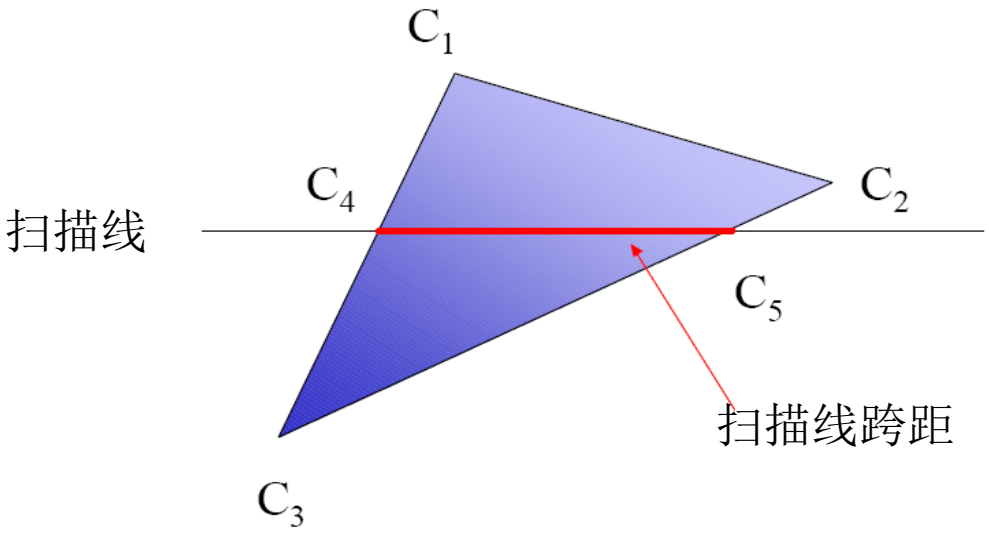
OpenGL与环绕数

- 在OpenGL的剖分算法以及NURBS曲面的裁剪中环绕数有重要应用



明暗处理中的填充方法

- 采用插值方法



在帧缓冲区中的填充

- 在流水线尾部进行填充
  - 只接受凸多边形
  - 非凸多边形需要已被剖分
  - 顶点处的亮度（颜色）已经计算出来（Gouraud明暗处理算法）
  - 与Z缓冲区算法结合在一起
    - 跟踪扫描线，插值亮度
    - 增量方法的应用不大

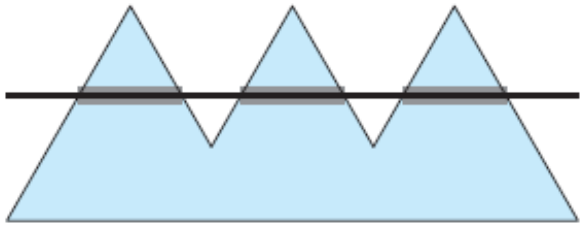
种子填充方法

- 如果已知位于多边形内部（WHITE）的一个点，那么可以递归填充
- 把多边形内部转化为内部要填充的颜色（BLACK）

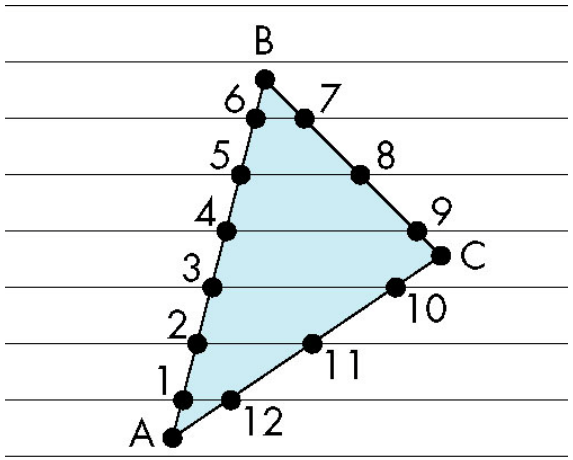
```
flood_fill(int x, int y){
    if(read_pixel(x,y)==WHITE){
        write_pixel(x,y,BLACK);
        flood_fill(x-1,y);
        flood_fill(x+1,y);
        flood_fill(x,y+1);
        flood_fill(x,y-1);
    }
}
```

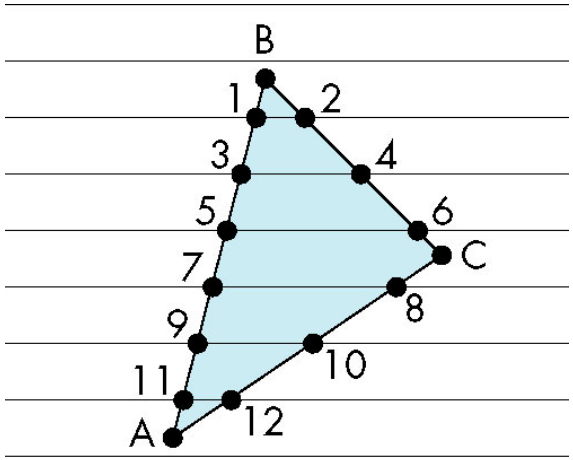
扫描线填充

- 通过维持一个特别的数据结构（结构中保存扫描线与多边形的交点）进行填充
  - 按扫描线进行排序
  - 逐扫描线跨距(span)进行填充

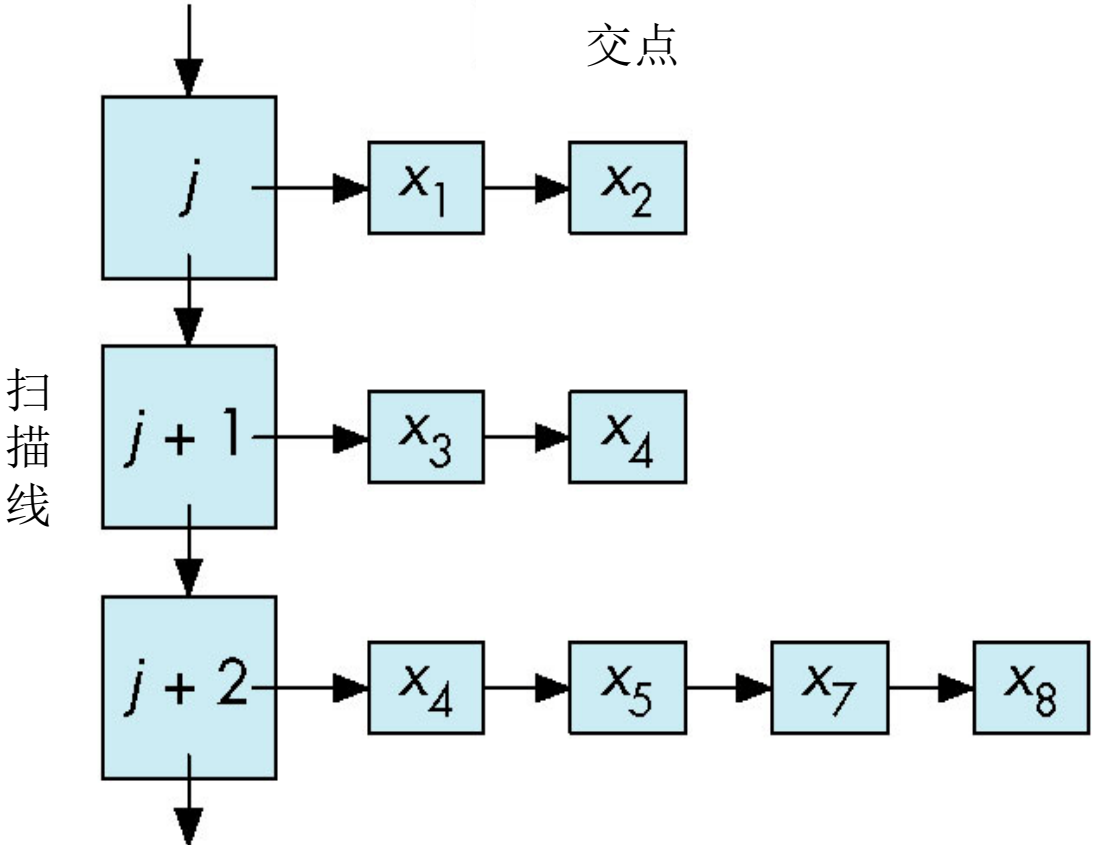


通过顶点列表生成的顶点顺序



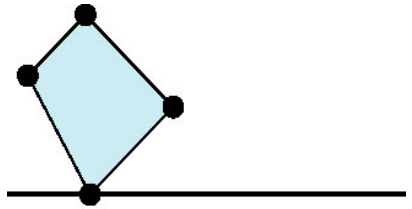


所期望的顺序

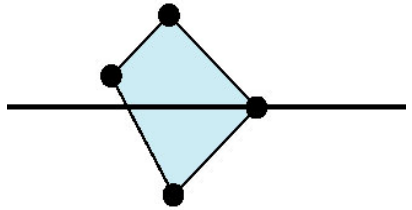

交点
扫描线
j
j + 1
j + 2
x1
x2
x3
x4
x4
x5
x7
x8

奇异情形

- 可以把大多数多边形填充算法应用到其它形状上
  - 必要的细节考虑
- 即使对于多边形，对于某些算法也需要仔细考虑
  - 奇偶检测方法



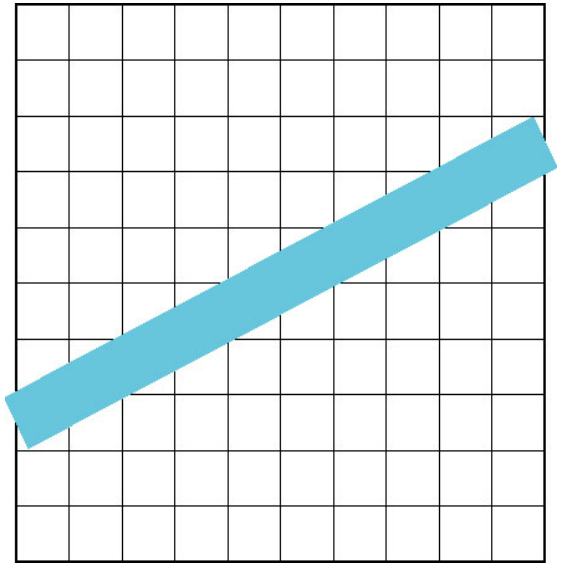
(a)



(b)

走样

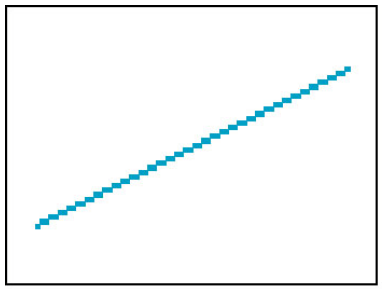
- 理想的光栅化直线应当是一个像素宽



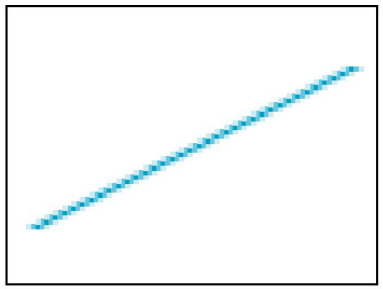
- 对于每个x选择最佳的y（或者反过来）会导致走样的光栅化直线

通过面积平均进行反走样

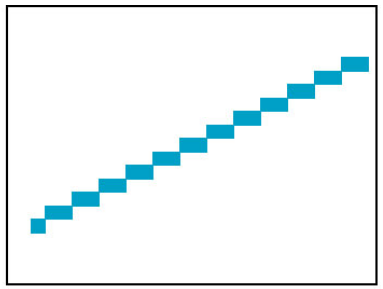
- 对于每个x，把理想直线所覆盖的像素面积乘以颜色

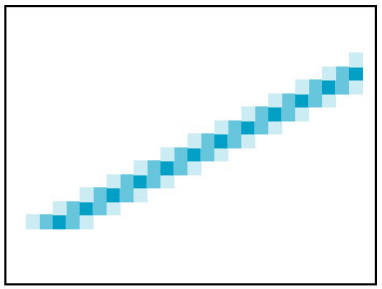


初始形状



反走样的结果

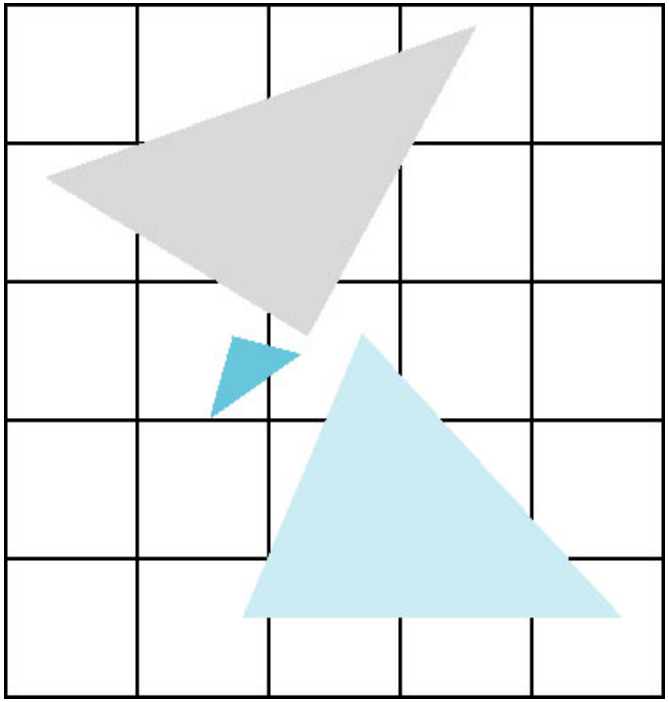




放大显示的结果

多边形的走样

- 对于多边形，走样问题可能非常严重
  - 边的锯齿形状
  - 小多边形被忽略
  - 需要进行颜色组合，从而可能一个多边形的颜色并不完全确定一个像素的颜色

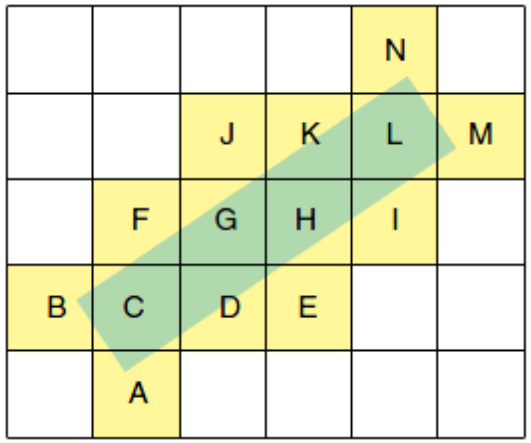


OpenGL中的反走样

- 多重采样（Multisampling，MSAA）
  - 在位于多边形边上的像素位置进行多重采样，可使绘制的图像边缘更加平滑
  - 与超采样（supersampling，SSAA）不同，多重采样对每个pixel只执行一次片元着色器（注：OpenGL 4.0及以上，支持对每个样本都执行一次片元着色器，通过glEnable(GL_SAMPLE_SHADING)开启）
  - 对每个样本，多重采样执行多次深度/模板测试，从而提升了子像素的空间采样精度
  - 启用方法
    - 申请多重采样缓存
    - glEnable(GL_MULTISAMPLE)；
  - 用途：主要用于多边形或点的反走样

OpenGL中的反走样

- 线段的反走样
  - 通过计算覆盖率＋融混来实现
  - 启用方法
    - glHint (GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
    - glEnable (GL_LINE_SMOOTH);
    - glEnable (GL_BLEND);
    - glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);



Thanks for your attention!