**计算机图形学实验**

**姓 名：曾文**

**学 号：20201120454**

**专 业：计算机科学与技术**

**教 师：钱文华**

实验一 直线段生成算法

实验时间：2022年3月16日

实验地点：信息学院2202机房

实验内容：熟悉OPENGL，通过示例程序生成直线段

实验目的：安装OPENGL，能编写代码运行，参考课本代码。

实验代码：

#include <GL/glut.h>

void init(void) {

glClearColor(0.5, 1.0, 1.0, 0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 200.0, 0.0, 150.0);

}

void linesegment(void) {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.4, 0.2);

glBegin(GL\_LINES);

glVertex2i(180, 15);

glVertex2i(10, 145);

glEnd();

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(50, 100);

glutInitWindowSize(400, 300);

glutCreateWindow("An Example OpenGL Program");

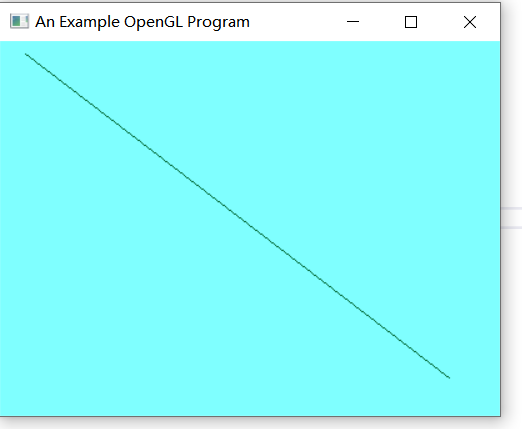
init();

glutDisplayFunc(linesegment);

glutMainLoop();

}

实验结果：



实验二 DDA直线生成算法

时间：2022年3月23日

地点：信息学院2204机房

1、实验内容：

熟悉OPENGL，通过DDA、中点生成算法生成直线段。

2、实验目的：

安装OPENGL，能编写代码运行，能根据键入点坐标正确显示直线段。

3、实验代码：

#include<GL\glut.h>

#include<iostream>

#include<cmath>

using namespace std;

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 200.0, 0.0, 150.0);

}

void LineDDA(int x1,int y1,int x2,int y2)

{

glColor3f(1.0, 0.0, 0.0); // 红色

glPointSize(2.0f);

int dm = 0;

if (abs(x2 - x1) >= abs(y2 - y1))

{

dm = abs(x2 - x1); // x 为计长方向

}

else

{

dm = abs(y2 - y1); // y 为计长方向

}

float dx = (float)(x2 - x1) / dm; // 当 x 为计长方向，dx = 1

float dy = (float)(y2 - y1) / dm; // 当 y 为计长方向，dy = 1

float x = x1;

float y = y1;

for (int i = 0; i < dm; ++i)

{

glBegin(GL\_POINTS);

glVertex2f((int)x, (int)y);

glEnd();

glFlush();

x += dx;

y += dy;

}

return;

}

/\*

交换两个int 类型的变量的值

\*/

void swap\_value(int\* a, int\* b)

{

int tmp = \*a;

\*a = \*b;

\*b = tmp;

}

// 窗口大小改变时调用的登记函数

void ChangeSize(GLsizei w, GLsizei h)

{

if (h == 0) h = 1;

// 设置视区尺寸

glViewport(0, 0, w, h);

// 重置坐标系统

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// 建立修剪空间的范围

if (w <= h)

glOrtho(0.0f, 250.0f, 0.0f, 250.0f \* h / w, 1.0, -1.0);

else

glOrtho(0.0f, 250.0f \* w / h, 0.0f, 250.0f, 1.0, -1.0);

}

void display(void)

{

// 用当前背景色填充窗口，如果不写这句会残留之前的图像

glClear(GL\_COLOR\_BUFFER\_BIT);

int x1, y1, x2, y2;

do {

cout << "请输入两点的坐标：" << endl;

cin >> x1 >> y1 >> x2 >> y2;

} while (0);

LineDDA(x1,y1,x2,y2);

return;

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(200, 200);

glutInitWindowSize(400, 400);

glutCreateWindow("Line");

do {

glutDisplayFunc(display);

} while (0);

glutReshapeFunc(ChangeSize);

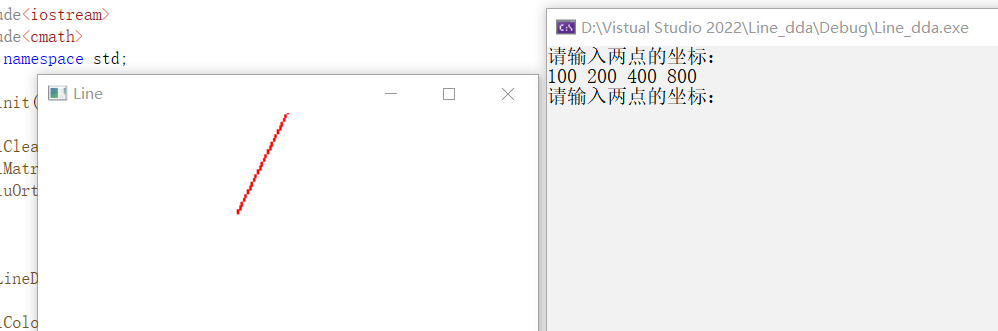
init();

glutMainLoop();

return 0;

}

4、实验结果：



实验三 Bresenham、 改进Bresenham算法生成直线段实验

时间：2022年3月30日

地点：信息学院2204机房

1、实验内容：

熟悉OPENGL，通过Bresenham中点、改进Bresenham算法生成直线段。

2、实验目的：

安装OPENGL，能编写代码运行，参考课本代码。

3、实验代码：

（1）Bresenham算法：

//Bresenham划线算法

//绘制斜率大于0且小于1的线段

#include <GL/glut.h>

#include <iostream>

#include <cmath>

using namespace std;

void init()

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 200.0, 0.0, 150.0);

}

void display()

{

int x1 = 10, y1 = 10, x2 = 150, y2 = 100;

//cout<<"please enter the positions of start point and the end point:x1,y1,x2,y2:"<<endl;

//cin>>x1>>y1>>x2>>y2;

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int x, y;

int temp1 = 2 \* dy;

int temp2 = 2 \* (dy - dx);

int p = temp1 - dx;

if (x1 > x2)

{

x = x2;

y = y2;

x2 = x1;

}

else {

x = x1;

y = y1;

}

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2i(x, y);

while (x < x2)

{

x++;

if (p < 0)

p += temp1;

else

{

y++;

p += temp2;

}

glVertex2i(x, y);

}

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(50, 100);

glutInitWindowSize(400, 300);

glutCreateWindow("Bresenham Draw Line");

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

（2）改进Bresenham算法：

#include<GL/glut.h>

#include<iostream>

#include<math.h>

using namespace std;

GLfloat pointsize = 1.0f;

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

GLint 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);

GLint cx = x, cy = y;

glVertex2i(x, y); //画第一个点

while (a <= x1) {

a++;

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

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) { //设置园的半径

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 1.0f, 0.0f);

glBegin(GL\_POINTS);

drawOneLine(2, 2, 400, 800);

glEnd();

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(400, 400);

glutCreateWindow("Bresenhamplus");

glClearColor(0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-500.0, 800.0, -500.0, 800.0);

glutDisplayFunc(display);

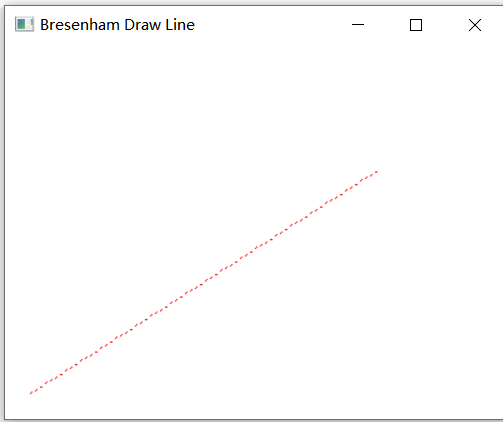
glutMainLoop();

return 0;

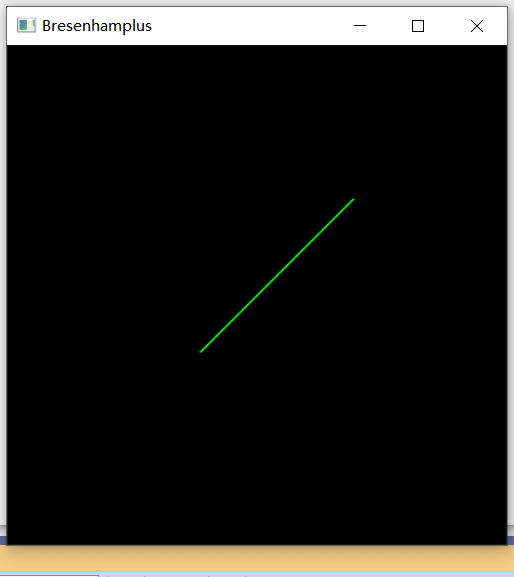
}

4、实验结果：

（1）Bresenham算法：



（2）改进Bresenham算法：



实验四 填充算法实验

时间：2022年4月6日

地点：信息学院2204机房

1、实验内容：

熟悉OPENGL，用扫描线填充算法填充多边形。

2、实验目的：

验证扫描线填充算法，指定任意的多边形边数，填充多边形。

3、实验代码：

//扫描线算法

#include<iostream>

#include<gl/glut.h>

#include<algorithm>

#include<vector>

#include<stack>

#include<queue>

using namespace std;

const int window\_width = 800, window\_height = 600;

const int maxn = 99999;

struct point

{

float x, y;

//point() {}

point(int xx, int yy) :

x(xx), y(yy) {}

};

vector<point> vertice; //顶点

typedef struct XET

{

float x;

float dx; // 从当前扫描线到下一条扫描线间x的增量，即斜率的倒数

float ymax; //该边所交的最高扫描线的坐标值ymax

XET\* next;

}AET, NET; //AET 活性边表； NET新边表

void draw\_a\_point(int x, int y);

void PolyScan();

void mymouse(int button, int state, int x, int y);

void display();

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 50);

glutInitWindowSize(window\_width, window\_height);

glutCreateWindow("扫描线填充");

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, window\_width, 0, window\_height);

glClearColor(1, 1, 1, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glutMouseFunc(&mymouse);

glutDisplayFunc(&display);

glutMainLoop();

return 0;

}

//画点函数

void draw\_a\_point(int x, int y)

{

glBegin(GL\_POINTS);

glColor3f(0, 1, 1);

glVertex2f(x, y);

glEnd();

glFlush();

}

void PolyScan()

{

/\*得到最高点的y坐标\*/

int Max\_Y = 0;

for (int i = 0; i < vertice.size(); i++)

/\*Max\_Y = max(Max\_Y, vertice[i].y);\*/

if (vertice[i].y > Max\_Y)

Max\_Y = vertice[i].y;

//初始化AET表

AET\* pAET = new AET;

pAET->next = NULL;

//初始化NET表

NET\* pNET[800]; //吊桶

for (int i = 0; i <= Max\_Y; i++)

{

pNET[i] = new NET;

pNET[i]->next = NULL;;

}

//扫描并且建立NET表

int len = vertice.size(); //顶点个数

for (int i = 0; i <= Max\_Y; i++)

{

for (int j = 0; j < len; j++) //扫描每个点

{

if (i == vertice[j].y)

{

//如果一个点和前一个点有一条边相连，则该点和后面一个点也相连

//！这个式子 便于最后一个顶点和第一个点相连 和 防止出现负数

//判断当前点的高低，使ymax、DX、DY的计算有变化

if (vertice[(j - 1 + len) % len].y > vertice[j].y)

{

//前一个点在当前点的上方

NET\* p = new NET;

p->x = vertice[j].x;

p->ymax = vertice[(j - 1 + len) % len].y;//与当前扫描线相交的活性边 的 最高点即为相邻顶点的y

float DX = vertice[(j - 1 + len) % len].x - vertice[j].x;

float DY = vertice[(j - 1 + len) % len].y - vertice[j].y;

p->dx = DX / DY;//dx为直线斜率的倒数

p->next = pNET[i]->next;

pNET[i]->next = p;

}

if (vertice[(j + 1) % len].y > vertice[j].y)

{

//后一个点在当前点的上方

NET\* p = new NET;

p->x = vertice[j].x;

p->ymax = vertice[(j + 1) % len].y;

float DX = vertice[(j + 1) % len].x - vertice[j].x;

float DY = vertice[(j + 1) % len].y - vertice[j].y;

p->dx = DX / DY;//dx为直线斜率的倒数

p->next = pNET[i]->next;

pNET[i]->next = p;

}

}

}

}

//建立并且更新活性边表AET

//各条扫描线i

for (int i = 0; i <= Max\_Y; i++)

{

/\*把新边表NET[i] 中的边结点用插入排序法插入AET表，使之按x坐标递增顺序排列\*/

//计算每条扫描线上不同线产生的新的交点x，更新AET

NET\* p = pAET->next;

while (p)

{

p->x = p->x + p->dx; //更新x坐标

p = p->next;

}

//断表排序,不再开辟空间

AET\* tq = pAET;

p = pAET->next;

tq->next = NULL;

while (p)//顺着链表往下走

{

//找到第一个比它大的数字tq->next->next->x，则从p->next到tq->next都是比p->x小的

while (tq->next != NULL && tq->next->x <= p->x)

tq = tq->next;

//插入p到tq和tq->next之间

NET\* t = p->next;

p->next = tq->next;

tq->next = p;

p = t;

tq = pAET;//回到头

}

/\*(改进算法) 取消求交，减少计算量\*/

//先从AET表中删除ymax==i的结点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//像素的取舍问题，保证多边形的“下闭上开”，避免填充扩大化（交点的个数应保证为偶数个）

AET\* q = pAET;

p = q->next;

while (p)

{

if (p->ymax == i)

{

q->next = p->next;

delete p;

p = q->next;

}

else

{

q = q->next;

p = q->next;

}

}

//若NET中有新点，将其用插入法插入AET，按x递增的顺序排列

p = pNET[i]->next;

q = pAET;

while (p)

{

while (q->next && p->x >= q->next->x)

q = q->next;

//插入p

NET\* t = p->next;

p->next = q->next;

q->next = p;

p = t;

q = pAET;//回到头

}

//配对后填充颜色

p = pAET->next;

while (p && p->next != NULL)

{

for (float j = p->x; j <= p->next->x; j++)

{

//扫描线画点

draw\_a\_point(j, i);

//cout << "(" << j << ", " << i << ")" << endl;

}

p = p->next->next;//考虑端点情况

}

}

glFlush();

}

void mymouse(int button, int state, int x, int y)

{

//左键

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

draw\_a\_point(x, window\_height - y);

point p(x, window\_height - y);

vertice.push\_back(p);

cout << "顶点" << vertice.size() << ": (" << x << ", " << window\_height - y << ")" << endl;

}

//右键

if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

{

glClearColor(1, 1, 1, 1);//设置绘制窗口颜色为白色

glColor3f(0, 1, 1);

//绘制多边形

glBegin(GL\_LINES);

for (int i = 0; i < vertice.size(); i++)

{

if (i == vertice.size() - 1)//画完最后一个点，使其闭合

{

glVertex2f(vertice[0].x, vertice[0].y);

glVertex2f(vertice[i].x, vertice[i].y);

}

else

{

glVertex2f(vertice[i].x, vertice[i].y);

glVertex2f(vertice[i + 1].x, vertice[i + 1].y);

}

}

glEnd();

glFlush();

}

//鼠标中键

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN)

{

//cout << "center: (" << x << ", " << y << ")" << endl;

//BoundaryFill4(x, window\_height - y);

//BoundaryFill4\_Stack(x, window\_height - y);

cout << "多边形顶点个数为" << vertice.size() << "。 " << "开始扫描线填充。" << endl;

PolyScan();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.1, 0.5, 0.3);

glPointSize(1);

glBegin(GL\_POINTS);

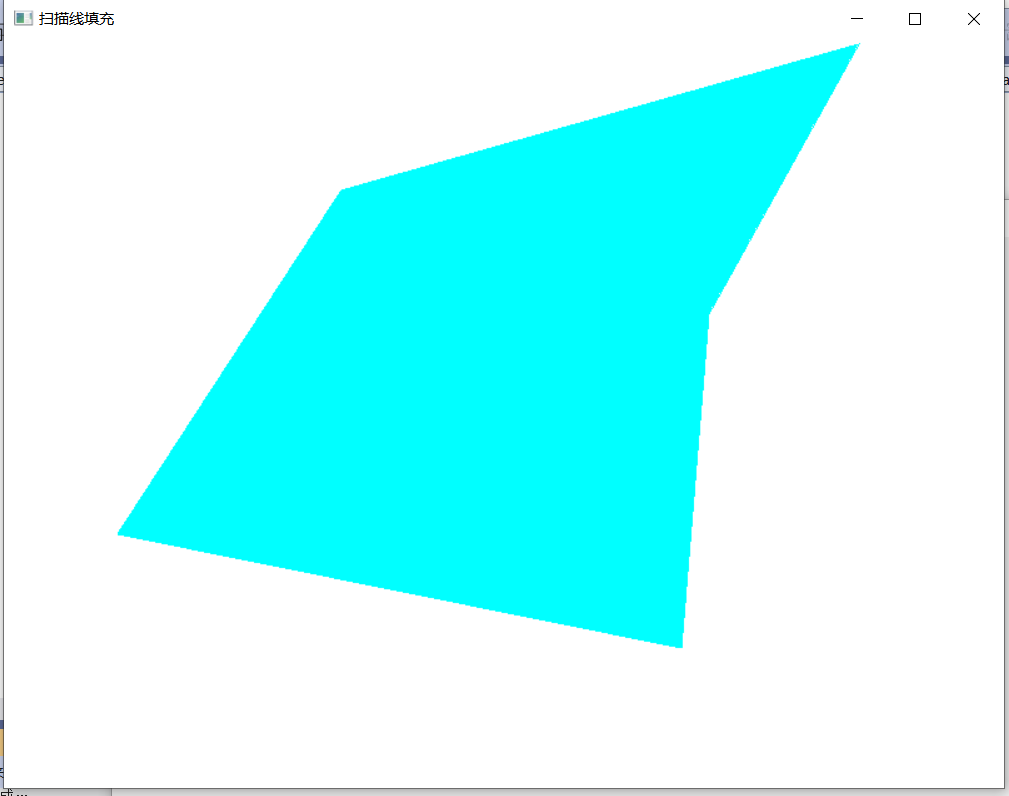
PolyScan();

glEnd();

glFlush();

}

4、实验结果：



实验五 填充算法实验

时间：2022年4月13日

地点：信息学院2202机房

1、实验内容：

（1）圆扫描转换。

（2）种子点填充算法。

2、实验目的：

（1）输入圆的半径，画出圆。

（2）输入多边形，种子点位置，填充多边形。

3、实验代码：

(1).Bresenham画圆：

#include<windows.h>

#include<GL/glut.h>

#include<iostream>

using namespace std;

void init(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(50, 100);

glutInitWindowSize(400, 400);

glutCreateWindow("Bresenham");

glClearColor(1.0, 1.0, 1.0, 0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 400, 0, 400);

}

void Bresenham\_Circle(int xc, int yc, int r)

{

int x, y, d;

x = 0;

y = r;

d = 3 - 2 \* r;

glVertex2i(x + xc, y + yc);

while (x < y)

{

if (d < 0)

{

d = d + 4 \* x + 6;

}

else

{

d = d + 4 \* (x - y) + 10;

y--;

}

x++;

glVertex2i(x + xc, y + yc);

glVertex2i(y + xc, x + yc);

glVertex2i(y + xc, -x + yc);

glVertex2i(x + xc, -y + yc);

glVertex2i(-x + xc, -y + yc);

glVertex2i(-y + xc, -x + yc);

glVertex2i(-x + xc, y + yc);

glVertex2i(-y + xc, x + yc);

}

}

void myDisplay(void)

{

int r;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.8, 0.2);

glPointSize(2);

glBegin(GL\_POINTS);

do {

cout << "请输入圆的半径：" << endl;

cin >> r;

} while (0);

Bresenham\_Circle(200, 200, r);

glEnd();

glFlush();

}

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

{

init(argc, argv);

do {

glutDisplayFunc(myDisplay);

} while (0);

glutMainLoop();

return 0;

}

1. 种子填充：

#include<iostream>

#include<vector>

#include<queue>

#include<stack>

#include<GL/glut.h>

using namespace std;

int window\_width = 800, window\_height = 600;

struct point

{

int x, y;

point() {}

point(int xx, int yy) :

x(xx), y(yy) {}

};

vector<point> vertice; //顶点

float newcolor[3] = { 0, 1, 1 }; //填充颜色

float boundarycolor[3] = { 0, 0, 0 }; //边界颜色

void draw\_a\_point(int x, int y);

bool is\_equal(float\* a, float\* b);

void BoundaryFill4(int x, int y);

//void BoundaryFill4\_Stack(int x, int y);

void mymouse(int button, int state, int x, int y);

void KeyBoards(unsigned char key, int x, int y);

//画点函数

void draw\_a\_point(int x, int y)

{

glBegin(GL\_POINTS);

glColor3fv(newcolor);

glVertex2f(x, y);

glEnd();

glFlush();

}

//判断函数

bool is\_equal(float\* a, float\* b)

{

return a[0] == b[0] && a[1] == b[1] && a[2] == b[2];

}

//用队列queue实现的 4连通区域的填充算法

void BoundaryFill4(int x, int y)

{

queue<point> q;

q.push(point(x, y));

while (!q.empty())

{

point now = q.front();

q.pop();

int nowx = now.x, nowy = now.y;

float color[3];

glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color); //读取像素颜色

if (!is\_equal(color, newcolor) && !is\_equal(color, boundarycolor))

{

draw\_a\_point(nowx, nowy);

q.push(point(nowx, nowy + 1)); //下方

q.push(point(nowx, nowy - 1));

q.push(point(nowx + 1, nowy));

q.push(point(nowx - 1, nowy)); //左方

}

}

}

void mymouse(int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

draw\_a\_point(x, window\_height - y);

point p(x, window\_height - y);

vertice.push\_back(p);

cout << "顶点" << vertice.size() << ": (" << x << ", " << y << ")" << endl;

}

if (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

{

glClearColor(1, 1, 1, 1);//设置绘制窗口颜色为白色

glColor3fv(boundarycolor);

glBegin(GL\_LINES);

for (int i = 0; i < vertice.size(); i++)

{

if (i == vertice.size() - 1)//画完最后一个点，使其闭合

{

glVertex2f(vertice[0].x, vertice[0].y);

glVertex2f(vertice[i].x, vertice[i].y);

}

else

{

glVertex2f(vertice[i].x, vertice[i].y);

glVertex2f(vertice[i + 1].x, vertice[i + 1].y);

}

}

glEnd();

glFlush();

vertice.clear();

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN)

{

cout << "center: (" << x << ", " << y << ")" << endl;

BoundaryFill4(x, window\_height - y);

//BoundaryFill4\_Stack(x, window\_height - y);

}

}

void KeyBoards(unsigned char key, int x, int y)

{

if (key == 32)

{

BoundaryFill4(x, window\_height - y);

glFlush();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.4, 0.2);

glPointSize(1);

glBegin(GL\_POINTS);

//BoundaryFill4(m, window\_height - n);

glEnd();

glFlush();

}

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

{

cout << "点击鼠标左键画点；" << endl << "点击鼠标右键结束画点，形成多边形；" << endl << "点击鼠标中键确定区域填充种子点位置。" << endl;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(400, 150);

glutInitWindowSize(window\_width, window\_height);

glutCreateWindow("区域种子填充");

glMatrixMode(GL\_PROJECTION);/\*设置为投影类型模式和其他观察参数\*/

//glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();/\*设置为投影类型模式和其他观察参数\*/

//gluOrtho2D(0, window\_width, window\_height, 0);/\*设置为投影类型模式和其他观察参数，观察窗口的大小要与画布大小一致，所以直接设置全局变量即可\*/

gluOrtho2D(0, window\_width, 0, window\_height); //上面的不可以！！！

glClearColor(1, 1, 1, 1);//设置绘制窗口颜色为白色

glClear(GL\_COLOR\_BUFFER\_BIT);

glutDisplayFunc(&display);//自己加的，回调函数

glutMouseFunc(&mymouse);

//glutKeyboardFunc(&KeyBoards);

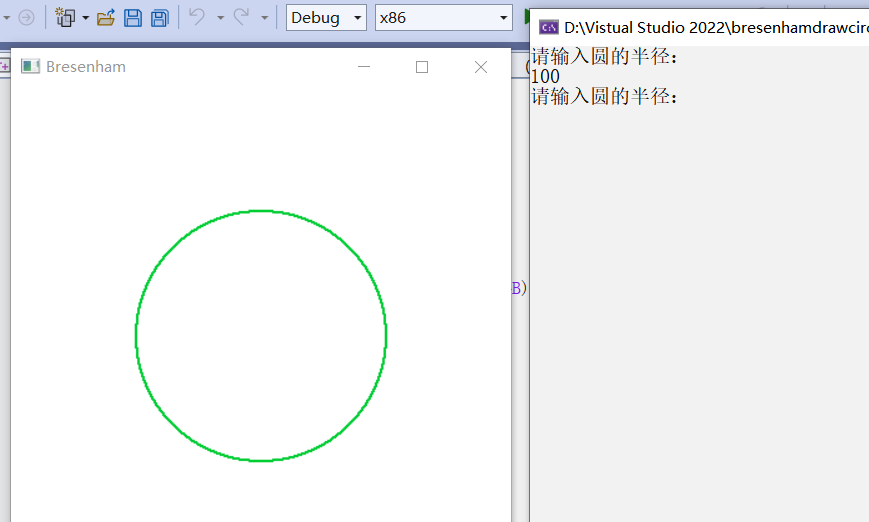
glutMainLoop();

return 0;

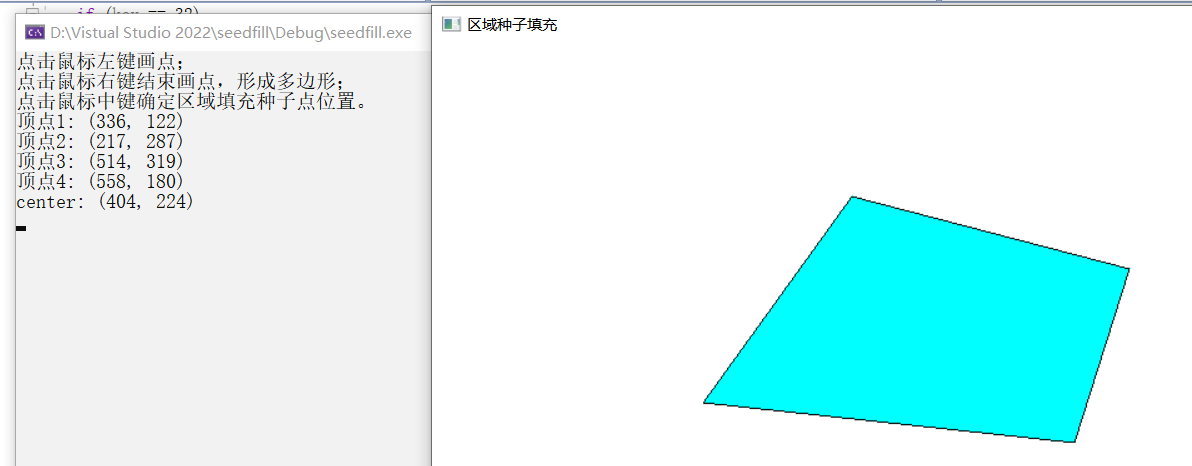
}

4、实验结果：

（1）Bresenham画圆：



（2）种子填充：



实验六 二维几何变换实验

实验时间：2022年4月20日

实验地点：信息学院机房

实验内容：教材P161，二维几何变换算法（平移、比例、旋转、对称）

实验目的：验证二维几何变换，熟悉变换矩阵；

实验代码：

#include <GL\glut.h>

#include <math.h>

#include<iostream>

GLfloat square[4][2];//定义一个正方形的顶点坐标数组

//平移（用键盘的上下左右键控制）

void Translate(GLfloat Tx, GLfloat Ty)

{

    for (int i = 0; i < 4; i++) {

        square[i][0] += Tx;

        square[i][1] += Ty;

    }

}

//比例/缩放（用键盘的a/A键控制x的缩放,d/D键控制y的缩放）

void Scale(GLfloat Sx, GLfloat Sy)

{

    for (int i = 0; i < 4; i++) {

        square[i][0] \*= Sx;

        square[i][1] \*= Sy;

    }

}

//旋转（用键盘的r/R键控制逆时针和正时针旋转）

void Rotate(GLfloat degree)

{

    for (int i = 0; i < 4; i++) {

        int tmpX = square[i][0], tmpY = square[i][1];

        square[i][0] = tmpX \* cos(degree) - tmpY \* sin(degree);

        square[i][1] = tmpX \* sin(degree) + tmpY \* cos(degree);

    }

}

//对称（x键关于x轴对称，y键关于y轴对称）

void Symmetry(int flag)

{

    for (int i = 0; i < 4; i++) {

        if (flag) {//等于1关于x轴对称

            square[i][1] \*= -1;

        }

        else {//等于0关于y轴对称

            square[i][0] \*= -1;

        }

    }

}

//错切（z/Z键向x轴负/正方向错切，c/C键向y轴负/正方向错切）

void Shear(GLfloat b, GLfloat c)

{

    for (int i = 0; i < 4; i++) {

        int tmpX = square[i][0], tmpY = square[i][1];

        square[i][0] = tmpX + tmpY \* c;

        square[i][1] = tmpX \* b + tmpY;

    }

}

//用户自定义初始化

void myinit()

{

    //初始化正方形顶点数组

    square[0][0] = -100.0;

    square[0][1] = 100.0;

    square[1][0] = 100.0;

    square[1][1] = 100.0;

    square[2][0] = 100.0;

    square[2][1] = -100.0;

    square[3][0] = -100.0;

    square[3][1] = -100.0;

}

//窗口大小变化的回调函数

void reshape(GLsizei w, GLsizei h)

{

    glViewport(0, 0, w, h);//设置视口大小比例始终与窗口一致

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-400, 400, -400, 400);

}

void display()

{

    //设置背景颜色为白色并清除颜色缓冲

    glClearColor(1.0, 1.0, 1.0, 1.0);

    glClear(GL\_COLOR\_BUFFER\_BIT);

    //画坐标系

    glColor3f(0.0, 0.0, 1.0);

    glBegin(GL\_LINES);

    //画x轴

    glVertex2f(400.0, 0.0);

    glVertex2f(-400.0, 0.0);

    //画y轴

    glVertex2f(0.0, 400.0);

    glVertex2f(0.0, -400.0);

    glEnd();

    glColor3f(0.0, 1.0, 0.0);

    glPointSize(4.0);

    glBegin(GL\_POINTS);

    //画x、y轴上的坐标

    for (GLfloat i = -400.0; i <= 400.0; i += 100.0) {

        glVertex2f(i, 0.0);

        glVertex2f(0.0, i);

    }

    glEnd();

    //画正方形

    glColor4f(1.0, 0.0, 0.0, 0.1);

    glLineWidth(2.0);

    glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

    glBegin(GL\_POLYGON);

    glVertex2f(square[0][0], square[0][1]);

    glVertex2f(square[1][0], square[1][1]);

    glVertex2f(square[2][0], square[2][1]);

    glVertex2f(square[3][0], square[3][1]);

    glEnd();

    //交换前后缓冲区

    glutSwapBuffers();

}

//键盘上特殊按键的回调函数

void processSpecialKeys(int key, int x, int y)

{

    switch (key) {

    case GLUT\_KEY\_LEFT:

        Translate(-100.0, 0);

        break;

    case GLUT\_KEY\_RIGHT:

        Translate(100.0, 0.0);

        break;

    case GLUT\_KEY\_UP:

        Translate(0.0, 100.0);

        break;

    case GLUT\_KEY\_DOWN:

        Translate(0.0, -100.0);

        break;

    }

    glutPostRedisplay();

}

//键盘普通按键的回调函数

void processNormalKeys(unsigned char key, int x, int y)

{

    switch (key) {

    case 97:    //"a"

        Scale(0.5, 1.0);

        break;

    case 65:    //"A"

        Scale(2.0, 1.0);

        break;

    case 100:   //"d"

        Scale(1.0, 0.5);

        break;

    case 68:    //"D"

        Scale(1.0, 2.0);

        break;

    case 114:   //"r"

        Rotate(15.0);

        break;

    case 82:    //"R"

        Rotate(-15.0);

        break;

    case 120:   //"x"

        Symmetry(1);

        break;

    case 121:   //"y"

        Symmetry(0);

        break;

    case 122:   //"z"

        Shear(0.0, -1.1);

        break;

    case 90:    //"Z"

        Shear(0.0, 1.1);

        break;

    case 99:    //"c"

        Shear(-1.1, 0.0);

        break;

    case 67:    //"C"

        Shear(1.1, 0.0);

        break;

    case 27:    //"esc"

        exit(0);

    }

    glutPostRedisplay();

}

//主函数

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

{

    glutInit(&argc, argv);

    //初始化OPENGL显示方式

    glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

    //设定OPENGL窗口位置和大小

    glutInitWindowSize(400, 400);

    glutInitWindowPosition(100, 100);

    //打开窗口

    glutCreateWindow("多边形的平移、比例（缩放）、旋转、对称和错切等二维仿射变换");

    //调用初始化函数

    myinit();

    //设定窗口大小变化的回调函数

    glutReshapeFunc(reshape);

    //设定键盘控制的回调函数

    glutSpecialFunc(processSpecialKeys);

    glutKeyboardFunc(processNormalKeys);

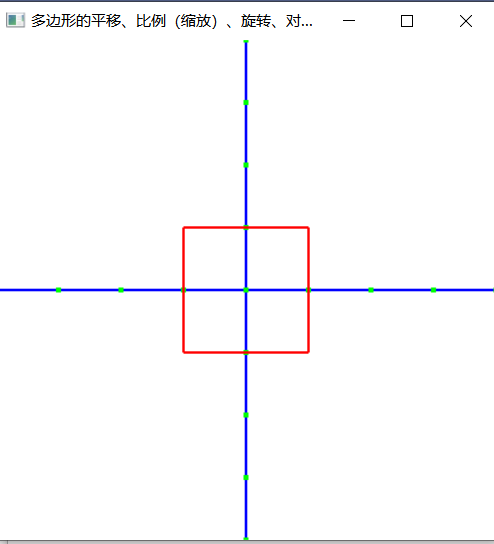
    glutDisplayFunc(display);

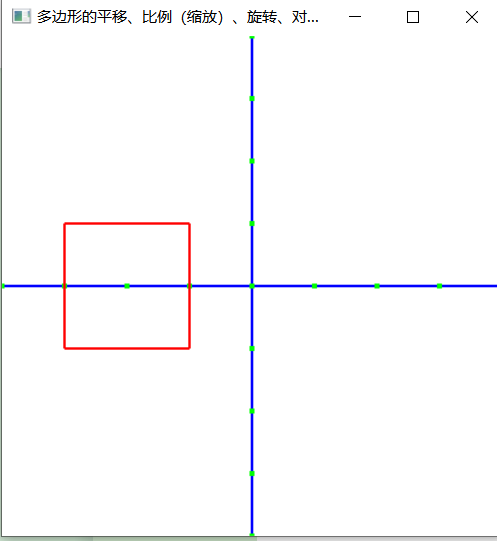
    glutMainLoop();

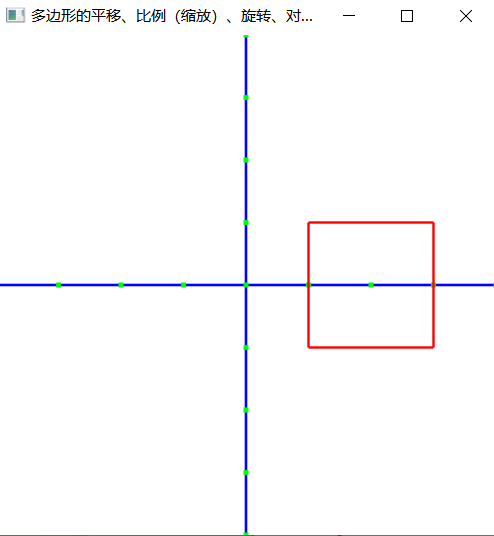
    return 0;

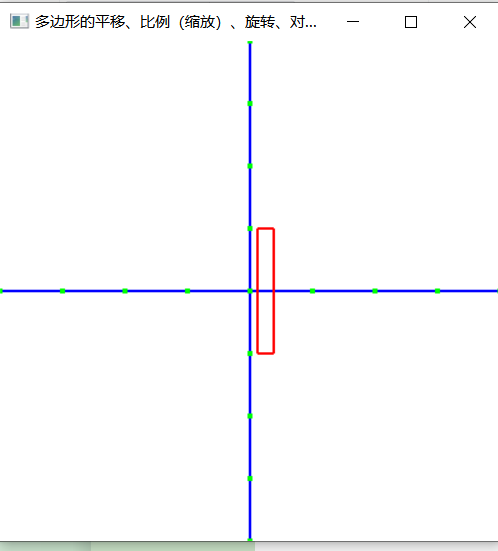
}

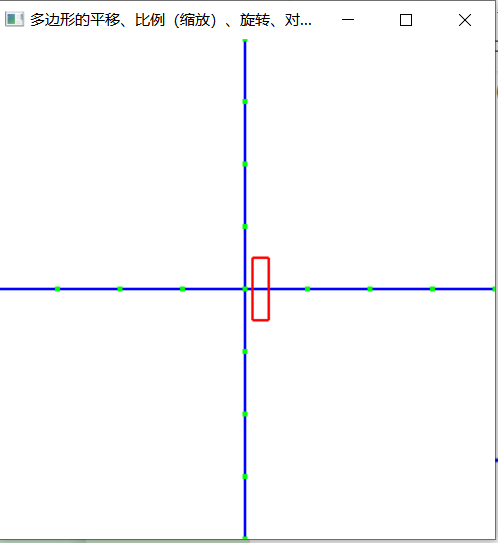
实验结果：











实验总结：

该程序实现了二维图形的基本变换，包括平移、比例、旋转、对称和错切变换，用户通过键盘来决定执行哪种操作。

实验七

实验时间：2022年4月26日

实验地点：信息学院机房

实验内容1：教材P458，GLUT鼠标函数

实验目的：调用鼠标函数完成相应功能，2-3个程序。

实验代码：

7.1：

#include<GL/glut.h>

GLsizei winWidth = 400, winHeight = 300;

void init(void) {

    glClearColor(0.0, 0.0, 1.0, 1.0);

    glMatrixMode(GL\_PROJECTION);

    gluOrtho2D(0.0, 200.0, 0.0, 150.0);

}

void displayFcn(void) {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(1.0, 0.0, 0.0);

    glPointSize(3.0);

}

void winReshapeFcn(GLint newWidth, GLint newHeight) {

    glViewport(0, 0, newWidth, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(0.0, GLdouble(newWidth), 0.0, GLdouble(newHeight));

    winWidth = newWidth;

    winHeight = newHeight;

}

void plotPoint(GLint x, GLint y) {

    glBegin(GL\_POINTS);

    glVertex2i(x, y);

    glEnd();

}

void mousePtPlot(GLint button, GLint action, GLint xMouse, GLint yMouse) {

    if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {

        plotPoint(xMouse, winHeight - yMouse);

    }

    glFlush();

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 100);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("xxx");

    init();

    glutDisplayFunc(displayFcn);

    glutReshapeFunc(winReshapeFcn);

    glutMouseFunc(mousePtPlot);

    glutMainLoop();

}

7.2：

#include<GL/glut.h>

GLsizei winWidth = 400, winHeight = 300;

GLint endPtCtr = 0;

class scrpt {

public:

    GLint x, y;

};

void init(void) {

    glClearColor(0.0, 0.0, 1.0, 1.0);

    glMatrixMode(GL\_PROJECTION);

    gluOrtho2D(0.0, 200, 0.0, 150.0);

}

void displayFcn(void) {

    glClear(GL\_COLOR\_BUFFER\_BIT);

}

void winReshapeFcn(GLint newWidth, GLint newHeight) {

    glViewport(0, 0, newWidth, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(0.0, GLdouble(newWidth), 0.0, GLdouble(newHeight));

    winWidth = newWidth;

    winHeight = newHeight;

}

void drawLineSegment(scrpt s1, scrpt s2) {

    glBegin(GL\_LINES);

    glVertex2i(s1.x, s1.y);

    glVertex2i(s2.x, s2.y);

    glEnd();

}

void polyline(GLint button, GLint action, GLint xMouse, GLint yMouse) {

    static scrpt endpt1, endpt2;

    if (endPtCtr == 0) {

        if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {

            endPtCtr = 1;

            endpt1.x = xMouse;

            endpt1.y = winHeight - yMouse;

        }

        else

            if (button == GLUT\_RIGHT\_BUTTON && action == GLUT\_DOWN)

                return;

    }

    else

        if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {

            endpt2.x = xMouse;

            endpt2.y = yMouse;

            drawLineSegment(endpt1, endpt2);

            endpt1 = endpt2;

        }

        else

            if (button == GLUT\_RIGHT\_BUTTON)

                return;

    glFlush();

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 100);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("Drawing   Interactive polyline");

    init();

    glutDisplayFunc(displayFcn);

    glutMouseFunc(polyline);

    glutMainLoop();

}

7.3：

#include<windows.h>

#include<gl/glut.h>

#include<math.h>

#include<iostream>

using namespace std;

#define x\_Screen 800

#define y\_Screen    600

#define little 50

#define middle 20

#define large   8

void myBackground()

{

    glClearColor(0.0, 0.0, 0.0, 1.0);

    glColor3f(1.0, 1.0, 1.0);

}

void myDisplay()

{

    glEnable(GL\_DEPTH\_TEST);

    //如果没有抗锯齿，则点为方形的。如果我们启动抗锯齿设置，则点是一个圆点。

    glEnable(GL\_POINT\_SMOOTH);

    glEnable(GL\_LINE\_SMOOTH);

    glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST); // Make round points, not square points

    glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST);  // Antialias the lines

    glEnable(GL\_BLEND);

    glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    int i;

    glBegin(GL\_POINTS);

    for (i = 0; i < little; i++)

        glVertex2f(50.0 + rand() % x\_Screen, 50.0 + rand() % y\_Screen);

    glEnd();

    glPointSize(2);

    glBegin(GL\_POINTS);

    for (i = 0; i < middle; i++)

        glVertex2f(50.0 + rand() % x\_Screen, 50.0 + rand() % y\_Screen);

    glEnd();

    glPointSize(8);

    glBegin(GL\_POINTS);

    for (i = 0; i < large; i++)

        glVertex2f(50.0 + rand() % x\_Screen, 50.0 + rand() % y\_Screen);

    glEnd();

    glBegin(GL\_POLYGON);

    for (i = 0; i < 64; i++)

        glVertex2f(600 + 50.0 \* cos((float)i / 10), 500 + 50.0 \* sin((float)i / 10));

    glEnd();

    glLineWidth(3);

    glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

    glEnable(GL\_BLEND);

    glEnable(GL\_LINE\_SMOOTH);

    glBegin(GL\_LINE\_STRIP);

    for (i = 0; i < 19; i++) {

        glVertex2f(rand() % 10 + i \* 70, rand() % 50 + 50.0 + (i % 2) \* 80);

    }

    glEnd();

    glutSwapBuffers();

}

void myChange(int w, int h)

{

    glViewport(0, 0, w, h);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(0.0, x\_Screen, 0.0, y\_Screen);

    glMatrixMode(GL\_MODELVIEW);

    glLoadIdentity();

}

void main()

{

    glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

    glutInitWindowSize(x\_Screen, y\_Screen);

    glutCreateWindow("Star");

    glutDisplayFunc(myDisplay);

    glutReshapeFunc(myChange);

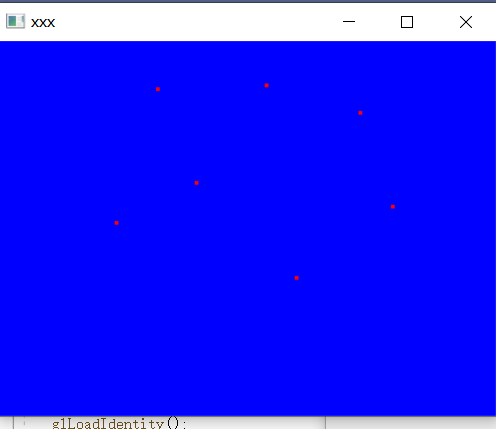
    myBackground();

    glutMainLoop();

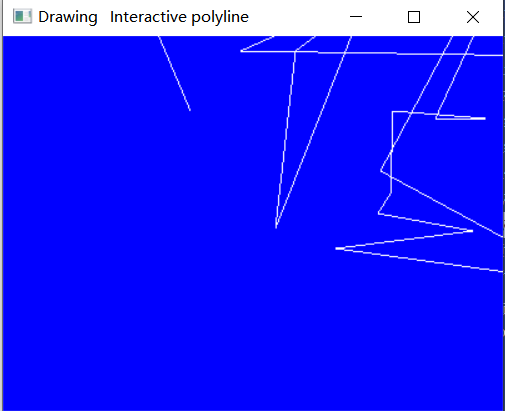
}

实验结果：

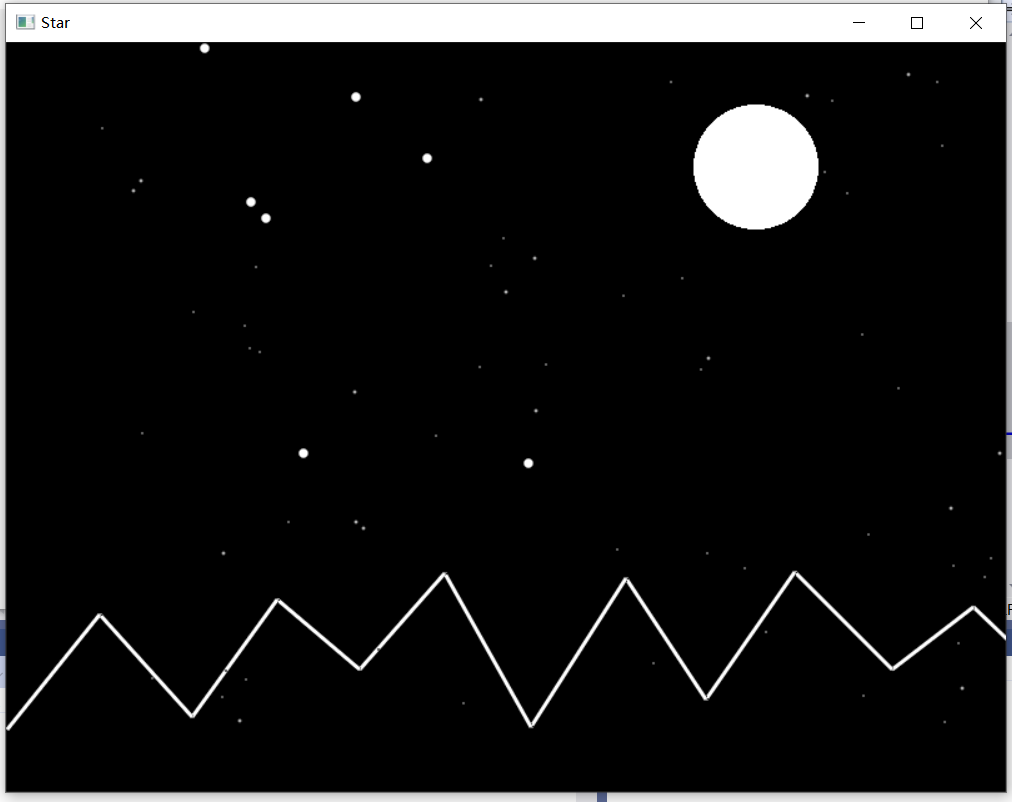
7.1：



7.2：



7.3：



实验总结：

程序7.1实现了用鼠标画点。

程序7.2实现了用鼠标画线。

程序7.3利用openGL库函数实现了反走样。

实验八 二维图像裁剪实验

实验时间：2022年5月3日

实验地点：信息学院机房

实验内容1：使用opengl，用Cohen-Sutherland线段裁剪算法对直线段进行裁剪

实验目的：验证Cohen-Sutherland裁剪算法，从键盘输入任意的直线段，用指定的裁剪窗口裁剪直线段。

实验代码：

#include <GL/glut.h>

#include <cstdio>

#define LEFT 1//0001

#define RIGHT 2//0010

#define BOTTOM 4//0100

#define TOP 8//1000

int x1 = 150, y1 = 50, x2 = 50, y2 = 250, XL = 100, XR = 300, YB = 100, YT = 200;  //(x1,y1)、(x2,y2)为直线段的端点，XL为左边界，XR为右边界，YB为下边界，YT为上边界

int x1\_init = 150, y1\_init = 50, x2\_init = 50, y2\_init = 250;  //将直线段端点备份，以便画出裁剪前的直线段

void encode(int x, int y, int& c)

{//|或运算两者有一个为1结果就为1

    c = 0;

    if (x < XL) c |= LEFT;//按照数字进行或运算，左边界

    else if (x > XR) c |= RIGHT;//右边界

    if (y < YB) c |= BOTTOM;//下边界

    else if (y > YT) c |= TOP;//上边界

}

//上左:1001; 中左:0001; 下左:0101;

//上中:1000; 中中:0000 下中:0100

//上右:1010  中右:0010  下右:0110

void CS\_LineClip()  //Cohen-Sutherland裁剪算法

{

    int x, y;

    int code1, code2, code;

    encode(x1, y1, code1);//先求出端点所在的区号

    encode(x2, y2, code2);//先求出端点所在的区号

    //循环处理

    while (code1 != 0 || code2 != 0)//只有两个端点同时都在矩形区域里才不使用函数

    {

        if (code1 & code2)//两个都为1才成立，这两个都在同一个区域中 直接不进行处理

            return;

        if (code1 != 0)//如果

            code = code1;

        else

            code = code2;

        if (LEFT & code)//线段与左边界相交(如果

        {

            x = XL;//求出的左边界的交点的x值

            y = y1 + (y2 - y1) \* (XL - x1) / (x2 - x1);//求出左边交点的y值

        }

        else if (RIGHT & code)//线段与右边界相交

        {

            x = XR;//求出右边界的交点的x值

            y = y1 + (y2 - y1) \* (XR - x1) / (x2 - x1);//根据斜率计算出右边界的交点的y值

        }

        else if (BOTTOM & code)//线段与下边界相交

        {

            y = YB;//求出下边界的交点的y值

            x = x1 + (x2 - x1) \* (YB - y1) / (y2 - y1);//求出下边界的交点的x值根据斜率

        }

        else if (TOP & code)//线段与上边界相交

        {

            y = YT;

            x = x1 + (x2 - x1) \* (YT - y1) / (y2 - y1);

        }

        if (code == code1)

        {

            x1 = x; y1 = y;  encode(x1, y1, code1);//将区域 外的端点覆盖掉

        }

        else

        {

            x2 = x; y2 = y; encode(x2, y2, code2);//将区域 外的端点覆盖掉

        }

    }

}

void mydisplay()  //显示函数

{

    //绘制方形边界

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(1.0, 0.0, 0.0);//设置背景颜色

    glPointSize(2);//设置点的大小

    glBegin(GL\_LINE\_LOOP);//绘制区域的操作

    glVertex2i(XL, YT);//绘制矩形

    glVertex2i(XL, YB);

    glVertex2i(XR, YB);

    glVertex2i(XR, YT);

    glEnd();

    glFlush();

    //绘制未裁剪前的线段

    glBegin(GL\_LINES);//绘制直线的操作

    glVertex2i(x1\_init, y1\_init);

    glVertex2i(x2\_init, y2\_init);

    glEnd();

    glFlush();

    //绘制裁剪后的线段

    glColor3f(0.0, 0.0, 0.0);

    glBegin(GL\_LINES);

    glVertex2i(x1, y1);

    glVertex2i(x2, y2);

    glEnd();

    glFlush();

    //绘制剪裁区域外的直线段

    x1\_init = 400, y1\_init = 50, x2\_init = 300, y2\_init = 250;

    glColor3f(0.0, 0.0, 0.0);

    glBegin(GL\_LINES);

    glVertex2i(x1\_init, y1\_init);

    glVertex2i(x2\_init, y2\_init);

    glEnd();

    glFlush();

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

    glutInitWindowPosition(500, 100);//窗口位置

    glutInitWindowSize(500, 400);

    glutCreateWindow("Cohen-Sutherland裁剪算法");

    glClearColor(1.0, 1.0, 1.0, 0.0);  //设置背景颜色

    glMatrixMode(GL\_PROJECTION);       // 设置投影参数

    gluOrtho2D(0.0, 600.0, 0.0, 400.0); // 设置场景的大小

    CS\_LineClip();  //执行一次裁剪算法

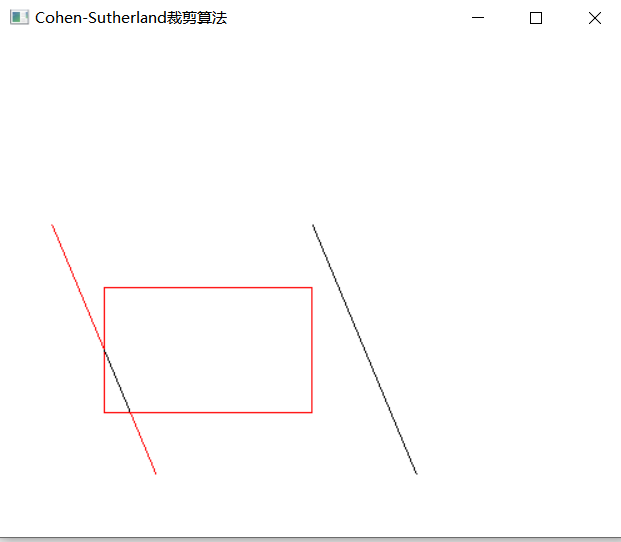
    glutDisplayFunc(&mydisplay);//调用绘制函数

    glutMainLoop();

    return 0;

}

实验结果：



实验总结：

该程序实现了Cohen-Sutherland裁剪算法裁剪直线，用户从键盘输入直线段，用指定的裁剪窗口裁剪直线。

实验九 三维图形几何变换实验

实验时间：2022年5月11日

实验地点：信息学院机房

实验内容2：教材P222，三维图形旋转、缩放变换、平移变换、错切变换、对称变换等任意变换。

实验目的：调用函数完成三维图形几何变换。

实验代码：

#include <GL/glut.h>

#include<stdlib.h>

#include<math.h>

GLsizei winWidth = 600, winHeight = 600;

GLint nVerts = 12;//点的个数

GLfloat tx = 0, ty = 1, tz = 3;//存储要平移的距离

GLfloat angle = 60;//旋转角度

GLfloat scale = 2.0;//缩放比例

typedef GLfloat M4[4][4];

M4 matComposite;

class pt3D { public: GLfloat x, y, z; };

                   pt3D verts[] = {

                           { 0.0, 1.0, 0.0 }, { -1.0, -1.0, 1.0 }, { 1.0, -1.0, 1.0 },

                           { 0.0, 1.0, 0.0 }, { 1.0, -1.0, 1.0 }, { 1.0, -1.0, -1.0 },

                           { 0.0, 1.0, 0.0 }, { 1.0, -1.0, -1.0 }, { -1.0, -1.0, -1.0 },

                           { 0.0, 1.0, 0.0 }, { -1.0, -1.0, -1.0 }, { -1.0, -1.0, 1.0 }

                   };//存放三维物体的各个点坐标，由于有4个三角形面，所以有12个点，设置顶点一致就好，拼接三角形，底面不做绘制

                   pt3D resultVerts[12];//存放变换后的矩阵，即最新的点坐标

class color { public: GLfloat r, g, b; };

                    color colors[] = {

                            { 1.0, 0.0, 0.0 }, { 0.0, 1.0, 0.0 }, { 0.0, 0.0, 1.0 }, { 1.0, 1.0, 0.0 }

                    };//存放每个面的颜色

                    void init()

                    {

                        glClearColor(1.0, 1.0, 1.0, 0.0);

                        glOrtho(-5.0, 5.0, -5.0, 5.0, -5.0, 5.0);

                        glMatrixMode(GL\_PROJECTION);

                    }

                    void m4SetIdentity(M4 matIdentity4x4)

                    {

                        GLint col, row;

                        for (row = 0; row < 4; row++) {

                            for (col = 0; col < 4; col++) {

                                matIdentity4x4[row][col] = (row == col);

                            }

                        }

                    }

                    void m4PreMultiply(M4 m1, M4 m2)

                    {

                        GLint row, col;

                        M4 matTemp;

                        for (row = 0; row < 4; row++) {

                            for (col = 0; col < 4; col++) {

                                matTemp[row][col] = m1[row][0] \* m2[0][col] + m1[row][1] \* m2[1][col] + m1[row][2] \* m2[2][col] + m1[row][3] \* m2[3][col];

                            }

                        }

                        for (row = 0; row < 4; row++) {

                            for (col = 0; col < 4; col++) {

                                m2[row][col] = matTemp[row][col];

                            }

                        }

                    }

                    void translate3D(GLfloat tx, GLfloat ty, GLfloat tz)

                    {

                        M4 matTranslate3D;

                        m4SetIdentity(matTranslate3D);

                        matTranslate3D[0][3] = tx;

                        matTranslate3D[1][3] = ty;

                        matTranslate3D[2][3] = tz;

                        m4PreMultiply(matTranslate3D, matComposite);

                    }

                    void transformVerts3D()

                    {

                        GLint k;

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

                            resultVerts[k].x = matComposite[0][0] \* verts[k].x + matComposite[0][1] \* verts[k].y + matComposite[0][2] \* verts[k].z + matComposite[0][3];

                            resultVerts[k].y = matComposite[1][0] \* verts[k].x + matComposite[1][1] \* verts[k].y + matComposite[1][2] \* verts[k].z + matComposite[1][3];

                            resultVerts[k].z = matComposite[2][0] \* verts[k].x + matComposite[2][0] \* verts[k].y + matComposite[2][2] \* verts[k].z + matComposite[2][3];

                        }

                    }

                    //画一个三棱锥，每个面设不同的颜色

                    void draw(pt3D\* mat)

                    {

                        int j;

                        for (int i = 0; i < 4; i++) {

                            glColor3f(colors[i].r, colors[i].g, colors[i].b);

                            for (j = i \* 3; j < i \* 3 + 3; j++) {

                                glVertex3f(mat[j].x, mat[j].y, mat[j].z);

                            }

                        }

                    }

                    void displayFunc()

                    {

                        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

                        glRotatef(30, 0.0f, 1.0f, 0.0f);

                        glBegin(GL\_LINES);//画坐标轴，可以省略，主要是为了看旋转变化

                        glColor3f(1.0, 0.0, 0.0);//y轴红色

                        glVertex3f(0, 0, 0);

                        glVertex3f(0, 4, 0);

                        glColor3f(0.0, 1.0, 0.0);

                        glVertex3f(0, 0, 0);

                        glVertex3f(4, 0, 0);//x轴绿色

                        glColor3f(0.0, 0.0, 1.0);

                        glVertex3f(0, 0, 0);

                        glVertex3f(0, 0, 3);//z轴蓝色

                        glEnd();

                        glBegin(GL\_TRIANGLES);                          // 绘制三角形

                        draw(verts);

                        glEnd();

                        glLoadIdentity();//不可以省略，否则后面对其它三维物体的变换也会发生在上面已经绘制的物体中

                        GLfloat scaleBack = 1 / scale;

                        glRotatef(angle, 0.0, 1.0, 0.0);

                        glScalef(1.0, scale, 1.0);

                        //GLfloat tx = 0, ty = 1.0, tz = 3;

                        m4SetIdentity(matComposite);

                        translate3D(tx, ty, tz);

                        transformVerts3D();

                        glBegin(GL\_TRIANGLES);

                        draw(resultVerts);

                        glEnd();

                        glScalef(1.0, scaleBack, 1.0);

                        glLoadIdentity();

                        glFlush();

                    }

                    /\*

                    键盘设置，下面的还可以改善：当旋转或平移到一定数值时，回到原始状态

                     \*/

                    void processSpecialKeys(int key, int x, int y)

                    {

                        switch (key) {

                        case GLUT\_KEY\_UP:ty += 0.1; break;

                        case GLUT\_KEY\_DOWN:ty -= 0.1; break;

                        case GLUT\_KEY\_LEFT:tz += 0.1; break;

                        case GLUT\_KEY\_RIGHT:tz -= 0.1; break;

                        case GLUT\_KEY\_PAGE\_UP:scale += 0.1; break;

                        case GLUT\_KEY\_PAGE\_DOWN:scale -= 0.1; break;

                        case GLUT\_KEY\_INSERT:angle += 10; break;

                        case GLUT\_KEY\_END:tx = 0; ty = 1; tz = 3; angle = 60; scale = 2; break;//回到原始状态

                        default:break;

                        }

                        displayFunc();

                    }

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

                    {

                        glutInit(&argc, argv);

                        glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

                        glutInitWindowPosition(50, 50);

                        glutInitWindowSize(winWidth, winHeight);

                        glutCreateWindow("3D");

                        init();

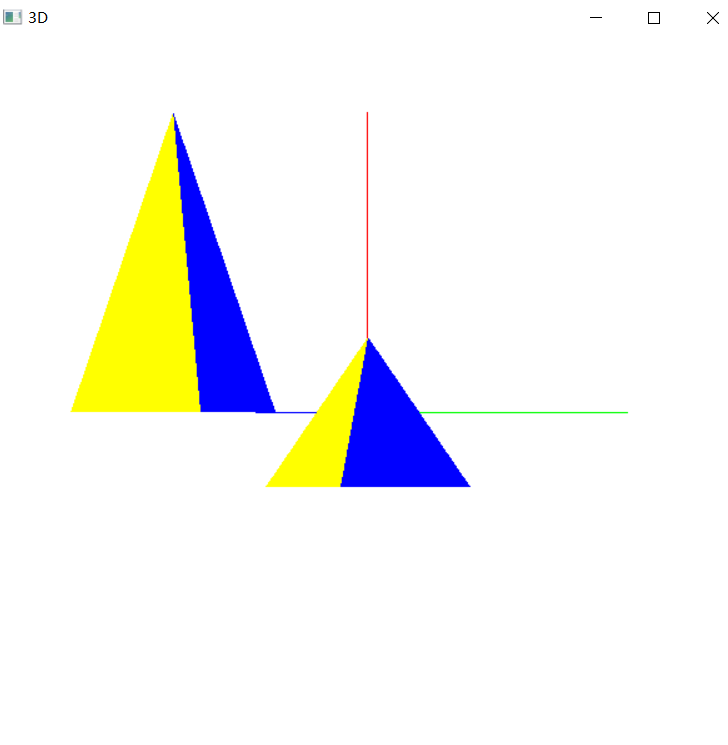
                        glutDisplayFunc(displayFunc);

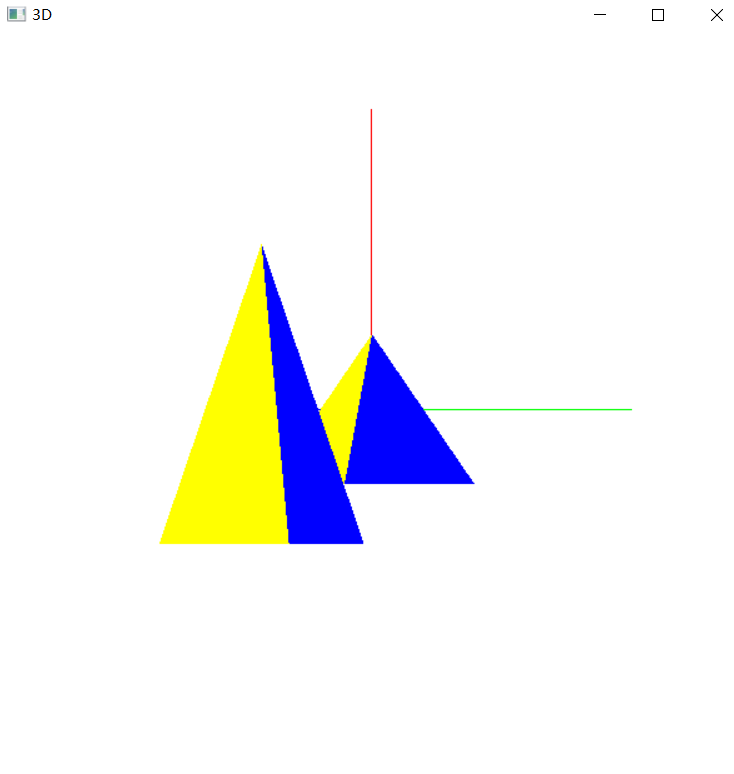
                        glutSpecialFunc(processSpecialKeys);

                        glutMainLoop();

                    }

实验结果：





实验总结：

该程序实现了三维图形的基本变换，包括平移、比例、旋转、对称和错切变换，用户通过键盘来决定执行哪种操作。

实验十 建模实验

实验时间：2022年5月17日

实验地点：信息学院机房

实验内容2：使用opengl，教材P541，颜色编码建模显示。

实验目的：调用函数完成颜色编码建模实验。

实验代码：

#include<GL/glut.h>

//设置一个窗口的大小

GLsizei winWidth = 500, winHeight = 500;

GLfloat xComplexMin = -2.00, xComplexMax = 0.50;

GLfloat yComplexMin = -1.25, yComplexMax = 1.25;

GLfloat complexWidth = xComplexMax - xComplexMin;

GLfloat complexHeight = yComplexMax - yComplexMin;

class complexNum {

public:

    GLfloat x, y;

};

struct  color { GLfloat r, g, b; };

void init(void)

{

    glClearColor(1.0, 1.0, 1.0, 0.0);

}

void plotpoint(complexNum z)

{

    glBegin(GL\_POINTS);

    glVertex2f(z.x, z.y);

    glEnd();

}

complexNum complexSquare(complexNum z)

{

    complexNum zSquare;

    zSquare.x = z.x \* z.x - z.y \* z.y;

    zSquare.y = 2 \* z.x \* z.y;

    return zSquare;

}

GLint mandelSqTransf(complexNum z0, GLint maxIter)

{

    complexNum z = z0;

    GLint count = 0;

    while ((z.x \* z.x + z.y \* z.y <= 4.0) && (count < maxIter)) {

        z = complexSquare(z);

        z.x += z0.x;

        z.y += z0.y;

        count++;

    }

    return count;

}

void mandelbrot(GLint nx, GLint ny, GLint maxIter)

{

    complexNum z, zIncr;

    color ptColor;

    GLint iterCount;

    zIncr.x = complexWidth / GLfloat(nx);

    zIncr.y = complexHeight / GLfloat(ny);

    for (z.x = xComplexMin; z.x < xComplexMax; z.x += zIncr.x)

        for (z.y = yComplexMin; z.y < yComplexMax; z.y += zIncr.y) {

            iterCount = mandelSqTransf(z, maxIter);

            if (iterCount >= maxIter)

                ptColor.r = ptColor.g = ptColor.b = 0.0;

            else if (iterCount > (maxIter / 10)) {

                ptColor.r = 1.0;

                ptColor.g = 0.5;

                ptColor.b = 0.0;

            }

            else if (iterCount > (maxIter / 10)) {

                ptColor.r = 1.0;

                ptColor.g = ptColor.b = 0.0;

            }

            else if (iterCount > (maxIter / 20)) {

                ptColor.b = 0.5;

                ptColor.r = ptColor.g = 0.0;

            }

            else if (iterCount > (maxIter / 100)) {

                ptColor.r = ptColor.b = 0.0;

                ptColor.g = 0.3;

            }

            else {

                ptColor.r = 0.0;

                ptColor.g = ptColor.b = 1.0;

            }

            glColor3f(ptColor.r, ptColor.g, ptColor.b);

            plotpoint(z);

        }

}

void displayFcn(void)

{

    GLint nx = 1000, ny = 1000, maxIter = 1000;

    glClear(GL\_COLOR\_BUFFER\_BIT);

    mandelbrot(nx, ny, maxIter);

    glFlush();

}

void winReshapeFcn(GLint newWidth, GLint newHeight) {

    glViewport(0, 0, newHeight, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(xComplexMin, xComplexMax, yComplexMin, yComplexMax);

    glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(50, 50);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("Mandelbrot 集");

    init();

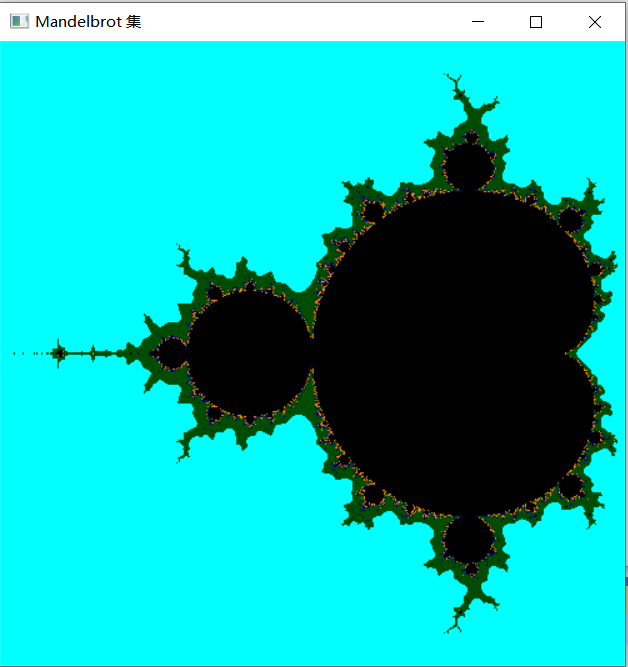
    glutDisplayFunc(displayFcn);

    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();

}

实验结果：



实验总结：

该程序实现了颜色编码建模显示。

实验十一 交互控制实验

实验时间：2022年5月17日

实验地点：信息学院机房

实验内容1：使用opengl，完成鼠标、键盘交互操作

实验目的：熟悉鼠标、键盘交互

实验代码：

#include<GL/glut.h>

#include<stdio.h>

GLsizei winWidth = 500, winHeight = 500;

char sixel;

float thera = 0;

float x = 0, y = 0, z = 0;

void init(void) {

    glClearColor(1.0, 1.0, 1.0, 0.0);

}

void displayWirePolyhedra(float x, float y, float z, float thera) {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.0, 0.0, 1.0);

    gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glScalef(1.0, 1.0, 1.0);

    glTranslatef(1.0, 2.0, 0.0);//下一个图形坐标

    glutSolidTeapot(1.5);

    //glutWireTeapot(1.5);//放大倍数

    glScalef(1.0, 1.0, 1.0);//缩放比

    glTranslatef(-1.0, -5.0, 0.0);//下一个图形坐标

    glRotatef(thera, x, y, z);

    glutWireTeapot(1.5);

    //glutSolidTeapot(2.0);

    glFlush();

}

void display() {

    displayWirePolyhedra(x, y, z, thera);

}

void winReshapeFcn(GLint newWidth, GLint newHeight) {

    glViewport(0, 0, newWidth, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);

    glMatrixMode(GL\_MODELVIEW);

    glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 100);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("   ");

    init();

    printf\_s("请选择绕拿一个轴旋转x,y,z \n");

    scanf\_s("%c", &sixel);

    getchar();

    if (sixel == 'x') {

        x = 1.0;

        y = 0.0;

        z = 0.0;

        printf\_s("请输入旋转的角度\n");

        scanf\_s("%f", &thera);

    }

    else if (sixel == 'y') {

        x = 0.0;

        y = 1.0;

        z = 0.0;

        printf\_s("请输入旋转的角度\n");

        scanf\_s("%f", &thera);

    }

    else if (sixel == 'z') {

        x = 0.0;

        y = 0.0;

        z = 1.0;

        printf\_s("请输入旋转的角度\n");

        scanf\_s("%f", &thera);

    }

    else {

        printf\_s("输入有误\n");

    }

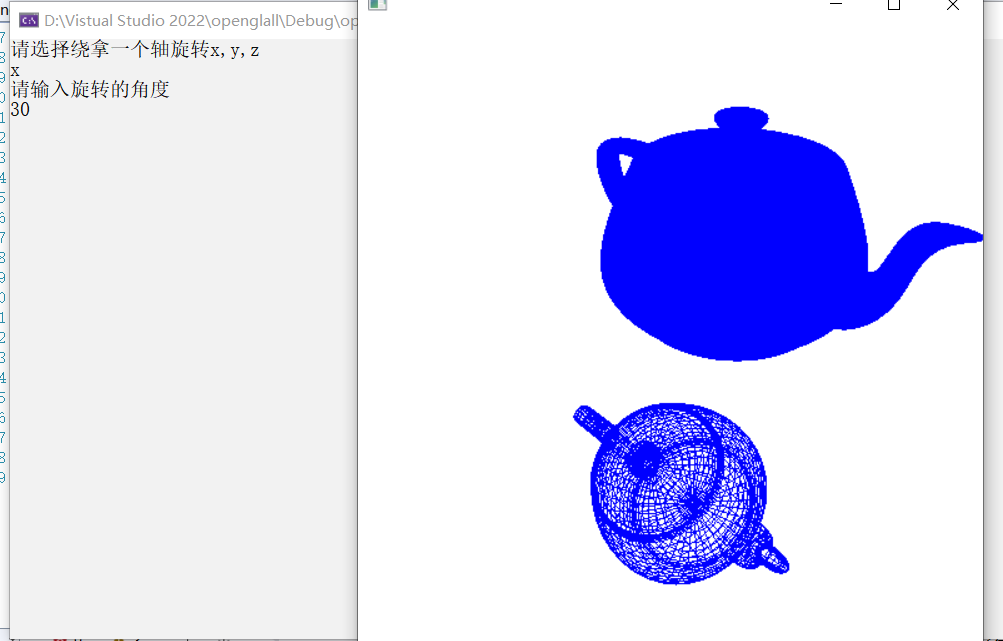
    glutDisplayFunc(display);

    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();

}

实验结果：



实验总结：

该程序创建了一个三维的茶壶，用户能通过鼠标、键盘对其进行操作。

实验十二 三维观察实验

实验时间：2022年5月25日

实验地点：信息学院机房

实验内容：1、使用opengl，完成投影变换等实验

2、P 264

实验目的：熟悉三维观察相关内容

实验代码：

12.1：

#include<GL/glut.h>

GLint winWidth = 600, winHeight = 600; // Initialdisplay - window size

GLfloat x0 = 100.0, y0 = 50.0, z0 = 50.0;

GLfloat xref = 50.0, yref = 50.0, zref = 0.0;// Viewing - coordinate origin

GLfloat Vx = 0.0, Vy = 1.0, Vz = 0.0;

GLfloat xwMin = -40.0, ywMin = -60.0, xwMax = 40.0, ywMax = 60.0;

GLfloat dnear = 25.0, dfar = 125.0;

void  init(void) {

    glClearColor(1.0, 1.0, 1.0, 0.0);

    glMatrixMode(GL\_MODELVIEW);

    gluLookAt(x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);

    glMatrixMode(GL\_PROJECTION);

    glFrustum(xwMin, xwMax, ywMin, ywMax, dnear, dfar);

}

void displayFcn(void) {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    /\*Set parameters for a square fi11 area. \*/

    glColor3f(0.0, 1.0, 0.0);

    glPolygonMode(GL\_FRONT, GL\_FILL);

    glPolygonMode(GL\_BACK, GL\_LINE);

    glBegin(GL\_QUADS);

    glVertex3f(0.0, 0.0, 0.0);

    glVertex3f(100.0, 0.0, 0.0);

    glVertex3f(100.0, 100.0, 0.0);

    glVertex3f(0.0, 100.0, 0.0);

    glEnd();

    glFlush();

}

void reshapeFcn(GLint newWidth, GLint newHeight) {

    glViewport(0, 0, newWidth, newHeight);

    winWidth = newWidth;

    winHeight = newHeight;

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(50, 50);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("Perspective View of A Square");

    init();

    glutDisplayFunc(displayFcn);

    glutReshapeFunc(reshapeFcn);

    glutMainLoop()；

}

12.2：

#include <stdlib.h>

#include <GL/glut.h>

float theta = 0.0;

void drawPyramid() //该金字塔在以原点为中心，边长为2的立方体范围内

{

    glBegin(GL\_TRIANGLES);

    glColor3f(1.0f, 0.0f, 0.0f);      //前面为红色

    glVertex3f(0.0f, 1.0f, 0.0f);   //前面三角形上顶点

    glVertex3f(-1.0f, -1.0f, 1.0f);  //前面三角形左顶点

    glVertex3f(1.0f, -1.0f, 1.0f);  //前面三角形右顶点

    glColor3f(0.0f, 1.0f, 0.0f);        //右面为绿色

    glVertex3f(0.0f, 1.0f, 0.0f);   //右面三角形上顶点

    glVertex3f(1.0f, -1.0f, 1.0f);  //右面三角形左顶点

    glVertex3f(1.0f, -1.0f, -1.0f);  //右面三角形右顶点

    glColor3f(0.0f, 0.0f, 1.0f);        //背面为蓝色

    glVertex3f(0.0f, 1.0f, 0.0f);  //背面三角形上顶点

    glVertex3f(1.0f, -1.0f, -1.0f);  //背面三角形左顶点

    glVertex3f(-1.0f, -1.0f, -1.0f);    //背面三角形右顶点

    glColor3f(1.0f, 1.0f, 0.0f);        //左面为黄色

    glVertex3f(0.0f, 1.0f, 0.0f);   //左面三角形上顶点

    glVertex3f(-1.0f, -1.0f, -1.0f);    //左面三角形左顶点

    glVertex3f(-1.0f, -1.0f, 1.0f); //左面三角形右顶点

    glEnd();

    glBegin(GL\_POLYGON);  //金字塔底面正方形

    glColor3f(0.5f, 0.5f, 0.5f);  //底面为灰色

    glVertex3f(-1.0f, -1.0f, 1.0f);

    glVertex3f(1.0f, -1.0f, 1.0f);

    glVertex3f(1.0f, -1.0f, -1.0f);

    glVertex3f(-1.0f, -1.0f, -1.0f);

    glEnd();

}

void display()

{

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); //清空颜色和深度缓存

    glMatrixMode(GL\_MODELVIEW);

    glLoadIdentity();

    //gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glTranslatef(0.0f, 0.0f, -5.0f);

    glRotatef(theta, 0.0f, 1.0f, 0.0f);

    drawPyramid();

    glutSwapBuffers();

}

void reshape(int w, int h) //重绘回调函数，在窗口首次创建或用户改变窗口尺寸时被调用

{

    glViewport(0, 0, w, h);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    //glFrustum(-1.0, 1.0, -1.0, 1.0, 3.1, 10.0);

    //gluPerspective(45,1,0.1,10.0);

    glOrtho(-2.0, 2.0, -2.0, 2.0, 2.0, 10.0);

}

void init()

{

    glClearColor(1.0, 1.0, 1.0, 1.0);

    glEnable(GL\_DEPTH\_TEST);     //启动深度测试模式

}

void myKeyboard(unsigned char key, int x, int y)

{

    if (key == 'a' || key == 'A')

        theta += 5.0;

    if (key == 's' || key == 'S')

        theta -= 5.0;

    if (key == 'c' || key == 'C')

        exit(0);

    if (theta > 360) theta -= 360;

    if (theta < 0) theta += 360;

    glutPostRedisplay(); //重新调用绘制函数

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGB);

    glutInitWindowSize(500, 500);

    glutInitWindowPosition(0, 0);

    glutCreateWindow("金字塔---A键:顺时针旋转,S键:逆时针旋转,C键:退出");

    glutReshapeFunc(reshape); //指定重绘回调函数

    glutDisplayFunc(display);

    glutKeyboardFunc(myKeyboard);   //指定键盘回调函数

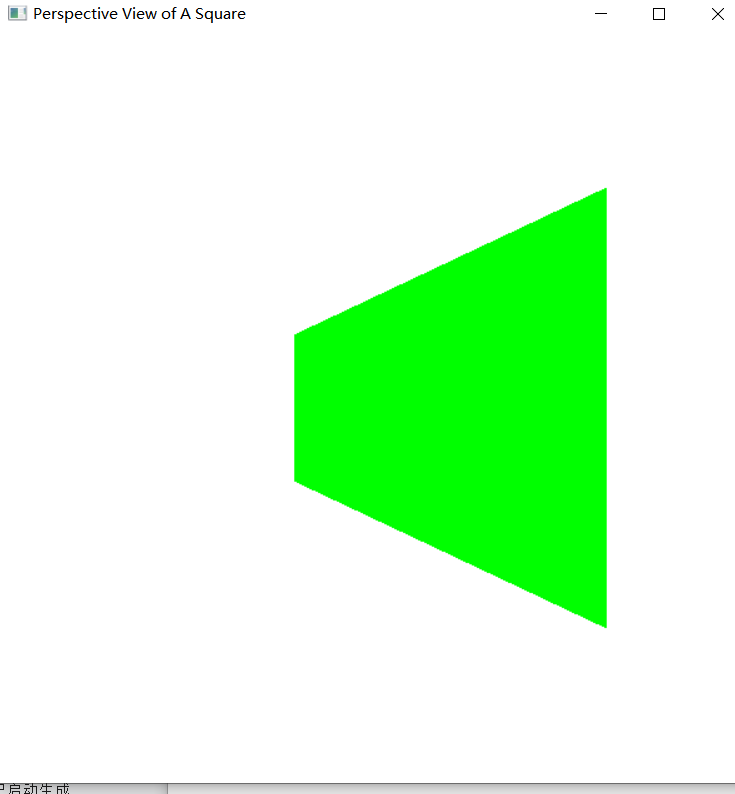
    init();

    glutMainLoop();

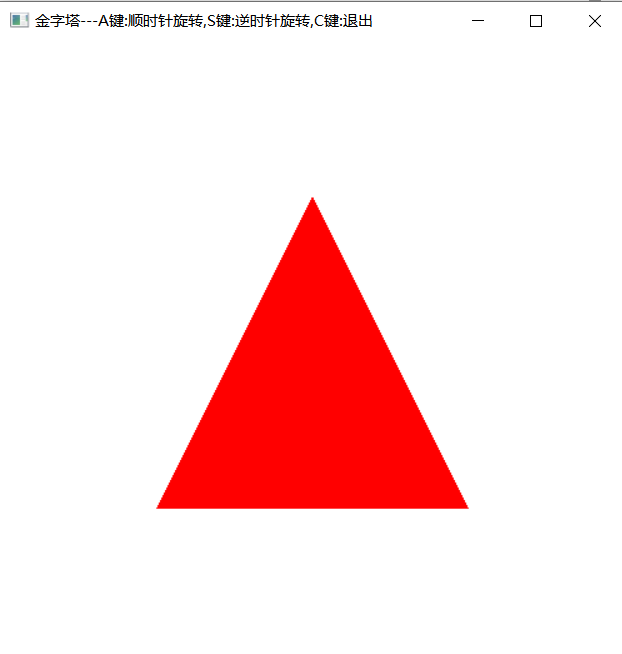
}

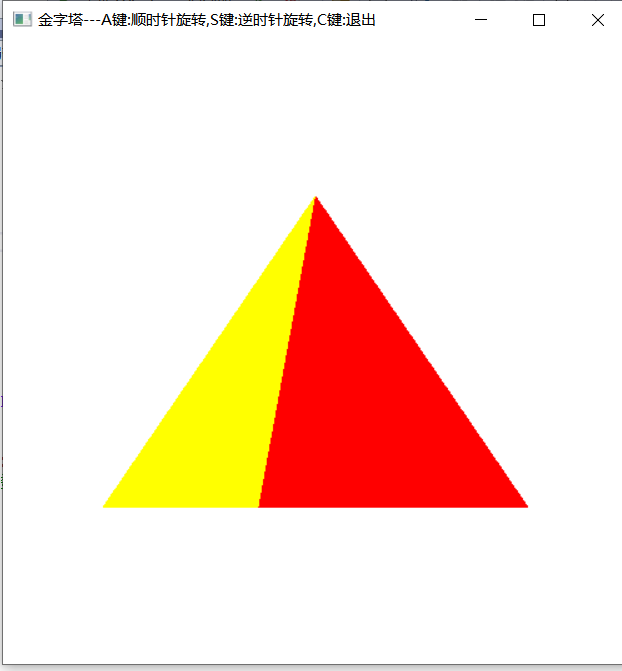
实验结果：

12.1：



12.2：





实验总结：

程序12.1将一个绿色梯形放在视区内展示。

程序12.2创建了一个金字塔，金字塔四个侧面的颜色依次为红、绿、蓝、黄（逆时针方向），用户通过键盘可控制其旋转。

实验十三 多面体实验

实验时间：2022年6月1日

实验地点：信息学院机房

实验内容：生成多面体线框图，P300、P307

实验目的：熟悉三维线框图相关内容

实验代码：

13.1：

#include <windows.h>

#include <GL/glut.h>

GLsizei winWidth = 500, winHeight = 500;

void init(void) {

    glClearColor(1.0, 1.0, 1.0, 0.0);

}

void displayWirePolyhedra(void) {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.0, 0.0, 1.0);

    gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glScalef(1.5, 2.0, 1.0);

    glutWireCube(1.0);

    glScalef(0.8, 0.5, 0.8);

    glTranslatef(-6.0, -5.0, 0.0);

    glutWireDodecahedron();

    glTranslatef(8.6, 8.6, 2.0);

    glutWireTetrahedron();

    glTranslatef(-3.0, -1.0, 0.0);

    glutWireOctahedron();

    glScalef(0.8, 0.8, 1.0);

    glTranslatef(4.3, -2.0, 0.5);

    glutWireIcosahedron();

    glFlush();

}

void winReshapeFcn(GLint newWidth, GLint newHeight) {

    glViewport(0, 0, newWidth, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);

    glMatrixMode(GL\_MODELVIEW);

    glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    //glutInitWindowPosition(100, 100);

    //glutInitWindowSize(winWidth, winHeight);

    int cx = glutGet(GLUT\_SCREEN\_WIDTH);

    int cy = glutGet(GLUT\_SCREEN\_HEIGHT);

    glutInitWindowPosition((cx - winWidth) / 2, (cy - winHeight) / 2);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("Wire-Frame Po1yhedra");

    init();

    glutDisplayFunc(displayWirePolyhedra);

    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();

}

13.2：

#include <windows.h>

#include <GL/glut.h>

GLsizei winWidth = 500, winHeight = 500;

void init(void)

{

    glClearColor(1.0, 1.0, 1.0, 0.0);

}

void wireQuadSurfs(void)

{

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.0, 0.0, 1.0);

    gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);

    glPushMatrix();

    glTranslatef(1.0, 1.0, 0.0);

    glutWireSphere(0.75, 8, 6);

    glPopMatrix();

    glPushMatrix();

    glTranslatef(1.0, -0.5, 0.5);

    glutWireCone(0.7, 2.0, 7, 6);

    glPopMatrix();

    GLUquadricObj\* cylinder;

    glPushMatrix();

    glTranslatef(0.0, 1.2, 0.8);

    cylinder = gluNewQuadric();

    gluQuadricDrawStyle(cylinder, GLU\_LINE);

    gluCylinder(cylinder, 0.6, 0.6, 1.5, 6, 4);

    glPopMatrix();

    glFlush();

}

void winReshapeFcn(GLint newWidth, GLint newHeight)

{

    glViewport(0, 0, newWidth, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glOrtho(-2.0, 2.0, -2.0, 2.0, 0.0, 5.0);

    glMatrixMode(GL\_MODELVIEW);

    glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(100, 100);

    glutInitWindowSize(winWidth, winHeight);

    glutCreateWindow("二次曲面");

    init();

    glutDisplayFunc(wireQuadSurfs);

    glutReshapeFunc(winReshapeFcn);

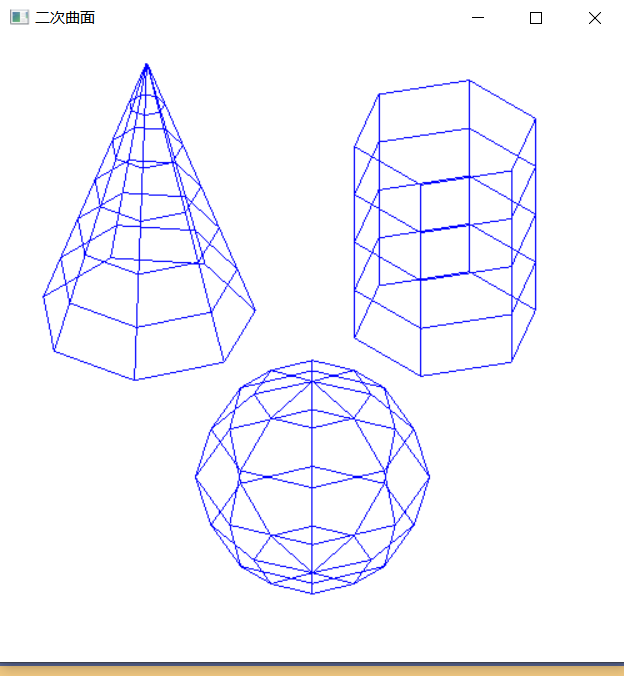
    glutMainLoop();

    return 0;

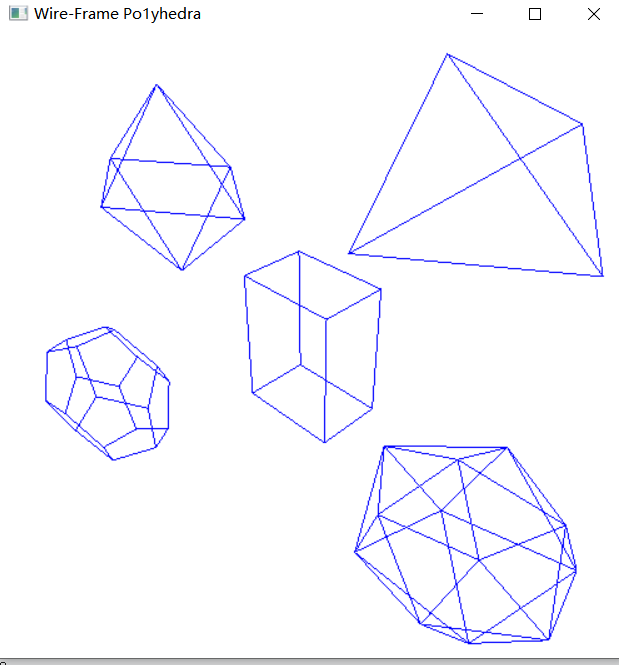
}

实验结果：

13.2：



13.1：



实验总结：

程序10.1将五个多面体展示出来。

程序10.2生成了三个二次曲面。

实验十四 曲线曲面生成实验

实验时间：2022年6月7日

实验地点：信息学院机房

实验内容：生成曲线或者曲面，P323

实验目的：熟悉Bezier、样条等相关内容

实验代码：

#include <GL/glut.h>

#include <stdio.h>

#include <math.h>

GLsizei winWIdth = 600, winHeight = 600;

GLfloat xwcMin = -50.0, xwcMax = 50.0, ywcMin = -50.0, ywcMax = 50.0;

class wcPt3D

{

public: GLfloat x, y, z;

};

void init(void)

{

    glClearColor(1.0, 1.0, 1.0, 0.0);

}

void plotPoint(wcPt3D p)

{

    glBegin(GL\_POINTS);

    glVertex2f(p.x, p.y);

    glEnd();

}

void binomialCoeffs(GLint n, GLint\* C)

{

    GLint i, j;

    for (i = 0; i <= n; i++)

    {

        C[i] = 1;

        for (j = n; j >= i + 1; j--)

        {

            C[i] \*= j;

        }

        for (j = n - i; j >= 2; j--)

        {

            C[i] /= j;

        }

    }

}

void computeBezPt(GLfloat u, wcPt3D\* bezPt, GLint nCtrlPts, wcPt3D\* ctrlPts, GLint\* C)

{

    GLint i, j = nCtrlPts - 1;

    GLfloat bezBlendFcn;

    bezPt->x = bezPt->y = bezPt->z = 0.0;

    for (i = 0; i < nCtrlPts; i++)

    {

        bezBlendFcn = C[i] \* pow(u, j) \* pow(1 - u, j - i);

        bezPt->x += ctrlPts[i].x \* bezBlendFcn;

        bezPt->y += ctrlPts[i].y \* bezBlendFcn;

        bezPt->z += ctrlPts[i].z \* bezBlendFcn;

    }

}

void bezier(wcPt3D\* ctrlPts, GLint nCtrlPts, GLint nBezPts)

{

    GLint\* C, k;

    GLfloat u;

    wcPt3D bezPt;

    C = new GLint[nCtrlPts];

    binomialCoeffs(nCtrlPts - 1, C);

    for (k = 0; k <= nBezPts; k++)

    {

        u = GLfloat(k) / GLfloat(nBezPts);

        computeBezPt(u, &bezPt, nCtrlPts, ctrlPts, C);

        plotPoint(bezPt);

    }

    delete[] C;

}

void displayFcn(void)

{

    GLint nCtrlPts = 4, nBezCurvePts = 1000;

    wcPt3D ctrlPts[4] = {

        {-40.0, -40.0, 0.0},

        {-10.0, 200.0, 0.0},

        {10.0, -200.0, 0.0},

        {40.0, 40.0, 0.0} };

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glPointSize(4);

    glColor3f(1.0, 0.0, 0.0);

    bezier(ctrlPts, nCtrlPts, nBezCurvePts);

    glFlush();

}

void winReshapeFcn(GLint newWidth, GLint newHeight)

{

    glViewport(0, 0, newWidth, newHeight);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);

    glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowPosition(50, 50);

    glutInitWindowSize(winWIdth, winHeight);

    glutCreateWindow("Bezier Curve");

    init();

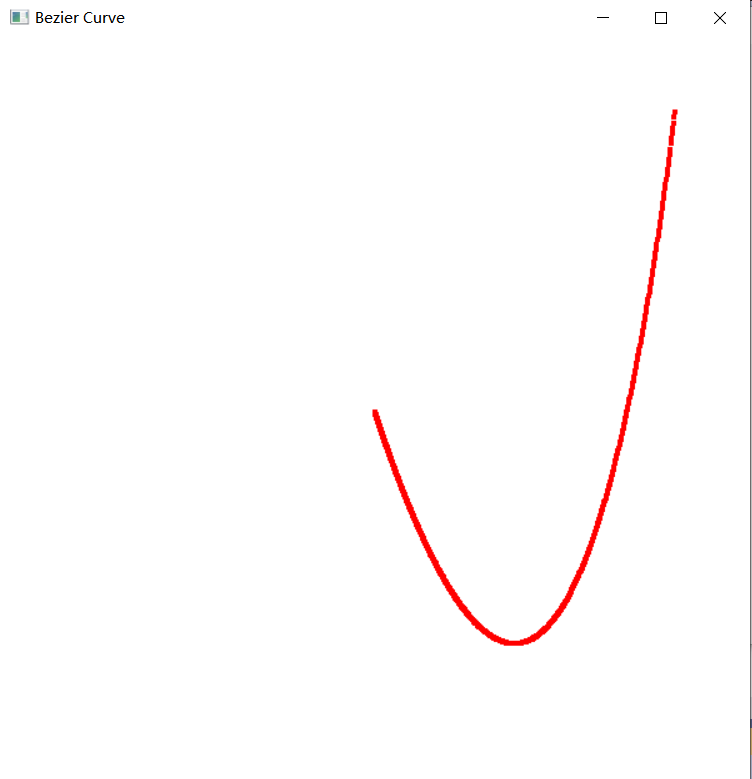
    glutDisplayFunc(displayFcn);

    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();

}

实验结果：



实验总结：

该程序利用插值方法生成一条曲线。

实验十五 消隐实验

实验时间：2022年6月15日

实验地点：信息学院机房

实验内容：完成消隐实验，采用Z-buffer算法完成消隐

实验目的：熟悉Z-buffer 、画家算法等相关内容

实验代码：

import cv2

import numpy as np

import time

XLine=1000

YLine=700

img  =  np.zeros((YLine, XLine, 3), np.uint8)#生成背景

pts=[]

z\_buffer=np.zeros((XLine,YLine),np.uint32)

z\_buffer+=100000  #无穷大

#为了满足需求而设置的值

count=0

def zfunc(x,y):

    return x+y

def draw\_func(event, x, y, flags, param):

    global pts

    if event == cv2.EVENT\_LBUTTONDOWN:

        print('(x:',x,',y:',y,')')

        str1 = '(x:'+ str(x) + ',y:'+ str(y) + ')'

        #cv2.putText(img,str1 , (x, y), cv2.FONT\_HERSHEY\_PLAIN,1.0, (0, 0, 255), thickness=1)

        cv2.circle(img,(x,y),1,(0,255,0),thickness=1)

        pts.append([x,y])

#活性边表与新边表中的节点

class Node:

    def \_\_init\_\_(self,xmin,ymax,k1,next=None):

        self.x=xmin#也就是x的初始值

        self.ymax=ymax

        self.k1=k1#也就是δx

        self.next=next

    def print(self):

        t=self

        while t!=None:

            print(t.x,t.ymax,t.k1,end="-->")

            t=t.next

        print()

def FillPoly(fpts,color):

    global z\_buffer,count

    count+=1

    #cv2.fillPoly(img,fpts,color)

    Ymin,Ymax=1000000,0

    for pt in fpts:

        if Ymin>pt[1]:

            Ymin=pt[1]

        if Ymax<pt[1]:

            Ymax=pt[1]

    #构造新边表

    NET=[x for x in range(Ymax+1)]

    for i in range(len(fpts)):

        #后面一个点的序号

        j=i+1 if i<len(fpts)-1 else 0

        if fpts[i][1]>fpts[j][1]:

            ymax,ymin=fpts[i][1],fpts[j][1]

            xmin=fpts[j][0]

        else:

            ymin,ymax=fpts[i][1],fpts[j][1]

            xmin=fpts[i][0]

        #与x轴平行,跳过该边

        if fpts[i][1]==fpts[j][1]:

            continue

        if fpts[i][0]==fpts[j][0]:

            k1=0

        else:

            k1=(fpts[i][0]-fpts[j][0])/(fpts[i][1]-fpts[j][1])

        if type(NET[ymin])!=type(66):

            t=NET[ymin]

            while t.next!=None:

                t=t.next

            t.next=Node(xmin,ymax,k1)

        else:

            NET[ymin]=Node(xmin,ymax,k1)

    for i,x in enumerate(NET):

        if type(x)!=type(66):

            print(i)

            x.print()

    AET=Node(0,0,0)

    print("YY")

    for y in range(Ymin,Ymax+1):

        #print(y)

        #慢慢画

        cv2.waitKey(3)

        cv2.imshow('src',img)

        #加入新边

        if type(NET[y])!=type(66):

            t=AET

            while t.next:

                t=t.next

            t.next=NET[y]

        #填充颜色

        #按x排序

        xs=[]

        t=AET

        while t.next:

            t=t.next

            xs.append(t.x)

        xs=sorted(xs)

        for i in range(1,len(xs),2):

            for x in range(int(xs[i-1]),int(xs[i])):

                #z=zfunc(x,y)  这里是zbuffer消隐操作

                z=count

                if z==3 and z\_buffer[x,y]==1:#这里是为了交作业 ヽ(ﾟ▽ﾟ)ノ

                    z\_buffer[x,y]=z

                    cv2.circle(img,(x,y),1,color,thickness=1)

                elif z<z\_buffer[x,y]:

                    z\_buffer[x,y]=z

                    cv2.circle(img,(x,y),1,color,thickness=1)

        #删除并更新边

        t=AET

        while t.next:

            ct=t.next

            ct.x+=ct.k1

            if ct.ymax==y:

                t.next=ct.next

            t=t.next

            if t==None:

                break

cv2.namedWindow('src')

cv2.setMouseCallback('src',draw\_func)

while(1):

    cv2.imshow('src',img)

    code=cv2.waitKey(100)

    if code == ord('q'):#按下q退出

        break

    elif code == ord('e'):#按下该键表示画完一个多边形

        tmp=np.array(pts,np.int32)

        tmp=[tmp.reshape((-1,1,2))]

        fpts=pts

        pts=[]

        #cv2.polylines(img,tmp,True,(255,255,255))

    #按下 r,g,b 填充多边形为相应的颜色

    elif code == ord('r'):

        FillPoly(fpts,(0,0,255))

    elif code == ord('y'):

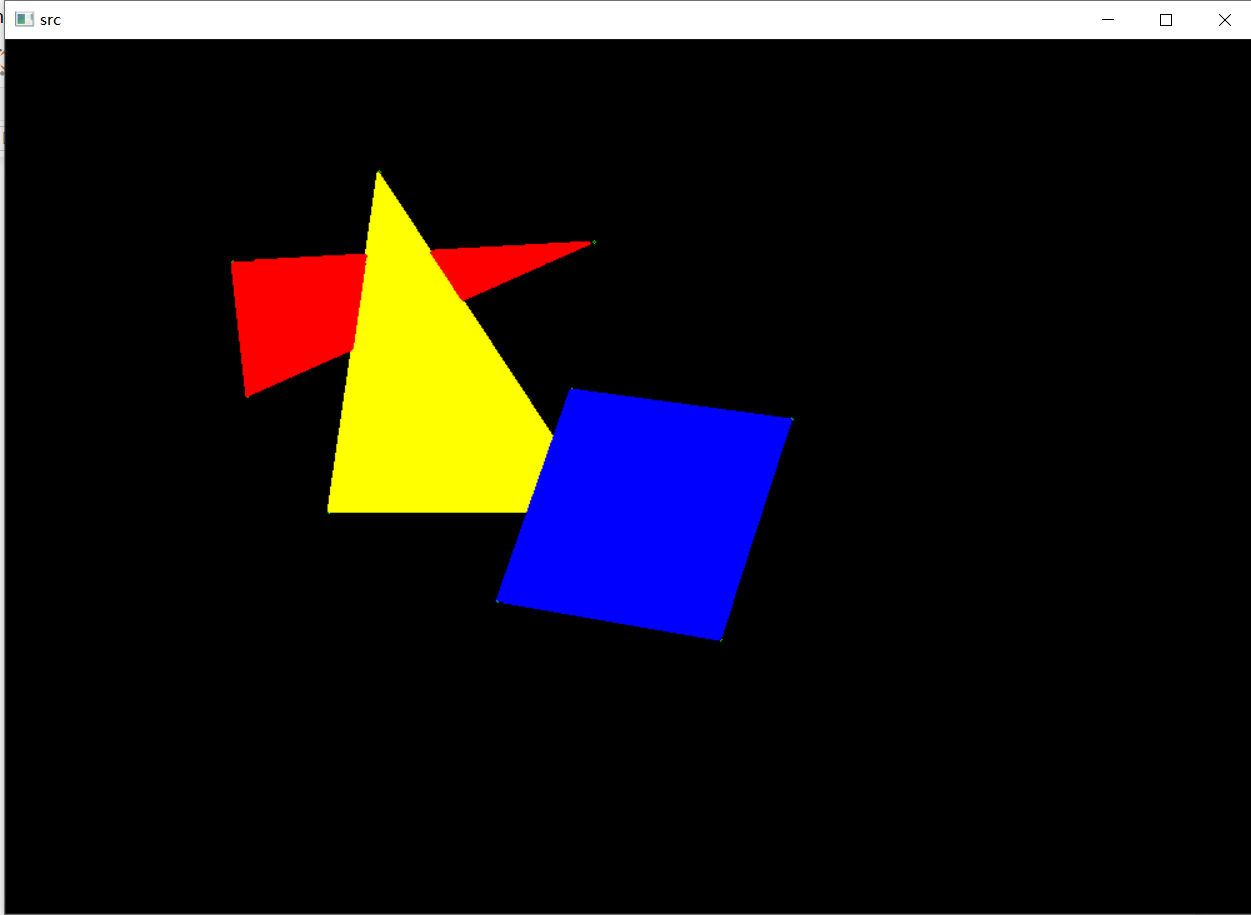
        FillPoly(fpts,(0,255,255))

    elif code == ord('b'):

        FillPoly(fpts,(255,0,0))

cv2.destroyAllWindows()

实验结果：



实验总结：

该程序实现了Zbuffer消隐算法。