

计算机图形学实验

实验二

学 号： 20191130038

姓 名： 郑广敏

专 业： 智能科学与技术

年 级： 2019级

任课教师： 吴昊

2020 年 10 月 16 日

**一、目的和要求：**

1. 进一步掌握OpenGL的绘制原理；
2. 直线生成算法的理解与实现；
3. 圆生成算法的理解与实现；
4. 椭圆生成算法的理解与实现；

**二、实验内容：**

## 1.实现DDA直线生成算法

读懂如下代码，理解执行的原理：

#include <stdlib.h>

#include <math.h>

inline int round (const float a) { return int (a + 0.5); }

void lineDDA (int x0, int y0, int xEnd, int yEnd){

int dx = xEnd - x0, dy = yEnd - y0, steps, k;

float xIncrement, yIncrement, x = x0, y = y0;

if (fabs (dx) > fabs (dy))

steps = fabs (dx);

else

steps = fabs (dy);

xIncrement = float (dx) / float (steps);

yIncrement = float (dy) / float (steps);

setPixel (round (x), round (y));

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

setPixel (round (x), round (y));

}

}

注意：

（1）代码没有主函数及其它OpenGL的初始化函数，请用前面实验一中的代码进行添加，该画线函数在回调函数myDraw()中调用执行；

例如：

void myDraw(void)

{

int x0=0, y0=0, x1=300, y1=200;

lineDDA(x0, y0, x1, y1);

}

//同时需要在main函数中注册回调函数,参考实验一

glutDisplayFunc(&myDraw);

（2）代码中两次调用setPixel(int , int)函数，该函数不是C的系统函数，需要我们自己去定义。实际上它是一个画点的函数，所以用OpenGL的画点的方法进行改写，参考如下：

void setPixel(GLint x, GLint y){

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

（3）函数前的修饰符inline表示该函数是一个内联函数。内联函数在编译时直接用函数体代码替代该函数被调用的位置，加快程序执行的速度，与宏定义非常类似。

## 2.实现Bresenham 直线生成算法：

#include <stdlib.h>

#include <math.h>

/\* Bresenham line-drawing procedure for |m| < 1.0. \*/

void lineBres (int x0, int y0, int xEnd, int yEnd){

//划线代码主体，由你们完成

}

注意：

（1）代码没有主函数及其它OpenGL的初始化函数，请用前面实验一中的代码进行添加，该画线函数在回调函数myDraw()中调用执行；

（2）代码中两次调用setPixel(int , int)函数，该函数不是C或C++的系统函数，需要我们自己去定义。

## 3.实现中点圆算法

#include <GL/glut.h>

struct scrPt {

GLint x, y;

};

void setPixel (GLint x, GLint y)

{

glBegin (GL\_POINTS);

glVertex2i (x, y);

glEnd ( );

}

void circlePlotPoints (scrPt circCtr, scrPt circPt)

{

//利用圆的对称性画8个点

setPixel (circCtr.x + circPt.x, circCtr.y + circPt.y);

setPixel (circCtr.x - circPt.x, circCtr.y + circPt.y);

setPixel (circCtr.x + circPt.x, circCtr.y - circPt.y);

setPixel (circCtr.x - circPt.x, circCtr.y - circPt.y);

setPixel (circCtr.x + circPt.y, circCtr.y + circPt.x);

setPixel (circCtr.x - circPt.y, circCtr.y + circPt.x);

setPixel (circCtr.x + circPt.y, circCtr.y - circPt.x);

setPixel (circCtr.x - circPt.y, circCtr.y - circPt.x);

}

void circleMidpoint (scrPt circCtr, GLint radius)

{

//画圆函数主体，由你们完成

}

注意：

（1）代码没有主函数及其它OpenGL的初始化函数，请用前面实验一中的代码进行添加，该函数在回调函数myDraw()中调用执行；

（2）代码中出现的结构体类型scrPt表示一个点的坐标，在回调函数myDraw()中调用执行void circleMidpoint (scrPt circCtr, GLint radius)时注意，circCtr是圆心的坐标，需要初始化。

例如：圆心的坐标是（200，200），那么可以写为：

srtPt point;

point.x=200;

point.y=200;

circleMidpoint(point, 100); //圆的半径为100

## 4.实现中点椭圆算法(选做题)

void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)

{

……..

}

**三 关键步骤的源码和注释**

1.

void lineDDA (int x0, int y0, int xEnd, int yEnd){

int dx = xEnd - x0, dy = yEnd - y0, steps, k;

float xIncrement, yIncrement, x = x0, y = y0;

if (fabs (dx) > fabs (dy))

steps = fabs (dx);

else

steps = fabs (dy);

xIncrement = float (dx) / float (steps);

yIncrement = float (dy) / float (steps);

setPixel(round (x), round (y));

for (k = 0; k < steps; k++) {

x += xIncrement;

y += yIncrement;

setPixel (round (x), round (y));

}

}

2.void lineBres (int x0, int y0, int xEnd, int yEnd){

int dx = xEnd - x0, dy = yEnd - y0;

int fu=0;

if(dx<0){

dx=x0-xEnd;

fu=(fu+1)%2;

}

if(dy<0){

dy=y0-yEnd;

fu=(fu+1)%2;

}

if(dy<=dx){//斜率绝对值小于等于1时

int twoDy=2\*dy,twoDyMinusDx=2\*(dy-dx);

int p=twoDy-dx;

int x,y;

if (x0 > xEnd){//让起始点变为x小的一端

x=xEnd;

y=yEnd;

xEnd=x0;

}

else{

x=x0,y=y0;

}

setPixel(x,y);//填充点

while(x<xEnd){

//计算下一个点

x++;

if(p<0){//p<0，y保持不变

p+=twoDy;

}

else{//p>=0，y移动一个单位

if(fu){

y--;

}

else{

y++;

}

p+=twoDyMinusDx;

}

setPixel(x,y);//填充点

}

}

if(dy>dx){//斜率绝对值大于1时

int twoDx=2\*dx,twoDxMinusDy=2\*(dx-dy);

int p=twoDx-dy;

int x,y;

if (y0 > yEnd){

x=xEnd;

y=yEnd;

yEnd=y0;

}

else{

x=x0,y=y0;

}

setPixel(x,y);

while(y<yEnd){

y++;

if(p<0){

p+=twoDx;

}

else{

if(fu){

x--;

}

else{

x++;

}

p+=twoDxMinusDy;

}

setPixel(x,y);

}

}

}

3.

void circlePlotPoints (scrPt circCtr, scrPt circPt)

{

//利用圆的对称性画8个点

setPixel (circCtr.x + circPt.x, circCtr.y + circPt.y);

setPixel (circCtr.x - circPt.x, circCtr.y + circPt.y);

setPixel (circCtr.x + circPt.x, circCtr.y - circPt.y);

setPixel (circCtr.x - circPt.x, circCtr.y - circPt.y);

setPixel (circCtr.x + circPt.y, circCtr.y + circPt.x);

setPixel (circCtr.x - circPt.y, circCtr.y + circPt.x);

setPixel (circCtr.x + circPt.y, circCtr.y - circPt.x);

setPixel (circCtr.x - circPt.y, circCtr.y - circPt.x);

}

void circleMidpoint (scrPt circCtr, GLint radius)

{

//画圆函数主体，由你们完成

scrPt circPt;

GLint p=1-radius;//决策变量

circPt.x=0,circPt.y=radius;

circlePlotPoints(circCtr,circPt);//填充点

while(circPt.x<circPt.y){//第一象限上半部分，其他靠对称

//计算下一个点

circPt.x++;

if(p<0)//p<0，y保持不变

p+=2\*circPt.x+1;

else{//p>=0,y下移一个单位

circPt.y--;

p+=2\*(circPt.x-circPt.y)+1;

}

circlePlotPoints(circCtr,circPt);//填充点

}

}

4.

void ellipsePlotPoints (int xCenter, int yCenter, int dx, int dy)

{

//利用椭圆的对称性画4个点R

setPixel (xCenter + dx, yCenter + dy);

setPixel (xCenter - dx, yCenter + dy);

setPixel (xCenter + dx, yCenter - dy);

setPixel (xCenter - dx, yCenter - dy);

}

void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)

{

//画椭圆函数主体，由你们完成

int Rx\_2=Rx\*Rx;

int Ry\_2=Ry\*Ry;

int twoRx\_2=2\*Rx\_2;

int twoRy\_2=2\*Ry\_2;

int p;//决策参数

int x=0,y=Ry;//填充点

int px=0,py=twoRx\_2\*y;

ellipsePlotPoints(xCenter,yCenter,x,y);

p=round(Ry\_2-(Rx\_2\*Ry)+0.25\*Rx\_2);

while(px<=py){//斜率绝对值小于等于1，绘制上半部分

//计算下一个点

x++;

px+=twoRy\_2;

if(p<0){//p<0,y保持不动

p+=Ry\_2+px;

}else{//p>=0,y下移一个单位

y--;

py-=twoRx\_2;

p+=Ry\_2+px-py;

}

ellipsePlotPoints(xCenter,yCenter,x,y);//填充点

}

//剩下的斜率绝对值大于1，绘制下半部分

p=round(Ry\_2\*pow(x+0.5,2)+Rx\_2\*pow(y-1,2)-Ry\_2\*Rx\_2);

while(y>0){

//计算下一个点

y--;

py-=twoRx\_2;

if(p>0){//p<0,x保持不动

p+=Ry\_2-py;

}else{//p>=0,x右移一个单位

x++;

px+=twoRy\_2;

p+=Ry\_2-py+px;

}

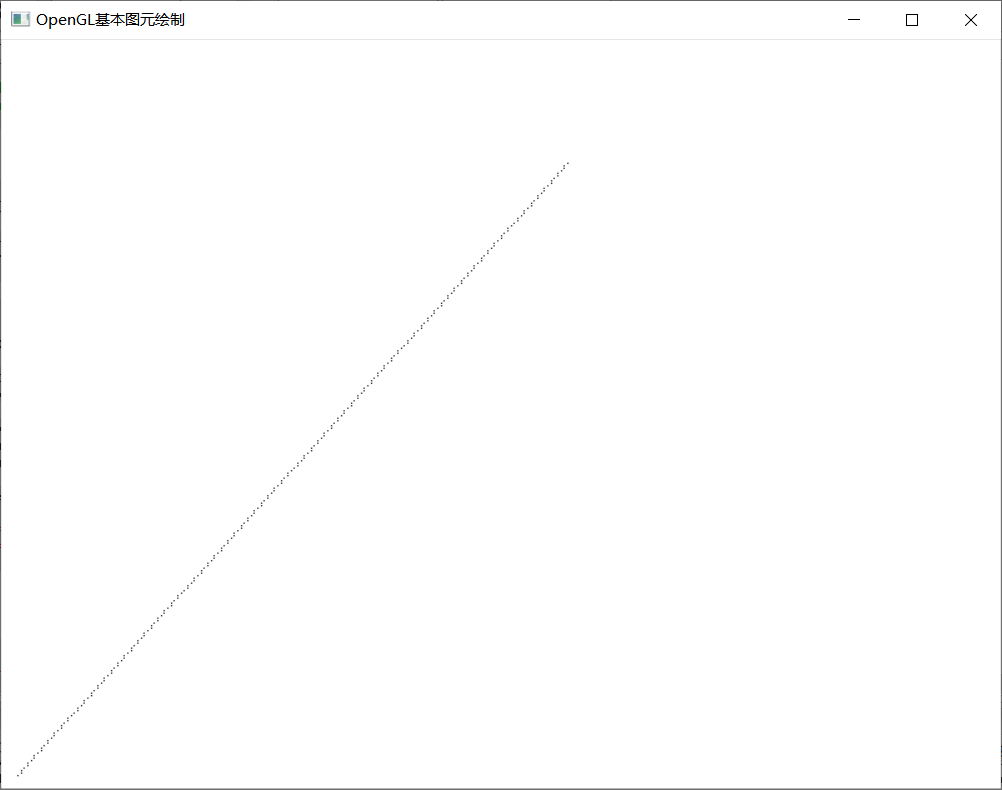
ellipsePlotPoints(xCenter,yCenter,x,y);//填充点

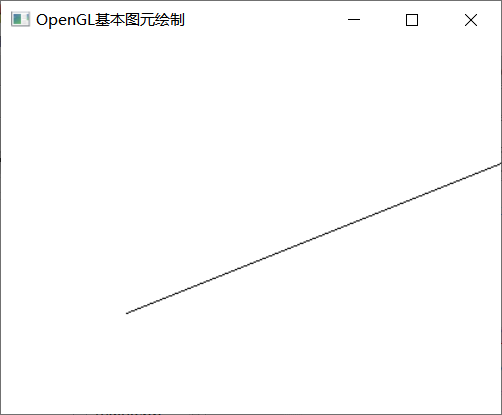
}

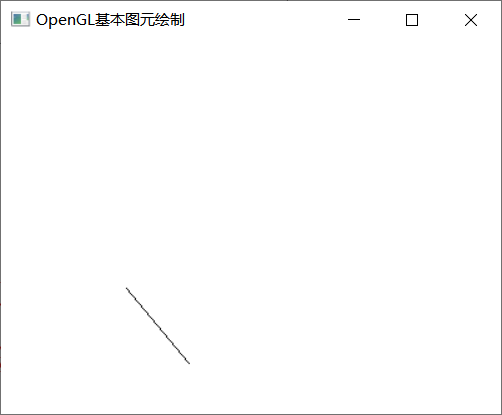
}

四 **实验结果（截图）**

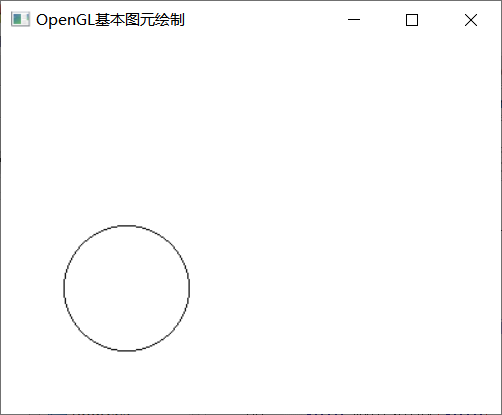
1.



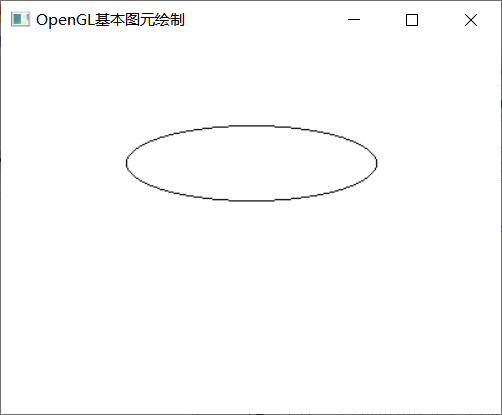




3.



4.



**五、实验分析**

DDA算法是一种非常简单的算法，是一种增量方法，通过不使用乘法来提高效率。但是DDA同时还涉及了浮点运算，其中还涉及了除法，使算法耗时。且浮点计算会降低生成点的准确度，是画出来的线呈现梯子的形状。

Bresenham算法对DDA的浮点运算进一步优化，对于生成点，不在采取计算的方法，而是使用预测的方法。将待决策点到直线的距离差作为决策参数，根据决策参数的正负来选择生成点，整个过程使用的都是整数计算大大提高了运行速度。

中点画圆算法与中点画椭圆算法采用了类似Bresenham算法的思路，通过预测的方法来生成点。避免了浮点运算的开销。此外充分利用了圆的八对称性和椭圆的四对称性减少了大量运算，提高了效率