《计算机图形学》12月报告

2021 年 1 月 22 日

1 综述

- 2020.09.17 完成绘制直线(DDA算法)
- 2020.09.20 完成绘制直线(Bresenham算法)
- 2020.09.21 完成绘制多边形
- 2020.09.21 完成绘制椭圆(中心椭圆生成算法)
- 2020.09.23 完成绘制曲线(Bezier算法)
- 2020.09.27 完成绘制曲线(B-spline算法)
- 2020.10.02 完成图元平移
- 2020.10.06 完成图元旋转、图元缩放
- 2020.10.11 完成对线段裁剪(Cohen-Sutherland算法)
- 2020.10.14 完成对线段裁剪(Liang-Barsky算法)
- 2020.10.27 完成gui线段绘制
- 2020.11.26 完成gui多边形绘制
- 2020.11.28 完成gui设置画笔
- 2020.11.29 完成gui重置画布
- 2020.12.02 完成gui椭圆绘制
- 2020.12.04 完成gui曲线绘制
- 2020.12.05 完成gui平移
- 2020.12.06 完成gui旋转
- 2020.12.06 完成gui缩放
- 2020.12.06 完成gui裁剪(Cohen-Sutherland算法)
- 2020.12.06 完成gui裁剪(Liang-Barsky算法)

2 算法介绍

2.1 绘制直线

2.1.1 DDA算法

算法原理:通过坐标轴上以单位间隔取样($\Delta x = 1$或$\Delta y = 1$)，因为取单位间隔，所以显见对应的$\Delta y = m, -m$或$\Delta x = \frac{1}{m}, -\frac{1}{m}$(m为斜率)，当起始点在左侧取正；起始点在右侧取负。

算法改进:通过四舍五入可以使绘制的直线更符合原直线函数

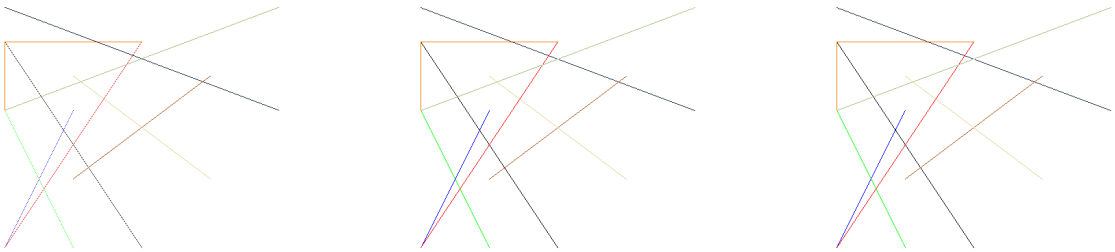
2.1.2 Bresenham算法

算法原理:通过坐标轴上以单位间隔取样($\Delta x = 1$或$\Delta y = 1$)，不失一般性取$0 < m < 1$，对于已经确定的点(x_k, y_k)下一个点有两个像素位置可选分别为y_k和$y_k + 1$；设x_k对应的真实值为y,y到y_k的距离为d_1,y到$y_k + 1$的距离为d_2。判断d_1和d_2值大小，若$d_1 > d_2$,则y离d_2更近，取$y_{k+1} = y_k + 1$,否则取$y_{k+1} = y_k$。m为其他情况同理。

2.1.3 对比分析

DDA设计大量浮点操作比起Bresenham更耗时；Bresenham可以使用并行算法；DDA连续大量浮点整数运算以及取整运算会导致所计算的像素位置偏离实际线段；总的来说Bresenham算法比DDA算法更准确更高效。

三种算法生成图像(从左到右分别是Naive、DDA、Bresenham算法):



2.2 绘制椭圆

2.2.1 中心椭圆生成算法

算法原理:算法类似于Bresenham算法，不失一般性，第一象限椭圆切线斜率绝对值为1时，将第一象限上半部分设为区域1(斜率小于等于1)，下半部分设为区域2(斜率大于1)；之所以这样分割，原理同Bresenham算法。根据参数方程定义椭圆函数为

$$f_{ellipse}(x, y) = r_y^2 * x^2 + r_x^2 * y^2 - r_x^2 * r_y^2 \tag{1}$$

在区域1中取x方向单位步长，再通过决策函数判断真实值与两候选像素之间那个位置更近，更新对应的y值，区域2同理；又因为椭圆四个象限轴对称，可以通过改变对应的符

号补全其余象限

2.3 绘制曲线

2.3.1 Bezier算法

算法原理:从Bezier曲线的定义出发：

$$p(t)=\sum_{i=0}^{n}P_iB_{i,n}(t)\qquad t\in[\mathbf{0},\mathbf{1}]$$

其中$P(i)$为控制顶点，n表示有$n+1$个控制顶点，$B_{i,n}(t)$为Bernstein基函数;其中$B_{i,n}=C_n^it^i(1-t)^{n-i}$；其中由Bezier提出的原基函数可以化简成Bernstein基函数。

t的选取尤为重要，t从0选取到1才能绘制出整个图像；t的选取方法有以下几种：均匀参数化、累加弦长参数化，向心参数化法；实现算法选用均匀参数化方法，该方法为节点在参数轴均匀分布，比如：0，$\frac{1}{100}$，$\frac{2}{100}$...，1

Bezier曲线生成函数如果使用上式的定义函数计算较为耗时，所以Bezier曲线生成函数使用 Casteljau递推算法生成曲线,递推式为：

$$P_i^r=\begin{cases}p_i & r=0;\\ (1-t)P_i^{r-1}+tP_{i+1}^{r-1} & r=1,...,n\quad i=0,...,n-r.\end{cases}\tag{2}$$

举例：不失一般性，假设绘制二次Bezier曲线，当$t=\frac{1}{3}$时，设有P_0,P_1,P_2三个点，在线段P_0P_1三分之一处画出P_0^1；在线段P_1P_2三分之一处画出P_1^1；连接两点$P_0^1P_1^1$，在线段$P_0^1P_1^1$三分之一处画出P_0^2;将整个t值都取一遍即可画出曲线图像.其他曲线图像同理

2.3.2 B-spline算法

算法原理：B-spline算法的定义为：

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

其中P_i是特征多边形的顶点；$B_{i,k}$称为k阶（k-1次）基函数，B-spline算法阶数是次数加1，这是和Bezier算法的一个不同之处；定义域的解释之后会给出，先给出基函数算法。

B-spline基函数的求出算法应用最广泛的是deBoor-cox递推算法：

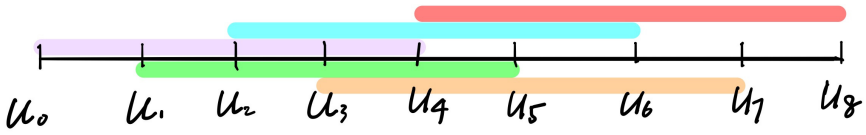
$$B_{i,k}(u)=\frac{u-u_i}{u_{i+k-1}-u_i}*B_{i,k-1}u+\frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}*B_{i+1,k+1}(u)$$

$$B_{i,1}(u)=\begin{cases}1 & u_i<u<u_{i+1};\\ 0 & Otherwise.\end{cases}$$

规定0/0=0

B-spline曲线的定义域为$u\in[u_{k-1},u_{n+1}]$。设U为所有节点矢量的集合，显见节点表个数为$n+k+1$个。举例说明，当$n=4,k=4$时，有$U=\{u_0,u_1,u_2,u_3,u_4,u_5,u_6,u_7,u_8\}$。

第一项为$P_0B_{0,4}(u)$，由deBoor-cox算法可知，其涉及到u_0到u_4五个点；第二项同理，设计到u_1到u_5五个点；其余同理；所以可以画出区间对应坐标轴：



区间合法所需要的条件为：区间内必须有足够基函数与顶点对应，也即区间中基函数覆盖较多的区间才是一个合法区间。所以上例中对应的合法区间为$u \in [u_3, u_5]$也就是$u \in [u_{k-1}, u_{n+1}]$

均匀B样条曲线的定义：当节点沿参数轴均匀等距分布，即$u_{i+1} - u_i = C > 0$时，为均匀B样条函数，比如：$\{0, 1, 2, 3, 4, 5, 6\}, \{0, 0.2, 0.4, 0.6, 0.8, 1\}$

B-spline基函数$(B_{i,k})$的递推公式计算需要用到这里的思想。这里还使用上面当$n = 4, k = 4$的例子，从上面的定义可知，我们可以把节点集合$U = \{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$写成这样$U = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$也即$u_0 = 0, u_1 = 1, ..., u_8 = 8$，当然U也可以写成其他集合，因为计算的时候是一个比例，所以对计算结果并没有影响，综上就可以简单的算出对应基函数的值。

2.3.3 对比分析

Bezier算法几个不足之处：

1.一旦确定特征多边形，就确定了曲线的阶次

2.Bezier曲线拼接复杂（需要满足几何连续性，参数连续性等）

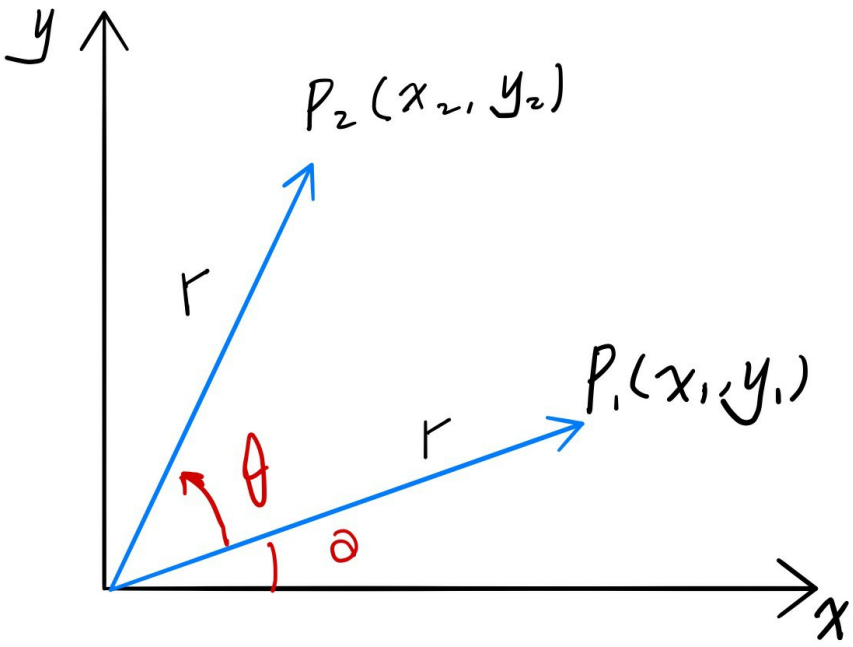
3.Bezier曲线不能作局部修改（只能整体修改）

2.4 图元平移

通过id读取对应数据，在执行对应的绘制操作之前，先将给出的特征点平移，即可做到整个图元的平移

2.5 图元旋转

二维旋转是将物体沿着xy平面内的圆弧路径重定位。旋转角θ取值逆时针为正，顺时针为负



旋转变换的二维表示有如下关系：

$$\begin{cases} x_2 = r * cos(\alpha + \theta) = r * cos\alpha * cos\theta - r * sin\alpha * sin\theta \\ y_2 = r * sin(\alpha + \theta) = r * cos\alpha * sin\theta + r * sin\alpha * cos\theta \end{cases}$$

又因为有$x_1 = r * cos\alpha, y_1 = r * sin\alpha$，所以有

$$\begin{cases} x_2 = r * cos(\alpha + \theta) = x_1 * cos\theta - y_1 * sin\theta \\ y_2 = r * sin(\alpha + \theta) = x_1 * sin\theta + y_1 * cos\theta \end{cases}$$

相对任意参考点的二维变换:如果要对某参考点(x,y)作二维几何变换，比如比例变换，旋转变换等。变换过程如下：

将参考点移至坐标原点

对原点进行二维几何变换

进行反平移，将参考点移回原来的位置

2.6 图元变换

二维比例变换改变物体的尺寸。通过顶点值(x_1, y_1)乘以比例系数S_x和S_y得到变换的坐标(x_2, y_2);

比例系数S_x为在x方向对物体的缩放，S_y在y方向的缩放。比例系数可以赋予任何正数，小于1缩小，大于1则放大；若$S_x = S_y$为等比例变换，若$S_x \neq S_y$则是非均匀比例变换。

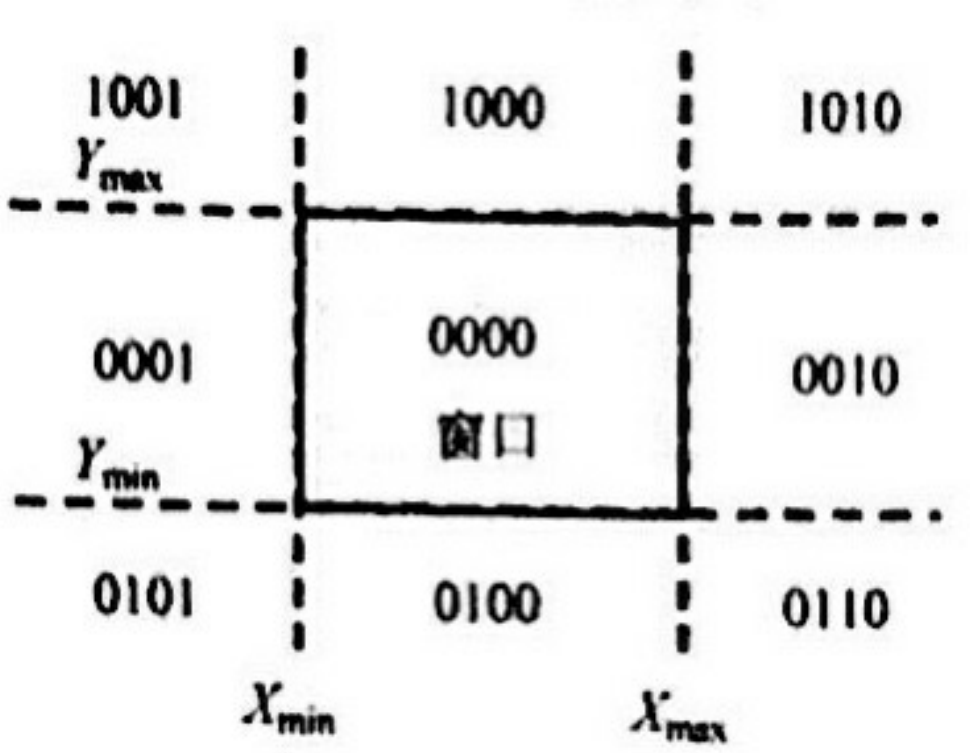
比例变换的二维表示为：

$$\begin{cases} x_2 = x_1 * S_x \\ y_2 = y_1 * S_y \end{cases}$$

2.7 对线段裁剪

2.7.1 Cohen-Sutherland算法

编码算法将整个画布分成9个区域，如下图所示：



根据线段端点所在位置，给每个端点一个四位二进制码（称为区域码）。**四位区域码的4位从左到右依次表示上、下、右、左**。区域码的任何为赋值为1代表端点落在相应的区域中，否则为0。

据区域编码规则可知，在确定区域码每位的值时，可通过比较端点坐标值(x, y)和裁剪边界来确定区域码各位的值：

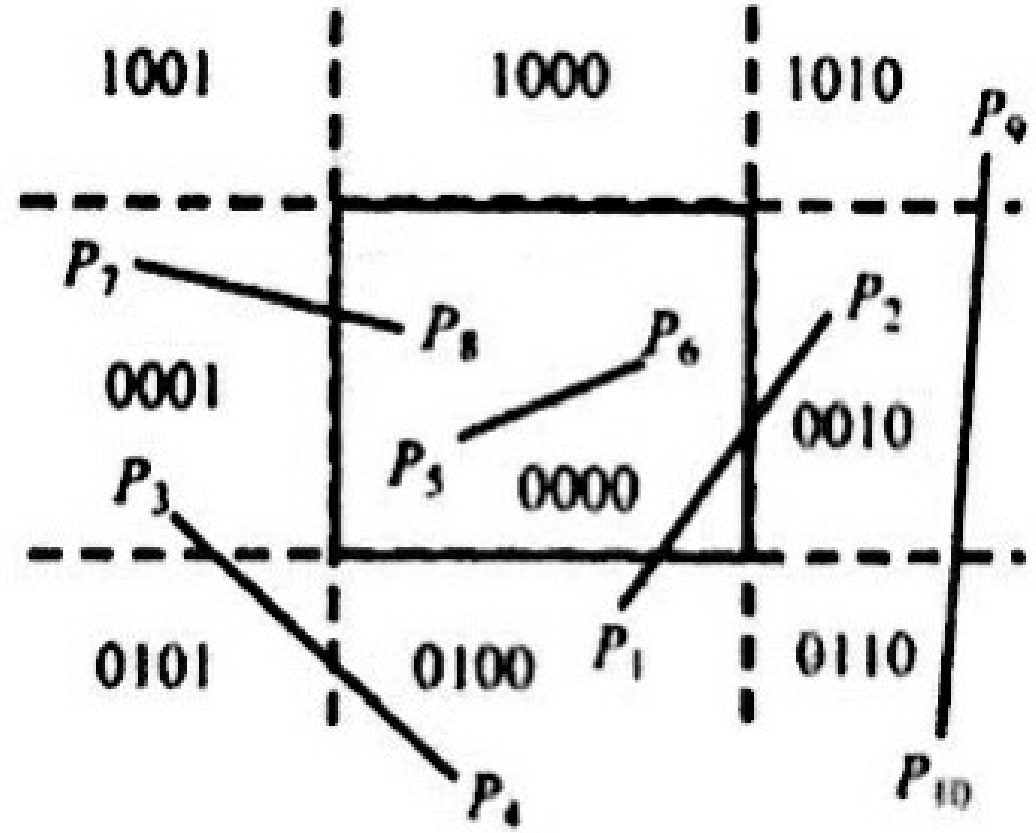
如果$x < x_{min}$，表示该点在裁剪窗口左边界的左边，则第1位置1，否则置0；

如果$x > x_{max}$，表示该点在裁剪窗口右边界的右边，则第2位置1，否则置0；

如果$y < y_{min}$，表示该点在裁剪窗口下边界的下边，则第3位置1，否则置0；

如果$y < y_{max}$，表示该点在裁剪窗口上边界的上边，则第4位置1，否则置0；

根据线段和裁剪窗口的关系可分三种情况处理：



线段完全在裁剪窗口之内 两个端点的区域码都为0000，则该线段完全在裁剪窗口内。如上图：P_5P_6

线段完全在裁剪窗口之外 两个端点的区域码相与的结果不为0000，则该线段完全在裁剪窗口之外。如上图：P_9P_{10}

其他 上图中P_1P_2, P_3P_4, P_7P_8都是此类问题；P_1P_2, P_7P_8显见属于一半落在窗口内一半落在窗口外，需要进行求交运算；而P_3P_4虽然完全落在窗口外但是条件不被第二种情况所适用也需要进行求交运算

求交过程：首先对线段外端点（落在窗口外的点）与一条裁剪边界比较来确定需要裁剪多少线段；然后，将线段的剩下部分与其他裁剪边界对比，直到该直线完全落在窗口内或者被舍弃。

实际算法实现只有在检测到区域码的某位为1时，才把线段和对应裁剪窗口进行求交运算。

2.7.2 Liang-Barsky算法

算法思想

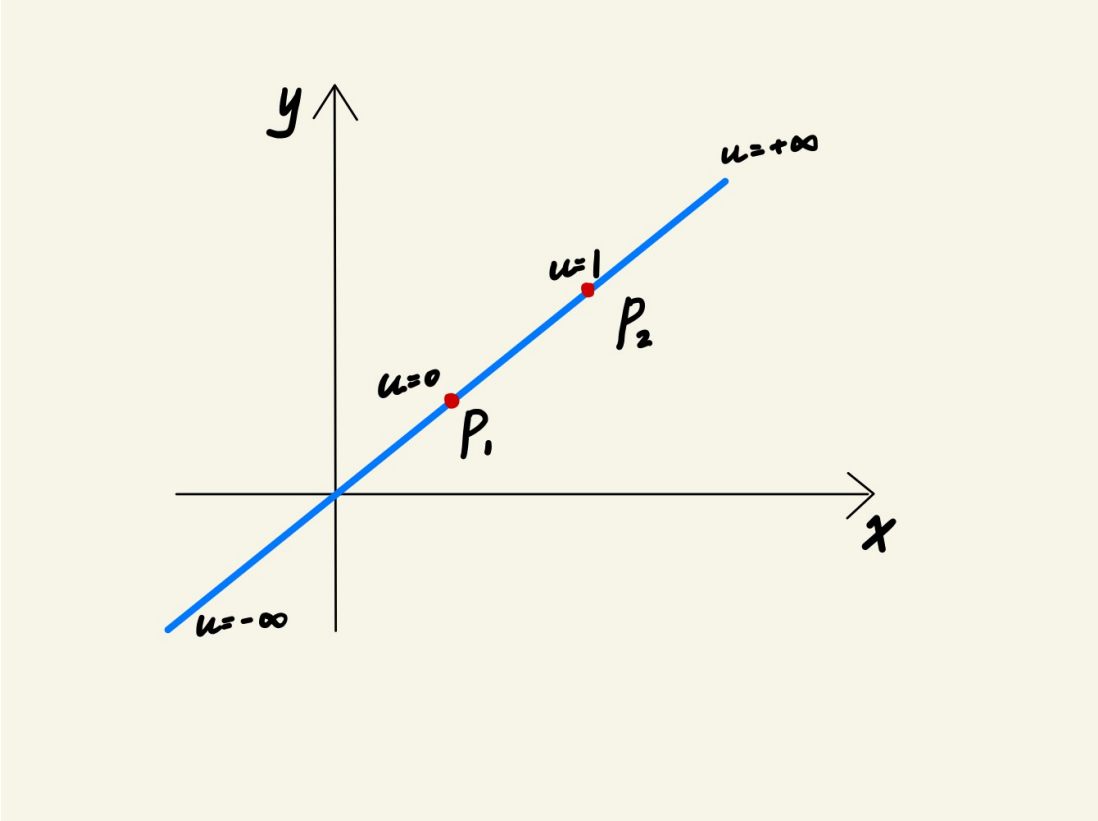
Liang-Barsky算法的主要思想有两部分：

- 用参数方程表示直线
- 将待裁剪直线看作是一个有方向的线

用参数方程表示直线

算法背景中提到Liang-Barsky算法的解决思路是，将裁剪线段和裁剪窗口看为点集裁剪的结果是两个点集的交集。**那么裁剪线段如何转换点集呢？很显然用参数方程来表示直线。**

设待裁剪线段为P_1P_2，其中$P_1 = (x_1, y_1), P_2 = (x_2, y_2)$，用参数关系u表示有下图关系：



显见有如下关系：

$$\begin{cases} x = x_1 + u * (x2 - x1) = x_1 + u * \Delta x & 0 \le u \le 1; \\ y = y_1 + u * (y2 - y1) = y_1 + u * \Delta y & 0 \le u \le 1. \end{cases}$$

当$u = 0$时，$x = x_1, y = y_1$也就是P_1；当$u = 1$时，$x = x_2, y = y_2$也就是P_2；当$u = 0.5$时，也就是该直线中点位置。

由上面三种情况可以很容易归纳出该图像的几何意义：也就是u值即可表示要裁剪线段的多少

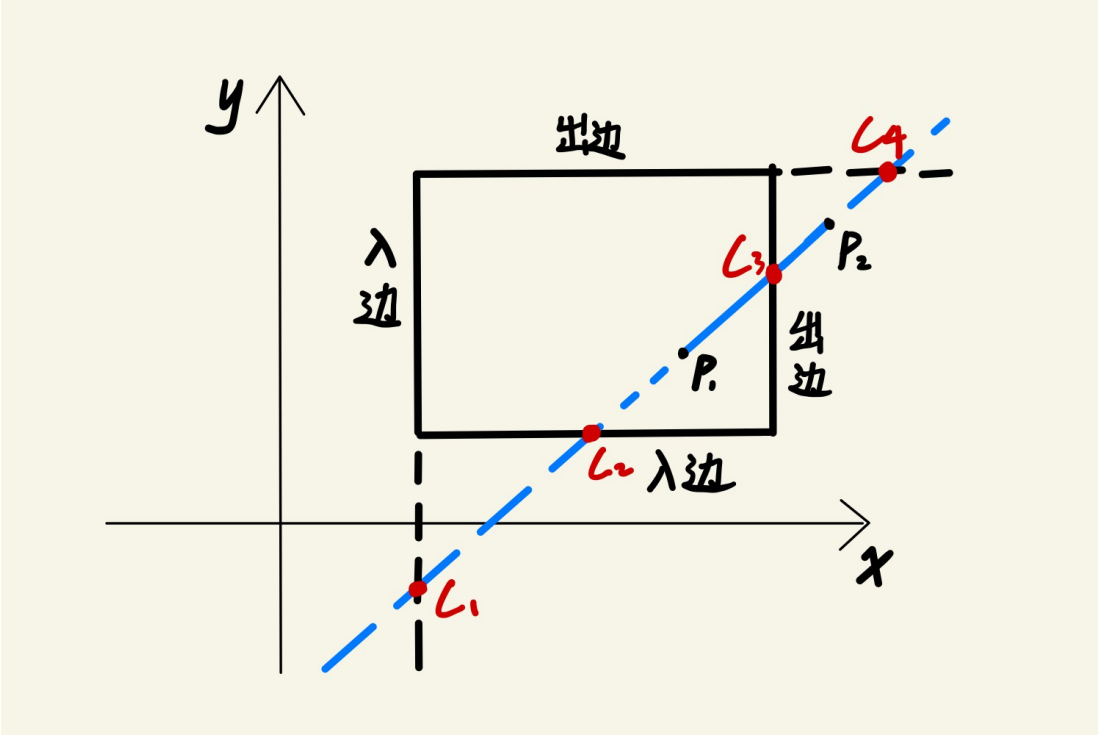
将裁剪直线看作是一个有方向的线

从上面我们知道u的取值可以决定线段要裁剪的多少，那么u到底如何取值变成了现在的首要目标？

我们将四个窗口的交边分别定义成两类边：入边和出边

- 入边：指从裁剪窗口之外进入到裁剪窗口方向的边
- 出边：指从裁剪窗口之内延伸到窗口之外的边

待裁剪线段和裁剪窗口必定会有四个交点（包括与裁剪窗口延长线的交点）分别设四个交点分别为c_1, c_2, c_3, c_4。设待裁剪直线为P_1P_2。则有下图：



显见要裁剪线段为P_1和c_3所夹线段，所以u的选取就要从P_1, c_3所对应u_1, u_2入手，则显见有如下关系式：

$u_1 = max(c_1, c_2, P_1), u_1$是两个入边和$P_1$对应u值的最小值 $u_2 = min(c_3, P_2, c_4), u_2$是两个出边和$P_2$对应u值的最大值

u_1, u_2的值要满足$u_1 < u_2$

只要求出u_1, u_2就能算出裁剪线段，但是要求出u_1, u_2的话，就又出现了两个新的问题：

- 如何算出四个交点c_1, c_2, c_3, c_4所对应的u值
- 如何确定哪两个边是出边，哪两个边是入边

四个交点对应的u值

在上面用参数方程表示直线章节中，我们提出了：

$$\begin{cases} x = x_1 + u * (x2 - x1) = x_1 + u * \Delta x & 0 \le u \le 1; \\ y = y_1 + u * (y2 - y1) = y_1 + u * \Delta y & 0 \le u \le 1. \end{cases}$$

我们不妨先考虑下，在u为何值时，(x, y)位于裁剪窗口之内？我们设裁剪窗口的上边界为y_{max}，下边界为y_{min}，左边界为x_{min}，右边界为x_{max}，结合上式有：

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

可以看出当

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

时，为裁剪直线和四个边界的交点值，所以我们可以很轻松的算出四个对应的u值，此处不在赘述。

出入边的确定

上面我们只提到了不等式的四个特殊情况，不失一般性，这里我们写出不等式的所有情况：

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

可化简为：

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

上面四种情况可以归纳成

$$u * p_k \leq q_k, k = 1, 2, 3, 4$$

使用穷举法可知：

- 当$p_k < 0$时，线段从裁剪边界延长线的外部延伸到内部，也就是入边 - 当$p_k > 0$时，线段从裁剪边界延长线的内部延伸到外部，也就是出边

显见，当$p_k = 0$时，且$q_k < 0$，则线段完全在边界外；若$q_k \geq 0$，则线段完全在边界内

3 系统介绍

3.1 gui设置画笔

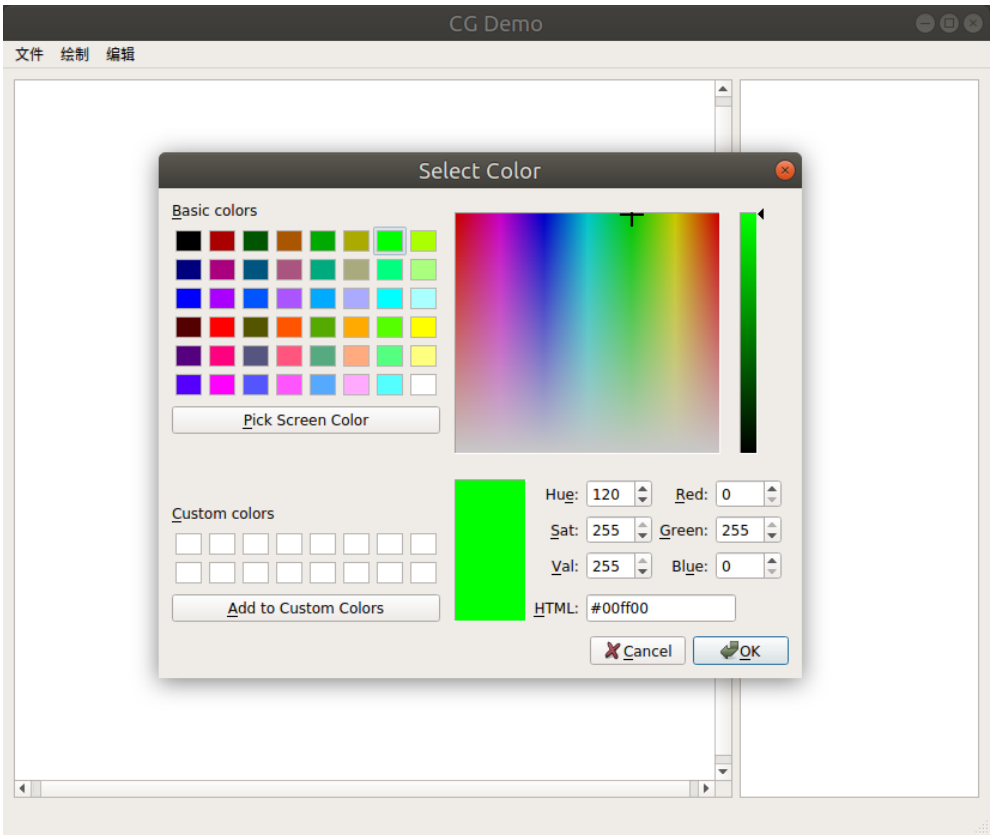
需要实现的部分：

- set_pen_action函数

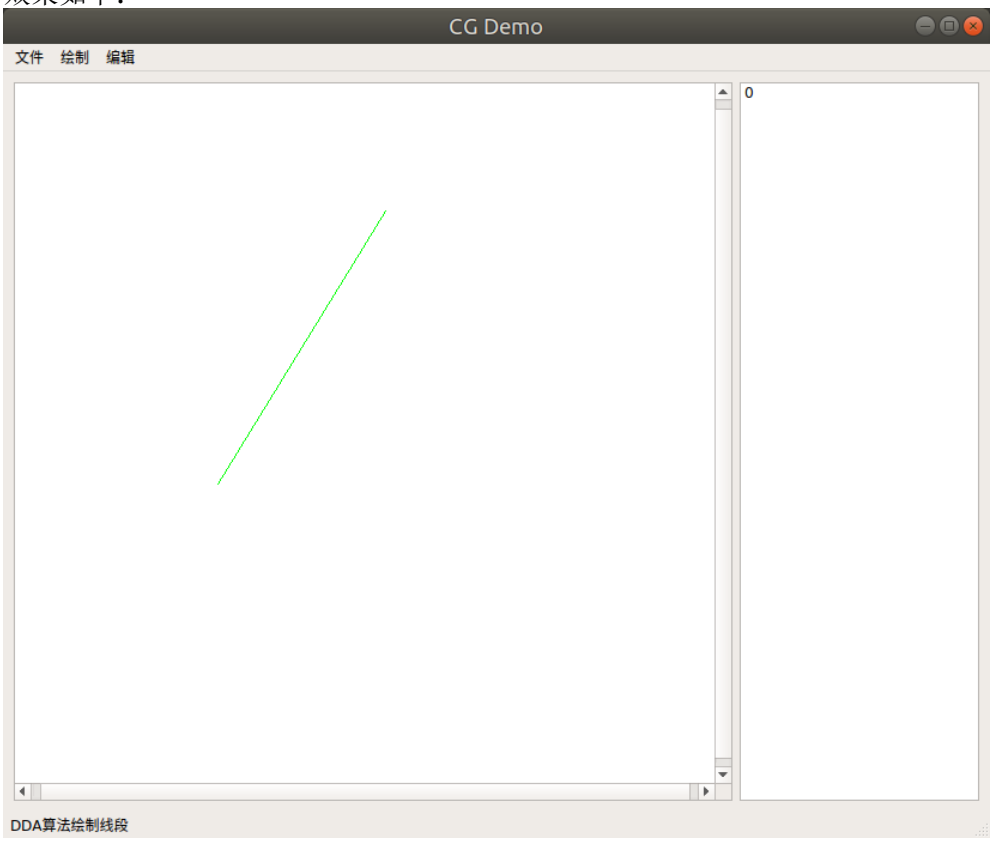
- MainWindow中的set_pen_action函数的设置的color参数传到MyItem中完成颜色的设置

set_pen_action通过QColorDialog实现，并在MainWindow类中设置setcolor参数。再通过MyCanvas类中的main_window参数将setcolor传入到MyItem类中，在MyItem中将画笔颜色设置成setcolor

设置画笔颜色：


CG Demo
文件 绘制 编辑
Select Color
Basic colors
Pick Screen Color
Custom colors
Add to Custom Colors
Hue: 120
Sat: 255
Val: 255
Red: 0
Green: 255
Blue: 0
HTML: #00ff00
Cancel
OK

效果如下：


CG Demo
文件 绘制 编辑
0
DDA算法绘制线段

3.2 gui重置画布

需要实现的内容：

- reset_canvas_action函数
- clearItem函数

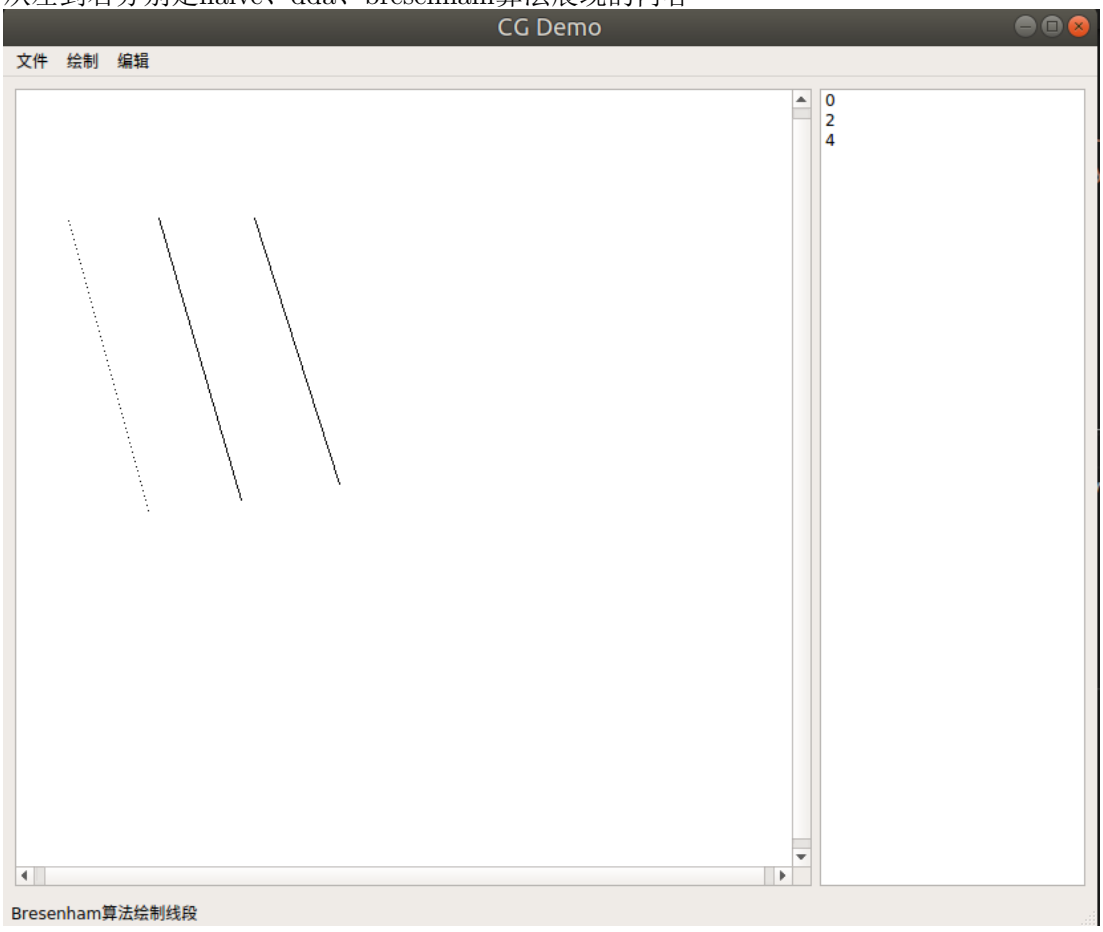
通过QDialog获取新设置的高和宽；再利用clearItem函数将所有画好的图形都删掉，并且将各参数重置为初始值；再将画布高和宽置为Dialog中得到的高和宽即可

3.3 gui绘制直线

给出的代码中已经完成了start_draw_line函数，mousePressEvent、mouseMoveEvent、mouseReleaseEvent、paint、boudingRect的line部分，其余部分需要（如：椭圆、曲线等）需要自己完成

需要完成的部分：连接dda和bresenham算法的槽函数，以及补充line_dda_action和line_bresenham_action函数调用algorithm模块中的dda算法和bresenham算法，即可完成绘制直线部分

从左到右分别是naive、dda、bresenham算法展现的内容



3.4 gui绘制多边形

与直线部分的代码相似，需要自己实现的函数及已有函数补充功能有：

- start_draw_polygon

- mousePressEvent的polygon分支
- mouseMoveEvent的polygon分支
- mouseReleaseEvent的polygon分支
- paint的polygon分支
- boundingRect的polygon分支
- init中连接槽函数

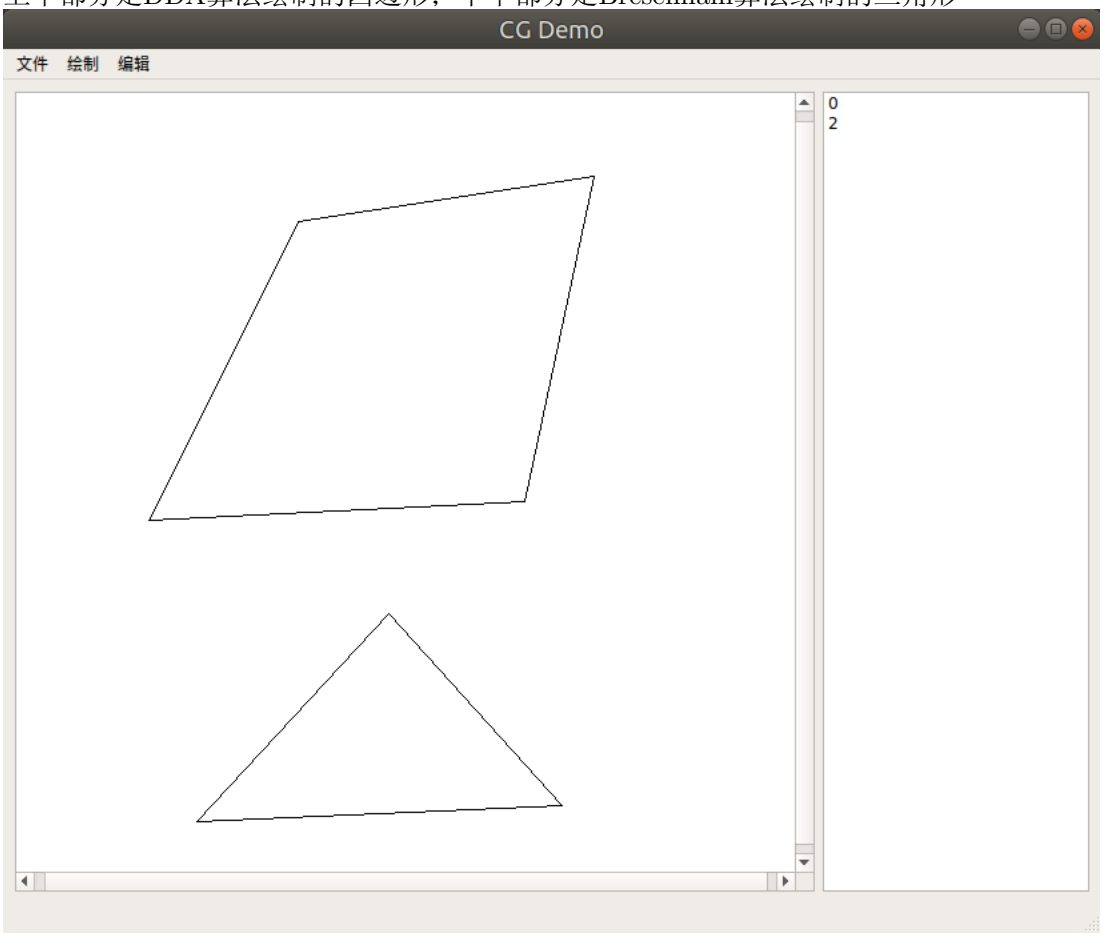
除mousePressEvent、mouseMoveEvent、mouseReleaseEvent外，其余内容和line相似，不做赘述。

mousePressEvent：实现的时候要分情况讨论，在最初没有点在画布的时候，要将两点看作是一条直线，添加到temp_item中；在temp_item中有其他点的时候，还需要进一步分情况讨论。需要判断新加入的点是否和第一个点离的很近，如果离的很近（代码实现是判断两点的距离之差的绝对值是否小于10）有可能是多边形首尾连接需要该图像保存下来，此处还定义了新的变量polygonCompleted来辅助mouseReleaseEvent的实现。

mouseMoveEvent：只需要在p_list的尾部加入鼠标指向点即可

mouseReleaseEvent：利用mousePressEvent中提到的polygonCompleted来区分多边形是否完成。如果没有完成，继续向p_list尾部中加入点；如果polygonCompleted显示完成则不需要任何操作，将temp_item重新置为None即可

上半部分是DDA算法绘制的四边形；下半部分是Bresenham算法绘制的三角形



3.5 gui绘制椭圆

与直线部分的代码实现几乎一致，需要自己实现的函数及已有函数补充功能有：

- start_draw_ellipse
- mousePressEvent的ellipse分支
- mouseMoveEvent的ellipse分支
- mouseReleaseEvent的ellipse分支
- paint的ellipse分支
- boundingRect的ellipse分支
- init中连接槽函数

实现和直线基本一致，仅有几处关键词不同

3.6 gui绘制椭圆

与多边形部分的代码实现相似，需要自己实现的函数及已有函数补充功能有：

- start_draw_curve
- mousePressEvent的curve分支
- mouseMoveEvent的curve分支
- mouseReleaseEvent的curve分支
- paint的curve分支
- boundingRect的curve分支
- init中连接槽函数

与多边形不同的是，结束条件为记录的点的数量，当达到指定数量结束一个绘制

3.7 gui平移

需要自己实现的函数及已有函数补充功能有：

- start_translate
- mousePressEvent的translate分支
- mouseMoveEvent的translate分支
- mouseReleaseEvent的translate分支
- init中连接槽函数

在start_translate将参数初始化为被选中的图元；在mousePressEvent中记录图形移动的初始点；在mouseMoveEvent利用目前鼠标所在位置以及mousePressEvent中记录的初始点计算出x轴y轴的位移，调用algorightm中的translate方法将图元移动；在mouseReleaseEvent中将位移后的图元保存下来即可完成平移功能

3.8 gui旋转

需要自己实现的函数及已有函数补充功能有：

- start_rotate
- init中连接槽函数

在start_rotate获取中心点及旋转角度来进行变换

3.9 gui缩放

与旋转部分的实现类似，需要自己实现的函数及已有函数补充功能有：

- start_scale
- init中连接槽函数

在start_scale获取中心点及缩放倍数来进行变换

3.10 gui裁剪

与平移部分实现类似，需要自己实现的函数及已有函数补充功能有：

- start_clip
- mousePressEvent的clip分支
- mouseMoveEvent的clip分支
- mouseReleaseEvent的clip分支
- init中连接槽函数

在start_clip将参数初始化为被选中的图元；在mousePressEvent中记录图形裁剪的初始点；在mouseMoveEvent利用目前鼠标所在位置以及记录裁剪的终点，调用algorightm中的clip方法将图元移动；在mouseReleaseEvent中将位移后的图元保存下来即可完成平移功能

4 总结

4.1 9月完成情况

- 完成绘制直线(DDA算法)
- 完成绘制直线(Bresenham算法)
- 完成绘制多边形
- 完成绘制椭圆(中心椭圆生成算法)
- 完成绘制曲线(Bezier算法)
- 完成绘制曲线(B-spline算法)

4.2 10月完成情况

- 图元平移
- 图元旋转、图元缩放
- 对线段裁剪(Cohen-Sutherland算法)
- 对线段裁剪(Liang-Barsky算法)
- gui线段绘制

4.3 11月完成情况

- gui多边形绘制
- gui设置画笔
- gui重置画布

4.4 12月完成情况

- gui椭圆绘制
- gui曲线绘制
- gui平移
- gui旋转
- gui缩放
- gui线段裁剪(Cohen-Sutherland算法)
- gui线段裁剪(Liang-Barsky算法)