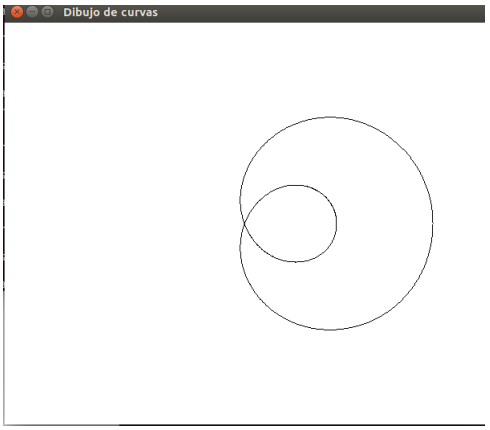
Práctica de Laboratorio 3

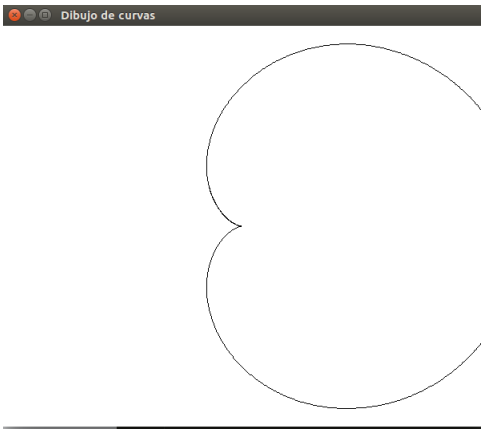
Christofer Fabián Chávez Carazas
Universidad Nacional de San Agustín de Arequipa
Escuela Profesional de Ciencia de la Computación
Computación Gráfica

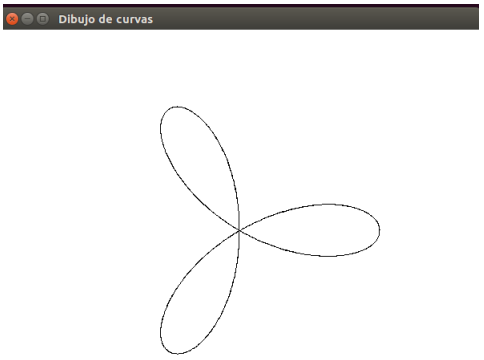
2 de noviembre de 2017

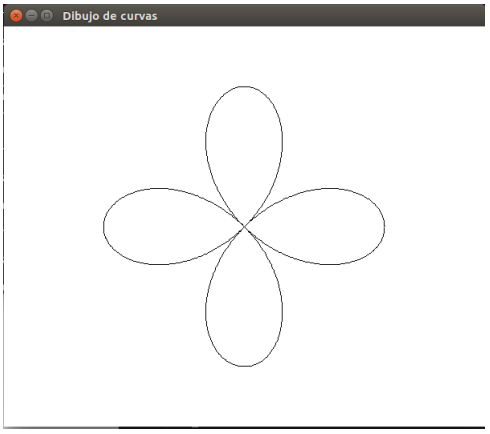
1. **Compile y ejecute el siguiente código**

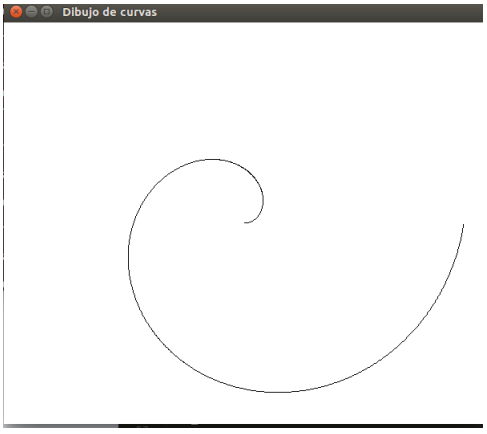
   Resultados:











2. **Implique que función cumple cada línea de código:**

   Lo primero que hace el programa es pedirnos qué tipo de curva va a dibujar. Esto lo hace en la siguiente función:

```
void displayFcn(void){
    GLint curveNum;
    glClear(GL_COLOR_BUFFER_BIT);
    cout<<"\nIngrese el entero correspondiente a\n";
    cout<<"una de las siguientes cuevas.\n";
```

```
    cout<<"Presione otra tecla para salir.\n";
    cout<<"\n1-caracola, 2-cardioide, 3-trebol(3 hojas), 4-trebol(4 hojas), 5-espiral: ";
    cin>>curveNum;

    if(curveNum == 1 or curveNum == 2 or curveNum == 3 or curveNum == 4 or curveNum == 5){
        drawCurve(curveNum);
    }
    else exit(0);
    glFlush();
}
```

La función principal del programa es *drawCurve*.

```
    const GLint a = 175, b = 60;

    GLfloat r, theta, dtheta = 1.0 / float(a);
    GLint x0 = 300, y0 = 250;
```

Las variables a y b son el ancho y el largo que va a tener la cueva. La variable *dtheta* es el desplazamiento de *theta*. Las variables *x0* y *y0* son las coordenadas del punto central de la curva.

```
    curvePt[0].x = x0;
    curvePt[0].y = y0;

    switch(curveNum){
        case limacon:
            curvePt[0].x += a + b; break;
        case cardioid:
            curvePt[0].x += a + a; break;
        case threeLeaf:
            curvePt[0].x += a; break;
        case fourLeaf:
            curvePt[0].x += a; break;
        case spiral:
            break;
        default:
            break;
    }
```

En esta parte del código se inicializa el punto donde comenzará a dibujarse la curva, y esto se hace dependiendo de la curva que queramos dibujar.

```
    theta = dtheta;
    while(theta < twoPI){
        switch(curveNum){
            case limacon:
                r = a * cos(theta) + b; break;
            case cardioid:
                r = a * (1 + cos(theta)); break;
            case threeLeaf:
                r = a * cos(3 * theta); break;
            case fourLeaf:
                r = a * cos(2 * theta); break;
            case spiral:
                r = (a / 4.0) * theta; break;
            default:
                break;
        }

        curvePt[1].x = x0 + r * cos(theta);
        curvePt[1].y = y0 + r * sin(theta);
        lineSegment(curvePt[0], curvePt[1]);

        curvePt[0].x = curvePt[1].x;
        curvePt[0].y = curvePt[1].y;
        theta += dtheta;
    }
```

En esta parte del código se halla el radio, según la curva que queramos dibujar, para poder hallar el siguiente punto de la curva. Luego, se dibuja una linea desde el punto actual hasta el punto recién hallado. Después, el punto actual se vuelve el punto recién

hallado y se suma a *theta* el desplazamiento para la siguiente iteración.

3. **Implemente el algoritmo DDA para el trazado de líneas**

```
#include <GL/glut.h>
#include <iostream>
#include <cmath>


GLsizei winWidth = 600, winHeight = 500;

struct Point{
    GLint x;
    GLint y;
};

void init(void){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 600.0, 0.0, 500.0);
}

void drawPoint(Point p){
    glBegin(GL_POINTS);
        glVertex2i(p.x, p.y);
    glEnd();
}

void drawLine(Point a, Point b){
    glBegin(GL_LINES);
        glVertex2i(a.x, a.y);
        glVertex2i(b.x, b.y);
    glEnd();
}

void DDA(Point ini, Point final){
    GLint deltaX = final.x - ini.x;
    GLint deltaY = final.y - ini.y;
    GLint pasos = -1;
    GLint incrementoX = -1;
    GLint incrementoY = -1;
    Point actual;
    if(abs(deltaX) > abs(deltaY)) pasos = abs(deltaX);
    else pasos = abs(deltaY);
    incrementoX = deltaX / pasos;
    incrementoY = deltaY / pasos;
    actual.x = ini.x;
    actual.y = ini.y;
    drawPoint(actual);
    for(int i = 0; i < pasos; i++){
        actual.x += incrementoX;
        actual.y += incrementoY;
        drawPoint(actual);
    }
}

void display(){
    Point a;
    Point b;
    a.x = 50;
    a.y = 50;
    b.x = 400;
    b.y = 400;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    DDA(a,b);
    glFlush();

}


int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Dibujo de curvas");

    init();
    glutDisplayFunc(display);


    glutMainLoop();

}
```

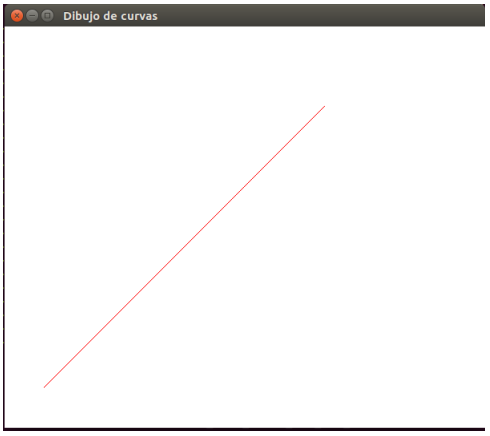


Figura 1: Resultados

4. Implemente el algoritmo Bresenham para el trazado de líneas

```
#include <GL/glut.h>
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;

GLsizei winWidth = 600, winHeight = 500;

struct Point{
    GLint x;
    GLint y;
};

void init(void){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 600.0, 0.0, 500.0);
}

void drawPoint(Point p){
    glBegin(GL_POINTS);
        glVertex2i(p.x, p.y);
    glEnd();
}

void drawLine(Point a, Point b){
    glBegin(GL_LINES);
        glVertex2i(a.x, a.y);
        glVertex2i(b.x, b.y);
    glEnd();
}

void bresenham(Point ini, Point fin){
    GLint dX = fin.x - ini.x;
    GLint dY = fin.y - ini.y;
    GLint IncYi;
    GLint IncXi;
    GLint IncXr;
    GLint IncYr;
    GLint avR;
    GLint av;
    GLint avI;
    if(dY >= 0) IncYi = 1;
    else{
        dY = -dY;
        IncYi = -1;
    }
    if(dX >= 0) IncXi = 1;
    else{
        dX = -dX;
        IncXi = -1;
    }
    if(dX >= dY){
        IncYr = 0;
        IncXr = IncXi;
    }
    else{
        IncXr = 0;
        IncYr = IncYi;
```

```
        swap(dX,dY);
    }
    Point actual;
    actual.x = ini.x;
    actual.y = ini.y;
    avR = 2 * dY;
    av = avR - dX;
    avI = av - dX;
    while(actual.x != fin.x){
        drawPoint(actual);
        if(av >= 0){
            actual.x += IncXi;
            actual.y += IncYi;
            av += avI;
        }
        else{
            actual.x += IncXr;
            actual.y += IncYr;
            av += avR;
        }
    }
}


void display(){
    Point a;
    Point b;
    a.x = 50;
    a.y = 400;
    b.x = 400;
    b.y = 50;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    bresenham(a,b);
    glFlush();

}


int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Dibujo de curvas");

    init();
    glutDisplayFunc(display);


    glutMainLoop();

}
```

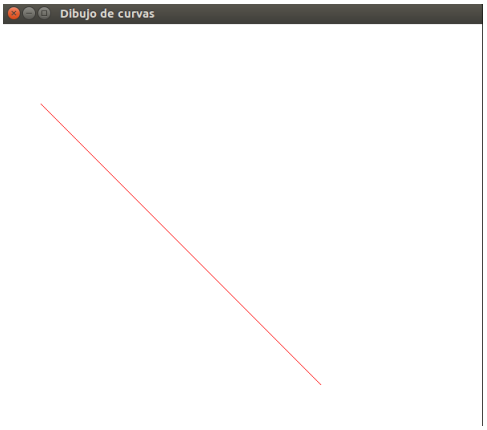
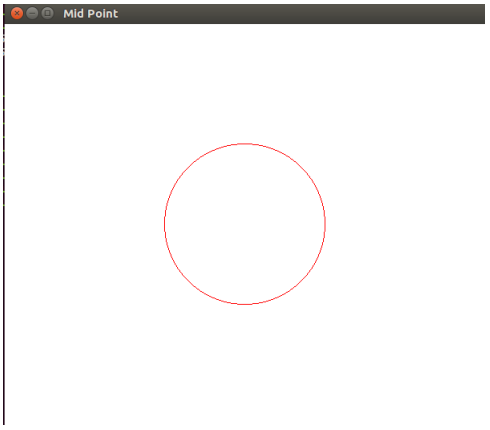


Figura 2: Resultados

5. **Implemente el algoritmo del punto medio para la generación de círculos**

```
#include <GL/glut.h>
#include <iostream>
```

```
#include <cmath>
#include <algorithm>

using namespace std;

GLsizei winWidth = 600, winHeight = 500;

struct Point{
    GLint x;
    GLint y;
};

void init(void){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 600.0, 0.0, 500.0);
}

void setPixel(GLint x, GLint y){
    glBegin(GL_POINTS);
        glVertex2i(x,y);
    glEnd();
}


void circlePlotPoints(Point circCtr, Point circPt){
    setPixel(circCtr.x + circPt.x,circCtr.y + circPt.y);
    setPixel(circCtr.x - circPt.x,circCtr.y + circPt.y);
    setPixel(circCtr.x + circPt.x,circCtr.y - circPt.y);
    setPixel(circCtr.x - circPt.x,circCtr.y - circPt.y);
    setPixel(circCtr.x + circPt.y,circCtr.y + circPt.x);
    setPixel(circCtr.x - circPt.y,circCtr.y + circPt.x);
    setPixel(circCtr.x + circPt.y,circCtr.y - circPt.x);
    setPixel(circCtr.x - circPt.y,circCtr.y - circPt.x);
}

void circleMidPoint(Point circCtr, GLint radius){
    Point circPt;
    GLint p = 1 - radius;
    circPt.x = 0;
    circPt.y = radius;
    circlePlotPoints(circCtr, circPt);
    while(circPt.x < circPt.y){
        circPt.x++;
        if(p < 0) p += 2 * circPt.x + 1;
        else{
            circPt.y--;
            p += 2 * (circPt.x - circPt.y) + 1;
        }
        circlePlotPoints(circCtr, circPt);
    }
}

void display(){
    Point center;
    center.x = 300;
    center.y = 250;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    circleMidPoint(center, 100);
    glFlush();
}

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Mid Point");

    init();
    glutDisplayFunc(display);


    glutMainLoop();

}
```



6. Implementar el algoritmo del punto medio para la generación de elipses

```
#include <GL/glut.h>
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;

GLsizei winWidth = 600, winHeight = 500;

struct Point{
    GLint x;
    GLint y;
};

void init(void){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 600.0, 0.0, 500.0);
}

void setPixel(GLint x, GLint y){
    glBegin(GL_POINTS);
        glVertex2i(x,y);
    glEnd();
}

void elipcePlotPoints(Point centro, Point actual){
    setPixel(centro.x + actual.x, centro.y + actual.y);
    setPixel(centro.x - actual.x, centro.y + actual.y);
    setPixel(centro.x + actual.x, centro.y - actual.y);
    setPixel(centro.x - actual.x, centro.y - actual.y);
}

void elipceMidPoint(Point centro, GLint radiusX, GLint radiusY){
    GLdouble p1, p2;
    GLint rX2, rY2;
    Point actual;
    actual.x = 0;
    actual.y = radiusY;
    rX2 = pow(radiusX,2);
    rY2 = pow(radiusY,2);
    p1 = rY2 - (rX2 * radiusY) + (0.25 * rX2);
    while((rY2 * actual.x) < (rX2 * actual.y)){
        actual.x++;
        if(p1 < 0) p1 += (2 * rY2 * actual.x) + rY2;
        else{
            actual.y--;
            p1 += (2 * rY2 * actual.x) - (2 * rX2 * actual.y) + rY2;
        }
        elipcePlotPoints(centro,actual);
    }
    p2 = (rY2) * pow((actual.x + 0.5), 2) + (rX2) * pow((actual.y - 1), 2) - (rX2 * rY2);
    while(actual.y > 0){
        actual.y--;
        if(p2 > 0) p2 -= (2 * rX2 * actual.y) + rX2;
        else{
            actual.x++;
            p2 += (2 * rY2 * actual.x) - (2 * rX2 * actual.y) + rX2;
        }
        elipcePlotPoints(centro,actual);
    }
}

void display(){
```

```
    Point center;
    center.x = 300;
    center.y = 250;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    elipceMidPoint(center, 50, 200);
    elipceMidPoint(center, 200, 50);

    glFlush();
}

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Mid Point");

    init();
    glutDisplayFunc(display);


    glutMainLoop();

}
```

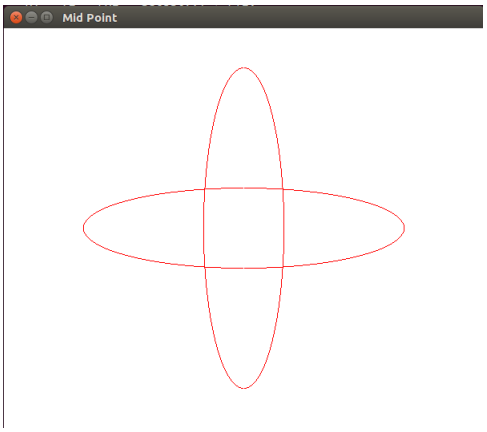


Figura 3: Resultados

7. **Realice un diseño artístico (Dibujo) con los algoritmos implementados en 3,4,5 y 6**

   Función que utiliza los anteriores algoritmos para dibujar arte.

```
void arte(){
    Point rec1;
    Point rec2;
    Point rec3;
    Point rec4;
    Point centro;
    centro.x = winWidth / 2;
    centro.y = winHeight / 2;
    GLint desplRec = 5;
    GLint finalRec = 50;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    for(GLint i = desplRec; i < finalRec; i += desplRec){
        rec1.x = i; rec1.y = i;
        rec2.x = i; rec2.y = winHeight - i;
        rec3.x = winWidth - i; rec3.y = winHeight - i;
        rec4.x = winWidth - i; rec4.y = i;
        DDA(rec1, rec2);
        bresenham(rec2, rec3);
        DDA(rec3, rec4);
        bresenham(rec1, rec4);
    }
    GLint desplCirc = 5;
    GLint finalCirc = 50;
    GLint desplXElip = 5;
```

```
    GLint desplYElip = 1;
    GLint finalElip = 50;
    centro.x -= 200;
    for(int k = 0; k < 5; k++){
        circleMidPoint(centro, 1);
        for(GLint i = desplCirc; i < finalCirc; i += desplCirc){
            circleMidPoint(centro, i);
        }
        centro.x += 100;
    }

    centro.x = winWidth / 2;
    centro.y -= 100;
    GLint rX = 1;
    GLint rY = 2;
    while(rX != finalElip and rY != finalElip){
        elipceMidPoint(centro, rX, rY);
        rX += desplXElip;
        rY += desplYElip;
    }
    rX -= desplXElip;
    rY -= desplYElip;
    centro.y += 200;
    elipceMidPoint(centro, rX, rY);
    circleMidPoint(centro, rY);

    for(int i = rY / 2; i != 5; i--){
        circleMidPoint(centro, i);
    }
    glFlush();
}
```

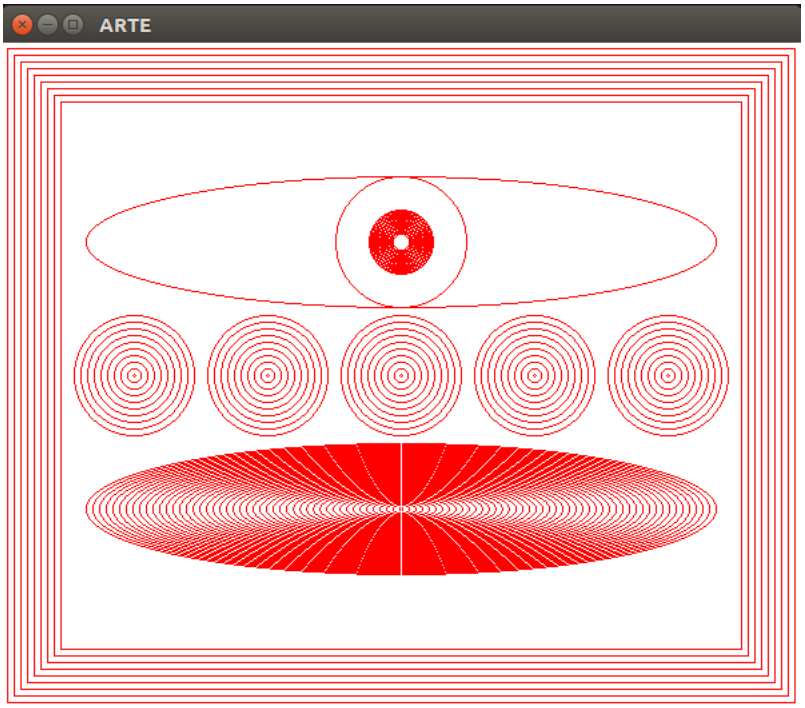


Figura 4: Resultados