计算机图形学实验

姓 名：李志辉

学 号：**20201120283**

专 业：计算机科学与技术

教 师：钱文华

目录

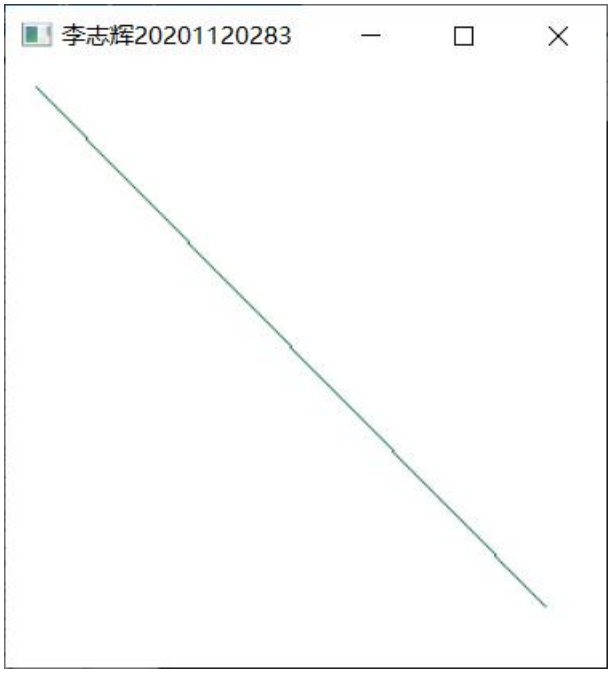
实验一 直线段生成算法 1
实验二 DDA 直线生成算法 3
实验三 Bresenham、 改进 Bresenham 算法生成直线段实验 8
实验四 填充算法实验 11
实验五 填充算法实验 19
实验六 填充算法实验 26
实验七 GLUT 鼠标函数实验、反走样技术 33
实验八 二维图像裁剪实验 42
实验九 三维图形几何变换实验 46
实验十 可编程着色实验 48
实验十一 交互控制实验 53
实验十二 三维观察实验 55
实验十三 多面体实验 57
实验十四 曲线曲面生成实验 62
实验十五 消隐实验 65

实验一 直线段生成算法

实验时间：2022 年 3 月 16 日
实验地点：信息学院 2202 机房
实验内容：熟悉 OPENGL，通过示例程序生成直线段
实验目的：安装 OPENGL，能编写代码运行，参考课本代码。
实验代码：

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

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();
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 100);
    glutCreateWindow("An Example OpenGL Program");

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

实验结果：



实验总结：

熟悉并安装了 OPENGL，成功生成了直线段。

实验二 DDA 直线生成算法

实验时间：2022 年 3 月 23 日

实验地点：信息学院 2204 机房

实验内容：熟悉 OPENGL，通过 DDA、中点算法生成直线段

实验目的：安装 OPENGL，能编写代码运行，参考课本代码。

实验代码：

2.1 DDA 算法

```
#include<GL\glut.h>
#include<iostream>
#include<cmath>
using namespace std;
//DDA 算法绘制直线
void DrowDDA(int x1, int y1, int x2, int y2)
{
    glColor3f(1.0, 0.0, 0.0);       //所要画线的颜色为红色
    glPointSize(3.0f);
    int m = 0;
    //选择两个点的坐标的距离的最大值
    if (abs(x2 - x1) >= abs(y2 - y1))
    {
        m = abs(x2 - x1);              // x 为计长方向
    }
    else
    {
        m = abs(y2 - y1);              // y 为计长方向
    }
    float dx = (float)(x2 - x1) / m;   // 当 x 为计长方向，dx = 1
    float dy = (float)(y2 - y1) / m;   // 当 y 为计长方向，dy = 1
    float x = x1;
    float y = y1;
    for (int i = 0; i < m; ++i)//循环设置定点
    {
        glBegin(GL_POINTS);//把每一个顶点做为一个定点处理，绘制 N 个定点
        glVertex2f((int)x, (int)y);//设置定点
        glEnd();
```

```
            glFlush();
            x += dx;
            y += dy;
        }
}

void display(void)
{
    // 用当前背景色填充窗口，如果不写这句会残留之前的图像
    glClear(GL_COLOR_BUFFER_BIT);
    int x1 = 0, y1 = 0, x2 = 500, y2 = 500;
    int x12 = 0, y12 = 0, x22 = 500, y22 = 500;
    DrowDDA(x1, y1, x2, y2);//使用 DDA 算法绘制直线
}
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);//初始化 glut 的库
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(200, 200);//窗口的位置
    glutInitWindowSize(400, 400);//窗口的大小
    glutCreateWindow("李志辉 20201120283");
    glutDisplayFunc(display);//调用函数
    glClearColor(1.0, 1.0, 1.0, 0.0);  //窗口的背景为白色
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 150.0);
    glutMainLoop();
    return 0;
}
```

2.2 中点画线算法

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


void MidPLine(int x0, int y0, int x1, int y1)
{
    glColor3f(0.0f, 0.0f, 1.0f);//设置颜色，蓝色
    glPointSize(3);//栅格化点，直径为 3

    int a, b, d, x, y, temp, tag = 0;
    if (abs(x1 - x0) < abs(y1 - y0))    //若斜率的绝对值大于 1，将坐标 x 和坐标 y 互换
    {
        temp = x0, x0 = y0, y0 = temp;
        temp = x1, x1 = y1, y1 = temp;
```

```
        tag = 1;
    }
    if (x0 > x1)//保证 x0<x1
    {
        temp = x0, x0 = x1, x1 = temp;
        temp = y0, y0 = y1, y1 = temp;
    }
    a = y0 - y1;//判别式的 a
    b = x1 - x0;//判别式的 b
    d = a + b / 2;//判别式的初值
    if (y0 < y1)//斜率为正
    {
        x = x0; y = y0;
        glBegin(GL_POINTS);//画起点
        glVertex2i(x, y);
        glEnd();
        while (x < x1)
        {
            if (d < 0) //判别式<0，取点 P2，增量为 a+b
            {
                x++; y++; d = d + a + b;
            }
            else//判别式>=0，取点 P1，增量为 a
            {
                x++; d += a;
            }
            if (tag)//斜率大于 1
            {
                glBegin(GL_POINTS);
                glVertex2i(y, x);
                glEnd();
            }
            else
            {
                glBegin(GL_POINTS);
                glVertex2i(x, y);
                glEnd();
            }
        }  /* while */
    }
    else//斜率为负(y0>=y1)
    {
        x = x1;
        y = y1;
```

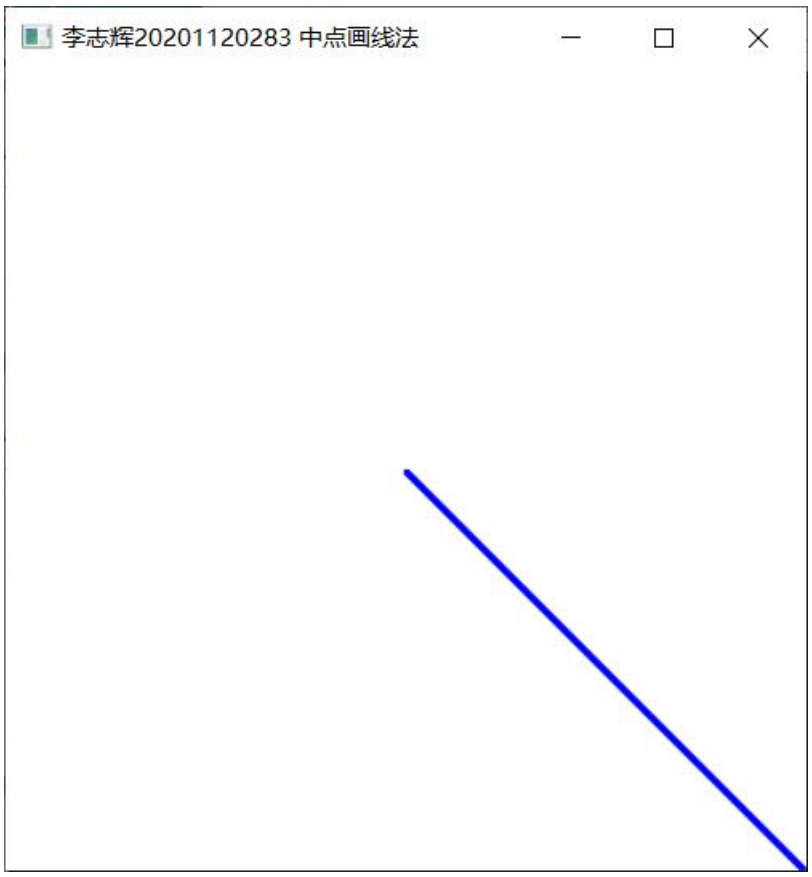
```
        glBegin(GL_POINTS);//画起点
        glVertex2i(x, y);
        glEnd();
        while (x > x0)
        {
            if (d < 0)//判别式<0，增量为-a+b
            {
                x--; y++; d = d - a + b;
            }
            else//判别式>=0，增量为-a
            {
                x--; d -= a;
            }
            if (tag)//斜率大于 1
            {
                glBegin(GL_POINTS);
                glVertex2i(y, x);
                glEnd();
            }
            else
            {
                glBegin(GL_POINTS);
                glVertex2i(x, y);
                glEnd();
            }
        }  /* while */
    }
}
void myDisplay()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);//清除颜色，白色
    glClear(GL_COLOR_BUFFER_BIT);//消除缓冲区，使用上述清除颜色消除


    //MidPLine(0, 0, 200, 200);
    //MidPLine(200, 200, 0, 0);
    //MidPLine(0, 200, 100, 100);
    MidPLine(200, 200, 400, 0);

    glFlush();//强制刷新
}

void Reshape(int w, int h)
{
```

```
    glViewport(0, 0, (GLsizei)w, (GLsizei)h); //定义视口大小
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);//使左下角坐标为（0，0），右上角坐标为（w,h）
}

void main(int argc, char* argv[])
{
    glutInit(&argc, argv);//初始化 GLUT
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);//显示模式
    glutInitWindowPosition(100, 100);//窗口位置,窗口左上角在屏幕的坐标
    glutInitWindowSize(400, 400);//窗口大小
    glutCreateWindow("中点画线法");//创建窗口，参数是窗口的标题
    glutDisplayFunc(myDisplay);//告诉 GLUT 哪个函数负责绘图，即注册一个绘图函数 myDisplay
    glutReshapeFunc(Reshape); //窗口发生改变时，使用什么函数进行重绘
    glutMainLoop();//处理永不结束的循环监听
}
```

实验结果：

2.1 DDA 算法

2.2 中点画线算法



实验总结：学会了通过 DDA、中点画线算法生成直线段

实验三 Bresenham、 改进 Bresenham 算法生成直线段实验

实验时间：2022 年 3 月 30 日

实验地点：信息学院 2204 机房

实验内容：熟悉 OPENGL，通过 Bresenham 中点、改进 Bresenham 算法生成直线段

实验目的：安装 OPENGL，能编写代码运行，参考课本代码。

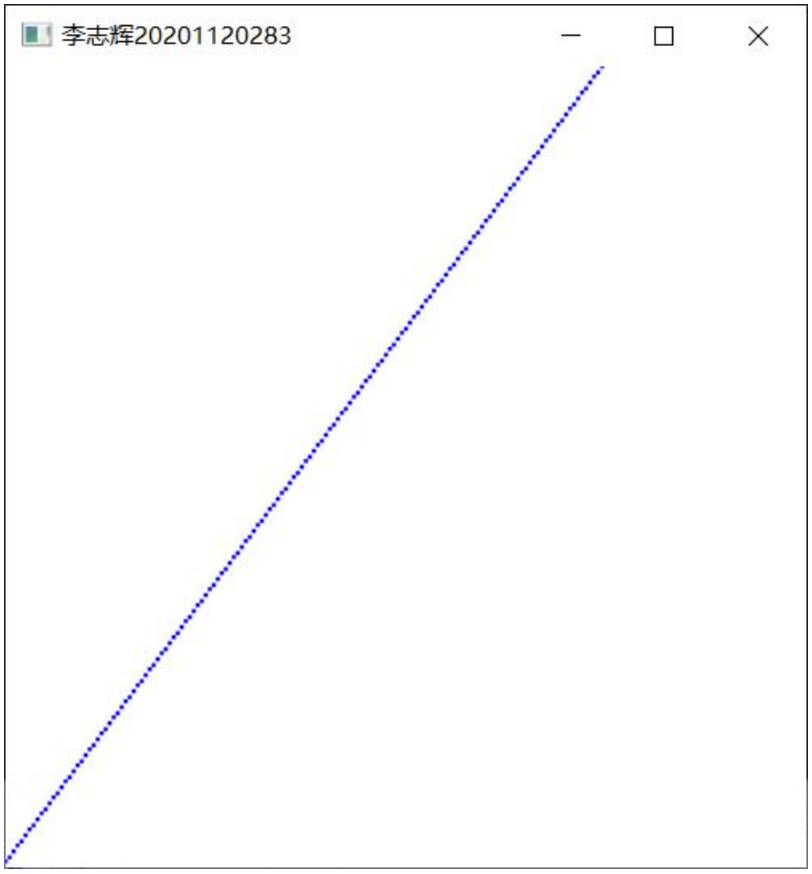
实验代码：

```
#include<GL\glut.h>
#include<iostream>
#include<cmath>
using namespace std;
void swapvalue(int* a, int* b);//函数的预定义
```

```
void DrowBres(int x1, int y1, int x2, int y2)
{
    glColor3f(0.0, 0.0, 1.0);       // 蓝色
    glPointSize(2.0f);
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    // 两点重合
    if (dx == 0 && dy == 0)
    {
        glBegin(GL_POINTS);
        glVertex2f(x1, y1);
        glEnd();
        glFlush();
        return;
    }
    int flag = 0;       // 将斜率变换到 0 <= |k| <= 1 区间
    if (dx < dy)
    {
        flag = 1;
        swapvalue(&x1, &y1);//交换两个变量的值
        swapvalue(&x2, &y2);//交换两个变量的值
        swapvalue(&dx, &dy);//交换两个变量的值
    }
    int tx = (x2 - x1) > 0 ? 1 : -1;
    int ty = (y2 - y1) > 0 ? 1 : -1;
    int curx = x1;
    int cury = y1;
    int dS = 2 * dy;
    int dT = 2 * (dy - dx);
    int d = dS - dx;
    while (curx != x2)
    {
        if (d < 0)
            d += dS;
        else
        {
            cury += ty;
            d += dT;
        }
        if (flag)
        {
            glBegin(GL_POINTS);
            glVertex2f(cury, curx);//画点
            glEnd();
```

```
                glFlush();
            }
            else
            {
                glBegin(GL_POINTS);
                glVertex2f(curx, cury);//画点
                glEnd();
                glFlush();
            }
            curx += tx;
        }
}void swapvalue(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void display(void)
{
    // 用当前背景色填充窗口，如果不写这句会残留之前的图像
    glClear(GL_COLOR_BUFFER_BIT);
    int x1 = 0, y1 = 0, x2 = 500, y2 = 500;
    int x12 = 0, y12 = 0, x22 = 500, y22 = 500;
    DrowBres(x12, y12, x22, y22);//使用 Bresenham 画直线
}
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);//初始化 glut 的库
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);//使用单缓存或者 RGB moshi
    glutInitWindowPosition(200, 200);//窗口的位置
    glutInitWindowSize(400, 400);//窗口的大小
    glutCreateWindow("李志辉 20201120283");
    glutDisplayFunc(display);//调用函数
    glClearColor(1.0, 1.0, 1.0, 0.0);  //窗口的背景为白色
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 150.0);
    glutMainLoop();//让 glut 程序进入事件循环。在一个 glut 程序中最多只能调用一次。一旦调
用，会直到程序结束才返回。
    return 0;
}
```

实验结果：



实验总结：学会了通过 Bresenham 中点、改进 Bresenham 算法生成直线段。

实验四 填充算法实验

实验时间：2022 年 4 月 6 日

实验地点：信息学院机房 2204

实验内容 1：教材 P66，填充六边形

实验内容 2：使用 opengl，用扫描线填充算法填充多边形

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。

实验代码：

4.1 P66，填充六边形

```
#include<GL/glut.h>
#include<stdlib.h>
#include<math.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 regHexagon(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(regHexagon);
    glutReshapeFunc(winReshapeFcn);

    glutMainLoop();
}
```

4.2 用扫描线填充算法填充多边形

```
#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;
```

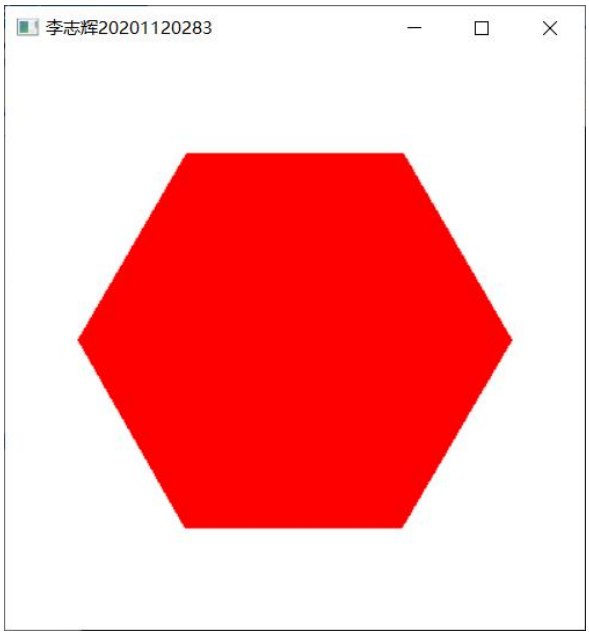
```
    }
    glClear(GL_COLOR_BUFFER_BIT);         //赋值的窗口显示.
    glColor3f(1.0, 0.0, 0.0);              //设置直线的颜色红色
    glBegin(GL_POINTS);
    /******扫描并建立 NET 表************************************************************/
    for (i = 0; i <= MaxY; i++)
    {
        for (int j = 0; j < POINTNUM; 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;
        }
        //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;
```

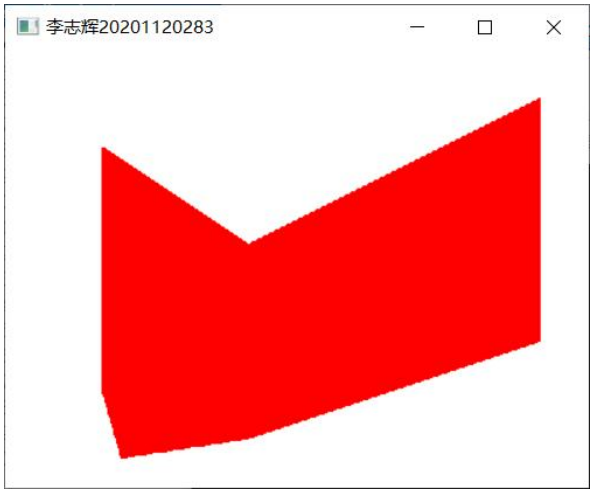
```
            p = s;
            q = pAET;
        }
        /******配对填充颜色
*****************************************************************/

        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
}
```

实验结果：

4.1 P66，填充六边形



4.2 用扫描线填充算法填充多边形



实验总结：通过本节课的学习，我验证了扫描线填充算法，并完成了指定任意的多边形边数，填充多边形。

实验五 填充算法实验

实验时间：2022 年 4 月 13 日

实验地点：信息学院机房 2202

实验内容 1：圆扫描转换

实验目的 1：输入圆的半径，画出圆。

实验内容 2：种子点填充算法

实验目的 2：输入多边形，种子点位置，填充多边形。

实验代码：

5.1 输入圆的半径，画出圆。

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

const int n = 50;
const GLfloat R = 0.9f;
const GLfloat PI = 3.24250265357f;
void myDisplay(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    for (int i = 0; i < n; i++) {
        glVertex2f(R * cos(2 * PI / n * i), R * sin(2 * PI / n * i));
    }
    glEnd();
    glFlush();
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("李志辉 20201120283");
    glutDisplayFunc(&myDisplay);
    glutMainLoop();
    return 0;
```

```
}
```

5.2 输入多边形，种子点位置，填充多边形

```
#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;
                }
            }
```

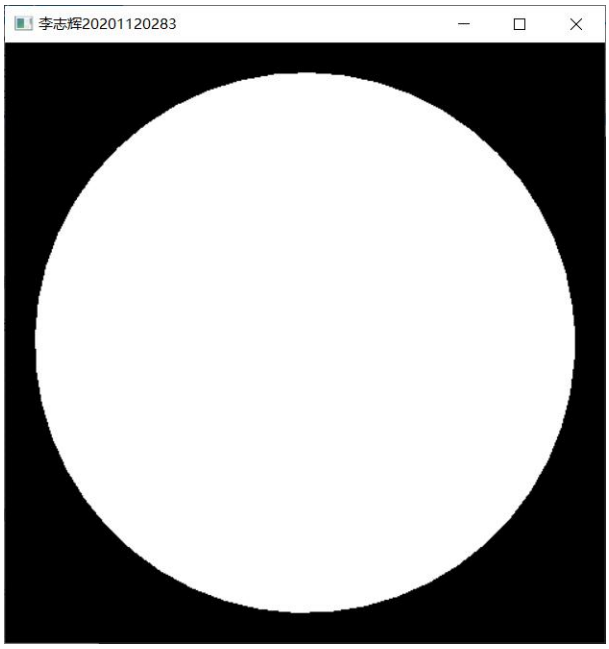
```
        }
    }
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i <= MaxY; i++) {
        AET* p = new AET;
        p = pAET->next;
        AET* n = new AET;
        //将新边表中的活性边按照从左到右的顺序排序
        if (pNET[i]->next && pNET[i]->next->next) {
            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
```

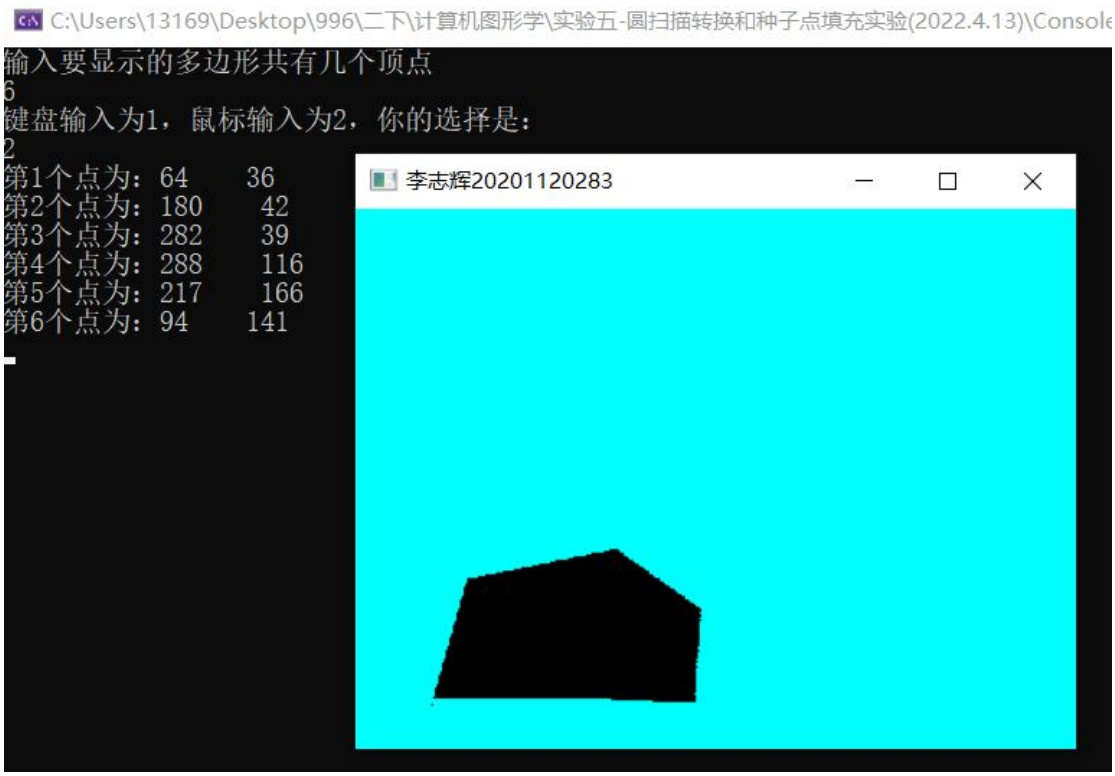
```
    cout << "输入要显示的多边形共有几个顶点" << endl;
    cin >> n;
    cout << "键盘输入为 1，鼠标输入为 2，你的选择是：" << endl;
    int x;
    cin >> x;
    return x;
}

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("李志辉 20201120283");//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();
}
```

实验结果：

5.1 输入圆的半径，画出圆。



5.2 输入多边形，种子点位置，填充多边形



实验总结：完成了圆扫描转换以及种子点填充算法。

实验六 填充算法实验

实验时间：2022 年 4 月 20 日

实验地点：信息学院机房

实验内容：教材 P161，二维几何变换算法（平移、比例、旋转、对称）

实验目的：验证二维几何变换，熟悉变换矩阵；

实验代码：

6.1 教材 P161 二维几何变换程序

```
#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();
}
```

6.2 额外的练习程序

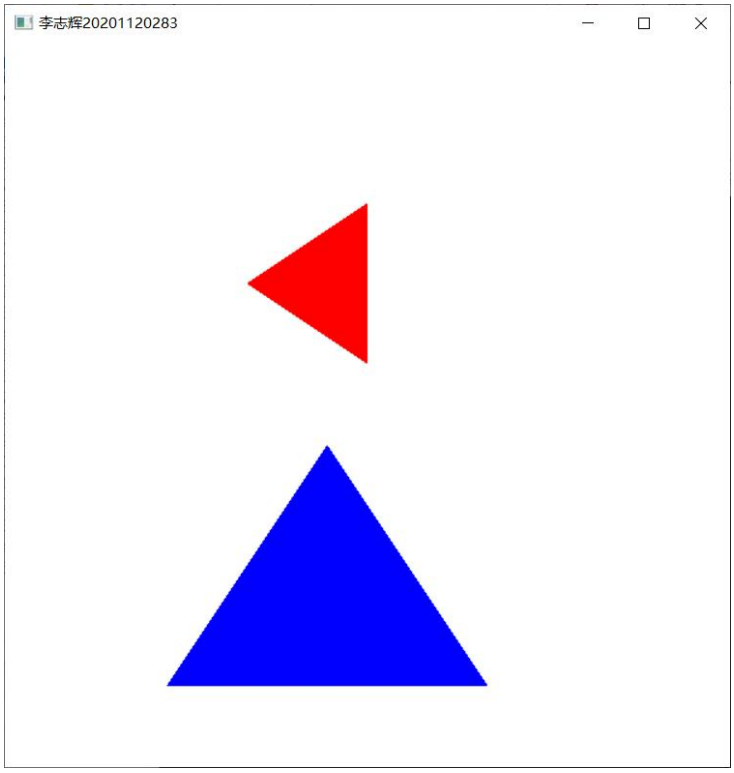
```
#include <gl\glut.h>
#include <iostream>
using namespace std;

//transilate 时，x，y 上的增量
float i = 0, j = 0;
//rotate 时的角度增量
GLint angle = 0;
//scale 时，x,y 上的增量
float s1 = 1, s2 = 1;

void init()
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 600.0, 0.0, 600.0);
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef(300 + i, 300 + j, 0);
    glRotatef(angle % 360, 0, 0, 1);
    glScalef(s1, s2, 0);

    glColor3f(0, 0.6, 0.5);
    glRectf(-100, 100, 100, -100);

    glFlush();
    glutSwapBuffers();
}
```

```
void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'w':
        j = j + 5; break;
    case 's':
        j = j - 5; break;
    case 'a':
        i = i - 5; break;
    case 'd':
        i = i + 5; break;
    case 'q':
        angle += 10; break;
    case 'e':
        angle -= 10; break;
    case 'z':
        s1 += 0.1; break;
    case 'x':
        s2 += 0.1; break;
    case 'c':
        s1 -= 0.1; break;
    case 'v':
        s2 -= 0.1; break;
    }
    glutPostRedisplay();
}

void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(400, 50);
    glutInitWindowSize(400, 400);
    glutCreateWindow("李志辉 20201120283： 二维几何变换");

    init();

    glutDisplayFunc(display);

    cout << " w,s,a,d键 分别为上下左右平移" << endl;
    cout << " q,e键 分别为逆时针旋转，顺时针旋转" << endl;
    cout << " z,x,c,v键 分别为长宽的放大缩小" << endl;
    glutKeyboardFunc(keyboard);
```

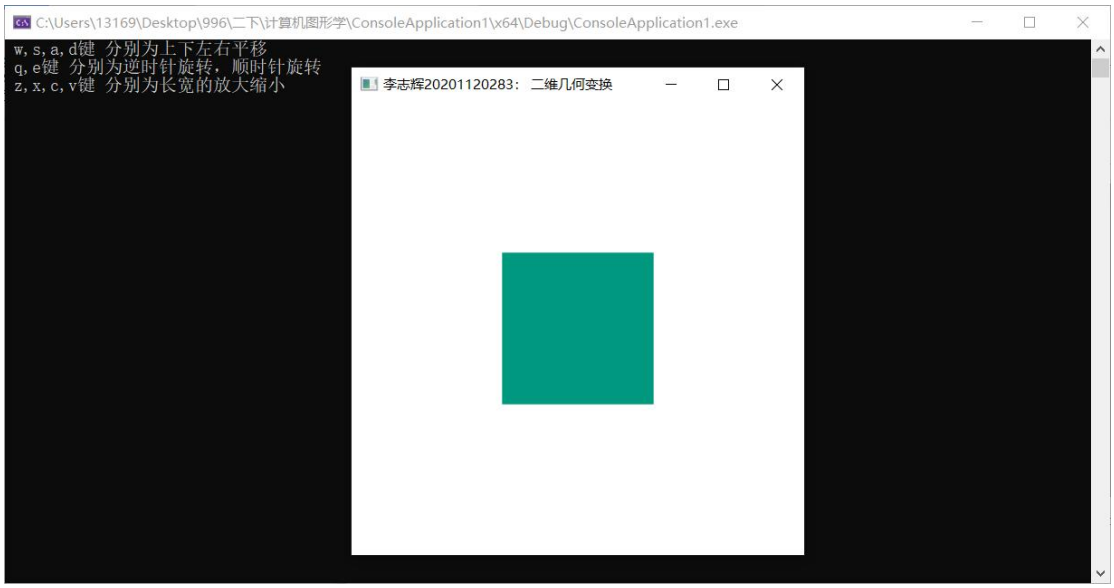
```
    glutMainLoop();
}
```

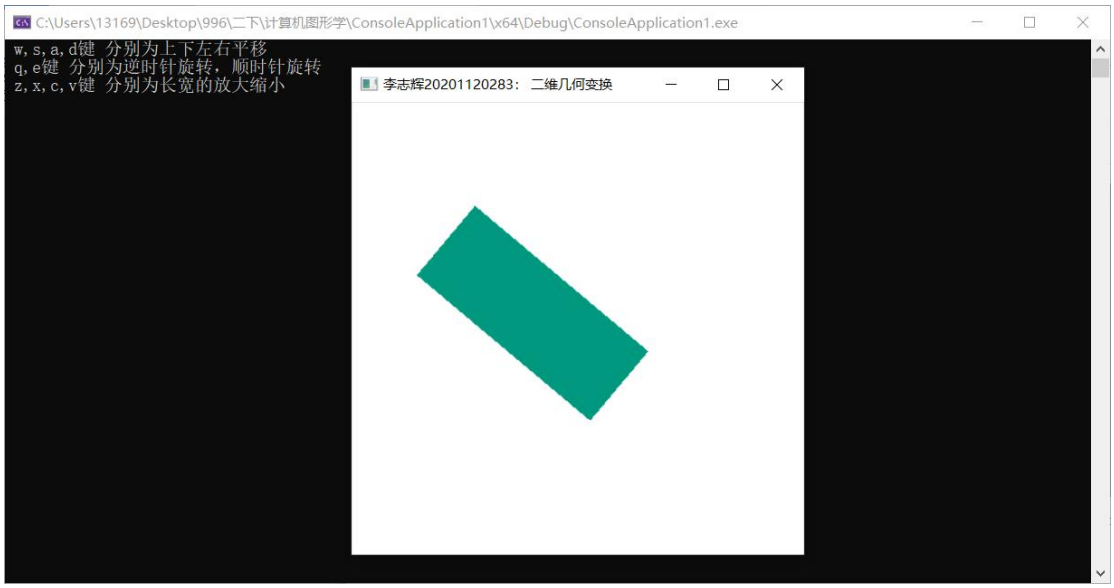
实验结果：

6.1 教材 P161 二维几何变换程序



6.2 额外的练习程序





实验总结：通过本次实验明白了二维几何变换的原理与过程，并完成了一个简单的程序。

实验七 GLUT 鼠标函数实验、反走样技术

实验时间：2022 年 4 月 26 日

实验地点：信息学院机房

实验内容 1：教材 P458，GLUT 鼠标函数

实验目的：调用鼠标函数完成相应功能，2-3 个程序。

实验内容 2：使用 opengl，实现任一反走样技术。

实验代码：

7.1 教材 P458 鼠标函数程序

```
#include "stdafx.h"
#include<GL/glut.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<iostream>
#include<cmath>
```

```
#include<vector>

GLsizei winWidth = 400, winHeight = 300;

void init(void) {
     glClearColor(0.0, 0.0, 1.0, 1.0);
     glMatrixMode(GL_PROJECTION);
     gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}

void displayFcn(void) {
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(1.0, 0.0, 0.0);
     glPointSize(30.0);
}

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("李志辉 20201120283");
```

```
    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMouseFunc(mousePtPlot);
    glutMainLoop();
}
```

7.2 其余鼠标函数程序一

```
#include "stdafx.h"
#include<GL/glut.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<iostream>
#include<stdio.h>
#include<stdlib.h>

GLsizei winWidth = 400, winHeight = 300;
GLint endPtCtr = 0;

class scrPt {
public:
    GLint x, y;
};

void init(void) {
    glClearColor(0.0, 0.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 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 yMosue) {
    static scrPt endPt1, endPt2;
    if (endPtCtr == 0) {
        if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
            endPt1.x = xMouse;
            endPt1.y = winHeight - yMosue;
            endPtCtr = 1;
        }
        else
            if (button == GLUT_RIGHT_BUTTON)
                exit(0);
    }
    else
        if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
            endPt2.x = xMouse;
            endPt2.y = winHeight - yMosue;
            drawLineSegment(endPt1, endPt2);

            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("李志辉 20201120283");

    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMouseFunc(polyline);
```

```
    glutMainLoop();
}
```

7.3 其余鼠标函数程序二

```
#include<iostream>
#include<math.h>
#include<windows.h>
#include<gl/glut.h>

// myInit
void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.5f, 0.4f, 0.9f);
    glPointSize(5.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluOrtho2D(0.0, 640, 0.0, 480);
}
///myDisplay//
void myDisplay()
{
    glBegin(GL_POINTS);
    glVertex2i(1, 1);
}
/// myMouse///
void myMouse(int button, int state, int x, int y)
{
    if (state == GLUT_DOWN)
    {
        glBegin(GL_POINTS);
        glVertex2i(x, 480 - y);
        glEnd();
        glFlush();
    }
    else if (button == GLUT_RIGHT_BUTTON)
    {
        glClearColor(0.8, 0.6, 0.7, 0.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
    }
}
///main///
void main(int argc, char** argv)
```

```
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("mouse");

    myInit();
    glutDisplayFunc(myDisplay);
    glutMouseFunc(myMouse);
    glutMainLoop();
}
```

7.4 反走样程序

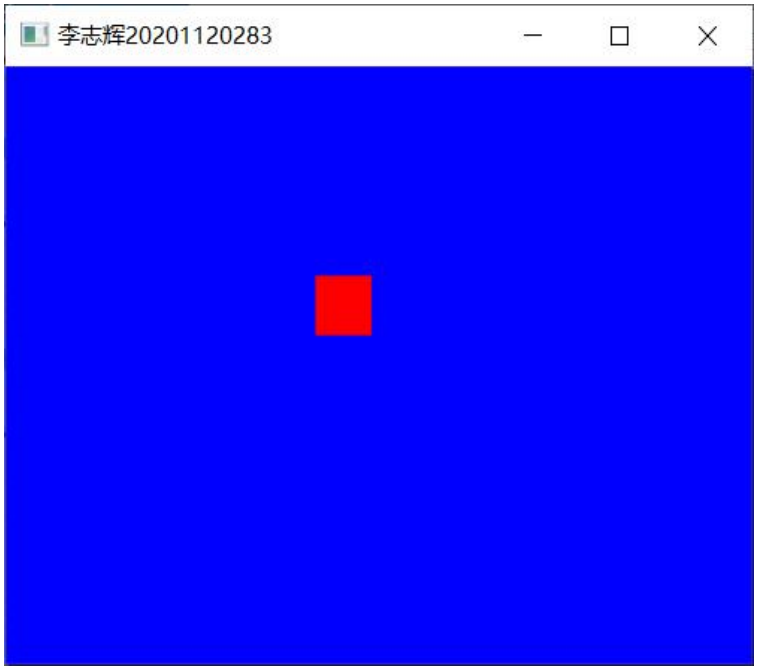
```
#include <GL/glut.h>
#include <stdio.h>

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("李志辉 20201120283");
    init();
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

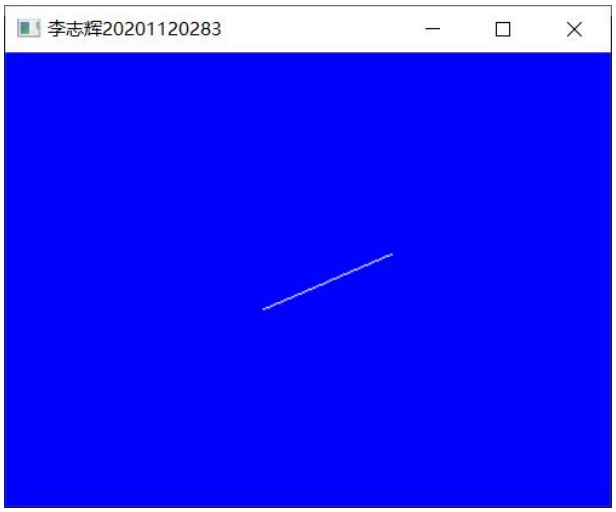
实验结果：

7.1 教材 P458 鼠标函数程序



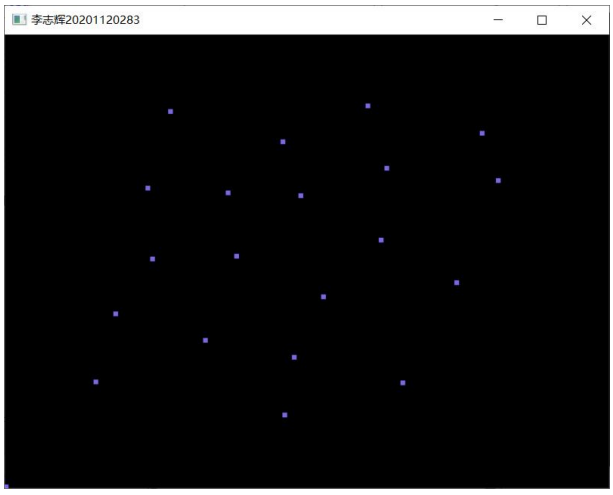
鼠标单击在鼠标位置生成红色正方形

7.2 其余鼠标函数程序



鼠标点击生成线段

7.3 其余鼠标函数程序二



鼠标点击生成点且留存

7.4 反走样程序

其中一条经过反走样处理

实验总结：通过本次实验学会并练习了一些通过鼠标控制的程序，也了解了反走样技术，并完成了一个简单的反走样程序。

实验八 二维图像裁剪实验

实验时间：2022 年 5 月 3 日

实验地点：信息学院机房

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

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

实验代码：

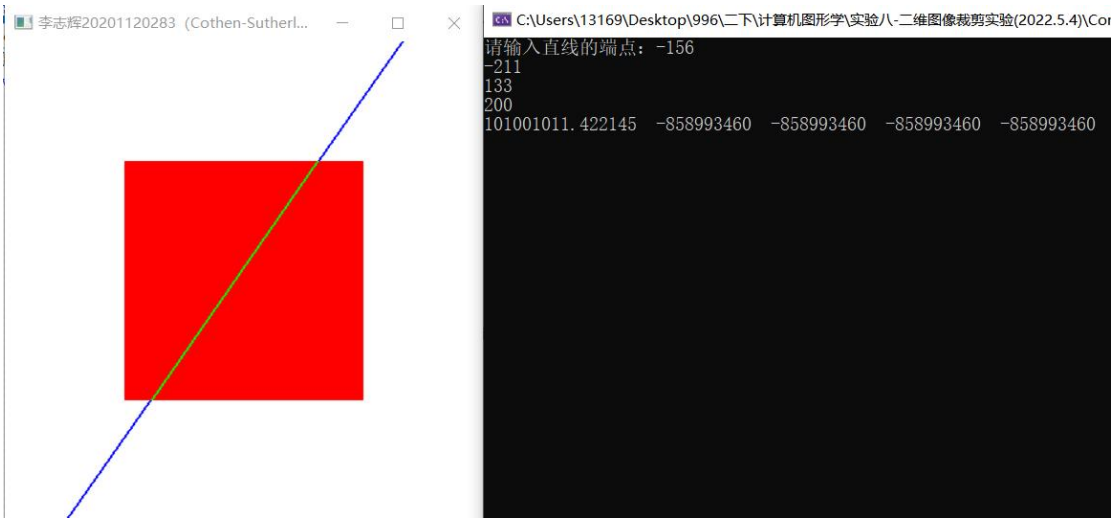
```
#include <GL/glut.h>
#include <iostream>

GLint winWidth = 400, winLength = 400, vWidth = 200, vLength = 200;
void Cothen_Sutherland();
class point {
public:
    int x, y, d1 = 0, d2 = 0, d3 = 0, d4 = 0;
};
point P[2];
void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-200, 200, -200, 200);
    glLineWidth(2);
    glEnable(GL_BLEND);
}
void disPlayFnc() {
    std::cout << "请输入直线的端点：";
    std::cin >> P[0].x >> P[0].y >> P[1].x >> P[1].y;
    for (int i = 0; i < 2; i++) {
        if (P[i].x > vWidth / 2)
```

```
            P[i].d4 = 1;
        if (P[i].x < -vWidth / 2)
            P[i].d3 = 1;
        if (P[i].y > vLength / 2)
            P[i].d2 = 1;
        if (P[i].y < -vLength / 2)
            P[i].d1 = 1;
        printf("%d%d%d%d", P[i].d1, P[i].d2, P[i].d3, P[i].d4);
    }
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2i(P[0].x, P[0].y);
    glVertex2i(P[1].x, P[1].y);
    glEnd();
    glFlush();
    Cothen_Sutherland();
}
void Cothen_Sutherland() {
    if ((P[0].d1 | P[1].d1 + P[0].d2 | P[1].d2 + P[0].d3 | P[1].d3 + P[0].d4 | P[1].d4) ==
0) {
        glBegin(GL_POLYGON);
        glColor4f(1.0, 0.0, 0.0, 0.5);
        glVertex2i(-100, 100);
        glVertex2i(100, 100);
        glVertex2i(100, -100);
        glVertex2i(-100, -100);
        glEnd();
        glBegin(GL_LINES);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2i(P[0].x, P[0].y);
        glVertex2i(P[1].x, P[1].y);
        glEnd();
        glFlush();
        return;
    }
    if ((P[0].d1 & P[1].d1 + P[0].d2 & P[1].d2 + P[0].d3 & P[1].d3 + P[0].d4 & P[1].d4) !=
0) {
        glBegin(GL_POLYGON);
        glColor4f(1.0, 0.0, 0.0, 0.5);
        glVertex2i(-100, 100);
        glVertex2i(100, 100);
        glVertex2i(100, -100);
        glVertex2i(-100, -100);
```

```
        glEnd();
        glFlush();
        return;
    }
    double k = (double(P[0].y) - P[1].y) / (double(P[0].x) - P[1].x);
    printf("%lf", k);
    double a[4];
    a[0] = k * (-100.0 - P[0].x) + P[0].y;//left
    a[1] = k * (100.0 - P[0].x) + P[0].y;//right
    a[2] = (100.0 - P[0].y) / k + P[0].x;//top
    a[3] = (-100.0 - P[0].y) / k + P[0].x;//bottom
    point newp[2];
    int count = 0;
    if (P[0].d1 + P[0].d2 + P[0].d3 + P[0].d4 == 0) {
        newp[count] = P[0];
        count++;
    }
    else if (P[1].d1 + P[1].d2 + P[1].d3 + P[1].d4 == 0) {
        newp[count] = P[1];
        count++;
    }
    for (int i = 0; i < 4; i++) {
        if (a[i] <= 100 && a[i] >= -100) {
            switch (i)
            {
            case 0:newp[count].x = -100, newp[count].y = a[i]; count++; break;
            case 1:newp[count].x = 100, newp[count].y = a[i]; count++; break;
            case 2:newp[count].x = a[i], newp[count].y = 100; count++; break;
            case 3:newp[count].x = a[i], newp[count].y = -100; count++; break;
            default:
                break;
            }
        }
        if (count > 1)
            break;
        printf("  %d  %d", newp[count].x, newp[count].y);
    }
    glBegin(GL_POLYGON);
    glColor4f(1.0, 0.0, 0.0, 0.5);
    glVertex2i(-100, 100);
    glVertex2i(100, 100);
    glVertex2i(100, -100);
    glVertex2i(-100, -100);
    glEnd();
```

```
    glBegin(GL_LINES);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(newp[0].x, newp[0].y);
    glVertex2i(newp[1].x, newp[1].y);
    glEnd();
    glFlush();
    return;
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(winWidth, winLength);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("李志辉 20201120283（Cothen-Sutherland 裁剪算法）");

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

实验结果：



实验总结：通过本次实验，我对 Cohen-Sutherland 裁剪算法有了一定的了解，明白了其中的原理并可以完成一些简单的小程序。

实验九 三维图形几何变换实验

实验时间：2022 年 5 月 11 日

实验地点：信息学院机房

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

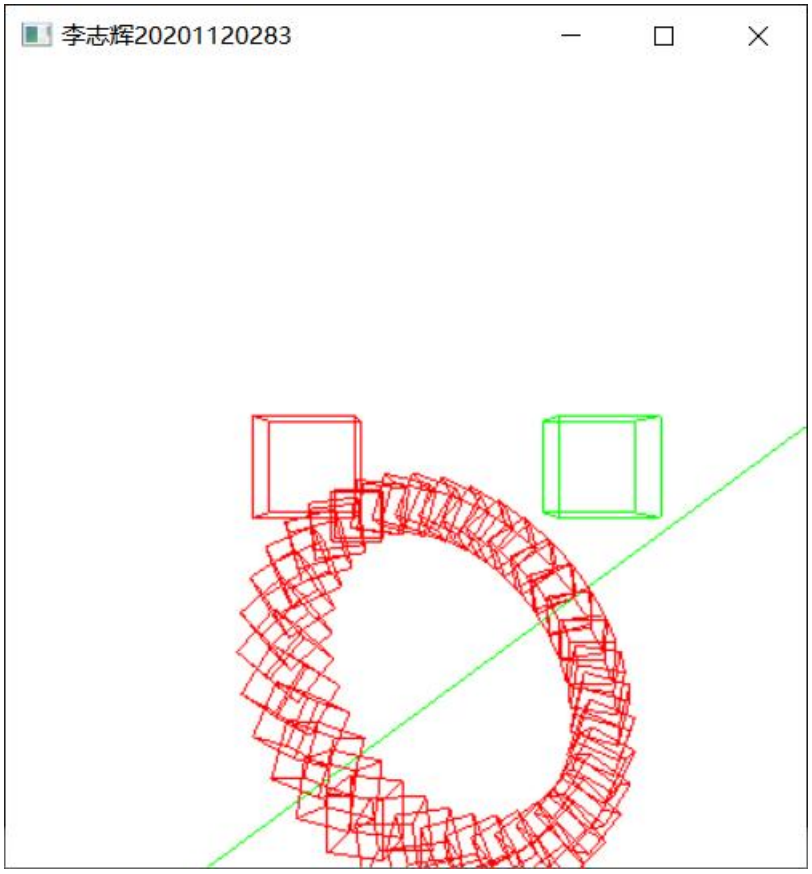
实验目的：调用函数完成三维图形几何变换。

实验代码：

```
#include <GL\glut.h>
#include <iostream>
#include <cstdio>

void init()
{
    glClearColor(1, 1, 1, 1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45, 1, 0, 100);
}
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(-1, 0, -10);
    glColor3f(1, 0, 0);
    glutWireCube(1);

    glTranslatef(3, 0, 0);
    glColor3f(0, 1, 0);
    glutWireCube(1);
    glFlush();

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(-1, -1, -20);
    glColor3f(0, 1, 0);
    glBegin(GL_LINES);
    //绘制出旋转轴
```

```
    glVertex3f(-25, -30, -25);
    glVertex3f(20, 15, 20);
    glEnd();
    glColor3f(1, 0, 0);
    for (int i = 0; i < 36; i++)
    {
        //逆变换
        glTranslatef(5, 0, 5);
        glRotatef(-45, 0, 0, 1);
        glRotatef(45, 1, 0, 0);
        glRotatef(10, 0, 1, 0);
        //再绕 x 轴顺时针旋转 45 度使旋转轴与 y 轴重合
        glRotatef(-45, 1, 0, 0);
        //先绕 z 轴逆时针旋转 45 度使旋转轴位于 yz 平面上
        glRotatef(45, 0, 0, 1);
        //先将旋转轴平移至经过原点
        glTranslatef(-5, 0, -5);
        glutWireCube(1);
    }
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(500, 500);
    glutInitWindowSize(400, 400);
    glutCreateWindow("李志辉 20201120283");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

实验结果：



实验总结：通过本次实验，了解并掌握了三维图形的一些基本几何变换，本次实验展示了旋转与对称。

实验十 可编程着色实验

实验时间：2022 年 5 月 17 日

实验地点：信息学院机房

实验内容 1：使用 opengl，片元着色器着色，P523

实验内容 2：使用 opengl，教材 P541，颜色编码建模显示。

实验目的：验证片元着色器算法，获得着色结果。

实验代码：

10.1 教材 523 页片元着色器着色

```
#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;
}
```

10.2 教材 541 页颜色编码建模显示

```
#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) {
```

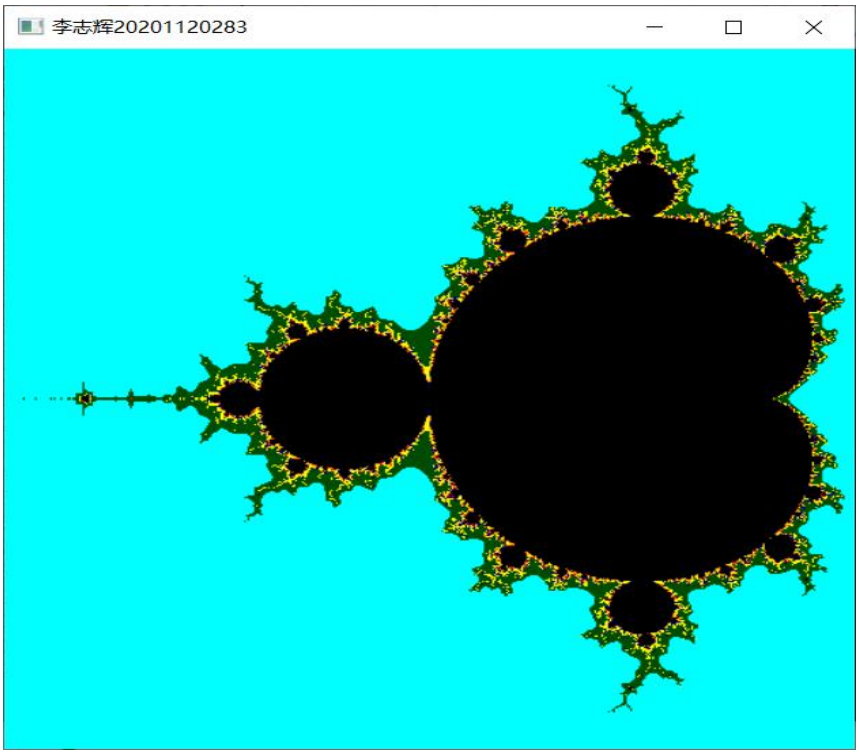
```
    glViewport(0, 0, newHeight, newHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(xComplexMin, xComplexMax, yComplexMin, yComplexMax);
    glClear(GL_COLOR_BUFFER_BIT);
}
void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("李志辉 20201120283");
    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMainLoop();

}
```

实验结果：

10.1 教材 523 页片元着色器着色

本次实验由于特殊情况无法运行

10.2 教材 541 页颜色编码建模显示



实验总结：通过本次的实验，我了解到了 opengl 中一种新的语言——着色器语言，但由于电脑问题，无法运行正常的着色程序。另外，完成了教材 541 页的颜色编码建模显示实验，对颜色编码建模有了一定的了解。

实验十一　交互控制实验

实验时间：2022 年 5 月 17 日

实验地点：信息学院机房

实验内容 1：使用 opengl，完成鼠标、键盘交互操作

实验目的：熟悉鼠标、键盘交互

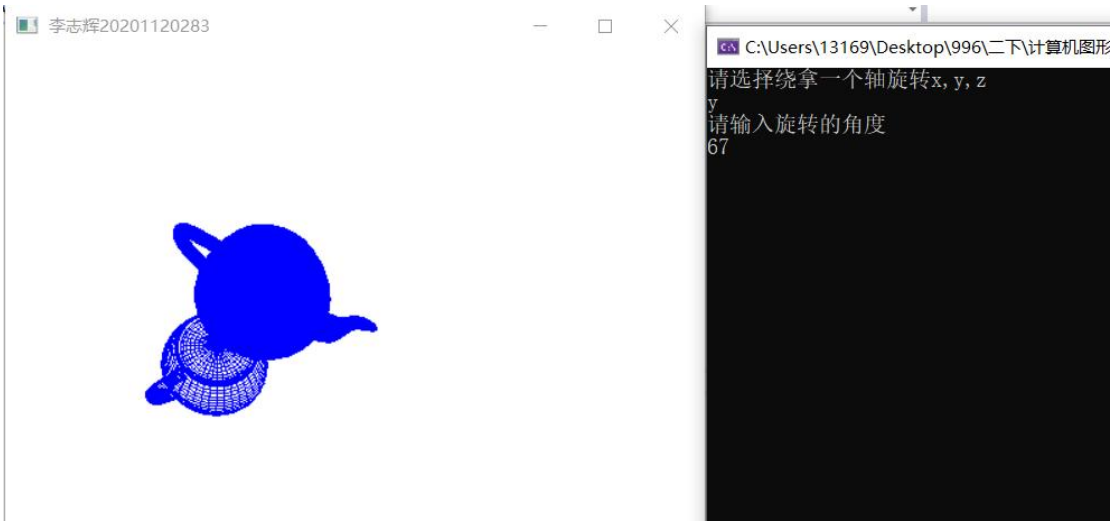
实验代码：

```
#include<GL/glut.h>
#include<stdio.h>
GLsizei winWidth = 500, winHeight = 500;
char sixel;
float thera = 0;
float x = 0, y = 0, z = 0;
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 0.0);
}


void displayWirePolyhedra(float x, float y, float z, float thera) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 1.0);
    gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glScalef(1.0, 1.0, 1.0);
    glTranslatef(1.0, 2.0, 0.0);//下一个图形坐标
    glutSolidTeapot(1.5);
    //glutWireTeapot(1.5);//放大倍数
    glScalef(1.0, 1.0, 1.0);//缩放比
    glTranslatef(-1.0, -5.0, 0.0);//下一个图形坐标
```

```
    glRotatef(thera, x, y, z);
    glutWireTeapot(1.5);
    //glutSolidTeapot(2.0);
    glFlush();
}

void display() {

    displayWirePolyhedra(x, y, z, thera);
}

void winReshapeFcn(GLint newWidth, GLint newHeight) {
    glViewport(0, 0, newWidth, newHeight);
    glMatrixMode(GL_PROJECTION);
    glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glClear(GL_COLOR_BUFFER_BIT);
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("   ");
    init();
    printf_s("请选择绕拿一个轴旋转 x,y,z \n");
    scanf_s("%c", &sixel);
    getchar();
    if (sixel == 'x') {
        x = 1.0;
        y = 0.0;
        z = 0.0;
        printf_s("请输入旋转的角度\n");
        scanf_s("%f", &thera);
    }
    else if (sixel == 'y') {
        x = 0.0;
        y = 1.0;
        z = 0.0;
        printf_s("请输入旋转的角度\n");
        scanf_s("%f", &thera);
    }
    else if (sixel == 'z') {
```

```
            x = 0.0;
            y = 0.0;
            z = 1.0;
            printf_s("请输入旋转的角度\n");
            scanf_s("%f", &thera);
        }
        else {
            printf_s("输入有误\n");
        }
        glutDisplayFunc(display);
        glutReshapeFunc(winReshapeFcn);
        glutMainLoop();
}
```

实验结果：



实验总结：通过本次实验学会了通过键盘和鼠标交互操作来达到想要实现的效果。

实验十二　三维观察实验

实验时间：2022 年 5 月 25 日

实验地点：信息学院机房

实验内容 1：使用 opengl，完成投影变换等实验，P 264

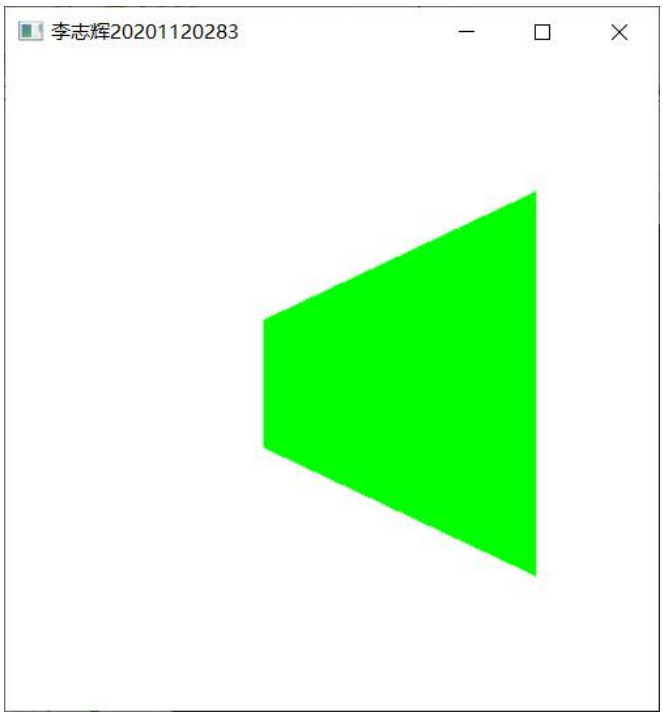
实验目的：熟悉三维观察相关内容

实验代码：

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

GLint winWidth = 600, winHeight = 600;

GLfloat x0 = 100.00, 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 = 60.0;

GLfloat dnear = 25.0, dfar = 125.0;

void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    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 Squuare");

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

实验结果：



实验总结：通过本次实验明白了投影变换的相关知识并完成了一个简单的程序。

实验十三　多面体实验

实验时间：2022 年 6 月 1 日

实验地点：信息学院机房

实验内容：生成多面体线框图，P300、P307

实验目的：熟悉三维线框图相关内容

实验代码：

13.1 教材 P300 线框图程序

```
#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();
}
```

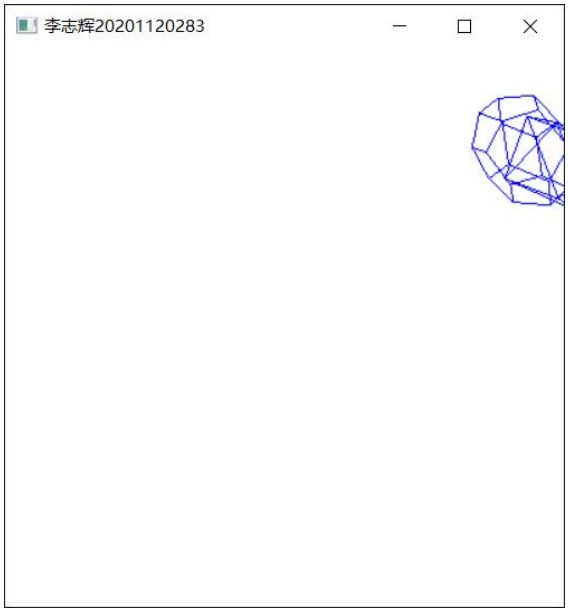
13.2 教材 P307 线框图程序

```
#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();
}
```

实验结果：

13.1 教材 P300 线框图程序



13.2 教材 P307 线框图程序

实验总结：通过本次实验学会了画一些简单的三维线框图。

实验十四　曲线曲面生成实验

实验时间：2022 年 6 月 7 日

实验地点：信息学院机房

实验内容：生成曲线或者曲面，P323

实验目的：熟悉 Bezier、样条等相关内容

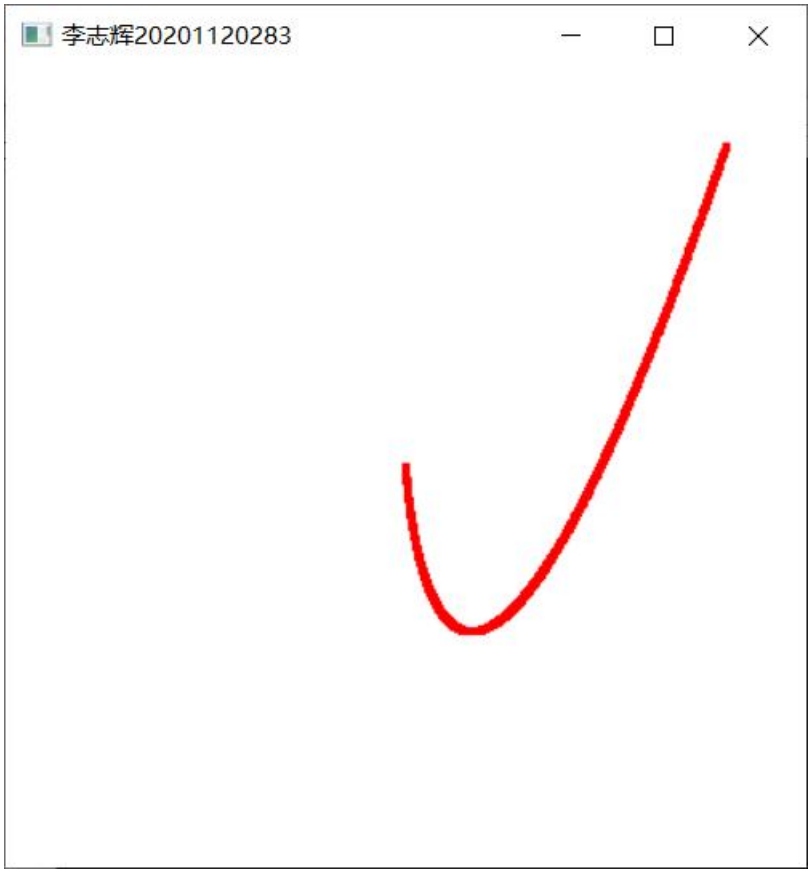
实验代码：

```
#include<GL/glut.h>
#include<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, 0.0);
}

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

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

    for (k = 0; k <= n; k++) {
        G[k] = 1;
```

```
        for (j = n; k >= k + 1; j--)
            G[k] *= j;
        for (j = n - k; j >= 2; j--)
            G[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, 30);
    glutInitWindowSize(400, 400);
    glutCreateWindow("李志辉 20201120283");

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

    glutMainLoop();
}
```

实验结果：



实验总结：通过本次实验熟悉 Bezier、样条等相关内容，并完成了教材 323 页的生成曲线示例程序。

实验十五 消隐实验

实验时间：2022 年 6 月 15 日

实验地点：信息学院机房

实验内容：完成消隐实验，采用 Z-buffer 算法完成消隐

实验目的：熟悉 Z-buffer 、画家算法等相关内容

实验代码：

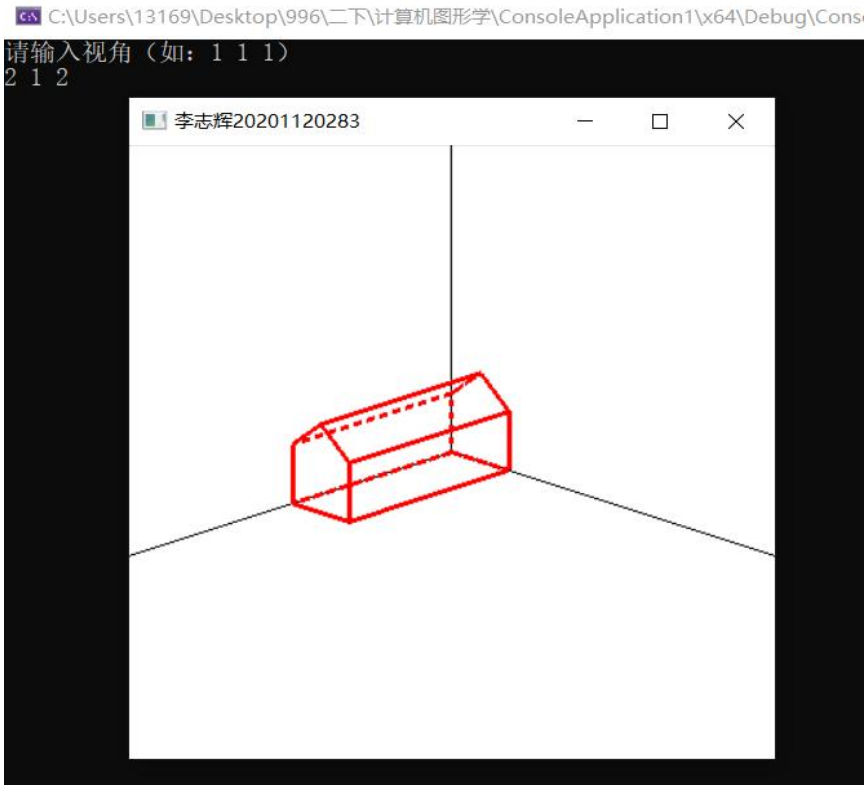
```
#include <iostream>
#include "Windows.h"
#include "math.h"
```

```
#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("李志辉 20201120283");

    Init();

    glutDisplayFunc(myDisplay);
    glutReshapeFunc(Reshape);

    glutMainLoop();
    return 0;
}
```

实验结果：



实验总结：通过本次实验熟悉并掌握了采用 Z-buffer 算法完成消隐。