计算机图形学实验

姓 名：谢俊杰

学 号：20201060337

专 业：计算机科学与技术

教 师：钱文华

实验一 示例程序生成直线段实验

时间：2022 年 3 月 16 日
地点：信息学院 2202

1、实验内容：

（1）安装 OpenGL
（2）通过示例程序生成直线段

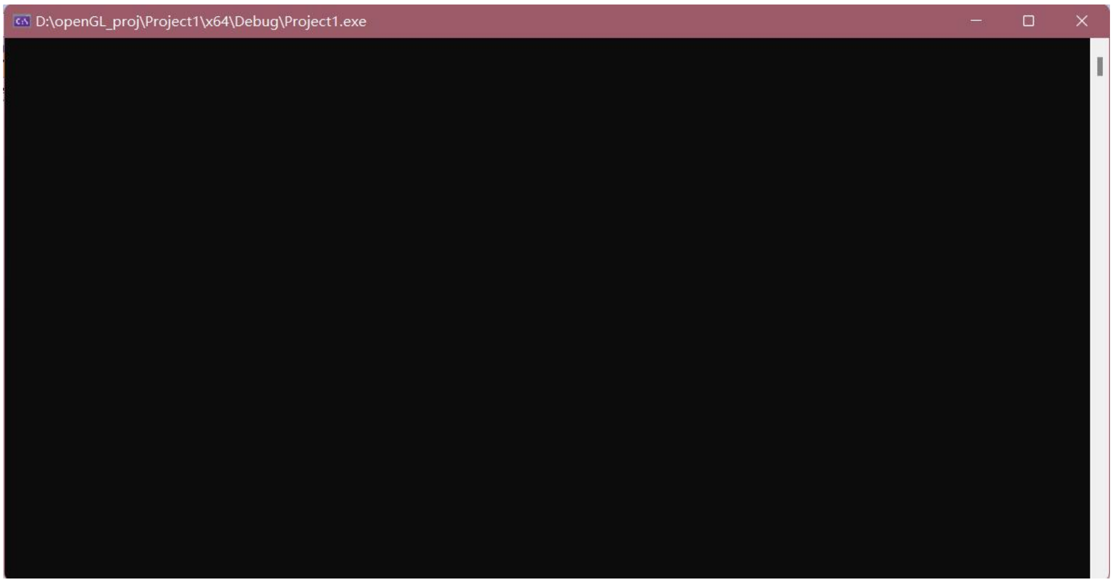
2、实验目的：

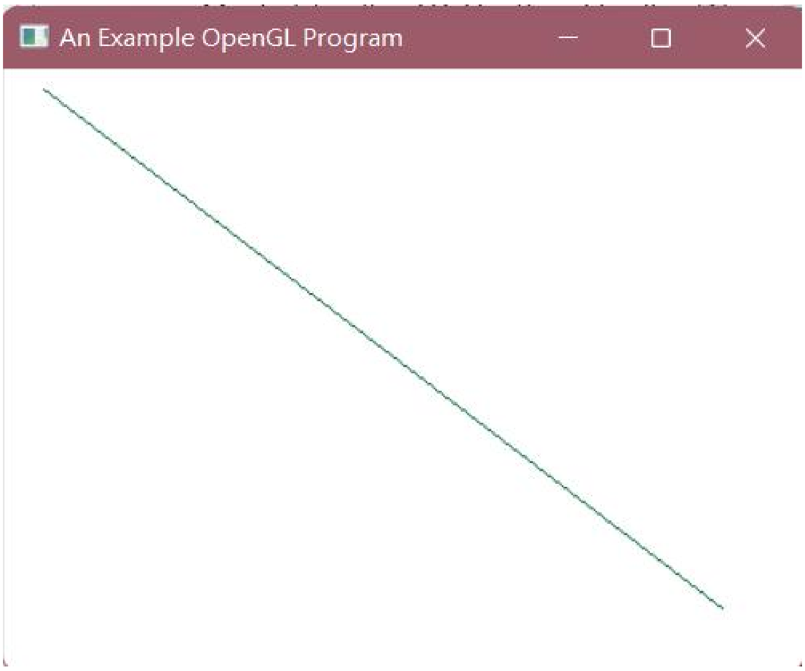
通过实验掌握下列知识:
（1）OpenGL glut的安装；
（2）OpenGL编程初步；
（3）熟悉OpenGL glut下的编程框架；
（4）使用OpenGL绘制点线等图元。

3、实验代码：

```
#include <windows.h>
#include <gl/glut.h>
#include <math.h>
#define GL_PI 3.1415f


void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
    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();
}

int 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();
    return 0;
}
```

4、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 2202

1、实验内容：

熟悉 OPENGL，通过 DDA、中点算法生成直线段

2、实验目的：

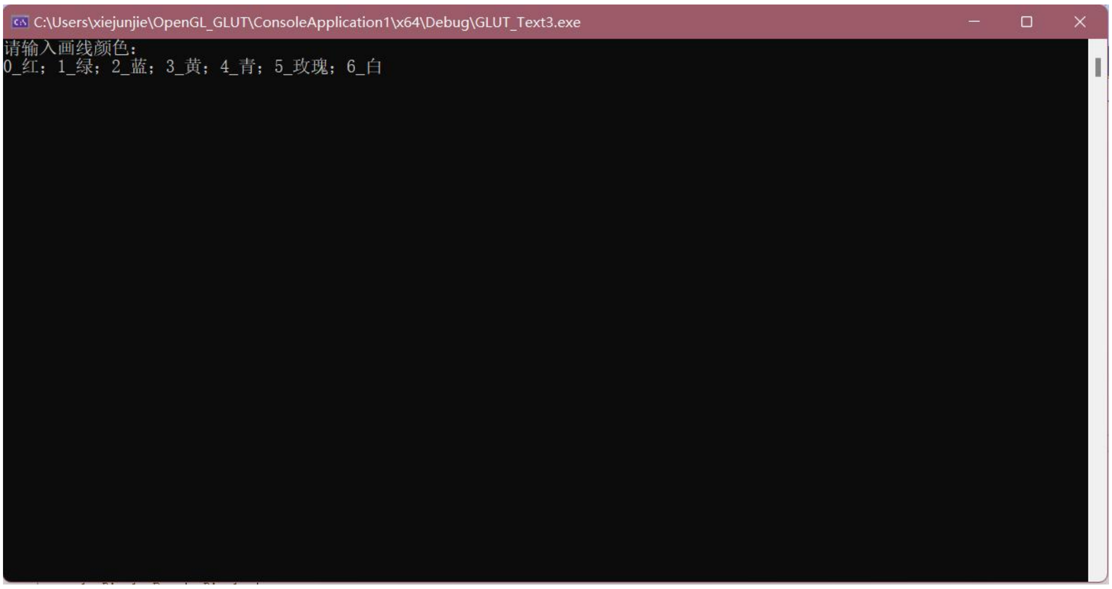
装 OPENGL，能编写代码运行，参考课本代码

3、实验代码：

```
#include <gl/glut.h>
#include <stdio.h>
#include <stdlib.h>

int size;
int color;
void lineDDA(int x0, int y0, int x1, int y1) {
    int x, dx, dy, y;
    float m;
    dx = x1 - x0;
    dy = y1 - y0;
    m = dy / dx;
    y = y0;
    switch (color){
    case 0:
        glColor3f(1, 0, 0);
        break;
    case 1:
        glColor3f(0, 1, 0);
        break;
    case 2:
        glColor3f(0, 0, 1);
        break;
    case 3:
        glColor3f(1, 1, 0);
        break;
    case 4:
        glColor3f(0, 1, 1);
        break;
    case 5:
        glColor3f(1, 0, 1);
        break;
    case 6:
        glColor3f(1, 1, 1);
        break;
    default:
        printf("无效的输入！\n");
    }
    glPointSize(size);
    for (x = x0; x <= x1; x++) {
        glBegin(GL_POINTS);
        glVertex2i(x, (int)(y + 0.5));
        glEnd();
        y = y + m;
    }
}

void myDisplay(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    lineDDA(10, 10, 200, 300);
    glBegin(GL_LINES);
    glColor3f(1.0, 0.0f, 0.0f);
    glVertex2f(100.0, 0.0);
    glEnd();
    glFlush();
}

void Init() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

void Reshape(int w, int h) {
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
}

int main(int argc, char** argv) {
    printf("请输入画线颜色：\n");
    printf("0_红；1_绿；2_蓝；3_黄；4_青；5_玫瑰；6_白\n");
    scanf_s("%d", &color);
    printf("请输入画线宽度：\n");
    scanf_s("%d", &size);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(400, 400);
    glutCreateWindow("DDALine");
    Init();
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(Reshape);
    glutMainLoop();
    return 0;
}
```

4、实验结果：

程序执行后，所显示的界面如下图所示：

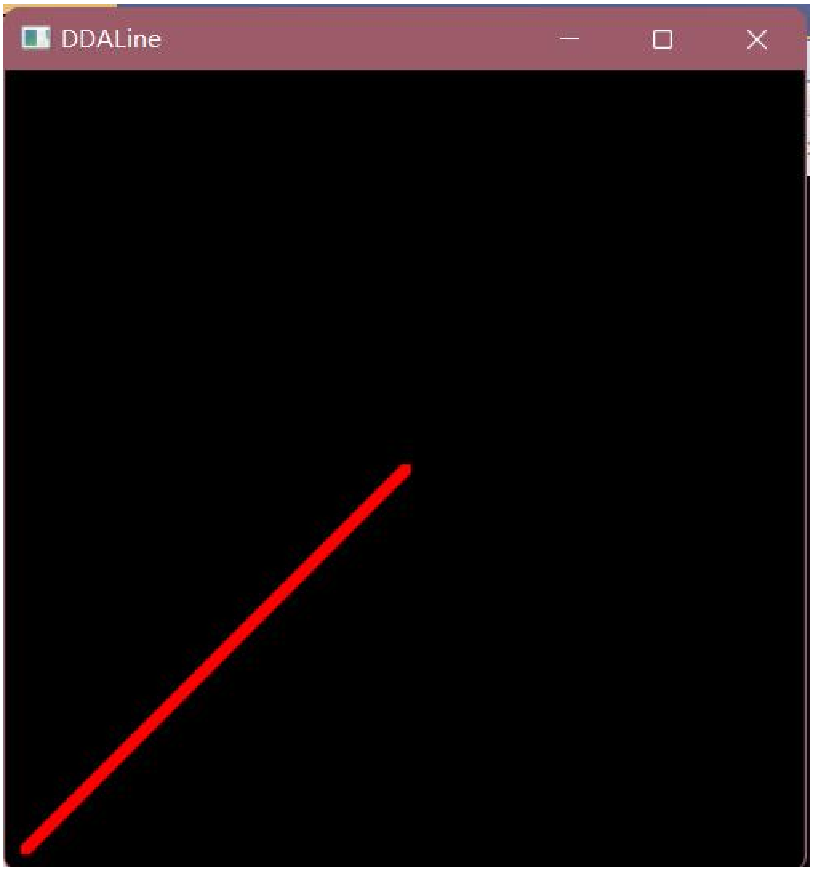


C:\Users\xiejunjie\OpenGL_GLUT\ConsoleApplication1\x64\Debug\GLUT_Text3.exe

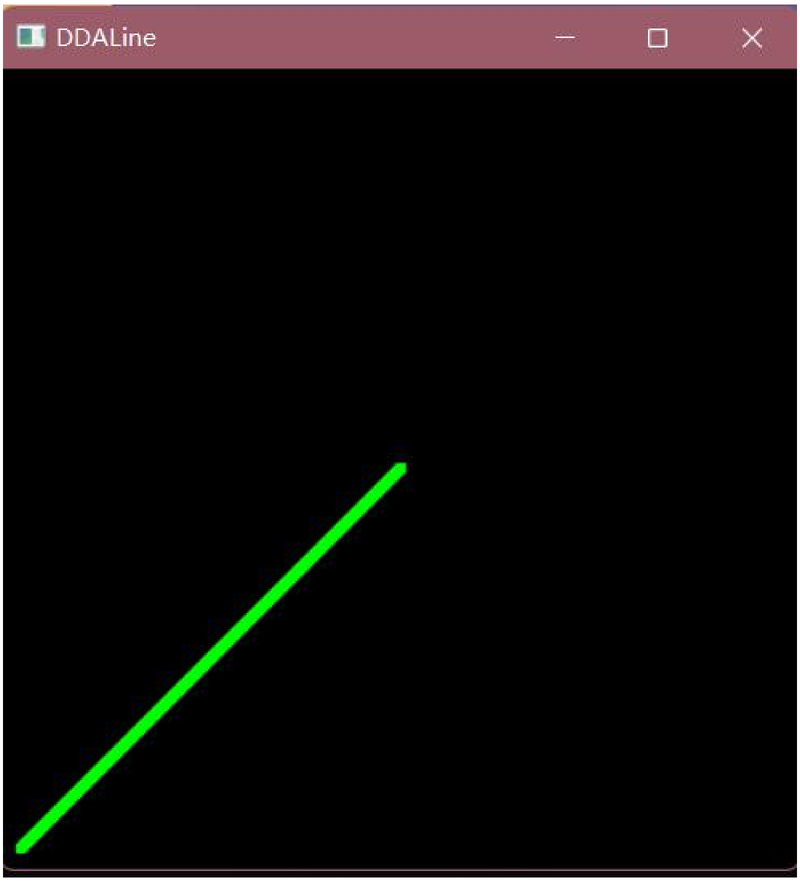
```
请输入画线颜色：
0_红；1_绿；2_蓝；3_黄；4_青；5_玫瑰；6_白
```

如下图所示，依次输入各个颜色所指示的编号，所画直线宽度为 5：

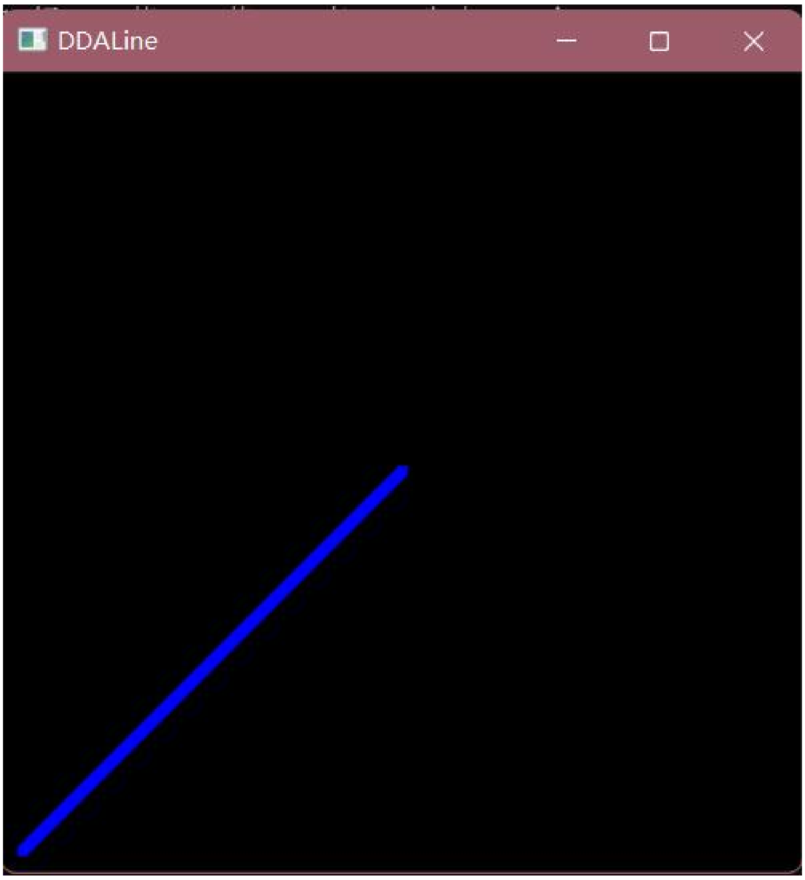
红色：



绿色：



蓝色：


DDALine

实验三　　Bresenham 算法、改进 Bresenham 算法生成直线段

实验时间：
实验地点：

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。

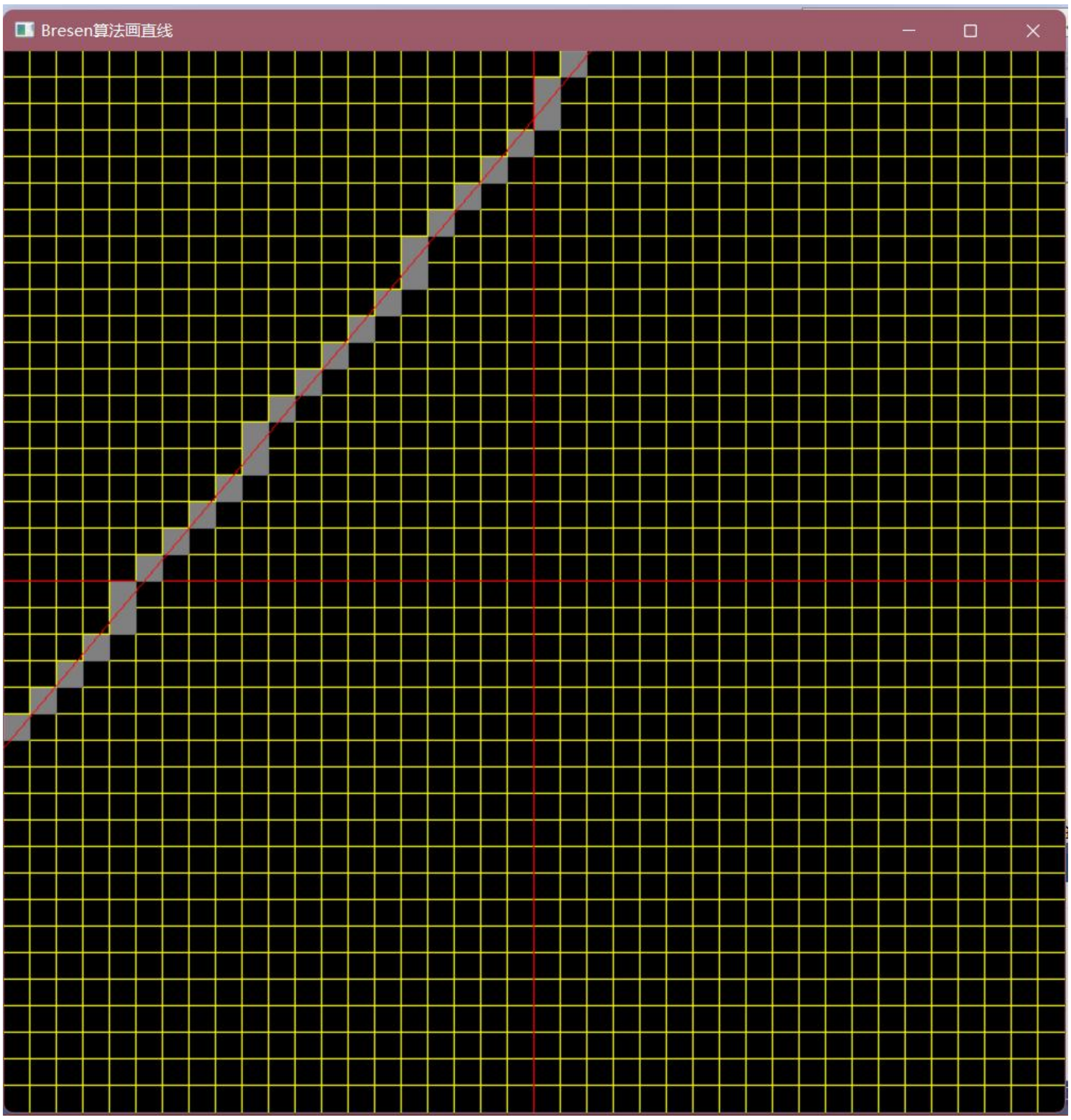
3、实验代码：

```
#include<gl/glut.h>
#include<algorithm>
using namespace std;
float window_size = 800;
int numbers = 20;
int xs = -115, ys = -119, xe = 35, ye = 59;

void InitEnvironment()  //对环境进行初始化操作
{
    glClearColor(0.0, 0.0, 0.0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluOrtho2D(-numbers, numbers, -numbers, numbers);
}

void draw_point(float x, float y) {
    glColor3f(0.5, 0.5, 0.5);
    glRectf(x, y, x + 1, y + 1);
}
void draw_point2(float x, float y) {
    draw_point(x, y);
    glPointSize(8);
    glColor3f(0.0, 0.0, 1.0);
    glBegin(GL_POINTS);
    glVertex3f(x, y, 0.0);
    glEnd();
    glFlush();
}

void Bresenhamline(int x0, int y0, int x1, int y1) {
    //支持各种斜率，支持两点颠倒
    int x, y, dx, dy;
    float k, e;
```

```
dx = x1 - x0, dy = y1 - y0;
if (dy == 0) {//平行坐标轴
    x = min(x0, x1);
    for (int i = 0; i <= abs(dx); i++) {
        draw_point(x, y0);
        x++;
    }
    return;
}
if (dx == 0) {//斜率不存在
    y = min(y0, y1);
    for (int i = 0; i <= abs(dy); i++) {
        draw_point(x0, y);
        y++;
    }
    return;
}
k = float(dy) / float(dx);
x = x0, y = y0;

if (k >= 0) {
    e = -0.5;
    x = min(x0, x1);
    y = min(y0, y1);
}
else {
    e = 0.5;
    x = min(x0, x1);
    y = max(y0, y1);
}
if (0 <= k && k <= 1) {
    for (int i = 0; i <= abs(dx); i++) {
        draw_point(x, y);
        x++, e += k;
        if (e >= 0) { y++; e--; }
    }
}
else if (k > 1) {
    k = float(dx) / float(dy);
    for (int i = 0; i <= abs(dy); i++) {
        draw_point(x, y);
        y++, e += k;
        if (e >= 0) { x++; e--; }
    }
    }
    else if (-1 <= k && k < 0) {
        for (int i = 0; i <= abs(dx); i++) {
            draw_point(x, y);
            x++, e += k;
            if (e <= 0) { y--; e++; }
        }
    }
    else if (k < -1) {
        k = float(dx) / float(dy);
        for (int i = 0; i <= abs(dy); i++) {
            draw_point(x, y);
            y--, e += k;
            if (e <= 0) { x++; e++; }
        }
    }
}
void myDisplay(void) {
    //绘制坐标系
    glColor3f(1.0, 1.0, 0);
    for (int i = 1; i < numbers * 2; i++) {
        glBegin(GL_LINES);
        glVertex2f(-numbers + i, -numbers);
        glVertex2f(-numbers + i, numbers);
        glVertex2f(-numbers, -numbers + i);
        glVertex2f(numbers, -numbers + i);
        glEnd();
    }
    glColor3f(1.0, 0, 0);
    glBegin(GL_LINES);
    glVertex2f(-numbers, 0);
    glVertex2f(numbers, 0);
    glVertex2f(0, -numbers);
    glVertex2f(0, numbers);
    glEnd();

    Bresenhamline(xs, ys, xe, ye);
    //绘制初始直线
    glBegin(GL_LINES);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(xs, ys); glVertex2f(xe, ye);
    glEnd();
    glFlush();
}
```

```
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);   //初始化
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(window_size, window_size);
    glutCreateWindow("Bresen 算法画直线");
    InitEnvironment();   //初始化
    glutDisplayFunc(&myDisplay);   //回调函数
    glutMainLoop();    //持续显示，当窗口改变会重新绘制图形
    return 0;
}
```

4、实验结果：



实验四　填充算法实验

1、实验内容：（1）教材 P66，填充六边形

（2）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：

（1）填充六边形

```
#include <gl/glut.h>
#include <math.h>
#include <stdlib.h>

const double TWO_PI = 6.2831853;

GLsizei winWidth = 400, winHeight = 400;
GLuint regHex;

class screenPt {
private:
    GLint x, y;
public:
    screenPt() {
        x = y = 0;
    }
    void setCoords(GLint xCoord, GLint yCoord) {
        x = xCoord;
        y = yCoord;
    }
    GLint getx() const {
        return x;
    }
    GLint gety() const {
        return y;
    }
};

static void init(void) {
    screenPt hexVertex, circCtr;
    GLdouble theta;
    GLint k;

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

    glClearColor(1.0, 1.0, 1.0, 0.0);
    regHex = glGenLists(1);
    glNewList(regHex, GL_COMPILE);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for (k = 0; k < 6; k++) {
        theta = TWO_PI * k / 6.0;
        hexVertex.setCoords(circCtr.getx() + 150 * cos(theta), circCtr.gety() + 150 * sin(theta));
        glVertex2i(hexVertex.getx(), hexVertex.gety());
    }
    glEnd();
    glEndList();
}

void regHexgon(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glCallList(regHex);
    glFlush();
}

void winReshapeFcn(int newWidth, int 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_List Example");
    init();
    glutDisplayFunc(regHexgon);
    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();
}
```

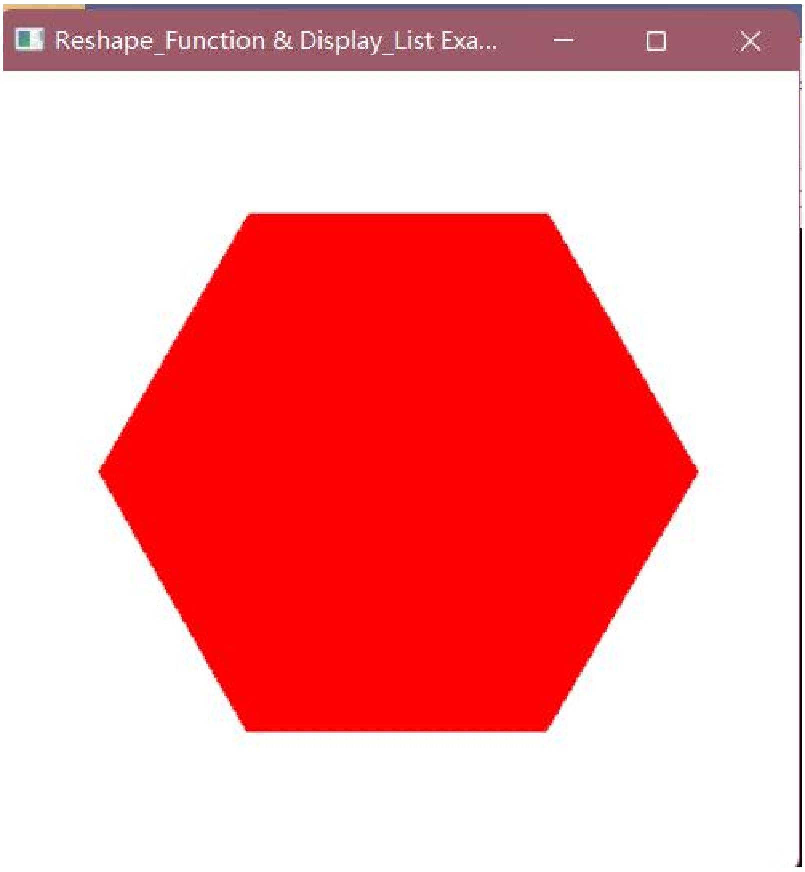
(2)填充多边形

```
#include <gl/glut.h>
#include <windows.h>
const int POINTNUM = 7;      //多边形点数.
```

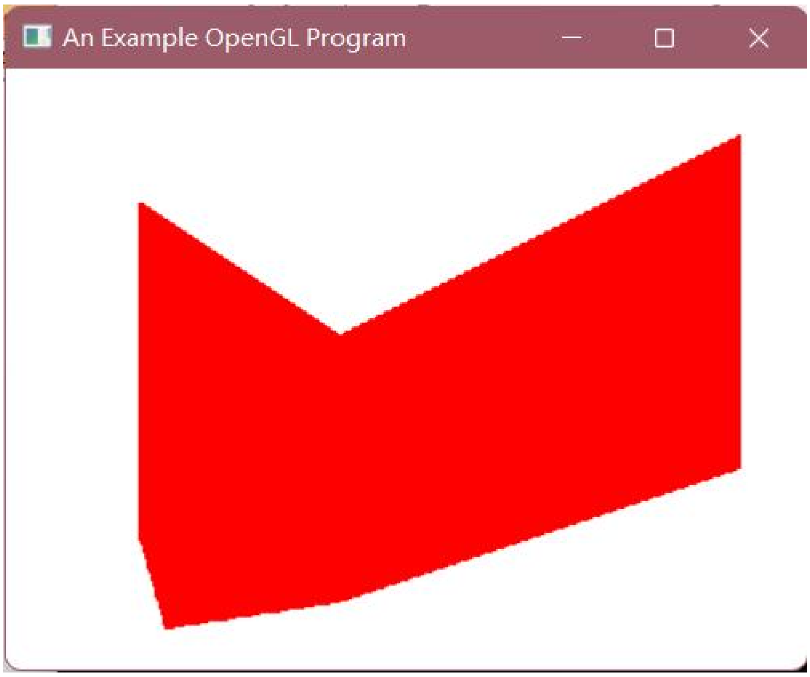
```
/******定义结构体用于活性边表 AET 和新边表 NET***********************************/
typedef struct XET
{
    float x;
    float dx, ymax;
    XET* next;
}AET, NET;

/******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(1.0, 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;
        }
        //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：

（1）填充六边形实验代码运行结果



（2）填充多边形实验代码运行结果



实验五 圆扫描转换和种子点填充实验

1、实验内容：

（1）圆的扫描转换

（2）种子点填充

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）圆的扫描转换

```
#include <GL/glut.h>

void circle(int x, int y, int x0, int y0) {
    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 Breseham(int x0, int y0, double r) {
    int x = 0;
    int y = (int)r;
    int d = int(3 - 2 * r);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    while (y >= x) {
        circle(x, y, x0, y0);
        if (d < 0)
            d += 4 * x + 6;
        else {
            d += 4 * (x - y) + 10;
            y--;
        }
        x++;
    }
    glEnd();
}

void display(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);

    Breseham(500, 500, 200.0);
    glFlush();
}
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize(800, 800);
    glutCreateWindow("Text5.1");
    gluOrtho2D(0, 1000, 0, 1000);
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

（2）种子点填充算法

```
#include <iostream>
#include<GL/glut.h>
#include <windows.h>
using namespace std;
int n;

struct vertex {
    float ver_x;
    float ver_y;
};
typedef struct XET {
    float x;
    float dx, ymax;
    XET* next;
}AET, NET;

struct point {
    float x;
    float y;
};

vertex* ver;
int c = 0;

void input(GLint button, GLint state, GLint x, GLint y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        ver[c].ver_x = x;
        ver[c].ver_y = y;
        cout << "第" << c + 1 << "个点为：" << x << "    " << y << endl;
        c++;
    }
}

void keyFromBoard() {
    for (int i = 0; i < n; i++) {
        int x, y;
        cin >> x >> y;
        ver[i].ver_x = x;
        ver[i].ver_y = y;
    }
}

void fillwith() {
    int MaxY = 0;
    int i;

    for (i = 0; i < n; i++) {
        if (ver[i].ver_y >= MaxY) {
            MaxY = ver[i].ver_y;
        }
    }
    AET* pAET = new AET;
    pAET->next = NULL;
    NET* pNET[1024];
    for (i = 0; i <= MaxY; i++) {
        pNET[i] = new NET;
        pNET[i]->next = NULL;
    }

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.9, 0.5, 0.6);
    glBegin(GL_POINTS);
    for (i = 0; i < MaxY; i++) {
        for (int j = 0; j < n; j++) {
            if (ver[j].ver_y == i) {
                if (ver[(j + 1 + n) % n].ver_y > ver[j].ver_y) {
                    NET* p = new NET;
                    p->x = ver[j].ver_x;
                    p->ymax = ver[(j + 1 + n) % n].ver_y;
                    p->dx = (ver[(j + 1 + n) % n].ver_x - ver[j].ver_x) / (ver[(j + 1 + n) % n].ver_y - ver[j].ver_y);
                    p->next = pNET[i]->next;
                    pNET[i]->next = p;
                }
                if (ver[(j - 1 + n) % n].ver_y > ver[j].ver_y) {
                    NET* p = new NET;
                    p->x = ver[j].ver_x;
                    p->ymax = ver[(j - 1 + n) % n].ver_y;
                    p->dx = (ver[(j - 1 + n) % n].ver_x - ver[j].ver_x) / (ver[(j - 1 + n) % n].ver_y - ver[j].ver_y);
                    p->next = pNET[i]->next;
                    pNET[i]->next = p;
                }
            }
        }
    }
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i <= MaxY; i++) {
        AET* p = new AET;
        p = pAET->next;
        AET* n = new AET;
        //将新边表中的活性边按照从左到右的顺序排序
        if (pNET[i]->next && pNET[i]->next->next) {
            if (pNET[i]->next->dx > 0) {
                NET* t = new NET;
                t = pNET[i]->next;
                n = pNET[i]->next->next;
                t->next = NULL;
                n->next = NULL;
                pNET[i]->next = n;
                n->next = t;
            }
        }
        //更新活性边表中的活性边 x 坐标的值
        while (p) {
            p->x = p->x + p->dx;
            p = p->next;
        }
        p = pAET->next;
        n = pAET;
```

```
//删掉扫描线高度等同于 ymax 的废弃点
while (p) {
    if (p->ymax == i) {
        n->next = p->next;
        free(p);
        p = n->next;
    }
    else {
        p = p->next;
        n = n->next;
    }
}
//插入新点，按照顺序插入
p = pAET->next;
n = pAET;
NET* a = new NET;
a = pNET[i]->next;
if (a) {
    NET* b = new NET;
    b = a;
    while (b->next) {
        b = b->next;
    }
    if (!pAET->next) {
        pAET->next = a;
    }
    else {
        while (p) {
            if (a->x < p->x) {
                b->next = p;
                n->next = a;
                break;
            }
            if (!p->next) {
                p->next = a;
                break;
            }
            n = n->next;
            p = p->next;
        }
    }
}
//填充 2
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;
        }
    }
    glEnd();
    glFlush();
}

int init(void) {
    glClearColor(0.0, 1.0, 1.0, 0.0);//画完图形后的背景颜色
    glMatrixMode(GL_PROJECTION);
    //gluOrtho2D(x1, x2, y1, y2)窗口会显示在二维坐标内 x1<x<x2,y1<y<y2 这个区域的点
    gluOrtho2D(0.0, 600.0, 0.0, 450.0);//窗口的显示的值的范围
    cout << "输入要显示的多边形共有几个顶点" << endl;
    cin >> n;
    cout << "键盘输入为 1，鼠标输入为 2，你的选择是：" << endl;
    int x;
    cin >> x;
    return x;
}

int main(int argc, char* argv) {

    glutInit(&argc, &argv);//初始化 GLUT 库
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);//指定一个颜色为 RGE 显示的窗口或者单缓冲
区窗口
    glutInitWindowPosition(50, 100);//设置窗口位置，50: 距离屏幕左边的像素数。100：距离
屏幕上边的像素数
    glutInitWindowSize(400, 300); //设置窗口大小
    glutCreateWindow("种子点填充算法");//设置窗口的标题

    int x = init();
    ver = (vertex*)malloc(sizeof(vertex) * n);//输入顶点以（x,y）格式

    if (x == 1) {
        keyFromBoard();
    }
    else if (x == 2) {
        //鼠标左点击
        for (int i = 0; i < n; i++) {
            glutMouseFunc(input);//鼠标点击时会调用该方法
        }
    }

    glutDisplayFunc(fillwith);
    glutMainLoop();
}
```

实验六　二维几何变换实验

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

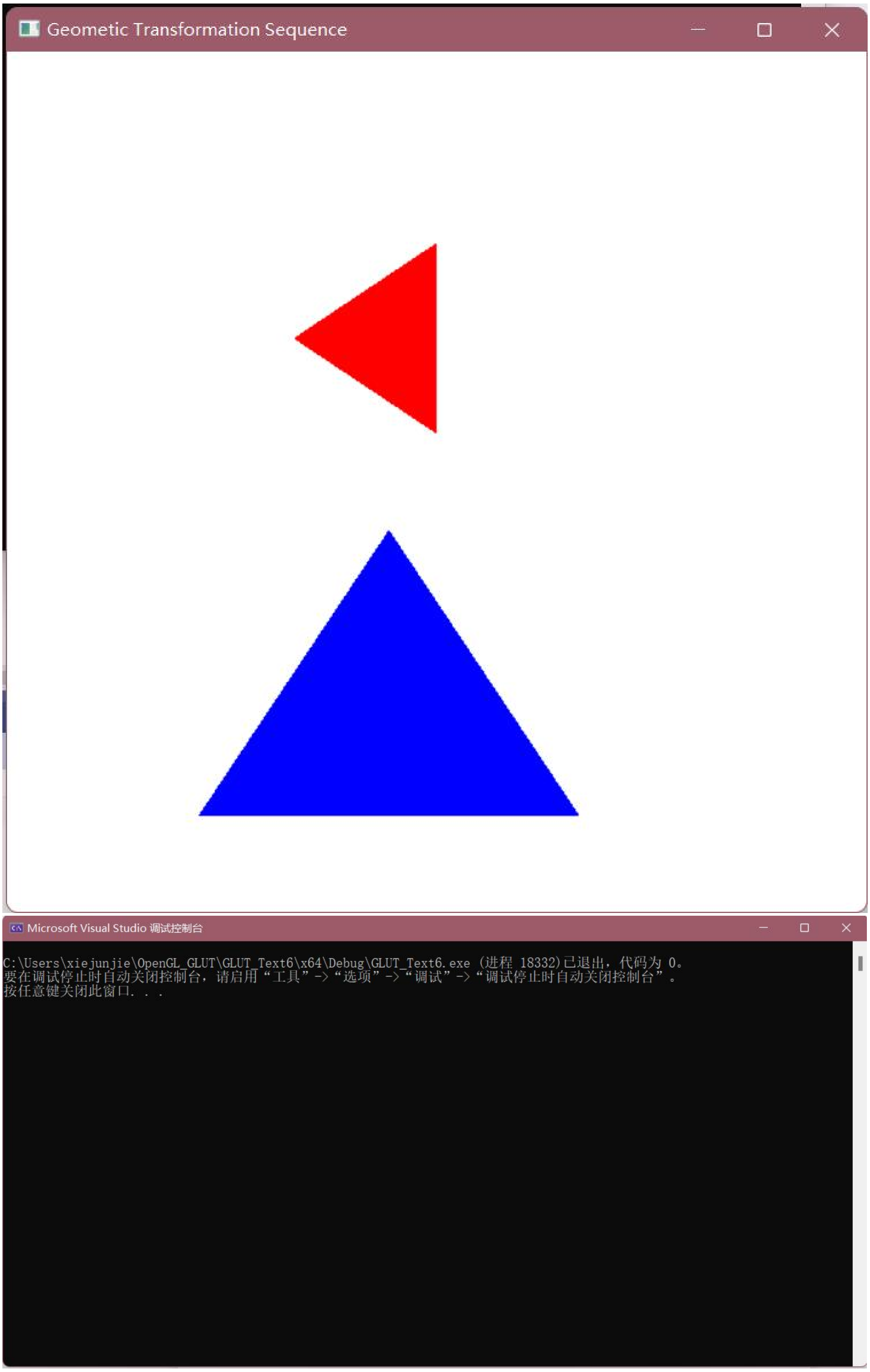
3、实验代码

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

GLsizei winWidth = 600, winHeight = 600;

GLfloat xwcMin = 0.0, xwcMax = 225.0;
GLfloat ywcMin = 0.0, ywcMax = 225.0;
class wcPt2D {
public:
    GLfloat x, y;
};

typedef GLfloat Matrix3x3[3][3];

Matrix3x3 matComposite;

const GLdouble pi = 3.14159;
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
}

void matrix3x3SetIdentity(Matrix3x3 matIdent3x3) {
    GLint row, col;

    for (row = 0; row < 3; row++)
        for (col = 0; col < 3; col++)
            matIdent3x3[row][col] = (row == col);
}

void matrix3x3PreMultply(Matrix3x3 m1, Matrix3x3 m2) {
    GLint row, col;
    Matrix3x3 matTemp;

    for (row = 0; row < 3; row++)
        for (col = 0; col < 3; col++)
            matTemp[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] = matTemp[row][col];
}

void translate2D(GLfloat tx, GLfloat ty) {
    Matrix3x3 matTransl;

    matrix3x3SetIdentity(matTransl);

    matTransl[0][2] = tx;
    matTransl[1][2] = ty;

    matrix3x3PreMultply(matTransl, matComposite);
}

void rotate2D(wcPt2D pivotPt, GLfloat theta) {
    Matrix3x3 matRot;

    matrix3x3SetIdentity(matRot);
    matRot[0][0] = cos(theta);
    matRot[0][1] = -sin(theta);
    matRot[0][2] = pivotPt.x * (1 - cos(theta)) + pivotPt.y * sin(theta);

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

    matrix3x3PreMultply(matRot, matComposite);
}

void scale2D(GLfloat sx, GLfloat sy, wcPt2D fixedPt) {
    Matrix3x3 matScale;

    matrix3x3SetIdentity(matScale);

    matScale[0][0] = sx;
    matScale[0][2] = (1 - sx) * fixedPt.x;
    matScale[1][1] = sy;
    matScale[1][2] = (1 - sy) * fixedPt.y;

    matrix3x3PreMultply(matScale, matComposite);
}
```

```
void transformVerts2D(GLint nVerts, wcPt2D* verts) {
    GLint k;
    GLfloat temp;

    for (k = 0; k < nVerts; k++) {
        temp = matComposite[0][0] * verts[k].x + matComposite[0][1] * verts[k].y +
matComposite[0][2];
        verts[k].y = matComposite[1][0] * verts[k].x + matComposite[1][1] * verts[k].y +
matComposite[1][2];
        verts[k].x = temp;
    }
}

void triangle(wcPt2D* verts) {
    GLint k;

    glBegin(GL_TRIANGLES);
    for (k = 0; k < 3; k++)
        glVertex2f(verts[k].x, verts[k].y);
    glEnd();
}

void displayFcn(void) {
    GLint nVerts = 3;
    wcPt2D verts[3] = { {50.0,25.0},{150.0,25.0},{100.0,100.0} };

    wcPt2D centroidPt;

    GLint k, xSum = 0, ySum = 0;
    for (k = 0; k < nVerts; k++) {
        xSum += verts[k].x;
        ySum += verts[k].y;
    }
    centroidPt.x = GLfloat(xSum) / GLfloat(nVerts);
    centroidPt.y = GLfloat(ySum) / GLfloat(nVerts);

    wcPt2D pivPt, fixedPt;
    pivPt = centroidPt;
    fixedPt = centroidPt;

    GLfloat tx = 0.0, ty = 100.0;
    GLfloat sx = 0.5, sy = 0.5;
    GLdouble theta = pi / 2.0;
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 0.0, 1.0);
    triangle(verts);

    matrix3x3SetIdentity(matComposite);

    scale2D(sx, sy, fixedPt);
    rotate2D(pivPt, theta);
    translate2D(tx, ty);

    transformVerts2D(nVerts, verts);

    glColor3f(1.0, 0.0, 0.0);
    triangle(verts);

    glFlush();
}

void winReshapeFcn(GLint newWidth, GLint 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("Geometic Transformation Sequence");

    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();
}
```

4、实验结果


Geometic Transformation Sequence
Microsoft Visual Studio 调试控制台

```
C:\Users\xiejunjie\OpenGL_GLUT\GLUT_Text6\x64\Debug\GLUT_Text6.exe (进程 18332)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

实验七　GLUT 鼠标函数实验

1、实验内容

（1）教材 P458，GLUT 鼠标函数；

（2）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

（1）

```
#include <gl/glut.h>

GLsizei winWidth = 400, winHeight = 300;//初始化窗口的尺寸

void init(void) {
    glClearColor(0.0, 0.0, 1.0, 1.0);//将窗口颜色设置为蓝色
    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("Mouse Plot Points");
    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMouseFunc(mousePtPlot);
    glutMainLoop();
}
```

（2）

```
#include <gl/glut.h>
#include <stdio.h>
#include <stdlib.h>
GLsizei winWidth = 400, winHeight = 300;
GLint ptCtr = 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, 0.0, 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;
}
```

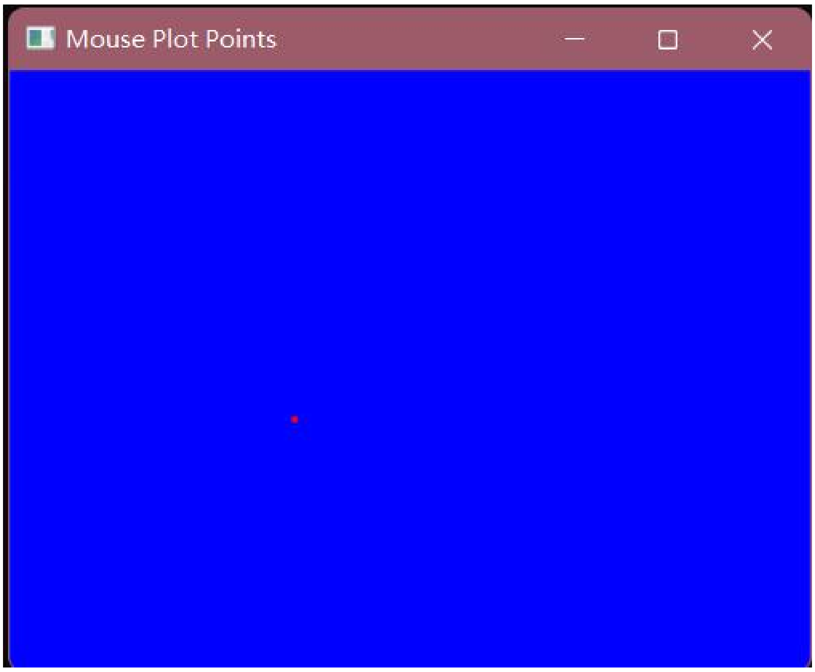
```
void drawLineSegment(scrPt endPt1, scrPt endPt2) {
    glBegin(GL_LINES);
    glVertex2i(endPt1.x, endPt1.y);
    glVertex2i(endPt2.x, endPt2.y);
    glEnd();
}

void polyline(GLint button, GLint action, GLint xMouse, GLint yMouse) {
    static scrPt endPt1, endPt2;
    if (ptCtr == 0) {
        if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
            endPt1.x = xMouse;
            endPt1.y = winHeight - yMouse;
            ptCtr = 1;
        }
        else {
            if (button == GLUT_RIGHT_BUTTON) {
                exit(0);
            }
        }
    }
    else {
        if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
            endPt2.x = xMouse;
            endPt2.y = winHeight - yMouse;
            drawLineSegment(endPt1, endPt2);
        }
        else {
            if (button == GLUT_RIGHT_BUTTON) {
                exit(0);
            }
        }
    }
    glFlush();
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow(″Draw Interactive Polyline″);
    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMouseFunc(polyline);
    glutMainLoop();
}
```

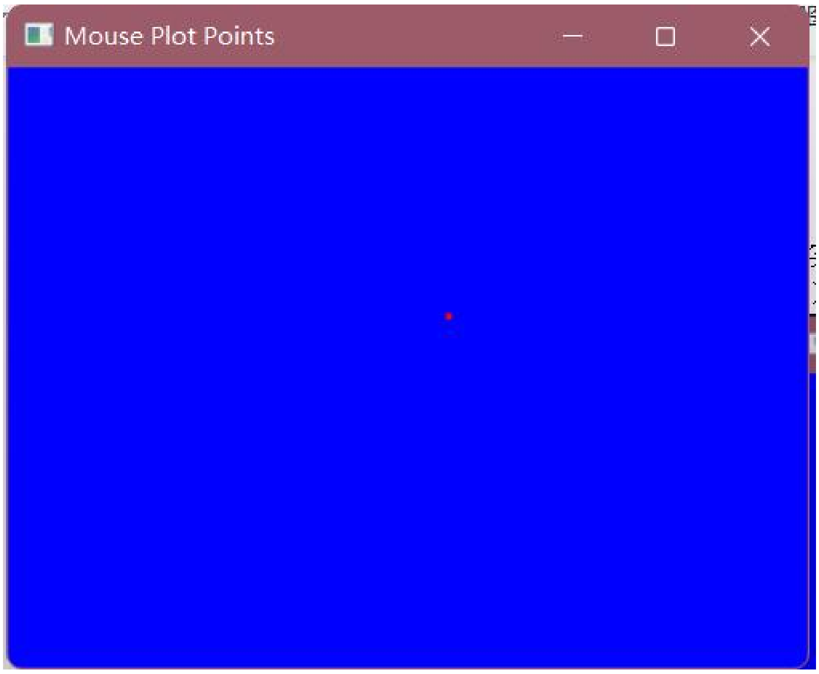
（3）

```
#include <GL/glut.h>
#include <stdio.h>

static float rotAngle = 0.;

void init(void)
{
    GLfloat values[2];
    glGetFloatv(GL_LINE_WIDTH_GRANULARITY, values);
    printf("GL_LINE_WIDTH_GRANULARITY value is %3.1f\n", values[0]);

    glGetFloatv(GL_LINE_WIDTH_RANGE, values);
    printf("GL_LINE_WIDTH_RANGE values are %3.1f %3.1f\n", values[0], values[1]);

    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glHint(GL_LINE_SMOOTH_HINT, GL_DONT_CARE);

    glLineWidth(1.5);

    glClearColor(0.0, 0.0, 0.0, 0.0);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 1.0, 0.0);
    glPushMatrix();
    glRotatef(-rotAngle, 0.0, 0.0, 0.1);
    glBegin(GL_LINES);
    glVertex2f(-0.5, 0.5);
    glVertex2f(0.5, -0.5);
    glEnd();
    glPopMatrix();

    glColor3f(0.0, 0.0, 1.0);
    glPushMatrix();
    glRotatef(rotAngle, 0.0, 0.0, 0.1);
    glBegin(GL_LINES);
    glVertex2f(0.5, 0.5);
    glVertex2f(-0.5, -0.5);
    glEnd();
    glPopMatrix();

    glFlush();
}

void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        gluOrtho2D(-1.0, 1.0, -1.0 * (GLfloat)h / (GLfloat)w, 1.0 * (GLfloat)h /
(GLfloat)w);
    else
        gluOrtho2D(-1.0 * (GLfloat)w / (GLfloat)h, 1.0 * (GLfloat)w / (GLfloat)h, -1.0,
1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'r': case 'R':
        rotAngle += 20.;
        if (rotAngle >= 360.0) rotAngle = 0.0;
        glutPostRedisplay();
        break;
    case 27:
        exit(0);
        break;
    default:
        break;
    }
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400, 300);
    glutCreateWindow("OpenGL 反走样技术");
    init();
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

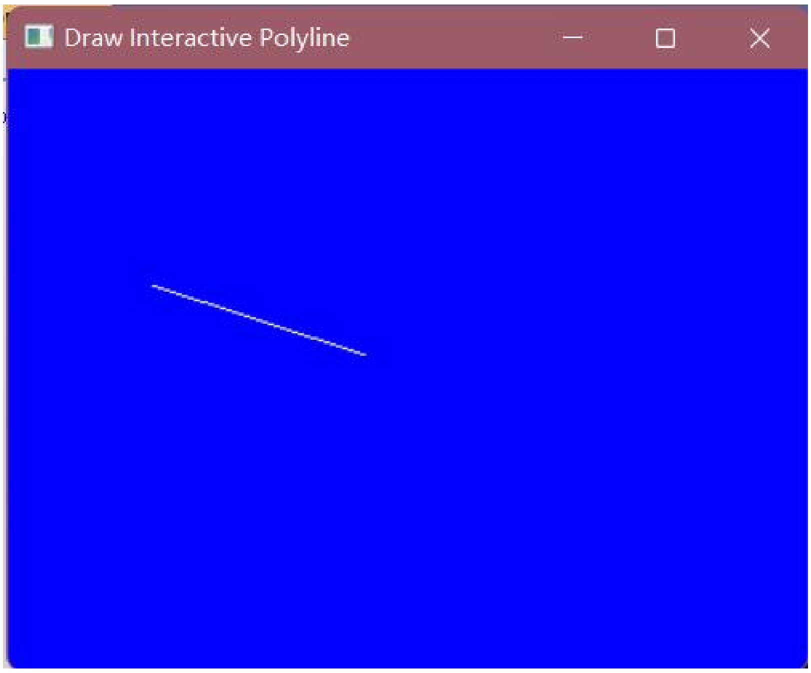
4、实验结果

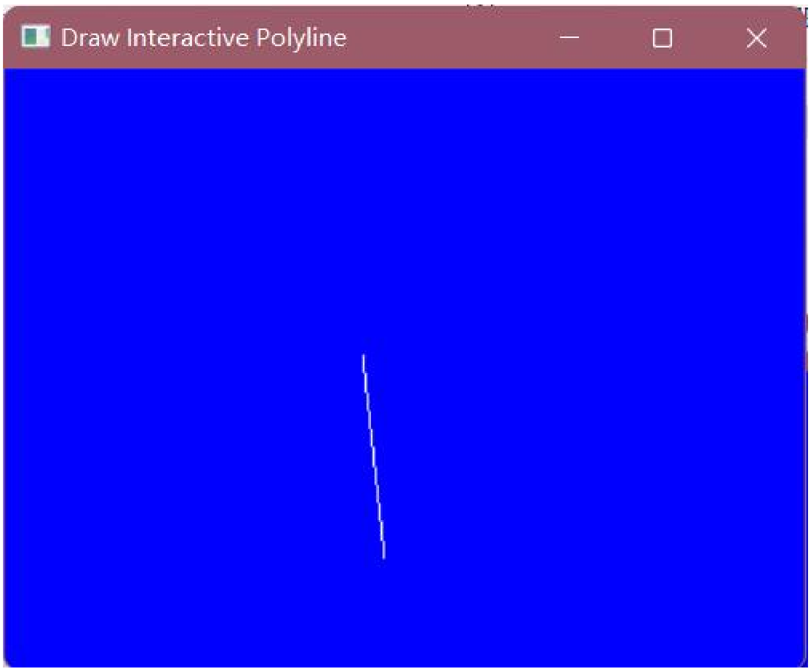
（1）



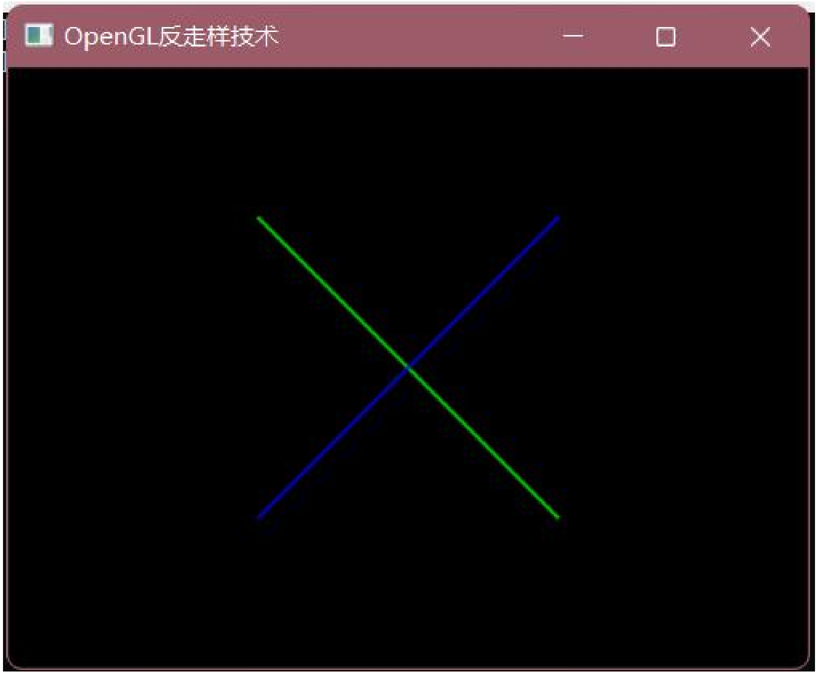


（2）





（3）



实验八　二维图像裁剪实验

1、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

2、实验目的

验证 Cohen-Sutherland 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

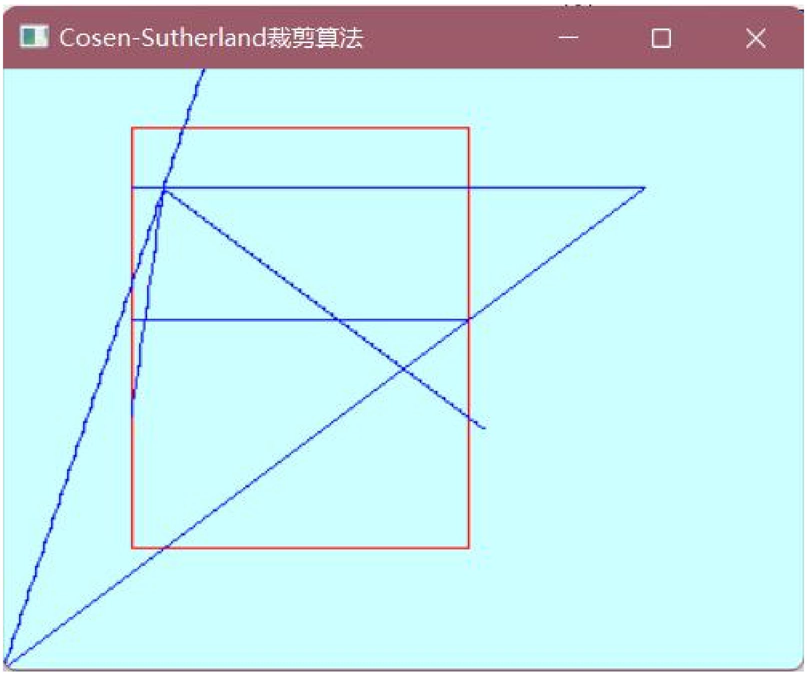
```
#include <gl/glut.h>
#include <math.h>
#include<stdio.h>
```

```
class wcPt2D{
public:
    GLfloat x, y;
};
const GLint winLeftBitCode = 0x1;
const GLint winRightBitCode = 0x2;
const GLint winBottomBitCode = 0x4;
const GLint winTopBitCode = 0x8;

inline GLint round(const GLfloat a) {
    return GLint(a + 0.5);
}

inline int inside(int code) {
    return int(!code);
}

inline int reject(int code1, int code2) {
    return int(code1 & code2);
}

inline int accept(int code1, int code2) {
    return int(!(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 draw_pixel(int ix, int iy) {
    glBegin(GL_POINTS);
    glVertex2i(ix, iy);
    glEnd();
}

void lineDDA(int x0, int y0, int x_end, int y_end, double a, double b, double c) {
    glColor3f(a, b, c);
    int dx = x_end - x0;
    int dy = y_end - 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);
    draw_pixel(round(x), round(y));
    for (k = 0; k < steps; k++) {
        x += xIncrement;
        y += yIncrement;
        draw_pixel(round(x), round(y));
    }
}

void lineClipCoSuth(wcPt2D winMin, wcPt2D winMax, wcPt2D p1, wcPt2D p2) {
    GLubyte code1, code2;
    GLint done = false, plotLine = false;
    GLfloat m;
    while (!done) {
        code1 = encode(p1, winMin, winMax);
        code2 = encode(p2, winMin, winMax);
        if (accept(code1, code2)) {
            done = true;
            plotLine = true;
        }
        else {
            if (reject(code1, code2))
                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);
                if (code1 & winLeftBitCode) {
                    p1.y += (winMax.x - p1.x) * m;
                    p1.x = winMax.x;
                }
                else {
                    if (code1 & winRightBitCode) {
                        p1.y += (winMax.x - p1.x) * m;
                        p1.x = winMin.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;
                            }
                        }
                    }
                }
            }
        }
    }
    if (plotLine)
        lineDDA(round(p1.x), round(p1.y), round(p2.x), round(p2.y), 0.0, 0.0, 1.0);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    wcPt2D winMin, winMax, p1, p2, q1, q2, t1, t2, m1, m2;
    winMin.x = 80;
    winMin.y = 100;
    winMax.x = 290;
    winMax.y = 500;
    lineDDA(80, 100, 80, 450, 1.0, 0.0, 0.0);
    lineDDA(80, 100, 290, 100, 1.0, 0.0, 0.0);
    lineDDA(290, 100, 290, 450, 1.0, 0.0, 0.0);
    lineDDA(80, 450, 290, 450, 1.0, 0.0, 0.0);
    p1.x = 0;
    p1.y = 0;
    p2.x = 400;
    p2.y = 400;
    q1.x = 0;
    q1.y = 0;
    q2.x = 100;
    q2.y = 400;
    t1.x = 100;
    t1.y = 400;
    t2.x = 400;
    t2.y = 400;
    m1.x = 300;
    m1.y = 200;
    m2.x = 100;
    m2.y = 400;
    lineClipCoSuth(winMin, winMax, p1, p2);
    lineClipCoSuth(winMin, winMax, q1, q2);
    lineClipCoSuth(winMin, winMax, t1, t2);
    lineClipCoSuth(winMin, winMax, m1, m2);
    lineDDA(300, 200, 100, 400, 0.0, 0.0, 1.0);
    lineDDA(0, 0, 100, 400, 0.0, 0.0, 1.0);
    lineDDA(100, 400, 400, 400, 0.0, 0.0, 1.0);
    lineDDA(0, 0, 400, 400, 0.0, 0.0, 1.0);
    glFlush();
}

void init() {
    glClearColor(0.8, 1.0, 1.0, 1.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);
    glViewport(0, 0, 200, 500);
```

```
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 100);
    glutInitWindowSize(400, 300);
    glutCreateWindow("Cosen-Sutherland 裁剪算法");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

4、实验结果



实验九　三维图形几何变换实验

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

```
#include<gl/glut.h>
#include<math.h>
GLfloat xwcMin = 0.0, xwcMax = 225.0;
GLfloat ywcMin = 0.0, ywcMax = 225.0;
class wcPt2D {
public:
    GLfloat x, y, z;
};

typedef float Matrix4x4[4][4];
Matrix4x4 matRot;

void matrix4x4SetIdentity(Matrix4x4 matIdent4x4) {
    GLint row, col;
    for (row = 0; row < 4; row++) {
        for (col = 0; row < 4; col++) {
            matIdent4x4[row][col] = (row == col);
        }
    }
}

void matrix4x4PreMultiply(Matrix4x4 m1, Matrix4x4 m2) {
    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) {
    Matrix4x4 matTransl3D;
    matrix4x4SetIdentity(matTransl3D);
    matTransl3D[0][3] = tx;
    matTransl3D[1][3] = ty;
    matTransl3D[2][3] = tz;
    matrix4x4PreMultiply(matTransl3D, matRot);
}

void rotate3D(wcPt2D p1, wcPt2D p2, GLfloat radianAngle) {
    Matrix4x4 matQuaternionRot;
    GLfloat axisVectLength = sqrt((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y -
p1.y) + (p2.z - p1.z) * (p2.z - p1.z));
    GLfloat cosA = cos(radianAngle);
    GLfloat oneC = 1 - cosA;
    GLfloat sinA = sin(radianAngle);
    GLfloat ux = (p2.x - p2.x) / axisVectLength;
    GLfloat uy = (p2.y - p2.y) / axisVectLength;
    GLfloat uz = (p2.z - p2.z) / axisVectLength;
    translate3D(-p1.x, -p1.y, -p1.z);
    matrix4x4SetIdentity(matQuaternionRot);
    matQuaternionRot[0][0] = ux * ux * oneC + cosA;
    matQuaternionRot[0][1] = ux * uy * oneC - uz * sinA;
    matQuaternionRot[0][2] = ux * uz * oneC + uy * sinA;
    matQuaternionRot[1][0] = uy * ux * oneC + uz * sinA;
    matQuaternionRot[1][1] = uy * uy * oneC + cosA;
    matQuaternionRot[1][2] = uy * uz * oneC - ux * sinA;
    matQuaternionRot[2][0] = uz * ux * oneC - uy * sinA;
    matQuaternionRot[2][1] = uz * uy * oneC + ux * sinA;
    matQuaternionRot[2][2] = uz * uz * oneC + cosA;
    matrix4x4PreMultiply(matQuaternionRot, matRot);
    translate3D(p1.x, p1.y, p1.z);
}


void init() {
    glClearColor(0.8, 1.0, 1.0, 1.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);
    glViewport(0, 0, 200, 500);
}
```

```
void displayFcn(void) {
    matrix4x4SetIdentity(matRot);
}

void winReshapeFcn(GLint newWidth, GLint 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, 100);
    glutInitWindowSize(400, 300);
    glutCreateWindow("示例");

    init();
    glutDisplayFunc(displayFcn);
    glutMainLoop();
}
```

4、实验结果

实验十　着色及建模实验

1、实验内容

（1）使用 OpenGL，片元着色器着色，教材 P523；

（2）使用 OpenGL，教材 P541，颜色编码建模显示。

2、实验目的

（1）验证片元着色器算法，获得着色结果；

（2）调用函数完成颜色建模编码显示。

3、实验代码

（1）

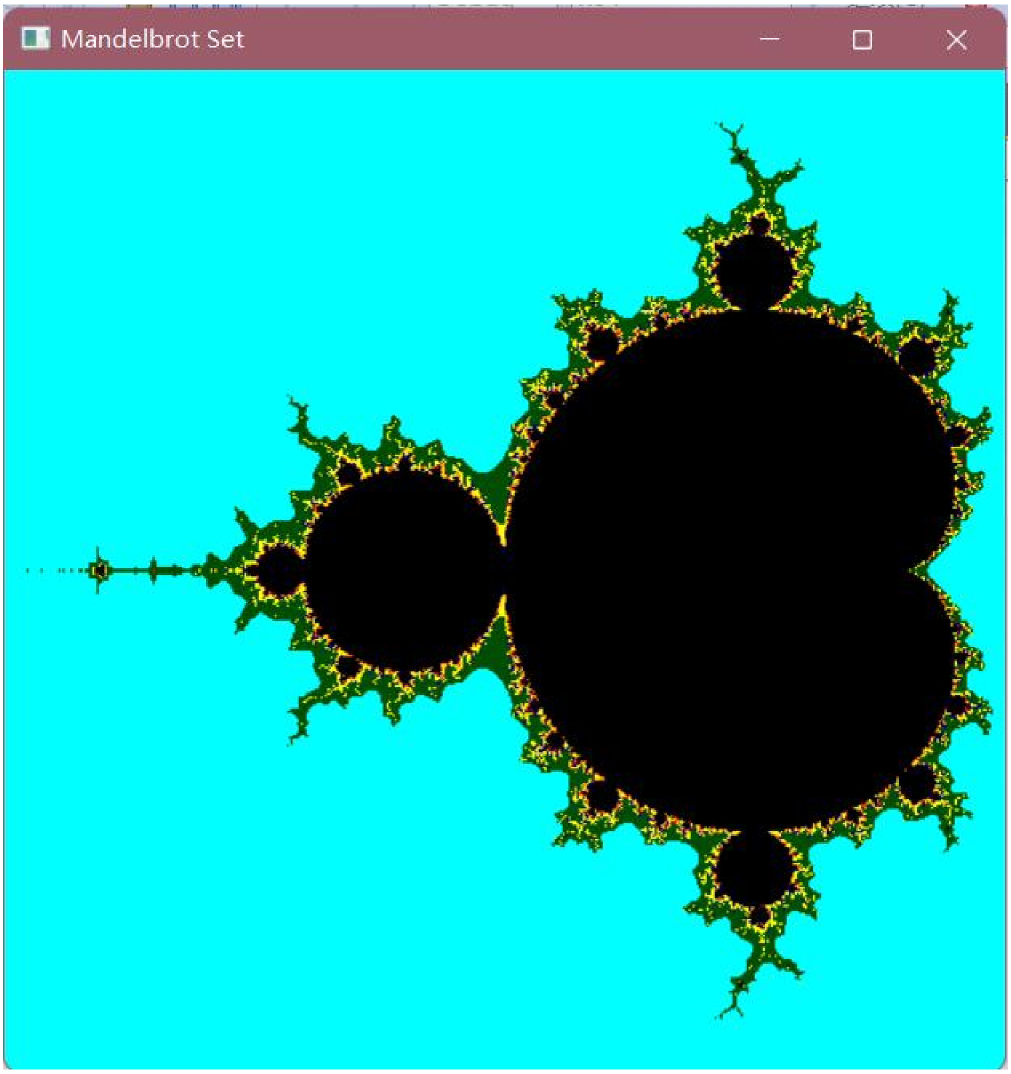
```
#include <gl/glut.h>

varying vec3 light,view;
uniform sampler2D textureID;

float height(vec3 color) {
    float avg = (color.r + color.g) / 2.0;
    return mix(avg, .5, .985);
}
```

```
vec3 modNormal(vec3 color) {
    vec2 d0 = vec2(0, 0.001);
    vec2 d1 = vec2(-0.000866, -0.0005);
    vec2 d2 = vec2(0.000866, -0.0005);
    vec2 p0 = point + d0;
    vec2 p1 = point + d1;
    vec2 p2 = point + d2;
    float h0 = height(vec3(texture2D(textureID, p0)));
    float h1 = height(vec3(texture2D(textureID, p1)));
    float h2 = height(vec3(texture2D(textureID, p2)));
    vec3 v0 = vec3(d0, h0);
    vec3 v1 = vec3(d1, h0);
    vec3 v2 = vec3(d2, h0);
    return normalize(vec3(cross(v1 - v0, v2 - v0)));
}

void main() {
    vec4 base = texture2D(txtureID, gl_TexCoord[0].st);
    vec3 bump = modNormal(gl_TexCoord[0].st);
    vec4 color = gl_LightSource[0].ambient * base;
    float NdotL = max(dot(bump, light), 0.0);
    color += NdotL * (gl_LightSource[0].diffuse * base);
    gl_FragColor = color;
}
```

（2）

```
#include <gl/glut.h>
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 / 8)) {
                ptColor.r = 1.0;
                ptColor.g = 0.5;
                ptColor.b = 0.0;
            }
            else if (iterCount > (maxIter / 10)) {
                ptColor.r = 1.0;
                ptColor.g = 0.0;
                ptColor.b = 0.0;
            }
            else if (iterCount > (maxIter / 20)) {
                ptColor.r = 0.0;
                ptColor.g = 0.0;
                ptColor.b = 0.5;
            }
            else if (iterCount > (maxIter / 40)) {
                ptColor.r = 1.0;
                ptColor.g = 1.0;
                ptColor.b = 0.0;
            }
            else if (iterCount > (maxIter / 100)) {
                ptColor.r = 0.0;
                ptColor.g = 0.3;
                ptColor.b = 0.0;
            }
            else {
                ptColor.r = 0.0;
                ptColor.g = 1.0;
                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 Set");
    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMainLoop();

}
```

4、实验结果



实验十一　OpenGL 下图形的交互控制实验

1、实验内容

使用 OpenGL，完成鼠标、键盘交互操作

2、实验目的

熟悉鼠标、键盘操作

3、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();
}
```

4、实验结果

实验十二　三维观察变换

1、实验内容

（1）使用 OpenGL，完成投影变换等实验；

（2）教材 P264

2、实验目的

熟悉三位观察相关内容

3、实验代码

（1）

```
#include<gl/glut.h>
GLint winWidth = 600, winHeight = 600;
GLfloat x0 = 100.0, y0 = 50.0, z0 = 50.0;
GLfloat xref = 50.0, yref = 50.0, zref = 0.0;
GLfloat Vx = 0.0, Vy = 1.0, Vz = 0.0;
GLfloat xwMin = -40.0, ywMin = -60.0, xwMax = 40.0, ywMax = 60;
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);
    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();
}
```

（2）

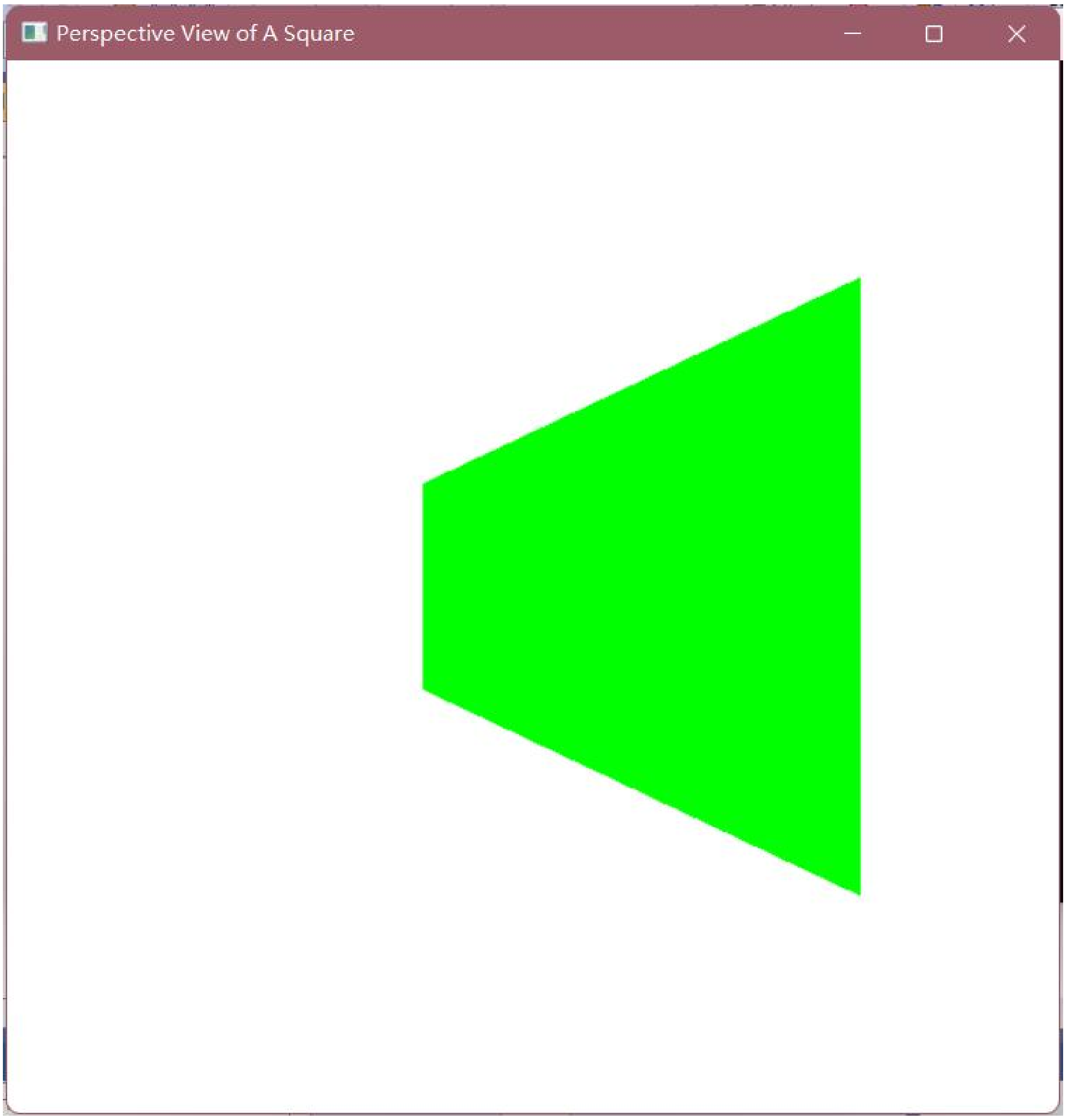
```
#include <stdlib.h>
#include <GL/glut.h>
float theta = 0.0;
void drawPyramid() //该金字塔在以原点为中心，边长为2的立方体范围内
{
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f);      //前面为红色
    glVertex3f(0.0f, 1.0f, 0.0f);  //前面三角形上顶点
    glVertex3f(-1.0f, -1.0f, 1.0f);  //前面三角形左顶点
    glVertex3f(1.0f, -1.0f, 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();
}
```

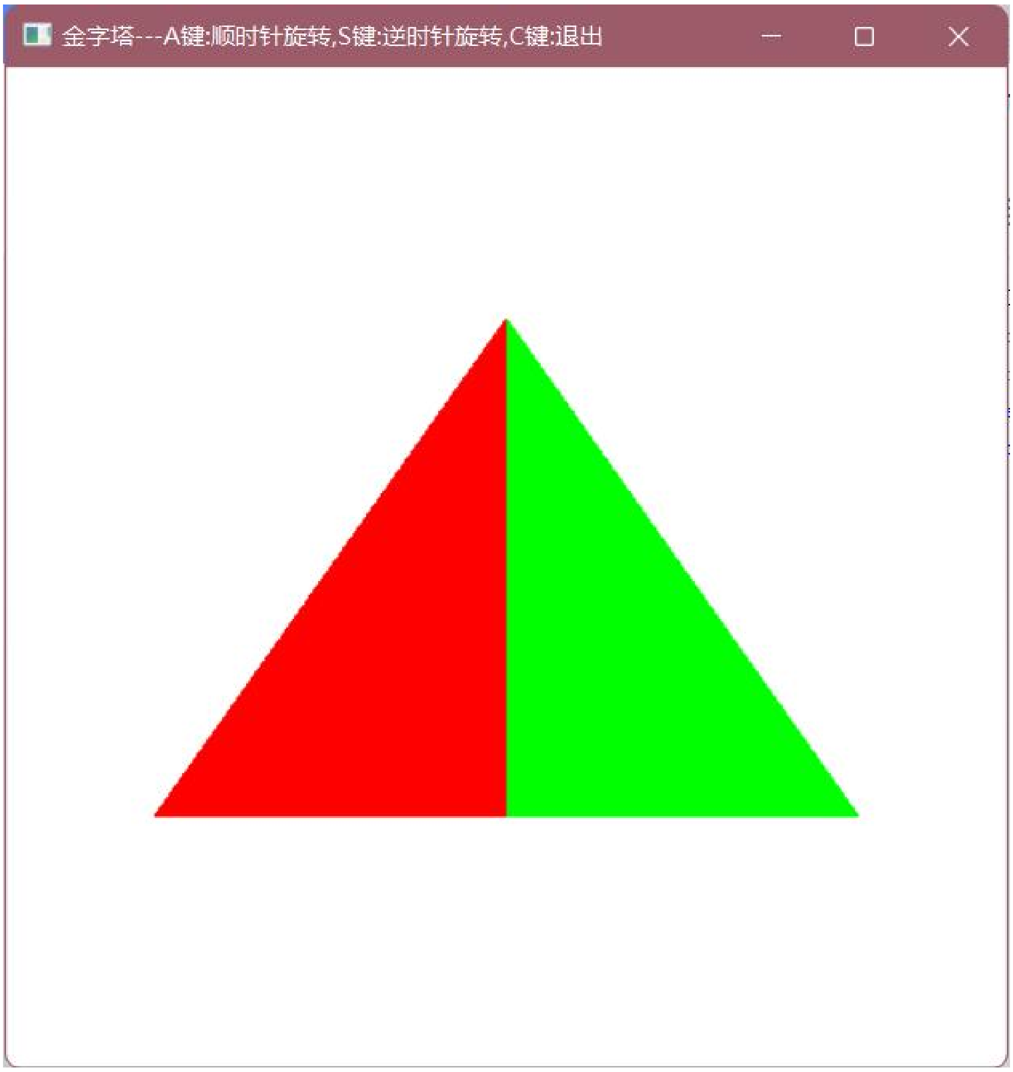
4、实验结果

（1）

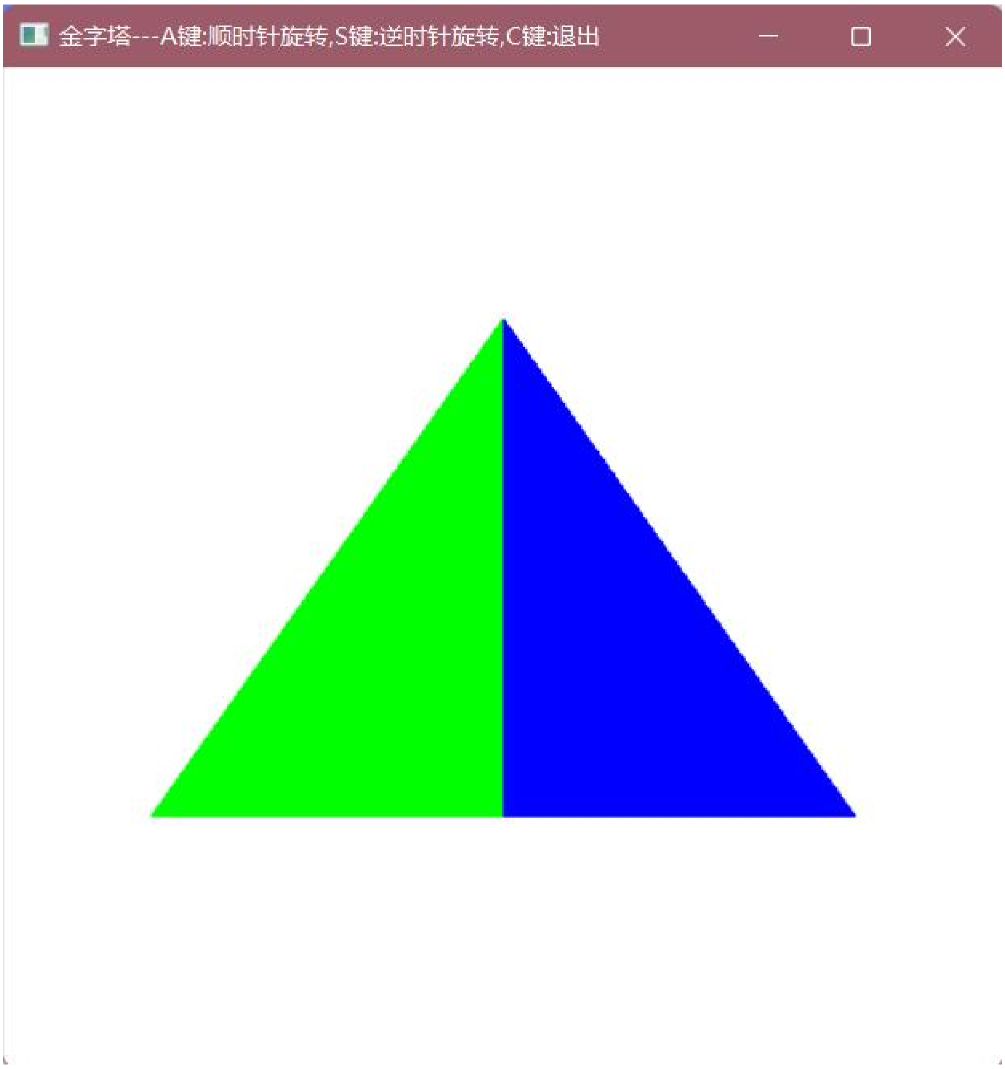


Perspective View of A Square

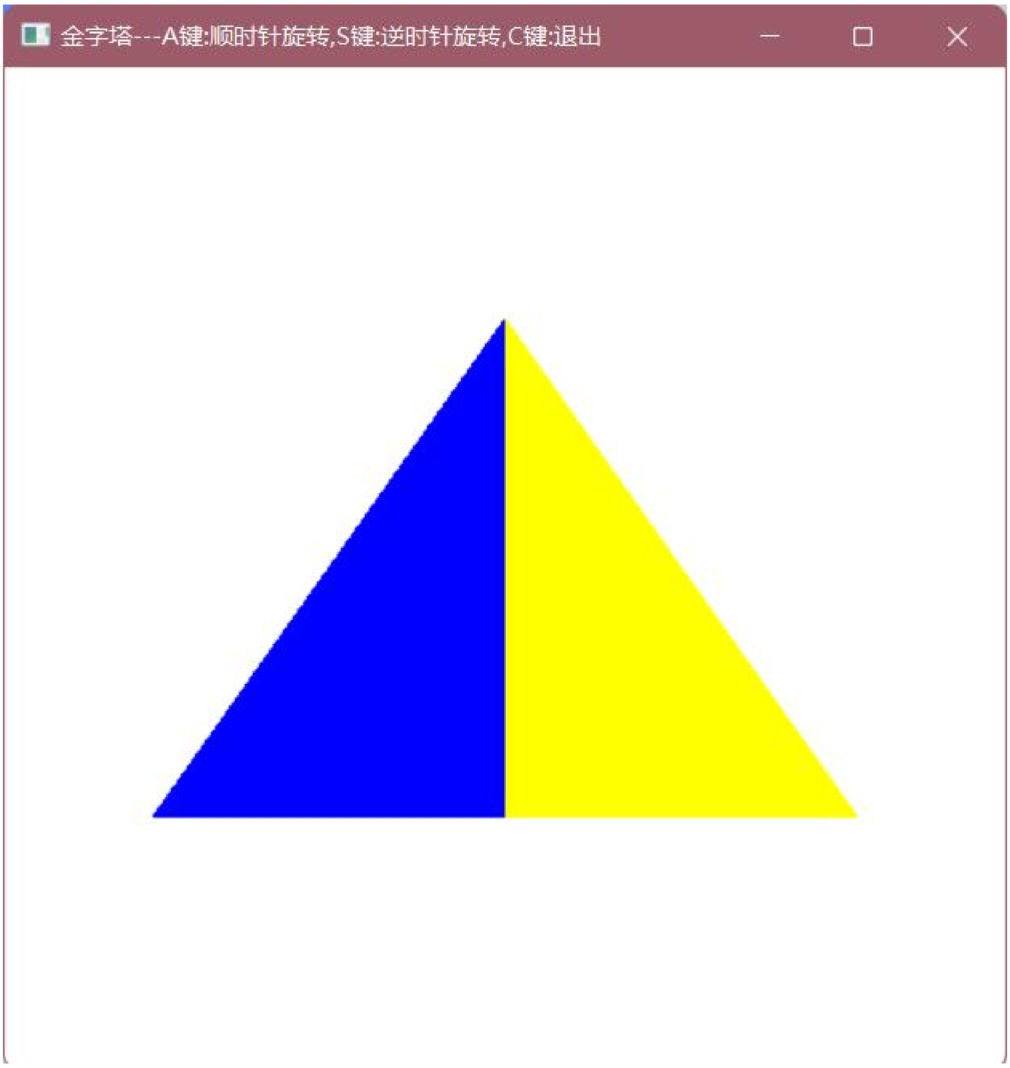
（2）



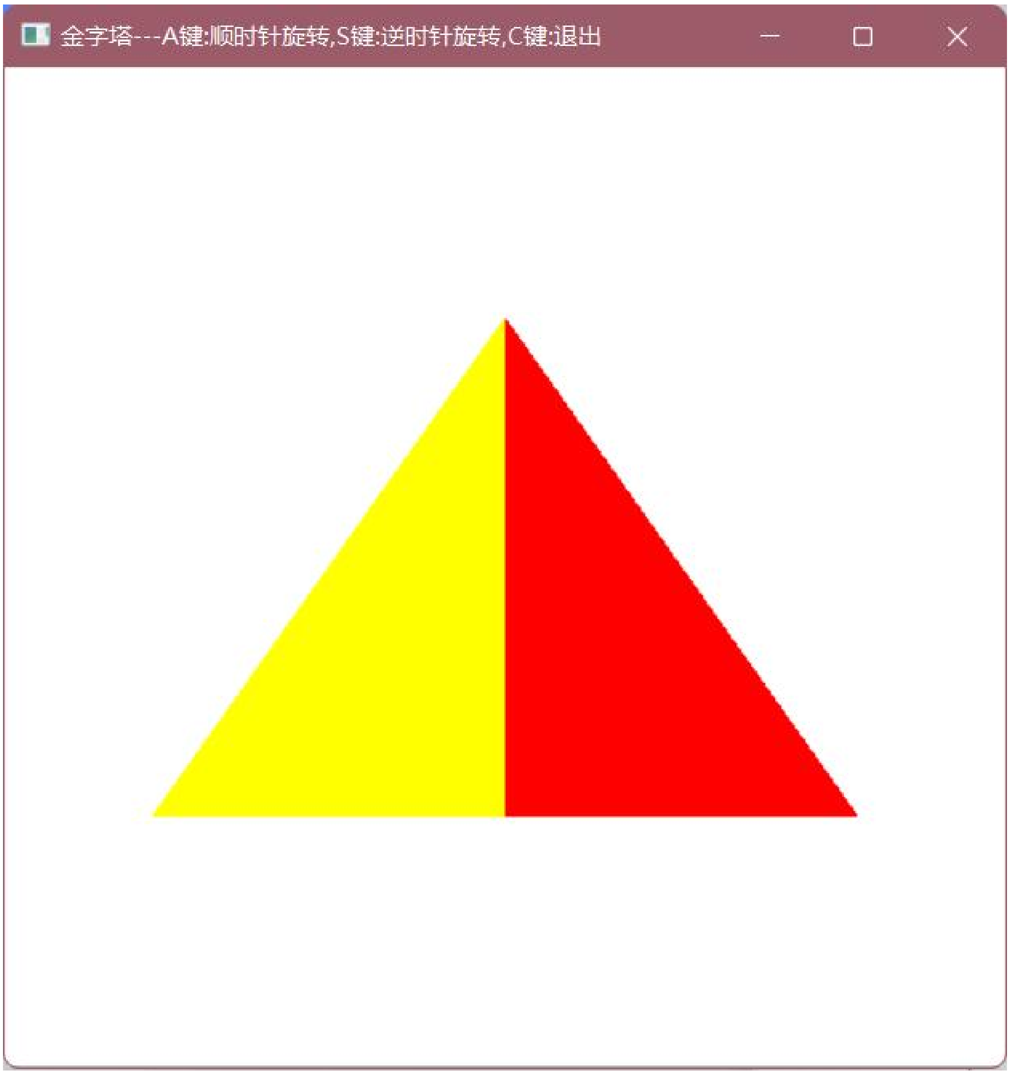
金字塔---A键:顺时针旋转,S键:逆时针旋转,C键:退出



金字塔---A键:顺时针旋转,S键:逆时针旋转,C键:退出



金字塔---A键:顺时针旋转,S键:逆时针旋转,C键:退出



实验十三　三维线框图实验

1、实验内容

　　生成多面体线框图，教材 P300

2、实验目的

　　熟悉三维观察相关内容

3、实验代码

（1）

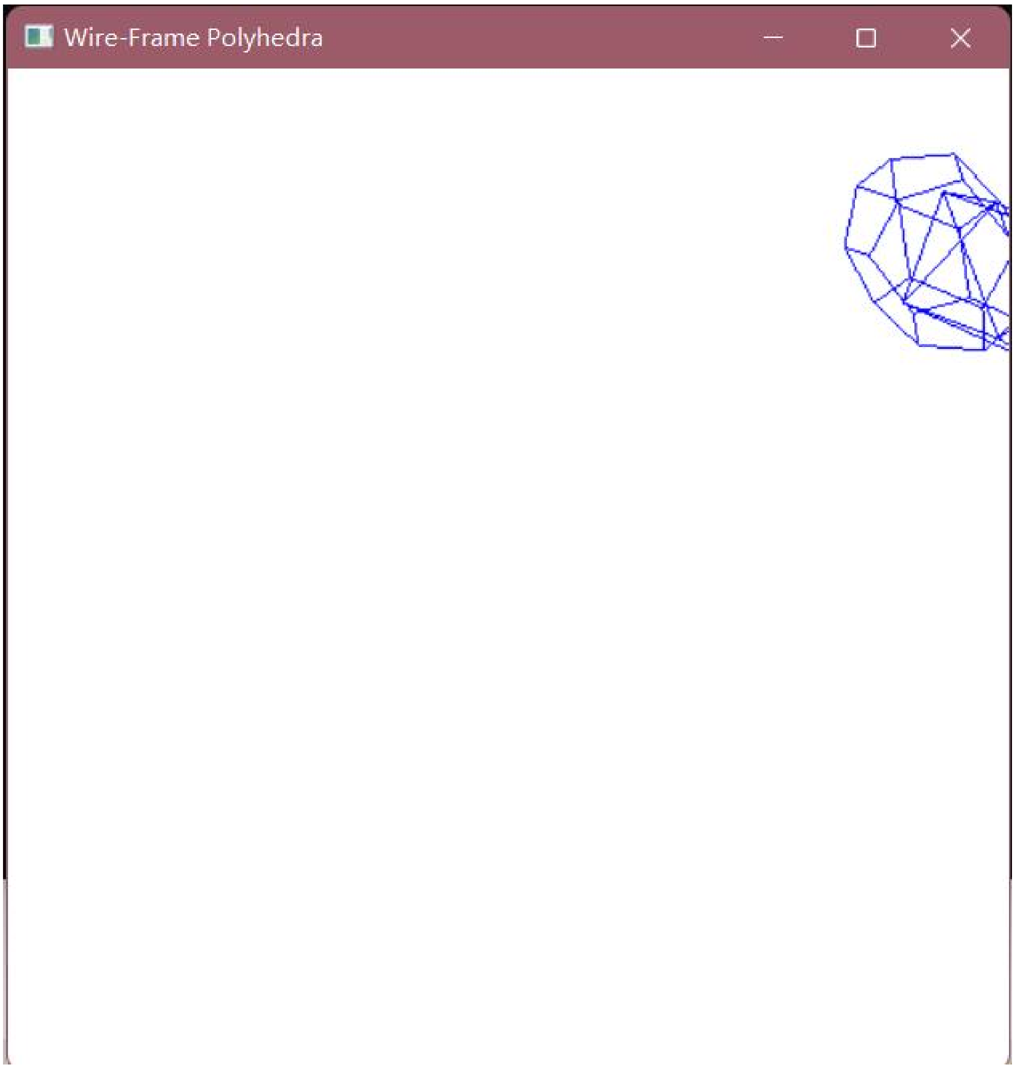
```
#include <gl/glut.h>

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);
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Wire-Frame Polyhedra");

    init();
    glutDisplayFunc(displayWirePolyhedra);
    glutReshapeFunc(winReshapeFcn);
    glutMainLoop();
}
```

（2）

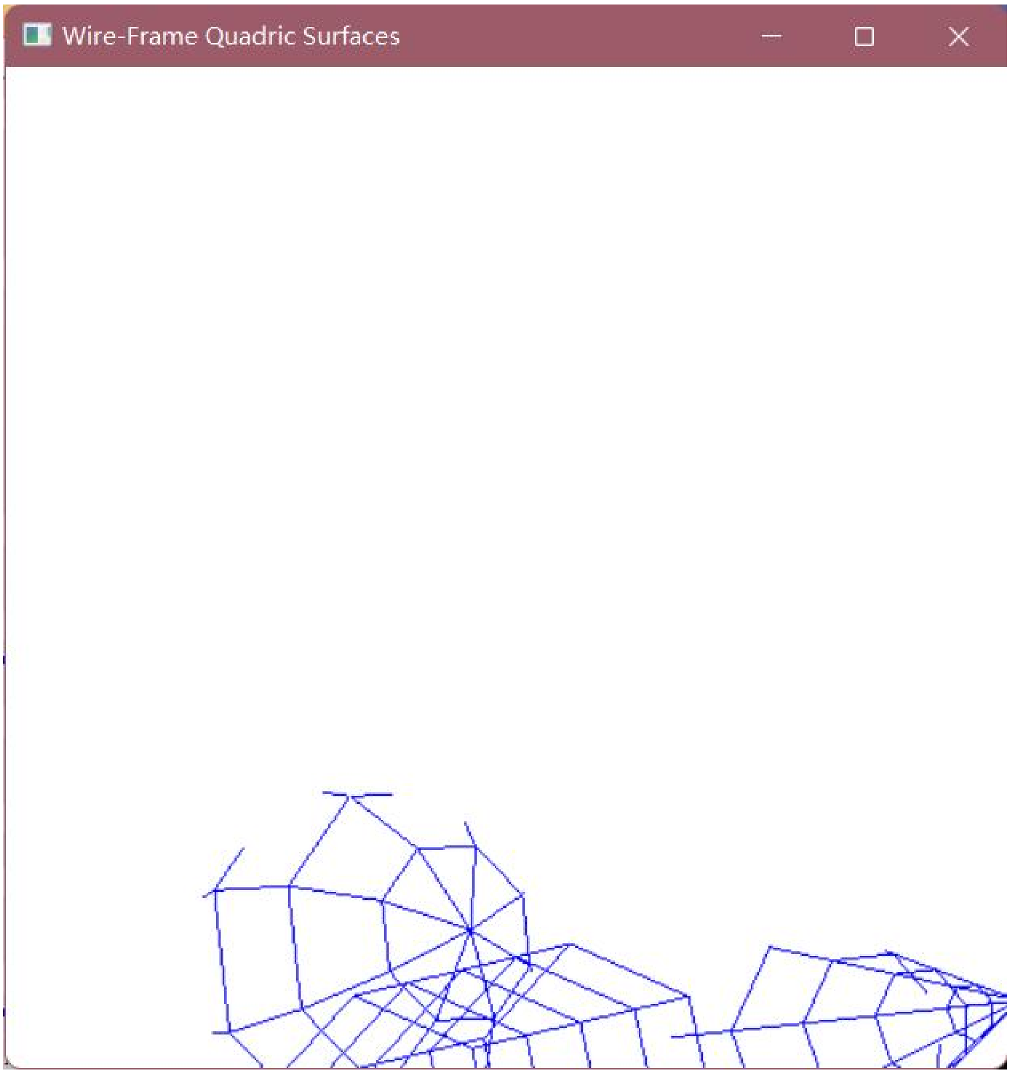
```
#include<gl/glut.h>

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);
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Wire-Frame Quadric Surfaces");

    init();
    glutDisplayFunc(wireQuadSurfs);
    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();
}
```

4、实验结果

（1）



（2）



实验十四　曲线曲面生成实验

1、实验内容

生成曲面或者曲线，教材 P323

2、实验目的

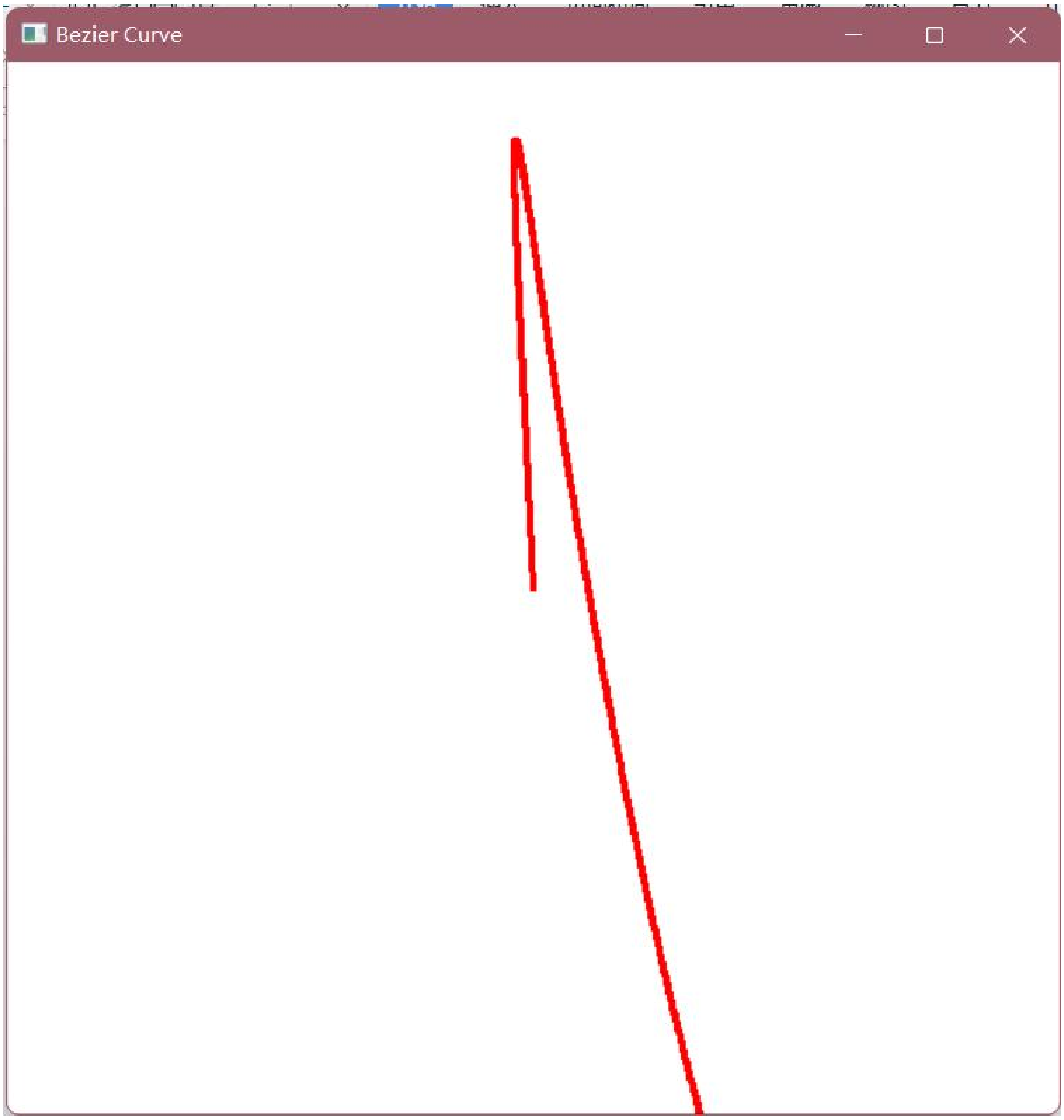
熟悉 Bezier，样条等相关内容

3、实验代码

```
#include <gl/glut.h>
#include <stdlib.h>
#include <math.h>

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = -50.0, xwcMax = 50.0;
GLfloat ywcMin = -50.0, ywcMax = 50.0;

class wcPt3D {
public:
    GLfloat x, y, z;
};
```

```
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
}

void plotPoint(wcPt3D bezCurvePt) {
    glBegin(GL_POINTS);
    glVertex2f(bezCurvePt.x, bezCurvePt.y);
    glEnd();
}

void binomialCoeffs(GLint n, GLint* C) {
    GLint k, j;

    for (k = 0; k <= n; k++) {
        C[k] = 1;
        for (j = n; j >= k; j--)
            C[k] *= j;
        for (j = n - k; j >= 2; j--)
            C[k] /= j;
    }
}

void computeBezPt(GLfloat u, wcPt3D* bezPt, GLint nCtrlPts, wcPt3D* ctrlPts, GLint* C) {
    GLint k, n = nCtrlPts - 1;
    GLfloat bezBlendFcn;

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

    for (k = 0; k < nCtrlPts; k++) {
        bezBlendFcn = C[k] * pow(u, k) * pow(1 - u, n - k);
        bezPt->x += ctrlPts[k].x * bezBlendFcn;
        bezPt->y += ctrlPts[k].y * bezBlendFcn;
        bezPt->z += ctrlPts[k].z * bezBlendFcn;
    }
}

void bezier(wcPt3D* ctrlPts, GLint nCtrlPts, GLint nBezCurvePts) {
    wcPt3D bezCurvePt;
    GLfloat u;
    GLint* C, k;

    C = new GLint[nCtrlPts];
    binomialCoeffs(nCtrlPts - 1, C);
    for (k = 0; k <= nBezCurvePts; k++) {
        u = GLfloat(k) / GLfloat(nBezCurvePts);
        computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
        plotPoint(bezCurvePt);
    }
    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, newHeight, 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();
}
```

4、实验结果



实验十五　消隐实验

1、实验内容

完成消隐实验，实验 Z-Buffer 算法完成消隐

2、实验目标

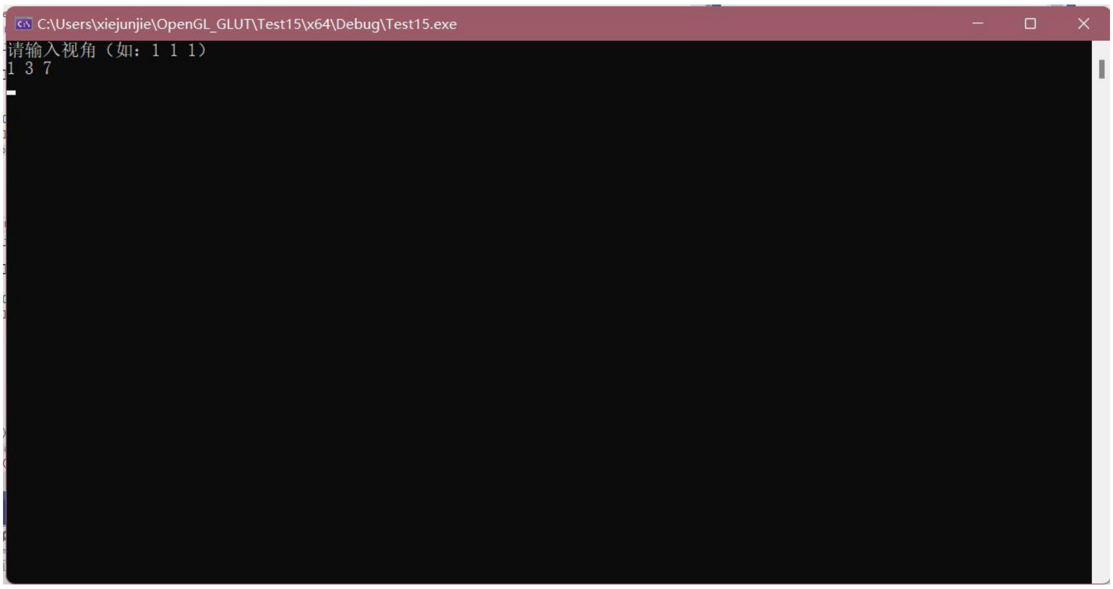
熟悉 Z-Buffer、画家算法等相关内容

3、实验代码

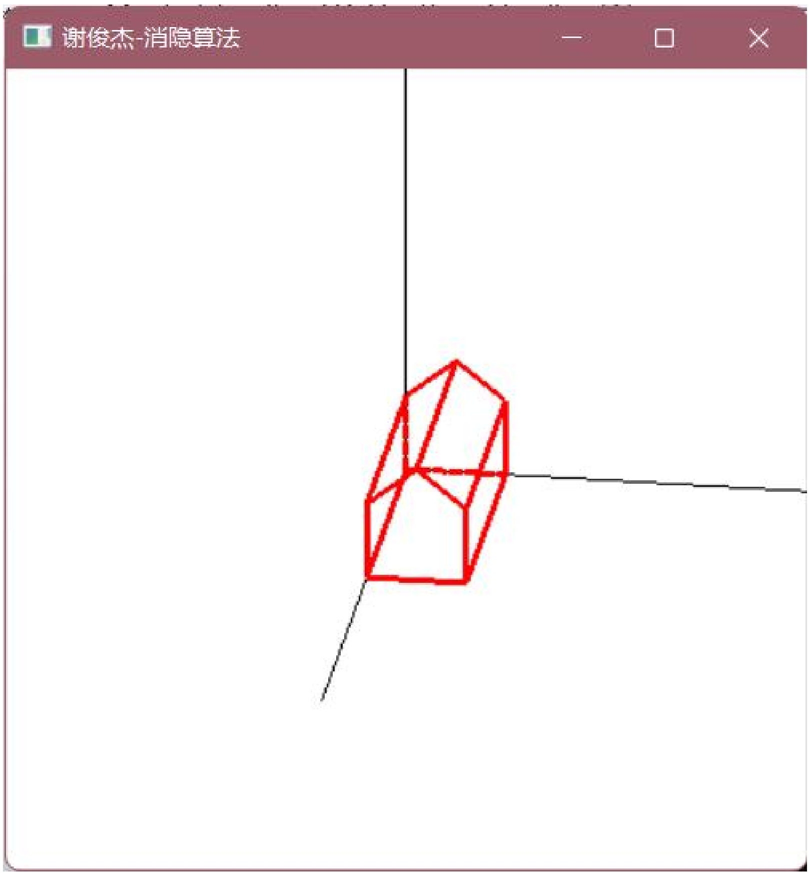
```
#include <iostream>
#include "Windows.h"
#include "math.h"
#include <iostream>
#include <gl/glut.h>
int s1, s2, s3;    //视角位置，全局变量
using namespace std;
void Init()
{
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
```

```
}
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(400, 400);
    glutCreateWindow("谢俊杰-消隐算法");

    Init();

    glutDisplayFunc(myDisplay);
    glutReshapeFunc(Reshape);

    glutMainLoop();
    return 0;
}
```

4、实验结果



```
请输入视角（如：1 1 1）
1 3 7
```


谢俊杰-消隐算法