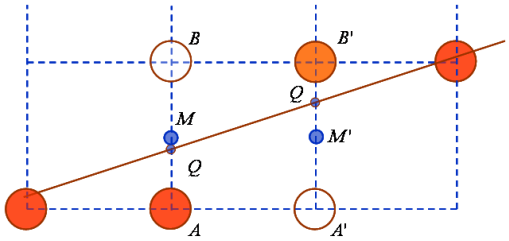


Algoritmo de punto medio para rectas.

Laboratorio 03

Diseño e implementación de primitivas gráficas

Objetivo: El objetivo de esta práctica, es que el alumno conozca y sepa cómo aplicar los principales algoritmos de discretización de líneas en 2D para rectas: algoritmo básico, algoritmo DDA (Digital Differential Analyzer), algoritmo de punto medio (BRESENHAM[1]); de igual manera que efectúe la implementación del algoritmo de punto medio para circunferencias.

Duración de la Práctica: 2 Horas.

Lugar de realización: Laboratorio de cómputo.

El conocimiento requerido para realizar esta Práctica es de haber asimilado los conceptos básicos de C/C++ y OpenGL.

El desarrollo tendrá la siguiente estructura de temas:

1. Introducción
2. Discretización de líneas
3. Algoritmo Básico
4. Algoritmo DDA
5. Algoritmo de Punto Medio (de Bresenham)
6. Programa ejemplo con OpenGL (para líneas)
7. Ejercicios propuestos
8. Circunferencias
9. Algoritmo basado en una representación paramétrica
10. Algoritmo basado en una representación explícita
11. Algoritmo de Punto Medio para circunferencias
12. Algoritmo de Punto Medio para elipses
13. Ejercicios propuestos
14. Referencias

[1] Jack E. Bresenham: el algoritmo para rectas fue publicado en la **IBM Systems Journal**, con el artículo **"Algorithm for computer control of a digital plotter"** en enero de 1965, mientras que la versión para circunferencias fue presentada con el artículo **"A linear algorithm for incremental digital display of circular arcs"** en la revista **Communications of the ACM**, en el año1977.

1. INTRODUCCIÓN

En este taller se presentan, esencialmente, algoritmos que tienen que ver con la conversión a puntos de primitivas gráficas para dibujar líneas.

Estos algoritmos se encuentran normalmente implementados en el procesador interno de las pantallas de barrido y en algunos trazadores, por lo que el programador de aplicaciones gráficas no tendrá, en general, que programarlos.

No obstante, es útil conocerlos, al menos por las siguientes razones:

- Pueden utilizarse para convertir impresoras gráficas en periféricos gráficos cómodos de usar.
- Ayudan a comprender mejor los problemas de eficiencia asociados a la conversión a píxeles.
- Son la base para implementaciones en hardware.
- Son imprescindibles para diseñar un periférico de barrido.

Un paquete gráfico raster aproxima las primitivas gráficas matemáticas (ideales), descritas en términos de un plano cartesiano, a un conjunto de píxeles de una determinada intensidad de gris o color. Estos píxeles son almacenados como mapas de bits en un buffer de memoria gráfica.

2. DISCRETIZACION DE LÍNEAS

Un algoritmo de discretización de líneas calcula las coordenadas de los píxeles que están sobre o cerca de una línea ideal infinitamente delgada sobrepuesta a una trama bidimensional (vea la Figura No. 1).

En principio nos gustaría que la secuencia de píxeles estuviera lo más cerca posible de la línea ideal y que además fuera lo más recta posible.

Para visualizar la geometría supondremos que mostraremos un píxel como un punto circular centrado en la posición (x, y) de la trama.

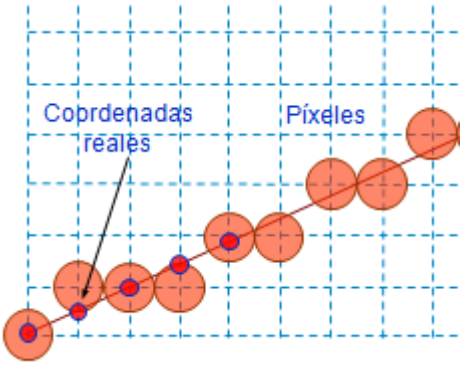


Figura No. 1. Relación de píxeles con la recta "ideal"

3. ALGORITMO BÁSICO

Es la forma más simple de abordar el problema, recordar que si no se tiene en cuenta el valor de la pendiente esta podría dar una representación grafica de la recta de manera errada.
El respectivo código en C/C++ se representa a continuación:

```
void recta_simple (int x0, int y0, int x1, int y1)
{
    int x;
    float dx, dy, m;
    dx = x1 - x0;
    dy = y1 - y0;
    m = dy/dx;
    b = y0 - m*x0;
    y = y0;
    for (x=x0; x<=x1; x++)
        {
            pintar(x, round(y), atributo);
            y = m*x + b;
        }
}
```

donde la función **pintar** puede ser diseñada de la siguiente manera (criterio del programador):

```
void pintar(int x,int y,char atributo)
{
    // Puede ser empleando glPoint(x,y);
    // Se sugiere que se implemente lo sugerido en las clases
    // detalles del atributo
}
```

4. ALGORITMO DDA (Digital Differential Analyzer)

Este algoritmo, llamado también de algoritmo incremental, representa una mejora con respecto al algoritmo Básico, donde aminora el número de operaciones básicas (elementales).

```
void recta_dda(int x0, int y0, int x1, int y1)
{
        int x;
        float dx, dy, m;
        dx = x1-x0;
        dy = y1-y0;
        m = dy/dx;
        y = y0;
        for(x=x0; x<=x1; x++)
                {
                pintar(x, round(y), atributo);
                y += m;
                }
}
```

5. ALGORITMO DE PUNTO MEDIO (BRESENHAM)

Una versión optimizada para el diseño de una recta viene dada por la aritmética entera, además de un número reducido de operaciones básicas con respecto a las anteriores.

```
void recta_punto_medio(int x0, int y0, int x1, int y1)
{
        int dx, dy, dE, dNE, d, x, y;
        dx = x1 - x0;
        dy = y1- y0;
        d = 2*dy - dx;
        dE = 2*dy;
        dNE = 2*(dy - dx);
        x = x0;
        y = y0;
        pintar(x, y, atributo);
        while (x<x1)
                {
                if(d<=0){
                        d += dE;
                        x++;
                        }
                else{
                        d += dNE;
                        x++;
                        y++;
                        }
                pintar(x, y, atributo);
                }
}
```

Recuerde que estos dos últimos algoritmos están restrictos al valor de la pendiente menor que uno, por lo tanto, deberá hacer las consideraciones del caso para el ploteo adecuado de la recta.

6. PROGRAMA EJEMPLO CON OPENGL

Queremos generar una recta usando el algoritmo básico, para esto tenga en cuenta:

$$y = mx + b$$

$$m = \frac{\Delta x}{\Delta y} = \frac{y1 - y0}{x1 - x0}$$

donde m es el valor de la pendiente.

Un programa realizado con el lenguaje C++ para resolver el problema planteado, se describe a continuación:

```
#include <math.h>
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

void inicio(void);
void display(void);
void myreshape(int, int);
void abasico(int, int, int, int);
void ingresoDatos(void);
int px0,py0,px1,py1;

int main(int argc, char** argv)
{
   ingresoDatos();
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(200, 200);
   glutInitWindowPosition(100,100);
   glutCreateWindow("Algoritmos de primitivas graficas");

   inicio();
   glutDisplayFunc(display);
   glutReshapeFunc(myreshape);
   glutMainLoop();
   return 0;
}

void inicio(void)
{
   glClearColor(1.0,1.0,1.0,0.0);  //parametros: rojo, amarillo y azul, el cuarto es el parametro
alpha
   glShadeModel(GL_FLAT);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
```

```
    glPushMatrix();          // salva el estado actual de la matriz
    glColor3f(0,0,1);        // Azul
    glPointSize(2);  // Fije el grosor de pixel = 2
    abasico(px0, py0, px1, py1);
    glPopMatrix();     // recupera el estado del matriz
    glFlush();
}

void myreshape(int w, int h)
{
    glViewport(0,0,(GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void abasico(int x0,int y0,int x1,int y1)
{
    int x;
    float dx, dy, m, b, y;
    dx = x1 - x0;
    dy = y1 - y0;
    m = dy/dx;
    b = y0 - m*x0;
    y = y0;
    glBegin(GL_POINTS);
    for(x=x0; x<=x1; x++)
       {
          y = m*x + b;
          cout << x <<"  "<< y <<"  "<< (int)ceil(y-0.5) <<"\n";
          glVertex2f(x, (int)ceil(y-0.5));
       }
    glEnd();
}

// opcional
void ingresoDatos(void)
{
    // lo que Ud desee
    cout <<"\n leer px0="; cin >> px0;
    cout <<"\n leer py0="; cin >> py0;
    cout <<"\n leer px1="; cin >> px1;
    cout <<"\n leer py1="; cin >> py1;
}
```

Observación:

Note que, en el código, que se ha fijado el grosor del píxel a 2 unidades, y que la aplicación muestra lo sugerido por el algoritmo. Pero, si se realiza una maximización de su ventana, notará que el trazo continuo de la recta se pierde, tendiendo que cambiar el grosor del píxel a uno apropiado. Esto en realidad no representa problema alguno, al contrario de otras APIs (algunas), OpenGL trabaja con píxeles con entradas en punto flotante, es decir no necesariamente valores enteros, consiguiendo mejor detalle gráfico. Cuando se encapsula dos coordenadas con el modo GL_LINES, OpenGL hace las cuentas adicionales para mostrar un trazo continuo. Esto quiere decir, que en las futuras aplicaciones no nos debemos preocupar si debemos plotear coordenadas reales en general.

7. EJERCICIOS PROPUESTOS

Ejercicio 01:

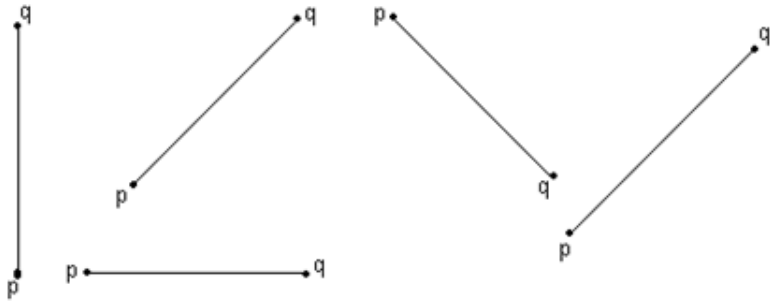
Realice los cambios necesarios para que se grafique la recta en el caso en que los puntos inicial y final no estén ordenados según las abscisas.

Ejercicio 02:

Emplee los menús jerárquicos a fin de gestionar la visualización de cualquiera de los tres algoritmos, uno a continuación del otro, no lo superponga.

Ejercicio 03:

Diseñar la función **LINEA (p, q)** que permita pintar una línea considerando todos los casos (línea vertical, horizontal, oblicua, etc.) utilizando el algoritmo de punto medio.
Debe considerar que **p(x_0, y_0)** y **q(x_f, y_f)** son los puntos inicial y final de la recta respectivamente.



Ejercicio 04:

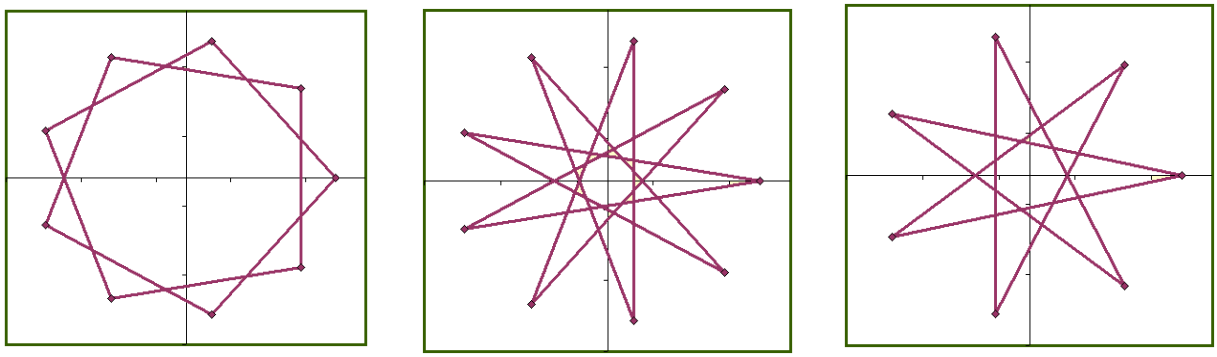
Diseñar una aplicación gráfica que incorpore un “método” para insertar un “punto” en la ventana gráfica, empleando el controlador del ratón. La idea es, presionar el botón izquierdo (o derecho) y que se registre en un array las coordenadas capturadas, luego dibujar el segmento de recta con los dos puntos, con la función LINEA.

Ejercicio 05:

Implementar una aplicación para dibujar un polígono estrellado partiendo de un polígono regular de N lados. Un polígono regular estrellado puede construirse a partir del regular convexo uniendo vértices no consecutivos de forma continua. Vea las figuras adyacentes.

El heptágono regular genera dos estrellados, 7/2 y 7/3:

El eneágono genera dos estrellados, 9/2 y 9/4:



8. CIRCUNFERENCIAS

Las circunferencias son probablemente las curvas más utilizadas en los gráficos elementales; generalmente, se utilizan como bloques de construcción para generar imágenes artísticas o técnicas de resolución (por ejemplo, vea la Figura No. 4).

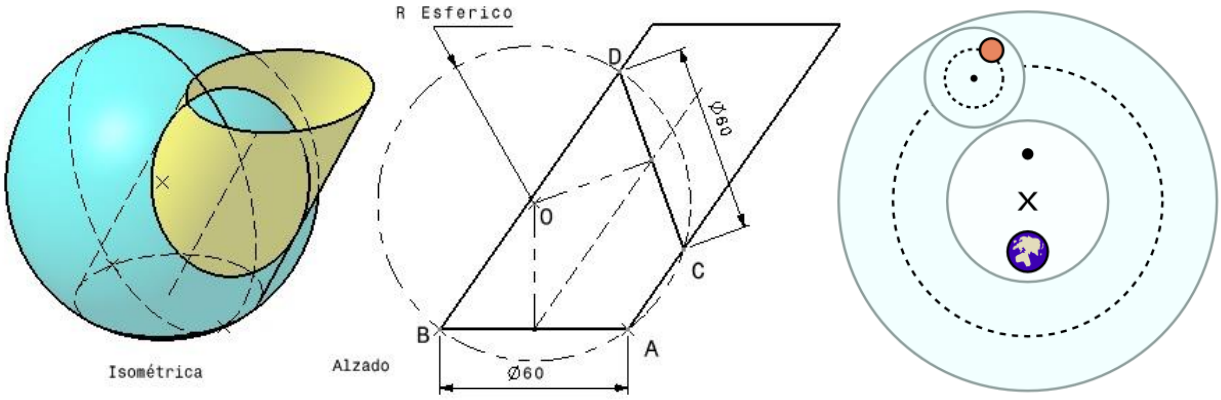


Figura No. 4. Uso de circunferencias en los dibujos

De forma análoga al diseño de rectas, también se dispone de algoritmos para diseñar circunferencias vía la discretización de los píxeles. Para efectos de una presentación mas simple o sencilla trabajaremos con la circunferencia centrada en el origen y luego mediante una traslación la llevaremos a su posición original.

Además, aprovecharemos el hecho de que este tipo de cónica posee propiedades simétricas con respecto a los ejes cartesianos y con respecto a la recta identidad en el I cuadrante (Figura No. 5).

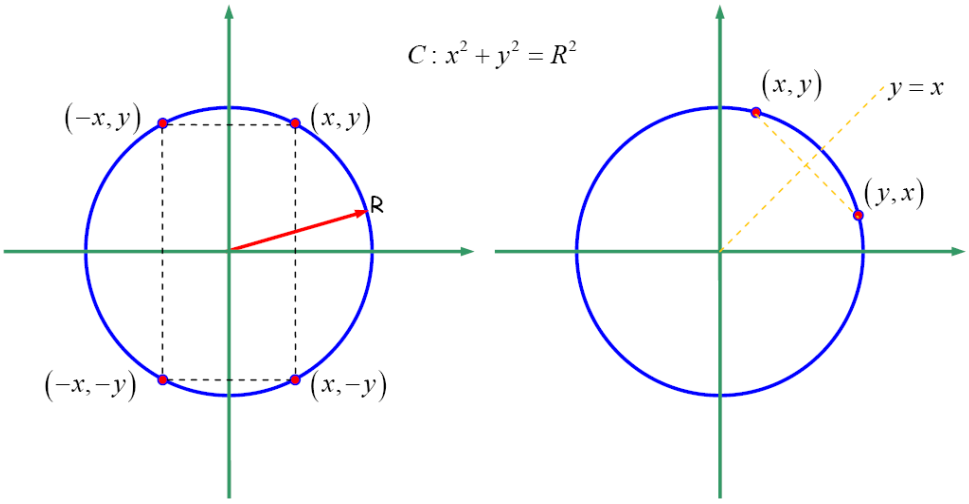


Figura No. 5. Propiedades simétricas de la circunferencia

Esto nos conduce a discretizar en un solo octante y en especial en aquella región en donde se puede pasterizar horizontalmente; es decir en el II octante donde las coordenadas (x, y) de la circunferencia satisfacen la relación x < y, posteriormente se procede a hacer las copias necesarias en los siete octantes restantes.

9. ALGORITMO BASADO EN UNA REPRESENTACIÓN PARAMÉTRICA

Para esto sólo se considera lo siguiente:

$$x = R\cos\theta$$
$$y = R\,\text{sen}\,\theta$$

El ángulo deberá variar de $\frac{\pi}{4}$ a $\frac{\pi}{2}$ para permitir recorrer el II octante (Figura No 3).

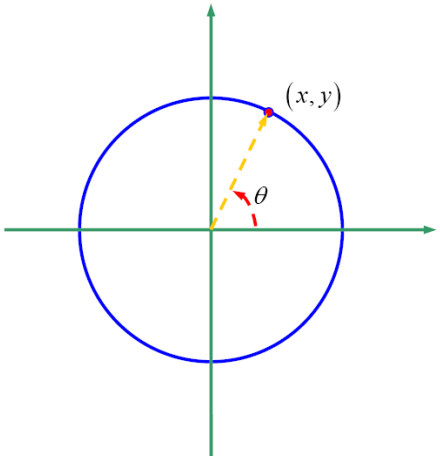


Figura No. 6. Representación paramétrica de la circunferencia

Esto permite plantear el esquema como sigue:

```
void circunferencia_parametrica(int R)
{
        float x,y;
        float PI=3.1415……..;
        float teta=PI/4;
        delta=0.1;
        while (teta<PI/2)
                {
                x=R*cos(teta);
                y=R*sin(teta);
                teta+=delta;
                pintar(round(x),round(y),atributo);
                }
}
```

Observación:

Round(x)=truncamiento(x+0.5)

10. ALGORITMO BASADO EN UNA REPRESENTACIÓN EXPLÍCITA

Se desprende de la ecuación general de la circunferencia (Figura No 7.):

$$y = \mp\sqrt{R^2 - x^2}$$

Obviamente el cálculo de las coordenadas va ha ser precedida por un costo adicional o una sobrecarga de operaciones diferentes a las básicas o elementales, como es la potencia y la función raíz cuadrada.

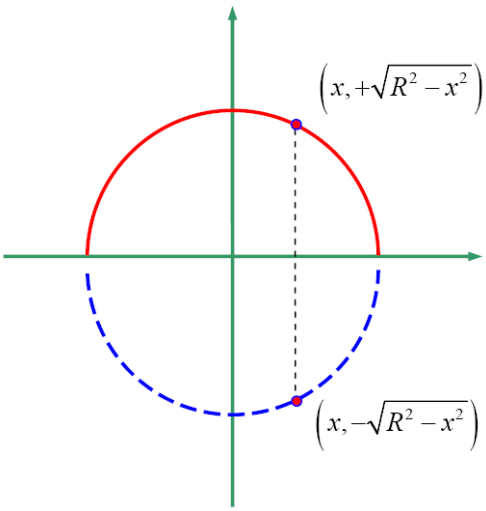


Figura No. 7. Representación explicita de la circunferencia.

El siguiente esquema traduce lo comentado:

```
void circunferencia_explicita(int R)
{
        float x=0,y=R;
        while (x<y)
                {
                y=sqrt(pow(R,2)-pow(x,2));
                pintar(x,round(y),atributo);
                x++;
                }
}
```

11. ALGORITMO DE PUNTO MEDIO PARA CIRCUNFERENCIAS

Finalmente abordamos la versión de punto medio para circunferencias, la cuál realiza operaciones aritméticas enteras tal como se ha planteado y desarrollado en las clases respectivas.

A continuación, se presenta el código propuesto en **Computer Graphics: Principles and Practice,** la cual presenta algunos ajustes para evitar el cociente 5/4>1:

```
void circunferencia_punto_medio(int R)
{
        // discretizacion valida en el II octante
        int x=0;
        int y=R,d=1-R;
        pintar(x,y,atributo);
        while (x<y){
                if (d<0)
                d+=2*x+3;
                else{
                d+=2(x-y)+5;
                y--
                    }
                x++;
                pintar(x,y,atributo);
                }
}
```

12. ALGORITMO DE PUNTO MEDIO PARA LA ELIPSE

El planteamiento que se utiliza aquí es similar a aquel empleado en el despliegue de una circunferencia. Dado los parámetros r_x, r_y, (x_c, y_c), se determina los puntos (x, y) para una elipse en posición estándar centrada en el origen y luego se altera los puntos, de modo que la elipse este centrada en (x_c, y_c).

El método de punto medio para elipse se aplica a lo largo del primer cuadrante en dos partes, de acuerdo con la elipse con la pendiente $r_x < r_y$, como se muestra a continuación.

Se procesa este cuadrante tomando pasos unitarios en la dirección de x donde la pendiente de la curva tiene una magnitud menor que -1 y tomando pasos unitarios en la dirección de y donde la pendiente tiene una magnitud mayor que -1.

Las regiones 1 y 2 pueden procesarse de varias maneras. Se puede iniciar en la posición (0, r_y) y pasar en el sentido del reloj a lo largo de la trayectoria elíptica en el primer cuadrante, al alternar de pasos unitarios en x a pasos unitarios en y cuando la pendiente adquiere un valor menor que -1.

De modo alternativo, se puede iniciar en (r_x, 0) y seleccionar puntos en el sentido contrario al de las manecillas del reloj, alternando de pasos unitarios en y a pasos unitarios en x cuando la pendiente adquiere un valor mayor que -1.

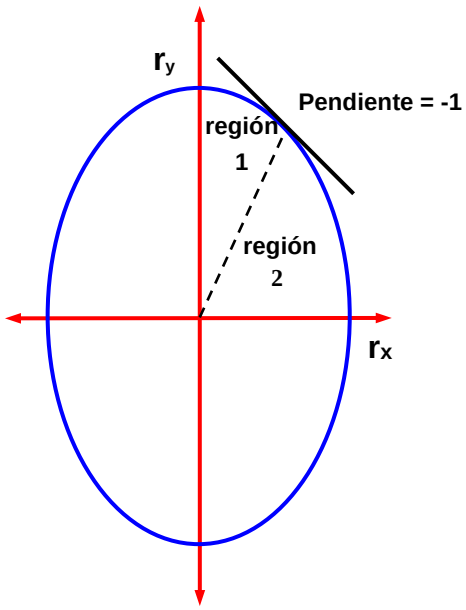


Figura No. 5. Punto medio para elipses

La pendiente de la curva (la tangente) se calcula a partir de la ecuación:

$$f_{elipse}(x,y) = r_y^{\,2}x^2 + r_x^{\,2}y^2 - r_x^{\,2}r_y^{\,2} = 0$$

con $(x_c, y_c) = (0,0)$, además $\frac{dy}{dx} = -\frac{2r_y^{\,2}x}{2r_x^{\,2}y}$

En la frontera entre la región 1 y la región 2:

$$\frac{dy}{dx} = -1$$

Análogamente:

$\frac{dy}{dx} < -1$ en la región 1 y $\frac{dy}{dx} > -1$ en la región 2

y en la frontera:

$$2r_y^{\,2}x = 2r_x^{\,2}y$$

Por lo tanto, se mueve hacia fuera de la región 1 siempre que:

$$2r_y^{\,2}x \geq 2r_x^{\,2}y\,.$$

13. EJERCICIOS PROPUESTOS

Ejercicio 01:

Proceda como en el laboratorio anterior e implemente las tres alternativas (descrita antes) para representar la circunferencia, para esto el usuario deberá ingresar el radio respectivo. El programa mostrará los gráficos con las tres técnicas.

Ejercicio 02 (Importante):

Ahora haga las modificaciones necesarias para graficar una circunferencia centrada en las coordenadas (x_0, y_0):

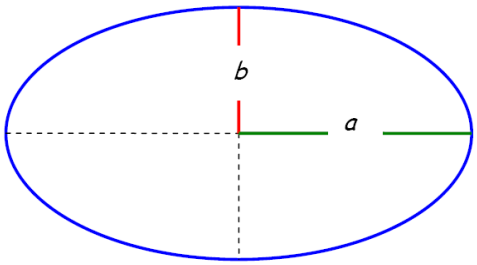
$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

Naturalmente se deberá leer dichas coordenadas además del radio, y los atributos que el usuario desea considerar.

Ejercicio 03:

Implementar el algoritmo de punto medio respectivo para graficar una elipse que tiene como ecuación:

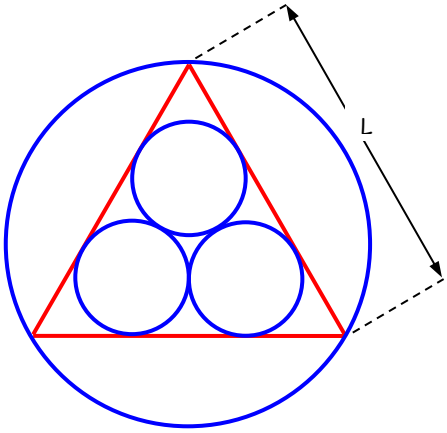
$$\frac{x^{\,2}}{a^2} + \frac{y^{\,2}}{b^2} = 1$$



El usuario deberá ingresar los valores de **a** y **b**, así mismo solo deberá trabajar en el primer cuadrante y razonar usando simetría. Desarrolle las modificaciones necesarias para obtener el despliegue gráfico de la elipse de ecuación general.

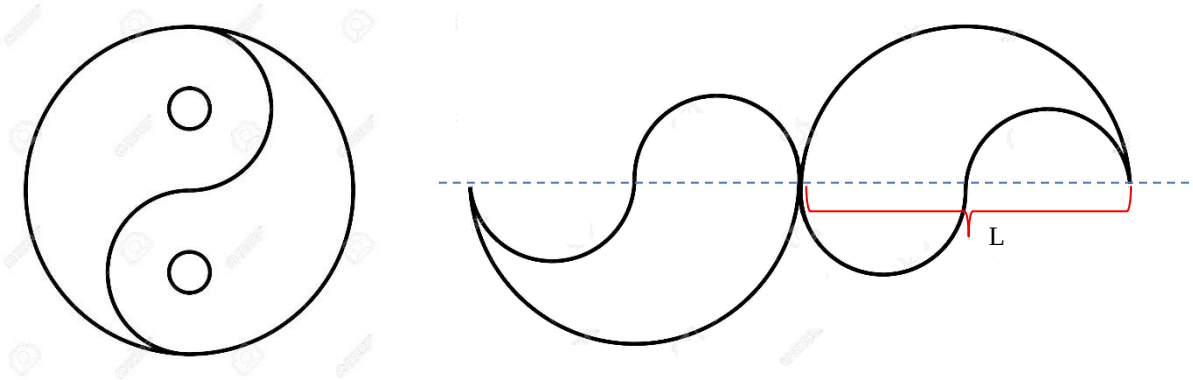
Ejercicio 04:

Implemente un programa que lea el valor de **L**, y que permita mostrar un triángulo equilátero de lado **L** inscrito en una circunferencia, además de 3 circunferencias pequeñas de igual radio contenidas en el mencionado triangulo. Se le asignará el color azul al triangulo y rojo a las circunferencias, vea la gráfica que se adjunta.



Ejercicio 05:

Realizar un programa con OpenGL que permita dibujar los siguientes diseños geométricos. Las dimensiones dependerán del valor de L (a elección suya), considere el grosor y el color de su agrado.



14. REFERENCIAS

Referencias:

1. Sellers G, Wright R, & Haemel N. (2016). **OpenGL Superbible** (7ma edición). Indiana. Addison-Wesley.
2. Shreiner D, Sellers G, Kessenich J. (2013). **OpenGL Programming Guide: The Official Guide to Learning OpenGL**. Michigan. Addison-Wesley.
3. Mark J. Kilgard. (1996). **The OpenGL Utility Toolkit (GLUT) Programming Interface** (API Version 3). Silicon Graphics Inc.
4. Mark Segal, Kurt Akeley. (1998). **The OpenGL Graphics System: A Specification** (Version 1.2). Silicon Graphics Inc.
5. Foley J. D., A.van Dam, S.K.Feiner and J.F. Hughes (1990). **Computer Graphics. Principles and Practice (Second Edition).** Addison-Wesley.