《计算机图形学》10月进展报告

欧阳鸿荣 161220096

（南京大学 计算机科学与技术系 南京 210093）

1.综述

系统名	语言和框架	IDE	编译器
YoungPaint	C++和Qt 5.11.2	Qt Creator	MinGW 5.3.0

截止11月底，我项目目前能完成的功能是：

一、基本功能：

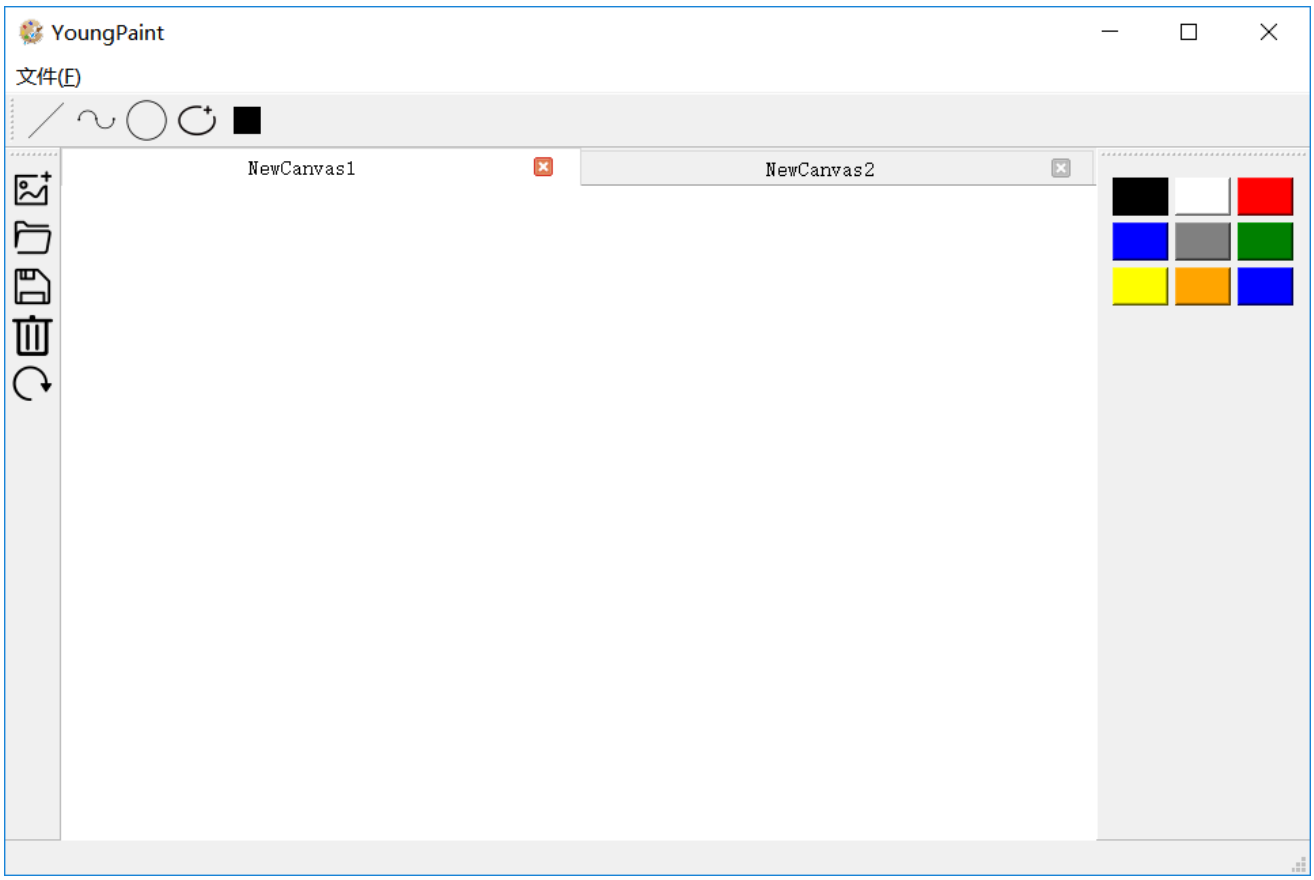
1. **二维图形的输入功能：**
    - 直线、圆，椭圆，多边形的输入实现
        - 类画图软件，用鼠标交互
    - 填充区域的输入
        - 实现了类似油漆桶的功能
        - 鼠标点击区域，洪泛填充与区域颜色相同的区域
2. **二维图形的编辑功能：**
    - 直线、圆，椭圆，多边形的编辑
        - 直线能编辑起点、终点
        - 圆能编辑半径
        - 椭圆能编辑长轴a和短轴b的长度
        - 多边形能编辑任意顶点
        - 鼠标点击拖动交互编辑
    - 二维图形的裁剪功能：
        - 直线的裁剪
            - 使用梁友栋算法
        - 裁剪窗口可用鼠标点击拖动输入
        - 裁剪后的图形仍然可以编辑
3. **二维图形的变换功能**
    - 直线、圆、椭圆、多边形的平移
    - 直线、圆、椭圆、多边形的旋转
        - 任意角度旋转
        - 直线、圆的旋转实现了精度控制
            - 旋转次数不多的情况下，长度/半径误差在1以内
    - 直线、圆、椭圆、多边形的缩放
    - 对变换后的图形仍然可以编辑
4. **二维图形的存储功能**

- 讲绘制出来的图形保存为图像

二、扩展功能

1. 画布的创建
2. 颜色的选择
3. 增加了画笔和笔刷的功能
4. 清屏和撤销的功能

将项目打包成了一个.exe文件，放在161220096_系统工程\Demo\YoungPaint里供助教测试，虽然在很多人电脑里测试都可以运行，但是程序依赖是否都有这个还不得而知，权当给助教一个参考。



2.算法介绍

(1).直线算法 —— DDA直线算法

(a)基本原理

数值差分分析DDA(digital differential analyzer)：直接利用△x或△y的线段扫描转换算法，利用光栅特性(屏幕单位网格表示像素列阵)，使用x或y方向单位增量(△x或△y=±1)来离散取样，并逐步计算沿线路径各像素位置。在一个坐标轴上以单位间隔对线段离散取样，确定另一个坐标轴上最靠近线段路径的对应整数值。

(b)理论绘制过程

1.具有正斜率

1.1 从左到右生成线段。

若斜率 $m \leq 1$，在x方向以单位间隔($\triangle x = 1$)取样，以增量形式顺序计算每个y值：

$$y_{k+1} = y_k + m \quad (1)$$

下标k取整数值从k=1开始递增。m为0与1间的任意实数，计算出的坐标值必须取整。

1.2 从右端点到左端点生成线段

取$\triangle x = -1$

$$y_{k+1} = y_k - m \quad (2)$$

1.3 若斜率m>1，从左到右生成线段。

在y方向以单位间隔$\triangle y = 1$取样，顺序计算每个x值：

$$x_{k+1} = x_k + \frac{1}{m} \quad (3)$$

下标k取整数值从1开始递增，直至最后端点。计算出的坐标值必须取整。

1.4 若从右到左生成线段

取$\triangle y = -1$

$$x_{k+1} = x_k - \frac{1}{m} \quad (4)$$

2.具有负斜率

2.1 假如斜率的绝对值小于1时，

起始端点在左侧，设置$\triangle x = 1$并用方程(1)计算y； 起始端点在右侧，设置$\triangle x = -1$并用方程(2)得到y

2.2 假如斜率的绝对值大于1时，

起始端点在左侧，用$\triangle y = 1$和方程(3)， 当起始端点在右侧，$\triangle y = -1$和方程(4)。

(c)算法的C++实现

```
void LineController::MyDrawLineDDA(QPainter *painter, QPoint &start, QPoint &end)
{
    //首先先在这里实现我的直线算法
    qDebug()<<"MyDrawLine DDA"<<endl;

    int x1 = start.x();
    int y1 = start.y();
    int x2 = end.x();
    int y2 = end.y();

    double dx=x2-x1;
    double dy=y2-y1;
    double e=(fabs(dx)>fabs(dy))?fabs(dx):fabs(dy);
    double x=x1;
    double y=y1;

    dx/=e;
    dy/=e;

    for(int i=1;i<=e;i++){

        QPoint temPt((int)(x+0.5), (int)(y+0.5));
        painter->drawPoint(temPt);

        x+=dx;
        y+=dy;
    }
}
```

(2).圆算法 —— 中点圆算法

(a)基本原理

避免平方根运算，直接采用像素与圆距离的平方作为判决依据。通过检验两候选像素中点与圆周边界的相对位置关系(圆周边界的内或外)来选择像素。

(b)优点

适应性强：易应用于其它圆锥曲线。误差可控：对于整数圆半径，生成与Bresenham算法相同的像素位置。且所确定像素位置误差限制在半个像素以内。

(c)理论绘制过程

根据圆的对称性，只绘制了八分之一圆，其余部分通过对称性即可得到坐标。使用经过改良的中点圆算法，使用递推，减少了计算量，并且避免了浮点运算

算法过程

1. 输入圆半径r和圆心(x_c, y_c)，并得到圆心在原点的圆周上的第一点为 $(x_0, y_0) = (0, r)$
2. 计算圆周点$(0, r)$的初始决策参数值为：$p_0 = \frac{5}{4} - r$
3. 从k=0开始每个取样位置x_k位置处完成下列检测
   - 若p_k < 0，选择像素位置：(x_{k+1}, y_k)；且$p_{k+1} = p_k + 2x_{k+1} + 1$
   - 若$p_k > 0$，选择像素位置：$(x_{k+1}, y_k - 1)$；且$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$
   - 其中有：$2x_{k+1} = 2x_k + 2$, 且 $2y_{k+1} = 2y_k - 2$
4. 确定在其它七个八分圆中的对称点
5. 将计算出的像素位置(x, y)移动到中心在(x_c, y_c)的圆路径上，即：对像素位置进行平移

   $x = x + x_c, y = y + y_c$

   重复步骤3到5，直到$x > y$

(d)算法的C++实现

```
//中点圆算法
void CycleController::MyDrawCycle(QPainter *painter, QPoint &start, QPoint &end)
{
    //首先先在这里实现我的画圆算法
    qDebug()<<"MyDrawCycle "<<endl;
    int x0 = start.x();
    int y0 = start.y();
    double R = this->getLength(start,end);
    int x,y,p;
    x=0;
    y=R;
    p=3-2*R;
    for(;x<=y;x++)
    {
        this->drawEighthCycle(painter,x0,y0,x,y);
        if(p>=0){
           p+=4*(x-y)+10;
           y--;
        }else{
            p+=4*x+6;
        }
    }
}
//由于中点圆算法只对八分之一圆进行绘制，因此用该函数将八分之一圆投射成完整的圆
void CycleController::drawEighthCycle(QPainter *painter, int x0, int y0, int x, int y)
{
    QPoint temPt1(x0+x,y0+y);QPoint temPt2(x0+y,y0+x);
    QPoint temPt3(x0+x,y0-y);QPoint temPt4(x0+y,y0-x);
    QPoint temPt5(x0-x,y0-y);QPoint temPt6(x0-y,y0-x);
    QPoint temPt7(x0-x,y0+y);QPoint temPt8(x0-y,y0+x);

    painter->drawPoint(temPt1);painter->drawPoint(temPt2);
    painter->drawPoint(temPt3);painter->drawPoint(temPt4);
    painter->drawPoint(temPt5);painter->drawPoint(temPt6);
    painter->drawPoint(temPt7);painter->drawPoint(temPt8);
}
```

(3).椭圆算法 —— 中点椭圆算法

(a)基本原理

考虑椭圆沿长轴和短轴不同而修改圆生成算法来实现椭圆生成。

给定长短轴参数r_x, r_y(假设$r_x < r_y$)和椭圆中心位置(x_c, y_c)，利用平移：先确定中心在原点的标准位置的椭圆点(x, y)；然后将点变换为圆心在(x_c, y_c)的点。再利用对称性：生成第一象限内的椭圆弧，再利用对称性求出其它三个象限的对应点。

(b)理论绘制过程

根据圆的对称性，只绘制了八分之一圆，其余部分通过对称性即可得到坐标。使用经过改良的中点圆算法，使用递推，减少了计算量，并且避免了浮点运算

算法过程

1. 输入r_x、r_y和中心(x_c, y_c)，得到中心在原点的椭圆上的第一个点：$(x_0, y_0) = (0, r_y)$
2. 计算区域1中决策参数的初值为: $p1_0 = r_y^2 - r_x^2 r_y + r_x^2/4$
3. 在区域1中每个x_k位置处，从k=0开始，完成下列测试
   - 若p_{lk} < 0，椭圆的下一个离散点为(x_{k+1}, y_k)， 且：$pl_{k+1} = pl_k + 2r_y^2 x_k + 1 + r_y^2$。
   - 若$p_{lk} > 0$，椭圆的下一个离散点为$(x_{k+1}, y_k - 1)$， 且：$p1k + 1 = pl_k + 2r_y^2 x_k - 2r_x^2 y_{k+1} + r_y^2$
   - 其中有：$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2$; $2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$
   - 循环到：$2r_y^x \geq 2r_x^2 y$
4. 使用区域1中最后点(x_0, y_0)计算区域2参数初值为$p2_0 = r_y^2(x_0 + 1/2)^2 + r_x^2(y_0 - 1) - r_x^2 r_y^2$
5. 在区域2的每个y_k位置处，从k=0开始，完成下列检测
   - •假如$p2_k > 0$，椭圆下一点选为$(x_k, y_k - 1)$且：$p2_{k+1} = p2_k + 3r_x^2 - 2r_x^2 y$,
   - •否则，沿椭圆的下一个点为$(x_{k+1}, y_k - 1)$, 且：$p2_{k+1} = p2_k + 2r_y^2 x - 2r_x^y + 2r_y^2 + 3r_x^2$
   - 使用与区域1中相同的x和y增量计算
   - 循环直至$(r_x, 0)$
6. 确定其它三个像限中对称的点。
7. 将每个计算出的像素位置(x, y)平移到中心在(x_c, y_c)的椭圆轨迹上，并且按照坐标之绘点
   $x = x + x_c, y = y + y_c$

(c)算法的C++实现

```
//由于中点椭圆算法只对四分之一椭圆进行绘制，因此用该函数将四分之一椭圆投射成完整的椭圆
void EllipseController::drawQuarterEllipse(QPainter *painter, int x0, int y0, int x,
int y)
{
    QPoint temPt1(x0+x,y0+y);
    QPoint temPt2(x0+x,y0-y);
    QPoint temPt3(x0-x,y0+y);
    QPoint temPt4(x0-x,y0-y);

    painter->drawPoint(temPt1);
    painter->drawPoint(temPt2);
    painter->drawPoint(temPt3);
    painter->drawPoint(temPt4);
}

//中点椭圆算法
void EllipseController::MyDrawEllipse(QPainter *painter, QPoint &start, QPoint &end)
{
    //首先先在这里实现我的椭圆算法
    qDebug()<<"MyDrawEllipse "<<endl;
    int x0 = start.x(); //椭圆中心
    int y0 = start.y();
    int rx = abs(end.x()-x0); //椭圆长短轴
    int ry = abs(end.y()-y0);

    double rx_2 = rx*rx;
    double ry_2 = ry*ry;

    double p1 = ry_2 - rx_2*ry + rx_2/4; //区域1中决策参数
    int x = 0;
    int y = ry;
    drawQuarterEllipse(painter,x0, y0, x, y);//第一个点

    //区域一 切线斜率k<=1
    while (ry_2*x <= rx_2*y){

        if (p1 < 0){
            p1 += 2*ry_2*x +3*ry_2;
        }else{
            p1 += 2*ry_2*x - 2*rx_2*y + 2*rx_2+ 3*ry_2;
            y--;
        }
        x++;
        drawQuarterEllipse(painter,x0, y0, x, y);
    }
    //区域二 切线斜率k > 1
    //使用区域1中最后点(x0,y0)来计算区域2中参数初值
    p1 = ry_2*(x+1/2)*(x+1/2)+rx_2*(y-1)*(y-1)-rx_2*ry_2;

    while (y > 0){
```

```
        if (pl < 0){
            pl += 2*ry_2*x - 2*rx_2*y + 2*ry_2 + 3*rx_2;
            x++;
        }
        else{
            pl += 3*rx_2 - 2*rx_2*y;
        }
        y--;
        drawQuarterEllipse(painter,x0, y0, x, y);
    }
}
```

3.系统介绍

3.1 实验环境

操作系统（运行平台）	**Windows 10**
开发语言	C++
开发环境	Qt Creator
编译环境	MinGW 5.3.0

3.2 系统组织

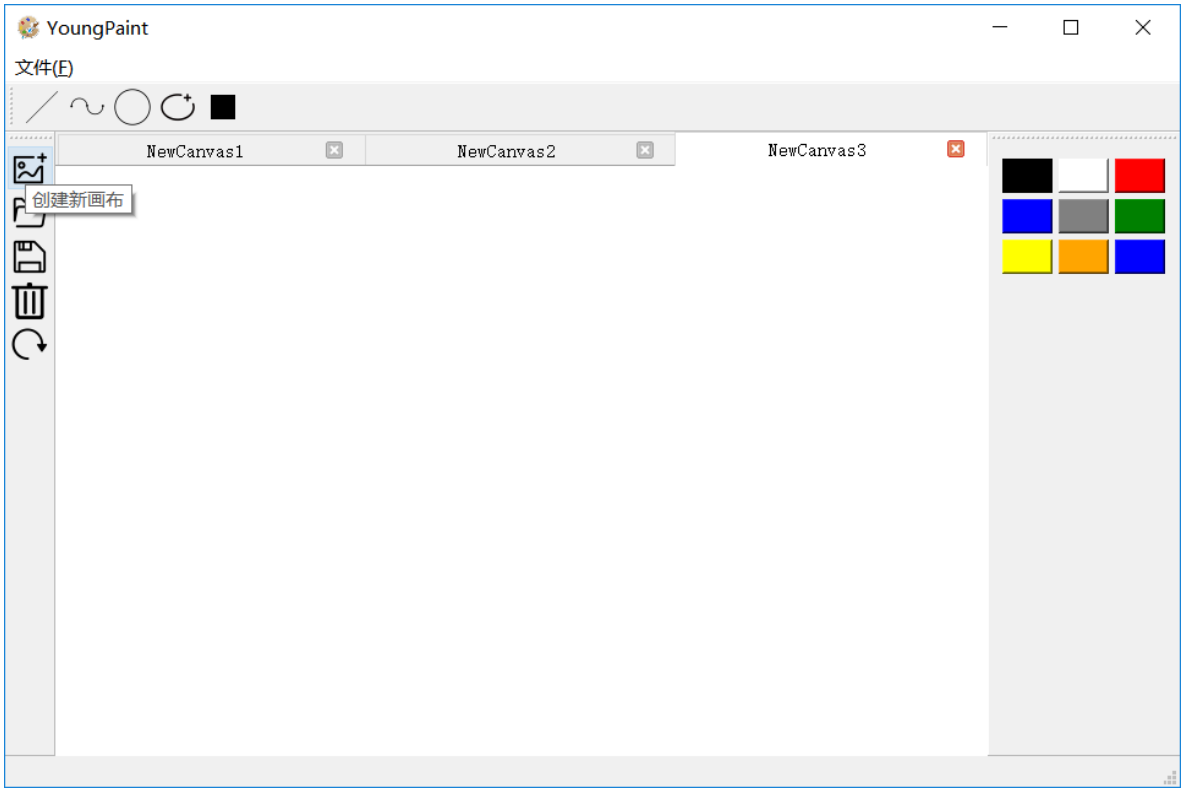
本系统使用Qt提供图形界面，通过Qt集成的QOpenGLWidget类来提供画布，实验主要分为以下类

类名	**继承于**	**类功能**
MainWindow	QMainWindow	提供整个系统的框架，并提供基础功能，如选择绘图类型（目前是直线，圆，椭圆），撤销，清屏，新建图像，保存图像等选项，是交互的窗口。
Canvas_GL	QOpenGLWidget	每个Canvas_GL类都是一个绘图的画布，用户在上面绘图
ColorPanel	QWidget	尚未完全实现，用于提供更为便捷的功能选择
FigureController		图形控制的接口，用于定义绘图行为（目前只有绘制，后期预计在此接口上实现更多功能）
LineController	FigureController	控制直线的绘制，实现了DDA直线算法
CycleController	FigureController	控制圆的绘制，实现了中点圆和Bresenham画圆法
EllipseController	FigureController	控制椭圆的绘制，实现了重点椭圆绘制算法
Figure		图形类的基类，用于记录画过的图形，便于后期操作
Line	Figure	直线，记录了直线的起点和终点
Cycle	Figure	圆，记录了圆心和半径
Ellipse	Figure	椭圆，记录了代表长短轴的矩形

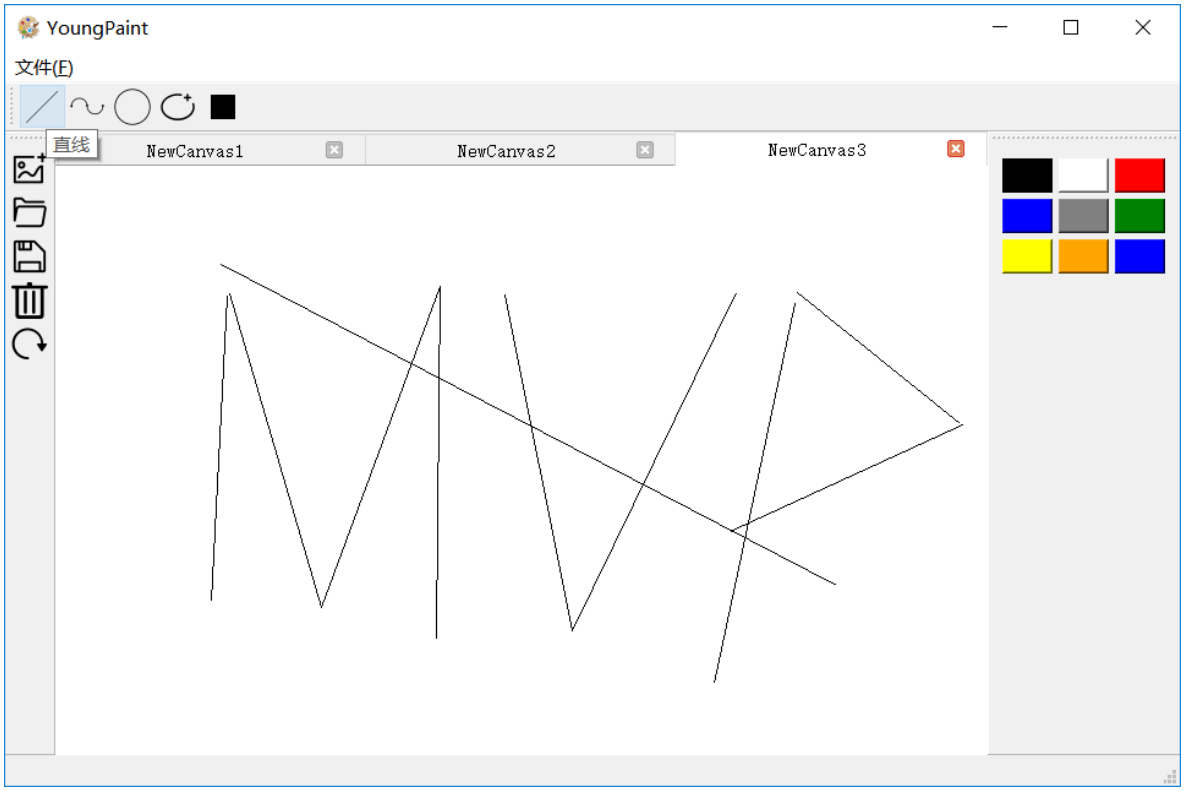
3.3 功能演示

(1) 打开画面

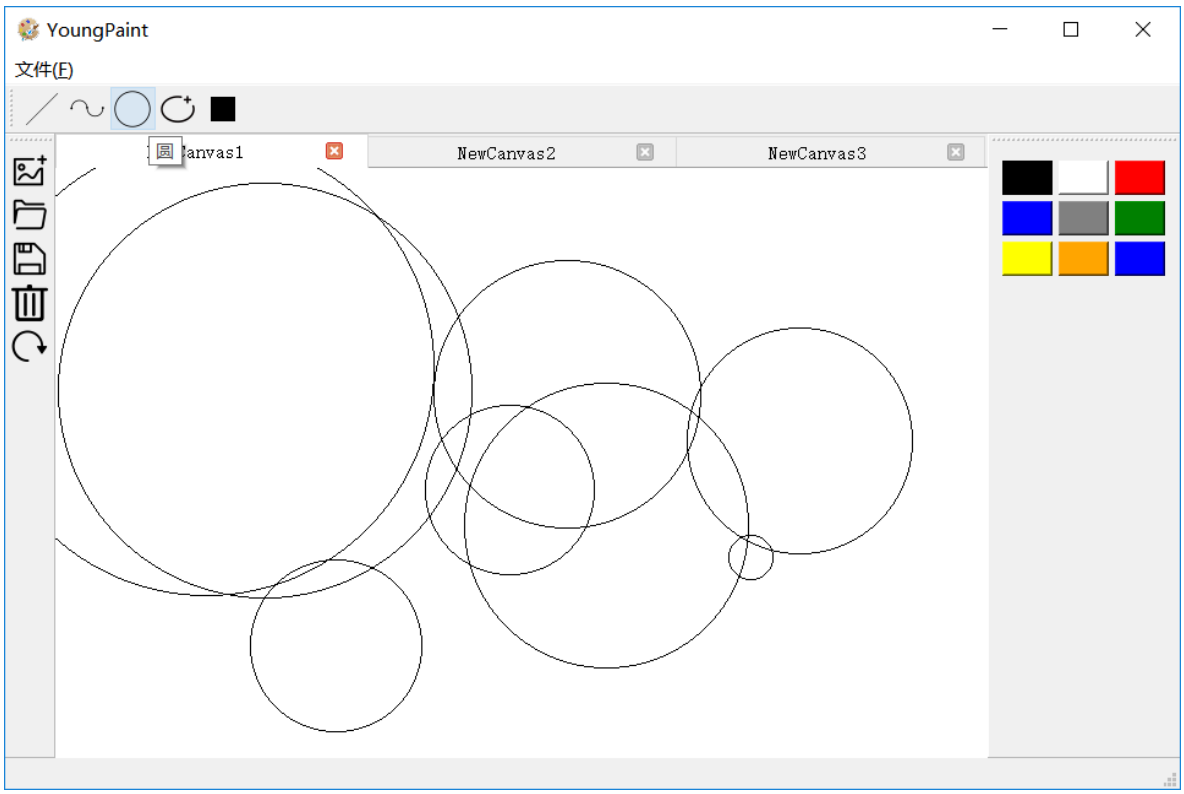
(2) 创建新画布：点击左侧工具栏的创建新画布，则可以创建画布，并且可以创建多个画布



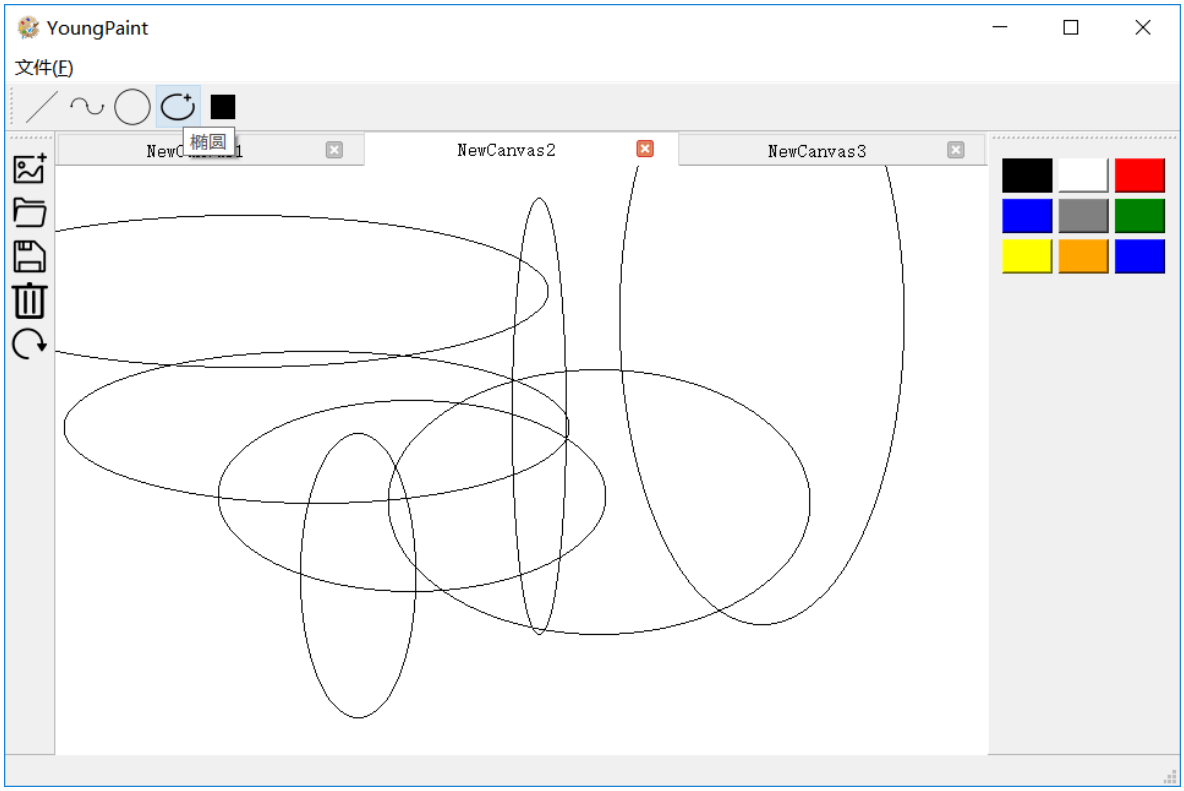
(3) 画直线：点击上方工具栏的直线，则可绘制直线，鼠标点击确定起点，释放确定终点



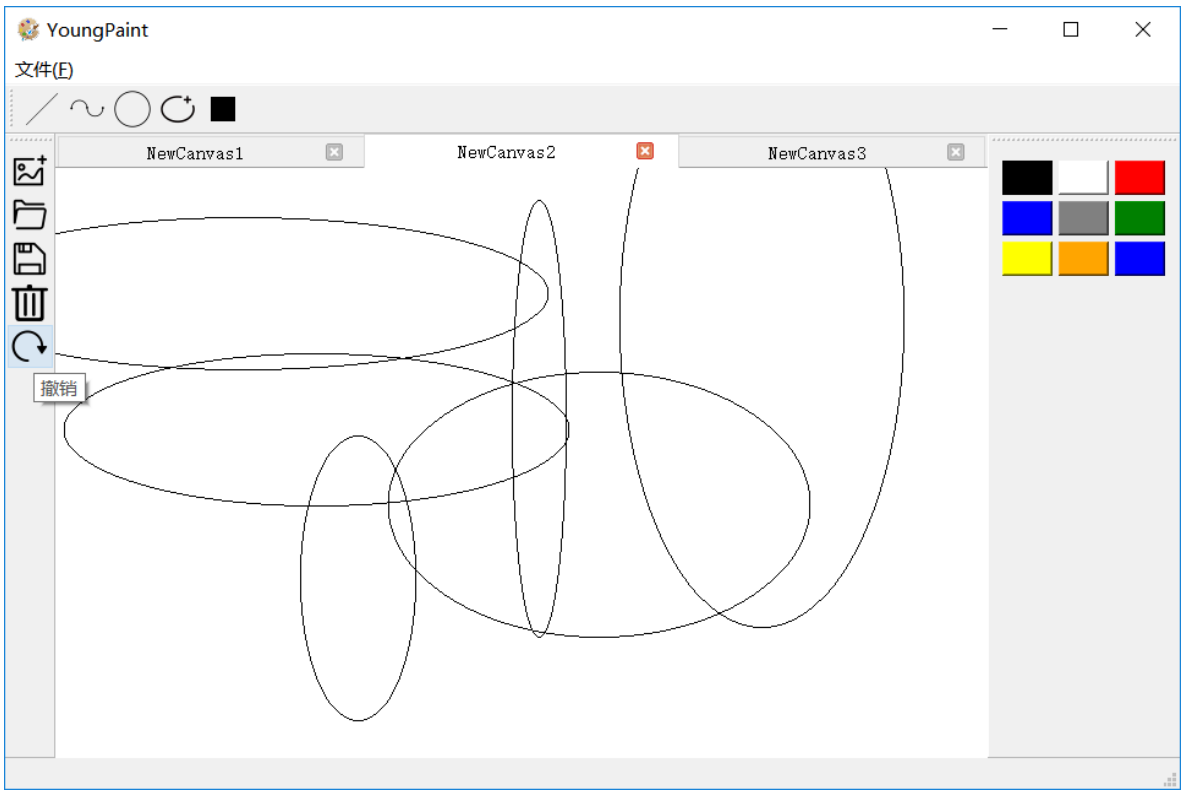
(4) 画圆：点击上方工具栏的圆，则可绘制圆。鼠标点击确定圆心，释放确定半径



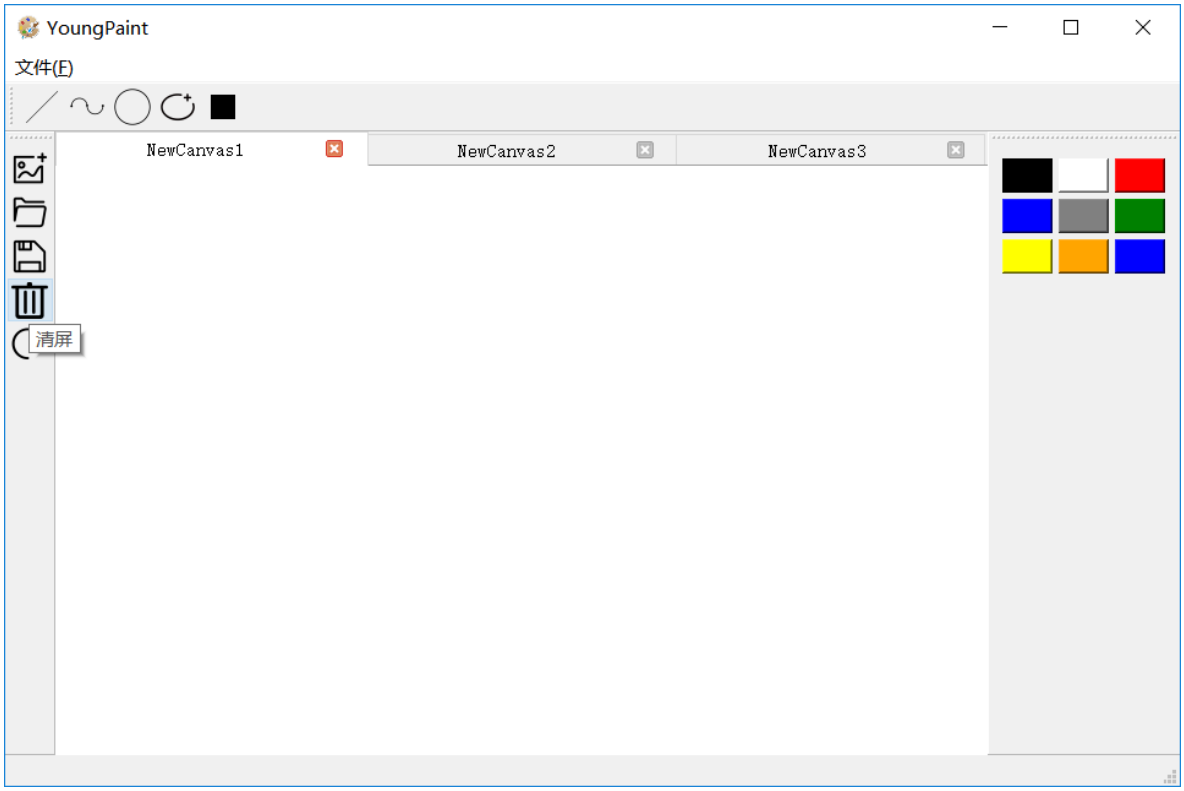
(5) 画椭圆：点击上方工具栏的椭圆，则可绘制椭圆。鼠标点击确定中心，释放确定长轴和短轴



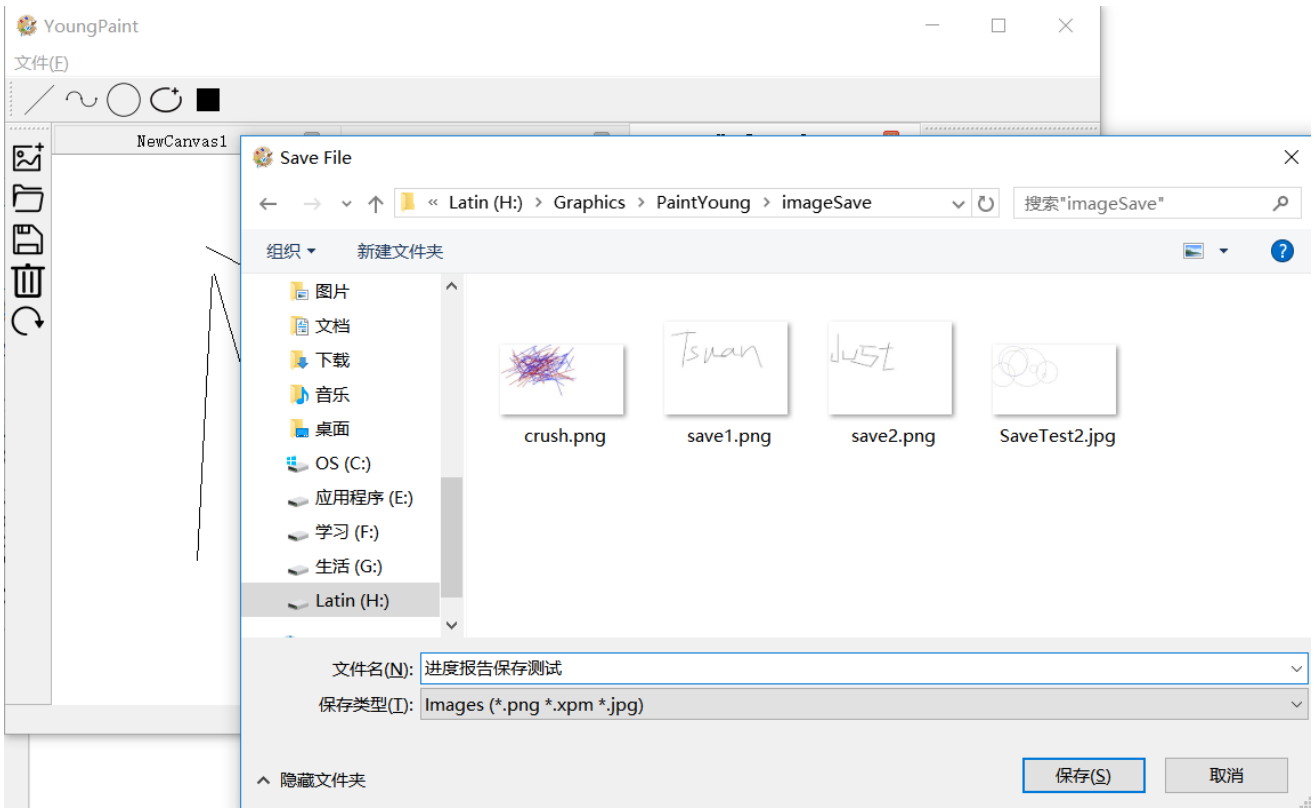
(6) 撤销：点击左侧工具栏撤销按钮，即可撤销



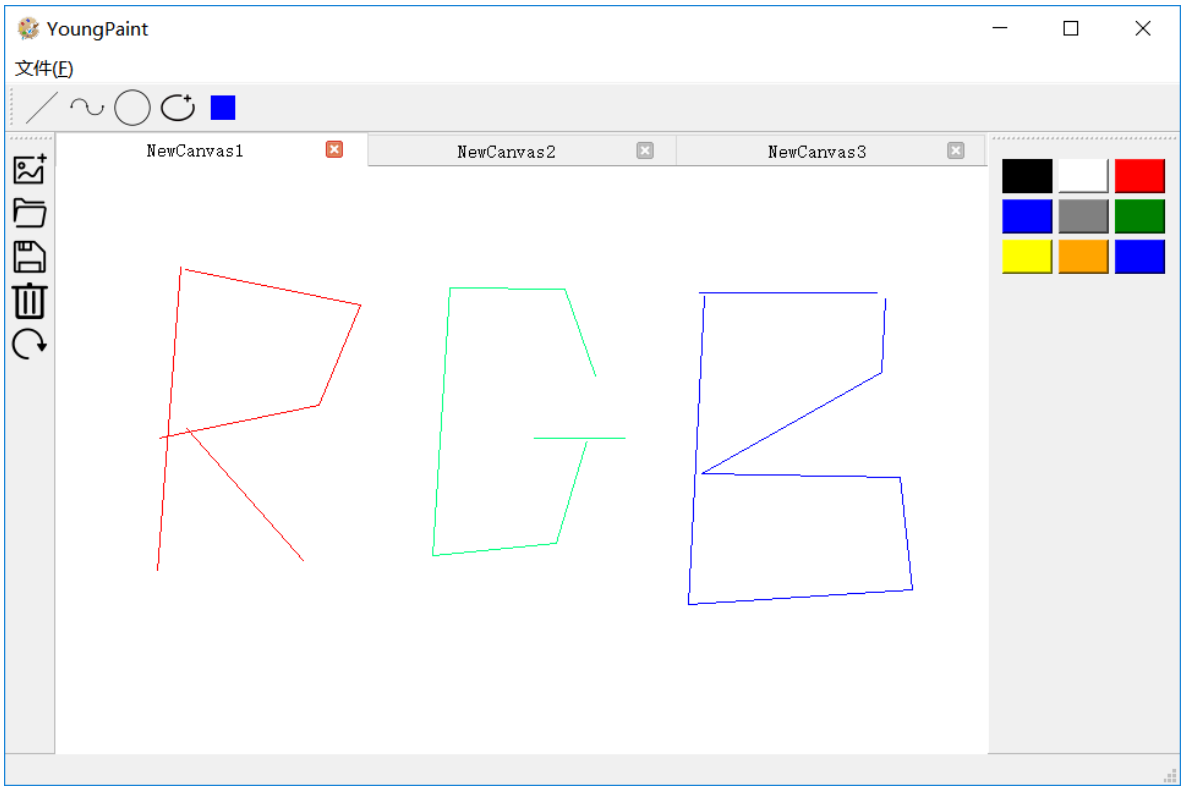
(7) 清屏：点击左侧工具栏清屏按钮，即可清屏



(8) 保存：点击左侧工具栏保存按钮，即可保存



(9) 颜色选择：点击上方工具栏颜色选择按钮，即可选择颜色



4.总结

本绘图系统名为YoungPaint，基于C++和Qt 5.11.2，于Qt Creator上开发，编译环境为MinGW 5.3.0

在9-10月关于图形学的学习中，基于我在课上所学的理论知识，以及课外对于Qt的交互、界面设计的学习，在截止10月底的系统中，我实现了二维图形中直线，圆以及椭圆的输入，并且实现创建多个窗口，画笔颜色的选择，绘画的撤销以及图像的保存功能。

这次实验是我第一次写具有图形交互的的实验，感觉十分有趣。把图形学课上的理论同实践相结合并且不断探索，不断阅读各种文档资料学习新知识的感觉也不错。尤其是双缓冲绘图的实现，起初我为了实现类似画图程序的动态效果而自己实现了一个，后来听同学说这就是双缓冲技术，独立探索出了这样的技巧让我感觉我的确是有在学习东西的，这也让我对于该实验有着更大的兴趣。尽管由于其他原因，本次10月份的程序不能说尽善尽美，但是基于我对于程序的理解，一遍上着高级程序设计课学习C++各种高级性质，我尽可能把我的知识和设计体现在代码上。

5.参考文献

[1] 孙正兴,周良,郑洪源,谢强.计算机图形学教程.北京:机械工业出版社,2006.

[2] 陈家骏,郑滔.程序设计教程用 C++语言编程(第 3 版).北京:机械工业出版社,2015.

（附其他参考资料）

[3] Qt学习社区上的《Qt基础教程之Qt学习之路》

[4] Qt官方文档

[5] Qt5.9.4利用QOpenGLWidget类进行opengl绘图 https://blog.csdn.net/cpwwhsu/article/details/79773235

[6] Qt学习之路-简易画板3(双缓冲绘图) https://blog.csdn.net/u012891055/article/details/41727391?utm_source=blogxgwz0