第二次实验报告

姓名:徐泽庭

学号:10185102108

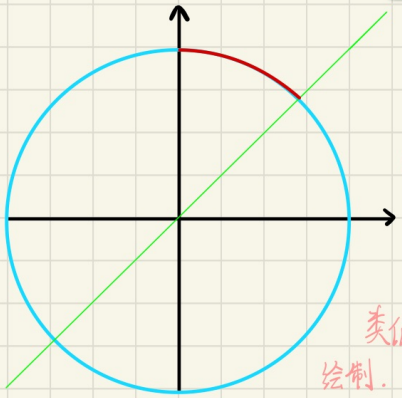
日期:2021/3/21

问题描述:

使用中点画圆法绘制圆和椭圆

求解思路:

中点画圆法

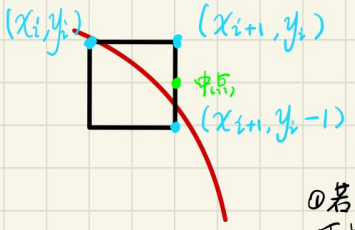


对于整个圆，我们首先将圆弧分为4个部分，
画出第一象限后经过旋转或对称变换得到其下的图形

于是对第一象限的弧进行研究，发现画出其中半即可绘出另一半

类似于Bresenham算法，在红色圆弧区域，以x为主方向进行绘制。

判别依据：



圆的方程：$x^2+y^2=r^2$

即 $x^2+y^2-r^2=0$

中点坐标：$(x_{i+1},y_i-\frac{1}{2})$

将中点坐标代入 $G=x^2+y^2-r^2$

①若 $G>0$，说明中点坐标在圆外。
下点离圆弧近，取 (x_{i+1},y_i-1) 作为下一绘制点

②若 $G=0$，说明中点坐标在圆上。
取 (x_{i+1},y_i-1) 或 (x_{i+1},y_i) 均可

③若 $G<0$，说明中点坐标在圆内
取 (x_{i+1},y_i)

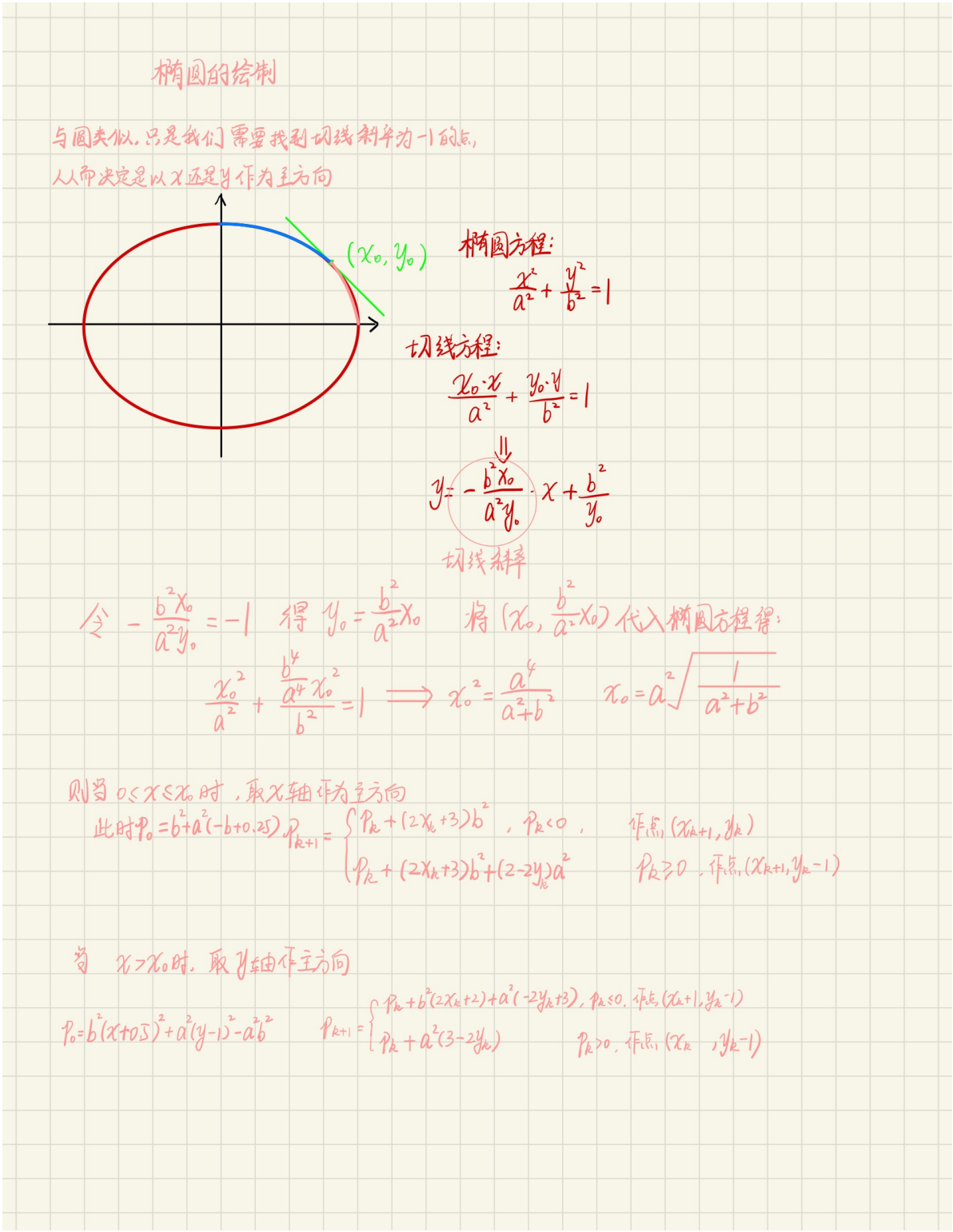
迭代思想求决策参数 p_k

$$p_k=G(x_k+1,y_k-\tfrac{1}{2})=(x_k+1)^2+(y_k-\tfrac{1}{2})^2-R^2$$

若 $p_k<0$，则 $p_{k+1}=G(x_k+2,y_k-\tfrac{1}{2})=(x_k+2)^2+(y_k-\tfrac{1}{2})^2-R^2=p_k+2x_k+3$

若 $p_k\geq 0$，则 $p_{k+1}=G(x_k+2,y_k-\tfrac{3}{2})^2=(x_k+2)^2+(y_k-\tfrac{3}{2})^2-R^2=p_k+2(x_k-y_k)+5$

由此得到了 p_{k+1} 与 p_k 之间的关系，类似于 Bresenham 算法，只要判断 p_k 与0的关系即可画出下一点。



程序代码：

绘制圆：

```
import wx
import time
class CIRCLE(wx.Frame):
    def __init__(self):
        #定义窗口的大小和位置
        super().__init__(None, title='中点画圆法', size=(1000, 1000), pos=(30,
30))

```

```
        #输入参数
        self.x, self.y, self.r = map(int, input("请依次输入圆心坐标和半径:
\n").split())
        #将事件与函数进行绑定
        self.Bind(wx.EVT_PAINT, self.On_Paint)

        #颜色表
        self.color = [
            '#FFB6C1',
            '#FF00FF',
            '#0000CD',
            '#98FB98',
            '#FFFF00',
            '#FF4500',
            '#B22222',
            '#00BFFF',
        ]

    #绘图函数
    def DrawCircle(self, a, b, R, dc):
        L4=[] #存储第四象限的数组(对于窗口的坐标系而言是第一象限)
        x, y, p = 0, R, 1 - R
        #将p设置为1-R能够避免浮点运算，而且其绘图的拟合程度较好
        while (x < y):
            if p < 0:
                p += (2 * x + 3)
            else:
                p += (2 * x - 2 * y + 5)
                y -= 1
            x+=1
            L4.append((x,y)) #八分之一点的集合
            L4.append((y,x)) #将点做关于y=-x(对于窗口坐标系而言是y=x)的对称
            #L4所得点的集合是四分之一圆弧的点

        L3 = [(-1*e[0],e[1])for e in L4] #第三象限
        L2 = [(-1 * e[0], -1*e[1]) for e in L4] #第二象限
        L1 = [(e[0], -1*e[1]) for e in L4] #第一象限

        #为绘图时按照顺时针绘画，将各个象限进行排序
        L1.sort()
        L2.sort()
        L3.sort(reverse=True)
        L4.sort(reverse=True)

        #按照一四三二的象限顺序生成总的点集
        p_points = [(e[0]+a,e[1]+b)for e in L1+L4+L3+L2]
        points = list(set(p_points))
        points.sort(key=p_points.index)

        #这里利用time.sleep显示出动画效果
        #为了颜值设置了每隔STEP个点换一支画笔颜色
        STEP = 400
        for i,point in enumerate(points):
            if i % STEP == 0:
                dc.SetPen(wx.Pen(self.color[(i // STEP) % len(self.color)]))
            dc.DrawPoint(point)
            time.sleep(0.003)

```

```
    #定义绘图消息的接收
    def On_Paint(self, event):
        dc = wx.PaintDC(self)
        a, b, R = self.x, self.y, self.r
        self.DrawCircle(a, b, R, dc)

app = wx.App() #app实例化

circle = CIRCLE() #窗口类实例化

circle.Show() #进行绘图

app.MainLoop() #主循环
```

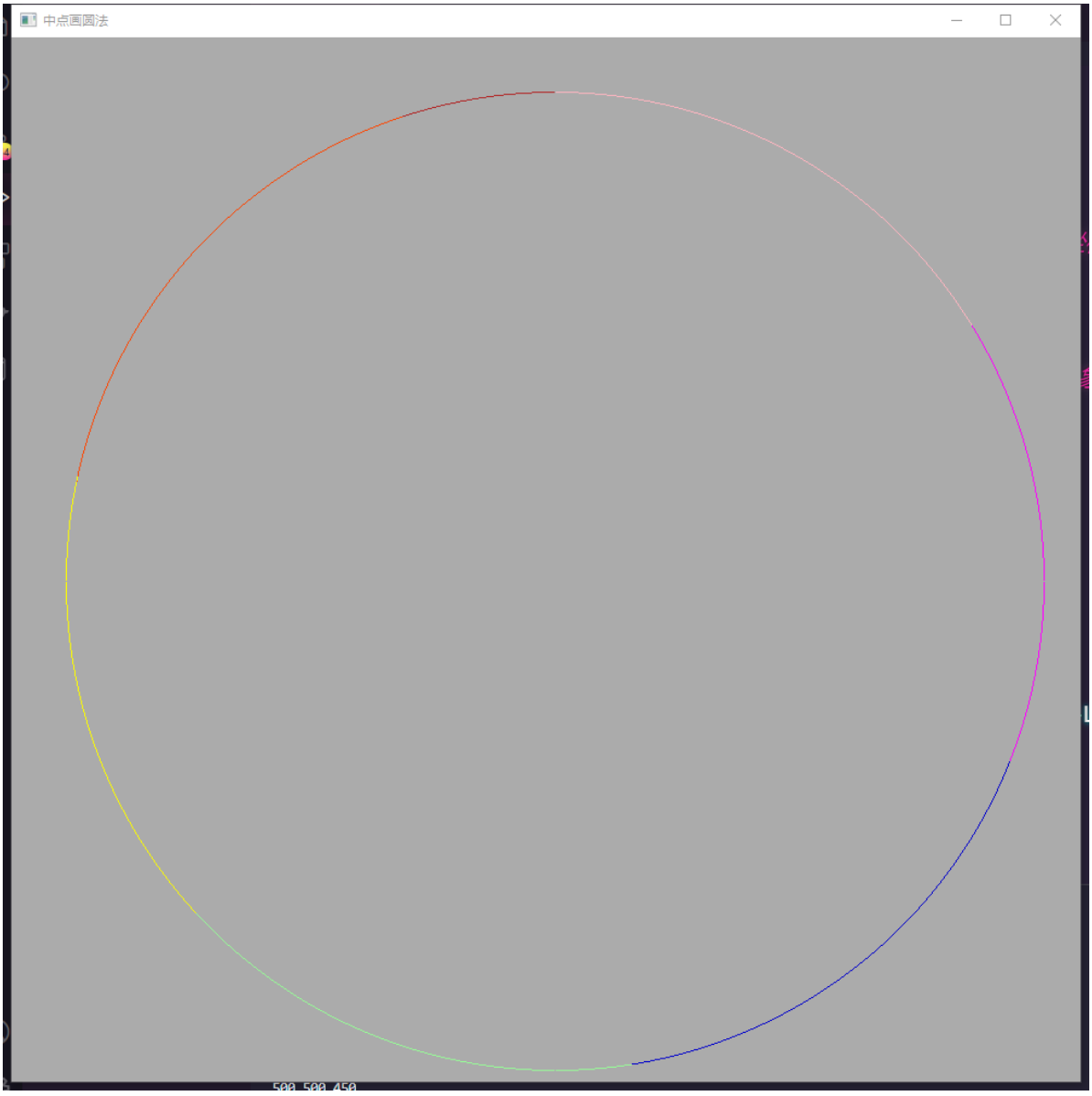
绘制椭圆：

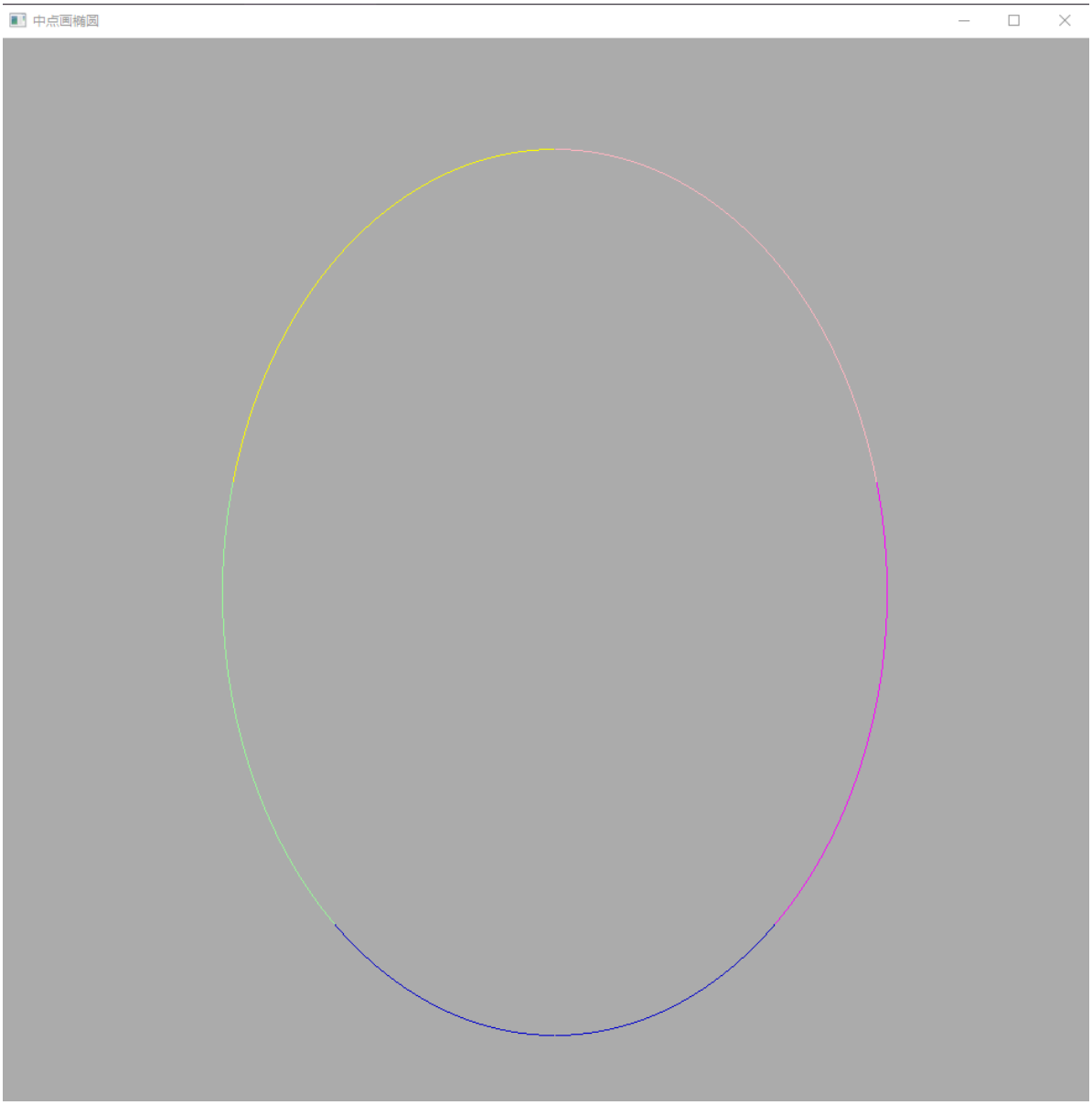
```
import wx
import time


class OVAL(wx.Frame):
    def __init__(self):
        #定义窗口的大小和位置
        super().__init__(None, title='中点画椭圆', size=(1000, 1000), pos=
(30, 30))

        #输入参数
        self.Ox, self.Oy, self.a, self.b = map(
            int,
            input("请依次输入椭圆中心坐标和长短半轴：\n").split())
        #将事件与函数进行绑定
        self.Bind(wx.EVT_PAINT, self.On_Paint)

        #颜色表
        self.color = [
            '#FFB6C1',
            '#FF00FF',
            '#0000CD',
            '#98FB98',
            '#FFFF00',
            '#FF4500',
            '#B22222',
            '#00BFFF',
        ]

    #绘图函数
    def DrawOval(self, Ox, Oy, a, b, dc):
        changed = 0 if a > b else 1  #用来判断取x或y为主方向
        a, b = max(a, b), min(a, b)
        point_list4 = []  #存储第四象限的点集
        A = a**2
        B = b**2
        x0 = int((A**2 / (A + B))**0.5)  #记录切线斜率为1的点横轴坐标
        x = 0
        y = b
```

```
39
40          p = B + A * (0.25 - b)
41
42          while x <= x0:
43              if changed:
44                  point_list4.append((y, x))
45              else:
46                  point_list4.append((x, y))
47              if p < 0:
48                  p += B * (3 + 2 * x)
49              else:
50                  p += B * (3 + 2 * x) + A * (2 - 2 * y)
51                  y -= 1
52              x += 1
53
54          p = B * (x + 0.5)**2 + A * (y - 1)**2 - A * B
55
56          while y > 0:
57              if changed:
58                  point_list4.append((y, x))
59              else:
60                  point_list4.append((x, y))
61              if p > 0:
62                  p += A * (3 - 2 * y)
63              else:
64                  p += A * (3 - 2 * y) + B * (2 * x + 2)
65                  x += 1
66              y -= 1
67          point_list1 = [(x, -1 * y) for x, y in point_list4]  #第一象限的点
68          point_list2 = [(-1 * x, -1 * y) for x, y in point_list4]  #第二象限
    的点
69          point_list3 = [(-1 * x, y) for x, y in point_list4]  #第三象限的点
70          #设置顺时针顺序进行绘画
71          point_list1.sort()
72          point_list2.sort()
73          point_list3.sort(reverse=True)
74          point_list4.sort(reverse=True)
75          points_O = point_list1 + point_list4 + point_list3 + point_list2
76
77          #平移坐标系
78          points = [(x + Ox, y + Oy) for x, y in points_O]
79
80          #这里利用time.sleep显示出动画效果
81          #为了颜值设置了每隔STEP个点换一支画笔颜色
82
83          STEP = 400
84          for i, point in enumerate(points):
85              if i % STEP == 0:
86                  dc.SetPen(wx.Pen(self.color[(i // STEP) %
    len(self.color)]))
87              dc.DrawPoint(point)
88              time.sleep(0.002)
89
90      #定义绘图消息的接收
91      def On_Paint(self, event):
92          dc = wx.PaintDC(self)
93          self.DrawOval(self.Ox, self.Oy, self.a, self.b, dc)
94
```

```
95
96  app = wx.App()  #app实例化
97
98  oval = OVAL()  #窗口类实例化
99
100 oval.Show()  #进行绘图
101
102 app.MainLoop()  #主循环
```

实验结果





实验心得

- 在设置决策参数p的时候将p赋值为1-R，目的是为了避免浮点运算，且基本不会影响绘图的准确性
- 我自己在制作绘图点的数组时，为了能够让动画显示出顺时针的绘画过程，需要对每一个象限的点分别进行排序，而且这些象限的点要满足窗口坐标系和常用直角坐标系之间的映射关系，而且在对四个象限的点进行合并之后要对其去重，且去重后的列表其元素的位置关系要保持不变。
- 有关四个象限的点之间的变换，大部分想的是做对称变换，即(x,y),(x,-y),(-x,y),(-x,-y)，我也想过做旋转变换，即(x,y),(y,-x),(-x,-y),(-y,x)，这样看来其实差别不大，肉眼基本观察不出来，但是为了和后面的椭圆相照应，决定放弃旋转变换，因为椭圆并不能通过旋转得到。
- 有关椭圆的绘制，已经很尽量避免了乘方开方等运算，但我觉得，这个实验的最终任务不是让我们把程序实现得多么快速精炼，更多的是让我们学习其中的算法，中点画圆和椭圆的思想一定要掌握