《计算机图形学》系统技术报告

张帅

(南京大学 计算机科学与技术系, 南京 210093)

摘 要: 本绘图系统采用了面向对象设计，以 C++11 为基础，交互使用了 Qt，基础图形的绘制使用了 OpenGL，直线绘制使用了 Bresenham 算法，圆的绘制使用了中点圆生成算法，椭圆绘制用了中点椭圆生成算法，曲线绘制使用了 3 次 Bézier 曲线，多边形填充使用了多边形扫描填充算法，线段裁剪使用了梁友栋-Barsky 算法，多边形裁剪使用了逐边裁剪算法。

1 算法概述

1.1 绘制算法

1.1.1 直线绘制：Bresenham 算法

直线绘制采用了 Bresenham 算法，此算法优点在于避免了浮点运算，运算量小，从而可以得到实时的屏幕响应。

设起始点到终点坐标为$(x_0,y_0),(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$，第$k$步的决策参数为$p_k$，递推公式如下：

$$y_{k+1}=\begin{cases}y_k, & p_k<0\\ y_k+1, & p_k\geq 0\end{cases} \tag{1.1.1-1}$$

$$p_{k+1}=\begin{cases}p_k+2\Delta y, & p_k<0\\ p_k+2\Delta y-2\Delta x, & p_k\geq 0\end{cases} \tag{1.1.1-2}$$

此处以斜率在 0 到 1 之间为例：

1. 计算$\Delta x=x_k-x_0$，$\Delta y=y_k-y_0$；
2. 绘制直线左侧的起始点(x_0,y_0)，并计算初始决策变量$p_0=2\Delta y-2\Delta x$；
3. 根据上述递推公式从x_0开始，每次 x 递增 1，并在此 x 位置处对应的 y 和决策变量 p，直至x_k。

对于斜率大于 1 的情况，只需将递增变量由 x 改为 y，递推公式中y_{k+1}与y_k的关系改为x_{k+1}与x_k的关系即可。

对于斜率为负的情况，只需将递推公式中的y_k+1改为y_k-1即可。

1.1.2 圆的绘制：中点圆生成算法

圆的绘制采用了中点圆算法，同样避免了浮点运算，使用递推的方式，节省了计算量。

根据圆的对称性，只需要绘制 1/8 圆便可得到完整的圆。从圆心正上方顺时针绘制 1/8 圆，然后通过对称绘制剩余 7/8 圆。如下为 1/8 圆绘制方法：

设圆心坐标为(x_c,y_c)，半径为 r，起始点到终点坐标为(x_c+x_0,y_c+y_0)，(x_c+x_1,y_c+y_1)，(x_c+x_2,y_c+y_2)，…，(x_c+x_n,y_c+y_n)，第k步的决策参数为p_k，递推公式如下：

$$y_{k+1}=\begin{cases}y_k, & p_k<0\\ y_k-1, & p_k\geq 0\end{cases} \tag{1.1.2-1}$$

$$p_{k+1}=\begin{cases}p_k+4(2x_k+3), & p_k<0\\ p_k+4(2x_k-2y_k+3), & p_k\geq 0\end{cases} \tag{1.1.2-2}$$

圆绘制过程如下：

1. 绘制圆心正上方的点(x_c+x_0,y_c+y_0)，即$(x_c,y_c+\mathrm{r})$，计算初始决策参数$p_0=5-4r$；

2. 在x_k处按照如上的递推公式(1.1.2-1)与(1.1.2-2)进行计算，得到y_{k+1}与p_{k+1}，并绘制点$(x_c+x_{k+1},y_c+y_{k+1})$，其中$x_{k+1}=x_k+1$；

3. 将 2 中所得的点对称到其他 7/8 圆得到相应的对称点并绘制；

4. 重复 2~3，直至$x_k\geq y_k$。

1.1.3 椭圆绘制：中点椭圆生成算法

将椭圆按照象限分为 4 个部分，先绘制出第一象限的部分，然后根据对称性绘制出其余 3 个象限的部分。

第一象限的部分分为两个区域：区域 1：切线斜率小于 1；区域 2：切线斜率大于 1. 生成过程如下：

1. 输入椭圆中心(x_c,y_c)和长短轴径r_x和r_y;

2. 计算得到中心在原点的椭圆上的第一个点$(x_c,\ y_c)$=$(0,\ r_y)$;

3. 根据公式(1.1.3-2)计算区域 1 中决策参数的初值为$p1_0$;

4. 在区域 1 中每个x_k位置处，从$k=0$开始，反复按照公式(1.1.3-3)更新决策参数，若决策参数小于 0 则y_k不变，反之y_k加 1，一直循环到$2r_y^2x\geq 2r_x^2y$为止，记第一个满足此条件的点为(x_0,y_0);

5. 在区域 2 的每个y_k位置处，从$k=0$开始，反复按照公式(1.1.3-6)更新决策参数，若决策参数小于 0 则x_k不变，反之x_k加 1，一直循环到$(r_x,0)$为止;

6. 根据第一象限的点对称画出其他三个象限的点;

7. 将每个计算出的像素位置(x,y)平移到中心到(x_c,y_c)的椭圆轨迹上，并按平移坐标值画点(x,y)，其中$\begin{cases}x=x+x_c\\ y=y+y_c\end{cases}$.

区域 1：

决策参数：
$$p1_k=f_{ellipse}\left(x_{k+1},y_k-\frac{1}{2}\right)=r_y^2(x_k+1)^2+r_x^2(y_k-\frac{1}{2})^2-r_x^2r_y^2. \tag{1.1.3-1}$$

初始取值：
$$p1_0=r_y^2-r_x^2r_y+\frac{r_x^2}{4}. \tag{1.1.3-2}$$

更新公式：
$$\begin{cases}p1_{k+1}=p1_k+2r_y^2x_k+3r_y^2, & p_k<0\\ p1_{k+1}=p1_k+2r_y^2x_k-2r_x^2y_k+2r_x^2+3r_y^2, & p_k\geq 0\end{cases}. \tag{1.1.3-3}$$

区域 2：

决策参数：
$$p2_k=f_{ellipse}\left(x_k+\frac{1}{2},y_k-1\right)=r_y^2(x_k+\frac{1}{2})^2+r_x^2(y_k-1)^2-r_x^2r_y^2. \tag{1.1.3-4}$$

初始取值：
$$p2_0=r_y^2(x_0+\tfrac{1}{2})^2+r_x^2(y_0-1)^2-r_x^2r_y^2. \tag{1.1.3-5}$$

更新公式：

$$\begin{cases} p2_{k+1} = p2_k - 2r_x^2 y_k + 3r_x^2, & p_k > 0 \\ p2_{k+1} = p2_k + 2r_y^2 x_k - 2r_x^2 y_k + 2r_y^2 + 3r_x^2, & p_k \le 0 \end{cases}. \quad (1.1.3\text{-}6)$$

1.1.4 曲线绘制：三次 Bézier 曲线

在三阶 Bézier 曲线中有四个控制点，设第 i 个控制点为P_i，则P_0和P_3分别为曲线的起点和终点。曲线上点的计算公式为：

$$P(u) = \sum_{i=0}^{3} P_i BEZ_{i,3}(u) \quad (1.1.4\text{-}1)$$

$$P(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u)P_2 + u^3 P_3 . (0 \le u \le 1)$$

将u从 0 累计到 1，从而可以计算得到曲线上从P_0到P_3的所有点。（其中u每次增加 0.001，以保证曲线的连续）

1. 2 填充算法

1.2.1 多边形填充：多边形扫描填充算法

先后使用了有序边表与活化边表，得到各扫描行的填充范围，从而进行填充。

边表使用了如下数据结构：

```
struct Edge
{
    Edge() {}
    Edge(double xi, double dx, int ymax)
    {
        this->xi = xi;
        this->dx = dx;
        this->ymax = ymax;
    }
    double xi; //本扫描行该边的起始点横坐标
    double dx; //该边y每增加1，x的增加量，即斜率的倒数
    int ymax; //该边最高点的y值
};
```

a. 生成有序边表：扫描多边形中所有边，跳过所有点 y 值相同的水平边，将所有边最下方的点加入到有序边表中，填入对应的 xi，根据该边起始点计算出 dx。若该点在上下侧，则 ymax 为该边的最高点 y 值；若该点在左右两侧，则 ymax 为该边最高点 y 值-1（为了防止重复绘制顶点带来的内外侧判断错误）。

b. 根据有序边表生成活化边表：扫描当前扫描线及以下的所有有序边表，将 ymax 大于等于当前 y 值的 Edge 项加入到本扫描线对应的活化边表中，加入时 xi 需要根据当前 y 值和有序边表中的 xi，dx 计算，其余项保持不变。然后根据 xi 对当前扫描线的 Edge 项从小到大排序。

c. 根据活化边表生成各扫描行填充范围：遍历当前扫描行所有 Edge 项，两两扫描，从索引为偶数的项开始，填充到下一个项的 xi 为止。

1.2.2 圆的填充算法

圆的填充实现较为简单，通过圆的方程计算出圆轮廓上纵坐标为y对应左右侧点的横坐标值x_l和x_r，填充x_l和x_r中间的所有点即可。设圆心为(x_0, y_0)，半径为r，x_l和x_r计算公式如下

$$\begin{cases} x_l = x_0 - \sqrt{r^2 + (y - y_0)^2} \\ x_r = x_0 + \sqrt{r^2 + (y - y_0)^2} \end{cases} \tag{1.2.2-1}$$

1.3 变换算法

图形变换包括平移、旋转和缩放。设原先点为$P_1(x_1, y_1)$，变换后点为$P_2(x_2, y_2)$。

1.3.1 平移

设平移向量为$\mathrm{T}(t_x, t_y)$，有

$$\begin{cases} x_2 = x_1 + t_x \\ y_2 = x_1 + t_y \end{cases} \tag{1.3.1-1}$$

1.3.2 旋转

设旋转基准点为$C(x_0, y_0)$，顺时针旋转角为θ，有

$$\begin{cases} x_2 = x_0 + (x_1 - x_0)cos\theta - (y_1 - y_0)sin\theta \\ y_2 = y_0 + (x_1 - x_0)sin\theta + (y_1 - y_0)cos\theta \end{cases} \tag{1.3.2-1}$$

1.3.3 缩放

设缩放基准点为$C(x_0, y_0)$，在x方向和y方向的缩放系数分别为S_x和S_y，有

$$\begin{cases} x_2 = x_1 S_x + x_0(1 - S_x) \\ y_2 = y_1 S_y + y_0(1 - S_y) \end{cases} \tag{1.3.3-1}$$

1.4 裁剪算法

设裁剪窗口矩形横坐标范围为$[xw_{min}, xw_{max}]$，纵坐标范围为$[yw_{min}, yw_{max}]$.

1.4.1 线段裁剪：梁友栋-Barsky 算法

设线段为AB，$A(x_1, y_1)$，$B(x_2, y_2)$，$\Delta x = x_2 - x_1$，$\Delta y = y_2 - y_1$.

设变量p_i表示第i侧$\overrightarrow{AB}$为从内到外/从外到内，$p_i > 0$表示外->内；$p_i = 0$表示线段平行于该边界；$p_i < 0$表示内->外。($0 \le$ i ≤ 3, 分别表示左、右、上、下边界)

$$\begin{cases} p_0 = -\Delta x \\ p_1 = \Delta x \\ p_2 = -\Delta y \\ p_3 = \Delta y \end{cases} \tag{1.4.1-1}$$

定义变量q_i如下

$$\begin{cases} q_0 = x_1 - xw_{min} \\ q_1 = xw_{max} - x_1 \\ q_2 = y_1 - yw_{min} \\ q_3 = yw_{max} - y_1 \end{cases} \tag{1.4.2-2}$$

设直线的参数$u(0 \le u \le 1)$，$u = 0$时在A点，$u = 1$时在B点.

设r $= \frac{q_i}{p_i}$，则r为直线与第 i 边界交点对应的直线参数 u 的值。

设线段裁剪之后两端点对应u值分别为u_1, u_2. 不妨设$u_1 \le u_2$.

对四个方向的p和q进行遍历，

若$p_i = 0$，说明线段平行于该侧边界。若此时$q_i < 0$，说明线段在外部，则整个线段均被裁掉，直接返回；否则线段在内部，本边界与线段没有交点，不做处理；

若$p_i < 0$，说明$\overrightarrow{AB}$从外到内穿过该边界，令$u_1 = \max(u_1, r)$;

若$p_i > 0$，说明$\overrightarrow{AB}$从内到外穿过该边界，令$u_2 = \min(u_2, \mathrm{r})$.

遍历完成之后，若$u_1 > u_2$则说明整个线段在裁剪窗口之外，直接舍弃；反之保留$u \in [u_1, u_2]$区间内的线段。

1.4.2 多边形裁剪：逐边裁剪算法

设多边形原先顶点表为 vertices.

用 4 条边界线依次对多边形进行裁剪，

设本次裁剪得到的新顶点表为 vtxs，

对 vertices 表中各顶点 i 进行遍历，

若点 vertices[i]在边界线外侧，

若点 vertices [(i+1)%vertexes.size()]在边界线外侧，vtxs 不加入任何点；

若点 vertices [(i+1)%vertexes.size()]在边界线内侧，vtxs 加入 i 与 i+1 形成的边与边界线的交点，和 i+1 号顶点；

若点 vertices[i]在边界线内侧，

若点 vertices [(i+1)%vertexes.size()]在边界线外侧，vtxs 只加入 i 与 i+1 形成的边与边界线的交点；

若点 vertices [(i+1)%vertexes.size()]在边界线外侧，vtxs 只加入 i+1 号顶点

顶点遍历完成之后将新顶点表 vtxs 赋给 vertices.

所有边界裁剪完成之后的顶点表即为新多边形的顶点表，根据此顶点表生成新多边形即可。

2 系统框架设计

2.1 开发环境

操作系统	Windows 10 家庭中文版
编程语言	C++ 11
外部库	Qt 5.4，OpenGL 2.7
编译器	MinGW 4.9.1 32bit
IDE	Qt Creator 3.3.0

2.2 系统架构概览

用 Qt 向用户提供交互，用 OpenGL 提供底层绘制，考虑到 Qt 与 OpenGL 的兼容性，使用了继承自 QWidget 的子类 QGLWidget 实现 Qt 环境下的 OpenGL 绘制。

QMainWindow 负责与用户交互，并以标签页的形式集成了 QGLWidget；

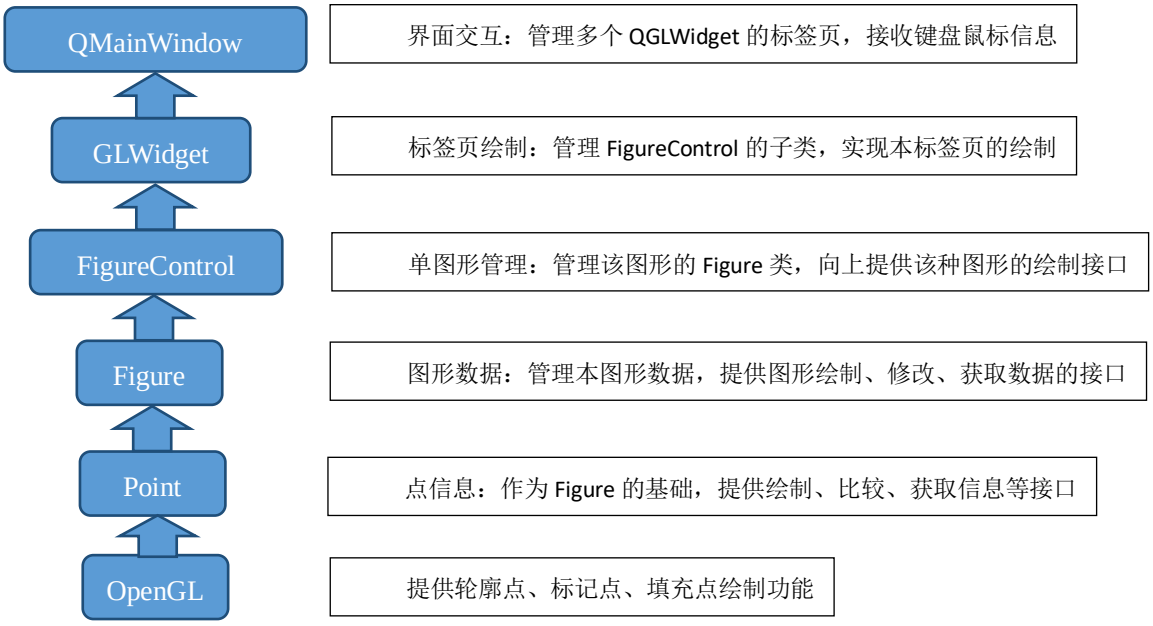
QGLWidget 负责绘制本标签页的内容，并集成了各类 FigureControl，用于将本标签页的交互信号送给不同的图形控制类；

FigureControl 负责该类型图形的交互，有 LineControl、CircleControl、EllipseControl、PolygonControl 等，用于分别提供不同图形的不同交互方式；

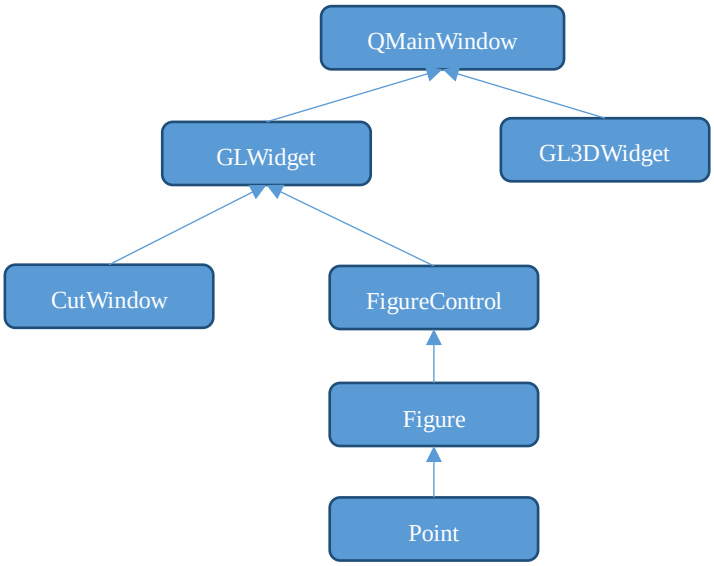
Figure 负责保存该图形的数据信息，包括顶点、圆心、半径以及轮廓点、填充点等，集成了 Point 类，图形绘制通过调用 Point 的 draw 函数完成；

Point 负责实现绘制单个普通点或标记点，并向上提供其他常用功能函数支持。

系统核心架构如下图



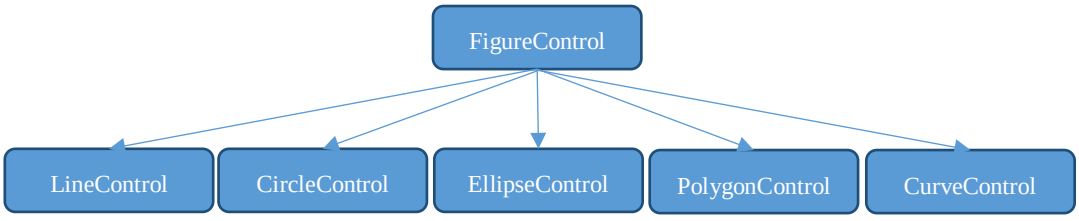
其中 GLWidget 派生出一个 GL3DWidget，作为 3D 标签页，用于 3D 图形的绘制；在 FigureControl 层有一个 CutWindow 类，用于绘制裁剪窗口。系统架构中的聚集关系如下图



Point 类作为轮廓和填充的构成部分聚集在 Figure 类中；Figure 类的数组放在 FigureControl 中，由 FigureControl 处理本 Figure 的相关交互；CutWindow 用于管理裁剪窗口；GLWidget 为 2D 画布的标签页，GL3DWidget 继承自 GLWidget，作为 3D 画布标签页；所有标签页聚集在 QMainWindow 中，由 QMainWindow 与用户交互并向下发送消息。

2.3 图形管理类：FigureControl

图形管理类之间的关系较为简单，主要用于存储本图形的数据，并进行与本图形相关的交互，如鼠标点击、键盘按下、图形绘制、设置聚焦、裁剪当前图形等，类继承关系如下图



2.4 图形类：Figure

Figure 为所有图形的虚基类，继承关系如下图

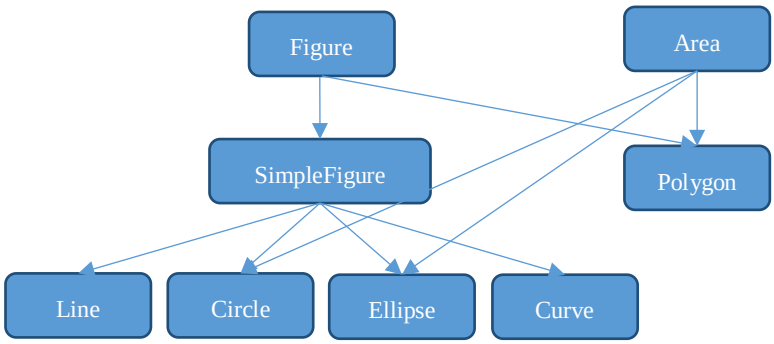


Figure 类提供所有图形都应有的共同接口：

- 基础绘制：绘制图形、绘制图形标记点、清空图形
- 基础变换：平移、旋转、缩放
- 判断某点是否在该图形上

```
class Figure
{
public:
        virtual void draw()=0; //绘制图形本身
        virtual void markDraw()=0; //绘制除draw以外的标记：标记点(+矩形框)
        virtual void clear()=0; //清除图形中的点并释放空间

        virtual void translate(const Point &offset)=0; //平移
        virtual void rotate(double angle)=0; //旋转
        virtual void scale(double s)=0; //缩放

        virtual bool isOn(const Point &p)=0; //点p是否在本图形上(含标记点、center、handle)
        virtual bool isOnPlain(const Point &p)=0; //点p是否在图形本身上

};
```

SimpleFigure 是对所有简单图形的抽象，SimpleFigure 的轮廓均直接由点构成。而 Polygon 的边为 Line，所以不属于 SimpleFigure。SimpleFigure 中提供了通用的 draw()、clear()、isOn()。

Area 是对所有可填充图形的抽象，用于填充点管理，提供了与填充相关的函数。

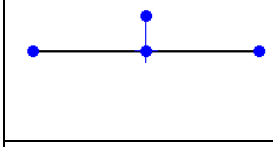
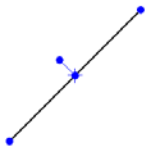
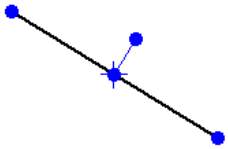
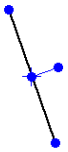
Line、Circle、Ellipse、Curve、Polygon 分别为直线、圆、椭圆、曲线、多边形。其中由于 Ellipse 和 Polygon 与 OpenGL 库中命名冲突，所以在实现中命名为 MyEllipse 和 MyPolygon，但相应代码文件名仍为 Ellipse 和 Polygon.

3 软件测试

3. 1 直线测试

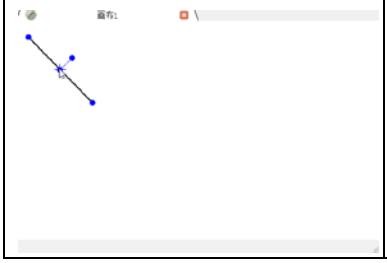
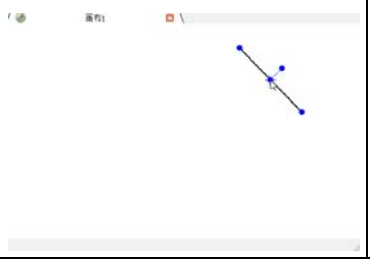
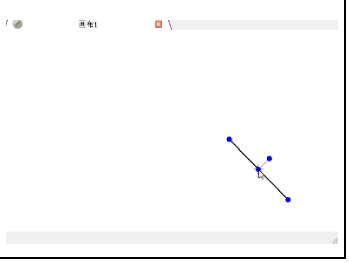
3.1.1 绘制

点击绘制不同斜率直线以验证算法正确性，在各种斜率情况下，直线绘制都有很好的效果，如下图

k=0	0<k<1	k=1	k>1
			
k 不存在	-1<k<0	k=-1	k<-1
			

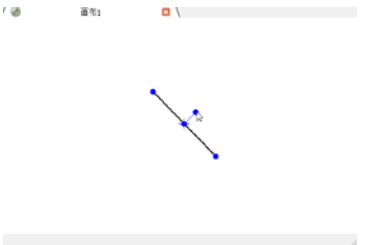
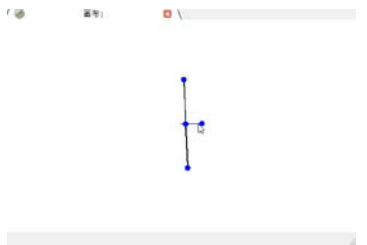
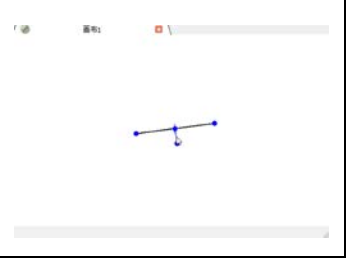
3.1.2 平移

拖动中心标记点改变直线位置，直线形状完全保持并跟随箭头自由移动，如下图

初始状态	右移一段距离	下移一段距离
		

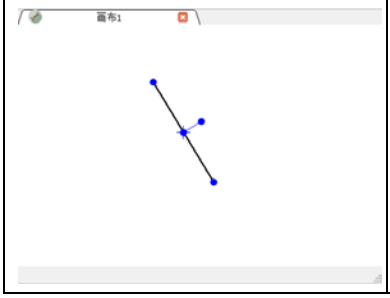
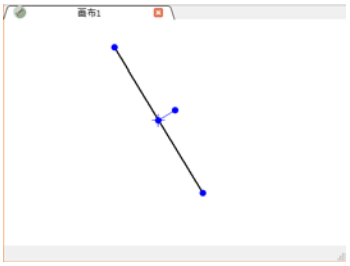
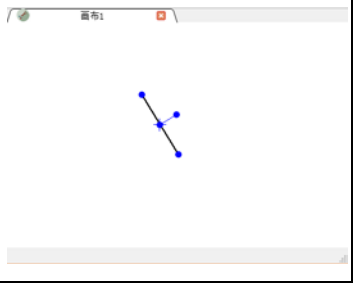
3.1.3 旋转

拖动旋转标记点让直线旋转，如下图

初始状态	顺时针旋转一个锐角	顺时针旋转一个钝角
		

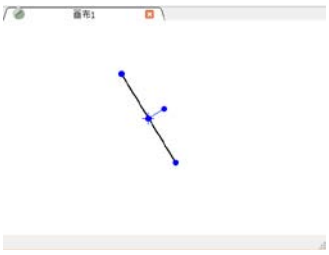
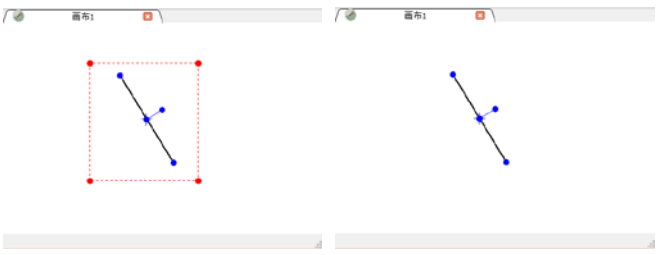
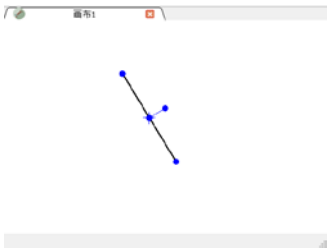
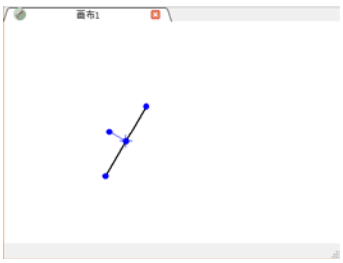
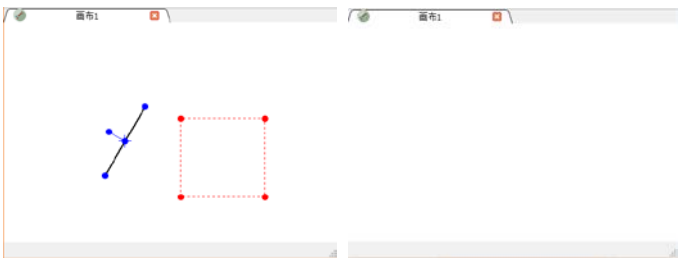
3.1.4 缩放

点击放大/缩小改变直线大小，如下图

初始状态	放大两次	缩小两次
		

3.1.5 裁剪

绘制不同的裁剪窗口对线段进行裁剪

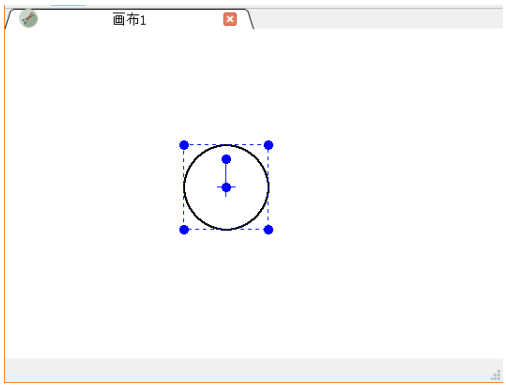
初始状态	全部在裁剪窗口内，直线没有任何变化
	
初始状态	部分在裁剪窗口内，在窗口外的直线部分被裁掉
	
初始状态	全部在裁剪窗口外，直线被删掉
	

3. 2 圆测试

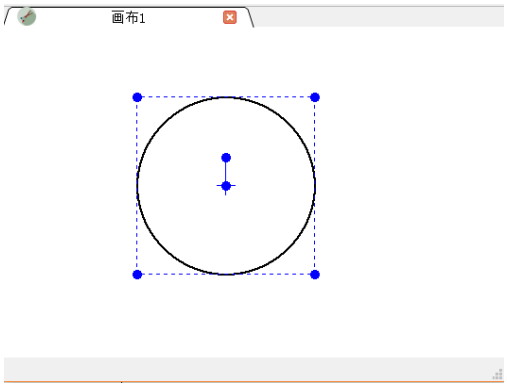
3.2.1 绘制

点击后拖动绘出一个圆

半径较小



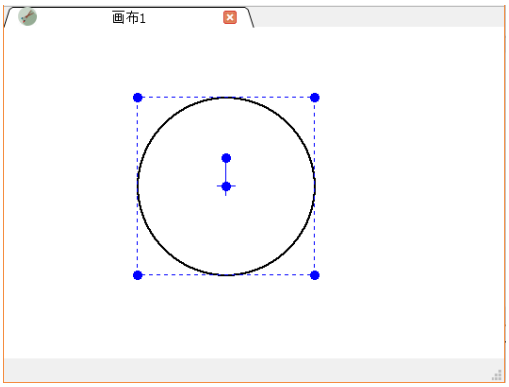
半径较大



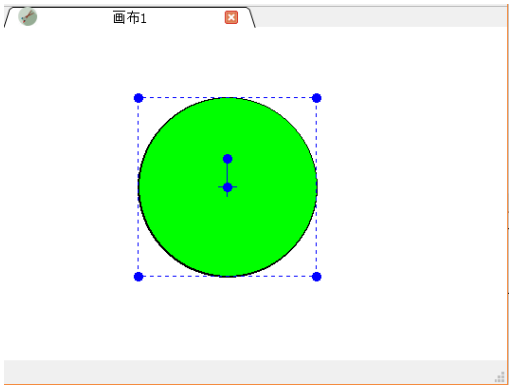
3.2.2 填充

点击填充图标进行填充

填充前



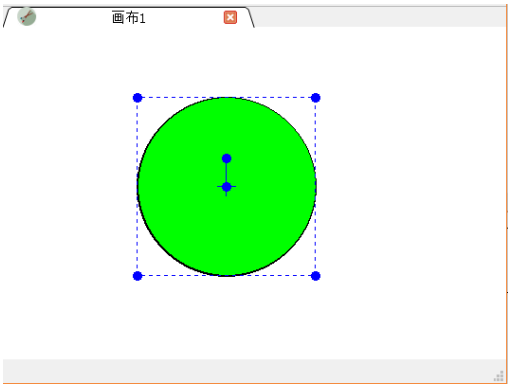
填充后



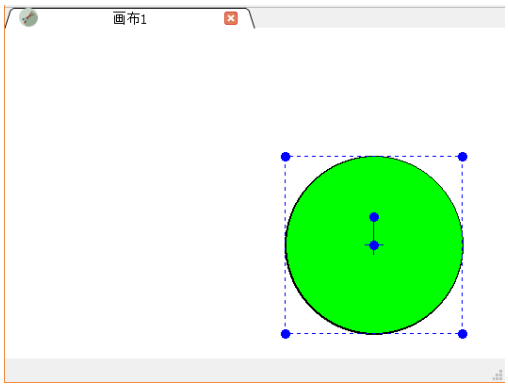
3.2.3 平移

拖动中心标记点移动圆

初始位置

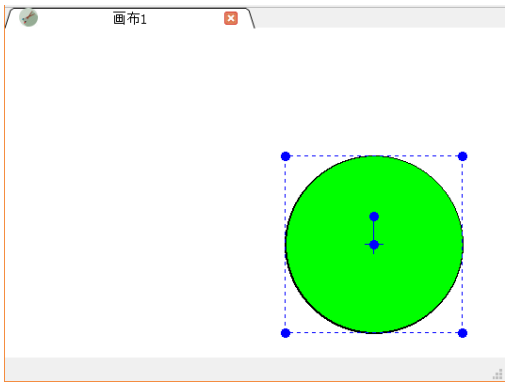
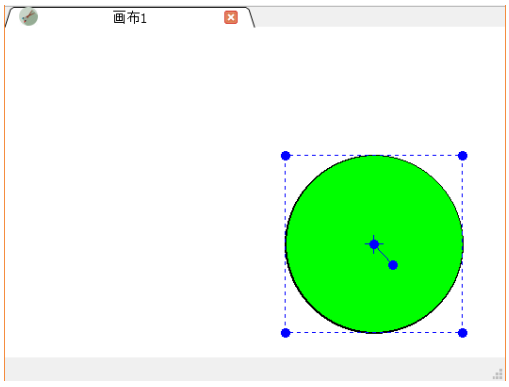


移动后



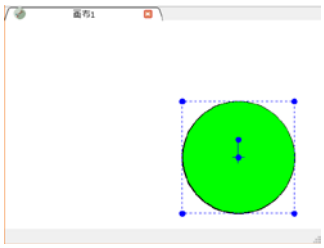
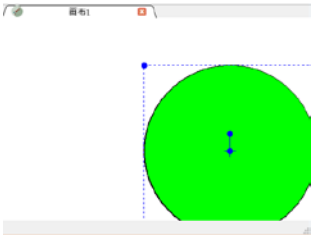
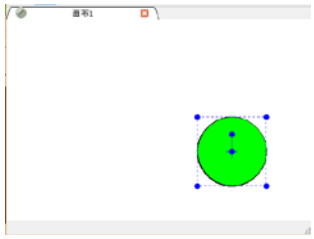
3.2.4 旋转

拖动旋转标记点旋转圆

初始状态	旋转一定角度后
	

3.2.5 缩放

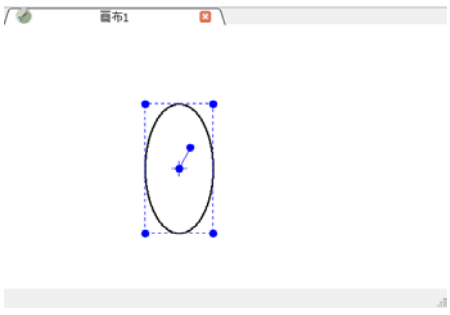
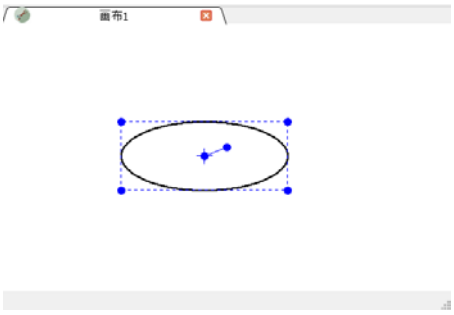
点击放大/缩小改变圆的大小

初始状态	放大	缩小
		

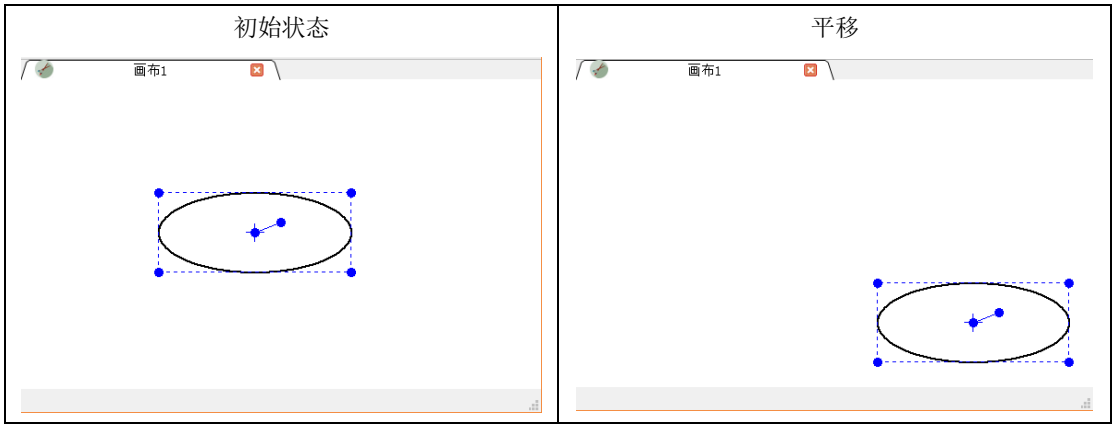
3. 3 椭圆测试

3.3.1 绘制

点击并拖动绘制出一个椭圆

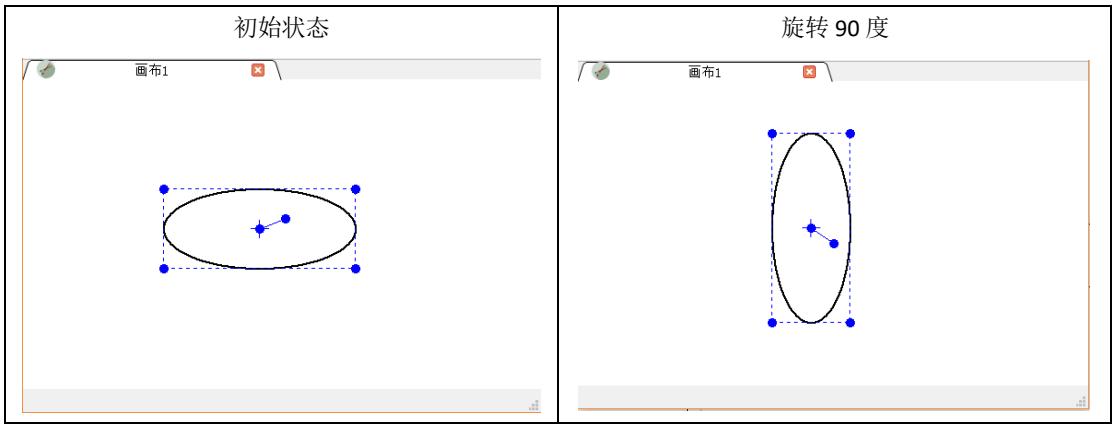
横轴>纵轴	横轴<纵轴
	

3.3.2 平移

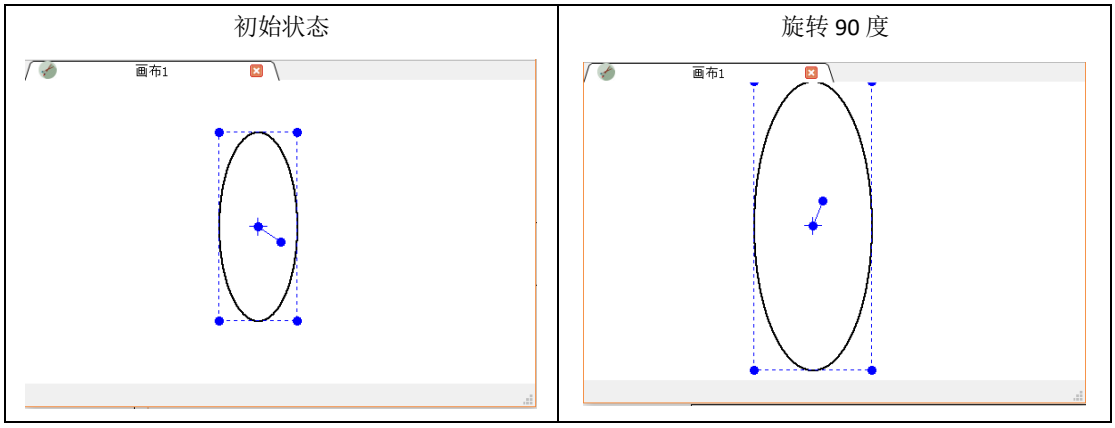


3.3.3 旋转

拖动旋转标记点至水平或垂直位置



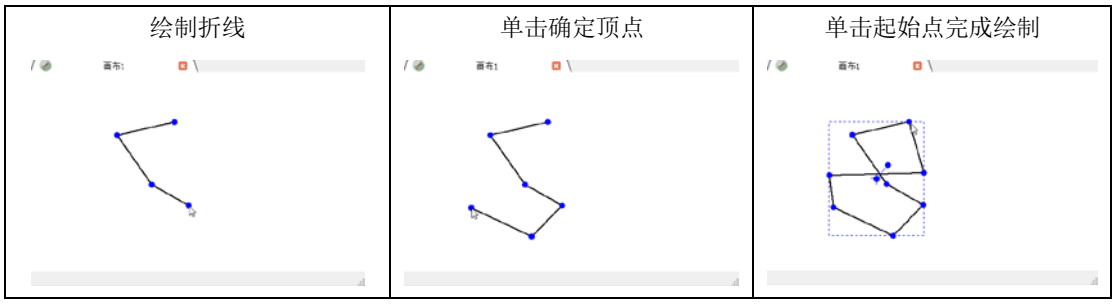
3.3.4 缩放



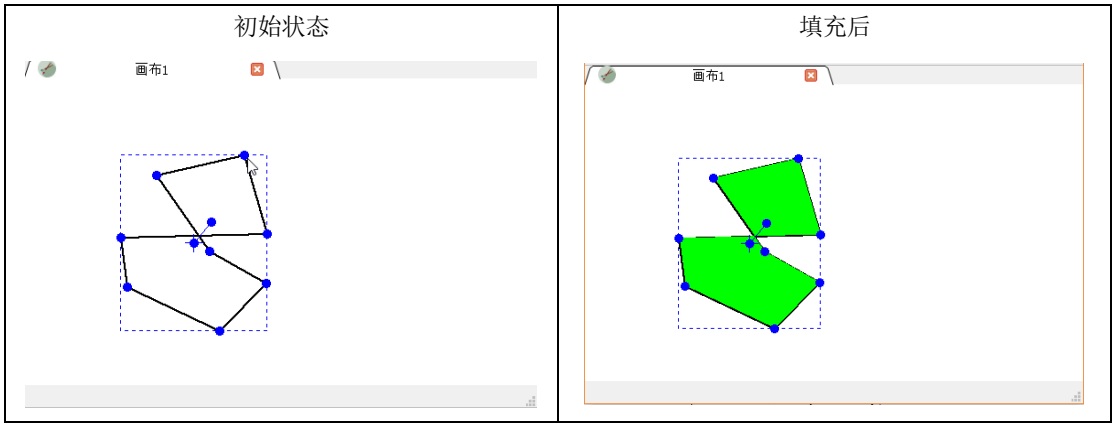
3. 4 多边形测试

3.4.1 绘制

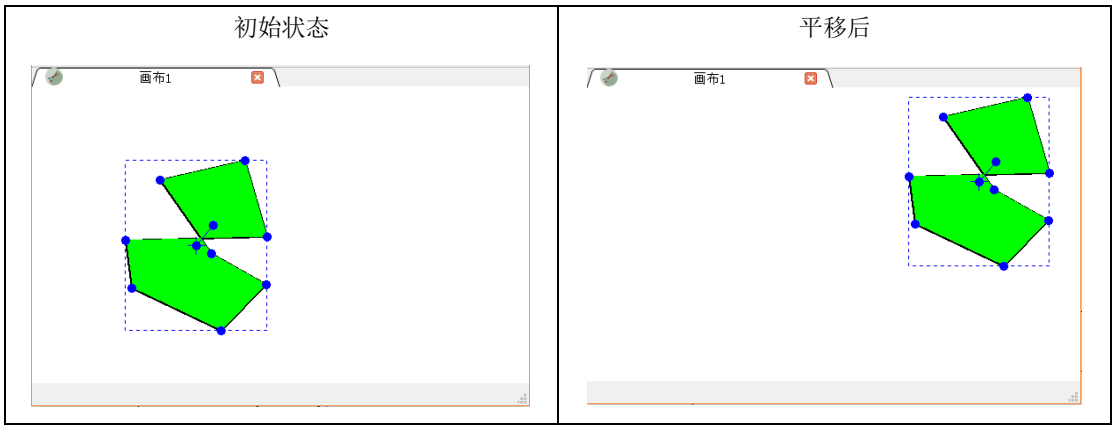
点击绘制多个点，以第一个点为终点得到一个多边形



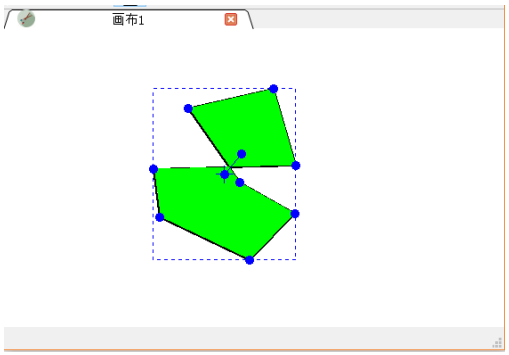
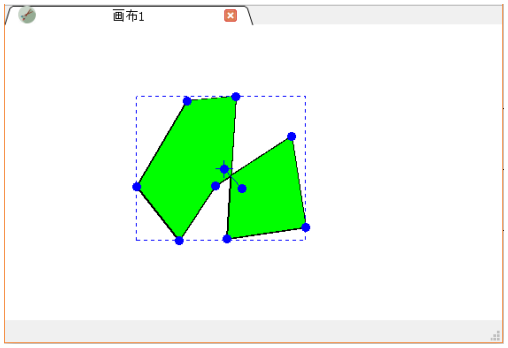
3.4.2 填充



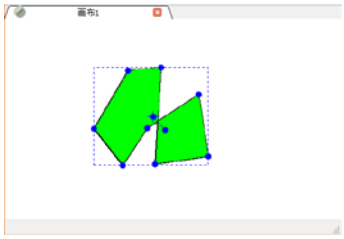
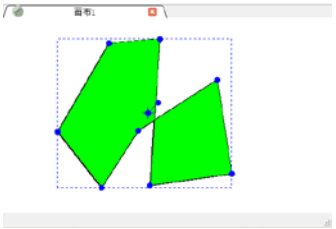
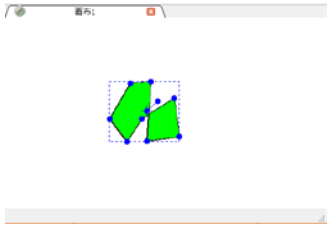
3.4.3 平移



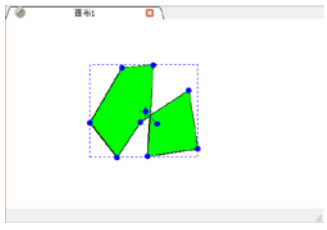
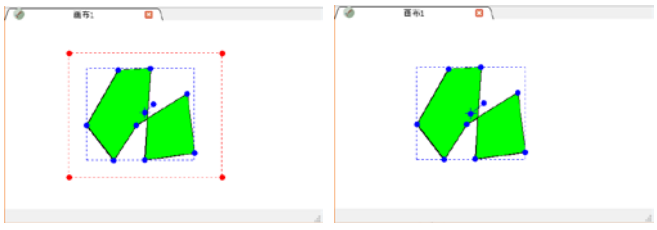
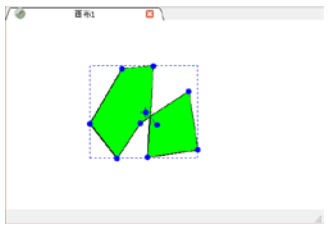
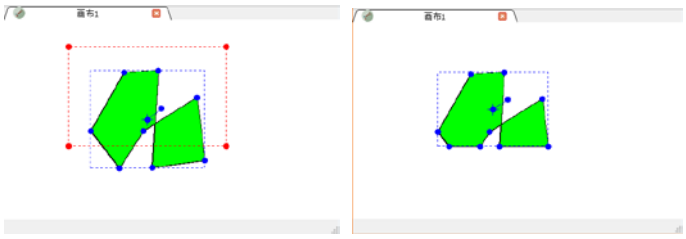
3.4.4 旋转

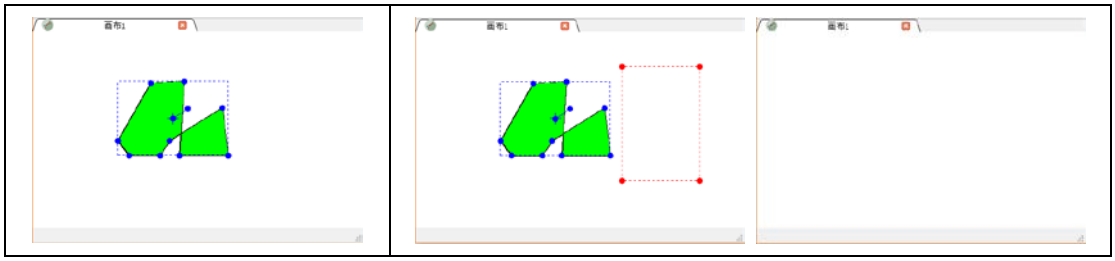
初始状态	旋转后
	

3.4.5 缩放

初始状态	放大	缩小
		

3.4.6 裁剪

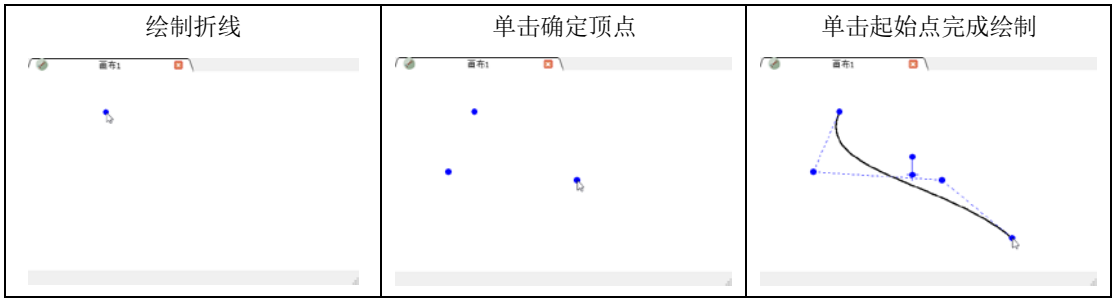
初始状态	全部在裁剪窗口内，没有任何变化
	
初始状态	部分在裁剪窗口内，在窗口外的部分被裁掉
	
初始状态	全部在裁剪窗口外，多边形被删掉



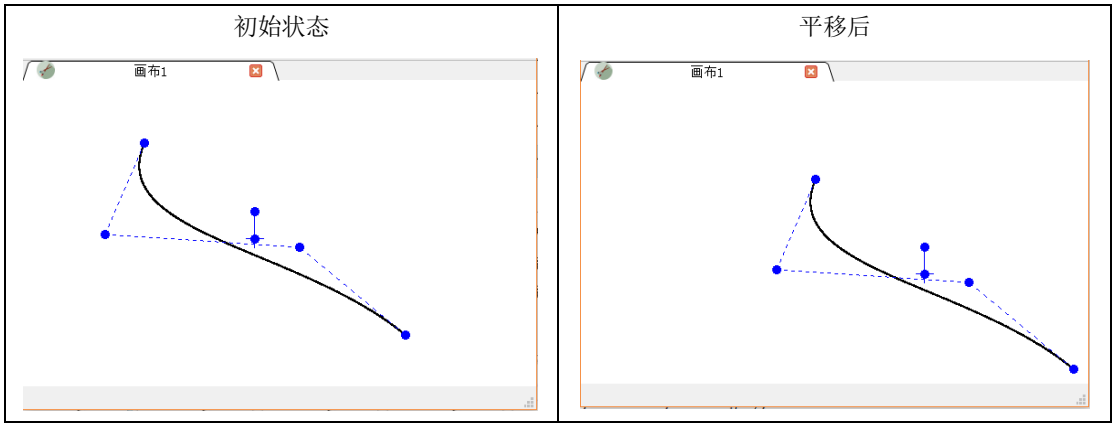
3.5 曲线测试

3.5.1 绘制

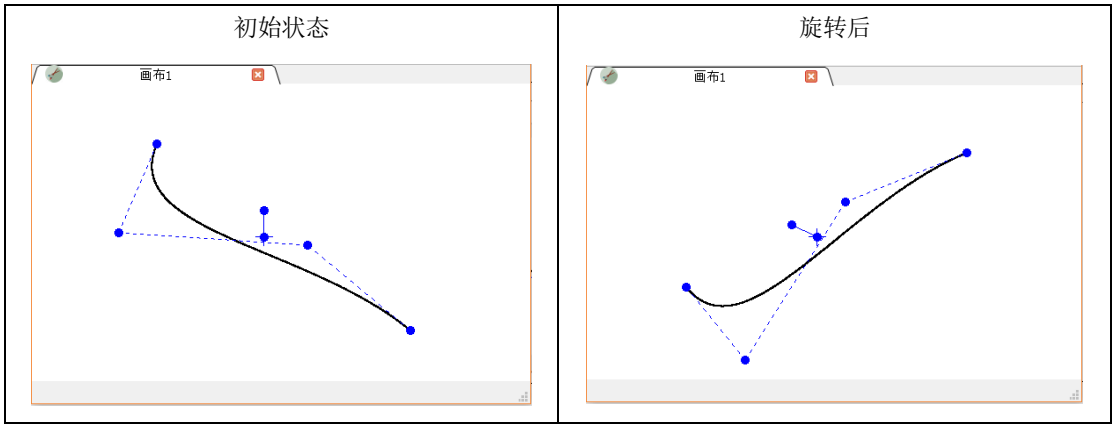
点击 4 个控制点自动生成贝塞尔曲线



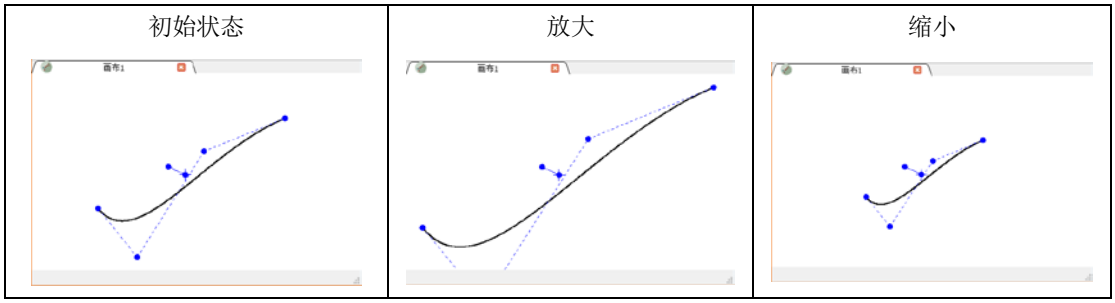
3.5.2 平移



3.5.3 旋转



3.5.4 缩放



4 结束语

本技术报告详尽地介绍了绘图系统所使用的所有算法，以及系统整体架构。报告中通过图表解释了消息如何从用户的鼠标键盘操作一层层传递到底层 OpenGL，然后绘制出丰富多彩的图形。另外，报告中也对图形绘制算法进行了充分的测试，以确保图形绘制能够方便快捷地完成。

参考文献

[1] 孙正兴,周良,郑洪源,谢强.计算机图形学教程.北京:机械工业出版社,2006.

[2] Edward Angel,段菲.OpenGL 编程基础(第 3 版).北京:清华大学出版社,2008.

[3] 陈家骏,郑滔.程序设计教程用 C++语言编程(第 3 版).北京:机械工业出版社,2015.