计算机图形学大作业实验报告

学号：**171860004** 姓名： **戴若石** 邮箱：2062499982@qq.com

1 实验内容

实验要求是使用Python3语言完成一个绘图系统，能够绘制多种图形并对图形进行编辑操作。

我实现的绘图系统功能包括：

- 绘制图形：绘制线段（DDA、Bresenham）、绘制多边形、绘制椭圆、绘制曲线（Bezier、B-spline）。
- 编辑图形：平移、旋转（对椭圆除外）、缩放、对线段裁剪。
- 文件操作：重置画布、保存画布。
- 其他：设置画笔颜色。

2 代码框架

总共有三个代码文件：

2.1 cg_algorithms.py

这个文件主要实现图像绘制和编辑的算法，其他两个文件可以直接调用这个文件中写好的算法代码。

2.2 cg_cli.py

这个文件实现命令行界面程序。

由于命令行界面程序不用实时显示生成的每一个图元，只需要在保存画布的时候将图元画出并保存即可。故用字典存每个图元的控制点信息和算法信息，在保存画布的时候调用cg_algorithms里写好的算法将图元依次画出并保存即可。

用numpy库中的数组模拟点阵图像的点阵，在需要显示颜色的点对应的数组处存储颜色信息。

用PIL库中的Image类生成图像并保存。

2.3 cg_gui.py

这个文件实现用户交互界面程序，其中使用了PyQt5库。

1. 用MyCanvas类实现画布的窗体，这个类继承自QGraphicsView。

   这个类中实现了对于绘制或编辑图元时鼠标在画布上的点击移动等事件的处理方法，也实现了对于图元选择的处理方法。

2. 用MyItem类实现图元，这个类继承自QGraphicsItem。

   这个类中通过调用cg_algorithm里写好的算法实现了对应图元的绘制（paint）方法和其选择框的计算方法。

3. 用MainWindow类实现主窗口，这个类继承自QMainWindow类。

   这个类中主要实现了交互界面的窗口设置，文件操作，以及对绘制编辑图元的方法的调用。

3 算法介绍

对于绘制算法，凭借算法输入可以确定需要显示的图元的数学方程式。算法所需要完成的是输出离这个方程式距离最近的整数点集。

对于编辑算法，输入为待编辑的图元编号和图元控制点坐标，对这些坐标进行相应变换以得到最终结果。

3.1 绘制线段

两点确定一条直线，算法输入为两个端点的坐标$(x_0, y_0), (x_1, y_1)$。

绘制线段的两个算法都是选取x或y方向为取样方向，以最小单位1为增量的取值，依次算出取样位置对应的点的坐标。

取增长更快的方向为取样方向（即斜率的绝对值小于1时取x方向， 斜率的绝对值大于1时取y方向），这样可以保证画出来的线段的连贯性。

3.1.1 DDA（数值差分分析）算法

通过输入的两个端点坐标可以确定直线的斜率$m = \frac{y_1 - y_0}{x_1 - x_0}$，因此可以根据取样方向的增量算出另一方向对应的增量值。

以斜率小于1为例，此时取x方向为取样方向，$x_{k+1} = x_k + 1$，从k=0开始依次计算y_{k+1}：$y_{k+1} = int(y_k + m)$，其中int()代表取整。

3.1.2 Bresenham算法

由于取增长更快的方向为取样方向，那么非取样方向的增长一定慢于取样方向。取样方向的增量为1，也就是说非取样方向要么没有增长，要么增长为1。也就是说当(x_k, y_k)已经确定了，那只需要判断(x_{k+1}, y_k)和$(x_{k+1}, y_k + 1)$哪个点离直线更加接近即可确定y_{k+1}的值。

由于只需要判断哪个离得更近，所以不需要计算出距离，只需要计算出与距离的差值符号相同的决策参数。

Bresenham算法流程如下：

决策参数初始化为$p_0 = 2\Delta y - \Delta x$。

从k=0开始依次确定y_{k+1}值并更新决策参数如下：

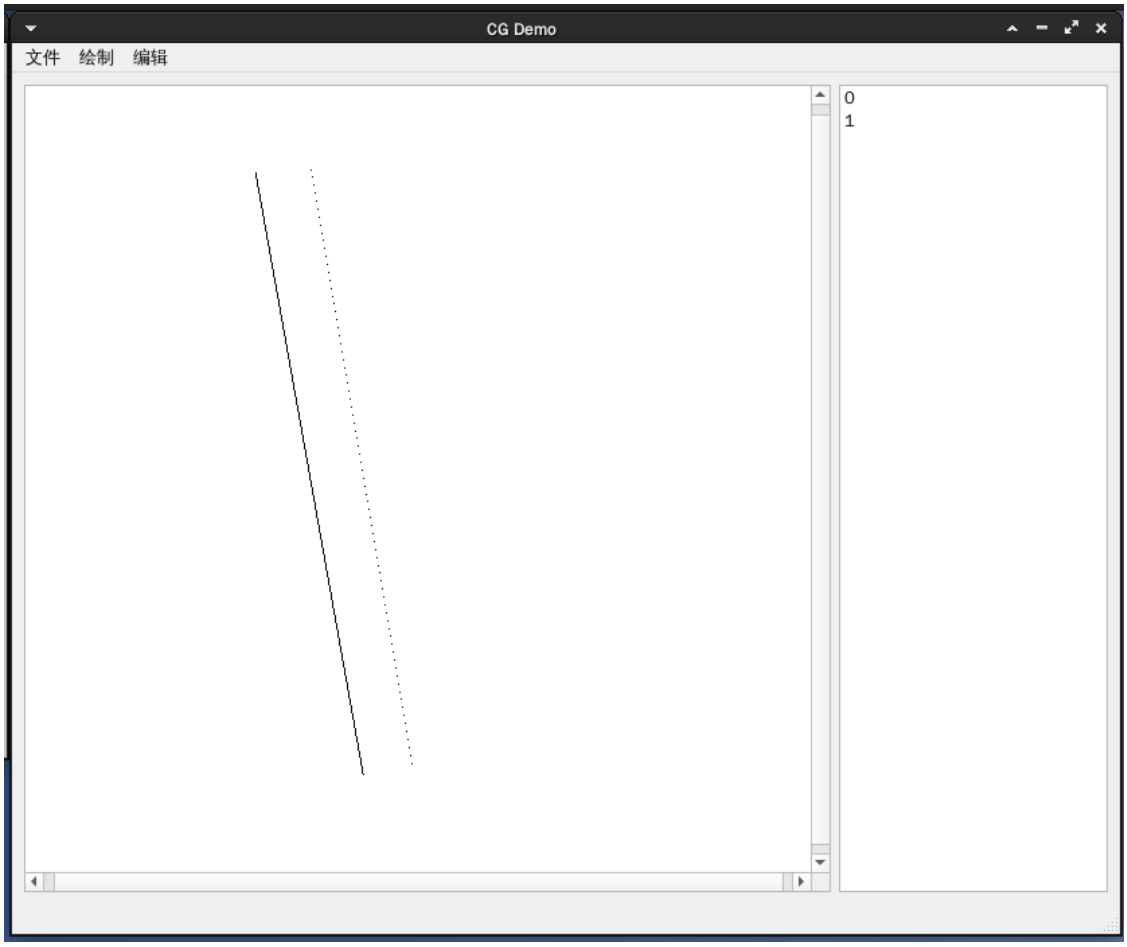
- 当$p_k < 0$时，$y_{k+1} = y_k$，$p_{k+1} = p_k + 2\Delta y$
- 当$p_k > 0$时，$y_{k+1} = y_k + 1$，$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

3.1.3 算法比较

CG_demo中给出了一个Naive算法是直接以x方向为取样方向，依次按照直线方程计算对应的y值。

- DDA算法与Naive算法相比，优点是在斜率很大时不会出现画出来的是不连贯的点集而非线段的问题，且计算过程中采用累加而非浮点数与整数相乘，性能更好。

  下图中左边为DDA所画，右边为Naive所画。



- DDA算法与Bresenham算法相比，缺点是计算斜率时会有误差，且在之后的累加过程中误差会积累得越来越大。

3.2 绘制多边形

绘制多边形的输入是顶点集。只需要依次调用对应的绘制线段算法将多变形的边依次画出即可。

3.3 绘制椭圆（中点圆生成算法）

绘制椭圆的输入是椭圆举行包围框的左上角和右下角顶点坐标，根据这两个坐标可以确定椭圆的中心点、长轴长和短轴长。

由于椭圆的对称性质，只需要确定第一象限的四分之一椭圆轨迹即可得到整个椭圆。

绘制椭圆的算法的核心思想与Bresenham绘制线段算法一样，都是计算决策参数以确定非取样方向的增量。

将四分之一椭圆轨迹根据切线斜率的绝对值与1的关系分成两个部分，防止出现椭圆轨迹不连贯的现象。

设椭圆中心为(x_c, y_c)，椭圆x方向轴长为r_x，椭圆y方向轴长为r_y。绘制椭圆算法流程如下：

令(x_0, y_0)为$(0, r_y)$，初始化决策参数为$p_0 = r_y^2 - r_x^2 r_y + r_x^2/4$

从$k = 0$开始循环测试，循环到$2r_y^2 x \geq 2r_x^2 y$，这部分$x_{k+1} = x_k + 1$:

- 当$p_k < 0$时，$y_{k+1} = y_k$，$p_{k+1} = p_k + 2r_y^2 x_{k+1} + r_y^2$
- 当$p_k > 0$时，$y_{k+1} = y_k - 1$，$p_{k+1} = p_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$

根据第一个部分最后的点(x_1, y_1)计算第二个部分的决策参数初始值为
$p_0 = r_y^2(x_1 + 1/2)^2 + r_x^2(r_y - 1)^2 - r_x^2 r_y^2$

从k=0开始循环测试，循环到$y \leq 0$，这部分$y_{k+1} = y_k - 1$:

- 当$p_k > 0$时，$x_{k+1} = x_k$，$p_{k+1} = p_k - 2r_x^2 y_{k+1} + r_x^2$
- 当$p_k < 0$时，$x_{k+1} = x_k + 1$，$p_{k+1} = p_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

每一个点(x_k, y_k)在确定后都向结果点集中加入四个点：
$(x_c + x_k, y_c + y_k), (x_c + x_k, y_c - y_k), (x_c - x_k, y_c + y_k), (x_c - x_k, y_c - y_k)$

3.4 绘制曲线

曲线绘制可以绘制两种不同的曲线：Bezier曲线和B-spline曲线。

3.4.1 Bezier曲线

- 曲线介绍：

核心思想：曲线可用参数形式表示，对参数进行离散化取值以得到曲线上的点。

Bezier曲线的参数定义为：$P(u) = \sum_{k=0}^{n} P_k BEZ_{k,n}(u)$

其中$n+1$为控制点个数，P_k为第k个控制点坐标，$BEZ_{k,n}(u)$是Bernstein基函数，定义如下：
$BEZ_{i,n}(u) = C(n,i)u^i(1-u)^{n-i}$

$BEZ_{k,n}(u)$有降阶公式如下：

$BEZ_{k,n}(u) = (1-u)BEZ_{k,n-1}(u) + uBEZ_{k-1,n-1}(u)$

且$BEZ_{0,n} = (1-u)BEZ_{0,n-1}, BEZ_{n,n} = uBEZ_{n-1,n-1}$

带入参数定义展开得：

$$
\begin{aligned}
&P(u) = (1-u)P_0 BEZ_{0,n-1}(u) + (1-u)P_1 BEZ_{1,n-1}(u) + uP_1 BEZ_{0,n-1}(u) + \\
&(1-u)P_2 BEZ_{2,n-1}(u) + uP_2 BEZ_{1,n-1}(u) + \cdots + (1-u)P_{n-1} BEZ_{n-1,n-1}(u) + \\
&uP_{n-1} BEZ_{n-2,n-1}(u) + nP_n BEZ_{n-1,n-1}(u) \\
&= ((1-u)P_0 + uP_1)BEZ_{0,n-1}(u) + ((1-u)P_1 + uP_2)BEZ_{1,n-1}(u) + \cdots + \\
&((1-u)P_{n-1} + uP_n)BEZ_{n-1,n-1}(u) \\
&= \sum_{k=0}^{n-1}((1-u)P_k + uP_{k+1})BEZ_{k,n-1}(u) \\
&= (1-u)\sum_{k=0}^{n-1} P_k BEZ_{k,n-1}(u) + u\sum_{k=0}^{n-1} P_{k+1} BEZ_{k,n-1}(u)
\end{aligned}
$$

根据以上式子，可以写出$P(u) = P_0^n$的递归生成式如下：

当$r = 0$时，$P_i^r = P_i$（P_i为第i个控制点）。

当r > 0时，$P_i^r = (1-u)P_i^{r-1} + uP_{i+1}^{r-1}$，其中$i = 0, 1, 2, \cdots, n-r$，$r = 1, 2, \cdots, n$。

- 实现方式

由于上述递归生成式较简单，故而使用循环完成，最外层对u进行循环，接着对r进行循环，接着对i进行循环。r循环层内需记录上一次循环的结果用于下一次循环的生成。

对于u的离散化取值暂时取定制，步长为1/10000。

3.4.2 B-spline曲线

- 曲线介绍：

B-spline曲线是对Bezier曲线的一般化推广。核心思想是把参数区间分成很多小区间，每一段连续的小区间分别构成曲线的一部分。

$n+1$个控制点生成的$k+1$阶（k次）B-spline曲线的定义如下：

$P(u)=\sum_{i=0}^{n}P_iN_{i,k+1}(u)\qquad u\in[u_k,u_{n+1})$

其中$N_{i,k+1}(u)$是B样条的基函数，是一个非递减的参数为u的$k+1$阶分段多项式，定义如下：

当$k=0$时，如果$u\in[t_i,t_{i+1})$，$N_{i,0}(u)=1$，否则$N_{i,0}(u)=0$。

当$k>1$时，$N_{i,k+1}(u)=\frac{u-u_i}{u_{i+k}-u_i}B_{i,k}(u)+\frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}}B_{i+1,k}(u)$。

- 实现方式：

需要实现的算法是三次均匀B样条曲线，即$k=3$且对参数区间均匀划分。

B样条基函数的计算通过写了一个递归调用的函数实现。对参数u进行一次取值按照B-spline曲线的定义计算曲线上点的坐标。

3.4.3 对比

- 曲线样式不同：Bezier曲线的端点即为首末控制点，B-spline曲线不通过首末控制点。由于这个区别，在实现GUI的绘制曲线部分时，绘制两种不同的曲线，选取控制点的方式和顺序也不一样。
- 曲线性质不同：调整单个控制点的位置时，整个Bezier曲线都会受其影响，因为曲线上的每个点都与每个控制点相关；而B-spline曲线只有部分段会受其影响。即，对曲线的部分调整B-spline更方便。

3.5 图元平移、旋转和缩放

核心思想是对图元的输入点列表进行平移旋转和缩放，例如线段就是对于线段两端点进行对应操作。这样最后得到的图元就会整体进行平移旋转和缩放。

由于绘制椭圆只实现了对于长轴平行或垂直于x轴的椭圆的绘制，故而椭圆无法进行任意角度的旋转，只能旋转90°的倍数。用户交互界面中椭圆无法旋转。

旋转的实现思路是将平面直角坐标系转换为极坐标系，旋转完成之后再将极坐标系中的点映射到平面直角坐标系中就能得到结果。为此实现了一个函数cal_r，输入平面直角坐标，输出将这个坐标映射到极坐标系后得到的极坐标中的角度。

3.6 对线段裁剪

输入线段的两个端点坐标，输出裁剪窗口内的线段的端点坐标。

对线段裁剪有两个算法：Cohen-Sutherland算法和Liang-Barsky算法。

3.6.1 Cohen-Sutherland算法

核心思想：通过编码测试来减少需要计算交点的次数。

首先对于线段的两个端点，将端点坐标与裁剪窗口的四个边界进行比较得到每一位的编码，再通过编码的计算比较筛出完全在窗口内不需要裁剪，和部分完全在窗口外的情况。

接下来依次计算窗口的四个边界与线段所在直线的交点，并进行判断和修改线段端点坐标，最终得到裁剪结果。

3.6.2 Liang-Barsky算法

核心思想：从参数方程的角度考虑线段。

端点坐标为$(x_1,y_1),(x_2,y_2)$的线段的参数方程为：

$x=x_1+u(x_2-x_1)$

$y = y_1 + u(y_2 - y_1)$

其中，$u \in [0,1]$。令$\Delta x = x_2 - x_1, \Delta y = y_2 - y_1$。

用左上角坐标为(x_{min}, y_{min})，右下角坐标为(x_{max}, y_{max})的裁剪窗口去裁剪这个线段即需要求出参数区间，使得区间内的参数u满足如下不等式：

$$\begin{cases} x_{min} \le x_1 + u\Delta x \le x_{max} \\ y_{min} \le y_1 + u\Delta y \le y_{max} \end{cases}$$

可将其拆成四个不等式如下：

$$\begin{cases} u \cdot (-\Delta x) \le x_1 - x_{min} \\ u \cdot \Delta x \le x_{max} - x_1 \\ u \cdot (-\Delta y) \le y_1 - y_{min} \\ u \cdot \Delta y \le y_{max} - y_1 \end{cases}$$

令$p_1 = -\Delta x, p_2 = \Delta x, p_3 = -\Delta y, p_4 = \Delta y$，
$q_1 = x_1 - x_{min}, q_2 = x_{max} - x_1, q_3 = y_1 - y_{min}, q_4 = y_{max} - y_1$。

则上述不等式组可简化为：$u \cdot p_k \le q_k, (k = 1,2,3,4)$。

$k = 1,2,3,4$分别对应了左、右、下、上四条边界。

当$p_k = 0$时说明线段与对应边界平行，此时需要特殊处理。

当$p_k < 0$时说明对于对应边界，线段方向是从窗口外指向窗口内，即对应边界是一条入边边界；同理，当$p_k > 0$时说明对应边界是一条出边边界。

$u = \frac{q_k}{p_k}$是对应边界与线段交点的参数值，想要求出裁剪后的线段的端点，即需要在0和两条入边与线段的交点中选取一个最大的参数值u_1，在1和两条出边与线段的交点中选取一个最小的参数值u_2，再根据这两个参数值求出端点坐标：

$(x_1 + u_1 \cdot \Delta x, y_1 + u_1 \cdot \Delta y), (x_1 + u_2 \cdot \Delta x, y_1 + u_2 \cdot \Delta y)$。

3.6.3 算法比较

相比Liang-Barsky算法，Cohen-Sutherland算法虽然也对一些情况进行了筛选，但是还是过于繁琐，需要考虑的情况太多了，写代码的时候也很容易出错。

而Liang-Barsky算法通过数学参数化的方法，将裁剪窗口的四条边归于同一类，可以进行统一处理，即精简了代码，也使得代码的可读性更高。

4 总结

1. 对于图形学的一些算法有了更深的理解，实现过后对于算法的一些细节以及不同算法在实现上的优缺点更加了解。
2. 实现过程中对于python语言有了更好的运用。

参考资料

课程ppt: CG-03-Primitives、CG-05-Trimming、CG-06-Curve and Surface

菜鸟教程（python语言参考） https://www.runoob.com/python/python-tutorial.html

B-spline曲线参考 https://zhuanlan.zhihu.com/p/50450278