**Session 3 : Implement Bresenham’s algorithm in OpenGL**

**Intended Learning Outcome:**

1. Students will be able to draw a line using Bresenham’s algorithm in OpenGL with two endpoints.
2. Students will be able to implement Bresenham’s algorithm.

**Expected Skills:**

1. Calculating any integer point between two endpoints of a line.
2. Clear idea about drawing a line using Bresenham’s algorithm.

**Tools Required:**

1. CodeBlocks
2. OpenGL and GLUT using CodeBlocks.

**Session Detail:**

**Bresenham’s Line drawing Algorithm**

****

1. Input the two line end-points, storing the left end-point in (*x0, y0*)
2. Plot the point (*x0, y0*)

3. Calculate the constants *x*, *y*, 2 *y*, and (2 *y -* 2 *x*) and get the first value for the decision parameter as:

*p*0 2 *y* *x*

1. At each *xk* along the line, starting at *k = 0*, perform the following test. If *pk* *< 0*, the next point to plot is *(xk+1, yk)* and:

*pk* 1 *pk* 2 *y*

1. Repeat step 4 (Δx – 1) times

After getting Bresenham’s algorithm’s theoretical concept, students will write a program to implement Bresenham’s algorithm.

**Code for Bresenham’s algorithm:**

#include <stdio.h>

#include <GL/gl.h>

#include <GL/glut.h>

float x1,y1,x2,y2,m,i,j,p;

int dx=0,dy=0;

void display(void)

{

/\* clear all pixels \*/

glClear (GL\_COLOR\_BUFFER\_BIT);

/\* draw white polygon (rectangle) with corners at

* (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0) \*/

glEnd();

glColor3f (0.0, 1.0, 0.0); glBegin(GL\_POINTS); p=(2\*dy)-dx; for(i=x1,j=y1;i<=x2,j<=y2; ){

if(p>=0){

i=i+1;

j=j+1;

if((i>x2)||(j>y2)){

break;

}

printf("%0.2f %0.2f\n",i,j); glVertex3f ((i/100), (j/100), 0.0); p=p+(2\*dy)-(2\*dx);

}

else if(p<0){ i=i+1; if((i>x2)||(j>y2)){

break;

}

printf("%0.2f %0.2f\n",i,j); glVertex3f ((i/100), (j/100), 0.0); p=p+(2\*dy);

}

}

glEnd();

/\* don't wait!

* start processing buffered OpenGL routines \*/

glFlush ();

}

void init (void)

{

/\* select clearing (background) color \*/ glClearColor (0.0, 0.0, 0.0, 0.0);

/\* initialize viewing values \*/ glMatrixMode(GL\_PROJECTION); glLoadIdentity();

glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);



}

/\*

* Declare initial window size, position, and display mode
* (single buffer and RGBA). Open window with "hello"
* In its title bar. Call initialization routines.
* Register callback function to display graphics.
* Enter main loop and process events.

\*/

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

{

printf("Enter first point: ");

scanf("%f %f",&x1,&y1);

printf("Enter second point: ");

scanf("%f %f",&x2,&y2);

dx=x2-x1;

dy=y2-y1;

glutInit(&argc, argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize (500, 500);



glutInitWindowPosition (100, 100);

glutCreateWindow ("hello");

init ();

glutDisplayFunc(display);

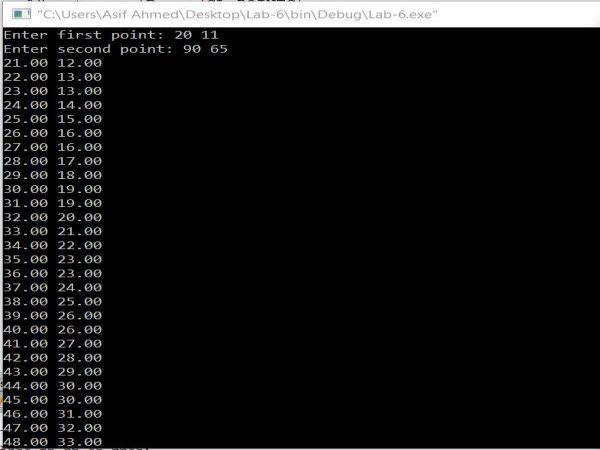
glutMainLoop();

return 0; /\* ISO C requires main to return int. \*/

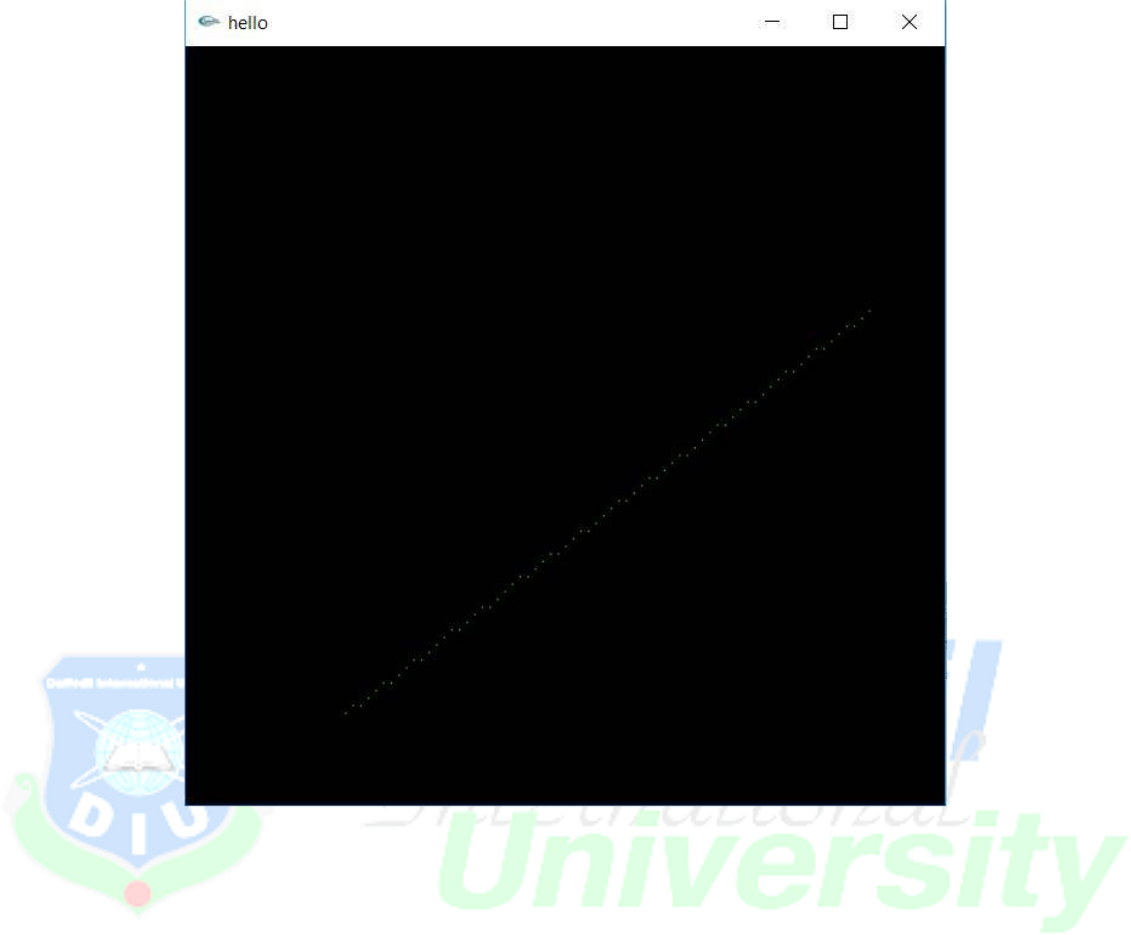
}

**Sample input:**

N.B.: (0<=x<100) and (0<=y<100)



**Sample output:**

****