实验一　椭圆的扫描转换算法的实现

姓名 _刘子言_　学号_20002462_　专业班级_计 203_　成绩_____

实验日期_2022.4.21_　实验地点_线上_　指导教师(签名)_李建华_

一．实验目的

1、学习椭圆的中点 Bresenham 算法。

2、实现交互式绘制椭圆。

二．实验工具与设备

计算机，开发软件为 CodeBlocks （配置 OpenGL）

三、实验内容

1、在本科学习平台（s.ecust.edu.cn）资料栏下，下载椭圆的中点 Bresenham 算法代码文件 onMidPointEllispe.cpp；

2、参考教材 3.4 节和 3.6 节内容，添加“椭圆”菜单项，通过橡皮筋技术，交互确定矩形对角线位置，根据矩形宽、高，确定椭圆的长、短轴长度。

1） 如果是橡皮筋时，显示矩形和内切于矩形的椭圆

2） 如果是橡皮筋结束，仅显示椭圆

3、由于椭圆的中点 Bresenham 算法要求椭圆的圆心须在坐标系原点上，可以利用 glTranslated (GLdouble x, GLdouble y, GLdouble z);将世界坐标原点平移到矩形中心，椭圆绘制结束，再反平移，恢复世界坐标原点的原来位置。

实验代码：

1、onMidPointEllispe.h

```
#include <vector>
#include <windows.h>
#include <GL/glut.h>
using namespace std;
typedef struct Point {
    int x, y;
    Point(int a = 0, int b = 0) {
        x = a, y = b;
    }
} point;
void    onMidPointEllispe(int a,int b);
```

2、main.cpp

```
#include <iostream>
#include<math.h>
using namespace std;
#include "onMidPointEllispe.h"

static GLsizei iMode = 0;
int iPointNum = 0;                          //已确定点的数目
int x_1=0,x_2=0,y_1=0,y_2=0;                //确定的两点坐标
int screenWidth = 600, screenHeight = 400;

void Initial(void)//初始化窗口
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //设置窗口背景颜色为白色
}

void ChangeSize(int w, int h)
{
    screenWidth = w;
    screenHeight = h;
    glViewport(0, 0, w, h);                 //指定窗口显示区域
    glMatrixMode(GL_PROJECTION);            //设置投影参数
    glLoadIdentity();
    gluOrtho2D(0.0,screenWidth,0.0,screenHeight);
}

void MousePlot(GLint button, GLint action, GLint xMouse, GLint yMouse)
{
    if(button == GLUT_LEFT_BUTTON && action == GLUT_DOWN)//左击绘制
    {
        if(iPointNum == 0 || iPointNum == 2)
        {
            iPointNum = 1;
            x_1 = xMouse;
            y_1 = screenHeight - yMouse;
        }
        else
        {
            iPointNum = 2;
            x_2 = xMouse;
            y_2 = screenHeight - yMouse;
            glutPostRedisplay();            //窗口执行重新绘制操作
        }
    }
```

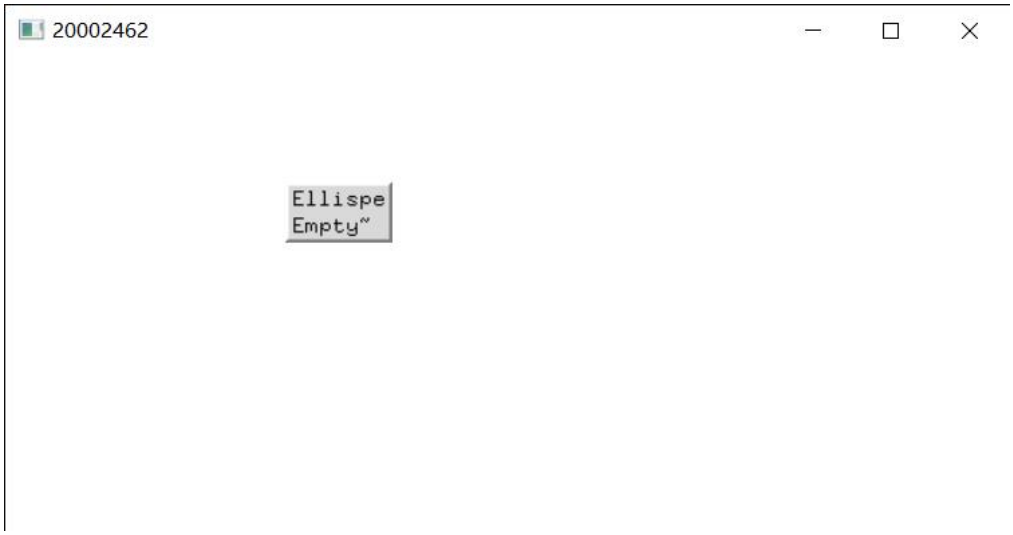
```
    if(button == GLUT_RIGHT_BUTTON && action == GLUT_DOWN)//右击清除
    {
        iPointNum = 0;
        glutPostRedisplay();                                //窗口执行重新绘制操作
    }
}

void PassiveMouseMove (GLint xMouse, GLint yMouse)//鼠标移动过程中
{
    if(iPointNum == 1)
    {
        x_2 = xMouse;
        y_2 = screenHeight - yMouse;
    }
    glutPostRedisplay();                            //窗口执行重新绘制操作
}

void ProcessMenu(int value)//处理菜单响应
{
    iMode = value;
    glutPostRedisplay();
}

void onMidPointEllispe(int a,int b)//绘制椭圆(用长短半轴)用中点 Bresenham 算法扫描转换
{
    int x,y;
    float d1,d2;
    glBegin(GL_POINTS);
    x=0;y=b;
    d1=b*b+a*a*(-b+0.25);
    glVertex2i(x,y);
    glVertex2i(-x,-y);
    glVertex2i(-x,y);
    glVertex2i(x,-y);
    while(b*b*(x+1)<a*a*(y-0.5))
    {
        if(d1<=0)
        {
            d1+=b*b*(2*x+3);
            x++;
        }
        else
        {
            d1+=b*b*(2*x+3)+a*a*(-2*y+2);
```

```
            x++; y--;
        }
        glVertex2f(x,y);
        glVertex2f(-x,-y);
        glVertex2f(-x,y);
        glVertex2f(x,-y);
    }
    d2=b*b*(x+0.5)*(x+0.5)+a*a*(y-1)*(y-1)-a*a*b*b;
    while(y>0)
    {
        if(d2<=0)
        {
            d2+=b*b*(2*x+2)+a*a*(-2*y+3);
            x++; y--;
        }
        else
        {
            d2+=a*a*(-2*y+3);
            y--;
        }
        glVertex2f(x,y);
        glVertex2f(-x,-y);
        glVertex2f(-x,y);
        glVertex2f(x,-y);
    }
    glEnd();
    glFlush();
}

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT); //用当前背景色填充窗口
    glColor3f(1.0f, 0.0f, 0.0f); //设置矩形颜色为红色

    switch(iMode)
    {
        case 1:    //椭圆菜单
            if(iPointNum == 1)                //绘制矩形
            {
                glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
                //参数 GL_FRONT_AND_BACK：表示显示模式将适用于物体的所有面；
                //参数 GL_LINE：表示显示线段，多边形用轮廓显示
                glRectf(x_1,y_1,x_2,y_2);
            }
```

```
            glColor3f(0.0f, 1.0f, 0.0f); //设置椭圆颜色为绿色
            glTranslated((x_1+x_2)/2, (y_1+y_2)/2, 0); //将世界坐标原点平移到矩形中心
            onMidPointEllispe(abs((x_2-x_1)/2),   abs((y_2-y_1)/2)); //绘制椭圆，输入长短轴
            glTranslated(-(x_1+x_2)/2,   -(y_1+y_2)/2,   0); //反平移
            break;
        case 2:    //空菜单
            break;
        default: break;
    }
    glutSwapBuffers(); //交换缓冲区
}

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB); //使用“双缓存”以及 RGB 模型
    glutInitWindowSize(600,400);
    glutInitWindowPosition(200,200);
    glutCreateWindow("20002462");

    //创建菜单并定义菜单回调函数
    glutCreateMenu(ProcessMenu);
    glutAddMenuEntry("Ellispe", 1);                  //菜单 1 椭圆
    glutAddMenuEntry("Empty~",2);                    //菜单 2 设为空
    glutAttachMenu(GLUT_RIGHT_BUTTON);       //将主菜单与鼠标右键关联

    glutDisplayFunc(Display);
    glutReshapeFunc(ChangeSize);                     //指定窗口在整形回调函数
    glutMouseFunc(MousePlot);                        //指定鼠标响应函数
    glutPassiveMotionFunc(PassiveMouseMove);      //指定鼠标移动响应函数

    Initial();
    glutMainLoop();
    return 0;
}
```

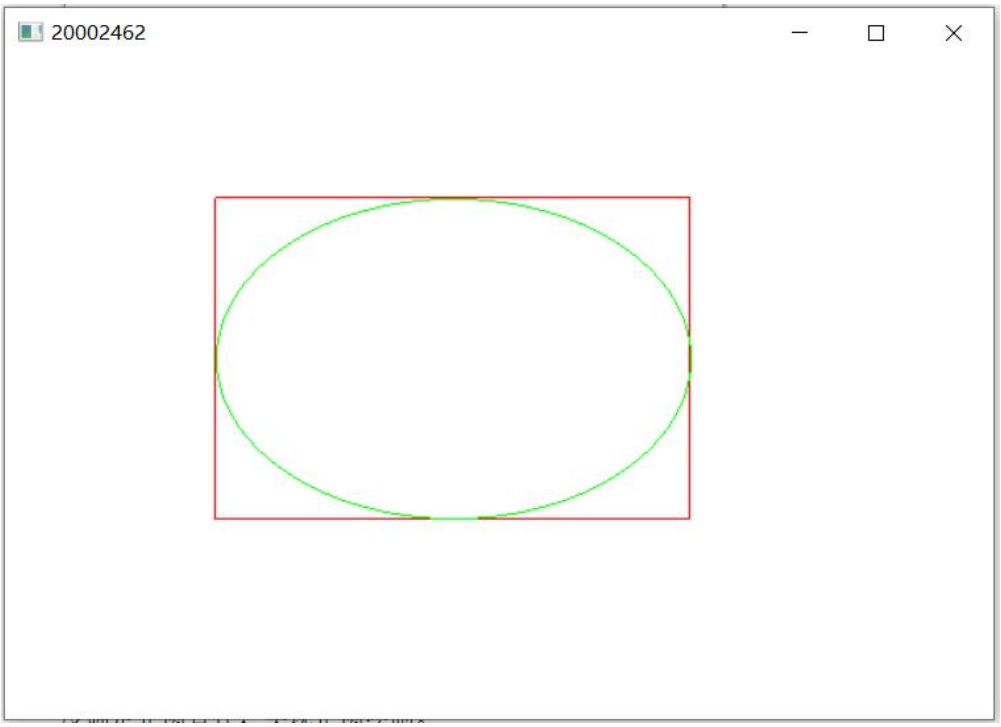
实验运行结果截图：

1、鼠标右击显示菜单栏——

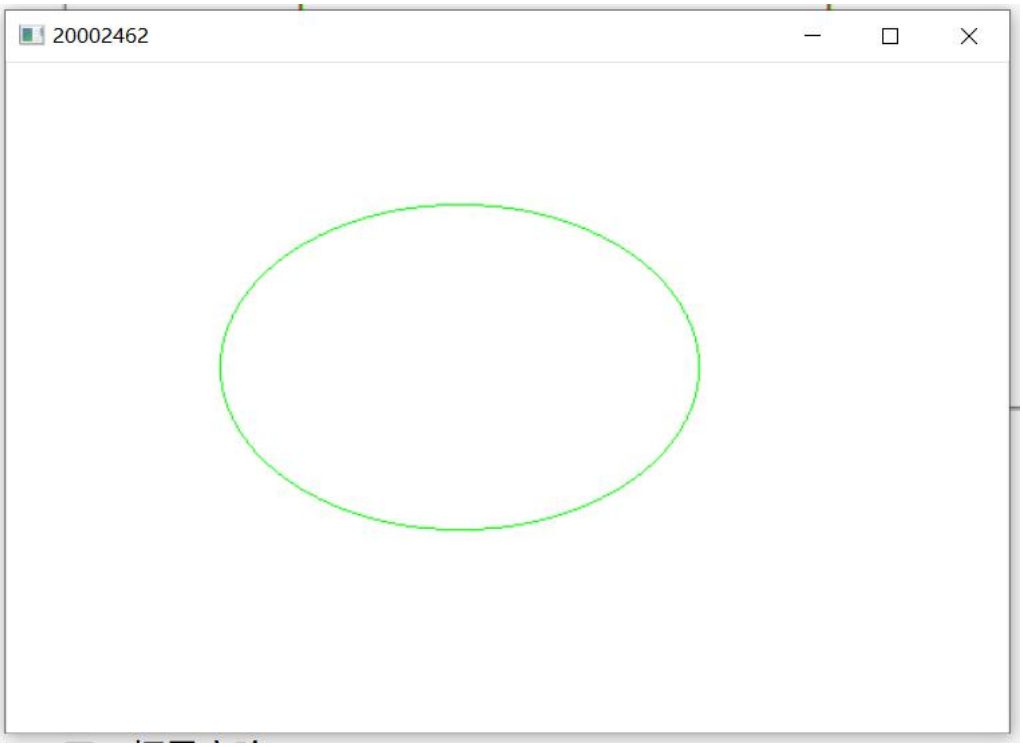


2、选择菜单 Ellpspe，左击鼠标，绘制椭圆——

（1）橡皮筋时，显示矩形和内切于矩形的椭圆



（2）再次点击鼠标，橡皮筋结束，仅显示椭圆



四、拓展实验

利用 glutGetModifiers()函数，通过组合 SHIFT 键，控制正方形边长第 2 个位置，根据正方形长度确定正圆直径，实现正圆绘制。

int glutGetModifiers(void);

这个函数仅仅只能在处理按键消息或者鼠标消息函数里被调用，通过函数的返回值检测是否有组合键被按下。

这里函数的返回值是三个 glut.h 里预定义的常量里的一个，或它们的或组合。这三个常量是：

1）GLUT_ACTIVE_SHIFT: 当按下 SHIFT 键或以按下 CAPS LOCK，注意两者同时按下时，不会返回这个值。

2）GLUT_ACTIVE_CTRL: 返回它，当按下 CTRL 键。

3）GLUT_ACTIVE_ATL：返回它,当按下 ATL 键。

例如：GLUT_ACTIVE_CTRL|GLUT_ACTIVE_ALT：按下组合键 CTRL+ALT

实验代码：

1、头文件与上一实验相同

2、main.cpp

```
#include <iostream>
#include<math.h>
using namespace std;
#include "onMidPointEllispe.h"

static GLsizei iMode = 0;
int iPointNum = 0;                              //已确定点的数目
int x_1=0,x_2=0,y_1=0,y_2=0;                    //确定的两点坐标
int screenWidth = 600, screenHeight = 400;

void Initial(void)//初始化窗口
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //设置窗口背景颜色为白色
}

void ChangeSize(int w, int h)
{
    screenWidth = w;
    screenHeight = h;
    glViewport(0, 0, w, h);                     //指定窗口显示区域
    glMatrixMode(GL_PROJECTION);                //设置投影参数
    glLoadIdentity();
    gluOrtho2D(0.0,screenWidth,0.0,screenHeight);
}

void MousePlot(GLint button, GLint action, GLint xMouse, GLint yMouse)
{
```

```
switch(iMode)
{
    case 1://椭圆菜单
        if(button == GLUT_LEFT_BUTTON && action == GLUT_DOWN)//左击绘制
        {
            if(iPointNum == 0 || iPointNum == 2)
            {
                iPointNum = 1;
                x_1 = xMouse;
                y_1 = screenHeight - yMouse;
            }
            else
            {
                iPointNum = 2;
                x_2 = xMouse;
                y_2 = screenHeight - yMouse;
                glutPostRedisplay();                    //窗口执行重新绘制操作
            }
        }
        break;
    case 2://椭圆与正方形(按 shift 键)都可实现的菜单
        if(button == GLUT_LEFT_BUTTON && action == GLUT_DOWN)//左击绘制
        {
            if(iPointNum == 0 || iPointNum == 2)
            {
                iPointNum = 1;
                x_1 = xMouse;
                y_1 = screenHeight - yMouse;
            }
            else
            {
                iPointNum = 2;//绘制过程中
                if( glutGetModifiers() == GLUT_ACTIVE_SHIFT)//判断是否有按 shift 键
                {
                    x_2 = xMouse;//以 x1 与 x2 的差作为圆的直径
                    y_2 = y_1 - (x_2-x_1);//y 的增量与 x 相同，保证绘制的是正方形和正方形
                }
                else//否则，没有按 shift 就还是绘制椭圆和矩形
                {
                    x_2 = xMouse;
                    y_2 = screenHeight - yMouse;
                }
                glutPostRedisplay();                    //窗口执行重新绘制操作
            }
```

```
            }
            break;
        default: break;
    }
    if(button == GLUT_RIGHT_BUTTON && action == GLUT_DOWN)//右击清除
    {
        iPointNum = 0;
        glutPostRedisplay();                    //窗口执行重新绘制操作
    }
}

void PassiveMouseMove (GLint xMouse, GLint yMouse)//鼠标移动过程中
{
    if(iPointNum == 1)
    {
        switch(iMode)
        {
            case 1://椭圆菜单
                x_2 = xMouse;
                y_2 = screenHeight - yMouse;
                break;
            case 2://椭圆与正方形(按 shift 键)都可实现的菜单
                if( glutGetModifiers() == GLUT_ACTIVE_SHIFT)
                {
                    x_2 = xMouse;
                    y_2 = y_1 - (x_2-x_1);
                }
                else
                {
                    x_2 = xMouse;
                    y_2 = screenHeight - yMouse;
                }
                break;
            default: break;
        }
        glutPostRedisplay();                    //窗口执行重新绘制操作
    }
}

void ProcessMenu(int value)//处理菜单响应
{
    iMode = value;
    glutPostRedisplay();
}
```

```
void onMidPointEllispe(int a,int b)//绘制椭圆(用 a,b 长短半轴)用中点 Bresenham 算法扫描转换
{
    int x,y;
    float d1,d2;
    glBegin(GL_POINTS);
    x=0;y=b;
    d1=b*b+a*a*(-b+0.25);
    glVertex2i(x,y);
    glVertex2i(-x,-y);
    glVertex2i(-x,y);
    glVertex2i(x,-y);
    while(b*b*(x+1)<a*a*(y-0.5))
    {
        if(d1<=0)
        {
            d1+=b*b*(2*x+3);
            x++;
        }
        else
        {
            d1+=b*b*(2*x+3)+a*a*(-2*y+2);
            x++; y--;
        }
        glVertex2f(x,y);
        glVertex2f(-x,-y);
        glVertex2f(-x,y);
        glVertex2f(x,-y);
    }
    d2=b*b*(x+0.5)*(x+0.5)+a*a*(y-1)*(y-1)-a*a*b*b;
    while(y>0)
    {
        if(d2<=0)
        {
            d2+=b*b*(2*x+2)+a*a*(-2*y+3);
            x++; y--;
        }
        else
        {
            d2+=a*a*(-2*y+3);
            y--;
        }
        glVertex2f(x,y);
        glVertex2f(-x,-y);
```

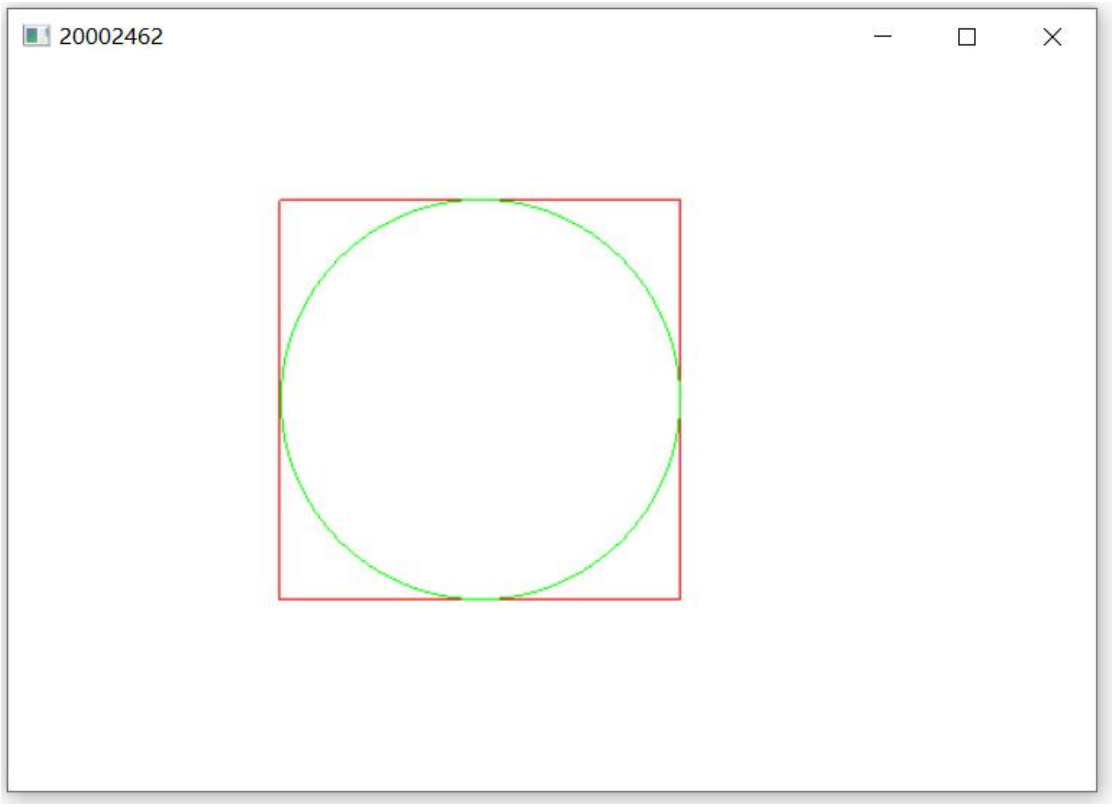
```
            glVertex2f(-x,y);
            glVertex2f(x,-y);
        }
        glEnd();
        glFlush();
}

void Display(void)
{
        glClear(GL_COLOR_BUFFER_BIT); //用当前背景色填充窗口
        glColor3f(1.0f, 0.0f, 0.0f); //设置矩形颜色为红色

        switch(iMode)
        {
            case 1://椭圆菜单
            case 2://椭圆与正方形(按 shift 键)都可实现的菜单
                if(iPointNum == 1)                  //绘制矩形
                {
                    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
                    //参数 GL_FRONT_AND_BACK：表示显示模式将适用于物体的所有面；
                    //参数 GL_LINE：表示显示线段，多边形用轮廓显示
                    glRectf(x_1,y_1,x_2,y_2);
                }
                glColor3f(0.0f, 1.0f, 0.0f); //设置椭圆颜色为绿色
                glTranslated((x_1+x_2)/2, (y_1+y_2)/2, 0); //将世界坐标原点平移到矩形中心
                onMidPointEllispe(abs((x_2-x_1)/2),   abs((y_2-y_1)/2)); //绘制椭圆，输入长短轴
                glTranslated(-(x_1+x_2)/2,   -(y_1+y_2)/2,   0); //反平移
                break;
            default: break;
        }
        glutSwapBuffers(); //交换缓冲区
}

int main(int argc, char* argv[])
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB); //使用“双缓存”以及 RGB 模型
        glutInitWindowSize(600,400);
        glutInitWindowPosition(200,200);
        glutCreateWindow("20002462");

        //创建菜单并定义菜单回调函数
        glutCreateMenu(ProcessMenu);
        glutAddMenuEntry("Ellispe", 1);
```

```
    glutAddMenuEntry("Ellispe+Circle", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON); //将主菜单与鼠标右键关联

    glutDisplayFunc(Display);
    glutReshapeFunc(ChangeSize);                    //指定窗口在整形回调函数
    glutMouseFunc(MousePlot);                       //指定鼠标响应函数
    glutPassiveMotionFunc(PassiveMouseMove);        //指定鼠标移动响应函数

    Initial();
    glutMainLoop();
    return 0;
}
```

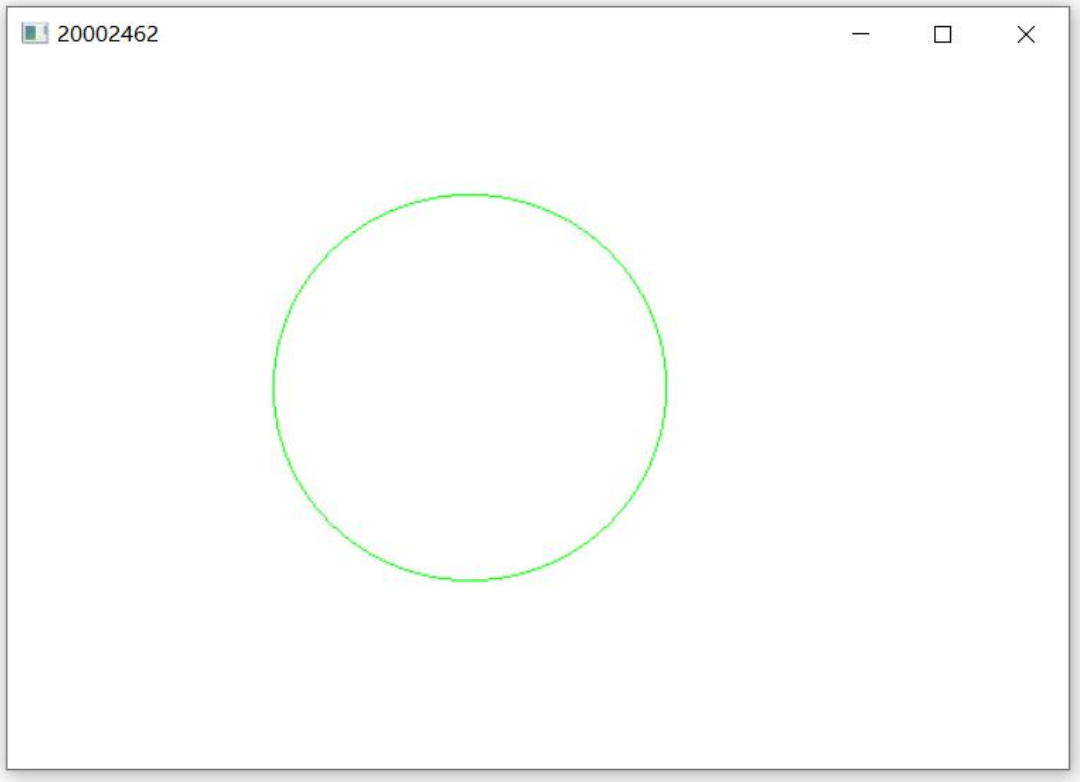
实验运行结果截图：

1、鼠标右击显示菜单栏——

2、选择菜单 Ellispe+Circle，左击鼠标，如果按 shift 键，则可以绘制正圆+正方形——



3、再次左击鼠标，橡皮筋结束，正方形消失，只显示正圆——



五．思考题

1．什么是“扫描转换”？一般什么时候需要扫描转换？

答：

（1）“扫描转换”是将光栅化与按扫描线顺序绘制图形相结合的过程。

本质上，图形的生成是在指定的输出设各上，根据坐标描述构造二维几何图形。随机扫描显示器和向量绘图仪等模拟设备能将输出指令保存在显示文件中，再由指令直接绘制出图形。而对于更具广泛意义的光栅扫描显示器等数字设备来说，图形的输出是将输出平面，如光栅扫描显示屏幕，看做像素的矩阵，在该矩阵上确定一个像素的集合来逼近该图形。这里的图形生成算法针对后一种图形的光栅化的情形，给出在光栅扫描显示器等数字设备上确定一个最佳逼近于图形的像素集的过程，称为图形的“扫描转换”。

（2）绘制图形时有时直接使用相关的绘制函数，无法满足真实感图形绘制的要求。此时，我们选择扫描转换，运用光栅扫描显示器，在像素点阵中确定最佳逼近于理想图形的像素点集，并用指定颜色显示这些像素点集，实现基本图形的光栅化，并与按扫描线顺序绘制图形的过程相结合，实现更具真实感的图形绘制。