

C# Y .NET 8

Parte 13. Gráficos 3D

2024-07

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	3
Acerca del autor.....	4
Licencia de este libro	4
Licencia del software	4
Marcas registradas	4
Dedicatoria	6
Proyección 3D a 2D	7
Girando un objeto 3D y mostrarlo en 2D	13
Matriz de Giro en X.....	13
Matriz de Giro en Y	13
Matriz de Giro en Z.....	14
Giro centrado: Cálculo de constantes.....	21
Giro centrado: Uso de las constantes.....	26
Combinando los tres giros: Cálculo de constantes.....	34
Combinando los tres giros: Uso de constantes	39
Líneas ocultas.....	47
Gráfico Matemático en 3D.....	54
Gráfico Polar en 3D	66
Gráfico de sólido de revolución	77

Tabla de ilustraciones

Ilustración 1: Proyección de objeto 3D en pantalla 2D	7
Ilustración 2: Esquema de proyección.....	8
Ilustración 3: Proyección 3D a 2D de una figura tridimensional: un cubo	12
Ilustración 4: Diseño de pantalla	14
Ilustración 5: Giro figura en 3D	19
Ilustración 6: Giro en el eje X de la figura 3D. No hay giro en los otros ejes.....	19
Ilustración 7: Giro en el eje Y de la figura 3D. No hay giro en los otros ejes.....	20
Ilustración 8: Giro en el eje Z de la figura 3D. No hay giro en los otros ejes.....	20
Ilustración 9: Constantes para después calcular los puntos en pantalla	25
Ilustración 10: Diseño de ventana.....	26
Ilustración 11: Cubo proyectado por defecto.....	31
Ilustración 12: Cubo proyectado con giro en X	32
Ilustración 13: Cubo proyectado con giro en Y	33
Ilustración 14: Cubo proyectado con giro en Z	33
Ilustración 15: Constantes para cuadrar el cubo en la pantalla.....	38
Ilustración 16: Diseño de ventana.....	39
Ilustración 17: Cubo proyectado, sin giros.....	44
Ilustración 18: Cubo proyectado con giros.....	45
Ilustración 19: Cubo proyectado con giros.....	46
Ilustración 20: Líneas ocultas.....	52
Ilustración 21: Líneas ocultas.....	53
Ilustración 22: Diseño de ventana.....	55
Ilustración 23: Ecuación en 3D proyectada	63
Ilustración 24: Ecuación en 3D proyectada y girada	64
Ilustración 25: Ecuación 3D proyectada y girada	65
Ilustración 26: Diseño de ventana.....	67
Ilustración 27: Ecuación polar 3D	75
Ilustración 28: Ecuación polar 3D	76
Ilustración 29: Sólido de revolución.....	85
Ilustración 30: Sólido de revolución.....	86

Acerca del autor

Rafael Alberto Moreno Parra

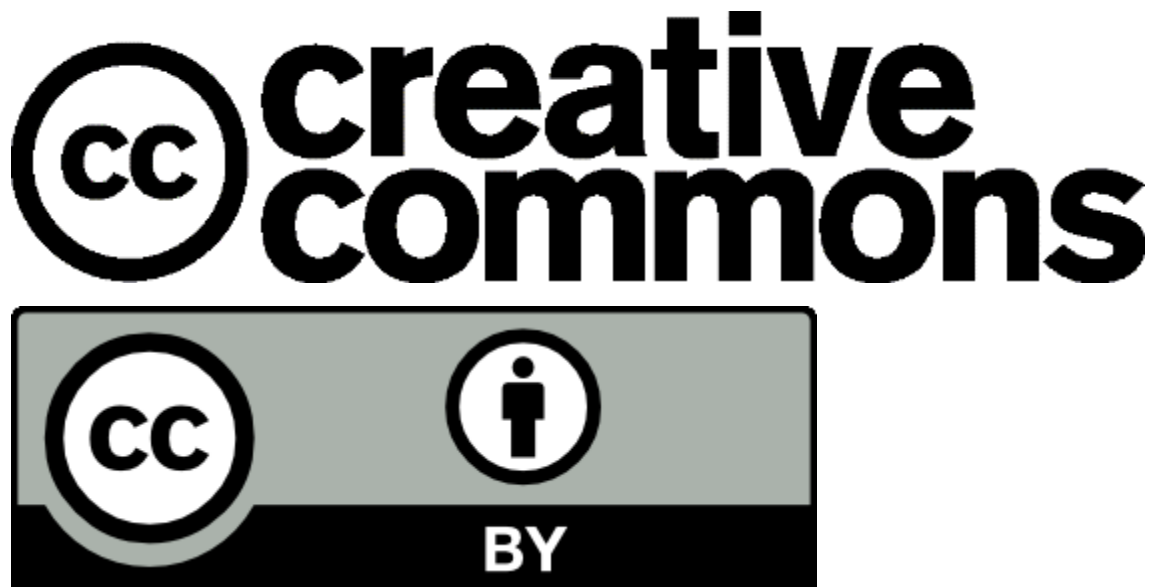
ramsoftware@gmail.com o enginelifelife@hotmail.com

Sitio Web: <http://darwin.50webs.com> (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: <https://github.com/ramsoftware>

Youtube: <https://www.youtube.com/@RafaelMorenoP>

Licencia de este libro



Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License" [1]



Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2022 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

A mis padres, a mi hermana....

Y a mi tropa gatuna: Sally, Suini, Grisú, Capuchina, Milú,
Arián, Frac y mis recordados Tinita, Tammy, Vikingo y
Michu.

Proyección 3D a 2D

Dado un objeto tridimensional que flota entre la persona y una pantalla plana, ¿Cómo se proyectaría este objeto tridimensional en la pantalla? Ese objeto tiene coordenadas (X, Y, Z) y deben convertirse a coordenadas planas (X_{plano}, Y_{plano}) . Hay que considerar la distancia de la persona u observador a ese objeto. Si la persona está muy cerca del objeto 3D, lo verá grande, si se aleja el observador, lo verá pequeño.

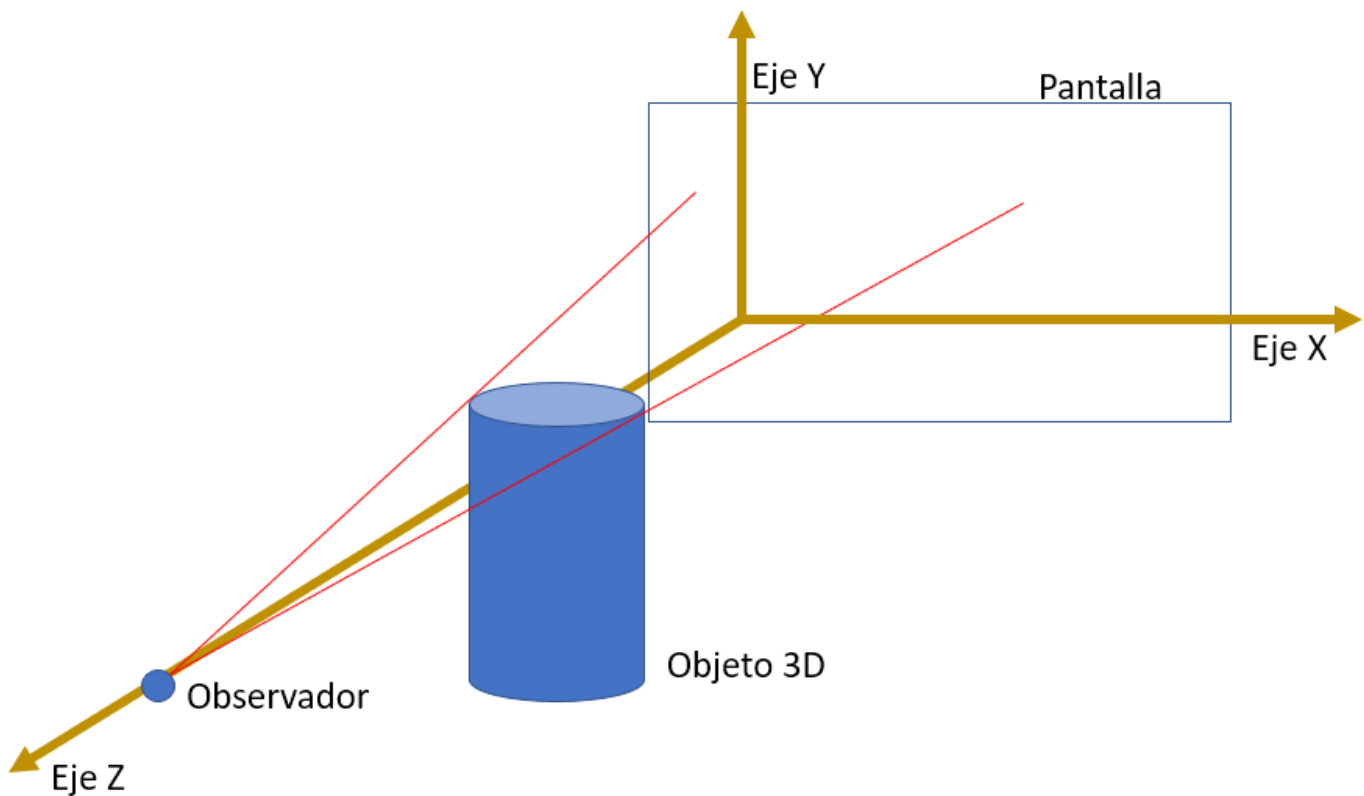


Ilustración 1: Proyección de objeto 3D en pantalla 2D

Las ecuaciones para pasar coordenadas X, Y, Z a coordenadas planas X_{plano}, Y_{plano} , considerando la distancia del observador ($Z_{Persona}$) es:

$$X_{plano} = (Z_{Persona} * X) / (Z_{Persona} - Z)$$

$$Y_{plano} = (Z_{Persona} * Y) / (Z_{Persona} - Z)$$

Demostración, si ponemos el valor de $X = 0$ para ver sólo el comportamiento de Y , este sería el gráfico

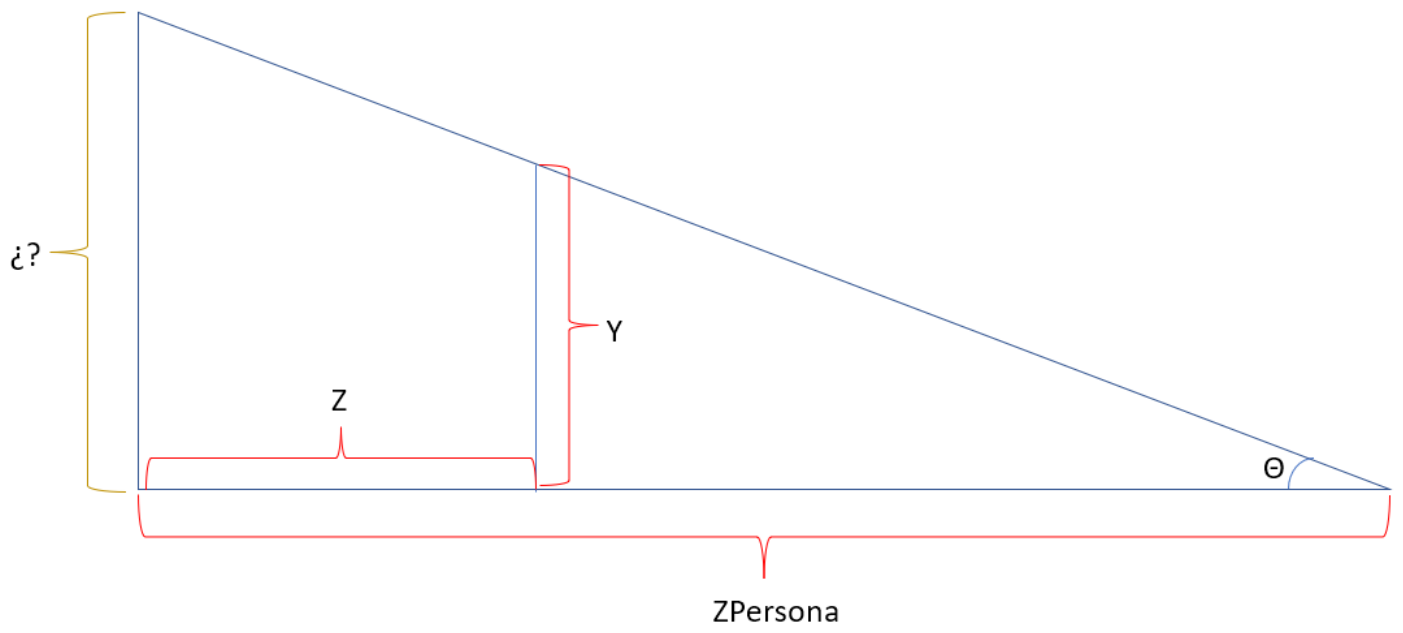


Ilustración 2: Esquema de proyección

Necesitamos hallar el valor de ¿? Que sería Yplano, ¿Cómo? La respuesta está en el ángulo Θ , más concretamente en la tangente que sería así:

$$\tan \theta = \frac{Y}{ZPersona - Z}$$

Pero también sabemos que:

$$\tan \theta = \frac{¿?}{ZPersona}$$

Luego igualamos:

$$\frac{¿?}{ZPersona} = \frac{Y}{ZPersona - Z}$$

Despejando:

$$¿? = \frac{ZPersona * Y}{ZPersona - Z}$$

Luego:

$$Yplano = \frac{ZPersona * Y}{ZPersona - Z}$$

Igualmente, para Xplano sería:

$$Xplano = \frac{ZPersona * X}{ZPersona - Z}$$

La implementación:

M/001.cs

```
namespace Graficos {  
  
    public partial class Form1 : Form {  
        public Form1() {  
            InitializeComponent();  
        }  
    }  
}
```

```

}

private void Form1_Paint(object sender, PaintEventArgs e) {
    Graphics Lienzo = e.Graphics;
    Pen Lapiz = new(Color.Blue, 3);

    //Distancia de la persona (observador)
    //al plano donde se proyecta la "sombra" del cubo
    int ZPersona = 180;

    Cubo Figura3D = new();
    Figura3D.Convierte3Da2D(ZPersona);
    Figura3D.Dibuja(Lienzo, Lapiz);
}
}

internal class Cubo {
    //Un cubo tiene 8 puntos con
    //coordenadas espaciales X, Y, Z
    private List<int> Punto;

    //Luego tiene 8 coordenadas planas
    private List<int> pX;
    private List<int> pY;

    //Constructor
    public Cubo() {
        //Ejemplo de coordenadas
        //espaciales X,Y,Z
        Punto = [
            50, 50, 50,
            100, 50, 50,
            100, 100, 50,
            50, 100, 50,
            50, 50, 100,
            100, 50, 100,
            100, 100, 100,
            50, 100, 100 ];

        pX = [];
        pY = [];
    }

    //Convierte de 3D a 2D
    public void Convierte3Da2D(int ZPersona) {
        for (int cont = 0; cont < Punto.Count; cont += 3) {
            //Extrae las coordenadas espaciales
            int X = Punto[cont];

```

```

        int Y = Punto[cont + 1];
        int Z = Punto[cont + 2];

        //Convierte a coordenadas planas
        int Xp = (ZPersona * X) / (ZPersona - Z);
        int Yp = (ZPersona * Y) / (ZPersona - Z);

        //Agrega las coordenadas planas a la lista
        pX.Add(Xp);
        pY.Add(Yp);
    }
}

//Dibuja el cubo
public void Dibuja(Graphics lienzo, Pen lapiz) {
    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[1], pY[1]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[2], pY[2]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[3], pY[3]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[0], pY[0]);

    lienzo.DrawLine(lapiz, pX[4], pY[4], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[5], pY[5], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[6], pY[6], pX[7], pY[7]);
    lienzo.DrawLine(lapiz, pX[7], pY[7], pX[4], pY[4]);

    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[4], pY[4]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[7], pY[7]);
}
}
}

```

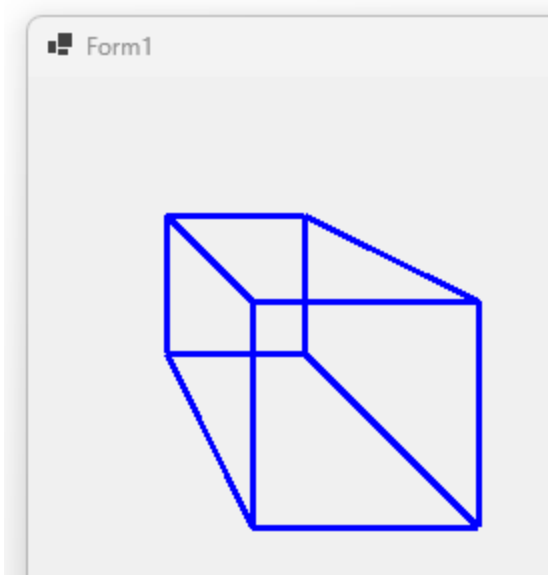


Ilustración 3: Proyección 3D a 2D de una figura tridimensional: un cubo

Girando un objeto 3D y mostrarlo en 2D

En https://www.cs.buap.mx/~iolmos/graficacion/5_Transformaciones_geometricas_3D.pdf se explican las tres rotaciones en los ejes X, Y, Z por separado. Nota: La matriz en el texto referenciado está como (columna, fila) en este libro es (columna, fila), de todos modos, las ecuaciones de giro son las mismas.

Para aplicar el giro, se requiere una matriz de transformación. Estas serían las matrices:

Matriz de Giro en X

$$\begin{bmatrix} X_{giro} \\ Y_{giro} \\ Z_{giro} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(angX) & \sin(angX) \\ 0 & -\sin(angX) & \cos(angX) \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicándolo:

$$X_{giro} = 1 * X + 0 * Y + 0 * Z$$

$$Y_{giro} = 0 * X + \cos(angX) * Y - \sin(angX) * Z$$

$$Z_{giro} = 0 * X + \sin(angX) * Y + \cos(angX) * Z$$

Matriz de Giro en Y

$$\begin{bmatrix} X_{giro} \\ Y_{giro} \\ Z_{giro} \end{bmatrix} = \begin{bmatrix} \cos(angY) & 0 & -\sin(angY) \\ 0 & 1 & 0 \\ \sin(angY) & 0 & \cos(angY) \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicándolo:

$$X_{giro} = \cos(angY) * X + 0 * Y + \sin(angY) * Z$$

$$Y_{giro} = 0 * X + 1 * Y + 0 * Z$$

$$Z_{giro} = -\sin(angY) * X + 0 * Y + \cos(angY) * Z$$

Matriz de Giro en Z

$$\begin{bmatrix} X_{giro} \\ Y_{giro} \\ Z_{giro} \end{bmatrix} = \begin{bmatrix} \cos(angZ) & \text{sen}(angZ) & 0 \\ -\text{sen}(angZ) & \cos(angZ) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicándolo:

$$X_{giro} = \cos(angZ) * X - \text{sen}(angZ) * Y + 0 * Z$$

$$Y_{giro} = \text{sen}(angZ) * X + \cos(angZ) * Y + 0 * Z$$

$$Z_{giro} = 0 * X + 0 * Y + 1 * Z$$

Como es una aplicación gráfica de escritorio, entonces así debe ser la ventana:

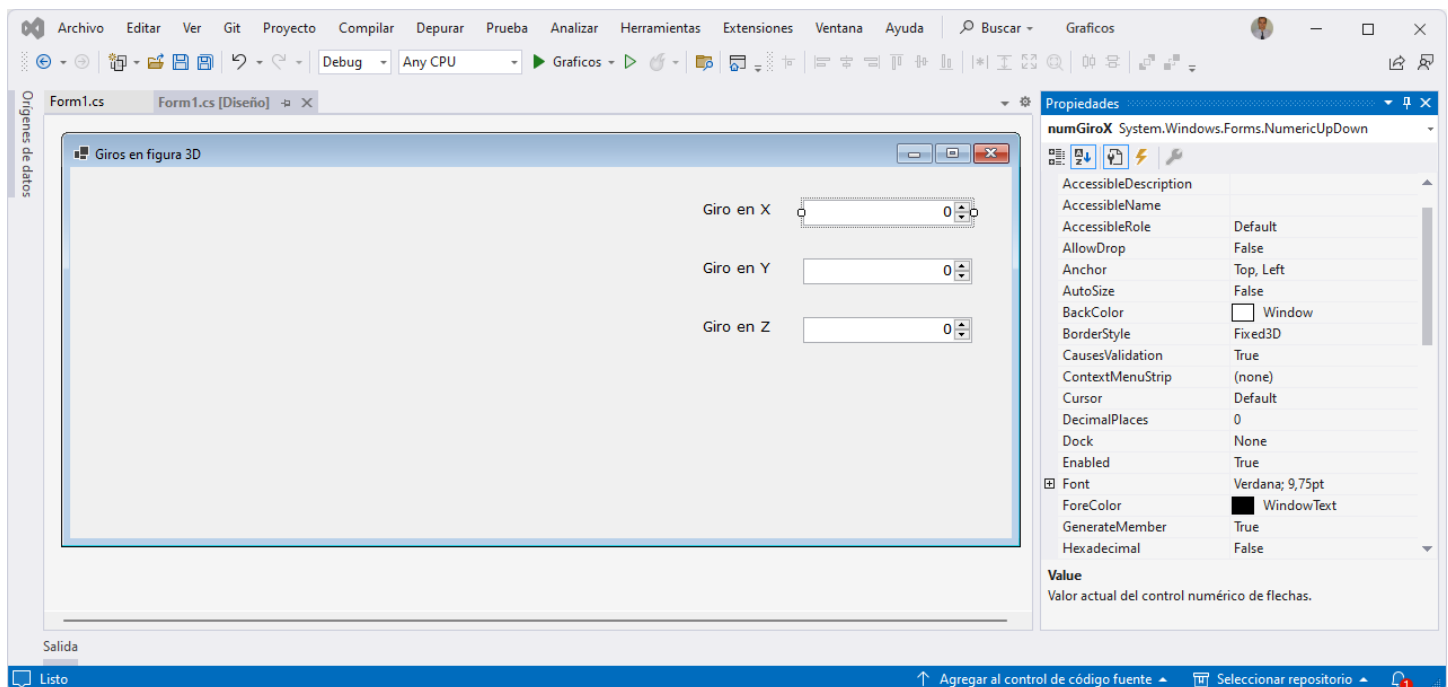


Ilustración 4: Diseño de pantalla

Se utilizan los siguientes controles:

Tipo	Nombre
Label	IblGiroX
Label	IblGiroY
Label	IblGiroZ
NumericUpDown	numGiroX
NumericUpDown	numGiroY
NumericUpDown	numGiroZ

```

namespace Graficos {

    public partial class Form1 : Form {
        //El cubo que se proyecta y gira
        Cubo Figura3D;

        public Form1() {
            InitializeComponent();
            Figura3D = new Cubo();
            Figura3D.AplicaGiro(0, 0);
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapiz = new(Color.Blue, 3);

            int ZPersona = 180;
            Figura3D.Convierte3Da2D(ZPersona);
            Figura3D.Dibuja(Lienzo, Lapiz);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            //Se anulan los dos valores de los otros ángulos
            numGiroY.Value = 0;
            numGiroZ.Value = 0;

            //Sólo puede girar en un ángulo
            int AnguloX = Convert.ToInt32(numGiroX.Value);
            Figura3D.AplicaGiro(0, AnguloX); //0 es giro en X
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            numGiroX.Value = 0;
            numGiroZ.Value = 0;
            int AnguloY = Convert.ToInt32(numGiroY.Value);
            Figura3D.AplicaGiro(1, AnguloY); //1 es giro en Y
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            numGiroX.Value = 0;
            numGiroY.Value = 0;
            int AnguloZ = Convert.ToInt32(numGiroZ.Value);
            Figura3D.AplicaGiro(2, AnguloZ); //2 es giro en Z
        }
    }
}

```

```

        Refresh();
    }
}

internal class Cubo {
    //Un cubo tiene 8 puntos con
    //coordenadas espaciales X, Y, Z
    private List<int> Punto;

    //Un cubo tiene 8 coordenadas
    //espaciales X, Y, Z que han sido giradas
    private List<double> Giradas;

    //Luego tiene 8 coordenadas planas
    private List<int> pX;
    private List<int> pY;

    //Constructor
    public Cubo() {
        //Ejemplo de coordenadas
        //espaciales X,Y,Z
        Punto = [
            50, 50, 50,
            100, 50, 50,
            100, 100, 50,
            50, 100, 50,
            50, 50, 100,
            100, 50, 100,
            100, 100, 100,
            50, 100, 100 ];

        Giradas = [];
        pX = [];
        pY = [];
    }

    public void AplicaGiro(int CualAnguloGira, int ValorAngulo) {
        //Dependiendo de cuál ángulo gira
        switch (CualAnguloGira) {
            case 0: GiroX(ValorAngulo); break;
            case 1: GiroY(ValorAngulo); break;
            case 2: GiroZ(ValorAngulo); break;
        }
    }

    //Gira en X
    private void GiroX(double AnguloGrados) {
        double AnguloRadianes = AnguloGrados * Math.PI / 180;
    }
}

```



```

double[,] Matriz = new double[3, 3] {
    {1, 0, 0},
    {0, Math.Cos(AnguloRadianes), Math.Sin(AnguloRadianes)},
    {0, -Math.Sin(AnguloRadianes), Math.Cos(AnguloRadianes) }
};

AplicaMatrizGiro(Matriz);
}

//Gira en Y
private void GiroY(double AnguloGrados) {
    double AnguloRadianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {Math.Cos(AnguloRadianes), 0, -Math.Sin(AnguloRadianes)},
        {0, 1, 0},
        {Math.Sin(AnguloRadianes), 0, Math.Cos(AnguloRadianes) }
    };

    AplicaMatrizGiro(Matriz);
}

//Gira en Z
private void GiroZ(double AnguloGrados) {
    double AnguloRadianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {Math.Cos(AnguloRadianes), Math.Sin(AnguloRadianes), 0},
        {-Math.Sin(AnguloRadianes), Math.Cos(AnguloRadianes), 0},
        {0, 0, 1 }
    };

    AplicaMatrizGiro(Matriz);
}

private void AplicaMatrizGiro(double[,] Mt) {
    Giradas.Clear();

    //Gira las 8 coordenadas
    for (int Cont = 0; Cont < Punto.Count; Cont += 3) {
        //Extrae las coordenadas espaciales
        int X = Punto[Cont];
        int Y = Punto[Cont + 1];
        int Z = Punto[Cont + 2];

        //Hace el giro
        double Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
    }
}

```

```

        double Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
        double Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];

        //Guarda en la lista de giro
        Giradas.Add(Xg);
        Giradas.Add(Yg);
        Giradas.Add(Zg);
    }
}

//Convierte de 3D a 2D las coordenadas giradas
public void Convierte3Da2D(int ZPersona) {
    pX.Clear();
    pY.Clear();

    for (int cont = 0; cont < Giradas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Giradas[cont];
        double Y = Giradas[cont + 1];
        double Z = Giradas[cont + 2];

        //Convierte a coordenadas planas
        int Xp = Convert.ToInt32(ZPersona * X / (ZPersona - Z));
        int Yp = Convert.ToInt32(ZPersona * Y / (ZPersona - Z));

        //Agrega las coordenadas planas a la lista
        pX.Add(Xp);
        pY.Add(Yp);
    }
}

//Dibuja el cubo
public void Dibuja(Graphics lienzo, Pen lapiz) {
    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[1], pY[1]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[2], pY[2]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[3], pY[3]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[0], pY[0]);

    lienzo.DrawLine(lapiz, pX[4], pY[4], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[5], pY[5], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[6], pY[6], pX[7], pY[7]);
    lienzo.DrawLine(lapiz, pX[7], pY[7], pX[4], pY[4]);

    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[4], pY[4]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[7], pY[7]);
}

```

```
}  
}
```

Ejemplo de ejecución:



Ilustración 5: Giro figura en 3D

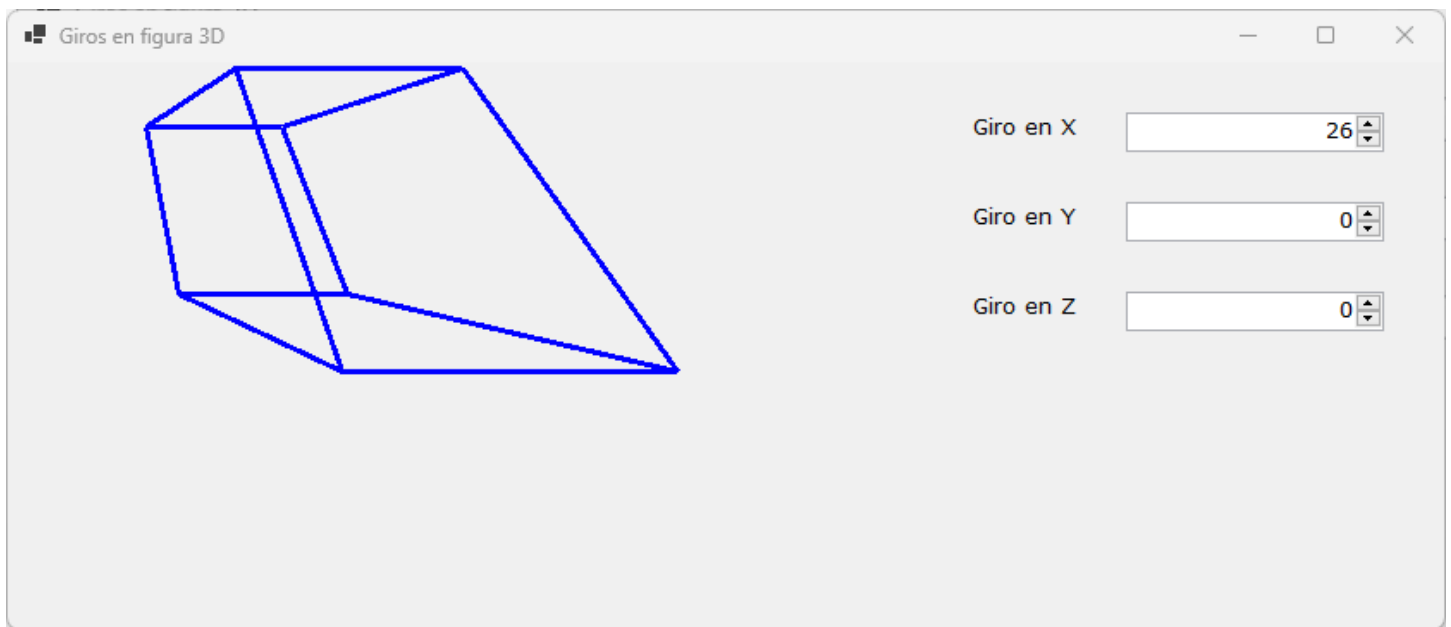


Ilustración 6: Giro en el eje X de la figura 3D. No hay giro en los otros ejes.



Ilustración 7: Giro en el eje Y de la figura 3D. No hay giro en los otros ejes.



Ilustración 8: Giro en el eje Z de la figura 3D. No hay giro en los otros ejes.

Giro centrado: Cálculo de constantes

En el ejemplo anterior, el punto (0,0,0) se encuentra en la parte superior izquierda de la ventana, el cubo está a un lado, no en el centro, por lo que girar el cubo también implica desplazarlo. Llega a un punto que se sale de la ventana y no se puede ver. Se hace un cambio al cubo para que su centroide esté en la posición (0,0,0), además de mejorar la proyección en pantalla.

El pasar el cubo proyectado a la pantalla física es el que requiere más cuidado porque hay que cuadrarlo considerando la distancia del observador y los diferentes giros que puede hacer el cubo. El problema es que el cubo se puede ver bien en la posición predeterminada (ángulos en 0,0,0), pero al girarlo, se salen los vértices de los límites de la pantalla o ventana, luego hay que ajustar la conversión de plano a pantalla de tal manera que por más que lo gire en cualquier ángulo, no se salga el dibujo de la ventana. Para lograr eso, se sabe que el cubo tiene coordenadas de -0.5 a 0.5 (es decir de lado=1) tanto en X, Y, y Z. Se deja fija la distancia del observador, en este caso con un valor de 5 (suficientemente alejado de la figura). Con esos valores entonces se probaron todos los ángulos de giro en X de 0 a 360 grados, en Y de 0 a 360 grados y en Z de 0 a 360 grados, así:

Desde ángulo en X = 0 grados hasta 360 grados → Calcule el X plano mínimo, Y plano mínimo, X plano máximo, Y plano máximo

Desde ángulo en Y = 0 grados hasta 360 grados → Calcule el X plano mínimo, Y plano mínimo, X plano máximo, Y plano máximo

Desde ángulo en Z = 0 grados hasta 360 grados → Calcule el X plano mínimo, Y plano mínimo, X plano máximo, Y plano máximo

Con eso se obtienen los valores X plano mínimo, Y plano mínimo, X plano máximo, y Y plano máximo. Con esos valores se calcularán más adelante como poner los puntos en pantalla.

El siguiente software de salida por consola es para dar con esos valores extremos:

M/003.cs

```
namespace Ejemplo {  
  
    internal class Cubo {  
        //Un cubo tiene 8 coordenadas  
        //espaciales X, Y, Z  
        private List<double> Coordenadas;  
  
        //Se convierten en 8 coordenadas  
        //espaciales X, Y, Z al girarse  
        private List<double> Giradas;  
    }  
}
```

```

//Luego tiene 8 coordenadas planas
private List<double> PlanoX;
private List<double> PlanoY;

//Constructor
public Cubo() {
    //Coordenadas espaciales X,Y,Z
    //para hacer el cubo
    Coordenadas = [
        -0.5, -0.5, -0.5,
        0.5, -0.5, -0.5,
        0.5, 0.5, -0.5,
        -0.5, 0.5, -0.5,
        -0.5, -0.5, 0.5,
        0.5, -0.5, 0.5,
        0.5, 0.5, 0.5,
        -0.5, 0.5, 0.5 ];

    Giradas = [];
    PlanoX = [];
    PlanoY = [];
}

//Gira en X
private void GiroX(double AnguloGrados) {
    double Radianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {1, 0, 0},
        {0, Math.Cos(Radianes), Math.Sin(Radianes)},
        {0, -Math.Sin(Radianes), Math.Cos(Radianes) }
    };

    AplicaMatrizGiro(Matriz);
}

//Gira en Y
private void GiroY(double AnguloGrados) {
    double Radianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {Math.Cos(Radianes), 0, -Math.Sin(Radianes)},
        {0, 1, 0},
        {Math.Sin(Radianes), 0, Math.Cos(Radianes) }
    };

    AplicaMatrizGiro(Matriz);
}

```

```

//Gira en Z
private void GiroZ(double AnguloGrados) {
    double Radianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {Math.Cos(Radianes), Math.Sin(Radianes), 0},
        {-Math.Sin(Radianes), Math.Cos(Radianes), 0},
        {0, 0, 1}
    };

    AplicaMatrizGiro(Matriz);
}

private void AplicaMatrizGiro(double[,] Mt) {
    Giradas.Clear();

    //Gira las 8 coordenadas
    for (int cont = 0; cont < Coordenadas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Coordenadas[cont];
        double Y = Coordenadas[cont + 1];
        double Z = Coordenadas[cont + 2];

        //Hace el giro
        double Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
        double Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
        double Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];

        //Guarda en la lista de giro
        Giradas.Add(Xg);
        Giradas.Add(Yg);
        Giradas.Add(Zg);
    }
}

//Convierte de 3D a 2D las coordenadas giradas
private void Convierte3Da2D(int ZPersona) {
    for (int cont = 0; cont < Giradas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Giradas[cont];
        double Y = Giradas[cont + 1];
        double Z = Giradas[cont + 2];

        //Convierte a coordenadas planas
        double Xp = ZPersona * X / (ZPersona - Z);
        double Yp = ZPersona * Y / (ZPersona - Z);
    }
}

```

```

        //Agrega las coordenadas planas a la lista
        PlanoX.Add(Xp);
        PlanoY.Add(Yp);
    }
}

//Calcula los extremos de las coordenadas
//del cubo al girar y proyectarse
public void CalculaExtremo(int ZPersona) {
    double maximoX = double.MinValue;
    double minimoX = double.MaxValue;
    double maximoY = double.MinValue;
    double minimoY = double.MaxValue;

    for (double angX = 0; angX <= 360; angX++) {
        GiroX(angX);
        Convierte3Da2D(ZPersona);
    }

    for (double angY = 0; angY <= 360; angY++) {
        GiroY(angY);
        Convierte3Da2D(ZPersona);
    }

    for (double angZ = 0; angZ <= 360; angZ++) {
        GiroZ(angZ);
        Convierte3Da2D(ZPersona);
    }

    for (int cont = 0; cont < PlanoX.Count; cont++) {
        if (PlanoX[cont] < minimoX) minimoX = PlanoX[cont];
        if (PlanoX[cont] > maximoX) maximoX = PlanoX[cont];
        if (PlanoY[cont] < minimoY) minimoY = PlanoY[cont];
        if (PlanoY[cont] > maximoY) maximoY = PlanoY[cont];
    }

    Console.WriteLine("Proyección a 2D del cubo");
    Console.WriteLine("MinimoX: " + minimoX);
    Console.WriteLine("MaximoX: " + maximoX);
    Console.WriteLine("MinimoY: " + minimoY);
    Console.WriteLine("MaximoY: " + maximoY);
}

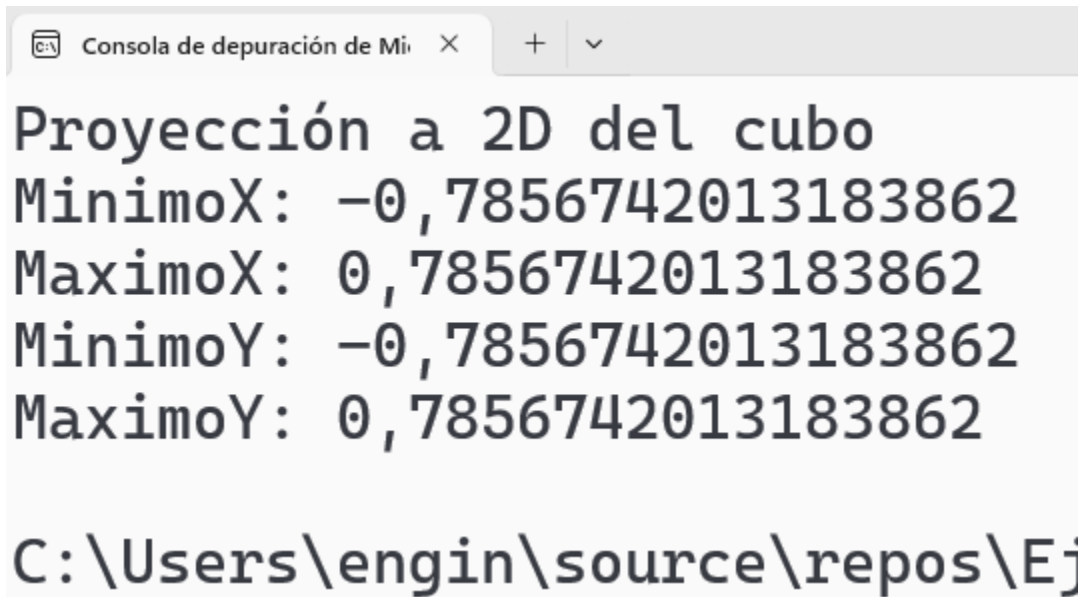
}

class Program {
    static void Main() {
        Cubo Figura3D = new();
    }
}

```



```
Figura3D.CalculaExtremo(5);  
    }  
}  
}
```



The screenshot shows a console window titled 'Consola de depuración de Mi' with a search bar and a dropdown menu. The output text is as follows:

```
Proyección a 2D del cubo  
MinimoX: -0,7856742013183862  
MaximoX: 0,7856742013183862  
MinimoY: -0,7856742013183862  
MaximoY: 0,7856742013183862  
  
C:\Users\engin\source\repos\Ej
```

Ilustración 9: Constantes para después calcular los puntos en pantalla

Esas constantes son las que se usan para generar las coordenadas en pantalla física.

Nota 1: Si cambia la distancia del observador al plano o cambia el tamaño de la figura 3D hay que recalcular nuevas constantes.

Nota 2: Esas constantes se usarán en futuras implementaciones.

Giro centrado: Uso de las constantes

Se rehace el programa de proyección del cubo con el uso de constantes calculadas en el programa anterior.

Como es una aplicación gráfica de escritorio, entonces así debe ser la ventana:

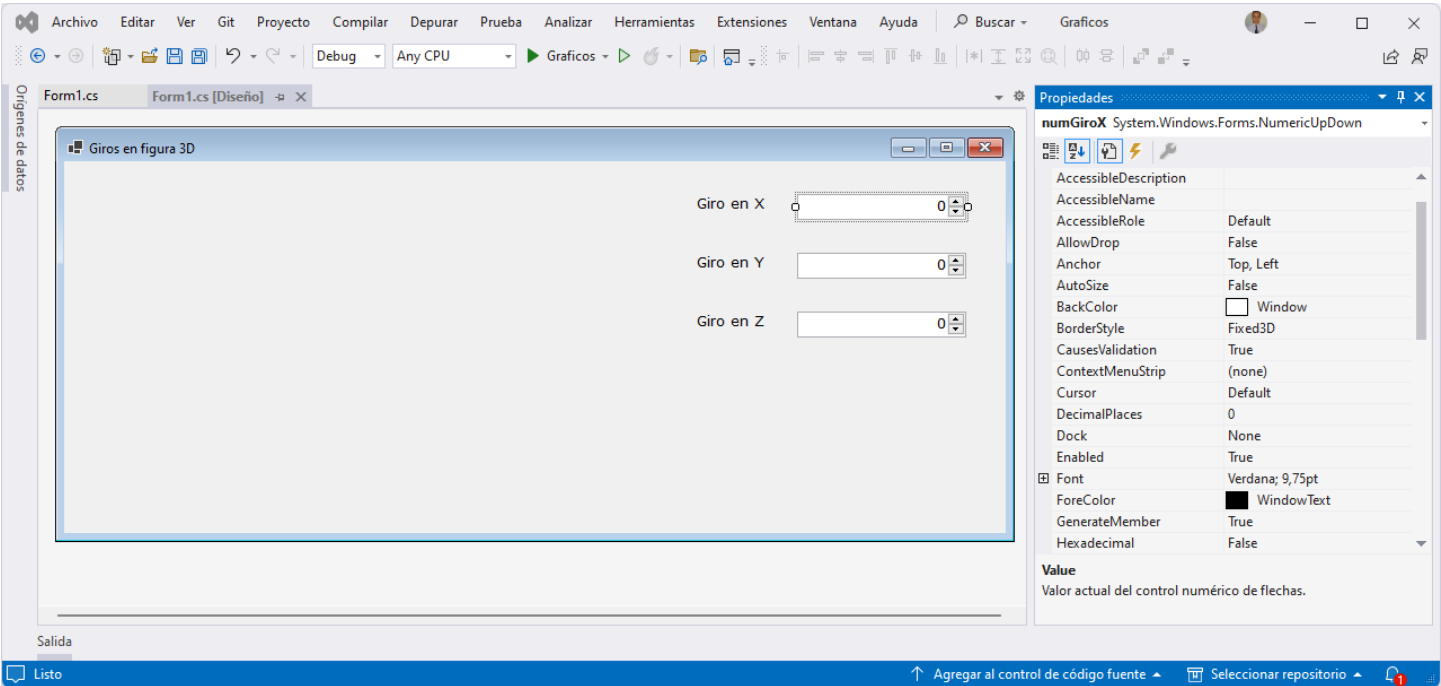


Ilustración 10: Diseño de ventana

Se utilizan los siguientes controles:

Tipo	Nombre
Label	IblGiroX
Label	IblGiroY
Label	IblGiroZ
NumericUpDown	numGiroX
NumericUpDown	numGiroY
NumericUpDown	numGiroZ

M/004.cs

```
namespace Graficos {
    public partial class Form1 : Form {
        //El cubo que se proyecta y gira
        Cubo Figura3D;

        public Form1() {
            InitializeComponent();
        }
    }
}
```

```

        Figura3D = new Cubo();
        Figura3D.AplicaGiro(0, 0);
    }

    private void numGiroX_ValueChanged(object sender, EventArgs e) {
        numGiroY.Value = 0;
        numGiroZ.Value = 0;
        int AnguloX = Convert.ToInt32(numGiroX.Value);
        Figura3D.AplicaGiro(0, AnguloX);
        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        numGiroX.Value = 0;
        numGiroZ.Value = 0;
        int AnguloY = Convert.ToInt32(numGiroY.Value);
        Figura3D.AplicaGiro(1, AnguloY);
        Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        numGiroX.Value = 0;
        numGiroY.Value = 0;
        int AnguloZ = Convert.ToInt32(numGiroZ.Value);
        Figura3D.AplicaGiro(2, AnguloZ);
        Refresh();
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics Lienzo = e.Graphics;
        Pen Lapiz = new(Color.Black, 2);

        int ZPersona = 180;
        Figura3D.Convierte3Da2D(ZPersona);
        Figura3D.CuadraPantalla(20, 20, 500, 500);
        Figura3D.Dibuja(Lienzo, Lapiz);
    }
}

internal class Cubo {
    //Un cubo tiene 8 coordenadas
    //espaciales X, Y, Z
    private List<double> Coordenadas;

    //Un cubo tiene 8 coordenadas
    //espaciales X, Y, Z que han sido giradas
    private List<double> Giradas;

```

```

//Luego tiene 8 coordenadas planas
private List<double> PlanoX;
private List<double> PlanoY;

//Luego tiene 8 coordenadas de pantalla
private List<int> pX;
private List<int> pY;

//Constructor
public Cubo() {
    //Ejemplo de coordenadas espaciales X,Y,Z
    Coordenadas = new List<double>() {
        -0.5, -0.5, -0.5,
        0.5, -0.5, -0.5,
        0.5, 0.5, -0.5,
        -0.5, 0.5, -0.5,
        -0.5, -0.5, 0.5,
        0.5, -0.5, 0.5,
        0.5, 0.5, 0.5,
        -0.5, 0.5, 0.5 };

    Giradas = [];
    PlanoX = [];
    PlanoY = [];
    pX = [];
    pY = [];
}

public void AplicaGiro(int CualAnguloGira, int ValorAngulo) {
    //Dependiendo de cuál ángulo gira
    switch (CualAnguloGira) {
        case 0: GiroX(ValorAngulo); break;
        case 1: GiroY(ValorAngulo); break;
        case 2: GiroZ(ValorAngulo); break;
    }
}

//Gira en X
private void GiroX(double AnguloGrados) {
    double Radianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {1, 0, 0},
        {0, Math.Cos(Radianes), Math.Sin(Radianes)},
        {0, -Math.Sin(Radianes), Math.Cos(Radianes) }
    };
};

```

```

    AplicaMatrizGiro(Matriz);
}

//Gira en Y
private void GiroY(double AnguloGrados) {
    double Radianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {Math.Cos(Radianes), 0, -Math.Sin(Radianes)},
        {0, 1, 0},
        {Math.Sin(Radianes), 0, Math.Cos(Radianes)}
    };

    AplicaMatrizGiro(Matriz);
}

//Gira en Z
private void GiroZ(double AnguloGrados) {
    double Radianes = AnguloGrados * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {Math.Cos(Radianes), Math.Sin(Radianes), 0},
        {-Math.Sin(Radianes), Math.Cos(Radianes), 0},
        {0, 0, 1}
    };

    AplicaMatrizGiro(Matriz);
}

private void AplicaMatrizGiro(double[,] Mt) {
    Giradas.Clear();

    //Gira las 8 coordenadas
    for (int cont = 0; cont < Coordenadas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Coordenadas[cont];
        double Y = Coordenadas[cont + 1];
        double Z = Coordenadas[cont + 2];

        //Hace el giro
        double Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
        double Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
        double Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];

        //Guarda en la lista de giro
        Giradas.Add(Xg);
        Giradas.Add(Yg);
        Giradas.Add(Zg);
    }
}

```

```

    }
}

//Convierte de 3D a 2D las coordenadas giradas
public void Convierte3Da2D(int ZPersona) {
    PlanoX.Clear();
    PlanoY.Clear();

    for (int cont = 0; cont < Giradas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Giradas[cont];
        double Y = Giradas[cont + 1];
        double Z = Giradas[cont + 2];

        //Convierte a coordenadas planas
        double Xp = ZPersona * X / (ZPersona - Z);
        double Yp = ZPersona * Y / (ZPersona - Z);

        //Agrega las coordenadas planas a la lista
        PlanoX.Add(Xp);
        PlanoY.Add(Yp);
    }
}

//Convierte las coordenadas planas en coordenadas de pantalla
public void CuadraPantalla(int XpIni, int YpIni,
                           int XpFin, int YpFin) {
    //Los valores extremos de las coordenadas del cubo
    double maximoX = 0.785674201318386;
    double minimoX = -0.785674201318386;
    double maximoY = 0.785674201318386;
    double minimoY = -0.785674201318386;

    //Las constantes de transformación
    double conX = (XpFin - XpIni) / (maximoX - minimoX);
    double conY = (YpFin - YpIni) / (maximoY - minimoY);

    //Deduce las coordenadas de pantalla
    pX.Clear();
    pY.Clear();
    for (int cont = 0; cont < PlanoX.Count; cont++) {
        double Xpant = conX * (PlanoX[cont] - minimoX) + XpIni;
        double Ypant = conY * (PlanoY[cont] - minimoY) + YpIni;
        pX.Add(Convert.ToInt32(Xpant));
        pY.Add(Convert.ToInt32(Ypant));
    }
}
}

```

```
//Dibuja el cubo
public void Dibuja(Graphics lienzo, Pen lapiz) {
    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[1], pY[1]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[2], pY[2]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[3], pY[3]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[0], pY[0]);

    lienzo.DrawLine(lapiz, pX[4], pY[4], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[5], pY[5], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[6], pY[6], pX[7], pY[7]);
    lienzo.DrawLine(lapiz, pX[7], pY[7], pX[4], pY[4]);

    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[4], pY[4]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[7], pY[7]);
}
}
```

Ejemplo de ejecución:

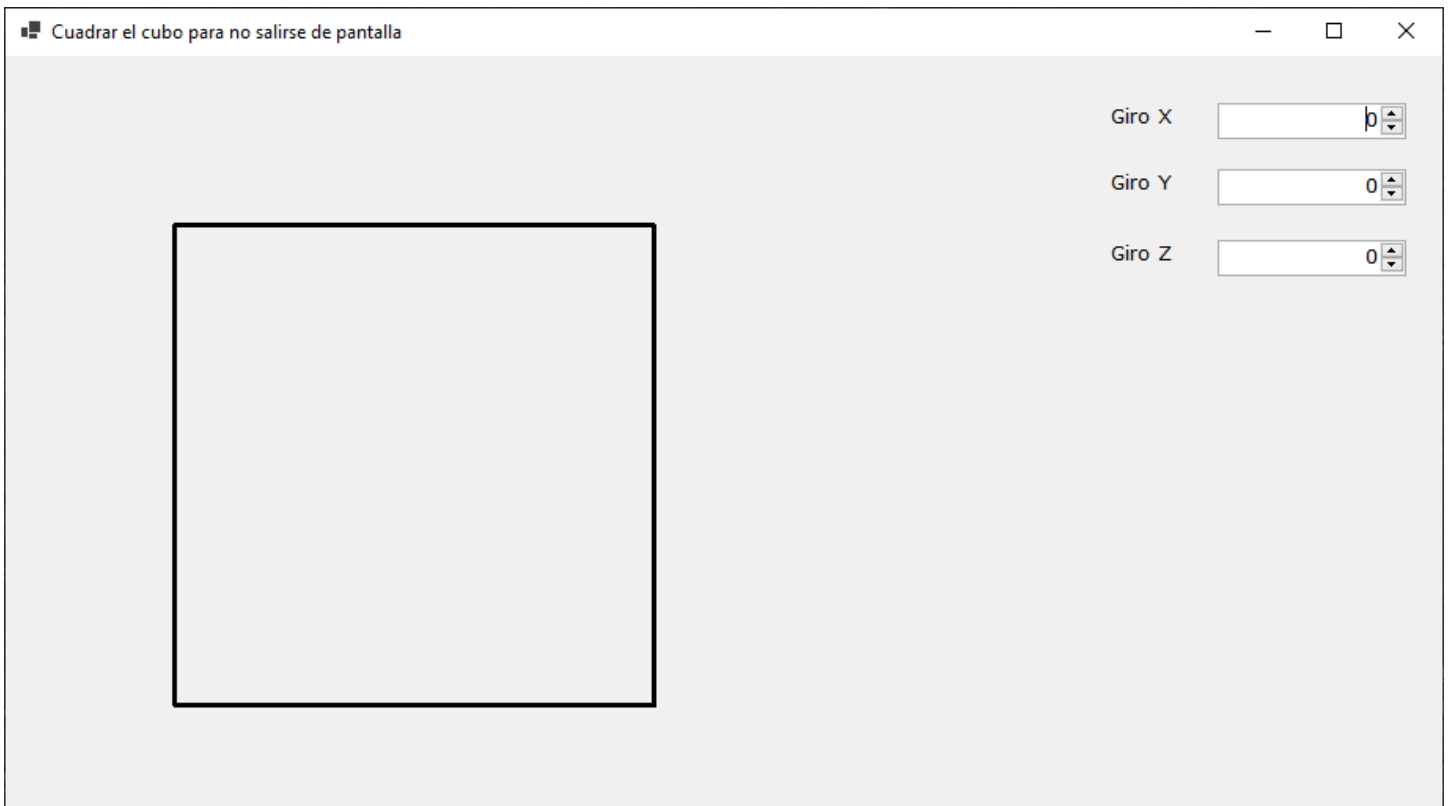


Ilustración 11: Cubo proyectado por defecto

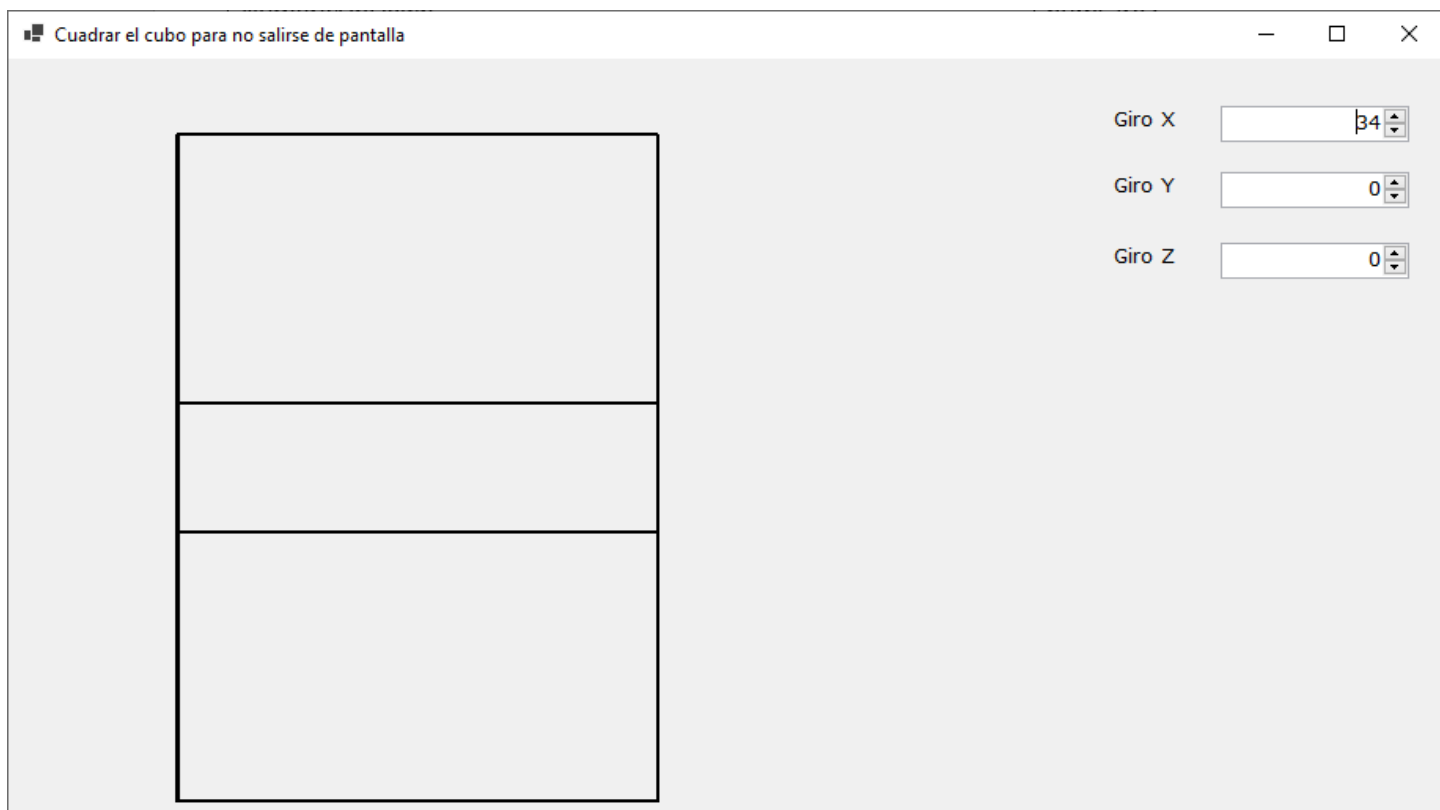


Ilustración 12: Cubo proyectado con giro en X

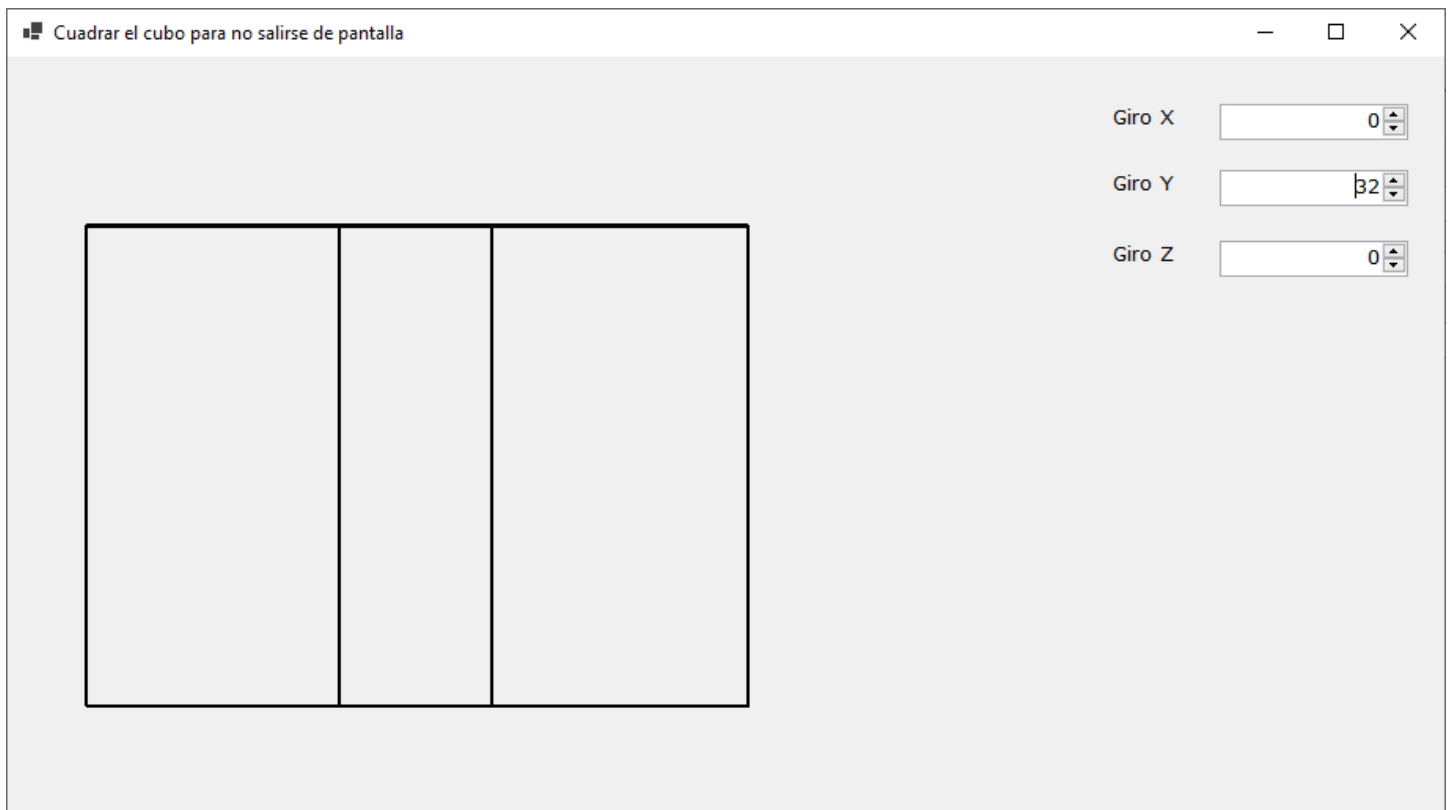


Ilustración 13: Cubo proyectado con giro en Y

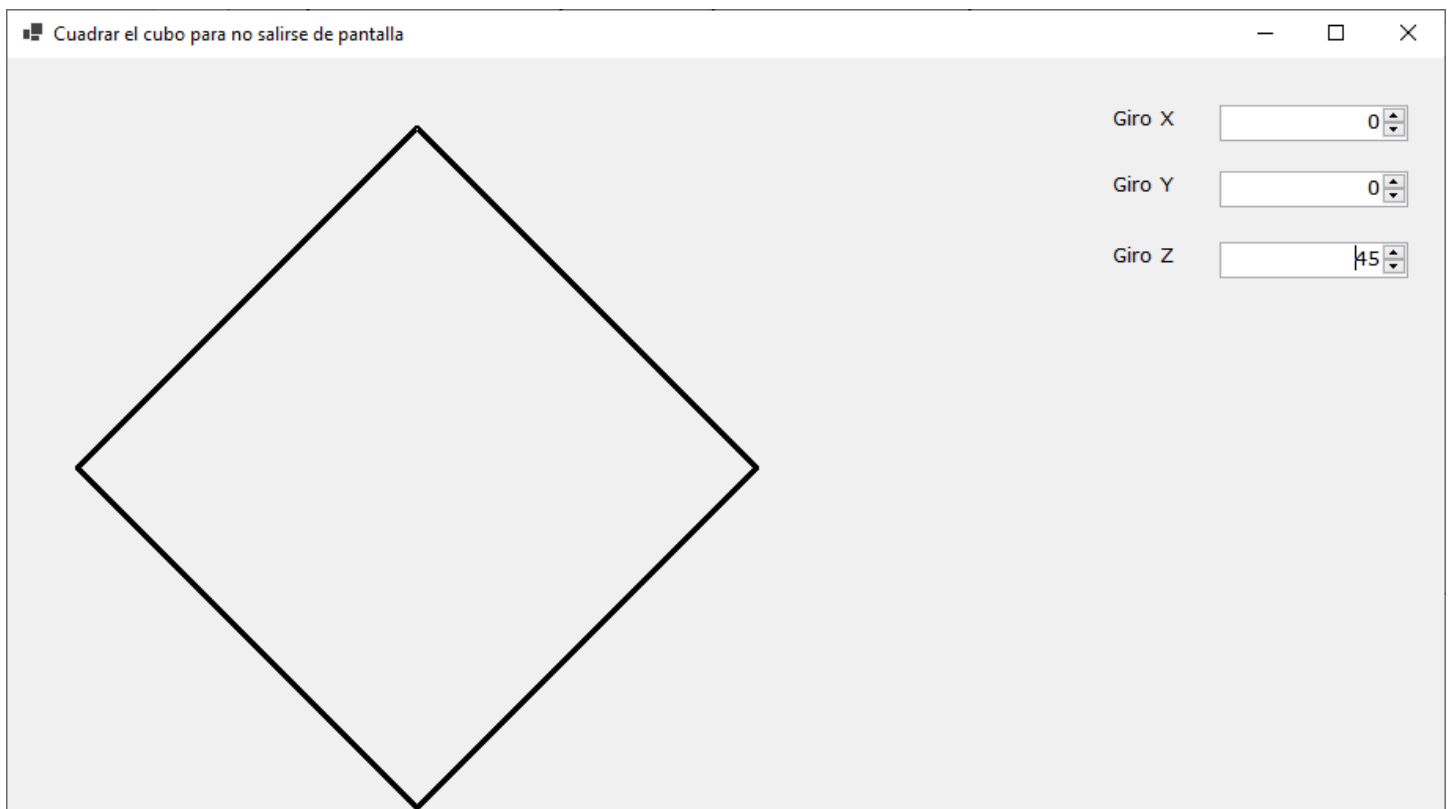


Ilustración 14: Cubo proyectado con giro en Z

Combinando los tres giros: Cálculo de constantes

En el programa anterior, sólo se puede girar en un ángulo, no hay forma de girarlo en los tres ángulos. ¿Cómo lograr el giro en los tres ángulos? La respuesta está en multiplicar las tres matrices de giro y con la matriz resultante se procede a hacer los giros.

$$\text{Cos}X = \cos(\text{ang}X)$$

$$\text{Sen}X = \sin(\text{ang}X)$$

$$\text{Cos}Y = \cos(\text{ang}Y)$$

$$\text{Sen}Y = \sin(\text{ang}Y)$$

$$\text{Cos}Z = \cos(\text{ang}Z)$$

$$\text{Sen}Z = \sin(\text{ang}Z)$$

$$\begin{bmatrix} X_{giro} \\ Y_{giro} \\ Z_{giro} \end{bmatrix} = \begin{bmatrix} \text{Cos}Y * \text{Cos}Z & -\text{Cos}X * \text{Sin}Z + \text{Sin}X * \text{Sin}Y * \text{Cos}Z & \text{Sin}X * \text{Sin}Z + \text{Cos}X * \text{Sin}Y * \text{Cos}Z \\ \text{Cos}Y * \text{Sin}Z & \text{Cos}X * \text{Cos}Z + \text{Sin}X * \text{Sin}Y * \text{Sin}Z & -\text{Sin}X * \text{Cos}Z + \text{Cos}X * \text{Sin}Y * \text{Sin}Z \\ -\text{Sin}Y & \text{Sin}X * \text{Cos}Y & \text{Cos}X * \text{Cos}Y \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Los pasos para dibujar el cubo fueron los siguientes:

Paso 1: Coordenadas del cubo, donde su centro esté en la posición 0, 0, 0. Las coordenadas van de -0.5 a 0.5, luego cada lado mide 1. Cada coordenada tiene valores posX, posY, posZ

Paso 2: Se gira cada coordenada del cubo usando la matriz de giro XYZ. Cada coordenada girada es posXg, posYg, posZg.

Paso 3: Se proyecta la coordenada girada a un plano XY, el valor Z queda en cero. Esa coordenada plana es planoX, planoY

El pasar el cubo proyectado a la pantalla física es el que requiere más cuidado porque hay que cuadrarlo considerando la distancia del observador y los diferentes giros que puede hacer el cubo. El problema es que el cubo se puede ver bien en la posición predeterminada (ángulos en 0,0,0), pero al girarlo, se salen los vértices de los límites de la pantalla o ventana, luego hay que ajustar la conversión de plano a pantalla de tal manera que por más que lo gire en cualquier ángulo, no se salga el dibujo de la ventana. Para lograr eso, se sabe que el cubo tiene coordenadas de -0.5 a 0.5 (es decir de lado=1) tanto en X, Y, y Z. Se deja fija la distancia del observador, en este caso con un valor de 5 (suficientemente alejado de la figura). Con esos valores entonces se probaron todos los ángulos de giro en X de 0 a 360 grados, en Y de 0 a 360 grados y en Z de 0 a 360 grados, así:

Desde anguloX = 0 grados hasta 360 grados

Desde anguloY = 0 grados hasta 360 grados

Desde anguloZ = 0 grados hasta 360 grados

Eso significa que se probaron $361 \times 361 \times 361 = 47.045.881$ situaciones posibles. Un número muy alto que tomó tiempo en hacerlo (depende de la velocidad del computador) para saber las coordenadas máximas planas al girarlo en todos esos ángulos.

Esta es la implementación:

M/005.cs

```
namespace Ejemplo {

    internal class Cubo {
        //Un cubo tiene 8 coordenadas
        //espaciales X, Y, Z
        private List<double> Coordenadas;

        //Un cubo tiene 8 coordenadas
        //espaciales X, Y, Z que han sido giradas
        private List<double> Giradas;

        //Luego tiene 8 coordenadas planas
        private List<double> PlanoX;
        private List<double> PlanoY;

        //Constructor
        public Cubo() {
            //Ejemplo de coordenadas
            //espaciales X,Y,Z
            Coordenadas = [
                -0.5, -0.5, -0.5,
                0.5, -0.5, -0.5,
                0.5, 0.5, -0.5,
                -0.5, 0.5, -0.5,
                -0.5, -0.5, 0.5,
                0.5, -0.5, 0.5,
                0.5, 0.5, 0.5,
                -0.5, 0.5, 0.5 ];

            Giradas = [];
            PlanoX = [];
            PlanoY = [];
        }

        public void GirarFigura(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;
```

```

double CosX = Math.Cos(angXr);
double SinX = Math.Sin(angXr);
double CosY = Math.Cos(angYr);
double SinY = Math.Sin(angYr);
double CosZ = Math.Cos(angZr);
double SinZ = Math.Sin(angZr);

//Matriz de Rotación

//https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
double[,] Mt = new double[3, 3] {
{CosY*CosZ,-CosX*SinZ+SinX*SinY*CosZ,SinX*SinZ+CosX*SinY*CosZ},
{CosY*SinZ,CosX*CosZ+SinX*SinY*SinZ,-SinX*CosZ+CosX*SinY*SinZ},
{-SinY,SinX*CosY,CosX*CosY}
};

Giradas.Clear();

//Gira las 8 coordenadas
for (int cont = 0; cont < Coordenadas.Count; cont += 3) {
    //Extrae las coordenadas espaciales
    double X = Coordenadas[cont];
    double Y = Coordenadas[cont + 1];
    double Z = Coordenadas[cont + 2];

    //Hace el giro
    double Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
    double Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
    double Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];

    //Guarda en la lista de giro
    Giradas.Add(Xg);
    Giradas.Add(Yg);
    Giradas.Add(Zg);
}
}

//Convierte de 3D a 2D las coordenadas giradas
private void Convierte3Da2D(int ZPersona) {
    PlanoX.Clear();
    PlanoY.Clear();
    for (int cont = 0; cont < Giradas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Giradas[cont];
        double Y = Giradas[cont + 1];
        double Z = Giradas[cont + 2];
    }
}

```

```

        //Convierte a coordenadas planas
        double Xp = ZPersona * X / (ZPersona - Z);
        double Yp = ZPersona * Y / (ZPersona - Z);

        //Agrega las coordenadas planas a la lista
        PlanoX.Add(Xp);
        PlanoY.Add(Yp);
    }
}

//Calcula los extremos de las coordenadas
//del cubo al girar y proyectarse
public void CalculaExtremo(int ZPersona) {
    double maximoX = double.MinValue;
    double minimoX = double.MaxValue;
    double maximoY = double.MinValue;
    double minimoY = double.MaxValue;

    for (double angX = 0; angX <= 360; angX++) {
        for (double angY = 0; angY <= 360; angY++) {
            for (double angZ = 0; angZ <= 360; angZ++) {
                GirarFigura(angX, angY, angZ);
                Convierte3Da2D(ZPersona);

                for (int cont = 0; cont < PlanoX.Count; cont++) {
                    if (PlanoX[cont] < minimoX)
                        minimoX = PlanoX[cont];

                    if (PlanoX[cont] > maximoX)
                        maximoX = PlanoX[cont];

                    if (PlanoY[cont] < minimoY)
                        minimoY = PlanoY[cont];

                    if (PlanoY[cont] > maximoY)
                        maximoY = PlanoY[cont];
                }
            }
        }
    }

    Console.WriteLine("MinimoX: " + minimoX);
    Console.WriteLine("MaximoX: " + maximoX);
    Console.WriteLine("MinimoY: " + minimoY);
    Console.WriteLine("MaximoY: " + maximoY);
}
}

```

```
class Program {  
    static void Main() {  
        Cubo Figura3D = new();  
        Figura3D.CalculaExtremo(5);  
    }  
}
```

MinimoX: -0,8793154376917781
MaximoX: 0,8793154376917781
MinimoY: -0,8793153987523792
MaximoY: 0,8793153987523792

Ilustración 15: Constantes para cuadrar el cubo en la pantalla

Esas constantes son las que se usan para generar las coordenadas en pantalla física.

Nota 1: Si cambia la distancia del observador al plano o cambia el tamaño de la figura 3D hay que recalcular nuevas constantes.

Nota 2: Esas constantes se usarán en futuras implementaciones.

Combinando los tres giros: Uso de constantes

Como es una aplicación gráfica de escritorio, entonces así debe ser la ventana:

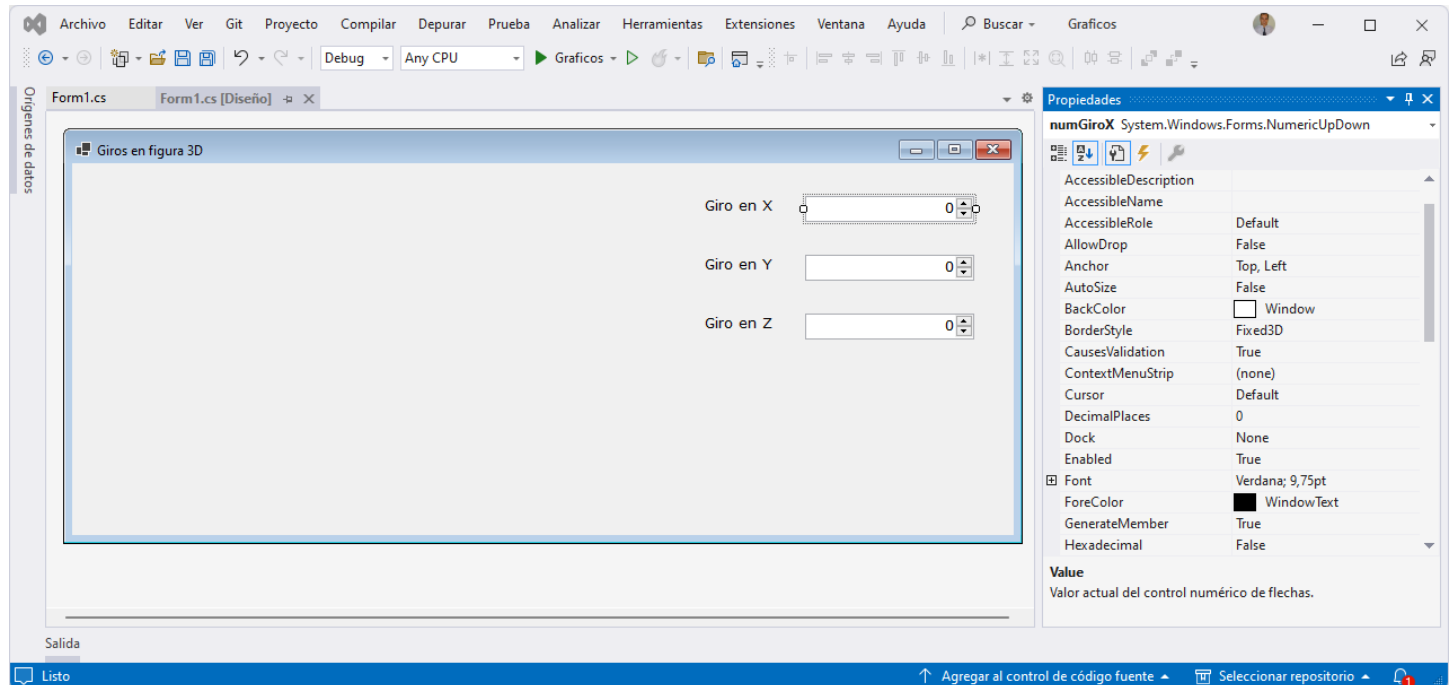


Ilustración 16: Diseño de ventana

Se utilizan los siguientes controles:

Tipo	Nombre
Label	IblGiroX
Label	IblGiroY
Label	IblGiroZ
NumericUpDown	numGiroX
NumericUpDown	numGiroY
NumericUpDown	numGiroZ

```

namespace Graficos {

    public partial class Form1 : Form {
        //El cubo que se proyecta y gira
        Cubo Figura3D;

        public Form1() {
            InitializeComponent();
            Figura3D = new Cubo();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapiz = new(Color.Black, 1);

            int AnguloX = Convert.ToInt32(numGiroX.Value);
            int AnguloY = Convert.ToInt32(numGiroY.Value);
            int AnguloZ = Convert.ToInt32(numGiroZ.Value);

            int ZPersona = 5;
            Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ);
            Figura3D.Convierte3Da2D(ZPersona);
            Figura3D.CuadraPantalla(20, 20, 500, 500);
            Figura3D.Dibuja(Lienzo, Lapiz);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }
    }

    internal class Cubo {
        //Un cubo tiene 8 coordenadas
        //espaciales X, Y, Z
        private List<double> Coordenadas;

        //Un cubo tiene 8 coordenadas
        //espaciales X, Y, Z que han sido giradas
        private List<double> Giradas;
    }
}

```



```

//Luego tiene 8 coordenadas planas
private List<double> PlanoX;
private List<double> PlanoY;

//Luego tiene 8 coordenadas de pantalla
private List<int> pX;
private List<int> pY;

//Constructor
public Cubo() {
    //Ejemplo de coordenadas espaciales X,Y,Z
    Coordenadas = new List<double>() {
        -0.5, -0.5, -0.5,
        0.5, -0.5, -0.5,
        0.5, 0.5, -0.5,
        -0.5, 0.5, -0.5,
        -0.5, -0.5, 0.5,
        0.5, -0.5, 0.5,
        0.5, 0.5, 0.5,
        -0.5, 0.5, 0.5 };

    Giradas = [];
    PlanoX = [];
    PlanoY = [];
    pX = [];
    pY = [];
}

public void GirarFigura(double angX, double angY, double angZ) {
    double CosX = Math.Cos(angX * Math.PI / 180);
    double SinX = Math.Sin(angX * Math.PI / 180);
    double CosY = Math.Cos(angY * Math.PI / 180);
    double SinY = Math.Sin(angY * Math.PI / 180);
    double CosZ = Math.Cos(angZ * Math.PI / 180);
    double SinZ = Math.Sin(angZ * Math.PI / 180);

    //Matriz de Rotación

    //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
    double[,] Mt = new double[3, 3] {
        {CosY*CosZ,-CosX*SinZ+SinX*SinY*CosZ,SinX*SinZ+CosX*SinY*CosZ},
        {CosY*SinZ,CosX*CosZ+SinX*SinY*SinZ,-SinX*CosZ+CosX*SinY*SinZ},
        {-SinY,SinX*CosY,CosX*CosY}
    };

    Giradas.Clear();
}

```

```

//Gira las 8 coordenadas
for (int cont = 0; cont < Coordenadas.Count; cont += 3) {
    //Extrae las coordenadas espaciales
    double X = Coordenadas[cont];
    double Y = Coordenadas[cont + 1];
    double Z = Coordenadas[cont + 2];

    //Hace el giro
    double Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
    double Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
    double Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];

    //Guarda en la lista de giro
    Giradas.Add(Xg);
    Giradas.Add(Yg);
    Giradas.Add(Zg);
}
}

//Convierte de 3D a 2D las coordenadas giradas
public void Convierte3Da2D(int ZPersona) {
    PlanoX.Clear();
    PlanoY.Clear();

    for (int cont = 0; cont < Giradas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        double X = Giradas[cont];
        double Y = Giradas[cont + 1];
        double Z = Giradas[cont + 2];

        //Convierte a coordenadas planas
        double Xp = ZPersona * X / (ZPersona - Z);
        double Yp = ZPersona * Y / (ZPersona - Z);

        //Agrega las coordenadas planas a la lista
        PlanoX.Add(Xp);
        PlanoY.Add(Yp);
    }
}

//Convierte las coordenadas planas en coordenadas de pantalla
public void CuadraPantalla(int XpIni, int YpIni,
                           int XpFin, int YpFin) {
    //Los valores extremos de las coordenadas del cubo
    double maximoX = 0.87931543769177811;
    double minimoX = -0.87931543769177811;
    double maximoY = 0.87931539875237918;
    double minimoY = -0.87931539875237918;

```

```

//Las constantes de transformación
double conX = (XpFin - XpIni) / (maximoX - minimoX);
double convY = (YpFin - YpIni) / (maximoY - minimoY);

//Deduce las coordenadas de pantalla
pX.Clear();
pY.Clear();
for (int cont = 0; cont < PlanoX.Count; cont++) {
    double Xpant = conX * (PlanoX[cont] - minimoX) + XpIni;
    double Ypant = convY * (PlanoY[cont] - minimoY) + YpIni;
    pX.Add(Convert.ToInt32(Xpant));
    pY.Add(Convert.ToInt32(Ypant));
}
}

//Dibuja el cubo
public void Dibuja(Graphics lienzo, Pen lapiz) {
    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[1], pY[1]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[2], pY[2]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[3], pY[3]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[0], pY[0]);

    lienzo.DrawLine(lapiz, pX[4], pY[4], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[5], pY[5], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[6], pY[6], pX[7], pY[7]);
    lienzo.DrawLine(lapiz, pX[7], pY[7], pX[4], pY[4]);

    lienzo.DrawLine(lapiz, pX[0], pY[0], pX[4], pY[4]);
    lienzo.DrawLine(lapiz, pX[1], pY[1], pX[5], pY[5]);
    lienzo.DrawLine(lapiz, pX[2], pY[2], pX[6], pY[6]);
    lienzo.DrawLine(lapiz, pX[3], pY[3], pX[7], pY[7]);
}
}
}

```

Ejemplo de ejecución:

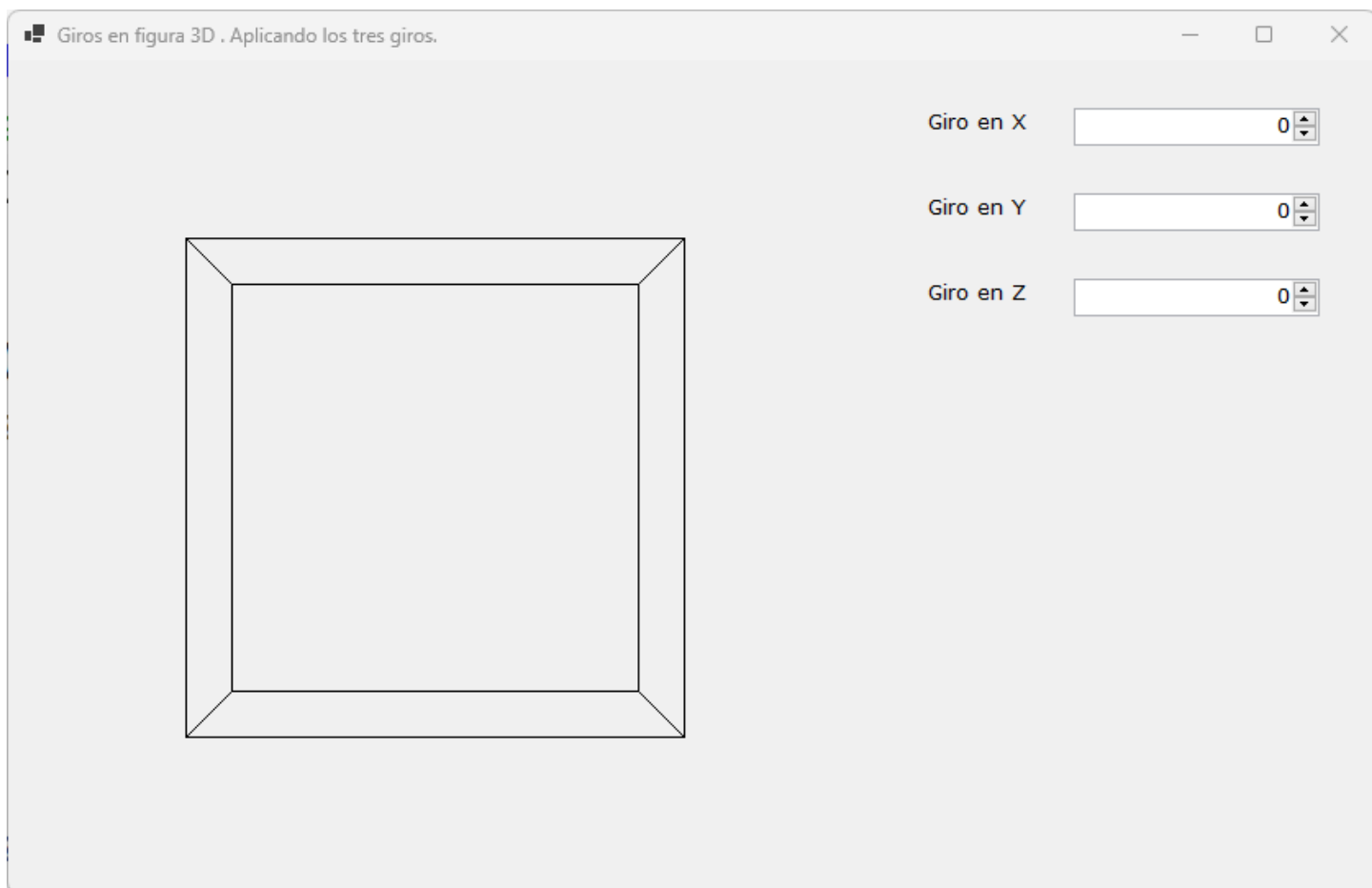


Ilustración 17: Cubo proyectado, sin giros

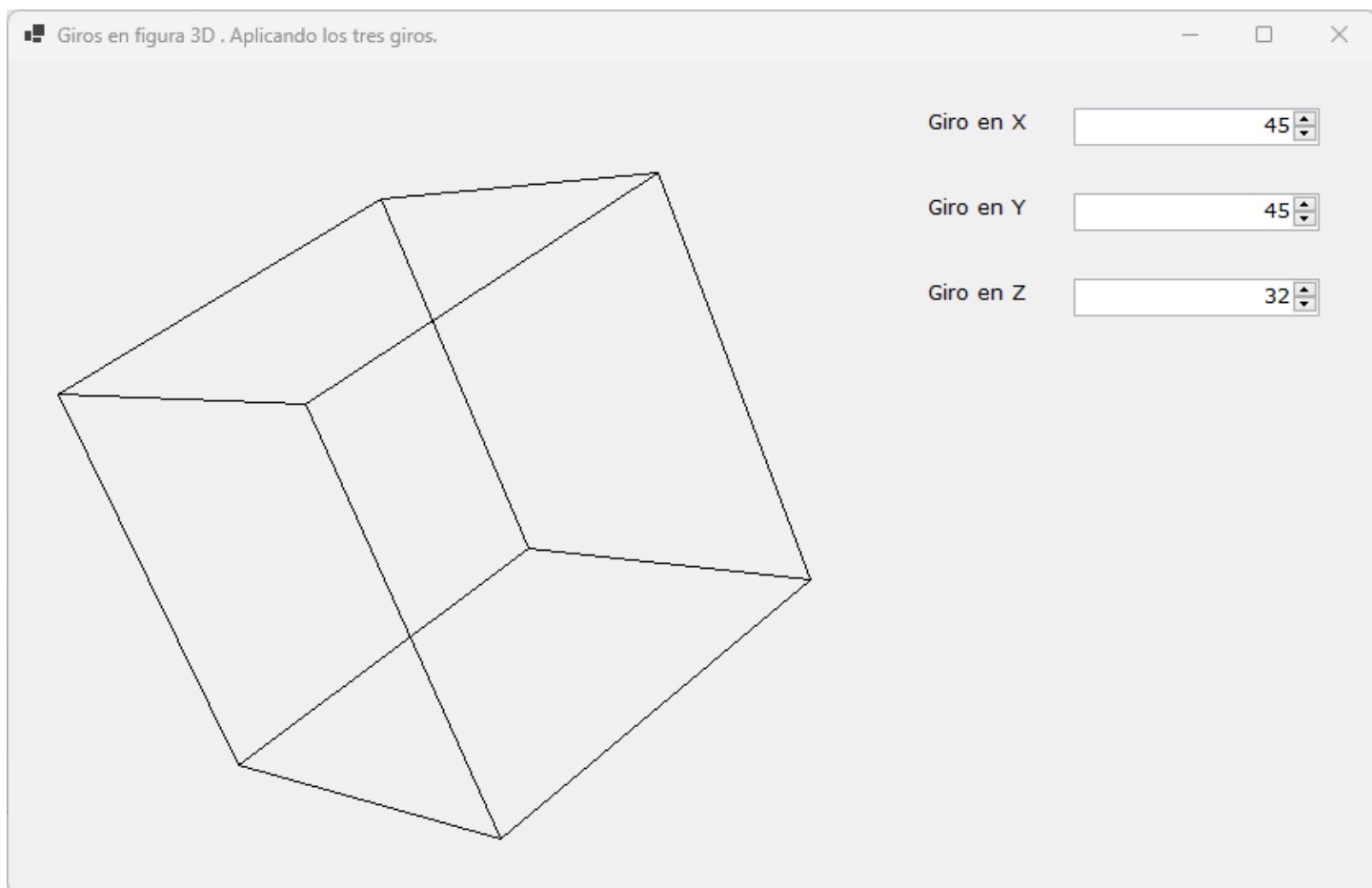


Ilustración 18: Cubo proyectado con giros

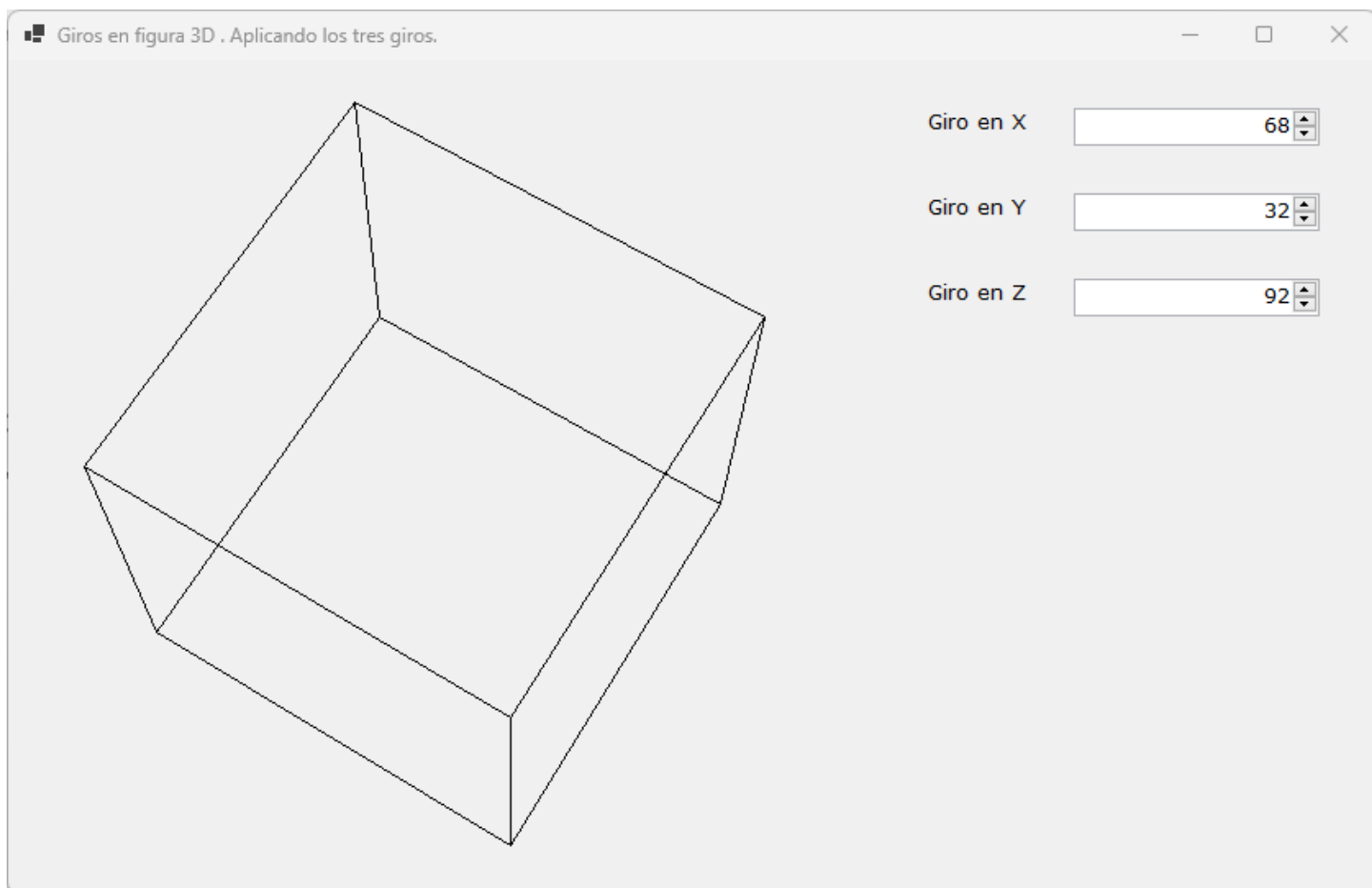


Ilustración 19: Cubo proyectado con giros

Líneas ocultas

El problema con los ejemplos anteriores de giros es nuestra vista, en vez de interpretar que el cubo gira, lo que a veces vemos es que la figura se distorsiona y es porque el cubo hecho de líneas verticales y horizontales nos confunde, y no sabemos que está atrás y que está adelante. Para evitar esto, se hace un cambio y es dibujar el cubo no con líneas sino con polígonos [2]. Un cubo tiene seis caras, luego serían seis polígonos para dibujarlo. Estos serían los pasos:

1. Escribir las cuatro coordenadas XYZ que servirán para componer cada uno de los seis polígonos.
2. Girar cada coordenada XYZ de cada polígono.
3. Determinar el valor Z promedio de cada polígono.
4. Ordenar los polígonos del Z promedio más pequeño al más grande. Eso nos daría que polígono está más lejos y cuál más cerca al observador.

Dibujar en ese orden cada polígono, primero rellenando el área del polígono con el color del fondo y luego dibujando el perímetro con algún color visible.

M/007.cs

```
namespace Graficos {
    public partial class Form1 : Form {
        //El cubo que se proyecta y gira
        Cubo Figura3D;

        public Form1() {
            InitializeComponent();
            Figura3D = new Cubo();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapiz = new(Color.Black, 1);
            Brush Relleno = new SolidBrush(Color.White);

            int AnguloX = Convert.ToInt32(numGiroX.Value);
            int AnguloY = Convert.ToInt32(numGiroY.Value);
            int AnguloZ = Convert.ToInt32(numGiroZ.Value);

            int ZPersona = 5;
            Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ);
            Figura3D.Convierte3Da2D(ZPersona);
            Figura3D.CuadraPantalla(20, 20, 500, 500);
            Figura3D.Dibuja(Lienzo, Lapiz, Relleno);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }
    }
}
```

```

private void numGiroY_ValueChanged(object sender, EventArgs e) {
    Refresh();
}

private void numGiroZ_ValueChanged(object sender, EventArgs e) {
    Refresh();
}
}

internal class Cubo {
    //Un cubo tiene 8 coordenadas
    //espaciales X, Y, Z
    private List<Poligono> poligono;

    //Constructor
    public Cubo() {
        double P = -0.5;
        double Q = 0.5;
        poligono =
        [
            new Poligono(P, Q, P, Q, Q, P, Q, P, P, P, P, P, P),
            new Poligono(P, Q, Q, Q, Q, Q, Q, P, Q, P, P, Q),
            new Poligono(P, Q, P, P, Q, Q, P, P, Q, P, P, P),
            new Poligono(Q, Q, P, Q, Q, Q, Q, P, Q, Q, P, P),
            new Poligono(P, Q, P, Q, Q, P, Q, Q, Q, P, Q, Q),
            new Poligono(P, P, P, Q, P, P, Q, P, Q, P, P, Q),
        ];
    }

    public void GirarFigura(double angX, double angY, double angZ) {
        double CosX = Math.Cos(angX * Math.PI / 180);
        double SinX = Math.Sin(angX * Math.PI / 180);
        double CosY = Math.Cos(angY * Math.PI / 180);
        double SinY = Math.Sin(angY * Math.PI / 180);
        double CosZ = Math.Cos(angZ * Math.PI / 180);
        double SinZ = Math.Sin(angZ * Math.PI / 180);

        //Matriz de Rotación

        //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
        double[, ] Mt = new double[3, 3] {
            {CosY*CosZ, -CosX*SinZ+SinX*SinY*CosZ, SinX*SinZ+CosX*SinY*CosZ},
            {CosY*SinZ, CosX*CosZ+SinX*SinY*SinZ, -SinX*CosZ+CosX*SinY*SinZ},
            {-SinY, SinX*CosY, CosX*CosY}
        };

        //Gira los 8 polígonos
    }
}

```



```

        for (int cont = 0; cont < poligono.Count; cont++) {
            poligono[cont].Girar(Mt);
        }
    }

    //Convierte de 3D a 2D las coordenadas giradas
    public void Convierte3Da2D(int ZPersona) {
        for (int cont = 0; cont < poligono.Count; cont++) {
            poligono[cont].Convierte3Da2D(ZPersona);
        }
    }

    //Convierte las coordenadas planas en coordenadas de pantalla
    public void CuadraPantalla(int XpIni, int YpIni,
                               int XpFin, int YpFin) {
        //Los valores extremos de las coordenadas del cubo
        double MaximoX = 0.87931543769177811;
        double MinimoX = -0.87931543769177811;
        double MaximoY = 0.87931539875237918;
        double MinimoY = -0.87931539875237918;

        //Las constantes de transformación
        double conX = (XpFin - XpIni) / (MaximoX - MinimoX);
        double conY = (YpFin - YpIni) / (MaximoY - MinimoY);

        for (int cont = 0; cont < poligono.Count; cont++) {
            poligono[cont].CuadraPantalla(conX, conY,
                                           MinimoX, MinimoY,
                                           XpIni, YpIni);
        }

        //Ordena del polígono más alejado al más cercano,
        //de esa manera los polígonos de adelante
        //son visibles y los de atrás son borrados.
        poligono.Sort();
    }

    //Dibuja el cubo
    public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
        for (int cont = 0; cont < poligono.Count; cont++) {
            poligono[cont].Dibuja(lienzo, lapiz, relleno);
        }
    }
}

internal class Poligono : IComparable {
    //Un polígono son cuatro(4) coordenadas espaciales

```

```

public double X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4;

//Las coordenadas de giro
public double X1g, Y1g, Z1g;
public double X2g, Y2g, Z2g;
public double X3g, Y3g, Z3g;
public double X4g, Y4g, Z4g;

//Las coordenadas planas (segunda dimensión)
public double X1sd, Y1sd, X2sd, Y2sd, X3sd, Y3sd, X4sd, Y4sd;

//Las coordenadas en pantalla
public int X1p, Y1p, X2p, Y2p, X3p, Y3p, X4p, Y4p;

//Centro del polígono
public double Centro;

public Poligono(double X1, double Y1, double Z1,
                double X2, double Y2, double Z2,
                double X3, double Y3, double Z3,
                double X4, double Y4, double Z4) {
    this.X1 = X1; this.Y1 = Y1; this.Z1 = Z1;
    this.X2 = X2; this.Y2 = Y2; this.Z2 = Z2;
    this.X3 = X3; this.Y3 = Y3; this.Z3 = Z3;
    this.X4 = X4; this.Y4 = Y4; this.Z4 = Z4;
}

//Gira en XYZ
public void Girar(double[,] Mt) {
    X1g = X1 * Mt[0, 0] + Y1 * Mt[1, 0] + Z1 * Mt[2, 0];
    Y1g = X1 * Mt[0, 1] + Y1 * Mt[1, 1] + Z1 * Mt[2, 1];
    Z1g = X1 * Mt[0, 2] + Y1 * Mt[1, 2] + Z1 * Mt[2, 2];

    X2g = X2 * Mt[0, 0] + Y2 * Mt[1, 0] + Z2 * Mt[2, 0];
    Y2g = X2 * Mt[0, 1] + Y2 * Mt[1, 1] + Z2 * Mt[2, 1];
    Z2g = X2 * Mt[0, 2] + Y2 * Mt[1, 2] + Z2 * Mt[2, 2];

    X3g = X3 * Mt[0, 0] + Y3 * Mt[1, 0] + Z3 * Mt[2, 0];
    Y3g = X3 * Mt[0, 1] + Y3 * Mt[1, 1] + Z3 * Mt[2, 1];
    Z3g = X3 * Mt[0, 2] + Y3 * Mt[1, 2] + Z3 * Mt[2, 2];

    X4g = X4 * Mt[0, 0] + Y4 * Mt[1, 0] + Z4 * Mt[2, 0];
    Y4g = X4 * Mt[0, 1] + Y4 * Mt[1, 1] + Z4 * Mt[2, 1];
    Z4g = X4 * Mt[0, 2] + Y4 * Mt[1, 2] + Z4 * Mt[2, 2];

    Centro = (Z1g + Z2g + Z3g + Z4g) / 4;
}

```

```

//Convierte de 3D a 2D (segunda dimensión)
public void Convierte3Da2D(double ZPersona) {
    X1sd = X1g * ZPersona / (ZPersona - Z1g);
    Y1sd = Y1g * ZPersona / (ZPersona - Z1g);

    X2sd = X2g * ZPersona / (ZPersona - Z2g);
    Y2sd = Y2g * ZPersona / (ZPersona - Z2g);

    X3sd = X3g * ZPersona / (ZPersona - Z3g);
    Y3sd = Y3g * ZPersona / (ZPersona - Z3g);

    X4sd = X4g * ZPersona / (ZPersona - Z4g);
    Y4sd = Y4g * ZPersona / (ZPersona - Z4g);
}

//Cuadra en pantalla física
public void CuadraPantalla(double conX, double conY,
    double MinimoX, double MinimoY,
    int XPIni, int YPIni) {
    X1p = Convert.ToInt32(conX * (X1sd - MinimoX) + XPIni);
    Y1p = Convert.ToInt32(conY * (Y1sd - MinimoY) + YPIni);

    X2p = Convert.ToInt32(conX * (X2sd - MinimoX) + XPIni);
    Y2p = Convert.ToInt32(conY * (Y2sd - MinimoY) + YPIni);

    X3p = Convert.ToInt32(conX * (X3sd - MinimoX) + XPIni);
    Y3p = Convert.ToInt32(conY * (Y3sd - MinimoY) + YPIni);

    X4p = Convert.ToInt32(conX * (X4sd - MinimoX) + XPIni);
    Y4p = Convert.ToInt32(conY * (Y4sd - MinimoY) + YPIni);
}

//Hace el gráfico del polígono
public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
    //Pone un color de fondo al polígono
    //para borrar lo que hay detrás
    Point Punto1 = new(X1p, Y1p);
    Point Punto2 = new(X2p, Y2p);
    Point Punto3 = new(X3p, Y3p);
    Point Punto4 = new(X4p, Y4p);
    Point[] poligono = [Punto1, Punto2, Punto3, Punto4];
    Lienzo.FillPolygon(Relleno, poligono);
    Lienzo.DrawPolygon(Lapiz, poligono);
}

//Usado para ordenar los polígonos
//del más lejano al más cercano

```

```
//https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-
a-property-in-the-object
public int CompareTo(object obj) {
    Poligono Comparar = obj as Poligono;
    if (Comparar.Centro < Centro) return 1;
    if (Comparar.Centro > Centro) return -1;
    return 0;
}
}
```

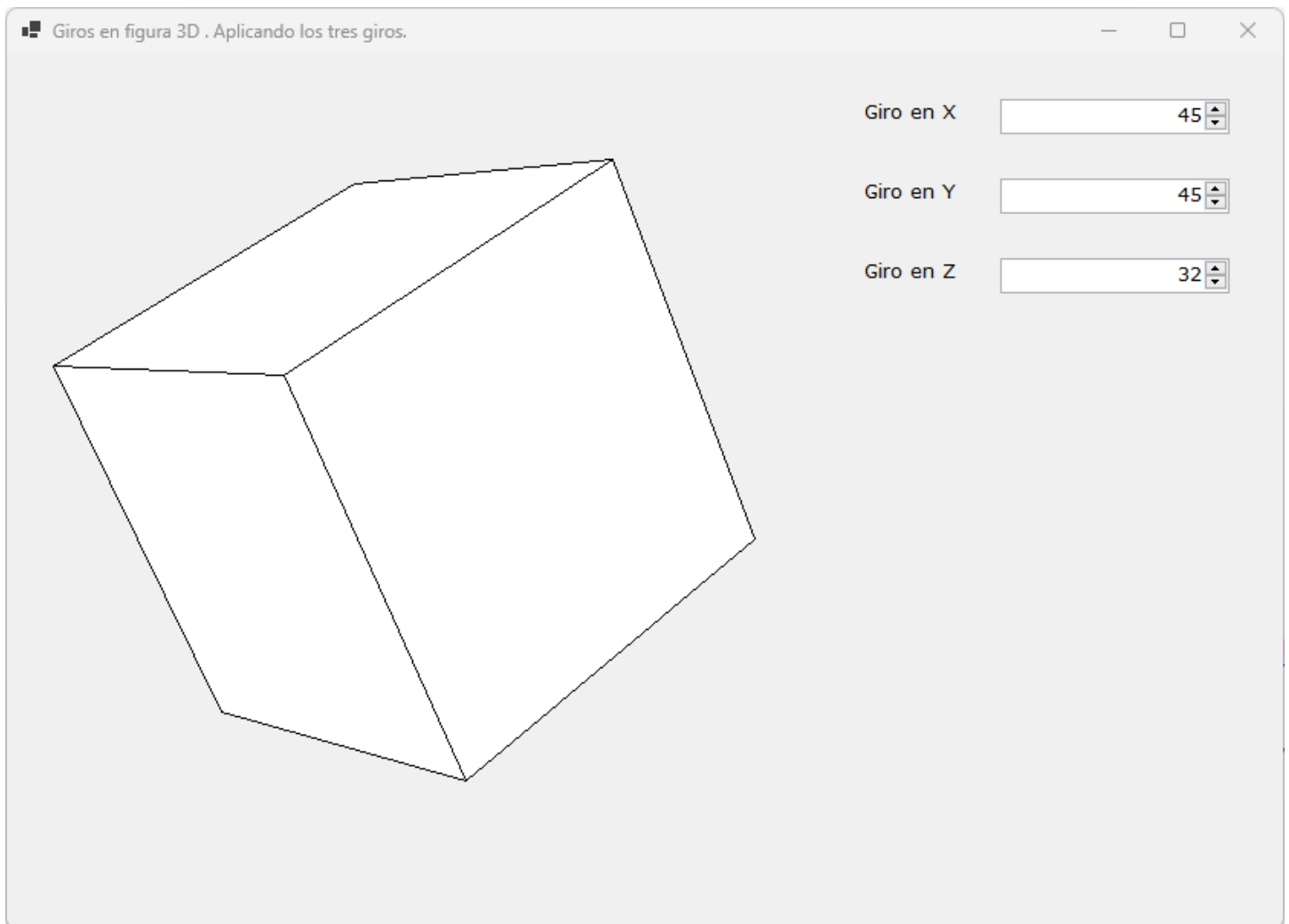


Ilustración 20: Líneas ocultas

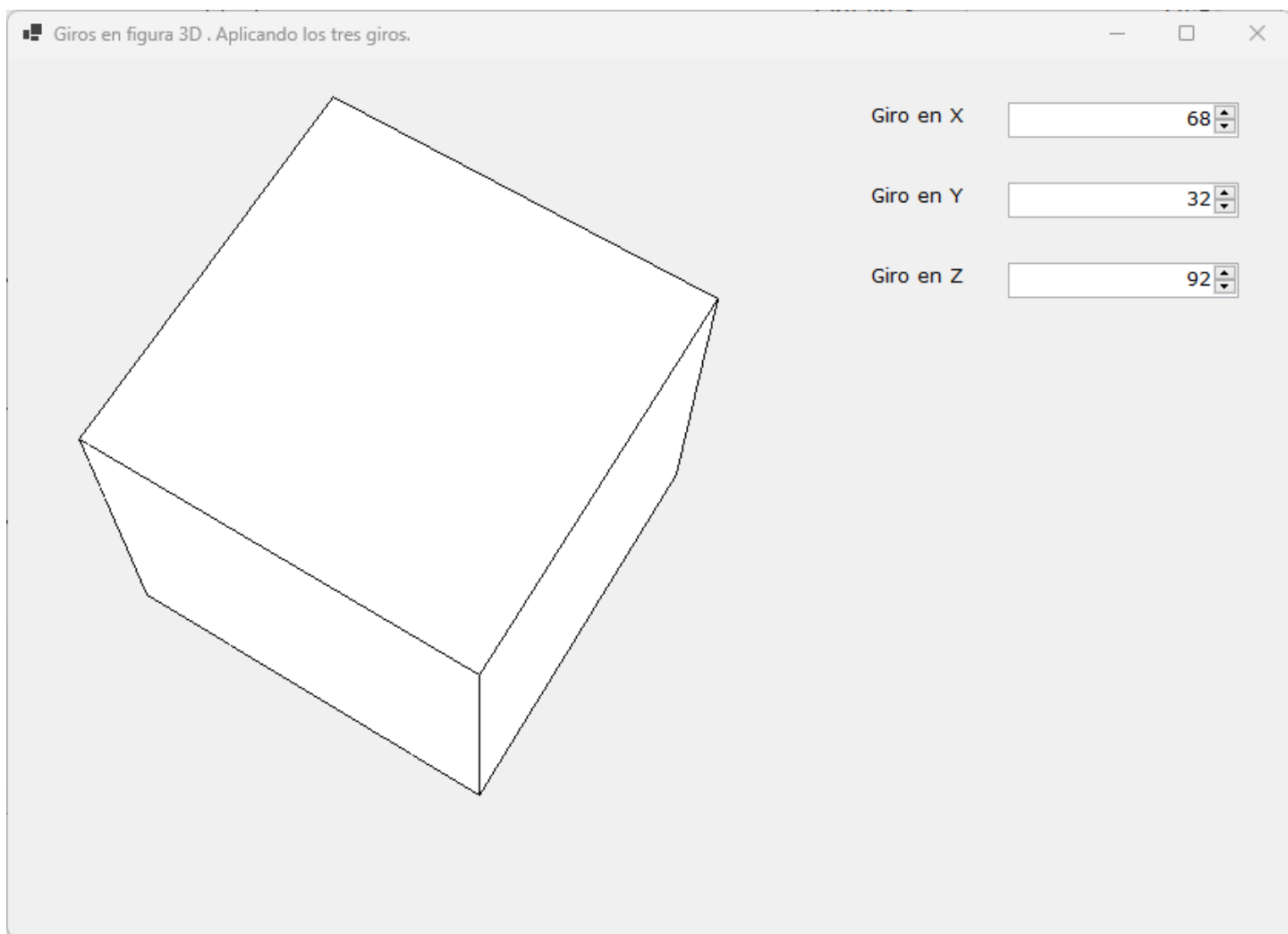


Ilustración 21: Líneas ocultas

Gráfico Matemático en 3D

Cuando tenemos una ecuación del tipo $Z = F(X,Y)$, tenemos una ecuación con dos variables independientes y una variable dependiente. Su representación es en 3D. Por ejemplo:

$$Z = \sqrt[2]{X^2 + Y^2} + 3 * \text{Cos}(\sqrt[2]{X^2 + Y^2}) + 5$$

Los pasos para hacer el gráfico son los siguientes:

Paso 1: Saber el valor donde inicia X hasta donde termina. Igual sucede con Y, donde inicia Y hasta donde termina.

Paso 2: Con esos valores se calcula el valor Z. Al final se tiene un conjunto de coordenadas posX, posY, posZ. Se va a hacer uso de polígonos, por lo que se calculan cuatro coordenadas para formar el polígono.

Paso 3: Se normalizan los valores de posX, posY, posZ para que queden entre 0 y 1, luego se le resta -0.5 ¿Para qué? Para que los puntos (realmente polígonos) queden contenidos dentro de un cubo de lado=1, cuyo centro está en 0,0,0. Ver los dos temas anteriores como se muestra el cubo.

Paso 4: Luego se aplica el giro en los tres ángulos. Se obtiene posXg, posYg, posZg

Paso 5: Se ordenan los polígonos del más profundo (menor valor de posZg) al más superficial (mayor valor de posZg). Por esa razón, la clase Polígono hereda de IComparable

Paso 6: Con Xg, Yg, Zg se proyecta a la pantalla, obteniéndose el planoX, planoY.

Paso 7: Con planoX, planoY se aplican las constantes que se calcularon anteriormente y se obtiene la proyección en pantalla, es decir, pantallaX, pantallaY

Paso 8: Se dibujan los polígonos, primero rellenándolos con el color del fondo y luego se gráfica el perímetro de ese polígono.

Como es una aplicación gráfica de escritorio, entonces así debe ser la ventana:

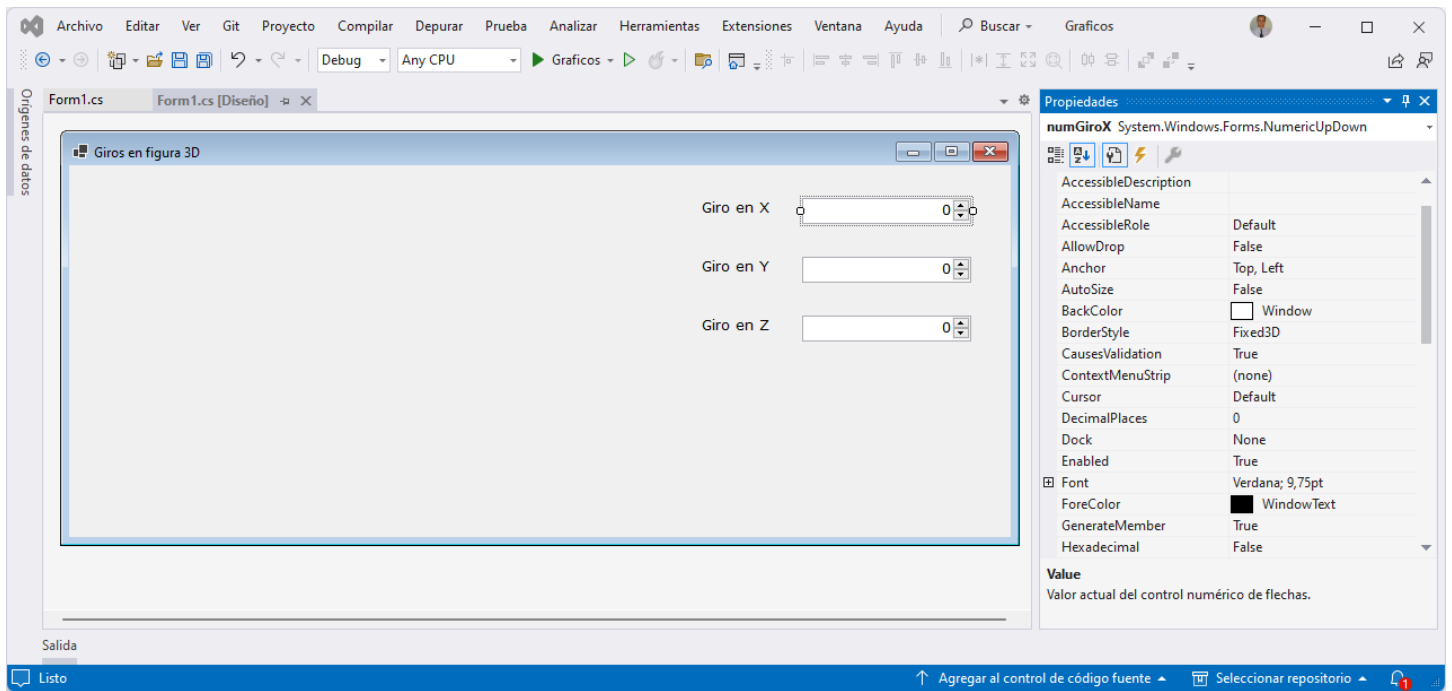


Ilustración 22: Diseño de ventana

Se utilizan los siguientes controles:

Tipo	Nombre
Label	lblGiroX
Label	lblGiroY
Label	lblGiroZ
NumericUpDown	numGiroX
NumericUpDown	numGiroY
NumericUpDown	numGiroZ

```

namespace Graficos {
    public partial class Form1 : Form {
        //El objeto que se encarga de calcular
        //y cuadrar en pantalla la ecuación  $Z = F(X,Y)$ 
        Grafico Grafico3D;

        public Form1() {
            InitializeComponent();
            Grafico3D = new Grafico();

            //Hace los cálculos de la ecuación primero.
            double MinX = -9;
            double MinY = -9;
            double MaxX = 9;
            double MaxY = 9;
            int numLineas = 30;
            Grafico3D.CalculaEcuacion(MinX, MinY, MaxX, MaxY, numLineas);
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapis = new(Color.Black, 1);
            Brush Relleno = new SolidBrush(Color.White);

            int AnguloX = Convert.ToInt32(numGiroX.Value);
            int AnguloY = Convert.ToInt32(numGiroY.Value);
            int AnguloZ = Convert.ToInt32(numGiroZ.Value);

            int ZPersona = 5;

            //Tamaño de la pantalla
            int XpIni = 0;
            int YpIni = 0;
            int XpFin = 800;
            int YpFin = 800;

            //Después de los cálculos, entonces aplica giros,
            //conversión a 2D y cuadrar en pantalla
            Grafico3D.CalculaGrafico(AnguloX, AnguloY, AnguloZ,
                                    ZPersona,
                                    XpIni, YpIni,
                                    XpFin, YpFin);
            Grafico3D.Dibuja(Lienzo, Lapis, Relleno);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }
    }
}

```



```

}

private void numGiroY_ValueChanged(object sender, EventArgs e) {
    Refresh();
}

private void numGiroZ_ValueChanged(object sender, EventArgs e) {
    Refresh();
}
}

internal class Grafico {

    //Donde almacena los poligonos
    private List<Poligono> poligono;

    public void CalculaEcuacion(double MinimoX, double MinimoY,
                                double MaximoX, double MaximoY,
                                int numLineas) {
        //Inicia el listado de polígonos que forma la figura
        poligono = [];

        //Mínimos y máximos para normalizar
        double MinimoZ = double.MaxValue;
        double MaximoZ = double.MinValue;

        //Calcula cada polígono dependiendo de la ecuación
        double IncrX = (MaximoX - MinimoX) / numLineas;
        double IncrY = (MaximoY - MinimoY) / numLineas;

        for (double ValX = MinimoX; ValX <= MaximoX; ValX += IncrX)
            for (double ValY = MinimoY; ValY <= MaximoY; ValY += IncrY) {

                //Calcula los 4 valores del eje Z del polígono
                double Z1 = Ecuacion(ValX, ValY);
                double Z2 = Ecuacion(ValX + IncrX, ValY);
                double Z3 = Ecuacion(ValX + IncrX, ValY + IncrY);
                double Z4 = Ecuacion(ValX, ValY + IncrY);

                //Si un valor de Z es inválido se pone en cero
                if (double.IsNaN(Z1) || double.IsInfinity(Z1)) Z1 = 0;
                if (double.IsNaN(Z2) || double.IsInfinity(Z2)) Z2 = 0;
                if (double.IsNaN(Z3) || double.IsInfinity(Z3)) Z3 = 0;
                if (double.IsNaN(Z4) || double.IsInfinity(Z4)) Z4 = 0;

                //Captura el mínimo valor de Z de todos los polígonos
                if (Z1 < MinimoZ) MinimoZ = Z1;
                if (Z2 < MinimoZ) MinimoZ = Z2;
            }
        }
    }
}

```

```

        if (Z3 < MinimoZ) MinimoZ = Z3;
        if (Z4 < MinimoZ) MinimoZ = Z4;

        //Captura el máximo valor de Z de todos los polígonos
        if (Z1 > MaximoZ) MaximoZ = Z1;
        if (Z2 > MaximoZ) MaximoZ = Z2;
        if (Z3 > MaximoZ) MaximoZ = Z3;
        if (Z4 > MaximoZ) MaximoZ = Z4;

        //Añade un polígono a la lista
        poligono.Add(new Poligono(ValX, ValY, Z1,
                                   ValX + IncrX, ValY, Z2,
                                   ValX + IncrX, ValY + IncrY, Z3,
                                   ValX, ValY + IncrY, Z4));
    }

    //Luego normaliza los puntos X,Y,Z
    //para que queden entre -0.5 y 0.5
    for (int cont = 0; cont < poligono.Count; cont++)
        poligono[cont].Normaliza(MinimoX, MinimoY, MinimoZ,
                                   MaximoX + IncrX, MaximoY + IncrY, MaximoZ);
}

//Aquí está la ecuación 3D que se desee
//graficar con variable XY
private double Ecuacion(double X, double Y) {
    double Z = Math.Sqrt(X * X + Y * Y);
    Z += 3 * Math.Cos(Math.Sqrt(X * X + Y * Y)) + 5;
    return Z;
}

public void CalculaGrafico(double AngX, double AngY, double AngZ,
                           int ZPersona,
                           int XpIni, int YpIni,
                           int XpFin, int YpFin) {

    //Genera la matriz de rotación
    double CosX = Math.Cos(AngX * Math.PI / 180);
    double SinX = Math.Sin(AngX * Math.PI / 180);
    double CosY = Math.Cos(AngY * Math.PI / 180);
    double SinY = Math.Sin(AngY * Math.PI / 180);
    double CosZ = Math.Cos(AngZ * Math.PI / 180);
    double SinZ = Math.Sin(AngZ * Math.PI / 180);

    //Matriz de Rotación

```

https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions

```

        double[,] Matriz = new double[3, 3] {
{CosY*CosZ,-CosX*SinZ+SinX*SinY*CosZ,SinX*SinZ+CosX*SinY*CosZ},
{CosY*SinZ,CosX*CosZ+SinX*SinY*SinZ,-SinX*CosZ+CosX*SinY*SinZ},
{-SinY,SinX*CosY,CosX*CosY}
        };

        //Los valores extremos al girar la figura en X, Y, Z
        //(de 0 a 360 grados), porque está contenida
        //en un cubo de 1*1*1
        double MaximoX = 0.87931543769177811;
        double MinimoX = -0.87931543769177811;
        double MaximoY = 0.87931543769177811;
        double MinimoY = -0.87931543769177811;

        //Las constantes de transformación
        double ConstanteX = (XpFin - XpIni) / (MaximoX - MinimoX);
        double ConstanteY = (YpFin - YpIni) / (MaximoY - MinimoY);

        //Gira los polígonos, proyecta a 2D y cuadra en pantalla
        for (int cont = 0; cont < poligono.Count; cont++)
            poligono[cont].CalculoPantalla(Matriz, ZPersona,
                                            ConstanteX, ConstanteY,
                                            MinimoX, MinimoY,
                                            XpIni, YpIni);

        //Ordena del polígono más alejado al más cercano,
        //de esa manera los polígonos de adelante son
        //visibles y los de atrás son borrados.
        poligono.Sort();
    }

    //Dibuja el polígono
    public void Dibuja(Graphics Lienzo, Pen Lapis, Brush Relleno) {
        for (int Cont = 0; Cont < poligono.Count; Cont++)
            poligono[Cont].Dibuja(Lienzo, Lapis, Relleno);
    }
}

internal class Poligono : IComparable {
    //Un polígono son cuatro(4) coordenadas espaciales
    public double X1, Y1, Z1;
    public double X2, Y2, Z2;
    public double X3, Y3, Z3;
    public double X4, Y4, Z4;

    //Las coordenadas en pantalla
    public int X1p, Y1p, X2p, Y2p, X3p, Y3p, X4p, Y4p;
}

```

```

//Centro del polígono
public double Centro;

public Poligono(double X1, double Y1, double Z1,
                double X2, double Y2, double Z2,
                double X3, double Y3, double Z3,
                double X4, double Y4, double Z4) {
    this.X1 = X1; this.Y1 = Y1; this.Z1 = Z1;
    this.X2 = X2; this.Y2 = Y2; this.Z2 = Z2;
    this.X3 = X3; this.Y3 = Y3; this.Z3 = Z3;
    this.X4 = X4; this.Y4 = Y4; this.Z4 = Z4;
}

//Normaliza puntos polígono entre -0.5 y 0.5
public void Normaliza(double MinX, double MinY, double MinZ,
                     double MaxX, double MaxY, double MaxZ) {
    X1 = (X1 - MinX) / (MaxX - MinX) - 0.5;
    Y1 = (Y1 - MinY) / (MaxY - MinY) - 0.5;
    Z1 = (Z1 - MinZ) / (MaxZ - MinZ) - 0.5;

    X2 = (X2 - MinX) / (MaxX - MinX) - 0.5;
    Y2 = (Y2 - MinY) / (MaxY - MinY) - 0.5;
    Z2 = (Z2 - MinZ) / (MaxZ - MinZ) - 0.5;

    X3 = (X3 - MinX) / (MaxX - MinX) - 0.5;
    Y3 = (Y3 - MinY) / (MaxY - MinY) - 0.5;
    Z3 = (Z3 - MinZ) / (MaxZ - MinZ) - 0.5;

    X4 = (X4 - MinX) / (MaxX - MinX) - 0.5;
    Y4 = (Y4 - MinY) / (MaxY - MinY) - 0.5;
    Z4 = (Z4 - MinZ) / (MaxZ - MinZ) - 0.5;
}

//Gira en XYZ, convierte a 2D y cuadra en pantalla
public void CalculoPantalla(double[,] Mt, double ZPersona,
                           double conX, double conY,
                           double MinimoX, double MinimoY,
                           int XPIni, int YPIni) {
    double X1g = X1 * Mt[0, 0] + Y1 * Mt[1, 0] + Z1 * Mt[2, 0];
    double Y1g = X1 * Mt[0, 1] + Y1 * Mt[1, 1] + Z1 * Mt[2, 1];
    double Z1g = X1 * Mt[0, 2] + Y1 * Mt[1, 2] + Z1 * Mt[2, 2];

    double X2g = X2 * Mt[0, 0] + Y2 * Mt[1, 0] + Z2 * Mt[2, 0];
    double Y2g = X2 * Mt[0, 1] + Y2 * Mt[1, 1] + Z2 * Mt[2, 1];
    double Z2g = X2 * Mt[0, 2] + Y2 * Mt[1, 2] + Z2 * Mt[2, 2];
}

```

```

double X3g = X3 * Mt[0, 0] + Y3 * Mt[1, 0] + Z3 * Mt[2, 0];
double Y3g = X3 * Mt[0, 1] + Y3 * Mt[1, 1] + Z3 * Mt[2, 1];
double Z3g = X3 * Mt[0, 2] + Y3 * Mt[1, 2] + Z3 * Mt[2, 2];

double X4g = X4 * Mt[0, 0] + Y4 * Mt[1, 0] + Z4 * Mt[2, 0];
double Y4g = X4 * Mt[0, 1] + Y4 * Mt[1, 1] + Z4 * Mt[2, 1];
double Z4g = X4 * Mt[0, 2] + Y4 * Mt[1, 2] + Z4 * Mt[2, 2];

//Usado para ordenar los polígonos
//del más lejano al más cercano
Centro = (Z1g + Z2g + Z3g + Z4g) / 4;

//Convierte de 3D a 2D (segunda dimensión)
double X1sd = X1g * ZPersona / (ZPersona - Z1g);
double Y1sd = Y1g * ZPersona / (ZPersona - Z1g);

double X2sd = X2g * ZPersona / (ZPersona - Z2g);
double Y2sd = Y2g * ZPersona / (ZPersona - Z2g);

double X3sd = X3g * ZPersona / (ZPersona - Z3g);
double Y3sd = Y3g * ZPersona / (ZPersona - Z3g);

double X4sd = X4g * ZPersona / (ZPersona - Z4g);
double Y4sd = Y4g * ZPersona / (ZPersona - Z4g);

//Cuadra en pantalla física
X1p = Convert.ToInt32(conX * (X1sd - MinimoX) + XPIni);
Y1p = Convert.ToInt32(conY * (Y1sd - MinimoY) + YPIni);

X2p = Convert.ToInt32(conX * (X2sd - MinimoX) + XPIni);
Y2p = Convert.ToInt32(conY * (Y2sd - MinimoY) + YPIni);

X3p = Convert.ToInt32(conX * (X3sd - MinimoX) + XPIni);
Y3p = Convert.ToInt32(conY * (Y3sd - MinimoY) + YPIni);

X4p = Convert.ToInt32(conX * (X4sd - MinimoX) + XPIni);
Y4p = Convert.ToInt32(conY * (Y4sd - MinimoY) + YPIni);
}

//Hace el gráfico del polígono
public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
    //Pone un color de fondo al polígono
    //para borrar lo que hay detrás
    Point Punto1 = new(X1p, Y1p);
    Point Punto2 = new(X2p, Y2p);
    Point Punto3 = new(X3p, Y3p);
    Point Punto4 = new(X4p, Y4p);
    Point[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];
}

```

```

        //Dibuja el polígono relleno y su perímetro
        Lienzo.FillPolygon(Relleno, ListaPuntos);
        Lienzo.DrawPolygon(Lapiz, ListaPuntos);
    }

    //Usado para ordenar los polígonos
    //del más lejano al más cercano
    public int CompareTo(object obj) {
        Poligono OrdenCompara = obj as Poligono;
        if (OrdenCompara.Centro < Centro) return 1;
        if (OrdenCompara.Centro > Centro) return -1;
        return 0;
        //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-
        by-a-property-in-the-object
    }
}
}

```

Ejemplo de ejecución:

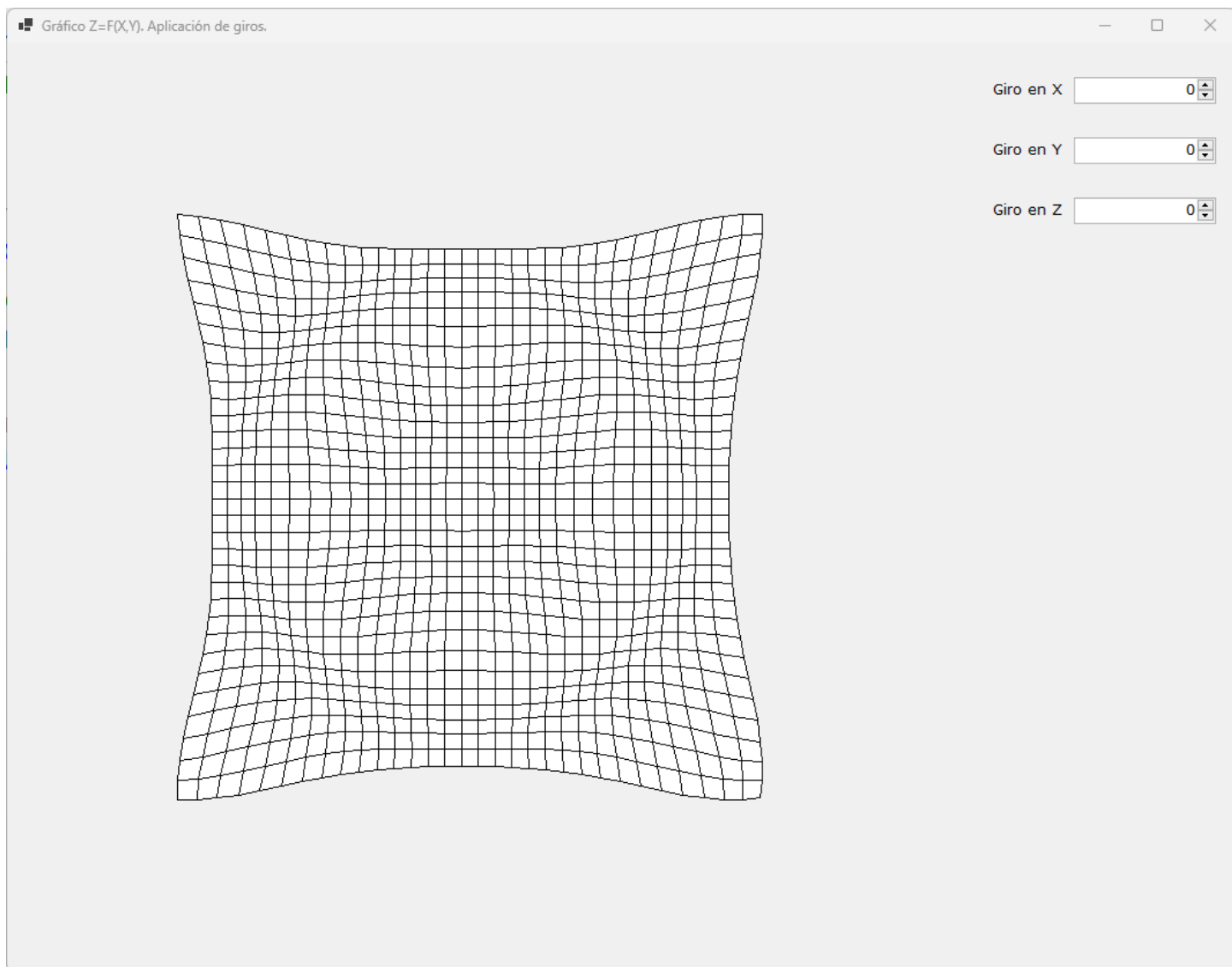


Ilustración 23: Ecuación en 3D proyectada

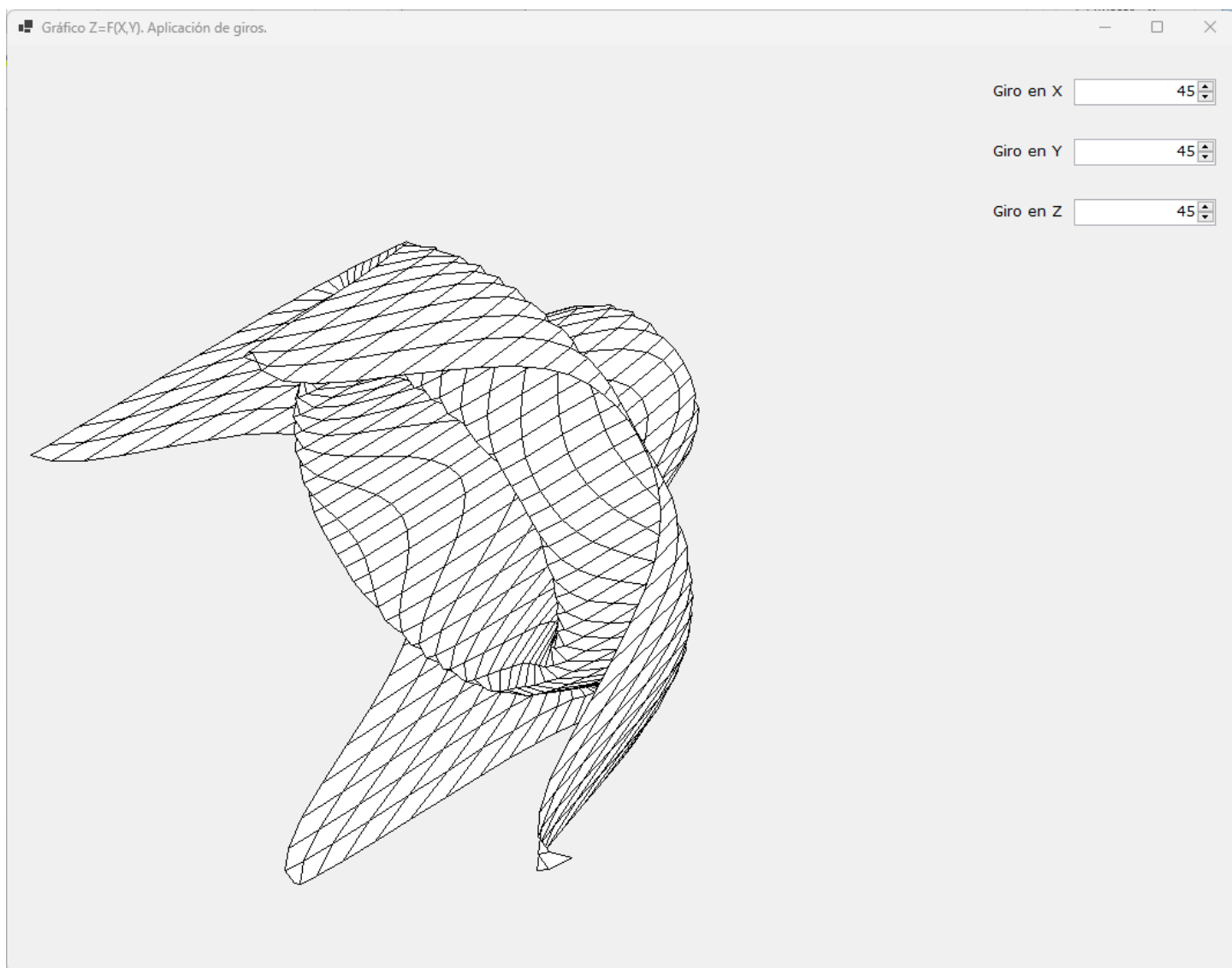


Ilustración 24: Ecuación en 3D proyectada y girada

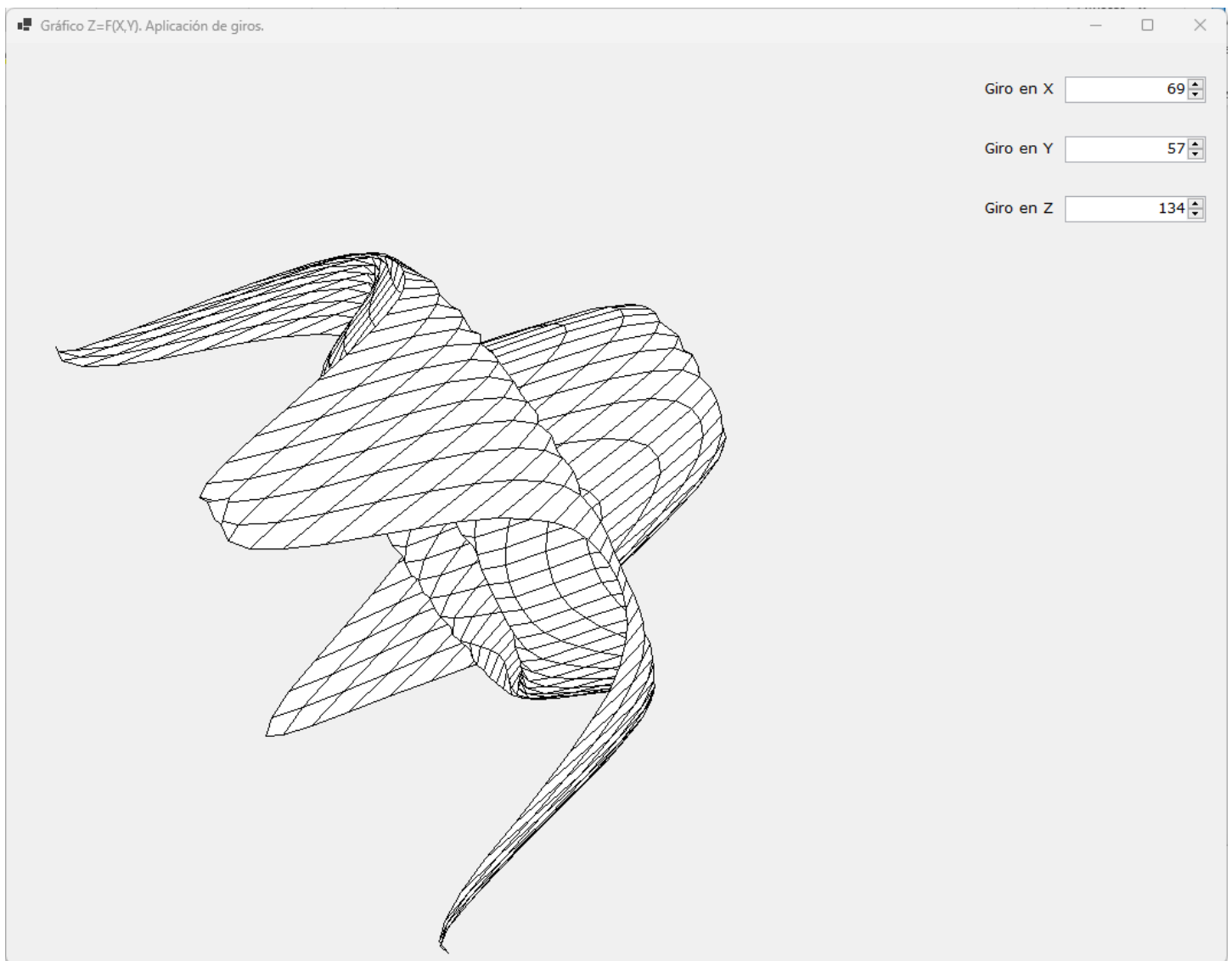


Ilustración 25: Ecuación 3D proyectada y girada

Se puede cambiar la ecuación aquí:

```
//Aquí está la ecuación 3D que se desee
//graficar con variable XY
private double Ecuacion(double X, double Y) {
    double Z = Math.Sqrt(X * X + Y * Y);
    Z += 3 * Math.Cos(Math.Sqrt(X * X + Y * Y)) + 5;
    return Z;
}
```

Gráfico Polar en 3D

Los gráficos polares en 3D trabajan con dos ángulos: Theta y Phi. Ver:

<https://mathematica.stackexchange.com/questions/83867/how-to-make-a-3d-plot-using-polar-coordinates> o <https://stackoverflow.com/questions/55031161/polar3d-plot-with-theta-phi-and-radius> o <https://mathworld.wolfram.com/SphericalCoordinates.html> o https://en.wikipedia.org/wiki/Spherical_coordinate_system

Por ejemplo, esta ecuación:

$$r = \text{Cos}(\varphi) + \text{Sen}(\theta)$$

Los pasos para dibujar el gráfico polar 3D son:

Paso 1: φ va de 0 a 2π , θ va de 0 a π y con eso se obtienen los valores de r

Paso 2: Luego se hace la traducción a coordenadas XYZ con estas fórmulas:

$$x = r * \text{Cos}(\varphi) * \text{Sen}(\theta)$$

$$y = r * \text{Sen}(\varphi) * \text{Sen}(\theta)$$

$$z = r * \text{Cos}(\theta)$$

Paso 3: Se hace el mismo procedimiento con los gráficos 3D de XYZ

Como es una aplicación gráfica de escritorio, entonces así debe ser la ventana:

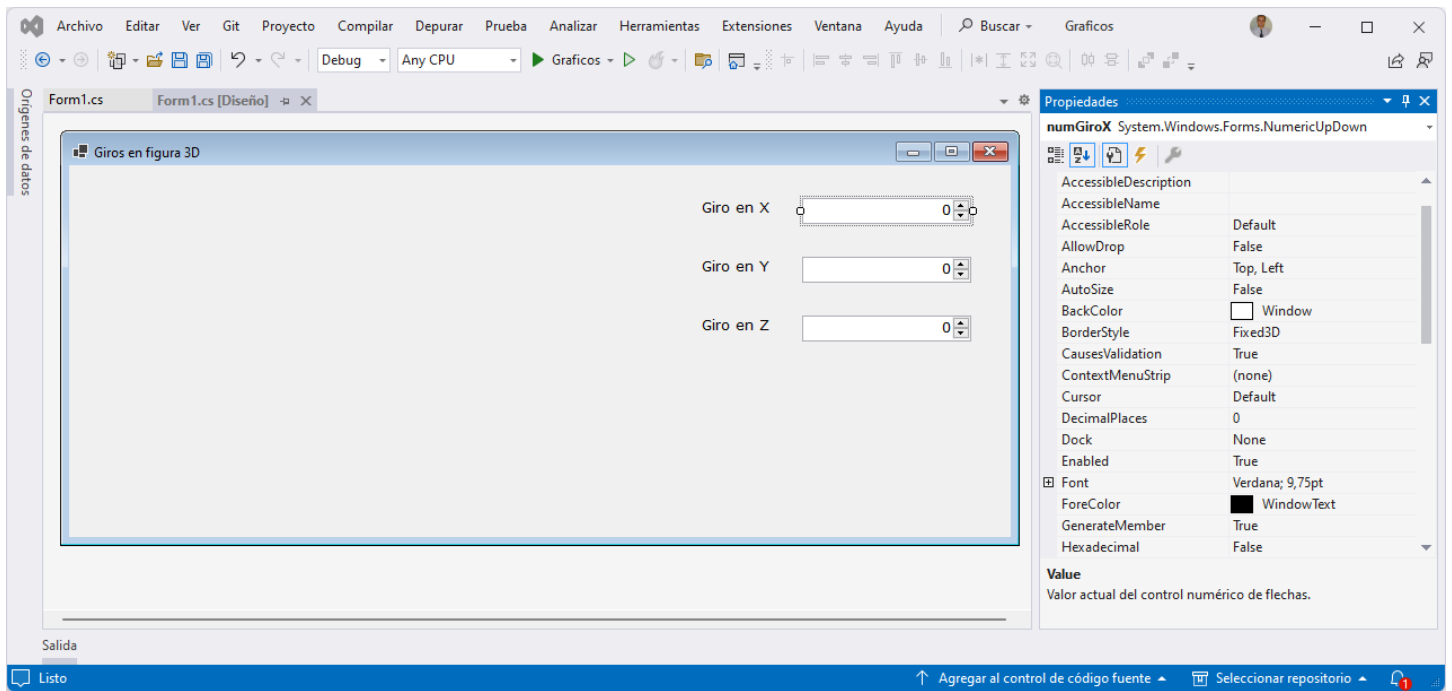


Ilustración 26: Diseño de ventana

Se utilizan los siguientes controles:

Tipo	Nombre
Label	lblGiroX
Label	lblGiroY
Label	lblGiroZ
NumericUpDown	numGiroX
NumericUpDown	numGiroY
NumericUpDown	numGiroZ

```

namespace Graficos {
    public partial class Form1 : Form {
        //El objeto que se encarga de calcular
        //y cuadrar en pantalla la ecuación polar
        Grafico Grafico3D;

        public Form1() {
            InitializeComponent();
            Grafico3D = new Grafico();

            int numLineas = 70;
            Grafico3D.CalculaEcuacion(numLineas);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapiz = new(Color.Black, 1);
            Brush Relleno = new SolidBrush(Color.White);

            int AnguloX = Convert.ToInt32(numGiroX.Value);
            int AnguloY = Convert.ToInt32(numGiroY.Value);
            int AnguloZ = Convert.ToInt32(numGiroZ.Value);

            int ZPersona = 5;
            int XPantallaIni = 0;
            int YPantallaIni = 0;
            int XPantallaFin = 800;
            int YPantallaFin = 800;

            //Después de los cálculos, entonces aplica giros,
            //conversión a 2D y cuadrar en pantalla
            Grafico3D.CalculaGrafico(AnguloX, AnguloY, AnguloZ,
                                    ZPersona,
                                    XPantallaIni, YPantallaIni,
                                    XPantallaFin, YPantallaFin);
        }
    }
}

```

```

        Grafico3D.Dibuja(Lienzo, Lapis, Relleno);
    }
}

internal class Grafico {

    //Donde almacena los poligonos
    private List<Poligono> poligono;

    public void CalculaEcuacion(int numLineas) {
        //Inicia el listado de polígonos que forma la figura
        poligono = [];

        //Mínimos y máximos para normalizar
        double MinimoX = double.MaxValue;
        double MaximoX = double.MinValue;
        double MinimoY = double.MaxValue;
        double MaximoY = double.MinValue;
        double MinimoZ = double.MaxValue;
        double MaximoZ = double.MinValue;

        //Calcula cada polígono dependiendo de la ecuación
        double MinTheta = 0, MaxTheta = 360, MinPhi = 0, MaxPhi = 360;

        double IncTheta = (MaxTheta - MinTheta) / numLineas;
        double IncPhi = (MaxPhi - MinPhi) / numLineas;

        for (double Th = MinTheta; Th <= MaxTheta; Th += IncTheta)
            for (double Phi = MinPhi; Phi <= MaxPhi; Phi += IncPhi) {

                //Calcula los 4 valores del eje Z del polígono

                //Con los ángulos
                double Theta1 = Th * Math.PI / 180;
                double Phi1 = Phi * Math.PI / 180;
                double R1 = Ecuacion(Theta1, Phi1);
                if (double.IsNaN(R1) || double.IsInfinity(R1)) R1 = 0;

                //Conversión a coordenadas X,Y,Z
                double X1 = R1 * Math.Cos(Phi1) * Math.Sin(Theta1);
                double Y1 = R1 * Math.Sin(Phi1) * Math.Sin(Theta1);
                double Z1 = R1 * Math.Cos(Theta1);

                double Theta2 = (Th + IncTheta) * Math.PI / 180;
                double Phi2 = Phi * Math.PI / 180;
                double R2 = Ecuacion(Theta2, Phi2);
                if (double.IsNaN(R2) || double.IsInfinity(R2)) R2 = 0;
                double X2 = R2 * Math.Cos(Phi2) * Math.Sin(Theta2);
            }
    }
}

```

```

double Y2 = R2 * Math.Sin(Phi2) * Math.Sin(Theta2);
double Z2 = R2 * Math.Cos(Theta2);

double Theta3 = (Th + IncTheta) * Math.PI / 180;
double Phi3 = (Phi + IncPhi) * Math.PI / 180;
double R3 = Ecuacion(Theta3, Phi3);
if (double.IsNaN(R3) || double.IsInfinity(R3)) R3 = 0;
double X3 = R3 * Math.Cos(Phi3) * Math.Sin(Theta3);
double Y3 = R3 * Math.Sin(Phi3) * Math.Sin(Theta3);
double Z3 = R3 * Math.Cos(Theta3);

double Theta4 = Th * Math.PI / 180;
double Phi4 = (Phi + IncPhi) * Math.PI / 180;
double R4 = Ecuacion(Theta4, Phi4);
if (double.IsNaN(R4) || double.IsInfinity(R4)) R4 = 0;
double X4 = R4 * Math.Cos(Phi4) * Math.Sin(Theta4);
double Y4 = R4 * Math.Sin(Phi4) * Math.Sin(Theta4);
double Z4 = R4 * Math.Cos(Theta4);

if (X1 < MinimoX) MinimoX = X1;
if (X2 < MinimoX) MinimoX = X2;
if (X3 < MinimoX) MinimoX = X3;
if (X4 < MinimoX) MinimoX = X4;

if (Y1 < MinimoY) MinimoY = Y1;
if (Y2 < MinimoY) MinimoY = Y2;
if (Y3 < MinimoY) MinimoY = Y3;
if (Y4 < MinimoY) MinimoY = Y4;

if (Z1 < MinimoZ) MinimoZ = Z1;
if (Z2 < MinimoZ) MinimoZ = Z2;
if (Z3 < MinimoZ) MinimoZ = Z3;
if (Z4 < MinimoZ) MinimoZ = Z4;

if (X1 > MaximoX) MaximoX = X1;
if (X2 > MaximoX) MaximoX = X2;
if (X3 > MaximoX) MaximoX = X3;
if (X4 > MaximoX) MaximoX = X4;

if (Y1 > MaximoY) MaximoY = Y1;
if (Y2 > MaximoY) MaximoY = Y2;
if (Y3 > MaximoY) MaximoY = Y3;
if (Y4 > MaximoY) MaximoY = Y4;

if (Z1 > MaximoZ) MaximoZ = Z1;
if (Z2 > MaximoZ) MaximoZ = Z2;
if (Z3 > MaximoZ) MaximoZ = Z3;
if (Z4 > MaximoZ) MaximoZ = Z4;

```

```

        //Añade un polígono a la lista
        poligono.Add(new Poligono(X1, Y1, Z1,
                                   X2, Y2, Z2,
                                   X3, Y3, Z3,
                                   X4, Y4, Z4));
    }

    //Luego normaliza los puntos X,Y,Z
    //para que queden entre -0.5 y 0.5
    for (int cont = 0; cont < poligono.Count; cont++)
        poligono[cont].Normaliza(MinimoX, MinimoY, MinimoZ,
                                   MaximoX, MaximoY, MaximoZ);
}

//Aquí está la ecuación polar 3D que se desee graficar
//con variable Theta y PHI
public double Ecuacion(double Theta, double Phi) {
    return Math.Cos(Phi + Theta) - Math.Sin(Theta - Phi);
}

public void CalculaGrafico(double AngX, double AngY, double AngZ,
                           int ZPersona,
                           int XpIni, int YpIni,
                           int XpFin, int YpFin) {

    //Genera la matriz de rotación
    double CosX = Math.Cos(AngX * Math.PI / 180);
    double SinX = Math.Sin(AngX * Math.PI / 180);
    double CosY = Math.Cos(AngY * Math.PI / 180);
    double SinY = Math.Sin(AngY * Math.PI / 180);
    double CosZ = Math.Cos(AngZ * Math.PI / 180);
    double SinZ = Math.Sin(AngZ * Math.PI / 180);

    //Matriz de Rotación:
https://en.wikipedia.org/wiki/Rotation\_formalisms\_in\_three\_dimensions
    double[,] Matriz = new double[3, 3] {
        {CosY*CosZ,-CosX*SinZ+SinX*SinY*CosZ,SinX*SinZ+CosX*SinY*CosZ},
        {CosY*SinZ,CosX*CosZ+SinX*SinY*SinZ,-SinX*CosZ+CosX*SinY*SinZ},
        {-SinY,SinX*CosY,CosX*CosY}
    };

    //Los valores extremos al girar la figura en X, Y, Z
    //(de 0 a 360 grados), porque está contenida
    //en un cubo de 1*1*1
    double MaximoX = 0.87931543769177811;
    double MinimoX = -0.87931543769177811;

```

```

double MaximoY = 0.87931543769177811;
double MinimoY = -0.87931543769177811;

//Las constantes de transformación
double conX = (XpFin - XpIni) / (MaximoX - MinimoX);
double conY = (YpFin - YpIni) / (MaximoY - MinimoY);

//Gira los polígonos, proyecta a 2D y cuadra en pantalla
for (int cont = 0; cont < poligono.Count; cont++)
    poligono[cont].CalculoPantalla(Matriz, ZPersona,
                                    conX, conY,
                                    MinimoX, MinimoY,
                                    XpIni, YpIni);

//Ordena del polígono más alejado al más cercano,
//de esa manera los polígonos de adelante son
//visibles y los de atrás son borrados.
poligono.Sort();
}

//Dibuja el polígono
public void Dibuja(Graphics Lienzo, Pen Lapis, Brush Relleno) {
    for (int Cont = 0; Cont < poligono.Count; Cont++)
        poligono[Cont].Dibuja(Lienzo, Lapis, Relleno);
}

internal class Poligono : IComparable {
    //Un polígono son cuatro(4) coordenadas espaciales
    public double X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4;

    //Las coordenadas en pantalla
    public int X1p, Y1p, X2p, Y2p, X3p, Y3p, X4p, Y4p;

    //Centro del polígono
    public double Centro;

    public Poligono(double X1, double Y1, double Z1,
                    double X2, double Y2, double Z2,
                    double X3, double Y3, double Z3,
                    double X4, double Y4, double Z4) {
        this.X1 = X1; this.Y1 = Y1; this.Z1 = Z1;
        this.X2 = X2; this.Y2 = Y2; this.Z2 = Z2;
        this.X3 = X3; this.Y3 = Y3; this.Z3 = Z3;
        this.X4 = X4; this.Y4 = Y4; this.Z4 = Z4;
    }

    //Normaliza puntos polígono entre -0.5 y 0.5

```



```

public void Normaliza(double MinimoX, double MinimoY, double MinimoZ,
                     double MaximoX, double MaximoY, double MaximoZ) {
    X1 = (X1 - MinimoX) / (MaximoX - MinimoX) - 0.5;
    Y1 = (Y1 - MinimoY) / (MaximoY - MinimoY) - 0.5;
    Z1 = (Z1 - MinimoZ) / (MaximoZ - MinimoZ) - 0.5;

    X2 = (X2 - MinimoX) / (MaximoX - MinimoX) - 0.5;
    Y2 = (Y2 - MinimoY) / (MaximoY - MinimoY) - 0.5;
    Z2 = (Z2 - MinimoZ) / (MaximoZ - MinimoZ) - 0.5;

    X3 = (X3 - MinimoX) / (MaximoX - MinimoX) - 0.5;
    Y3 = (Y3 - MinimoY) / (MaximoY - MinimoY) - 0.5;
    Z3 = (Z3 - MinimoZ) / (MaximoZ - MinimoZ) - 0.5;

    X4 = (X4 - MinimoX) / (MaximoX - MinimoX) - 0.5;
    Y4 = (Y4 - MinimoY) / (MaximoY - MinimoY) - 0.5;
    Z4 = (Z4 - MinimoZ) / (MaximoZ - MinimoZ) - 0.5;
}

//Gira en XYZ, convierte a 2D y cuadra en pantalla
public void CalculoPantalla(double[,] Mt, double ZPersona,
                           double ConstanteX, double ConstanteY,
                           double MinimoX, double MinimoY,
                           int XpIni, int YpIni) {
    double X1g = X1 * Mt[0, 0] + Y1 * Mt[1, 0] + Z1 * Mt[2, 0];
    double Y1g = X1 * Mt[0, 1] + Y1 * Mt[1, 1] + Z1 * Mt[2, 1];
    double Z1g = X1 * Mt[0, 2] + Y1 * Mt[1, 2] + Z1 * Mt[2, 2];

    double X2g = X2 * Mt[0, 0] + Y2 * Mt[1, 0] + Z2 * Mt[2, 0];
    double Y2g = X2 * Mt[0, 1] + Y2 * Mt[1, 1] + Z2 * Mt[2, 1];
    double Z2g = X2 * Mt[0, 2] + Y2 * Mt[1, 2] + Z2 * Mt[2, 2];

    double X3g = X3 * Mt[0, 0] + Y3 * Mt[1, 0] + Z3 * Mt[2, 0];
    double Y3g = X3 * Mt[0, 1] + Y3 * Mt[1, 1] + Z3 * Mt[2, 1];
    double Z3g = X3 * Mt[0, 2] + Y3 * Mt[1, 2] + Z3 * Mt[2, 2];

    double X4g = X4 * Mt[0, 0] + Y4 * Mt[1, 0] + Z4 * Mt[2, 0];
    double Y4g = X4 * Mt[0, 1] + Y4 * Mt[1, 1] + Z4 * Mt[2, 1];
    double Z4g = X4 * Mt[0, 2] + Y4 * Mt[1, 2] + Z4 * Mt[2, 2];

    //Usado para ordenar los polígonos del más lejano al más cercano
    Centro = (Z1g + Z2g + Z3g + Z4g) / 4;

    //Convierte de 3D a 2D (segunda dimensión)
    double X1sd = X1g * ZPersona / (ZPersona - Z1g);
    double Y1sd = Y1g * ZPersona / (ZPersona - Z1g);

```

```

double X2sd = X2g * ZPersona / (ZPersona - Z2g);
double Y2sd = Y2g * ZPersona / (ZPersona - Z2g);

double X3sd = X3g * ZPersona / (ZPersona - Z3g);
double Y3sd = Y3g * ZPersona / (ZPersona - Z3g);

double X4sd = X4g * ZPersona / (ZPersona - Z4g);
double Y4sd = Y4g * ZPersona / (ZPersona - Z4g);

//Cuadra en pantalla física
X1p = Convert.ToInt32(ConstanteX * (X1sd - MinimoX) + XpIni);
Y1p = Convert.ToInt32(ConstanteY * (Y1sd - MinimoY) + YpIni);

X2p = Convert.ToInt32(ConstanteX * (X2sd - MinimoX) + XpIni);
Y2p = Convert.ToInt32(ConstanteY * (Y2sd - MinimoY) + YpIni);

X3p = Convert.ToInt32(ConstanteX * (X3sd - MinimoX) + XpIni);
Y3p = Convert.ToInt32(ConstanteY * (Y3sd - MinimoY) + YpIni);

X4p = Convert.ToInt32(ConstanteX * (X4sd - MinimoX) + XpIni);
Y4p = Convert.ToInt32(ConstanteY * (Y4sd - MinimoY) + YpIni);
}

//Hace el gráfico del polígono
public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
    //Pone un color de fondo al polígono
    //para borrar lo que hay detrás
    Point Punto1 = new(X1p, Y1p);
    Point Punto2 = new(X2p, Y2p);
    Point Punto3 = new(X3p, Y3p);
    Point Punto4 = new(X4p, Y4p);
    Point[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];

    //Dibuja el polígono relleno y su perímetro
    Lienzo.FillPolygon(Relleno, ListaPuntos);
    Lienzo.DrawPolygon(Lapiz, ListaPuntos);
}

//Usado para ordenar los polígonos
//del más lejano al más cercano
public int CompareTo(object obj) {
    Poligono OrdenCompara = obj as Poligono;
    if (OrdenCompara.Centro < Centro) return 1;
    if (OrdenCompara.Centro > Centro) return -1;
    return 0;
    //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-
    by-a-property-in-the-object
}

```

Ejemplo de ejecución:

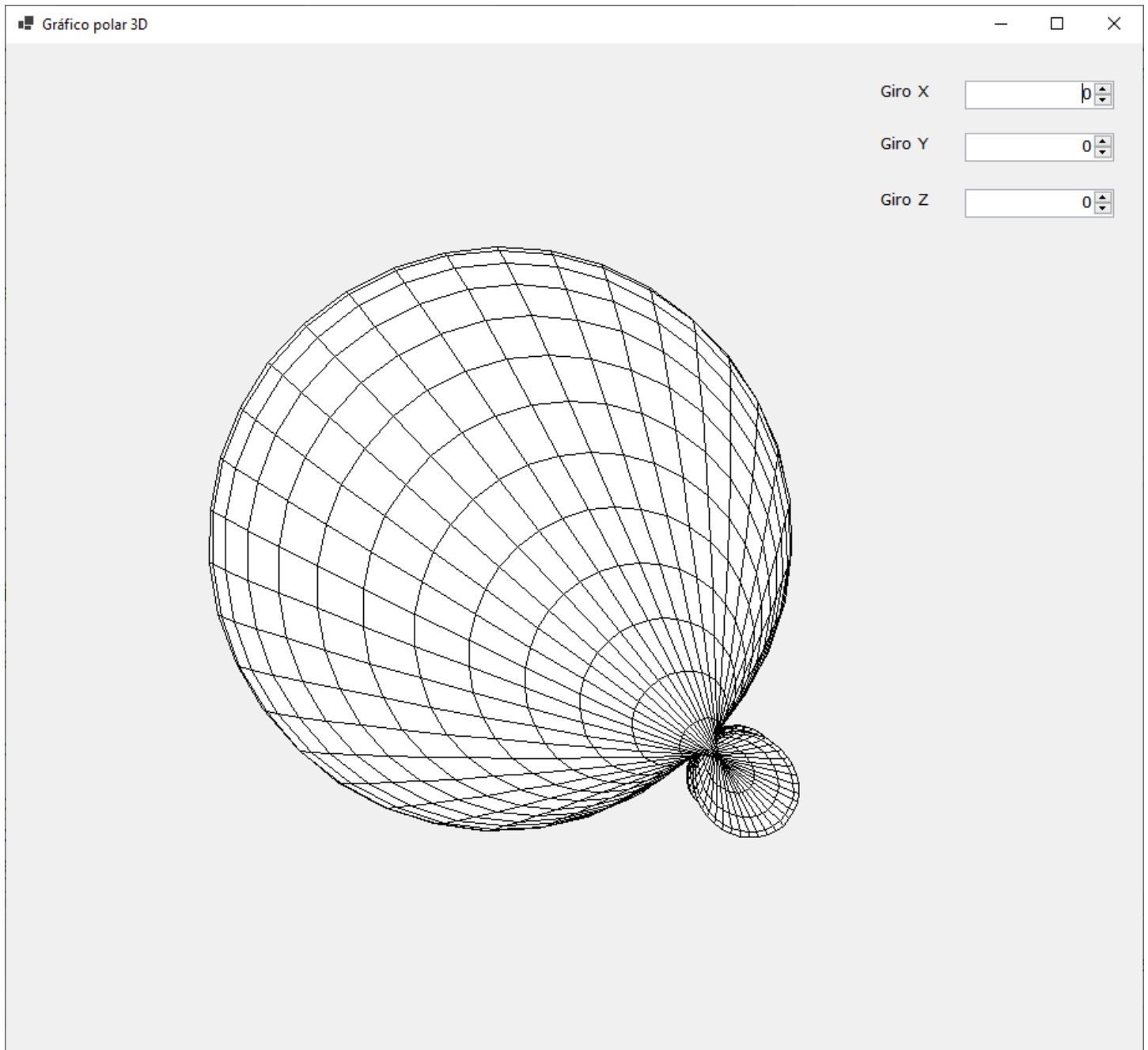


Ilustración 27: Ecuación polar 3D

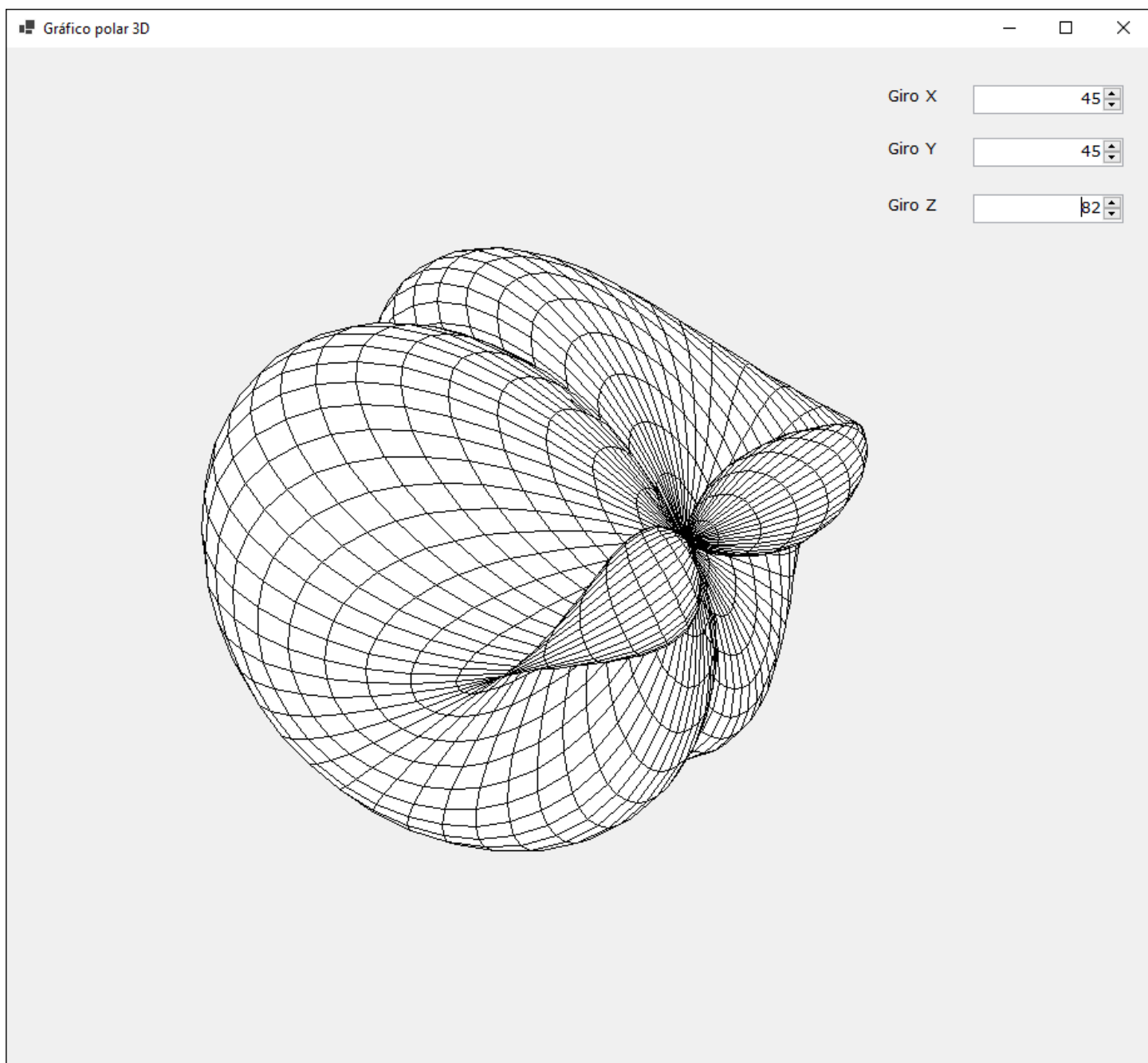


Ilustración 28: Ecuación polar 3D

Gráfico de sólido de revolución

Un sólido de revolución puede nacer de tomar una ecuación del tipo $Y=F(X)$, y luego poner a girar el gráfico resultante en el eje X. Ver más en:

https://es.wikipedia.org/wiki/S%C3%B3lido_de_revoluci%C3%B3n

Los pasos para hacer el gráfico son los siguientes:

Paso 1: Saber el valor donde inicia X hasta donde termina. Así se calcula Y. Un típico gráfico $Y=F(X)$ en 2D.

Paso 2: Ese gráfico en 2D va a girar sobre el eje X.

Paso 3: Se aplica la matriz de giro en el eje X. Cómo se va a hacer uso de polígonos, se calculan cuatro coordenadas (X,Y,Z) para formar cada polígono que conformará la figura 3D.

Paso 4: Se normalizan los valores de (X,Y,Z) para que queden entre 0 y 1, luego se le resta -0.5 ¿Para qué? Para que los puntos (realmente polígonos) queden contenidos dentro de un cubo de lado=1, cuyo centro está en 0,0,0. Ver los dos temas anteriores como se muestra el cubo.

Paso 5: Para cada valor (X,Y,Z) se aplica el giro en los tres ángulos (la matriz para hacer girar en 3D). Se obtiene X_g , Y_g , Z_g

Paso 6: Se ordenan los polígonos del más profundo (menor valor de Z_g) al más superficial (mayor valor de Z_g). Por esa razón, la clase Polígono hereda de IComparable

Paso 7: Con X_g , Y_g , Z_g se proyecta a la pantalla, obteniéndose el planoX, planoY.

Paso 8: Con planoX, planoY se aplican las constantes que se calcularon para proyectar el cubo y se obtiene los datos de pantalla, es decir, pantallaX, pantallaY

Paso 9: Se dibujan los polígonos, primero rellenándolos con el color del fondo y luego se gráfica el perímetro de ese polígono.

```

namespace Graficos {

    public partial class Form1 : Form {
        //El objeto que se encarga de calcular
        //y cuadrar en pantalla la ecuación que
        //genera el sólido de revolución
        Grafico Grafico3D;

        public Form1() {
            InitializeComponent();
            Grafico3D = new Grafico();

            double MinXreal = 0;
            double MaxXreal = 180;
            int numLineas = 40;
            Grafico3D.Calcula(MinXreal, MaxXreal, numLineas);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapiz = new(Color.Black, 1);
            Brush Relleno = new SolidBrush(Color.White);

            int AnguloX = Convert.ToInt32(numGiroX.Value);
            int AnguloY = Convert.ToInt32(numGiroY.Value);
            int AnguloZ = Convert.ToInt32(numGiroZ.Value);

            //Extremos de la ventana para dibujar
            int ZPersona = 5;
            int XpIni = 0;
            int YpIni = 0;
            int XpFin = 800;
            int YpFin = 800;
        }
    }
}

```

```

        //Después de los cálculos, entonces aplica giros,
        //conversión a 2D y cuadrar en pantalla
        Grafico3D.CalculaGrafico(AnguloX, AnguloY, AnguloZ,
                                ZPersona,
                                XpIni, YpIni,
                                XpFin, YpFin);
        Grafico3D.Dibuja(Lienzo, Lapiz, Relleno);
    }
}

internal class Poligono : IComparable {
    //Un polígono son cuatro(4) coordenadas espaciales
    public double X1, Y1, Z1;
    public double X2, Y2, Z2;
    public double X3, Y3, Z3;
    public double X4, Y4, Z4;

    //Las coordenadas en pantalla
    public int X1p, Y1p, X2p, Y2p, X3p, Y3p, X4p, Y4p;

    //Centro del polígono
    public double Centro;

    public Poligono(double X1, double Y1, double Z1,
                    double X2, double Y2, double Z2,
                    double X3, double Y3, double Z3,
                    double X4, double Y4, double Z4) {
        this.X1 = X1; this.Y1 = Y1; this.Z1 = Z1;
        this.X2 = X2; this.Y2 = Y2; this.Z2 = Z2;
        this.X3 = X3; this.Y3 = Y3; this.Z3 = Z3;
        this.X4 = X4; this.Y4 = Y4; this.Z4 = Z4;
    }

    //Normaliza puntos polígono entre -0.5 y 0.5
    public void Normaliza(double MinX, double MinY, double MinZ,
                          double MaxX, double MaxY, double MaxZ) {
        X1 = (X1 - MinX) / (MaxX - MinX) - 0.5;
        Y1 = (Y1 - MinY) / (MaxY - MinY) - 0.5;
        Z1 = (Z1 - MinZ) / (MaxZ - MinZ) - 0.5;

        X2 = (X2 - MinX) / (MaxX - MinX) - 0.5;
        Y2 = (Y2 - MinY) / (MaxY - MinY) - 0.5;
        Z2 = (Z2 - MinZ) / (MaxZ - MinZ) - 0.5;

        X3 = (X3 - MinX) / (MaxX - MinX) - 0.5;
        Y3 = (Y3 - MinY) / (MaxY - MinY) - 0.5;
        Z3 = (Z3 - MinZ) / (MaxZ - MinZ) - 0.5;
    }
}

```

```

X4 = (X4 - MinX) / (MaxX - MinX) - 0.5;
Y4 = (Y4 - MinY) / (MaxY - MinY) - 0.5;
Z4 = (Z4 - MinZ) / (MaxZ - MinZ) - 0.5;
}

//Gira en XYZ, convierte a 2D y cuadra en pantalla
public void CalculoPantalla(double[,] Mt, double ZPersona,
    double conX, double conY,
    double MinimoX, double MinimoY,
    int XpIni, int YpIni) {
    double X1g = X1 * Mt[0, 0] + Y1 * Mt[1, 0] + Z1 * Mt[2, 0];
    double Y1g = X1 * Mt[0, 1] + Y1 * Mt[1, 1] + Z1 * Mt[2, 1];
    double Z1g = X1 * Mt[0, 2] + Y1 * Mt[1, 2] + Z1 * Mt[2, 2];

    double X2g = X2 * Mt[0, 0] + Y2 * Mt[1, 0] + Z2 * Mt[2, 0];
    double Y2g = X2 * Mt[0, 1] + Y2 * Mt[1, 1] + Z2 * Mt[2, 1];
    double Z2g = X2 * Mt[0, 2] + Y2 * Mt[1, 2] + Z2 * Mt[2, 2];

    double X3g = X3 * Mt[0, 0] + Y3 * Mt[1, 0] + Z3 * Mt[2, 0];
    double Y3g = X3 * Mt[0, 1] + Y3 * Mt[1, 1] + Z3 * Mt[2, 1];
    double Z3g = X3 * Mt[0, 2] + Y3 * Mt[1, 2] + Z3 * Mt[2, 2];

    double X4g = X4 * Mt[0, 0] + Y4 * Mt[1, 0] + Z4 * Mt[2, 0];
    double Y4g = X4 * Mt[0, 1] + Y4 * Mt[1, 1] + Z4 * Mt[2, 1];
    double Z4g = X4 * Mt[0, 2] + Y4 * Mt[1, 2] + Z4 * Mt[2, 2];

    //Usado para ordenar los polígonos
    //del más lejano al más cercano
    Centro = (Z1g + Z2g + Z3g + Z4g) / 4;

    //Convierte de 3D a 2D (segunda dimensión)
    double X1sd = X1g * ZPersona / (ZPersona - Z1g);
    double Y1sd = Y1g * ZPersona / (ZPersona - Z1g);

    double X2sd = X2g * ZPersona / (ZPersona - Z2g);
    double Y2sd = Y2g * ZPersona / (ZPersona - Z2g);

    double X3sd = X3g * ZPersona / (ZPersona - Z3g);
    double Y3sd = Y3g * ZPersona / (ZPersona - Z3g);

    double X4sd = X4g * ZPersona / (ZPersona - Z4g);
    double Y4sd = Y4g * ZPersona / (ZPersona - Z4g);

    //Cuadra en pantalla física
    X1p = Convert.ToInt32(conX * (X1sd - MinimoX) + XpIni);
    Y1p = Convert.ToInt32(conY * (Y1sd - MinimoY) + YpIni);

```



```

X2p = Convert.ToInt32(conX * (X2sd - MinimoX) + XpIni);
Y2p = Convert.ToInt32(conY * (Y2sd - MinimoY) + YpIni);

X3p = Convert.ToInt32(conX * (X3sd - MinimoX) + XpIni);
Y3p = Convert.ToInt32(conY * (Y3sd - MinimoY) + YpIni);

X4p = Convert.ToInt32(conX * (X4sd - MinimoX) + XpIni);
Y4p = Convert.ToInt32(conY * (Y4sd - MinimoY) + YpIni);
}

//Hace el gráfico del polígono
public void Dibuja(Graphics Lienzo, Pen Lapis, Brush Relleno) {
    //Pone un color de fondo al polígono
    //para borrar lo que hay detrás
    Point Punto1 = new(X1p, Y1p);
    Point Punto2 = new(X2p, Y2p);
    Point Punto3 = new(X3p, Y3p);
    Point Punto4 = new(X4p, Y4p);
    Point[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];

    //Dibuja el polígono relleno y su perímetro
    Lienzo.FillPolygon(Relleno, ListaPuntos);
    Lienzo.DrawPolygon(Lapis, ListaPuntos);
}

//Usado para ordenar los polígonos
//del más lejano al más cercano
public int CompareTo(object obj) {
    Poligono OrdenCompara = obj as Poligono;
    if (OrdenCompara.Centro < Centro) return 1;
    if (OrdenCompara.Centro > Centro) return -1;
    return 0;
    //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-
    by-a-property-in-the-object
}
}

internal class Grafico {

    //Donde almacena los poligonos
    private List<Poligono> poligono;

    public void Calcula(double minX, double maxX, int numLineas) {
        //Inicia el listado de polígonos
        //que forma la figura
        poligono = [];

        //Mínimos y máximos para normalizar

```

```

double MinimoX = double.MaxValue;
double MaximoX = double.MinValue;
double MinimoY = double.MaxValue;
double MaximoY = double.MinValue;
double MinimoZ = double.MaxValue;
double MaximoZ = double.MinValue;

double IncAng = 360 / numLineas;
double IncX = (maxX - minX) / numLineas;
double radian = Math.PI / 180;

//Usando la matriz de Giro en X, se toma cada punto
//del gráfico 2D Y=F(X) y se hace girar en X
for (double ang = 0; ang < 360; ang += IncAng)
    for (double X = minX; X <= maxX; X += IncX) {

        //Primer punto
        double X1 = X;
        double Y1 = Ecuacion(X1);
        if (double.IsNaN(Y1) || double.IsInfinity(Y1)) Y1 = 0;

        //Hace giro
        double X1g = X1;
        double Y1g = Y1 * Math.Cos(ang * radian);
        double Z1g = Y1 * Math.Sin(ang * radian);

        //Segundo punto
        double X2 = X + IncX;
        double Y2 = Ecuacion(X2);
        if (double.IsNaN(Y2) || double.IsInfinity(Y2)) Y2 = 0;

        //Hace giro
        double X2g = X2;
        double Y2g = Y2 * Math.Cos(ang * radian);
        double Z2g = Y2 * Math.Sin(ang * radian);

        //Tercer punto ya girado
        double X3g = X2;
        double Y3g = Y2 * Math.Cos((ang + IncAng) * radian);
        double Z3g = Y2 * Math.Sin((ang + IncAng) * radian);

        //Cuarto punto ya girado
        double X4g = X1;
        double Y4g = Y1 * Math.Cos((ang + IncAng) * radian);
        double Z4g = Y1 * Math.Sin((ang + IncAng) * radian);

        //Obtener los valores extremos para poder normalizar
        if (X1g < MinimoX) MinimoX = X1g;
    }

```

```

        if (X2g < MinimoX) MinimoX = X2g;
        if (X3g < MinimoX) MinimoX = X3g;
        if (X4g < MinimoX) MinimoX = X4g;

        if (Y1g < MinimoY) MinimoY = Y1g;
        if (Y2g < MinimoY) MinimoY = Y2g;
        if (Y3g < MinimoY) MinimoY = Y3g;
        if (Y4g < MinimoY) MinimoY = Y4g;

        if (Z1g < MinimoZ) MinimoZ = Z1g;
        if (Z2g < MinimoZ) MinimoZ = Z2g;
        if (Z3g < MinimoZ) MinimoZ = Z3g;
        if (Z4g < MinimoZ) MinimoZ = Z4g;

        if (X1g > MaximoX) MaximoX = X1g;
        if (X2g > MaximoX) MaximoX = X2g;
        if (X3g > MaximoX) MaximoX = X3g;
        if (X4g > MaximoX) MaximoX = X4g;

        if (Y1g > MaximoY) MaximoY = Y1g;
        if (Y2g > MaximoY) MaximoY = Y2g;
        if (Y3g > MaximoY) MaximoY = Y3g;
        if (Y4g > MaximoY) MaximoY = Y4g;

        if (Z1g > MaximoZ) MaximoZ = Z1g;
        if (Z2g > MaximoZ) MaximoZ = Z2g;
        if (Z3g > MaximoZ) MaximoZ = Z3g;
        if (Z4g > MaximoZ) MaximoZ = Z4g;

        poligono.Add(new Poligono(X1g, Y1g, Z1g,
                                   X2g, Y2g, Z2g,
                                   X3g, Y3g, Z3g,
                                   X4g, Y4g, Z4g));
    }

    //Luego normaliza los puntos X,Y,Z
    //para que queden entre -0.5 y 0.5
    for (int cont = 0; cont < poligono.Count; cont++)
        poligono[cont].Normaliza(MinimoX, MinimoY, MinimoZ,
                                   MaximoX, MaximoY, MaximoZ);
}

//Aquí está la ecuación Y=F(X) que se desee
//graficar para hacer el sólido de revolución.
public double Ecuacion(double X) {
    return 1.7 * Math.Sin((2 * X - 3) * Math.PI / 180);
}

```

```

public void CalculaGrafico(double AngX, double AngY, double AngZ,
                           int ZPersona,
                           int XpIni, int YpIni,
                           int XpFin, int YpFin) {

    //Genera la matriz de rotación
    double CosX = Math.Cos(AngX * Math.PI / 180);
    double SinX = Math.Sin(AngX * Math.PI / 180);
    double CosY = Math.Cos(AngY * Math.PI / 180);
    double SinY = Math.Sin(AngY * Math.PI / 180);
    double CosZ = Math.Cos(AngZ * Math.PI / 180);
    double SinZ = Math.Sin(AngZ * Math.PI / 180);

    //Matriz de Rotación

    //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
    double[,] Matriz = new double[3, 3] {
{CosY*CosZ,-CosX*SinZ+SinX*SinY*CosZ,SinX*SinZ+CosX*SinY*CosZ},
{CosY*SinZ,CosX*CosZ+SinX*SinY*SinZ,-SinX*CosZ+CosX*SinY*SinZ},
{-SinY,SinX*CosY,CosX*CosY}
    };

    //Los valores extremos al girar la figura en
    //X, Y, Z (de 0 a 360 grados), porque está
    //contenida en un cubo de 1*1*1
    double MaximoX = 0.87931543769177811;
    double MinimoX = -0.87931543769177811;
    double MaximoY = 0.87931543769177811;
    double MinimoY = -0.87931543769177811;

    //Las constantes de transformación
    double conX = (XpFin - XpIni) / (MaximoX - MinimoX);
    double conY = (YpFin - YpIni) / (MaximoY - MinimoY);

    //Gira los polígonos, proyecta a 2D y cuadra en pantalla
    for (int cont = 0; cont < poligono.Count; cont++)
        poligono[cont].CalculoPantalla(Matriz, ZPersona,
                                         conX, conY,
                                         MinimoX, MinimoY,
                                         XpIni, YpIni);

    //Ordena del polígono más alejado al más cercano,
    //de esa manera los polígonos de adelante son
    //visibles y los de atrás son borrados.
    poligono.Sort();
}

```

```

//Dibuja el polígono
public void Dibuja(Graphics Lienzo, Pen Lapis, Brush Relleno) {
    for (int Cont = 0; Cont < poligono.Count; Cont++)
        poligono[Cont].Dibuja(Lienzo, Lapis, Relleno);
    }
}
}

```

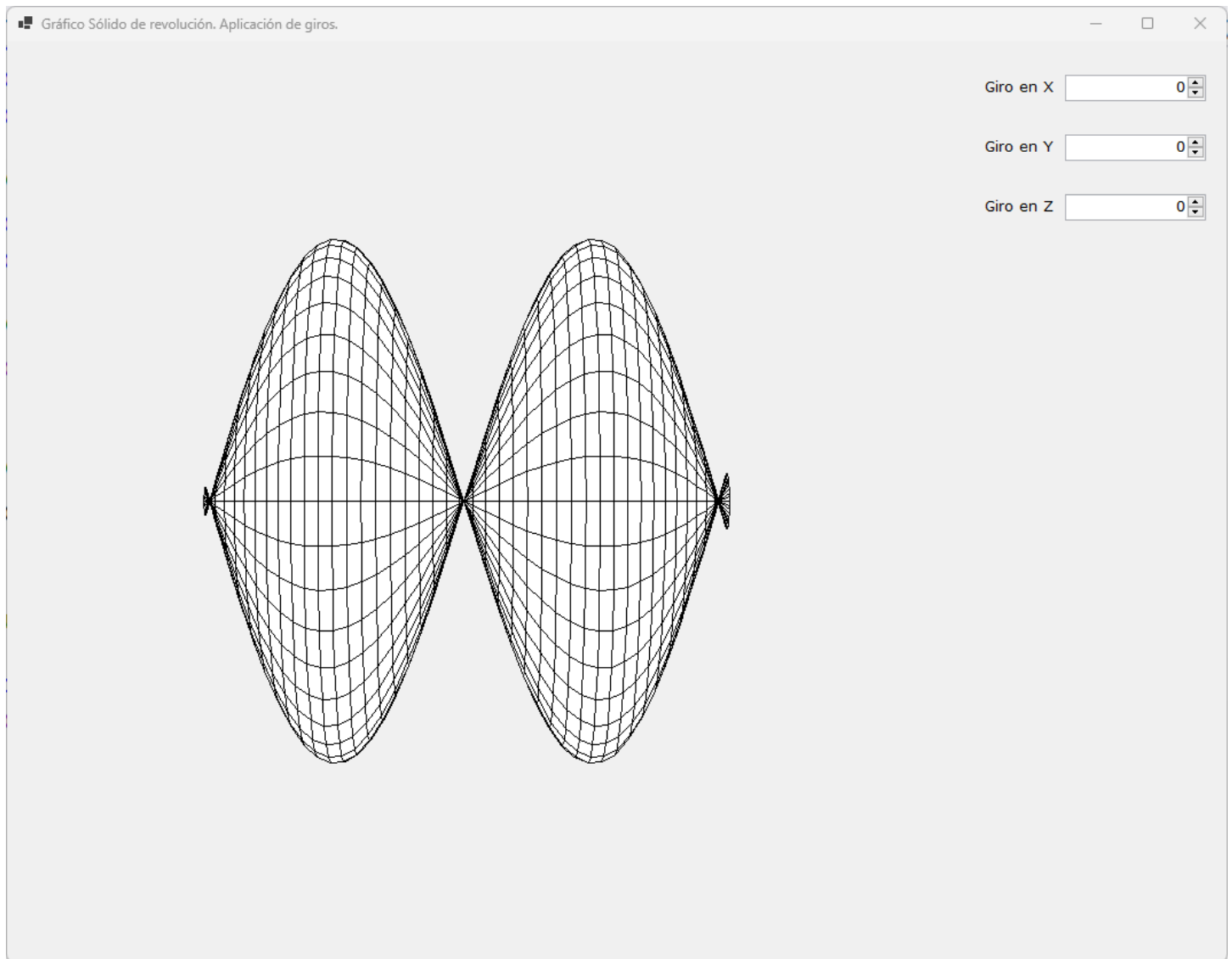


Ilustración 29: Sólido de revolución

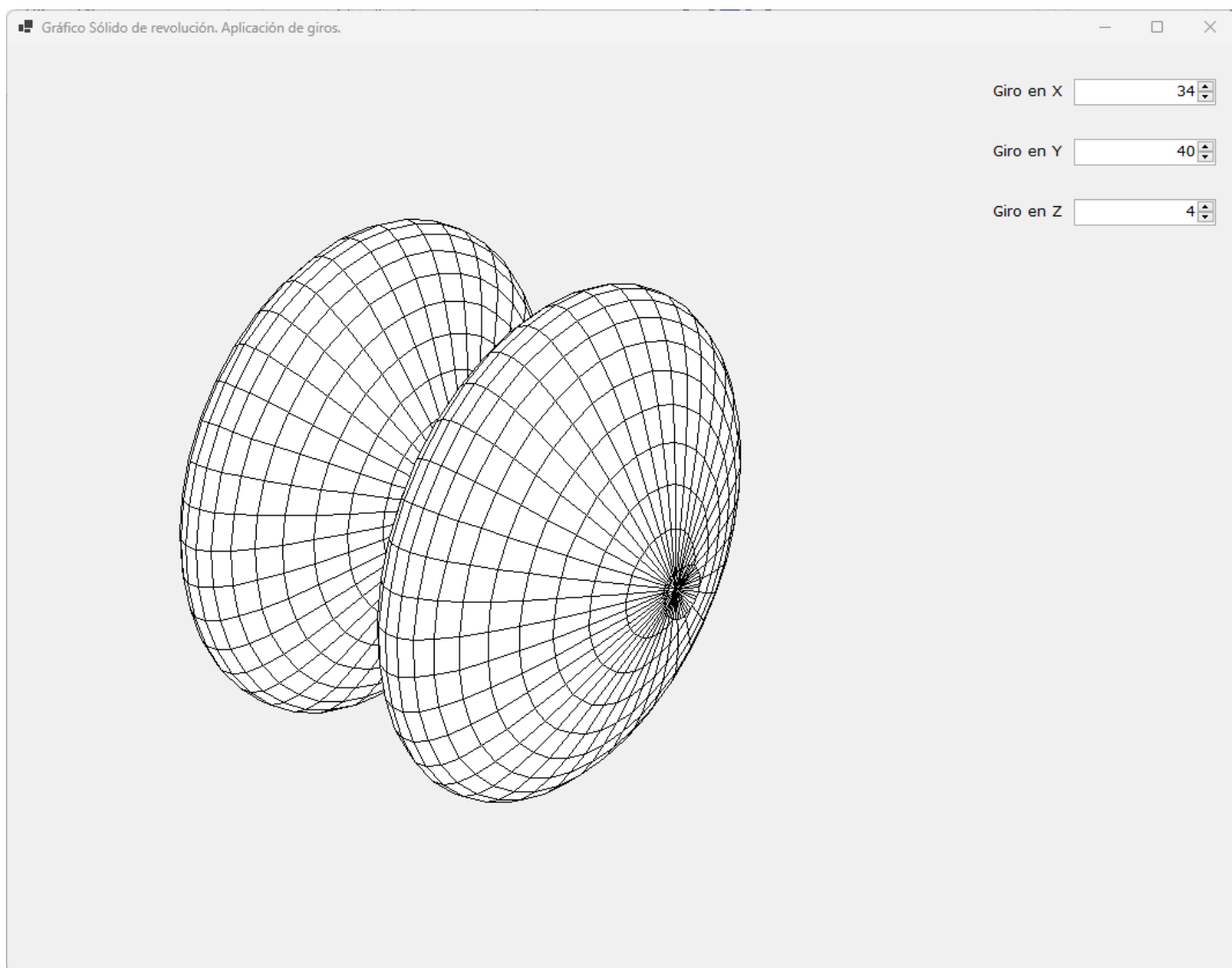


Ilustración 30: Sólido de revolución