《计算机图形学》5 月报告

151110079，姚荣春，15895873878@163.com

2019 年 5 月 30 日

1 综述

图形学大作业期中报告，计划使用 python3+pyqt5 来实现。目前实现了命令行界面和图形化界面，可以从文本文件中读取命令执行，也可以通过简单的 GUI 进行绘图。功能上实现了重置画布，保存画布，设置画笔颜色，绘制线段，绘制多边形，绘制椭圆，绘制曲线，图元平移这几个功能。本次报告不涉及到性能测试，所有功能完成后再进行统一的性能测试。

2 系统框架设计

1. 程序入口文件与 GUI 实现：graphic.py
2. 画布相关与命令处理：canvas.py
3. 保存输出图像：bmp.py
4. 核心代码模块：line.py polygon.py curve.py ellipse.py

需要下载安全 pyqt：pip3 install PyQt5

通过 python3 graphic.py 命令来启动图形化界面或者通过 python3 graphic.py cmd.txt 命令来执行命令行命令。graphic 类初始化后形成一个简单的图形界面，之后通过将用户鼠标点击的信息转化为一系列核心代码所能执行的命令，通过调用 line.py 等类中的绘制函数进行图形绘制。canvas.py 继承了 bmp.py 因此是具有保存 bmp 功能的一个画布类，所有的图形也都通过相关类来实现，比如多边形类，直线类。画布类使用一个队列来保存画布上的所有图形类的实例，绝大部分的功能都是通过画布类的方法来实现，通过向队列中添加相关图形的类的实例来达到在画布中增加图形的目的。各个图形对应的一些比如平移裁剪等功能在各自类的相应方法中实现。

3 系统功能介绍

使用命令行：将命令储存在 cmd.txt 文件中，运行 python3 graphic.py cmd.txt 来执行 cmd.txt 中的命令

使用图形化界面：运行 python3 graphic.py 会启动图形化界面，具体功能如下：

1. 画直线：点击右下角的直线按钮后在图像上点击用于确定直线的两个点，再次点击直线按钮后画布上会显示出直线，默认使用 DDA 算法。

2. 画多边形：点击右下角的多边形按钮后，在画布上点击多个用于确定多边形的点，再次点击多边形按钮后，画布上会显示出多边形，默认使用 DDA 算法。

3. 画椭圆：点击右下角的椭圆按钮后，在画布上点击三个点，第一个点的坐标用于确定圆心（x,y），第二个点的横坐标记为 x2, 则 x2-x 作为横轴长度。第三个的点的纵坐标记为 y3 则 y3-x 作为椭圆的纵轴长度。再次点击多边形按钮后，画布上会显示出椭圆。

4. 画曲线：点击右下角的曲线按钮，在图像上点击用于确定曲线的多个控制点，再次点击曲线按钮后画布上会显示出一条曲线，默认使用 B-spline 算法且使用的是 Clamped 节点设置方法。

5. 保存图片：点击右下角的图片保存按钮即可完成图片保存，输出图片名默认为 output.jpg。

6. 修改画笔颜色，保存时设置图片名称等功能尚未实现

4 算法介绍

1. 重置画布

实现思想：在画布类中使用 numpy 数组来保存 R,B,G 三种颜色，重置的时候清空 numpy 数组并重新根据大小申请空间，并调用垃圾回收函数

2. 保存画布

实现原理：根据博客上的 BMP 格式详解直接构造 BMP 文件头和结构头，采用 24 位色彩按照 BGR 的顺序讲画布中的数据以二进制方式写入到文件

参考文献 BMP 格式详解 https://www.cnblogs.com/wainiwann/p/7086844.html

3. 设置画笔颜色

实现思想：根据参数直接修改画布类中的色彩

4. 绘制线段

DDA 算法实现原理：记斜率为 r，若 r<1 则沿 x 轴从 x1 点开始向 x2 点增长，记初始 y 值为 y1, 所计算点所对应的 y 值为 $y' + r$ 其中 y' 为上一个点计算的 y 值，而最终涂色的像素的 y 坐标为 $y' + r$ 的四舍五入。若 r>1 则从 y1 开始增长 y 的值，利用 y 去计算所对应的 x 的值。

Bresenham 算法实现原理：同 DDA 中一样根据斜率选取利用 x 坐标计算 y 坐标或者反

之。以利用 x 计算 y 为例 (并且 x 的取值按照 y 增长的方向), 因为 y 只有可能是 y'+1 或者维持 y' 不变。因此, 记 y=rx+c 计算出来的 y 值为 y_{real} 在 $y'+1-y_{real} < y_{real}-y'$ 的情况下坐标为 y'+1 否则维持不动。所以问题变为如何确定不等式是否成立。$r = (y2-y1)/(x2-x1)$ 如果 x2>x1 则左右同乘 $x2-x1$ 否则乘 $x1-x2$, 那么确定不等式是否成立只需要进行整数乘法, 从而避免了浮点数运算。如1中所示, (end_x-start_x)*(2*tmp_y+1-2*start_y)<2*(end_y-start_y)*(index_x-start_x) 为 Bresenham 实现的核心部分：即不等式是否成立的指示变量。

```
def draw_Bresenham(self,x1,y1,x2,y2):
    start_x=x1
    end_x=x2
    tmp_y=y1
    end_y=y2
    start_y=y1
    start_x=x1
    if(abs(y2-y1)<abs(x2-x1)):
        if(y2<y1):
            start_x=x2
            end_x=x1
            tmp_y=y2
            end_y=y1
            start_y=y2
        self.step=-1 if start_x>end_x else 1
        for index_x in range(start_x,end_x,self.step):
            if((end_x-start_x)*(2*tmp_y+1-2*start_y)<2*(end_y-start_y)*(index_x-start_x)):
                if(end_x>start_x):
                    tmp_y+=1
            else:
                if(end_x<start_x):
                    tmp_y+=1
            #self.point_vec[index_x][tmp_y]=1
            self.point_vec.append((index_x,tmp_y))
```

图 1: Bresenham 算法代码

5. 绘制多边形

实现方法：继承直线的类，调用绘制直线方法进行绘制多边形

6. 绘制椭圆

实现原理：中心圆算法。与直线的 Bresenham 算法思想相近，设椭圆的圆心为（x,y），横纵轴长度分别为 rx,ry，从（x,y+ry）起始，圆周曲线的斜率小于 1，设当前点的纵坐标为 yt, 当前横坐标为 xt, 那么 xt+1 点的纵坐标应为 yt 或 yt-1, 用于判别是否使用 yt-1 的不等式为 $\frac{xt-x}{rx}^2 + \frac{yt-0.5}{ry}^2 - 1$ 即（xt,yt-0.5）是否在椭圆内部。如果在椭圆内，则说明 ty 作为纵坐标更为合适，如果不在则以 yt-1 作为当前的纵坐标。当斜率小于 1 时，从（x+rx,y）起始，不断移动纵坐标来计算横坐标，直到斜率等于 1。判定（xt,yt）点附近是否斜率为 1 可以用以下不等式：ry*ry*xt-rx*rx*yt。当第一象限内的所有点都画完之后，根据第一象限内的点坐标来确定其他象限内对应点的坐标。椭圆算法的部分核心代码如图2所示。

7. 绘制曲线

参考文章：

```
for index_x in range(1,rx):
    if(ry*ry*index_x-rx*rx*y_index_high>0):
        print("reach break")
        break

    p= 4*ry*ry*index_x*index_x+rx*rx*(4*y_index_high*y_index_high-4*y_index_high+1-4*ry*ry)
    if(p>0):
        y_index_high-=1

    (self.point_vec).append((index_x+self.x,y_index_high+self.y))
    (self.point_vec).append((self.x-index_x,y_index_high+self.y))
    (self.point_vec).append((self.x-index_x,self.y-y_index_high))
    (self.point_vec).append((index_x+self.x,self.y-y_index_high))

for index_y in range(1,ry):
    if(ry*ry*x_index_high-rx*rx*index_y<=0):
        print("reach break")
        break

    p= 4*rx*rx*index_y*index_y+ry*ry*(4*x_index_high*x_index_high-4*x_index_high+1-4*rx*rx)
    if(p>0):
        x_index_high-=1
    (self.point_vec).append((x_index_high+self.x,index_y+self.y))
    (self.point_vec).append((self.x-x_index_high,index_y+self.y))
    (self.point_vec).append((self.x-x_index_high,self.y-index_y))
    (self.point_vec).append((x_index_high+self.x,self.y-index_y))
```

图 2: 中心圆算法代码

http://pages.mtu.edu/ shene/COURSES/cs3621/NOTES/
https://zhuanlan.zhihu.com/p/50626506

B-spline 算法理解：B-spline 算法根据 n 个控制点来绘制曲线。每个点的横纵坐标都由各个控制点的坐标乘以其对该点的影响参数来决定，为了确定每个控制点对计算点的影响大小，B-spline 引入了一个算法公式3。第 i 个控制点的影响因子为 $F(u,i) = N_{i,deg}(t)$。算法公式的计算所需要的参数为阶数和节点数组，这两个数据是受到人为定义的。人们可以通过改变阶数与节点数组的分布来画出不用的 B 样条曲线。

$$
\begin{aligned}
N_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \\
N_{i,p}(u) &= \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u)
\end{aligned}
$$

图 3: B 样条基本公式

1

B-spline 算法基本公式：图3为 B 样条曲线的基本公式，B 样条曲线通过此公式来计算曲线上各个点的横纵坐标，其中参数 u 的意义为当前点在曲线中的比重，比如曲线中点的比重为 0.5，起始点为 0。数组 u 为节点数组，其中的点由 0 向 1 单调递增，节点个数为阶数 + 控制点个数 +1，其中的值可以有多种分布形式，如果均匀分布则曲线为均匀 B 样条曲线即节点数组中两个点之间的大小差值为节点数分之一。除了均匀分布以外还有 clamped 等分布方式。

B-spline 算法实现原理：记 u 为所生成点所对应整条曲线的比重（u 为 0-1 之间的小数），比重 u 所对应的坐标点为 $C(u) = \Sigma_{i=0}^{n-1} N_{i,deg}(t) P_i$，其中 n 为控制点个数即为画曲线

命令所输入的点的个数，P_i 为第 i 个点的横纵坐标，deg 为阶数。B-spline 算法的具体实现如图4所示，支持任何合理阶数，由于命令行不要求输入阶数，我将阶数默认设置为 3，所此采用的节点分布为 clamped，因为这样画出的曲线初始点和末位点与第一个控制节点最后一个控制节点分别重合看起来美观。

```
def draw_Bspline(self):

    self.dim=3 #set the dim

    assert(self.n>=self.dim+1)
    self.node_size = self.n+self.dim+1
    self.u=[]

    #set the note val
    u_val=0.0
    inter = 1.0/(self.node_size-2*self.dim-1)
    for i in range(self.node_size):
        if(i<=self.dim):
            self.u.append(u_val)
        else:
            u_val+=inter
            if(u_val>=1.0):
                u_val=1.0
            self.u.append(u_val)

    start_rate=0.000
    last_val=(-1,-1)
    for i in range(1000):

        val=self.B_spline(start_rate)
        if(val!=last_val):
            last_val=val
            self.point_vec.append(val)
            #print(val)
        start_rate+=0.001
```

图 4: B 样条算法实现

Bezier 算法原理：下次再写。

5 总结

...期末在写