Report

图形学实验 PA0
光栅图形学

实现直线绘制、圆形绘制以及区域填充算法

姓名: 勾天润

学号: 2020012321

班级: 无 03

课程: 计算机图形学基础

指导教师：胡事民

助教：曹耕晨、彭浩洋

March 19, 2023

Contents

1. 代码逻辑

1.1 直线绘制

直线绘制部分使用 Bresenham 算法.
当 $|k| < 1$ 时，需要从 x 较小的点开始画，每次 e 更新 k.

Algorithm 1 Bresenham Algorithm

Require: $(x_1, y_1), (x_2, y_2)$
Ensure: $\{(x, y)\}$ to draw
  if $x_1 = x_2 || y_1 = y_2$ **then** 绘制竖线或横线 **return**
  end if
  $k \leftarrow dy/dx$
  if $-1 < k < 1$ **then**
    $x \leftarrow min(x1, x2), x\_ \leftarrow max(x1, x2), y \leftarrow y_x,$
    if $k > 0$ **then**
      $e \leftarrow -0.5$
      while $x \leq x\_$ **do**
        Draw (x, y)
        $x \leftarrow x + 1$
        $e \leftarrow e + k$
        if $e > 0$ **then**
          $e \leftarrow e - 1$
          $y \leftarrow y + 1$
        end if
      end while
    else if $k < 0$ **then**
      $e \leftarrow 0.5$
      while $x \leq x\_$ **do**
        Draw (x, y)
        $x \leftarrow x + 1$
        $e \leftarrow e + k$
        if $e < 0$ **then**
          $e \leftarrow e + 1$
          $y \leftarrow y - 1$
        end if
      end while
    end if
  else if $k \geq 1 || k \leq -1$ **then** ……
  end if

当 $|k| >= 1$ 时，需要从 y 较小的点开始画，每次 e 更新 1/k.
容易出问题的地方是，如果只按上述算法做，可能画的最后一个点并不是直线实际的最后一个点。比如在测试样例 1 中，从 (255,0) 到 (0,300) 的直线本应接触边界，使得后续的填充时不会让颜色漏出直线左上方的区域。如果按照上述做法，直线终点将为 (1,300) 而非 (0,300), 导致后续填充出问题。

1.2 圆形绘制

圆形绘制采用**中点画圆法**，核心思想与画直线部分类似，都是考察刚已画过的点，其右 (上) 侧像素是否该画，否则画该像素上或下 (左或右) 侧像素。得到某个应该画的像素后，可利用对称性获得其余 7 个应该画的像素。
在计算判别式时，最好将圆心平移回原点，判别后再加上圆心坐标，进行绘制。
加入了裁剪逻辑。需要判断当前处理像素是否在图片中，如果不是则不能更改颜色、访问颜色，否则为非法处理。(填充算法同理)

1.3 区域填充

区域填充部分参考了教材 2.3.2 节。使用了栈的数据结构，每次涂完连续的一行区域后，找到其上下的四连通区域，入栈。这样可以保证，每块区域只入栈一次。
注意需要判断当前要涂的颜色和原来颜色是否一致，若相同则可直接 return，防止代码一直进行出栈入栈陷入死循环。

```
Vector3f oldcolor=img.GetPixel(x,y);
if(oldcolor==color)return;//如果本来就是要涂的颜色，直接return
int xl,xr,i;//刚画完的一条线段，其左右端点横坐标
bool spanNeedFill;//当前区域是否需要涂色的flag
Seed pt;pt.x=x;pt.y=y;
stack<Seed>s;
s.push(pt);
while(!s.empty()){
    pt=s.top();//取出栈顶元素，往左往右分别画到颜色不是oldcolor为止
    s.pop();
    x=pt.x;
    y=pt.y;
    while(inside(img,x)){//往右画
        if(img.GetPixel(x,y)==oldcolor){
            img.SetPixel(x,y,color);
            x++;
        }
        else break;
    }
    xr=x-1;x=pt.x-1;
    while(inside(img,x)){//往左画
        if(img.GetPixel(x,y)==oldcolor){
            img.SetPixel(x,y,color);
            x--;
        }
        else break;
    }
    xl=x+1;
    //处理上面一条扫描线
    x=xl;
```

```
31      y++;if(y<img.Height()){
32      while(x<=xr){
33          spanNeedFill=false;
34          while(inside(img,x)){
35              if(img.GetPixel(x,y)==oldcolor){
36                  spanNeedFill=true;
37                  x++;
38              }
39              else break;
40          }
41          if(spanNeedFill){
42              pt.x=x-1;pt.y=y;
43              s.push(pt);
44              spanNeedFill=false;
45          }
46          while(inside(img,x)){
47              if(img.GetPixel(x,y)!=oldcolor&&x<=xr)x++;
48              else break;
49          }
50      }
51      }
52      //处理下面一条扫描线
53      x=xl;
54      y=y-2;if(y>=0){
55      while(x<=xr){
56          spanNeedFill=false;
57          while(inside(img,x)){
58              if(img.GetPixel(x,y)==oldcolor){
59                  spanNeedFill=true;
60                  x++;
61              }
62              else break;
63          }
64          if(spanNeedFill){
65              pt.x=x-1;pt.y=y;
66              s.push(pt);
67              spanNeedFill=false;
68          }
69          while(inside(img,x)){
70              if(img.GetPixel(x,y)!=oldcolor&&x<=xr)x++;
71              else break;
72          }
73      }
74      }
75  }
```

2. 代码参考

本次作业完全参考课本上的算法讲解与代码，在其基础上进行改进。

3. 代码测试过程

3.1

画直线过程中，出现只能画竖线横线的情况。经分析，c++ 中 int/int 只求商。可将 dy 或 dx 转为 double 型

3.2

绘制斜率为负的直线时，曾尝试将图片水平翻转、绘制正直线再翻转回来，但出现了奇怪的报错，就暂时搁置了这一想法。

3.3

填充过程中，发现灰色区域“越界”，放大观察图片，偶然发现是直线没能把边界封住，于是增加了 setpixel(终点) 的语句。

3.4

在自己做样例进行测试过程中，发现如果两次执行相同的填充命令会陷入死循环，于是增加了“oldcolor==color”的判断。

经过调试，代码可以完整绘制提供的测试样例。我自己编写了一个测试样例，画了一只 mole

4. 建议

Image 类可以添加一个用于判断像素是否在图像上的函数，

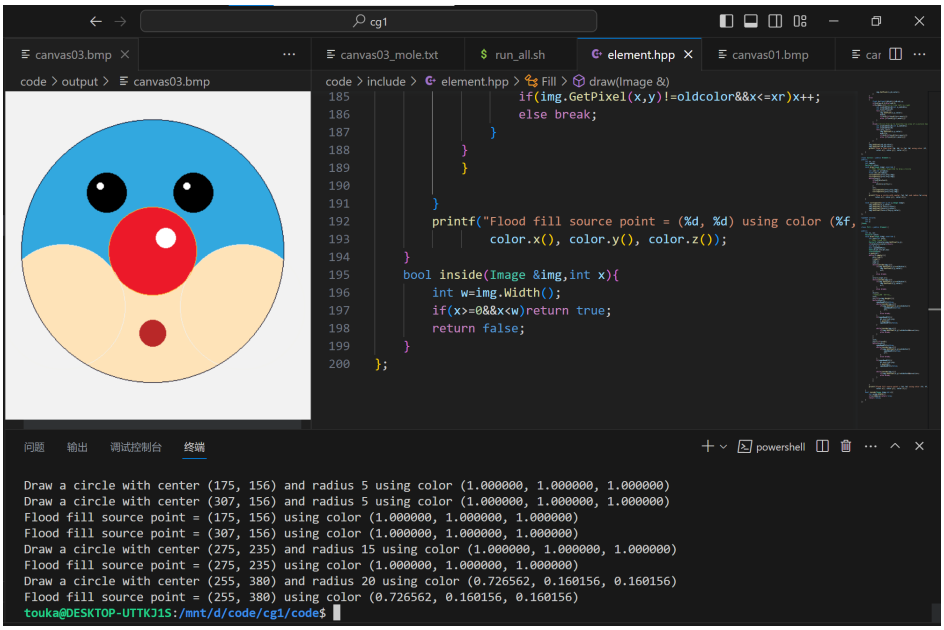

Figure 1: mole