

《计算机图形学》4月报告

161240005 陈勇虎

2019 年 4 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，或者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。后续将会继续添加图形用户界面，以鼠标交互的方式进行图元的绘制的变换。

(如果用Qt打开源文件，需注意路径不含有中文名，比如原来压缩后文件夹中会有我的名字...)

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 Δx 或 Δy 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($\Delta x < \Delta y = 1$)

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,

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

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

$$p_{i+1} = p_i + 2(\Delta y - \Delta x) \quad p_i \geq 0 \quad (3)$$

$$p_{i+1} = p_i + 2\Delta y \quad p_i < 0 \quad (4)$$

$$p_1 = 2\Delta y - \Delta x \quad (5)$$

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

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 多边形绘制算法待完善

- 1.原理分析
- 2.理解分析
- 3.性能测试

2.3 椭圆生成算法Done

- 1.原理分析
- 2.理解分析
- 3.性能测试

2.4 曲线生成算法

2.4.1 Bezier算法Done

- 1.原理分析
- 2.理解分析
- 3.性能测试

2.4.2 B-spline算法(TODO)

- 1.原理分析
- 2.理解分析
- 3.性能测试

2.5 裁剪算法(TODO)

2.5.1 Cohen-Sutherland算法

- 1.原理分析
- 2.理解分析
- 3.性能测试

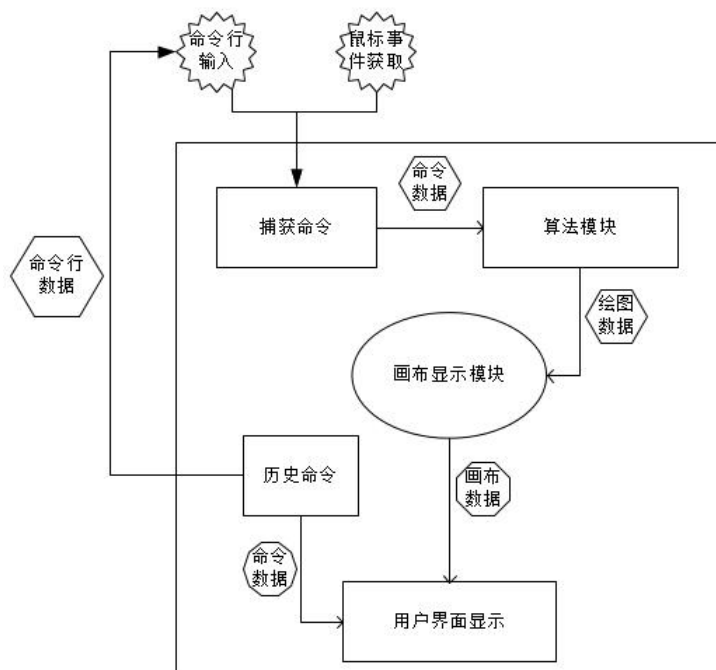
2.5.2 liang-Barsky算法(TODO)

- 1.原理分析
- 2.理解分析
- 3.性能测试

2.6 TODO

3 系统介绍

3.1 系统框架设计

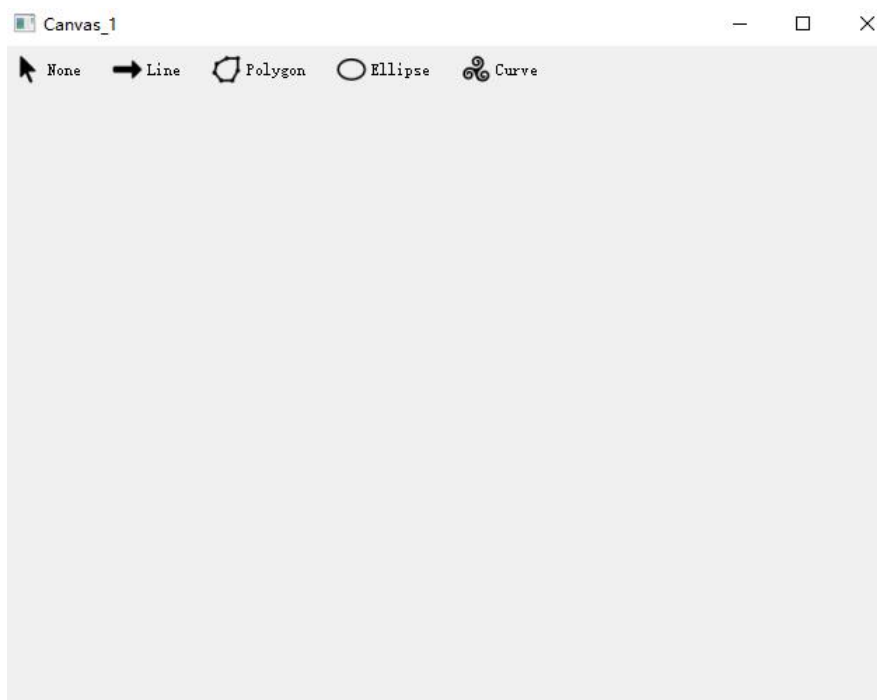


3.2 运行界面

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. 读取命令方式

在主界面输入命令即可，命令支持为：

1. 重置画布：resetCanvas width height canvas_id(int)

清空id为canvas_id的画布，并重新设置宽高canvas_id缺省为1.

width, height: int

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

2. 保存画布：

saveCanvas name canvas_id(int)

将id为canvas_id画布保存为位图name.bmp canvas_id缺省为1.

name: string

3. 设置画笔颜色：

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

setColor R G B canvas_id(int)

R, G, B: int

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

4.绘制线段:

drawLine id x1 y1 x2 y2 algorithm canvas_id(int)

在id为canvas_id的画布上绘制直线,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”

5.绘制多边形:

drawPolygon id n algorithm x1 y1 x2 y2 ...xn yn canvas_id(int)

在id为canvas_id的画布上绘制多边形,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”

6.绘制椭圆 (中点圆生成算法):

drawEllipse id x y rx ry canvas_id(int)

在id为canvas_id的画布上绘制椭圆,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

7.绘制曲线:

drawCurve id n algorithm x1 y1 x2 y2 ...xn yn canvas_id(int)

在id为canvas_id的画布上绘制曲线,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”

8.对图元平移:

translate id dx dy canvas_id(int)

对画布id为canvas_id的画布上序号为id的图元进行平移,canvas_id缺省为1.

id: int 要平移的图元编号

dx, dy: float 平移向量

9.对图元旋转:

rotate id x y r canvas_id(int)

对画布id为canvas_id的画布上序号为id的图元进行旋转,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 (°)

10.对图元缩放:

scale id x y s canvas_id(int)

对画布id为canvas_id的画布上序号为id的图元进行缩放,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

11.对线段裁剪:

clip id x1 y1 x2 y2 algorithm canvas_id(int)

对画布id为canvas_id的画布上序号为id的图元进行裁剪,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”

12.打印历史命令

list canvas_id(int)

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

13.输出历史命令到文件

output name canvas_id(int)

name:string 输出文件名


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

2.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NJU_CG_boxed.exe文件, 则*.txt在此文件夹下的script中)

3.3.2 鼠标操作方式(已经完成的部分)

直线绘制

直线绘制, 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;

多边形绘制

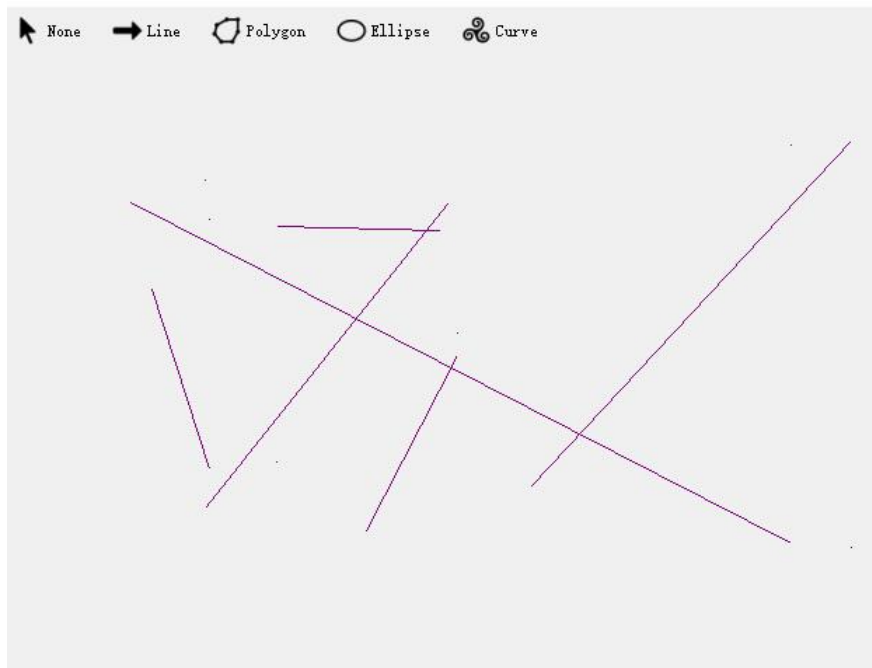
多边形绘制, 点击画布界面中的图标,随后在界面依次左击多边形的各个顶点的位置, 最后再点击一次右键, 即可产生所要的多边形;

TODO

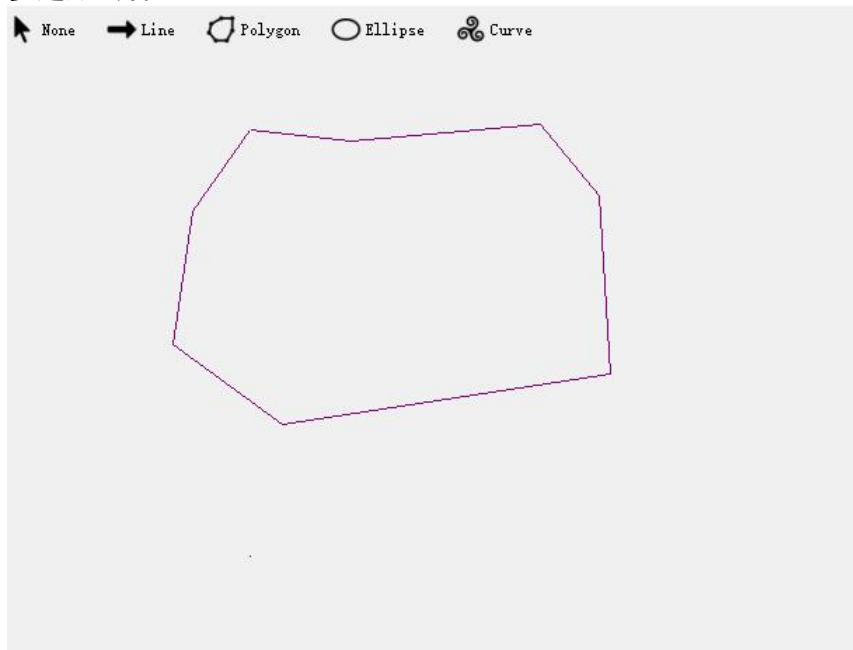
3.4 运行结果

3.4.1 鼠标运行结果展示

直线绘制



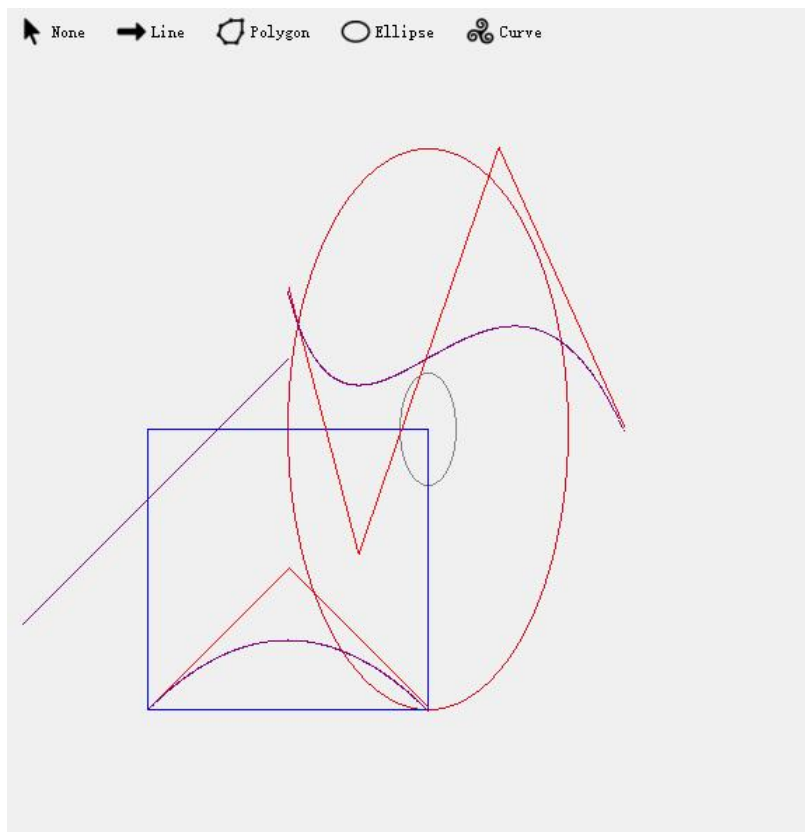
多边形绘制



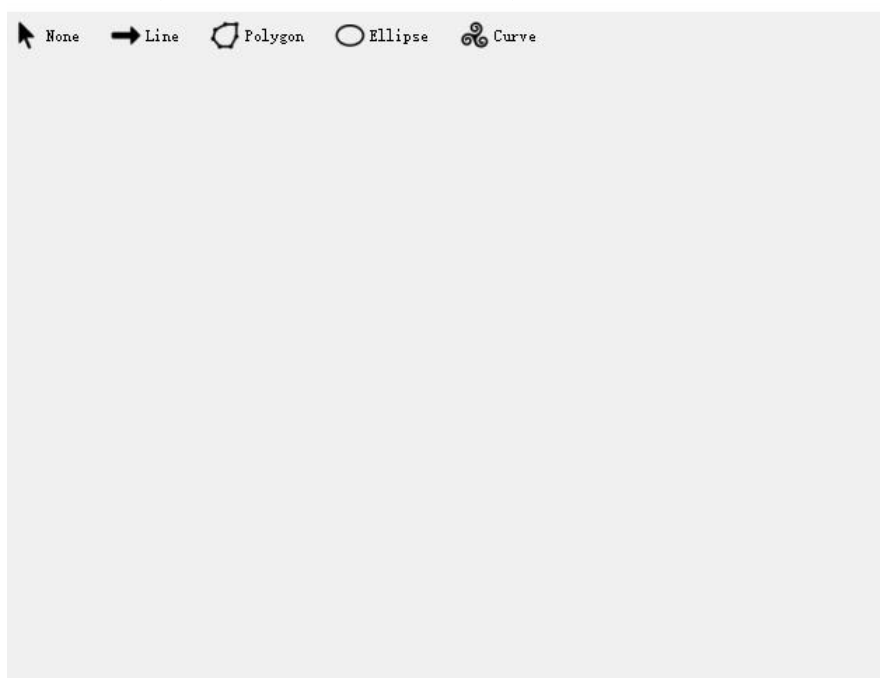
TODO

3.4.2 命令运行结果展示

重置画布 原来的画布



重置画布后(可以通过打印历史命令确认)



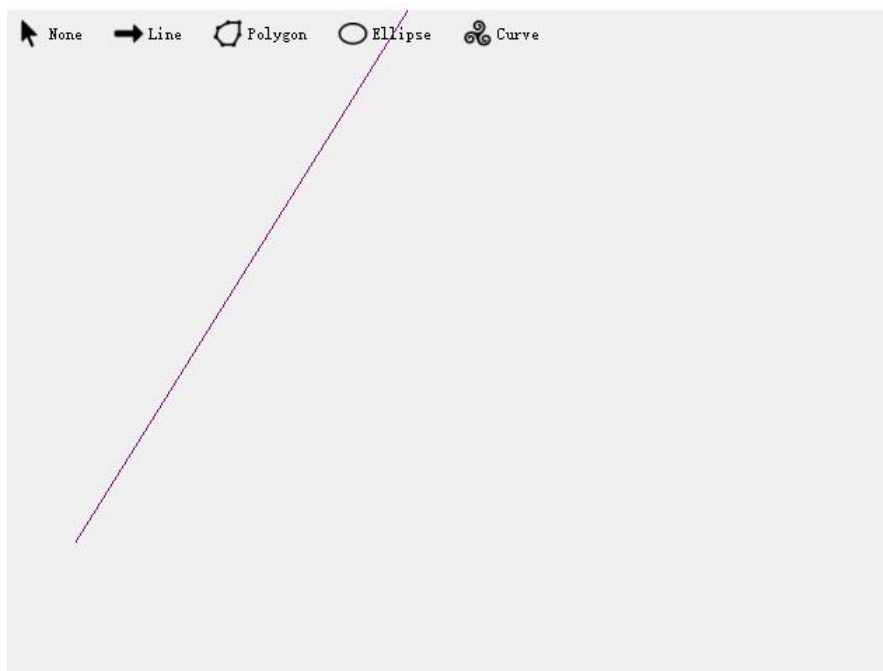
保存画布

例如若保存画布1的内容，命名为Mydraw; 输入saveCanvas Mydraw 1(可省略)即可, 可以在NJU_CG/NJU_CG/images目录中找到Mydraw.bmp文件(如果是运行了NJU_CG_boxed.exe文件，则MyDraw在此文件夹下的images中)

直线绘制

例如我们输入drawLine 1 50 100 300 500 DDA 4，即在第四个画布上使用DDA算法绘制一条直线，直线起点为(50,100)，中点为(300,500)，图元id为1.(4缺省的话，则是在第一

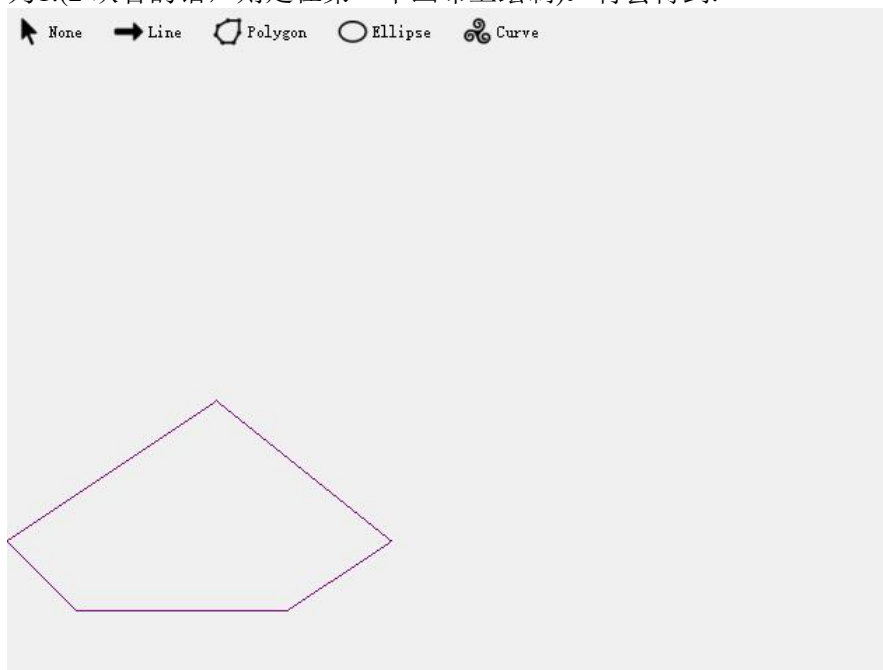
个画布上绘制)。将会得到:



Bresenham算法同理

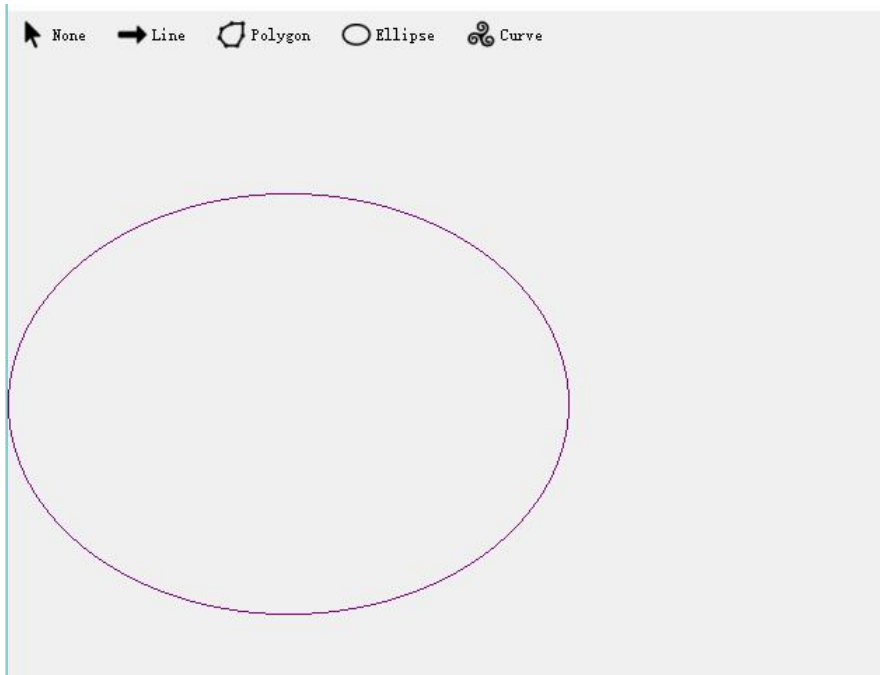
多边形绘制

例如我们输入`drawPolygon 1 5 DDA 50 50 200 50 275 100 150 200 0 100 2`，即在第二个画布上使用DDA算法绘制一个多边形，一共有5个点，分别为(50,50)，(200,50),(275,100),(150,200),(0,100),图元id为1.(2缺省的话，则是在第一个画布上绘制)。将会得到:



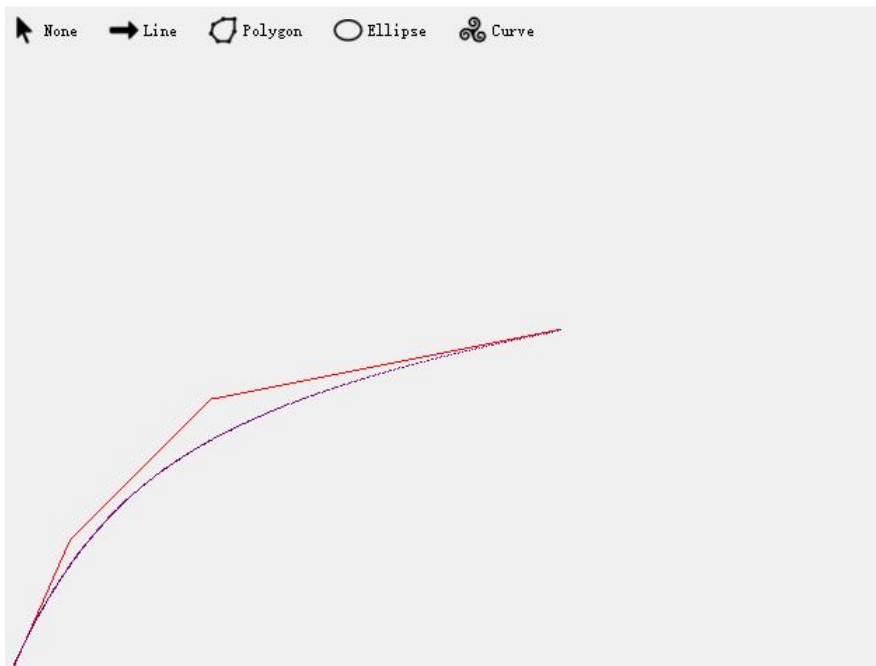
椭圆绘制

例如我们输入`drawEllipse 1 200 200 200 150 3`，即在第三个画布上绘制一个椭圆，圆心为(200,200),长轴为200，短轴为150,图元id为1.(3缺省的话，则是在第一个画布上绘制)。将会得到:



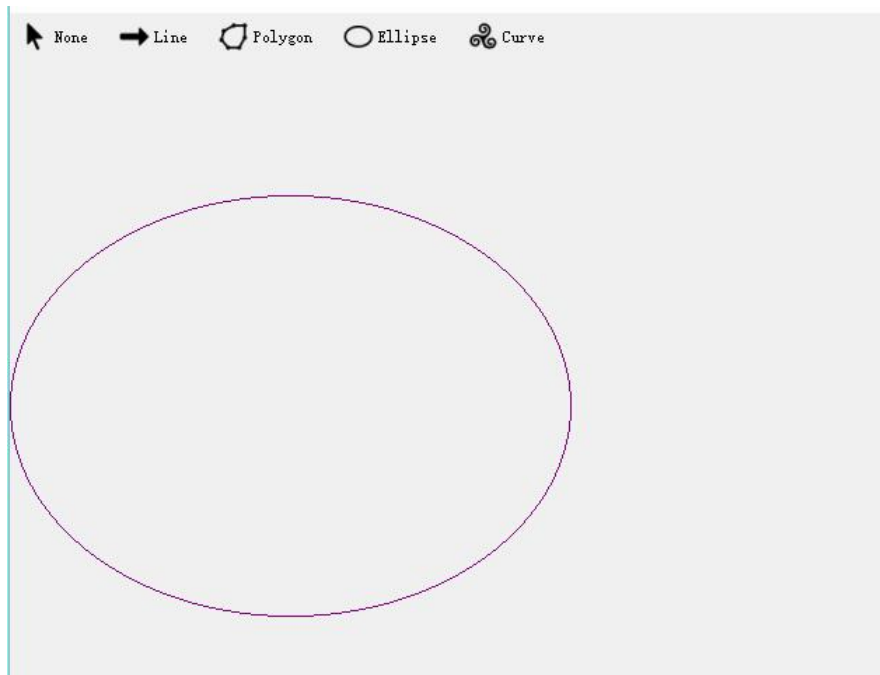
曲线绘制

例如我们输入`drawCurve 1 5 Bezier 10 10 50 100 100 150 200 400 250 2`，即在第二个画布上使用Bezier算法绘制曲线，控制点数量为5，控制点坐标为(10,10),(50,100),(100,150),(150,200)(400,250)元id 为1.(2 缺省的话，则是在第一个画布上绘制)。将会得到:

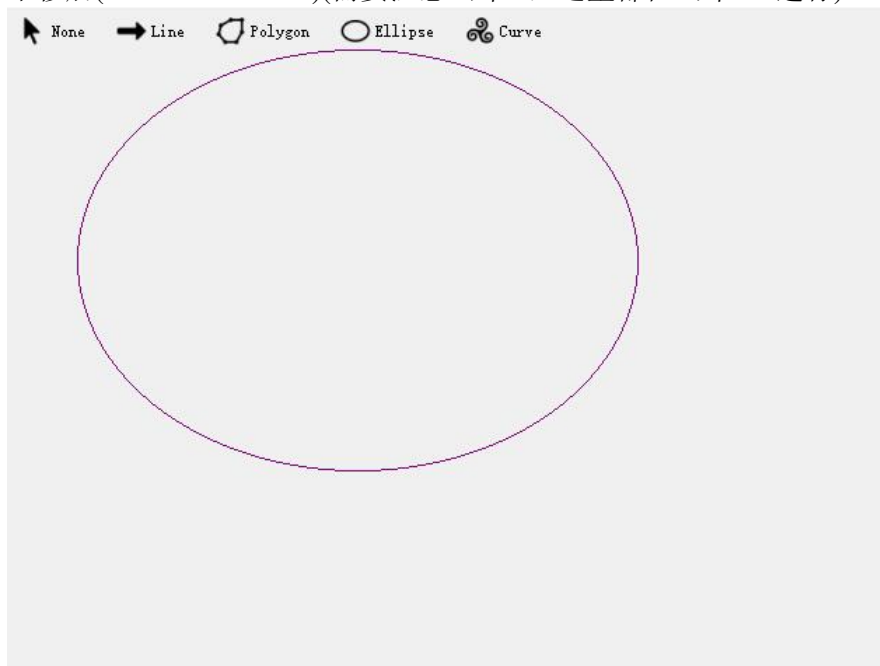


图元平移

平移前(`drawEllipse 1 200 200 200 150`):

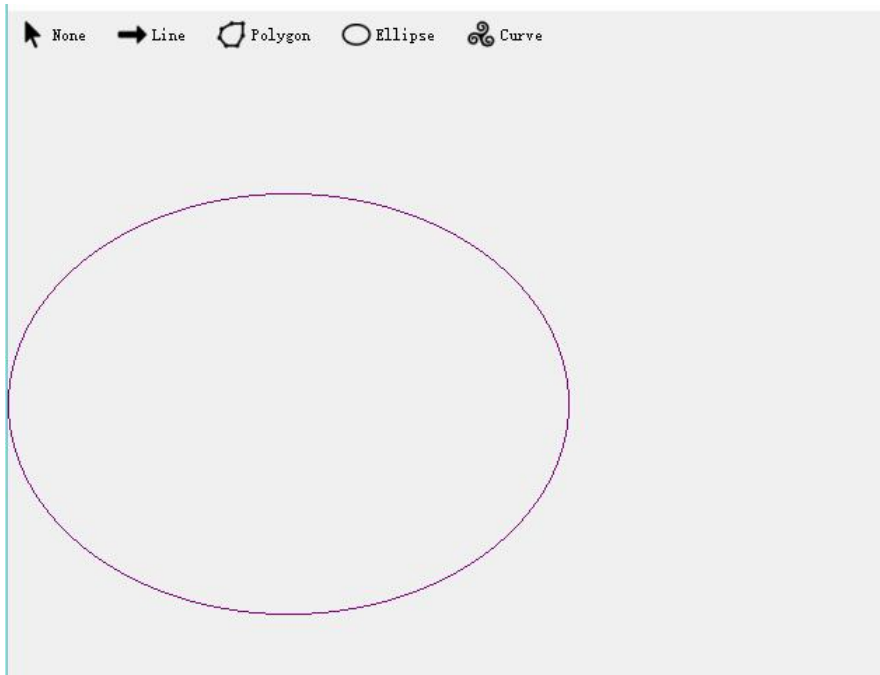


平移后(`translate 1 50 100`)(需要注意画布id, 这里都在画布1上进行):

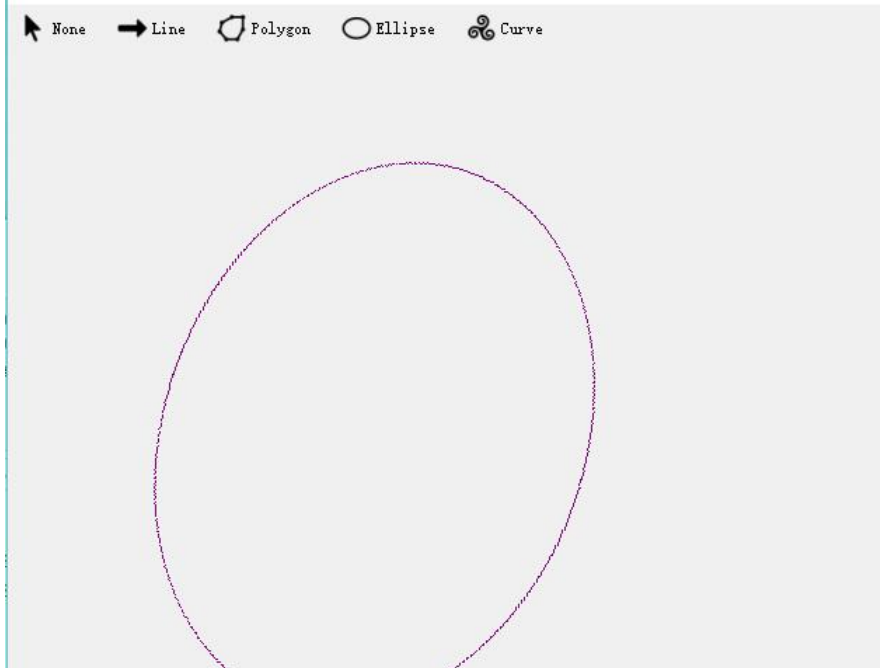


图元旋转

旋转前(`drawEllipse 1 200 200 200 150`):

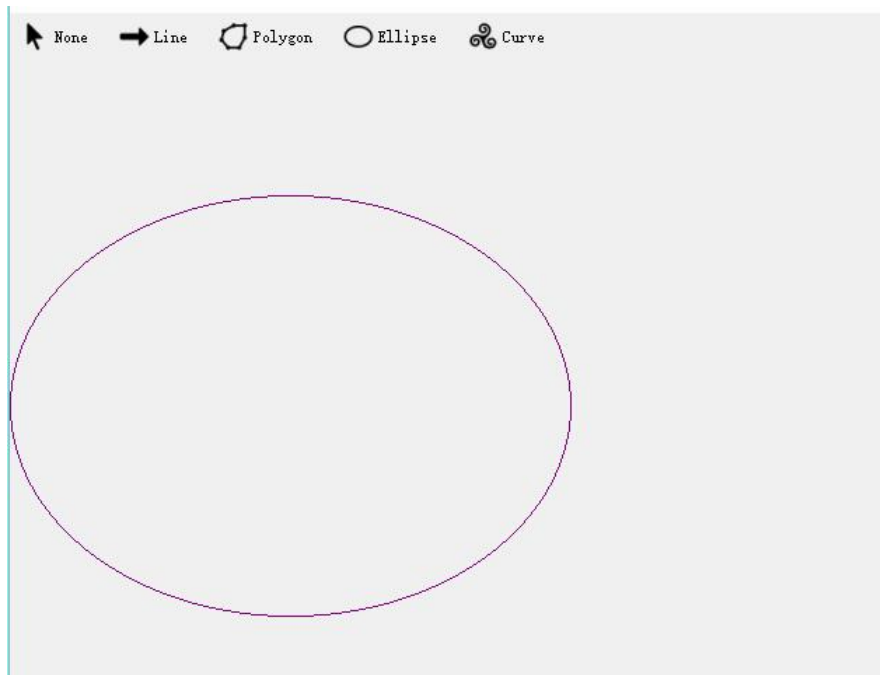


旋转后(rotate 1 250 250 70)(需要注意画布id, 这里都在画布1上进行):

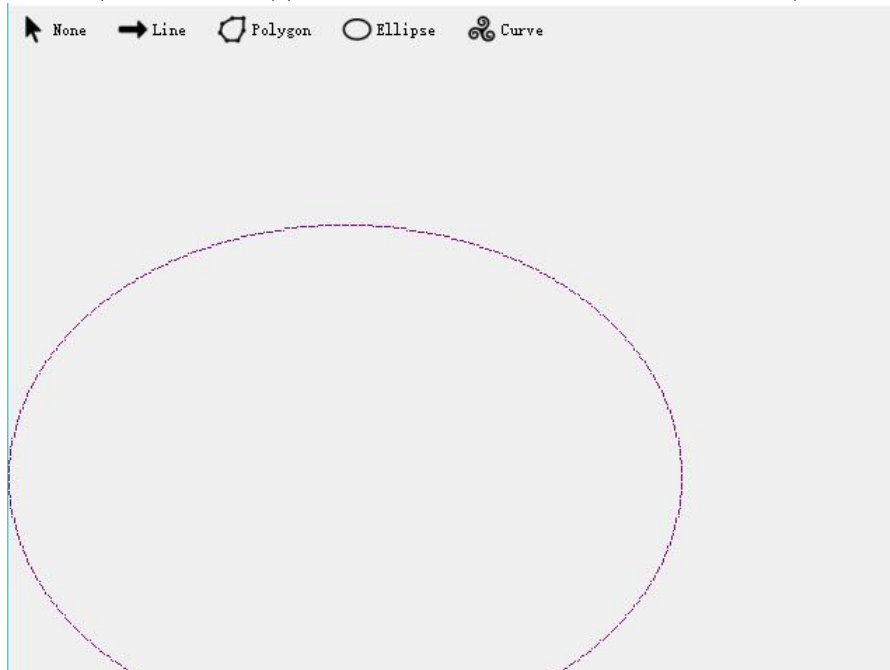


图元缩放

缩放前(drawEllipse 1 200 200 200 150):



缩放后(scale 1 0 0 1.2)(需要注意画布id, 这里都在画布1上进行):



线段裁剪**TODO**

4 总结

...

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[2]