**DEERWALK INSTITUTE OF TECHNOLOGY**

**Tribhuvan University**

**Faculties of Computer Science**

# Bachelors of Science in Computer Science and Information Technology (BSc. CSIT)

## Course: Computer Graphics (CSC209)

**Year/Semester: II/III**

## A Lab report on:

## Implementation of Bresenham's Line Drawing Algorithm


Submitted by:
Name: Arun Mainali
Roll No.: 1307

Submitted to:
Binod Sitaula
Department of Computer Science

# ❖ LAB 2

**OBJECTIVE:**

Implementation of Bresenham's Line Drawing Algorithm for:

i. Left to Right
ii. Right to Left

**THEORY:**

Bresenham's Line Drawing Algorithm:

**Bresenham's Line Drawing Algorithm**

Bresenham's Line Drawing Algorithm is a method for converting a geometric line into pixels on a screen. This algorithm, devised by Bresenham, is highly efficient as it uses only integer addition, subtraction, and multiplication operations, which are computationally fast.

The core idea is to select the next pixel that has the smallest distance from the actual line. Here's how the algorithm works:

Assume the line starts at pixel $P_1'$ ($x_1'$, $y_1'$) and moves horizontally toward $P_2'(x_2', y_2')$. At each step, a pixel is chosen in one of two positions:

1. The pixel directly to the right (lower-bound for the line).
2. The pixel to the right and upward (upper-bound for the line).

The goal is to select the pixel that best approximates the true line by minimizing the vertical distance from the line.

The method works as follows:

**Pixel Selection:**

Assume a pixel $P_1'(x_1', y_1')$,then select subsequent pixels as we work our way to the night, one pixel position at a time in the horizontal direction toward $P_2'(x_2', y_2')$.

Once a pixel in choose at any step

The next pixel is

1. Either the one to its right (lower-bound for the line)

2. One top its right and up (upper-bound for the line)

The line is best approximated by those pixels that fall the least distance from the path between $P_1'$, $P_2'$.

To chooses the next one between the bottom pixel S and top pixel T.

If S is chosen

We have $x_{i+1}=x_i+1$ and $y_{i+1}=y_i$

If T is chosen

We have $x_{i+1}=x_i+1$ and $y_{i+1}=y_i+1$

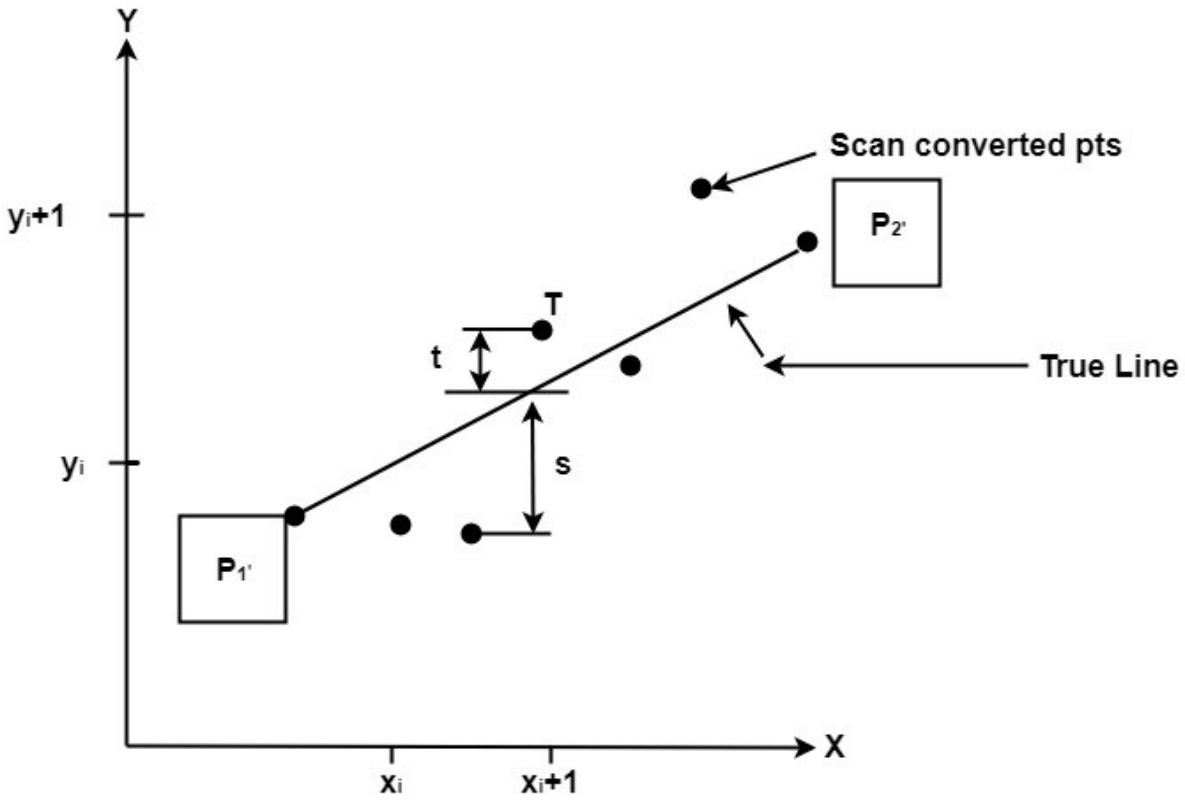



**Fig: Scan Converting a line.**

The actual y coordinates of the line at $x = x_{i+1}$ is

$y=mx_{i+1}+b$

$$y = m(x_i + 1) + b$$

**Calculating Distance:**

The distance from S to the actual line in y direction

$s = y - y_i$

The distance from T to the actual line in y direction

$t = (y_i+1) - y$

Now consider the difference between these 2 distance values

$s - t$

When $(s-t) < 0 \Longrightarrow s < t$

The closest pixel is S

When $(s-t) \geq 0 \Longrightarrow s < t$

The closest pixel is T

This difference is

$s-t = (y-yi)-[(yi+1)-y]$

$= 2y - 2yi - 1$

$$s - t = 2m(x_i + 1) + 2b - 2y_i - 1$$ [Putting the value of (1)]

**Decision Variable:**

Substituting m by $\frac{\triangle y}{\triangle x}$ and introducing decision variable

$d_i = \triangle x\,(s-t)$

$d_i = \triangle x\,(2\frac{\triangle y}{\triangle x}(x_i+1)+2b-2y_i-1)$

$=2\triangle xy_i-2\triangle y-1\triangle x.2b-2y_i\triangle x-\triangle x$

$d_i = 2\triangle y.x_i-2\triangle x.y_i+c$

Where $c = 2\triangle y+\triangle x\,(2b-1)$

We can write the decision variable $d_{i+1}$ for the next slip on

$d_{i+1} = 2\triangle y.x_{i+1}-2\triangle x.y_{i+1}+c$

$d_{i+1}-d_i = 2\triangle y.(x_{i+1}-x_i)- 2\triangle x(y_{i+1}-y_i)$

Since x_(i+1)=$x_i$+1,we have

$d_{i+1}+d_i=2\triangle y.(x_i+1-x_i)- 2\triangle x(y_{i+1}-y_i)$

**Special Cases**

If chosen pixel is at the top pixel T (i.e., $d_i \geq 0) \Longrightarrow y_{i+1}=y_i+1$

$d_{i+1}=d_i+2\triangle y-2\triangle x$

If chosen pixel is at the bottom pixel T (i.e., $d_i<0) \Longrightarrow y_{i+1}=y_i$

$d_{i+1}=d_i+2\triangle y$

Finally, we calculate $d_1$

$d_1=\triangle x[2m(x_1+1)+2b-2y_1-1]$

$d_1=\triangle x[2(mx_1+b-y_1)+2m-1]$

Since $mx_1+b-y_i=0$ and $m = \frac{\Delta y}{\Delta x}$, we have

$d_1=2\triangle y-\triangle x$

**Advantages:**

1. The algorithm uses only integer arithmetic, making it simple and efficient.
2. It avoids generating duplicate points.
3. It is hardware-friendly due to the lack of multiplication and division.
4. It is faster than the Digital Differential Analyzer (DDA) as it avoids floating-point calculations.

**Disadvantages:**

1. It is primarily designed for basic line drawing and may not produce smooth lines.
2. For smoother rendering, other algorithms may be better suited.

**Algorithm Steps:**

1. **Start** the algorithm.
2. Declare variables $x_1$, $x_2$, $y_1$, $y_2$, d, $i_1$, $i_2$, Δx, Δy
3. Input the values of $x_1$, $y_1$ (starting point) and $x_2$, $y_2$ (ending point).
4. Compute:
    - $\Delta x=x_2-x_1$
    - $\Delta y=y_2-y_1$
    - $i_1=2 \cdot \Delta y$

- $i_2=2\cdot(\Delta y-\Delta x)$
- $d=i_1-\Delta x$

5. Determine the initial point (x,y)(x, y)(x,y):
   - If $\Delta x<0$: $x=x_2$, $y=y_2$, $x_{end}=x_1$
   - Otherwise: $x=x_1$, $y=y_1$, $x_{end}=x_2$
6. Plot the initial point (x,y)
7. Check if the line is complete:
   - If $x \geq x_{end}$ , stop.
8. Compute the next pixel:
   - If d<: $d=d+i_1$
   - Otherwise: $d=d+i_2$, $y=y+1$
9. Increment x=x+1
10. Draw the next pixel at (x,y)
11. Repeat from Step 7.

## PROGRAM CODE 1:

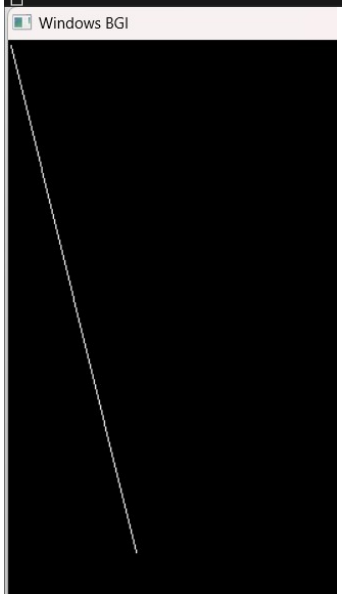
```
#include <graphics.h>
#include <iostream>
#include <cmath>
using namespace std;
// Function for Bresenham's Line Algorithm
void bresenhamLine(int x0, int y0, int x1, int y1) {
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = (x1 > x0) ? 1 : -1; // Sign for x increment
    int sy = (y1 > y0) ? 1 : -1; // Sign for y increment
    bool interchange = false;
```

```
    // Check if |m| > 1; if true, interchange roles of x and y
    if (dy > dx) {
        swap(dx, dy);
        interchange = true;
    }
    int p = 2 * dy - dx; // Initial decision parameter
    int x = x0, y = y0; // Starting point
    // Loop through the line
    for (int i = 0; i <= dx; i++) {
        putpixel(x, y, WHITE); // Plot the current point
        if (p < 0) {
            if (interchange){
                y += sy;}
            else{
                x += sx;
            p += 2 * dy;
        } else {
            x += sx;
            y += sy;
            p += 2 * (dy - dx);
        }
    }
}
int main() {
```

```
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    int x0, y0, x1, y1;
    cout << "Enter the starting point (x0, y0): ";
    cin >> x0 >> y0;
    cout << "Enter the ending point (x1, y1): ";
    cin >> x1 >> y1;
    // Call Bresenham's line function
    bresenhamLine(x0, y0, x1, y1);
    getch();
    closegraph();
    return 0;
}
```

}**SAMPLE OUTPUT**:

```
PS C:\Users\bijan\Downloads\graphics.h-project-template-main>
me\src\bresanham.exe"
Enter the starting point (x0, y0): 100 400
Enter the ending point (x1, y1): 2 4
```



## **CONCLUSION**:

From this lab, we were able to draw a line using Bresenham's Line Drawing Algorithm and display it using C graphic functions.