《计算机图形学课内实验》

实验报告

班级：计算机 03

姓名：杨　　南

学号：10055076

日期：2012-10-25

一、实验目的及要求

1、理解直线的 DDA 算法、直线的 Bresenham 画线算法、中点圆算法、中点椭圆算法；

2、学会利用图形函数库编程。

二、实验环境

GUI Library：OpenGL 3.7

IDE:Visual Studio 2010

所用语言：c++

三、实验内容

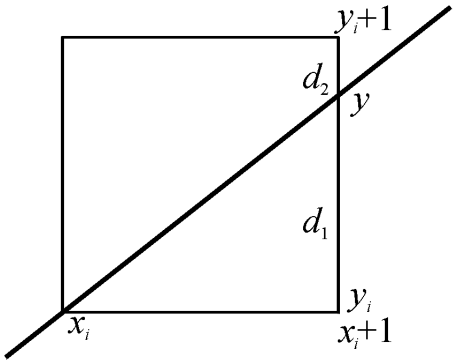
编程分别用 bresenham 算法、中点圆算法、中点椭圆算法画出一个圆、椭圆、及直线。

四、数据结构和算法描述

一）算法描述

1、bresenham 划线算法

我们以斜率大于 0 小于 1 的线段来进行考虑。以单位 x 间隔进行取样。每次绘制了当前像素点(xk,yk)之后，需要确定下一个要绘制的点是 (xk+1,yk) 还是 (xk+1,yk+1)，需要判断哪一个点像素点更接近线路径。



在取样位置，我们使用 d1 和 d2 来表示两个像素和线路径的偏移。在像素列位置 xk+1 处的直线上的 y 坐标可计算为：

$$y=m(xk+1)+b$$

那么：

$$d1=y-yk=m(xk+1)+b-yk$$

$$d2=(yk+1)-y=yk+1-m(xk+1)-b$$

要确定那个像素点更接近于线路径，需测试这两个像素的偏移的差：

$$d1-d2=2m(xk+1)-2yk+2b-1$$

为了使用整数进行计算，将决策参数定义为：

$$pk=dx*(d1-d2)=2dy*xk-2dx*yk+c$$

pk 和 d1-d2 符号相同。其中 m=dy/dx，c=2dy+dx(2b-1)。C 是常量，与像素位置无关且会在循环计算 pk 时被消除。假如 yk 处的像素比 yk+1 处的像素更接近于线路径，则 pk 为负，绘制下面的像素。反之为正，绘制上面的像素。

我们可以利用递增整数运算得到后继的决策参数值。由 k 的式子可以得出：

$$pk+1=2dy*xk+1-2dx*yk+1+c$$

则：

$$pk+1-pk=2dy(xk+1-xk)-2dx(yk+1-yk)$$

$$pk+1=pk+2dy-2dx(yk+1-yk)$$

而 yk+1-yk 取 1 或者 0 取决于 pk 的符号。

而起始像素位置(x0,y0)的参数 p0 可以计算得出：

$$p0=2dy-dx$$

由此可以得出斜率大于 0 小于 1 的 Bresenham 画线算法步骤如下：

1）输入线段两个端点坐标，将左端点存储在(x0,y0)中；

2）画出第一个点。

3）计算常量 dx,dy,2dy,2dy-2dx。得出第一个决策参数：

$$p0=2dy-dx$$

4）k=0 开始，在沿线路径的每个 xk 处，进行如下计算：

若 pk<0，下一个要绘制的点是(xk+1,yk),且

$$pk+1=pk+2dy。$$

5）重复步骤 4.

其余斜率下的算法可以按照上述方法推出，这里不再赘述。

2、中点画圆算法

1）输入圆心位置和圆的半径，得到圆周上的第一个点 Point1;

（假设起始点为坐标原点，后面将通过坐标平移来处理非圆心在圆点）

2）计算决策关键参数的初始值，P = 5/4 - r;

3）在每个 Xn 的位置，从 n = 0 开始,更具决策值 P 来判断：

如果 P<0,下一个点的位置为(Xn+1,Yn);

并且执行 P = P + 2*(Xn+1)+1;

如果 P>=0,下一个点的位置为(Xn+1,Yn-1);

并且执行 P = P + 2*(Xn+1)+1 - 2*(Yn-1);

4）通过对称原理计算其他 7 个对称相关点；

5）移动坐标到圆心点(x1,y1)

X = X + x1;

Y = Y + y1;

6）如果 X<Y 重复执行 3 到 5 的步骤，否则结束该算法

3、中点椭圆算法

椭圆中点生成算法是将椭圆在第一象限中分为两个部分：

（1)对于斜率绝对值小于 1 的区域内在 x 方向取单位量；

（2)对于斜率绝对值大于 1 的区域内在 y 方向取单位量；

斜率可以通过椭圆的标准方程中获得

K = - (ry*ry)*x/(rx*rx)*y;

这里中点椭圆生成算法同样和 Bresenham 算法有很多相似之处，同样有一个决定下一个位置的关键值 P 来做权衡处理。在中点画椭圆算法中，通过平移的方法将假设圆心在坐标原点，然后计算，最后再平移到真实原心位置。

中点椭圆算法内容：

1）输入椭圆的两个半径 r1 和 r2，并且输入椭圆的圆心。设置初始点(x0,y0)的位置为(0,r2);

2）计算区域 1 中央决策参数的初始值

p = ry*ry - rx*rx*ry + 1/4*(rx*rx);

3）在区域 1 中的每个 Xn 为止，从 n = 0 开始，直到|K|(斜率）小于-1 时后结束；

（1）如果 p < 0 ，绘制下一个点(x+1,y)，并且计算

p = p + r2*r2*(3+2*x);

（2）如果 P >=0 ,绘制下一个点(x+1,y-1),并且计算

p = p + r2*r2*(3+2*point.x) - 2*r1*r1*(y-1)

4）设置新的参数初始值；

p = ry*ry(X0+1/2)*(X0+1/2) + rx*rx*(Y0-1) - rx*rx*ry*ry;

5）在区域 2 中的每个 Yn 为止，从 n = 0 开始，直到 y = 0 时结束。

（1）如果 P>0 的情况下,下一个目标点为(x,y-1)，并且计算

p = p - 2rx*rx*(Yn+1) + rx*rx;

（2）如果 p<=0 的情况下，下一个目标点为(x+1,y-1)，并且计算

p = p - 2rx*rx*Y(n+1) + 2ry*ry*(Xn+1)+rx*rx;

6）更具对称性原理计算其他 3 个象限的坐标。

7）急速拿出中心位置在(x1,y1)的位置

x = x + x1;

y = y + y1;

二）数据结构描述

```
//定义屏幕点类
class screenPt
{
private:
    GLint x,y;
public:
    screenPt();
    void setCoords(GLint e,Glint   );//初始化对象
    GLint getX()const;//获取 X 坐标
    GLint getY()const;//获取 Y 坐标
    void incrementX();//x 自增 1
    void decrementX();//x 自减 1
    void incrementY();//y 自增 1
    void decrementY();//y 自减 1
};
//ellipse algorithm
void ellipse_plot_points(screenPt ,GLint ,GLint );//调用画点函数，画出对称点
void Ellipse(screenPt ,GLint   ,Glint   );//中点椭圆算法核心程序
void draw_ellipse(void);

void circleMidpoint(Glint ,Glint ,GLint radius);//中点画圆算法核心程序
void circle_plot_points(GLint ,GLint ,screenPt );//画出对称点
void draw_circle();
```

void BresenhamDraw(screenPt startPt,screenPt endPt);//划线预处理，判断各种情况

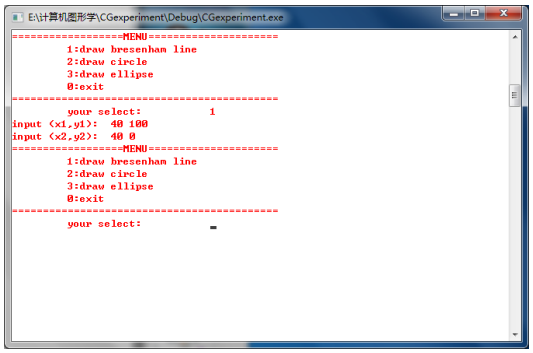
void Bresenham(screenPt ,screenPt);//B 式划线算法核心程序

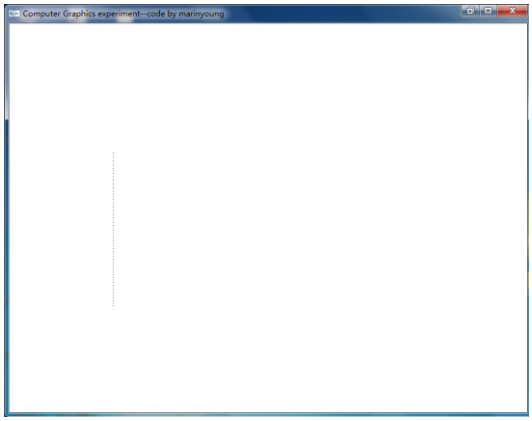
void draw_line(void);

五、调试过程及实验结果

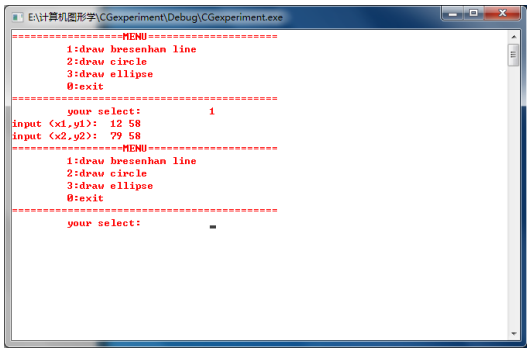
1、bresenham 划线

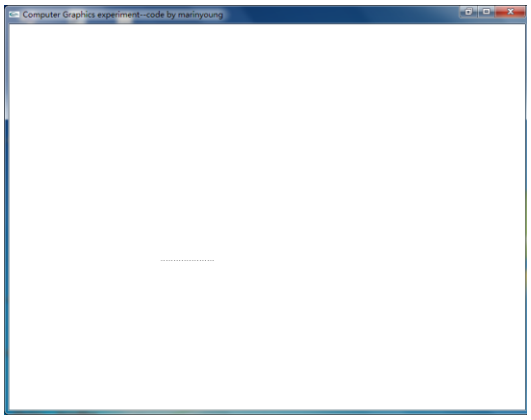
1）k 不存在；



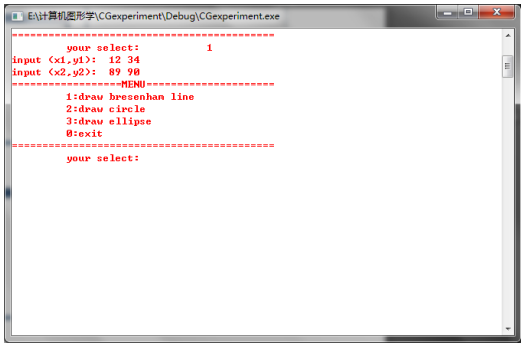


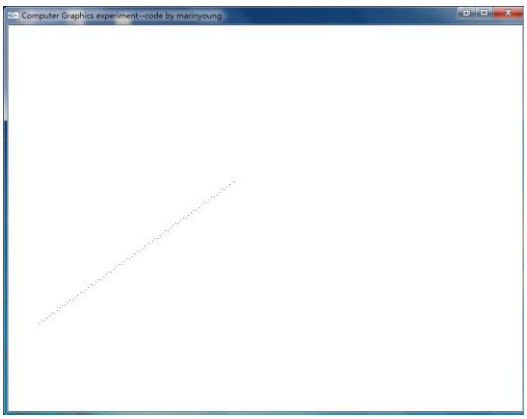
2）k=0



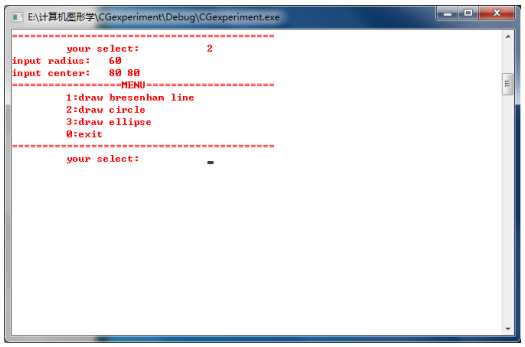


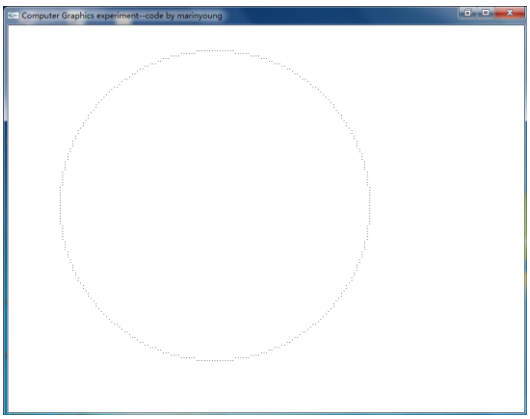
3）任意斜率


E:\计算机图形学\CGexperiment\Debug\CGexperiment.exe
your select: 1
input (x1,y1): 12 34
input (x2,y2): 89 90
MENU
1:draw bresenhan line
2:draw circle
3:draw ellipse
0:exit
your select:

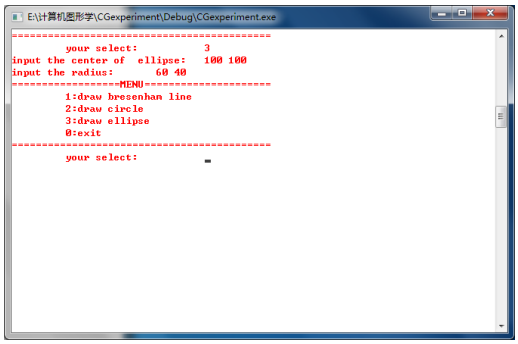

Computer Graphics experiment--code by marinyoung

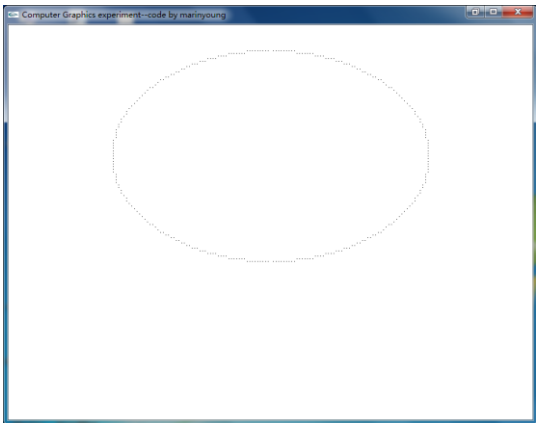
2、中点画圆


E:\计算机图形学\CGexperiment\Debug\CGexperiment.exe
your select: 2
input radius: 60
input center: 80 80
MENU
1:draw bresenhan line
2:draw circle
3:draw ellipse
0:exit
your select:


Computer Graphics experiment--code by marinyoung

3、中点椭圆


E:\计算机图形学\CGexperiment\Debug\CGexperiment.exe
your select: 3
input the center of ellipse: 100 100
input the radius: 60 40
MENU
1:draw bresenhan line
2:draw circle
3:draw ellipse
0:exit
your select:


Computer Graphics experiment--code by marinyoung

六、总结

实验首先要求对于算法相当熟悉。最初开始写代码时，由于老师还没有讲实验的具体要求，并没有调用任何图形化函数库，只是单纯地在 cmd 下输出每一个像素点的坐标。在编写 B 式划线算法程序的时候，由于书上给出的不是完整的算法，自己推导出的直线斜率|k|>1 时的算法，由此编写程序经常报错，故而参考了网上的相关资料，重新修改了算法，终于能顺利输出所有像素点。

本以为实验就这样结束了，结果在下一次课堂上，老师对实验又提出了新的要求，要求用图形化界面绘制图形，于是在原有代码基础上添加了绘图模块。

由于对 openGL 函数库不是特别熟悉，导致不能正确调用划线函数、所画图像失帧等问题。经过仔细的查阅相关资料尤其是 openGL 函数库的用法之后，逐渐掌握了画图函数的调用方法，此后的实验还是相当顺利的。

整个程序均采用 Glint 数据类型，故而绘制出来的图形是离散化的，这个采用 GLfloat 数据类型就可以改进。另外，由于自己屏幕分辨率的问题，绘制出来的圆看起来像是椭圆，原以为是程序的问题，后来绘制了一个（0，0）、（100,100）的对角线测试，才发现不是程序的问题。

回过头看，程序的人机交互还是较为简陋的，并且没有进行输入异常化处理。另外，提供的功能较少，如没有添加文件操作，不能保存绘制的图形等。这些问题将在今后的试验中加以改正。但是，作为一个验证性实验，该程序已经充分验证了基本的绘图算法，加深了自己对算法的理解，锻炼了自己的编程能力。对今后的学习大有裨益。

七、附录（源程序清单）

```
//CDecperiment.h
//code by marinyoung
//IDE:Visual Studio 2010
//GUI library:OpenGL 3.7
//last modification:October 19th,2012
#ifndef CGEXPERIMENT_H
#define CGEXPERIMENT_H
#include<iostream>
#include<GL/glut.h>
#include<cstdlib>
#include<windows.h>

#define Squre(x) (x)*(x)

const int width=700,high=700;

using namespace std;

class screenPt
{
private:
    GLint x,y;

public:
    screenPt()
    {
        x=y=0;
    }
    void setCoords(GLint xCoordValue,GLint yCoordValue)
    {
        x=xCoordValue;
        y=yCoordValue;
    }
    GLint getX()const
    {
        return x;
    }
    GLint getY()const
    {
        return y;
    }
```

```
    void incrementX()
    {
        x++;
    }
    void decrementX()
    {
        x--;
    }
    void incrementY()
    {
        y++;
    }
    void decrementY()
    {
        y--;
    }
};
//draw a point
void setPixel(GLint x,GLint y)
{
    glColor3f (0, 0, 0);
    glBegin(GL_POINTS);
        glVertex2i(x,y);
        Sleep(10);
    glEnd();
    glFlush();
}
//function statement
void ellipse_plot_points(screenPt ,GLint ,GLint );
void circle_plot_points(GLint ,GLint ,screenPt );
void Bresenham(screenPt ,screenPt );

/*=====ellipse algorithm=====*/
void Ellipse(screenPt center,GLint Rx,GLint Ry)
{
    //the first area
    screenPt ellipsePt;
    ellipsePt.setCoords(0,Ry);
    int p=Squre(Ry)-Squre(Rx)*Ry+Squre(Rx)/4;
    while(Squre(Ry)*ellipsePt.getX()<Squre(Rx)*ellipsePt.getY())
    {
        ellipsePt.incrementX();
        if(p<0)
        {
```

```
            p=p+2*Squre(Ry)*ellipsePt.getX()+Squre(Ry);
        }
        else
        {
            ellipsePt.decrementY();

    p=p+2*Squre(Ry)*ellipsePt.getX()-2*Squre(Rx)*ellipsePt.getY()+Squre(
Ry);
        }
        ellipse_plot_points(center,ellipsePt.getX(),ellipsePt.getY());
    }
    //the second area
    p=Squre(Ry)*Squre(ellipsePt.getX()+1/2)+Squre(Rx)*Squre(ellipsePt.ge
tY()-1)-Squre(Rx)*Squre(Ry);
    while(ellipsePt.getY()>=0)
    {
        ellipsePt.decrementY();
        if(p>0)
        {
            p=p-2*Squre(Rx)*ellipsePt.getY()+Squre(Rx);
        }
        else
        {
            ellipsePt.incrementX();

    p=p+2*Squre(Ry)*ellipsePt.getX()-2*Squre(Rx)*ellipsePt.getY()+Squre(
Rx);
        }
        ellipse_plot_points(center,ellipsePt.getX(),ellipsePt.getY());
    }
}
void ellipse_plot_points(screenPt center,GLint Rx,GLint Ry)
{
    setPixel(center.getX()+Rx,center.getY()+Ry);
    setPixel(center.getX()-Rx,center.getY()+Ry);
    setPixel(center.getX()+Rx,center.getY()-Ry);
    setPixel(center.getX()-Rx,center.getY()-Ry);
}

void draw_ellipse(void)
{
    GLint center_x,center_y;
    screenPt centerPt;
    GLint Rx,Ry;
```

```
    cout<<"input the center of  ellipse:\t";
    cin>>center_x>>center_y;
    centerPt.setCoords(center_x,center_y);
    cout<<"input the radius:\t";
    cin>>Rx>>Ry;
    glClearColor(1.0,1.0,1.0,0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    Ellipse(centerPt,Rx,Ry);
}
/*=====circle algorithm=====*/
void circleMidpoint(GLint Xc,GLint Yc,GLint radius)
{
    screenPt circlePt;
    GLint p=1-radius;
    circlePt.setCoords(0,radius);

    circle_plot_points(Xc,Yc,circlePt);
    while(circlePt.getX()<circlePt.getY())
    {
        circlePt.incrementX();
        if(p<0)
            p+=2*circlePt.getX()+1;
        else
        {
            circlePt.decrementY();
            p+=2*(circlePt.getX()-circlePt.getY())+1;
        }
        circle_plot_points(Xc,Yc,circlePt);
    }
}
void circle_plot_points(GLint Xc,GLint Yc,screenPt circlePt)
{
    setPixel(Xc+circlePt.getX(),Yc+circlePt.getY());
    setPixel(Xc-circlePt.getX(),Yc+circlePt.getY());
    setPixel(Xc+circlePt.getX(),Yc-circlePt.getY());
    setPixel(Xc-circlePt.getX(),Yc-circlePt.getY());

    setPixel(Xc+circlePt.getY(),Yc+circlePt.getX());
    setPixel(Xc-circlePt.getY(),Yc+circlePt.getX());
    setPixel(Xc+circlePt.getY(),Yc-circlePt.getX());
    setPixel(Xc-circlePt.getY(),Yc-circlePt.getX());
}
void draw_circle()
{
```

```
    GLint x,y,radius;
    cout<<"input radius:\t";
    cin>>radius;
    cout<<"input center:\t";
    cin>>x>>y;
    glClearColor(1.0,1.0,1.0,0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    circleMidpoint(x,y,radius);
}
/*=====bresenham algorithm=====*/
void swap(GLint *x,GLint *y)
{
    GLint temp=*x;
    *x=*y;
    *y=temp;
}

//initialize and Draw Line
void BresenhamDraw(screenPt startPt,screenPt endPt)
{
    screenPt linePt;

linePt.setCoords(startPt.getY()<endPt.getY()?startPt.getY():endPt.getY(
),

startPt.getX()<endPt.getX()?startPt.getX():endPt.getX());
    GLint y_max=startPt.getY()>endPt.getY()?startPt.getY():endPt.getY(),

x_max=startPt.getX()>endPt.getX()?startPt.getX():endPt.getX();
    GLint x1,y1,x2,y2;
    //if k doesn't exist
    if(startPt.getX()==endPt.getX())
    {
        //the same point
        if(startPt.getY()==endPt.getY())
        {
            std::cout<<"error!"<<endl;
            return;
        }
        //k doesn't exist
        else
        {
            while(linePt.getY()!=y_max)
            {
```

```
                linePt.incrementY();
                setPixel(startPt.getX(),linePt.getY());
            }
        }
    }
    //if k=0
    else if(startPt.getY()==endPt.getY())
    {
        while(linePt.getX()!=x_max)
        {
            linePt.incrementX();
            setPixel(linePt.getX(),startPt.getY());
        }
    }
    //swap with each other if startPt.getX()>endPt.getX()
    else if(startPt.getX()>endPt.getX())
    {
        x1=startPt.getX();
        y1=startPt.getY();
        x2=endPt.getX();
        y2=endPt.getY();
        swap(x1,x2);
        swap(y1,y2);
        startPt.setCoords(x1,y1);
        endPt.setCoords(x2,y2);
        Bresenham(startPt,endPt);
    }
    else
        Bresenham(startPt,endPt);
}
//Bresenham Algorithm
void Bresenham(screenPt startPt,screenPt endPt)
{
    GLint dx=endPt.getX()-startPt.getX(),dy=endPt.getY()-startPt.getY();
    GLint p=2*dy-dx;
    bool
m=(abs(endPt.getY()-startPt.getY())<=abs(endPt.getX()-startPt.getX()));
    // |k|<1
    if(m)
    {
        //0<k<1
        if(startPt.getY()<=endPt.getY())
        {
            for(int k=0; k<dx; k++)
```

```
        {
            setPixel(startPt.getX(),startPt.getY());
            startPt.incrementX();
            if(p<0)
                p+=2*dy;
            else
            {
                startPt.incrementY();
                p+=2*(dy-dx);
            }
        }
    }
    //-1<k<0
    else
    {
        p=dx-2*dy;
        for(int k=0; k<dx; k++)
        {
            setPixel(startPt.getX(),startPt.getY());
            startPt.incrementX();
            if(p<0)
                p-=2*dy;
            else
            {
                startPt.decrementY();
                p-=2*(dx+dy);
            }
        }
    }
}
//|k|>1
else
{
    //k>1
    if(startPt.getY()<=endPt.getY())
    {
        p=2*dx-dy;
        for(int k=0; k<dy; k++)
        {
            setPixel(startPt.getX(),startPt.getY());
            startPt.incrementY();
            if(p<0)
                p-=2*dy;
            else
```

```
                {
                    startPt.incrementX();
                    p-=2*(dx+dy);
                }
            }
        }
        //k<-1
        else
        {
            setPixel(startPt.getX(),startPt.getY());
            p=2*dx+dy;
            for(int k=0; k<dy; k++)
            {
                startPt.decrementY();
                if(p<0)
                    p+=2*dx;
                else
                {
                    startPt.incrementX();
                    p+=2*(dx+dy);
                }
            }
        }
    }
}
void draw_line(void)
{
    screenPt startPt,endPt;
    GLint x1,y1,x2,y2;
    cout<<"input (x1,y1):\t";
    cin>>x1>>y1;
    startPt.setCoords(x1,y1);
    cout<<"input (x2,y2):\t";
    cin>>x2>>y2;
    endPt.setCoords(x2,y2);
    glClearColor(1.0,1.0,1.0,0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    BresenhamDraw(startPt,endPt);
}
void draw_coordinate()
{
    for(int k=0;k<high;k++)//draw y
    {
        glColor3f (0, 0, 0);
```

```
        glBegin(GL_POINTS);
            glVertex2i(width/2,k);
        glEnd();
        glFlush();
    }
    for(int k=0;k<width;k++)//draw x
    {
        glColor3f (0, 0, 0);
        glBegin(GL_POINTS);
            glVertex2i(k,high/2);
        glEnd();
        glFlush();
    }
}
void menu()
{
    int select=1;
    draw_coordinate();
    while(select)
    {
        cout<<"==================MENU====================="<<endl;
        cout<<"\t 1:draw bresenham line"<<endl;
        cout<<"\t 2:draw circle"<<endl;
        cout<<"\t 3:draw ellipse"<<endl;
        //cout<<"\t 4:clear the screen"<<endl;
        cout<<"\t 0:exit"<<endl;
        cout<<"==========================================="<<endl;
        cout<<"\t your select:\t\t";
        cin>>select;
        switch(select)
        {
        case 0:
            break;
        case 1:
            draw_line();
            break;
        case 2:
            draw_circle();
            break;
        case 3:
            draw_ellipse();
            break;
         case 4:
            //glClear (GL_COLOR_BUFFER_BIT);
```

```
            // break;
        default:
            break;
        }
    }
    return;
}
#endif
```

```
//main.cpp
//code by marinyoung
//last modification:October 19th,2012
#include <windows.h>
#include <GL/glut.h>
#include "CGexperiment.h"

void init(void)
{
    glClearColor(1.0,1.0,1.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0,200.0,0.0,150.0);
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutInitWindowPosition(-1, -1);
    glutInitWindowSize(800, 600);
    glutCreateWindow ("Computer Graphics experiment--code by marinyoung");

    init();
    glutDisplayFunc(menu);
    glutMainLoop();
    return 0;
}
```