计算机图形学　11 月报告

夏宇　171180554

(南京大学 计算机科学与技术系, 南京　210093)

摘　要:　通过学习课本上所讲的线条、椭圆画线算法，以及平移、旋转、填充的相关算法，并且利用 Qt Creator+ PyQt5 进行软件系统 GUI 的编写开发。通过手动实现课本中的相关画图算法，以及构建完整的图形界面系统，来实现作业所需要的画图、打开、保存以及图形显示的相关功能。

1　引言

计算机图形学大作业要求实现相关图元绘制算法，以及相关联的图元操作手段，并且要求能够：一、通过命令行读取命令文件绘制；二、通过图形界面获取鼠标信息绘制图片的两种方式进行输入操作。在作业要求的基础上，通过对一个基本的画图软件的需求进行分析，来确定所需要实现的功能以及采用的框架系统。并且，对已经实现和待实现的算法进行相关的解释，对已经实现和待采用的系统软件框架做一个描述。根据我的对图形软件的需求分析和自己希望了解桌面应用开发的意向，我采用 QT Creator + PyQt5 进行系统框架的搭建，随着了解的进一步深入，我也在考虑更换为 QSS 来构建更加简洁、扁平化的 UI 界面。

2　软件需求与功能分析

根据作业的要求以及软件的用途进行分析，来确定所需要的功能以及具体的实现方法：

2.1　基本绘图需求

图形学作业对与具体算法的实现有着具体的要求，基本要求的图元有 4 种：线段（DDA 和 Bresenham）、多边形（DDA 和 Bresenham）、椭圆（中间圆生成算法）、曲线（Bezier 和 B-spline）。还要求了基本的图形操作：平移、旋转、放缩。以及画布的设置、画笔颜色的设置及文件的保存。这些都通过实验说明的 PPT 有了具体的说明。所以这些操作需要在命令行和图形界面均需要实现。

2.2　算法及系统性能要求

对图形的绘制应该能够实时输出，所以需要绘图算法以及显示系统能够高效的进行相关操作。

2.3　扩展绘图需求

除了实验要求的功能之外，对于一个基本的绘图软件还应该有如下功能：

- 扩展图形的绘制，如矩形、三角形、箭头等。
- 对已绘制图元的修改、编辑操作
- 类似画笔的功能，在界面上拖动鼠标可以获得连续曲线
- 画笔的设置拓展，在设置颜色之外，还应该能够设置画笔的宽度
- 单步操作的的保存与撤销功能，方便对图形的修改
- 3D 图形显示功能

2.4　图形交互界面需求

为了方便用户的使用，对图形的交互界面有相关需求，首先需要能够便捷得进行用户所需要的操作。其次，还需要即时展示用户所需要的信息，以及扁平清爽的 UI 交互界面。

这方面可以参考比较成熟的画图软件，比如 Windows 中的“画图”软件，具体而言：

- 点击图形按钮，之后在画布上进行点击、拖动等操作实现图形绘制；点击图形旋转、平移按钮，可以通过点击图形来选中所需要的图元进行直接的操作；可以通过鼠标的点击进行画笔颜色、画笔宽度的选择；可以通过鼠标进行操作的撤销等等。
- 实时展示用户所绘制的图形，在进行图形绘制的时候，可以实时显示图形随鼠标所选中位置移动的变化；实时显示当前的画笔颜色、宽度；实时显示当前鼠标所在画布的坐标。
- 干净清爽的 UI 操作界面，控件的布局、颜色设置符合美观；对于较大的图形，能够设置滚动栏显示全部图形。

2.5 文件管理需求

实现文件的基本的打开、保存操作。考虑到：弹出窗口式的文件打开、保存操作；应该使画布尺寸随打开的图片变化；在存在未保存图形的时候关闭窗口能提醒保存操作。

综合以上需求，我选择 PyQt5 以及 QT Creator 进行开发工作，通过 python 可以便于进行高层操作而无需过于关注底层内存管理系统的操作，能够提高开发的效率；使用 QT 可以方便得构建简介美观的图形界面，并且能方便的满足对颜色选择、文件操作的需求。本月意境实现了线段、多边形、椭圆的绘制，以及图像的打开、保存操作，还形成了基本的图元输出管理方法以及图形界面系统。

3 实验平台介绍

实验平台	Windows10 Python 3.8
开发工具	PyQt5.10.1
打包工具	PyInstaller 3.2.1
建议分辨率	1920x1080
缩放比例	150%

4 图元算法介绍

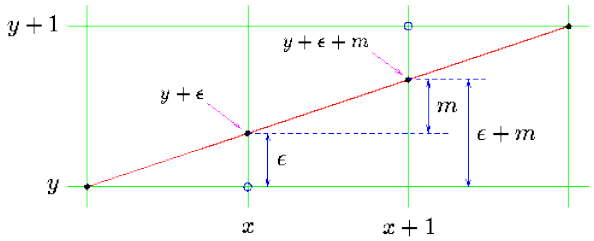
本月已近实现的线段的绘制操作、多边形的绘制操作以及椭圆的绘制操作，还有曲线的绘制以及图形的剪切、填充、平移等操作待完成。

下面现就绘制线段、多边形、椭圆以及填充的相关算法进行介绍：

4.1 Bresenham画线算法

计算机显示屏幕上是由一个个的像素组成的，而在数学中，一条直线是连续的不可分线条。所以，为了显示出线条的轨迹线条，我们还需要决定屏幕上哪些像素应该被选中并且记录下来最终输出。因此，我们需要一个能够根据线段方程高效选择所需要输出的像素的算法，这便是 Bresenham 算法的作用。

由于显示直线的象素点只能取整数值坐标，可以假设直线上第 i 个象素点坐标为(xi，yi)，它是直线上点(xi，yi)的最佳近似，并且 xi=xi(假设 m<1)，如下图所示。那么，直线上下一个象素点的可能位置是(xi+1，yi)或(xi+1，yi+1)。



由图中可以知道，在 x=xi+1 处，直线上点的 y 值是 y=m(xi+1)+b，该点离象素点(xi+1，yi)和象素点(xi+1，yi+1)的距离分别是 d1 和 d2：

$$d1=y-yi=m(xi+1)+b-yi \qquad (1)$$

$$d2=(yi+1)-y=(yi+1)-m(xi+1)-b \qquad (2)$$

这两个距离差是

$$d1-d2=2m(xi+1)-2yi+2b-1\ (2-10) \qquad (3)$$

我们来分析公式：

(1)当此值为正时，d1>d2，说明直线上理论点离(xi+1，yi+1)象素较近，下一个象素点应取(xi+1，yi+1)。

(2)当此值为负时，d1<d2，说明直线上理论点离(xi+1，yi)象素较近，则下一个象素点应取(xi+1，yi)。

(3)当此值为零时，说明直线上理论点离上、下两个象素点的距离相等，取哪个点都行，假设算法规定这种情况下取(xi+1，yi+1)作为下一个象素点。

因此只要利用(d1-d2)的符号就可以决定下一个象素点的选择。为此，我们进一步定义一个新的判别式：

$$pi=\triangle x\times(d1-d2)=2\triangle y\cdot xi-2\triangle x\cdot yi+c \qquad (4)$$

式(4)中的△x=(x2-x1)>0，因此 pi 与(d1-d2)有相同的符号；这里△y=y2-y1，m=△y/△x；c=2△y+△x(2b-1)。

下面对式(4)作进一步处理，以便得出误差判别递推公式并消除常数 c。

将式(4)中的下标 i 改写成 i+1，得到：

$$pi+1=2\triangle y\cdot xi+1-2\triangle x\cdot yi+1+c \qquad (5)$$

将式(5)减去(4)，并利用 xi+1=xi+1，可得：

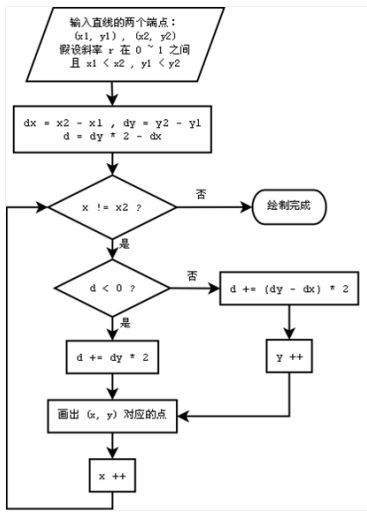
$$pi+1=pi+2\triangle y-2\triangle x\cdot(yi+1-yi) \qquad (6)$$

再假设直线的初始端点恰好是其象素点的坐标，即满足：

$$y1=mx1+b\ (2-14) \qquad (7)$$

由式(4)和式(7)得到 p1 的初始值：

$$p1=2\triangle y-\triangle x \qquad (8)$$



Bresenha 流程图

这样，我们可利用误差判别变量，第 i+1 步的判别变量 pi+1 仅与第 i 步的判别变量 pi、直线的两个端点坐标分量差△x 和△y 有关，运算中只含有整数相加和乘 2 运算，而乘 2 可利用算术左移一位来完成，因此这个算法速度快并易于硬件实现。[1]

4.2 Bresenham中心圆算法

Bresenham 画圆算法。Bresenham 算法的主要思想是：以坐标原点（0，0）为圆心的圆可以通过 0 度到 45°的弧计算得到，即 x 从 0 增加到半径，然后利用对称性计算余下的七段圆弧。当 x 从 0 增加到时，y 从 R 递减到。

设圆的半径为 R，则圆的方程为：

$$f(x，y)＝(x+1)2+y2-R2＝0 \qquad (1)$$

假设当前列（x=xi 列）中最接近圆弧的像素已经取为 P(xi，yi)，根据第二卦限 1/8 圆的走向，下一列（x=xi+1 列）中最接近圆弧的像素只能在 P 的正右方点 H(xi+1，yi)或右下方点 L(xi+1，yi-1)中选择，如图 1 所示。Bresenham 画圆算法采用点 T(x，y)到圆心的距离平方与半径平方之差 D(T)作为选择标准，即

$$D(T)=(x+1)2+y2-R2 \qquad (2)$$

通过比较 H、L 两点各自对实圆弧上点的距离大小，即根据误差大小来选取，具有最小误差的点为绘制点。根据公式(2)得：对 H(xi+1，yi)点有：D(H)=(xi+1)2+yi2-R2；对 L(xi+1，yi-1)点有：D(L)=(xi+1)2+(yi-1)2-R2；根据 Bresenham 画圆算法，则选择的标准是：

如果|D(H)|<|D(L)|，那么下一点选取 H(xi+1，yi);

如果|D(H)|>|D(L)|，那么下一点选取 L(xi+1，yi-1);

如果|D(H)|=|D(L)|，那么下一点可以取 L(xi+1，yi-1)，也可以选取 H(xi+1，yi)，约定选取 H(xi+1，yi)。

综合上述情况，得：

当|D(H)|>|D(L)|时，选取 L 点(xi+1，yi-1)为绘制点坐标；

当|D(H)|<|D(L)|时，选取 H 点(xi+1，yi)为绘制点坐标。

然后将选取的点坐标作为当前坐标，重复上述过程直至 xi=或者 yi=为止，(xi，yi)的初始值为（0，R）。以上便是 Bresenham 算法的主要思想，但是上述算法是在一个假设下：以坐标原点（0，0）为圆心。该假设实际上只是为了方便算法的研究。以上只是对这个算法所做的简答表示，具体的实现还需要做很大的改进。另外，如果完全按照 Bresenham 画圆算法，那么就会涉及到浮点运算，会使得计算量很大。

因此有改进算法(以 Python 代码表示)，以避免大量的浮点数运算：

```
# init temp point
tempX = 0
tempY = shortR
# paint first half
p1 = SquareShortR - SquareLongR * shortR + SquareLongR / 4.0
while tempX * SquareShortR < tempY * SquareLongR:
    if p1 < 1:
        p1 += 2 * SquareShortR * tempX + 3 * SquareShortR
        tempX += 1
    else:
        p1 += 2 * SquareShortR * tempX + 3 * SquareShortR - 2 * SquareLongR * tempY + 2 *
          SquareLongR
        tempX += 1
        tempY -= 1
```

```
            # addPonit() can add the four point at the symmetry place
            self.addPoint(centerX, centerY, tempX, tempY, horizontalFocus)
# paint last half
p2 = SquareShortR * (tempX + 0.5) * (tempX + 0.5) + SquareLongR * (tempY - 1) * (tempY - 1) \
      - SquareShortR * SquareLongR
while tempY > 0:
    if p2 > 0:
        p2 -= 2 * SquareLongR * tempY + 3 * SquareLongR
        tempY -= 1
    else:
        p2 += 2 * SquareShortR * tempX + 3 * SquareShortR - 2 * SquareLongR * tempY +
                2 * SquareLongR
            tempX += 1
            tempY -= 1
        self.addPoint(centerX, centerY, tempX, tempY, horizontalFocus)
```

4.3 泛滥填充算法

主要方法为用水平扫描线从上到下扫描由多条首尾相连的线段构成的多边形，每根扫描线与多边形的某些边产生一系列交点。将这些交点按照坐标排序，将排序后的点两两成对，作为线段的两个端点，以所填的颜色画水平直线。

具体方法：

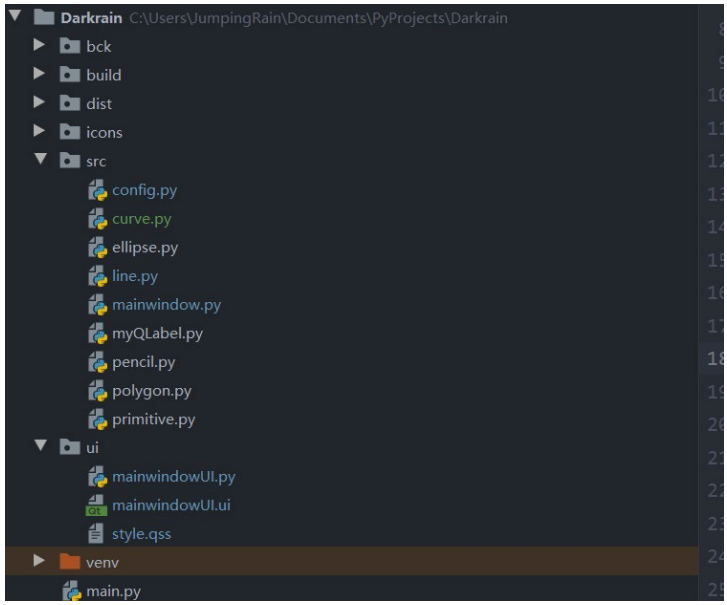
- 初始化扫描线，然后更新活化边表，第一次更新的为最小 y 值为扫描线值的两条边；
- 只要活化边表不为空时• 执行下面的循环：
  - 以活化边表中相邻两个边的 x 为填充范围的横坐标• 以扫描线值为 y 坐标• 填充这一个区域（其实就是画这根线）
  - 填充完后• 扫描线值加 1
  - 根据扫描线值移除移已经处理完的活化边（扫描线值大于该边最大 y 值），将活化边表中相邻两个边的 x (之前用于确定填充横坐标区间的值）更新，也即加上各自斜率的倒数
  - 更新一下活化边表，根据活化边表中每条边的 x 来进行排序
- 循环直至边表为空（所有边都处理完），为了防止对多边形编辑后填充不正确• 每次编辑完后• 都要重新校正边表的值。

5 软件系统介绍

本实验的 GUI 框架以及逻辑主要由 PyQt5 完成，并且用 QT Creator 辅助 UI 设计。

5.1 实验代码框架

实验主要代码框架如下：mainwindow 为主窗口类；mainwindowUI 为由 QT Creator 创建的 UI 界面；myQlabel 为基于 QLabel 重构的类; primitive 为图元基类；line、ellipse、polygon 为派生子类。



在 mainwindow.py 中，首先导入 mainwindowUI 中的 UI 内容，而后创建基于 MyQLabel 的 picLabel 实例，初始化按钮、菜单动作。而后进入主执行活动。下面介绍几个重要的操作：

设置 piclabel:

```
# set myQLabel
self.picLabel = myQLabel.MyLabel(self.scrollAreaWidgetContents, self, QtCore.QSize(500, 400))
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.picLabel.sizePolicy().hasHeightForWidth())
self.picLabel.setSizePolicy(sizePolicy)
self.picLabel.setCursor(QtGui.QCursor(QtCore.Qt.CrossCursor))
self.picLabel.setAutoFillBackground(True)
self.picLabel.setFrameShape(QtWidgets.QFrame.Panel)
self.picLabel.setFrameShadow(QtWidgets.QFrame.Raised)
self.picLabel.setText("")
self.picLabel.setObjectName("picLabel")
self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.picLabel)
self.picLabel.setMouseTracking(True)
```

初始化 MyQLabel:

```
class MyLabel(QtWidgets.QLabel):
    def __init__(self, parent, parentWindow, qsize):
        # init parent class QLabel
        super().__init__(parent)

        # get a pointer to mainWindow
        self.parentWindow = parentWindow

        # set size of PicLabel
        self.setMinimumSize(qsize)

        # init PixMap
        self.mainPixMap = QPixmap(qsize)
        self.mainPixMap.fill(QColor("white"))
        self.setPixmap(self.mainPixMap)
        self.tempPixMap = self.mainPixMap.copy()
        self.setPixmap(self.tempPixMap)

        # init painter
        self.mainPen = QPen()
        self.mainPen.setColor(QColor("black"))
        self.mainPen.setWidth(2)
        self.painter = QPainter()
        self.painter.begin(self.mainPixMap)
        self.painter.setPen(self.mainPen)

        # init mode
        self.mode = LineMode
```

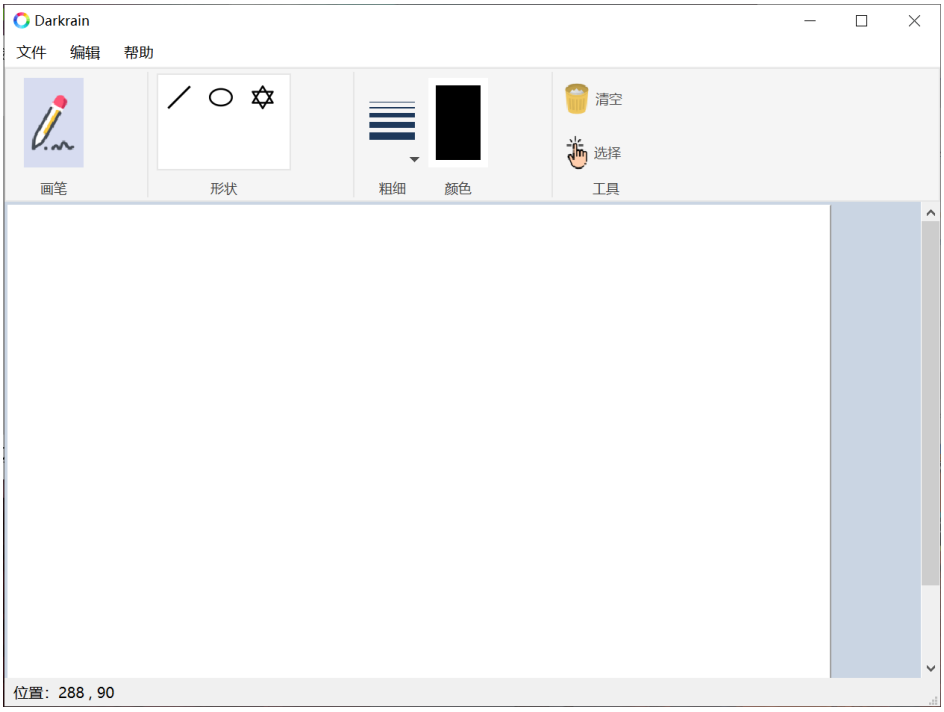
刷新显示操作：

此处为了加速显示，会在每次创建新图元之前保存 tempPixMap，在刷新时先将 tempPixMap 赋值给 MainPixMap，而后再由 QPainter 将新图元的点显示到屏幕上，大大加速了屏幕上存在多个图元时的显示速度。而且显示速度始终能保持稳定，不会因为显示过多而变慢。

```
def reShowPic(self) -> None:
    LOG("reshow start")
    self.painter.drawPoint(20,20)
    self.painter.drawPixmap(0, 0, self.mainPixMap.width(),
                            self.mainPixMap.height(), self.tempPixMap)
    print("middle")
    if 0 <= self.nowPrim < len(self.primList):
        for point in self.primList[self.nowPrim].pointList:
            self.painter.drawPoint(point)
    self.setPixmap(self.mainPixMap)
    LOG("reshow end")
```

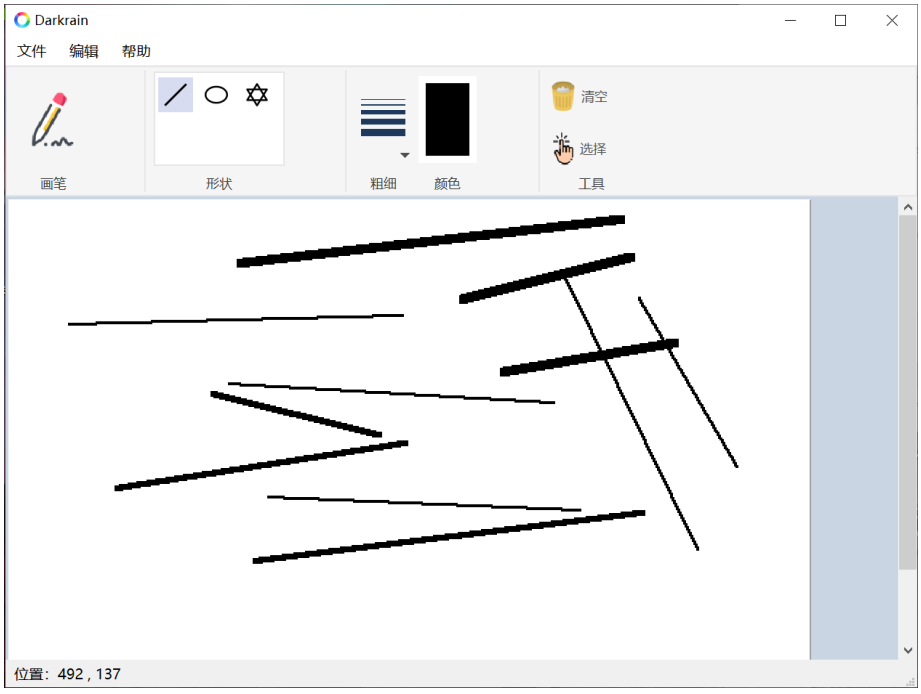
5.2 应用UI及简单功能演示

初始界面如下：

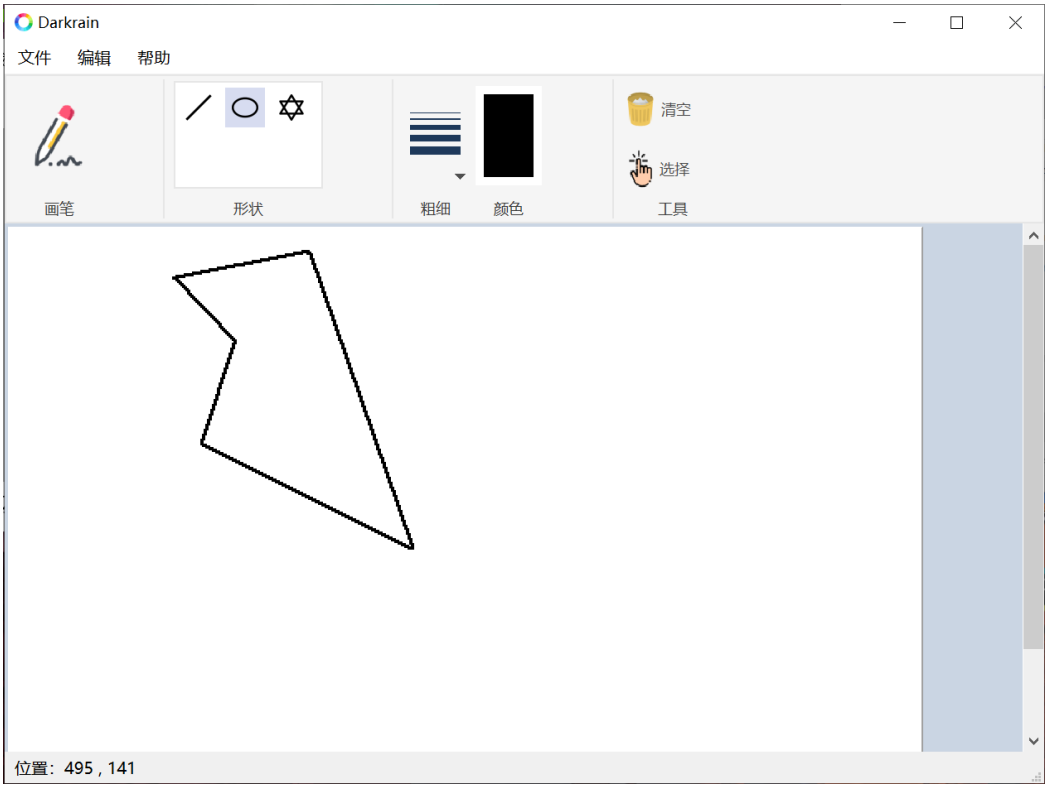


简单图元操作：

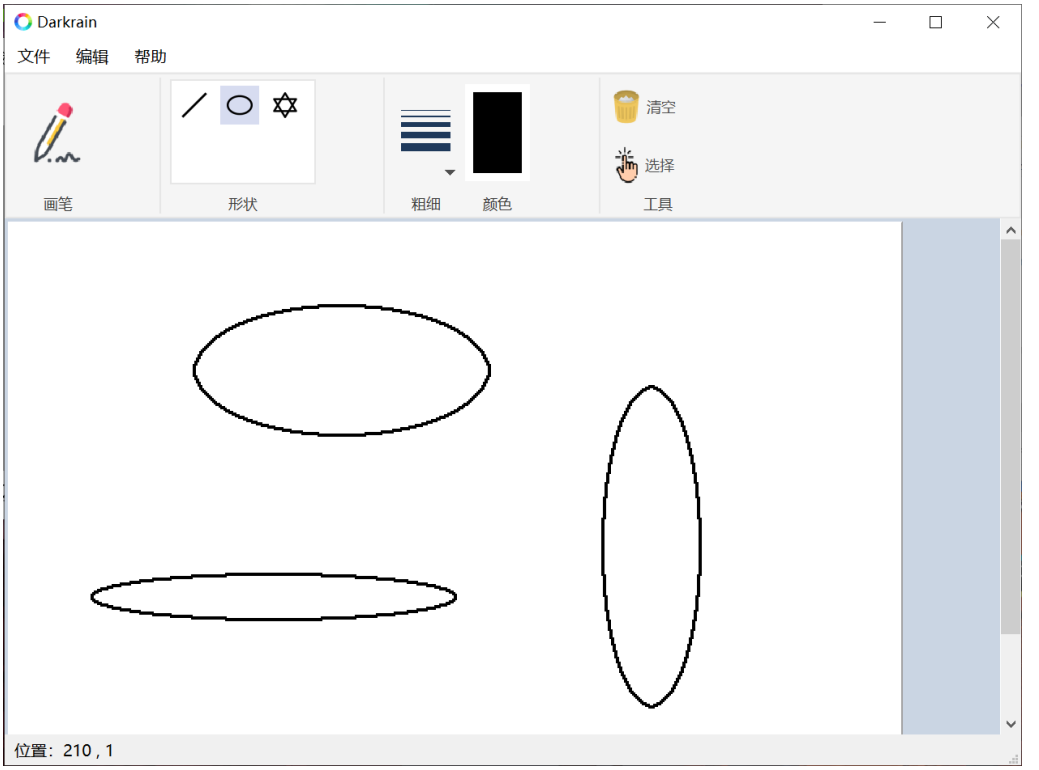
直线：



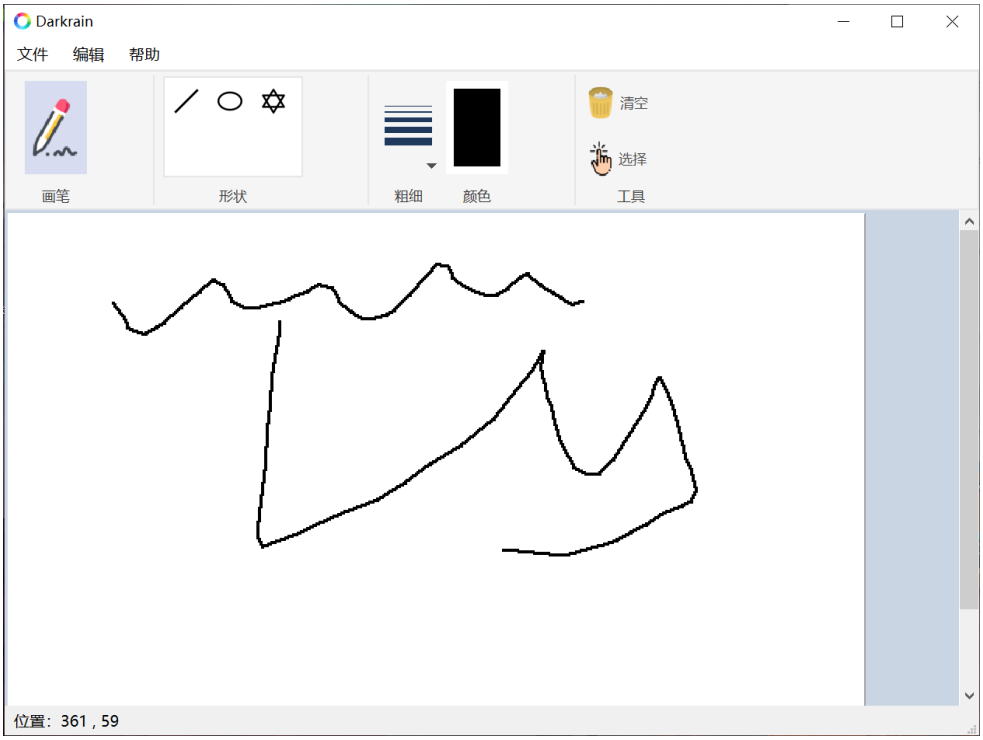
多边形：


Darkrain
文件 编辑 帮助
画笔
形状
粗细
颜色
清空
选择
工具
位置：495，141

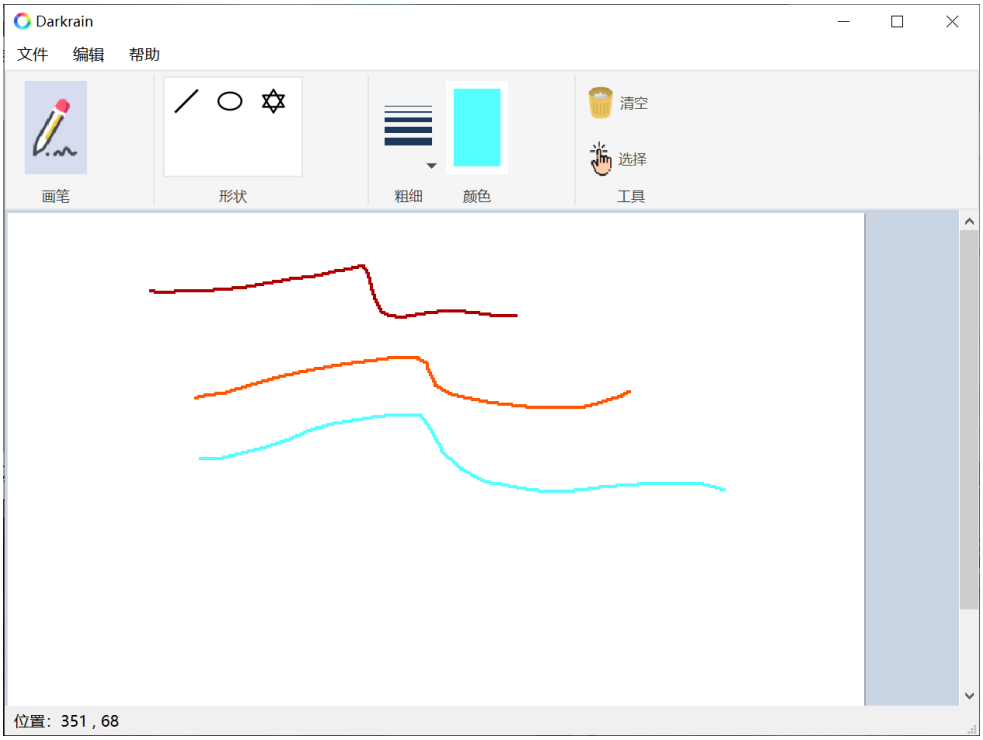
椭圆：


Darkrain
文件 编辑 帮助
画笔
形状
粗细
颜色
清空
选择
工具
位置：210，1

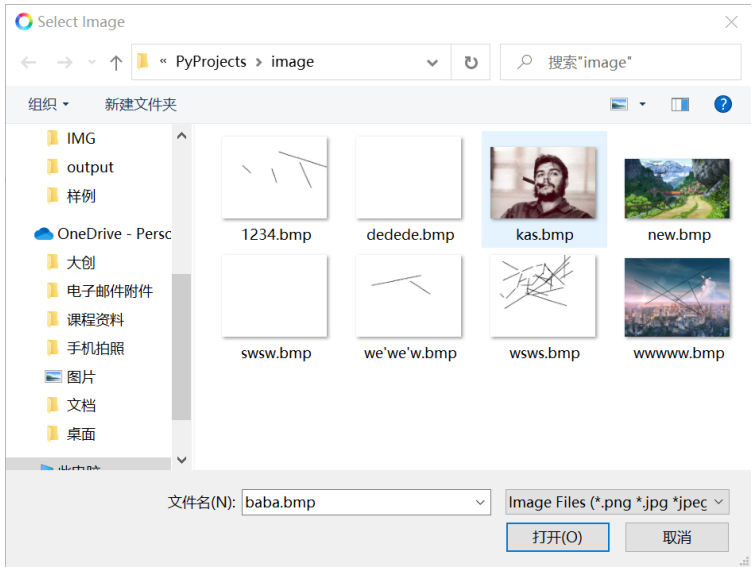
画笔：

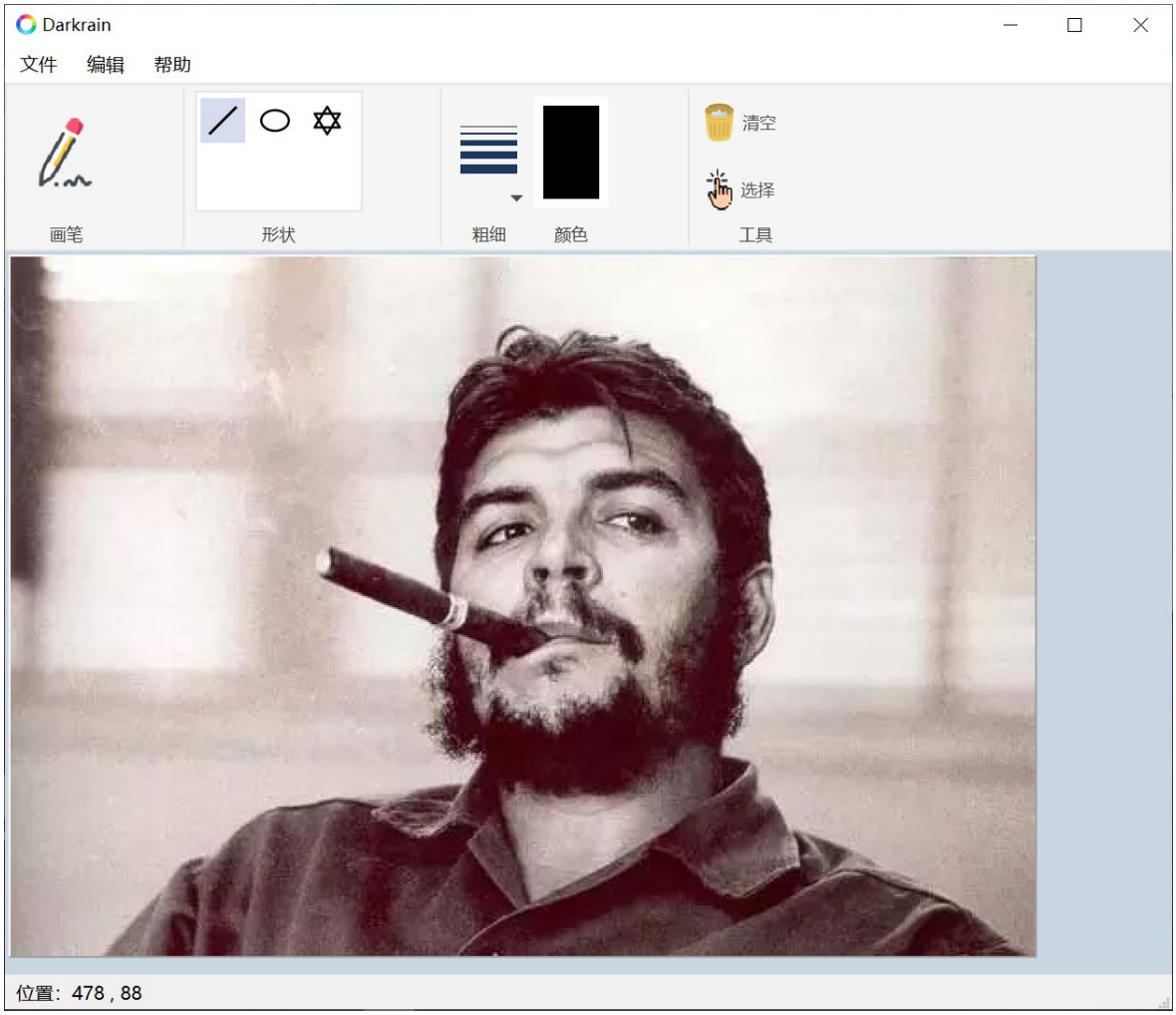

Darkrain
文件
编辑
帮助
画笔
形状
粗细
颜色
清空
选择
工具
位置：361，59

不同颜色：

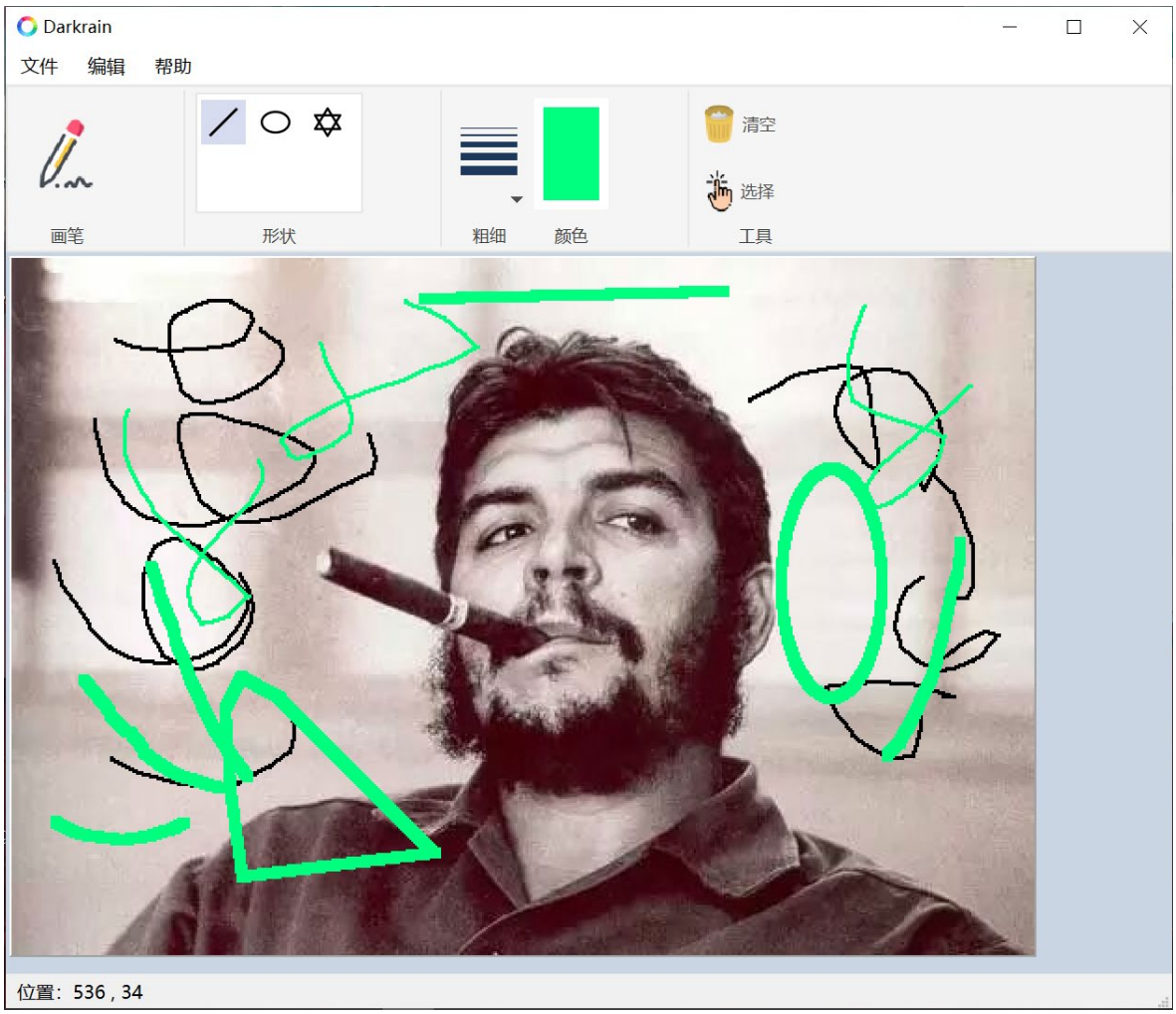

Darkrain
文件
编辑
帮助
画笔
形状
粗细
颜色
清空
选择
工具
位置：351，68

打开一幅图片：（打开后画布尺寸随之改变）





画线、多边形、椭圆绘制操作演示效果（显示图片）：



注意在鼠标移动过颜色图标时会有特殊效果：

普通：

特殊：

6 结束语

通过学习课本上所讲的线条、椭圆画线算法，以及平移、旋转、填充的相关算法，并且利用 Qt Creator+ PyQt5 进行软件系统 GUI 的编写开发。通过手动实现课本中的相关画图算法，以及构建完整的图形界面系统，来实现作业所需要的画图、打开、保存以及图形显示的相关功能。

下一步，我会首先补充完善所需要的相应功能，再考虑利用 QSS 进一步完善 UI 界面的显示，并且逐步加入所设计的增强功能。

References:

[1] Bresenham 算法原理: https://blog.csdn.net/yzh1994414/article/details/82860187

[2] 课程资料