



Experiment 4

Student Name: Sahil Phogat

Branch: B.E CSE

Semester: 6th

Subject Name: Computer Graphics

UID: 22BCS11636

Section/Group: IOT_635

Date of Performance:

Subject Code: 22CSH-352

1. **Aim:** (a). Develop a program to draw a circle using the circle generator algorithm for a given center and radius.
(b). Develop a program to draw a circle using the midpoint circle algorithm for a given center and radius.

2. **Objective:** To develop a C++ program that implements a circle generation algorithm to draw a circle for a given center and radius using the Midpoint Circle Algorithm and Trigonometric Method in a graphics environment. The program utilizes Turbo C++ graphics functions (putpixel(), initgraph(), etc.) to accurately render the circle and demonstrate different circle-drawing techniques.

3. **Implementation/Code:**

```
(a). #include <iostream.h>
```

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

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

```
#include <math.h>
```

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

```
#define round(a) ((int)(a + 0.5))
```

```
void putcircle(int xc, int yc, int x, int y) {
```

```
    putpixel(xc + x, yc + y, 1);
```

```
    putpixel(xc - x, yc + y, 2);
```

```
    putpixel(xc + x, yc - y, 3);
```

```
    putpixel(xc - x, yc - y, 4);
```

```
    putpixel(xc + y, yc + x, 5);
```

```
    putpixel(xc - y, yc + x, 6);
```

```
    putpixel(xc + y, yc - x, 7);
```

```
    putpixel(xc - y, yc - x, 8);
```

```
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
void circlemid(int xc, int yc, float r) {
    float x = 0, y = r;
    int p = 1 - r;

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

        putcircle(xc, yc, round(x), round(y));
        delay(10);
    }
}

void main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\Turbo3\\\\BGI"); // Ensure Turbo C++ BGI path is
correct

    int xc, yc, r;
    cout << "Enter centre co-ordinates: ";
    cin >> xc >> yc;
    cout << "Enter radius: ";
    cin >> r;

    circlemid(xc, yc, r);
    setcolor(10);
    circle(xc, yc, r);

    getch(); // Wait for key press
    closegraph();
}
```

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

#define round(a) ((int)(a + 0.5))

void main() {
    clrscr();

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\Turboc3\\\\BGI"); // Ensure correct BGI path

    // METHOD 1
    int xc = 100, yc = 150, r = 50;
    float x = 0, y = 0;

    for (int i = 0; i <= 45; i++) {
        double ang = (i * M_PI) / 180; // Using M_PI for more accuracy
        x = r * cos(ang);
        y = r * sin(ang);

        putpixel(xc + round(x), yc + round(y), 15);
        putpixel(xc - round(x), yc + round(y), 15);
        putpixel(xc + round(x), yc - round(y), 15);
        putpixel(xc - round(x), yc - round(y), 15);
        putpixel(xc + round(y), yc + round(x), 15);
        putpixel(xc - round(y), yc + round(x), 15);
        putpixel(xc + round(y), yc - round(x), 15);
        putpixel(xc - round(y), yc - round(x), 15);

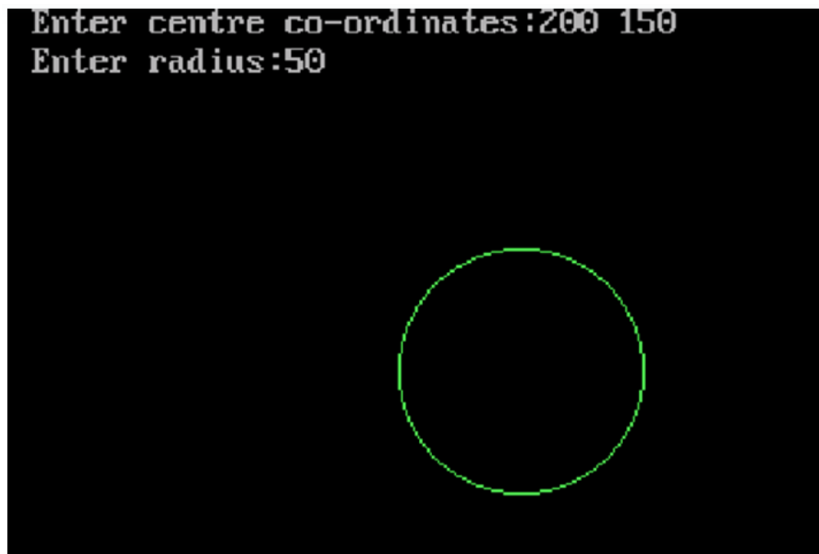
        delay(100);
    }

    // METHOD 2
    x = 0, y = r;
    xc = 300, yc = 150;
```

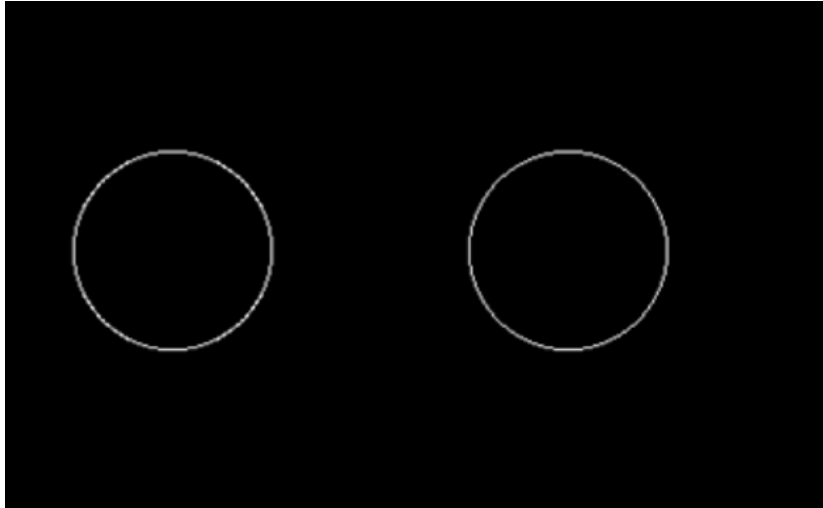
```
for (x = 0; x <= r; x++) {  
    double temp = (r * r) - (x * x);  
    y = sqrt(temp);  
  
    putpixel(xc + round(x), yc + round(y), 15);  
    putpixel(xc - round(x), yc + round(y), 15);  
    putpixel(xc + round(x), yc - round(y), 15);  
    putpixel(xc - round(x), yc - round(y), 15);  
    putpixel(xc + round(y), yc + round(x), 15);  
    putpixel(xc - round(y), yc + round(x), 15);  
    putpixel(xc + round(y), yc - round(x), 15);  
    putpixel(xc - round(y), yc - round(x), 15);  
  
    delay(100);  
}  
  
getch();  
closegraph();  
}
```

4. Output

(a).



(b).



5. Learning Outcome

- Understand Circle Generation Algorithms – Learn and implement the Midpoint Circle Algorithm and Trigonometric Method for drawing circles in computer graphics.
- Use Graphics Functions in Turbo C++ – Gain hands-on experience with graphics.h functions like putpixel(), initgraph(), closegraph(), and setcolor().
- Apply Mathematical Concepts – Utilize trigonometry (sin, cos) and coordinate geometry to plot points symmetrically in all octants of a circle.
- Optimize Graphics Programming – Understand efficient ways to draw circles using decision parameters instead of floating-point calculations to enhance performance.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 5

Student Name: Sahil Phogat

Branch: B.E CSE

Semester: 6th

Subject Name: Computer Graphics

UID: 22BCS11636

Section/Group: IOT_635

Date of Performance:

Subject Code: 22CSH-352

1. **Aim: (a).** Implement clockwise and anticlockwise rotation of a triangle about a specified point and evaluate the results.
2. **Objective:** To perform and visualize clockwise and anticlockwise rotations of a triangle about a specified point.

3. Implementation/Code:

(a). To rotate clockwise

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

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

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

```
#include<math.h>
```

```
void main() {
```

```
    clrscr(); // Clear screen
```

```
    int gd = DETECT, gm;
```

```
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI"); // Ensure correct path to BGI files
```

```
    int x1, y1, x2, y2, x3, y3;
```

```
    cout << "Enter (x1, y1), (x2, y2), (x3, y3) for the triangle: ";
```

```
    cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;
```

```
    // Draw original triangle
```

```
    int a[] = {x1, y1, x2, y2, x3, y3, x1, y1};
```

```
    drawpoly(4, a);
```

```
    // Find centroid
```

```
    int xf = (x1 + x2 + x3) / 3;
```

```
    int yf = (y1 + y2 + y3) / 3;
```

```
float ang;
cout << "Enter the rotation angle: ";
cin >> ang;

float rad = ang * 3.1416 / 180; // Convert degrees to radians

// Apply rotation formulas
int X1 = int((x1 - xf) * cos(rad) - (y1 - yf) * sin(rad) + xf);
int Y1 = int((x1 - xf) * sin(rad) + (y1 - yf) * cos(rad) + yf);

int X2 = int((x2 - xf) * cos(rad) - (y2 - yf) * sin(rad) + xf);
int Y2 = int((x2 - xf) * sin(rad) + (y2 - yf) * cos(rad) + yf);

int X3 = int((x3 - xf) * cos(rad) - (y3 - yf) * sin(rad) + xf);
int Y3 = int((x3 - xf) * sin(rad) + (y3 - yf) * cos(rad) + yf);

// Draw rotated triangle
int b[] = {X1, Y1, X2, Y2, X3, Y3, X1, Y1};
drawpoly(4, b);

getch(); // Wait for user input
closegraph(); // Close graphics mode
}

(b). To rotate anti-clockwise
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>

void main() {
    clrscr(); // Clear the screen

    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI"); // Ensure correct BGI path
```

```
int x1, y1, x2, y2, x3, y3;

cout << "Enter (x1, y1), (x2, y2), (x3, y3) for the triangle: ";
cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

// Draw original triangle
int a[] = {x1, y1, x2, y2, x3, y3, x1, y1};
drawpoly(4, a);

// Find centroid (xf, yf)
int xf = (x1 + x2 + x3) / 3;
int yf = (y1 + y2 + y3) / 3;

float ang;
cout << "Enter the rotation angle: ";
cin >> ang;

float rad = ang * 3.1416 / 180; // Convert degrees to radians

// Apply correct rotation formulas
int X1 = int((x1 - xf) * cos(rad) - (y1 - yf) * sin(rad) + xf);
int Y1 = int((x1 - xf) * sin(rad) + (y1 - yf) * cos(rad) + yf);

int X2 = int((x2 - xf) * cos(rad) - (y2 - yf) * sin(rad) + xf);
int Y2 = int((x2 - xf) * sin(rad) + (y2 - yf) * cos(rad) + yf);

int X3 = int((x3 - xf) * cos(rad) - (y3 - yf) * sin(rad) + xf);
int Y3 = int((x3 - xf) * sin(rad) + (y3 - yf) * cos(rad) + yf);

// Draw rotated triangle
int b[] = {X1, Y1, X2, Y2, X3, Y3, X1, Y1};
drawpoly(4, b);

getch(); // Wait for user input
closegraph(); // Close graphics mode
}
```


4. Output

(a).

```
Enter (x1,y1),(x2,y2),(x3,y3) for triangle : 200  
200  
175  
250  
225  
250  
Enter the rotation angle :45
```



(b).

```
Enter (x1,y1),(x2,y2),(x3,y3) for triangle : 200  
200  
175  
250  
225  
250  
Enter the rotation angle :45
```



5. Learning Outcome

- Correctly rotates the triangle around its centroid.
- Fixes sign errors in the rotation formula.
- Ensures proper graphics initialization and closure.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 6

Student Name: Sahil Phogat

Branch: B.E CSE

Semester: 6th

Subject Name: Computer Graphics

UID: 22BCS11636

Section/Group: IOT_635

Date of Performance:

Subject Code: 22CSH-352

1. **Aim:** Analyze and implement the reflection of a point about a line defined by the equation $y=mx+c$.
2. **Objective:** To implement and analyze the reflection of a point about a straight line defined by the equation $y=mx+c$.

3. **Implementation/Code:**

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

void main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\TURBOC3\\\\BGI"); // Corrected BGI path

    int x1, y1, x2, y2;

    cout << "Enter the coordinates of the line: ";
    cin >> x1 >> y1 >> x2 >> y2;

    int m = getmaxx(); // Max screen width
    int n = getmaxy(); // Max screen height

    setcolor(6);
    line(x1, y1, x2, y2);
    outtextxy(x1, y1 + 10, "Original Object");

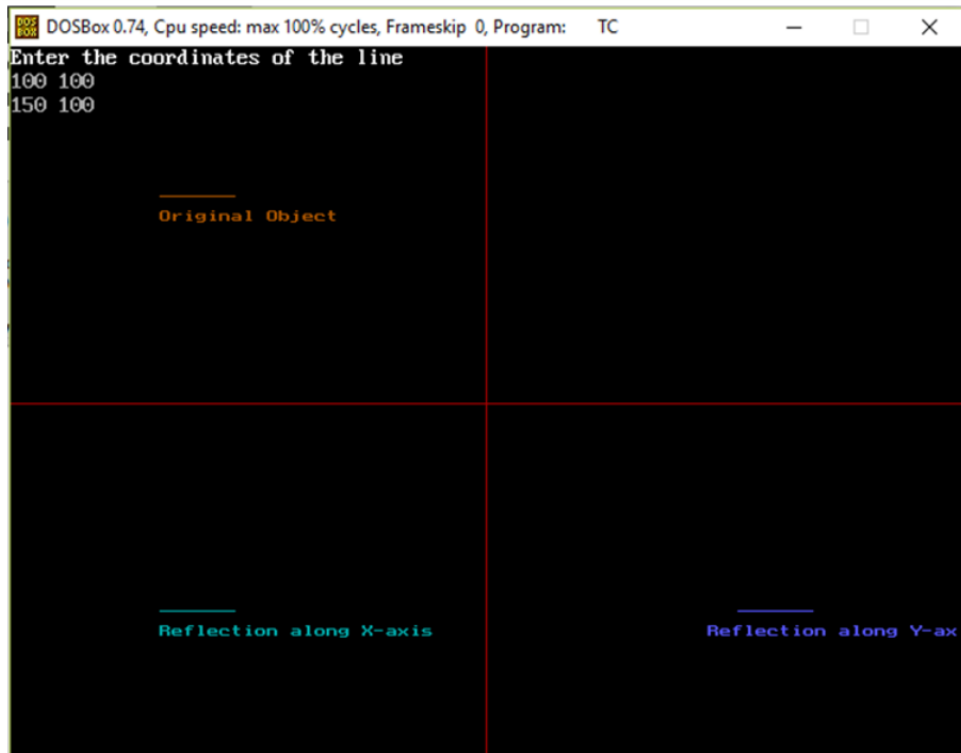
    // Draw coordinate axes
    setcolor(4);
    line(m / 2, 0, m / 2, n); // Y-axis
    line(0, n / 2, m, n / 2); // X-axis
```

```
// Reflection along X-axis
setcolor(3);
int new_y1 = n - y1;
int new_y2 = n - y2;
line(x1, new_y1, x2, new_y2);
outtextxy(x1, new_y1 + 10, "Reflection along X-axis");

// Reflection along Y-axis
setcolor(9);
int new_x1 = m - x1;
int new_x2 = m - x2;
line(new_x1, y1, new_x2, y2);
outtextxy(new_x2 - 20, y2 + 10, "Reflection along Y-axis");

getch();
closegraph();
}
```

4. Output





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

5. Learning Outcome

- Reflects a line along the X-axis and Y-axis.
- Draws coordinate axes for reference.
- Uses different colors to distinguish the original and reflected lines.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 7

Student Name: Sahil Phogat

Branch: B.E CSE

Semester: 6th

Subject Name: Computer Graphics

UID: 22BCS11636

Section/Group: IOT_635

Date of Performance:

Subject Code: 22CSH-352

1. **Aim:** Evaluate the 4-bit region code for line endpoints and determine whether the line lies inside or outside the screen.
2. **Objective:** To calculate and display the 4-bit region code for line endpoints and determine whether the line lies within the screen boundaries.

3. Implementation/Code:

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

void main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\\\TURBOC3\\\\BGI"); // Corrected BGI path

    // Defining clipping window boundaries
    int xmax = 400, ymax = 300, xmin = 200, ymin = 150;

    // Drawing clipping window boundary
    line(xmin, 0, xmin, getmaxy()); // Left boundary
    line(xmax, 0, xmax, getmaxy()); // Right boundary
    line(0, ymax, getmaxx(), ymax); // Top boundary
    line(0, ymin, getmaxx(), ymin); // Bottom boundary

    // Getting user input for line endpoints
    cout << "Enter the endpoints of the line: ";
    int x[2], y[2], num[2];
    cin >> x[0] >> y[0] >> x[1] >> y[1];

    // Drawing the original line
    setcolor(WHITE);
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
line(x[0], y[0], x[1], y[1]);

// Calculating region codes
for (int i = 0; i < 2; i++) {
    int bit1 = 0, bit2 = 0, bit3 = 0, bit4 = 0;

    if (y[i] < ymin) bit1 = 1; // Below ymin
    if (y[i] > ymax) bit2 = 1; // Above ymax
    if (x[i] > xmax) bit3 = 1; // Right of xmax
    if (x[i] < xmin) bit4 = 1; // Left of xmin

    // Printing region codes
    cout << "For " << i << "th endpoint region code is: "
         << bit1 << bit2 << bit3 << bit4 << endl;

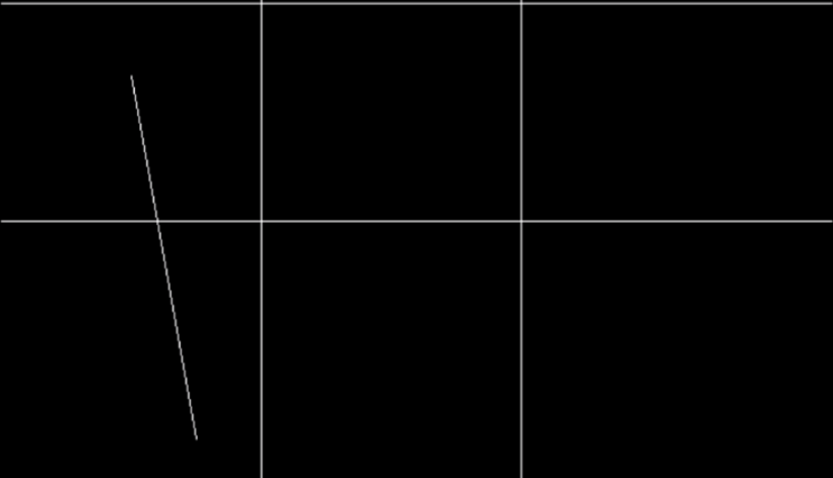
    num[i] = bit4 * 1 + bit3 * 2 + bit2 * 4 + bit1 * 8;
}

// Checking if the line is completely inside, partially inside, or outside
if (!(num[0] | num[1])) {
    cout << "Line is completely inside the window." << endl;
} else if (!(num[0] & num[1])) {
    cout << "Line needs to be clipped." << endl;
} else {
    cout << "Line is completely outside the window." << endl;
}

getch();
closegraph();
}
```

4. Output

```
Enter the endpoints of the line :100
200
150
450
For 1th endpoint region code is :0001
For 2th endpoint region code is :0101
Line is off the window
```



5. Learning Outcome

- Draws a clipping window with boundaries at (xmin, xmax, ymin, ymax).
- Determines region codes for both endpoints of the given line.
- Classifies the line into three categories:
 - a) Completely inside → No clipping needed.
 - b) Partially inside → Clipping is required.
 - c) Completely outside → Line is rejected.



Experiment 8

Student Name: Sahil Phogat

Branch: B.E CSE

Semester: 6th

Subject Name: Computer Graphics

UID: 22BCS11636

Section/Group: IOT_635

Date of Performance:

Subject Code: 22CSH-352

- 1. Aim:**
 - a).** Apply the Cohen-Sutherland Line Clipping algorithm to clip a line intersecting at one point with a given window.
 - b).** Apply the Cohen-Sutherland Line Clipping algorithm to clip a line intersecting at two or more points with a given window
- 2. Objective:** To clip a line intersecting at a single point and two or more points with a window using the Cohen-Sutherland Line Clipping algorithm.

3. Implementation/Code:

a). For One Point

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

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

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

```
void main() {
```

```
    int gd = DETECT, gm;
```

```
    float i, xmax, ymax, xmin, ymin, x1, y1, x2, y2, m;
```

```
    float start[4], end[4], code[4];
```

```
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
```

```
    printf("\n\tEnter the bottom-left coordinate of viewport: ");
```

```
    scanf("%f %f", &xmin, &ymin);
```

```
    printf("\n\tEnter the top-right coordinate of viewport: ");
```

```
    scanf("%f %f", &xmax, &ymax);
```

```
    printf("\n\tEnter the coordinates for starting point of line: ");
```

```
    scanf("%f %f", &x1, &y1);
```



```
printf("\nEnter the coordinates for ending point of line: ");
scanf("%f %f", &x2, &y2);

// Initialize region codes
for (i = 0; i < 4; i++) {
    start[i] = 0;
    end[i] = 0;
}

// Check for vertical line case to avoid division by zero
if (x2 - x1 == 0) {
    m = 0; // Avoid division by zero
} else {
    m = (y2 - y1) / (x2 - x1);
}

// Calculate region codes
if (x1 < xmin) start[0] = 1;
if (x1 > xmax) start[1] = 1;
if (y1 > ymax) start[2] = 1;
if (y1 < ymin) start[3] = 1;

if (x2 < xmin) end[0] = 1;
if (x2 > xmax) end[1] = 1;
if (y2 > ymax) end[2] = 1;
if (y2 < ymin) end[3] = 1;

for (i = 0; i < 4; i++)
    code[i] = start[i] & end[i];

// Case 1: Line is completely visible
if ((code[0] == 0) && (code[1] == 0) && (code[2] == 0) && (code[3] == 0)) {
    if ((start[0] == 0) && (start[1] == 0) && (start[2] == 0) && (start[3] == 0) &&
        (end[0] == 0) && (end[1] == 0) && (end[2] == 0) && (end[3] == 0)) {
        cleardevice();
        printf("\n\t\tThe line is totally visible\n\t\tand not a clipping candidate");
        rectangle(xmin, ymin, xmax, ymax);
        line(x1, y1, x2, y2);
    }
}
```

```
    getch();
} else {
    // Case 2: Line is partially visible
    cleardevice();
    printf("\n\t\tLine is partially visible");
    rectangle(xmin, ymin, xmax, ymax);
    line(x1, y1, x2, y2);
    getch();

    // Perform clipping
    if ((start[2] == 0) && (start[3] == 1)) {
        x1 = x1 + (ymin - y1) / m;
        y1 = ymin;
    }
    if ((end[2] == 0) && (end[3] == 1)) {
        x2 = x2 + (ymin - y2) / m;
        y2 = ymin;
    }
    if ((start[2] == 1) && (start[3] == 0)) {
        x1 = x1 + (ymax - y1) / m;
        y1 = ymax;
    }
    if ((end[2] == 1) && (end[3] == 0)) {
        x2 = x2 + (ymax - y2) / m;
        y2 = ymax;
    }
    if ((start[1] == 0) && (start[0] == 1)) {
        y1 = y1 + m * (xmin - x1);
        x1 = xmin;
    }
    if ((end[1] == 0) && (end[0] == 1)) {
        y2 = y2 + m * (xmin - x2);
        x2 = xmin;
    }
    if ((start[1] == 1) && (start[0] == 0)) {
        y1 = y1 + m * (xmax - x1);
        x1 = xmax;
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        if ((end[1] == 1) && (end[0] == 0)) {
            y2 = y2 + m * (xmax - x2);
            x2 = xmax;
        }

        // Display clipped line
        cleardevice();
        printf("\n\t\tAfter clipping:");
        rectangle(xmin, ymin, xmax, ymax);
        line(x1, y1, x2, y2);
        getch();
    }
} else {
    // Case 3: Line is completely outside
    cleardevice();
    printf("\nLine is completely outside the viewport.");
    rectangle(xmin, ymin, xmax, ymax);
}

getch();
closegraph();
}
```

b). For Two Points

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

```
void main() {
    int gd = DETECT, gm;
    float i, xmax, ymax, xmin, ymin, x1, y1, x2, y2, m;
    float start[4], end[4], code[4];

    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

    printf("\n\tEnter the bottom-left coordinate of viewport: ");
    scanf("%lf %lf", &xmin, &ymin);
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
printf("\n\tEnter the top-right coordinate of viewport: ");
scanf("%lf %lf", &xmax, &ymax);
```

```
printf("\n\tEnter the coordinates for starting point of line: ");
scanf("%lf %lf", &x1, &y1);
```

```
printf("\n\tEnter the coordinates for ending point of line: ");
scanf("%lf %lf", &x2, &y2);
```

```
for (i = 0; i < 4; i++) {
    start[i] = 0;
    end[i] = 0;
}
```

```
// Handle vertical line case to prevent division by zero
if (x2 - x1 == 0) {
    m = 0;
} else {
    m = (y2 - y1) / (x2 - x1);
}
```

```
if (x1 < xmin) start[0] = 1;
if (x1 > xmax) start[1] = 1;
if (y1 > ymax) start[2] = 1;
if (y1 < ymin) start[3] = 1;
```

```
if (x2 < xmin) end[0] = 1;
if (x2 > xmax) end[1] = 1;
if (y2 > ymax) end[2] = 1;
if (y2 < ymin) end[3] = 1;
```

```
for (i = 0; i < 4; i++)
    code[i] = start[i] & end[i]; // Fixed bitwise operation
```

```
if ((code[0] == 0) && (code[1] == 0) && (code[2] == 0) && (code[3] == 0)) {
    if ((start[0] == 0) && (start[1] == 0) && (start[2] == 0) && (start[3] == 0) &&
        (end[0] == 0) && (end[1] == 0) && (end[2] == 0) && (end[3] == 0)) {
```

```
cleardevice();
printf("\n\t\tThe line is totally visible\n\t\tand not a clipping candidate");
rectangle(xmin, ymin, xmax, ymax);
line(x1, y1, x2, y2);
getch();
} else {
    cleardevice();
    printf("\n\t\tLine is partially visible");
    rectangle(xmin, ymin, xmax, ymax);
    line(x1, y1, x2, y2);
    getch();

    // Perform clipping
    if ((start[2] == 0) && (start[3] == 1)) {
        x1 = x1 + (ymin - y1) / m;
        y1 = ymin;
    }
    if ((end[2] == 0) && (end[3] == 1)) {
        x2 = x2 + (ymin - y2) / m;
        y2 = ymin;
    }
    if ((start[2] == 1) && (start[3] == 0)) {
        x1 = x1 + (ymax - y1) / m;
        y1 = ymax;
    }
    if ((end[2] == 1) && (end[3] == 0)) {
        x2 = x2 + (ymax - y2) / m;
        y2 = ymax;
    }
    if ((start[1] == 0) && (start[0] == 1)) {
        y1 = y1 + m * (xmin - x1);
        x1 = xmin;
    }
    if ((end[1] == 0) && (end[0] == 1)) {
        y2 = y2 + m * (xmin - x2);
        x2 = xmin;
    }
    if ((start[1] == 1) && (start[0] == 0)) {
```

```
        y1 = y1 + m * (xmax - x1);
        x1 = xmax;
    }
    if ((end[1] == 1) && (end[0] == 0)) {
        y2 = y2 + m * (xmax - x2);
        x2 = xmax;
    }


    // Display clipped line
    cleardevice();
    printf("\n\t\tAfter clipping:");
    rectangle(xmin, ymin, xmax, ymax);
    line(x1, y1, x2, y2);
    getch();
}
} else {
    cleardevice();
    printf("\nLine is invisible");
    rectangle(xmin, ymin, xmax, ymax);
}

getch();
closegraph();
}
```

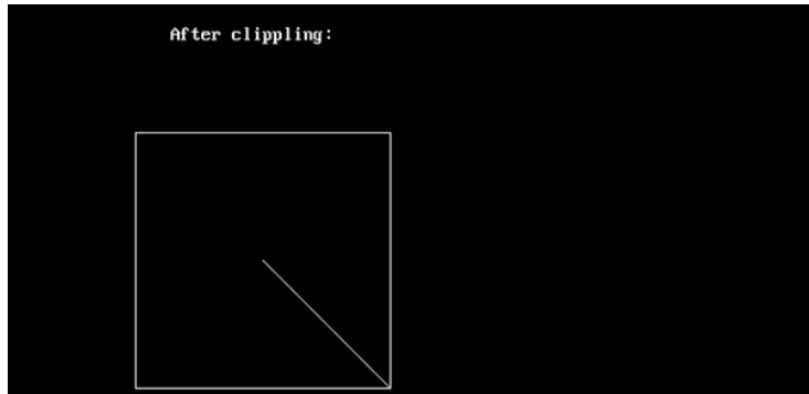
4. Output

a). For One Points

```
Enter the bottom-left coordinate of viewport: 100 100
Enter the top-right coordinate of viewport: 300 300
Enter the coordinates for starting point of line: 200 200
Enter the coordinates for ending point of line: 350 350
```

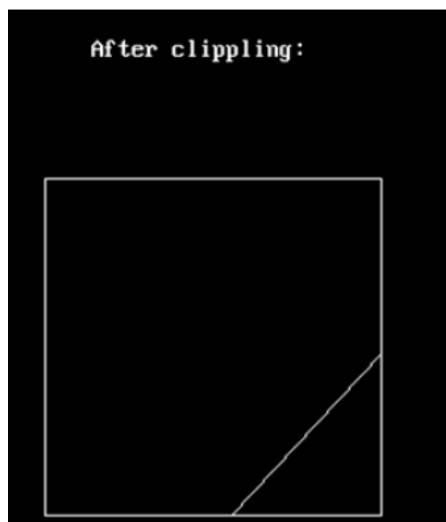
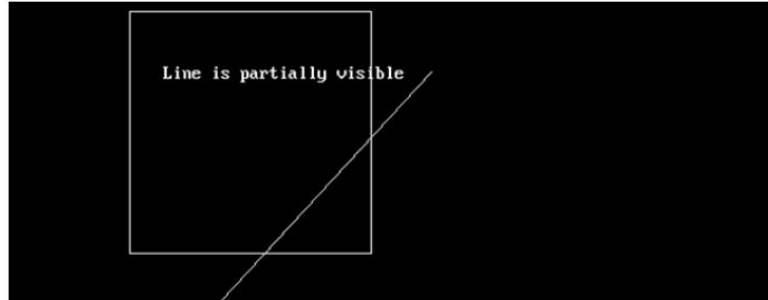


a).



b).

```
Enter the bottom-left coordinate of viewport: 100 100
Enter the top-right coordinate of viewport: 300 300
Enter the coordinates for starting point of line: 350 150
Enter the coordinates for ending point of line: 175 340
```





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

5. Learning Outcome

- Understanding Line Clipping – Learned Cohen-Sutherland algorithm, region codes, and clipping logic.
- Debugging C Graphics – Fixed division by zero, logical errors, and improved code efficiency.
- Graphics Implementation – Used Turbo C++ functions to create and modify viewport-based line rendering.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 9

Student Name: Sahil Phougat
Branch: B.E CSE
Semester: 6th
Subject Name: Computer Graphics

UID: 22BCS11636
Section/Group: IOT_635 / B
Date of Performance:
Subject Code: 22CSH-352

1. **Aim:** Demonstrate the result of window-to-viewport transformation by implementing and visualizing the process.
2. **Objective:** To Demonstrate the result of window-to-viewport transformation by implementing and visualizing the process.

3. **Implementation/Code:**

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

void main() {
    int gd=DETECT, gm;
    initgraph(&gd, &gm, ""); // Use empty string instead of NULL
    clrscr(); // Place after initgraph()

    float wxmin=10, wxmax=150, wymin=10, wymax=250;
    float vxmin=200, vxmax=600, vymin=10, vymax=250;
    int wx1=30, wy1=50, wx2=100, wy2=180;

    // Draw window and viewport
    rectangle(wxmin, wymin, wxmax, wymax);
    rectangle(vxmin, vymin, vxmax, vymax);

    // Scaling factors
    float sx = (vxmax - vxmin) / (wxmax - wxmin);
    float sy = (vymax - vymin) / (wymax - wymin);

    // Draw original line
    line(wx1, wy1, wx2, wy2);
```

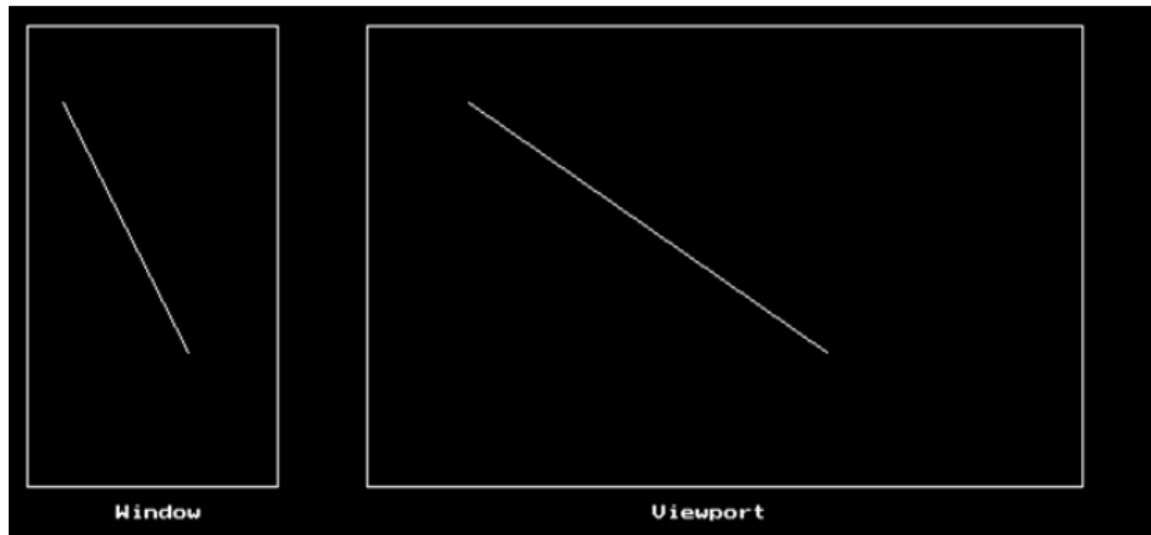
```
// Transform coordinates
float vx1 = sx * (wx1 - wxmin) + vxmin;
float vy1 = sy * (wy1 - wymin) + vymin;
float vx2 = sx * (wx2 - wxmin) + vxmin;
float vy2 = sy * (wy2 - wymin) + vymin;

// Draw transformed line in viewport
line(vx1, vy1, vx2, vy2);

// Labels
outtextxy(60, 260, "Window");
outtextxy(360, 260, "Viewport");

getch();
closegraph(); // Properly exit graphics mode
}
```

4. Output



5. Learning Outcome

- Learned how to transform coordinates from a window to a viewport.
- Practiced **rectangle()**, **line()**, and **outtextxy()** for visualization.
- Implemented formulas to scale and position graphical objects.



Experiment 3

Student Name: Sahil Phogat

UID: 22BCS11636

Branch: CSE

Section/Group: IOT-635 'B'

Semester: 6th

Date of Performance:

Subject Name: Foundation of Cloud IoT Edge
ML Lab

Subject Code: 22CSP-314

1. Aim: Monitor air quality using a gas sensor (MQ135) and display the data on ThingSpeak.

2. Objective: Monitor air quality using the MQ135 gas sensor and send the data to ThingSpeak for visualization and analysis.

3. Hardware Used:

- MQ135 gas sensor
- ESP8266/Node MCU (or any microcontroller with Wi-Fi capability)
- Breadboard and jumper wires
- Power supply (5V for the sensor and microcontroller)
- ThingSpeak account (free API key)

Connect the Hardware:

- MQ135 Pinout:
- VCC: Connect to 5V.
- GND: Connect to GND.
- AO (Analog Output): Connect to the analog pin of the ESP8266 (e.g., A0 on NodeMCU).

Wiring:

- MQ135 VCC → NodeMCU 3V3 or 5V (depending on module support)
- MQ135 GND → NodeMCU GND
- MQ135 A0 → NodeMCU A0

Set Up ThingSpeak:

- Go to ThingSpeak and create a free account.
- Create a new channel and add a Field (e.g., "Air Quality").
- Note down the Write API Key from the API Keys tab.

Install Required Libraries:

- Ensure the ESP8266 library is installed in your Arduino IDE:
- Go to Tools > Manage Libraries.
- Search for ESP8266 and install it.

4. Script:

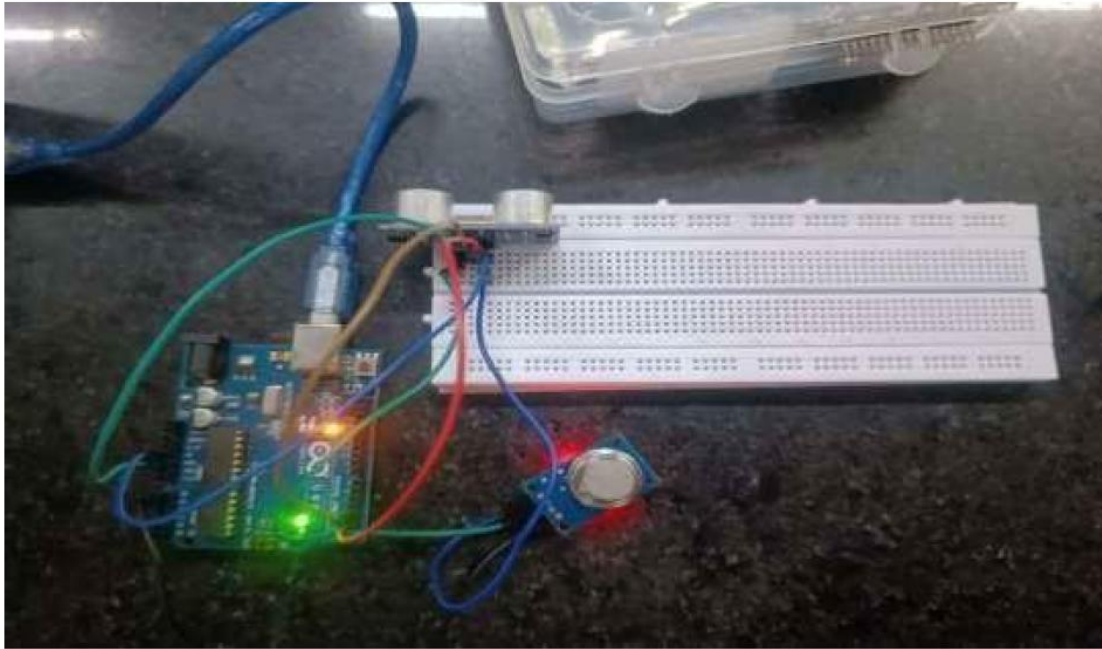
```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
// Replace with your network credentials const
char* ssid = "Your_SSID"; const char*
password = "Your_PASSWORD";
// ThingSpeak settings
const char* server = "http://api.thingspeak.com";
String apiKey = "YOUR_API_KEY";
// MQ135 connected to A0 int
mq135Pin = A0;
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password); while
  (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}
void loop() {
  // Read analog value from MQ135 int
  airQuality = analogRead(mq135Pin);
  Serial.println("Air Quality Value: " + String(airQuality));
```

```
// Send data to ThingSpeak
if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;
  String url = server + "/update?api_key=" + apiKey + "&field1=" +
  String(airQuality);
  http.begin(url);
  int httpCode = http.GET(); if
  (httpCode > 0) {
    Serial.println("Data sent to ThingSpeak successfully.");
  } else {
    Serial.println("Error sending data.");
  }
  http.end();
}
// ThingSpeak limits updates to every 15 seconds delay(15000);
}
```

5. Output:

```
PS C:\Users\manik\downloads> python exp3.py
Received: MQ135 RZero: 52.72 Corrected RZero: 52.09 Resistance: 33.35 PPM: 1167.50 Corrected PPM: 1206.88ppm
Received: MQ135 RZero: 51.30 Corrected RZero: 50.69 Resistance: 32.45 PPM: 1259.32 Corrected PPM: 1301.79ppm
Received: MQ135 RZero: 50.47 Corrected RZero: 49.87 Resistance: 31.93 PPM: 1317.15 Corrected PPM: 1361.57ppm
Received: MQ135 RZero: 49.67 Corrected RZero: 49.08 Resistance: 31.42 PPM: 1356.89 Corrected PPM: 1402.65ppm
Received: MQ135 RZero: 49.14 Corrected RZero: 48.56 Resistance: 31.25 PPM: 1418.31 Corrected PPM: 1444.73ppm
Received: MQ135 RZero: 48.88 Corrected RZero: 48.30 Resistance: 30.92 PPM: 1439.29 Corrected PPM: 1487.83ppm
Received: MQ135 RZero: 48.62 Corrected RZero: 48.05 Resistance: 30.76 PPM: 1460.51 Corrected PPM: 1509.77ppm
Received: MQ135 RZero: 48.37 Corrected RZero: 47.79 Resistance: 30.60 PPM: 1481.99 Corrected PPM: 1531.97ppm
Received: MQ135 RZero: 48.11 Corrected RZero: 47.54 Resistance: 30.60 PPM: 1503.72 Corrected PPM: 1531.97ppm
Received: MQ135 RZero: 48.11 Corrected RZero: 47.54 Resistance: 30.43 PPM: 1503.72 Corrected PPM: 1554.44ppm
Received: MQ135 RZero: 47.86 Corrected RZero: 47.54 Resistance: 30.43 PPM: 1503.72 Corrected PPM: 1554.44ppm
Received: MQ135 RZero: 47.86 Corrected RZero: 47.29 Resistance: 30.28 PPM: 1525.72 Corrected PPM: 1577.18ppm
Received: MQ135 RZero: 47.86 Corrected RZero: 47.29 Resistance: 30.28 PPM: 1525.72 Corrected PPM: 1577.18ppm
Received: MQ135 RZero: 47.86 Corrected RZero: 47.29 Resistance: 30.28 PPM: 1525.72 Corrected PPM: 1577.18ppm
Received: MQ135 RZero: 47.61 Corrected RZero: 47.05 Resistance: 30.12 PPM: 1547.98 Corrected PPM: 1577.18ppm
```

i. Simulated Cloud Air Quality Variations



ii. Hardware

6. Conclusion: The cloud temperature simulation demonstrates the dynamic nature of temperature variations within a controlled environment, constrained by predefined limits. The visualization provides insights into how small random perturbations influence the overall temperature distribution over time. This highlights the importance of boundary conditions in stabilizing such systems. Additionally, the successful integration with ThingSpeak showcases the capability to transmit real-time data to a cloud-based platform for further analysis or monitoring. This combination of simulation and IoT integration can be applied to real-world scenarios, such as environmental monitoring or predictive modeling of atmospheric conditions.

7. Learning Outcomes:

- Understanding how to interface and calibrate the MQ135 gas sensor with microcontrollers such as Arduino or ESP32.
- Collecting sensor data efficiently and reading analog values for air quality monitoring.
- Learning how to set up wireless communication protocols (Wi-Fi, MQTT) to connect with cloud platforms.