计算机学院实验报告

实验题目： 直线与画圆算法		学号：202000120101
日期：2022.9.13	班级：20.2	姓名：尹国泰
Email：1018693208@qq.com		

实验目的：

理解光栅化，掌握几何数据如何经过一系列变换后转化为像素从而呈现在显示设备上。

实验环境介绍：

操作系统：Window10

编译器环境：MinGW，VSCode

OpenGL 环境：freeglut

解决问题的主要思路：

1. 配置好 OpenGL 环境
2. 实现 DDA，Bresenham 画线算法以及中点画圆算法

其中常规的中点画圆算法将 d 的值设置为 d=1.25-r，可以将 d 扩大 4 倍，即将 d 设置为 d=5-4*r 来规避浮点运算

3. 实现简单的交互操作来确定使用的算法以及线段和圆的相关参数
4. 运行并进行测试

实验步骤：

1. 进行 OpenGL 环境的配置
2. DDA 画线算法的实现

```
void DDALine(int x0,int y0,int x1,int y1){
    glBegin(GL_POINTS);
    glColor3f(255,255,255);
    float k=1.0*(y1-y0)/(x1-x0);
    if(abs(k)<=1){//abs(k)<=1
        if(x0>x1){
            swap(x0,x1);
            swap(y0,y1);
        }
        float y=y0;
        for(int x=x0;x<=x1;x++){
            glVertex2i(x,int(y+0.5));
            y=y+k;
        }
```

```
    }else{//abs(k)>1
        if(y0>y1){
            swap(x0,x1);
            swap(y0,y1);
        }
        float x=x0;
        k=1/k;
        for(int y=y0;y<=y1;y++){
            glVertex2i(int(x+0.5),y);
            x=x+k;
        }
    }
    glEnd();
}
```

3. Bresenham 画线算法的实现

```
void IntegerBresenhamline(int x0,int y0,int x1,int y1){
    glBegin(GL_POINTS);
    glColor3f(0,255,0);
    float k=1.0*(y1-y0)/(x1-x0);
    if(abs(k)<=1){
        if(x0>x1){
            swap(x0,x1);
            swap(y0,y1);
        }
        int x,y,dx,dy,e;
        dx=x1-x0,dy=y1-y0;
        x=x0,y=y0;
        if(k>=0){//abs(k)<=1 且 k>=0
            e=-dx;
            for(int i=0;i<=dx;i++){
                glVertex2i(x,y);
                x++,e=e+2*dy;
                if(e>=0){
                    y++,e=e-2*dx;
                }
            }
        }else{//abs(k)<=1 且 k<0
            e=dx;
            for(int i=0;i<=dx;i++){
                glVertex2i(x,y);
                x++,e=e+2*dy;
                if(e<=0){
                    y--,e=e+2*dx;
```

```
                }
            }
        }
    }else{
        if(y0>y1){
            swap(x0,x1);
            swap(y0,y1);
        }
        int x,y,dx,dy,e;
        dx=x1-x0,dy=y1-y0;
        x=x0,y=y0;
        if(k>=0){//abs(k)>1 且 k>=0
            e=-dy;
            for(int i=0;i<=dy;i++){
                glVertex2i(x,y);
                y++,e=e+2*dx;
                if(e>=0){
                    x++,e=e-2*dy;
                }
            }
        }else{//abs(k)>1 且 k<0
            e=dy;
            for(int i=0;i<=dy;i++){
                glVertex2i(x,y);
                y++,e=e+2*dx;
                if(e<=0){
                    x--,e=e+2*dy;
                }
            }
        }
    }
    glEnd();
}
```

4. 整型中点画圆算法的实现

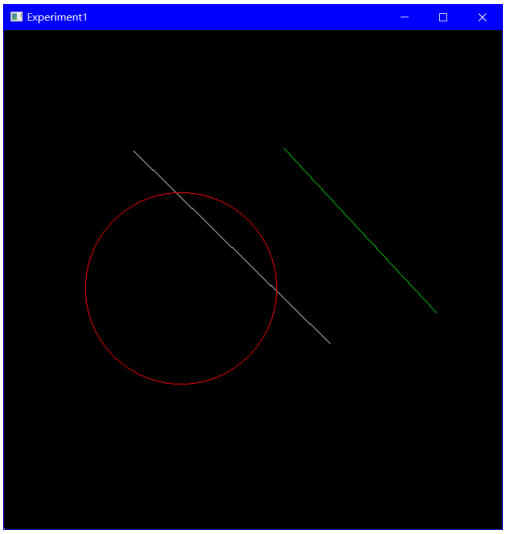
```
void CirclePoints(int cx,int cy,int x,int y){
    glBegin(GL_POINTS);
    glColor3f(255,0,0);
    glVertex2i(cx+x,cy+y); glVertex2i(cx+y,cy+x);
    glVertex2i(cx-x,cy+y); glVertex2i(cx+y,cy-x);
    glVertex2i(cx+x,cy-y); glVertex2i(cx-y,cy+x);
    glVertex2i(cx-x,cy-y); glVertex2i(cx-y,cy-x);
    glEnd();
}
void MidPointCircle(int cx,int cy,int r){
```

```
    int x,y;
    int d;//float d;
    x=0,y=r,d=5-4*r;//d=1.25-r;将 d 扩大 4 倍来规避浮点运算
    CirclePoints(cx,cy,x,y);
    while(x<=y){
        if(d<0){
            d+=8*x+12;
        }else{
            d+=8*(x-y)+20;
            y--;
        }
        x++;
        CirclePoints(cx,cy,x,y);
    }
}
```

5. 简单的交互设计

```
void myMouse(int button, int state, int x, int y){
    if (button == GLUT_LEFT_BUTTON&&state == GLUT_DOWN) {
        mx0=x,my0=h-y;//按下左键确定线段及圆的直径起点
    }
    if (button == GLUT_LEFT_BUTTON&&state == GLUT_UP) {
        mx1=x,my1=h-y;//松开左键确定线段及圆的直径终点
    }
    if (button == GLUT_RIGHT_BUTTON&&state == GLUT_DOWN){
        clearoption^=1;//单击右键来切换是否只保留最后一次画图操作
    }
}
void myDisplay(void) {
    if(clearoption) glClear(GL_COLOR_BUFFER_BIT);
    if(option=='1') DDALine(mx0,my0,mx1,my1);//按下键盘数字 1 运行 DDA
画线算法
    if(option=='2') IntegerBresenhamline(mx0,my0,mx1,my1);//按下键盘
数字 2 运行 Bresenham 画线算法
    if(option=='3'){//按下键盘数字 3 运行中点画圆算法
        int r= sqrt((mx0-mx1)*(mx0-mx1)+(my0-my1)*(my0-my1))/2;
        MidPointCircle((mx0+mx1)/2,(my0+my1)/2,r);
    }
    glFlush();//强制刷新缓冲，保证绘图命令将被执行
}
void myKeyboard(unsigned char key, int x, int y){
    option=key;
}
```

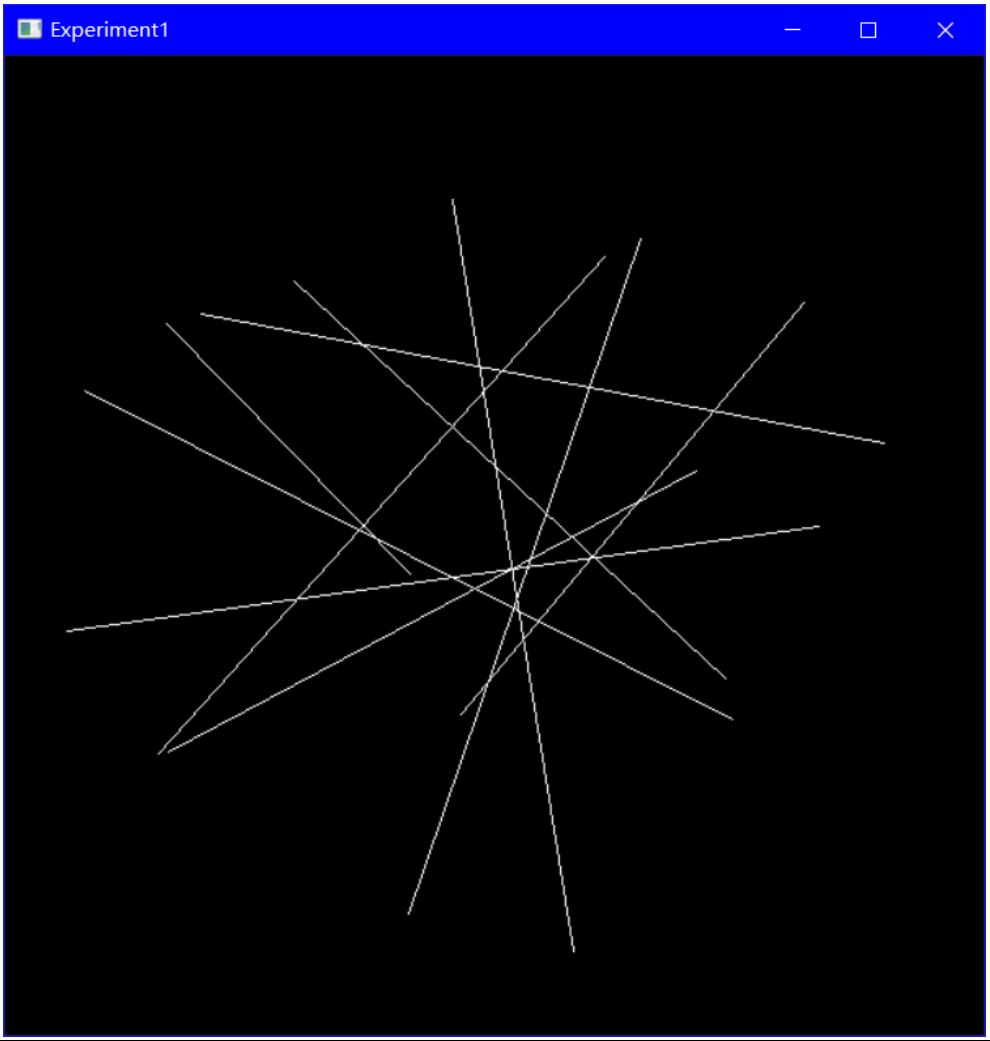
6. 编译运行并进行测试



实验结果展示及分析：

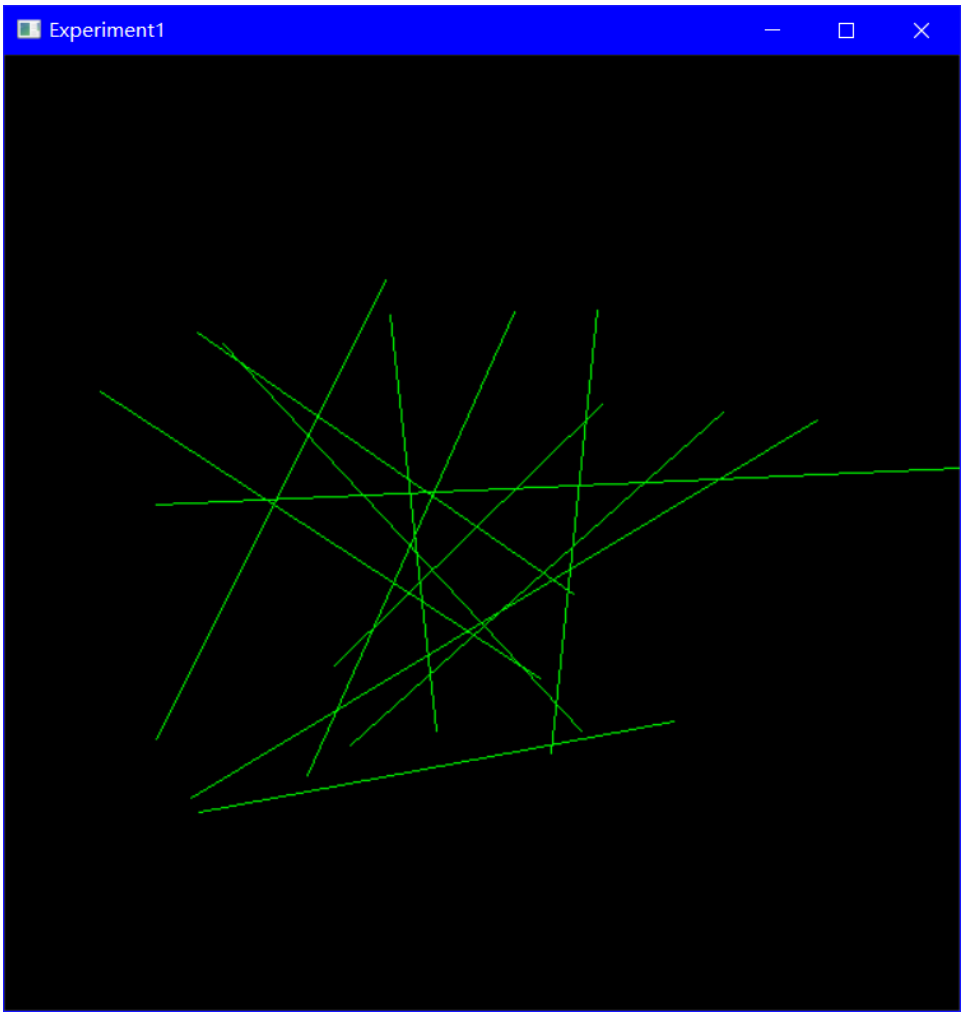
1. DDA 画线算法的测试

运行程序后，按下键盘数字 1 运行 DDA 画线算法，按下左键确定线段起点，松开左键确定线段终点，这样分别画出 k>0, k<0, |k|>1, |k|<1 的多条线段，均能够正常画出且没有出现断断续续的点，这说明我们对于|k|>1, |k|<1 两种情况分别讨论并实现的算法是成功的



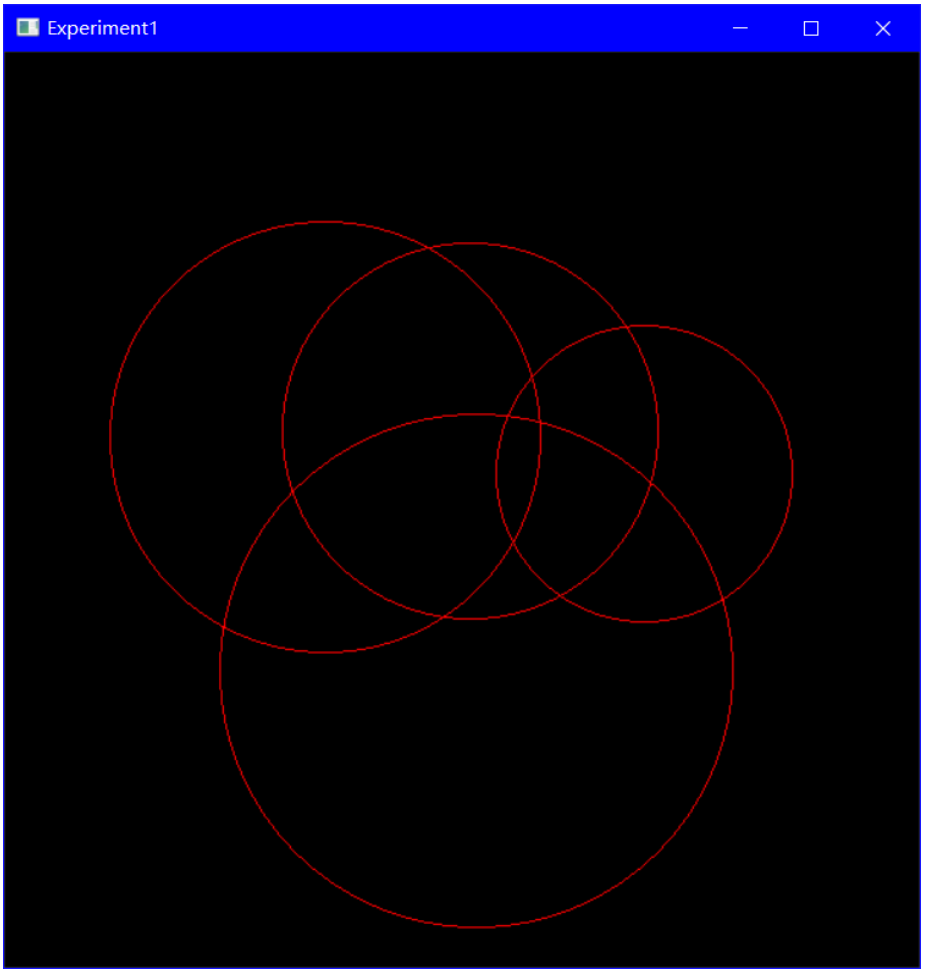
2. 整型 Bresenham 画线算法的测试

运行程序后，按下键盘数字 2 运行 Bresenham 画线算法，分别画出 k>0, k<0, |k|>1, |k|<1 的多条线段，同样能够正常画出且没有出现断断续续地点，说明我对 k>1, k<-1, 0<k<1, -1<k<0 这四种情况分别实现的算法是正确的



3. 整型中点画圆算法的测试

运行程序后，按下键盘数字 3 运行中点画圆算法，按下左键确定圆的直径起点，松开左键确定圆的直径终点，像这样画出多个圆，显示正常，说明我改进后的整型中点画圆算法是正确的



实验中存在的问题及解决：

1. 画线时出现断断续续的线，且斜率绝对值越大，线上的点越稀疏。

分析解决：这是因为只实现了|k|<1 的画线算法，当|k|>1 时，取样点变少，线上的点十分稀疏，只需要再设计出|k|>1 的算法，将 y 的值递增，来确定 x 的值即可。

2. 中点画圆算法中出现 d=1.25-r，包含浮点运算，导致效率降低。

分析解决：可以将 d 的值扩大 4 倍，即 d'=4*d=5-4*r，这样 d 也是一个整型数，规避了浮点运算。