扫描转换与区域填充

冯结青

浙江大学 CAD&CG国家重点实验室

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形扫描转换与区域填充的比较

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较

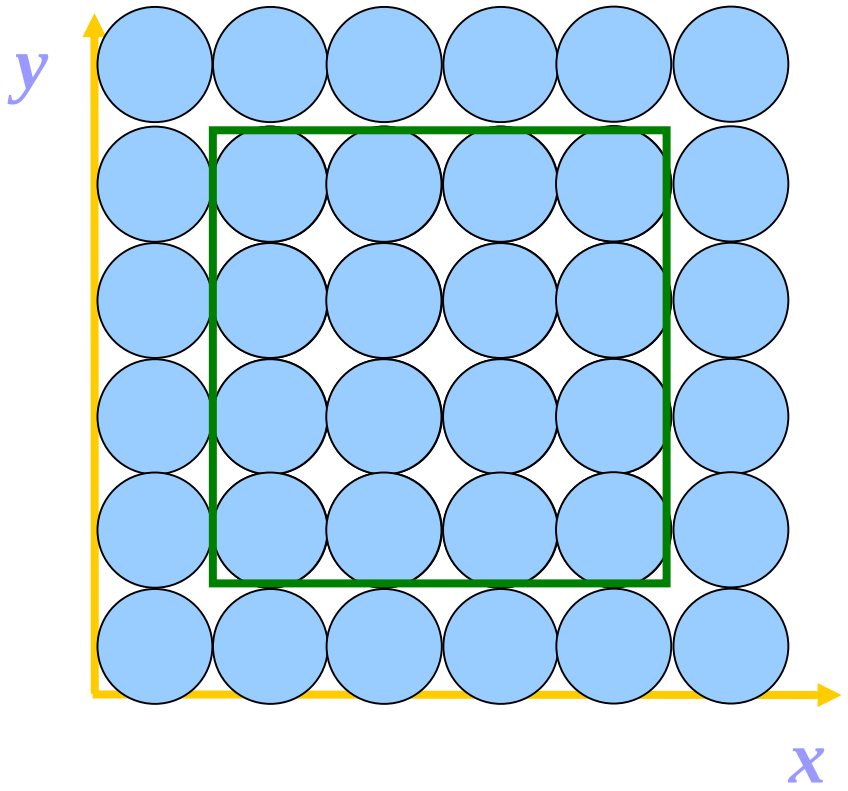
多边形扫描转换的边填充算法

- 扫描线算法的不足
  - 数据结构复杂
  - 建立ET、AEL时，需要进行排序
- 边填充算法：利用求补运算~消除排序
  - 求补运算

    $\bar{M}=A\sim M$

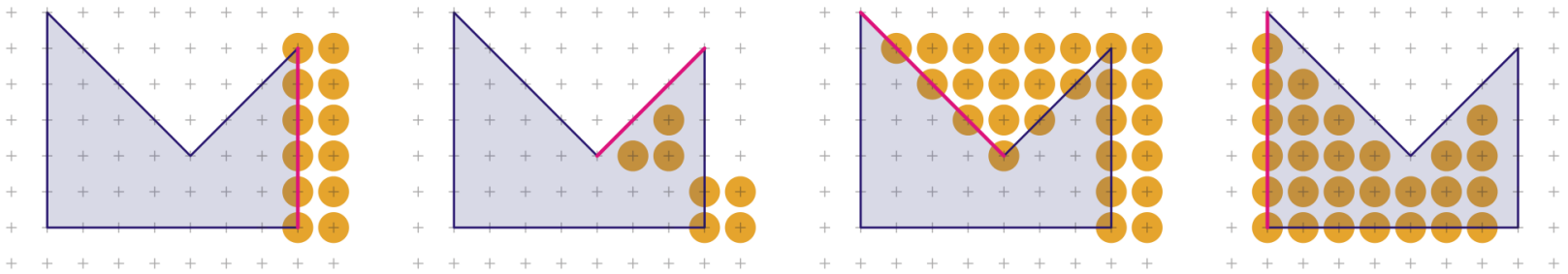
    $M=\bar{\bar{M}}=A\sim\bar{M}=A\sim(A\sim M)$
  - A~：填充一次，多边形颜色
  - A~~：填充两次，背景颜色

求补运算实例

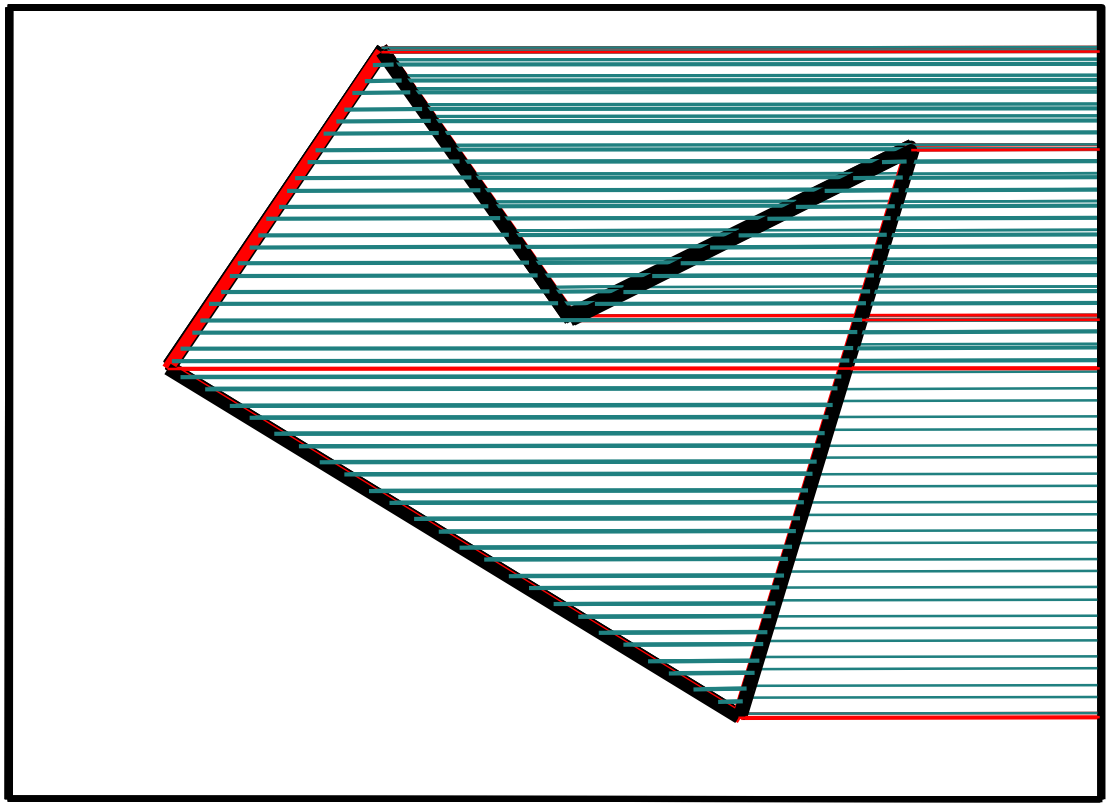


边填充算法的过程

- 按任意顺序处理多边形的每条边
- 在处理每一条边时
  - 求出该边与扫描线的交点
  - 将每一条扫描线上交点右方的所有像素求补
- 多边形的所有边处理完毕之后，填充完成



边填充算法的示意图



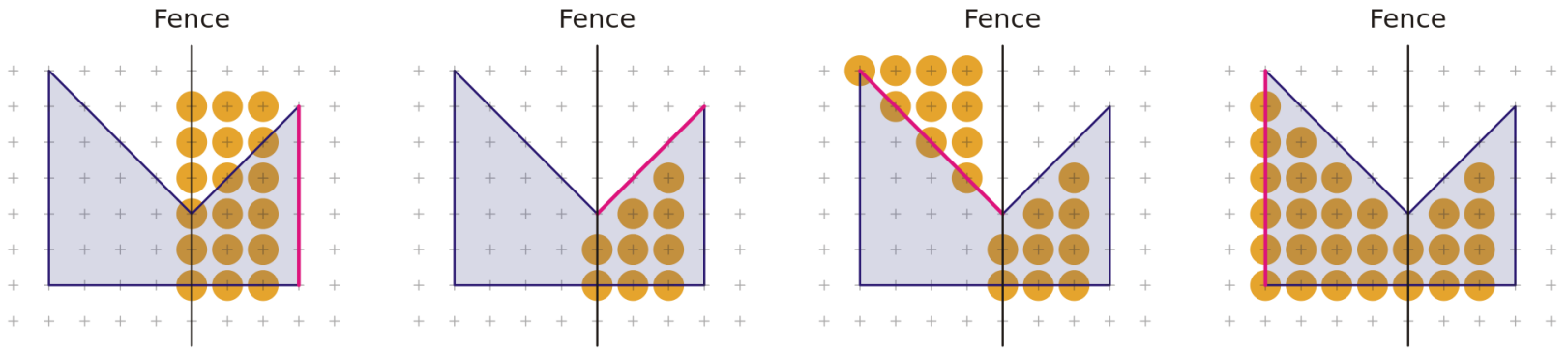
边填充算法示意图

边填充算法讨论

- 边填充算法多边形的要求？
- 优点：数据结构简单，适用于具有帧缓存的图形系统
- 缺点：对于复杂图形，每一像素可能被访问多次，输入输出量比扫描线算法大很多

栅栏边填充算法

- 对于多边形的每条边
  - 对于每条与该边相交的扫描线：交点x_{inter}
  - if $x_{inter} \geq x_{fence}$，像素：$x_{fence} \leq x_{pixel} < x_{inter}$求补
  - if $x_{inter} < x_{fence}$，像素：$x_{inter} \leq x_{pixel} < x_{fence}$求补



栅栏边填充算法

- 栅栏边填充算法的交点？
- 如何选择栅栏边？
- 栅栏边是否越多越好？

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较

多边形扫描转换的边标志算法

- 目标
  - 减少每个像素的访问次数
  - 保持算法的简单性
- 思想
  - 首先用特殊颜色将多边形的边标记出来
  - 再将多边形内各个扫描线区间着色

边标志算法

步骤1：勾画边界

边界颜色: bdry_color

多边形为 $P_i=(x_i,y_i)$, i=0,1,…,n.

x_i,y_i为整数; $P_0=P_n$

思考：勾画边界算法与Bresenham算法结果是否一致？有何区别？

```
for ( i = 0; i < n; i++ ) {
    Δy = y_{i+1}-y_i;
    if  (Δy ≠ 0) {
        Δx = (x_{i+1}-x_i) / Δy;
        if (Δy > 0) {
            x=x_i;   ymin=y_i;   ymax=y_{i+1};
        }
        else{
            x=x_{i+1}; ymin=y_{i+1}; ymax=y_i;
        }
        for (y=ymin+1; y≤ymax; y++) {
            x = x + Δx ;
            setpixel (framebuffer, x, y, bdry_color);
        }
    }
}
```

边标志算法

步骤2：逐条扫描线对多边形内部着色

边界颜色: bdry_color
多边形内颜色：plyg_color
背景颜色：bkgd_color

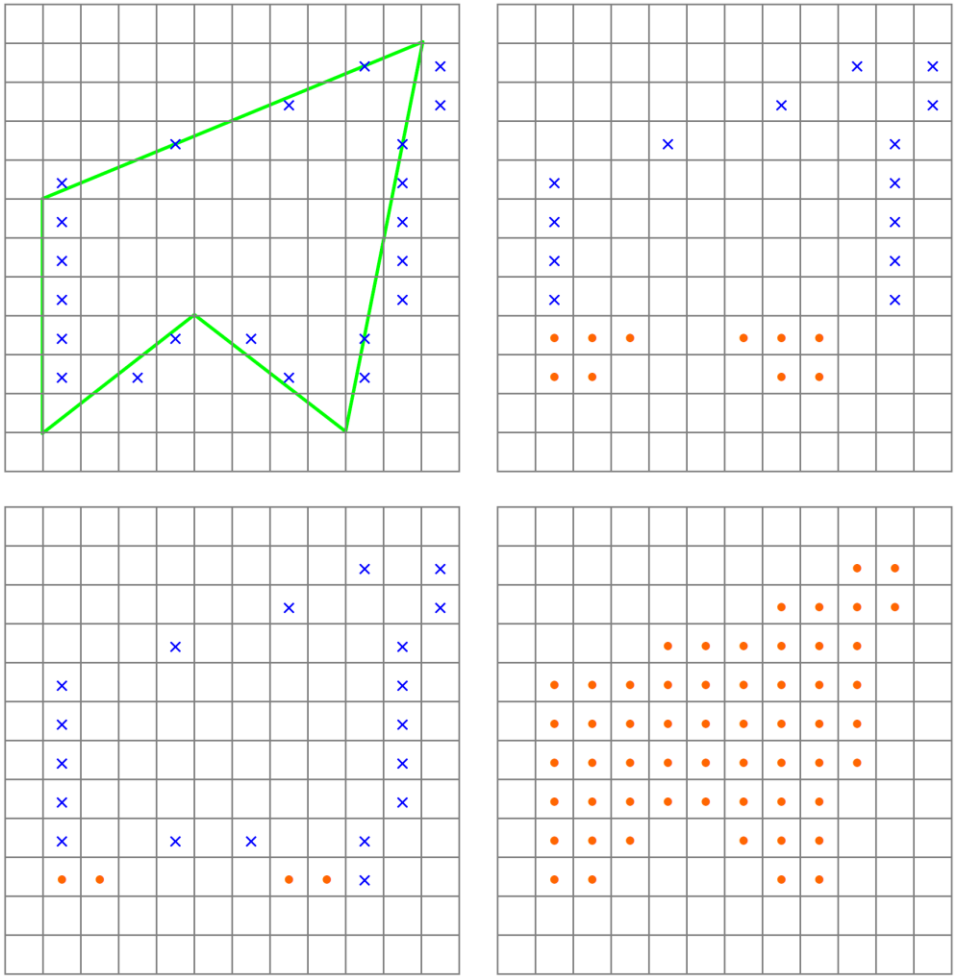
getpixel(framebuffer,x,y): 返回帧缓存中像素(x,y)的颜色值

多边形包围盒：
[Xmin, Xmax; Ymin, Ymax]

int_flag：像素是否位于多边形内部标记，内部true

```
for ( y = Ymin; y ≤ Ymax; y++ ) {
    int_flag = false;
    for  (x = Xmin; x ≤ Xmax; x++ ) {
        if (getpixel(framebuffer,x,y)== bdry_color)
            int_flag = ! int_flag;
        if (int_flag )
            setpixel (framebuffer, x, y, plyg_color);
        else
            setpixel (framebuffer, x, y, bkgd_color);
    }
}
```

边标志算法示意图



边标志算法讨论

- 每个像素最多访问两次
- 实现效率高
  - 软件实现：与扫描线算法效率一样高
  - 硬件实现：比扫描线算法效率高一倍

主要内容

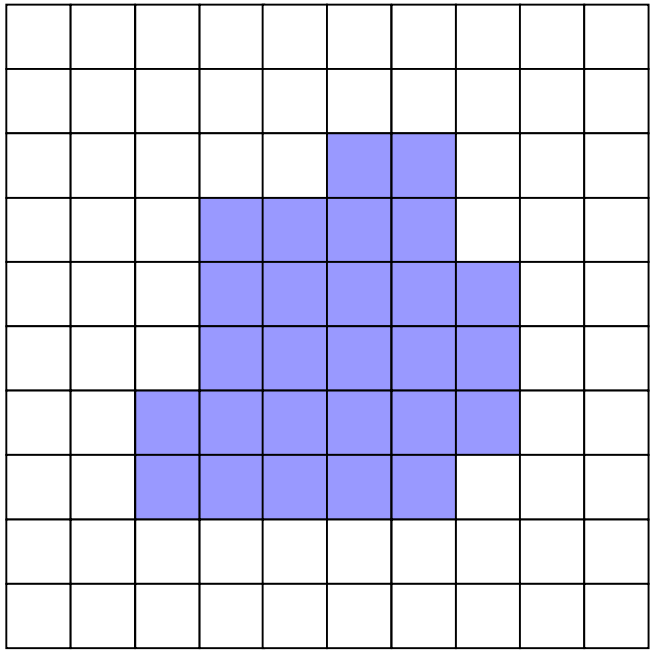
- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较

点阵表示的区域填充

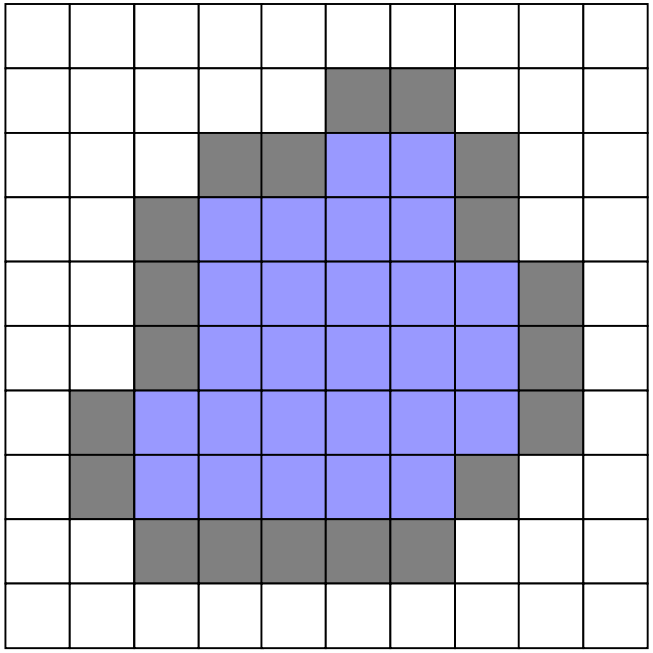
- 区域的定义：已经表示成点阵的像素集合
- 区域的表示：
  - 内部表示：把给定区域内的像素枚举出来
    - 区域内所有像素都着同一种颜色
    - 区域边界像素不能着上述颜色
  - 边界表示：把区域边界上的像素枚举出来
    - 边界上所有像素都着同一种颜色
    - 区域内部像素不能着上述颜色

区域表示

- 内部表示

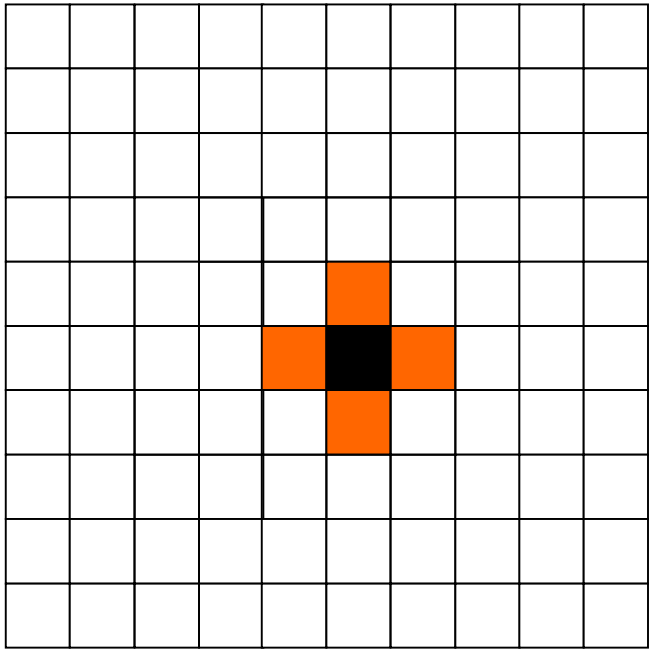


- 边界表示

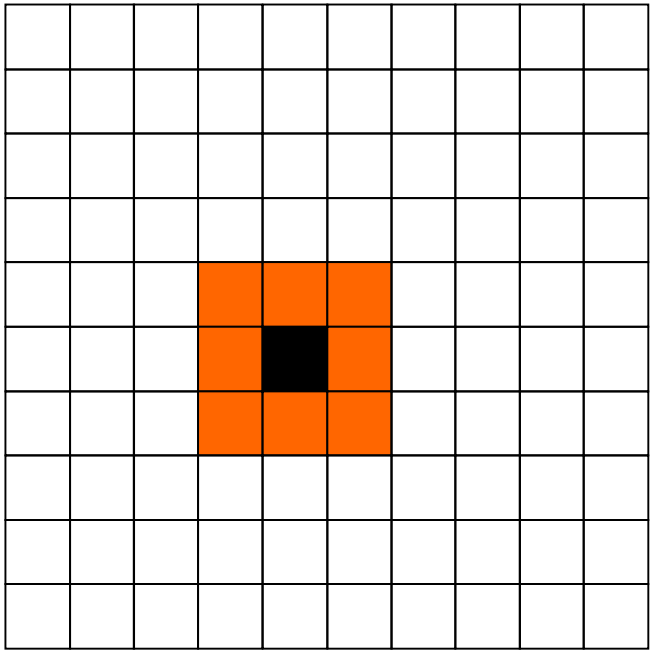


区域的类型

- 四连通邻域



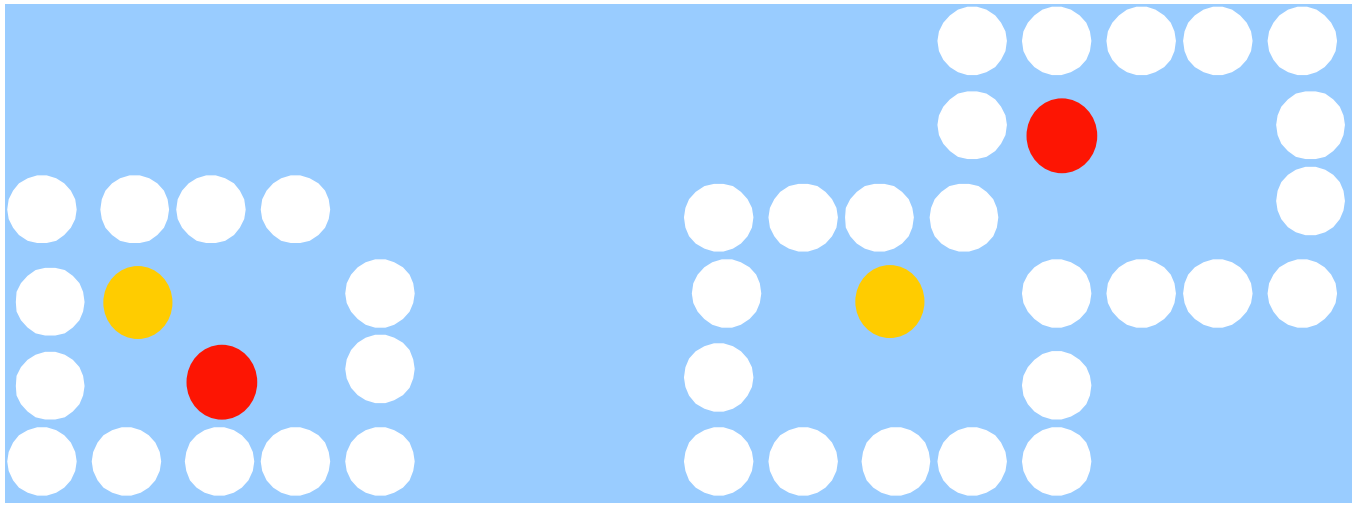
- 八连通邻域



区域的类型

- 四连通区域：区域内任意两个像素，从一个像素出发，可以通过上、下、左、右四种运动，到达另一个像素
- 八连通区域：区域内任意两个像素，从一个像素出发，可以通过水平、垂直、正对角线、反对角线八种运动，到达另一个像素

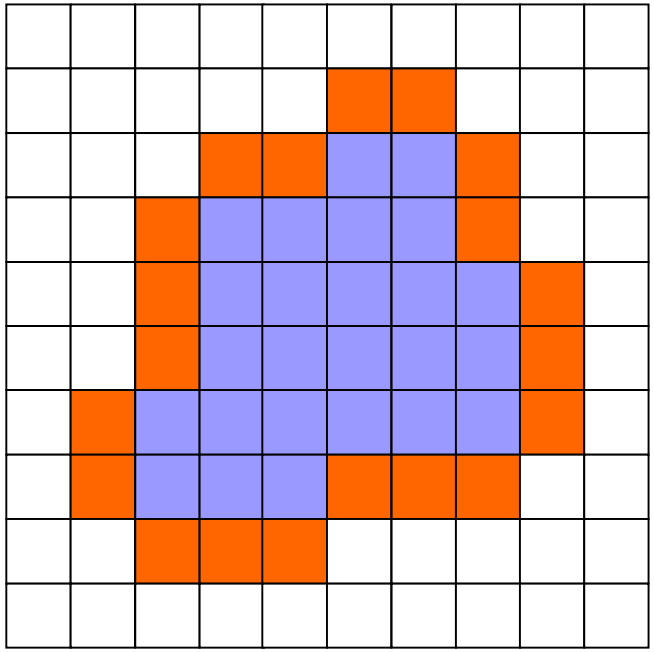
区域的类型



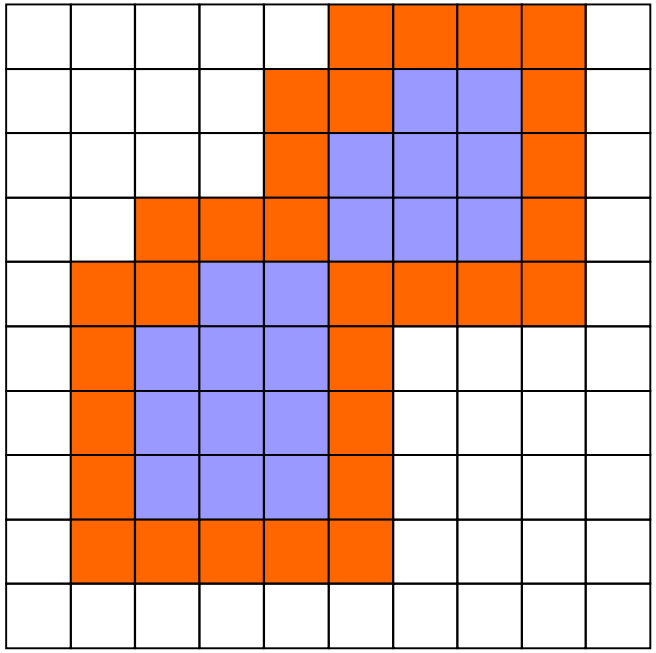
四连通区域　　八连通区域

区域的类型

- 四连通区域



- 八连通区域



区域的类型

- 四连通和八连通区域的关系
  - 四连通区域 $\subset$ 八连通区域 (反之不成立)
  - 四连通区域的边界是八连通区域
  - 八连通区域的边界是四连通区域

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较

内部表示区域种子填充算法

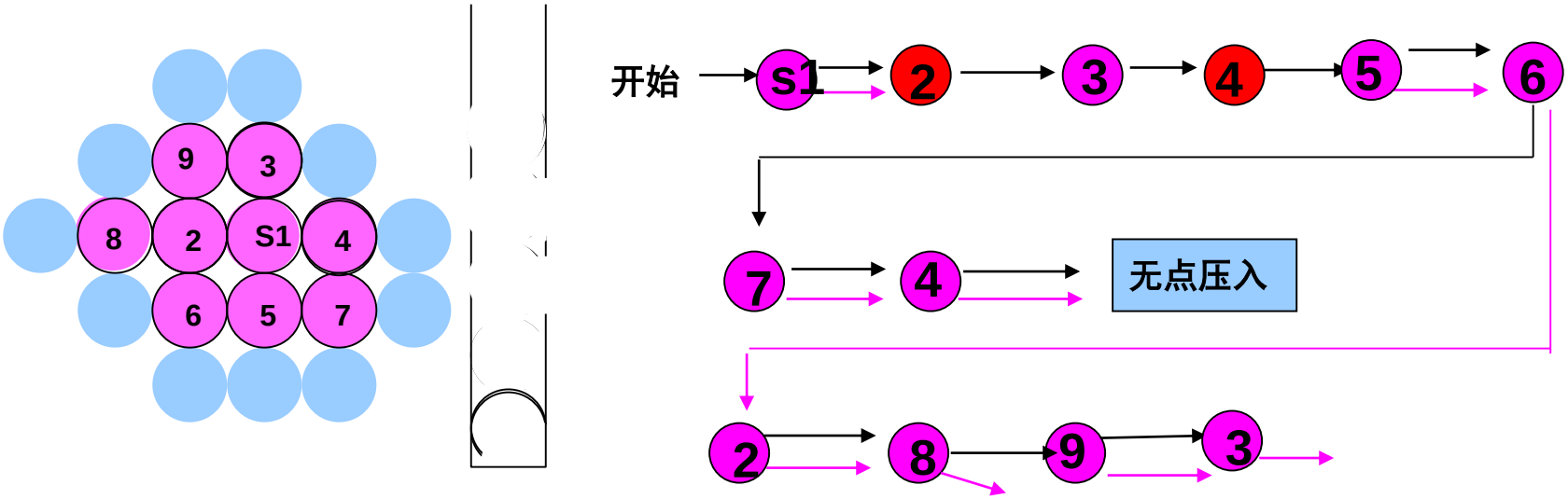
- 假设内部表示区域为G，其中的像素原有颜色为G0，需要填充的颜色为G1。
- 算法需要提供一个种子点(x,y)，它的颜色为G0。
- 具体算法如下(四连通区域)

内部表示区域种子填充算法

```
Flood_Fill_4(x, y, G0, G1)
{
    if(GetPixel(x,y) ==G0 ) // GetPixel(x,y) 返回(x,y)的颜色
    {
        SetPixel(x, y, G1);  //将(x,y)的添上颜色G1
        Flood_Fill_4(x-1, y, G0, G1);
        Flood_Fill_4(x, y+1, G0, G1);
        Flood_Fill_4(x+1, y, G0, G1);
        Flood_Fill_4(x, y-1, G0, G1);
    }
}
```

基于递归实现的种子填充算法

遍历像素的顺序：左、上、右、下



边界表示区域种子填充算法

```
Fill_Boundary_4_Connnected(x, y, BoundaryColor, InteriorColor)
// (x,y) 种子像素的坐标；
// BoundaryColor 边界像素颜色； InteriorColor 需要填充的内部像素颜色
{
    if(GetPixel(x,y) != BoundaryColor && GetPixel(x,y)!= InteriorColor )
     // GetPixel(x,y): 返回像素(x,y)颜色
    {
        SetPixel(x, y, InteriorColor); // 将像素(x, y)置成填充颜色
        Fill_Boundary_4Connnected(x, y+1, BoundaryColor, InteriorColor)；
        Fill_Boundary_4Connnected(x, y-1, BoundaryColor, InteriorColor)；
        Fill_Boundary_4Connnected(x-1, y, BoundaryColor, InteriorColor)；
        Fill_Boundary_4Connnected(x+1, y, BoundaryColor, InteriorColor)；
    }
}
```

基于递归的区域填充

- 课后练习：基于递归的八连通区域填充算法
  - 内部表述的八连通区域种子填充
  - 边界表示的八连通区域种子填充

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较

基于扫描线的区域填充

- 基于递归的区域填充算法
  - 优点：程序简单，表达清晰
  - 不足：递归算法，系统堆栈反复进出，费时、费内存
- 改进：区域填充的扫描线算法
  - 适用类型：基于边界表示的四连通区域
  - 基本思想：首先填充当前扫描线的一个区间，然后确定与这一区间相邻的上下扫描线上的区间

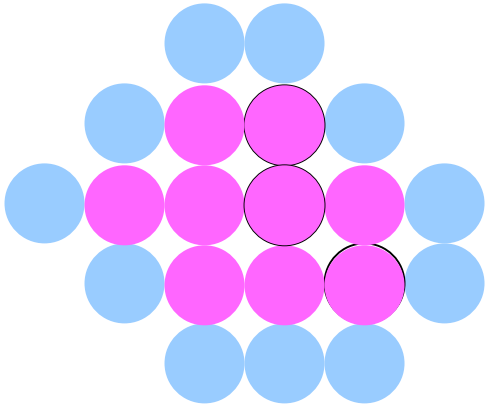
扫描线区域填充算法

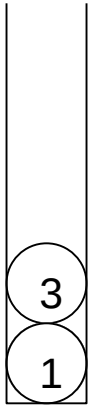
1. (初始化) 堆栈置空，将种子点(x,y)压入堆栈
2. (出栈) 如果堆栈为空，算法结束。否则取栈顶元素(x,y)作为种子点
3. (区间填充) 从种子点开始，沿y坐标左右两个方向逐个像素填充，其值置为new-color，直至边界像素

扫描线区域填充算法

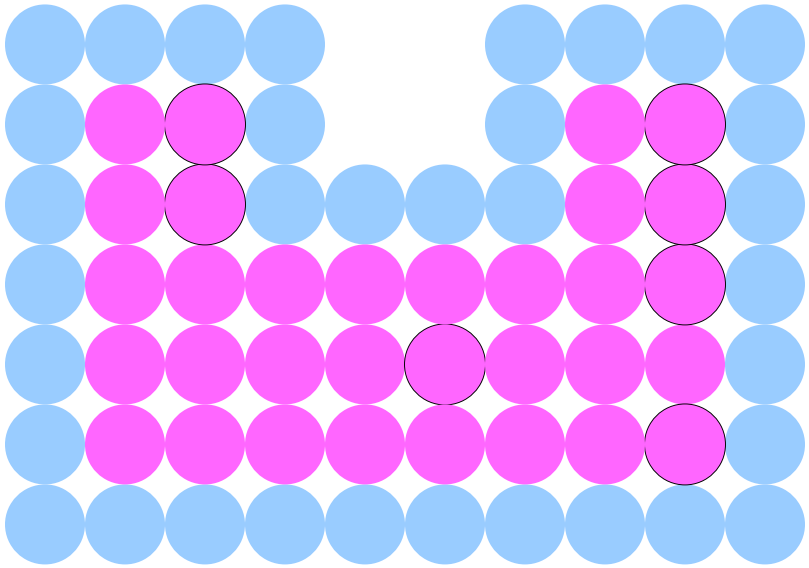
4. (定范围) 记第3步所得区间为$[x_{left},x_{right}]$
5. (进栈) 分别在当前扫描线的上下两条扫描线上，确定位于$[x_{left},x_{right}]$内的区间。如果区间内的像素颜色为new-color或boundary-color，则转步骤2；否则，将区间右端点压入堆栈，转步骤2

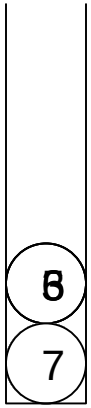
扫描线区域填充算法实例1





扫描线区域填充算法实例2





扫描线种子填充算法的优点

- 对于每一个待填充的区段，只向堆栈中压入一个种子点即可 (v.s. 全部压入堆栈)。所以种子点进出堆栈少，省内存
- 每弹出一个种子点，可以填充一个区间(v.s. 逐一填充)，所以填充效率高

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较

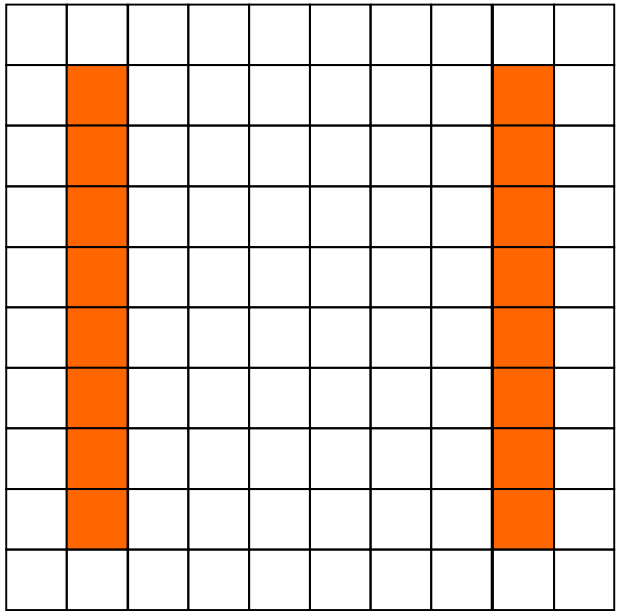
多边形扫描转换与区域填充比较

- 基本思想不同
  - 多边形扫描转换将多边形顶点表示转换为点阵表示，扫描过程利用了多边形的各种连贯性
  - 区域填充只改变区域的颜色，不改变区域的表示方法。填充过程利用了区域的连贯性

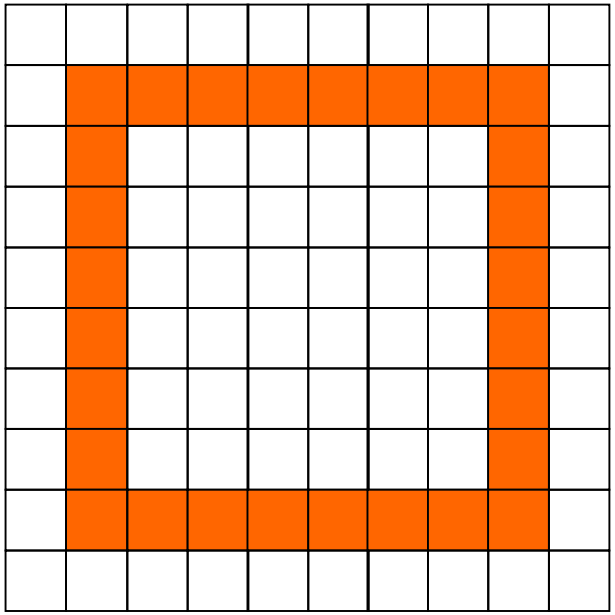
多边形扫描转换与区域填充比较

- 对边界的要求不同
  - 多边形扫描转换只要求每一条扫描线与多边形有偶数个交点
  - 区域填充中
    - 四连通区域必须是封闭的八连通边界
    - 八连通区域必须是封闭的四连通边界

多边形扫描转换与区域填充比较



多边形扫描转换允许边界



区域填充允许边界

多边形扫描转换与区域填充比较

- 出发点不同
  - 区域填充：知道需要区域内一个种子点(复杂计算)
  - 多边形扫描转换：没有要求

课后阅读：《计算机图形学教程》(修订版)唐荣锡等，科学出版社，2000年。p.112-129（Ch7.1~7.3)

主要内容

- 多边形扫描转换
  - 边填充算法
  - 边标志算法
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
  - 基于扫描线的四连通区域种子填充算法
- 多边形的扫描转换与区域填充的比较