**计算机图形学实验**

**姓 名： 许豪**

**学 号：20201060336**

**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

时间：2022年3月16日

地点：信息学院2202机房

1. 实验内容：

OpenGL glut的安装；

初步熟悉OpenGL编程及其编程框架；

熟悉OpenGL，通过示例程序生成直线段等图元；

1. 实验目的：

安装OPENGL，能编写代码运行；

参考课本代码，初步熟悉OpenGL编程及其编程框架；

1. 实验代码：

#include<Windows.h>

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

#include <stdlib.h>

static int slices = 16;

static int stacks = 16;

/\* GLUT callback Handlers \*/

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);// 指定清空颜色（背景色）为白色

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 200.0, 0.0, 150.0); //指定二维坐标系中被显示的区域

}

void lineSegment(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);// 清空显示窗口

glColor3f(0.0, 0.4, 0.2);// 指定前景色（当前绘制颜色）为蓝色

glBegin(GL\_LINES);

glVertex2i(180, 15);// 指定顶点

glVertex2i(10, 145);

glEnd();

glFlush();// 使绘制立即反映到屏幕上

}

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

glutInit(&argc, argv); // 初始 GLUT.

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);//设定显示模式

glutInitWindowPosition(50, 100); // 设定窗口位置

glutInitWindowSize(400, 300);// 设定窗口大小

glutCreateWindow("An Example OpenGL Program");// 用前面指定参数创建窗口，并定义窗口名称

init(); // 进行一些初始化工作

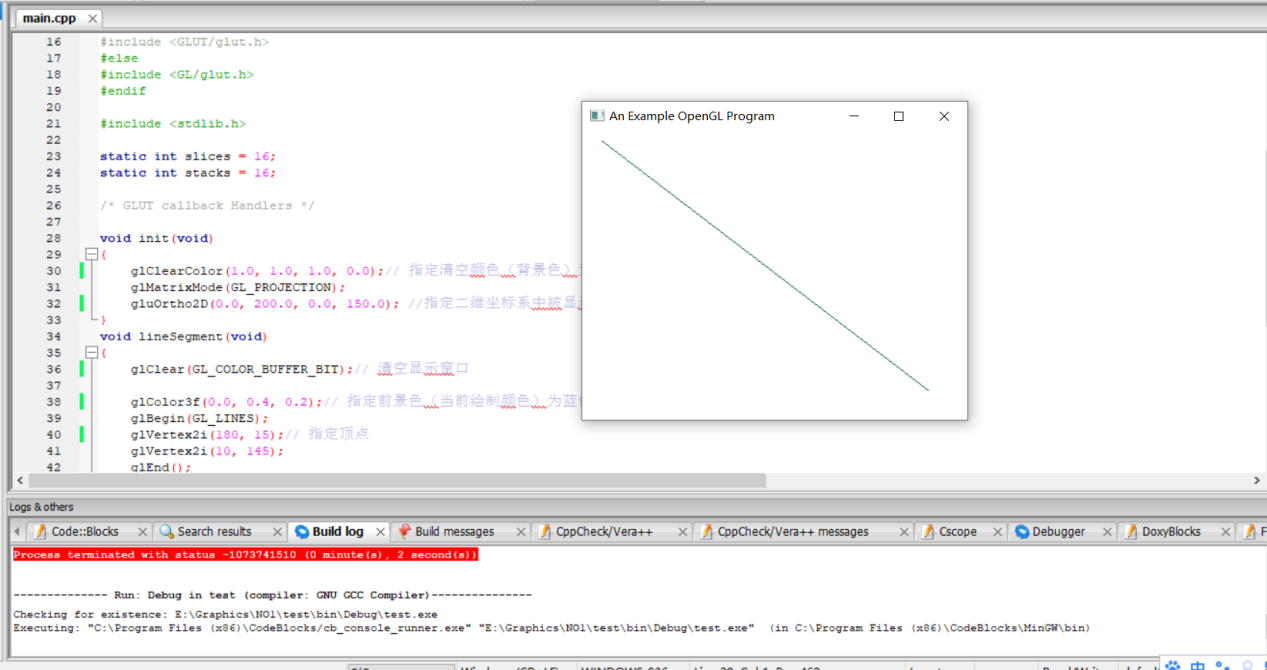
glutDisplayFunc(lineSegment);// 指定绘制的回调函数

glutMainLoop(); // 进入无穷循环，等待事件处理

return 0;

}

4、实验结果：



实验体会：

这个实验主要是根据书上的一个实例来体会opengl的操作，实现画线算法以及后面的各种图形图像算法的主要部分就是利用glut.h这个包中的函数来实现，所以我们需要了解其中函数的功能和使用方法。如main中第一步要glutInit(&argc, argv); // 初始GLUT，可以使用 glutCreateWindow("An Example OpenGL Program");在窗口创建时给一个标题，利用glutDisplayFunc(lineSegment);// 指定绘制的回调函数，在窗口中显示，然后利用glutMainLoop(); // 进入无穷循环，等待事件处理，将已创建的显示窗口连同其中的图形内容激活，这个函数语句必须是最后一条。到此main函数的基本形式已经确定下来了，后序只需更改回调函数。在visual studio运行opengl的时候，每次必定出现LNK1104 无法打开文件“freeglutd.lib”的错误，必须在每个程序包中安装：项目->管理NuGet程序包->浏览->搜索 NupenGL 安装第一个即可。

实验二 DDA直线生成算法

时间：2022年3月23日

地点：信息学院2202机房

1. 实验内容：

熟悉OpenGL，通过DDA、中点算法生成直线段

1. 实验目的：

安装OpenGL，能编写代码运行，参考课本代码。

初步熟悉OpenGL编程及其编程框架；

熟悉OpenGL，通过示例程序生成直线段等图元

1. 实验代码：

#include<Windows.h>\

#include<math.h>

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

#include <stdlib.h>

static int slices = 16;

static int stacks = 16;

void setPixel(GLint x,GLint y);

int round(float a);

/\* GLUT callback Handlers \*/

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

{

glClear(GL\_COLOR\_BUFFER\_BIT);// 清空显示窗口

glColor3f(0.0, 0.4, 0.2);// 指定前景色（当前绘制颜色）为蓝色

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

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

if(abs(dx)>abs(dy))

steps = abs(dy);

else

steps = abs(dx);

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));

}

}

void setPixel(GLint x,GLint y)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

int round(float a)

{

return int(a+0.5);

}

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);// 清空显示窗口

lineDDA(0, 0, 200000, 200000);

glFlush();// 使绘制立即反映到屏幕上

}

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);// 指定清空颜色（背景色）为白色

glColor3f(1.0, 0.0, 0.0); /\* draw in red \*/

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 200.0, 0.0, 150.0); //指定二维坐标系中被显示的区域

glMatrixMode(GL\_MODELVIEW);

}

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

{

glutInit(&argc, argv);// 初始 GLUT

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);//设定显示模式

glutInitWindowPosition(50, 100);// 设定窗口位置

glutInitWindowSize(400, 300);// 设定窗口大小

glutCreateWindow("DDA");// 用前面指定参数创建窗口，并定义窗口名称

gluOrtho2D(0.0, 500.0, 0.0, 500.0);//指定二维坐标系中被显示的区域

init(); // 进行一些初始化工作

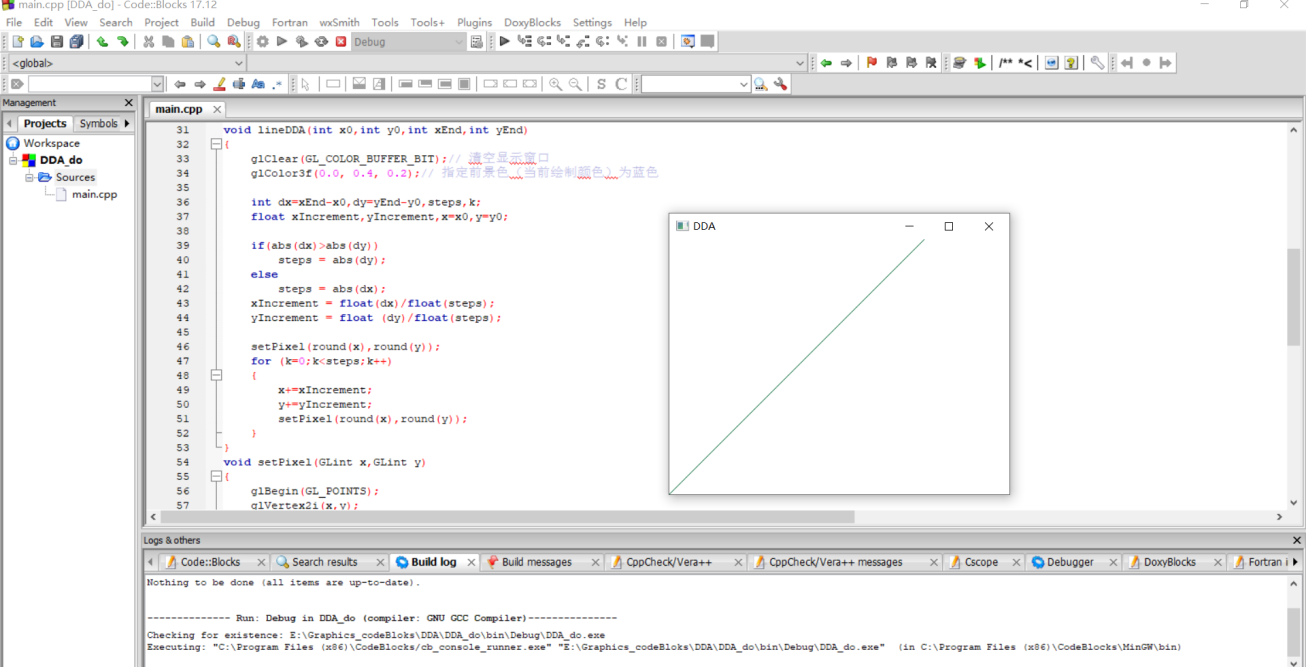
glutDisplayFunc(myDisplay);// 指定绘制的回调函数

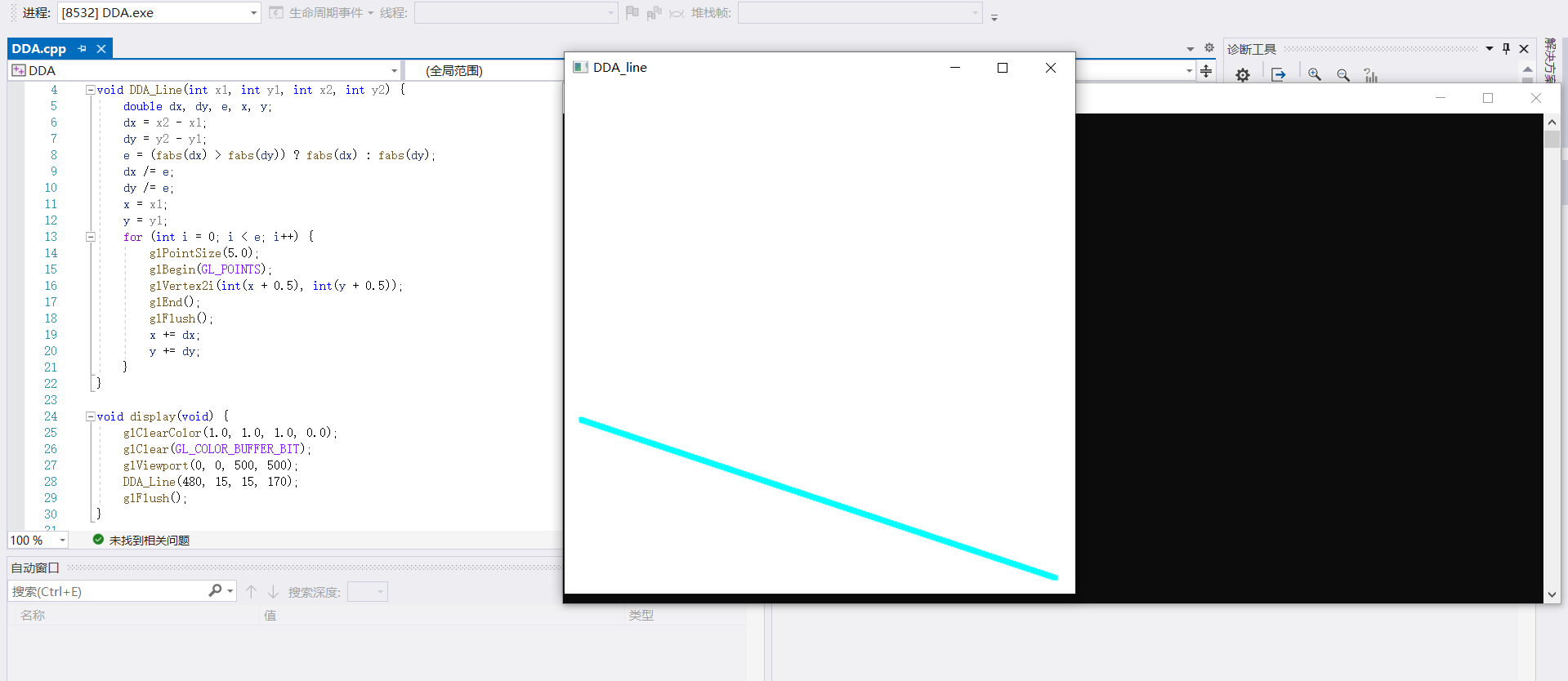
glutMainLoop();// 进入无穷循环，等待事件处理

return 0;

}

4、实验结果：





实验体会：

DDA算法原理是利用差分原理，实现思路如下:声明double类的变量dx，dy，e，x，y，其中dx和dy作为步进长度，e作为循环次数，x和y记录每次画点的坐标；dx=x2-x1，dy=y2-y1，计算斜率，如果斜率小于1则选择x为主步进方向，反之y为主步进方向。循环e次，每次循环画一个点，这里我选择x每次增大1，同时要将变化后的点从浮点类型转换为整型，同时更新下一次的x和y值。具体实现代码和之前的划线算法类似，只需根据原理写出DDA算法函数即可。

实验三 Bresenham算法生成直线段

时间：2022年3月30日

地点：信息学院2202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，了解Bresenham算法的原理。

3.实验代码：

#ifndef ABC\_H\_INCLUDED

#define ABC\_H\_INCLUDED

#endif // ABC\_H\_INCLUDED

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include<Windows.h>

#include <GL/glut.h>

#endif

#include <stdlib.h>

static int slices = 16;

static int stacks = 16;

void setPixel(GLint x,GLint y);

/\* GLUT callback Handlers \*/

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

{

int dx = abs(xEnd-x0),dy=abs(yEnd-y0);

int p=2\*dy-dx;

int twoDY=2\*dy;

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

int x,y;

if(x0>xEnd)

{

x=xEnd;

y=yEnd;

xEnd=x0;

}

else

{

x=x0;

y=y0;

}

setPixel(x,y);

while(x<xEnd)

{

x++;

if(p<0)

p+=twoDY;

else

{

y++;

p+=twoDyMinusDx;

}

setPixel(x,y);

}

}

void setPixel(GLint x,GLint y)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);// 指定清空颜色（背景色）为白色

glColor3f(1.0, 0.0, 0.0); /\* draw in red \*/

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 200.0, 0.0, 150.0); //指定二维坐标系中被显示的区域

glMatrixMode(GL\_MODELVIEW);

}

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);// 清空显示窗口

lineBres(0, 0, 100000, 100000);

glFlush();// 使绘制立即反映到屏幕上

}

#ifdef \_\_APPLE\_\_

#else

#include<Windows.h>

#include <GL/glut.h>

#endif

#include <stdlib.h>

#include "abc.h"

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

{

glutInit(&argc, argv);// 初始 GLUT

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);//设定显示模式

glutInitWindowPosition(50, 100);// 设定窗口位置

glutInitWindowSize(400, 300);// 设定窗口大小

glutCreateWindow("Bresenham");// 用前面指定参数创建窗口，并定义窗口名称

gluOrtho2D(0.0, 400, 0.0, 300);//指定二维坐标系中被显示的区域

init(); // 进行一些初始化工作

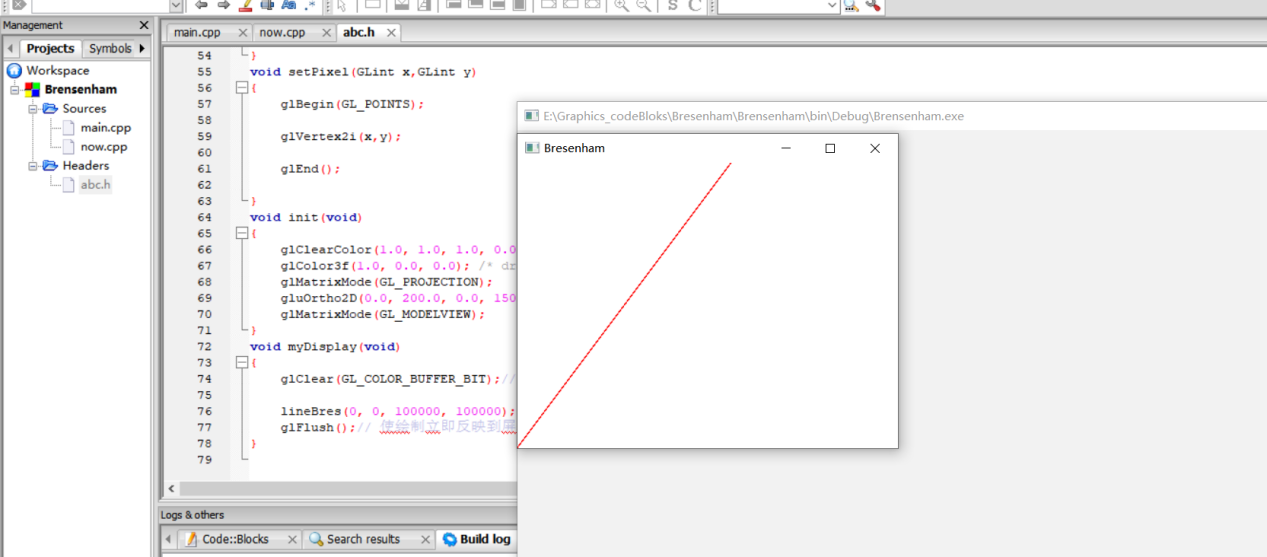
glutDisplayFunc(myDisplay);// 指定绘制的回调函数

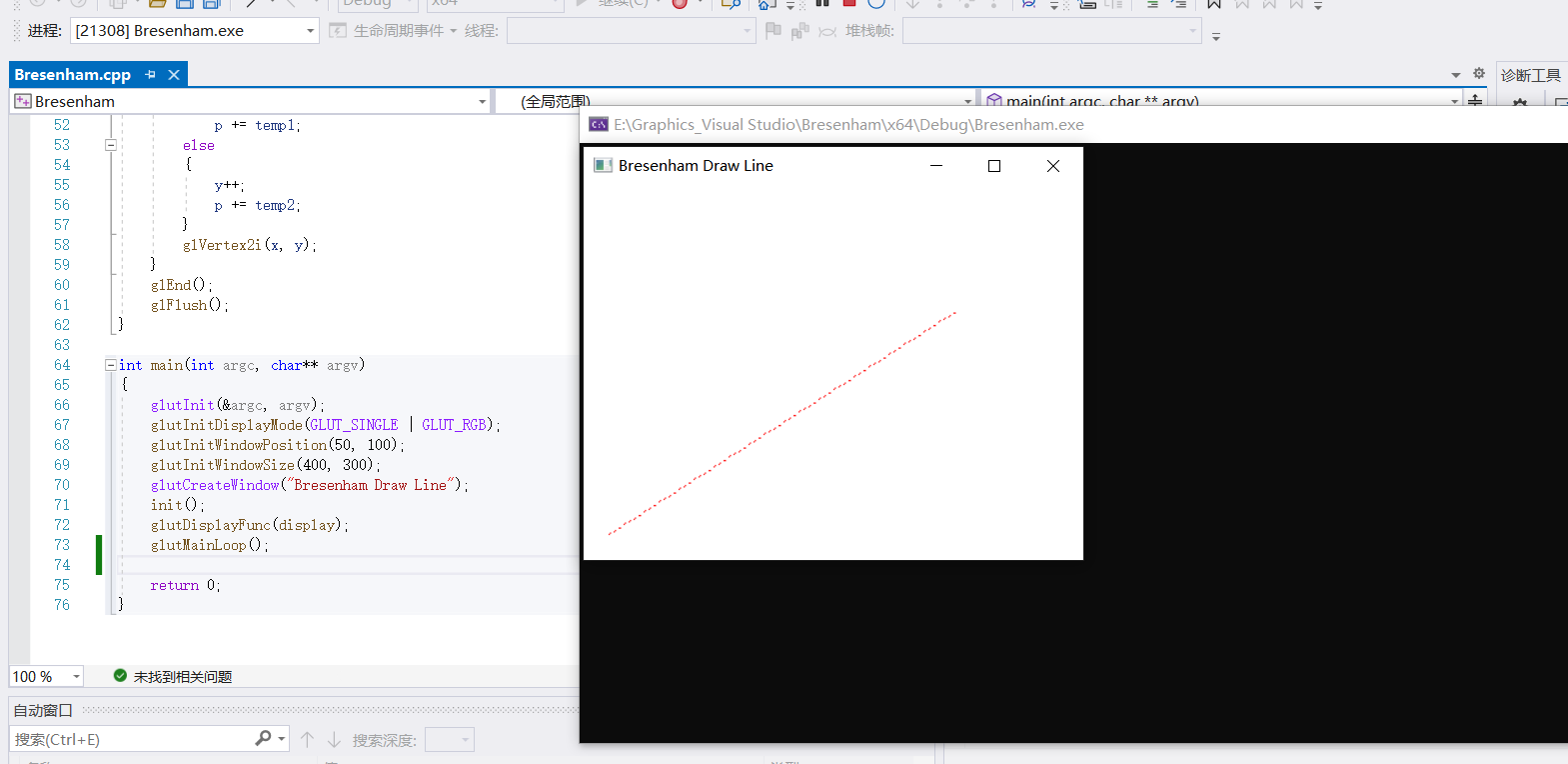
glutMainLoop();// 进入无穷循环，等待事件处理

return 0;

}

4.实验结果：





实验体会：

Bresenham画直线的算法主要解决的问题是如何判断下一点的位置：它也是采用递推步进的办法，令每次最大变化方向的坐标步进一个象素，同时另一个方向的坐标依据误差判别式的符号来决定是否也要步进一个象素。总得来说，此算法是利用光栅[网格](https://so.csdn.net/so/search?q=%E7%BD%91%E6%A0%BC&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_43481350/article/details/_blank)上的点到真实直线上的点的距离（成为误差项），来标定下一个点的位置初始值。利用d的判定函数来判断是否方向要递增。Bresenham算法对任意斜率的直线段具有通用性，对于斜率为整且大于1的直线段，只需要交换x和y之间的规则。对于负斜率，除了一个坐标递减而另一个坐标地政外，其余程序是类似的。

实验四 填充算法实验

时间：2022年4月6 日

地点：信息学院2202机房

1.实验内容：

教材P66，填充六边形。

使用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：

**填充六边形**：实现教材P66页代码即可。

#include<GL/glut.h>

#include<math.h>

#include<stdlib.h>

const double TWO\_PI = 6.2831853;

/\*Initial display-window size\*/

GLsizei winWidth = 400, winHeight = 400;

GLuint regHex;

class screenPt

{

private:

GLint x, y;

public:

/\*Default Constructor:initalizes coordinate position to(0,0).\*/

screenPt() { x = y = 0; }

void setGoords(GLint xGoord, GLint yGoord) { x = xGoord; y = yGoord; }

GLint getx()const { return x; }

GLint gety()const { return y; }

};

static void init(void)

{

screenPt hexVertex, circCtr;

GLdouble theta;

GLint k;

/\*Set circle center coordinates.\*/

circCtr.setGoords(winWidth / 2, winHeight / 2);

glClearColor(1.0, 1.0, 1.0, 0.0);//Display-window color=white.

/\*Set up a display list for a red regular hexagon.

\*Vertice for the hexagon are six equally spaced

\*points around the circumference of a circle.

\*/

regHex = glGenLists(1);//Get an identifier for the display list.

glNewList(regHex, GL\_COMPILE);

glColor3f(1.0, 0.0, 0.0);//Set fill color for hexagon to red.

glBegin(GL\_POLYGON);

for (k = 0; k < 6; k++)

{

theta = TWO\_PI \* k / 6.0;

hexVertex.setGoords(circCtr.getx() + 150 \* cos(theta),

circCtr.gety() + 150 \* sin(theta));

glVertex2i(hexVertex.getx(), hexVertex.gety());

}

glEnd();

glEndList();

}

void regHexagon(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glCallList(regHex);

glFlush();

}

void winReshapeFcn(GLint newWidth, GLint newHeight)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

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("Reshape-Function&Display-ListExample");

init();

glutDisplayFunc(regHexagon);

glutReshapeFunc(winReshapeFcn);

glutMainLoop();

}

**扫描线填充算法**：

#include "gl/glut.h"

#include "windows.h"

const int POINTNUM = 7; //多边形点数.

/\*\*\*\*\*\*定义结构体用于活性边表AET和新边表NET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct XET

{

float x;

float dx, ymax;

XET\* next;

}AET, NET;

/\*\*\*\*\*\*定义点结构体point\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct point

{

float x;

float y;

}

polypoint[POINTNUM] = { 250,50,550,150,550,400,250,250,100,350,100,100,120,30 };//多边形顶点

void PolyScan()

{

/\*\*\*\*\*\*计算最高点的y坐标(扫描到此结束)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int MaxY = 0;

int i;

for (i = 0; i < POINTNUM; i++)

if (polypoint[i].y > MaxY)

MaxY = polypoint[i].y;

/\*\*\*\*\*\*\*初始化AET表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

AET\* pAET = new AET;

pAET->next = NULL;

/\*\*\*\*\*\*初始化NET表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NET\* pNET[1024];

for (i = 0; i <= MaxY; i++)

{

pNET[i] = new NET;

pNET[i]->next = NULL;

}

glClear(GL\_COLOR\_BUFFER\_BIT); //赋值的窗口显示.

glColor3f(0.0, 0.0, 0.0); //设置直线的颜色红色

glBegin(GL\_POINTS);

/\*\*\*\*\*\*扫描并建立NET表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (i = 0; i <= MaxY; i++)

{

for (int j = 0; j < POINTNUM; j++)

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

{ //一个点跟前面的一个点形成一条线段，跟后面的点也形成线段

if (polypoint[(j - 1 + POINTNUM) % POINTNUM].y > polypoint[j].y)

{

NET\* p = new NET;

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

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

p->dx = (polypoint[(j - 1 + POINTNUM) % POINTNUM].x - polypoint[j].x) / (polypoint[(j - 1 + POINTNUM) % POINTNUM].y - polypoint[j].y);

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

pNET[i]->next = p;

}

if (polypoint[(j + 1 + POINTNUM) % POINTNUM].y > polypoint[j].y)

{

NET\* p = new NET;

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

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

p->dx = (polypoint[(j + 1 + POINTNUM) % POINTNUM].x - polypoint[j].x) / (polypoint[(j + 1 + POINTNUM) % POINTNUM].y - polypoint[j].y);

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

pNET[i]->next = p;

}

}

}

/\*\*\*\*\*\*建立并更新活性边表AET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (i = 0; i <= MaxY; i++)

{

//计算新的交点x,更新AET

NET\* p = pAET->next;

while (p)

{

p->x = p->x + p->dx;

p = p->next;

}

//更新后新AET先排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//断表排序,不再开辟空间

AET\* tq = pAET;

p = pAET->next;

tq->next = NULL;

while (p)

{

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

tq = tq->next;

NET\* s = p->next;

p->next = tq->next;

tq->next = p;

p = s;

tq = pAET;

}

//(改进算法)先从AET表中删除ymax==i的结点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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;

}

}

//将NET中的新点加入AET,并用插入法按X值递增排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

p = pNET[i]->next;

q = pAET;

while (p)

{

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

q = q->next;

NET\* s = p->next;

p->next = q->next;

q->next = p;

p = s;

q = pAET;

}

/\*\*\*\*\*\*配对填充颜色\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

p = pAET->next;

while (p && p->next)

{

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

glVertex2i(static\_cast<int>(j), i);

p = p->next->next;//考虑端点情况

}

}

glEnd();

glFlush();

}

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

//窗口的背景颜色设置为白色

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 600.0, 0.0, 450.0);

}

void main(int argc, char\* argv)

{

glutInit(&argc, &argv); //I初始化 GLUT.

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //设置显示模式：单个缓存和使用RGB模型

glutInitWindowPosition(50, 100); //设置窗口的顶部和左边位置

glutInitWindowSize(400, 300); //设置窗口的高度和宽度

glutCreateWindow("An Example OpenGL Program"); //创建显示窗口

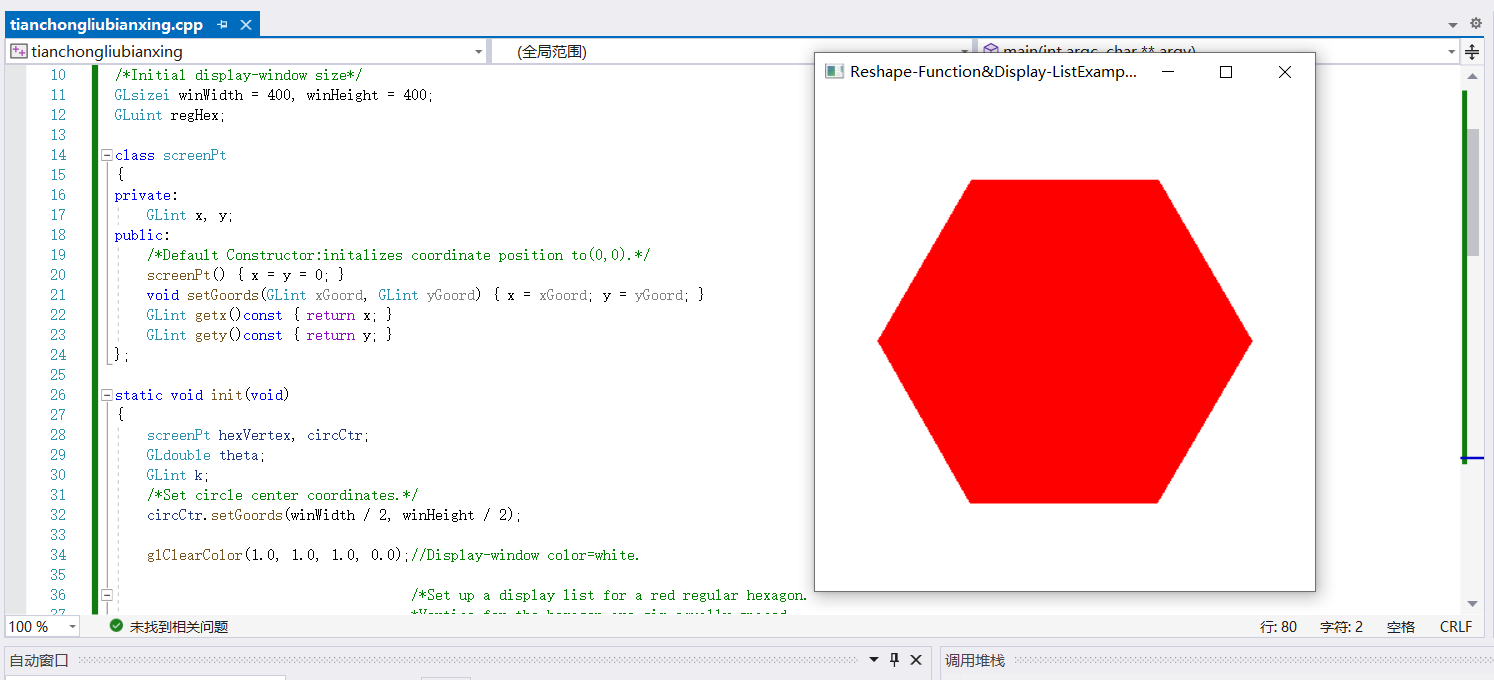
init(); //调用初始化过程

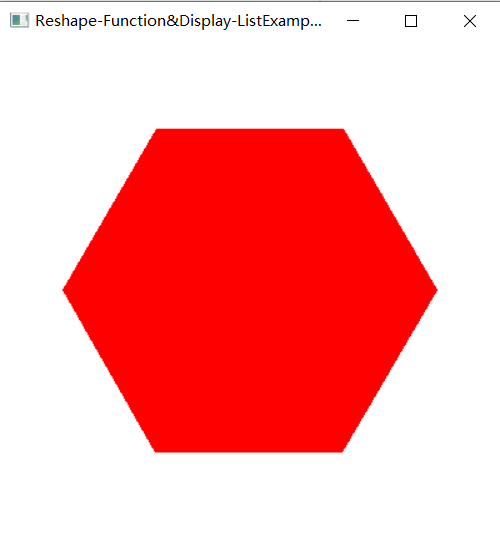
glutDisplayFunc(PolyScan); //图形的定义传递给我window.

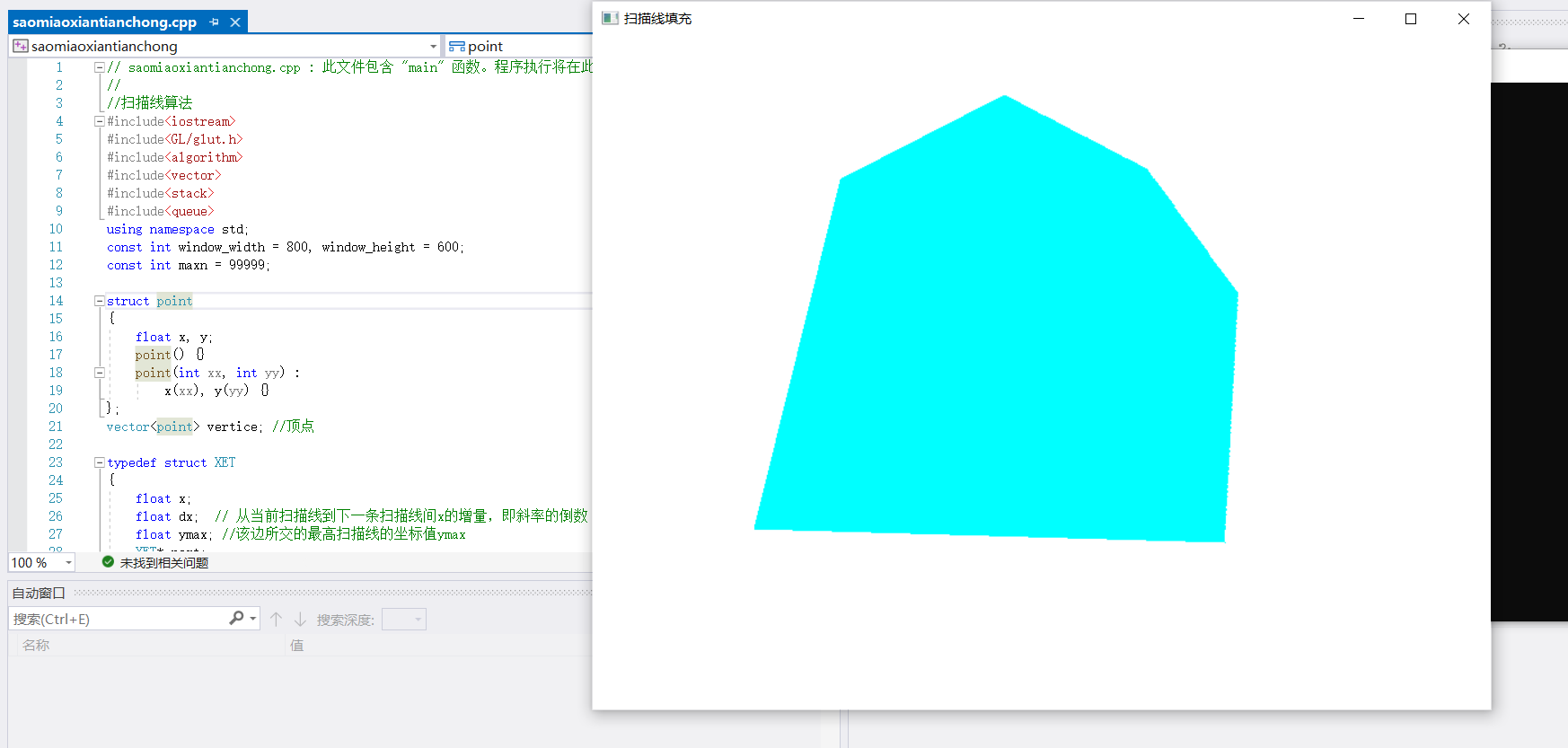
glutMainLoop(); //显示所有的图形并等待

}

4.实验结果：







实验体会:算法流程大概是

1.使用鼠标逐个输入多边形的定点，并且使用数组vertexes记录点的位置；

2.构建边表：

初始化边表数组的长度为绘图区域的高度；

由始至终遍历数组，分别将获取的顶点写入相应的边表位置：

若该点的相邻点在该点上方，则淘汰；

若该点的相邻点在该点的下方，则左下方点的边表元素应在右下方点所对应边表元素的右侧；

遍历完成，边表创建完成。

3.逐行构建活化边表并填充：

创建活化边表，初始化为空，仅包含首指针元素；

行遍历边表，并将全部得到的顶点写入活化边表；

活化边表中的数据项应随着扫描线的下移，在本层所有操作开始之前将dY递减、将x值逐层加上dX，并删除dY为0的节点；

当活化边表被清空时，遍历结束。

实验五 圆的扫描填充和种子点填充实验

时间：2022年4月13 日

地点：信息学院2202机房

1.实验内容：

圆的扫描转换。

种子点填充算法

1. 实验目的：

输入圆的半径，画出圆。

输入多边形，种子点位置，填充多边形。

3.实验代码：

圆

#include <iostream>

#include<GL/glut.h>

#include<stdio.h>

#include<math.h>

void setpoint(int x0, int y0, int x, int y)//同时绘制八个点(对称)

{

glColor3f(1.0f, 1.0f, 0.0);

glVertex2f((x0 + x), (y0 + y));

glVertex2f((x0 + y), (y0 + x));

glVertex2f((x0 + y), (y0 - x));

glVertex2f((x0 + x), (y0 - y));

glVertex2f((x0 - x), (y0 - y));

glVertex2f((x0 - y), (y0 - x));

glVertex2f((x0 - y), (y0 + x));

glVertex2f((x0 - x), (y0 + y));

}

void midpointcircle(int x0, int y0, int r)//中点画圆法

{

int x = 0;

int y = r;

int d = 1 - r; // 为了简化算法，摆脱浮点数，所以对d进行取整计算所以从1.25-r变到了1-r

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POLYGON);

setpoint(x0, y0, x, y);

while (x < y)

{

if (d < 0)

{

d += 2 \* x + 3;

}

else

{

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

y--;

}

x++;

setpoint(x0, y0, x, y);

}

glEnd();

glFlush();

}

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)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

int x0 = 100, y0 = 100, r = 60;

printf("请输入圆的半径：");

scanf\_s("%d", &r);

midpointcircle(x0, y0, r);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(200, 200);

glutInitWindowSize(400, 400);

glutCreateWindow("圆");

glutDisplayFunc(display);

glutReshapeFunc(ChangeSize);

glClearColor(1.0, 1.0, 1.0, 0.0); //设置窗口背景的颜色

glutMainLoop();

return 0;

}

种子点填充

#include<iostream>

#include<vector>

#include<GL/glut.h>

#include<queue>

#include<stack>

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;

float newcolor[3] = { 1, 0, 0 };

float boundarycolor[3] = { 0, 0, 1 };

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);

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(50, 50);

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);

glutMouseFunc(&mymouse);

//glutKeyboardFunc(&KeyBoards);

glutMainLoop();

return 0;

}

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

{

glBegin(GL\_POINTS);

glColor3fv(newcolor);

glVertex2f(x, y);

glEnd();

glFlush();

}

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

{

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

}

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;

int temx, temy;

temx = nowx;

temy = nowy;

float color[3];

glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

glReadPixels(temx + 1, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temx = temx + 1;

glReadPixels(temx + 1, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(temx, nowy);

}

else

{

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

}

}

temx = nowx;

glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temy = temy + 1;

glReadPixels(nowx, temy + 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(nowx, temy);

}

else

{

q.push(point(nowx, temy));

break;

}

}

temy = nowy;

glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temx = temx - 1;

glReadPixels(temx - 1, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(temx, nowy);

}

else

{

q.push(point(temx, nowy));

break;

}

}

temx = nowx;

glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temy = temy - 1;

glReadPixels(nowx, temy - 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(nowx, temy);

}

else

{

q.push(point(nowx, temy));

break;

}

}

draw\_a\_point(nowx, nowy);

}

}

}

void BoundaryFill4\_Stack(int x, int y)

{

stack < point> q;

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

while (!q.empty())

{

point now = q.top();

q.pop();

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

int temx, temy;

int flag = 0;

temx = nowx;

temy = nowy;

float color[3];

glReadPixels(nowx, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

glReadPixels(nowx + 1, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

flag = flag + 1;

}

glReadPixels(nowx, nowy - 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

flag = flag + 1;

}

glReadPixels(nowx - 1, nowy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

flag = flag + 1;

}

glReadPixels(nowx, nowy + 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

flag = flag + 1;

}

draw\_a\_point(nowx, nowy);

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

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

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

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

}

if (flag != 0)

{

glReadPixels(temx + 1, temy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temx = temx + 1;

glReadPixels(temx + 1, temy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(temx, temy);

}

else

{

q.push(point(temx, temy));

}

}

temx = nowx; temy = nowy;

glReadPixels(temx, temy - 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temy = temy - 1;

glReadPixels(temx, temy - 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(temx, temy);

}

else

{

q.push(point(temx, temy));

break;

}

}

temx = nowx; temy = nowy;

glReadPixels(temx - 1, temy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temx = temx - 1;

glReadPixels(temx - 1, temy, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(temx, temy);

}

else

{

q.push(point(temx, temy));

break;

}

}

temx = nowx; temy = nowy;

glReadPixels(temx, temy + 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

temy = temy + 1;

glReadPixels(temx, temy + 1, 1, 1, GL\_RGB, GL\_FLOAT, color);

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

{

draw\_a\_point(temx, temy);

}

else

{

q.push(point(temx, temy));

break;

}

}

draw\_a\_point(nowx, nowy);

}

}

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

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

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

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

}

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);

glLineWidth(4);

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)

{

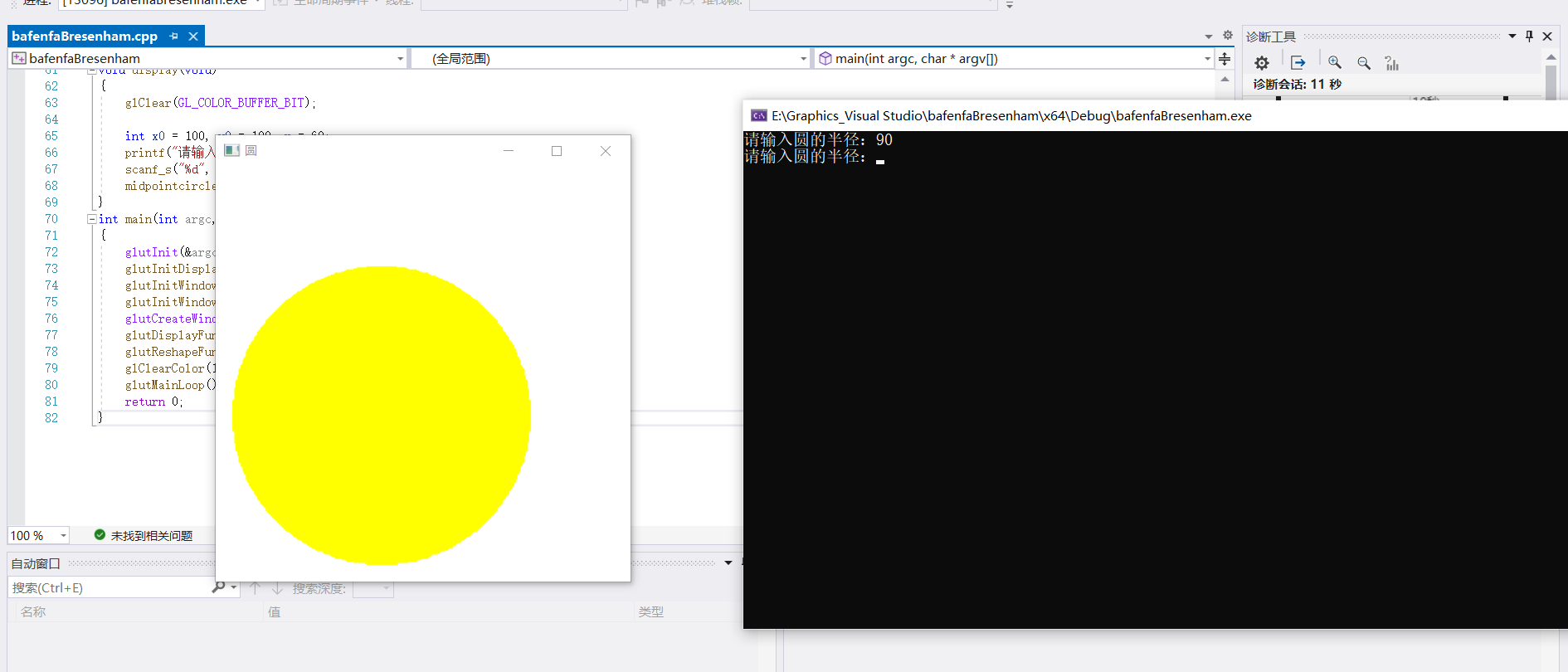
BoundaryFill4(x, window\_height - y);

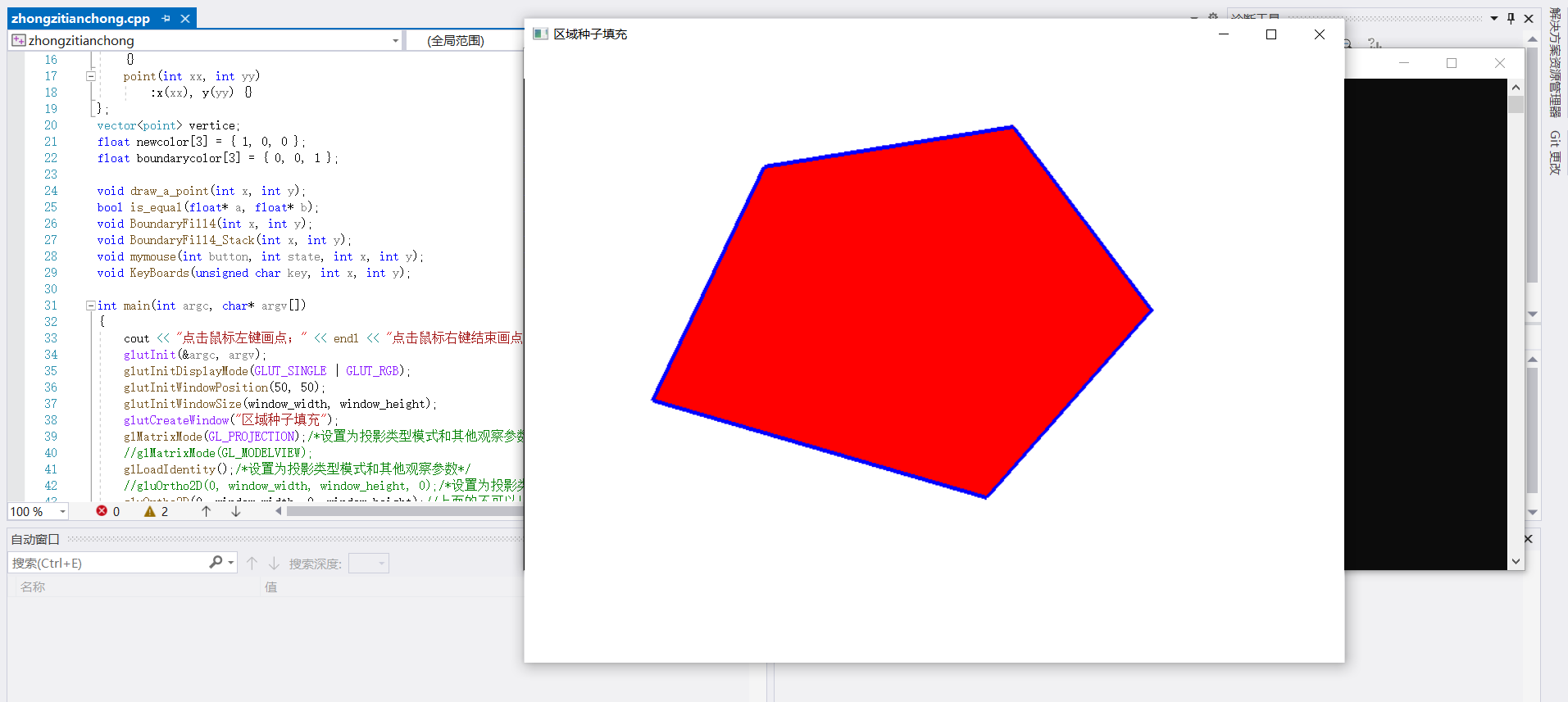
glFlush();

}

}

4.实验结果：





实验体会：

画圆改进：用d-0.25代替d（中点Bresenham算法）通过更换坐标系调整算法步骤：

1.输入圆的半径R。

2.计算初始值d=1-R、x=0、y=R。

3.绘制点(x,y)及其在八分圆中的另外七个对称点。

4.判断d的符号,若d≤0，则先将d更新为d+2x+3，再将(x,y)更新为(x+1,y)；否则先将d更新为d+2(x-y)+5，再将(x,y)更新为(x+1,y-1)。

5.当x<y时，重复步骤3和4。否则结束

种子填充算法是一种区域填充递归算法，分为：

1）四连通：

从区域内一点出发，分别从上，下，左，右移动判断该像素点与要填充的颜色是否相同。如果不同则变为填充颜色。如果该点为边界颜色则结束换另一个方向。

2)八连通：

从区域内一点出发，分别从上，下，左，右，左上，左下，右上，右下移动判断该像素点与要填充的颜色是否相同。如果不同则变为填充颜色。如果该点为边界颜色则结束换另一个方向。

由于八连通可能会出现点到达边界后从对角方向穿过去，所以每次获取点颜色时，需要同时获取该点附近的上下左右四点的颜色与边界判断。如果该点上下左右四点颜色与边界颜色相同则结束循环改变方向。

实验六 二维几何变换实验

时间：2022年4月20 日

地点：信息学院2202机房

1.实验内容：

教材P161，二维几何变换算法（平移、比例、旋转、对称）

2.实验目的：

验证二维几何变换，熟悉变换矩阵。

3.实验代码：

#include<gl/glut.h>

#include<iostream>

#include<cmath>

#include<vector>

#define PI 3.14159265358

//q,w,e,r分别为平移，绘制多边形，旋转，缩放

#define TRANSLATE 0

#define DRAWPOLYGON 1

#define ROTATE 2

#define SCALE 3

int tran\_x, tran\_y;

int \_xtmp, \_ytmp;//作为缩放变量用

int mode = DRAWPOLYGON;//默认为绘制模式

using namespace std;

const int winwidth = 800;

const int winheight = 640;

struct position {

double x;

double y;

};

typedef GLfloat Matrix3x3[3][3];

Matrix3x3 matComposite;//复合矩阵

vector<position> xy;

position tmp;

void DrawPolygon();

void dragmouse(int x, int y);

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

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

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

//设置为单位矩阵

void matrix3x3SetIdentity(Matrix3x3 mat) {

GLint row, col;

for (row = 0; row < 3; row++) {

for (col = 0; col < 3; col++)

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

}

}

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0);//设置绘制窗口颜色为白色

glClear(GL\_COLOR\_BUFFER\_BIT);//清除窗口显示内容

/\*设置为投影类型模式和其他观察参数\*/

glPointSize(3.0f);

glColor3f(1.0, 0.0, 0.0);//设置颜色为红

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, winwidth, winheight, 0);

matrix3x3SetIdentity(matComposite);

}

//矩阵相乘

void matrixpreMultiply(Matrix3x3 m1, Matrix3x3 m2) {

GLint row, col;

Matrix3x3 tmp;

for (row = 0; row < 3; row++) {

for (col = 0; col < 3; col++) {

tmp[row][col] = m1[row][0] \* m2[0][col] + m1[row][1] \*

m2[1][col] + m1[row][2] \* m2[2][col];

}

}

for (row = 0; row < 3; row++) {

for (col = 0; col < 3; col++) {

m2[row][col] = tmp[row][col];

}

}

}

//平移

void translate2D(GLfloat tx, GLfloat ty) {

Matrix3x3 matTransl;

matrix3x3SetIdentity(matTransl);//设置为单位矩阵

matTransl[0][2] = tx;

matTransl[1][2] = ty;

matrixpreMultiply(matTransl, matComposite);

}

//旋转

void rotate2D(int x, int y, float theta) {

theta = theta / 180 \* PI;

Matrix3x3 matRot;

matrix3x3SetIdentity(matRot);

matRot[0][0] = cos(theta);

matRot[0][1] = -sin(theta);

matRot[0][2] = x \* (1 - cos(theta)) + y \* sin(theta);

matRot[1][0] = sin(theta);

matRot[1][1] = cos(theta);

matRot[1][2] = y \* (1 - cos(theta)) - x \* sin(theta);

matrixpreMultiply(matRot, matComposite);

}

//缩放

void scale2D(GLfloat sx, GLfloat sy, int x, int y) {

Matrix3x3 matScale;

matrix3x3SetIdentity(matScale);

matScale[0][0] = sx;

matScale[0][2] = (1 - sx) \* x;

matScale[1][1] = sy;

matScale[1][2] = (1 - sy) \* y;

matrixpreMultiply(matScale, matComposite);

}

//复合矩阵

void transformVerts2D() {

GLfloat tmp;

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

tmp = matComposite[0][0] \* xy[i].x + matComposite[0][1] \* xy[i].y + matComposite[0][2];

xy[i].y = matComposite[1][0] \* xy[i].x + matComposite[1][1] \* xy[i].y + matComposite[1][2];

xy[i].x = tmp;

}

DrawPolygon();

matrix3x3SetIdentity(matComposite);

}

//绘制多边形

void DrawPolygon() {

// glEnable(GL\_POLYGON\_STIPPLE);

glPolygonMode(GL\_BACK, GL\_LINE);//设置反面为线性模式

glPolygonMode(GL\_FRONT, GL\_LINE);//设置正面为线性模式

glClear(GL\_COLOR\_BUFFER\_BIT);//清除窗口显示内容

glBegin(GL\_POLYGON);

for (unsigned int i = 0; i < xy.size(); i++) {

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

}

glEnd();

glFlush();

}

//鼠标拖动

void dragmouse(int x, int y) {

float ssx = 1, ssy = 1;

switch (mode)

{

//q,w,e,r

case TRANSLATE:

translate2D(x - tran\_x, y - tran\_y);

transformVerts2D();

tran\_x = x;

tran\_y = y;

break;

case DRAWPOLYGON:

break;

case ROTATE:

if (x <= \_xtmp && y >= \_ytmp)

rotate2D(tran\_x, tran\_y, -8);

else

rotate2D(tran\_x, tran\_y, 8);

transformVerts2D();

\_xtmp = x;

\_ytmp = y;

break;

case SCALE:

/\*不等比例缩放\*/

if (x > \_xtmp) {

ssx += 0.01f;

}

else if (x < \_xtmp && ssx>0) {

ssx -= 0.01f;

}

if (y < \_ytmp) {

ssy += 0.01f;

}

else if (y > \_ytmp && ssy > 0) {

ssy -= 0.01f;

}

/\*等比例缩放

if(x<=\_xtmp&&y>=\_ytmp){

ssx-=0.01f;

ssy-=0.01f;

}else{

ssx+=0.01f;

ssy+=0.01f;

}\*/

scale2D(ssx, ssy, tran\_x, tran\_y);

transformVerts2D();

\_xtmp = x;

\_ytmp = y;

break;

default:

break;

}

}

//鼠标监听

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

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

switch (mode)

{

//q,w,e,r

case TRANSLATE:

tran\_x = x;

tran\_y = y;

break;

case DRAWPOLYGON:

tmp.x = x;

tmp.y = y;

xy.push\_back(tmp);

DrawPolygon();

break;

case ROTATE:

tran\_x = x;

tran\_y = y;

\_xtmp = x;

\_ytmp = y;

break;

case SCALE:

tran\_x = x;

tran\_y = y;

break;

default:

break;

}

}

}

//键盘监听

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

//清空删除

if (key == 'a') {

glClear(GL\_COLOR\_BUFFER\_BIT);//清除窗口显示内容

glFlush();

xy.clear();

}

}

void myKeyboardUp(unsigned char key, int x, int y) {

switch (key)

{

case 'q':mode = TRANSLATE;

break;

case 'w':mode = DRAWPOLYGON;

break;

case 'e':mode = ROTATE;

break;

case 'r':mode = SCALE;

break;

default:

break;

}

}

void myDisplay() {

glFlush();

}

void mymenu(int id) {

if (id == 0)

mode = 0;

else if (id == 1)

mode = 1;

else if (id == 2)

mode = 2;

else if (id == 3)

mode = 3;

}

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

glutInit(&argc, argv);//初始化

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);//设置绘制模式

glutInitWindowPosition(500, 300);

glutInitWindowSize(winwidth, winheight);

glutCreateWindow("二维图形的变换");//创建窗口

int id = glutCreateMenu(mymenu);

glutAddMenuEntry("pingyi",0);

glutAddMenuEntry("绘制多边形",1);

glutAddMenuEntry("xuanzhuan",2);

glutAddMenuEntry("suofang",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

init();

glutDisplayFunc(myDisplay);

glutMouseFunc(mymouse);//鼠标监听回调函数

glutMotionFunc(dragmouse);//鼠标拖动

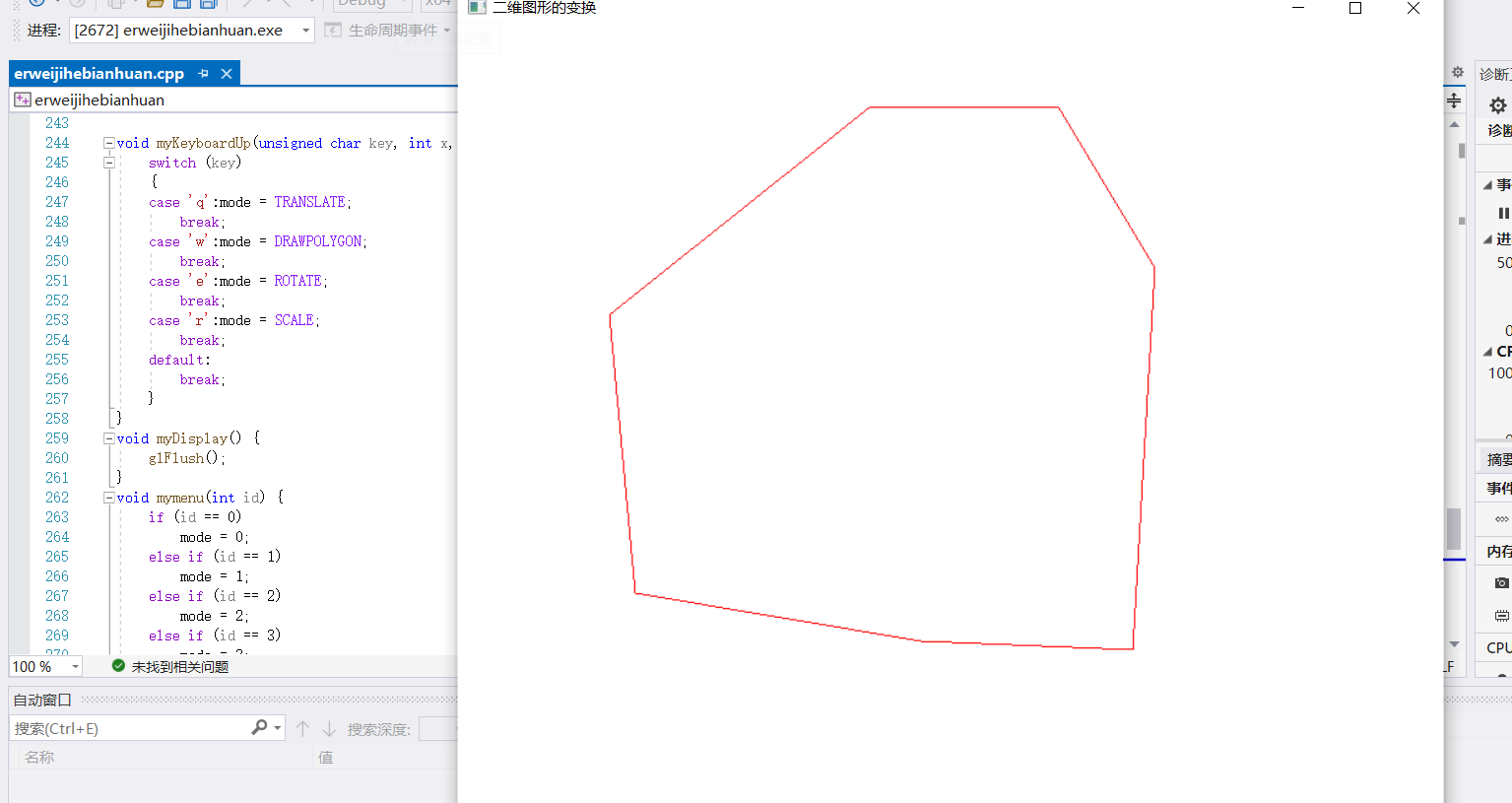
glutKeyboardFunc(myKeyboard);//键盘监听

glutKeyboardUpFunc(myKeyboardUp);//键盘弹起状态

glutMainLoop();

}

4.实验结果：



实验体会：图形的几何变换是指对图形的几何信息经过平移、比例、旋转、反射和错切等变换后产生新的图形，是图形在方向、尺寸和形状方面的变换。以上变换可用统一的变换矩阵形式来实现，只需利用特定的变换矩阵即可。而复合变换可以通过上面基础变换得到。几何变换均可表示成P’=P\*T的形式（规范化齐次矩阵）。

实验七 GLUT鼠标函数实验和反走样实验

时间：2022年4月27 日

地点：信息学院2202机房

1.实验内容：

教材P458，GLUT鼠标函数

使用opengl，实现任一反走样技术。

2.实验目的：

调用鼠标函数完成相应功能，2~3个程序。

3.实验代码：

鼠标实验：前面已经实现（代码不再展示了）

反走样实验：

#include <iostream>

#include <GL/glut.h>

#define NO 0

#define YES 1

int Drawing;

void Initialization()

{

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

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

glEnable(GL\_POINT\_SMOOTH); //启用点反走样

glHint(GL\_POINT\_SMOOTH\_HINT, GL\_NICEST);

glEnable(GL\_LINE\_SMOOTH); //启用直线反走样，初始化函数中所调用的那3条命令才真正起作用

glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST);

Drawing = NO;

}

void OnDisplay(void)

{

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

if (Drawing == YES) //启用混合状态，初始化函数中所调用的那3条命令才真正起作用

{

glEnable(GL\_BLEND);

}

else //关闭混合状态

{

glDisable(GL\_BLEND);

}

glColor4f(1.0f, 0.0f, 0.0f, 1.0f); //当前绘图色为绿色

glLineWidth(10);

glBegin(GL\_LINE\_STRIP);

{

glVertex2f(-3.0f, -0.5f);

glVertex2f(0.0f, 0.5f);

glVertex2f(3.0f, -0.5f);

}

glEnd();

glPointSize(10);

glBegin(GL\_POINTS);

{

glVertex2f(0.0f, 1.0f);

}

glEnd();

glutSwapBuffers();

}

void OnReShape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); //将当前矩阵指定为投影模式

glLoadIdentity();

if (h != 0)

{

GLfloat aspect = (float)w / (float)h;

if (w < h)

{

gluOrtho2D(-3, 3, -3 \* aspect, 3 \* aspect);

}

else

{

gluOrtho2D(-3 / aspect, 3 / aspect, -3, 3);

}

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void CreateMen(void)

{

glutAddMenuEntry("zhengchangxianshi",NO);

glutAddMenuEntry("fanzouyang", YES);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void OnMenu(int value)

{

Drawing = value;

glutPostRedisplay();

}

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

{

glutInit(&argc, argv);

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

glutInitWindowSize(600, 480);

glutCreateWindow("OpenGL反走样");

glutReshapeFunc(OnReShape);

glutDisplayFunc(OnDisplay);

glutCreateMenu(OnMenu);

Initialization();

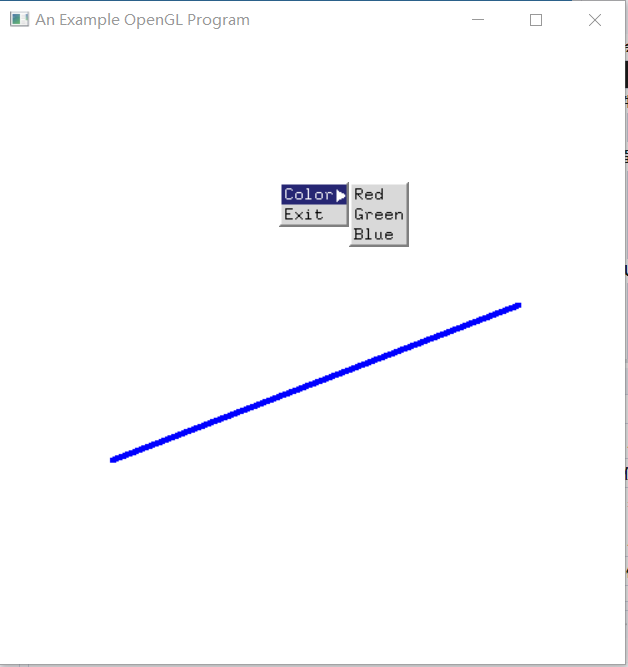
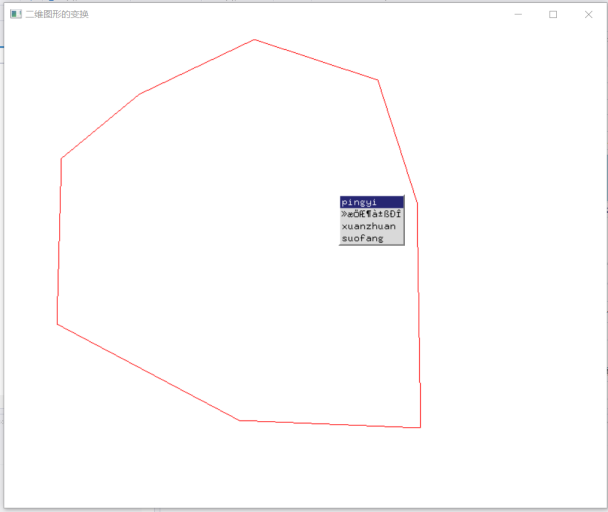
CreateMen();

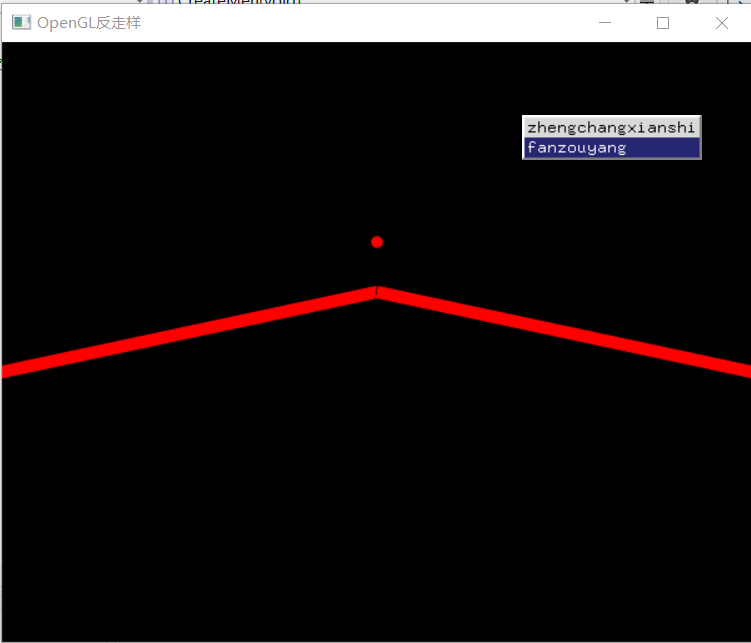
glutMainLoop();

return 0;

}

4.实验结果：





实验体会：在OpenGL中处理鼠标事件非常的方便，GLUT已经注册好了函数，只要调用方法即可。使用函数glutMouseFunc,就可以注册函数，这样当发生鼠标事件时就会自动调用我们的方法。GLUT提供鼠标动作检测能力。有两种GLUT处理的motion：active motion和passive motion。Active motion是指鼠标移动并且有一个鼠标键被按下。Passive motion是指当鼠标移动时，并有没鼠标键按下。如果一个程序正在追踪鼠标，那么鼠标移动期间，每一帧将产生一个结果。

绘制的物体边缘部分出现锯齿的现象称之为走样。反走样是减轻这种现象的方法。有前滤波和后滤波。

实验八 二维图像裁剪实验

时间：2022年5月3 日

地点：信息学院2202机房

1.实验内容：

使用opengl，用Cohen-Sutherland线段裁剪算法对直线段进行裁剪

2.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

3.实验代码：

**Cohen-Sutherland线段裁剪算法：**

#include <Windows.h>

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

#include <stdlib.h>

#include <iostream>

#include <stdlib.h>

#include <math.h>

#include<stdio.h>

//全局变量

int a, b, a1, b1, pp0, pq0, pp1, pq1;

//画线

void setPixel(GLint x, GLint y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

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 x0, int y0, int xEnd, int yEnd) {

int dx = xEnd - x0;

int dy = yEnd - y0;

int steps, k;

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

if (abs(dx) > abs(dy))

steps = abs(dx);

else

steps = abs(dy);

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

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

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

{

x += xIncrement;

y += yIncrement;

setPixel(x, y);

}

}

//裁剪

class wcPt2D {

public:

GLfloat x, y;

};

inline GLint round(const GLfloat a) {

return GLint(a + 0.5);

}

const GLint winLeftBitCode = 0x1;

const GLint winRightBitCode = 0x2;

const GLint winBottomBitCode = 0x4;

const GLint winTopBitCode = 0x8;

inline GLint inside(GLint code) {

return GLint(!code);

}

inline GLint reject(GLint code1, GLint code2) {

return GLint(code1 & code2);

}

inline GLint accept(GLint code1, GLint code2) {

return GLint(!(code1 | code2));

}

GLubyte encode(wcPt2D pt, wcPt2D winMin, wcPt2D winMax) {

GLubyte code = 0x00;

if (pt.x < winMin.x)

code = code | winLeftBitCode;

if (pt.x > winMax.x)

code = code | winRightBitCode;

if (pt.y < winMin.y)

code = code | winBottomBitCode;

if (pt.y > winMax.y)

code = code | winTopBitCode;

return(code);

}

void swapPts(wcPt2D\* p1, wcPt2D\* p2) {

wcPt2D tmp;

tmp = \*p1;

\*p1 = \*p2;

\*p2 = tmp;

}

void swapCodes(GLubyte\* c1, GLubyte\* c2) {

GLubyte tmp;

tmp = \*c1;

\*c1 = \*c2;

\*c2 = tmp;

}

void lineClipCohSuth(wcPt2D winMin, wcPt2D winMax, wcPt2D p1, wcPt2D p2) {

GLubyte code1, code2;

GLint done = false, plotLine = false;

GLfloat m;

int x0 = 0; int y0 = 0; int x1 = 0; int y1 = 0;

while (!done) {

code1 = encode(p1, winMin, winMax);

code2 = encode(p2, winMin, winMax);

if (accept(code1, code2)) {

done = true;

plotLine = true;

}//简取

else if (reject(code1, code2))//简弃

done = true;

else {

if (inside(code1)) {

swapPts(&p1, &p2);

swapCodes(&code1, &code2);

}

if (p2.x != p1.x)

m = (p2.y - p1.y) / (p2.x - p1.x);//计算k

if (code1 & winLeftBitCode) {

p1.y += (winMin.x - p1.x) \* m;

p1.x = winMax.x;

}

else if (code1 & winBottomBitCode) {

if (p2.x != p1.x)

p1.x += (winMin.y - p1.y) / m;

p1.y = winMin.y;

}

else if (code1 & winTopBitCode) {

if (p2.x != p1.x)

p1.x += (winMax.y - p1.y) / m;

p1.y = winMax.y;

}

}//else

}//while

if (plotLine) {

LineDDA(round(p1.x), round(p1.y), round(p2.x), round(p2.y));

}

}

void cutwindow(int xmin, int ymin, int xmax, int ymax) {

LineDDA(xmin, ymin, xmin, ymax);

LineDDA(xmin, ymin, xmax, ymin);

LineDDA(xmin, ymax, xmax, ymax);

LineDDA(xmax, ymin, xmax, ymax);

}

void display() {//DDA演示

printf("DDA演示\n");

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

wcPt2D min;

min.x = a;

min.y = b;

wcPt2D max;

max.x = a1;

max.y = b1;

wcPt2D p1;

p1.x = pp0;

p1.y = pq0;

wcPt2D p2;

p2.x = pp1;

p2.y = pq1;

cutwindow(min.x, min.y, max.x, max.y);//绘制窗口

lineClipCohSuth(min, max, p1, p2);

glEnd();

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(500, 300);

glutInitWindowSize(400, 300);

glutCreateWindow("直线裁剪");

init();

printf("请输入裁剪窗口左下角坐标\n");

scanf("%d %d", &a, &b);

printf("请输入裁剪窗口右上角坐标\n");

scanf("%d %d", &a1, &b1);

printf("请输入要裁剪的直线第一点坐标\n");

scanf("%d %d", &pp0, &pq0);

printf("请输入要裁剪的直线第二点坐标\n");

scanf("%d %d", &pp1, &pq1);

glutDisplayFunc(display);

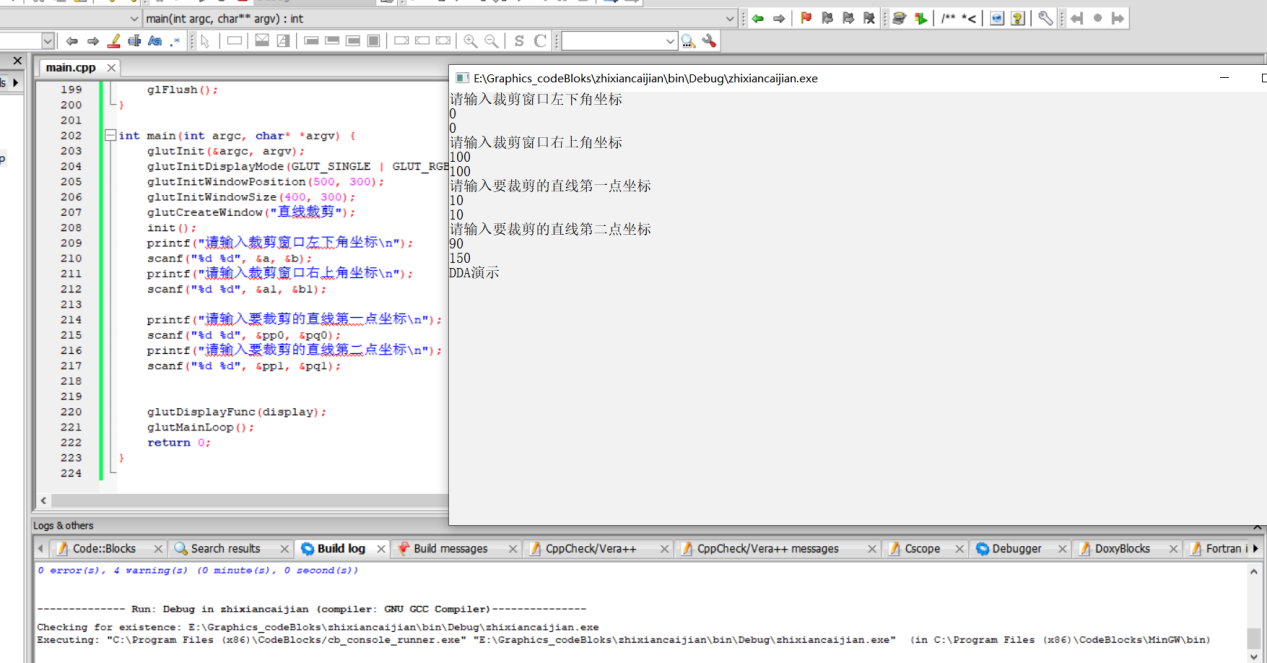
glutMainLoop();

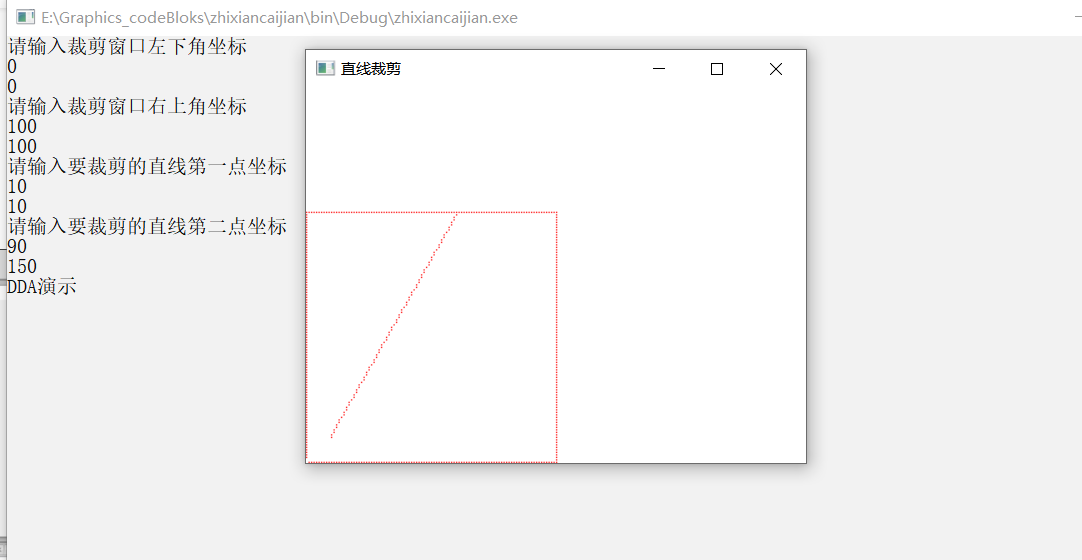
return 0;

}

4.实验结果：

**Cohen-Sutherland线段裁剪算法：**





实验体会：裁剪，计算机内部存储的图形往往比较大，而屏幕显示的只是图形的一部分，因此需要确定图形哪些部分落在显示区之内，哪些落在显示区之外，这个选择的过程就称为裁剪。要裁剪一条直线段，首先要判断此直线的位置：它是否完全落在裁剪窗口内？它是否完全在窗口外？如果不满足以上两个条件，则计算它与一个或多个裁剪边界的交点。裁剪一条线段时，先求出端点p1和p2的编码code1和code2，然后进行二进制“或”运算和“与”运算。算法的基本流程，1.检查线段 P1P2 是否为完全可见，或完全不可见，对于这两种情况或完全取之，或完全弃之，否则转 2 2.找到 P1P2 在窗口外的一个端点 P1 (或 P2）3.用窗口的边与 P1P2 的交点取代端点 P1 (或 P2）4.P1P2 线段是否完全可见，若是，则结束，否则转 2 继续执行。（反复判断简取之和简弃之）

实验九 三维图形几何变换实验

时间：2022年5月3 日

地点：信息学院2202机房

1.实验内容：

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。

2.实验目的：

调用函数完成三维图形几何变换。

3.实验代码：

// sanweijihebianhuan.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <Windows.h>

#include <GL/glut.h>

#include <math.h>

#include<Windows.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

GLsizei winWidth = 600, winHeight = 600;

GLuint regHex;//显示表标识

/\*设置世界坐标范围\*/

GLfloat xwcMin = 0.0, xwcMax = 225.0;

GLfloat ywcMin = 0.0, ywcMax = 225.0;

class wcPt3d

{

public:

GLfloat x, y, z;

};

typedef GLfloat Matrix4x4[4][4];

Matrix4x4 matComposite;//3D变换复合矩阵

const GLdouble pi = 3.14159;

static void init(void)

{

//初始化函数，并加入表

glClearColor(1.0, 1.0, 1.0, 0.0);//设置为白色背景

}

void Matrix4x4SetIdentity(Matrix4x4 matIdent4x4)

{

//生成一个单位矩阵

GLint row, col;

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

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

{

matIdent4x4[row][col] = (row == col);

}

}

void matrix4x4Premultiply(Matrix4x4 m1, Matrix4x4 m2)

{

//矩阵的乘法，结果储存在M2中

GLint row, col;

Matrix4x4 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)

{

/\*通过偏移量tx,ty产生平移矩阵matTransl之后，再通过矩阵的乘法，实现对象的平移\*/

Matrix4x4 matTransl;//平移矩阵

Matrix4x4SetIdentity(matTransl);

matTransl[0][3] = tx;

matTransl[1][3] = ty;

matTransl[2][3] = ty;

/\*矩阵乘法，平移\*/

matrix4x4Premultiply(matTransl, matComposite);

}

void rotate3d(wcPt3d p1, wcPt3d p2, GLfloat theta)

{

Matrix4x4 matRot;//旋转矩阵

GLfloat axisVecLength = sqrt((p2.x - p1.x) \* (p2.x - p1.x) + (p2.y - p1.y) \* (p2.x - p1.x) + (p2.z - p1.z) \* (p2.z - p1.z));

GLfloat cosTheta = cos(theta);

GLfloat oneC = 1 - cosTheta;

GLfloat sinTheta = sin(theta);

GLfloat ux = (p2.x - p1.x) / axisVecLength;

GLfloat uy = (p2.y - p1.y) / axisVecLength;

GLfloat uz = (p2.z - p1.z) / axisVecLength;

/\*将旋转向量平移到原点\*/

translate3D(-p1.x, -p1.y, -p1.z);

Matrix4x4SetIdentity(matRot);

matRot[0][0] = ux \* ux \* oneC + cosTheta;

matRot[0][1] = ux \* uy \* oneC - uz \* sinTheta;

matRot[0][2] = ux \* uz \* oneC + uy \* sinTheta;

matRot[1][0] = uy \* uz \* oneC + uz \* sinTheta;

matRot[1][1] = uy \* uy \* oneC + cosTheta;

matRot[1][2] = uy \* uz \* oneC + uy \* sinTheta;

matRot[2][0] = uz \* ux \* oneC - uy \* sinTheta;

matRot[2][1] = uz \* uy \* oneC + ux \* sinTheta;

matRot[2][2] = uz \* uz \* oneC + cosTheta;

matrix4x4Premultiply(matRot, matComposite);

translate3D(p1.x, p1.y, p1.z);//平移回原来的位置

}

void translate3d(GLint nverts, wcPt3d\* verts)

{

//对象顶点与复合变换矩阵的乘法，得到坐标变换后的点

GLint k;

GLfloat temp;

for (k = 0; k < nverts; ++k)

{

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

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

verts[k].x = temp;

}

}

void triangle(wcPt3d\* verts)

{

//输出一个空间三角形

GLint k;

glBegin(GL\_TRIANGLES);

cout << "当前顶点坐标为：";

for (k = 0; k < 3; ++k)

{

glVertex3f(verts[k].x, verts[k].y, verts[k].z);

cout << verts[k].x << "," << verts[k].y << "," << verts[k].z << endl;

}

glEnd();

}

void displayFcn()

{

GLint nverts = 3;

wcPt3d verts[3] = { { 50.0, 25.0,0 }, { 150.0, 25.0 ,50}, { 100.0, 100.0,100 } };//定义一个三角形

GLdouble theta = pi / 4.0;//旋转45°

wcPt3d p1, p2;

p1.x = 0; p1.y = 0; p1.z = 0;

p2.x = 0; p2.y = 0; p2.z = 1;//定义旋转轴为(0,0,1)

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

triangle(verts);//蓝色三角形为原始三角形

Matrix4x4SetIdentity(matComposite);

glColor3f(1.0, 0.0, 0.0);//红色为移动后的三角形

rotate3d(p1, p2, theta);//旋转

translate3d(nverts, verts);

glColor3f(1.0, 0.0, 0.0);//红色为移动后的三角形

triangle(verts);

glFlush();

}

void winReshapeFcn(int newWidth, int newHeight)

{

//窗口重定形函数

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();//将当前的用户坐标系的原点移到了屏幕中心：类似于一个复位操作

glOrtho(xwcMin, xwcMax, ywcMin, ywcMax, -55, 0);

glClear(GL\_COLOR\_BUFFER\_BIT);

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(50, 50);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Example");

init();

glutDisplayFunc(displayFcn);

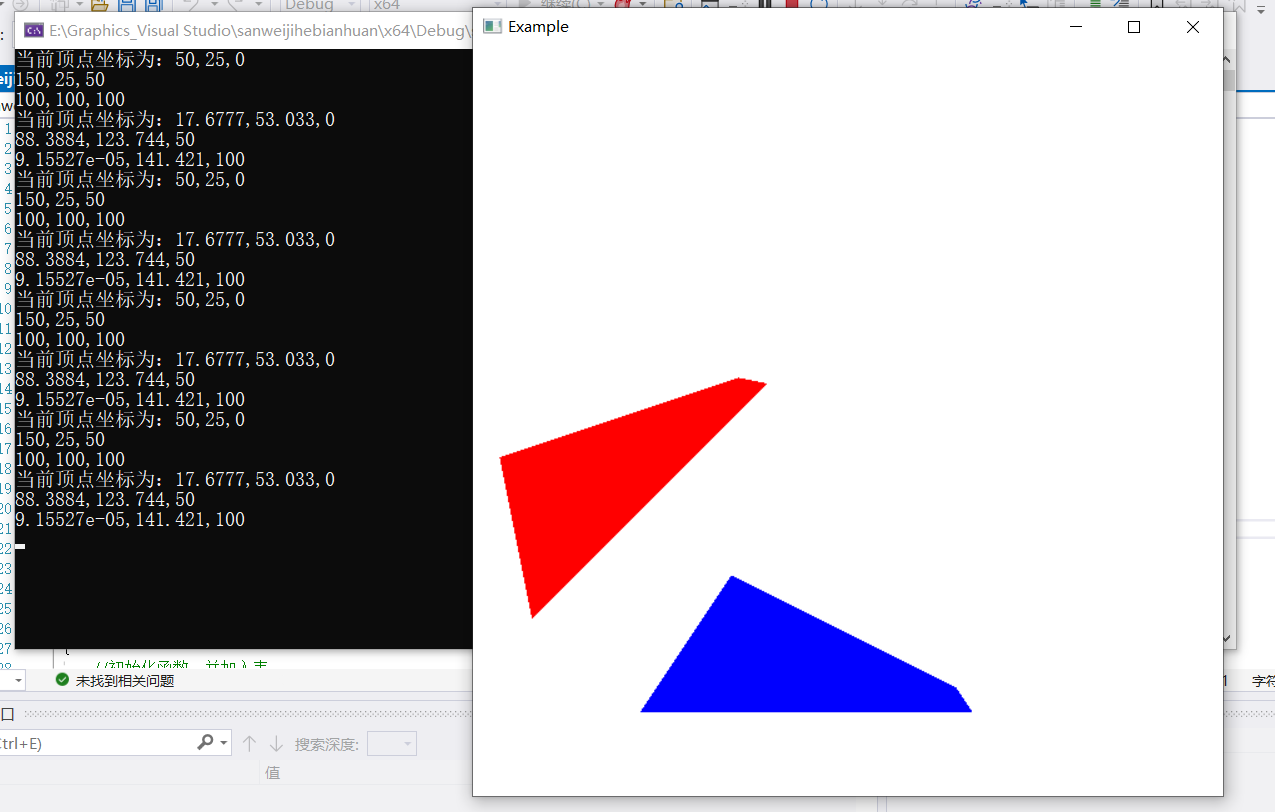
glutReshapeFunc(winReshapeFcn);

glutMainLoop();

return 0;

}

4.实验结果：



实验体会：OpenGL中的三维变换是通过矩阵变换来实现的。无论是移动，缩放，或者是旋转都是在一个矩阵的基础上乘上另一个矩阵来实现的。三维复合变换是指图形作一次以上的变换，变换结果是每次变换矩阵相乘。类似地，针对任意方向轴的变换可用五个步骤来完成：(1)使任意方向轴的起点与坐标原点重合，此时进行平移变换。(2)使方向轴与某一坐标轴重合，此时需进行旋转变换，且旋转变换可能不止一次。(3)针对该坐标轴完成变换。(4)用逆旋转变换使方向轴回到其原始方向。(5)用逆平移变换使方向轴回到其原始位置。这其实也是变换的基本流程。

实验十 着色及建模实验

时间：2022年5月17 日

地点：信息学院2202机房

1.实验内容：

使用opengl，片元着色器着色，P523

使用opengl，教材P541，颜色编码建模显示。

2.实验目的：

验证片元着色器算法，获得着色结果。

调用函数完成颜色编码建模实验。

3.实验代码：

// yansebianmajianmo.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include<Windows.h>

#include <iostream>

#include<GL/glut.h>

//GLsiei intialsiseot the dsplay windows

GLsizei winWidth=500,winHeight=500;

/\*Set 1imits for the rectangular area in complex plane.\*/

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){

/\*Set display -window colorto white.8\*/

glClearColor(1.0, 1.0, 1.0, 0.0);

}

void plotPoint(complexNum z) {

glBegin(GL\_POINTS);

glVertex2f(z.x,z.y);

glEnd();

}

/\*Calculate the square ofa complex number. \*/

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;

/\* Quit whenz\*z > 4 \*/

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)

/\*Set point color to black .\*/

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

else if (iterCount > (maxIter / 8)) {

/\* Set point color to orange.\*/

ptColor.r = 1.0;

ptColor.g = 0.5;

ptColor.b = 0.0;

}

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

/\*set point color to red.\*/

ptColor.r = 1.0;

ptColor.g = ptColor.b = 0.0;

}

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

/\*Set point color to dark blue. \*/

ptColor.b = 0.5;

ptColor.r = ptColor.g = 0.0;

}

else if (iterCount > (maxIter / 40)) {

/\*Set point color to yellow. \*/

ptColor.r = ptColor.g = 1.0;

ptColor.b = 0.0;

}

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

/\*set pointcolor to dark green \*/

ptColor.r = 0.0;

ptColor.b = 0.0;

ptColor.g = 0.3;

}

else {

/\*set pointcolor to cyan \*/

ptColor.r = 0.0;

ptColor.b = ptColor.g = 1.0;

}

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

plotPoint(z);

}

}

void displayFcn() {

/\*set number of x and y sudsvs. and the max iterations.\*/

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);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(50, 50);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Mandelbrot Set");

init();

glutDisplayFunc(displayFcn);

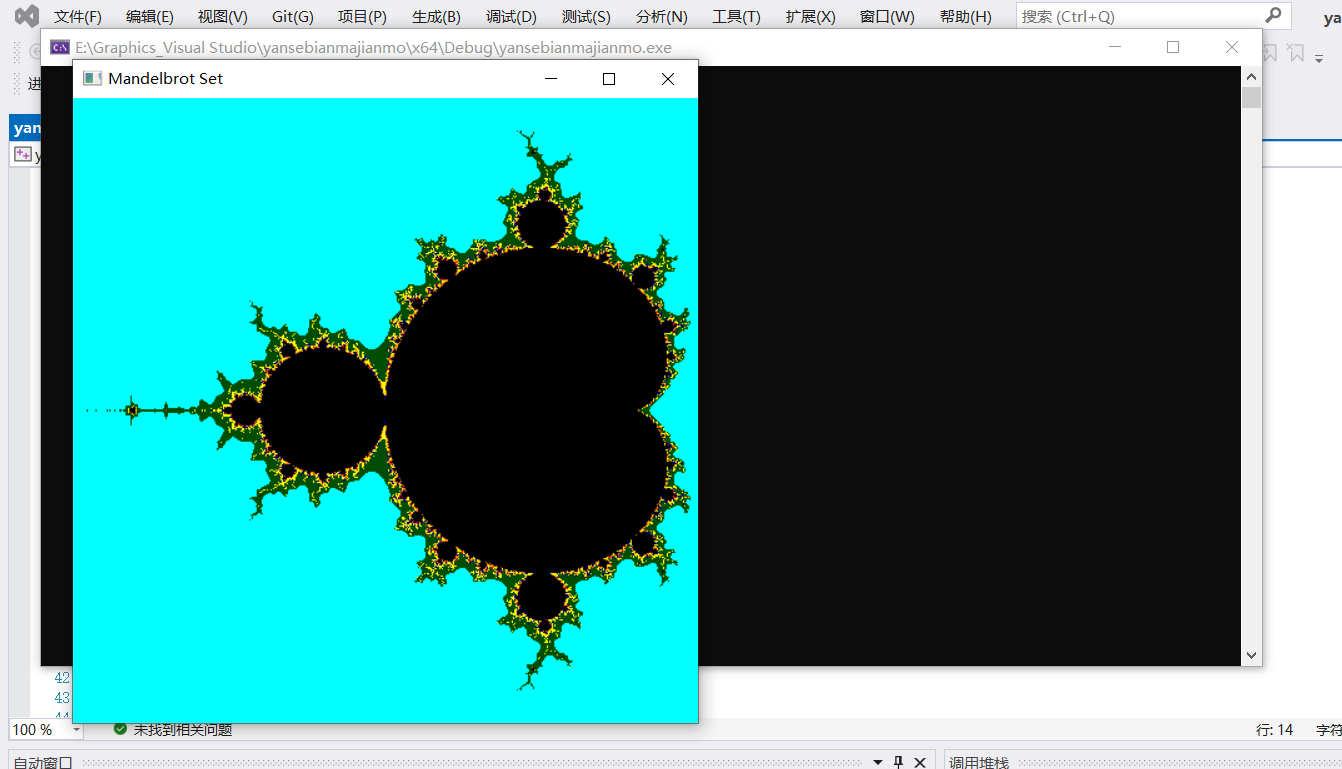
glutReshapeFunc(winReshapeFcn);

glutMainLoop();

return 0;

}

4.实验结果：



实验体会：OpenGL支持两种颜色模式：一种是RGBA，一种是颜色索引模式。无论哪种颜色模式，计算机都必须为每一个像素保存一些数据。不同的是，RGBA模式中，数据直接就代表了颜色；而颜色索引模式中，数据代表的是一个索引，要得到真正的颜色，还必须去查索引表。通过改变下面代码中glColor3f的参数值，绘制不同颜色。glColor系列函数，在参数类型不同时，表示“最大”颜色的值也不同。使用glIndex\*系列函数可以在颜色表中选择颜色。其中最常用的可能是glIndexi，它的参数是一个整形。void glIndexi(GLint c);GLUT工具包提供了设置颜色表的函数glutSetColor。这个实验就是对书上代码的复述和实现，参数的选择极易搞错，一旦输错就会出现运行界面空白的情况。

实验十一 交互控制实验

时间：2022年5月17 日

地点：信息学院2202机房

1.实验内容：

实验opengl，完成鼠标、键盘交互操作。

2.实验目的：

熟悉鼠标键盘交互。

3.实验代码：

// jiaohushiyan.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <iostream>

#include<Windows.h>

#include <GL/glut.h>

#include <math.h>

#define DEG\_TO\_RAD 0.017453 //角度转为弧度的参数，即 2\*PI/360

float theta = 30.0; //直线与X轴正方向的夹角

float length = 200.0; //直线的长度

float x = 300.0, y = 200.0; //直线的第一个端点

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 640.0, 0.0, 480.0);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(x, y);

glVertex2f(x + length \* cos(DEG\_TO\_RAD \* theta),

y + length \* sin(DEG\_TO\_RAD \* theta));

glVertex2f(x + length \* cos(DEG\_TO\_RAD \* (theta + 30)),

y + length \* sin(DEG\_TO\_RAD \* (theta + 30)));

glEnd();

glutSwapBuffers(); //交换前后台缓存

}

void idleFunc()

{

theta += 0.1;

if (theta > 360) theta -= 360;

glutPostRedisplay(); //重新调用绘制函数

}

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

{

//按下鼠标左键

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

theta += 5.0;

//按下鼠标右键

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

theta -= 5.0;

if (theta > 360) theta -= 360;

if (theta < 0) theta += 360;

glutPostRedisplay(); //重新调用绘制函数

}

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

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(640, 480);

glutCreateWindow("Draw Triangle with Double Buffer");

init();

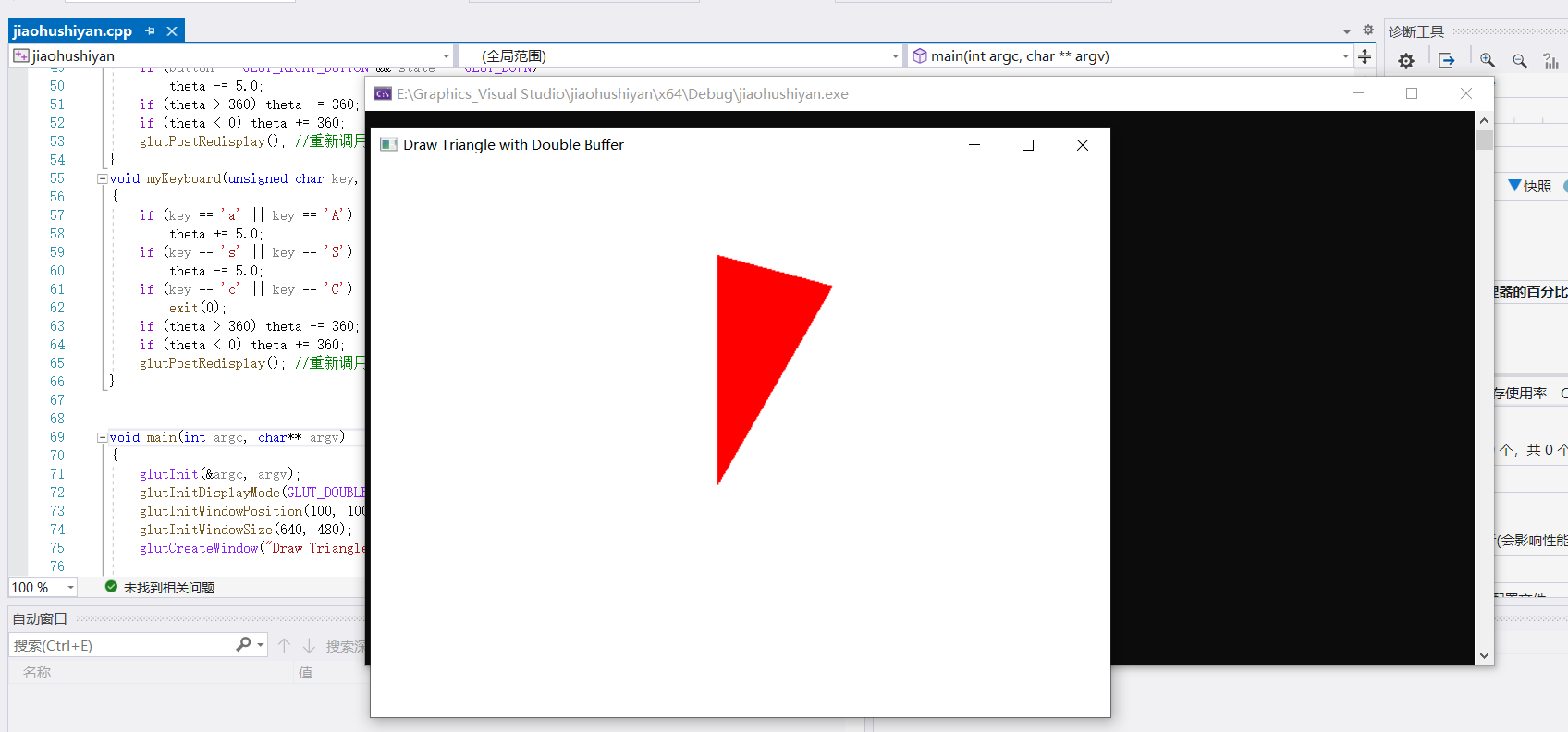
glutDisplayFunc(display);

glutMouseFunc(myMouse); //指定空闲回调函数

glutMainLoop();

}

4.实验结果：



实验体会：

OpenGL中通过鼠标和键盘跟程序交互的实现需要实现注册鼠标和键盘响应事件，在一定条件下，该事件被触发，事件里的程序被执行，达到交互的目的。

通过glutMouseFunc(&OnMouse)注册鼠标事件，OnMouse是鼠标事件的响应，函数格式是void OnMouse(int button,int state,int x,int y)；

通过glutKeyboardFunc(&KeyBoards)注册键盘事件，KeyBoards是键盘事件的响应，函数格式void KeyBoards(unsigned char key,int x,int y)；

对于鼠标func:形如void func(int button, int state, int x, int y);

button:鼠标的按键,为以下定义的常量

GLUT\_LEFT\_BUTTON:鼠标左键

GLUT\_MIDDLE\_BUTTON:鼠标中键

GLUT\_RIGHT\_BUTTON:鼠标右键

state:鼠标按键的动作,为以下定义的常量

GLUT\_UP:鼠标释放

GLUT\_DOWN:鼠标按下

x,y:鼠标按下式,光标相对于窗口左上角的位置

当点击鼠标时调用.

实验十二 三维观察变换

时间：2022年5月25日

地点：信息学院机房

1.实验内容：

使用opengl，完成投影变换等实验

P 264

2.实验目的：

熟悉三维观察相关内容

3.实验代码：

// work\_12.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include<Windows.h>

#include <iostream>

#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

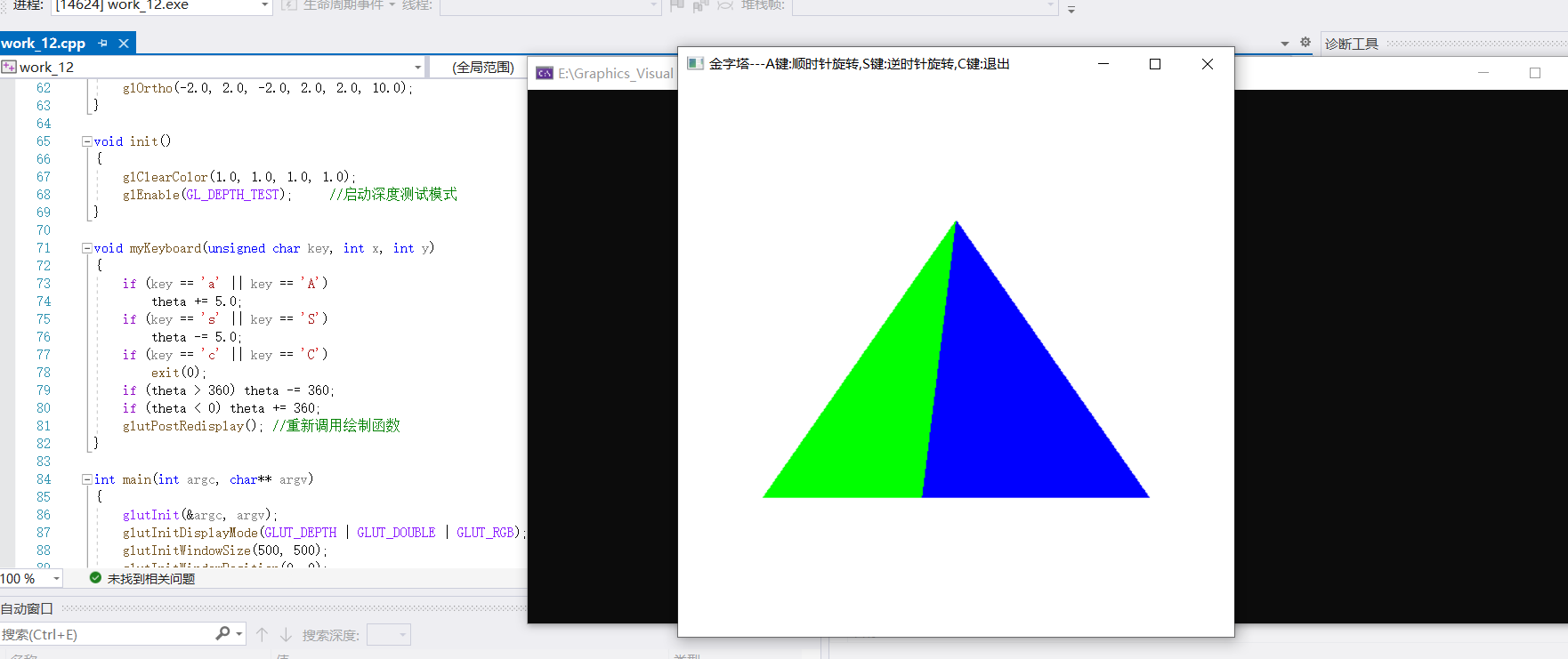
init();

glutMainLoop();

return 0;

}

1. 实验结果：



实验体会：

上述程序中用到了OpenGL深度缓存函数，下面进行介绍。当把三维物体投影到一个平面时，需要判定三维物体的哪个面是可见的，哪各面是不可见的，各个面之间怎样进行遮挡？所以，不启动深度缓存检测，就不能得到正确的投影效果。要启动深度缓存检测，必须修改GLUT初始化功能，使得显示模式包含深度缓存。事实上，投影变换完成两部分内容：（1）确定哪些对象可见，哪些不可见。OpenGL的投影函数确定一个长方体（正交投影）或一个棱台（透视投影）为视域体，这个区域里面的对象才能被观察到，这个区域之外的对象被裁剪掉，不会出现在图像中。（2）确定位于视域体之内的对象上的某个特定位置出现在投影平面上的位置。

实验十三 多面体实验

时间：2022年6月1日

地点：信息学院机房

1实验内容：

生成多面体线框图，P300

2实验目的：

熟悉三维观察相关内容

3实验代码：

// work\_13.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <iostream>

#include <GL/glut.h>

GLsizei winWidth = 500, winHeight = 500;

static 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);//绘制立方体线框图

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, (GLsizei)newWidth, (GLsizei)newHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

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("Wire-Frame Example");

init();

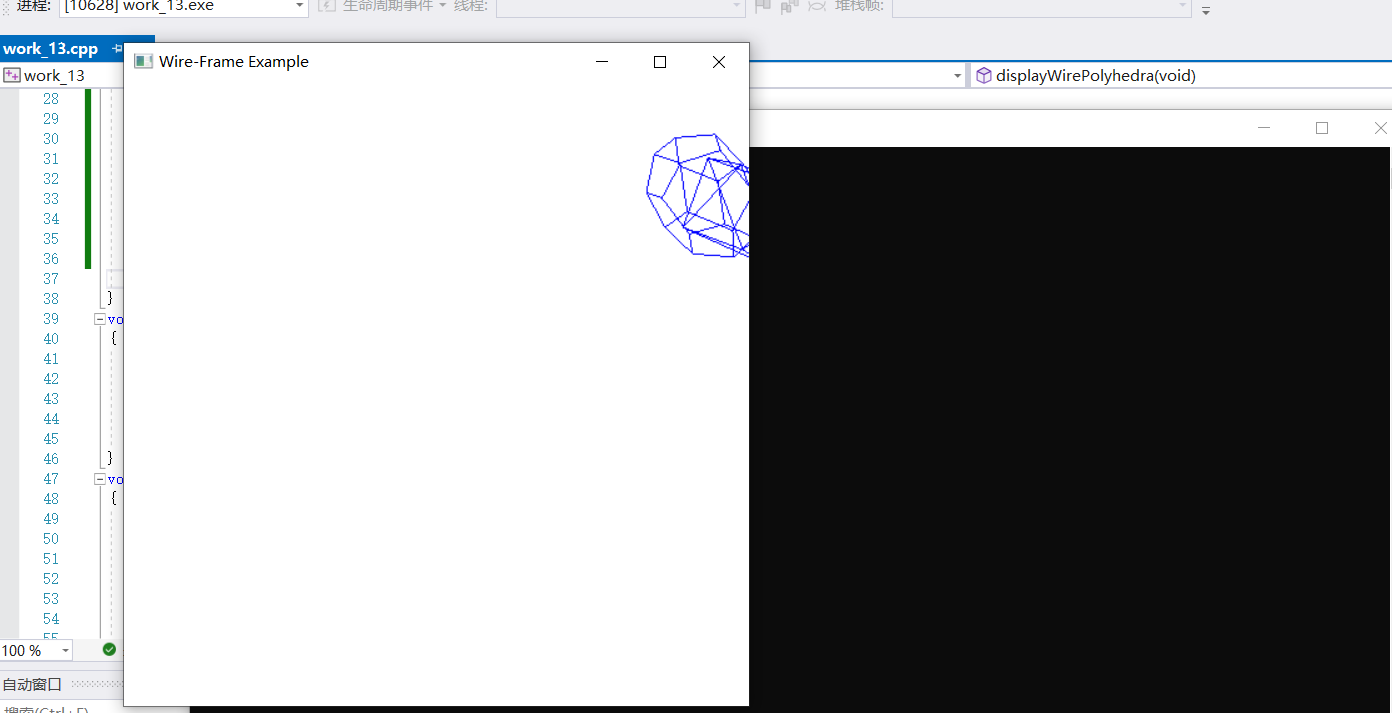
glutDisplayFunc(displayWirePolyhedra);

glutReshapeFunc(winReshapeFcn);

glutMainLoop();

}

4.实验结果：



实验体会：

某些标准的形体–五种规则多面体–由GLUT库中的子程序预先定义。这些多面体也称为柏拉图式实体，由它的每一面是一个单位规则多边形来标识。GLUT中总共有10个函数用来生成这些实体 glutWireTetrahedron();//规则三棱锥线框图

glutSolidTetrahedron();//规则三棱锥实体图glutWireCube(1.0);//规则立方体线框图glutSolidCube(1.0);glutWireOctahedron();//规则八面体线框图glutSolidOctahedron();glutWireDodecahedron();//规则十二面体线框图

glutSolidOctahedron();glutWireIcosahedron();//规则二十面体线框图

glutSolidIcosahedron();

实验十四 曲线曲面生成实验

时间：2022年6月7日

地点：信息学院机房

1.实验内容：

生成曲线或者曲面，P323

2.实验目的：

熟悉Bezier、样条等相关内容

3.实验代码：

// work\_14.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <windows.h>

#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);

}

int 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);

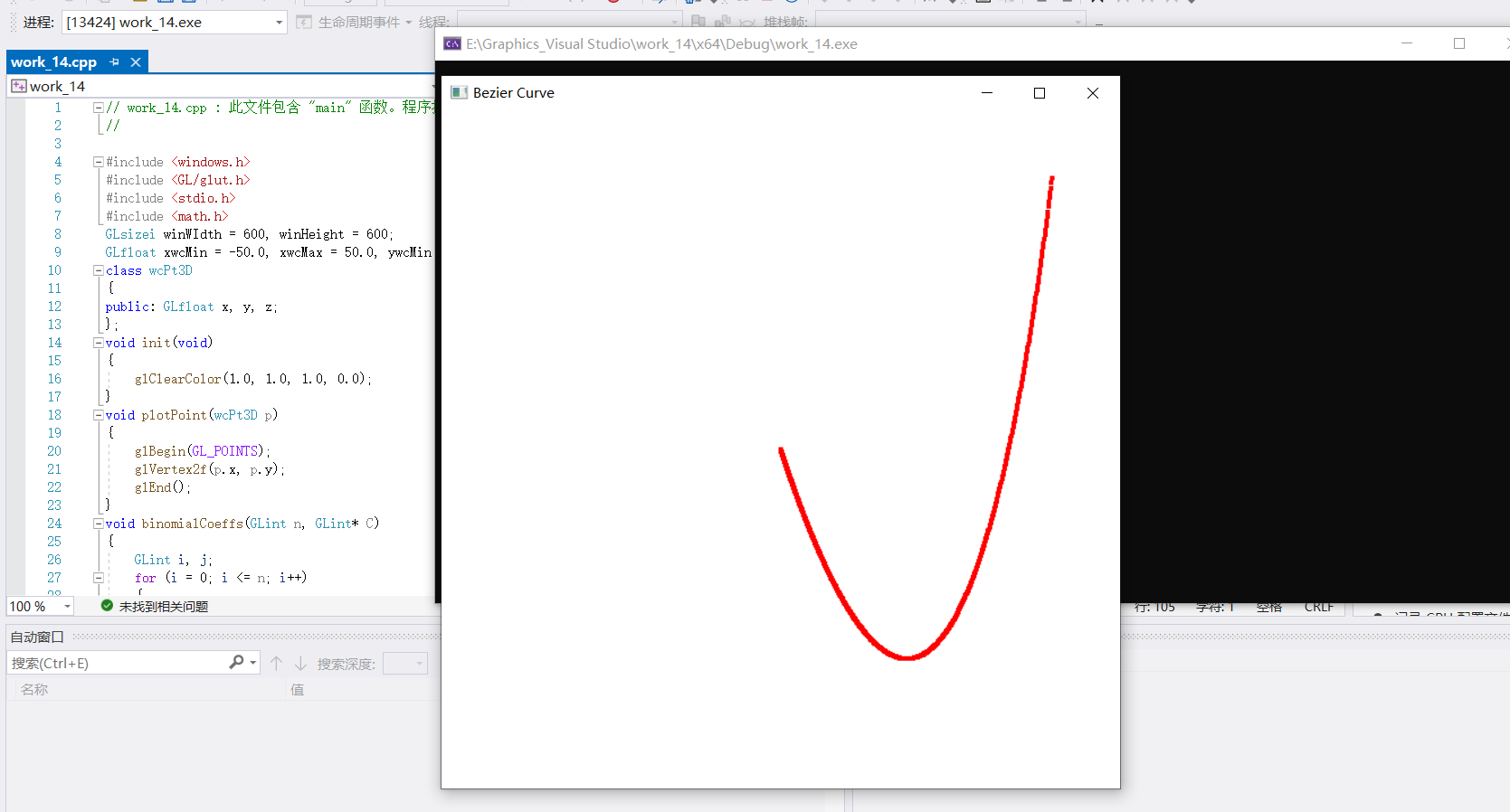
glutReshapeFunc(winReshapeFcn);

glutMainLoop();

return 0;

}

4、实验结果：



实验体会：

Bezier曲线通过控制曲线上的四个点（起始点、终止点以及两个相互分离的中间点）来创造、编辑图形。其中起重要作用的是位于曲线中央的控制线。这条线是虚拟的，中间与贝塞尔曲线交叉，两端是控制端点。移动两端的端点时贝塞尔曲线改变曲线的曲率（弯曲的程度）；移动中间点（也就是移动虚拟的控制线）时，贝塞尔曲线在起始点和终止点锁定的情况下做均匀移动。在计算机图形学中，样条曲线指由多项式曲线段连接而成的曲线，在每段的边界处满足特定的连续性条件。样条曲面(spline surface)可以使用两组正交样条曲线进行描述。如果我们用丄表示控制点的位置，则样条曲面上的任意一点可用样条曲线很合函数的积来计算。

实验十五 消隐实验

时间：2022年6月15日

地点：信息学院机房

1.实验内容：

完成消隐实验，采用Z-buffer算法完成消隐

2.实验目的：

熟悉Z-buffer 、画家算法等相关内容

3.实验代码：

// work\_15.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <iostream>

#include <iostream>

#include "Windows.h"

#include "math.h"

#include <iostream>

#include <gl/glut.h>

int s1, s2, s3; //视角位置，全局变量

using namespace std;

void Init()

{

glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

}

void Reshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

//3D

glOrtho(-w / 2, w / 2, -h / 2, h / 2, -300, 300);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void XYZ(void)

{

//坐标轴

glLineWidth(1); glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex3i(0, 0, 0); glVertex3i(320, 0, 0);

glVertex3i(0, 0, 0); glVertex3i(0, 240, 0);

glVertex3i(0, 0, 0); glVertex3i(0, 0, 300);

glEnd();

glFlush();

}

void myDisplay(void)

{

//顶点表

int x[10] = { 50, 50, 25, 0, 0, 50, 50, 25, 0, 0 };

int y[10] = { 0, 40, 60, 40, 0, 0, 40, 60,40, 0 };

int z[10] = { 140,140,140,140,140, 0, 0, 0, 0,0 };

//面点表

int f[7] = { 0,1,2,3,4,5,6 }; //面的号码

int p[7] = { 6,5,6,5,5,5,5 }; //面的顶点数

int fp[7][6] = { {0,1,2,3,4,0},{0,5,6,1,0,0},{5,9,8,7,6,5},{9,4,3,8,9,0},

{1,6,7,2,1,0},{3,2,7,8,3,0},{0,4,9,5,0,0} }; //面的顶点序

int i, j, k, SN;

int p1, p2, p3, u1, u2, u3, v1, v2, v3;

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

gluLookAt(s1, s2, s3, 0, 0, 0, 0, 1, 0);

XYZ();

glLineWidth(3); glColor3f(1.0, 0.0, 0.0);

//算法

for (i = 0; i < 7; i++)

{

p1 = fp[i][0]; p2 = fp[i][1]; p3 = fp[i][2]; //取前三个顶点

//计算法线

u1 = x[p2] - x[p1]; u2 = y[p2] - y[p1]; u3 = z[p2] - z[p1];

v1 = x[p3] - x[p2]; v2 = y[p3] - y[p2]; v3 = z[p3] - z[p2];

//计算法线与视角的点乘

SN = s1 \* (u2 \* v3 - u3 \* v2) + s2 \* (u3 \* v1 - u1 \* v3) + s3 \* (u1 \* v2 - u2 \* v1);

if (SN < 0) f[i] = -1; //法线与视角点乘小于零

}

for (i = 0; i < 7; i++)

{//消隐的部分

if (f[i] == -1)

{

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(2, 0x3333);

glBegin(GL\_LINE\_STRIP);

for (j = 0; j < p[i]; j++)

{

k = fp[i][j]; glVertex3i(x[k], y[k], z[k]);

}

// Sleep(1000);

glEnd(); glFlush();

glDisable(GL\_LINE\_STIPPLE);

}

//可见的部分

else

{

glBegin(GL\_LINE\_STRIP);

for (j = 0; j < p[i]; j++)

{

k = fp[i][j]; glVertex3i(x[k], y[k], z[k]);

}

// Sleep(1000);

glEnd(); glFlush();

}

}

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

cout << "请输入视角（如：1 1 1）" << endl;

cin >> s1 >> s2 >> s3;

glutInitWindowPosition(100, 100);

glutInitWindowSize(840, 680);

glutCreateWindow("FirstDemo");

Init();

glutDisplayFunc(myDisplay);

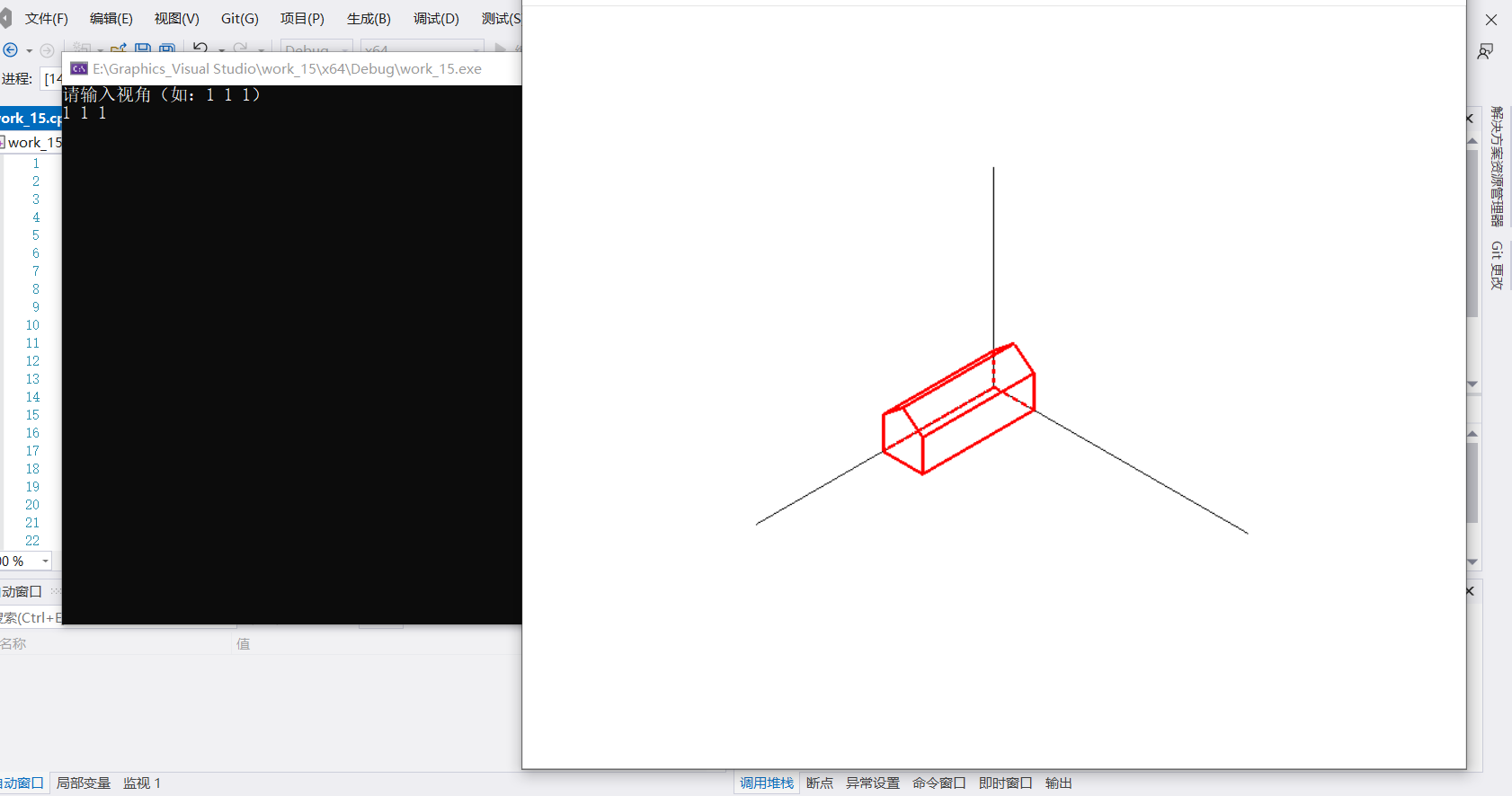
glutReshapeFunc(Reshape);

glutMainLoop();

return 0;

}

4.实验结果：



实验体会：算法思想是先将Z缓冲器中各单元的初始值置为最小值，当要改变某个象素的颜色值时，首先检查当前多边形的深度值是否大于该象素原来的深度值（保存在该象素所对应的Z缓冲器的单元中），如果大于原来的Z值，说明当前多边形更靠近观察点，用它的颜色替换原象素的颜色。假设xoy面为投影面，z轴为观察方向，过屏幕上任意象素点(x,y)做平行于z轴的射线R，与物体表面相交于p1,p2点，p1和p2点的z值称为该点的深度值，z-buffer算法比较p1和p2的z值，将最大的z值存入z缓冲器中，显然屏幕上(x,y)显示p1点的颜色。