<https://zhuanlan.zhihu.com/p/331804824>

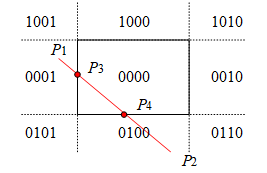


## openGL中点裁剪算法（鼠标交互实现）

## 算法原理

在编码裁剪算法中，为了快速判断一条直线段与矩形窗口的位置关系，采用了下图所示的空间划分和编码方案。

裁剪一条线段时，先求出两端点所在的区号code1和code2，若code1 = 0且code2 = 0，则说明线段的两个端点均在窗口内，那么整条线段必在窗口内，应取之；若code1和code2经按位“与”运算的结果不为0，则说明两个端点同在窗口的上方、下方、左方或右方。这种情况下，对线段的处理是弃之。如果上述两种条件都不成立，则按第三种情况处理。求出线段与窗口某边的交点，在交点处把线段一分为二，其中必有一段完全在窗口外，可弃之，对另一段则重复上述处理。



## 实现代码

#include <iostream>

#include<gl/glut.h>

using namespace std;

#define OVER 10e-6

int height = 700; int width = 700;

double xMax = 400, xMin = 200, yMax = 400, yMin = 200;//设置矩形的对角线两点的坐标

double X1, Y1, X2, Y2;//线段的端点

double cutx1, cuty1, cutx2, cuty2;//切割的点

bool bDrawLine = true;

bool beCut = true;

int getTheCodeValue(double x, double y)

{

int direct[4], codeValue = 0;

if (y > yMax) // 点在上边

{

direct[0] = 1;

direct[1] = 0;

}

else if (y < yMin)

{

direct[0] = 0;

direct[1] = 1;

}

else

{

direct[0] = 0;

direct[1] = 0;

}

if (x > xMax) // 点在矩形的右边

{

direct[2] = 1;

direct[3] = 0;

}

else if (x < xMin)

{

direct[2] = 0;

direct[3] = 1;

}

else

{

direct[2] = 0;

direct[3] = 0;

}

for (int i = 0; i < 4; i++)

{

codeValue += direct[i] \* pow(2, 3 - i);

// cout<< direct[i] \* pow(2, 3 - i)<<" ";

}

//cout << endl;

return codeValue;

}

bool judgeInRec(int codeValueOne, int codeValueTwo) // 判断这条线段是不是明显的在矩形内

{

if (codeValueOne == 0 && codeValueTwo == 0)

{

return true;

}

else

{

return false;

}

}

bool judgeOutRec(int codeValueOne, int codeValueTwo) // 判断这条线段明显的在矩形外

{

if (codeValueOne & codeValueTwo)//按位与

{

return true;

}

else

{

return false;

}

}

void findPoint(double lowx, double lowy, double highx, double highy)

{

double middlex = (lowx + highx) / 2.0;

double middley = (lowy + highy) / 2.0;

int valueCodeOne = getTheCodeValue(lowx, lowy);

int valueCodeTwo = getTheCodeValue(highx, highy);

if (judgeOutRec(valueCodeOne, valueCodeTwo)) // 线段明显在矩形外

{

//cout << "线段明显在矩形外" << endl;

return;

}

else

{

if (judgeInRec(valueCodeOne, valueCodeTwo))

{

//cout << "线段明显在矩形里面"<<endl;

return;

}

else

{

if (abs((pow(middlex, 2) + pow(middley, 2)) - (pow(lowx, 2) + pow(lowy, 2))) < OVER)//注意要用绝对值

{

if (beCut) {

cout << "第一个点";

cutx1 = middlex;

cuty1 = middley;

beCut = false;

}

else {

cout << "第二个点";

cutx2 = middlex;

cuty2 = middley;

beCut = true;

}

cout << middlex << " " << middley << endl;

return;

}

else

{

findPoint(lowx, lowy, middlex, middley);

findPoint(middlex, middley, highx, highy);

}

}

}

}

void mymouse(int button, int state, int x, int y) {

switch (button)

{

case GLUT\_LEFT\_BUTTON:

if (state == GLUT\_DOWN)

{

if (bDrawLine)

{

X1 = x;

Y1 = height - y;

bDrawLine = false;

glutPostRedisplay();//

}

else

{

X2 = x;

Y2 = height - y;

bDrawLine = true;

glutPostRedisplay();//

}

}

break;

default:

break;

}

}

void drawline\_color() {

glBegin(GL\_LINES);

glColor4f(0.0, 0.0, 0.0, 1.0);

glVertex2f(cutx1, cuty1);

glVertex2f(cutx2, cuty2);

glEnd();

}

void drawlines() {

glBegin(GL\_LINES);

glColor4f(1.0, 1.0, 1.0, 1.0);

glVertex2f(X1, Y1);

glVertex2f(X2, Y2);

glEnd();

findPoint(X1, Y1, X2, Y2);

drawline\_color();

//glutPostRedisplay();

}

void Reshape(GLsizei w, GLsizei h) {

// glViewport(0, 0, w, h);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluOrtho2D(0.0, w, 0.0, h);

width = w;

height = h;

}

void Display(void) {

//glClearColor(0.0, 0.0, 0.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

//glMatrixMode(GL\_MODELVIEW);

glBegin(GL\_POLYGON);

glColor4f(0.0, 1.0, 1.0, 1.0);

glVertex2f(200, 400);

glVertex2f(400, 400);

glVertex2f(400, 200);

glVertex2f(200, 200);

glEnd();

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (bDrawLine)

drawlines();

glutSwapBuffers();

}

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

{

glutInit(&argc, argv);

glutInitWindowPosition(100, 100);

glutInitWindowSize(width, height);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutCreateWindow("Line");

glutDisplayFunc(Display);

glutMouseFunc(mymouse);

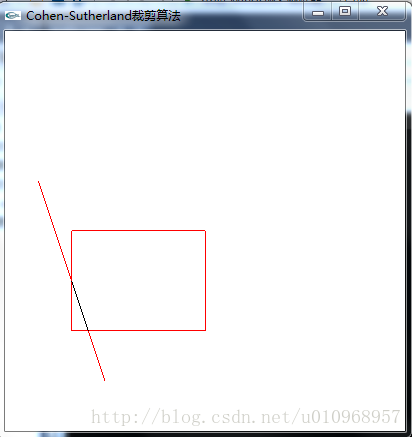
//glutMotionFunc(mymotion);

glutReshapeFunc(Reshape);

glutMainLoop();

return 0;

}



# 图形学初步--裁剪算法之Liang－Barsky算法

一、概念

裁剪是CG中许多重要问题的基础，裁剪最典型的用途是确定场景中或画面中位于给定区域之内的部分。由于在一个典型的场景之中，需要对大量的点、线段进行裁剪，因此裁剪算法的效率十分重要。

关于裁剪有一些很常见的算法，比如说Cohen-Sutherland线段细分裁剪算法、中点分割算法、Cyrus-Beck算法、Liang-Barsky算法。这篇文章重点讲Liang-Barsky算法。剩下的那些算法有空再补。

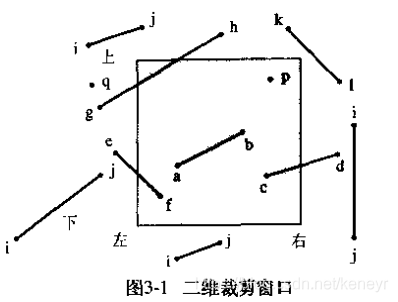
二、Liang-Barsky算法

参考博客：

https://www.cnblogs.com/keguniang/p/9688126.html

https://blog.csdn.net/pleasecallmewhy/article/details/8393445

首先知道直线和裁剪框的位置关系，如下图：



可以看到直线可能完全在裁剪框的外面，也可能完全在裁剪框的里面，或者和裁剪框相交。

所以是裁剪的规则是这样的：

**如果直线的两个端点都在窗口边界之内，如ab，则直线保留；**

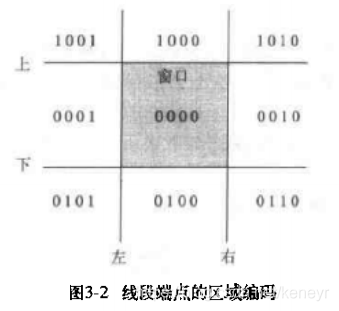
**如果直线的一个端点在窗口边界之内，另一个端点在窗口边界之外，如ef，则应从直线与边界的交点处裁剪掉边界之外的线段；**

**如果直线的两个端点都在窗口边界之外，有两种情况：**

**一种情况如ij，直线全部在窗口之外，应该全部裁减掉；**

**另一种情况如gh，直线横穿窗口边界，应该保留相交的片段，裁减掉剩余的片段；**

那么怎么去判断呢？ 在讲述判断步骤之前我们要了解一个基础理论，那就是编码



图中阴影部分是裁剪框，我们把这个区域分成了9份，采用了四位数标识线段的端点位于九个区域的哪个区域位置，从右往左数：

**第一个位置为1-------如果线段端点位于窗口左侧**

**第二个位置为1------如果线段端点位于窗口右侧**

**第三个位置为1------如果线段端点为窗口下面**

**第四个位置为1------如果线段位于窗口上面**

详细见网址：

[**https://blog.csdn.net/keneyr/article/details/83871170**](https://blog.csdn.net/keneyr/article/details/83871170)

最终实现的代码如下：

#include<stdio.h>

#include <windows.h>

#include <GL/glut.h>

#include <math.h>

#include<iostream>

using namespace std;

float xmin, xmax, ymin, ymax;

void myinit(void)

{

glShadeModel(GL\_FLAT);

glClearColor(1.0, 1.0, 1.0, 0.0);

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

gluOrtho2D(0.0, 1.0, 0.0, 1.0 \* (GLfloat)h / (GLfloat)w);

else

gluOrtho2D(0.0, 1.0 \* (GLfloat)w / (GLfloat)h, 0.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

}

int Clip(float p, float q, float\* tL, float\* tU)

{

int flag = 1;/\*flag为标志变量0表示舍弃1表示可见\*/

float r;

if (p < 0.0)

{

r = q / p;

if (r > \*tU)

flag = 0;

else if (r > \*tL) {

\*tL = r;/\*m取进入点最大参数值\*/

}

}

else if (p > 0.0) {

r = q / p;

if (r < \*tL)

flag = 0;

else if (r < \*tU) {

\*tU = r;/\*n取离开点的最小值\*/

}

}

else if (q < 0 && p == 0) //平行于边界而且在界外的线

flag = 0;

return flag;

}

void myclip()

// line clipping algorithm

{

float dx, dy, x1, tL, tU, x2, y1, y2;

tL = 0, tU = 1.0;

printf("请输入线段的两个顶点坐标x1,y1,x2,y2:\n");

//scanf("%f%f%f%f", &x1, &y1, &x2, &y2);

cin >> x1 >> y1 >> x2 >> y2;

glBegin(GL\_LINES);

glColor4f(0.0, 0.0, 0.0, 0.0);

glVertex2f(x1, y1); // line startpoint

glVertex2f(x2, y2); // line endpoint

glEnd();

dx = x2 - x1;

if (Clip(-dx, x1 - xmin, &tL, &tU))

if (Clip(dx, xmax - x1, &tL, &tU)) {

dy = y2 - y1;

if (Clip(-dy, y1 - ymin, &tL, &tU))

if (Clip(dy, ymax - y1, &tL, &tU))

{

if (tU < 1.0)

{

x2 = x1 + tU \* dx;//通过n求得裁剪后的p2端点

y2 = y1 + tU \* dy;

}

if (tL > 0.0)

{

x1 = x1 + tL \* dx;//通过m求得裁剪后的p1端点

y1 = y1 + tL \* dy;

}

glBegin(GL\_LINES);

glColor4f(1.0, 0.0, 0.0, 1.0);

glVertex2f(x1, y1); // clipped line startpoint

glVertex2f(x2, y2); // clipped line endpoint

glEnd();

}

}

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

printf("请分别输入矩形的左右下上边界：\n");

// scanf("%f%f%f%f", &xmin, &xmax, &ymin, &ymax);

cin >> xmin >> xmax >> ymin >> ymax;

glColor4f(1.0, 1.0, 0.0, 0.75);

glBegin(GL\_POLYGON);

glVertex2f(xmin, ymin); // Bottom Left

glVertex2f(xmax, ymin); // Bottom Left

glVertex2f(xmax, ymax); // Bottom Right

glVertex2f(xmin, ymax); // Bottom Right

glEnd();

myclip();

glFlush();

}

/\* Main Loop

\* Open window with initial window size, title bar,

\* RGBA display mode, and handle input events.

\*/

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGBA);

//define size and the relative positon of the applicaiton window on the display

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

//init the defined window with "argv[1]" as topic showed on the top the window

glutCreateWindow(argv[0]);

// opengl setup

myinit();

//define callbacks

glutDisplayFunc(display);

glutReshapeFunc(myReshape);

//enter the loop for display

glutMainLoop();

return 1;

}

# https://img-blog.csdnimg.cn/20181109082236386.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L2tlbmV5cg==,size_16,color_FFFFFF,t_70

# 基于OpenGL的直线段的裁剪算法

在Opengl应用框架下实现C-S裁剪算法或L-B裁剪算法。完成一个四边形对两条线段的裁剪：四边形的左上角和右下角顶点分别为（100，100），（300，200），线段1的两个端点为（150,50）,(250,150)。

实验代码-Cohen-Sutherland裁剪算法

#include <GL/glut.h>

#include <cstdio>

#define LEFT 1

#define RIGHT 2

#define BOTTOM 4

#define TOP 8

int x1=150,y1=50, x2=50, y2=250, XL=100, XR=300, YB=100, YT=200; //(x1,y1)、(x2,y2)为直线段的端点，XL为左边界，XR为右边界，YB为下边界，YT为上边界

int x1\_init = 150, y1\_init = 50, x2\_init = 50, y2\_init = 250; //将直线段端点备份，以便画出裁剪前的直线段

int encode(int x,int y)

{

int c = 0;

if (x < XL) c |= LEFT;

if (x > XR) c |= RIGHT;

if (y < YB) c |= BOTTOM;

if (y > YT) c |= TOP;

return c;

}

void CS\_LineClip() //Cohen-Sutherland裁剪算法

{

int x, y;

int code1, code2, code;

code1 = encode(x1, y1);

code2 = encode(x2, y2);

while (code1 != 0 || code2 != 0)

{

if (code1 & code2)

return;

if (code1 != 0)

code = code1;

else

code = code2;

if (LEFT & code)

{

x = XL;

y = y1 + (y2 - y1)\*(XL - x1) / (x2 - x1);

}

else if (RIGHT & code)

{

x = XR;

y = y1 + (y2 - y1)\*(XR - x1) / (x2 - x1);

}

else if (BOTTOM & code)

{

y = YB;

x = x1 + (x2 - x1)\*(YB - y1) / (y2 - y1);

}

else if (TOP & code)

{

y = YT;

x = x1 + (x2 - x1)\*(YT - y1) / (y2 - y1);

}

if (code == code1)

{

x1 = x; y1 = y; code1 = encode(x1, y1);

}

else

{

x2 = x; y2 = y; code2 = encode(x2, y2);

}

}

}

void init() //初始化函数

{

glClearColor(1.0, 1.0, 1.0, 0.0); //设置背景颜色

glMatrixMode(GL\_PROJECTION); // 设置投影参数

gluOrtho2D(0.0, 600.0, 0.0, 400.0); // 设置场景的大小

CS\_LineClip(); //执行一次裁剪算法

}

void mydisplay() //显示函数

{

//绘制方形边界

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glPointSize(2);

glBegin(GL\_LINE\_LOOP);

glVertex2i(XL, YT);

glVertex2i(XL, YB);

glVertex2i(XR, YB);

glVertex2i(XR, YT);

glEnd();

glFlush();

//绘制未裁剪前的线段

glBegin(GL\_LINES);

glVertex2i(x1\_init, y1\_init);

glVertex2i(x2\_init, y2\_init);

glEnd();

glFlush();

//绘制裁剪后的线段

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2i(x1, y1);

glVertex2i(x2, y2);

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(400, 400);

glutCreateWindow("Cohen-Sutherland裁剪算法");

init();

glutDisplayFunc(&mydisplay);

glutMainLoop();

return 0;

}