CS 352

Computer Graphics & Visualization

Assignment - 4

Name – Anjani Kumar

Roll No - 210001004

Question – 1: Bresenham's Algorithm

Code:

```
#include <GL/glut.h>
#include<bits/stdc++.h>
using namespace std;

// Declare global variables to store circle parameters
float x_cen, y_cen, radius;
vector<float> x_coor, y_coor;

// Display callback function
void displayCB() {
// Initialize variables for the circle drawing algorithm
float x_comp = 0, y_comp = radius;
int decision_parameter = 3 - 2 * radius;

// Add the initial set of points to the vectors
x_coor.push_back(0);
y_coor.push_back(radius);

x_coor.push_back(0);
y_coor.push_back(-radius);

x_coor.push_back(-radius);
y_coor.push_back(0);

x_coor.push_back(radius);
y_coor.push_back(0);
```

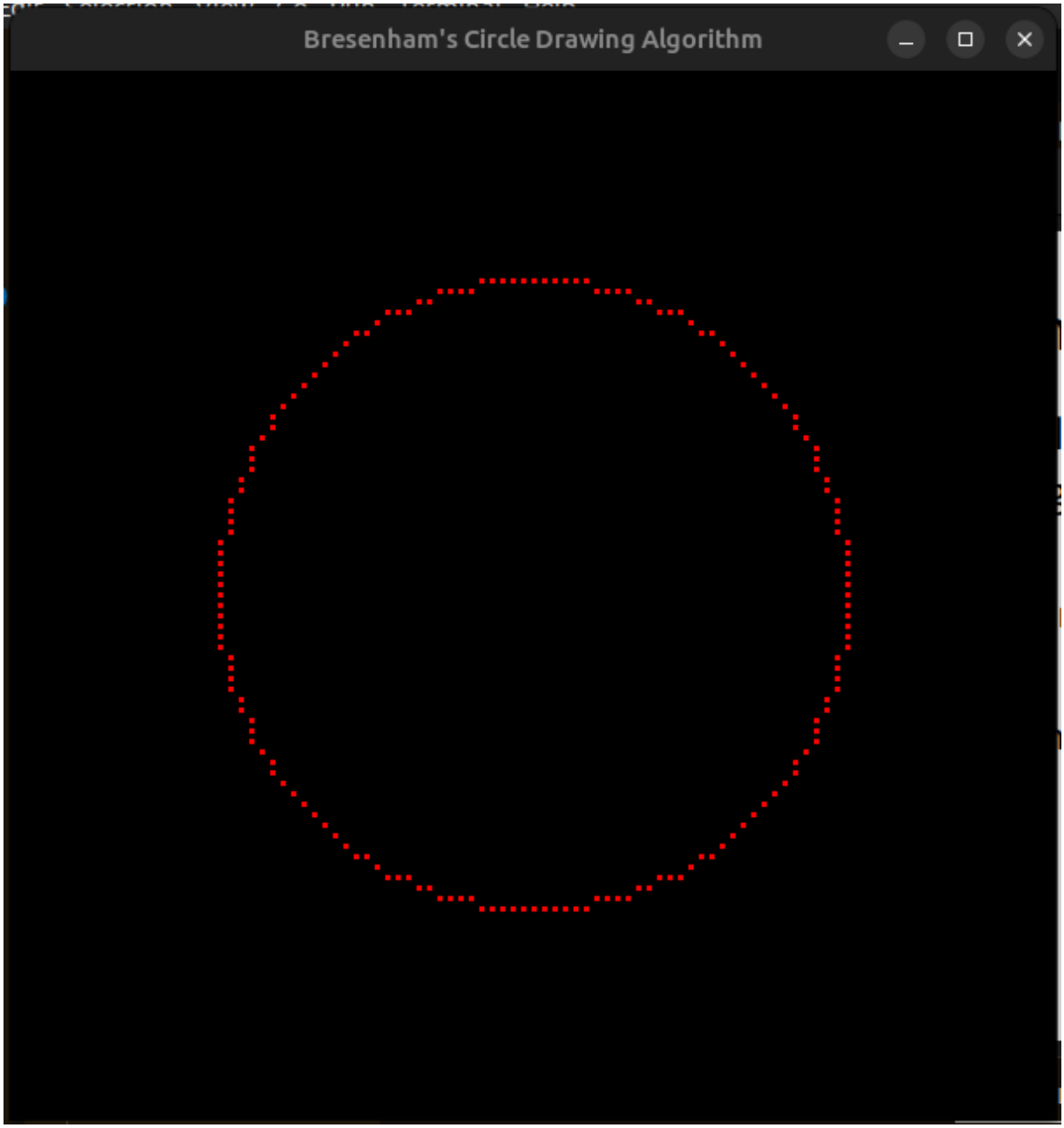
```
// Loop to generate points for the circle
while (x_comp != y_comp) {
if (decision_parameter < 0) {
decision_parameter += 4 * x_comp + 6;
x_comp++;
} else {
decision_parameter += 4 * (x_comp - y_comp) + 10;
y_comp--;
x_comp++;
}
// Add points in all eight octants of the circle
x_coor.push_back(x_comp);
y_coor.push_back(y_comp);

x_coor.push_back(y_comp);
y_coor.push_back(x_comp);

x_coor.push_back(-y_comp);
y_coor.push_back(x_comp);

x_coor.push_back(-x_comp);
y_coor.push_back(y_comp);

x_coor.push_back(-x_comp);
y_coor.push_back(-y_comp);

x_coor.push_back(-y_comp);
y_coor.push_back(-x_comp);

x_coor.push_back(y_comp);
```

```
y_coor.push_back(-x_comp);

x_coor.push_back(x_comp);
y_coor.push_back(-y_comp);
}

// Plot the generated points
for (int i = 0; i < x_coor.size(); i++) {
glBegin(GL_POINTS);
glVertex2f(x_cen + x_coor[i], y_cen + y_coor[i]);
glEnd();
}
glFlush(); // Ensure all OpenGL commands are executed
}

// Main function
int main(int argc, char** argv) {
// Prompt the user to enter circle parameters
cout << "Enter the coordinates of the Center: \n";
cout << "Enter the x-coordinate: ";
cin >> x_cen;
cout << "Enter the y-coordinate: ";
cin >> y_cen;
cout << "Enter the radius of the circle: ";
cin >> radius;

// Initialize GLUT
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(600, 600);
glutCreateWindow("Bresenham's Circle Drawing Algorithm");
```

```
// Set up OpenGL properties
glClearColor(1, 1, 1, 0.0);
glColor3f(1, 0, 0);
glPointSize(3.0);
gluOrtho2D(0, 100, 0, 100);

// Register the display callback function
glutDisplayFunc(displayCB);

// Enter the GLUT event processing loop
glutMainLoop();

return 0;
}
```

Screenshots of the Output:

```
Enter the coordinates of the Center:
Enter the x-coordinate: 50
Enter the y-coordinate: 50
Enter the radius of the circle: 30
```


Bresenham's Circle Drawing Algorithm

Question 2: Mid Point Algorithm for Circle Drawing

Code:

```
#include <GL/glut.h>
#include<bits/stdc++.h>
using namespace std;

// Declare variables to store circle parameters
float x_cen, y_cen, radius;
vector<float> x_coor, y_coor;

// Display callback function to draw the circle
void displayCB() {
// Initialize starting components and decision parameter
float x_comp = 0, y_comp = radius;
int decision_parameter = 1 - radius;

// Add initial points to vectors
x_coor.push_back(0);
y_coor.push_back(radius);

x_coor.push_back(0);
y_coor.push_back(-radius);

x_coor.push_back(-radius);
y_coor.push_back(0);

x_coor.push_back(radius);
y_coor.push_back(0);
// Mid-point circle algorithm
```

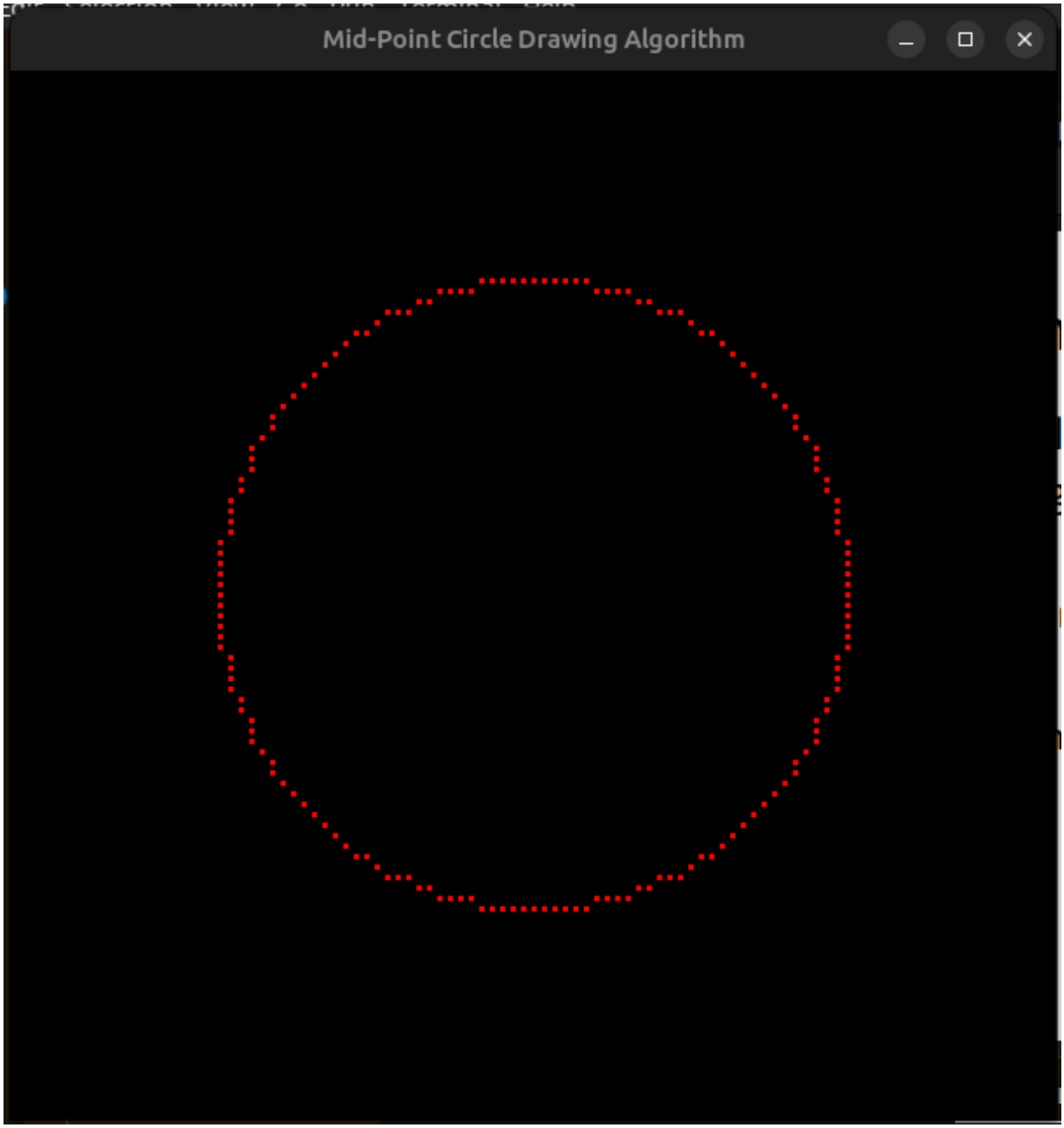
```
while (x_comp != y_comp) {
if (decision_parameter < 0) {
decision_parameter += 2 * x_comp + 3;
x_comp++;
} else {
decision_parameter += 2 * (x_comp - y_comp) + 5;
y_comp--;
x_comp++;
}
// Add points based on symmetry
x_coor.push_back(x_comp);
y_coor.push_back(y_comp);

x_coor.push_back(y_comp);
y_coor.push_back(x_comp);

x_coor.push_back(-y_comp);
y_coor.push_back(x_comp);
x_coor.push_back(-x_comp);
y_coor.push_back(y_comp);

x_coor.push_back(-x_comp);
y_coor.push_back(-y_comp);

x_coor.push_back(-y_comp);
y_coor.push_back(-x_comp);

x_coor.push_back(y_comp);
y_coor.push_back(-x_comp);

x_coor.push_back(x_comp);
```

```
y_coor.push_back(-y_comp);
}

// Plot the calculated points
for (int i = 0; i < x_coor.size(); i++) {
glBegin(GL_POINTS);
glVertex2f(x_cen + x_coor[i], y_cen + y_coor[i]);
glEnd();
}
glFlush();
}

int main(int argc, char** argv) {
// Prompt user to enter circle parameters
cout << "Enter the coordinates of the Center: \n";
cout << "Enter the x-coordinate: ";
cin >> x_cen;
cout << "Enter the y-coordinate: ";
cin >> y_cen;

cout << "Enter the radius of the circle: ";
cin >> radius;

// Initialize OpenGL and create window
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(600, 600);
glutCreateWindow("Mid-Point Circle Drawing Algorithm");
// Set OpenGL properties
glClearColor(1, 1, 1, 0.0);
glColor3f(1, 0, 0);
```

```
glPointSize(3.0);
gluOrtho2D(0, 100, 0, 100);

// Register the display callback function
glutDisplayFunc(displayCB);

// Enter the GLUT event processing loop
glutMainLoop();

return 0;
}
```

ScreenShot of the Ouput:

```
Enter the coordinates of the Center:
Enter the x-coordinate: 50
Enter the y-coordinate: 50
Enter the radius of the circle: 30
```


Mid-Point Circle Drawing Algorithm

Question 3: DDA Algorithm for Circle Drawing

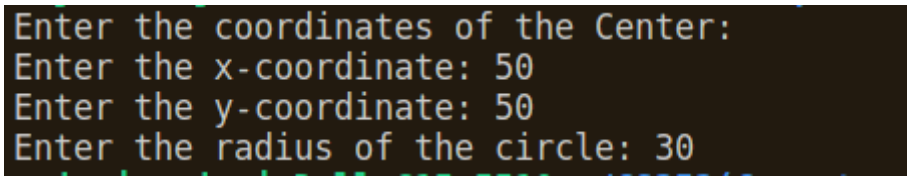
Code:

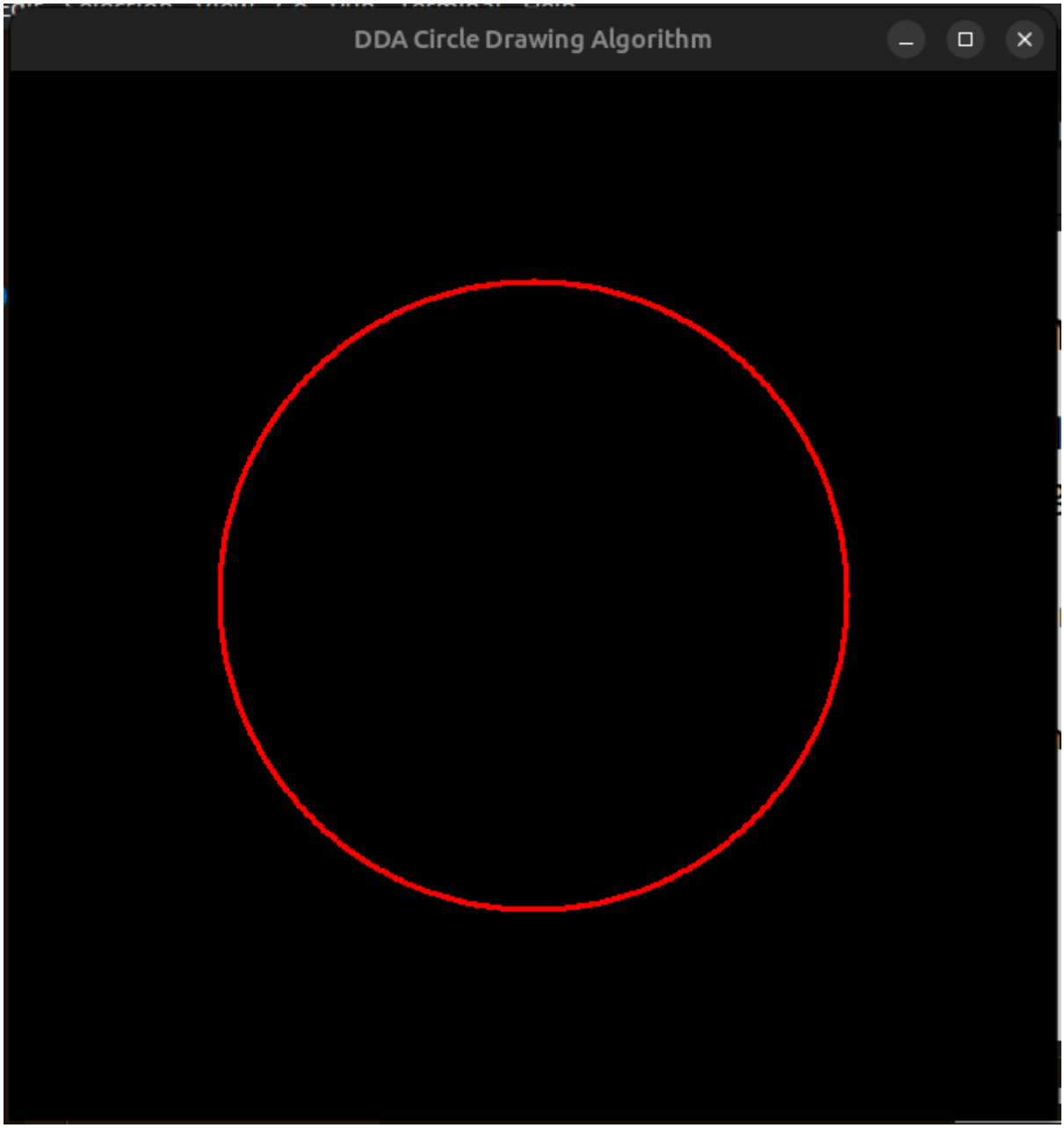
```
#include <GL/glut.h>
#include<bits/stdc++.h>
using namespace std;

// Declare variables to store circle parameters
float x_cen, y_cen, radius;
vector<float> x_coor, y_coor;

// Display callback function to draw the circle using DDA algorithm
void displayCB() {
// Iterate through angles to generate circle points
for (float angle = 0.0; angle <= 0.80; angle = angle + 0.01) {
// Calculate components using trigonometric functions
float x_comp = radius * cos(angle);
float y_comp = radius * sin(angle);

// Add points based on symmetry
x_coor.push_back(x_comp);
y_coor.push_back(y_comp);

x_coor.push_back(y_comp);
y_coor.push_back(x_comp);

x_coor.push_back(-y_comp);
y_coor.push_back(x_comp);
x_coor.push_back(-x_comp);
y_coor.push_back(y_comp);

x_coor.push_back(-x_comp);
y_coor.push_back(-y_comp);
```

```
x_coor.push_back(-y_comp);
y_coor.push_back(-x_comp);

x_coor.push_back(y_comp);
y_coor.push_back(-x_comp);

x_coor.push_back(x_comp);
y_coor.push_back(-y_comp);
}

// Plot the calculated points
for (int i = 0; i < x_coor.size(); i++) {
glBegin(GL_POINTS);
glVertex2f(x_cen + x_coor[i], y_cen + y_coor[i]);
glEnd();
}
// Flush the drawing commands
glFlush();
}

// Main function
int main(int argc, char** argv) {
// Prompt user to enter circle parameters
cout << "Enter the coordinates of the Center: \n";
cout << "Enter the x-coordinate: ";
cin >> x_cen;
cout << "Enter the y-coordinate: ";
cin >> y_cen;

cout << "Enter the radius of the circle: ";
cin >> radius;
```

```
// Initialize OpenGL and create window
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(600, 600);
glutCreateWindow("DDA Circle Drawing Algorithm");
// Set OpenGL properties
glClearColor(1, 1, 1, 0.0);
glColor3f(1, 0, 0);
glPointSize(3.0);
gluOrtho2D(0, 100, 0, 100);

// Register the display callback function
glutDisplayFunc(displayCB);

// Enter the GLUT event processing loop
glutMainLoop();

return 0;
}
```

Screenshot of the Output:




DDA Circle Drawing Algorithm