

Title: Lab Report No.1

Course title: Computer Graphics Laboratory

Course code: CSE-304

3rd Year 1st Semester Examination 2022

Date of Submission: 9 July 2023



Submitted to-

Dr. Mohammad Shorif Uddin

Professor

Department of Computer Science and Engineering

Jahangirnagar University

Savar, Dhaka-1342

Dr. Morium Akter

Associate Professor

Department of Computer Science and Engineering

Jahangirnagar University

Savar, Dhaka-1342

Sl	Class Roll	Exam Roll	Name
01	388	202200	Md. Tanvir Hossain Saon

Department of Computer Science and Engineering

Jahangirnagar University

Savar, Dhaka, Bangladesh

Name of the Experiment: Scan Convert a Point.

Introduction:

Scan converting a point is a crucial step in computer graphics, where the coordinates of a point are transformed into pixel coordinates for display on a screen or printing. It is a fundamental operation that serves as the foundation for rendering various geometric shapes and objects. This process involves mapping continuous point coordinates onto a discrete grid of pixels.

Source Code:

```
#include <graphics.h>
#include <iostream>
#include <conio.h> using
namespace std; int
main()
{
    int gd = DETECT, gm;
    int a, b;
    cout << "Enter the x-coordinate: ";
    cin >> a;
    cout << "Enter the y-coordinate: ";
    cin >> b;
    initgraph(&gd, &gm, "");
    putpixel(a, b, WHITE);
    getch();
    closegraph();
    return 0;

}
```

Output:

```
Enter the x-coordinate: 40  
Enter the y-coordinate: 50
```



Windows BGI

Discussion:

Scan converting a point entails converting its continuous coordinates into pixel coordinates to accurately represent it on a digital display. The most basic approach is the nearest-neighbor algorithm, which assigns the pixel closest to the point's coordinates as its representative pixel. While this method is simple and fast, it can produce jagged edges and aliasing artifacts when the point falls between pixels.

To address aliasing issues, techniques like anti-aliasing and super sampling are employed. Anti-aliasing involves blending the colors of neighboring pixels based on the point's coverage within each pixel, resulting in smoother edges and reduced aliasing artifacts. Super sampling involves sampling multiple positions within a pixel and averaging the results to produce higher-quality and more accurate representations, albeit at a higher computational cost.

Another consideration in scan converting a point is its size or thickness. Typically, points are represented by single pixels, which can be visually indistinguishable at small sizes. To overcome this limitation, techniques such as using larger pixels or employing point sprites, which are textured quads or circles that can be scaled, are utilized to represent points of varying sizes.

In addition to basic scan conversion, other factors come into play depending on the graphics system or application. For instance, in 3D graphics, points need to be transformed from 3D world coordinates to 2D screen coordinates through perspective projection and clipping. The point's attributes, such as color, transparency, and texture coordinates, may also be taken into account during scan conversion to ensure accurate rendering.

In conclusion, scan converting a point is an essential process in computer graphics that involves mapping continuous point coordinates onto a grid of pixels. The choice of algorithm, anti-aliasing techniques, and representation of point size significantly impact the quality and fidelity of the rendered image. By employing appropriate scan conversion techniques, graphics programmers can achieve accurate and visually pleasing representations of points in digital graphics.

Name of the Experiment: Scan Convert a Line (DDA Algorithm).

Introduction:

Scan converting a point is a crucial step in computer graphics, where the coordinates of a point are transformed into pixel coordinates for display on a screen or printing. It is a fundamental operation that serves as the foundation for rendering various geometric shapes and objects. This process involves mapping continuous point coordinates onto a discrete grid of pixels.

Source Code:

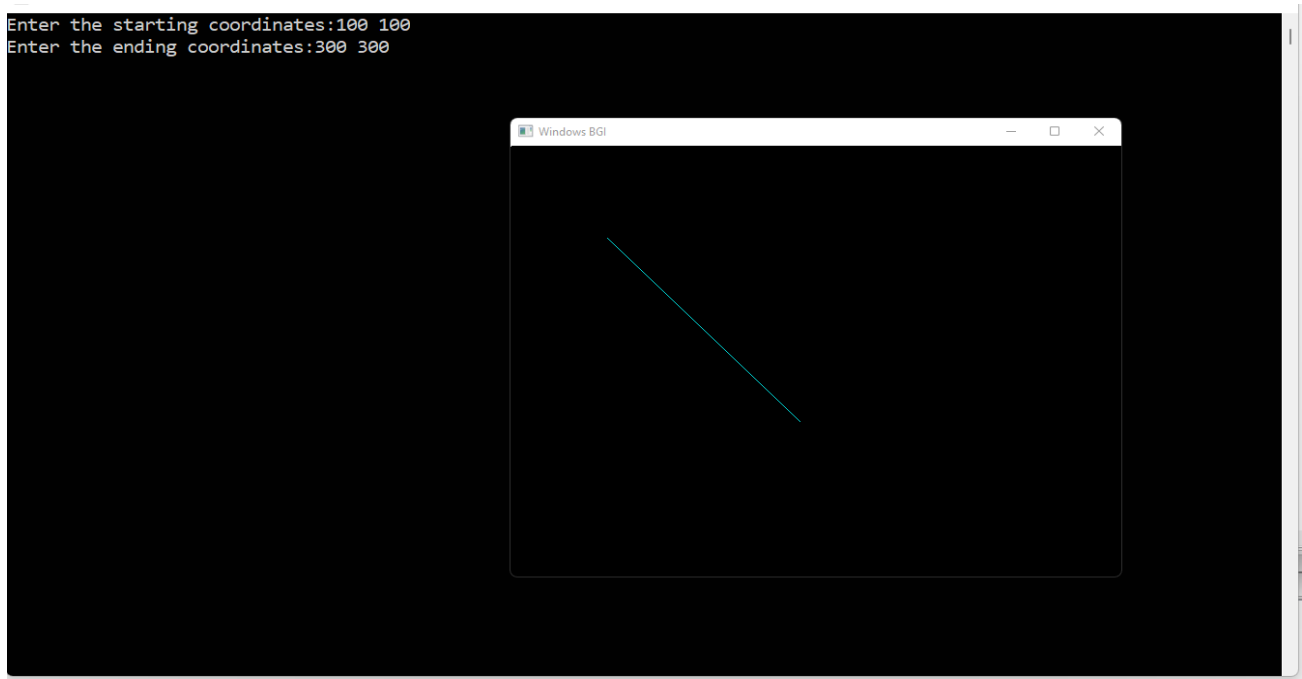
```
#include <graphics.h>
#include <iostream>
#include <conio.h>
using namespace std;
int main() {
    int gd = DETECT, gm;
    int x1, y1, x2, y2;
    int steps, xinc, yinc, dx, dy;
    cout << "Enter the starting coordinates:";
    cin >> x1 >> y1;
    cout << "Enter the ending coordinates:"; cin
    >> x2 >> y2;
    initgraph(&gd, &gm, "");
    dx = x2 - x1;
    dy = y2 - y1;

    if(abs(dx) > abs(dy)) {
        steps = abs(dx);
    } else {
        steps = abs(dy);
    }
    xinc = dx / steps;
    yinc = dy / steps;

    for (int i = 1; i <= steps; i++) {
        putpixel(x1, y1, CYAN);
        delay(10);
        x1 = x1 + xinc;
        y1 = y1 + yinc;
    }
    getch();
}
```

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

Output:



Discussion:

Scan converting a point entails converting its continuous coordinates into pixel coordinates to accurately represent it on a digital display. The most basic approach is the nearest-neighbor algorithm, which assigns the pixel closest to the point's coordinates as its representative pixel. While this method is simple and fast, it can produce jagged edges and aliasing artifacts when the point falls between pixels.

In conclusion, scan converting a point is an essential process in computer graphics that involves mapping continuous point coordinates onto a grid of pixels. The choice of algorithm, anti-aliasing techniques, and representation of point size significantly impact the quality and fidelity of the rendered image. By employing appropriate scan conversion techniques, graphics programmers can achieve accurate and visually pleasing representations of points in digital graphics.

Name of the Experiment: Scan Convert a Line (Bresenham's Algorithm).

Introduction:

Scan converting a line is a fundamental operation in computer graphics that involves determining which pixels should be plotted to accurately represent a line on a rasterized image. Various algorithms have been developed for this purpose, including Bresenham's algorithm. Bresenham's algorithm is known for its efficiency and ability to produce visually accurate results. It is widely used in graphics systems and applications for rendering lines with optimal performance.

Source Code:

```
#include <graphics.h>
#include <iostream>
#include <conio.h> using
namespace std;
void drawline(int x0, int y0, int x1, int y1) {
    int dx, dy, p, x, y;
    dx = x1 - x0;
    dy = y1 - y0; x
    = x0;
    y = y0;
    p = 2 * dy - dx;

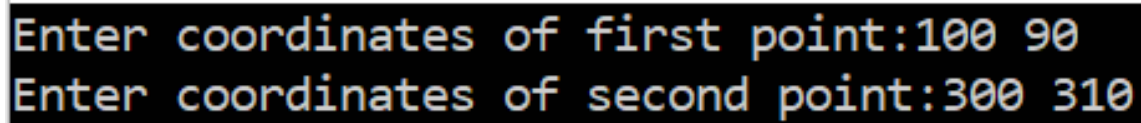
    while (x < x1) {
        if (p >= 0) {
            putpixel(x, y, BLUE);
            y = y + 1;
            p = p + 2 * dy - 2 * dx;
        } else {
            putpixel(x, y, BLUE);
            p = p + 2 * dy;
        }
        x = x + 1;
        delay(10);
    }
}

int main() {
    int gd = DETECT, gm, error, x0, y0, x1, y1;

    cout << "Enter coordinates of first point:"; cin
    >> x0 >> y0;
```

```
cout << "Enter coordinates of second point:"; cin  
>> x1 >> y1;  
initgraph(&gd, &gm, "");  
drawline(x0, y0, x1, y1);  
  
getch();  
closegraph();  
return 0;  
}
```

Output:



```
Enter coordinates of first point:100 90  
Enter coordinates of second point:300 310
```



Discussion:

Bresenham's algorithm, named after its creator Jack E. Bresenham, is a popular method for scan converting lines in computer graphics. It is based on the concept of calculating an error term at each step to determine the best pixel to plot along the line's path. By utilizing integer calculations and avoiding expensive floating-point operations, Bresenham's algorithm offers high performance and efficiency.

The algorithm starts by determining the differences in x and y coordinates between the line's starting and ending points. Based on the signs of these differences, the slope of the line is determined. This information is crucial for accurately plotting the line and ensuring it is correctly rendered on the screen.

Bresenham's algorithm uses an iterative approach to scan convert the line. It tracks the current position on the line and calculates an error term that determines the next pixel to plot. The algorithm progressively updates the error term and adjusts the position based on comparisons with the values of dx and dy. This process continues until the algorithm reaches the ending point of the line.

One of the key advantages of Bresenham's algorithm is its ability to produce accurate and visually pleasing line representations. It ensures that the line stays close to the true mathematical path, resulting in clean and crisp lines without gaps or overlaps between pixels. Additionally, the algorithm's efficiency makes it suitable for real-time rendering and applications with limited computational resources.

While Bresenham's algorithm is primarily designed for lines with slopes between 0 and 1, adaptations and extensions have been developed to handle other line slopes and support thicker lines. These variations of the algorithm allow for greater versatility and application in a wider range of scenarios.

In conclusion, Bresenham's algorithm is a widely used and efficient method for scan converting lines in computer graphics. Its ability to accurately determine which pixels to plot along the line's path, combined with its high performance, has made it a popular choice in graphics systems and applications. By leveraging integer calculations and an iterative process, Bresenham's algorithm enables the generation of visually pleasing and precise line representations in digital images.

Name of the Experiment: Scan Convert a Circle (Bresenham's Circle Algorithm).

Introduction:

Scan converting a circle is a fundamental operation in computer graphics that involves determining which pixels should be plotted to accurately represent a circle on a rasterized image. Bresenham's circle algorithm, an extension of Bresenham's line algorithm, is a popular method for scan converting circles. It is known for its efficiency and ability to produce visually accurate results, making it widely used in graphics systems and applications.

Source Code:

```
#include <stdio.h>
#include <dos.h> #include
<graphics.h>

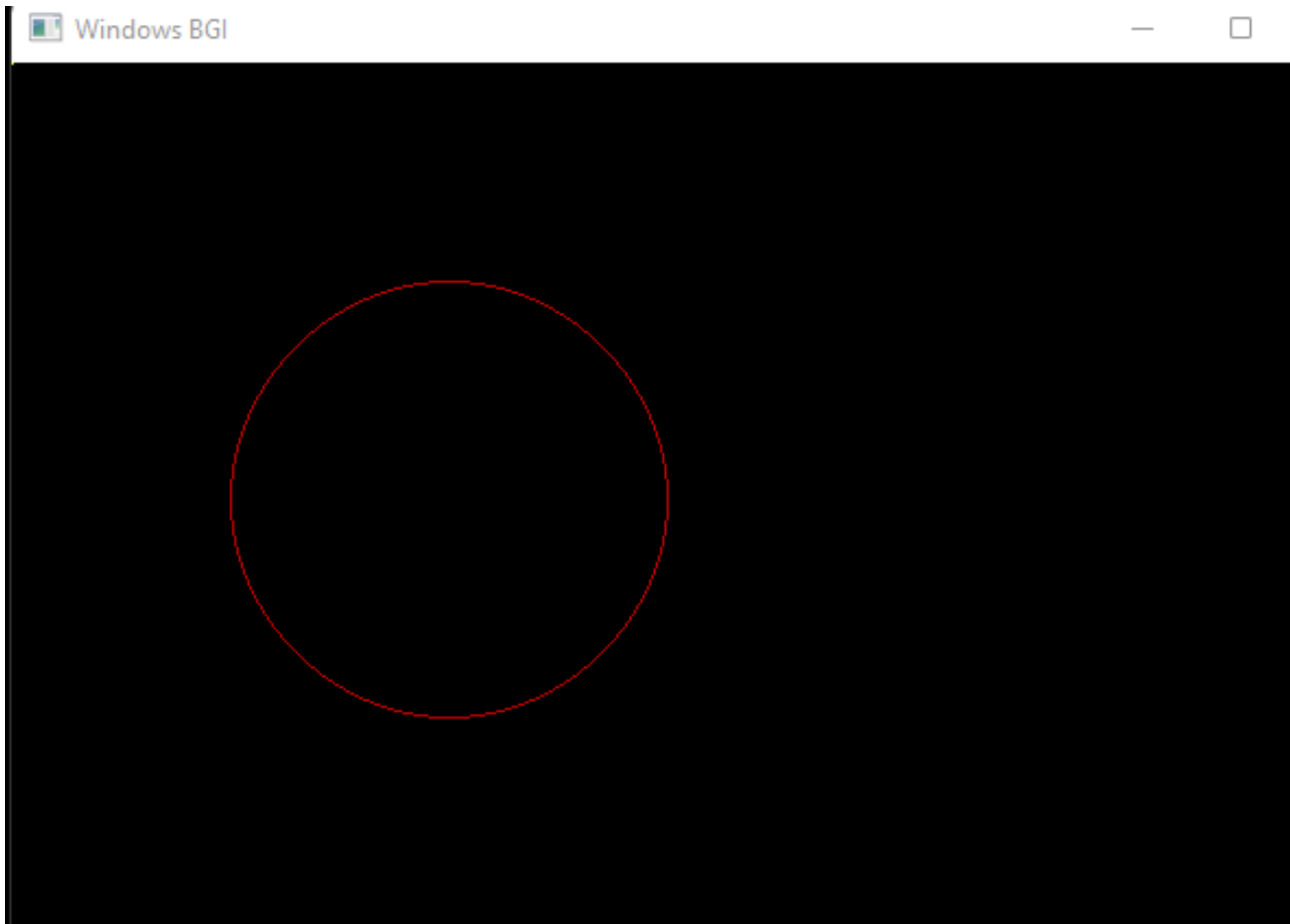
void drawCircle(int xc, int yc, int x, int y)
{
    putpixel(xc+x, yc+y, RED);
    putpixel(xc-x, yc+y, RED);
    putpixel(xc+x, yc-y, RED);
    putpixel(xc-x, yc-y, RED);
    putpixel(xc+y, yc+x, RED);
    putpixel(xc-y, yc+x, RED);
    putpixel(xc+y, yc-x, RED);
    putpixel(xc-y, yc-x, RED);
}

void circleBres(int xc, int yc, int r)
{
    int x = 0, y = r;
    int d = 3 - 2 * r;
    drawCircle(xc, yc, x, y); while
    (y >= x)
    {
        x++;

        if (d > 0)
        {
            y--;
            d = d + 4 * (x - y) + 10;
        }
        else
            d = d + 4 * x + 6;
        drawCircle(xc, yc, x, y); delay(50);
    }
}
```

```
int main()
{
    int xc = 200, yc = 200, r = 100;
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    circleBres(xc, yc, r);
    getch();
    closegraph();
    return 0;
}
```

Output:



Discussion:

Bresenham's circle algorithm, developed by Jack E. Bresenham, is a well-known method for scan converting circles in computer graphics. It utilizes integer calculations and avoids costly floating-point operations, providing high performance and efficiency.

The algorithm starts with the coordinates of the circle's center and its radius. It iteratively calculates the coordinates of pixels that lie on the circle's circumference, ensuring an accurate representation.

Bresenham's circle algorithm uses a decision criterion to determine the best pixel to plot at each step. It considers the current position and the midpoint between two potential pixels, evaluating which one lies closest to the true circle path. This decision is based on an error term that tracks the difference between the actual distance and the ideal distance from the circle's circumference.

The algorithm proceeds by updating the error term and adjusting the position based on comparisons with the radius and the x and y coordinates. This process continues until the entire circle is scan converted.

One of the main advantages of Bresenham's circle algorithm is its efficiency. By using integer calculations and avoiding complex mathematical computations, the algorithm achieves fast and accurate results. It is particularly valuable in real-time graphics applications and systems with limited computational resources.

Bresenham's circle algorithm produces visually pleasing circle representations with smooth edges and minimal aliasing artifacts. The algorithm ensures that the plotted pixels lie close to the true mathematical path of the circle, resulting in accurate and visually appealing renderings.

While Bresenham's circle algorithm is primarily designed for circles with equal horizontal and vertical radii, adaptations and extensions have been proposed to handle circles with different radii or to support filled circles.

In conclusion, Bresenham's circle algorithm is a widely used and efficient method for scan converting circles in computer graphics. By utilizing integer calculations and a decision criterion based on error terms, the algorithm accurately determines which pixels to plot along the circle's circumference. Its efficiency and ability to produce visually accurate results have made it a popular choice in graphics systems and applications. Bresenham's circle algorithm allows for the generation of precise and visually pleasing circle representations in digital images.