计算机图形学系统报告

张玄逸 201220194

联系方式：1822771416@qq.com

南京大学 计算机科学与技术系

一、综述

本实验实现了一个简单画图工具，可以用鼠标操作，在基本图形方面可以画出直线，椭圆，多边形及曲线，在图形处理方面可以进行平移，旋转，剪切，在功能方面可以调整画笔，重置画布，保存画布等。

二、算法介绍

1.线段绘制算法

- **DDA算法**

  原理：通过在一个坐标轴上以单位间隔（如1）对线段取样，计算另一个坐标轴上最靠近线段路径的对应整数值。x采用从 0 开始的整数值，作为第一个点并增加 1 直到到达端点。y 值四舍五入到最接近的整数以对应于屏幕像素。

  - 考虑具有正斜率的线，如果斜率m小于等于 1，我们以 dx=1 进行采样，计算方式为：

    x = x + 1

    y = y + m
  - 对于斜率m大于 1 的线，我们颠倒 x 和 y 的作用，即以 dy=1 采样，并将连续的 x 值计算为

    y = y + 1

    x = x + 1/m

  具有负斜率的线方法相似，如果斜率绝对值小于 1，我们设置 dx=1，否则dy=1。

  注意：要判断斜率为0和不存在的情况

- **Bresenham算法**

  原理：它是一种精确有效的光栅线段生成算法，以某个坐标轴为单位取样，在每一个取样位置处确定选择哪个像素位置。与DDA不同的是，它通过引入整型参量定义来衡量两候选像素与线路径上实际点间在某方向上的相对偏移，并利用对整型参量符号的检测来确定最接近于实际线路径的像素。

  考虑斜率m<=1的情况，此时x_k对应的y上下分别有y_{k+1}和y_k，两个候选像素与y的数值偏移分别为：d_1 = y - y_k = m(x_k+1) + b - y_k

  d_1 = y_{k+1} - y = y_k + 1 - m(x_k+1) - b

  这两个点的距离差为：d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1

  令△x和△y分别为相邻点的单位偏移量，则上式可写为：

p_k = $\triangle x$(d_1 - d_2) = 2$\triangle y x_k$ - 2$\triangle x y_k$ + c，c = 2$\triangle y$ + $\triangle x$(2b - 1)是一个常量

此时就可以根据p_k的正负判断应选择哪个点。p_k为负时选择较低点，为正时选择较高点

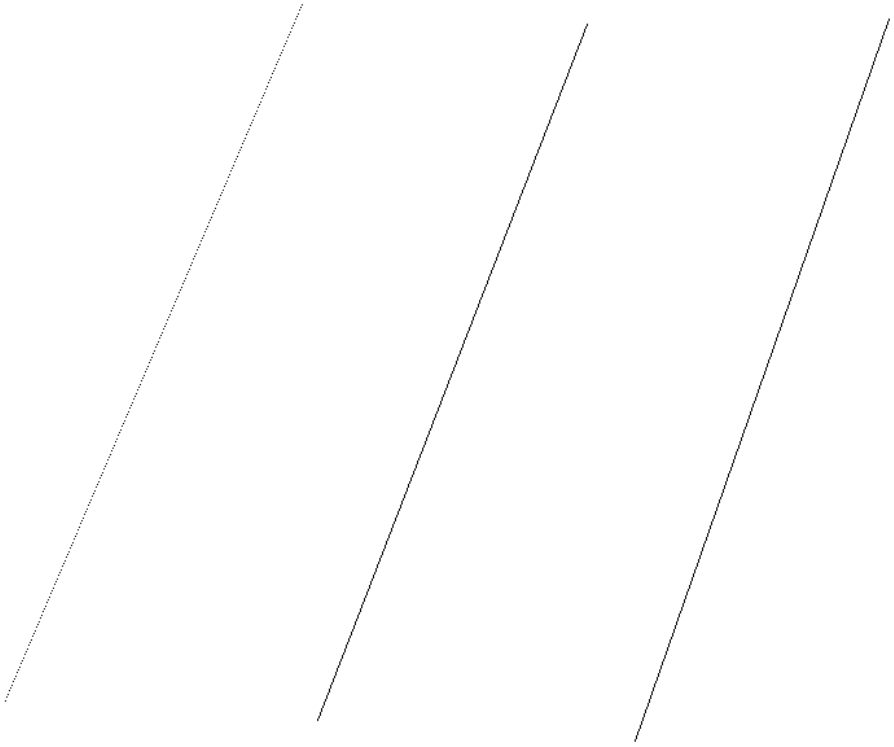
相邻两个p相减可以得到：p_{k+1} = p_k + 2$\triangle y$ - 2$\triangle x$ (y_{k+1} - y_k)

- p_k > 0 时，y_{k+1} 取高像素，即(y_{k+1} - y_k) = 1，p_{k+1} = p_k + 2$\triangle y$ - 2$\triangle x$
- p_k < 0 时，y_{k+1} 取低像素，即(y_{k+1} - y_k) = 0，p_{k+1} = p_k + 2$\triangle y$

经过反复迭代就可以得到线段上的所有点集，而第一个端点的参数p_0 = 2$\triangle y$ - $\triangle x$

斜率大于1时可以将xy倒转求值，负斜率类似。

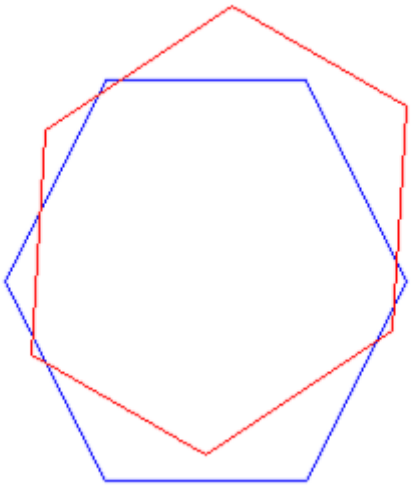
注意：平行线，竖直线，对角线要特殊处理。另外，对于p_k=0的情况一定要保持选择偏上或偏下的像素点。



注：最左为naive，中间为DDA算法，最右为Bresenham算法

2.多边形绘制算法

因为多边形可以看作多条首尾相连的线段，所以多边形的绘制方法和线段绘制算法相似。



3.椭圆绘制算法

采用中点椭圆生成算法。

首先，因为平面椭圆有两条对称轴，因此只需要计算第一象限即可。另外，可以先将圆心平移到（0，0）绘制，之后再平移到原来的位置。在斜率绝对值小于1的区域，沿x轴正方向取单位步长，每次在两个y的候选像素中选择一个；在斜率绝对值大于1的区域，沿y轴负方向取单位步长，每次在两个x的候选像素中选择一个。

定义椭圆函数f(x,y) = $r_x^2x^2$ + $r_y^2y^2$ - $r_x^2r_y^2$，通过函数的正负值可以判定点在圆内，圆外还是圆上。检测对每对候选像素的中点进行。

从（0,r_y）开始，起始的决策参数 p_k = $r_y^2 - r_x^2r_y + \frac{r_x^2}{4}$

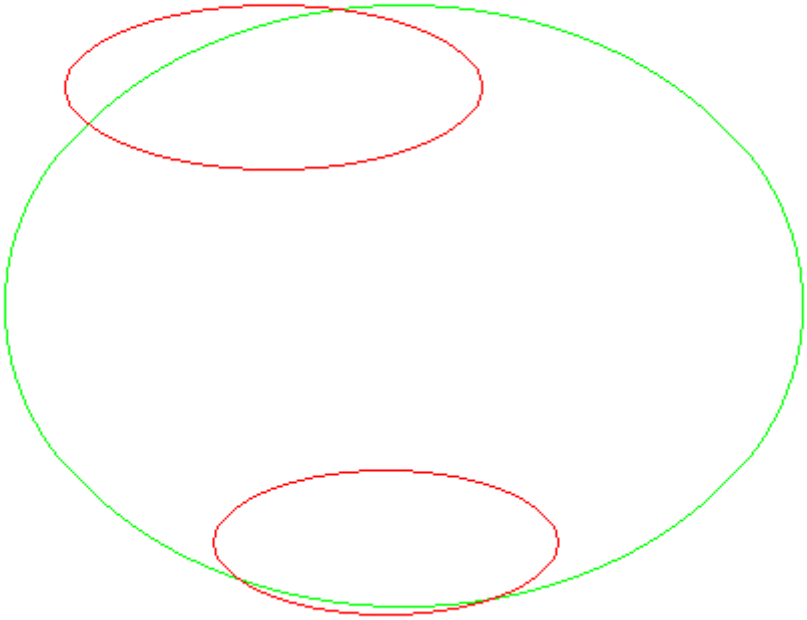
斜率绝对值小于1的区域：

- 若p_k为负
  - 中点在圆内，选择较高点y_{k+1}（即y不变），下一对候选像素为y_{k+1}与y_k
  - $p_{k+1} = p_k + 2r_y^2x_k + 3r_y^2$
- 若p_k为正
  - 中点在圆外，选择较低点y_k（即y-1），下一对候选像素为y_k与y_{k-1}
  - $p_{k+1} = p_k + 2r_y^2x_k - 2r_x^2y_k + 2r_x^2 + 3r_y^2$

当$r_y^2x_k > r_x^2y_k$时，将最后一个点(x_k, y_k)作为斜率大于1的起始点，参数初始化为
$r_y^2(x + \frac{1}{2})^2 + r_x^2(y-1)^2 - r_x^2r_y^2$

斜率绝对值大于1的区域：

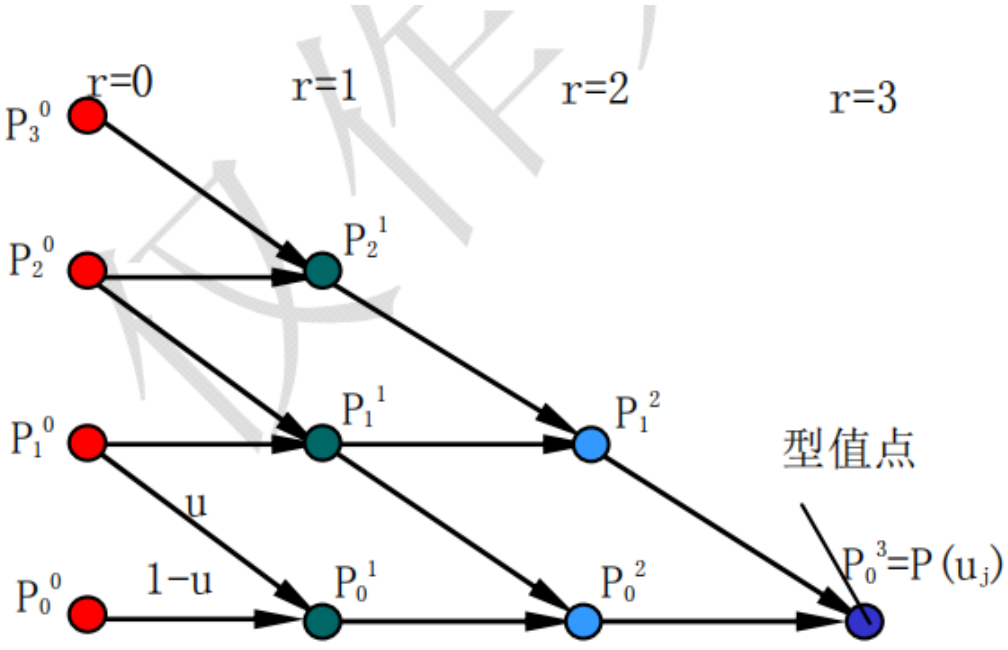
- 若p_k为负
  - 中点在圆内，选择较低点x_k（即x+1），下一对候选像素为x_{k+1}与x_k
  - $p_{k+1} = p_k + 2r_y^2x_k - 2r_x^2y_k + 2r_y^2 + 3r_x^2$
- 若p_k为正
  - 中点在圆外，选择较高点x_{k-1}（即x不变），下一对候选像素为x_k与x_{k-1}
  - $p_{k+1} = p_k + 2r_x^2y_k + 3r_x^2$



4.曲线绘制算法

- Bezier算法

  直接用曲线方程计算量太大，所以采用分割递推方法。



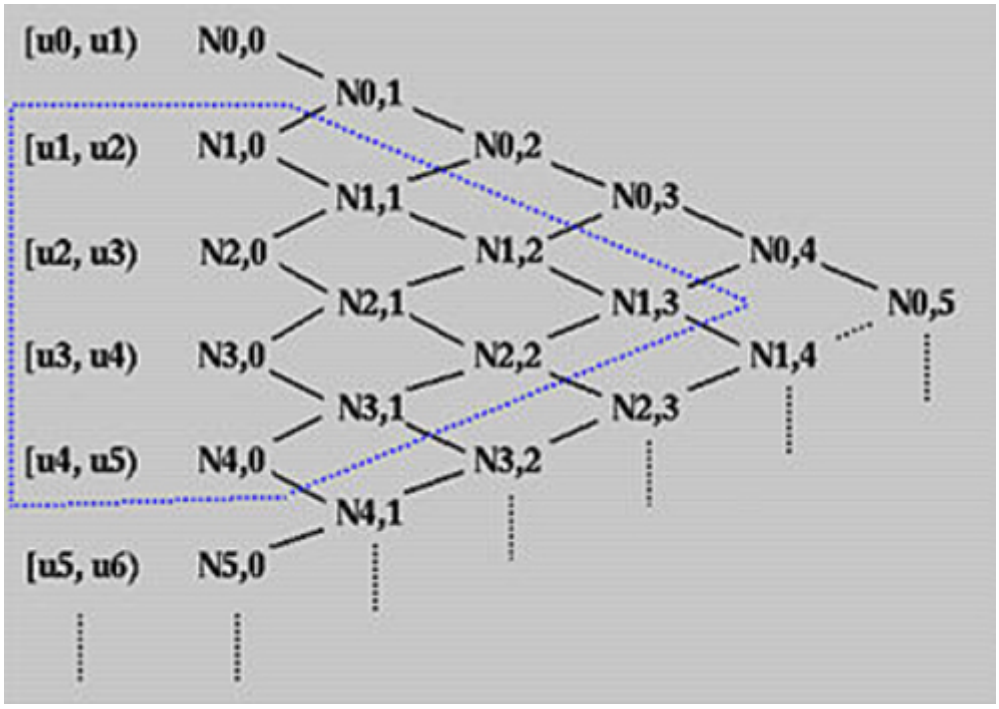
对于某一个参数u，它需要所有n个顶点来迭代计算。

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

其中n为顶点个数，P_i为第 i 个顶点向量。

- B-spline算法

  与Bezier算法不同的是，B样条算法不需要用所有n个顶点来计算某一个u的对应点，而是只取递推三角形的前k阶。这样增加了曲线的局部性和灵活度。

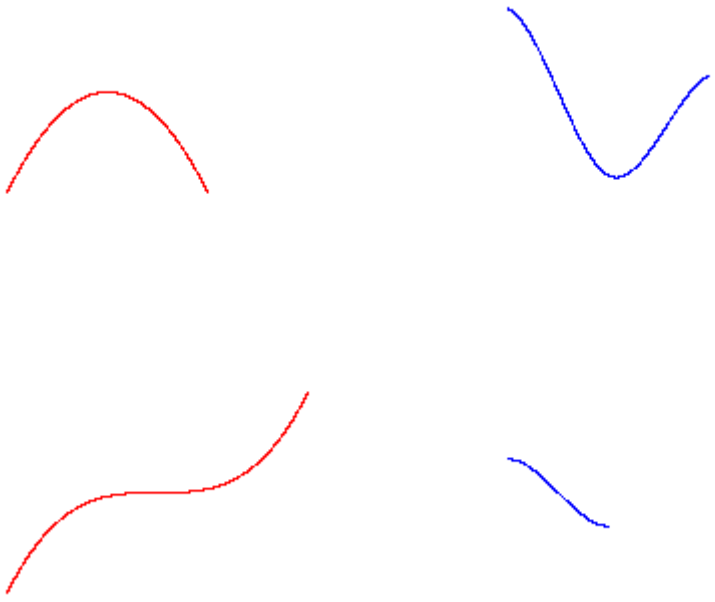


基函数$B_{i,k}(u) = \frac{u-u_i}{u_{i+k-1}-u_i}B_{i,k-1}(u) + \frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}B_{i+1,k-1}(u)$

当$u \in [u_i, u_{i+1}]$时，$B_{i,1}(u) = 1$

当$u \notin [u_i, u_{i+1}]$时，$B_{i,1}(u) = 0$

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



红为Bezier算法，蓝为B-spline算法。

图元平移算法

若平移向量为(t_x, t_y)，则对于图形上的每一个点(x,y)，有

- x = x + t_x
- y = y + t_y

图元旋转算法

当基准点为坐标原点时，旋转变换方程为：

- $x_2 = x_1 cos\theta - y_1 sin\theta$
- $y_2 = x_1 sin\theta + y_1 cos\theta$

图元缩放算法

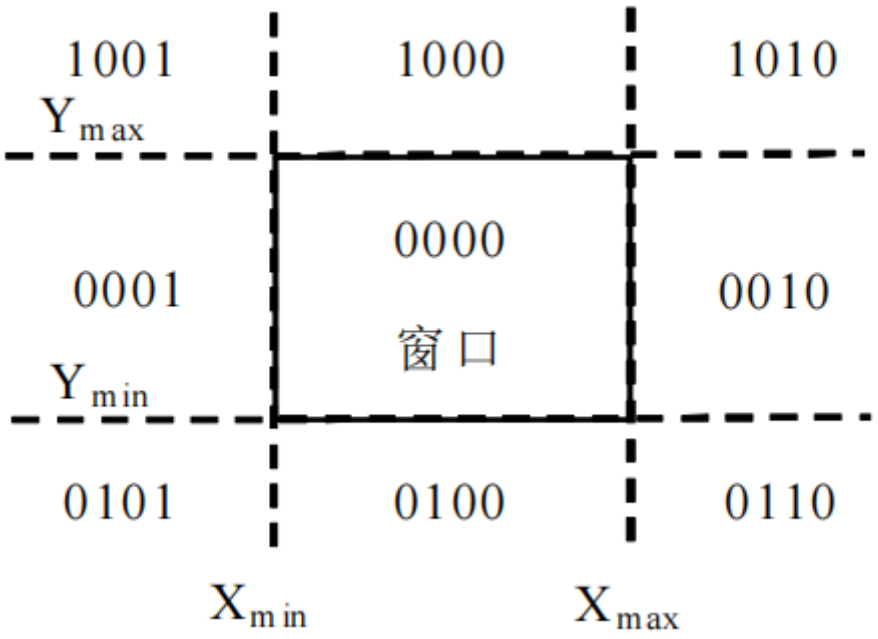
若缩放中心为(t_x, t_y)，缩放倍数为r，则对于图形上的每一个点(x,y)，有

- x = (x - t_x) * r + t_x
- y = (y - t_y) * r + t_y

线段裁剪算法

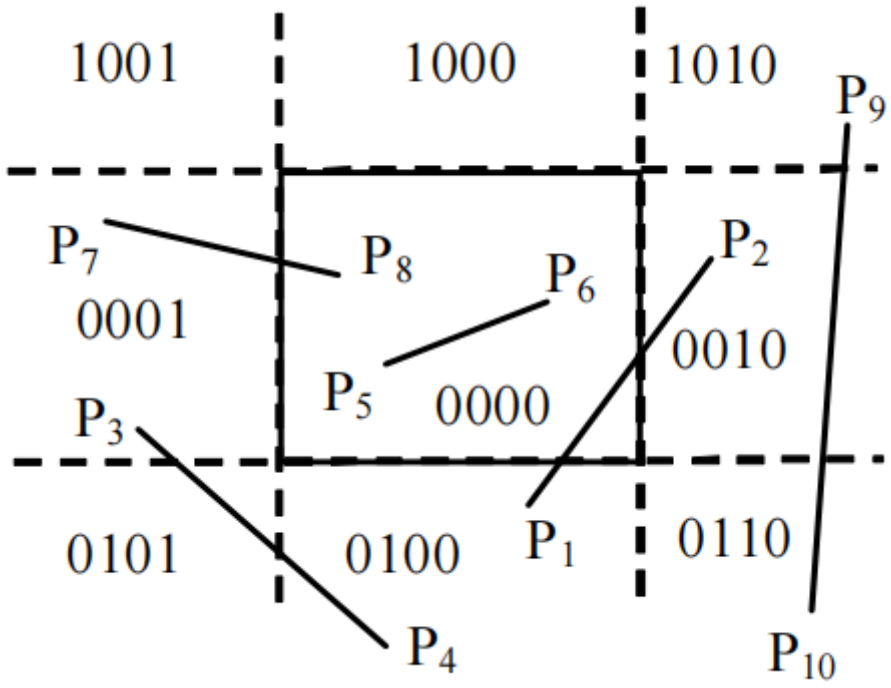
- **Cohen-Sutherland算法**

  编码算法以下图所示的 9 个区域为基础，根据每条线段的端点坐标所在的区域，每个端点均赋以四位二进制码，称为区域码。区域码的各位表明线段端点对于裁剪窗口的四个相对坐标位置。我们需要获取线段的两个端点对应的编码。



  从右到左的1-4位分别是 $x - x_{min}, x_{max} - x, y - y_{min}, y_{max} - y$ 的符号位。

  线段有三种情况。



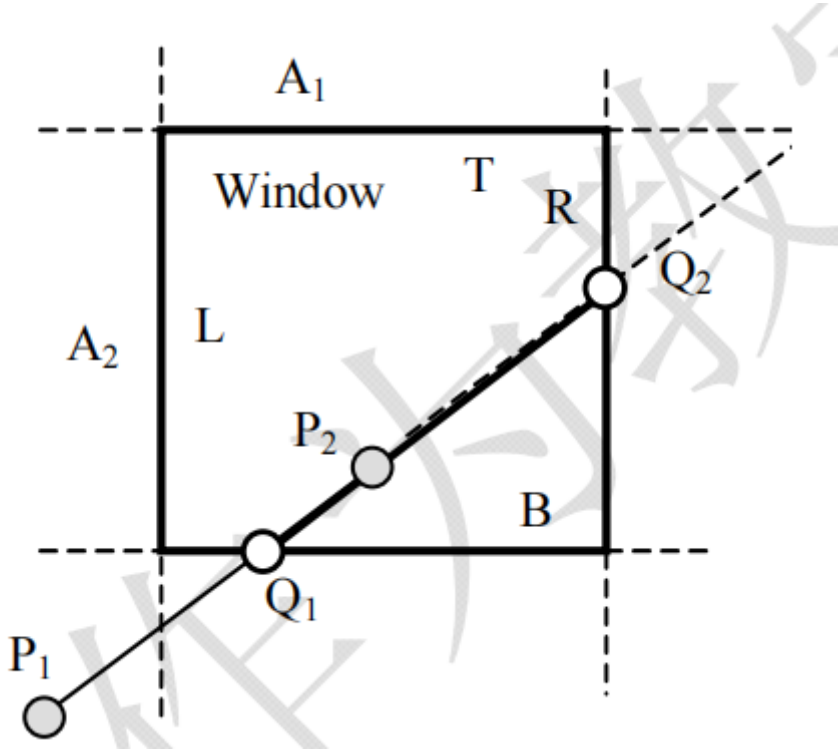
- 两个编码均为0000：线段完全在窗口内，停止裁剪
- 两个编码与结果不为0，说明完全在窗口外，全部舍弃
- 其它：部分在窗口外，此时需要根据某一端编码不同数位上的01判断与哪个窗口边界相交，并计算去掉外面的部分。

处理完后继续循环，直到进入前两个情况之一。

- Liang-Barsky算法

  记线段P1P2（及其延长线）与窗口的交点为Q1,Q2，称Q1Q2为诱导窗口。将P1P2用参数方程表示：P=P1+u(P2-P1)（对P1P2而言0<=u<=1），则Q1Q2对应的参数为u1，u2。其可见部分的区间为

  $$[max(u_{p1}=0,u_1),min(u_{p2}=1,u_2)]$$



裁剪条件的参数化表示如下

- $xw_{min} <= x_1 + u \triangle x <= xw_{max}$
- $yw_{min} <= y_1 + u \triangle y <= yw_{max}$

  xw_{min} 等变量是窗口边界。

可统一表示为$u * p_k <= q_k$，其中k=0,1,2,3，对应窗口的左，右，下，上边界。

- $p_1 = x_1 - x_2, q_1 = x_1 - xw_{min}$
- $p_2 = x_2 - x_1, q_2 = xw_{max} - x_1$
- $p_3 = y_1 - y_2, q_3 = y_1 - yw_{min}$
- $p_4 = y_2 - y_1, q_2 = yw_{max} - y_1$

根据p_k可以确定线段与窗口的位置关系。

- p_k = 0：直线平行于窗口一条边界，此时只需要再根据q值即可判断是否保留
- p_k < 0：直线从外到内
- p_k > 0：直线从内到外

因此当p_k != 0 时 u = q_k/p_k，p<0时用于更新u_1（取最大值），反之更新u_2（取最小值）

三、系统设计

- **代码结构**

  cg_algorithm.py：所有基本绘制函数

  cg_cli.py：类似命令行的实现方式。通过读取文件获取命令，调用cg_algorithm模块将点信息储存，在save函数调用时将所有点绘制到画布上并保存。

  cg_gui.py：图形化界面的画布，与cg_cli.py相互独立，只调用cg_algorithm模块。

- **功能设计**

  - **画布外观**

    首先调整整体窗口自适应大小，画布大小调整为650*550，图标大小为40*40。

    ```
    QApplication.setAttribute(Qt.AA_EnableHighDpiScaling)
    ...
    self.scene.setSceneRect(0, 0, height, width)  #绘画范围
    self.canvas_widget = MyCanvas(self.scene, self)
    self.canvas_widget.setFixedSize(height, width)  #画布窗口大小
    ```

  - **画布重置**

    点击“画布重置”按钮，可以清空整个画布及选择框。之后弹出对话框，可以重新设置画布大小。

  - **画布保存**

    点击“画布保存”按钮可以将画布保存为bmp位图，自选路径。

    保存通过QFileDialog类获取用户自选的路径与名称。

```
filename = QFileDialog.getSaveFileName(self, '选择存储位置', 'utitled',
    'Images (*.bmp)')
if filename[0]:
    pix = self.canvas_widget.grab(
        self.canvas_widget.sceneRect().toRect())
    pix.save(filename[0])
```

- **画笔设置**

  使用QColorDialog类获取想要的画笔颜色并重新赋值，实现颜色的改变。

- **绘制线段（椭圆）**

  只允许左键点击。因为线段与椭圆都是点击——松开就完成一次绘制，因此只需要通过点击松开事件的重写就可以定位图形的两个起始点和终点。

- **绘制多边形（曲线）**

  由于这两者需要进行多次点击——松开才能完成一个图形的绘制，所以设定双击结束绘制。通过增加双击事件的重写，实现一个多边形（曲线）绘制过程的结束。注意一次双击事件会伴随着两次单击——松开事件，每次松开都会给图形的控制点列表末尾加上当前坐标。

- **菜单图标设置**

  之前软件采用的是单纯的多级按钮来实现功能的选择，结构为Menubar->Menu->Action，分别代表菜单栏，一级菜单和二级菜单（一般用于选择画线算法）。现在需要在每个一级菜单Menu上附加一个图标Icon，只需要在之前添加菜单时的menubar.addMenu函数中加入图标参数即可。

```
line_menu = draw_menu.addMenu('线段')#之前的代码
line_menu = menubar.addMenu(
    QIcon(QPixmap("resource/line.png").scaled(icon_size, icon_size)),
    '')  #现在的代码，添加了图标
```

但有一些菜单没有更低一级的选项，比如重置画布就属于一级菜单（因为它应该有一个单独的图标），但它并不需要下一级。现在的重置有了图标，但点击后会出现“重置画布”的二级菜单，点击这个按钮后才会执行重置，这并不是我想要的。

我先尝试了能否直接实现Menu的点击事件，发现并没有触发。查阅文档后得知，“此信号是为层次结构中的主父菜单发出的。因此，只有父菜单需要连接到插槽”。Menu只是充当一个Label的作用，它只负责提供下一级按钮而不会相应trigger函数，因此不能直接连接。

然后我将trigger即点击事件改为abouttoshow事件，这样的确能在点击Menu时就发出信号，但abouttoshow是在向用户显示菜单内容前发出的信号，这意味着用户有可能没有真正点击它。比如用户先点击了“重置画布”，此时该按钮处于被选择状态，而当鼠标移到“退出”按钮上还没有点击时，“退出”照样会尝试显示自己的菜单给用户看，这时触发abouttoshow信号致使软件退出。这与用户的本意相违背，因此也不是好的解决方案。

```
set_pen_act.triggered.connect(self.set_pen_action)
set_pen_act.abouttoshow.connect(self.set_pen_action)
#若set_pen_act为Menu，则第一个信号无效，第二个信号错误
```

继续查资料得知，菜单栏Menubar除了能添加Menu外，还可以直接添加Action（二级菜单），并且Action也能附加图标。

```
reset_canvas_act = menubar.addAction('')
reset_canvas_act.setIcon(
QIcon(QPixmap("resource/reset.png").scaled(icon_size, icon_size)))
#重置画布
'''
reset_canvas_act = QAction(
QIcon(QPixmap("resource/reset.png").scaled(icon_size, icon_size)),
'')
menubar.addAction(reset_canvas_act)#错误写法，不会显示图标
'''
```

- **程序图标设置**

  窗口右上角的图标只需要一个setWindowIcon函数即可完成设置，但电脑任务栏中的程序图标却始终无法设置，查资料得知需要加入以下两行，使得任务栏图标与窗口图标一致。

  ```
  import ctypes
  ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID("myappid")
  ```

  但这个方法只适用于windows系统，不确定linux系统下窗口任务栏图标能否成功显示。

- **特殊问题处理**

  - **绘画中途点击其它功能键**

    因为涉及多次点击的绘画功能只有多边形和曲线，因此只有它们会遭遇"被打断"的情况。我本来在鼠标双击事件中写了多边形与曲线的绘画结束函数，现在将其挪到新函数finish_poly_curve中，只要点击任何功能键（如保存，设置画笔，平移，画线等），都会调用该函数结束多边形或曲线的绘制。但在它们正常绘制完毕后再点击功能键，不应该调用finish函数，因此需要判断此时是否有正在绘制的图形。

  - **旋转角度计算**

    旋转角度采用数学公式计算，在少数情况下会导致出错，显示三角函数范围错误。多次实验后发现可能是电脑计算精度问题，在cos=1.000000,sin=0.000000的情况下，实际上电脑里储存的cos值可能比1更大。即使误差小于10的负6次方，但只要cos大于1，角度弧度的转换就会报错，这也就是问题的来源。解决方案是发现cos值大于1就改回1，修正为正确的数值。

四、参考资料

孙正兴《计算机图形学教程》，机械工业出版社，2006

Bresenham 线算法 - 维基百科

python Qt学习

QT官方文档

QT文档 2

QT按钮菜单