Date: 07/10/2025
Experiment No: 05
Experiment Name: Draw circle using Direct circle drawing algorithm and Mid point circle drawing Algorithm.

Introduction

This report explains and compares two basic computer graphics methods for drawing circles: the Direct Circle Drawing Algorithm and the Midpoint Circle Drawing Algorithm. The goal is to implement both, see how they generate circle pixels, and compare their efficiency. The Midpoint method is faster because it avoids complex floating-point calculations and uses symmetry to draw the circle more easily and efficiently.

Description:

Direct Circle Drawing Algorithm:

This method uses the basic circle equation $x^2 + y^2 = r^2$ to find points on the circle.

Steps:

1. Start with $x = 0$ and increase x until $x = r$.
2. For each x, calculate $y = \sqrt{(r^2 - x^2)}$.
3. Round y to the nearest integer and plot points at $(\pm x, \pm y)$ and $(\pm y, \pm x)$ for symmetry.

Drawback:
It's slow because it repeatedly uses square roots and floating-point calculations, making it inefficient for real-time graphics.

Midpoint Circle Drawing Algorithm:

This algorithm is an improved, faster method for drawing circles using only integers — no square roots or floating-point math. It decides the next pixel based on a simple decision value that checks which point is closer to the circle path.

Steps:

1. Start at (0, r).
2. Work only in one octant (0°–45°) and use symmetry to draw the rest.
3. Use a decision parameter (P) to pick the next point:
   If $P < 0$, choose $(x+1, y)$.
   If $P \geq 0$, choose $(x+1, y-1)$.
4. Update P using only addition and subtraction.

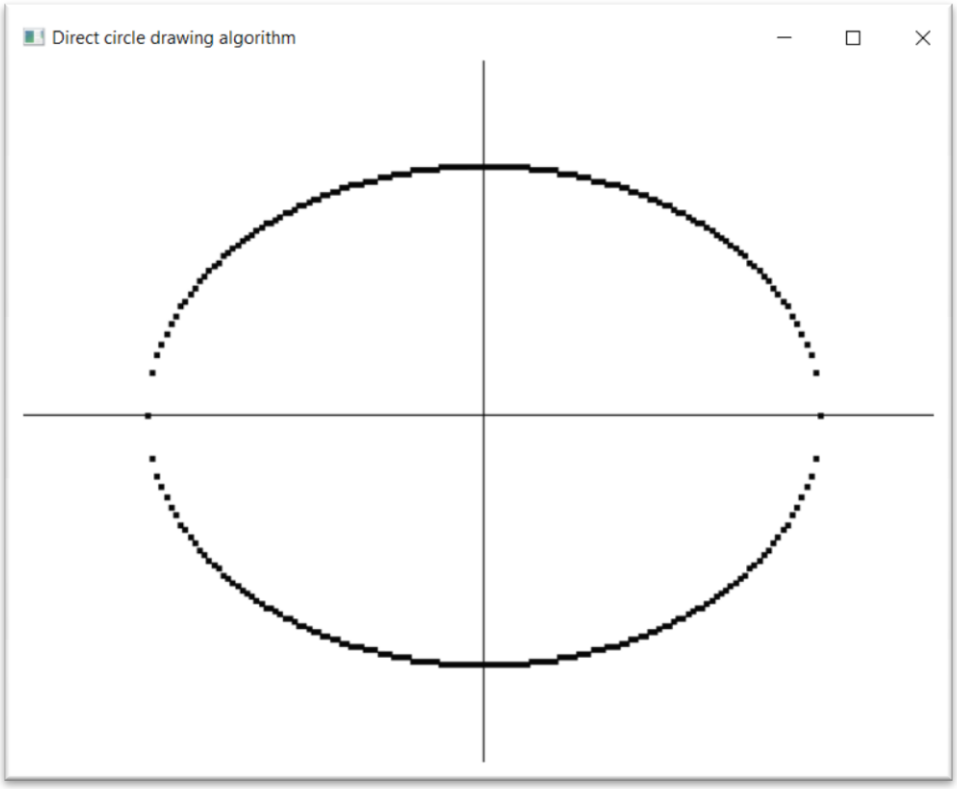
Advantage:
It's much faster and more efficient because it avoids complex math and uses symmetry to draw the whole circle easily.

Code:

```
// Direct circle drawing algorithm

#include <GL/glut.h>
#include <stdlib.h>
#include <iostream>
#include <math.h>
using namespace std;

float x, y, r;

void init(void){
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-100, 100, -100, 100);
}

void put_pixel(float x, float y){
   glColor3f(0.0f, 0.0f, 0.0f);
   glPointSize(4.0);
   glBegin(GL_POINTS);
   glVertex2f(x, y);
   glVertex2f(x, -y);
   glVertex2f(-x, y);
   glVertex2f(-x, -y);

   glEnd();

   glFlush();
}

void direct_circle(float x, float y, float r){
   y = r;

   put_pixel(x, y);

   while (x <= y){
      int y_update = round(sqrt(r * r - x * x));
      x = x + 1;
      put_pixel(x, y_update);
   }
}
```

```
void display(){

   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0f, 0.0f, 0.0f);

   // Draw axes
   glBegin(GL_LINES);
   glColor3f(0, 0, 0);
   glVertex2f(100, 0);
   glVertex2f(-100, 0);
   glVertex2f(0, 100);
   glVertex2f(0, -100);
   glEnd();

   direct_circle(x,  y,  r);
}

int main(int argc, char *argv[]){
   cout << "Enter Redius: ";
   cin >> r;

   cout << "Enter center point of X axis:";
   cin >> x;

   cout << "Enter center point of Y axis:";
   cin >> y;

   glutInit(&argc, argv);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(10, 10);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("GLUT Shapes");

   init();
   glutDisplayFunc(display);

   glutMainLoop();

   return 0;
}
```

Output:



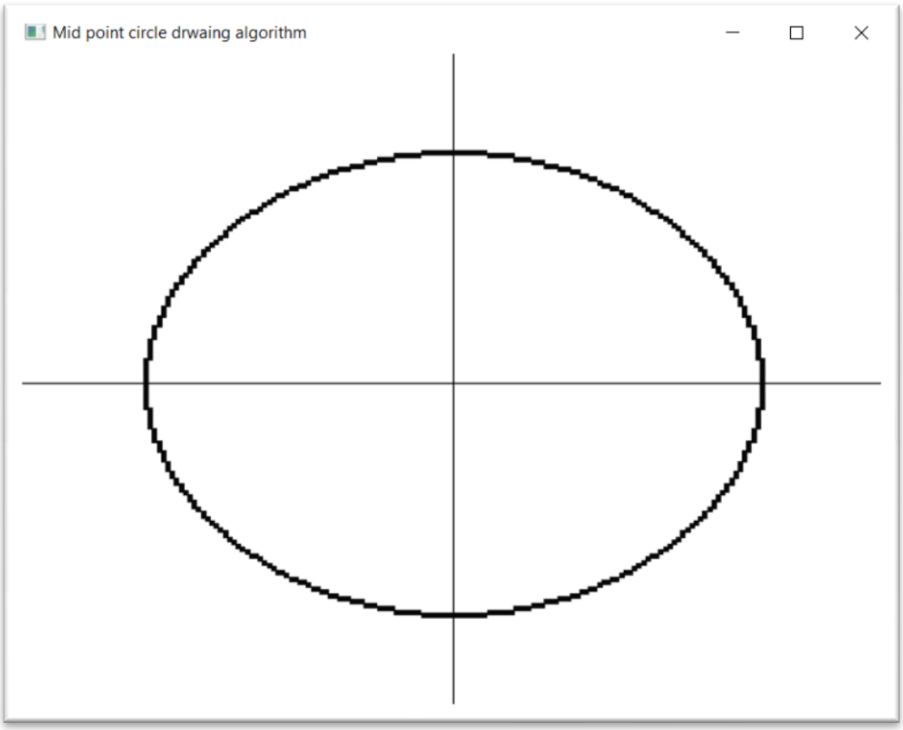
Code 2:

```
// mid point circle drawing algorithm

#include <GL/glut.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

float x, y, r;

void init(void){
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-100, 100, -100, 100);
```

```
}

void put_pixel(float x, float y){
   glColor3f(0.0f, 0.0f, 0.0f);
   glPointSize(4.0);
   glBegin(GL_POINTS);
   glVertex2f(x, y);
   glVertex2f(y, x);
   glVertex2f(y, -x);
   glVertex2f(x, -y);
   glVertex2f(-x, y);
   glVertex2f(-y, x);
   glVertex2f(-y, -x);
   glVertex2f(-x, -y);

   glEnd();

   glFlush();
}

void circle(float x, float y, float r){
   float p = 1 - r;
   y = r;

   while (x  <=  y){
      put_pixel(x,  y);

      if (p  <  0){
         x =  x + 1;
         y = y;
         p += 2 * x + 3;
      }
      else{
         x = x + 1;
         y = y - 1;
         p += 2 * (x - y) + 5;
      }
   }
}

void display(){
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0f, 0.0f, 0.0f);

   // Draw axes
   glBegin(GL_LINES);
```

```
    glColor3f(0, 0, 0);
    glVertex2f(100, 0);
    glVertex2f(-100, 0);
    glVertex2f(0, 100);
    glVertex2f(0, -100);
    glEnd();

    circle(x, y, r);
}

int main(int argc, char *argv[]){
    cout << "Enter Redius: ";
    cin >> r;

    cout << "Enter center point of X axis:";
    cin >> x;

    cout << "Enter center point of Y axis:";
    cin >> y;

    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(10, 10);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("Mid point cicle drawing algorithm");

    init();
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;
}
```

Output:



Conclusion:

Both the Direct and Midpoint Circle Drawing Algorithms can draw circles, but the Midpoint method is faster and more efficient. The Direct algorithm uses square roots and floating-point calculations, making it slower. In contrast, the Midpoint algorithm uses only integers and symmetry, giving smoother and quicker results making it better for real-time computer graphics.