Tribhuvan University
Institute Of Science and Technology

Prithvi Narayan Campus

BSc.CSIT Program

PRACTICAL REPORT

(Computer Graphics)

Submitted To

Mr. Dev Timilsina
Department of Computer Science & Information Technology
Prithvi Narayan Campus, Pokhara

Submitted By

Mahesh Kumar Udas
Roll No. 21
2080 Batch

2nd Year / 3rd Semester

INDEX

Name: Mahesh Kumar Udas	Roll No.: 21
Faculty: BSc.CSIT	Semester: Third
Subject: Computer Graphics	Year : Second

LAB 1

OBJECTIVE: To implement and understand the DDA (Digital Differential Analyzer) line drawing algorithm using C++ and the graphics library in Turbo C++.

THEORY:

The DDA Line Drawing Algorithm is an incremental scan conversion method used in computer graphics to draw lines between two points. It calculates intermediate positions by incrementing one coordinate and computing the other using the line equation.

The formula used is:

- dx = x2 - x1
- dy = y2 - y1
- step = max(|dx|, |dy|)
- x increment = dx / step
- y increment = dy / step

It is simple, fast, and works well for drawing straight lines.

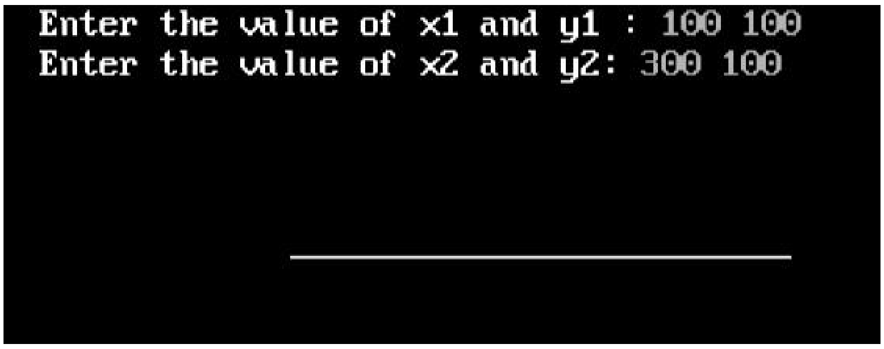
ALGORITHM:

1. Input the coordinates (x1, y1) and (x2, y2)
2. Calculate dx = x2 - x1 and dy = y2 - y1
3. Determine steps = max(|dx|, |dy|)
4. Calculate x increment = dx / steps
5. Calculate y increment = dy / steps
6. Set starting point (x, y) = (x1, y1)
7. Repeat steps to plot pixels until the end point is reached

PROGRAMS

```
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <dos.h>

void main(){
        float x,y,x1,y1,x2,y2,dx,dy,step;
        int i,gd=DETECT,gm;

        initgraph(&gd,&gm,"c:\\turboc3\\bgi");

        cout<<"Enter the value of x1 and y1 : ";
        cin>>x1>>y1;
        cout<<"Enter the value of x2 and y2: ";
        cin>>x2>>y2;

        dx=abs(x2-x1); dy=abs(y2-y1);

        if(dx>=dy) step=dx;
        else step=dy;

        dx=dx/step;
        dy=dy/step;

        x=x1; y=y1;

        i=1;
        while(i<=step){
                putpixel(x,y,10);
                x=x+dx;
                y=y+dy;
                i=i+1;
                delay(10);
        }
        getch();
        closegraph();
}
```

Output:



```
Enter the value of x1 and y1 : 100 100
Enter the value of x2 and y2: 300 100
```

RESULTS:

The DDA line drawing algorithm was successfully implemented and executed. The program accurately draws a line between the given points using pixel plotting.

CONCLUSION:

The DDA algorithm is a straightforward and efficient method for line drawing in computer graphics. It uses floating-point operations to increment coordinates and is ideal for lines with arbitrary slopes.

LAB 2

OBJECTIVE: To implement and understand the Bresenham's Line Drawing Algorithm in C++ using Turbo C++ and graphics library functions.

THEORY:

The Bresenham's Line Drawing Algorithm is an efficient method used in computer graphics to draw a straight line between two points. It uses only integer calculations, which makes it faster than the DDA algorithm, especially on systems where floating-point operations are expensive.

Key Concepts:

- Works using decision parameters to determine the next pixel position.
- Uses only integer addition, subtraction, and multiplication.
- Ideal for raster devices (like monitors) where pixel locations are integers.

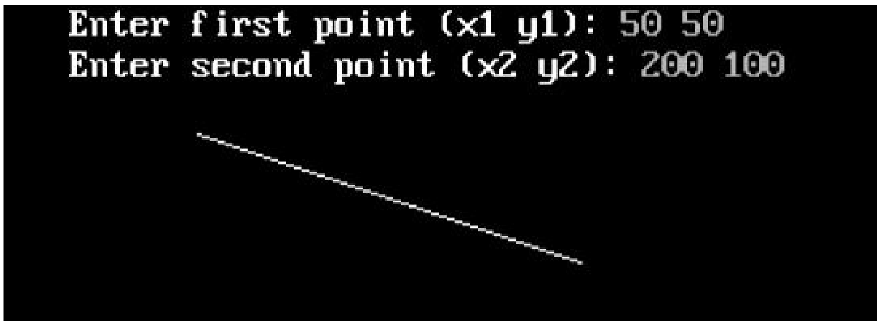
ALGORITHM:

```
1. Input the coordinates of the two endpoints (x1, y1) and (x2, y2).
2. Calculate dx = x2 - x1 and dy = y2 - y1.
3. Initialize the decision parameter m = 2*dy - dx.
4. Start at the first point (x1, y1).
5. For each step until x = x2:
     ● Plot the pixel using putpixel(x, y).
     ● Update the decision parameter m:
     ● If m < 0: increment x and update m += 2*dy.
     ● Else: increment both x and y, and update m += 2*dy - 2*dx.
```

PROGRAMS

```
#include <graphics.h>
#include <iostream.h>
#include <conio.h>

int main() {
    int x, y, x1, y1, x2, y2, dx, dy, m, i;
    int gd = DETECT, gm;

    initgraph(&gd, &gm,"c:\\turboc3\\bgi");

    cout << "Enter first point (x1 y1): ";
    cin >> x1 >> y1;
    cout << "Enter second point (x2 y2): ";
    cin >> x2 >> y2;

    dx = x2 - x1;
    dy = y2 - y1;
    m = 2 * dy - dx;
    x = x1;
    y = y1;

    for(i=0; i <= dx; i++) {
        putpixel(x, y, WHITE);

        if (m < 0) {
            x = x + 1;
            m = m + 2 * dy;
        } else {
            x = x + 1;
            y = y + 1;
            m = m + 2 * dy - 2 * dx;
        }
    }

    getch();
    closegraph();
    return 0;
}
```

Output:



```
Enter first point (x1 y1): 50 50
Enter second point (x2 y2): 200 100
```

RESULTS:

The program successfully implements Bresenham’s Line Drawing Algorithm and draws a line between two points using only integer calculations.

CONCLUSION:

The experiment demonstrates the working of Bresenham’s algorithm. It is more efficient than the DDA algorithm for line drawing, especially in terms of performance since it avoids floating-point arithmetic.

LAB 3

OBJECTIVE: To implement and understand the Midpoint Ellipse Drawing Algorithm using C++ and graphics.h library.

THEORY:

The Midpoint Ellipse Algorithm is used to draw ellipses based on the properties of symmetry. It divides the drawing of the ellipse into two regions:

- Region 1: Where the slope of the ellipse is < 1
- Region 2: Where the slope is >= 1

The algorithm uses a decision parameter to decide the next pixel location. Because of symmetry, for each point (x, y), the algorithm plots points in all four quadrants of the ellipse.

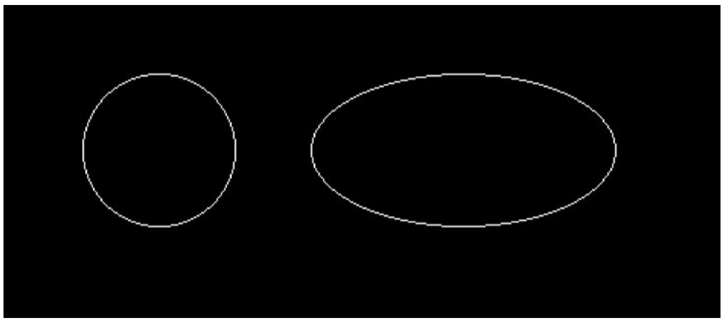
ALGORITHM:

1. Input center (xc, yc) and radii (rx, ry)
2. Initialize x = 0, y = ry
3. Region 1:
   - Use the decision parameter to decide whether to move in x or y
4. Region 2:
   - Transition after slope becomes > 1
   - Adjust decision parameters accordingly
5. Plot 4 symmetrical points each iteration using putpixel()
6. Repeat until full ellipse is drawn

PROGRAMS

```
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

// Function to draw a circle using the Midpoint
Algorithm
void drawCircle(int xc, int yc, int rad) {
    int pnc = 1 - rad; // Decision parameter
    int x = 0;
    int y = rad;

    while (x <= y) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);
        putpixel(xc + y, yc + x, WHITE);
        putpixel(xc - y, yc + x, WHITE);
        putpixel(xc + y, yc - x, WHITE);
        putpixel(xc - y, yc - x, WHITE);

        x++;
        if (pnc < 0) {
            pnc = pnc + 2 * x + 1;
        } else {
            y--;
            pnc = pnc + 2 * (x - y) + 1;
        }
    }
}

// Function to draw an ellipse using the
Midpoint Algorithm
void drawEllipse(int xc, int yc, int rx, int ry) {
    float x = 0, y = ry;

    float rxSq = rx * rx;
    float rySq = ry * ry;

    float dx = 2 * rySq * x;
    float dy = 2 * rxSq * y;

    float p1 = rySq - (rxSq * ry) + (0.25 *
rxSq);

    // Region 1
    while (dx < dy) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);

        x++;
        dx = 2 * rySq * x;

        if (p1 < 0) {
            p1 = p1 + dx + rySq;
        } else {
            y--;
            dy = 2 * rxSq * y;
            p1 = p1 + dx - dy + rySq;
        }
    }

    // Region 2
    float p2 = (rySq) * (x + 0.5) * (x + 0.5) +
(rxSq) * (y - 1) * (y - 1) - (rxSq * rySq);

    while (y >= 0) {
        putpixel(xc + x, yc + y, WHITE);
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);

        y--;
        dy = 2 * rxSq * y;

        if (p2 > 0) {
            p2 = p2 - dy + rxSq;
        } else {
            x++;
            dx = 2 * rySq * x;
            p2 = p2 + dx - dy + rxSq;
        }
    }
}


int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "c:\\turboc3\\bgi");

    // Draw a circle
    drawCircle(200, 200, 50);

    // Draw an ellipse
    drawEllipse(400, 200, 100, 50);

    getch();
    closegraph();
    return 0;
}
```

Output:



RESULTS:

The Midpoint Ellipse Drawing Algorithm was successfully implemented and executed. The output matched the expected ellipse shape with smooth curvature in all four quadrants.

CONCLUSION:

This lab helped understand the working of the Midpoint Ellipse Algorithm, which is efficient for drawing ellipses using only integer calculations and symmetrical pixel plotting. The decision parameter technique minimizes computational overhead.