

# 《计算机图形学》6月报告

161240005 陈勇虎

2019 年 6 月 15 日

## 1 综述

基于Qt的GUI开发框架，使用C++语言实现的一个图形学系统，该系统可以通过读取命令行中的命令，包括通过命令读取存储了指令序列的文本文件，调用算法实现图元的生成和变换等功能。同时，也支持鼠标交互的方式进行操作。

## 2 算法介绍

### 2.1 直线绘制算法

#### 2.1.1 数字差分分析(DDA)算法

##### 1.原理介绍

数字差分分析方法是利用计算两个坐标方向的差分来确定线段显示的屏幕像素位置的线段扫描转换算法，也可以看成是解直线的微分方程式,即:

$$\frac{dy}{dx} = \text{常数} \quad \text{或} \quad \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

其有限差分近似解为:

$$y_{i+1} = y_i + \Delta y$$
$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x$$

这里 $x_1, y_1$ 和 $x_2, y_2$ 是直线的端点,简单的DDA将会选择 $\Delta x$ 或 $\Delta y$ 中较大的一个.

##### 2.理解分析

具体实现代码可查看canvas.cpp文件,函数实现为:

```
void ReceiveDrawLine(int id,float x1,float y1,float x2,float y2,QString algorithm)
```

DDA方法计算像素位置将会比直接使用直线方程要快，这是因为它利用光栅特性消除了直线方程中的乘法，而在x, y方向使用合适的增量来逐步推出像素的位置，但是浮点增量的连续迭加重取整误差的累积会使长线段所计算的像素位置会偏离实际线段，而且取整操作和浮点运算十分耗时.

##### 3.性能测试

绘制端点为 $x_1 = (138, 141), x_2 = (281, 319)$ 的一条直线( $\Delta x < \Delta y = 1$ )

Table 1: DDA数据测试

y	x	real x	error
141	138	138	0
142	139	138.803	-0.196625
143	140	139.607	-0.39325
144	140	140.41	0.410126
145	141	141.214	0.213501
146	142	142.017	0.0168762
147	143	142.82	-0.179749
...	...	...	...
274	245	244.849	-0.151093
275	246	245.652	-0.347717
276	246	246.456	0.455658
...	...	...	...
313	276	276.181	0.180542
314	277	276.984	-0.0160828
315	278	277.787	-0.212708
316	279	278.591	-0.409332
317	279	279.394	0.394043
318	280	280.197	0.197418

表格中可以看出算法的误差，在部分点的计算中很大，具体可以运行附件中的DDA.cpp 查看.

## 2.1.2 Bresenham算法

### 1.原理分析

假定直线的斜率在0,1之间，且 $x_2 > x_1$ ，设在第i步中已经确定了最接近直线的第i个像素点 $(x_i, y_i)$ ，那么第i+1个像素点是 $(x_i + 1, y_i)$ 和 $(x_i + 1, y_i + 1)$ 中的一个。在 $x = x_i + 1$ 处直线上的点y值时 $y = m(x + 1) + b$ ，该点到点 $(x_i + 1, y_i)$ 和点 $(x_i + 1, y_i + 1)$  的距离分别是 $d_1$ 和 $d_2$ ：

$$d_1 = y - y_i = m(x + 1) + b - y_i$$

$$d_2 = (y_i + 1) - y = (y_i + 1) - m(x + 1) - b$$

这两个距离的差是：

$$d_1 - d_2 = 2m(x_i + 1) - 2y_i + 2b - 1$$

若差值为正，则 $d_1 > d_2$ ，下一个像素点应取 $(x_i + 1, y_i + 1)$ ；若此差值为负，则 $d_1 < d_2$ ，下一个像素点应取 $(x_i + 1, y_i)$ ；若此差值为零，则 $d_1 = d_2$ ，下一个像素点可取上述两个像素点的任意一个。

引入一个判别量 $p_i$ ,以方便对 $d_1 - d_2$ 的计算:

$$p_i = \Delta x(d_1 - d_2) = 2\Delta y x_i - 2\Delta x y_i + c$$

这里 $\Delta x = x_2 - x_1, \Delta y = y_2 - y_1; c = 2\Delta y + \Delta x(2b - 1), \Delta x > 0$ ,故 $p_i$ 与 $d_1 - d_2$ 同号.  
另一方面,

$$p_{i+1} = 2\Delta y \cdot x_{i+1} - 2\Delta x \cdot y_{i+1} + c$$

$$p_{i+1} - p_i = 2\Delta y - 2\Delta x(y_{i+1} - y_i)$$

$$p_{i+1} = p_i + 2(\Delta y - \Delta x) \quad p_i \geq 0$$

$$p_{i+1} = p_i + 2\Delta y \quad p_i < 0$$

$$p_1 = 2\Delta y - \Delta x$$

## 2.理解分析

算法中只有整数运算和乘2运算,对二进制数乘2可以利用移位实现,因此该算法运行快且易于硬件实现。

## 3.性能测试

绘制端点为 $x_1 = (138, 141), x_2 = (281, 319)$ 的一条直线

Table 2: Bresenham测试

y	x	real x	error
141	138	138	0
142	139	138.803	0.196625
143	140	139.607	0.393265
144	140	140.41	-0.41011
145	141	141.213	-0.213486
146	142	142.017	-0.016861
147	143	142.82	0.179779
148	144	143.624	0.376404
...	...	...	...
311	275	274.573	0.426971
312	275	275.376	-0.376404
313	276	276.18	-0.179779
314	277	276.983	0.0168457
315	278	277.786	0.213501
316	279	278.59	0.410095
317	279	279.393	-0.39325
318	280	280.197	-0.196625

## 2.2 多边形绘制算法

### 原理分析

多边形可以看成很多条直线的组合,因此可以找到每次绘制的多边形的某一条边,利用上面介绍过直线绘制方法进行绘制即可。

## 2.3 椭圆生成算法

### 1.原理分析

由中点圆法我们很容易推得椭圆的扫描转换算法,我们可以设椭圆圆心在坐标原点的一个标准椭圆,其方程为:

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

(1) 对于椭圆上的点,有  $F(x, y) = 0$

(2) 对于椭圆外的点,有  $F(x, y) > 0$

(3) 对于椭圆内的点,有  $F(x, y) < 0$

以弧上斜率为-1的点作为分界将第一象限椭圆分为上下两个部分。椭圆上一点(x,y)处的法向量为:

$$N(x, y) = \frac{\partial F}{\partial x}i + \frac{\partial F}{\partial y}j$$

这里(i,j)为x轴,y轴的单位向量.在分点的上部分,法向量的y分量更大,而在下部分,法向量的x分量更大。因此,若在当前中点,法向量( $2b^2(x_p + 1), 2a^2(y_p - 0.5)$ )的y分量比x分量更大,即:

$$b^2(x_p + 1) < a^2(y_p - 0.5)$$

而在下一个中点,不等号改变方向,则说明椭圆弧从上部分转为下部分。

设当前点( $x_i, y_i$ )已逼近理想椭圆,根据其逼近规律,对于上部分,下一点可能是( $x_i + 1, y_i$ )点,也可能是( $x_i + 1, y_i - 1$ )点,那么对于下一对候选像素的中点是( $x_i + 1, y_i - 0.5$ ),因此判别式为:

$$d_1 = F(x_i + 1, y_i - 0.5) = b^2(x_i + 1)^2 + a^2(y_i - 0.5)^2 - a^2b^2$$

当 $d_1 \geq 0$ 时,则中点( $x_i + 1, y_i - 0.5$ )位于椭圆之外或椭圆上,那么( $x_i + 1, y_i - 1$ )点距离理想椭圆弧近,故选( $x_i + 1, y_i - 1$ )点逼近该理想椭圆。进一步计算,如果确定了( $x_i + 1, y_i - 1$ )点,那么( $x_i + 1, y_i - 1$ )的下一对候选像素就是( $x_i + 2, y_i - 1$ )和( $x_i + 2, y_i - 2$ ),其中点为( $x_i + 2, y_i - 1.5$ )。因此中点( $x_i + 2, y_i - 1.5$ )的判别式为:

$$\begin{aligned} d'_i &= F(x_i + 2, y_i - 1.5) = b^2(x_i + 2)^2 + a^2(y_i - 1.5)^2 - a^2b^2 \\ &= d_1 + b^2(2x_i + 3) + a^2(-2y_i + 3) \end{aligned}$$

当 $d_1 \leq 0$ 时,则中点 $(x_i + 1, y_i - 0.5)$ 位于椭圆之外或椭圆上,那么 $(x_i + 1, y_i)$ 点距离理想椭圆弧近, 故选 $(x_i + 1, y_i)$ 点逼近该理想椭圆。进一步计算, 如果确定了 $(x_i + 1, y_i)$ 点,那么 $(x_i + 1, y_i)$ 的下一对候选像素就是 $(x_i + 2, y_i)$ 和 $(x_i + 2, y_i - 1)$ , 其中点为 $(x_i + 1.5, y_i - 0.5)$ 。因此中点 $(x_i + 1.5, y_i - 0.5)$ 的判别式为:

$$\begin{aligned} d'_i &= F(x_i + 1.5, y_i - 0.5) = b^2(x_i + 1.5)^2 + a^2(y_i - 0.5)^2 - a^2b^2 \\ &= d_1 + b^2(2x_i + 3) \end{aligned}$$

下面确定上部分的判别式 $d_1$ 的初始条件: 由于椭圆弧从 $(0, b)$ 开始画点,所以 $x_0 = 0, y_0 = b$ , 则第一点的中点坐标为 $(1, b - 0.5)$ , 相应判别式为:

$$d_1^0 = F(1, b - 0.5) = b^2 + a^2(-b + 0.25)$$

对于下部分,当前点 $(x_i, y_i)$ 的下一个点可能是 $(x_i, y_i - 1)$ 点, 也可能是 $(x_i + 1, y_i - 1)$ 点, 那么下一对候选像素的中点是 $(x_i + 0.5, y_i - 1)$ 。因此判别式为:

$$d_2 = F(x_i + 0.5, y_i - 1) = b^2(x_i + 0.5)^2 + a^2(y_i - 1)^2 - a^2b^2$$

当 $d_2 \geq 0$ 时,则中点 $(x_i + 0.5, y_i - 1)$ 位于椭圆之外或椭圆上,那么 $(x_i, y_i - 1)$ 点距离理想椭圆弧近, 故选 $(x_i, y_i - 1)$ 点逼近该理想椭圆。如果确定了 $(x_i, y_i - 1)$ 点, 那么 $(x_i, y_i - 1)$ 的下一对候选像素就是 $(x_i, y_i - 2)$ 和 $(x_i + 1, y_i - 2)$ ,其中点为 $(x_i + 0.5, y_i - 2)$ 。因此, 中点 $(x_i + 0.5, y_i - 2)$ 的判别式为:

$$\begin{aligned} d'_2 &= F(x_i + 0.5, y_i - 2) = b^2(x_i + 0.5)^2 + a^2(y_i - 2)^2 - a^2b^2 \\ &= d_2 + a^2(-2y_i + 3) \end{aligned}$$

当 $d_2 < 0$ 时,则中点 $(x_i + 0.5, y_i - 1)$ 位于椭圆之内,那么 $(x_i + 1, y_i - 1)$ 点距离理想椭圆弧近, 故选 $(x_i + 1, y_i - 1)$ 点逼近该理想椭圆。如果确定了 $(x_i + 1, y_i - 1)$ 点, 那么 $(x_i + 1, y_i - 1)$ 的下一对候选像素就是 $(x_i + 1, y_i - 2)$ 和 $(x_i + 2, y_i - 2)$ ,其中点为 $(x_i + 1.5, y_i - 2)$ 。因此, 中点 $(x_i + 1.5, y_i - 2)$ 的判别式为:

$$\begin{aligned} d'_2 &= F(x_i + 1.5, y_i - 2) = b^2(x_i + 1.5)^2 + a^2(y_i - 2)^2 - a^2b^2 \\ &= d_2 + b^2(2x_i + 2) + a^2(-2y_i + 3) \end{aligned}$$

下确定下部分的判别式 $d_2$ 的初始条件:显然, 下部分的起点应从弧上斜率为-1的点开始, 但弧上斜率为-1的点坐标计算复杂且为非整数,可以通过对椭圆方程式两遍微分

$$dF(x, y) = d(b^2x^2 + a^2y^2 - a^2b^2) = 2b^2xdx + 2a^2ydy = 0$$

故

$$\frac{dy}{dx} = -\frac{2b^2x}{2a^2y}$$

令 $\Delta x = 2b^2x, \Delta y = 2a^2y$ , 则有结论:当 $\Delta x = \Delta y$ 时, 此时的x,y坐标对应与弧上斜率为-1的点;当 $\Delta x < \Delta y$ 时,此时x,y坐标对应上部分;当 $\Delta x > \Delta y$ 时,此时的x,y坐标对应下部

分.又因 $2b^2x$ 与 $2a^2y$ 的计算在画上部分时已解决,故只需比较两者的大小就能达到判断所画上部分是否到达或越过弧上斜率为-1的点.

下部分中点判别式初值的处理如下:设前一点仍属于上部分,按上部分的递推公式推算,当前点已越过临界点,因此应把当前点的中点判别式 $d_2^0$ 转换为下部分起点的中点判别式 $d_2^0$ .

$$d_2^0 = (b(x + 0.5))^2 + (a(y - 1))^2 - a^2b^2$$

当上式中的 $d_2^0$ 作为下部分中点判别式的初始值,并运用前面给出的判别式的推算,就能到达(a,0)点处,从而完成四分之一弧的绘制.

## 2.理解分析

可以采用增量法就算判别式来提高计算效率.

## 2.4 曲线生成算法

### 2.4.1 Bezier曲线

曲线的Bezier形式可以让设计者极其明显地感觉到所给条件和输出曲线或者曲面的关系,对交互式设计非常有用.

给出型值点 $P_0, P_1, \dots, P_n$ , 它们所确定的n次Bezier曲线是:

$$P(t) = \sum_{i=0}^n B_{i,n}(t)P_i, 0 \leq t \leq 1$$

式中,基函数 $B_{i,n}(t)$ 是Bernstein多项式:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$$

这里Bezier曲线可以看成n+1个混合函数混合给定的n+1个顶点而产生的,混合函数用Bernstein多项式,所生成曲线是n次多项式.通常n+1个顶点也成为控制点,依次连接各控制点得到的多边形称为控制多边形.

## 1.原理分析

绘制Bezier有几何作图法, 分裂法等.这里简单介绍几何作图法, 也即de Casteljau算法, 它利用了Bezier曲线的分割递推性实现Bezier曲线的绘制.

记点 $P_k, P_{k+1}, \dots, P_{k+n}$ 可以生成的Bezier曲线为 $P_k^n, 0 \leq t \leq 1$

$$P_k^n = \sum_{i=k}^{k+n} B_{i,n}(t)P_i, 0 \leq t \leq 1$$

则下面的递推关系成立:

$$P_0^{(n)}(t) = (1-t)P_0^{n-1}(t) + tP_1^{n-1}(t) \quad (1)$$

因此,由点 $P_0, P_1, \dots, P_n$ 所确定的n次Bezier曲线在点t的值, 可以由点 $P_0, P_1, \dots, P_{n-1}$ 所确定的n-1次Bezier曲线在点t的值, 与由点 $P_1, P_2, \dots, P_n$ 所确定的n-1次Bezier曲线在点t的值,通过(1)式即可求得其线性组合.

$$\text{右端} = (1-t) \sum_{i=0}^{n-1} \binom{n-1}{i} t^i (1-t)^{n-i-1} P_i + t \sum_{i=0}^{n-1} \binom{n-1}{i} t^i (1-t)^{n-i-1} P_{i+1}$$

$$\begin{aligned}
&= (1-t)[(1-t)^{n-1}P_0 + \sum_{i=1}^{n-1} \binom{n-1}{i} t^i (1-t)^{n-i-1} P_i] \\
&\quad + t[\sum_{i=1}^{n-1} \binom{n-1}{i-1} t^{i-1} (1-t)^{n-i} P_i + t^{n-1} P_n] \\
&= (1-t)^n P_0 + \sum_{i=1}^{n-1} \binom{n-1}{i} t^i (1-t)^{n-i} P_i + \sum_{i=1}^{n-1} \binom{n-1}{i-1} t^i (1-t)^{n-i} P_i + t^n P_n \\
&= (1-t)^n P_0 + \sum_{i=1}^{n-1} (\binom{n-1}{i} + \binom{n-1}{i-1}) t^i (1-t)^{n-i} P_i + t^n P_n \\
&= \sum_{i=0}^n \binom{n}{i} t^i (1-t)^{n-i} P_i \\
&= \text{左端}
\end{aligned}$$

从而(1)式可以改写为:

$$P_0^{(n)}(t) = P_0^{(n-1)}(t) + (P_1^{(n-1)}(t) - P_0^{(n-1)}(t))(2)$$

(1)(2)表明了n次Bezier曲线上控制点在t时的值P(t),可以归结为计算两个n-1次Bezier曲线在t时的值的线性组合,这一过程可以继续下去,这就是Bezier曲线几何作图法的原理。

## 2.理解分析

几何作图法的优点就在于直观性强,计算速度快.

## 3.性能测试

假设给定4个型值点是 $P_0 = (1, 1), P_1 = (2, 3), P_2 = (4, 3), P_3 = (3, 1)$ , 则计算结果为:

Table 3: Bezier曲线的计算结果

t	$(1-t)^3$	$3t(1-t)^2$	$3t^2(1-t)$	$t^3$	P(t)
0	1	0	0	0	(1,1)
0.15	0.614	0.325	0.0574	0.0034	(1.5058,1.765)
0.35	0.275	0.444	0.239	0.043	(2.248,2.376)
0.5	0.125	0.375	0.375	0.125	(2.75,2.5)
0.65	0.043	0.239	0.444	0.275	(3.122,2.36)
0.85	0.0034	0.0574	0.325	0.614	(3.248,1.75)
1	0	0	0	1	(3,1)

## 2.4.2 B-spline曲线

(实验中只考虑三阶B样条曲线)

给定n+1个控制点 $P_0, P_1, P_2, \dots, P_n$ ,它们所确定的k阶B样条曲线是:

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

式中,基函数 $N_{i,k}(u)$ 为 $l=k$ 时的 $N_{i,l}(u)$ ,  $N_{i,l}(u)$ 递归定义如下:

$$\begin{cases} N_{i,1}(u) = \begin{cases} 1, & u_i \leq u \leq u_{i+1}, 0 \leq i \leq n+k-l \\ 0, & otherwise \end{cases} \\ N_{i,l}(u) = \frac{u-u_i}{u_{i+l-1}-u_i} N_{i,l-1}(u) + \frac{u_{i+1}-u}{u_{i+l}-u_{i+1}} N_{i+1,l-1}(u), u_i \leq u < u_{i+1}, 2 \leq l \leq k, 0 \leq i \leq n+k-l \end{cases}$$

式中,  $u_0, u_1, \dots, u_{n+k}$ , 是一个非递减的序列, 称为节点,  $(u_0, u_1, \dots, u_{n+k})$  构成节点向量。

$k$ 阶B样条曲线的定义域为  $u \in [u_{k-1}, u_{n+1}]$ , 曲线由  $n+1$  个混合函数描述。

### 1. 原理分析

给定控制顶点  $P_i (i = 0, 1, \dots, n)$ , 阶数  $k$  及节点向量  $(u_0, u_1, \dots, u_{n+k})$  后, 则可以定义一条  $k$ 阶B样条曲线。可以采用de Boor算法的递推公式进行计算绘制, 类似于Bezier曲线上的点的几何作图法。

先将  $u$  固定在区间  $[u_i, u_{i+1}) (k-1 \leq i \leq n)$ , 于是de Boor算法的递推公式如下:

$$\begin{aligned} P(u) &= \sum_{j=0}^n P_j N_{j,k}(u) = \sum_{j=i-k+1}^i P_j N_{j,k}(u) \\ &= \sum_{j=i-k+1}^i P_j \left[ \frac{u - u_j}{u_{j+k-1} - u_j} N_{j,k-1}(u) + \frac{u_{j+k} - u}{u_{j+k} - u_j} N_{j+1,k-1}(u) \right] \\ &= \sum_{j=i-k+1}^i \left[ \frac{u - u_j}{u_{j+k-1} - u_j} P_j + \frac{u_{j+k-1} - u}{u_{j+k-1} - u_j} P_{j-1} \right] N_{j,k-1}(u) \quad u \in [u_i, u_{i+1}) \end{aligned}$$

现令

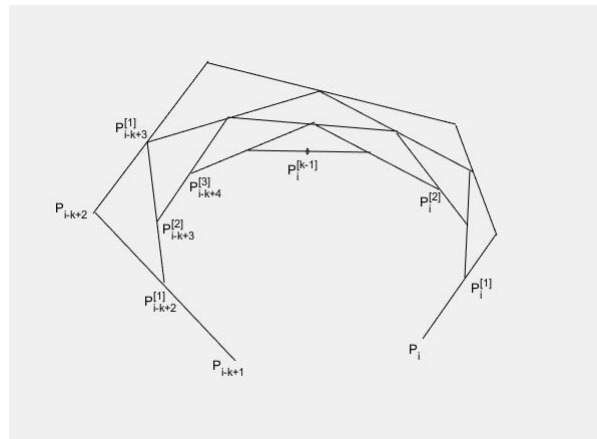
$$P_j^{[r]}(u) = \begin{cases} P_j, r = 0, j = i - k + 1, i - k + 2, \dots, i \\ \frac{u - u_j}{u_{j+k-r} - u_j} P_j^{[r-1]}(u) + \frac{u_{j+k-r} - u}{u_{j+k-r} - u_{j-1}} P_{j-1}^{[r-1]}(u) \\ r = 1, 2, \dots, k-1; j = i - k + r + 1, i - k + r + 2, \dots, i \end{cases}$$

则:

$$\begin{aligned} P(u) &= \sum_{j=i-k+1}^i P_j N_{j,k}(u) = \sum_{j=i-k+2}^i P_j^{[1]}(u) N_{j,k-1}(u) \\ &= \sum_{j=i-k+3}^i P_j^{[2]}(u) N_{j,k-2}(u) \\ &\dots = P_i^{[k-1]}(u) N_{i,1}(u) = P_i^{k-1}(u) \end{aligned}$$

### 2. 理解分析

de Boor算法有着直观的几何意义, 即以线段  $P_i^{[r]} P_{i+1}^{[r]}$  割去角  $P_i^{[r-1]}$ , 从多边形  $P_{i-k+1} P_{i-k+2} \dots P_i$ , 经过  $k-1$  层割角, 最后得到  $P(u)$  上的点  $P_i^{[k-1]}(u)$ , 如下所示:





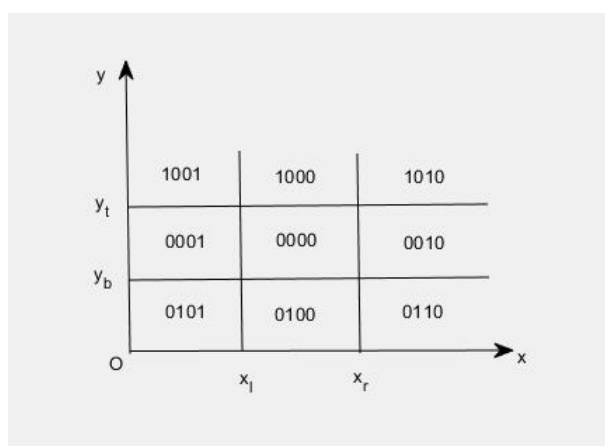
## 2.5 裁剪算法

### 2.5.1 Cohen-Sutherland算法

#### 原理分析

基本思想是:首先判断直线段是否全部在窗口内,是,则保留;不是,则再判断是否完全在窗口之外,如是,则舍弃。如果这两种情况都不属于,则将此线段分割,对分割后的子线段再进行如前判断,直至所有直线段和由直线分割出来的子线段都已经确定了是保留还是舍弃为止。

判断直线段对窗口的位置,可以通过判断直线段端点的位置来进行。用裁剪窗口的4条边界及其延长线把整个平面分成9个区域,然后对这些区域用4位二进制代码进行编码,每一区域中的点采用同一代码。编码规则如下:如果该区域在窗口的上方,则代码的第一位为1;如果该区域在窗口的下方,则代码的第二位1;如果该区域在窗口的右侧,则代码的第三位为1;如果该区域在窗口的左右,则代码的第四位为1。因此区域编码如下:



算法的基本步骤如下:

第一步:编码。设直线段的两个端点为 $P_1(x_1, y_1)$ 和 $P_2(x_2, y_2)$ 。根据上述编码规则,可以求出 $P_1, P_2$ 所在区域的代码 $c_1$ 和 $c_2$ 。

第二步:判别。根据 $c_1$ 和 $c_2$ 的具体值,可以有三种情况:

1) $c_1 = c_2 = 0$ ,这表明两端点全在窗口内,则整个直线段也在窗口内,应该保留,如Cohen-Sutherland例图中的线段AB。

2) $c_1 \times c_2 \neq 0$ ,这里的“ $\times$ ”是逻辑乘,即 $c_1$ 和 $c_2$ 至少有某一位同时为1,表现两端点必定同处于某一边界的同一外侧,则整个直线段全在窗口外,应该舍弃,如cohen-Sutherland例图中的线段CD。

3)如不属于上面两种情况,又可以分为以下三种情况:

①一个端点在内,另一个端点在外,如cohen-Sutherland例图中的EF;

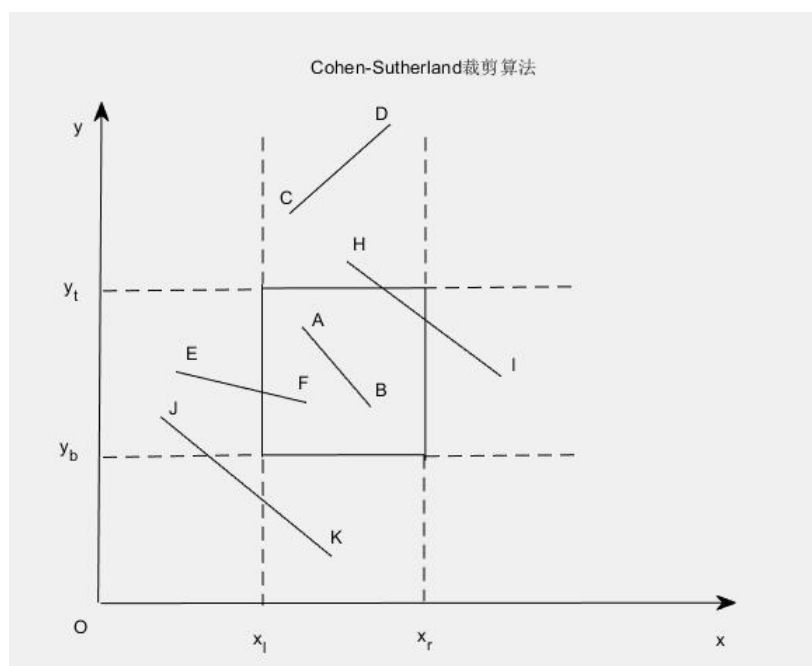
②两个端点在外,但直线段中部跨越窗口,如cohen-Sutherland例图中的HI;

③两个端点在外,且直线段也在外,如cohen-Sutherland例图中的JK;

第三步:求交。对不能确定取舍的直线段,求其与窗口边界及其延长线的交点,从而将直线段分割。求交点时,可以有针对性地与某一确定边界求交。如图中的直线段EF,知E所

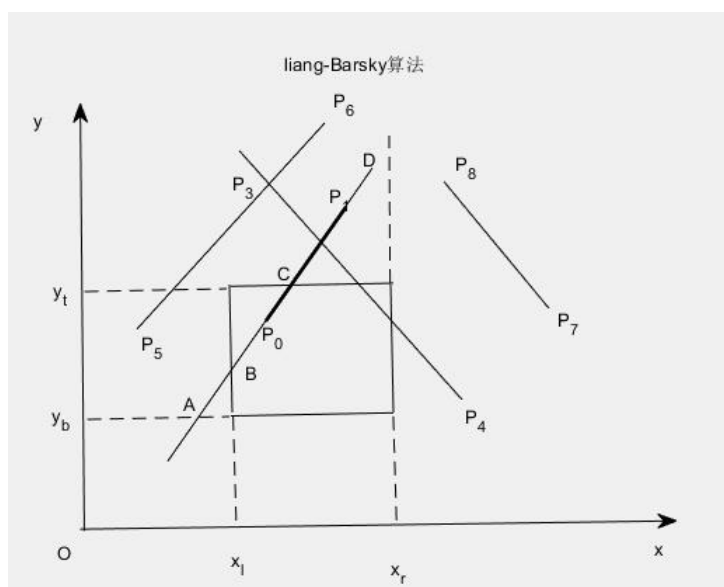
在区域的代码为0001, F所在区域代码为0000, 这表明E在窗口的左侧, 而F不在左侧, 则EF与 $x = x_l$ 必定相交。可求得交点 $E'$ , 从而可舍弃 $EE'$ , 而保留 $E'F$ 。

第四步:对剩下的线段 $E'F$ 重复以上各步。可以验证重复到第三遍的判断为止,这是剩下的直线段或者全在窗口内, 或者全在窗口外, 从而完成直线的裁剪。



## 2.5.2 liang-Barsky算法

原理分析



如上图所示:设要裁剪的直线段为 $P_0P_1$ ,  $P_i$ 的坐标为 $(x_i, y_i), i = 0, 1$ 。  $P_0P_1$ 和窗口边界交于A、B、C和D四个点。算法的基本思想是从A、B和 $P_0$ 中找出最靠近 $P_1$ 的点,在上图中该点是 $P_0$ ,从C、D和 $P_1$ 中找出最靠近 $P_0$ 的点,在上图中该点是C.那么 $P_0C$ 就是 $P_0P_1$ 裁剪后的可见部分。

在具体计算时,可以把 $P_0P_1$ 写成参数方程:

$$x = x_0 + \Delta x \cdot t$$

$$y = y_0 + \Delta y \cdot t$$

式中:  $\Delta x = x_1 - x_0$ ;  $\Delta y = y_1 - y_0$

把窗口边界四条边分成两类, 一类称为始边, 一类称为终边。当  $\Delta x \geq 0$  ( $\Delta y \geq 0$ ) 时, 称  $x = x_l$  ( $y = y_b$ ) 为始边, 称  $x = x_r$  ( $y = y_t$ ) 为终边。当  $\Delta x < 0$  ( $\Delta y < 0$ ) 时, 则称  $x = x_r$  ( $y = y_t$ ) 为始边,  $x = x_l$  ( $y = y_b$ ) 为终边。

求出  $P_0P_1$  和两条始边的交点的参数  $t'_0$  和  $t''_0$ , 令

$$t_0 = \max(t'_0, t''_0, 0)$$

则参数为  $t_0$  的点就是图中的 A、B 和  $P_0$  三条边中最靠近  $P_1$  的点。

求出  $P_0P_1$  和两条终边的交点的参数  $t'_1$  和  $t''_1$ , 令

$$t_1 = \min(t'_1, t''_1, 1)$$

则参数为  $t_1$  的点就是图中的 C、D 和  $P_1$  三条边中最靠近  $P_0$  的点。

当  $t_1 > t_0$  时,  $P_0P_1$  参数方程中参数  $t \in [t_0, t_1]$  的线段就是  $P_0P_1$  的可见部分。当  $t_0 > t_1$  时, 整个直线段为不可见。图中  $P_5P_6, P_7P_8$  即为这种情况。

为了确定终边和始边, 并求出  $P_0P_1$  与它们的交点, 可采用如下方法, 令:

$$Q_l = -\Delta x, \quad D_l = x_0 - x_l$$

$$Q_r = \Delta x, \quad D_r = x_r - x_0$$

$$Q_b = -\Delta y, \quad D_b = y_0 - y_b$$

$$Q_t = \Delta y, \quad D_t = y_t - y_0$$

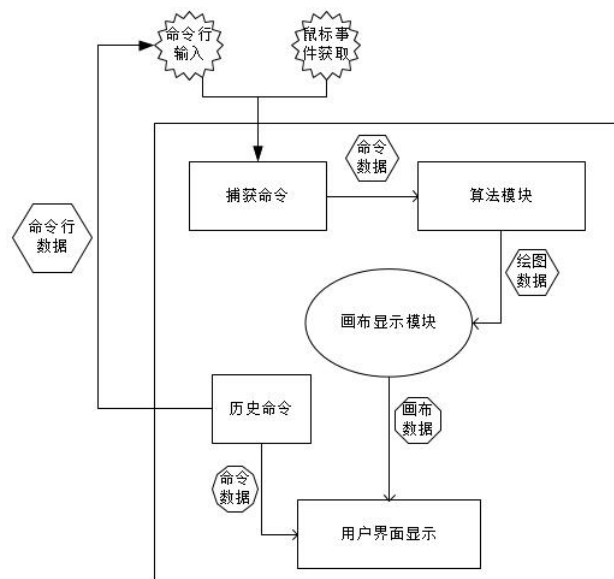
可知, 交点的参数为:

$$t_i = \frac{D_i}{Q_i}, \quad i = l, r, b, t$$

当  $Q_i < 0$  时, 求得的  $t_i$  必定是  $P_0P_1$  和始边的交点参数。当  $Q_i > 0$  时, 求得的  $t_i$  必定是  $P_0P_1$  和终边的交点参数。当  $Q_i = 0$  时, 若  $D_i < 0$ , 则  $P_0P_1$  是完全不可见的, 而当  $Q_i = 0$  时, 而对应的  $D_i \geq 0$ , 此时只需求出与  $y = y_t$  和  $y = y_b$  的交点决定直线段上可见部分即可。

## 3 系统介绍

### 3.1 系统框架设计

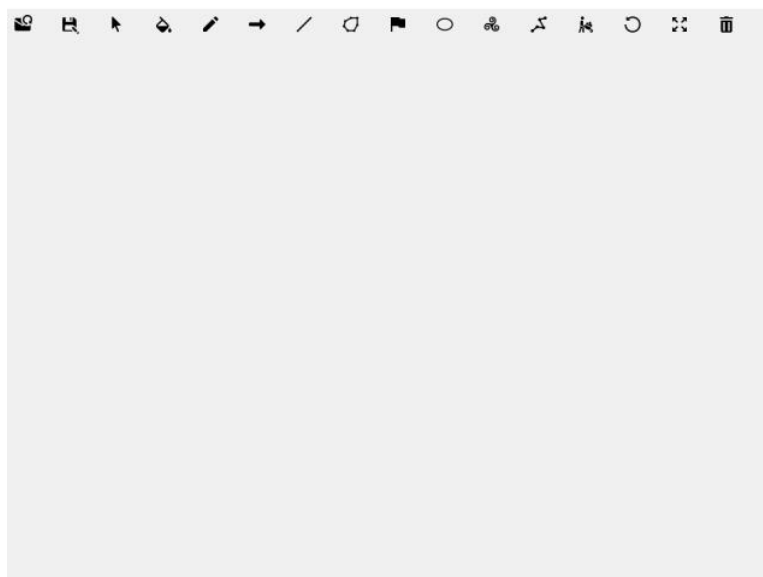


### 3.2 运行界面

#### 3.2.1 主界面



### 3.2.2 画布显示界面



### 3.2.3 历史命令界面



## 3.3 运行方式

系统支持两种操作，命令输入方式或者鼠标操作；

### 3.3.1 命令输入方式

#### 1. 读取文件方式

读取\*.txt文件，文件夹位于NJU\_CG/scripts文件夹下，在主界面命令行中输入input name(文件名，不包含后缀)即可。(如果是运行了\*.exe 文件，则\*.txt在此文件夹下的script中)。

## 2.读取命令方式

在主界面输入命令即可，命令支持为:

- (1) 打印历史命令: list canvas\_id(int,可缺省)

打印画布id为canvas\_id的画布上所有执行过的正确指令,其中canvas\_id 缺省为1.

- (2) 输出历史命令到文件: output name canvas\_id(int,可缺省)

打印id为canvas\_id的画布上所有执行过的正确指令,其中canvas\_id 缺省为1.

name:string 输出文件名

- (3) 重置画布: resetCanvas width height canvas\_id(int,可缺省)

清空id为canvas\_id的画布，并重新设置宽高，其中canvas\_id缺省为1

width, height: int

$100 \leq \text{width}, \text{height} \leq 1000$

- (4) 保存画布: saveCanvas name canvas\_id(int,可缺省)

将id为canvas\_id画布内容保存为位图name.bmp，其中canvas\_id缺省为1.

name: string

- (5) 设置画笔颜色: setColor R G B canvas\_id(int,可缺省)

将id为canvas\_id的画布的画笔颜色设置为R,G,B(输入决定),其中canvas\_id缺省为1.

R, G, B: int

$0 \leq R, G, B \leq 255$

- (6) 绘制线段: drawLine id x1 y1 x2 y2 algorithm canvas\_id\_id(int,可缺省)

在画布id为canvas\_id的画布上，以algorithm确定的算法绘制线段,线段端点分别为 $(x_1, y_1), (x_2, y_2)$ ,其中canvas\_id 缺省为1.

id: int 图元编号，每个图元的编号是唯一的

x1, y1, x2, y2 : float 起点、终点坐标

algorithm: string 绘制使用的算法，包括“DDA”和“Bresenham”

- (7) 绘制多边形: drawPolygon id n algorithm x1 y1 x2 y2 ...xn yn canvas\_id(int,可缺省)

在画布id为canvas\_id的画布上，以algorithm确定的算法绘制多边形,多边形的端点依次为 $(x_i, y_i), 1 \leq i \leq n$ ，其中canvas\_id 缺省为1.

id: int 图元编号，每个图元的编号是唯一的

n: int 顶点数


x1, y1, x2, y2 ... : float 顶点坐标

algorithm: string 绘制使用的算法，包括“DDA”和“Bresenham”

- (8) 绘制椭圆: `drawEllipse id x y rx ry canvas_id(int,可缺省)`  
在画布id为`canvas_id`的画布上绘制椭圆,其中`canvas_id`缺省为1.  
`id`: int 图元编号, 每个图元的编号是唯一的  
`x, y`: float 圆心坐标  
`rx, ry`: float 长短轴半径
- (9) 绘制曲线: `drawCurve id n algorithm x1 y1 x2 y2 ...xn yn canvas_id(int,可缺省)`  
在画布id为`canvas_id`的画布上,以`algorithm`确定的算法绘制曲线,其中`canvas_id`缺省为1.  
`id`: int 图元编号, 每个图元的编号是唯一的  
`n`: int 控制点数量  
`x1, y1, x2, y2 ...`: float 控制点坐标  
`algorithm`: string 绘制使用的算法, 包括“Bezier”和“B-spline”
- (10) 对图元平移: `translate id dx dy canvas_id(int,可缺省)`  
对画布id为`canvas_id`的画布上图元id为`id`的图元进行平移,其中`canvas_id`缺省为1.  
`id`: int 要平移的图元编号  
`dx, dy`: float 平移向量
- (11) 图元旋转: `rotate id x y r canvas_id(int,可缺省)`  
对画布id为`canvas_id`的画布上图元id为`id`的图元进行旋转,其中`canvas_id`缺省为1.  
`id`: int 要旋转的图元编号  
`x, y`: float 旋转中心  
`r`: float 顺时针旋转角度 (°)
- (12) 图元缩放: `scale id x y s canvas_id(int,可缺省)`  
对画布id为`canvas_id`的画布上图元id为`id`的图元进行缩放,其中`canvas_id`缺省为1.  
`id`: int 要缩放的图元编号  
`x, y`: float 缩放中心  
`s`: float 缩放倍数
- (13) 线段裁剪: `clip id x1 y1 x2 y2 algorithm canvas_id(int,可缺省)`  
对画布id为`canvas_id`的画布上图元id为`id`的直线图元, 以`algorithm`确定的算法进行裁剪,其中`canvas_id`缺省为1.  
`id`: int 要裁剪的图元编号  
`x1, y1, x2, y2`: float 裁剪窗口左下、右上角坐标  
`algorithm`: string 裁剪使用的算法, 包括“Cohen-Sutherland”和“Liang-Barsky”


### 3.3.2 鼠标操作方式

#### (1) 重置画布


点击画布界面中的图标,随后在弹出的对话框输入新的width和height, 点

击confirm即可.


(2) 保存画布

点击画布界面中的图标,随后在出现windows系统类似的保存对话框,进行相关的操作即可.


(3) 修改画笔颜色

点击画布界面中的图标,随后在出现windows系统类似的RGB调色对话框,进行相关的操作即可.


(4) 直线绘制(Bresenham算法)

点击画布界面中的图标,随后在界面中左击两个位置,即可产生一个以此两个位置为端点的直线;


(5) 直线绘制(DDA算法)

点击画布界面中的图标,随后在界面中左击两个位置,即可产生一个以此两个位置为端点的直线;


(6) 多边形绘制(Bresenham算法)

点击画布界面中的图标,随后在界面依次左击多边形的各个顶点的位置,最后再点击一次右键,即可产生所要的多边形;


(7) 多边形绘制(DDA算法)

点击画布界面中的图标,随后在界面依次左击多边形的各个顶点的位置,最后再点击一次右键,即可产生所要的多边形;


(8) 绘制椭圆

点击画布界面中的图标,随后在弹出的对话框输入相应id,x,y,rx,ry数据后,点击确认即可.


(9) 绘制曲线(Bezier曲线)

点击画布界面中的图标,随后在界面依次左击各个点击控制点的位置,最后再点击一次右键,即可生成Bezier曲线,为方便观察,控制点之间的线段也会绘制出来.

(10) 绘制曲线(B-spline曲线)


点击画布界面中的图标,随后在界面依次左击各个点击控制点的位置,最后再点击一次右键,即可生成B-spline曲线,为方便观察,控制点之间的线段也会绘制出来.

(11) 图元平移


点击画布界面中的图标,随机左键摁住需要平移的图元,鼠标移动,移动完成后,松开左键即可.




#### (12) 图元旋转

点击画布界面中的图标, 在弹出的对话框输入id,x,y,r等数据即可.

#### (13) 图元缩放

点击画布界面中的图标, 在弹出的对话框输入id,x,y,s等数据即可.

#### (14) 图元裁剪

点击画布界面中的图标, 在弹出的对话框输入id,x1,y1,x2,y2,algorithm等数据即可.(algorithm支持Cohen-Sutherland和Liang-Barsky)

### 3.4 运行结果

#### 3.4.1 鼠标运行结果展示


菜单选项功能依次为:

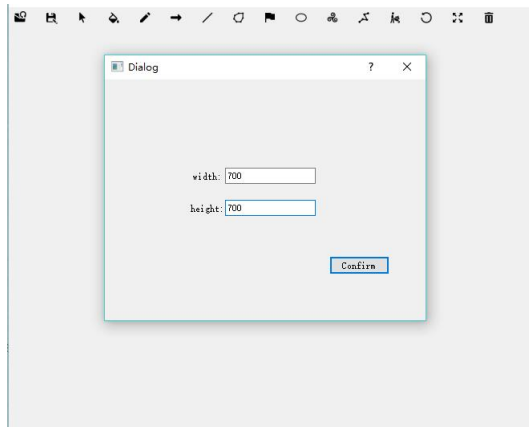
- (1) 重置画布:清空画布内容, 并重新设置画布的长和宽
- (2) 保存画布:保存画布内容, 并生成位图
- (3) 清空画笔模式属性:清空画笔的模式
- (4) 改变颜色:改变画笔的颜色
- (5) 绘制点模式:画笔进入绘制点的模式
- (6) 绘制直线模式(Bresenham 算法):画笔进入以Bresenham算法绘制直线的模式
- (7) 绘制直线模式(DDA算法):画笔进入以DDA算法绘制直线的模式
- (8) 绘制多边形模式(Bresenham 算法):画笔进入以Bresenham算法绘制多边形的模式
- (9) 绘制多边形模式(DDA 算法): 画笔进入以DDA算法绘制多边形的模式
- (10) 绘制椭圆模式:画笔进入绘制椭圆的模式
- (11) 绘制曲线模式(Bezier 算法):画笔进入以Bezier算法绘制曲线的模式
- (12) 绘制曲线模式(B-spline 算法):画笔进入以B-spline算法绘制曲线的模式
- (13) 平移模式按钮:进入图元平移模式
- (14) 旋转模式按钮:进入图元旋转模式
- (15) 放缩模式按钮:进入图元放缩模式
- (16) 裁剪模式按钮:进入图元裁剪模式

功能展示如下

- (1) 清空画笔模式, 绘制一个点功能就不做赘余的展示了


## (2) 重置画布

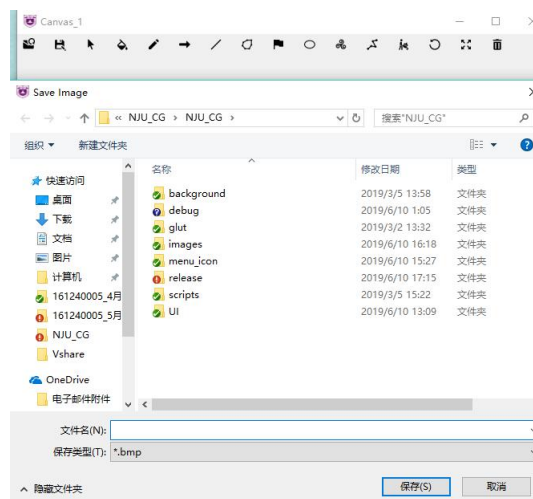
点击重置按钮后，画布清空，对话框提示输入新的画布度和高度



随后确认以后，即可看见相应的结果。

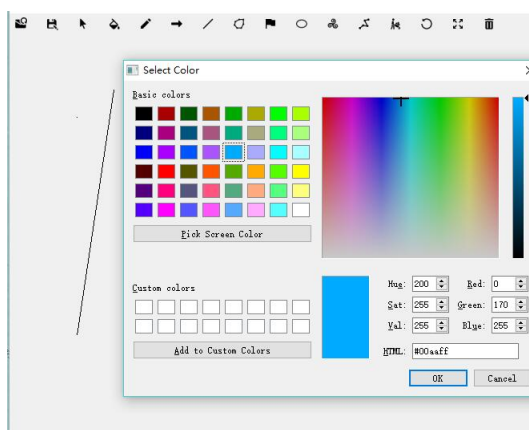
## (3) 保存画布

点击保存按钮后

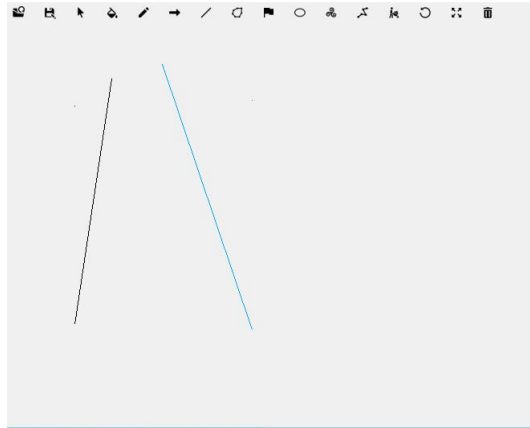



随后选择保存路径，确认以后，即可在对应的位置找到相应的bmp文件. 由于纯白色背景对眼睛不友好，于是背景色被修改成了以下RGB(219,207,202) 对应的颜色;

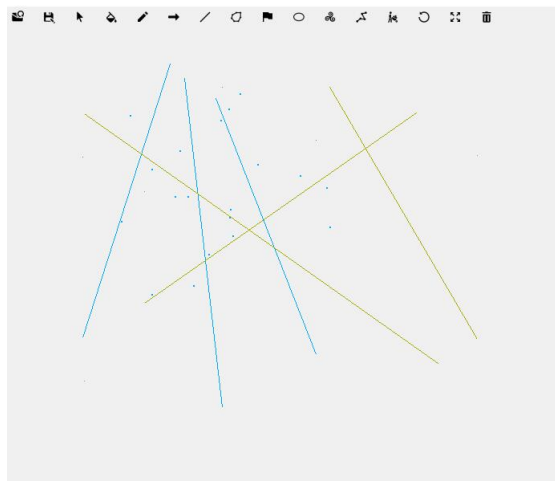
## (4) 改变颜色:改变画笔的颜色点击颜色修改按钮后,弹出对话框




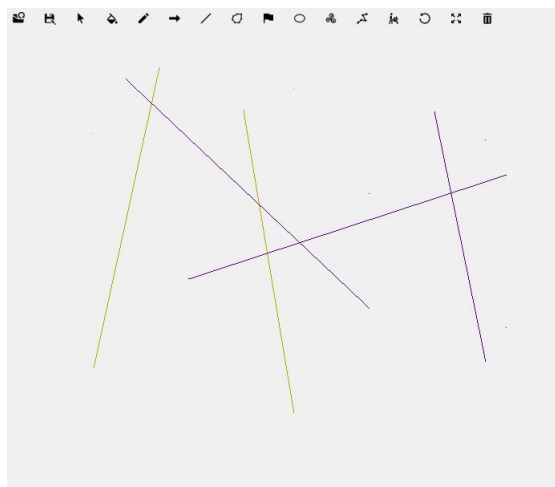
随后进行颜色选择，确认后再次绘制即可看见效果。




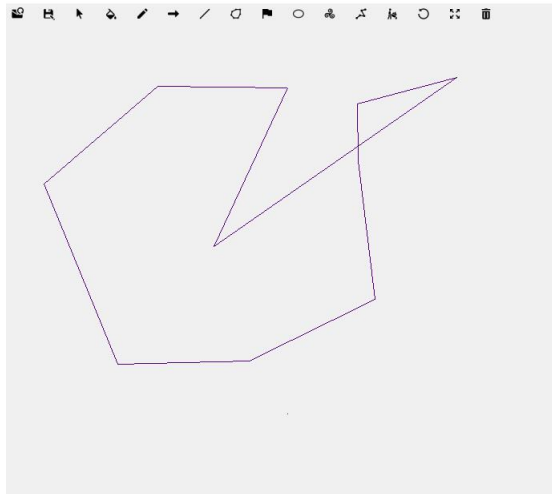
- (5) 绘制直线模式(Bresenham 算法):画笔进入以Bresenham算法绘制直线的模式  
点击绘制直线模式(Bresenham)按钮后随后进行绘制即可.绘制结果如下:




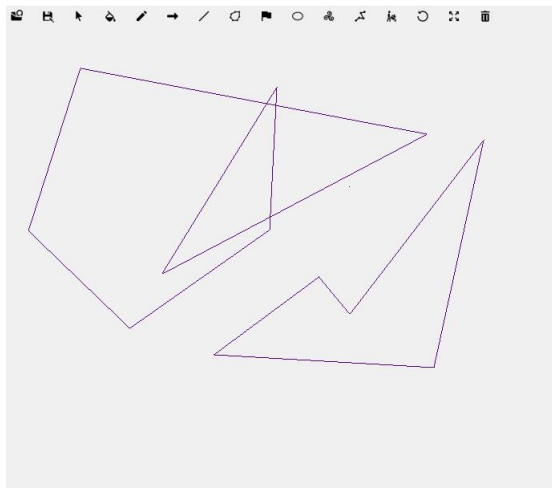
- (6) 绘制直线模式(DDA算法):画笔进入以DDA算法绘制直线的模式  
点击绘制直线模式(DDA)按钮后随后进行绘制即可.绘制结果如下:




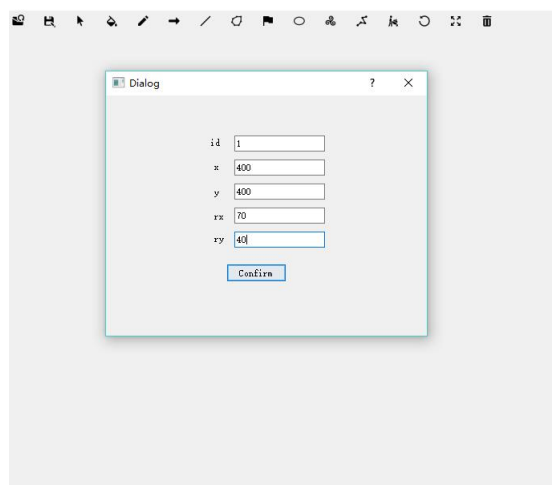
- (7) 绘制多边形模式(Bresenham 算法):画笔进入以Bresenham算法绘制多边形的模式:  
点击绘制多边形模式(Bresenham)按钮后随后进行绘制即可.绘制结果如下:



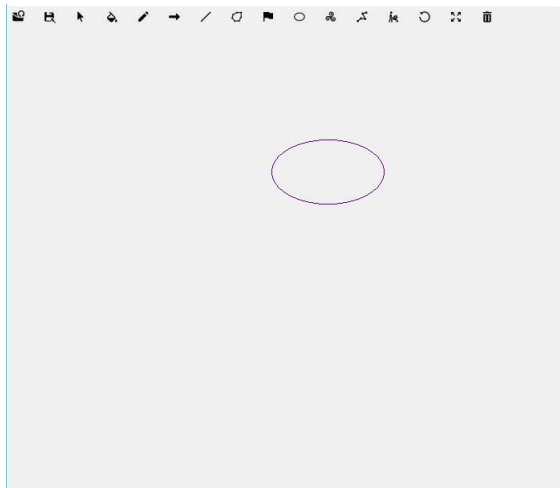
- (8) 绘制多边形模式(DDA 算法): 画笔进入以DDA算法绘制多边形的模式  
 点击绘制多边形模式(DDA)按钮后随后进行绘制即可.绘制结果如下:




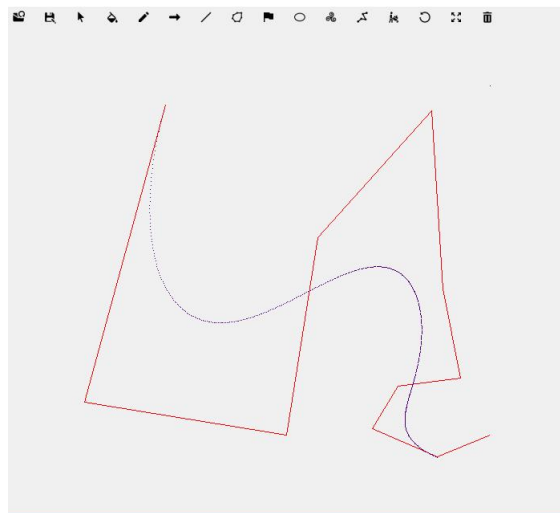
- (9) 绘制椭圆模式:画笔进入绘制椭圆的模式  
 点击绘制椭圆模式按钮后随后输入数据确认;




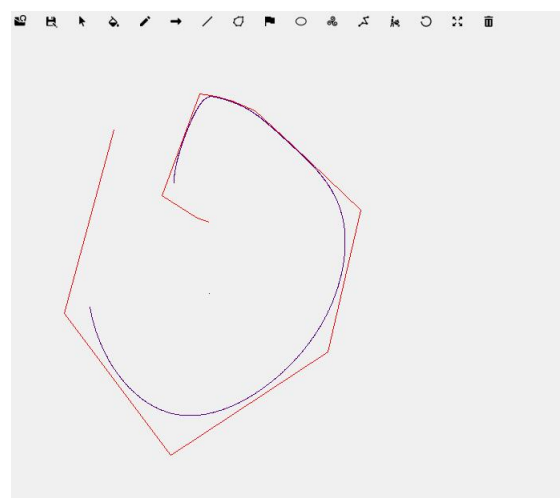
绘制结果如下:



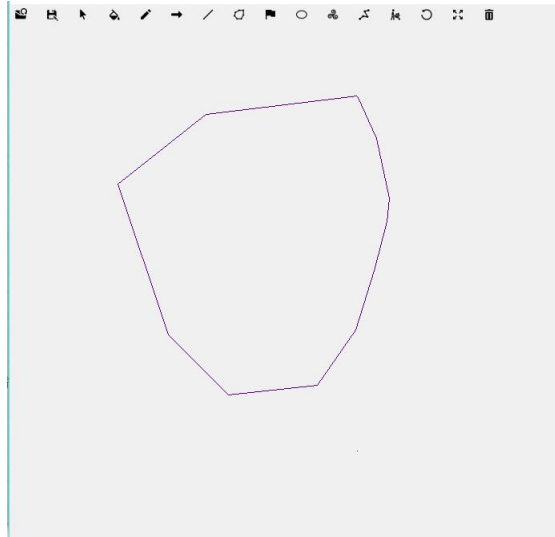
- (10) 绘制曲线模式(Bezier曲线):画笔进入以Bezier算法绘制曲线的模式  
 点击绘制曲线模式(Bezier)按钮 后随后进行绘制即可




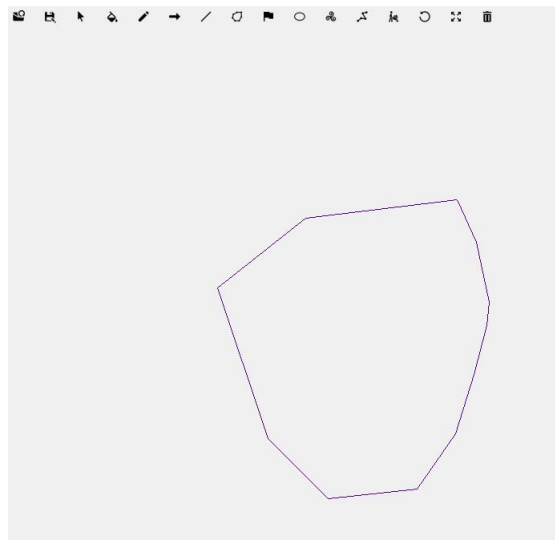
- (11) 绘制曲线模式(B-spline曲线):画笔进入以B-spline算法绘制曲线的模式  
 点击绘制曲线模式(B-spline)按钮 后随后进行绘制即可




- (12) 平移模式按钮:进入图元平移模式

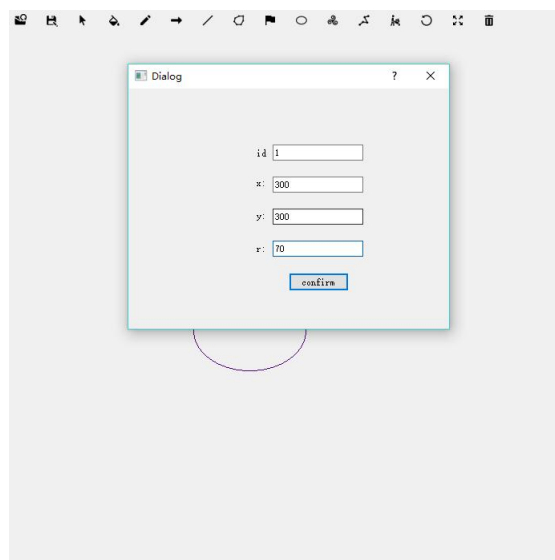


点击平移模式按钮后随后进行平移操作即可

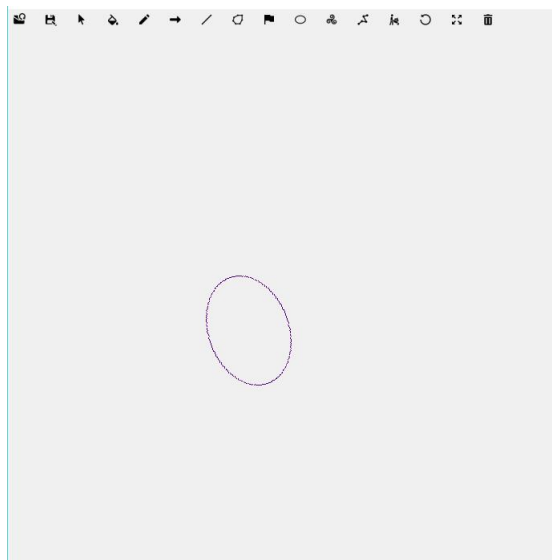


(13) 旋转模式按钮:进入图元旋转模式


点击旋转模式按钮后

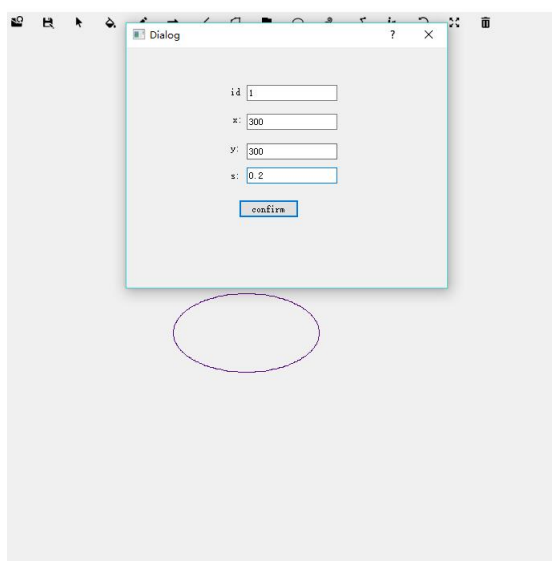


随后进行输入相关数据并确认即可

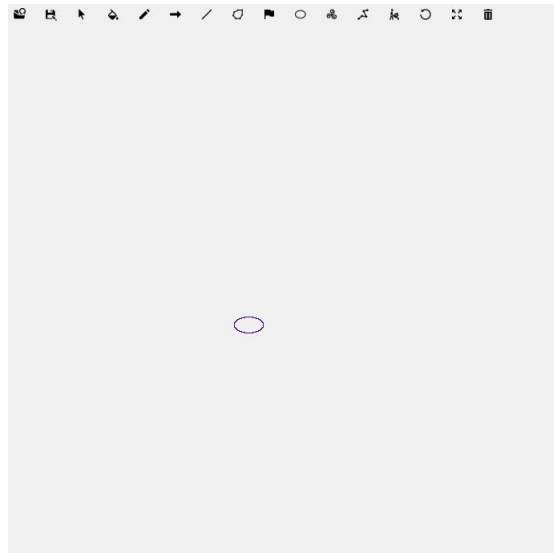



(14) 放缩模式按钮:进入图元放缩模式

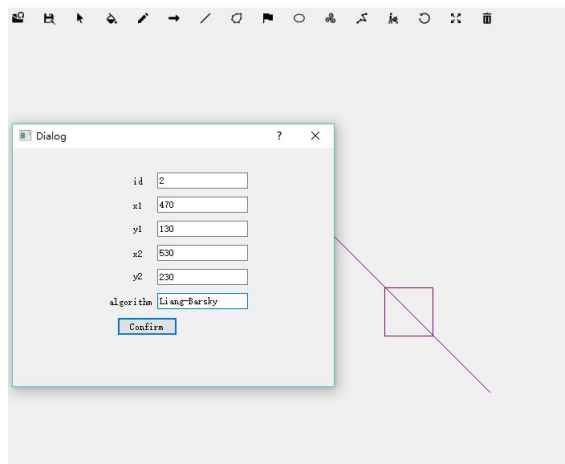
点击缩放模式按钮后



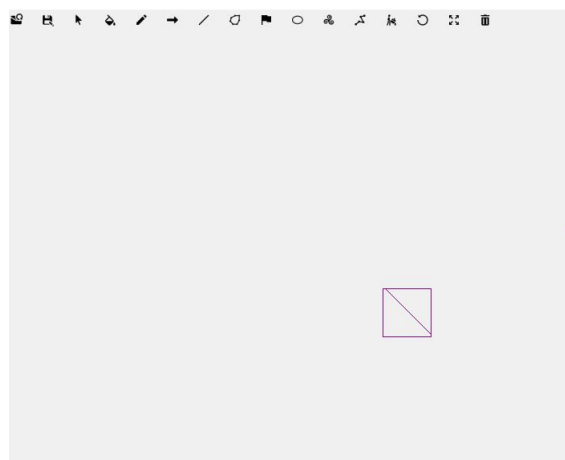
随后进行输入相关数据并确认即可



(15) 裁剪模式按钮:进入图元裁剪模式点击裁剪模式按钮后



随后进行输入相关数据并确认即可

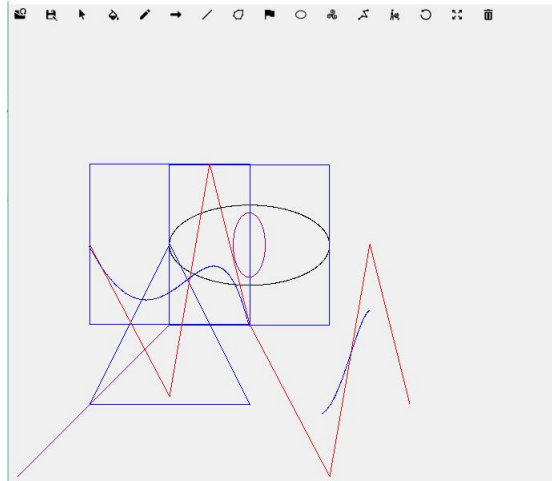


### 3.4.2 命令运行结果展示

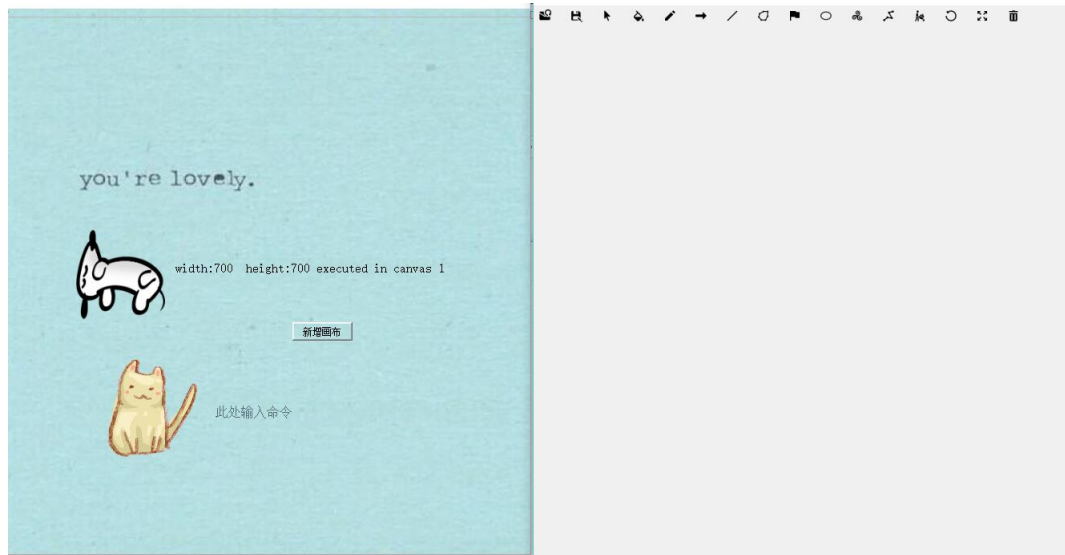
(1) 重置画布

原来的画布





输入resetCanvas 700 700 1(重置id为*canvas\_1*的画布时, 1可以缺省, 否则需输入对应的id)后



## (2) 保存画布

例如若保存画布1的内容, 命名为Mydraw;

输入saveCanvas Mydraw 1(1可以省略, 保存其他画布时不可以省略)即可,将可以在NJU\_CG/NJU\_CG/images 目录中找到Mydraw.bmp文件(如果是运行了\*.exe文件, 则MyDraw 在此文件夹下的images中)

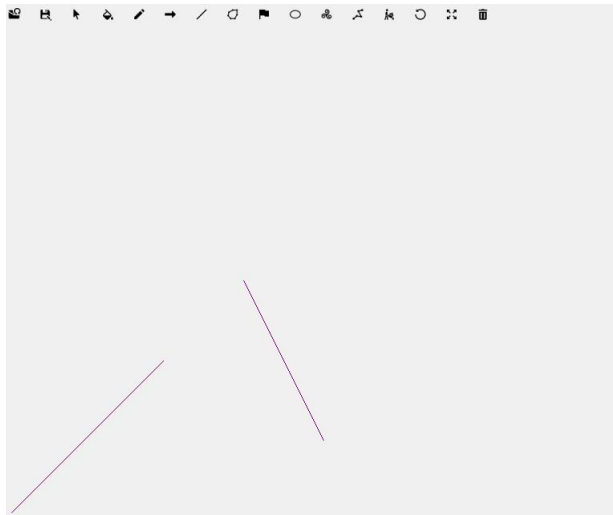
## (3) 直线绘制:

输入以下命令后(对id为*canvas\_1*的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

```
drawLine 1 10 10 200 200 DDA 1
```

```
drawLine 2 300 300 400 100 Bresenham 1
```

将会得到:



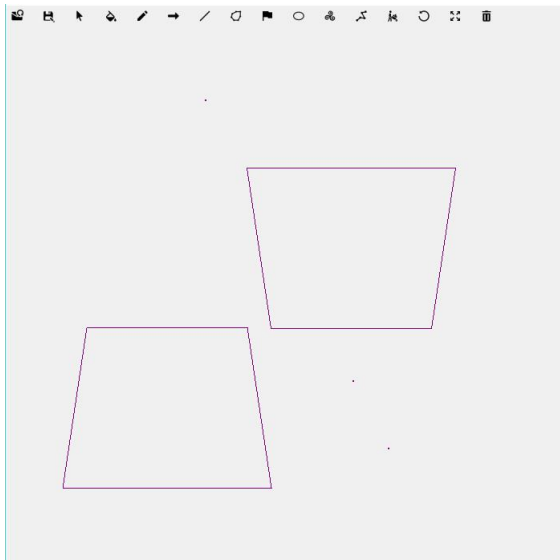
(4) 多边形绘制:

输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

```
drawPolygon 1 4 Bresenham 300 500 560 500 530 300 330 300 1
```

```
drawPolygon 2 4 DDA 70 100 330 100 300 300 100 300 1
```

将会得到:

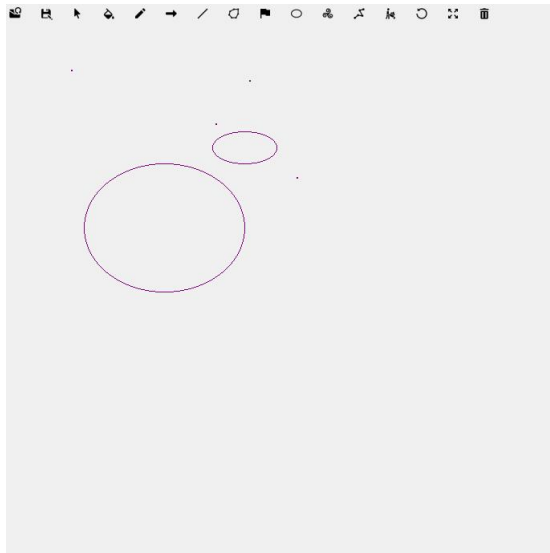


(5) 椭圆绘制: 输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

```
drawEllipse 1 300 300 40 20 1
```

```
drawEllipse 2 200 200 100 80 1
```

将会得到:



(6) 曲线绘制:

输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

```
drawCurve 1 7 B-spline 0 200 50 100 70 300 150 320 230 300 300 400 450 100 1
```

```
drawCurve 2 7 Bezier 0 400 50 300 70 500 150 520 230 500 300 600 450 300 1
```

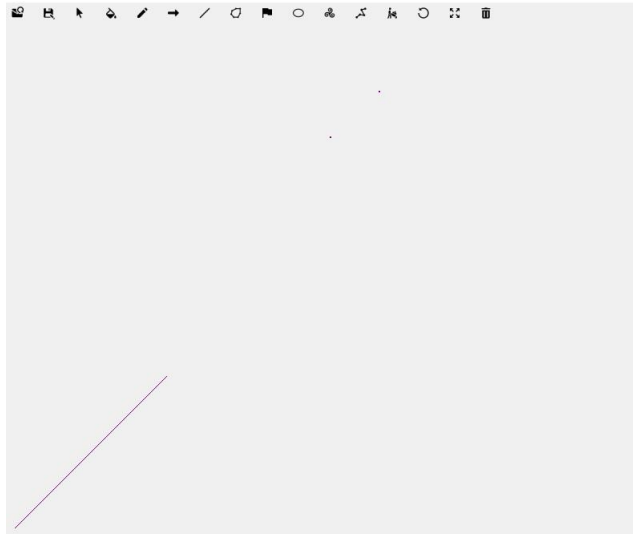
将会得到:



(7) 图元平移:

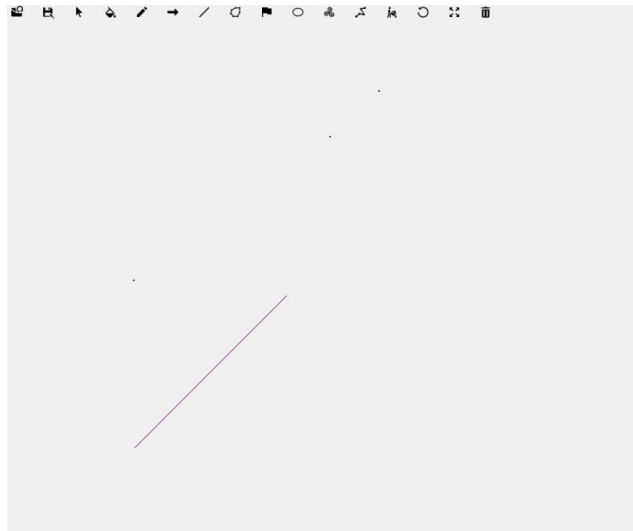
依次输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

```
drawLine 1 10 10 200 200 DDA 1
```



再输入`translate 1 150 100 1`

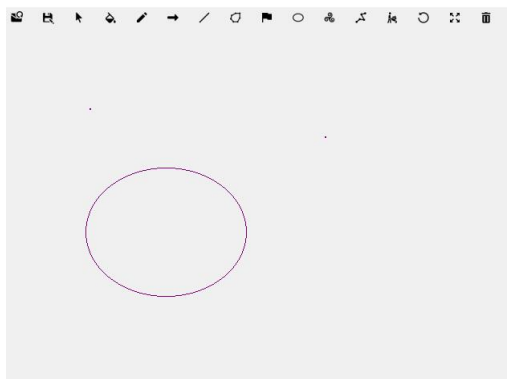
将会得到:



(8) 图元旋转:

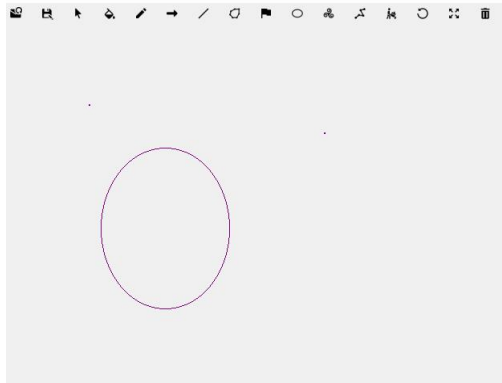
依次输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

`drawEllipse 1 200 200 100 80 1`



再输入rotate 1 200 200 90 1

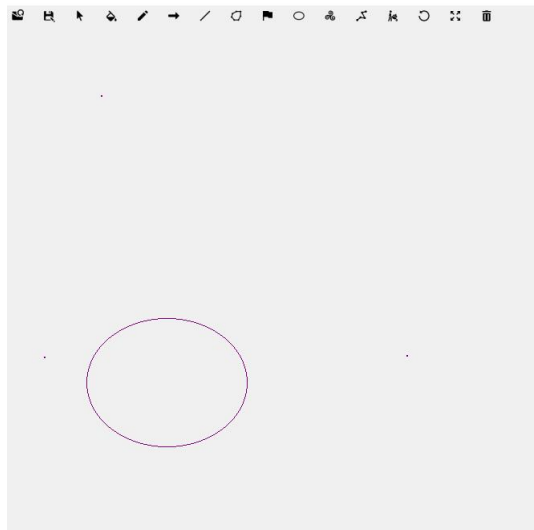
将会得到:



(9) 图元缩放:

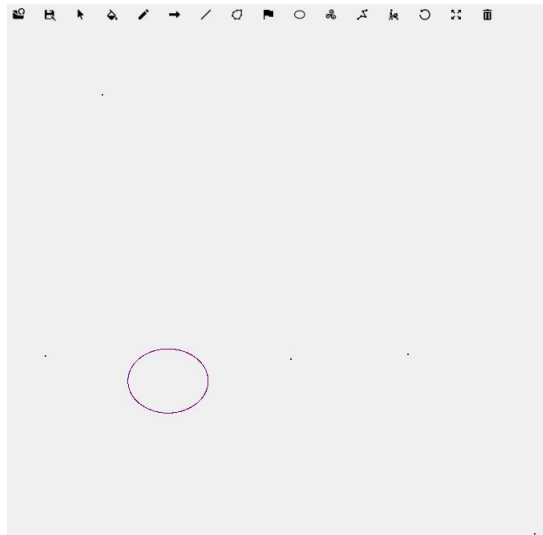
依次输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

drawEllipse 1 200 200 100 80 1



再输入scale 1 200 200 0.5 1

将会得到:



(10) 线段裁剪:

依次输入以下命令后(对id为`canvas_1`的画布时, 命令最后的1可以缺省, 否则需输入对应的id)后:

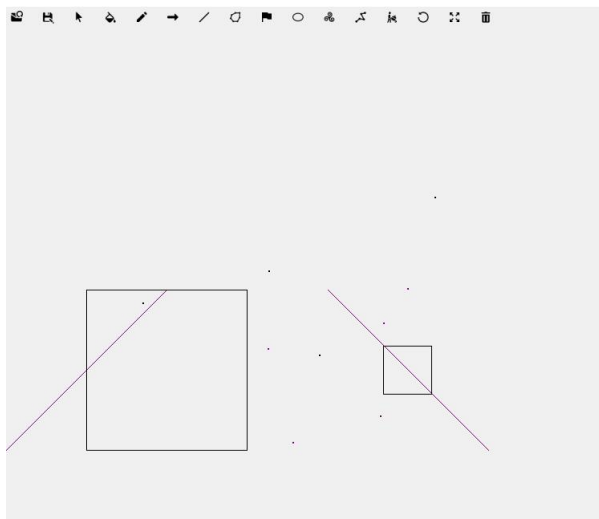
```
drawLine 1 0 100 201 301 DDA 1
```

```
drawLine 2 601 100 401 301 DDA 1
```

```
setColor 0 0 0 1
```

```
drawPolygon 3 4 DDA 100 100 300 100 300 300 100 300 1
```

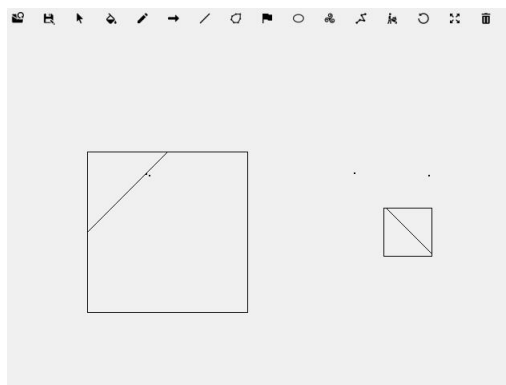
```
drawPolygon 4 4 DDA 470 170 530 170 530 230 470 230 1
```



再输入`clip 1 100 100 300 300 Cohen-Sutherland 1`

`clip 2 470 170 530 230 Liang-Barsky 1`

将会得到:



## 4 总结

通过Bresenham, DDA, 中点圆算法, de Boor, Cohen-Sutherland, Liang-Barsky 等算法的学习和实践, 可以实现计算机图形学中诸如直线绘制, 线段裁剪等有趣的图形学功能呢。深入的实践将有助于对图形学的认识和理解, 也可以为2D至3D的转换做好基础知识的准备。

### 参考文献

[1] 计算机图形学第2版徐长青, 许志闻, 郭晓新编机械工业出版社