计算机学院实验报告

实验题目：实验四：反走样算法		学号：202000120101
日期：2022.10.25	班级：20.2	姓名：尹国泰
Email：1018693208@qq.com		

实验目的：

了解反走样的算法思想，使用反走样技术处理简易模型边缘的锯齿感。

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. 本实验中我在多边形扫描转换算法的基础上增加了反走样算法，以实现当分辨率降低，即绘制所用像素点变大时，能够减轻多边形边缘的锯齿感。

2. 依据反走样算法思想，我将要绘制的单位像素分成N*N的子像素点，每个子像素点对单位像素有不同的灰度贡献。具体来说：

（1） 子像素点中心离单位像素中心越远，贡献越小

（2） N*N个子像素对单位像素贡献值之和为灰度最大值1。

（3） 当且仅当多边形扫描转换算法转换了某个子像素时，该子像素才会对其所在单位像素产生贡献

3. 扫描转换多边形即计算转换的子像素对单位像素产生了多少灰度贡献，扫描转换完成后，也就计算出了每个单位像素的灰度值。然后以单位像素大小绘制多边形，将RGB分别乘以灰度值作为新的RGB值。

实验步骤：

1. 首先设置单位像素边长，单位像素灰度值矩阵，子像素灰度贡献度矩阵

```
const int PixLength=5;//绘制的像素边长
float graylevel[w/PixLength+1][h/PixLength+1];//绘制的像素的灰度值
float weight[PixLength+1][PixLength+1];//每个绘制的像素的子像素贡献度
```

2. 求出子像素灰度贡献度矩阵每个子像素贡献度

```
void Setweight(){//设置子像素贡献度
    double sum=0;
    //maxdis 为所有子像素中心点距离单位像素中心点最远距离
    float maxdis=sqrt( (0.5-1.0*PixLength/2)*(0.5-
1.0*PixLength/2)+(0.5-1.0*PixLength/2)*(0.5-1.0*PixLength/2) );
    for(int i=0;i<PixLength;i++){
        for(int j=0;j<PixLength;j++){
```

```
            int x=i+0.5,y=j+0.5;
            //dis 为一个子像素中心点距离单位像素中心点距离
            float dis=sqrt( (x-1.0*PixLength/2)*(x-
1.0*PixLength/2)+(y-1.0*PixLength/2)*(y-1.0*PixLength/2) );
            //每个子像素的贡献度是与 dis 负相关的
            weight[i][j]=maxdis+1-dis;
            sum+=weight[i][j];
        }
    }
    //设置 sum 用于将一个单位像素的所有子像素贡献度之和化为 1
    for(int i=0;i<PixLength;i++){
        for(int j=0;j<PixLength;j++){
            weight[i][j]=weight[i][j]/sum;
        }
    }
}
```

3. 多边形扫描转换算法转换了某个子像素时，该子像素才会对其所在单位像素产生贡献

```
        curAET=headAET->next;
        while (curAET!= NULL && curAET->next != NULL){
            for (int j = (int)curAET->x; j < curAET->next->x; j++){
                //扫描转化时计算单位像素的灰度值
                //扫描转化到的子像素对其所在的单位像素产生贡献
                graylevel[j/PixLength][i/PixLength]
                                +=weight[j%PixLength][i%PixLength];
            }
            curAET=curAET->next->next;
        }
```

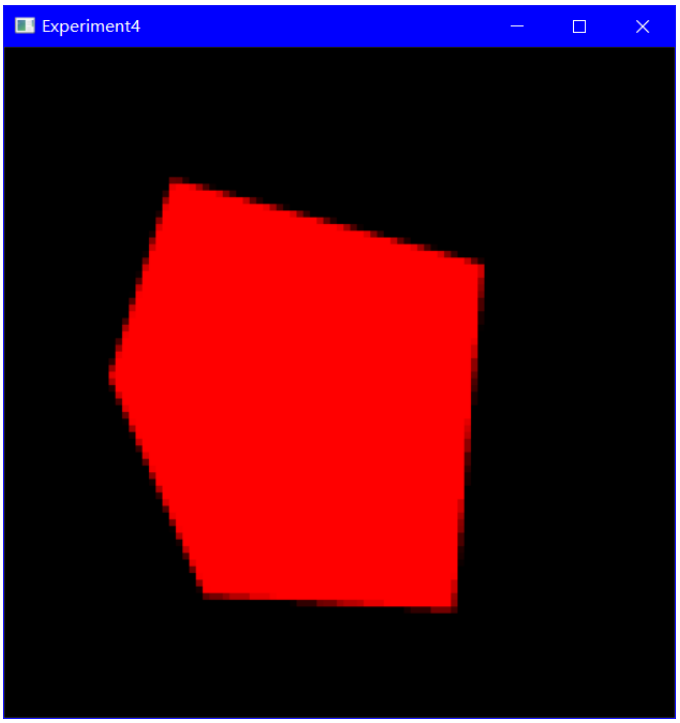
4. 以单位像素大小绘制多边形，将 RGB 分别乘以灰度值作为新的 RGB 值，下面的代码中包含使用反走样和不使用反走样算法的绘制，以便用于对比。

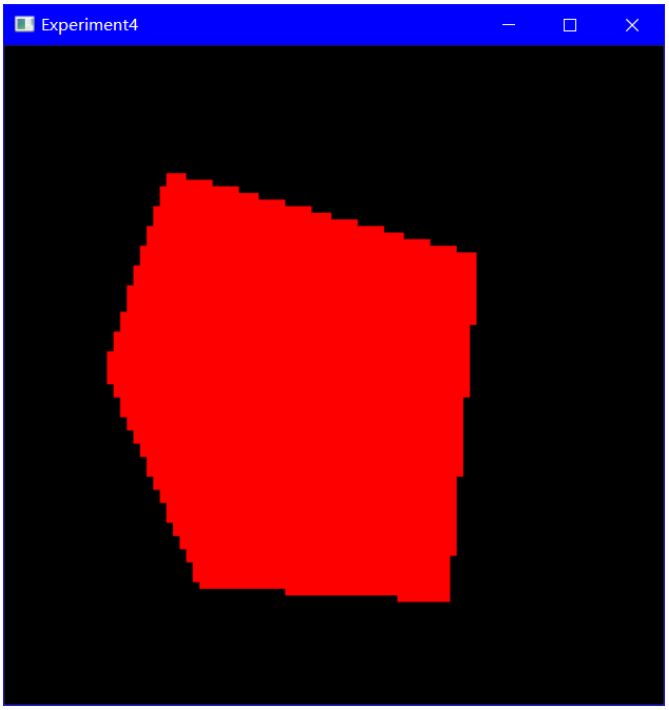
```
bool UseAntiAlia;
void DrawPoly(){//依据灰度值，以 PixLength 为像素边长填充
    glPointSize(PixLength);//设置绘制的像素的大小
    glBegin(GL_POINTS);
    float r=1.0,g=0.0,b=0.0;
    for(int i=0;i<=w/PixLength;i++){
        for(int j=0;j<=h/PixLength;j++){
            if(UseAntiAlia){//使用反走样，将 rgb 乘以灰度值
                glColor3f(r*graylevel[i][j],g*graylevel[i][j],b*gray
level[i][j]);
            }else{//不使用反走样
                if(graylevel[i][j]!=0) glColor3f(r,g,b);
                else glColor3f(0,0,0);
            }
```

```
            glVertex2f(i*PixLength,j*PixLength);//以 PixLength 为像素
边长
        }
    }
    glEnd();
}
```

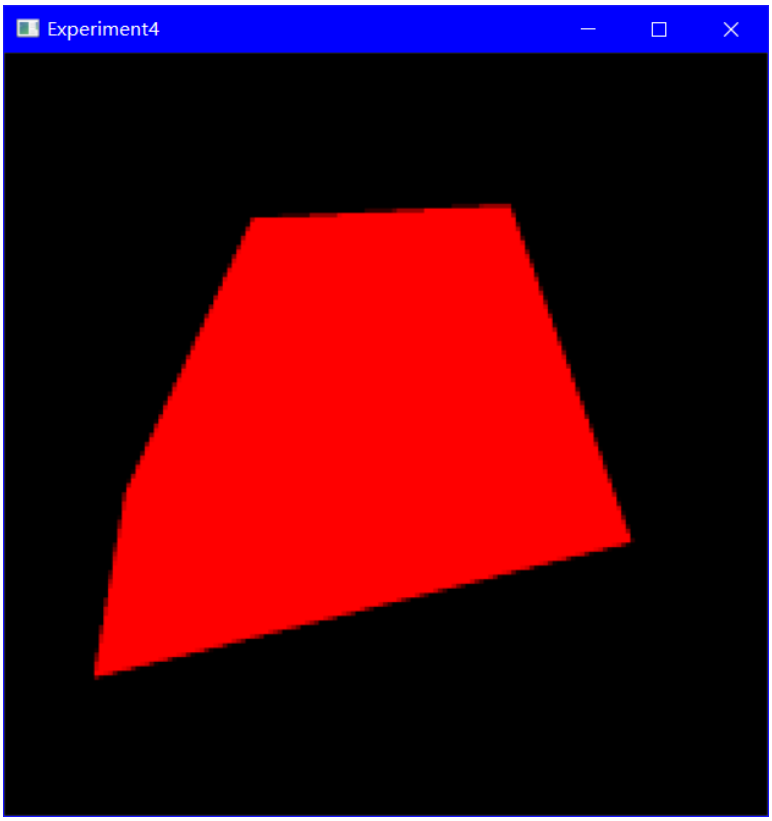
实验结果展示及分析：

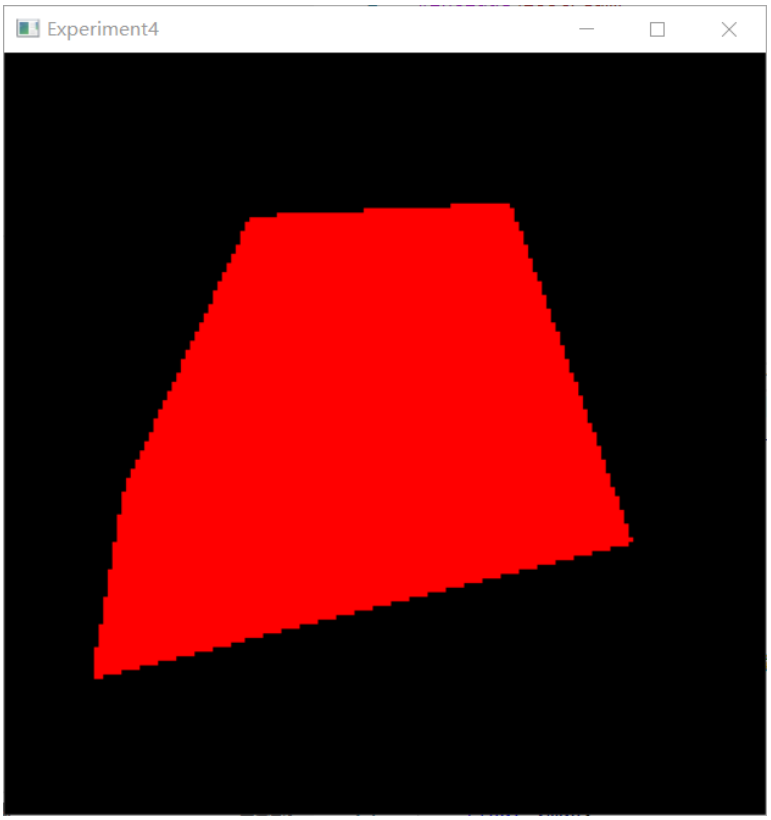
1. 以下为单位像素边长设置为 5 时，使用反走样和不使用反走样绘制的多边形





2. 以下为单位像素边长设置为 3 时，使用反走样和不使用反走样绘制的多边形





实验中存在的问题及解决：

1. 该如何去设置子像素对单位像素的贡献度？

（1） 若单位像素边长PixLength=N，我设置了一个weight[N][N]数组来存储每个子像素对单位像素的贡献度

（2） 首先子像素点中心离单位像素中心越远，贡献越小。

那么就要计算出像素点(i,j)中心距离单位像素中心点的距离dis(i,j)，并且让weight[i][j]与dis(i,j)成负相关

```
//dis为一个子像素中心点距离单位像素中心点距离
float dis=sqrt( (x-1.0*PixLength/2)*(x-1.0*PixLength/2)+(y-1.0*PixLength/2)*(y-1.0*PixLength/2) );
```

（3） 其次N*N个子像素对单位像素贡献值之和为灰度最大值1

我们只要算出 weight[i][j]占整个 weight 数组的比例即可。可以设置 weight[i][j]=maxdis+1-dis 先让 weight[i][j]既满足与 dis(i,j)成负相关又是正数，然后求出 weight 数组之和 sum，让 weight[i][j]除以 sum 得到所占比例。

附件：代码

E4.cpp

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

const int w=500,h=500;

const int PixLength=3;//绘制的像素边长
float graylevel[w/PixLength+1][h/PixLength+1];//绘制的像素的灰度值
float weight[PixLength+1][PixLength+1];//每个绘制的像素的子像素贡献度

struct point{
    int x, y;
    point(){}
    point(int _x, int _y)
        :x(_x), y(_y) {}
};
vector<point> vertex;//多边形顶点集合
```

```
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 drawbigpixelf(float x,float y){
    glColor3f(1, 0, 0);
    glPointSize(5);
    glBegin(GL_POINTS);
    glVertex2f(x, y);
    glEnd();
    glFlush();
}
bool inputend;
void clearET(ET* cur){
    while(cur!=NULL){
        ET* nxt=cur->next;
        delete cur;
        cur=nxt;
    }
}
void PolyScan(){
    //确定扫描线最低和最高值
    int ymin=h,ymax=0;
    for(auto i: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;
    headNET[h+5];
    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<vertex.size();j++){//遍历多边形顶点(按顺序)
        int pre=(j-1+vertex.size())%vertex.size();//前一个点在 vertex 中
的下标
        int aft=(j+1)%vertex.size();//后一个点在 vertex 中的下标
        if (vertex[pre].y > vertex[j].y){
            //与前一个点构成的边是一条新的边
            NET* cur=new NET;
```

```
            cur->x = vertex[j].x;
            cur->ymax = vertex[pre].y;
            float DX = vertex[pre].x-vertex[j].x;
            float DY = vertex[pre].y-vertex[j].y;
            cur->dx = DX/DY;
            cur->next = headNET[vertex[j].y]->next;
            headNET[vertex[j].y]->next = cur;
        }
        if (vertex[aft].y > vertex[j].y){
            //与后一个点构成的边是一条新的边
            NET* cur = new NET;
            cur->x = vertex[j].x;
            cur->ymax = vertex[aft].y;
            float DX = vertex[aft].x-vertex[j].x;
            float DY = vertex[aft].y-vertex[j].y;
            cur->dx = DX/DY;
            cur->next = headNET[vertex[j].y]->next;
            headNET[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 j = (int)curAET->x; j < curAET->next->x; j++){
                //扫描转化时计算单位像素的灰度值
                //扫描转化到的子像素对其所在的单位像素产生贡献
                graylevel[j/PixLength][i/PixLength]+=weight[j%PixLengt
h][i%PixLength];
            }
            curAET=curAET->next->next;
        }


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

        glFlush();
    }
    clearET(headAET);
    headAET=NULL;
    for(int i=0;i<=h;i++) clearET(headNET[i]),headNET[i]=NULL;
}
void Setweight(){//设置子像素贡献度
    double sum=0;
    //maxdis 为所有子像素中心点距离单位像素中心点最远距离
```

```
    float maxdis=sqrt( (0.5-1.0*PixLength/2)*(0.5-
1.0*PixLength/2)+(0.5-1.0*PixLength/2)*(0.5-1.0*PixLength/2) );
    for(int i=0;i<PixLength;i++){
        for(int j=0;j<PixLength;j++){
            int x=i+0.5,y=j+0.5;
            //dis 为一个子像素中心点距离单位像素中心点距离
            float dis=sqrt( (x-1.0*PixLength/2)*(x-
1.0*PixLength/2)+(y-1.0*PixLength/2)*(y-1.0*PixLength/2) );
            //每个子像素的贡献度是与 dis 负相关的
            weight[i][j]=maxdis+1-dis;
            sum+=weight[i][j];
        }
    }
    //设置 sum 用于将一个单位像素的所有子像素贡献度之和化为 1
    for(int i=0;i<PixLength;i++){
        for(int j=0;j<PixLength;j++){
            weight[i][j]=weight[i][j]/sum;
        }
    }
}

bool UseAntiAlia;
void DrawPoly(){//依据灰度值，以 PixLength 为像素边长填充
    glPointSize(PixLength);//设置绘制的像素的大小
    glBegin(GL_POINTS);
    float r=1.0,g=0.0,b=0.0;
    for(int i=0;i<=w/PixLength;i++){
        for(int j=0;j<=h/PixLength;j++){
            if(UseAntiAlia){//使用反走样，将 rgb 乘以灰度值
                glColor3f(r*graylevel[i][j],g*graylevel[i][j],b*grayle
vel[i][j]);
            }else{//不使用反走样
                if(graylevel[i][j]!=0) glColor3f(r,g,b);
                else glColor3f(0,0,0);
            }
            glVertex2f(i*PixLength,j*PixLength);//以 PixLength 为像素边
长
        }
    }
    glEnd();
}
void myKeyboard(unsigned char key, int x, int y){
    if (key == 9){
        glClear(GL_COLOR_BUFFER_BIT);
```

```
        UseAntiAlia^=1;
        DrawPoly();
        glFlush();
    }
}
void mymouse(int button, int state, int x, int y){
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        if(inputend){
            glClear(GL_COLOR_BUFFER_BIT);
            vertex.clear();
            inputend=0;
        }
        drawbigpixelf(x, h - y);
        point p(x, h - y);
        vertex.push_back(p);
    }//左键确定多边形的顶点

    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN){
        memset(graylevel,0,sizeof(graylevel));
        PolyScan();
        DrawPoly();
        glFlush();
        inputend=1;
    }//右键填充

    if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN){
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
        inputend=1;
    }//中键清空

}

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);
```

```
}
int main(int argc, char** argv) {
    Setweight();
    UseAntiAlia=1;
    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("Experiment4");
    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;
}
```