**Batch: D - 1 Roll No.: 16010122096**

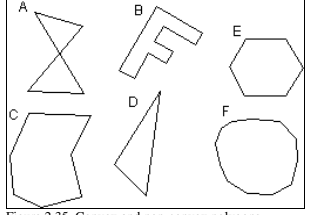
**Experiment No. 04**

|  |
| --- |
| **TITLE:** Draw the Following polygon/shape/curve.  a. Bresenham Circle Drawing Algorithm.  b. Other shapes |

**AIM:**

a. Generate the Circle using Bresenham Circle Drawing Algorithm

b. Draw polygon shown in following figure



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

CO1: Understand the basic concepts of computer graphics and OpenGL

CO4: Understand the computer Input & interaction, Curves and Computer Animation

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

[**https://www.gatevidyalay.com/bresenham-circle-drawing-algorithm/**](https://www.gatevidyalay.com/bresenham-circle-drawing-algorithm/)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm:**

1. **Bresenham’s Circle Drawing Algorithm:**

**Given-**

* Centre point of Circle = (X0, Y0)
* Radius of Circle = R

The points generation using Bresenham Circle Drawing Algorithm involves the following steps-

**Step-01:**

Assign the starting point coordinates (X0, Y0) as-

* X0 = 0
* Y0 = R

**Step-02:**

Calculate the value of initial decision parameter P0 as-

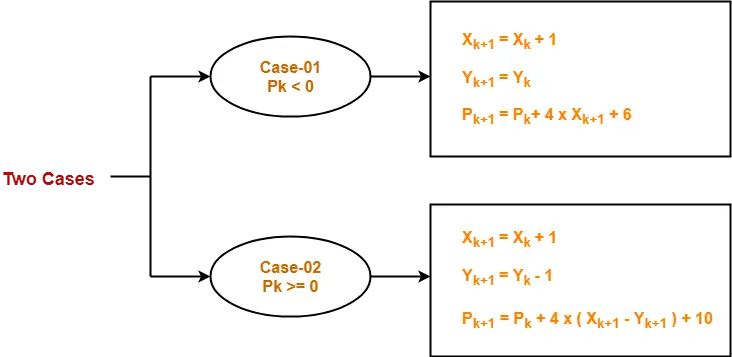
P0 = 3 – 2 x R

**Step-03:**

Suppose the current point is (Xk, Yk) and the next point is (Xk+1, Yk+1).

Find the next point of the first octant depending on the value of decision parameter Pk.

Follow the below two cases-

****

**Step-04:**

If the given centre point (X0, Y0) is not (0, 0), then do the following and plot the point-

* Xplot = Xc + X0
* Yplot = Yc + Y0

Here, (Xc, Yc) denotes the current value of X and Y coordinates.

**Step-05:**

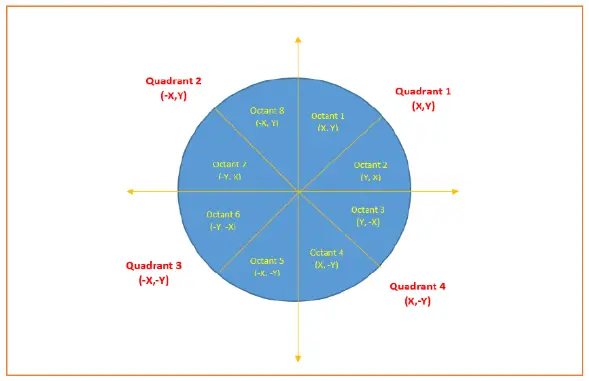
Keep repeating Step-03 and Step-04 until Xplot => Yplot.

**Step-06:**

Step-05 generates all the points for one octant.

To find the points for other seven octants, follow the eight symmetry property of circle.

This is depicted by the following figure-

****

**Implementation details:**

**a.**

**#include <bits/stdc++.h>**

**#include <GL/gl.h>**

**#include <GL/glut.h>**

**using namespace std;**

**int x, y, r;**

**void bresenham\_circle()**

**{**

**int xi = 0, yi = r;**

**int p = 3 - 2 \* r;**

**glBegin(GL\_POINTS);**

**while (xi <= yi)**

**{**

**// Plot the points in all 8 octants**

**glVertex2i(xi + x, yi + y);**

**glVertex2i(-xi + x, yi + y);**

**glVertex2i(xi + x, -yi + y);**

**glVertex2i(-xi + x, -yi + y);**

**glVertex2i(yi + x, xi + y);**

**glVertex2i(-yi + x, xi + y);**

**glVertex2i(yi + x, -xi + y);**

**glVertex2i(-yi + x, -xi + y);**

**if (p < 0)**

**{**

**p += 4 \* xi + 6;**

**}**

**else**

**{**

**yi--;**

**p += 4 \* (xi - yi) + 10;**

**}**

**xi++;**

**}**

**glEnd();**

**}**

**void display(void)**

**{**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**glBegin(GL\_LINES);**

**glVertex2i(-300, 0);**

**glVertex2i(300, 0);**

**glVertex2i(0, -280);**

**glVertex2i(0, 280);**

**glEnd();**

**glColor3f(0.0, 1.0, 0.0);**

**bresenham\_circle();**

**glFlush();**

**}**

**void myInit(void)**

**{**

**glClearColor(0.0, 0.0, 0.0, 0.0);**

**glMatrixMode(GL\_PROJECTION);**

**gluOrtho2D(-300, 300, -280, 280);**

**}**

**int main(int argc, char\*\* argv)**

**{**

**cout << "X Coordinate: ";**

**cin >> x;**

**cout << "Y Coordinate: ";**

**cin >> y;**

**cout << "Radius: ";**

**cin >> r;**

**glutInit(&argc, argv);**

**glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);**

**glutInitWindowSize(640, 480);**

**glutInitWindowPosition(100, 150);**

**glutCreateWindow("Bresenham's Circle Drawing");**

**glutDisplayFunc(display);**

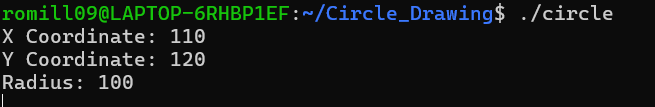
**myInit();**

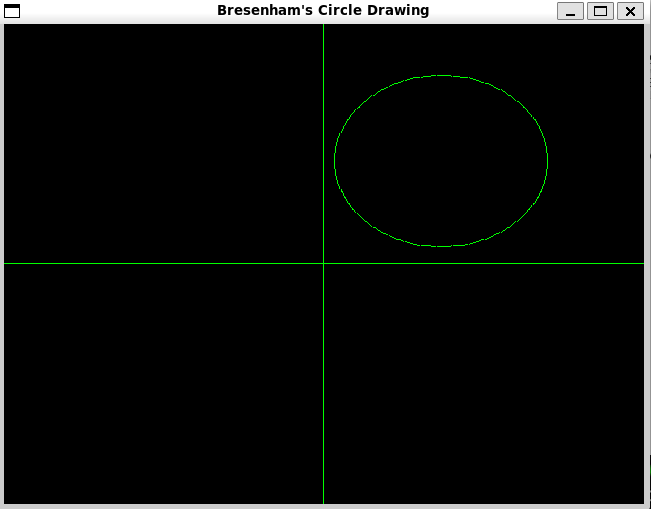
**glutMainLoop();**

**return 0;**

**}**

**Output:**



****

**b.**

**#include <bits/stdc++.h>**

**#include <GL/glut.h>**

**void A() {**

**glBegin(GL\_LINE\_LOOP);**

**glVertex2i(30, 90);**

**glVertex2i(70, 80);**

**glVertex2i(22, 30);**

**glVertex2i(75, 32);**

**glVertex2i(30, 90);**

**glEnd();**

**}**

**void B() {**

**glBegin(GL\_LINE\_LOOP);**

**glVertex2i(120, 120);**

**glVertex2i(170, 110);**

**glVertex2i(165, 100);**

**glVertex2i(135, 108);**

**glVertex2i(128, 95);**

**glVertex2i(145, 90);**

**glVertex2i(140, 80);**

**glVertex2i(120, 85);**

**glVertex2i(110, 45);**

**glVertex2i(100, 55);**

**glVertex2i(120, 120);**

**glEnd();**

**}**

**void C() {**

**glBegin(GL\_LINE\_LOOP);**

**glVertex2i(190, 80);**

**glVertex2i(250, 78);**

**glVertex2i(240, 50);**

**glVertex2i(252, 30);**

**glVertex2i(215, 20);**

**glVertex2i(180, 35);**

**glVertex2i(180, 60);**

**glVertex2i(190, 80);**

**glEnd();**

**}**

**void D() {**

**glBegin(GL\_LINE\_LOOP);**

**glVertex2i(260, 30);**

**glVertex2i(310, 70);**

**glVertex2i(280, 110);**

**glEnd();**

**}**

**void Hexagon(int xc, int yc, int r) {**

**glBegin(GL\_LINE\_LOOP);**

**for (int i = 0; i < 6; ++i) {**

**glVertex2i(xc + r \* cos(i \* M\_PI / 3), yc + r \* sin(i \* M\_PI / 3));**

**}**

**glEnd();**

**}**

**void Decagon(int xc, int yc, int r) {**

**glBegin(GL\_LINE\_LOOP);**

**for (int i = 0; i < 10; ++i) {**

**glVertex2i(xc + r \* cos(i \* M\_PI / 5), yc + r \* sin(i \* M\_PI / 5));**

**}**

**glEnd();**

**}**

**void display(void) {**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**glColor3f(0.0, 1.0, 0.0);**

**A();**

**B();**

**Hexagon(360, 70, 40);**

**C();**

**D();**

**Decagon(460, 70, 40);**

**glFlush();**

**}**

**void init(void) {**

**glClearColor(0.0, 0.0, 0.0, 0.0);**

**glMatrixMode(GL\_PROJECTION);**

**gluOrtho2D(0, 500, 0, 150);**

**}**

**int main(int argc, char\*\* argv) {**

**glutInit(&argc, argv);**

**glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);**

**glutInitWindowSize(500, 150);**

**glutInitWindowPosition(100, 100);**

**glutCreateWindow("Shapes");**

**init();**

**glutDisplayFunc(display);**

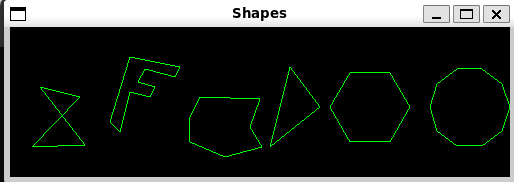
**glutMainLoop();**

**return 0;**

**}**

**Output:**

****

****

**Conclusion and discussion:**

The Bresenham Circle Drawing Algorithm efficiently generates a circle by minimizing calculations and errors, ensuring precise rendering. Additionally, implementing various shapes demonstrates the versatility of graphical algorithms in producing complex figures. These techniques are crucial for accurate and performance-efficient graphics in applications.

**Date: 19 / 08 / 2024**

**Signature of faculty in-charge**

**Post lab**

**Visit and explore and paste your screenshot**

[**https://cse18-iiith.vlabs.ac.in/exp/rasterization-line/**](https://cse18-iiith.vlabs.ac.in/exp/rasterization-line/)

[**https://cse18-iiith.vlabs.ac.in/exp/rasterization-polygon/**](https://cse18-iiith.vlabs.ac.in/exp/rasterization-polygon/)