计算机图形学大作业 12 月报告

姓名：高伟

学号：171860506

一．实验目的与要求

系统功能要求：

Part 1：核心算法模块（各种图元的生成、变换算法），具体的包括直线，圆，椭圆，多边形以及曲线的生成算法，包括图形的旋转、裁剪、平移和缩放操作。

Part 2：文件输入接口（读取包含了图元绘制指令序列的文本文件，依据指令调用①中的算法绘制图形以及保存图像），实现文件的读取以及文件指令的解析，并调用的 1 中的算法。

Part 3：用户交互接口（以鼠标交互的方式，通过鼠标事件获取所需参数并调用①中的算法将图元绘制到屏幕上），实现用户交互，模拟真实的画板。

二．开发环境与语言

本次实验在 windows 平台下使用 qt 开发框架(Qt Creator 4.9.0 (Enterprise))进行，使用的是 C++语言。

三．系统框架与系统功能的介绍

主要框架：

定义一个 mainwindow 类，继承于 Qmainwindow 类，其内部包含一个 ui 对象指针，用来表示主窗口，就是本次实验实现画图的图板，其他的成员变量见下图，包括一些图形集合的变量和图板基础属性以及一些辅助画图的变量。

```
Ui::MainWindow *ui;
bool leftpress;
QPixmap pixmap;
int presstype;
QVector<QRect> line_ass;
QVector<bool> clip_unshow;
QVector<QRect> rect_ass;
QVector<QRect> ellipse_ass;
QVector<QVector<QPoint>> polygon_ass;
QVector<QVector<QPoint>> curve_ass;
QVector<int> all_ass;
QVector<QPair<int,QColor>> id_color_ass;
```

leftpress 和 presstype 是 ui 操作的辅助变量，leftpress 表示鼠标左键的按下，presstype 表示鼠标左键按下的操作。

xxx_ass 表示对应的图形的集合，为了贴切画板的功能，在用户交互时还额外提供了画矩形的功能，all_ass 是所有图形的集合，id_color_ass 是一个 id，color 的 pair 型集合，其记录了图形的 id 和颜色，与 all_ass 严格一一对应。

```
QVector<bool> DDAline;
QVector<bool> DDApolygon;
QVector<bool> Beziercurve;
bool polygonfirst;
bool polygonmovefirst;

bool curvefirst;
```

DDAline，DDApolygon 和 Beziercurve 用来表示直线，多边形和曲线采用的算法，分别于 line_ass 以及 polygon_ass，curve_ass 严格对应。

```
QRect translaterect;
bool translate_in_rect;
QRect translate_move_rect;

QPoint rotatepoint;
int rotatedegrees;

QPoint scalepoint;
float scaletimes;

QRect cliprect;
```

这里是平移，旋转，缩放和裁剪操作的 ui 辅助变量。

```
void paintEvent(QPaintEvent *);
void mousePressEvent(QMouseEvent *);
void mouseReleaseEvent(QMouseEvent *);
void mouseMoveEvent(QMouseEvent *);
void wheelEvent(QWheelEvent *);
```

在 mainwindow 类中重写了 paintevent 函数和鼠标滚轮事件，鼠标和滚轮事件主要用于 ui 操作，其中滚轮主要用于缩放和旋转。paintevent 主要原理是顺序读取 all_ass 向量里的图形，确定是何种图形，找到对应的颜色，进行绘制，直到读到 all_ass 末尾。

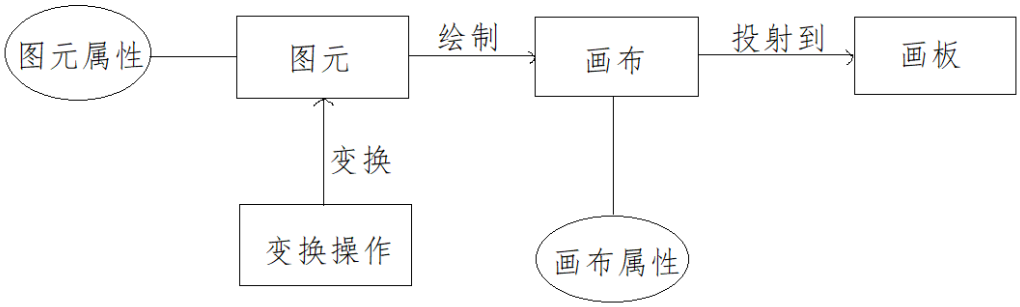
```
void f_resetCanvas(int w, int h);
void f_saveCanvas(QString name);
void f_setColor(int r, int g, int b);
void f_drawlineDDA(int x1, int y1, int x2, int y2,QPainter &painter);
void f_drawlinebresenham(int x1, int y1, int x2, int y2,QPainter &painter);
void f_drawrect(int x1, int y1, int x2, int y2, QPainter &painter);
void f_drawpolygonDDA(int n, int x[], int y[],QPainter &painter);
void f_drawpolygonbresenham(int n, int x[], int y[],QPainter &painter);
void f_drawellipse(int x0, int y0, int rx, int ry,QPainter &painter);
void f_drawcurvebezier(int n, int x[], int y[],QPainter &painter);
void f_drawcurvebspline(int n, int x[], int y[],QPainter &painter);
void f_translate(int id, int dx, int dy);
void f_rotate(int id, int xr, int yr, int r);
void f_scale(int id, int xf, int yf, float s);
int cs_judgebit(int x, int y, int left, int top, int rignt, int bottom);
void f_clipcs(int id, int left, int top, int rignt, int bottom);
void f_cliplb(int id, int left, int top, int rignt, int bottom);
void filedeal(QFile& file);
QRect& changepoint(int id, int &flag, int &count);
void getcontentpath(QString x);
```

这里是实现具体绘图的函数，函数的实现在 all_function.cpp 文件中。

```
if(argc == 3)
{
    QString path = argv[1];
    QString contentpath = argv[2];
    QFile file(path);
    w.getcontentpath(contentpath);
    w.filedeal(file);
}
w.show();
```

这是 main.cpp 中主要代码，当参数为 3 即参数包括文件和保存文件目录时，getcontentpath 用来获取文件目录，filedeal 用来处理文件。

如果不为 3 则正常运行画板 gui 功能。



画布是整个绘制的主体，而画板只是画布投射的地方，同时为了美观，将画布投射到画板上从（0,55）开始的地方。

接下来主要谈谈 filedeal 这个文件处理函数和 gui 实现：

主要原理：

打开文件后，读取文件指令时采用正则表达式实现指令匹配，事先设置好符合要求的指令格式，不符合此格式的都是非法指令。

每次从文件中 readline 一行，注意到在指令文件中绘制多边形和曲线的指令占用了两行，这里需要进行一行额外的读取。

```
rx[0].setPattern("resetCanvas\\s\\d{3,4}\\s\\d{3,4}");
rx[1].setPattern("saveCanvas\\s\\S+");
rx[2].setPattern("setColor\\s\\d{1,3}\\s\\d{1,3}\\s\\d{1,3}");
rx[3].setPattern("drawLine\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\sDDA");
rx[4].setPattern("drawLine\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\sBresenham");
rx[5].setPattern("drawPolygon\\s\\d{1,}\\s\\d{1,4}\\sDDA(\\s\\d{1,4})+");
rx[6].setPattern("drawPolygon\\s\\d{1,}\\s\\d{1,4}\\sBresenham(\\s\\d{1,4})+");
rx[7].setPattern("drawEllipse\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}");
rx[8].setPattern("drawCurve\\s\\d{1,}\\s\\d{1,4}\\sBezier(\\s\\d{1,4})+");
rx[9].setPattern("drawCurve\\s\\d{1,}\\s\\d{1,4}\\sB-spline(\\s\\d{1,4})+");
rx[10].setPattern("translate\\s\\d{1,}\\s-?\\d{1,4}\\s-?\\d{1,4}");
rx[11].setPattern("rotate\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s-?\\d{1,}");
rx[12].setPattern("scale\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,}.\\d*");
rx[13].setPattern("clip\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\sCohen-Sutherland");
rx[14].setPattern("clip\\s\\d{1,}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\s\\d{1,4}\\sLiang-Barsky");
```

每遇到符合规范的指令，根据其匹配的下标确定指令的操作，对指令字符串进行分解形成 stringlist，从其中获取 id，坐标等关键信息。

在用户交互方面，通过重写 mousePressEvent，mouseMoveEvent，mouseReleaseEvent，wheelEvent 函数来实现用户交互。

新图元的生成：

在 mousePressEvent 函数中，对于一个新的图元，每一次按下都在对应的图形集合和 all_ass 以及 id_color_ass 中 append 一个新的元素，直线，椭圆和矩形都是采用 qrect 进行存储，存储左上和右下的坐标(或者右上和左下的坐标)，曲线和多边形采用 QVector<QPoint>存储。

在 mouseMoveEvent 函数中，对于直线，椭圆，实时更新图形外接矩形端点的坐标，对于多边形，曲线，实时更新顶点或者控制点的坐标，并使用 update 函数进行窗口的重绘，使得图形可以进行实时的变化

在 mouseReleaseEvent 中，确定了最终图形相应点坐标，进行窗口重绘，并且通过将 leftpress 变量置为 false 来表示此图形绘制已经结束。

在绘制多边形时，则有一些不同，参考标准 windows 绘图板的多边形绘制效果，多边形的第一条线和直线一样，从第三个点开始每点一个点可以进行拖动并与前一个点相连接，当最后一个点与第一个点重合时，多边形绘制结束，由于像素太小，这里采用误差小于等于两个像素界定重合。

对于已有图元的平移等操作：

对于已有图元的 gui 平移等操作，为了简化考虑，只可以对最新的图元进行操作

平移：

定义一个变量 translaterect 用来存储平移图元的外接矩形，当鼠标左键点击在这个外接矩形中时，按住拖动鼠标左键，记录拖动的向量作为平移的向量，再调用平移实现的函数进行实现。

旋转：

定义一个变量 rotatepoint 来表示旋转中心，点击鼠标左键可随意更换，通过滚轮的滚动来实现旋转，滚轮向外侧滚动一格则顺时针旋转 1 度，向内侧滚动一格则逆时针旋转 1 度。

缩放：

定义一个变量 scalepoint 来表示缩放中心，点击鼠标左键可随意更换，通过滚轮的滚动来实现缩放，滚轮向外侧滚动则放大，反之则缩小

裁剪：

定义一个变量 cliprect 来表示裁剪区域，区域的生成和矩形的生成一致。

四．算法原理和理解

图元绘制：

绘制直线的 DDA 算法：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。

在实现时注意直线的斜率大于 1 和小于 1 的情况，可以通过变量进行整合，使其通过一种通用的方式进行绘制

绘制直线的 bresenham 算法：采用整数增量运算，根据光栅扫描原理(逐个像素和逐条扫描线显示图形)，线段离散过程中的每一放样位置上只可能有两个像素更接近于线段路径。Bresenham 算法引入一个整型参量来衡量“两候选像素与实际线路径点间的偏移关系”，通过对整型参量值符号的检测，选择候选像素中离实际线路径近的像素作为线的一个离散点。

在代码实现时同样要注意直线的斜率以及直线的绘制方向。

绘制多边形的 DDA 算法：n 多边形传入 n+1 组坐标，首尾坐标相同，这样 n+1 条边可以绘制多边形的 n 条边，这里认为多边形都是闭合的，之后采用直线绘制的 DDA 算法。

绘制多边形的 bresenham 算法：同上面，只是采用 bresenham 算法进行多边形边的绘制。

绘制椭圆的中点圆生成算法：

给定长短轴参数 rx、ry(假设 ry≤rx)和椭圆中心位置(xc,yc)，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(xc,yc)的点。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。在每个取样位置，按照椭圆函数在沿椭圆轨迹两个候选像素间中点求值的符号选择下一个像素。

依据椭圆弧切线斜率，第一像限椭圆(ry≤rx)可分成两部分。斜率绝对值小于 1 的区域 1 内沿 x 方向离散取样；斜率绝对值大于 1 的区域 2 内沿 y 方向离散取样。

区域一中：假如第 k 步选择的像素为(xk,yk)，将取样位置 xk+1 处两个候选像素间中点对 椭圆函数求值，即：计算决策参数：

p1k=fellipse(xk+1, yk-1/2)=ry2(xk+1)2+rx2(yk-1/2)2-rx2ry2

假如 p1k<0，中点位于椭圆内，扫描线 yk 上的像素更接近于椭圆边界，选择像素位置：(xk+1,yk) 。

假如 p1k≥0 ，中点在椭圆外或边界上，所选像素应在扫描线 yk-1 上，选择像素位置：(xk+1,yk-1) 。

区域二中同理，只是沿 y 方向离散取样

最后在实现第一象限的绘制后，利用椭圆的奇偶对称性画出其他三个象限的曲线，

最后进行圆心的平移，具体实现时平移的时机可以改变。

绘制曲线的 Bezier 算法：

给定 Pk=(xk,yk,zk)，(k=0,1,2,…,n)共 n+1 个控制点，这些点混合产生下列位置向量 P(u)，用来描述 P0 和 Pn 间的逼近 Bézier 多项式函数的路径(Bézier 曲线)。

$$P(u)=\sum_{k=0}^{n}P_k BEZ_{k,n}(u),\quad 0\le u\le 1$$

Bézier 曲线段逼近这些控制点→控制多边形大致勾画 Bézier 曲线的形状。

一条 n 次 Bézier 曲线被表示成它的 n+1 个控制顶点的加权和，权是 Bernstein 基函数。

德卡斯特里奥(de Casteljau)算法：

$$P_i^r = \begin{cases} P_i \\ (1-u)P_i^{r-1} + uP_{i+1}^{r-1} \end{cases} (i = 0,1,2,\cdots,n-r),(r = 1,2,\cdots n)$$

上图中第一行定义域为 r = 0

u 每次取值自增 0.004，生成不同的型值点，将其依次用直线连接

绘制曲线的 B 样条算法：

B 样条曲线的数学表达式为:

$$P(u)=\sum_{i=0}^{n}P_i B_{i,k}(u) \qquad u\in[u_{k-1},u_{n+1}]$$

$B_{i,k}$(i=0,1,..,n) 称为 k 阶（k-1 次)B 样条基函数，k 是刻画次数的。其中 k 可以是 2 到控制点个数 n+1 之间的任意整数。

de Boor-Cox 递推定义：

它的原理是，只要是 k 次多项式（k 次的 B 样条基函数），构造一种递推的公式，由 0 次构造 1 次，1 次构造 2 次，2 次构造 3 次...依次类推。

$$B_{i,1}(u)=\begin{cases}1 & u_i<x<u_{i+1}\\ 0 & Otherwise\end{cases} \qquad 并约定：\ \frac{0}{0}=0$$

$B_{i,1}(u)$是 0 次多项式，0 次多项式是常数。

$$B_{i,k}(u)=\frac{u-u_i}{u_{i+k-1}-u_i}B_{i,k-1}(u)+\frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}B_{i+1,k-1}(u)$$

与 Bezier 算法相同，u 同样每次自增 0.004，生成不同的型值点，将其依次用直线连接

图元变换：

平移：

原始位置 P(x,y)按平移距离 tx，ty 到新位置 P1(x1,y1)的移动：

x1 = x + tx, y1 = y + ty

(tx,ty)为平移向量

直线：平移线的每个端点

多边形：平移每个顶点的坐标

圆或椭圆：平移其中心坐标或者平移其外接矩形的端点（本次实现采用后者）

曲线：平移定义曲线的坐标位置，用新的坐标位置重构曲线

直线、矩形和椭圆：

```
int x0 = p.topLeft().x();
int y0 = p.topLeft().y();
int x1 = p.bottomRight().x();
int y1 = p.bottomRight().y();
p.setTopLeft(QPoint(x0+dx,y0+dy));
p.setBottomRight(QPoint(x1+dx,y1+dy));
```

多边形：

```
int x0 = polygon_ass[count][i].x();
int y0 = polygon_ass[count][i].y();
polygon_ass[count][i].setX(x0+dx);
polygon_ass[count][i].setY(y0+dy);
```

曲线：

```
int x0 = curve_ass[count][i].x();
int y0 = curve_ass[count][i].y();
curve_ass[count][i].setX(x0+dx);
curve_ass[count][i].setY(y0+dy);
```

旋转：

由于图形学中坐标轴与数学中坐标轴方向有所差异，经过计算，顺时针旋转角应该为正，逆时针为负

任意基准位置：

$x1 = xr + (x-xr)\cos\theta - (y-yr)\sin\theta$

$y1 = yr + (x-xr)\sin\theta + (y-yr)\cos\theta$

直线段：旋转每个线段端点

多边形：旋转每个顶点

曲线：旋转控制/取样点

直线：

```
int x0 = p.topLeft().x();
int y0 = p.topLeft().y();
int x1 = p.bottomRight().x();
int y1 = p.bottomRight().y();
int final_x0 = round(xr + (x0-xr)*cos(r*M_PI/180.0)-(y0-yr)*sin(r*M_PI/180.0));
int final_y0 = round(yr + (x0-xr)*sin(r*M_PI/180.0)+(y0-yr)*cos(r*M_PI/180.0));
int final_x1 = round(xr + (x1-xr)*cos(r*M_PI/180.0)-(y1-yr)*sin(r*M_PI/180.0));
int final_y1 = round(yr + (x1-xr)*sin(r*M_PI/180.0)+(y1-yr)*cos(r*M_PI/180.0));
p.setTopLeft(QPoint(final_x0,final_y0));
p.setBottomRight(QPoint(final_x1,final_y1));
```

多边形：

```
int x0 = polygon_ass[count][i].x();
int y0 = polygon_ass[count][i].y();
int final_x0 = round(xr + (x0-xr)*cos(r*M_PI/180)-(y0-yr)*sin(r*M_PI/180));
int final_y0 = round(yr + (x0-xr)*sin(r*M_PI/180)+(y0-yr)*cos(r*M_PI/180));
polygon_ass[count][i].setX(final_x0);
polygon_ass[count][i].setY(final_y0);
```

曲线：

```
int x0 = curve_ass[count][i].x();
int y0 = curve_ass[count][i].y();
int final_x0 = round(xr + (x0-xr)*cos(r*M_PI/180)-(y0-yr)*sin(r*M_PI/180));
int final_y0 = round(yr + (x0-xr)*sin(r*M_PI/180)+(y0-yr)*cos(r*M_PI/180));
curve_ass[count][i].setX(final_x0);
curve_ass[count][i].setY(final_y0);
```

缩放：

x1 = x*sx + xf(1-sx)

y1 = y*sy + yf(1-sy)

xf(1-sx)和 yf(1-sy)对任何点都是常数->原点缩放+平移

直线缩放：缩放两个端点

多边形：缩放每个顶点

其他物体：缩放定义物体的参数

直线，椭圆：

```
int x0 = p.topLeft().x();
int y0 = p.topLeft().y();
int x1 = p.bottomRight().x();
int y1 = p.bottomRight().y();
x0 = round(xf +(x0-xf)*s);
y0 = round(yf +(y0-yf)*s);
x1 = round(xf +(x1-xf)*s);
y1 = round(yf +(y1-yf)*s);
p.setTopLeft(QPoint(x0,y0));
p.setBottomRight(QPoint(x1,y1));
```

多边形：

```
int x0 = polygon_ass[count][i].x();
int y0 = polygon_ass[count][i].y();
x0 = round(xf +(x0-xf)*s);
y0 = round(yf +(y0-yf)*s);
polygon_ass[count][i].setX(x0);
polygon_ass[count][i].setY(y0);
```

曲线：

```
int x0 = curve_ass[count][i].x();
int y0 = curve_ass[count][i].y();
x0 = round(xf +(x0-xf)*s);
y0 = round(yf +(y0-yf)*s);
curve_ass[count][i].setX(x0);
curve_ass[count][i].setY(y0);
```

裁剪：

Cohen-Sutherland 算法：

核心思想：通过编码测试来减少要计算交点的次数 — 编码算法。

区域码的各位：线段端点与四条裁剪窗口边界的相对位置关系。

区域码各位从右到左编号：

位 1：上边界；

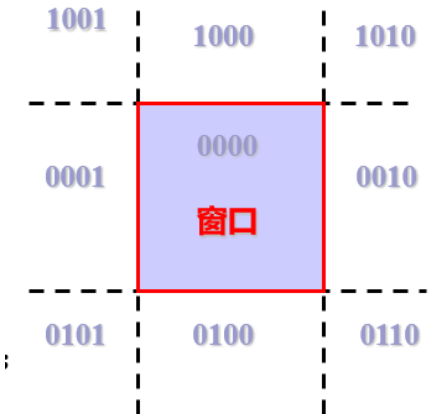
位 2：下边界；

位 3：右边界；

位 4：左边界。

区域码位＝1：端点落在相应位置上；

区域码位＝0：端点不在相应位置上。



y＞ ywmax ： 第 1 位置 1；否则，置 0；

y＜ ywmin ： 第 2 位置 1；否则，置 0；

x ＜ xwmin ： 第 3 位置 1；否则，置 0；

x ＞ xwmax ： 第 4 位置 1；否则，置 0。

1. 完全在窗口边界内的线段：两端点区域码均为 0000；

2. 完全在裁剪矩形外的线段：

两端点区域码同样位置都为 1。

对两个端点区域码进行逻辑与操作，结果不为 0000。

3. 不能确定完全在窗口内外的线段，进行求交运算

按“左-右-上-下”顺序用裁剪边界检查线段端点。

将线段的外端点与裁剪边界进行比较和求交，确定应裁剪掉的线段部分；

反复对线段的剩下部分与其它裁剪边界进行比较和求交，直到该线段完全被舍弃或找到位于窗口内的一段线段为止。

梁友栋-Barsky 裁剪算法：

一条两端点为 P1（x1，y1）、P2（x2，y2）的线段可以用参数方程形式表示：

$$x= x_1+ u\cdot(x_2-x_1) = x_1+ u\cdot\Delta x$$

$$y= y_1+ u\cdot(y_2-y_1) = y_1+ u\cdot\Delta y$$

如果点 P（x，y）位于由坐标（xmin，ymin）和（xmax，ymax）所确定的窗口内，那么下式成立：

$$xmin\le x1+ u\cdot\Delta x\le xmax$$

$$ymin\le y1+ u\cdot\Delta y\le ymax$$

这四个不等式可以表示为：$u \bullet pk \le qk$ ， k=1，2，3，4

其中，p、q 定义为：

$$p1=-\Delta x,\quad q1=x1-xmin$$

$$p2=\Delta x,\quad q2=xmax-x1$$

$$p3=-\Delta y,\quad q3=y1-ymin$$

$$p4=\Delta y,\quad q4=ymax-y1$$

任何平行于窗口某边界的直线，其 pk=0，k 值对应于相应的边界（k=1，2，3，4 对应于左、右、下、上边界）。

1、当 pk<0 时，线段从裁剪边界延长线的外部延伸到内部；

2、当 pk>0 时，线段从裁剪边界延长线的内部延伸到外部；

对于每条直线，可以计算出参数 u1 和 u2，该值定义了位于窗口内的线段部分：

1、u1 的值由线段从外到内遇到的矩形边界所决定（pk<0），对这些边界计算 rk=qk/pk，u1 取 0 和各个 r 值之中的最大值。

2、u2 的值由线段从内到外遇到的矩形边界所决定（pk>0），对这些边界计算 rk=qk/pk，u2 取 1 和各个 r 值之中的最小值。

3、如果 u1>u2，则线段完全落在裁剪窗口之外，应当被舍弃；否则，被裁剪线段的端点可以由 u1 和 u2 计算出来。

由于这两个裁剪算法代码量较大，因此详细细节见代码文件

Cohen-Sutherland 算法和梁友栋-Barsky 算法的比较：

梁友栋-Barsky 算法更新参数 u1、u2 仅需一次除法，线段与窗口的交点仅计算

一次就计算出 u1、u2 的最后值。

Cohen-Sutherland 算法即使对完全落在裁剪窗口之外的一条线段，也要对它反复求交点，而且每次求交计算都需要除法和乘法运算。

五．其他功能的实现：

重置画布：

```
void MainWindow::f_resetCanvas(int w, int h)
{
    my_width = w;
    my_height = h;
}
```

保存文件：

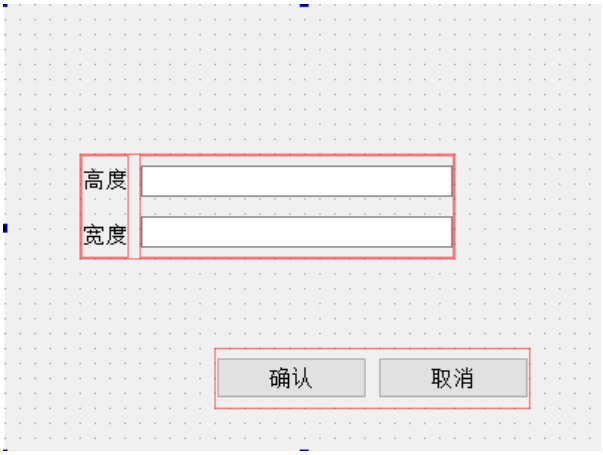
```
void MainWindow::f_saveCanvas(QString name)
{
    QPixmap pixmap(my_width,my_height+55);
    QPainter p(&pixmap);
    p.fillRect(QRect(0, 0, my_width, my_height+55), Qt::white);
    this->render(&p);
    QString res = contentpath+name+".bmp";
    pixmap.copy(QRect(0,55,my_width,my_height)).save(res);
}
```

设置颜色：

```
void MainWindow::f_setColor(int r, int g, int b)
{
    my_r = r;
    my_g = g;
    my_b = b;
}
```

打开和保存文件在 ui 中可以通过工具栏实现

重置画布也可以在工具栏实现：



ui 界面可以使用颜色对话框来改变 ui 绘图时的颜色。

```
QAction *changecolorAction = new QAction(tr("&颜色"),this);
changecolorAction->setIcon(QIcon(":/myicon/images/color.png"));
tbar->addAction(changecolorAction);

connect(changecolorAction,&QAction::triggered,this,&MainWindow::presscolor);
```

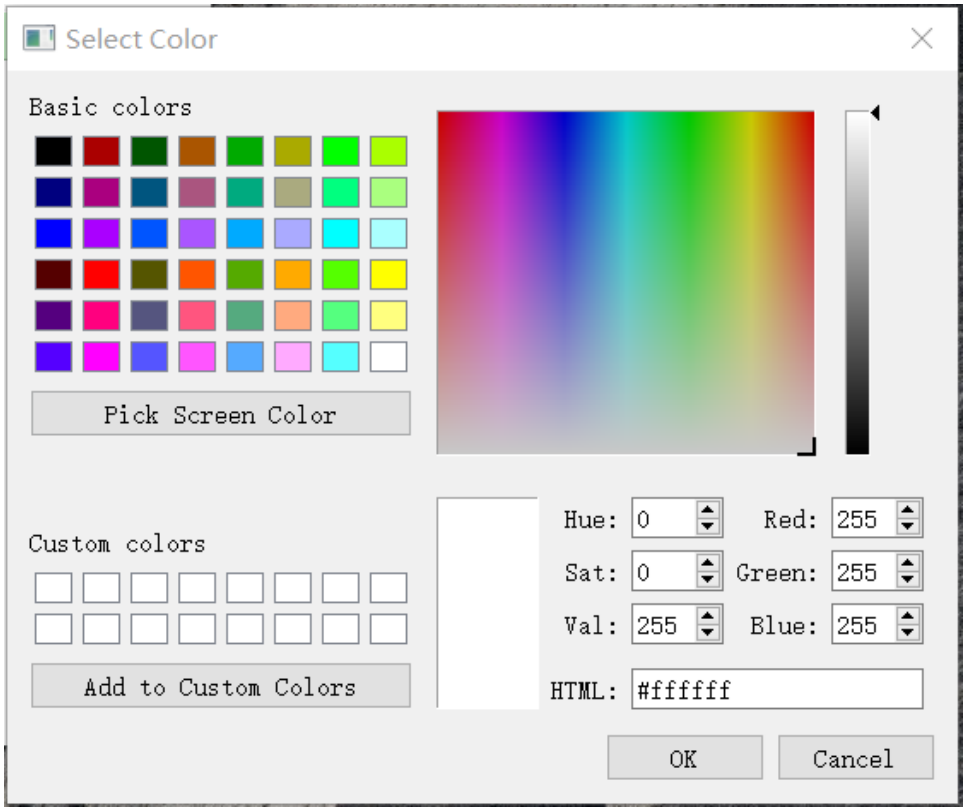
这里是 ui 界面的设置，同时将其与槽函数 presscolor 相连接

下面是 presscolor 函数的内容：

```
void MainWindow::presscolor()
{
    QColor color = QColorDialog::getColor();
    if(color.isValid())
    {
        my_r = color.red();
        my_g = color.green();
        my_b = color.blue();
    }
}
```

在 ui 界面中点击下面这个 action

就会弹出选择颜色的对话框



选完颜色后点击 OK 就可以获取到颜色，点击 cancel 则不会发生颜色的变化。

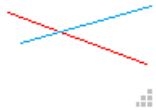
具体的 ui 操作将在系统说明书中展示，在此不加赘述。

六．实验结果

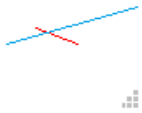
运行助教所给样例的运行结果：

output_1

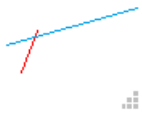
output_2



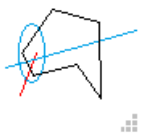
output_3



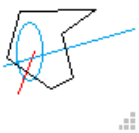
output_4



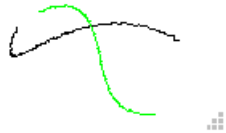
output_5



output_6



output_7



七. 参考网站

https://www.bilibili.com/video/av34085761?from=search&seid=13578330736137223665

https://www.cnblogs.com/liuyunfeifei/archive/2013/02/26/2933411.html

https://blog.csdn.net/technologyleader/article/details/82151188

https://blog.csdn.net/judgejames/article/details/94020929

http://www.voidcn.com/article/p-kwddxvzp-btc.html

https://www.cnblogs.com/nonsupport/p/8571163.html

https://blog.csdn.net/nanfeibuyi/article/details/80295663

https://www.cnblogs.com/weiweiqiao99/archive/2011/01/08/1930931.html

https://blog.csdn.net/a724699769/article/details/62216435

https://www.cnblogs.com/wanghuixi/p/9540694.html

https://www.cnblogs.com/xiaomanon/p/3868242.html

http://blog.sina.com.cn/s/blog_756b9f550100vdq1.html