云南大学

本科实验报告

课程名称： 计算机图形学实验

实验名称： 实验六 二维裁剪

学院（系）： 信息学院

专 业： 计算机科学与技术

年 级： 2020级

姓 名： 胡诚皓

学 号： 20201060330

指导教师： 吴昊

成 绩：

2022年 5 月 11 日

**一. 实验目的**

1. 掌握OpenGL下裁剪窗口与视区的设置；

2. Cohen-Sutherland 算法的原理与实现；

3. 梁友栋-Barskey 算法的原理与实现；

**二. 实验内容**

完整实现Cohen-Sutherland和梁友栋-Barskey线段裁减算法，理解裁减的基本原理，并继续体会怎样用键盘来控制图形的显示。

本次实验主要是在OpenGL下实现Cohen-Sutherland 算法和梁友栋-Barskey 算法，并加上键盘控制。代码框架如下

void myDisplay() {

wcPt2D winMin, winMax, p1, p2;

winMin.x = 100;

winMin.y = 200;//定义裁剪窗口

winMax.x = 400;

winMax.y = 400;

p1.x = 200;

p1.y = 200; //定义被裁剪的直线端点

p2.x = 300;

p2.y = 300;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0f, 0.0f, 0.0f);

glRectf(winMin.x, winMin.y, winMax.x, winMax.y); //绘制红色裁剪矩形

glColor3f(1.0f, 1.0f, 1.0f);

if (flag) {

lineClipCohSuth (winMin, winMax,p1,p2);// 绘制裁剪线段，由你们实现

} else {

glBegin(GL\_LINES);

glVertex2i(p1.x, p1.y);

glVertex2i(p2.x, p2.y);

glEnd();

}

glFlush();

}

void myInit() {

glClearColor(0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0, 640.0, 0.0, 480.0);

}

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);

}

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

if (key == 'c' || key == 'C') flag = true;

if (key == 'r' || key == 'R') flag = false;

glutPostRedisplay();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(640, 480);

glutCreateWindow("Cohen-Sutherland");

myInit();

glutDisplayFunc(myDisplay);

glutReshapeFunc(Reshape);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**三. 实验代码**

typedef unsigned char encode;

struct wcPt2D {

int x;

int y;

};

bool flag = false;

// 0000上下右左

encode getEncode(wcPt2D pt, wcPt2D min, wcPt2D max) {

encode code = 0x00;

if (pt.y > max.y) {

code |= 1 << 3;

}

if (pt.y < min.y) {

code |= 1 << 2;

}

if (pt.x > max.x) {

code |= 1 << 1;

}

if (pt.x < min.x) {

code |= 1 << 0;

}

return code;

}

void lineClipCohSutherland(wcPt2D minCoord, wcPt2D maxCoord, wcPt2D p1, wcPt2D p2) {

int xmin = minCoord.x, ymin = minCoord.y;

int xmax = maxCoord.x, ymax = maxCoord.y;

int x1 = p1.x, y1 = p1.y;

int x2 = p2.x, y2 = p2.y;

double m = (double) (y2 - y1) / (x2 - x1);

encode code1 = getEncode(p1, minCoord, maxCoord);

encode code2 = getEncode(p2, minCoord, maxCoord);

// 如果两点都在视窗内，则直接画线

if (code1 == 0 && code2 == 0) {

glBegin(GL\_LINES);

glVertex2i(p1.x, p1.y);

glVertex2i(p2.x, p2.y);

glEnd();

} else if (code1 & code2) {

// 如果两点都在视窗外，则不画线

return;

} else {

// 如果一点在视窗内，一点在视窗外，则画线

if (code1) {

// 如果第一个点在视窗外，则画第一个点

glVertex2i(x1, y1);

// 计算第一个点的新坐标

if (code1 & 1) {

y1 += m \* (xmin - x1);

x1 = xmin;

}

if (code1 & 2) {

y1 += m \* (xmax - x1);

x1 = xmax;

}

if (code1 & 4) {

x1 += (ymin - y1) / m;

y1 = ymin;

}

if (code1 & 8) {

x1 += (ymax - y1) / m;

y1 = ymax;

}

}

if (code2) {

// 如果第二个点在视窗外，则画第二个点

glVertex2i(x2, y2);

// 计算第二个点的新坐标

if (code2 & 1) {

y2 += m \* (xmin - x2);

x2 = xmin;

}

if (code2 & 2) {

y2 += m \* (xmax - x2);

x2 = xmax;

}

if (code2 & 4) {

x2 += (ymin - y2) / m;

y2 = ymin;

}

if (code2 & 8) {

x2 += (ymax - y2) / m;

y2 = ymax;

}

}

glBegin(GL\_LINES);

glVertex2i(x1, y1);

glVertex2i(x2, y2);

glEnd();

}

}

// 计算处理边界的p、q、r

int clipTest(float p, float q, float \*u1, float \*u2) {

bool res = true;// 用于记录线段是否要保留

float r;

if (p < 0) {// 由外到内的情况

r = q / p;

if (r > \*u2) {// r>u2，意味着已经超出线段右端点，是无效的

res = false;

} else if (r > \*u1) {// 更新u1为较大值

\*u1 = r;

}

} else if (p > 0) {// 由内到外的情况

r = q / p;

if (r < \*u1) {// r<u1，意味着已经超出线段左端点，是无效的

res = false;

} else if (r < \*u2) {// 更新u2为较小值

\*u2 = r;

}

} else {// p=0，线段平行于边界

if (q < 0) {// 线段完全在边界外，直接舍弃

res = false;

}

// p大于等于0时，线段完全在裁剪窗口内，直接保留

}

return res;

}

void lineClipBarskey(wcPt2D minCoord, wcPt2D maxCoord, wcPt2D p1, wcPt2D p2) {

float u1 = 0.0, u2 = 1.0, dx = p2.x - p1.x, dy = p2.y - p1.y;

int xmin = minCoord.x, ymin = minCoord.y;

int xmax = maxCoord.x, ymax = maxCoord.y;

if (clipTest(-dx, p1.x - xmin, &u1, &u2)) {// 左边界

if (clipTest(dx, xmax - p1.x, &u1, &u2)) {// 右边界

if (clipTest(-dy, p1.y - ymin, &u1, &u2)) {// 下边界

if (clipTest(dy, ymax - p1.y, &u1, &u2)) {// 上边界

// 得到了裁剪后的参数u1、u2，将线段端点缩到裁剪后位置

if (u2 < 1) {// 右端点要调整

p2.x = p1.x + u2 \* dx;

p2.y = p1.y + u2 \* dy;

}

if (u1 > 0) {// 左端点要调整

p1.x += u1 \* dx;

p1.y += u1 \* dy;

}

glBegin(GL\_LINES);

glVertex2i(p1.x, p1.y);

glVertex2i(p2.x, p2.y);

glEnd();

}

}

}

}

}

**四. 实验结果**

双击图标查看视频

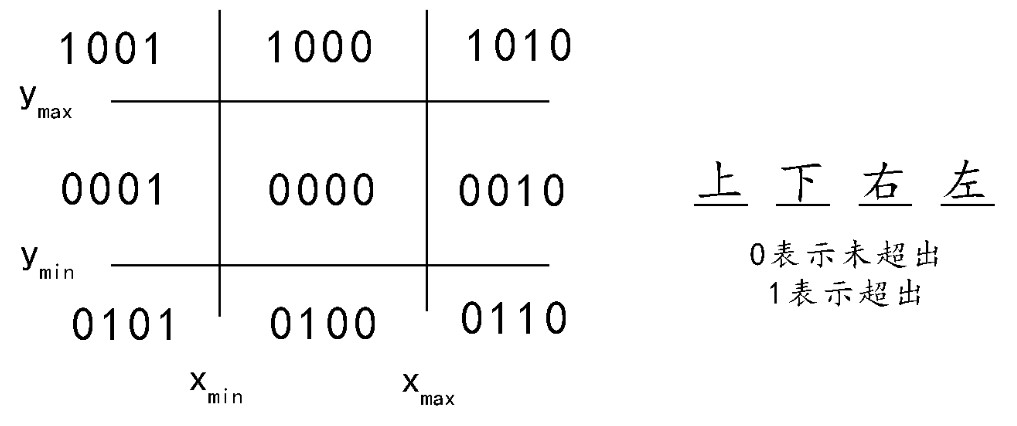




**五. 实验分析**

**Cohen-Sutherland算法**

根据裁剪窗口的边界坐标，将整个平面分为9个部分，用4位二进制进行编码，如下图所示



Sutherland裁剪对线段的两端点与也同样按照上述规则进行编码，分别记为。可以得到以下几种情况

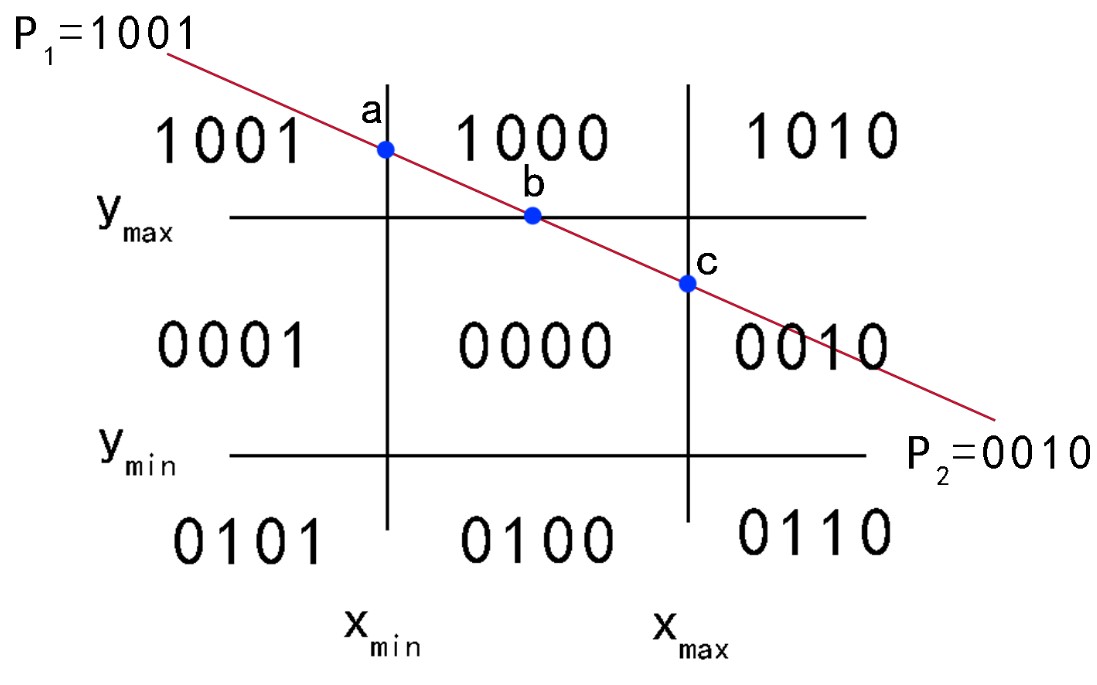
，说明该线段完全在裁剪窗口内部，保留即可

，即S1、S2按位与之后不为0，说明该线段完全在裁剪窗口外部，直接舍去该线段

S1与S2中不同值的位，表示这条线段横跨了对应的边界，分别对这些边界求交进行裁剪

记线段的斜率，那么线段与垂直边界交点的y坐标为；与水平边界交点的x坐标为，其中的x取、y取

用下图举个裁剪的例子。与左边界求交，裁去段；再与上边界求交，裁去段；再与右边界求交，裁去段，最后留下段，即为所求



**梁友栋-Barsky算法**

此算法是参数判断的一种优化，先写出直线段的参数化方程

再写出点裁剪条件的参数化表达

上式中其实包含了四个不等式，将他们统一表示为，k的值分别对应左、右、下、上边界，将定义列出

根据的情况进行分类

，说明该线段平行于对应边界

且，说明该线段完全在裁剪窗口外，直接舍去

且，说明该线段完全在裁剪窗口内，直接保留

，说明线段从对应边界的外侧向内侧延伸

，说明线段从对应边界的内侧向外侧延伸

在时，计算该直线段与边界交点的参数u值，很显然

将一开始线段两端点的u参数分别记为u1与u2，显然，该算法通过如下两条规则不断更新u1与u2，最后直接根据u1与u2画出裁剪后的线段

u1由线段的那些从对应边界的外侧向内侧延伸的部分更新。，其中为对应边界的

u2由线段的那些从对应边界的外侧向内侧延伸的部分更新。，其中同样为对应边界的

在求解过程中，若出现的情况，说明线段完全落在裁剪窗口之外，直接舍去。

在实际算法的实现中，在初始化u1为0、u2为1后，先计算各边界的p、q，然后遍历每一条边界的p、q，根据上面2条规则更新u1与u2，最后根据得到的u1、u2直接画出裁剪后的线段即可。