# q3

Thursday, May 02, 2024 9:09 PM

3) Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

```
// pg 134

// case 1 --> trivial acceptance

// case 2 --> trivial rejection

/*
      |             |
      |             |
      |             |
      |             |()
----------------------------------------
      |             |
      |             |
      | 0000        |
      |             |
      |             |
----------------------------------------
      |             |
      |             |
      |             |
      |             |
*/

#include <iostream>
#include <conio.h>
#include <math.h>
#include "dda.cpp"

#include <graphics.h>

using namespace std;
typedef unsigned int code;

double ymin = 100;
double ymax = 300;

double xmin = 100;
double xmax = 500;



enum
{
   TOP = 0x8,
   BOTTOM = 0x4,
   LEFT = 0x1,
   RIGHT = 0x2
};
```

```
class clip_window
{

public:
   double x_max, y_max;
   double x_min, y_min;




   clip_window()
   {

      // this->x_max = 2.0 / 3.0 * (getmaxx());
      // this->y_max = 2.0 / 3.0 * (getmaxy());
      // this->x_min = 1.0 / 3.0 * (getmaxx());
      // this->y_min = 1.0 / 3.0 * (getmaxy());


      this->x_max = xmax;
      this->y_max = ymax;
      this->x_min = xmin;
      this->y_min = ymin;
   }


   void print_points()
   {
      cout << "x max : " << x_max << " y max : " << y_max << endl;
      cout << "x min : " << x_min << " y min : " << y_min << endl;
   }




};


void find_code(double *p, clip_window *clip_w, code &c)
{
   if (p[1] > clip_w->y_max)
   { // top check
      c |= TOP;

   }
   else if (p[1] < clip_w->y_min)
   {
      c |= BOTTOM;

   }


   if (p[0] > clip_w->x_max)
   {
      c |= RIGHT;

   }
   else if (p[0] < clip_w->x_min)
   {
      c |= LEFT;

   }
}


void cohen_suther_clipp(double x1, double y1, double x2, double y2
,clip_window *clip_w)
{
```

```
    double *p1 = new double(2);
    double *p0 = new double(2);


    // defining two point of LINE


    p1[0] = x1;
    p1[1] = y1;


    p0[0] = x2;
    p0[1] = y2;


  //  clip_window *obj = new clip_window();
    code code1 = 0;
    code code2 = 0;


    find_code(p0, clip_w, code1);
    find_code(p1, clip_w, code2);

    int done=0;
    int accept=0;




    int itr = 1;




    do{


    cout<<"iteration no. "<<itr<<endl;
    itr++;


    code1 = 0;
    code2 = 0;
    find_code(p0, clip_w, code1);
    find_code(p1, clip_w, code2);




        if(!(code1|code2)){    // this is case when both code becomes 0000
            accept=1;
            done=1;
            break;
        }else if(code1 & code2){  // this is case when one LOGICAL AND IS
NON ZERO , trivial rejetced
            done=1;
            break;
        }
        else{


    double x, y;

    code outside = code1 ? code1 : code2;  // we will use one of the
outcode which is non zero


  if(code1 == outside){
cout<<"outside is for  :  x : "<<p0[0]<<" y : "<<p0[1]<<endl;
  }else{
```

```
    cout<<"outside is for  :  x : "<<p1[0]<<" y : "<<p1[1]<<endl;
  }



       if (outside & TOP)  // if it is non zero it means that the line has
intersection with this clippping edge.
       {
          cout<<"TOP CLIPPING"<<endl;
          x = p0[0] + ((p1[0] - p0[0]) *  (clip_w->y_max - p0[1]) )/ (p1[1] -
p0[1])  ;
          y = clip_w->y_max;



       }
       else if (outside & BOTTOM)
       {
          cout<<"BOTTOM CLIPPING"<<endl;
          x = p0[0] + (clip_w->y_min - p0[1]) * ((p1[0] - p0[0]) / (p1[1] -
p0[1]));
          y = clip_w->y_min;



       }
       else if (outside & RIGHT)
       {



          cout<<"RIGHT CLIPPING"<<endl;
          y = p0[1] + ((p1[1] - p0[1])*(clip_w->x_max - p0[0])) / (p1[0] -
p0[0]);
          x = clip_w->x_max;
       }
       else
       {
          cout<<"LEFT CLIPPING"<<endl;
          x = clip_w->x_min;
          y = p0[1] + ((p1[1] - p0[1]) / (p1[0] - p0[0])) * (clip_w->x_min -
p0[0]);
       }



       if(outside == code1){
       p0[0]=x;
       p0[1]=y;

       }
       else{
          p1[0]=x;
          p1[1]=y;

       }



       if(code1 == outside){
          p0[0] = x;          //changing p0 to new point ,
          p0[1] = y;



       }else{                 // changing p1 to new point ,
          p1[0]=x;
          p1[1]=y;
       }

       }


    }while(done == 0);


 if(accept){
```

```
    // draw line


    cout<<"x1 : "<<p0[0]<<"  , y1 : "<<p0[1]<<endl;
    cout<<"x2 : "<<p1[0]<<"  , y2 : "<<p1[1]<<endl;


    dda(p0[0],p0[1] , p1[0], p1[1],RED);
}



}

void drawWindow(clip_window *obj){
    dda(obj->x_min,obj->y_min,obj->x_max,obj->y_min,RED);
    dda(obj->x_min,obj->y_max,obj->x_max,obj->y_max,RED);
    dda(obj->x_min,obj->y_min,obj->x_min,obj->y_max,RED);
    dda(obj->x_max,obj->y_min,obj->x_max,obj->y_max,RED);
}

int main()
{









    int gd = DETECT, gm;
    char pathtodriver[] = "";
    initgraph(&gd, &gm, pathtodriver);


    clip_window *obj = new clip_window();




   drawWindow(obj);
   dda(10,300,200,200, GREEN);

   cohen_suther_clipp(10,300,200,200,obj);




    getch();
    closegraph();
}
```