

| **Title:** Drawing of line using computer graphics. |
| --- |

**Aim:** Generate the line using computer graphics program

Objectives:

1. Visit the Following link and perform the Vlab Experiment and provide the screenshots.

<https://cse18-iiith.vlabs.ac.in/exp/coordinate-systems/pretest.html>

1. Implement the Digital Differential Analyser (DDA) Line Drawing Algorithm.
2. Implement Bresenham Line Drawing Algorithm

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**Understand the basic concepts of computer graphics and OpenGL**

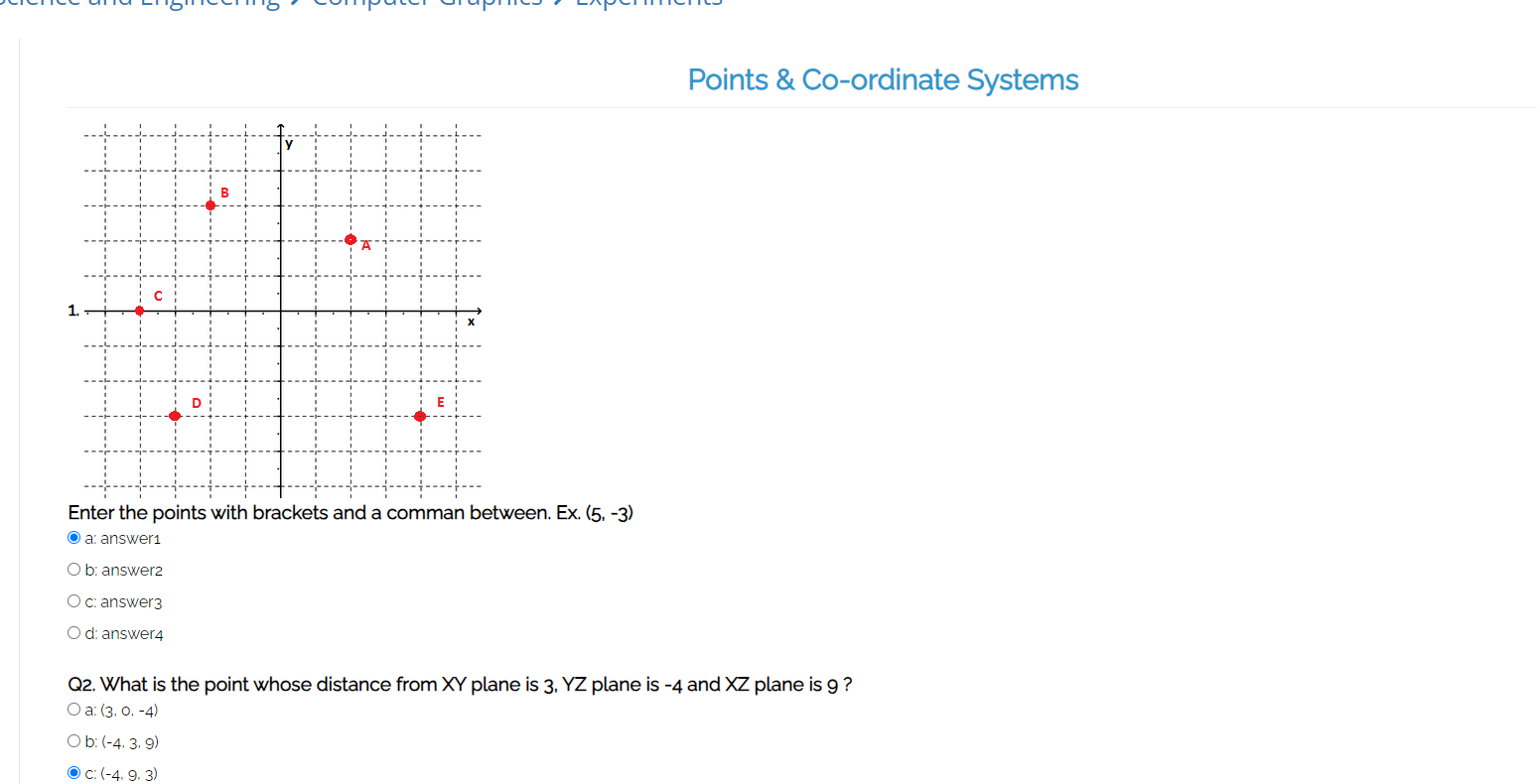
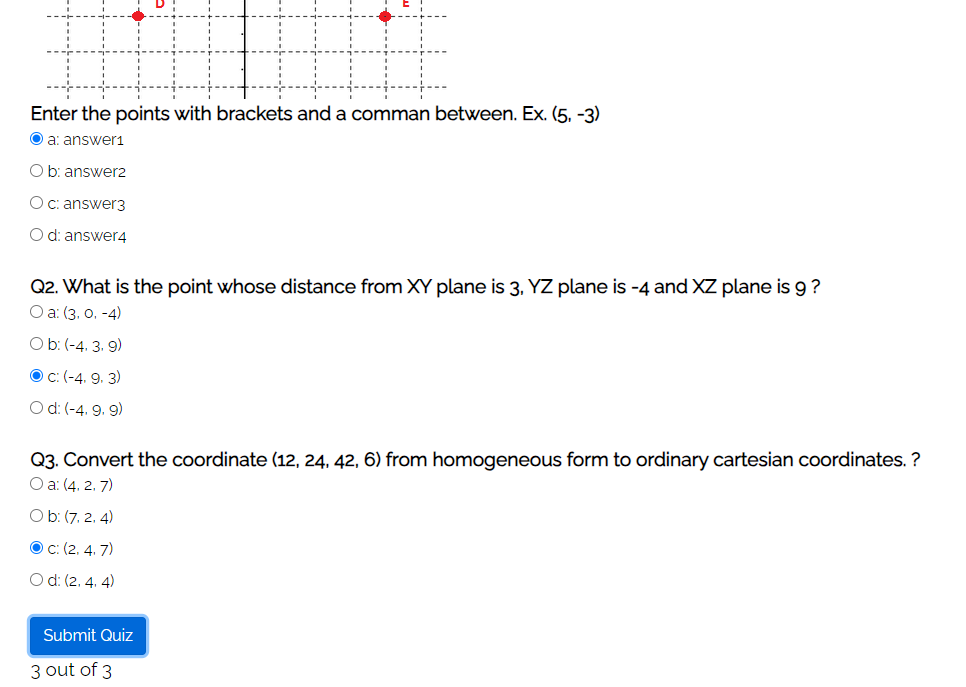
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

* google
* geeksforgeeks

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Screenshots from VLab**

**  
**

**Understanding from VLab:**

**Algorithm 1: DDA**

The DDA algorithm is a simple and efficient line drawing algorithm. It works by incrementing either the x or y coordinate and computing the corresponding y or x coordinate, respectively.  
Step 1: Start.

Step 2: We consider Starting point as (x1 , y1 ), and ending point (x2 , y2 ).

Step 3: Now, we have to calculate dx and dy. dx = x2 -x1 dy = y2 -y1 m = dy/dx

Step 4: Now, we calculate three cases.

* If m < 1 Then x change in Unit Interval y moves with deviation (xk+1, yk+1) = (xk+1, yk+m)
* If m > 1 Then x moves with deviation y change in Unit Interval (xk+1, yk+1) = (xk+1/m, yk+1)
* If m = 1 Then x moves in Unit Interval y moves in Unit Interval (xk+1, yk+1) = (xk+1, yk+1)

Step 5: We will repeat step 4 until we find the ending point of the line.

Step 6: Stop.

**Algorithm 2:Bresenham**

Bresenham's line drawing algorithm is another popular line-drawing algorithm that determines which points to plot to form a straight line between two given points.

Step 1: Start.

Step 2: Now, we consider Starting point as (x1 , y1 ) and ending point (x2 , y2 ). Step 3: Now, we have to calculate ▲x and ▲y. ▲x = x2 -x1 ▲y = y2 -y1

m = ▲y/▲x

Step 4: Now, we will calculate the decision parameter pk with following formula. pk = 2▲y-▲x

* Step 5: Theinitial coordinates of the line are (xk , yk ), and the next coordinatesare (xk+1, yk+1). Now, we are going to calculate two cases for decision parameter pk Case 1: If pk < 0 Then pk+1 =pk +2▲y xk+1 = xk +1 yk+1 = yk
* Case 2: If pk >= 0 Then pk+1 =pk +2▲y-2▲x xk+1 =xk +1 yk+1 =yk +1 Step 6: We will repeat step 5 until we found the ending point of the line and the total number of iterations =▲x-1.

Step 7: Stop

**Implementation details (Code can be in C/C++/Java/Python with and without using graphics library functions):**

**DDA:**

**#include <GLFW/glfw3.h>**

**#include <GL/glu.h>**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**void ddaLine(float x1, float y1, float x2, float y2) {**

**int dx = static\_cast<int>(x2 - x1);**

**int dy = static\_cast<int>(y2 - y1);**

**int steps = max(abs(dx), abs(dy));**

**float xIncrement = dx / static\_cast<float>(steps);**

**float yIncrement = dy / static\_cast<float>(steps);**

**float x = x1;**

**float y = y1;**

**glBegin(GL\_POINTS);**

**for (int i = 0; i <= steps; i++) {**

**glVertex2f(x, y);**

**x += xIncrement;**

**y += yIncrement;**

**}**

**glEnd();**

**}**

**void setupOpenGL() {**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**glPointSize(5.0f);**

**}**

**int main() {**

**if (!glfwInit()) {**

**cerr << "Failed to initialize GLFW" << endl;**

**return -1;**

**}**

**const int windowWidth = 400;**

**const int windowHeight = 400;**

**GLFWwindow\* window = glfwCreateWindow(windowWidth, windowHeight, "OpenGL Line Drawing", nullptr, nullptr);**

**if (!window) {**

**cerr << "Failed to create GLFW window" << endl;**

**glfwTerminate();**

**return -1;**

**}**

**glfwMakeContextCurrent(window);**

**glViewport(0, 0, windowWidth, windowHeight);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluOrtho2D(0.0, 30.0, 0.0, 30.0);**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**float x1, y1, x2, y2;**

**cout << "Enter the coordinates for the first point (x1 y1): ";**

**cin >> x1 >> y1;**

**cout << "Enter the coordinates for the second point (x2 y2): ";**

**cin >> x2 >> y2;**

**cout << "Drawing line from (" << x1 << ", " << y1 << ") to (" << x2 << ", " << y2 << ")" << endl;**

**while (!glfwWindowShouldClose(window)) {**

**setupOpenGL();**

**ddaLine(x1, y1, x2, y2);**

**glfwSwapBuffers(window);**

**glfwPollEvents();**

**}**

**glfwDestroyWindow(window);**

**glfwTerminate();**

**return 0;**

**}**

**Bresenham:**

**#include <GLFW/glfw3.h>**

**#include <GL/glu.h>**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**void bresenhamLine(int x1, int y1, int x2, int y2) {**

**int dx = abs(x2 - x1);**

**int dy = abs(y2 - y1);**

**int sx = (x1 < x2) ? 1 : -1;**

**int sy = (y1 < y2) ? 1 : -1;**

**int err = dx - dy;**

**while (true) {**

**glVertex2i(x1, y1);**

**if (x1 == x2 && y1 == y2) break;**

**int e2 = err \* 2;**

**if (e2 > -dy) {**

**err -= dy;**

**x1 += sx;**

**}**

**if (e2 < dx) {**

**err += dx;**

**y1 += sy;**

**}**

**}**

**}**

**void setupOpenGL() {**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**glPointSize(5.0f);**

**}**

**int main() {**

**if (!glfwInit()) {**

**cerr << "Failed to initialize GLFW" << endl;**

**return -1;**

**}**

**const int windowWidth = 640;**

**const int windowHeight = 480;**

**const int windowPosX = 100;**

**const int windowPosY = 150;**

**GLFWwindow\* window = glfwCreateWindow(windowWidth, windowHeight, "OpenGL Line Drawing", nullptr, nullptr);**

**if (!window) {**

**cerr << "Failed to create GLFW window" << endl;**

**glfwTerminate();**

**return -1;**

**}**

**glfwSetWindowPos(window, windowPosX, windowPosY);**

**glfwMakeContextCurrent(window);**

**glViewport(0, 0, windowWidth, windowHeight);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluOrtho2D(0.0, 30.0, 0.0, 30.0);**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**int x1, y1, x2, y2;**

**cout << "Enter the coordinates for the first point (x1 y1): ";**

**cin >> x1 >> y1;**

**cout << "Enter the coordinates for the second point (x2 y2): ";**

**cin >> x2 >> y2;**

**cout << "Drawing line from (" << x1 << ", " << y1 << ") to (" << x2 << ", " << y2 << ")" << endl;**

**while (!glfwWindowShouldClose(window)) {**

**setupOpenGL();**

**glBegin(GL\_POINTS);**

**bresenhamLine(x1, y1, x2, y2);**

**glEnd();**

**glfwSwapBuffers(window);**

**glfwPollEvents();**

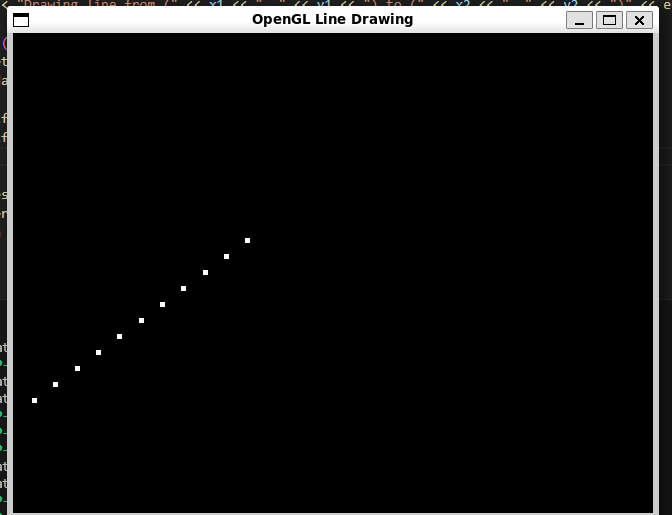
**}**

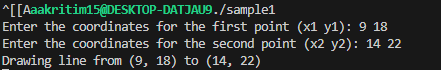
**glfwDestroyWindow(window);**

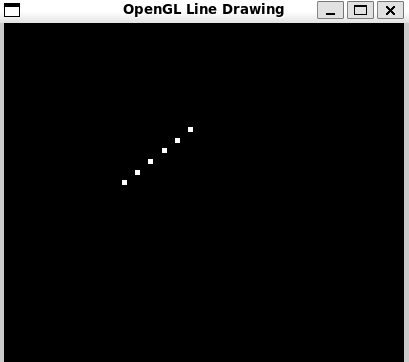
**glfwTerminate();**

**return 0;**

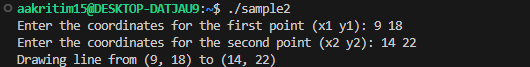
**}**

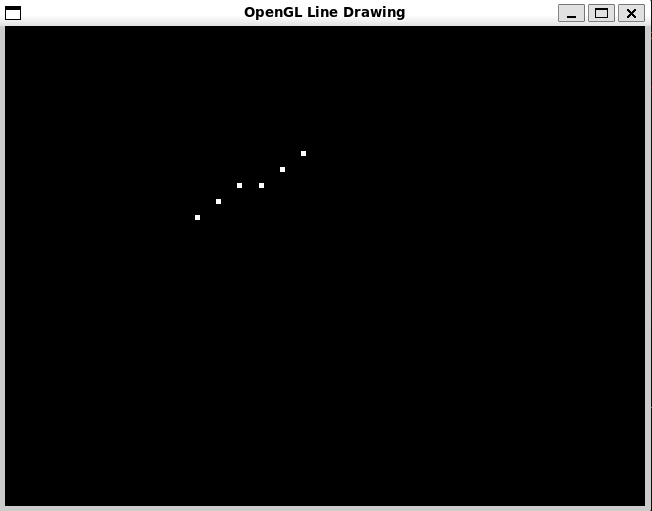
**Output(s) (final edited screen shot):  
DDA:  
  
  
**

****

****

**Bresenham:**

****

****

**Conclusion and discussion (Comparative - compulsory):**

Hence, two of the line drawing algorithms, DDA and Bresenham line drawing algorithms have been studied and implemented.

* Bresenham's Algorithm: Efficient and fast with integer operations, better for systems where floating-point arithmetic is slow or impractical. Preferred for applications where performance is crucial and integer arithmetic is available.
* DDA Algorithm: Conceptually simpler and useful for systems with good floating-point support, but can be slower and less precise due to floating-point operations.

**Date: 14/8/24**

**Signature of faculty in-charge**

**Explain Mid-point line drawing algorithm and implement it**

Principle:

● The line is drawn from left to right, pixel by pixel.

● For every confirmed pixel, there are two choices for the next pixel, the pixel on

its right ((Xp+1, Yp) in the figure) or the pixel on its upper right ((Xp+1, Yp+1)

in the figure).

● Using the mid-point algorithm, the choice between the two is made on the basis

of where the intercept of the input line lies when it cuts the line between the

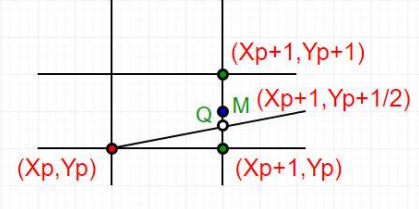
above points.

● If the line cuts above the midpoint ((Xp+1, Yp+0.5) in the figure), then the

upper right point is chosen and if the line cuts below the midpoint then the point

to the right is chosen.

Consider any given/calculated previous pixel P(Xp,Yp), there are two candidates for the next pixel closest to the line, E(Xp+1, Yp) and NE(Xp+1, Yp+1) (E stands for East and NE stands for North-East).



In Mid-Point algorithm:

● Find middle of two possible next points. Middle of E(Xp+1, Yp) and NE(Xp+1,

Yp+1) is M(Xp+1, Yp+1/2).

● If M is above the line, then choose E as next point.

● If M is below the line, then choose NE as next point.

These are few assumptions we make to keep the algorithm simple:

● We draw line from left to right.

● x1 < x2 and y1< y2

● Slope of the line is between 0 and 1. We draw a line from lower left to upper

right.

Cases other than above assumptions can be handled using reflection. Let

us consider a line y = mx + B.

//Midpoint Algorithm  
#include <GLFW/glfw3.h>

#include <GL/glu.h>

#include <iostream>

#include <cmath>

using namespace std;

void midpointLine(float x1, float y1, float x2, float y2) {

float dx = x2 - x1;

float dy = y2 - y1;

float d = 2 \* dy - dx;

float y = y1;

glBegin(GL\_POINTS);

for (float x = x1; x <= x2; x++) {

glVertex2f(x, y);

if (d > 0) {

y++;

d += 2 \* (dy - dx);

} else {

d += 2 \* dy;

}

}

glEnd();

}

void setupOpenGL() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glPointSize(5.0f);

}

int main() {

if (!glfwInit()) {

cerr << "Failed to initialize GLFW" << endl;

return -1;

}

const int windowWidth = 400;

const int windowHeight = 400;

GLFWwindow\* window = glfwCreateWindow(windowWidth, windowHeight, "Midpoint Line Drawing", nullptr, nullptr);

if (!window) {

cerr << "Failed to create GLFW window" << endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glViewport(0, 0, windowWidth, windowHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 30.0, 0.0, 30.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

float x1, y1, x2, y2;

cout << "Enter the coordinates for the first point (x1 y1): ";

cin >> x1 >> y1;

cout << "Enter the coordinates for the second point (x2 y2): ";

cin >> x2 >> y2;

cout << "Drawing line from (" << x1 << ", " << y1 << ") to (" << x2 << ", " << y2 << ")" << endl;

while (!glfwWindowShouldClose(window)) {

setupOpenGL();

midpointLine(x1, y1, x2, y2);

glfwSwapBuffers(window);

glfwPollEvents();

}

glfwDestroyWindow(window);

glfwTerminate();

return 0;

}

