第3-3章区域填充

课程目标

- 多边形区域填充算法
- 边填充算法
- 栅栏填充算法
- 边界标志算法
- 种子填充算法

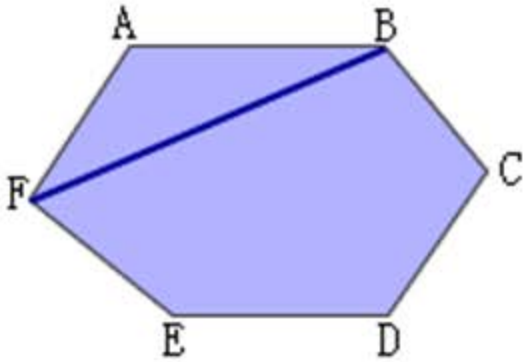
多边形的种类

多边形内任意选取不相同的两点，

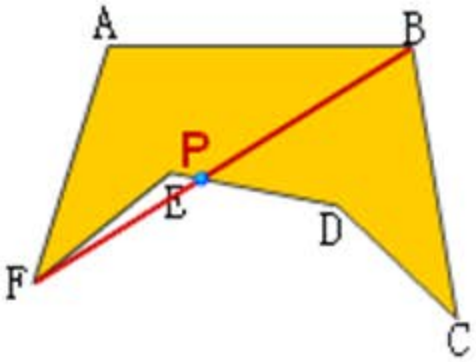
其连线上的所有点均在该多边形内，称为凸多边形；

否则，称为凹多边形 。

凸多边形



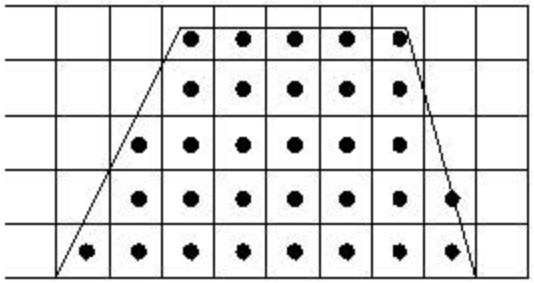
凹多边形



多边形的表示方法

多边形的表示方法：顶点表示和点阵表示。

多边形顶点表示



多边形点阵表示

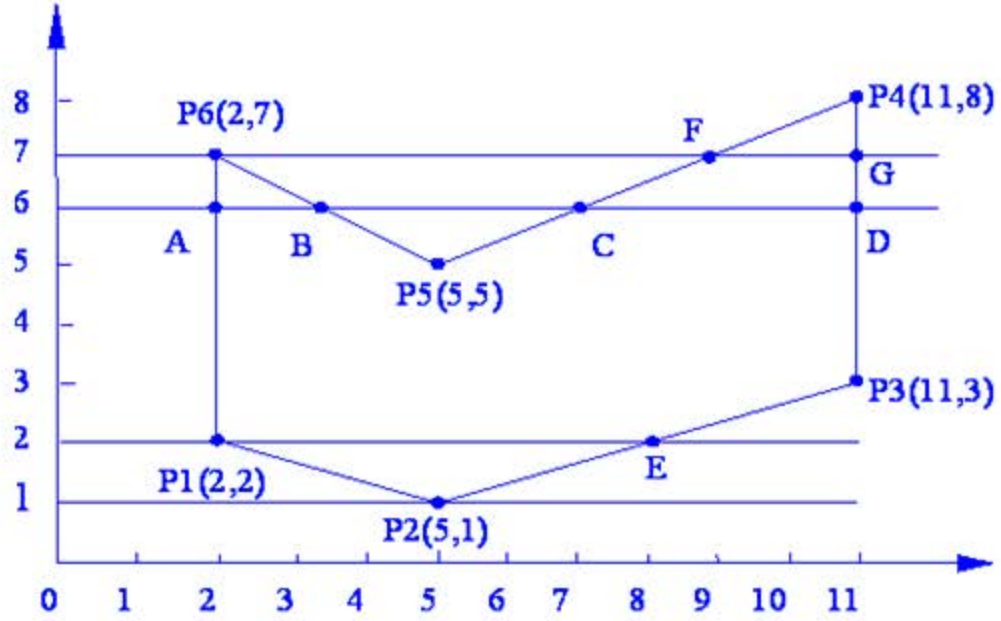
多边形区域填充:把多边形的顶点表示转换为点阵表示。

区域填充方法1--有序边表算法

基本思想：

按扫描线顺序，计算扫描线与多边形的相交区间，

用要求的颜色显示这些区间的象素，即完成填充工作。



区域填充方法1--有序边表算法

步骤：

（1）求交

I_4, I_3, I_2, I_1

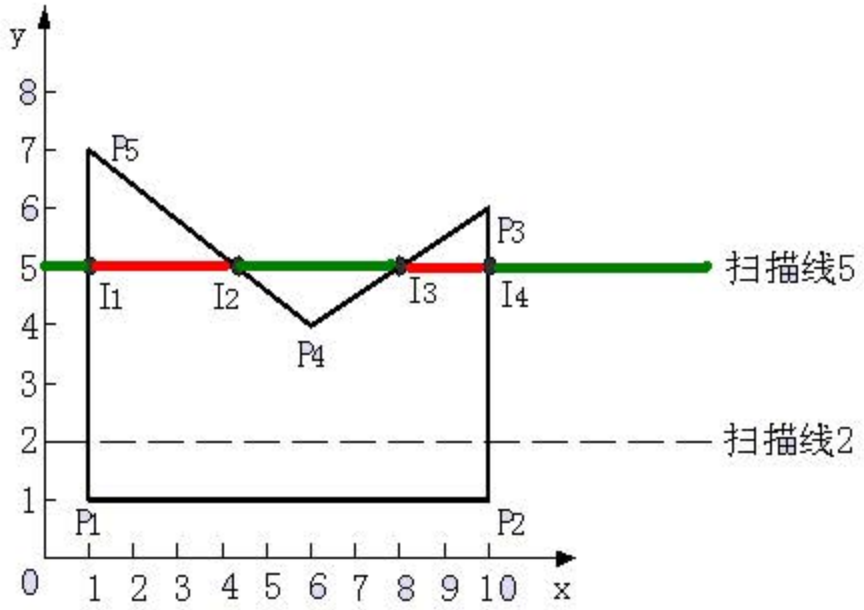
（2）排序

I_1, I_2, I_3, I_4

（3）交点配对

$(I_1, I_2), (I_3, I_4)$

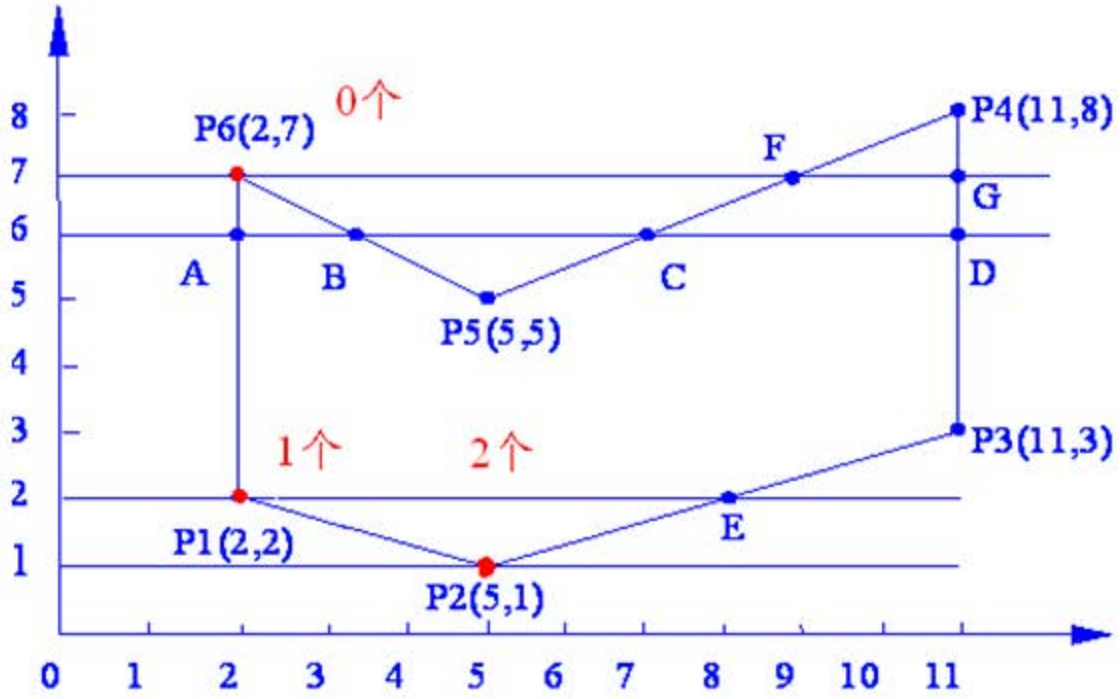
（4）区间填色



区域填充方法1--有序边表算法

特殊问题一：多边形顶点的计数

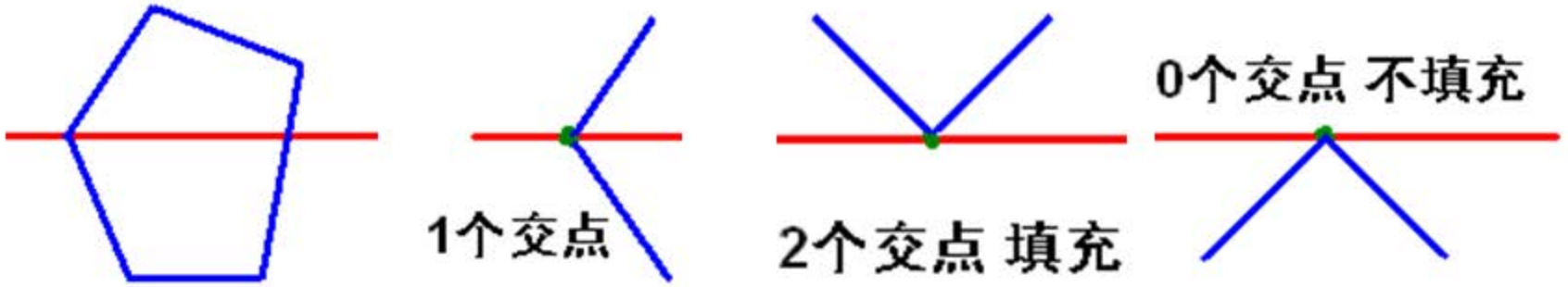
扫描线过顶点时，如何确定交点个数？



区域填充方法1--有序边表算法

解决：

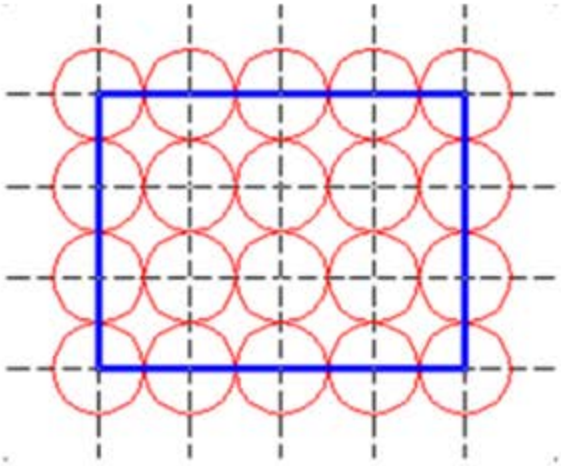
检查交于该顶点的两条边的另外两个端点的y值大于该顶点y值的个数

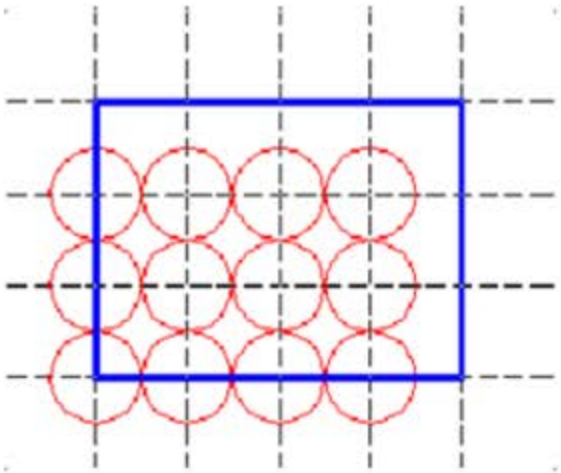


区域填充方法1--有序边表算法

特殊问题二：多边形边界上像素点的舍取问题?

多边形在填充前后因边界上的像素的取舍而导致形状和面积的改变





区域填充方法1--有序边表算法

解决：

顶点计数方法保证多边形填充时边界“下闭上开”

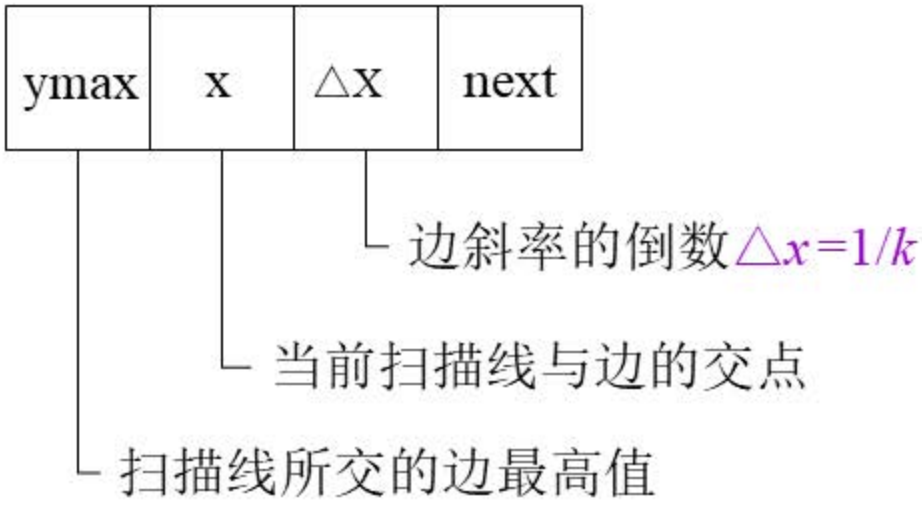
同理规定多边形边界“左闭右开”

区域填充方法1--有序边表算法

活性边：与当前扫描线相交的边。

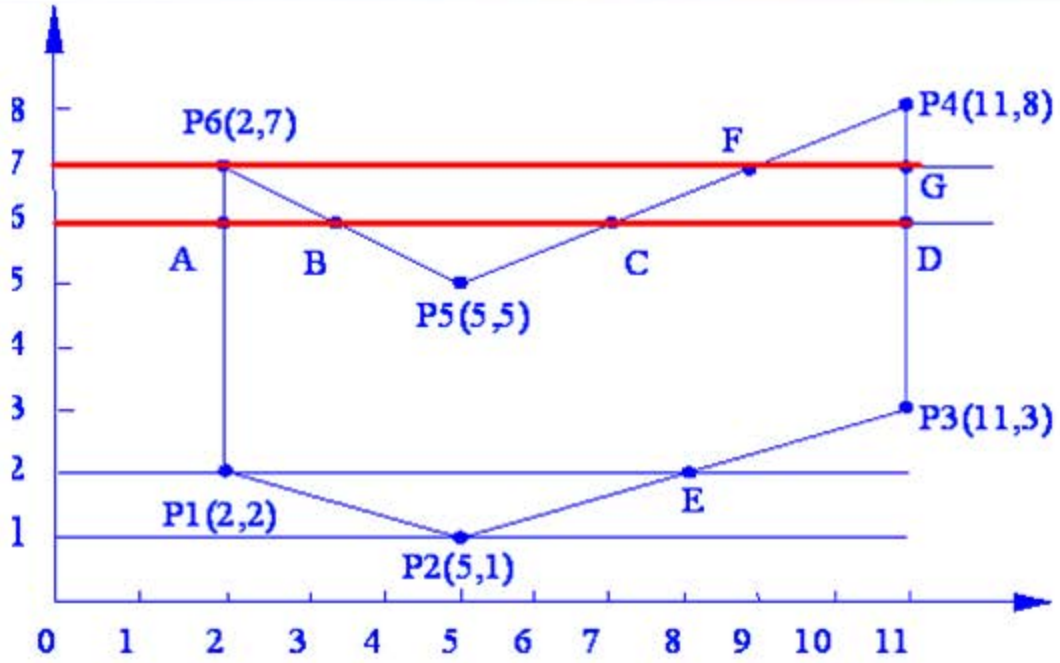
活性边表：把活性边按与扫描线交点x坐标递增的顺序存放在一个链表中。

结点存储结构定义：




```
typedef   struct Edge
{
    float  ymax;
    float   x;
    float   dx;
    struct Edge *next;
}Edge;
```

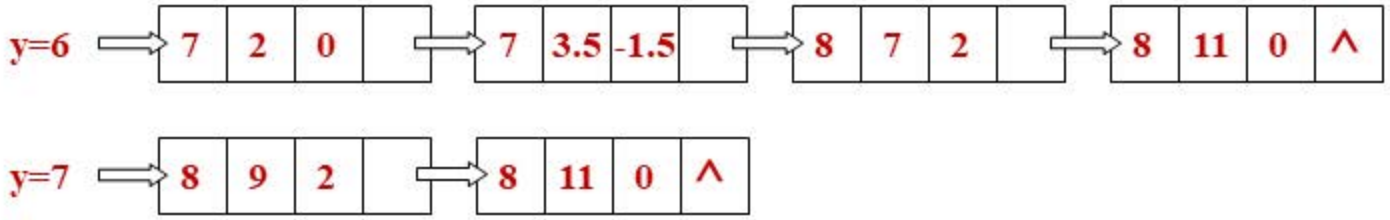
区域填充方法1--有序边表算法



活性边表的增量更新：

新边插入

旧边删除

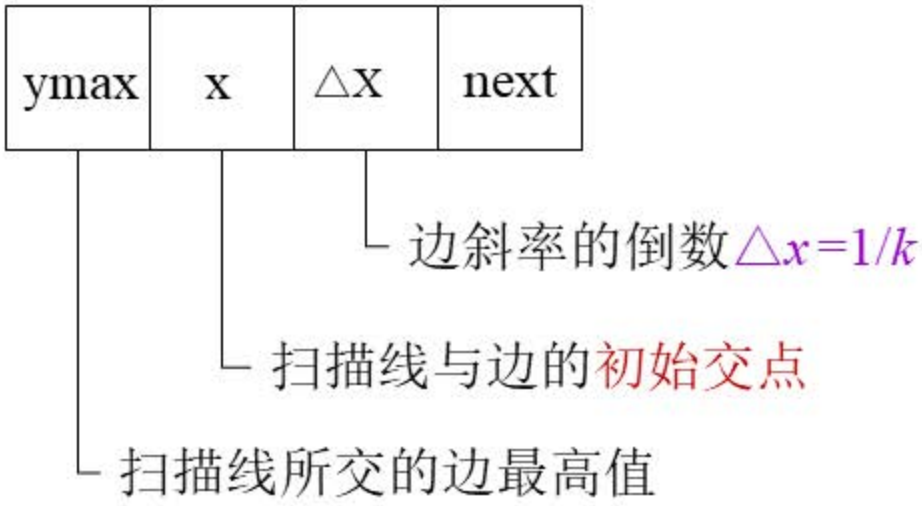


区域填充方法1--有序边表算法

新边表：对每条扫描线建立一个新边表。

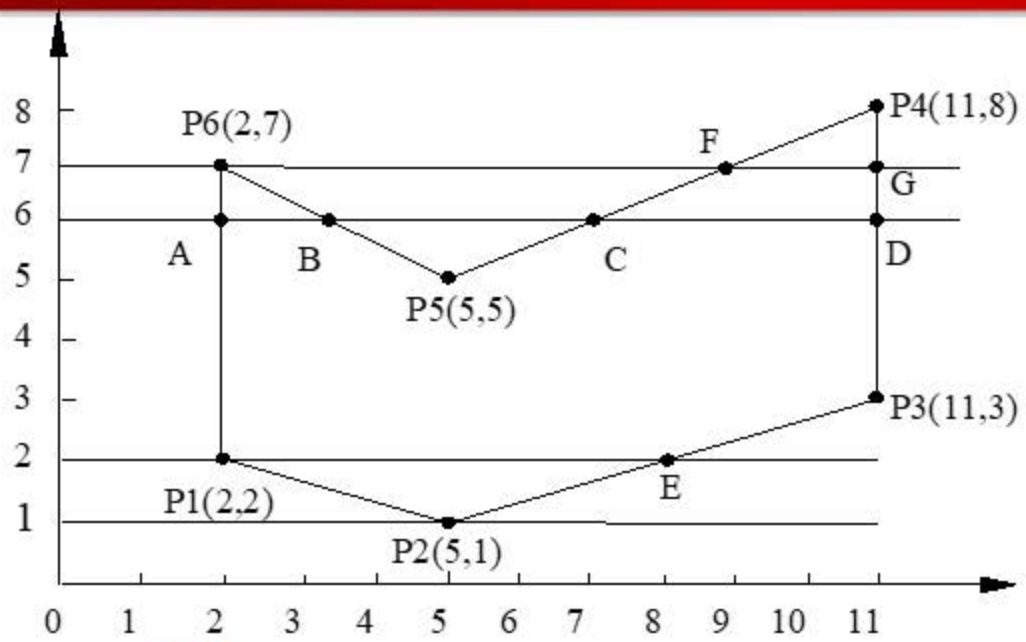
目的：助活性边表的增量更新。

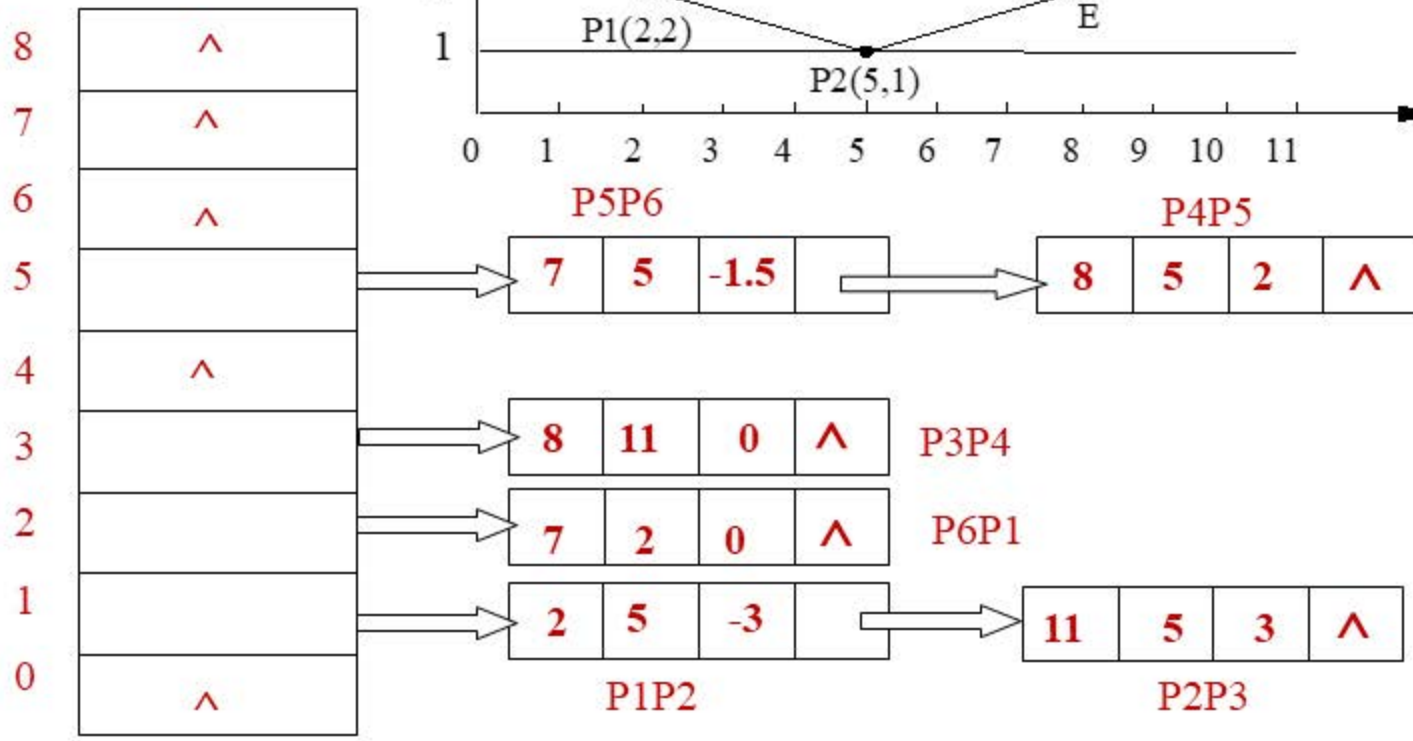
新边表中结点存储结构定义：




```
typedef   struct Edge
{
    float  ymax;
    float   x;
    float   dx;
    struct Edge *next;
}Edge;
```

区域填充方法1--有序边表算法





区域填充方法1--有序边表算法

```
void polyfill (多边形 polygon, int color) {
    for (各条扫描线i ) {
        初始化新边表头指针NET[i];
        把ymin = i 的边放进边表NET[i];
    }//初始化新边表
    y = 最低扫描线号;
    始化活性边表AET为空;
    for (各条扫描线i ){
      把新边表NET[i]中的边结点用插入排序法插入AET表，使之按x递增排列;
      遍历AET表，把配对交点区间(左闭右开)上的象素，用putpixel改写象素颜色值;
      遍历AET表，把ymax= i 的结点从AET表中删除，并把ymax > i 结点的x值递增Δx;
      若允许多边形的边自相交，则用冒泡排序法对AET表重新排序;
    }//活性边表增量更新
} /* polyfill */
```

ScanLineFill.cpp

区域填充方法1--有序边表算法

优点：

对每个像素只访问一次

与设备无关

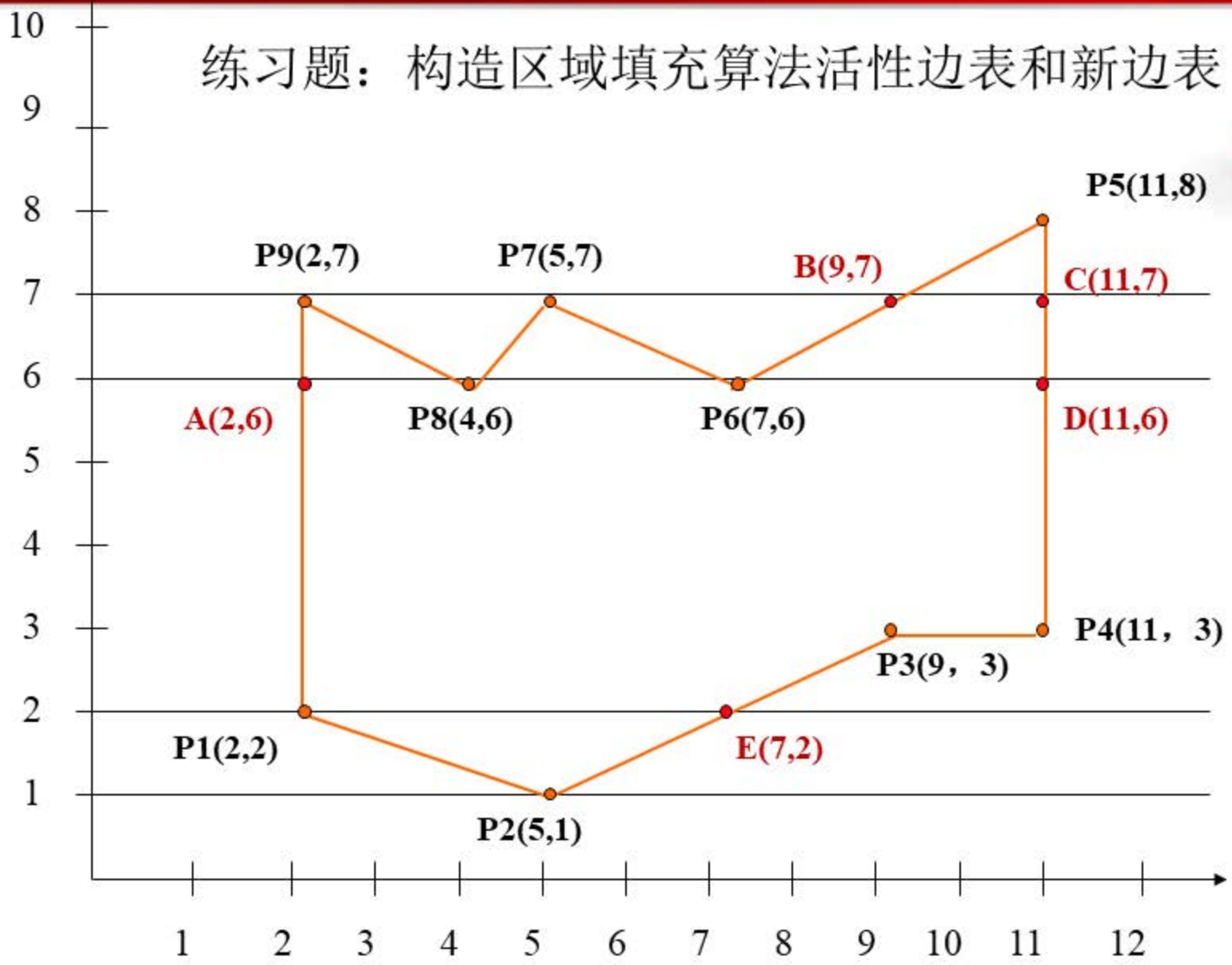
缺点：

数据结构复杂

只适合软件实现

区域填充方法1--有序边表算法

练习题：构造区域填充算法活性边表和新边表



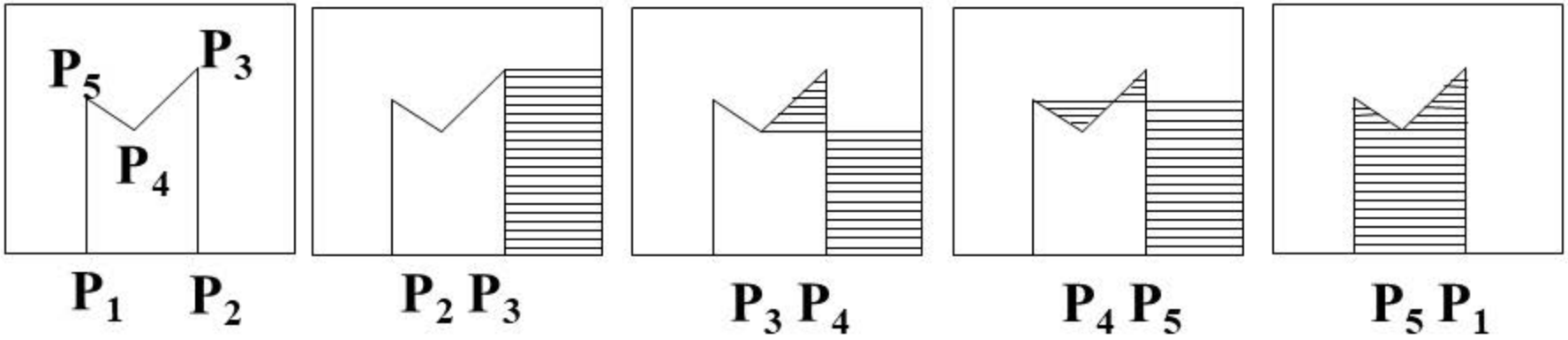
区域填充方法2--边填充算法

基本思想：

对于每一条扫描线和每条多边形的交点(x1, y1)，

将该扫描线上交点右方的所有像素取补。

对多边形的每条边作此处理，多边形的顺序随意。



区域填充方法2--边填充算法

优点：

最适合于有帧缓存的显示器

可按任意顺序处理多边形的边

仅访问与边有交点的扫描线右方像素，算法简单

缺点：

对复杂图形，每一像素可能被访问多次，输入/输出量大

图形输出不能与扫描同步进行，只有全部画完才能打印

区域填充方法3--栅栏填充算法

引入栅栏的目的？

为了减少边填充算法访问像素的次数。

何谓栅栏？

一条与扫描线垂直的直线；

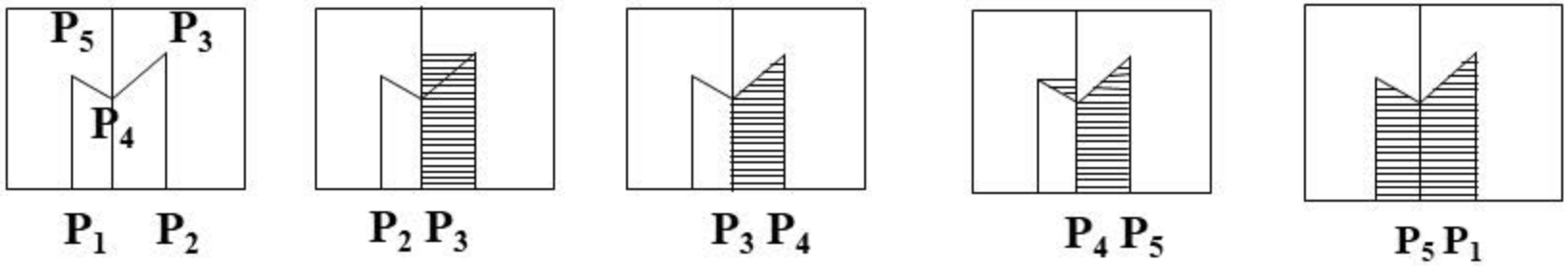
栅栏位置通常取过多边形顶点、且把多边形分为左右两半。

区域填充方法3--栅栏填充算法

基本思想：

若交点位于栅栏左侧，则将交点之右至栅栏之左的像素取补；

若交点位于栅栏右边，则将栅栏之右至交点之左的像素取补。



区域填充方法3--栅栏填充算法

栅栏填充算法

只是减少重复访问的像素的数目。

改进的边界标志算法

使得算法对每个像素仅访问一次。

区域填充方法4--边界标志算法

基本思想：

1. 对每条与多边形相交的扫描线，依从左到右顺序，逐个访问该扫描线上像素。

2. 使用布尔量inside指示当前点的状态：若点在多边形内，inside为真；否则，inside为假。

（实现方法： inside初始值为假，当像素点是边标志的点，将inside取反）

3. 若inside为真，则把该像素置为多边形色。

区域填充方法4--边界标志算法

```
void edgemark_fill(多边形定义  polydef, int color)
{     对多边形polydef 每条边进行直线扫描转换;
      inside = FALSE;
      for (每条与多边形polydef相交的扫描线y )
        for (扫描线上每个象素x )
        {    if(象素 x 被打上边标志) inside = ! (inside);
             if(inside! = FALSE) drawpixel (x, y, color);
             else  drawpixel (x, y, background);
        }
}
```

区域填充方法4--边界标志算法

优点：

用软件实现时，扫描线算法与边界标志算法的速度几乎相同

边界标志算法更适合硬件实现，这时它的执行速度比有序边表算法快一至两个数量级。

区域填充方法5--种子填充算法

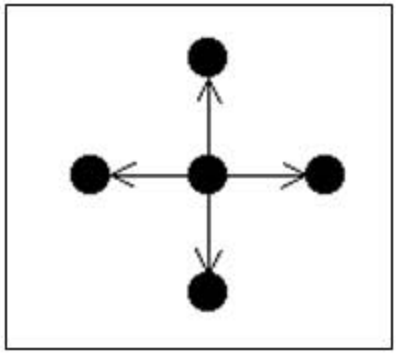
基本思想：

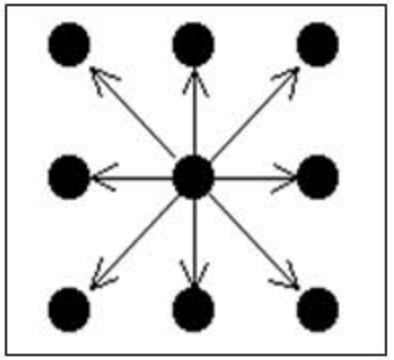
假设在多边形区域内部至少有一个像素（种子像素）是已知的，由此出发找到区域内所有其他像素，并对其进行填充。

区域连通方式：

四向连通

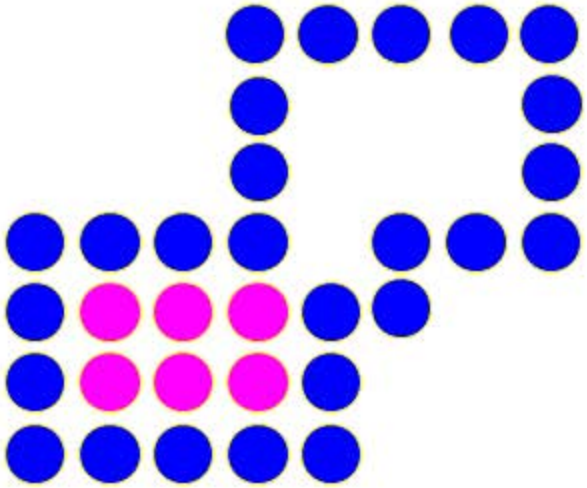
八向连通



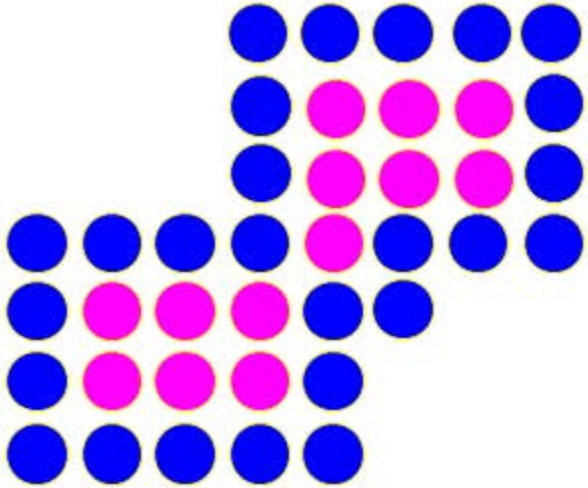


区域填充方法5--种子填充算法

区域连通方式对填充结果的影响



4连通填充结果



8连通填充结果

区域填充方法5--种子填充算法

4连通种子填充算法

(1)种子像素入栈

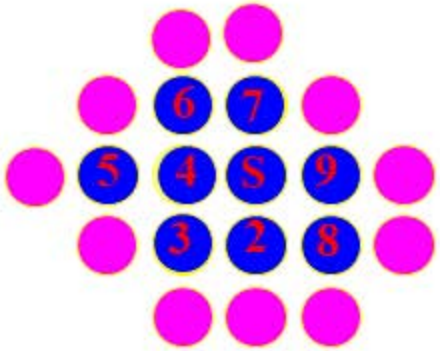
(2)当栈非空时，重复以下步骤：

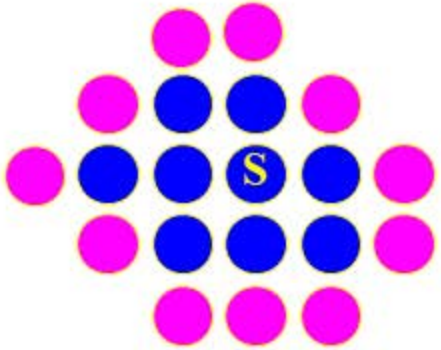
(2.1)栈顶像素出栈

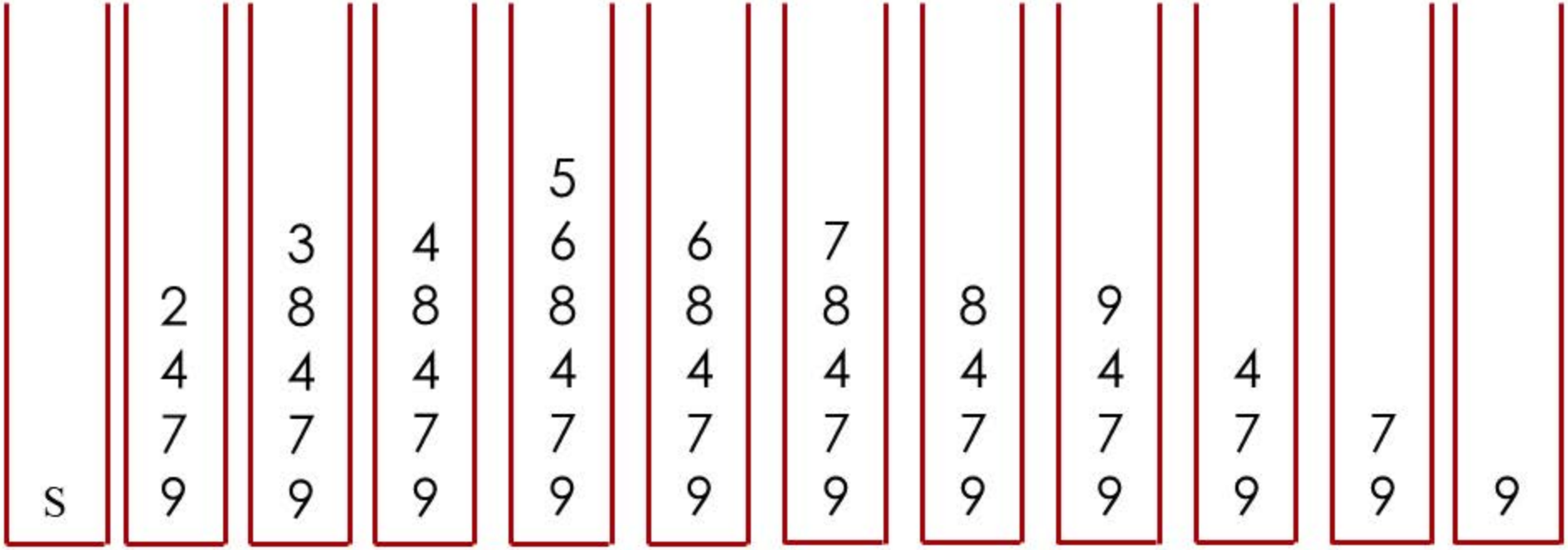
(2.2)将出栈象素置成填充色

(2.3)按左、上、右、下顺序检查与出栈象素相邻的四象素，若其中某象素不在边界上且未被置成填充色，则将其入栈

区域填充方法5--种子填充算法

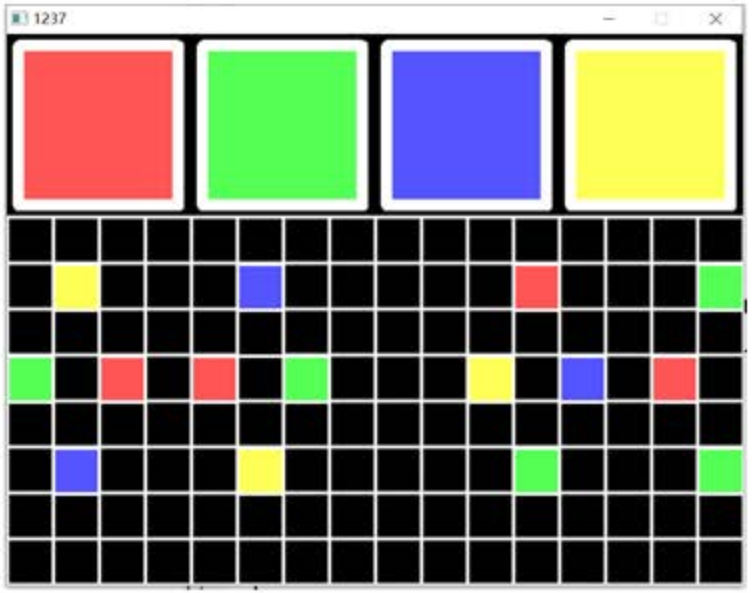






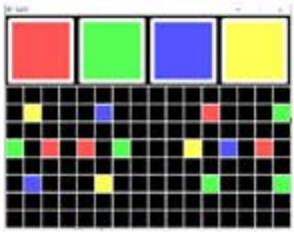
格子涂色

写一个“格子涂色”的游戏，要求：屏幕上有16x8的格子，屏幕底部有类似画笔中的选色区（随便放上一些常用的颜色），鼠标点击选择区的颜色后，就作为当前颜色，然后再点屏幕上的格子，就可以用刚才的颜色填涂相应格子。



格子涂色

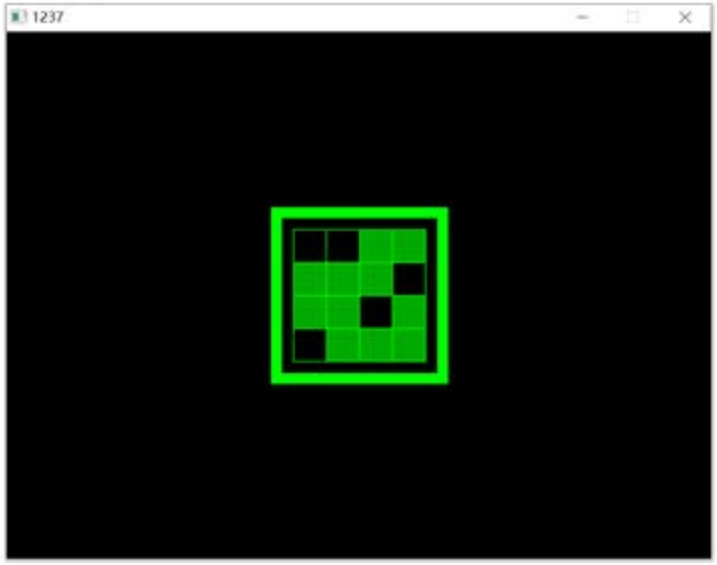
```
#include <graphics.h>
#include <conio.h>

int main()
{
  // 初始化图形窗口
  initgraph(640, 480);

  int i=10, b=150;
  setlinestyle(PS_SOLID, 10);
  setlinecolor(WHITE);

  setfillcolor(LIGHTRED);
  fillrectangle(10,10,149,149);
  setfillcolor(LIGHTGREEN);
  fillrectangle(10+i+b, 10, 149 + i + b, 149);
  setfillcolor(LIGHTBLUE);
  fillrectangle(10 + (i + b)*2, 10, 149+(i + b) * 2, 149);
  setfillcolor(YELLOW);
  fillrectangle(10 + (i + b) * 3, 10, 149+(i + b) * 3, 149);
  setlinecolor(WHITE);

  ExMessage m;     // 定义鼠标消息
  setlinestyle(PS_SOLID, 3);
  for (int m1 = 0; m1 < 641; m1 += 40) {
    line(m1, 159, m1, 479);
  }
    for (int n = 159; n < 480; n+=40) {
    line(0, n, 639, n);
  }


  setfillcolor(BLACK);
  while (true)
  {
    // 获取一条鼠标消息
    m = getmessage();
    int tempx,tempy;
    if (m.message == WM_LBUTTONDOWN)
    {
      if (m.y < 158)
      {
      for (int i = 160; i <= 640; i += 160)
        {
        if (m.x < i)
        {
          tempx = i;
          break;
        }

        }
      switch (tempx / 160)
      {
      case 1:
        setfillcolor(LIGHTRED);
        break;
      case 2:
        setfillcolor(LIGHTGREEN);
        break;
      case 3:
        setfillcolor(LIGHTBLUE);
        break;
      case 4:
        setfillcolor(YELLOW);
        break;
      }
      }
      else
      {
        for (int i = 0; i <= 640; i += 40)
        {
          if (m.x < i)
          {
            tempx = i;
            break;
          }
        }
        for (i = 160; i <= 480; i += 40)
        {
          if (m.y < i) {
            tempy = i;
            break;
          }
        }
        fillrectangle(tempx-40, tempy-40, tempx, tempy);
      }

    }
      else if(m.message == WM_RBUTTONUP)// 按鼠标右键退出程序
      return 0;
  }

  // 关闭图形窗口
  _getch();
  closegraph();
  return 0;
}
```



gezituse.cpp

点灯游戏

- 该游戏是经典的涂格子游戏，很有挑战性。



fillblank.cpp

课后习题

1. 已知多边形各顶点坐标为

P1(2,2),P2(2,4),P3(8,6),P4(12,2),P5(8,1),P6(6,2)，在用多边形区域填充时，请写出ET和全部AET内容

2. 画出4连通种子填充算法中堆栈中存放像素情况。

3. 列举区域填充算法，并比较其优缺点。