计算机学院实验报告

实验题目：实验三：Z-buffering算法		学号：202000120101
日期：2022.10.11	班级：20.2	姓名：尹国泰
Email：1018693208@qq.com		

实验目的：

了解消隐的算法思想。在屏幕上绘制两个不同的图形，利用 Z-buffering 算法实现离视点更近的图形显示在上层。

实验环境介绍：

操作系统：Window10

编译器环境：MinGW，VSCode

OpenGL 环境：freeglut

解决问题的主要思路：

1. 创建帧缓存器f_buffer[][]存储屏幕像素点颜色信息，初始化为背景色。创建z缓存器z_buffer[][]存储屏幕像素点显示的图形的z坐标，初始化为无穷小。
2. 使用多边形扫描转换算法依次转换每一个多边形，对于每个多边形上的每个像素点对f_buffer和z_buffer进行更新。
3. 所有多边形转换完毕后，根据f_buffer输出屏幕上每个像素点。
4. 设计键盘鼠标控制函数用于多边形的输入、绘制

实验步骤：

1. 创建帧缓存器 f_buffer[][]和 z 缓存器 z_buffer[][]，cntz 记录已转换的多边形数，输入多边形时，让新输入的多边形 z 坐标更大，即设置成 cntz（离我们更近）

```
struct frame{
    float r,g,b;
    frame(){}
    frame(float _r, float _g,float _b)
        :r(_r), g(_g), b(_b) {}
};
frame f_buffer[w+1][h+1];
int z_buffer[w+1][h+1],cntz;
```

2. 实现 z_buffer 算法。首先初始化 f_buffer 为背景色，z_buffer 为无穷小；然后扫描转换每个多边形，并更新 f_buffer 和 z_buffer；转换完毕，根据 f_buffer 输出屏幕像素点。

```
void exec_z_buffer(){
    //初始化 f_buffer 为背景色，z_buffer 为无穷小
    for(int i=0;i<=w;i++){
        for(int j=0;j<=h;j++){
```

```
            f_buffer[i][j].r=0;
            f_buffer[i][j].g=0;
            f_buffer[i][j].b=0;
            z_buffer[i][j]=-1e9;
        }
    }
    //扫描转换每个多边形，并更新 f_buffer 和 z_buffer
    for(auto p:polyset){
        PolyScan(p);
    }
    //转换完毕，根据 f_buffer 输出屏幕像素点
    for(int i=0;i<=w;i++){
        for(int j=0;j<=h;j++){
            frame frm=f_buffer[i][j];
            glBegin(GL_POINTS);
            glColor3f(frm.r, frm.g, frm.b);
            glVertex2f(i, j);
            glEnd();
        }
    }
}
```

3.多边形扫描转换中更新 f_buffer 和 z_buffer 的代码如下

```
        //以 AET 中的点两两配对的形式来进行填充
        curAET=headAET->next;
        while (curAET!= NULL && curAET->next != NULL){
            for (int x = (int)curAET->x; x < curAET->next->x; x++){
                if(poly.z > z_buffer[x][i]){
                    f_buffer[x][i].r=poly.f.r;
                    f_buffer[x][i].g=poly.f.g;
                    f_buffer[x][i].b=poly.f.b;
                    z_buffer[x][i]=poly.z;
                }
            }
            curAET=curAET->next->next;
        }
```

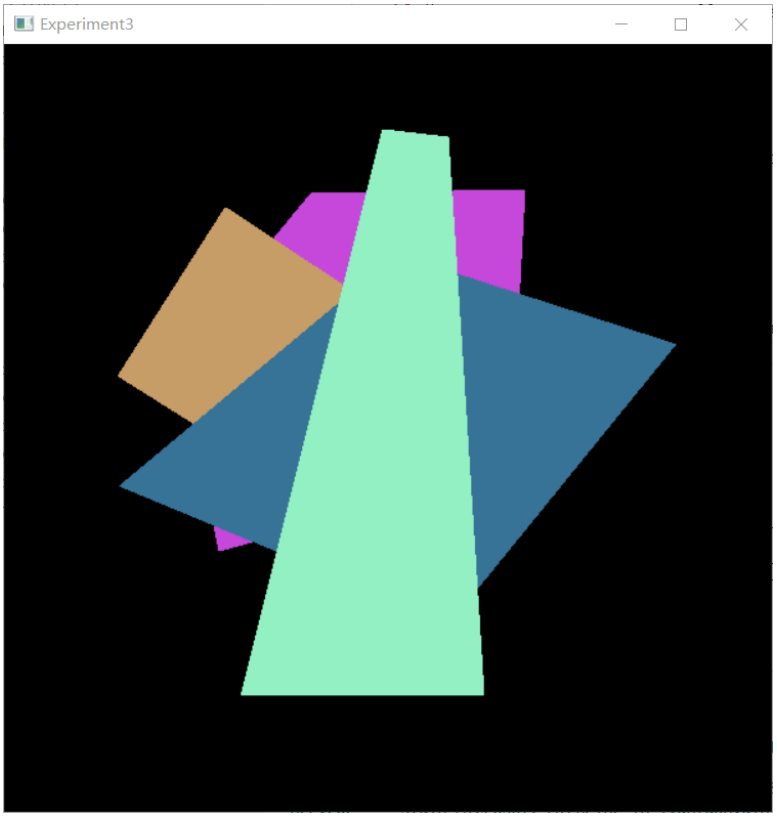
4. 设计键盘鼠标控制函数用于多边形的输入、绘制

```
//向多边形集合中加入一个新的多边形
void creatnewpoly(){
    polyset.push_back(Poly());
    int last=polyset.size()-1;
    polyset[last].z = ++cntz;
    polyset[last].f.r=randf();
```

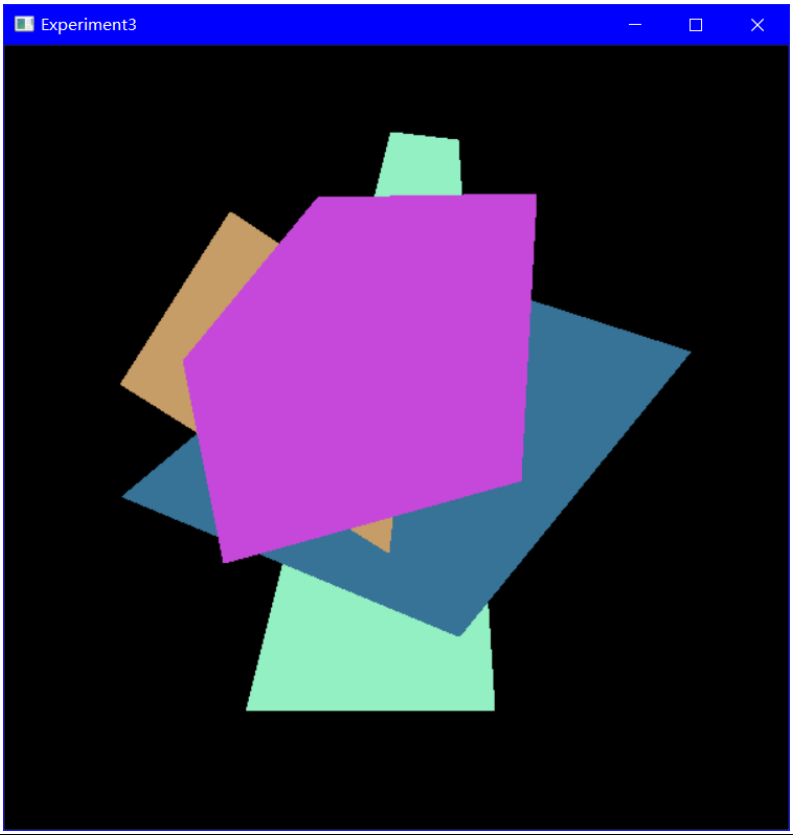
```
    polyset[last].f.g=randf();
    polyset[last].f.b=randf();
}
void mymouse(int button, int state, int x, int y){
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        point p(x, h - y);
        int last=polyset.size()-1;
        polyset[last].vertex.push_back(p);
    }//左键确定多边形的顶点

    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN){
        glClear(GL_COLOR_BUFFER_BIT);
        exec_z_buffer();
        glFlush();
        creatnewpoly();
    }//右键运行 z_buffer 输出所有多边形

    if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN){
        glClear(GL_COLOR_BUFFER_BIT);
        polyset.clear();
        cntz=0;
        creatnewpoly();
        glFlush();
    }//中键清空已输入的多边形
}
void myKeyboard(unsigned char key, int x, int y){
    //按下 Tab 键将所有已经输入的多边形 z 坐标取相反数
    if(key==9){
        for(int i=0;i<polyset.size();i++) {
            polyset[i].z=-polyset[i].z;
        }

        glClear(GL_COLOR_BUFFER_BIT);
        exec_z_buffer();
        glFlush();
        creatnewpoly();
    }
}
```

实验结果展示及分析：

1. 绘制输入的多个多边形，新输入的多边形 z 坐标更大



2. 按下 Tab 键能够让所有多边形的 z 坐标取相反数



实验中存在的问题及解决：

1. 一次 z_buffer 算法中要扫描转换多个多边形，在每一次扫描转换之前必须要清空 AET 和 NET，该如何实现？

可以创建一个clearET(ET* cur)函数，删除以cur为头结点的边表

```
void clearET(ET* cur){
    while(cur!=NULL){
        ET* nxt=cur->next;
        delete cur;
        cur=nxt;
    }
}
```

执行完每次扫描线转换后，通过以下方式删除存在的AET和NET

```
    clearET(headAET);
    headAET=NULL;
    for(int i=ymin;i<=ymax;i++) clearET(headNET[i]),headNET[i]=NULL;
```

2. 输入的不同多边形需要用不同的颜色来区分，有什么方法来让每次输入的多边形颜色区分度比较明显？

可以创建一个randf函数生成[0 ， 1.0]的随机浮点数

```
float randf(){
    return rand()/double(RAND_MAX);
}
```

初始化多边形时用三个randf()随机设置多边形的RGB

```
    polyset[last].f.r=randf();
    polyset[last].f.g=randf();
    polyset[last].f.b=randf();
```

附件：

```
#include<iostream>
#include <GL/glut.h>
#include<algorithm>
#include<vector>
#include<stack>
#include<queue>
#include<Windows.h>
#include<ctime>
using namespace std;

const int w=600,h=600;

float randf(){
    return rand()/double(RAND_MAX);
```

```
}
struct point{
    int x, y;
    point(){}
    point(int _x, int _y)
        :x(_x), y(_y) {}
};
struct frame{
    float r,g,b;
    frame(){}
    frame(float _r, float _g,float _b)
        :r(_r), g(_g), b(_b) {}
};
struct Poly{
    int z;
    frame f;
    vector<point> vertex;//多边形顶点集合
};
vector<Poly> polyset;


frame f_buffer[w+1][h+1];
int z_buffer[w+1][h+1],cntz;


typedef struct ET{
    float x;
    float dx;
    float ymax;
    ET* next;
}AET,NET;//活性边表，新边表
AET* headAET;
NET* headNET[h+5];
void clearET(ET* cur){
    while(cur!=NULL){
        ET* nxt=cur->next;
        delete cur;
        cur=nxt;
    }
}

void PolyScan(Poly poly){

    //确定扫描线最低和最高值
    int ymin=h,ymax=0;
```

```
    for(auto i:poly.vertex) ymin=min(ymin,i.y),ymax=max(ymax,i.y);

    //初始化活性边表和新边表
    headAET=new AET;
    headAET->next=NULL;

    for(int i=ymin;i<=ymax;i++){
        headNET[i] = new NET;
        headNET[i]->next = NULL;
    }
    //建立新边表 NET
    for(int j=0;j<poly.vertex.size();j++){//遍历多边形顶点(按顺序)
        int pre=(j-1+poly.vertex.size())%poly.vertex.size();//前一个点
在 poly.vertex 中的下标
        int aft=(j+1)%poly.vertex.size();//后一个点在 poly.vertex 中的下
标
        if (poly.vertex[pre].y > poly.vertex[j].y){
            //与前一个点构成的边是一条新的边
            NET* cur=new NET;
            cur->x = poly.vertex[j].x;
            cur->ymax = poly.vertex[pre].y;
            float DX = poly.vertex[pre].x-poly.vertex[j].x;
            float DY = poly.vertex[pre].y-poly.vertex[j].y;
            cur->dx = DX/DY;
            cur->next = headNET[poly.vertex[j].y]->next;
            headNET[poly.vertex[j].y]->next = cur;
        }
        if (poly.vertex[aft].y > poly.vertex[j].y){
            //与后一个点构成的边是一条新的边
            NET* cur = new NET;
            cur->x = poly.vertex[j].x;
            cur->ymax = poly.vertex[aft].y;
            float DX = poly.vertex[aft].x-poly.vertex[j].x;
            float DY = poly.vertex[aft].y-poly.vertex[j].y;
            cur->dx = DX/DY;
            cur->next = headNET[poly.vertex[j].y]->next;
            headNET[poly.vertex[j].y]->next = cur;
        }
    }
    //通过活性边表 AET 来进行区域填充
    for(int i=ymin;i<=ymax;i++){
        NET *curNET;
        AET *curAET,*preAET;
        //删除 AET 中到达 ymax 的边
```

```
        preAET=headAET;
        curAET=headAET->next;
        while (curAET){
            if (curAET->ymax == i){
                preAET->next = curAET->next;
                delete curAET;
                curAET = preAET->next;
            }else{
                preAET = preAET->next;
                curAET = curAET->next;
            }
        }

        //将 NET 中在 y=i 这一扫描线新的边用插入排序加入到 AET 中
        curNET=headNET[i]->next;
        while(curNET){
            curAET= headAET;
            while (curAET->next != NULL && curNET->x >
curAET->next->x)
                curAET = curAET->next;
            if(curAET->next != NULL && curNET->x == curAET->next->x
              &&curNET->dx > curAET->next->dx)
                curAET = curAET->next;
            AET *tmp=new AET;
            tmp->dx=curNET->dx;
            tmp->ymax=curNET->ymax;
            tmp->x=curNET->x;
            tmp->next=curAET->next;
            curAET->next=tmp;

            curNET = curNET->next;
        }
        //以 AET 中的点两两配对的形式来进行填充
        curAET=headAET->next;
        while (curAET!= NULL && curAET->next != NULL){
            for (int x = (int)curAET->x; x < curAET->next->x; x++){
                if(poly.z > z_buffer[x][i]){
                    f_buffer[x][i].r=poly.f.r;
                    f_buffer[x][i].g=poly.f.g;
                    f_buffer[x][i].b=poly.f.b;
                    z_buffer[x][i]=poly.z;
                }
            }
            curAET=curAET->next->next;
```

```
        }

        //更新 AET 中边的 x
        curAET=headAET->next;
        while(curAET){
            curAET->x+=curAET->dx;
            curAET=curAET->next;
        }

    }

    clearET(headAET);
    headAET=NULL;
    for(int i=ymin;i<=ymax;i++) clearET(headNET[i]),headNET[i]=NULL;
}

void exec_z_buffer(){
    //初始化 f_buffer 为背景色，z_buffer 为无穷小
    for(int i=0;i<=w;i++){
        for(int j=0;j<=h;j++){
            f_buffer[i][j].r=0;
            f_buffer[i][j].g=0;
            f_buffer[i][j].b=0;
            z_buffer[i][j]=-1e9;
        }
    }
    //扫描转换每个多边形，并更新 f_buffer 和 z_buffer
    for(auto p:polyset){
        PolyScan(p);
    }
    //转换完毕，根据 f_buffer 输出屏幕像素点
    for(int i=0;i<=w;i++){
        for(int j=0;j<=h;j++){
            frame frm=f_buffer[i][j];
            glBegin(GL_POINTS);
            glColor3f(frm.r, frm.g, frm.b);
            glVertex2f(i, j);
            glEnd();
        }
    }
}

void creatnewpoly(){
    polyset.push_back(Poly());
```

```
    int last=polyset.size()-1;
    polyset[last].z = ++cntz;
    polyset[last].f.r=randf();
    polyset[last].f.g=randf();
    polyset[last].f.b=randf();
}
void mymouse(int button, int state, int x, int y){
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        point p(x, h - y);
        int last=polyset.size()-1;
        polyset[last].vertex.push_back(p);
    }//左键确定多边形的顶点

    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN){
        glClear(GL_COLOR_BUFFER_BIT);
        exec_z_buffer();
        glFlush();
        creatnewpoly();
    }//右键填充

    if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN){
        glClear(GL_COLOR_BUFFER_BIT);
        polyset.clear();
        cntz=0;
        creatnewpoly();
        glFlush();
    }//中键清空
}
void myKeyboard(unsigned char key, int x, int y){
    if(key==9){
        for(int i=0;i<polyset.size();i++) {
            polyset[i].z=-polyset[i].z;
        }

        glClear(GL_COLOR_BUFFER_BIT);
        exec_z_buffer();
        glFlush();
        creatnewpoly();
    }
}
void display(){}

void Init(){
    //设置颜色
```

```
    glClearColor(0.0, 0.0, 0.0, 0.0);
    //颜色过渡形式
    glShadeModel(GL_SMOOTH);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);

    cntz=0;
    creatnewpoly();
    srand((int)time(0));

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


    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    //设置初始窗口的位置
    glutInitWindowPosition(100, 100);
    //设置初始窗口的大小
    glutInitWindowSize(w, h);
    //根据前面设置建立窗口，参数设置为变体

    glutCreateWindow("Experiment3");
    Init();
    //绘图时被调用的函数
    glutDisplayFunc(display);
    glutMouseFunc(mymouse);
    glutKeyboardFunc(myKeyboard);
    //进行消息循环，用于显示窗体，窗体关闭后自动退出循环
    glutMainLoop();
    return 0;
}
```