Vidyavardhini's College of Engineering & Technology

Department of Computer Science Engineering(Data Science)

Name:	ATHARVA SUDHAKAR RODGE
Roll No:	
Class/Sem:	SE/III
Experiment No.:	1
Title:	Digital Differential Analyzer Line Drawing Algorithm
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Experiment No. 1

Aim: Write a program to implement Digital Differential Analyzer Line Drawing Algorithm in C.

Objective: To implement DDA line drawing algorithm for drawing a line segment between two points (x1, y1) and B (x2, y2)

Theory: DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

DDA Working Mechanism

Case 1: Slope < 1 (m<1) (theta<45)x coordinates increase fast
dx is set to unit interval dx=1; die is computed; m = die/dx
therefore; dy = m Calculation for next pixel for line processed from left to right
$x_{k+1} = x_k + dx =$
$x_k + 1$ $y_{k+1} = y_k + dy$
$= y_k + m$
Calculation for next pixel for line processed from
right to left $x_{k+1} = x_k + dx = x_k - 1$
$y_{k+1} = y_k + dy = y_k - m$

Case 2: Slope > 1 (m>1) theta>45 y coordinates increase fast
dy is set to unit interval dy=1; dx is computed; m = dy/dx therefore;
dx = 1/m Calculation for next pixel for line processed from left to right
$x_{k+1} = x_k + dx = x_k$
$+ 1/m$ $y_{k+1} = y_k + dy =$
$y_k + 1$
Calculation for next pixel for line processed from
right to left $x_{k+1} = x_k + dx = x_k - 1/m$
$y_{k+1} = y_k + dy = y_k - 1$

Case 3: Slope =1 (m=1) theta=45
dx and dy is set to unit interval dx=1 and dy=1

Calculation for next pixel for line processed from left to right $x_{k+1} = x_k + dx = x_k + 1$

$y_{k+1} = y_k + dy = y_k + 1$

Calculation for next pixel for line processed from right to left $x_{k+1} = x_k + dx = x_k - 1$

$y_{k+1} = y_k + dy = y_k - 1$

Algorithm

DDA Algorithm:

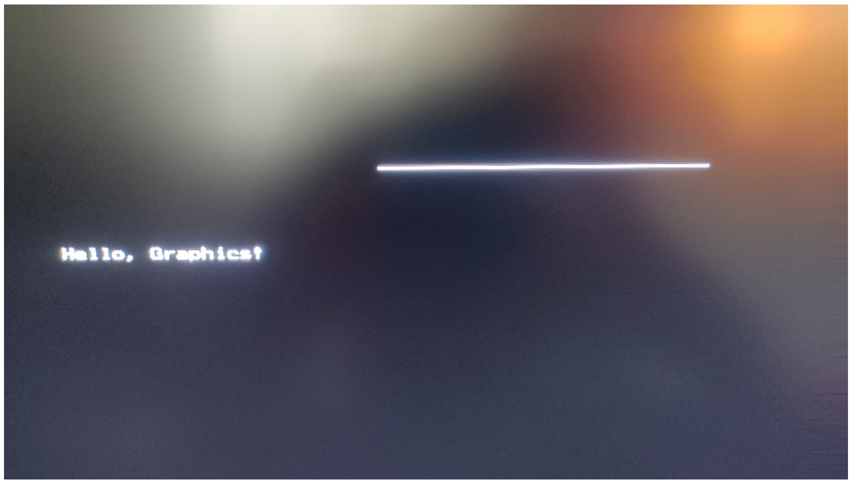
```
Step1: Start Algorithm
Step2: Declare $x_1,y_1,x_2,y_2,dx,dy,x,y$ as integer variables.
Step3: Enter value of $x_1,y_1,x_2,y_2$.
Step4: Calculate $dx = x_2-x_1$
Step5: Calculate $dy = y_2-y_1$
Step6: If ABS (dx) > ABS (dy)
        Then step = abs (dx)
        Else
Step7:
        $x_{inc}=dx/step$
        $y_{inc}=dy/step$
        assign $x = x_1$
        assign $y = y_1$
Step8: Set pixel (x, y)
Step9: $x = x + x_{inc}$
        $y = y + y_{inc}$
       Set pixels (Round (x), Round (y))
Step10: Repeat step 9 until $x = x_2$
Step11: End Algorithm
```

Code :

```
#include <stdio.h>
#include <graphics.h>
void drawLineDDA(int x1, int y1, int x2, int y2) {
   float dx = x2 - x1;
   float dy = y2 - y1;
   float steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);
   float xIncrement = dx / steps;
   float yIncrement = dy / steps;
   float x = x1;
   float y = y1;
   putpixel(x, y, WHITE);
   for (int i = 1; i <= steps; i++) {
      x += xIncrement;
      y += yIncrement;

      putpixel(round(x), round(y), WHITE);
   }
}
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");
   int x1, y1, x2, y2;
   printf("Enter the coordinates of the first point (x1 y1): ");
   scanf("%d %d", &x1, &y1);
   printf("Enter the coordinates of the second point (x2 y2): ");
   scanf("%d %d", &x2, &y2);
   drawLineDDA(x1, y1, x2, y2);
   delay(5000);
   closegraph();
   return 0;
}
```

Output :



```
Hello, Graphics!
```

Conclusion:

The Digital Differential Analyzer (DDA) algorithm is a straightforward and efficient method for drawing lines on a computer screen. It is based on linear interpolation using floating-point arithmetic to determine the pixel positions along the line. The algorithm is easy to understand and implement, making it suitable for simple graphics applications.

However, the DDA algorithm has some limitations. It relies on floating-point arithmetic, which may not be ideal for systems with limited computational resources. Moreover, rounding errors during calculations can accumulate, leading to slight deviations from the ideal line. Additionally, the DDA algorithm is sensitive to the choice of the number of steps, and an inappropriate number of steps may result in a loss of precision or graphical artifacts.

Vidyavardhini's College of Engineering & Technology

Department of Computer Science Engineering(Data Science)

Name:	ATHARVA SUDHAKAR RODGE
Roll No:	
Class/Sem:	SE/III
Experiment No.:	2
Title:	Bradenham's Line Drawing Algorithm
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Experiment No. 2

Aim: Write a program to implement Bradenham Line Drawing Algorithm in C.

Objective: To implement Bradenham line drawing algorithm for drawing a line segment between two points A (x1, y1) and B (x2, y2)

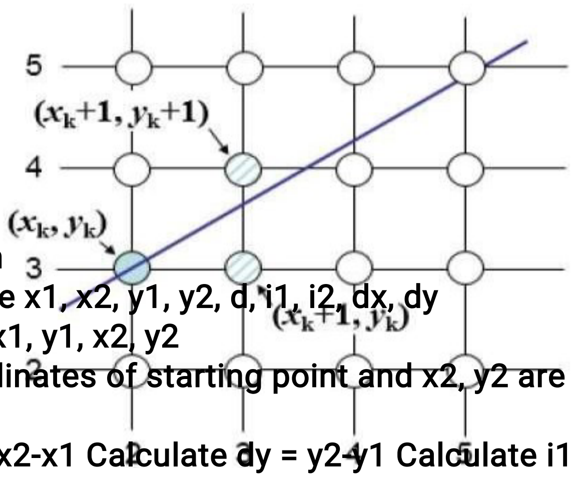
Theory :

Bresenham Algorithm

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer

addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.
In this method, next pixel selected is that one who has the least distance from true line

Basic Concept:
Move across the x axis in unit intervals and at each step choose between two different y coordinates.
For example, from position (2, 3) we must choose between (3, 3) and (3, 4).
We would like the point that is closer to the original line.
So, we must take decision to choose next point. So next pixels are selected based on the value of decision parameter p. The equations are given in below algorithm



Algorithm:

Step1: Start Algorithm
Step2: Declare variable x1, x2, y1, y2, d, i1, i2, dx, dy
Step3: Enter value of x1, y1, x2, y2
Where x1, y1are coordinates of starting point and x2, y2 are coordinates of Ending point
Step4: Calculate dx = x2-x1 Calculate dy = y2-y1 Calculate i1=2*dy Calculate

i2=2*(dy-dx) Calculate d=i1-dx
Step5: Consider (x, y) as starting point and xend as maximum possible value of x.
If dx < 0
Then x = x2 y = y2 xend=x1
If dx > 0 Then x = x1
y = y1 xend=x2
Step6: Generate point at (x,y)coordinates.
Step7: Check if whole line is generated.
If x > = xend Stop.
Step8: Calculate co-ordinates of the next pixel If d < 0
Then d = d + i1 If d ≥ 0
Then d = d + i2 Increment y = y + 1
Step9: Increment x = x + 1
Step10: Draw a point of latest (x, y) coordinates Step11: Go to step 7
Step12: End

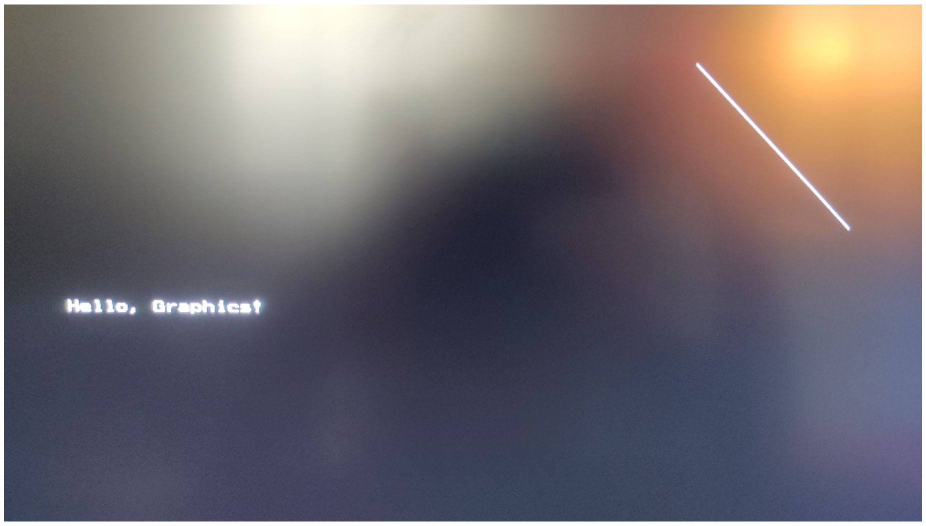
Code :

```
#include <stdio.h>
#include <graphics.h>
void drawLineBresenham(int x1, int y1, int x2, int y2) {
   int gd = DETECT, gm;
```

```
initgraph(&gd, &gm, NULL);
int dx = abs(x2 - x1);
int dy = abs(y2 - y1);
int slope_gt_one = 0;
if (dy > dx) {
   slope_gt_one = 1;
   int temp = x1;
   x1 = y1;
   y1 = temp;
   temp = x2;
   x2 = y2;
   y2 = temp;
   dx = abs(x2 - x1);
   dy = abs(y2 - y1);
}
int p = 2 * dy - dx;
int twoDy = 2 * dy;
int twoDyMinusDx = 2 * (dy - dx);
int x, y, xEnd;
if (x1 > x2) {
   x = x2;
   y = y2;
   xEnd = x1;
} else {
   x = x1;
   y = y1;
   xEnd = x2;
}
putpixel(x, y, WHITE);
while (x < xEnd) {
   x++;
   if (p < 0)
      p += twoDy;
   else {
      y++;
      p += twoDyMinusDx;
   }
   if (slope_gt_one)
      putpixel(y, x, WHITE);
   else
```

```
        putpixel(x, y, WHITE);
    }
    closegraph();
}
int main() {
    int x1, y1, x2, y2;
    printf("Enter the starting point (x1, y1): ");
    scanf("%d %d", &x1, &y1);
    printf("Enter the ending point (x2, y2): ");
    scanf("%d %d", &x2, &y2);
    drawLineBresenham(x1, y1, x2, y2);
    return 0;
}
```

Output



Conclusion:

The Bresenham Line Drawing Algorithm is a powerful and efficient method for drawing lines on a computer screen using integer arithmetic. It avoids the need for floating-point calculations, making it faster and more suitable for systems with limited computational resources.

One key advantage of the Bresenham algorithm is its ability to make decisions based on integer operations and avoid the overhead associated with floating-point arithmetic. This leads to a faster execution and is particularly beneficial in resource-constrained environments.

The algorithm is also versatile and can be extended to draw lines at any angle efficiently. However, it is specific to drawing lines and doesn't handle other shapes. In contemporary graphics programming, hardware acceleration and more advanced algorithms are often used for efficiency, but the Bresenham algorithm remains a fundamental concept and is widely used in various applications.

Vidyavardhini's College of Engineering & Technology

Department of Computer Science Engineering(Data Science)

Name:	ATHARVA SUDHAKAR RODGE
Roll No:	
Class/Sem: **Experiment No.:** **Title:** **Date of Performance:** **Date of Submission:**	SE/III 3 Midpoint Circle Drawing Algorithm
Marks:	
Sign of Faculty:	

Experiment No. 3

Aim: Write a program to implement Midpoint Circle Drawing Algorithm in C.

Objective :To implement midpoint circle drawing algorithm for drawing a circle with radius (R) and center (Xc, Yc)

Midpoint Circle Algorithm

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.

· We know that there are 360 degrees in a circle. First, we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.

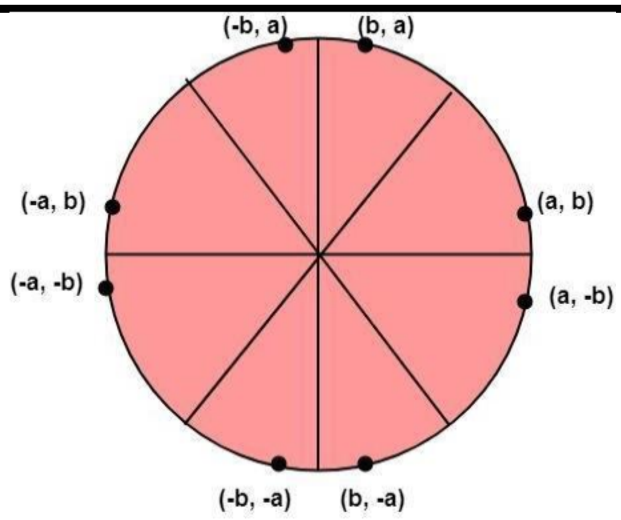
· Next we see that it is also symmetrical about the y axis, so now we only need to calculate the first 90 degrees.

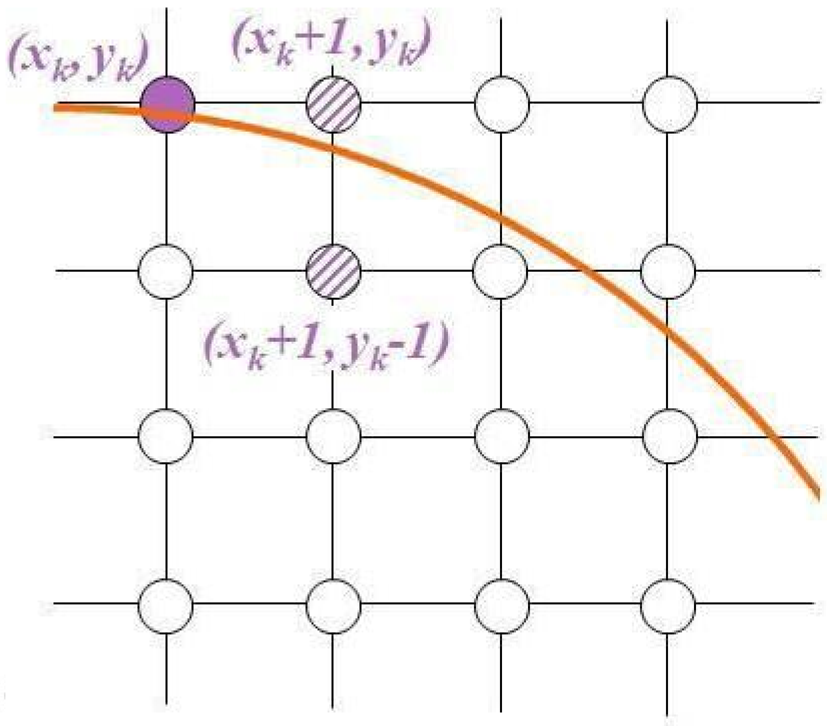
· Finally, we see that the circle is also symmetrical about the 45 degrees diagonal axis, so we only need to calculate the first 45 degrees.

· We only need to calculate the values on the border of the circle in the first octant. The other values may be determined by symmetry

Midpoint circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate. The pixel positions for determining symmetry are given in the below algorithm.

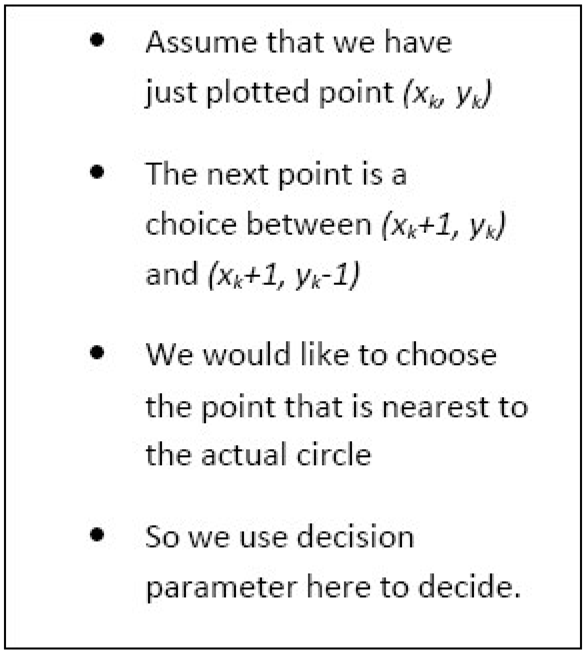
It is based on the following function for testing the spatial relationship between the arbitrary point (x, y) and a circle of radius r centered at the origin:





f

[> 0 for (x, y)outside the circle]



- Assume that we have just plotted point *(x_k, y_k)*
- The next point is a choice between *(x_k+1, y_k)* and *(x_k+1, y_k-1)*
- We would like to choose the point that is nearest to the actual circle
- So we use decision parameter here to decide.

Algorithm:

Plot the initial (x_k, y_k) i.e., $x_k = 0$ and $y_k = r$
$x_{k+1} = x_k + 1$
$y_{k+1} = y_k$
If $P_k >= 0$, then choose $y_{k+1} = y_k$

Repeat 3 and 4 until x becomes greater than or equal to y
To plot the entire circle, use the 8-way symmetry

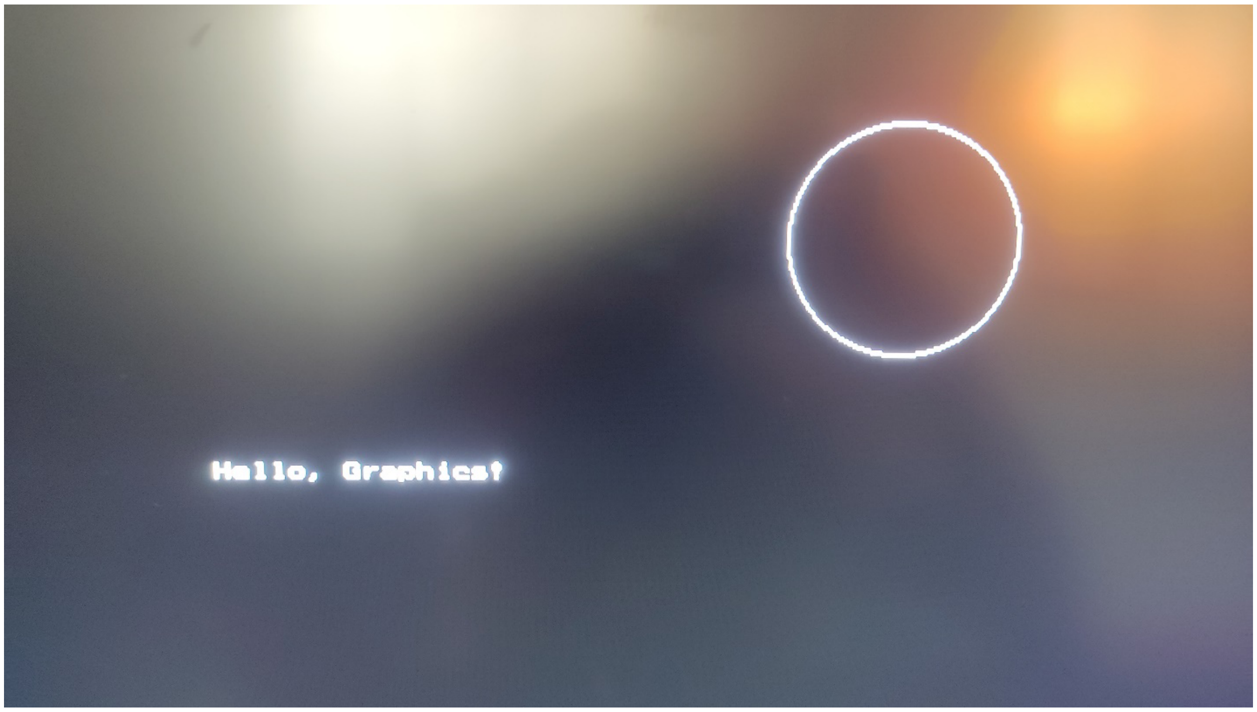
$$P_{k+1} = P_k + 2x_k + 3$$

Code:

```c
#include <stdio.h>
#include <graphics.h>
void drawCircleMidpoint(int xc, int yc, int radius) {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, NULL);
   int x = radius;
   int y = 0;
   int p = 1 - radius;
   putpixel(x + xc, y + yc, WHITE);
   if (radius > 0) {
      putpixel(x + xc, -y + yc, WHITE);
      putpixel(y + xc, x + yc, WHITE);
      putpixel(-y + xc, x + yc, WHITE);
   }
   while (x > y) {
      y++;
      if (p <= 0)
         p = p + 2 * y + 1;
      else {
         x--;
         p = p + 2 * y - 2 * x + 1;
      }
```

```
        if (x < y)
            break;
        putpixel(x + xc, y + yc, WHITE);
        putpixel(-x + xc, y + yc, WHITE);
        putpixel(x + xc, -y + yc, WHITE);
        putpixel(-x + xc, -y + yc, WHITE);
        if (x != y) {
            putpixel(y + xc, x + yc, WHITE);
            putpixel(-y + xc, x + yc, WHITE);
            putpixel(y + xc, -x + yc, WHITE);
            putpixel(-y + xc, -x + yc, WHITE);
        }
    }
    delay(5000);
    closegraph();
}
int main() {
    int xc, yc, radius;
    printf("Enter the center of the circle (xc, yc): ");
    scanf("%d %d", &xc, &yc);
    printf("Enter the radius of the circle: ");
    scanf("%d", &radius);
    drawCircleMidpoint(xc, yc, radius);
    return 0;
}
```

Output:



```
Hello, Graphics!
```

Conclusion:

The Midpoint Circle Drawing Algorithm is a simple and efficient algorithm for drawing circles. It is based on the concept of symmetry and reduces the computational load by taking advantage of the symmetry properties of a circle. The algorithm uses integer arithmetic and avoids the need for expensive floating-point operations.

One notable advantage of the Midpoint Circle Drawing Algorithm is its efficiency in terms of both time and space. It generates circle points using only integer addition and subtraction operations, making it suitable for systems with limited computational resources.

However, it is important to note that this algorithm is specifically designed for integer coordinates and may not produce accurate results for circles with very large radii. In such cases, other algorithms, such as the Bradenham Circle Algorithm, might be preferred. Overall, the Midpoint Circle Drawing Algorithm is a fundamental and widely used technique in computer graphics for rendering circular shapes efficiently.