《计算机图形学》6 月报告

151110079，姚荣春，15895873878@163.com

2019 年 6 月 16 日

1 综述

图形学大作业期末报告，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。功能上实现了要求的所有功能：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，图元旋转，图元缩放，线段的剪裁。最终报告分为两个部分，首先是同框架设计，主要说明了图形化界面，命令行如何使用，支持哪些功能，如何对不同功能代码模块进行分割与架构。第二部分为算法介绍：报告对各个功能的实现进行了相关算法的原理介绍并阐述了自己对算法实现的理解，并贴出了部分代码实现和对实现的具体解释。

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 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：

图形化界面组成部分：左侧为一个 800*800 的固定大小画布，画布可以随着 reset 的大小自适应调整点击坐标，即使是 reset 为 100*100 的画布也可以进行绘制。右上方第一个文本框依次展示了所有的图元的信息。右上方第二个文本框展示了当前点击，绘图时的中间信息。右下方的文本框可以输入命令行命令并点击执行命令行按钮来运行命令行命令。

1. 画直线：点击右下角的直线按钮后，会收到开始绘制直线的提示，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，并提示已经完成直线的绘制。默认使用 Bresenham 算法。

2. 画多边形：点击右下角的多边形按钮后，会收到开始绘制多边形的提示，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，并提示完成了多边形的绘制。默认使用 Bresenham 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-y 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. 右上角的文本框显示了所有存在的图元的信息，下方的文本框显示了所有的动态响应信息，比如点击鼠标的位置，开始执行的操作，最下方的文本框可以用于执行命令行命令，过程如下：输入命令，点击执行命令行按钮。

7. 清空画布：点击清空画布按钮即可清空当前的画布。

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)，假设直线的方程为 y=rx+c，r 为斜率，c 为可计算出的常数。设 (x', y') 为上一个已经计算完毕的点的坐标，那么下一个点的横坐标为 x'+1, 纵坐标 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。图1为 Bresenham 算法在斜率小于 1 时的代码实现。斜率小于 1 时，以 (x1,y1)，(x2,y2) 中 y 坐标较小的点为起始坐标点，不断的递增或递减 x 坐标直到到达终点，在这一过程中，根据确定好的 x 坐标来计算 y 坐标是否在前一个点的 y 坐标基础上增加 1 或者不变。那么如何确定 y 坐标的变化？我采用了如下判别式 (end_x-start_x)*(2*tmp_y+1-2*start_y)<2*(end_y-start_y)*(index_x-start_x) 作为 Bresenham 实现的核心部分：即不等式是否成立的指示变量。

```
1    def draw_Bresenham(self,x1,y1,x2,y2):
2        start_x=x1
3        end_x=x2
4        tmp_y=y1
5        end_y=y2
6        start_y=y1
7        start_x=x1
8        if(abs(y2-y1)<abs(x2-x1)):
9            if(y2<y1):
10               start_x=x2
11               end_x=x1
12               tmp_y=y2
13               end_y=y1
14               start_y=y2
15           self.step=-1 if start_x>end_x else 1
16           for index_x in range(start_x,end_x,self.step):
17               if((end_x-start_x)*(2*tmp_y+1-2*start_y)<2*(end_y-start_y)*(index_x-start_x)):
18                   if(end_x>start_x):
19                       tmp_y+=1
20               else:
21                   if(end_x<start_x):
22                       tmp_y+=1
23               #self.point_vec[index_x][tmp_y]=1
24               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：使用了非常简单的方法来实现绘制多边形的算法：继承绘制直线的类，调用绘制直线方法进行绘制多边形。算法代码如图2所示，第三行是对所有的控制节点的 x 坐标进行便利，对相应的直线调用直线绘制函数，当所有线段均绘制完毕后，多边形即绘制成功。

6. 绘制椭圆

中心圆算法基本原理：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：$ry \times ry \times xt - rx \times rx \times yt$。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。

中心圆算法实现：椭圆算法的部分核心代码如图3所示，第 18 行的 p 即为判断绘制第一象限的椭圆的时点（xt,yt-0.5）是否在椭圆内部的判别式，如果大于零则说明不在内部，需要将 y 坐标下降 1 作为新的点的 y 坐标。

7. 绘制曲线

参考文章：
http://pages.mtu.edu/ shene/COURSES/cs3621/NOTES/
https://zhuanlan.zhihu.com/p/50626506
https://www.cnblogs.com/hnfxs/p/3148483.html

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。

```
2   def draw_poly2(self):
3       for i in range(len(self.xvec)):
4           if(i==len(self.xvec)-1):
5               self.draw_line(self.xvec[-1],self.yvec[-1],self.xvec[0],self.yvec[0],algorithm)
6               break
7           x1_index=self.xvec[i]
8           y1_index=self.yvec[i]
9           x2_index=self.xvec[i+1]
10          y2_index=self.yvec[i+1]
11          self.draw_line(x1_index,y1_index,x2_index,y2_index,self.algorithm)
12
13
```

图 2: 多边形绘制算法

B-spline 曲线的特点：B-spline 曲线与贝塞尔曲线相比，有很多优点：一个 B 样条曲线可以是一个贝塞尔曲线，B 样条曲线满足贝塞尔曲线所具有的所有重要性质。B 样条曲线提供了比贝塞尔曲线更灵活的控制：可以通过改变节点的值分布或插入重节点来对曲线形状进行控制。同时，B 样条曲线具有局部性，阶数可以用户定义，而贝塞尔曲线基函数的次数等于控制顶点数减 1。除此之外 B 样条曲线还具有很多有用的特性，这里不一一列举。

B-spline 算法基本公式：图4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 样条基本公式的代码实现为图5。

B-spline 算法实现：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 算法的具体实现为图6和图5，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。

图6中第 18-27 行为对控制节点进行赋值，所采用的是 clamped 分布方式，即前阶数个点为 0，后阶数个点为 1，其余的点为均匀分配，这样绘制出的曲线较为美观。31-38 行为以 1000 个点的密度对曲线进行采样，如果新的点与上一个点坐标相同则抛弃，不同则保留。

```
1     def draw_ellipse(self,x,y,rx,ry):
2         print(x,y,rx,ry)
3         self.x = x
4         self.y = y
5         self.rx = rx
6         self.ry = ry
7         (self.point_vec).append((x,y+ry))
8         (self.point_vec).append((x,y-ry))
9         (self.point_vec).append((x-rx,y))
10        (self.point_vec).append((x+rx,y))
11        y_index_high = ry
12        x_index_high = rx
13
14        for index_x in range(1,rx):
15            if(ry*ry*index_x-rx*rx*y_index_high>0):
16                print("reach break")
17                break
18            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)
19            if(p>0):
20                y_index_high-=1
21            (self.point_vec).append((index_x+self.x,y_index_high+self.y))
22            (self.point_vec).append((self.x-index_x,y_index_high+self.y))
23            (self.point_vec).append((self.x-index_x,self.y-y_index_high))
24            (self.point_vec).append((index_x+self.x,self.y-y_index_high))
25        for index_y in range(1,ry):
26            if(ry*ry*x_index_high-rx*rx*index_y<=0):
27                print("reach break")
28                break
29            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)
30            if(p>0):
31                x_index_high-=1
32            (self.point_vec).append((x_index_high+self.x,index_y+self.y))
33            (self.point_vec).append((self.x-x_index_high,index_y+self.y))
34            (self.point_vec).append((self.x-x_index_high,self.y-index_y))
35            (self.point_vec).append((x_index_high+self.x,self.y-index_y))
36
```

图 3: 中心圆算法代码

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

图 4: B 样条基本公式

1

```
def N(self,i,p,u):
    if(p==0):
        if(self.u[i]<=u and u<self.u[i+1]):
            return 1.0
        else:
            return 0.0
    else:
        if(self.u[i+p]-self.u[i]==0):
            first_arg=0.0
        else:
            first_arg=((u-self.u[i])/(self.u[i+p]-self.u[i]))
        if((self.u[i+p+1]-self.u[i+1])==0):
            second_arg=0.0
        else:
            second_arg=((self.u[i+p+1]-u)/(self.u[i+p+1]-self.u[i+1]))


        ret_val=  first_arg*self.N(i,p-1,u)+second_arg*self.N(i+1,p-1,u)
        return ret_val
```

图 5: 基函数实现

```
    def B_spline(self,t):
        x_val=0.0
        y_val =0.0
        for i in  range(0,self.n):
            x_val+=self.N(i,self.dim,t)*self.xvec[i]
            y_val+=self.N(i,self.dim,t)*self.yvec[i]
        return (round(x_val),round(y_val))

    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 6: B 样条算法实现

1-7 行的函数为调用基函数 N 对各个控制点的权重进行计算。图5为基函数的实现。

Bezier 算法原理：贝塞尔曲线利用基函数生成对各个控制点的权重，这些权重累加起来为 1，各个点的权重与各个点的坐标相乘结果的累加为生成点的坐标。贝塞尔曲线具有全局性，基函数的阶数为控制点个数减 1。贝塞尔曲线的基函数如图7所示。

$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1,2,\cdots,n,\ i = 0,1,\cdots,n-k \end{cases}$$

图 7: 贝塞尔曲线基函数

```
1     def C(self,a,b):
2         return (math.factorial(a)/math.factorial(b))/math.factorial(a-b)
3
4     def draw_Bezier(self):
5         self.point_vec.append((self.xvec[0],self.yvec[0]))
6         self.point_vec.append((self.xvec[-1],self.yvec[-1]))
7
8         rate = 0.000
9         last_val=(-1,-1)
10        for i in range(1000):
11            t=rate
12            B_tx=0.0
13            B_ty=0.0
14            for j in range(0,self.n):
15                B_t =self.C(self.n-1,j)*((1-t)**(self.n-1-j))*(t**j)
16                B_tx += B_t*self.xvec[j]
17                B_ty += B_t*self.yvec[j]
18            #print((round(B_tx),round(B_ty)))
19            if((round(B_tx),round(B_ty))!=last_val):
20                last_val= (round(B_tx),round(B_ty))
21                self.point_vec.append((round(B_tx),round(B_ty)))
22            rate+=0.001
23
```

图 8: 贝塞尔曲线基函数

Bezier 算法实现：图8为贝塞尔曲线的代码实现。利用 python3 中 math 库的阶乘函数实现了贝塞尔曲线的基函数。循环中的 16-17 行为利用权重乘控制点坐标生成当前点的具体坐标。与 B 样条曲线相同，对曲线划分成 1000 份进行点生成，除去坐标相同的点，剩下的点组成了贝塞尔曲线。

8. 图元平移

图元平移算法原理：对平移图元的所有控制点进行平移，之后调用图元生成函数对图元进行重新生成，以线段为例，两个断点坐标同时平移，之后调用线段绘制函数生成平移后的线段。

9. 图元旋转

图元旋转算法原理：控制点 (x1, y1) 围绕点 (x, y) 进行逆时针旋转角度 d 后的横坐标为 $x+(x1-x)\times cos(d)-(y1-y)\times sin(d)$，纵坐标为 $y+(x1-x)\times sin(d)+(y1-y)\times cos(d)$。将图元所有的控制点都旋转过后，利用旋转后的控制点生成新的图形。

10. 图元缩放

图元缩放算法原理：控制点（x1，y1）围绕点（x，y）进行 r 倍缩放后的横坐标为 $x+(x1-x)*r$，纵坐标为 $y+(y1-y)*r$ 将图元的所有控制点都缩放后，调用图元生成函数生成新的图片。

11. 线段裁剪

Cohen-Sutherland 算法原理：Cohen 算法将剪裁矩形分为了 9 个部分，使用一个四位的 01 向量来表示这 9 个部分，如 0000 为点在剪裁矩形中的情况。当两个点都在矩形中即两个点都是 0000 时，不需要进行剪裁，当两个点的向量的与值不为 0 时，两个点所表示的线段必定在矩形外部，这时直接将线段删除即可，除了这两种情况下，通过对各个边界的所有可能交点进行计算，来画出剩下剪裁后的线段。Cohen 算法对矩形的区域划分如图9所示。

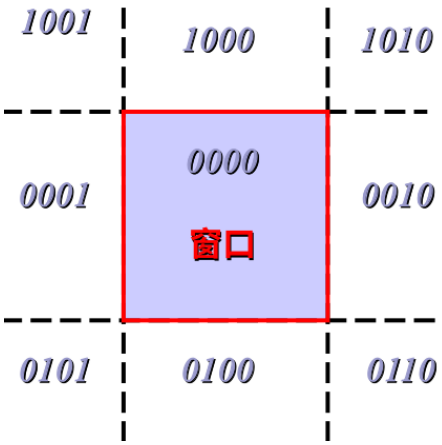

图 9: Cohen 算法的区域划分

参考文章：

https://blog.csdn.net/daisy___ben/article/details/51941608

Liang-Barsky 算法原理：如果斜率为正值，梁的算法将矩形的左边延长到了直线 x=xmin 上（xmin 为剪裁矩形的 y 坐标最小值），将矩形的上边延长到了直线 y=ymax 上（ymax 为剪裁矩形的 x 坐标最大值），这样一来线段所在的直线的四个交点就被移植到了四个直线上：x=xmax，y=ymin，y=ymax，x=xmin 梁算法利用了直线的另一种表现形式来计算交点：x=x1+u(x2-x1)，y=y1+u(y2-y1)，其中 x1，y1，x2，y2 为线段的两个控制点的坐标，u 的物理含义为这个点是线段的前 u 部分，因此 u 的取值为 0-1。梁剪裁算法的示意图为图10。

满足以下两个表达式的点一定位于剪裁矩形中：

xmin <= x1 + u(x2-x1) <= xmax

ymin <= y1 + u(y2-y1) <= ymax

不妨引入 p 和 q 来进一步简化上面的两个不等式：

p1 = -(x2-x1), q1 = x1 - xmin （左边界）

p2 = (x2-x1), q2 = xmax - x1 （右边界）

p3 = -(y2-y1), q3 = y1 - ymin （下边界）

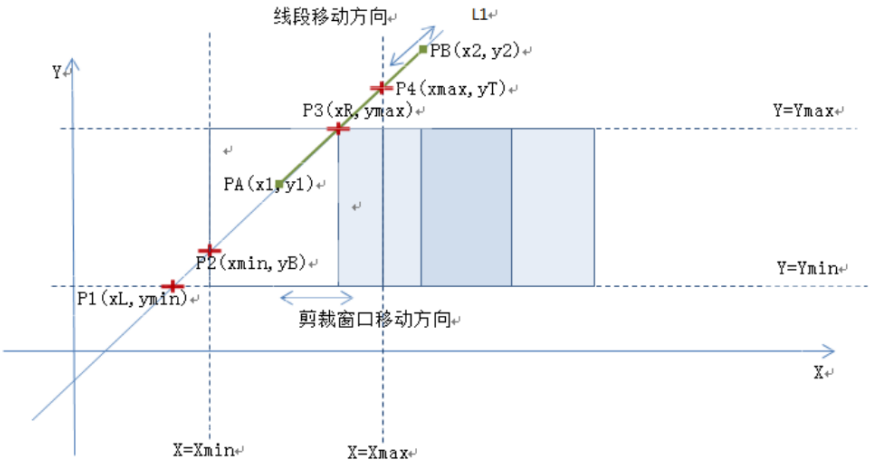

图 10: Liang-Barsky 算法的边界延伸

2

p4 = (y2-y1), q4 = ymax - y1 （上边界）

p 和 q 的值即定义了边界焦点的情况，以 u1 为例，即左边界的交点的位置：如果在矩阵中则需要满足 xmin <= x1 + u(x2-x1) <= xmax 与 ymin <= y1 + u(y2-y1) <= ymax，即为 p1t<=q1，因此边界条件的 t 为 q1/p1，以此类推可以得到所有的四个交点，将其中不在 0-1 范围内的所有点去除，剩下的点（有可能包含两个控制点）即为线段所需要的控制点。

5 总结

...期末在写