开发运行环境：VS2019+openGL开发

任务一：

1. 算法的设计思想：假定直线斜率k在0~1之间。此时，只需考虑x方向每次递增1个单位，决定y方向每次递增0或1。设直线当前点为(xi,y) ，直线当前光栅点为(xi,yi) 则下一个直线的点应为(xi+1,y+k) 下一个直线的光栅点或为右光栅点(xi+1,yi)（y方向递增量0）或为右上光栅点(xi+1,yi+1)（y方向递增量1）。记直线与它垂直方向最近的下光栅点的误差为d，有：d=(y+k)–yi，且 0≤d≤1 ，当d<0.5：下一个象素应取右光栅点(xi+1,yi) 当d≥0.5：下一个象素应取右上光栅点(xi+1,yi+1)  如果直线的（起）端点在整数点上，误差项d的初值：d0＝0， x坐标每增加1，d的值相应递增直线的斜率值k，即：d＝d + k。 一旦d≥1，就把它减去1，保证d的相对性，且在0-1之间。 令e=d-0.5，关于d的判别式和初值可简化成：     e的初值e0= -0.5，增量亦为k;     e<0时，取当前象素(xi,yi)的右方象素(xi+1,yi)； e>0时，取当前象素(xi,yi)的右上方象素(xi+1,yi+1)；e=0时，可任取上、下光栅点显示。 Bresenham算法的构思巧妙：它引入动态误差e，当x方向每次递增1个单位，可根据e的符号决定y方向每次递增 0 或 1。e<0，y方向不递增e>0，y方向递增1，x方向每次递增1个单位，e = e + k 因为e是相对量，所以当e>0时，表明e的计值将进入下一个参考点（上升一个光栅点），此时须：e = e - 1 。
2. 源代码：

#include<GL/glut.h>

#include<iostream>

#include<math.h>

using namespace std;

GLfloat pointsize = 1.0f;

double pi = acos(-1.0);

float a[8][3] = {

{0.5,0.5,0.5},

{0,0,255},

{0,128,0},

{0,255,255},

{255,0,0},

{255,0,255},

{255,255,0},

{255,255,255}

};

void drawOneLine(GLint x, GLint y, GLint x1, GLint y1) {

if (x1 == x) {

if (y1 > 0) {

glPointSize(pointsize);

glVertex2i(x, y);

for (GLfloat a = y; a <= y1; a = a + 0.001) {

glVertex2i(x, a);

}

}

else {

glPointSize(pointsize);

glVertex2i(x, y);

for (GLfloat a = y; a >= y1; a = a - 0.001) {

glVertex2i(x, a);

}

}

}

else if (y1 == x) {

if (x1 > 0) {

glPointSize(pointsize);

glVertex2i(x, y);

for (GLfloat a = x; a <= x1; a = a + 0.001) {

glVertex2i(a, y);

}

}

else {

glPointSize(pointsize);

glVertex2i(x, y);

for (GLfloat a = x; a >= x1; a = a - 0.001) {

glVertex2i(a, y);

}

}

}

else {

GLfloat a = x;

GLfloat m = (y1 - y) \* 1.0 / (x1 - x); //斜率

GLfloat b = y - m \* x;

GLfloat thethay = m \* a + b - y; //thetha y

GLfloat d0 = 2 \* thethay - 1; //初始化d0

glPointSize(pointsize);

GLfloat cx = x, cy = y;

glVertex2i(x, y); //画第一个点

std::cout << "a=" << sqrt(x1 \* x1 + y1 \* y1) << endl;

while (a <= x1) {

a += (GLfloat)0.0005;

thethay = m \* a + b - cy; //更新thetha y

if (y1 > y) {

if (d0 <= 0) { //更新d0

d0 += 2 \* thethay;

cy = cy;

}

else {

d0 += 2 \* thethay - 2;

cy = cy + 1;

}

glVertex2i(a, cy);

}

else {

if (d0 >= 0) { //更新d0

d0 += 2 \* thethay;

cy = cy;

}

else {

d0 += 2 \* thethay + 2;

cy = cy - 1;

}

glVertex2i(a, cy);

}

}

while (a >= x1) {

a -= (GLfloat)0.0005;

thethay = m \* a + b - cy; //更新thetha y

if (y1 > y) {

if (d0 <= 0) { //更新d0

d0 += 2 \* thethay;

cy = cy;

}

else {

d0 += 2 \* thethay - 2;

cy = cy + 1;

}

glVertex2i(a, cy);

}

else {

if (d0 >= 0) { //更新d0

d0 += 2 \* thethay;

cy = cy;

}

else {

d0 += 2 \* thethay + 2;

cy = cy - 1;

}

glVertex2i(a, cy);

}

}

}

}

void display(void) {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0f, 1.0f, 1.0f);

glBegin(GL\_POINTS);

for (int n = 0; n < 360; n = n + 5) {

int i = n / 45;

glColor3f(a[i][0],a[i][1],a[i][2]);

drawOneLine(0, 0, (GLint)(10000 \* cos(n / 180.0 \* pi)), (GLint)(10000 \* sin(n / 180.0 \* pi)));

}

glEnd();

glFlush();

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(500, 500);

glutCreateWindow("test");

glClearColor(0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-10000.0, 10000.0, -10000.0, 10000.0);

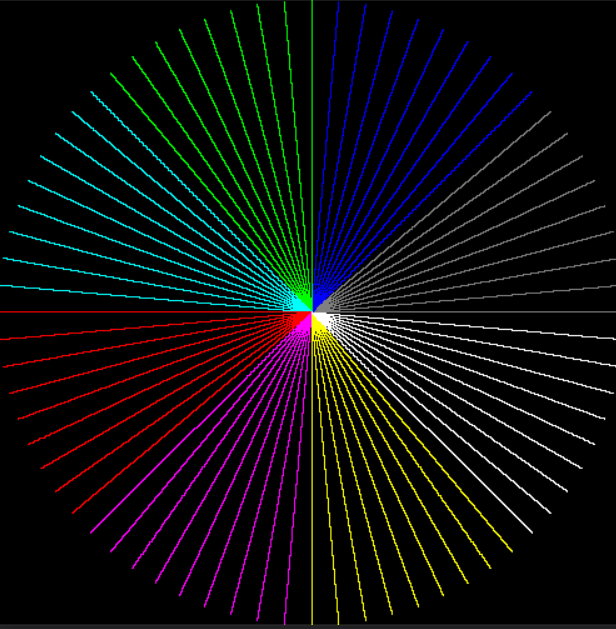
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

1. 运行结果及分析：



任务2

1. 算法的设计思想：Bresenham画圆算法又称中点画圆算法，与Bresenham 直线算法一样，其基本的方法是利用判别变量来判断选择最近的像素点，判别变量的数值仅仅用一些加、减和移位运算就可以计算出来。为了简便起见，考虑一个圆 心在坐标原点的圆，而且只计算八分圆周上的点，其余圆周上的点利用对称性就可得到。
2. 源代码：

#include <Windows.h>

#include <math.h>

#include <GL/glut.h>

#include <iostream>

#define PI acos(-1)

#define WINDOW\_WIDTH 600 //窗口宽度

#define GRID\_WIDTH 20//每个栅格的宽度

//颜色的枚举

enum Color {

red, green, blue, cyan, yellow

};

//栅格绘制

void table(int tableWidth) {

//画列线

glBegin(GL\_LINES);

for (int i = 0-tableWidth / 2; i <= tableWidth / 2; i += GRID\_WIDTH) {

glVertex2i(i, -tableWidth / 2);

glVertex2i(i, tableWidth / 2);

}

glEnd();

//画行线

glBegin(GL\_LINES);

for (int i = 0-tableWidth / 2; i <= tableWidth / 2; i += GRID\_WIDTH) {

glVertex2i(-tableWidth / 2, i);

glVertex2i(tableWidth / 2, i);

}

glEnd();

}

//坐标轴箭头绘制

void arrow(int radiusMin) {

glColor3f(1.0f, 1.0f, 1.0f);

GLint a1 = radiusMin - GRID\_WIDTH;

GLint b1 = 0;

GLint a2 = a1 - GRID\_WIDTH / 2;

GLint b2 = GRID\_WIDTH / 2;

glBegin(GL\_LINE\_STRIP);//x轴箭头

glVertex2i(a2, -b2);

glVertex2i(a1, b1);

glVertex2i(a2, b2);

glEnd();

glBegin(GL\_LINE\_STRIP);//y轴箭头

glVertex2i(b2, a2);

glVertex2i(b1, a1);

glVertex2i(-b2, a2);

glEnd();

}

//像素点绘制

void drawPoint(int x, int y, Color value) {

switch (value) {

case red:

glColor3f(1.0f, 0.0f, 0.0f);

break;

case green:

glColor3f(0.0f, 1.0f, 0.0f);

break;

case blue:

glColor3f(0.0f, 0.0f, 1.0f);

break;

case cyan:

glColor3f(0.0f, 1.0f, 1.0f);

break;

case yellow:

glColor3f(1.0f, 1.0f, 0.0f);

break;

}

// glPointSize((GLfloat)(GRID\_WIDTH/5));

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

//圆逼近点绘制，使用中点圆整数优化算法

void CirclePoints(int radius, int xpoint, int ypoint, Color value) { //xpoint,ypoint为逼近点所在位置

int x = xpoint, y = radius + ypoint;

int d = (int)1.25 - radius;

int dt = 3;

int db = -2 \* radius + 5;

while (y - ypoint >= x - xpoint) {

if (y - ypoint > x - xpoint) {

drawPoint(x, y, value);

drawPoint(y, x, value);

drawPoint(-y, -x, value);

drawPoint(-y, x, value);

drawPoint(y, -x, value);

drawPoint(x, -y, value);

drawPoint(-x, y, value);

drawPoint(-x, -y, value);

}

else if (x - xpoint == 0) {

drawPoint(x, y, value);

drawPoint(x, -y, value);

drawPoint(-y, x, value);

drawPoint(y, -x, value);

}

else if (x - xpoint == y - ypoint) {

drawPoint(x, y, value);

drawPoint(x, -y, value);

drawPoint(-x, y, value);

drawPoint(-x, -y, value);

}

if (d < 0) { //选择T

d += dt;

dt += 2;

db += 2;

}

else {//选择B

d += db;

dt += 2;

db += 4;

y--;

}

x++;

}

}

//中点圆整数算法

void MidPointCircleInt(int radius, int gridWidth, Color value) {

int x = 0, y = radius;

int d = (int)1.25 \* gridWidth \* gridWidth - radius \* gridWidth;

while (y >= x) {

if (y > x) {

CirclePoints(GRID\_WIDTH / 5, x, y, value);

CirclePoints(GRID\_WIDTH / 5, y, x, value);

CirclePoints(GRID\_WIDTH / 5, -y, -x, value);

CirclePoints(GRID\_WIDTH / 5, -y, x, value);

CirclePoints(GRID\_WIDTH / 5, y, -x, value);

CirclePoints(GRID\_WIDTH / 5, x, -y, value);

CirclePoints(GRID\_WIDTH / 5, -x, y, value);

CirclePoints(GRID\_WIDTH / 5, -x, -y, value);

}

else if (x == 0) {

CirclePoints(GRID\_WIDTH / 5, x, y, value);

CirclePoints(GRID\_WIDTH / 5, x, -y, value);

CirclePoints(GRID\_WIDTH / 5, -y, x, value);

CirclePoints(GRID\_WIDTH / 5, y, -x, value);

}

else if (x == y) {

CirclePoints(GRID\_WIDTH / 5, x, y, value);

CirclePoints(GRID\_WIDTH / 5, x, -y, value);

CirclePoints(GRID\_WIDTH / 5, -x, y, value);

CirclePoints(GRID\_WIDTH / 5, -x, -y, value);

}

if (d < 0) //选择T

d += 2 \* x \* gridWidth + 3 \* gridWidth \* gridWidth;

else {//选择B

d += 2 \* (x - y) \* gridWidth + 5 \* gridWidth \* gridWidth;

y -= gridWidth;

}

x += gridWidth;

}

}

//图像显示

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.4f, 0.4f, 0.4f);

int tableWidth = WINDOW\_WIDTH - WINDOW\_WIDTH % (2 \* GRID\_WIDTH);//正方形栅格总宽度

//栅格绘制

table(tableWidth);

int radiusMin = (tableWidth - tableWidth % (10 \* GRID\_WIDTH)) / 10;//最小圆的半径

MidPointCircleInt(radiusMin, GRID\_WIDTH, red);

CirclePoints(radiusMin, 0, 0, red);

MidPointCircleInt(radiusMin \* 2, GRID\_WIDTH, green);

CirclePoints(radiusMin \* 2, 0, 0, green);

MidPointCircleInt(radiusMin \* 3, GRID\_WIDTH, blue);

CirclePoints(radiusMin \* 3, 0, 0, blue);

MidPointCircleInt(radiusMin \* 4, GRID\_WIDTH, cyan);

CirclePoints(radiusMin \* 4, 0, 0, cyan);

MidPointCircleInt(radiusMin \* 5, GRID\_WIDTH, yellow);

CirclePoints(radiusMin \* 5, 0, 0, yellow);

//坐标轴箭头

arrow(radiusMin);

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGBA);

glutInitWindowPosition(350, 50);

glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_WIDTH);

glClearColor(0.0, 0.0, 0.0, 0.0);

glutCreateWindow("Task 2");

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-WINDOW\_WIDTH / 2, WINDOW\_WIDTH / 2, -WINDOW\_WIDTH / 2, WINDOW\_WIDTH / 2);

glutDisplayFunc(display);

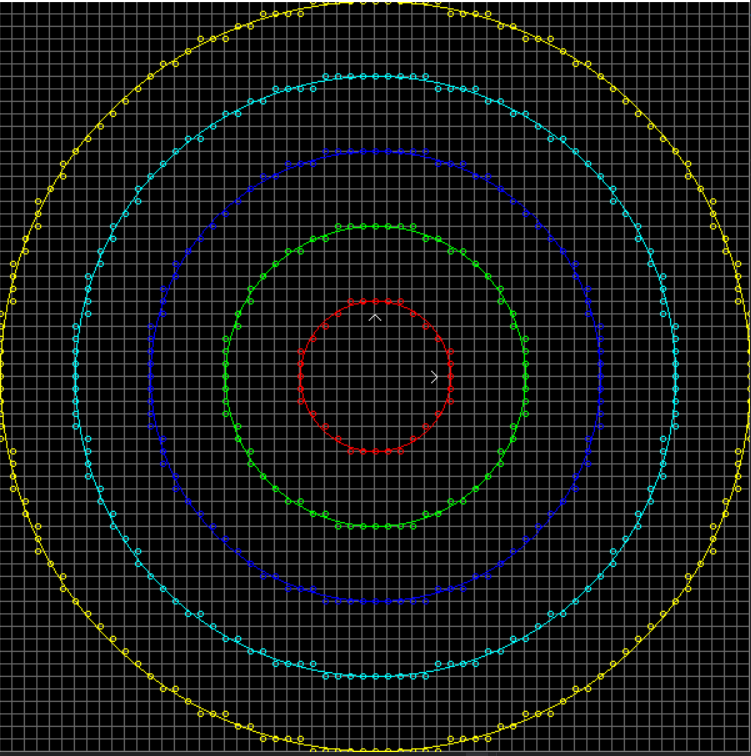
glutMainLoop();

return 0;

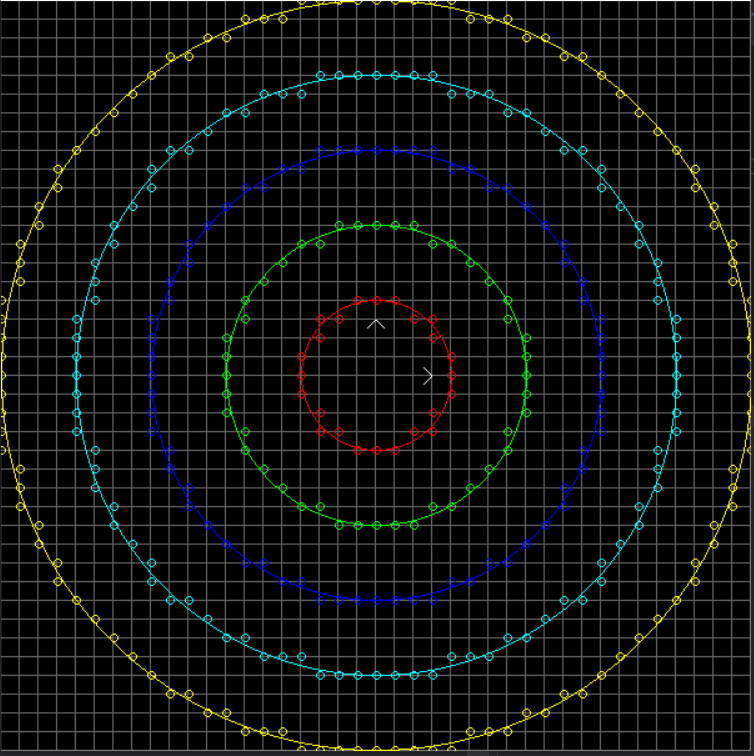
}

1. 运行结果及分析：

（1）光栅宽度为10：



（2）光栅宽度为15：



（3）光栅宽度为20：

