q4

Thursday, May 02, 2024 9:09 PM

4) Write a program to clip a polygon using Sutherland Hodgeman algorithm.

```
#include <iostream>
#include <vector>
#include <utility>
#include  <conio.h>
#include  <math.h>


#include "dda.cpp"



#include <graphics.h>


using namespace std;


// function for in and out check for any vertex , input a vertex and a clipping edge
// function to find a inttersection


int in_out_check(int x,int y,int edge,int x1,int y1,int x2,int y2){
    //out is 1
    //in is 0
    //bottom
    if(edge==0){
       return (y >= y1) ? 0 : 1;
    }else if(edge==1){  //right
        return (x <= x1) ? 0 : 1;
    }else if(edge ==2){  //top
        return (y <= y1) ? 0 : 1;
    }else{    // left
        return (x >= x1) ? 0 : 1;
    }
}


class Point {
    public:
        double x;
        double y;
    Point(double x1,double y1){
            x=x1;
            y=y1;
    }
};


void findIntersection( Point* p1, const Point* p2, Point* p3, Point* p4, Point *pp) {


    double x=0;
    double y =0;


     if((p1->x - p2->x) == 0){
         x = p1->x;


            if((p3->y - p4->y) == 0){
                y = p3->y;
            }else{
                 double m2 = (p4->y - p3->y) / (p4->x - p3->x);
                 double b2 = p3->y - (m2 * p3->x);
                 y = m2*x + b2;
            }
                pp->x = x;
                pp->y = y;

            return;
    }


    double m1 = (p2->y - p1->y) / (p2->x - p1->x);
    double m2 = (p4->y - p3->y) / (p4->x - p3->x);
    double b1 = p1->y - m1 * p1->x;
    double b2 = p3->y - m2 * p3->x;


  // p1 , p2 are point of window





if(p3->x - p4->x == 0){
   // vertical lines
   x = p3->x;
}else{
    x =(b2 - b1) / (m1 - m2);
}


   y = m1 * x + b1;
    pp->x = x;
    pp->y = y;
}


void print_points( vector<pair<double, double> > points){
    cout<<"size :: "<<points.size()<<endl;
    for(int i=0;i<points.size();i++){
        cout<<" "<<points[i].first<<" , "<<points[i].second<<endl;
    }
}
void clipping(int x1,int y1, int x2,int y2, vector<pair<double, double> >
points,int edge,vector<pair<double, double> >& new_points ){


    // loop for each vertex
    int v1 = -1;
    int v2 = -1;
    Point *p1 =new Point(x1,y1);
    Point *p2 =new Point(x2,y2);
    Point *p3 ;
    Point *p4 ;




    for(int i=0;i<(points.size()-1);i++){


        v1 = in_out_check(points[i].first, points[i].second,edge,x1,y1,x2,y2);
        v2 =
in_out_check(points[i+1].first,points[i+1].second,edge,x1,y1,x2,y2);
        cout<<"v1 , v2  "<<i<<" , "<<i+1<<" ::  "<<v1<<" "<<v2<<endl;


        p3 = new Point(points[i].first, points[i].second);
        p4 = new Point(points[i+1].first,points[i+1].second);

            if(v1 == 0 & v2==0){ // in to in
            // keep destination
             new_points.push_back(make_pair(p4->x,p4->y));
            }else if(v1==0 & v2==1){ // in to out , keep intersection
            // find intersection
            Point *pp = new Point(0,0);


            findIntersection(p1, p2, p3, p4,pp);
            new_points.push_back(make_pair(pp->x,pp->y));


            }else if(v1 ==1 & v2 ==0){ // out to in  . keep both


            // find intersection
            Point *pp = new Point(0,0);
            findIntersection(p1, p2, p3, p4,pp);
            new_points.push_back(make_pair(pp->x,pp->y));   // intersetion
            new_points.push_back(make_pair(p4->x,p4->y));  // destination


            }else{      // out to out
            }
    }


   delete p1;
   delete p2;
   delete p3;
   delete p4;


    new_points.push_back(make_pair(new_points[0].first,
new_points[0].second ));




}


void copy_vector(   vector<pair<double, double> >& points,
vector<pair<double, double> >new_points){


    for(int i=0;i<new_points.size();i++){
        points.push_back(make_pair(new_points[i].first,
new_points[i].second));
    }


}


void draw_object(  vector<pair<double, double> >& points,int colorr){
    int x1=0;
    int y1=0;


    int x2=0;
    int y2=0;


      x1 = points[0].first;
      y1 = points[0].second;


     for (int i = 1; i < points.size() ; ++i) {


            x2 = points[i].first;
            y2 = points[i].second;


            dda(x1,y1,x2,y2,colorr);
        x1 = x2;
        y1 = y2;


        // draw and switch
    }


}


int main(){


    vector<pair<double, double> >points ;
      vector<pair<double, double> > points_original ;






    points.push_back(make_pair(230,270));//1
    points.push_back(make_pair(300,170));//2


    points.push_back(make_pair(390,160));//3
    points.push_back(make_pair(320,80));//4


    points.push_back(make_pair(300,0));//5
    points.push_back(make_pair(230,70));




    points.push_back(make_pair(150,0));
    points.push_back(make_pair(150,80));


    points.push_back(make_pair(50,150));
    points.push_back(make_pair(160,170));


    points.push_back(make_pair(230,270));


    copy_vector(points_original,points);


    vector<pair<double, double> >window ;
    window.push_back(make_pair(100,50));
    window.push_back(make_pair(350,50));
    window.push_back(make_pair(350,200));
    window.push_back(make_pair(100,200));
    window.push_back(make_pair(100,50));


    int v = 10;
    int edges = 10;


    // loop for each edge of window
    // send one pair of coords


   for(int i=0; i<4; i++){
        vector<pair<double, double> >new_points;
        clipping(window[i].first,window[i].second, window[i+1].first,
window[i+1].second ,  points, i, new_points);
```

```
            points.clear();
            copy_vector(points,new_points);


            new_points.clear();




            print_points(points);


            cout<<"iteration no. :: "<<i<<endl;
         // copy the data from one to other

     }




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


      draw_object(window,BLUE);
      delay(1000);
     draw_object(points_original,RED);
      delay(1000);
     draw_object(points,GREEN);
      // bottom , right , top , left
      //    0 ,     1 ,     2,    3



   getch();
   closegraph();
      return 0;
   }
```