141220132 系统使用说明书

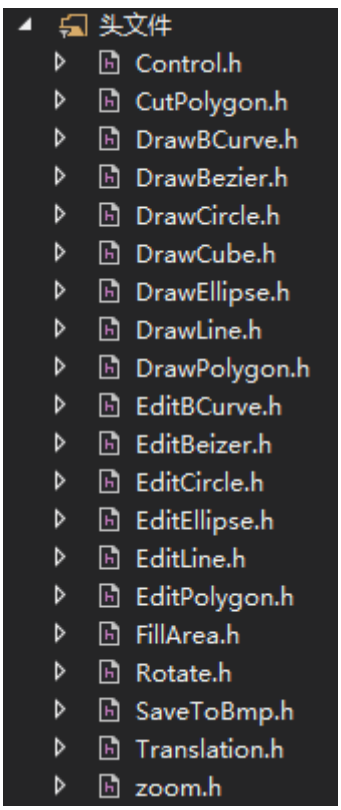
作者姓名 银琦

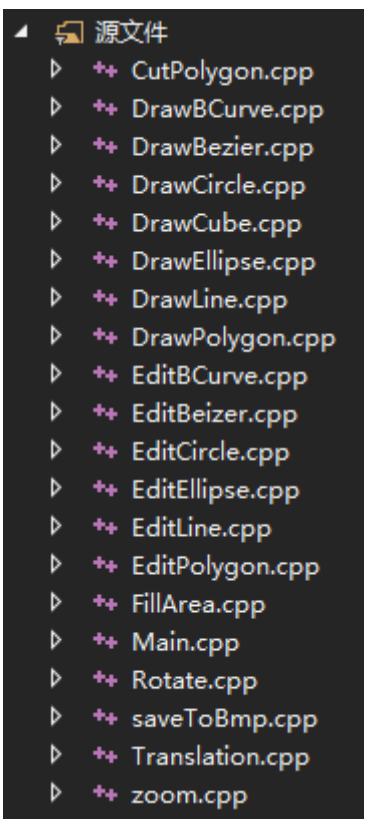
(南京大学 计算机科学与技术系，南京 210093)

摘 要: 使用 OpenGL 开发环境，实现了图形数据的输入、编辑、裁剪、变换、显示以及存储。

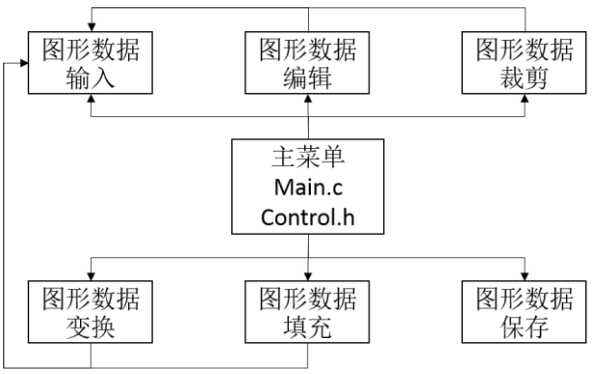
1 模块划分

1.1 程序的代码文件

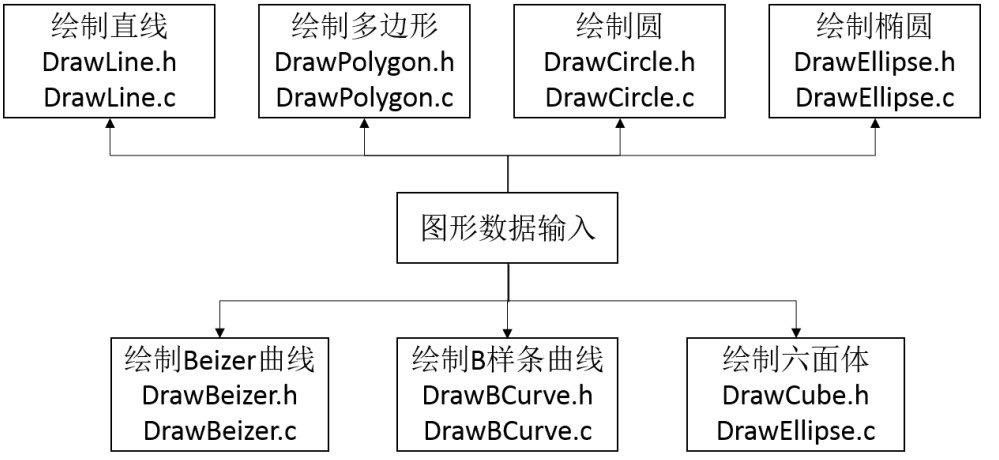




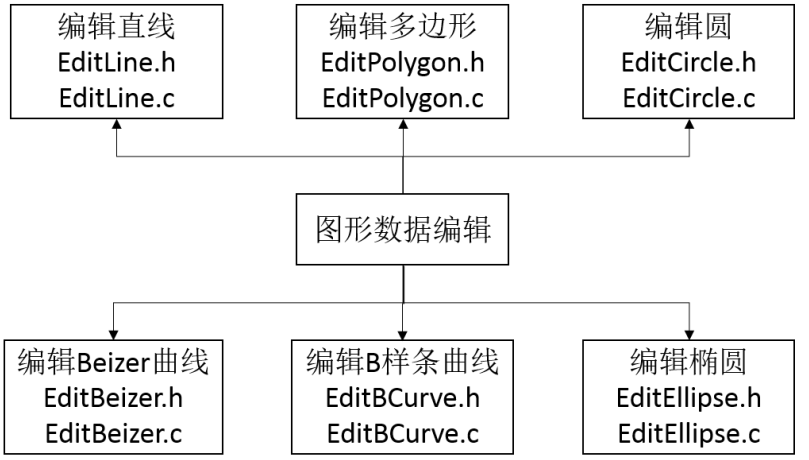
1.2 程序的总体模块划分图



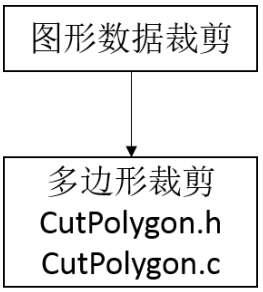
1.3 图形数据输入部分模块划分图



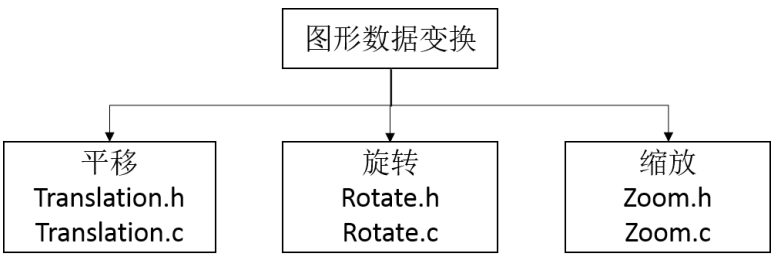
1.4 图形数据编辑部分模块划分图



1.5 图形数据裁剪部分模块划分图



1.6 图形数据变换部分模块划分图



2 各模块代码和实现功能介绍

2.1 主菜单

程序使用了在控制台添加菜单的方式与用户交互，供用户选择要操作的功能，鼠标右键点击可显示菜单，菜单项如图 2.1.1，包括图形数据输入，图形数据编辑，图形数据裁剪，图形数据变换，图形数据填充，图形数据保存，清空屏幕。

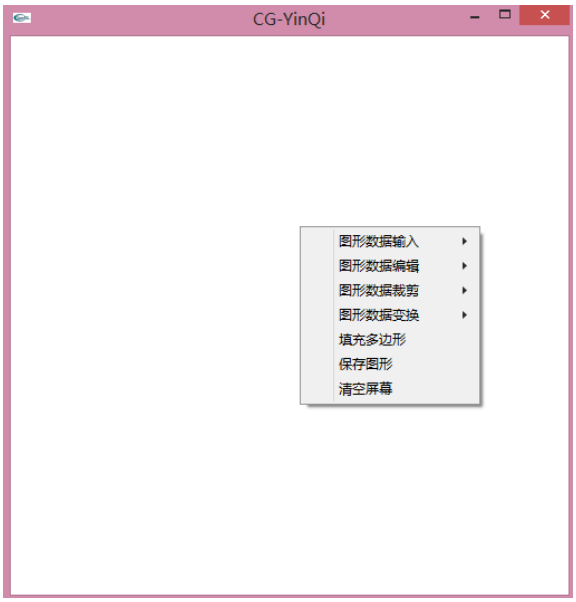


图 2.1.1

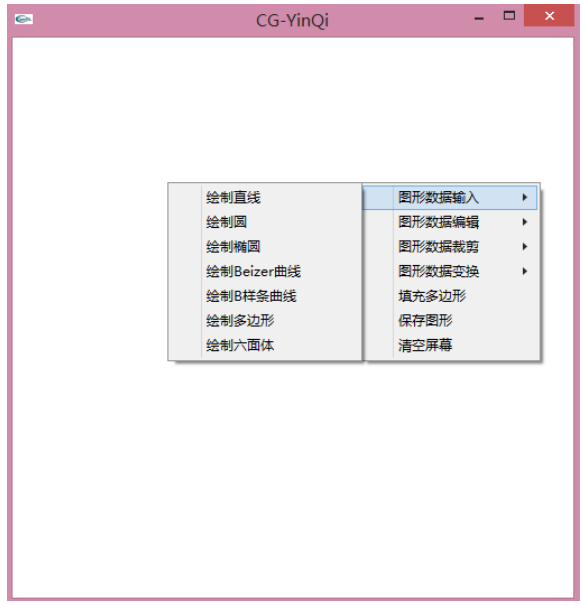


图 2.1.2

其中前四项还有子菜单，鼠标左键点击即可选择菜单项及子菜单项。

在图形数据输入中，可以选择绘制直线、绘制圆、绘制椭圆、绘制 Beizer 曲线、绘制 B 样条曲线、绘制多边形、绘制六面体（3D 多面体），如图 2.1.2。

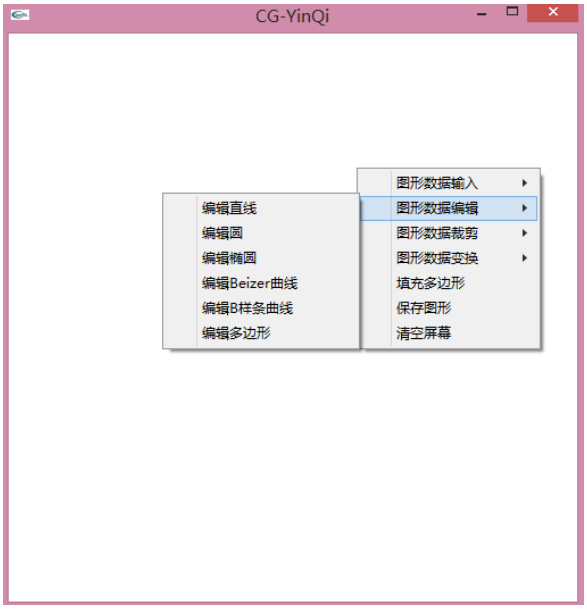


图 2.1.3

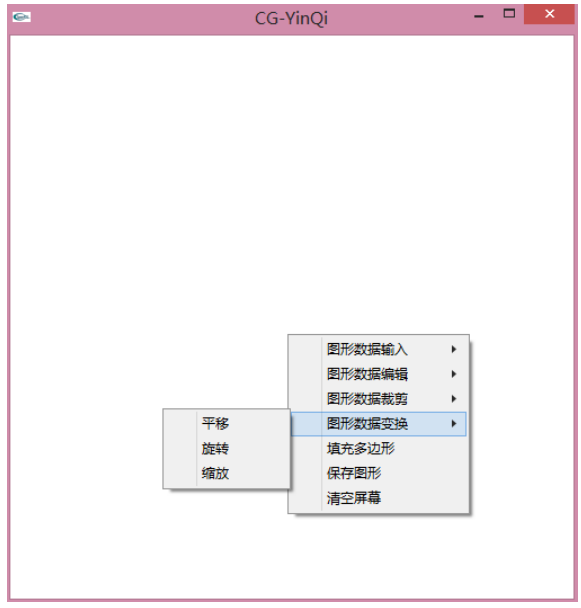


图 2.1.4

在图形数据编辑中，可以选择编辑直选、编辑圆、编辑椭圆、编辑 Beizer 曲线、编辑 B 样条曲线、编辑多边形，如图 2.1.3。

在图形数据裁剪中，包括对多边形的裁剪。

在图形数据变换中，包括对图形的平移、旋转、缩放，如图 2.1.4。

2.2 图形数据输入

2.2.1 绘制直线

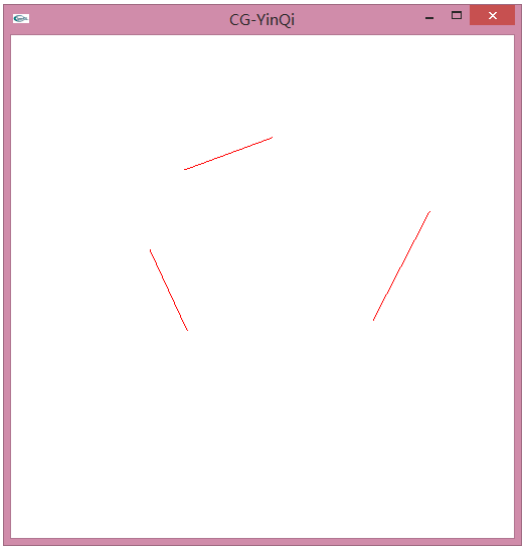
2.2.1.1 代码介绍

```
/*直线的数据结构*/
class linePt {
public:
    /*scrPt为自定义点的类型*/
    scrPt begin;  /*直线的起点*/
    scrPt end;    /*直线的终点*/
};
/*初始化一些变量*/
void initLine();
/*DDA绘制直线算法*/
void DDA(scrPt pointA, scrPt pointB);
/*绘制直线的鼠标操作*/
void drawLine(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*将直线存入数组*/
void saveLine(unsigned char key, int x, int y);
```

2.2.1.2 操作方式

鼠标左键点击屏幕，每次画直线鼠标只能点击两次，第一次点击为直线的起点，第二次点击为直线的终点，如果想继续画则需重新右键选择绘制直线功能。

2.2.1.3 运行结果



2.2.2 绘制圆

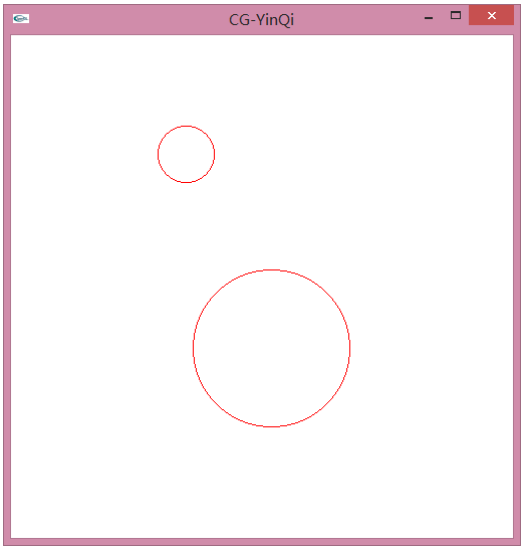
2.2.2.1 代码介绍

```
/*圆的数据结构*/
class circlePt {
public:
    scrPt center;
    double radius;
    double radiusY;     /*圆通过编辑成为椭圆后用*/
};
/*存储圆的数组*/
vector<circlePt> circle;
/*初始化一下变量*/
void initCircle();
/*画点*/
void setPixel(int x, int y);
/*中点圆算法画圆*/
void MidPoint_Circle(scrPt c, double r);
/*8路对称*/
void Cirpot(scrPt c, int x, int y);
/*绘制圆的鼠标操作*/
void drawCircle(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*将圆存入数组*/
void saveCircle(unsigned char key, int x, int y);
```

2.2.2.2 操作方式

鼠标左键点击屏幕，每次画圆鼠标只能点击两次，第一次点击为圆的圆心，第二次点击为圆上任意一点，如果想继续画则需重新右键选择绘制圆的功能。

2.2.2.3 运行结果



2.2.3 绘制椭圆

2.2.3.1 代码介绍

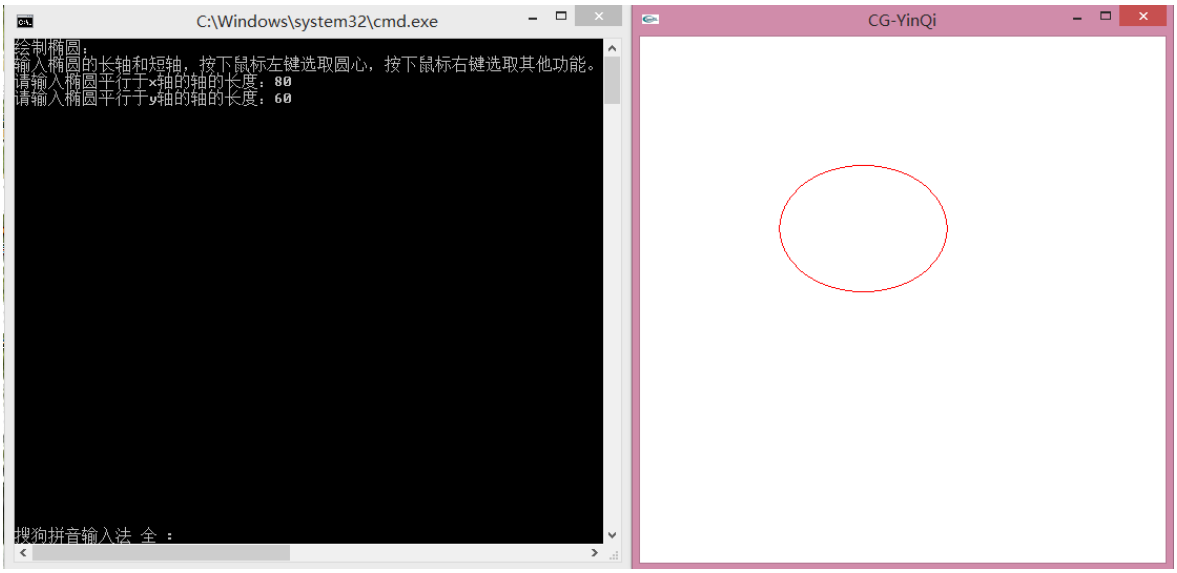
```
/*椭圆的数据结构*/
class ellipsePt {
public:
    scrPt center;
    double longAxis, minAxis;
};

/*存储椭圆的数组*/
vector<ellipsePt> ellipse;
/*初始化一些变量*/
void initEllipse();
/*画椭圆的鼠标操作*/
void drawEllipse(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*将椭圆存入数组*/
void saveEllipse(unsigned char key, int x, int y);
/*中点椭圆算法*/
void middle_ellipse(scrPt center, double longAxis, double minAxis);
```

2.2.3.2 操作方式

首先在控制台输入要绘制的椭圆的平行于 x 轴的轴的长度并按下回车，输入平行于 y 轴的轴的长度并按下回车，然后鼠标左键点击屏幕，即可绘制出椭圆，每次画椭圆只能点击屏幕一次，点击的点为椭圆的圆心。

注：只能绘制出长轴短轴平行于 x 轴和 y 轴的椭圆。

2.2.3.3 运行结果



2.2.4 绘制 Beizer 曲线

2.2.4.1 代码介绍

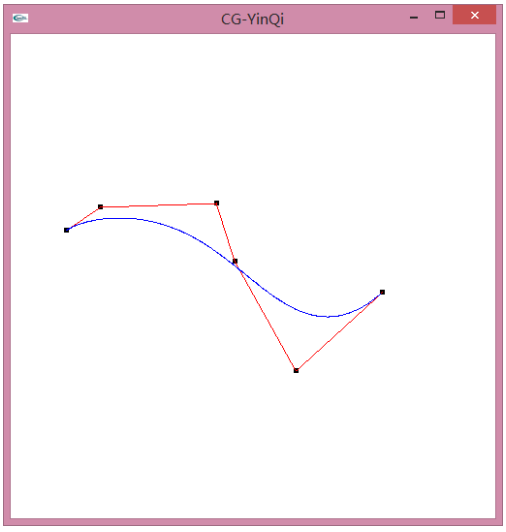
```
/*Beizer曲线的控制点的数据结构*/
class BezierCurvePt {
public:
    vector<scrPt> beizerPt;
};
/*存储Beizer曲线点的数组*/
vector<BezierCurvePt> bePt;
/*初始化一些变量*/
void initBeizer();
/*画Beizer曲线的鼠标操作*/
void drawBeizer(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*画Beizer曲线的算法*/
void drawBe(vector<scrPt>p);
/*求n!*/
int factorial(int n);
/*求组合排列*/
double C(int n, int i);
/*存储Beizer曲线的控制点*/
void savePoint(scrPt p);
/*键盘回调函数，结束绘制*/
void finishDraw(unsigned char key, int x, int y);
```

2.2.4.2 操作方式

鼠标左键点击屏幕，选取控制点，Beizer 曲线会实时的画出，控制点的个数不限，选取结束后按下回车结束选点。

注：当控制点的个数达到 14-15 个及以上时，曲线会变形，控制点个数少于 3 时不会绘制出曲线。

2.2.4.3 运行结果



2.2.5 绘制 B 样条曲线

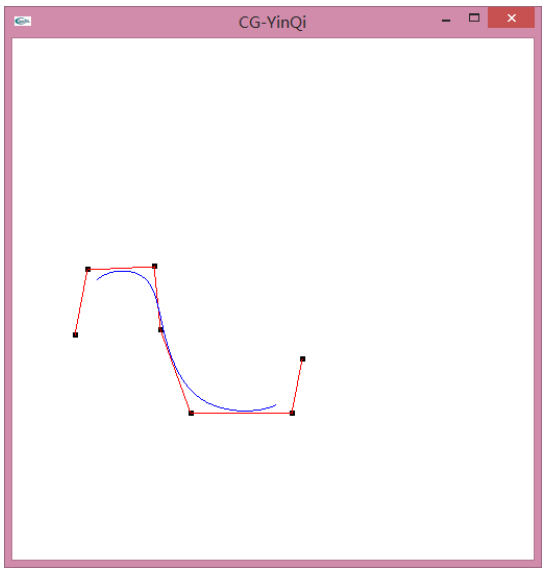
2.2.5.1 代码介绍

```
/*B样条曲线控制点的数据结构*/
class BCurvePt {
public:
    vector<scrPt> bPt;
};
/*存储B样条曲线控制点的数组*/
vector<BCurvePt> bCurve;
/*绘制B样条曲线的算法*/
void drawB(vector<scrPt>p);
/*初始化一些变量*/
void initBCurve();
/*绘制B样条曲线的鼠标操作*/
void drawBCurve(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*求n!*/
int factorial(int n);
/*求组合排列*/
double C(int n, int i);
/*存储B样条曲线的控制点*/
void savePoint(scrPt p);
/*键盘回调函数，结束绘制*/
void finishDraw(unsigned char key, int x, int y);
```

2.2.5.2 操作方式

鼠标左键点击屏幕，选取控制点，B 样条曲线不会实时的画出，控制点的个数不超过 25 个，选取结束后按下回车结束选点，并绘制出 B 样条曲线。

2.2.5.3 运行结果



2.2.6 绘制多边形

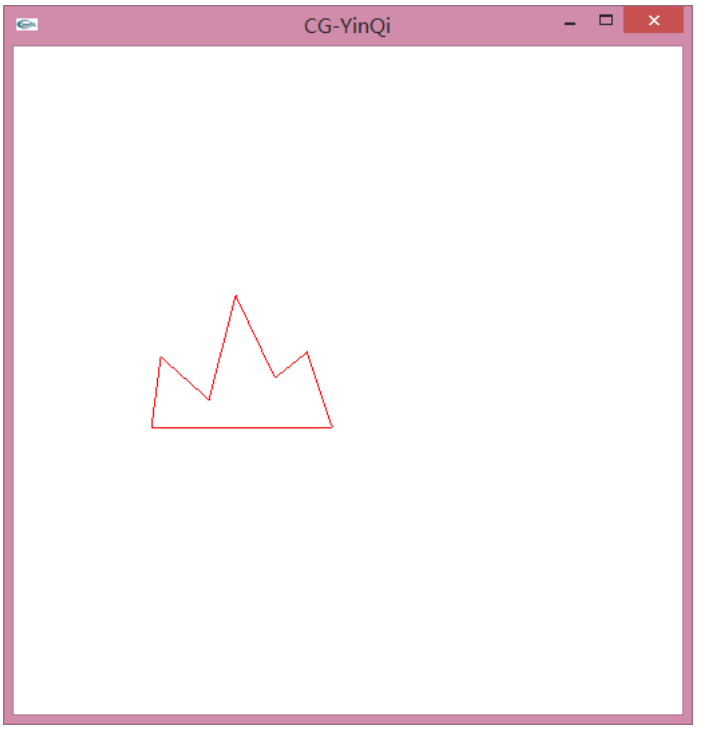
2.2.6.1 代码介绍

```
/*多边形的数据结构*/
class polygonPt {
public:
    vector<scrPt> point;
    char color;
};
/*存储多边形的数组*/
vector<polygonPt> polygon;
/*初始化一些变量*/
void initPolygon();
/*绘制多边形的鼠标操作*/
void drawPolygon(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*存储多边形的顶点*/
void savePoint(scrPt p);
/*键盘回调函数，结束绘制*/
void finishDraw(unsigned char key, int x, int y);
```

2.2.6.2 操作方式

鼠标左键点击屏幕，选取多边形的顶点，多次点击，连续绘制出直线，当选点结束后按下回车结束选点，同时添加多边形的最后一条边，完成多边形的绘制。

2.2.6.3 运行结果



2.2.7 绘制三维六面体

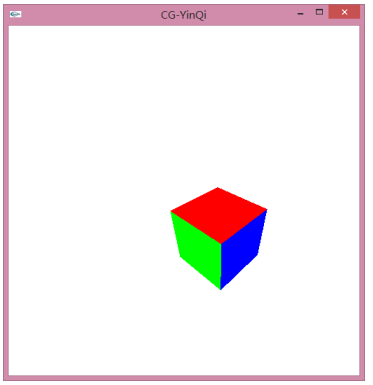
2.2.7.1 代码介绍

```
/*用于记录鼠标位置*/
struct PointMouse{
    int x;
    int y;
};
/*用于摄像机定位*/
struct CAMERA{
    GLfloat   xeye;
    GLfloat yeye;
    GLfloat zeye;
};
struct POLAR{/*球坐标*/
    float r;                    /*距离r*/
    float alpha;                      /*水平偏角α*/
    float fy;                  /*竖直偏角φ（单位均用角度）*/
};
void initDrawCube();/*初始化场景*/
/*将球坐标转化为直角坐标，并设定为摄像机的位置*/
void SetCamera(GLfloat x, GLfloat y);
void drawC();/*绘制图像*/
/*绘制好六面体后的鼠标操作*/
void drawCube(int button, int state, int x, int y);
void OnMouseMove(int x, int y); /*绘制好六面体后的鼠标操作*/
void Reshape(int w, int h); /*改变窗口大小时使用*/
/*键盘回调函数，退出子窗口时用*/
void Keyboard(unsigned char key, int x, int y);
```

2.2.7.2 操作方式

在菜单项中点击绘制六面体后会新建子窗口并自动绘制出六面体，按下鼠标并拖动可以旋转六面体。按下 Esc 可以退出子窗口，返回绘图界面。

2.2.7.3 运行结果



2.3 图形数据编辑

2.3.1 编辑直线

2.3.1.1 代码介绍

```
/*初始化一些变量*/
void initEditLine();
/*编辑直线的鼠标点击操作*/
void editLine(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*编辑直线的鼠标拖动操作*/
void editL(int x, int y);
/*计算鼠标点击的点到直线两端点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的点是直线的哪个端点*/
int judgePoint(scrPt click);
```

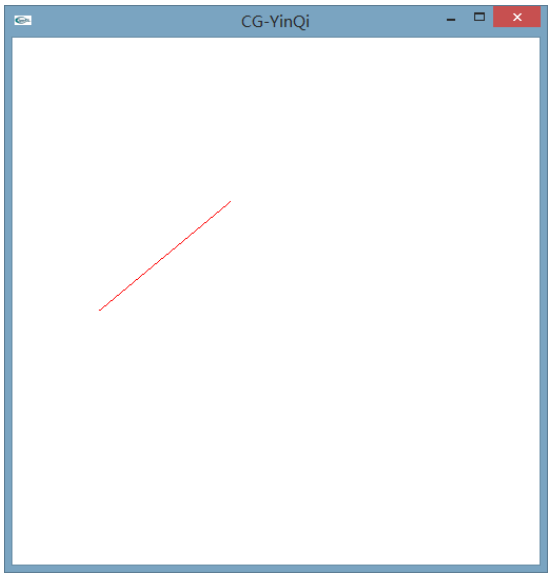
2.3.1.2 操作方式

在编辑直线模式下只可以编辑之前绘制的最后一条直线，鼠标左键选择直线端点，然后拖动鼠标，可以使该端点的位置移动并重新绘制出直线。

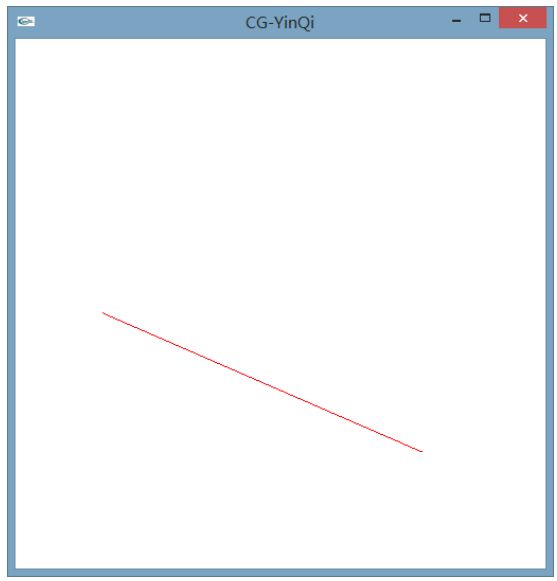
注：编辑直线的前提是已经绘制出了直线，否则无法编辑。以下所有对图形的编辑同理。

2.3.1.3 运行结果

鼠标点击直线的右端点并向下拖动。



直线编辑前



直线编辑后

2.3.2 编辑圆

2.3.2.1 代码介绍

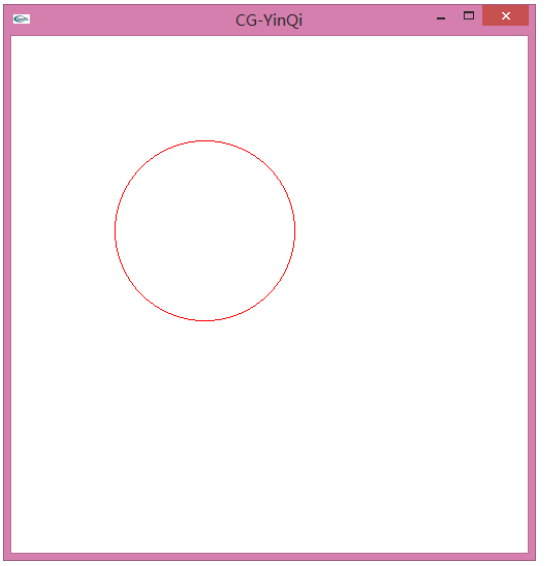
```
/*初始化一些变量*/
void initEditCircle();
/*编辑圆的鼠标点击操作*/
void editCircle(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*编辑圆的鼠标移动操作*/
void editC(int x, int y);
/*计算鼠标点击的点到圆平行于坐标轴的直径的端点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的是圆平行于坐标轴的直径的哪个端点*/
int judgePoint(scrPt click);
```

2.3.2.2 操作方式

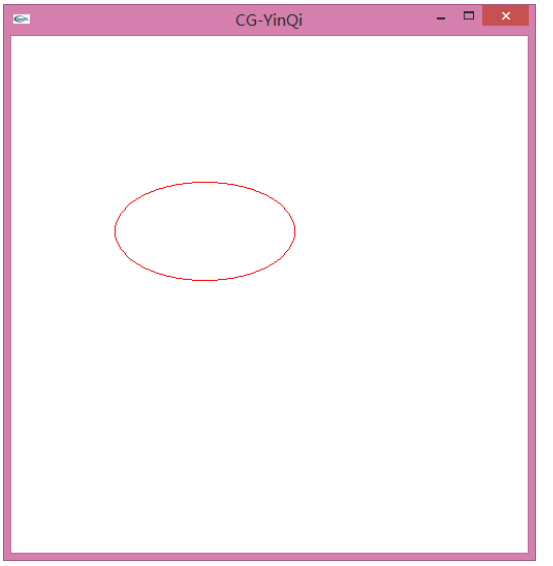
在编辑圆的模式下只可以编辑之前绘制的最后一个圆，设圆平行于坐标轴的直径的端点分别为 P1(上), P2(下), P3(左), P4(右)，点击 P1 或 P2 并在垂直方向移动鼠标或者点击 P3 或 P4 并在水平方向移动鼠标，可以将圆变成椭圆。

2.3.2.3 运行结果

鼠标点击圆的 P1（上方的）点并向下拖动。



圆编辑前



圆编辑后

2.3.3 编辑椭圆

2.3.3.1 代码介绍

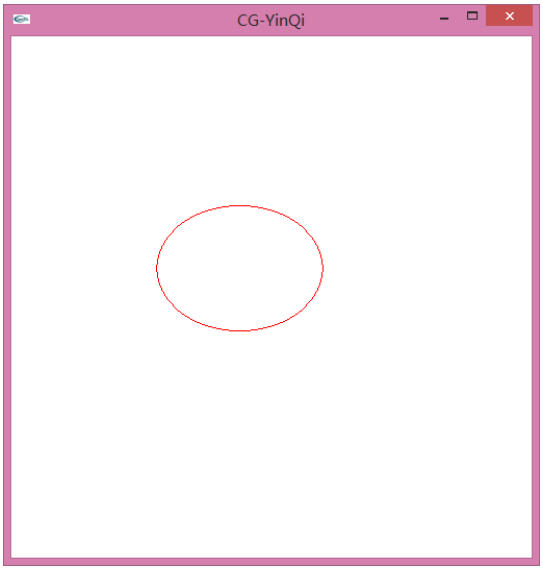
```
/*初始化一些变量*/
void initEditEllipse();
/*编辑椭圆的鼠标点击操作*/
void editEllipse(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*编辑椭圆的鼠标拖动操作*/
void editE(int x, int y);
/*计算鼠标点击的点到椭圆平行于坐标轴的长短轴的端点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的是椭圆平行于坐标轴的长短轴的哪个端点*/
int judgePoint(scrPt click);
```

2.3.3.2 操作方式

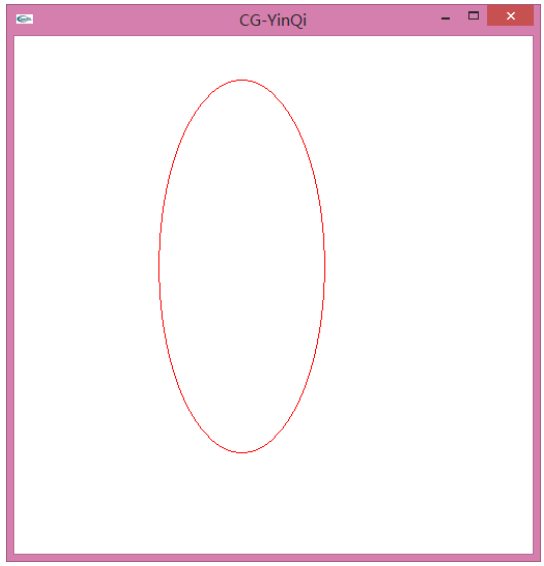
在编辑椭圆的模式下只可以编辑之前绘制的最后一个椭圆，设椭圆平行于坐标轴的长短轴的端点分别为 P1(上), P2(下), P3(左), P4(右)，点击 P1 或 P2 并在垂直方向移动鼠标或者点击 P3 或 P4 并在水平方向移动鼠标，可以改变椭圆形状。

2.3.3.3 运行结果

点击 P2（下方的）点并向下拖动。



椭圆编辑前



椭圆编辑后

2.3.4 编辑 Beizer 曲线

2.3.4.1 代码介绍

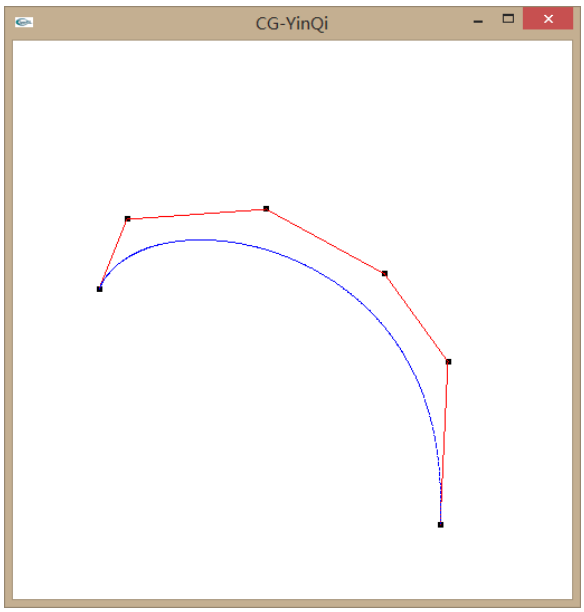
```
/*初始化一些变量*/
void initEditBeizer();
/*编辑Beizer曲线的鼠标点击操作*/
void editBeizer(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*编辑Beizer曲线的鼠标拖动操作*/
void editBe(int x, int y);
/*计算鼠标点击的点到Beizer曲线各控制点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的是Beizer曲线的哪个控制点*/
int judgePoint(scrPt click);
```

2.3.4.2 操作方式

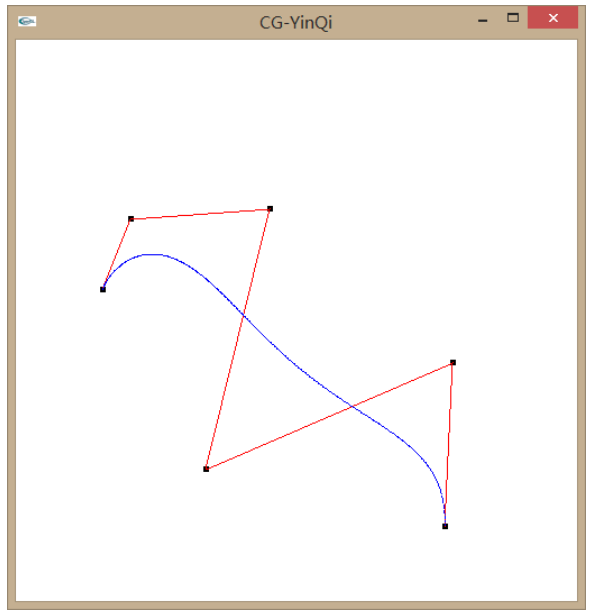
在编辑 Beizer 曲线的模式下只可以编辑之前绘制的最后一条 Beizer 曲线，点击 Beizer 曲线的任意一个控制点并拖动，可以更改该控制点的位置，并重新绘制 Beizer 曲线。

2.3.4.3 运行结果

改变了第三个控制点的位置。



Beizer 曲线编辑前



Beizer 曲线编辑后

2.3.5 编辑 B 样条曲线

2.3.5.1 代码介绍

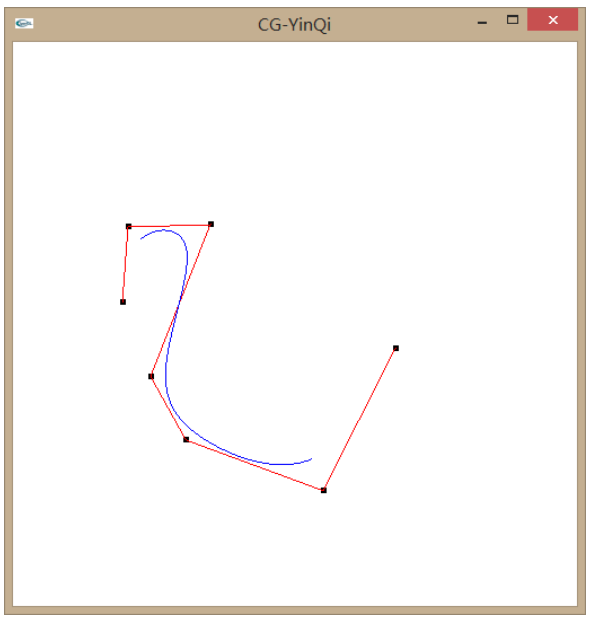
```
/*初始化一些变量*/
void initEditBCurve();
/*编辑B样条曲线的鼠标点击操作*/
void editBCurve(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*编辑B样条曲线的鼠标拖动操作*/
void editB(int x, int y);
/*计算鼠标点击的点到B样条曲线各控制点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的是B样条曲线的哪个控制点*/
int judgePoint(scrPt click);
```

2.3.5.2 操作方式

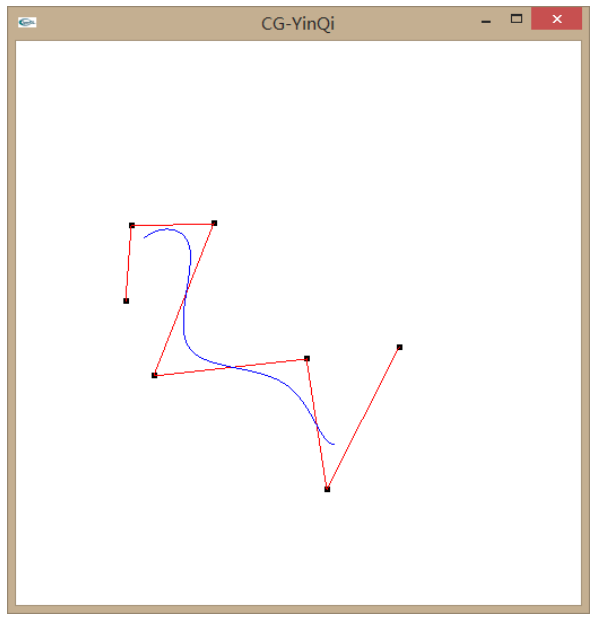
在编辑 B 样条曲线的模式下只可以编辑之前绘制的最后一条 B 样条曲线，点击 B 样条曲线的任意一个控制点并拖动，可以更改该控制点的位置，并重新绘制 B 样条曲线。

2.3.5.3 运行结果

更改了第五个控制点的位置。



B 样条曲线编辑前



B 样条曲线编辑后

2.3.6 编辑多边形

2.3.6.1 代码介绍

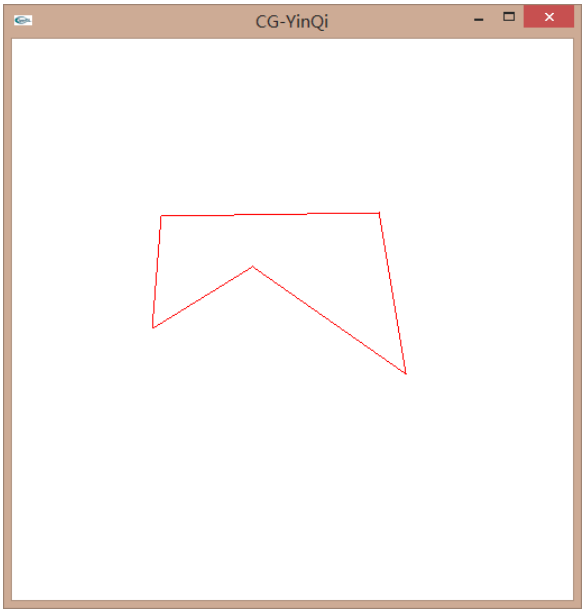
```
/*初始化一些变量*/
void initEditPolygon();
/*编辑多边形的鼠标点击操作*/
void editPolygon(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*编辑多边形的鼠标拖动操作*/
void editP(int x, int y);
/*计算鼠标点击的点到多边形各顶点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的是多边形的哪个顶点*/
int judgePoint(scrPt click);
```

2.3.6.2 操作方式

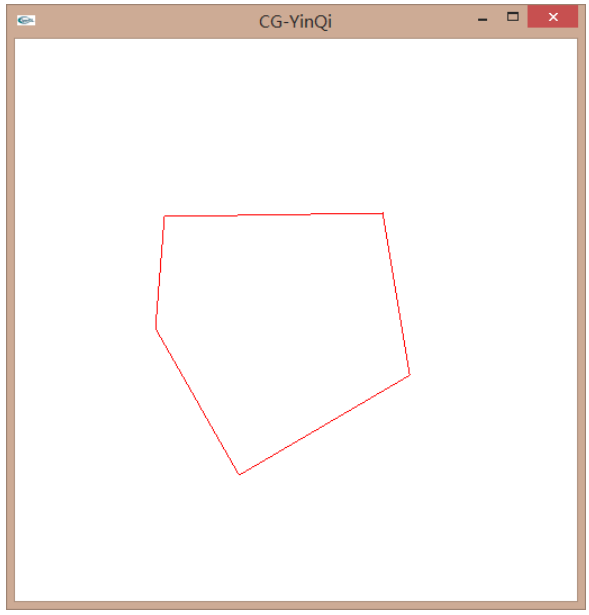
在编辑多边形的模式下只可以编辑之前绘制的最后一个多边形，点击多边形的任意一个顶点并拖动，可以更改该顶点的位置，并重新绘制该顶点的相邻两边。

2.3.6.3 运行结果

更改了多边形一个顶点的位置。



多边形编辑前



多边形编辑后

.

2.4 图形数据裁剪

2.4.1 裁剪多边形

2.4.1.1 代码介绍

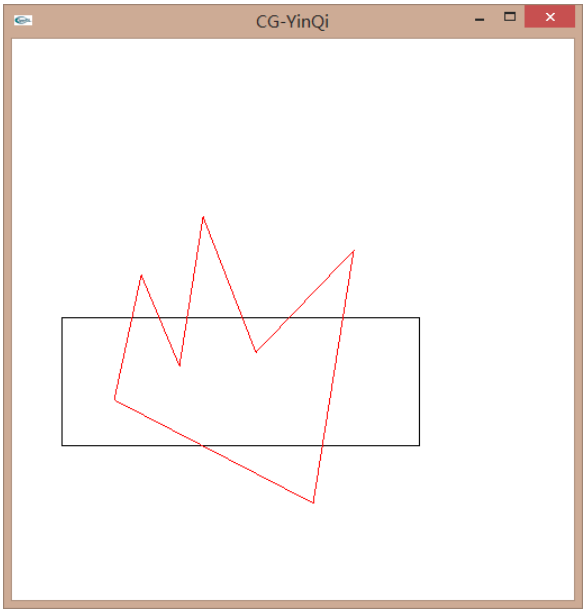
```
/*裁剪框的数据结构*/
class cutFrame {
public:
    scrPt leftUp;
    scrPt leftDown;
    scrPt rightUp;
    scrPt rightDown;
};
/*裁剪后的多边形的顶点*/
struct Point {
    scrPt p;
    int code;
} cfPoint[4], p1[20], p2[20], p3[20], p4[20];
/*初始化一些变量*/
void initCutPolygon();
/*绘制裁剪框*/
void drawCf(int x, int y);
/*编辑裁剪框*/
void editCf(int x, int y);
/*保存裁剪框*/
void saveCf();
/*裁剪多边形的鼠标操作*/
void cutPolygon(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*计算鼠标点击的点到裁剪框各顶点的距离*/
double calcuDis(scrPt p1, scrPt p2);
/*判断鼠标点击的是裁剪框的哪个顶点*/
int judgePoint(scrPt click);
/*左边界裁剪*/
void leftCut();
/*右边界裁剪*/
void rightCut();
/*下边界裁剪*/
void downCut();
//上边界裁剪
void upCut();
/*裁剪算法*/
void Sutherland_Hodgman();
/*键盘回调函数，完成裁剪*/
void finishDraw(unsigned char key, int x, int y)
```

2.4.1.2 操作方式

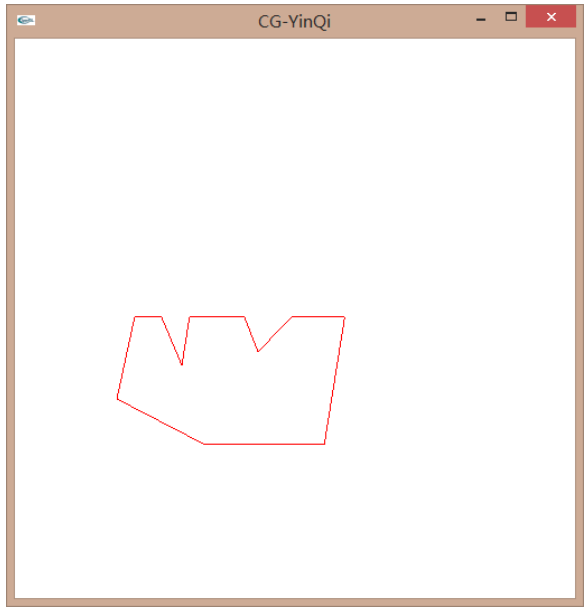
裁剪多边形时只可以裁剪最后一次绘制的多边形，如果没有绘制过多边形则无法裁剪。按下鼠标并拖动可绘制出裁剪框，点击裁剪框的四个顶点并拖动可以编辑裁剪框，编辑完成后按下回车执行裁剪。

2.4.1.3 运行结果

多边形裁剪前



绘制裁剪框



多边形裁剪后

2.5 图形数据变换

在图形数据变换模式下只能对最后一次绘制的图形进行图形变换！

2.5.1 平移

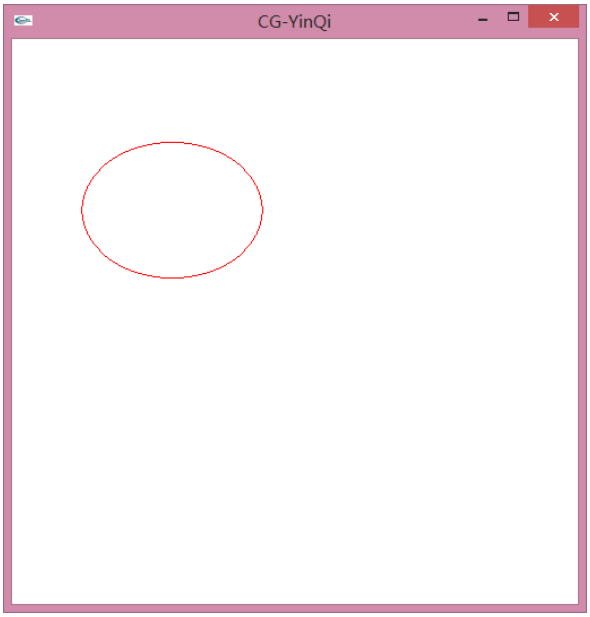
2.5.1.1 代码介绍

```
/*初始化一些变量*/
void initTranslation();
/*判断移动形状*/
void judgeShape();
/*平移图形时的鼠标点击操作*/
void translation(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*平移直线*/
void translateLine(int x, int y);
/*平移圆*/
void translateCircle(int x, int y);
/*平移椭圆*/
void translateEllipse(int x, int y);
/*平移Beizer曲线*/
void translateBeizer(int x, int y);
/*平移B样条曲线*/
void translateBCurve(int x, int y);
/*平移多边形*/
void translatePolygon(int x, int y);
```

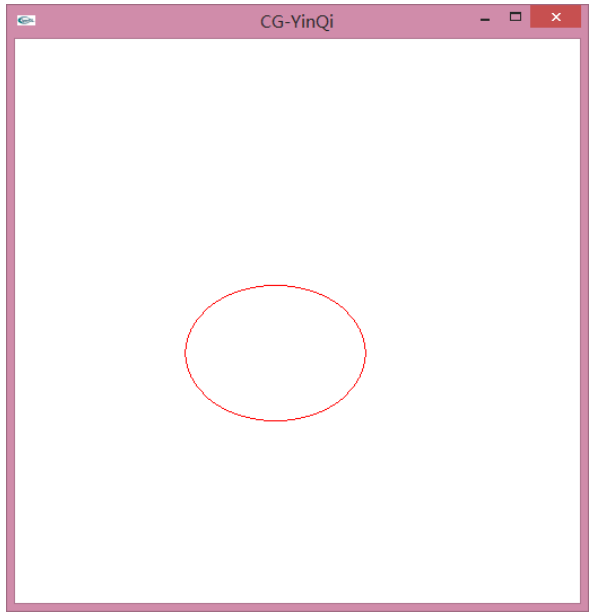
2.5.1.2 操作方式

按下鼠标左键并拖动，相应的图形会随着鼠标的移动而移动。

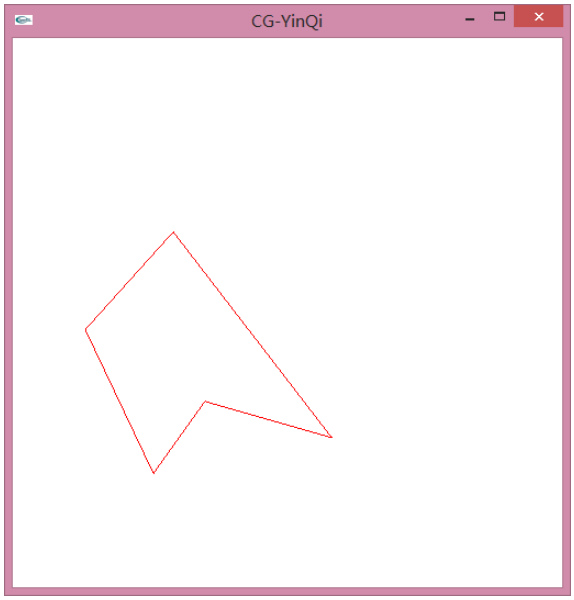
2.5.1.3 运行结果



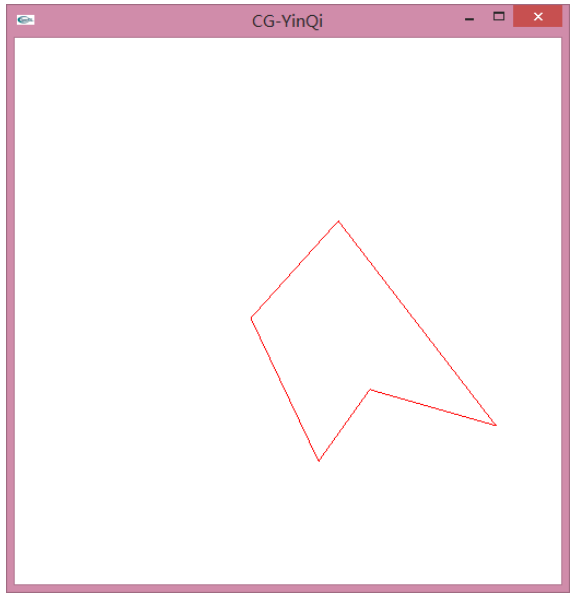
椭圆平移前



椭圆平移后



多边形平移前



多边形平移后

2.5.2 旋转

2.5.2.1 代码介绍

```
/*初始化一些变量*/
void initRotate();
/*判断旋转的图形*/
void judgeShapeR();
/*旋转的鼠标操作*/
void rotate(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*获取旋转的角度*/
double getAngle(int cX, int cY, int mX, int mY);
/*判断旋转的方向是顺时针还是逆时针*/
int getDirection(scrPt fP,scrPt lP, int x, int y);
/*旋转直线*/
void rotateLine(int x, int y);
/*旋转Beizer曲线*/
void rotateBeizer(int x, int y);
/*旋转B样条曲线*/
void rotateBCurve(int x, int y);
/*旋转多边形*/
void rotatePolygon(int x, int y);
/*旋转圆*/
void rotateCircle(int x, int y);
```

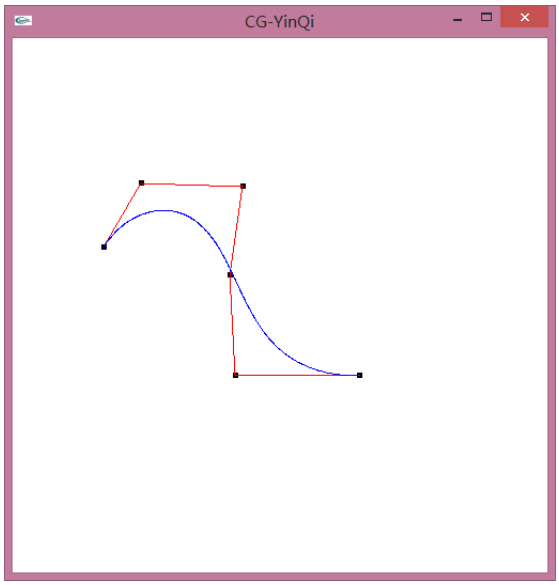
2.5.2.2 操作方式

首先在屏幕上点击一下，确定旋转中心，然后按下鼠标并顺时针或逆时针拖动，图形随着鼠标的移动而转动。

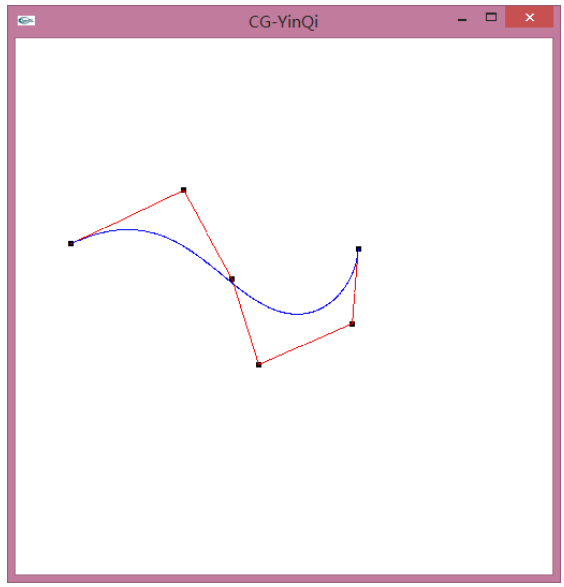
注：由于椭圆只实现了长短轴平行于坐标轴的椭圆，所以无法进行旋转。

2.5.2.3 运行结果

旋转 Beizer 曲线，旋转中心为第四个控制点。

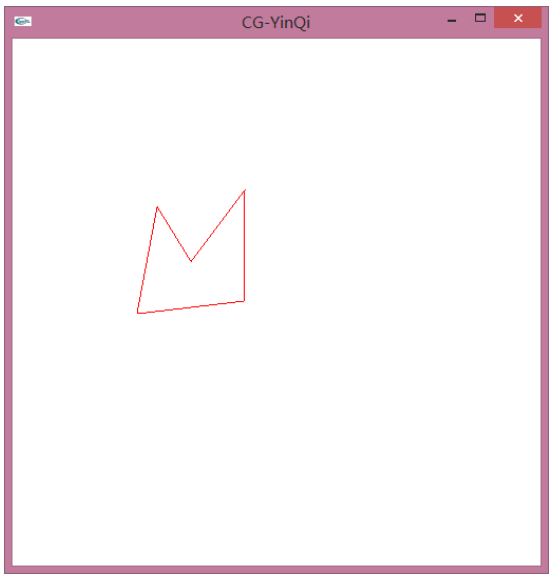


Beizer 曲线旋转前

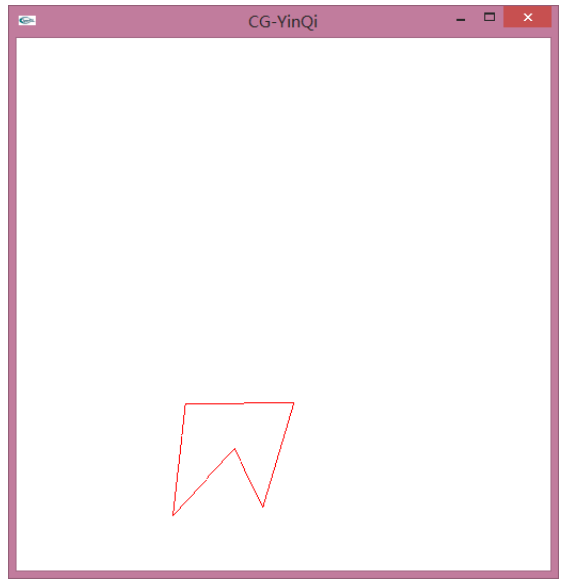


Beizer 曲线旋转后

旋转多边形，旋转中心在多边形外部。



多边形旋转前



多边形旋转后

2.5.3 缩放

2.5.3.1 代码介绍

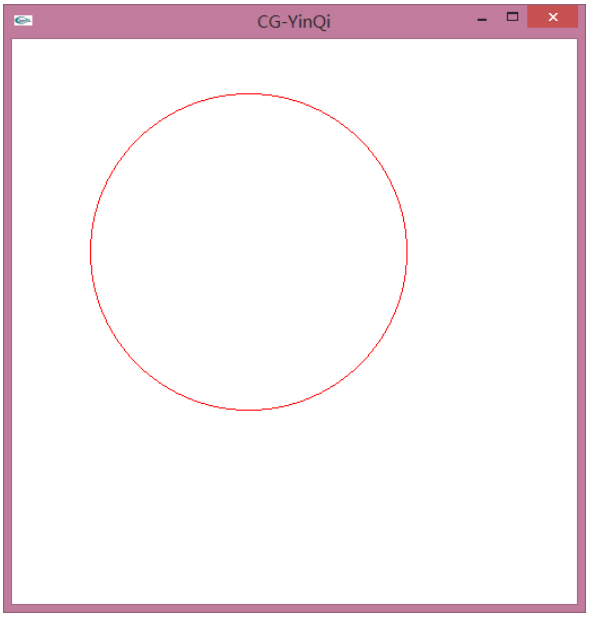
```
/*初始化一些变量*/
void initZoom();
/*判断旋转图形的形状*/
void judgeShapeZ();
/*旋转的鼠标操作*/
void zoom(GLint button, GLint action, GLint xMouse, GLint yMouse);
/*缩放圆*/
void zoomCircle(double amount);
/*缩放Beizer曲线*/
void zoomBeizer(double amount);
/*缩放B样条曲线*/
void zoomBCurve(double amount);
/*缩放多边形*/
void zoomPolygon(double amount);
/*缩放椭圆*/
void zoomEllipse(double amount);
/*缩放直线*/
void zoomLine(double amount);
```

2.5.3.2 操作方式

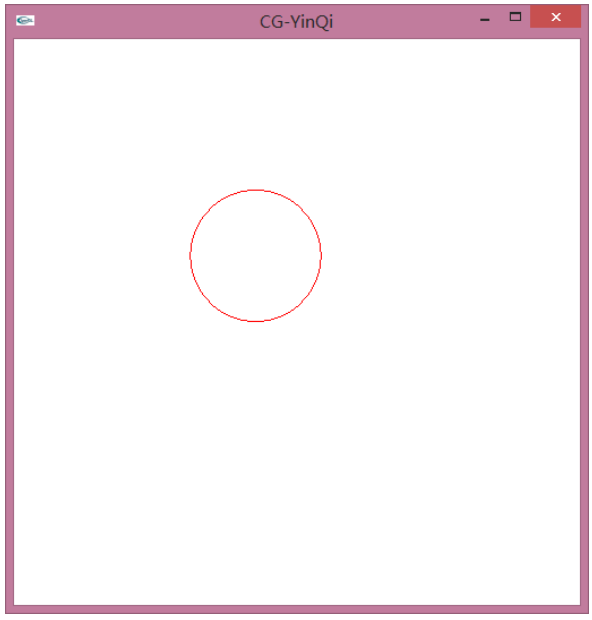
首先在屏幕上点击一下，确定缩放中心，然后向上或向下滚动鼠标滚轮，图形会放大或缩小。

2.5.3.3 运行结果

缩放圆，缩放中心接近圆心。

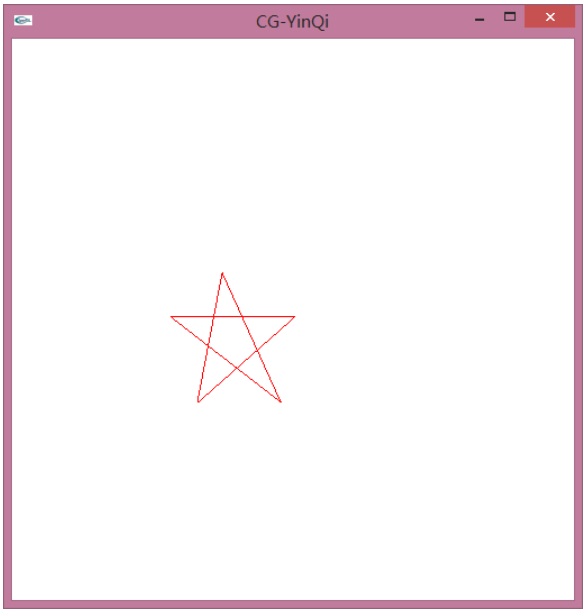


圆缩放前

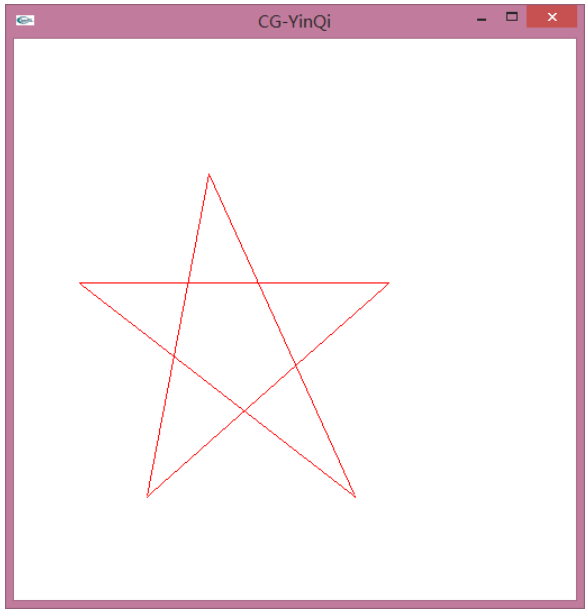


圆缩放后

缩放多边形，缩放中心接近几何中心。



多边形缩放前



多边形缩放后

2.6 图形数据填充

2.6.1 代码介绍

```
/*边的数据结构*/
typedef struct tagEDGE {
    double xi;
    double dx;
    double ymax;
}EDGE;
/*初始化一些变量*/
void initFillArea();
/*填充多边形的算法*/
void fillArea(vector<scrPt> point, char key);
/*获取多边形顶点最小值*/
int getPolygonMin(vector<scrPt> point);
/*获取多边形顶点最小值*/
int getPolygonMax(vector<scrPt> point);
/*初始化新边表*/
void initScanlineNewEdgeTable(vector<scrPt> point);
/*填充水平边*/
void horizontalEdgeFill(vector<scrPt> point);
/*比较函数*/
bool EdgeXiComparator(EDGE&e1, EDGE&e2);
/*将扫描线对应的所有新边插入到aet中*/
```

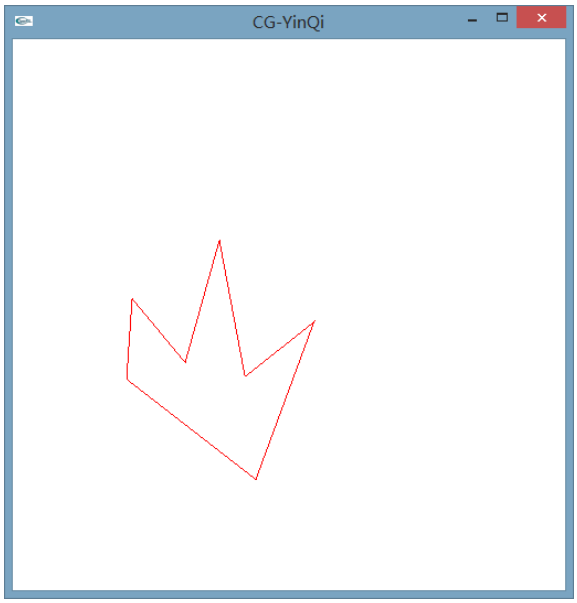
```
void insertNetlistToAet(int y);
/*执行具体的填充动作*/
void fillAetScanline(int y);
/*判断活动边*/
bool isEdgeOutOfActive(EDGE e, int y);
/*删除非活动边*/
void removeNonactiveedgeFromAet(int y);
/*更新活动边表中每项的xi值，并根据xi重新排序*/
void updateAndResortAet();
/*对每条扫描线进行处理*/
void processScanlineFill();
//extern void scanLinePolygonFill();
/*选择多边形的填充颜色*/
void chooseColor(unsigned char key, int x, int y);
```

2.6.2 操作方式

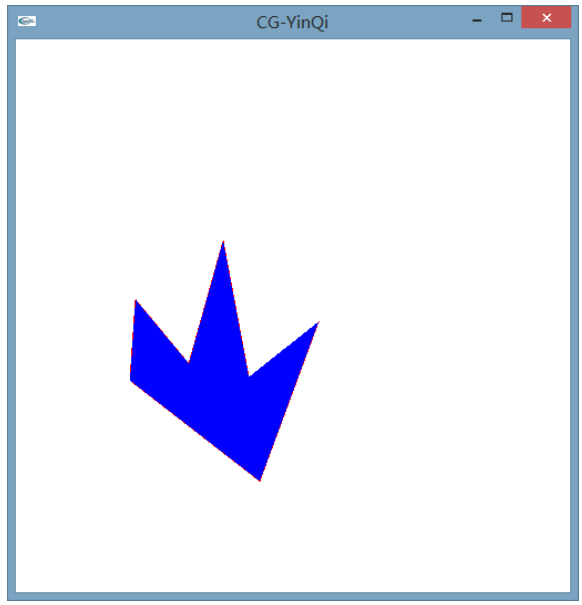
填充只能填充多边形，并且只能填充最后一次编辑的多边形，选择填充功能后，根据控制台的提示在键盘上输入字母选择相应的颜色。

2.6.3 运行结果

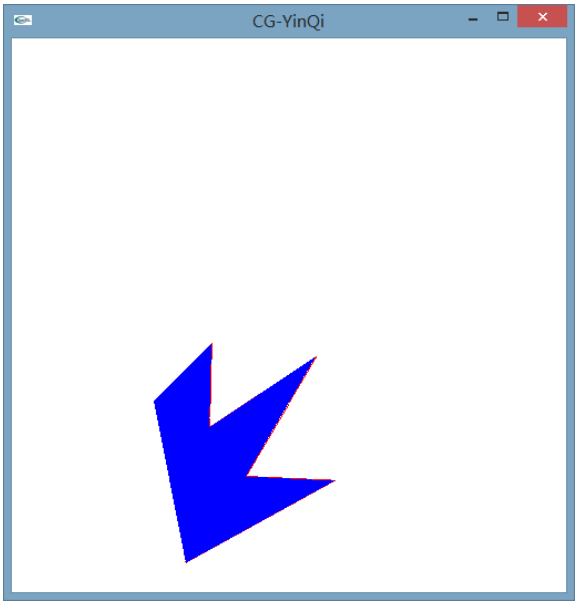
对多边形进行了填充以及填充后的一系列操作。



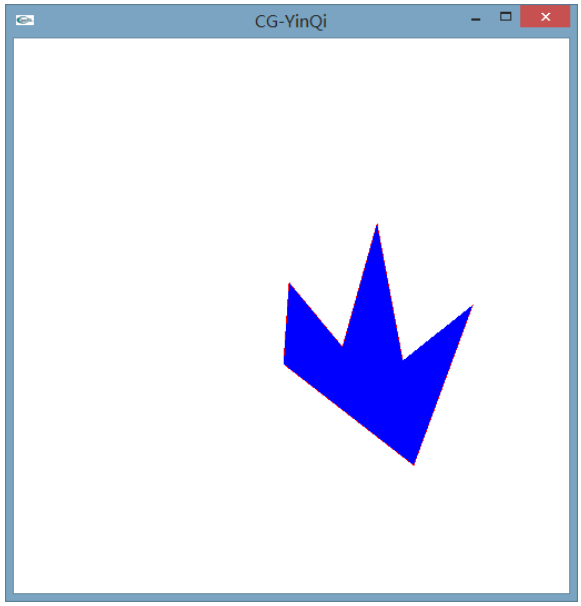
多边形填充前



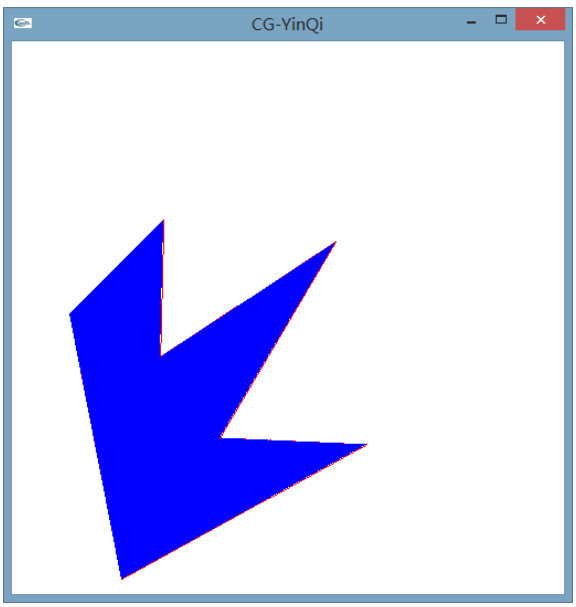
多边形填充后



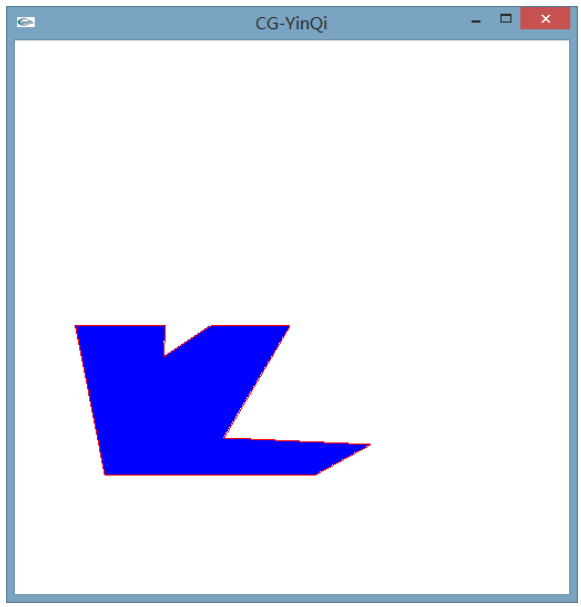
平移



旋转



缩放



裁剪

2.7 图形数据保存

2.7.1 代码介绍

```
/*从屏幕读取数据*/
bool screenshot(const char* filename);
/*写入文件，生成bmp图片*/
bool writeBMP(const char filename[], unsigned char* data, unsigned int w, unsigned int h);
```

2.7.2 操作方式

在菜单中点击保存图像，则图像会自动保存，文件名为 MyPicture.bmp，在工程目录下。

2.7.3 运行结果

2.8 清空屏幕

2.8.1 代码介绍

```
/*将所有存储图形的数组都清空，同时清空屏幕*/
void clearS();
```

2.8.2 操作方式

在菜单中选择清空屏幕即可。