



# CIRCUNFERENCIA DE BRESENHAM

DEL VALLE ARAGÓN ALEXIS RAFAEL

- COMPUTACIÓN GRÁFICA E  
INTERACCIÓN HUMANO  
COMPUTADORA
- GRUPO: 06
- FECHA: 9/ABRIL/2023

UNIVERSIDAD NACIONAL AUTÓNOMA DE  
MÉXICO  
FACULTAD DE INGENIERÍA

A través de ChatGPT realizar la búsqueda del algoritmo de una circunferencia de Bresenham y adaptarlo a un código para que sea funcional.



Ilustración 1 Consulta de ChatGPT

Código de consulta:

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

```
// Función que dibuja la circunferencia utilizando el algoritmo de Bresenham  
void drawCircle(int x0, int y0, int r) {
```

```
    int x = 0;  
    int y = r;  
    int d = 3 - 2*r;
```

```
    glBegin(GL_POINTS);  
    while (x <= y) {  
        glVertex2i(x0 + x, y0 + y);  
        glVertex2i(x0 + y, y0 + x);  
        glVertex2i(x0 - y, y0 + x);  
        glVertex2i(x0 - x, y0 + y);  
        glVertex2i(x0 - x, y0 - y);  
        glVertex2i(x0 - y, y0 - x);  
        glVertex2i(x0 + y, y0 - x);  
        glVertex2i(x0 + x, y0 - y);  
    }  
    glEnd();  
}
```

```

        if (d < 0) {
            d = d + 4*x + 6;
        } else {
            d = d + 4*(x-y) + 10;
            y--;
        }
        x++;
    }
    glEnd();
}

```

// Función que se encarga de renderizar la escena

```

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

```

```

    glColor3f(1.0, 1.0, 1.0); // Establecemos el color de la circunferencia a blanco

```

```

    drawCircle(200, 200, 100); // Dibujamos la circunferencia centrada en (200, 200) con radio 100

```

```

    glFlush();
}

```

// Función principal

```

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

```

```

    glutInit(&argc, argv);

```

```

    glutCreateWindow("Circunferencia de Bresenham");

```

```

    glutInitWindowSize(400, 400);

```

```

    glutInitWindowPosition(100, 100);

```

```

    glutDisplayFunc(display);

```

```







    glutMainLoop();

```

```

    return 0;
}

```

	glut.dll	18/08/1998 04:25 p. m.	Extensión de la ap...	166 KB
	glut.h	18/08/1998 04:24 p. m.	C/C++ Header	21 KB
	glut.lib	18/08/1998 04:25 p. m.	Object File Library	78 KB
	glut32.dll	18/08/1998 04:25 p. m.	Extensión de la ap...	166 KB
	glut32.lib	18/08/1998 04:25 p. m.	Object File Library	79 KB
	glutdlls37beta	08/04/2023 04:45 p. m.	Carpeta comprimi...	146 KB

Para realizar la ejecución pertinente, es necesario obtener la paquetería Glut directamente de [www.opengl.org](http://www.opengl.org) si, como en mi caso, no contaba con ella, y agregamos las extensiones según corresponda a las carpetas de nuestro proyecto.

Debug	08/04/2023 05:32 p. m.	Carpeta de archivos	
include	08/04/2023 04:48 p. m.	Carpeta de archivos	
lib	08/04/2023 04:50 p. m.	Carpeta de archivos	
Nueva carpeta	08/04/2023 04:46 p. m.	Carpeta de archivos	
shaders	26/02/2019 08:20 p. m.	Carpeta de archivos	
Circunferencia.cpp	08/04/2023 05:32 p. m.	C++ Source	3 KB
Circunferencia	08/04/2023 06:05 p. m.	Documento de Mi...	0 KB
glew32.dll	09/01/2019 09:55 p. m.	Extensión de la ap...	381 KB
glfw3.dll	09/01/2019 09:56 p. m.	Extensión de la ap...	70 KB
glut.dll	18/08/1998 04:25 p. m.	Extensión de la ap...	166 KB
glut32.dll	18/08/1998 04:25 p. m.	Extensión de la ap...	166 KB

Agregamos en las carpetas “include”, “lib”, y nuestra carpeta principal junto a nuestro código.cpp.

```

30  /*Ingresamos nuestros valores de entrada
31  | centro (X,Y) y radio de la circunferencia*/
32  float   x = 45.0f,
33          y = 69.0f,
34          radius = 300.87f;
35
36  void getResolution()

```

Dado que no se pueden obtener los valores de dibujo directamente desde la consola, es necesario ingresarlas dentro del código.

```

if ( radius > 0)    //Validamos que el radio sea positivo
{
    while (xPos <= yPos)
    {
        // Simétrico alrededor de los ejes cartesianos
        glVertex2f(x + xPos, y + yPos);
        glVertex2f(x - xPos, y + yPos);
        glVertex2f(x + xPos, y - yPos);
        glVertex2f(x - xPos, y - yPos);
        glVertex2f(x + yPos, y + xPos);
        glVertex2f(x - yPos, y + xPos);
        glVertex2f(x + yPos, y - xPos);
        glVertex2f(x - yPos, y - xPos);

        if (d < 0)
        {
            d += 4 * xPos + 6;
        }
        else
        {
            d += 4 * (xPos - yPos) + 10;
            yPos--;
        }
    }
}

```

Para llevar a cabo el algoritmo es necesario validar que el radio sea positivo.

```

91 //Plano cartesiano 2 veces el radio de cada eje.
92 // Eje x
93 glBegin(GL_LINES);
94 glVertex2f(-2*radius, x);
95 glVertex2f(2*radius, x);
96 glEnd();
97
98 // Eje y
99 glBegin(GL_LINES);
100 glVertex2f(y, -2*radius);
101 glVertex2f(y, 2*radius);
102 glEnd();
103
104 // Mostrar el resultado
105 glutSwapBuffers();
106 }

```

En el centro establecido (puede ser variable), se dibuja un plano cartesiano con una longitud 2 veces el radio ingresado para cada uno de los ejes.

```

108 int main(int argc, char** argv)
109 {
110
111     glutInit(&argc, argv);
112     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
113     //Pantalla 3 veces el radio en horizontal y vertical
114     //Es recomendable obtener un tamaño cuadrático para observar la circunferencia con
115     glutInitWindowSize(3*radius, 3*radius);
116     glutCreateWindow("Circunferencia de Bresenham");
117     //Posicion de pantalla desplazada 2 veces el radio del centro en X y Y
118     //Para obtener un tamaño medio.
119     gluOrtho2D(x-2*radius, x+ 2 * radius, y- 2 * radius, y+ 2 * radius);
120     glutDisplayFunc(display);
121     glutMainLoop();
122     glEnable(GL_DEPTH_TEST);
123     return 0;
124 }

```

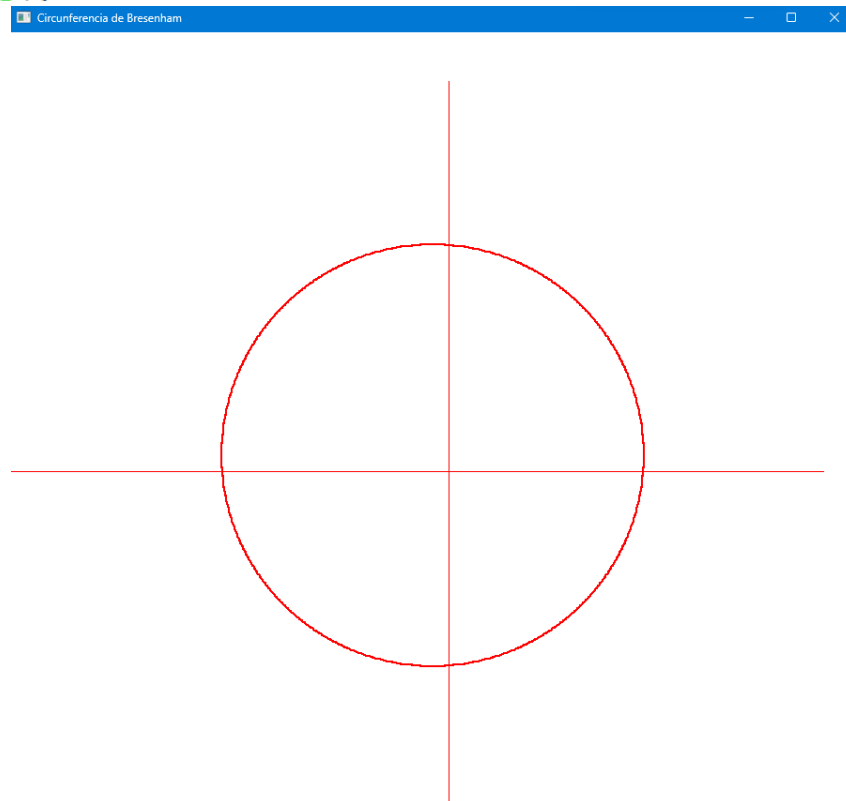
Por último adaptamos nuestra pantalla de muestra:

- Dado que trabajamos con una circunferencia y no queremos que se observe de manera distorsionada, ingresamos valores de tamaño cuadrático vertical y horizontal, es decir: el valor debe ser el mismo para ambos.
- La circunferencia debe ser observada con un tamaño mediano dentro de la ventana, así que ingresamos un tamaño 3 veces el radio ingresado, de esta forma no se visualizará ni muy pequeño ni muy grande.
- La posición de la ventana se encontrará ubicada en el centro de la circunferencia (x,y), desplazándose según se necesité, sumando y restando dos veces el valor de nuestro radio para que no se observe una figura recortada.

```

30  /*Ingresamos nuestros valores de entrada
31  | centro (X,Y) y radio de la circunferencia*/
32  float   x = 45.0f,
33          y = 69.0f,
34          radius = 300.87f;
35
36  void getResolution()
37  {

```



Circunferencia con centro (45,69) y radio= 300.87.