WORKSHEET 4

Student Name: Sumit Kumar **UID: 22BCS10048**

Section/Group: KRG_IOT-1-A **Semester: 6th Semester**

Branch: BE-CSE **Date of Performance: 13/02/2025**

Subject Name: Computer Graphics with Lab **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 and implement the circle generator and midpoint circle generator algorithm to draw a circle with a given centre and radius.

3. **Implementation Code:**

   (a) Circle Drawing Using a Basic Circle Generator Algorithm

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

using namespace std;

void drawCircle(int xc, int yc, int r) {
    for (int angle = 0; angle < 360; angle++) {
        int x = xc + r * cos(angle * M_PI / 180);
        int y = yc + r * sin(angle * M_PI / 180);
        putpixel(x, y, WHITE);
    }
}

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

    int xc, yc, r;
    cout << "Enter center (xc, yc) and radius: ";
    cin >> xc >> yc >> r;

    drawCircle(xc, yc, r);
   getch();
```

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

(b) Circle Drawing Using the Midpoint Circle Algorithm

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

using namespace std;

void plotPoints(int xc, int yc, int x, int 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);
}

void drawMidpointCircle(int xc, int yc, int r) {
    int x = 0, y = r, p = 1 - r;
    plotPoints(xc, yc, x, y);

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

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

    int xc, yc, r;
    cout << "Enter center (xc, yc) and radius: ";
    cin >> xc >> yc >> r;

    drawMidpointCircle(xc, yc, r);

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

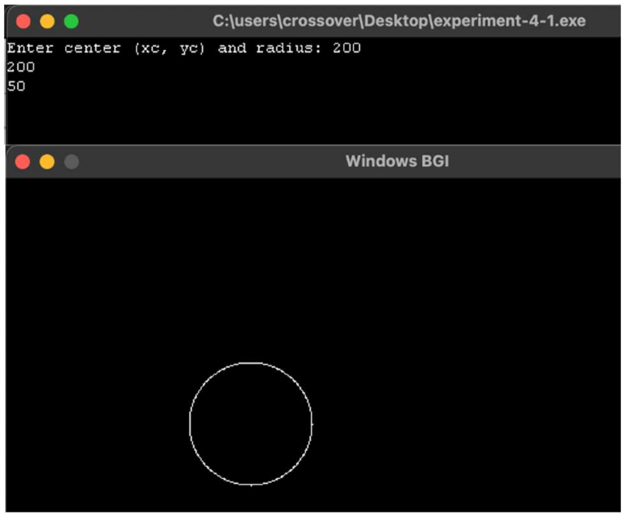
4. Explanation:

The first program implements a basic circle generator algorithm, utilizing trigonometric functions to plot points on the circle's circumference. It iterates through angles from 0 to 360 degrees, computing x and y coordinates using cosine and sine functions. This approach is simple but computationally expensive due to floating-point calculations and trigonometric function calls, making it less efficient for real-time graphics rendering.

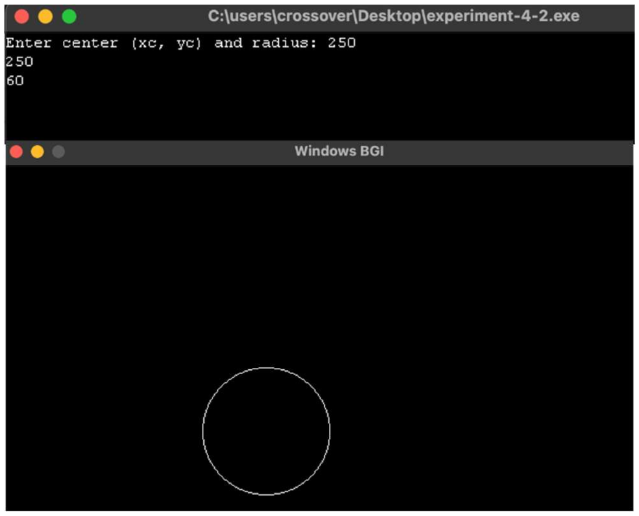
The second program uses the Midpoint Circle Algorithm, an optimized integer-based approach. It efficiently determines pixel positions using decision parameters, eliminating the need for floating-point arithmetic. The algorithm starts from the topmost point and symmetrically plots eight points in each iteration. The decision parameter helps choose the next pixel, reducing computational complexity. This method ensures smooth and accurate circle rendering with better performance.

Both programs demonstrate different techniques for drawing circles in computer graphics. The first method is straightforward but slow, whereas the midpoint algorithm is optimized for speed and accuracy. These techniques are fundamental in graphics programming, forming the basis for rendering curves and circular objects in applications like CAD software, games, and simulations.

5. Output:



```
Enter center (xc, yc) and radius: 200
200
50
```



```
Enter center (xc, yc) and radius: 250
250
60
```

(a) Circle Drawing Using a Basic Circle Generator Algorithm

(b) Circle Drawing Using the Midpoint Circle Algorithm.

6. Learning Outcomes:

- Understand trigonometric circle drawing and its computational limitations.
- Implement the Midpoint Circle Algorithm for optimized, efficient rendering.
- Learn pixel manipulation using putpixel() in BGI graphics.