**Graphics**

Lab Assignment



**Submitted to –**

**Komal Singh Gill**

**Submitted By–**

**Chetan Singh Chauhan**

**Roll no. – 201880014**

**Group – 2 MCA 2**

**PROBLEMS**

1. DDA Line Algorithm
2. Bresenham Line Algorithm
3. Midpoint Circle Algorithm
4. Boundary Fill Algorithm
5. Flood Fill Algorithm
6. 2D- Transformation
   1. Translation
   2. Scaling
   3. Rotation
7. Cohen Sutherland Algorithm
8. Liang Barsky Algorithm
9. Sutherland Hodgeman Algorithm
10. Weiler Atherton Algorithm

**Program of DDA Line Algorithm**

#include<stdio.h>

#include<math.h>

#include<GL/glut.h>

void disp(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

float x2=200,y2=80,x1=0,y1=0;

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

if(m<1)

{

glVertex2d(x1,y1);

while(x1<x2)

{

x1++;

y1=y1+m;

glVertex2d(x1,round(y1));

}

}

else

{

glVertex2d(x1,y1);

while(y1<y2)

{

y1++;

x1=x1+(1/m);

glVertex2d(round(x1),y1);

}

}

glEnd();

glFlush();

}

void Init()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(0.0,0.0,0.0);

gluOrtho2D(0,640,0,480);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(400,300);

glutInitWindowPosition(100,300);

glutCreateWindow("DDA\_LINE\_ALGO");

Init();

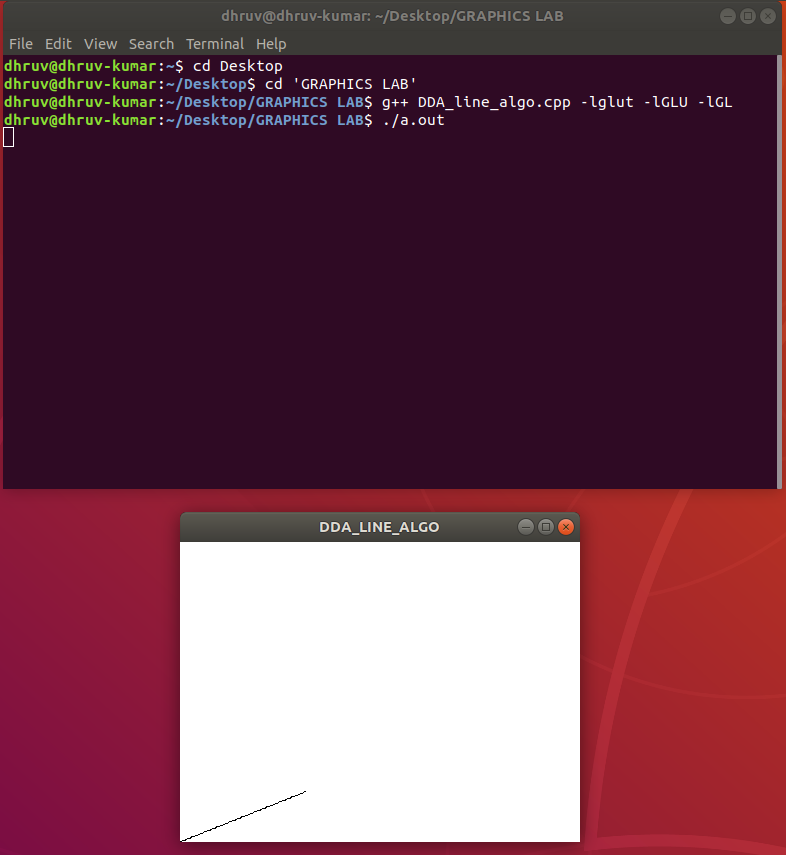
glutDisplayFunc(disp);

glutMainLoop();

return 0;

}

**OUTPUT**



**Program of Bresenham Algorithm**

#include<stdio.h>

#include<math.h>

#include<GL/glut.h>

void disp(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

int x1=100,x2=600,y1=150,y2=700;

int x=x1,y=y1,dx,dy,p;

dx=x2-x1;

dy=y2-y1;

p=2\*dy-dx;

glVertex2d(x,y);

while(x<x2)

{

x=x+1;

if(p>=0)

{

y=y+1;

glVertex2d(x,y);

p=p+2\*dy-2\*dx;

}

else

{

p=p+2\*dy-2\*dx;

}

x=x+1;

}

glEnd();

glFlush();

}

void Init()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(0.0,0.0,0.0);

gluOrtho2D(0,640,0,480);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(400,300);

glutInitWindowPosition(100,300);

glutCreateWindow("BRESENHAM\_LINE\_ALGO");

Init()

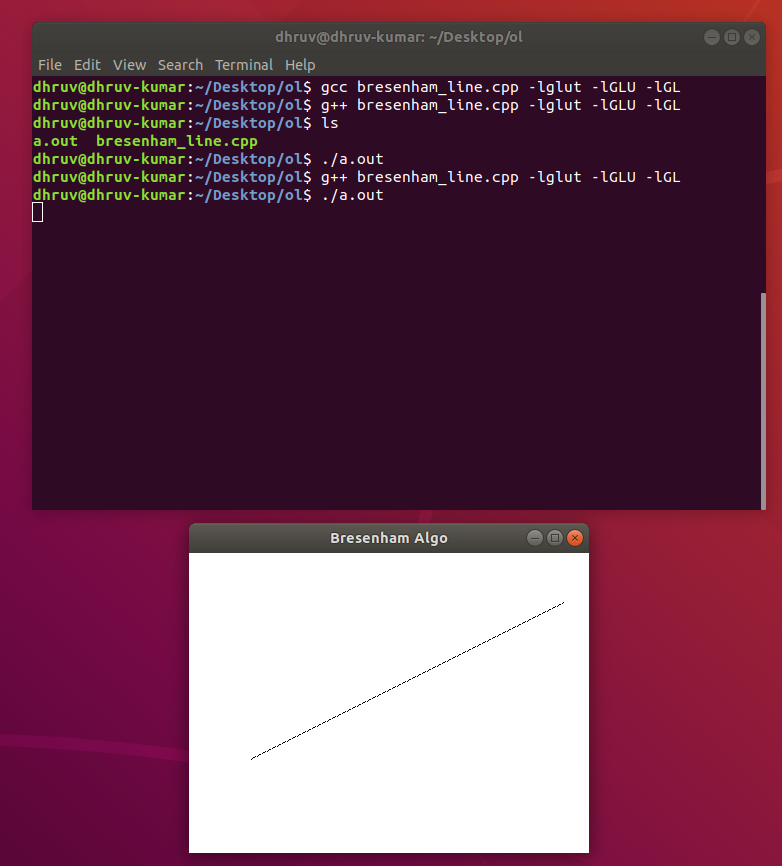
glutDisplayFunc(disp);

glutMainLoop();

return 0;

}

**Output**



**Program of Mid-point Circle Algorithm**

#include<stdio.h>

#include<math.h>

#include<GL/glut.h>

void disp(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

int x=0,y=250,p,r;

r=y;

p=1-r;

glVertex2d(x,y);

while(x<=y)

{

if(p<0)

{

p=p+2\*x+2\*1+1;//written in another format(p=p+2(x+1)+1)

}

else

{

p=p+2\*x+2\*1-2\*y+2\*1+1;//written in another format(p=p+2(x+1)-2(y-1)+1)

y--;

}

x++;

glVertex2d(x,y);

glVertex2d(-x,y);

glVertex2d(x,-y);

glVertex2d(-x,-y);

glVertex2d(y,x);

glVertex2d(-y,x);

glVertex2d(y,-x);

glVertex2d(-y,-x);

}

glEnd();

glFlush();

}

void Init()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(0.0,0.0,0.0);

gluOrtho2D(0,640,0,480);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(400,300);

glutInitWindowPosition(100,300);

glutCreateWindow("MID\_POINT\_CIRCLE");

Init();

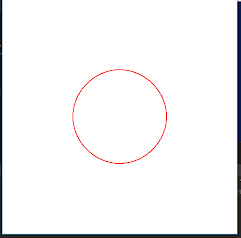
glutDisplayFunc(disp);

glutMainLoop();

return 0;

}

**Output**



=

**Flood Filling Algorithm**

#include <math.h>

#include <GL/glut.h>

struct Point {

GLint x;

GLint y;

};

struct Color {

GLfloat r;

GLfloat g;

GLfloat b;

};

void init() {

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0, 0.0, 0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 640, 0, 480);

}

Color getPixelColor(GLint x, GLint y) {

Color color;

glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, &color);

return color;

}

void setPixelColor(GLint x, GLint y, Color color) {

glColor3f(color.r, color.g, color.b);

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

glFlush();

}

void floodFill(GLint x, GLint y, Color oldColor, Color newColor) {

Color color;

color = getPixelColor(x, y);

if(color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b)

{

setPixelColor(x, y, newColor);

floodFill(x+1, y, oldColor, newColor);

floodFill(x, y+1, oldColor, newColor);

floodFill(x-1, y, oldColor, newColor);

floodFill(x, y-1, oldColor, newColor);

}

return;

}

void onMouseClick(int button, int state, int x, int y)

{

Color newColor = {1.0f, 0.0f, 0.0f};

Color oldColor = {1.0f, 1.0f, 1.0f};

floodFill(320, 240, oldColor, newColor);

}

void draw\_circle(Point pC, GLfloat radius) {

GLfloat step = 1/radius;

GLfloat x, y;

for(GLfloat theta = 0; theta <= 360; theta += step) {

x = pC.x + (radius \* cos(theta));

y = pC.y + (radius \* sin(theta));

glVertex2i(x, y);

}

}

void display(void) {

Point pt = {320, 240};

GLfloat radius = 50;

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

draw\_circle(pt, radius);

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(640, 480);

glutInitWindowPosition(200, 200);

glutCreateWindow("Flood\_Fill");

init();

glutDisplayFunc(display);

glutMouseFunc(onMouseClick);

glutMainLoop();

return 0;

}

**Output**

**Boundary Fill Algorithm**

#include <iostream>

#include <math.h>

#include <time.h>

#include <GL/glut.h>

using namespace std;

void delay(float ms){

clock\_t goal = ms + clock();

while(goal>clock());

}

void init(){

glClearColor(1.0,1.0,1.0,0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0,640,0,480);

}

void bound\_it(int x, int y, float\* fillColor, float\* bc){

float color[3];

glReadPixels(x,y,1.0,1.0,GL\_RGB,GL\_FLOAT,color);

if((color[0]!=bc[0] || color[1]!=bc[1] || color[2]!=bc[2])&&(

color[0]!=fillColor[0] || color[1]!=fillColor[1] || color[2]!=fillColor[2])){

glColor3f(fillColor[0],fillColor[1],fillColor[2]);

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

bound\_it(x+1,y,fillColor,bc);

bound\_it(x-2,y,fillColor,bc);

bound\_it(x,y+2,fillColor,bc);

bound\_it(x,y-2,fillColor,bc);

}

}

void mouse(int btn, int state, int x, int y){

y = 480-y;

if(btn==GLUT\_LEFT\_BUTTON)

{

if(state==GLUT\_DOWN)

{

float bCol[] = {1,0,0};

float color[] = {0,0,1};

//glReadPixels(x,y,1.0,1.0,GL\_RGB,GL\_FLOAT,intCol);

bound\_it(x,y,color,bCol);

}

}

}

void world(){

glLineWidth(3);

glPointSize(2);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(150,100);

glVertex2i(300,300);

glVertex2i(450,100);

glEnd();

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(640,480);

glutInitWindowPosition(200,200);

glutCreateWindow("Boundary fill");

glutDisplayFunc(world);

glutMouseFunc(mouse);

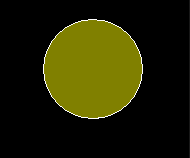
init();

glutMainLoop();

return 0;

}

**Output**

****

**Program of 2D- Transformations**

**1) Translation**

**2) Rotation**

**3) Scaling**

**Translation**

#include <GL/glut.h>

#include<iostream>

using namespace std;

int A[2][2], T[2];

void scaling(void)

{

glBegin(GL\_LINES);

int j = 0;

for (int i = 0; i < 2; i++)

{

glVertex2i(A[i][j], A[i][j + 1]);

}

j = 0;

for (int i = 0; i < 2; i++)

{

A[i][j] = A[i][j] + T[0];

A[i][j + 1] = A[i][j + 1] + T[1];

}

j = 0;

for (int i = 0; i < 2; i++)

{

glVertex2i(A[i][j], A[i][j + 1]);

}

glEnd();

glFlush();

}

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

{

cout <<"enter the values of x1,y1,x2,y2";

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

cin >> A[i][j];

}

}

cout <<"enter the values of tx,ty";

for (int i = 0; i <= 1; i++)

{

cin >> T[i];

}

glutInit( & argc, argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(400, 300);

glutInitWindowPosition(100, 100);

glutCreateWindow("translation!");

gluOrtho2D(0, 400, 0, 300);

glutDisplayFunc(scaling);

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0, 0.0, 0.0, 0);

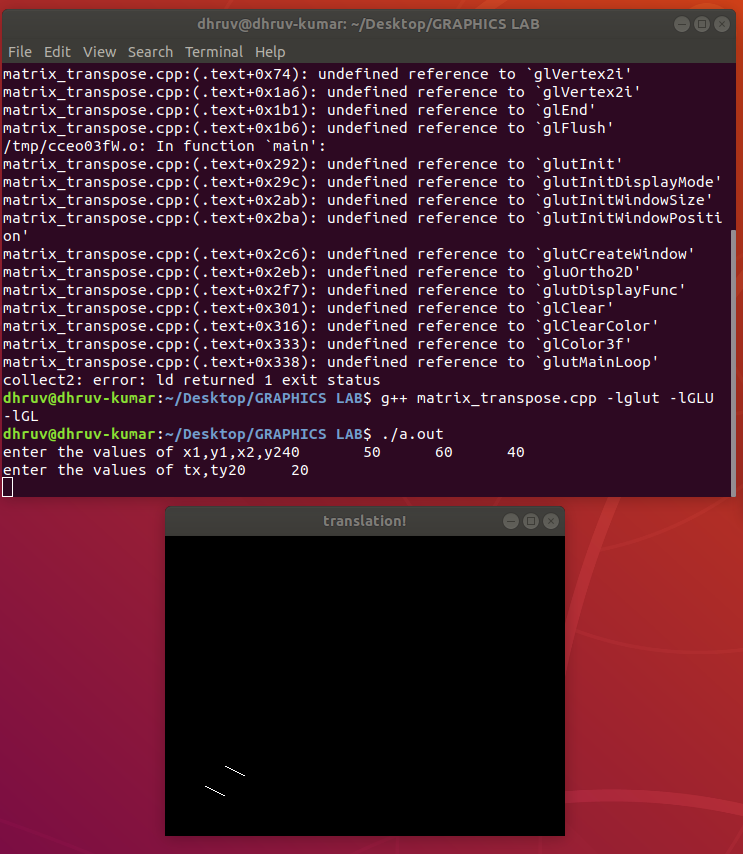
glColor3f(1.0, 1.0, 1.0);

glutMainLoop();

return 0;

}

**Output**



**Cohen Sutherland Clipping Algorithm**

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

#include<iostream>

void display();

using namespace std;

float xmin=-100;

float ymin=-100;

float xmax=100;

float ymax=100;

float xd1,yd1,xd2,yd2;

void init(void)

{

glClearColor(0.0,0,0,0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(-300,300,-300,300);

}

int code(float x,float y)

{

int c=0;

if(y>ymax)c=8;

if(y<ymin)c=4;

if(x>xmax)c=c|2;

if(x<xmin)c=c|1;

return c;

}

void cohen\_Line(float x1,float y1,float x2,float y2)

{

int c1=code(x1,y1);

int c2=code(x2,y2);

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

while((c1|c2)>0)

{

if((c1 & c2)>0)

{

exit(0);

}

float xi=x1;float yi=y1;

int c=c1;

if(c==0)

{

c=c2;

xi=x2;

yi=y2;

}

float x,y;

if((c & 8)>0)

{

y=ymax;

x=xi+ 1.0/m\*(ymax-yi);

}

else

if((c & 4)>0)

{

y=ymin;

x=xi+1.0/m\*(ymin-yi);

}

else

if((c & 2)>0)

{

x=xmax;

y=yi+m\*(xmax-xi);

}

else

if((c & 1)>0)

{

x=xmin;

y=yi+m\*(xmin-xi);

}

if(c==c1)

{

xd1=x;

yd1=y;

c1=code(xd1,yd1);

}

if(c==c2)

{

xd2=x;

yd2=y;

c2=code(xd2,yd2);

}

}

display();

}

void mykey(unsigned char key,int x,int y)

{

if(key=='c')

{ cout<<"Hello";

cohen\_Line(xd1,yd1,xd2,yd2);

glFlush();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,1.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(xmin,ymin);

glVertex2i(xmin,ymax);

glVertex2i(xmax,ymax);

glVertex2i(xmax,ymin);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2i(xd1,yd1);

glVertex2i(xd2,yd2);

glEnd();

glFlush();

}

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

{

printf("Enter line co-ordinates:");

cin>>xd1>>yd1>>xd2>>yd2;

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(600,600);

glutInitWindowPosition(0,0);

glutCreateWindow("Cohen Sutherland");

glutDisplayFunc(display);

glutKeyboardFunc(mykey);

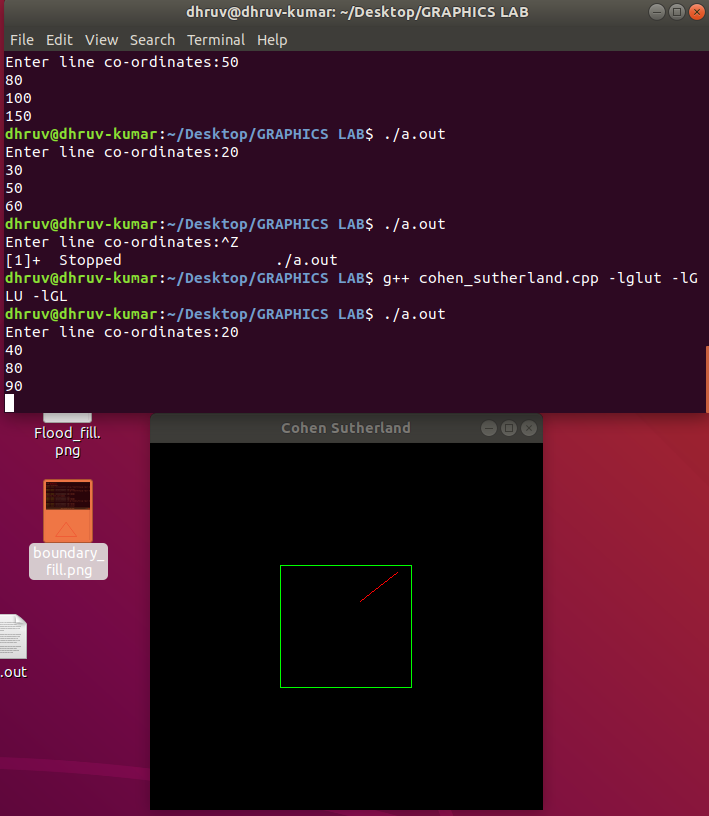
init();

glutMainLoop();

return 0;

}

**Output**

****

**Liang Barsky Line Clipping**

#include<GL/glut.h>

int x1=-80,x2=60,y3=-80,y2=50;

float u1=0,u2=1;

int xmin=-50,ymin=-50,xmax=50,ymax=50;

double p[4],q[4];

void init()

{

glClearColor(1.0,1.0,1.0,1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(-320,320,-240,240);

}

void clip(int x1,int y1,int x2,int y2)

{

int dx=x2-x1,dy=y2-y1,i;

double t;

p[0]=-dx;q[0]=x1-xmin;

p[1]=dx;q[1]=xmax-x1;

p[2]=-dy;q[2]=y1-ymin;

p[3]=dy;q[3]=ymax-y1;

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

{

if(p[i]==0 && q[i]<0)

return;

if(p[i]<0)

{

t=(q[i])/(p[i]);

if(t>u1 && t<u2)

{

u1=t;

}

}

else if(p[i]>0)

{

t=(q[i])/(p[i]);

if(t>u1 && t<u2)

{

u2=t;

}

}

}

if(u1<u2)

{

x1=x1+u1\*(x2-x1);

y1=y1+u1\*(y2-y1);

x2=x1+u2\*(x2-x1);

y2=y1+u2\*(y2-y1);

glBegin(GL\_LINES);

glVertex2i(x1,y1);

glVertex2i(x2,y2);

glEnd();

glFlush();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2i(x1,y3);

glVertex2i(x2,y2);

glEnd();

glFlush();

}

void myKey(unsigned char key,int x,int y)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

if(key=='c')

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

glColor3f(0.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2i(-50,-50);

glVertex2i(-50,50);

glVertex2i(-50,50);

glVertex2i(50,50);

glVertex2i(50,50);

glVertex2i(50,-50);

glVertex2i(50,-50);

glVertex2i(-50,-50);

glEnd();

glFlush();

clip(::x1,y3,x2,y2);

}

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(640,480);

glutInitWindowPosition(0,0);

glutCreateWindow("Clip");

glutDisplayFunc(display);

glutKeyboardFunc(myKey);

init();

glutMainLoop();

return 0;

}

Sutherland Hodgeman Algorithm

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

int xmin=-100;

int xmax=100;

int ymin=-100;

int ymax=100;

int xmi,xma,ymi,yma;

int cut;

void display()

{

glColor3f(0.0,1.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POLYGON);

glVertex2i(xmin,ymin);

glVertex2i(xmax,ymin);

glVertex2i(xmax,ymax);

glVertex2i(xmin,ymax);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(xmi,ymi);

glVertex2i(xma,ymi);

glVertex2i(xma,yma);

glVertex2i(xmi,yma);

glEnd();

glutSwapBuffers();

}

void clip()

{

while(xmi<xmin||xma>xmax||ymi<ymin||yma>ymax)

{

if(xmi<xmin)

{

cut=xmin-xmi;

xmi=xmi+cut;

}

else if(xma>xmax)

{

cut=xma-xmax;

xma=xma-cut;

}

else if(ymi<ymin)

{

cut=ymin-ymi;

ymi=ymi+cut;

}

else if(yma>ymax)

{

cut=yma-ymax;

yma=yma-cut;

}

glutPostRedisplay();

}

}

void mykey(unsigned char key,int x,int y)

{

if(key=='c')

{

clip();

}

}

void init(void)

{

glClearColor(0.0,0,0,0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(-300,300,-300,300);

}

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

{

printf("enter the points for left bottom of polygon:");

scanf("%d%d",&xmi,&ymi);

printf("Enter the points for right up of the polygon:");

scanf("%d%d",&xma,&yma);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(600,600);

glutInitWindowPosition(0,0);

glutCreateWindow("Clipping");

glutDisplayFunc(display);

glutKeyboardFunc(mykey);

init();

glutMainLoop();

return 0;

}

OUTPUT:

1. Weiler Atherton Algorithm

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <cstring> |
|  |

|  |
| --- |
| #include <cstdio> |
|  |

|  |
| --- |
| #include <fstream> |
|  |

|  |
| --- |
| #include <cstdlib> |
|  |

|  |
| --- |
| #include <ctime> |
|  |

|  |
| --- |
| #include <cmath> |
|  |

|  |
| --- |
| #include <vector> |
|  |

|  |
| --- |
| #include <list> |
|  |

|  |
| --- |
| #include <algorithm> |
|  |

|  |
| --- |
| #include <functional> |
|  |

|  |
| --- |
| #include <GL/glut.h> |
|  |

|  |
| --- |
| #include <GL/glu.h> |
|  |

|  |
| --- |
| #include <GL/gl.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #define Size 600 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| using namespace std; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| typedef float Color[3]; |
|  |

|  |
| --- |
| struct Point{ |
|  |

|  |
| --- |
| int x, y; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
| typedef struct IntersectionPoint { |
|  |

|  |
| --- |
| int pointFlag; |
|  |

|  |
| --- |
| int index0, index1; |
|  |

|  |
| --- |
| Point p; |
|  |

|  |
| --- |
| bool inFlag; |
|  |

|  |
| --- |
| int dis; |
|  |

|  |
| --- |
| }IP; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Polygon{ |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| vector<Point> pts; |
|  |

|  |
| --- |
| Polygon(void); |
|  |

|  |
| --- |
| ~Polygon(void); |
|  |

|  |
| --- |
| void drawPolygonLine(Color c); |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
| Polygon::Polygon(void){ |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| Polygon::~Polygon(void){ |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void Polygon::drawPolygonLine(Color c) { |
|  |

|  |
| --- |
| glColor3fv(c); |
|  |

|  |
| --- |
| glLineWidth(2.0); |
|  |

|  |
| --- |
| glBegin(GL\_LINE\_LOOP); |
|  |

|  |
| --- |
| int size = pts.size(); |
|  |

|  |
| --- |
| for (int i = 0; i < size; i++) |
|  |

|  |
| --- |
| glVertex2i(pts[i].x, pts[i].y); |
|  |

|  |
| --- |
| glEnd(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bool isPointInsidePolygon(Point p, Polygon& py) { |
|  |

|  |
| --- |
| int cnt = 0, size = py.pts.size(); |
|  |

|  |
| --- |
| for (int i = 0; i < size; i++) { |
|  |

|  |
| --- |
| Point p1 = py.pts[i]; |
|  |

|  |
| --- |
| Point p2 = py.pts[(i + 1) % size]; |
|  |

|  |
| --- |
| if (p1.y == p2.y) continue; |
|  |

|  |
| --- |
| if (p.y < min(p1.y, p2.y)) continue; |
|  |

|  |
| --- |
| if (p.y >= max(p1.y, p2.y)) continue; |
|  |

|  |
| --- |
| double x = (double)(p.y - p1.y) \* (double)(p2.x - p1.x) / (double)(p2.y - p1.y) + p1.x; |
|  |

|  |
| --- |
| if (x > p.x) cnt++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return (cnt % 2 == 1); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| int cross(Point& p0, Point& p1, Point& p2) { |
|  |

|  |
| --- |
| return ((p2.x - p0.x) \* (p1.y - p0.y) - (p1.x - p0.x) \* (p2.y - p0.y)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| bool onSegment(Point& p0, Point& p1, Point& p2) { |
|  |

|  |
| --- |
| int minx = min(p0.x, p1.x), maxx = max(p0.x, p1.x); |
|  |

|  |
| --- |
| int miny = min(p0.y, p1.y), maxy = max(p0.y, p1.y); |
|  |

|  |
| --- |
| if (p2.x >= minx && p2.x <= maxx && p2.y >= miny && p2.y <= maxy) return true; |
|  |

|  |
| --- |
| return false; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| bool segmentsIntersect(Point& p1, Point& p2, Point& p3, Point& p4) { |
|  |

|  |
| --- |
| int d1 = cross(p3, p4, p1); |
|  |

|  |
| --- |
| int d2 = cross(p3, p4, p2); |
|  |

|  |
| --- |
| int d3 = cross(p1, p2, p3); |
|  |

|  |
| --- |
| int d4 = cross(p1, p2, p4); |
|  |

|  |
| --- |
| if (((d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0)) && |
|  |

|  |
| --- |
| ((d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0))) |
|  |

|  |
| --- |
| return true; |
|  |

|  |
| --- |
| if (d1 == 0 && onSegment(p3, p4, p1)) return true; |
|  |

|  |
| --- |
| if (d2 == 0 && onSegment(p3, p4, p2)) return true; |
|  |

|  |
| --- |
| if (d3 == 0 && onSegment(p1, p2, p3)) return true; |
|  |

|  |
| --- |
| if (d4 == 0 && onSegment(p1, p2, p4)) return true; |
|  |

|  |
| --- |
| return false; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| Point getintersectPoint(Point p1, Point p2, Point p3, Point p4){ //未处理线段重合 |
|  |

|  |
| --- |
| Point p; |
|  |

|  |
| --- |
| int b1 = (p2.y - p1.y) \* p1.x + (p1.x - p2.x) \* p1.y; |
|  |

|  |
| --- |
| int b2 = (p4.y - p3.y) \* p3.x + (p3.x - p4.x) \* p3.y; |
|  |

|  |
| --- |
| int D = (p2.x - p1.x) \* (p4.y - p3.y) - (p4.x - p3.x) \* (p2.y - p1.y); |
|  |

|  |
| --- |
| int D1 = b2 \* (p2.x - p1.x) - b1 \* (p4.x - p3.x); |
|  |

|  |
| --- |
| int D2 = b2 \* (p2.y - p1.y) - b1 \* (p4.y - p3.y); |
|  |

|  |
| --- |
| p.x = D1 / D; |
|  |

|  |
| --- |
| p.y = D2 / D; |
|  |

|  |
| --- |
| return p; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void generateIntersectPoints(Polygon& pyclip, Polygon& py, list<IP>& iplist) { |
|  |

|  |
| --- |
| int clipSize = pyclip.pts.size(), pySize = py.pts.size(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for (int i = 0; i < clipSize; i++) { |
|  |

|  |
| --- |
| Point p1 = pyclip.pts[i]; |
|  |

|  |
| --- |
| Point p2 = pyclip.pts[(i + 1) % clipSize]; |
|  |

|  |
| --- |
| for (int j = 0; j < pySize; j++) { |
|  |

|  |
| --- |
| Point p3 = py.pts[j]; |
|  |

|  |
| --- |
| Point p4 = py.pts[(j + 1) % pySize]; |
|  |

|  |
| --- |
| if (segmentsIntersect(p1, p2, p3, p4)) { |
|  |

|  |
| --- |
| IP ip; |
|  |

|  |
| --- |
| ip.index0 = j; |
|  |

|  |
| --- |
| ip.index1 = i; |
|  |

|  |
| --- |
| ip.p = getintersectPoint(p1, p2, p3, p4); |
|  |

|  |
| --- |
| iplist.push\_back(ip); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| int getDistance(Point& p1, Point& p2) { |
|  |

|  |
| --- |
| return (p1.x - p2.x) \* (p1.x - p2.x) + (p1.y - p2.y) \* (p1.y - p2.y); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| bool distanceComparator(IP& ip1, IP& ip2) { |
|  |

|  |
| --- |
| return ip1.dis < ip2.dis; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void generateList(Polygon& py, list<IP>& iplist, list<IP>& comlist, int index) { |
|  |

|  |
| --- |
| int size = py.pts.size(); |
|  |

|  |
| --- |
| list<IP>::iterator it; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for (int i = 0; i < size; i++) { |
|  |

|  |
| --- |
| Point p1 = py.pts[i]; |
|  |

|  |
| --- |
| IP ip; |
|  |

|  |
| --- |
| ip.pointFlag = 0; |
|  |

|  |
| --- |
| ip.p = p1; |
|  |

|  |
| --- |
| comlist.push\_back(ip); |
|  |

|  |
| --- |
| list<IP> oneSeg; |
|  |

|  |
| --- |
| for (it = iplist.begin(); it != iplist.end(); it++) { |
|  |

|  |
| --- |
| if ((index == 0 && i == it->index0) || |
|  |

|  |
| --- |
| (index == 1 && i == it->index1)) { |
|  |

|  |
| --- |
| it->dis = getDistance(it->p, p1); |
|  |

|  |
| --- |
| it->pointFlag = 1; |
|  |

|  |
| --- |
| oneSeg.push\_back(\*it); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| oneSeg.sort(distanceComparator); |
|  |

|  |
| --- |
| for (it = oneSeg.begin(); it != oneSeg.end(); it++) |
|  |

|  |
| --- |
| comlist.push\_back(\*it); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void getPolygonPointInOut(list<IP>& polygonlist, Polygon& pyclip) { |
|  |

|  |
| --- |
| bool inFlag; |
|  |

|  |
| --- |
| list<IP>::iterator it; |
|  |

|  |
| --- |
| for (it = polygonlist.begin(); it != polygonlist.end(); it++) { |
|  |

|  |
| --- |
| if (it->pointFlag == 0) { |
|  |

|  |
| --- |
| if (isPointInsidePolygon(it->p, pyclip)) |
|  |

|  |
| --- |
| inFlag = true; |
|  |

|  |
| --- |
| else inFlag = false; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else { |
|  |

|  |
| --- |
| inFlag = !inFlag; |
|  |

|  |
| --- |
| it->inFlag = inFlag; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| bool operator==(Point& p1, Point& p2) { |
|  |

|  |
| --- |
| return p1.x == p2.x && p1.y == p2.y; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void getClipPointInOut(list<IP>& cliplist, list<IP>& polygonlist) { |
|  |

|  |
| --- |
| list<IP>::iterator it, it1; |
|  |

|  |
| --- |
| for (it = cliplist.begin(); it != cliplist.end(); it++) { |
|  |

|  |
| --- |
| if (it->pointFlag == 0) continue; |
|  |

|  |
| --- |
| for (it1 = polygonlist.begin(); it1 != polygonlist.end(); it1++) { |
|  |

|  |
| --- |
| if (it1->pointFlag == 0) continue; |
|  |

|  |
| --- |
| if (it->p == it1->p) it->inFlag = it1->inFlag; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void generateClipArea(list<IP>& polygonlist, list<IP>& cliplist){ |
|  |

|  |
| --- |
| list<IP>::iterator it, it1; |
|  |

|  |
| --- |
| Polygon py; |
|  |

|  |
| --- |
| Color c = {0.0, 0.0, 1.0}; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for (it = polygonlist.begin(); it != polygonlist.end(); it++) |
|  |

|  |
| --- |
| if (it->pointFlag == 1 && it->inFlag) break; |
|  |

|  |
| --- |
| py.pts.clear(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| while(true) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (it == polygonlist.end()) break; |
|  |

|  |
| --- |
| py.pts.push\_back(it->p); |
|  |

|  |
| --- |
| for (; it != polygonlist.end(); it++) { |
|  |

|  |
| --- |
| if (it->pointFlag == 1 && !it->inFlag) break; |
|  |

|  |
| --- |
| py.pts.push\_back(it->p); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| for (it1 = cliplist.begin(); it1 != cliplist.end(); it1++) |
|  |

|  |
| --- |
| if (it1->p == it->p) break; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for (; it1 != cliplist.end(); it1++) { |
|  |

|  |
| --- |
| if (it1->pointFlag == 1 && it1->inFlag) break; |
|  |

|  |
| --- |
| py.pts.push\_back(it1->p); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if (py.pts[0] == it1->p) { |
|  |

|  |
| --- |
| py.drawPolygonLine(c); |
|  |

|  |
| --- |
| py.pts.clear(); |
|  |

|  |
| --- |
| for (; it != polygonlist.end(); it++) |
|  |

|  |
| --- |
| if (it->pointFlag == 1 && it->inFlag) break; |
|  |

|  |
| --- |
| continue; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| for (; it != polygonlist.end(); it++) |
|  |

|  |
| --- |
| if (it->p == it1->p) break; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void weilerAtherton(Polygon& pyclip, Polygon& py){ |
|  |

|  |
| --- |
| list<IP> iplist, polygonlist, cliplist; |
|  |

|  |
| --- |
| generateIntersectPoints(pyclip, py, iplist); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| generateList(py, iplist, polygonlist, 0); |
|  |

|  |
| --- |
| generateList(pyclip, iplist, cliplist, 1); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| getPolygonPointInOut(polygonlist, pyclip); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| getClipPointInOut(cliplist, polygonlist); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| generateClipArea(polygonlist, cliplist); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void init() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| glClearColor(0.0, 0.0, 0.0, 0.0); |
|  |

|  |
| --- |
| glColor3f(1.0, 0.0, 0.0); |
|  |

|  |
| --- |
| glPointSize(1.0); |
|  |

|  |
| --- |
| glMatrixMode(GL\_PROJECTION); |
|  |

|  |
| --- |
| glLoadIdentity(); |
|  |

|  |
| --- |
| gluOrtho2D(0.0, Size - 1, 0.0, Size - 1); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void GenerateRandomSimplePolygon(Polygon &G, int M) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Point P; |
|  |

|  |
| --- |
| G.pts.clear(); |
|  |

|  |
| --- |
| for (int i = 0; i < M; ++ i) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| bool flag; |
|  |

|  |
| --- |
| do |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| P.x = rand() % Size; |
|  |

|  |
| --- |
| P.y = rand() % Size; |
|  |

|  |
| --- |
| flag = true; |
|  |

|  |
| --- |
| for (int j = 1; j < i - 1; ++ j) |
|  |

|  |
| --- |
| if (segmentsIntersect(G.pts[j - 1], G.pts[j], G.pts[i - 1], P)) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| flag = false; |
|  |

|  |
| --- |
| break; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| if (flag && i == M - 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for (int j = 2; j < i; ++ j) |
|  |

|  |
| --- |
| if (segmentsIntersect(G.pts[j - 1], G.pts[j], P, G.pts[0])) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| flag = false; |
|  |

|  |
| --- |
| break; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } while (!flag); |
|  |

|  |
| --- |
| G.pts.push\_back(P); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void KeyboardAction(unsigned char key, int x,int y) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| exit(0); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void display() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| glClear(GL\_COLOR\_BUFFER\_BIT); |
|  |

|  |
| --- |
| glEnable(GL\_POINT\_SMOOTH); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Polygon pyclip, py; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| //GenerateRandomSimplePolygon(pyclip, 4); |
|  |

|  |
| --- |
| //GenerateRandomSimplePolygon(py, 4); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Point p1, p2, p3, p4; |
|  |

|  |
| --- |
| p1.x = 553, p1.y = 495; |
|  |

|  |
| --- |
| p2.x = 351, p2.y = 175; |
|  |

|  |
| --- |
| p3.x = 486, p3.y = 71; |
|  |

|  |
| --- |
| p4.x = 61, p4.y = 86; |
|  |

|  |
| --- |
| pyclip.pts.push\_back(p1); |
|  |

|  |
| --- |
| pyclip.pts.push\_back(p2); |
|  |

|  |
| --- |
| pyclip.pts.push\_back(p3); |
|  |

|  |
| --- |
| pyclip.pts.push\_back(p4); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Point p5, p6, p7, p8; |
|  |

|  |
| --- |
| p5.x = 390, p5.y = 424; |
|  |

|  |
| --- |
| p6.x = 579, p6.y = 585; |
|  |

|  |
| --- |
| p7.x = 257, p7.y = 112; |
|  |

|  |
| --- |
| p8.x = 68, p8.y = 245; |
|  |

|  |
| --- |
| py.pts.push\_back(p5); |
|  |

|  |
| --- |
| py.pts.push\_back(p6); |
|  |

|  |
| --- |
| py.pts.push\_back(p7); |
|  |

|  |
| --- |
| py.pts.push\_back(p8); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int size = pyclip.pts.size(); |
|  |

|  |
| --- |
| for (int i = 0; i < size; ++ i) |
|  |

|  |
| --- |
| cout << pyclip.pts[i].x << " " << pyclip.pts[i].y << endl; |
|  |

|  |
| --- |
| cout << endl; |
|  |

|  |
| --- |
| size = py.pts.size(); |
|  |

|  |
| --- |
| for (int i = 0; i < size; ++ i) |
|  |

|  |
| --- |
| cout << py.pts[i].x << " " << py.pts[i].y << endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Color a = {1.0, 0.0, 0.0}; |
|  |

|  |
| --- |
| Color b = {0.0, 1.0, 0.0}; |
|  |

|  |
| --- |
| py.drawPolygonLine(a); |
|  |

|  |
| --- |
| pyclip.drawPolygonLine(b); |
|  |

|  |
| --- |
| weilerAtherton(pyclip, py); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| glFlush(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| { |
|  |

|  |
| --- |
| srand(time(NULL)); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| glutInit(&argc, argv); |
|  |

|  |
| --- |
| glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); |
|  |

|  |
| --- |
| glutInitWindowSize(Size, Size); |
|  |

|  |
| --- |
| glutInitWindowPosition(100, 100); |
|  |

|  |
| --- |
| glutCreateWindow("Weiler-Atherton Clipping Algorithm"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| glutKeyboardFunc(KeyboardAction); |
|  |

|  |
| --- |
| glutDisplayFunc(display); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| init(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| glutMainLoop(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return 0; |
|  |

}