

# C# Y .NET 8

## Parte 13. Gráficos 3D

2024-12

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 .....	5
En memoria .....	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.....	22
Giro centrado: Uso de las constantes.....	27
Combinando los tres giros: Cálculo de constantes.....	36
Combinando los tres giros: Uso de constantes .....	41
Líneas ocultas.....	50
Gráfico Matemático en 3D.....	58
Gráfico Matemático en 3D con líneas ocultas.....	67
Gráfico Matemático en 3D con colores según altura .....	76
Gráfico matemático en 3D ingresando la ecuación .....	91
Gráfico matemático en 3D, control por ratón .....	101
Gráfico Polar en 3D .....	115
Gráfico de sólido de revolución .....	126

## 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.....	20
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.....	21
Ilustración 9: Constantes para después calcular los puntos en pantalla .....	26
Ilustración 10: Diseño de ventana.....	27
Ilustración 11: Cubo proyectado por defecto.....	33
Ilustración 12: Cubo proyectado con giro en X .....	34
Ilustración 13: Cubo proyectado con giro en Y .....	35
Ilustración 14: Cubo proyectado con giro en Z .....	35
Ilustración 15: Constantes para cuadrar el cubo en la pantalla.....	40
Ilustración 16: Diseño de ventana.....	41
Ilustración 17: Cubo proyectado, sin giros.....	47
Ilustración 18: Cubo proyectado con giros.....	48
Ilustración 19: Cubo proyectado con giros.....	49
Ilustración 20: Líneas ocultas.....	56
Ilustración 21: Líneas ocultas.....	57
Ilustración 22: Diseño de ventana.....	59
Ilustración 23: Ecuación en 3D proyectada .....	60
Ilustración 24: Líneas ocultas.....	67
Ilustración 25: El valor de Z determina el color .....	76
Ilustración 26: Inclusive si gira, el valor de Z determina el color .....	77
Ilustración 27: Gráfico matemático 3D con colores .....	90
Ilustración 28: Ingresa la ecuación .....	91
Ilustración 29: Inicio de la aplicación.....	102
Ilustración 30: Se gira con el ratón .....	103
Ilustración 31: Gráfico ecuación polar 3D .....	116
Ilustración 32: Sólido de revolución.....	127

## Acerca del autor

Rafael Alberto Moreno Parra

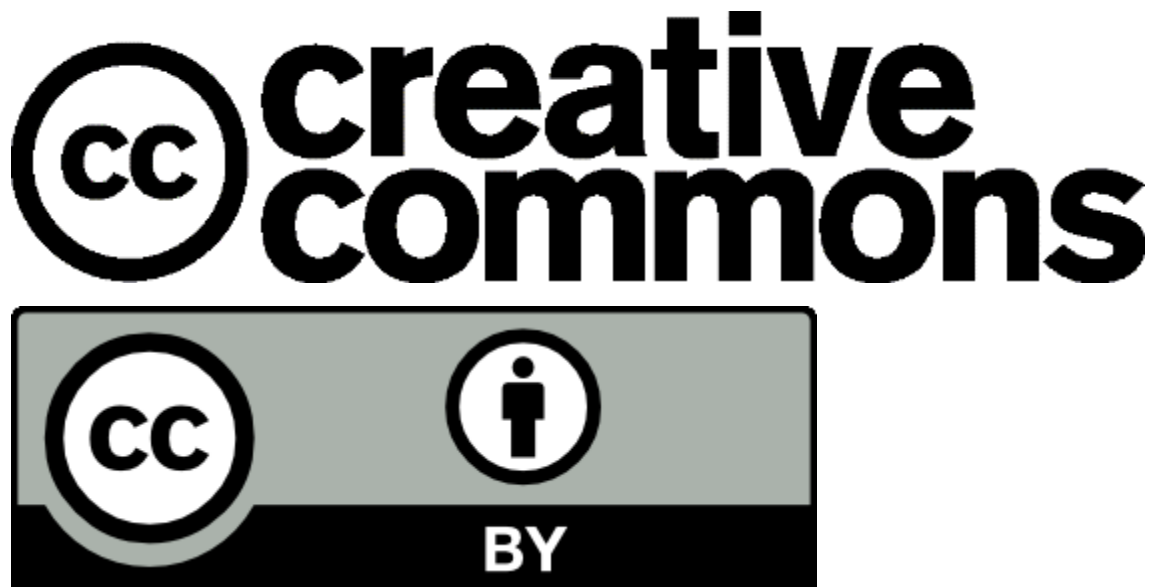
[ramsoftware@gmail.com](mailto:ramsoftware@gmail.com) o [enginelifelife@hotmail.com](mailto:enginelifelife@hotmail.com)

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/>

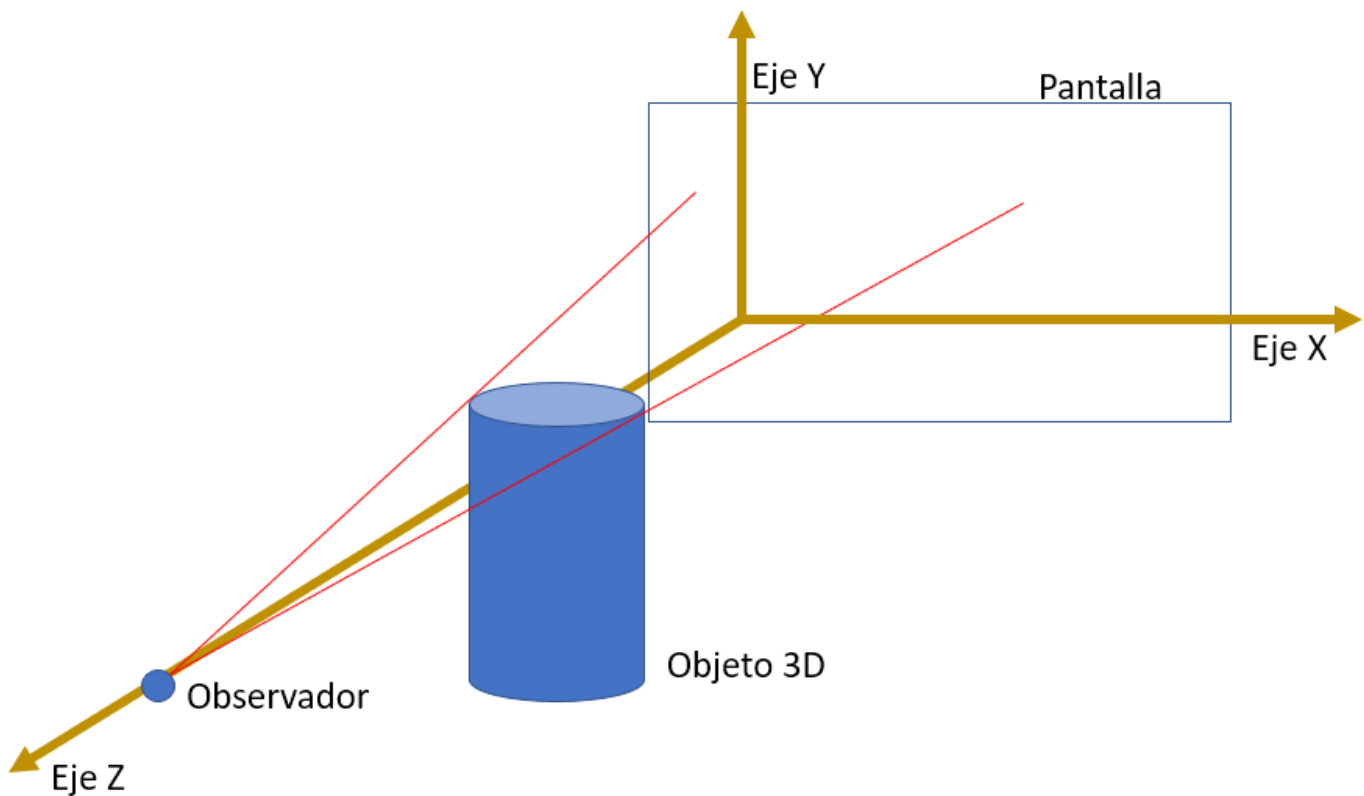
En memoria

En memoria de mi gata Capuchina.



## 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.



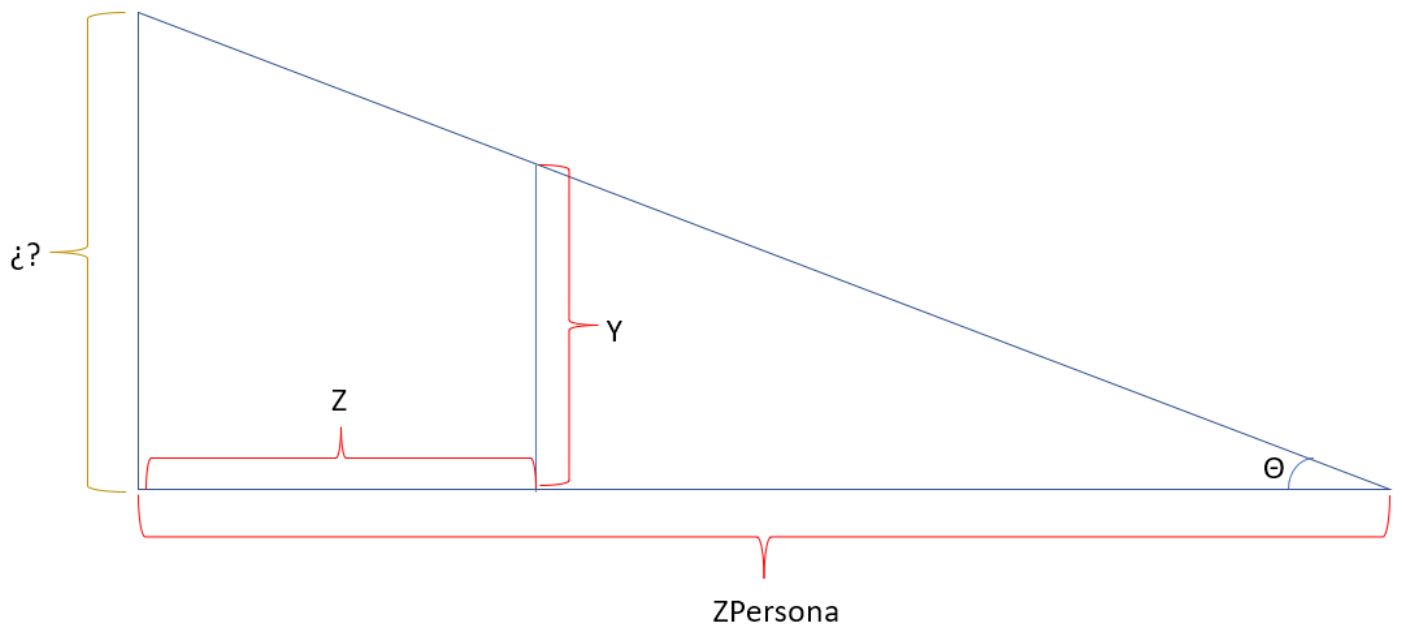
*Ilustración 1: Proyección de objeto 3D en pantalla 2D*

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  $\Theta$ , 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/001.7z/Cubo.cs

```
using System.Collections.Generic;
using System.Drawing;

namespace Graficos {
    internal class Cubo {
```

```

//Un cubo tiene 8 coordenadas espaciales X, Y, Z
private List<int> Coordenadas;

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

//Constructor
public Cubo() {
    //Ejemplo de coordenadas espaciales X,Y,Z
    Coordenadas = new List<int>() {
        50, 50, 50,
        100, 50, 50,
        100, 100, 50,
        50, 100, 50,
        50, 50, 100,
        100, 50, 100,
        100, 100, 100,
        50, 100, 100 };

    PlanoX = new List<int>();
    PlanoY = new List<int>();
}

//Convierte de 3D a 2D
public void Convierte3Da2D(int ZPersona) {
    for (int cont = 0; cont < Coordenadas.Count; cont += 3) {
        //Extrae las coordenadas espaciales
        int X = Coordenadas[cont];
        int Y = Coordenadas[cont + 1];
        int Z = Coordenadas[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
        PlanoX.Add(Xp);
        PlanoY.Add(Yp);
    }
}

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

```

```

        lienzo.DrawLine(lapiz, PlanoX[4], PlanoY[4], PlanoX[5], PlanoY[5]);
        lienzo.DrawLine(lapiz, PlanoX[5], PlanoY[5], PlanoX[6], PlanoY[6]);
        lienzo.DrawLine(lapiz, PlanoX[6], PlanoY[6], PlanoX[7], PlanoY[7]);
        lienzo.DrawLine(lapiz, PlanoX[7], PlanoY[7], PlanoX[4], PlanoY[4]);

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

```

M/001.7z/Form1.cs

```

//Proyección 3D a 2D de una figura tridimensional: un cubo
using System.Drawing;
using System.Windows.Forms;

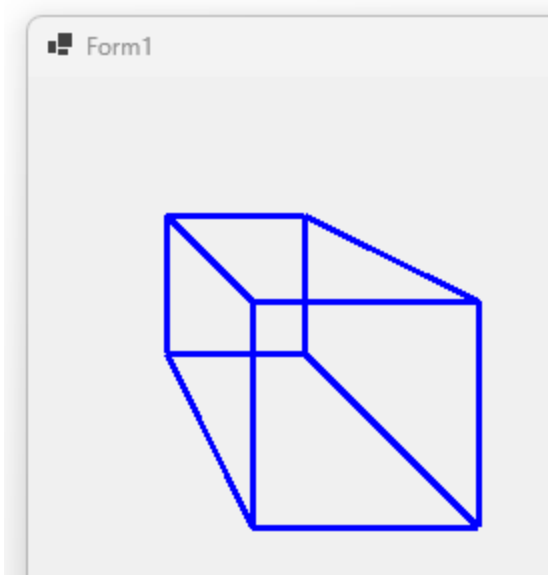
namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

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

            Cubo Figura3D = new Cubo();

            //Distancia de la persona (observador) al plano donde se proyecta la
"sombra" del cubo
            int ZPersona = 180;
            Figura3D.Convierte3Da2D(ZPersona);
            Figura3D.Dibuja(Lienzo, Lapiz);
        }
    }
}

```



*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](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:

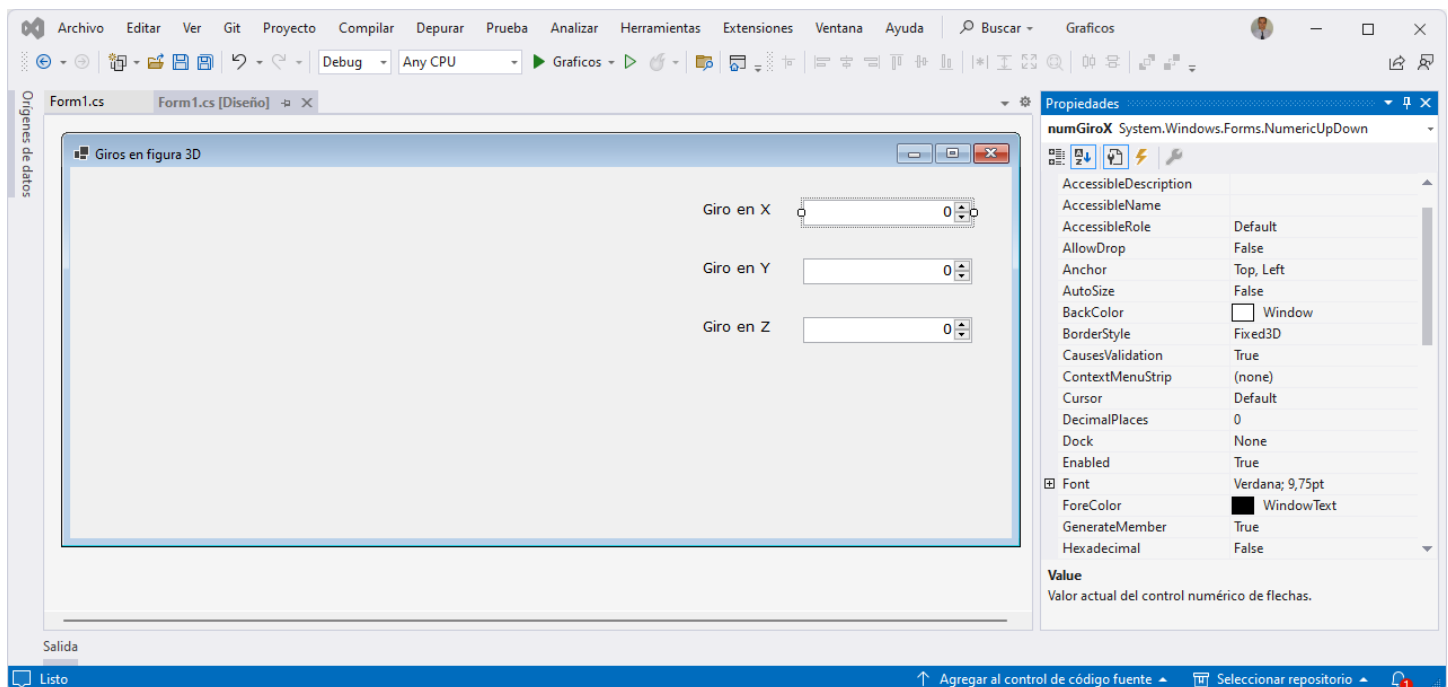


Ilustración 4: Diseño de pantalla

Se utilizan los siguientes controles:

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

```

using System;
using System.Collections.Generic;
using System.Drawing;

namespace Graficos {
    internal class Cubo {
        //Un cubo tiene 8 coordenadas espaciales X, Y, Z
        private List<int> 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<int> PlanoX;
        private List<int> PlanoY;

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

        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 angulo) {

```

```

double ang = angulo * Math.PI / 180;

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

AplicaMatrizGiro(Matriz);
}

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

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

    AplicaMatrizGiro(Matriz);
}

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

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

    AplicaMatrizGiro(Matriz);
}

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

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

        //Hace el giro
    }
}

```



```

    double posXg = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2,
0];
    double posYg = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2,
1];
    double posZg = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2,
2];

    //Guarda en la lista de giro
    Giradas.Add(posXg);
    Giradas.Add(posYg);
    Giradas.Add(posZg);
}
}

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

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

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

    lienzo.DrawLine(lapiz, PlanoX[4], PlanoY[4], PlanoX[5], PlanoY[5]);
    lienzo.DrawLine(lapiz, PlanoX[5], PlanoY[5], PlanoX[6], PlanoY[6]);
    lienzo.DrawLine(lapiz, PlanoX[6], PlanoY[6], PlanoX[7], PlanoY[7]);
    lienzo.DrawLine(lapiz, PlanoX[7], PlanoY[7], PlanoX[4], PlanoY[4]);

    lienzo.DrawLine(lapiz, PlanoX[0], PlanoY[0], PlanoX[4], PlanoY[4]);

```

```

        lienzo.DrawLine(lapiz, PlanoX[1], PlanoY[1], PlanoX[5], PlanoY[5]);
        lienzo.DrawLine(lapiz, PlanoX[2], PlanoY[2], PlanoX[6], PlanoY[6]);
        lienzo.DrawLine(lapiz, PlanoX[3], PlanoY[3], PlanoX[7], PlanoY[7]);
    }

}
}

```

M/002.7z/Form1.cs

```

using System;
using System.Drawing;
using System.Windows.Forms;

//Proyección 3D a 2D. Uso de giros en X, Y, Z pero tan sólo en uno
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, System.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();
}

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

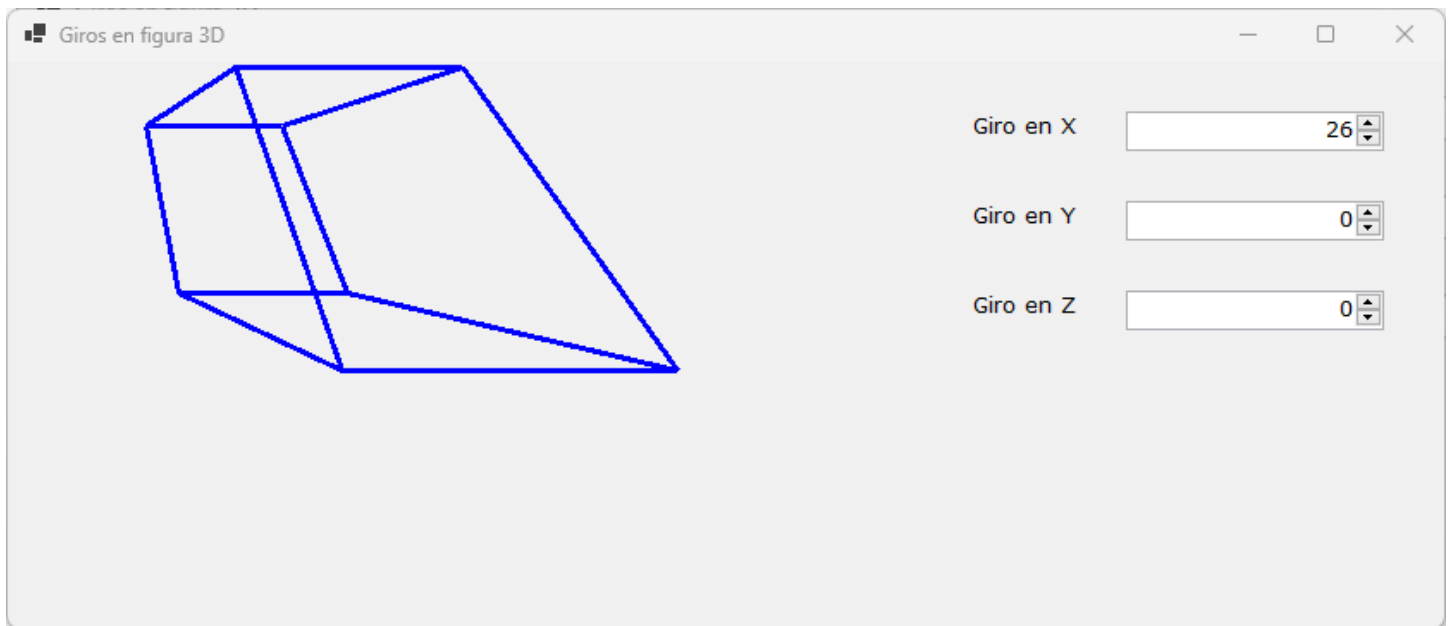
    int ZPersona = 180;
    Figura3D.Convierte3Da2D(ZPersona);
    Figura3D.Dibuja(Lienzo, Lapiz);
}
}
}

```

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/003.7z/Cubo.cs

```
using System;
using System.Collections.Generic;

namespace CalculaConstante {
    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 = 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 = new List<double>();
    PlanoX = new List<double>();
    PlanoY = new List<double>();
}

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

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

    AplicaMatrizGiro(Matriz);
}

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

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

    AplicaMatrizGiro(Matriz);
}

```

```

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

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

    AplicaMatrizGiro(Matriz);
}

private void AplicaMatrizGiro(double[,] Matriz) {
    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 posXg = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2,
0];
        double posYg = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2,
1];
        double posZg = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2,
2];

        //Guarda en la lista de giro
        Giradas.Add(posXg);
        Giradas.Add(posYg);
        Giradas.Add(posZg);
    }
}

//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);
    }

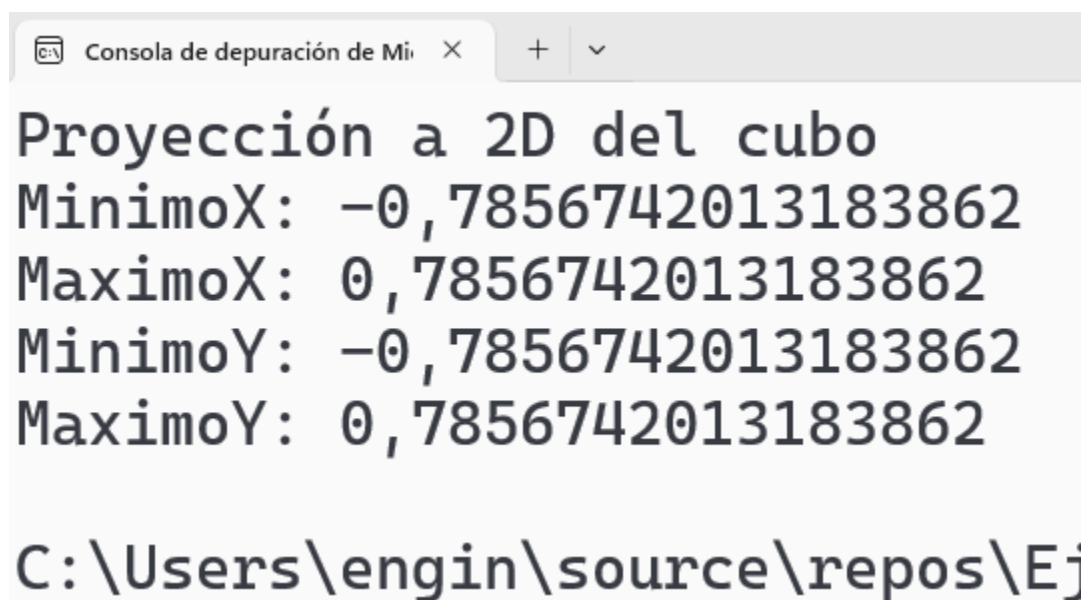
    for (int cont = 0; cont < PlanoX.Count; cont++) {
        if (PlanoX[cont] < minimoX) minimoX = PlanoX[cont];
        if (PlanoX[cont] > maximoX) maximoX = PlanoX[cont];
        if (PlanoY[cont] < minimoY) minimoY = PlanoY[cont];
        if (PlanoY[cont] > maximoY) maximoY = PlanoY[cont];
    }

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

```

```
using System;

namespace CalculaConstante {
    internal class Program {
        static void Main(string[] args) {
            Cubo Figura3D = new Cubo();
            Figura3D.CalculaExtremo(5);
            Console.ReadKey();
        }
    }
}
```



```
Consola de depuración de Mi... × + v

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:

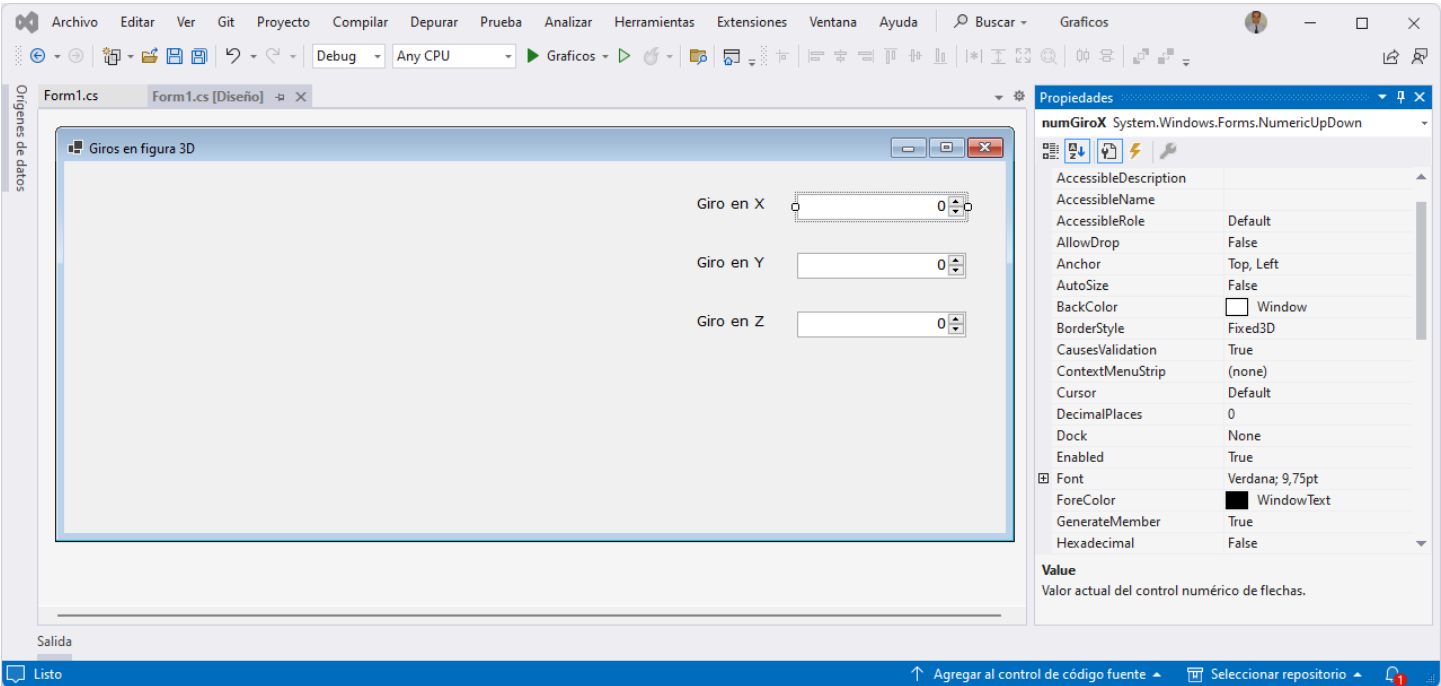


Ilustración 10: Diseño de ventana

Se utilizan los siguientes controles:

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

M/004.7z/Cubo.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;

namespace Graficos {
    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> PantallaX;
private List<int> PantallaY;

//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 = new List<double>();
    PlanoX = new List<double>();
    PlanoY = new List<double>();
    PantallaX = new List<int>();
    PantallaY = new List<int>();
}

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 angulo) {
    double ang = angulo * Math.PI / 180;

    double[,] Matriz = new double[3, 3] {
        {1, 0, 0},

```

```

        {0, Math.Cos(ang), Math.Sin(ang)},
        {0, -Math.Sin(ang), Math.Cos(ang) }
    };

    AplicaMatrizGiro(Matriz);
}

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

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

    AplicaMatrizGiro(Matriz);
}

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

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

    AplicaMatrizGiro(Matriz);
}

private void AplicaMatrizGiro(double[,] Matriz) {
    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 posXg = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2,
0];
        double posYg = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2,
1];
    }
}

```

```

    double posZg = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2,
2];

    //Guarda en la lista de giro
    Giradas.Add(posXg);
    Giradas.Add(posYg);
    Giradas.Add(posZg);
}
}

//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 XpantallaIni, int YpantallaIni, int
XpantallaFin, int YpantallaFin) {
    //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 convierteX = (XpantallaFin - XpantallaIni) / (maximoX -
minimoX);
    double convierteY = (YpantallaFin - YpantallaIni) / (maximoY -
minimoY);

    //Deduce las coordenadadas de pantalla
    PantallaX.Clear();

```

```

    PantallaY.Clear();
    for (int cont = 0; cont < PlanoX.Count; cont++) {
        int Xpantalla = Convert.ToInt32(convierteX * (PlanoX[cont] -
minimoX) + XpantallaIni);
        int Ypantalla = Convert.ToInt32(convierteY * (PlanoY[cont] -
minimoY) + YpantallaIni);
        PantallaX.Add(Xpantalla);
        PantallaY.Add(Ypantalla);
    }
}

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

    lienzo.DrawLine(lapiz, PantallaX[4], PantallaY[4], PantallaX[5],
PantallaY[5]);
    lienzo.DrawLine(lapiz, PantallaX[5], PantallaY[5], PantallaX[6],
PantallaY[6]);
    lienzo.DrawLine(lapiz, PantallaX[6], PantallaY[6], PantallaX[7],
PantallaY[7]);
    lienzo.DrawLine(lapiz, PantallaX[7], PantallaY[7], PantallaX[4],
PantallaY[4]);

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

```

```

using System;
using System.Drawing;
using System.Windows.Forms;

//Proyección 3D a 2D y giros. Figura centrada
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, System.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();
        }

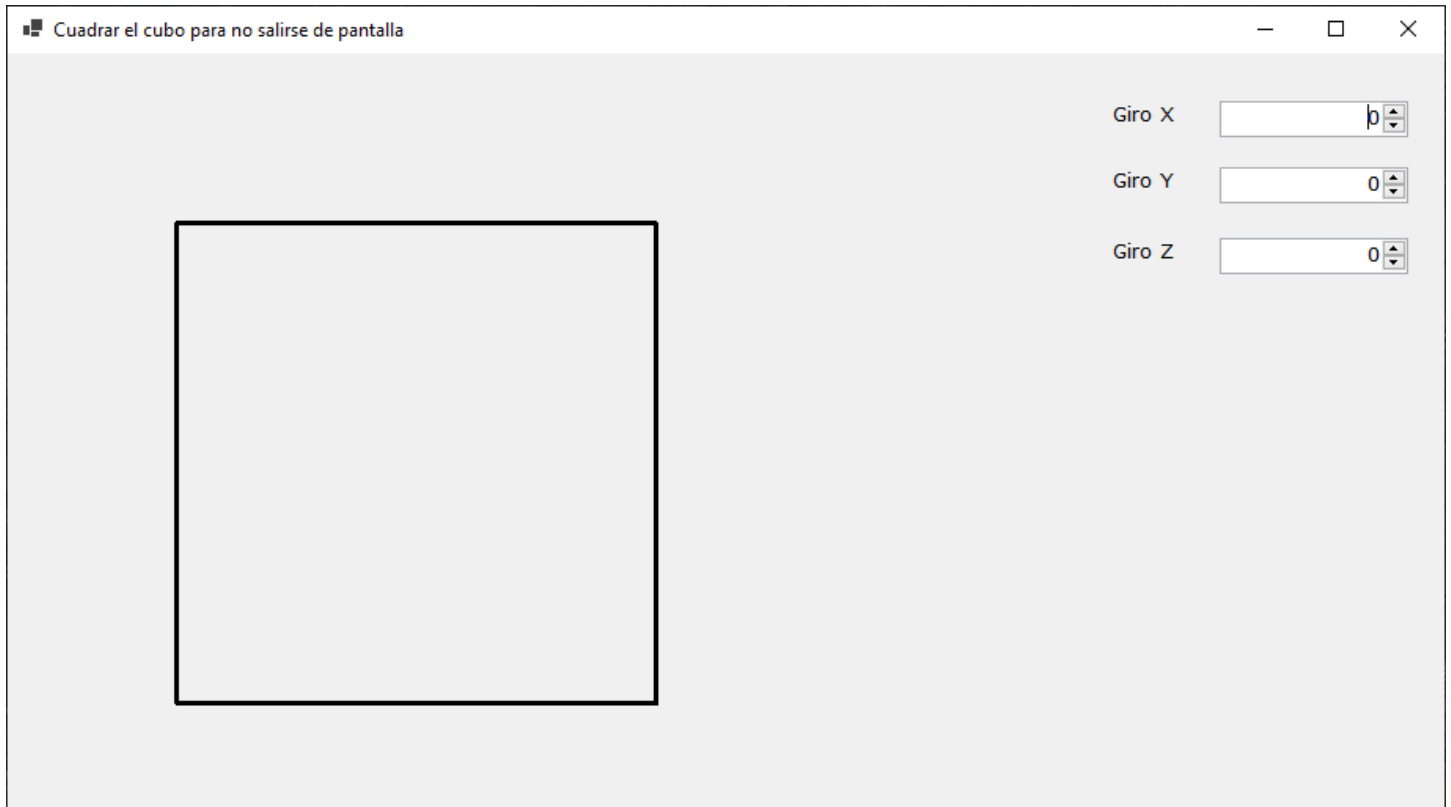
        //Pinta la proyección
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapiz = new Pen(Color.Black, 2);

```

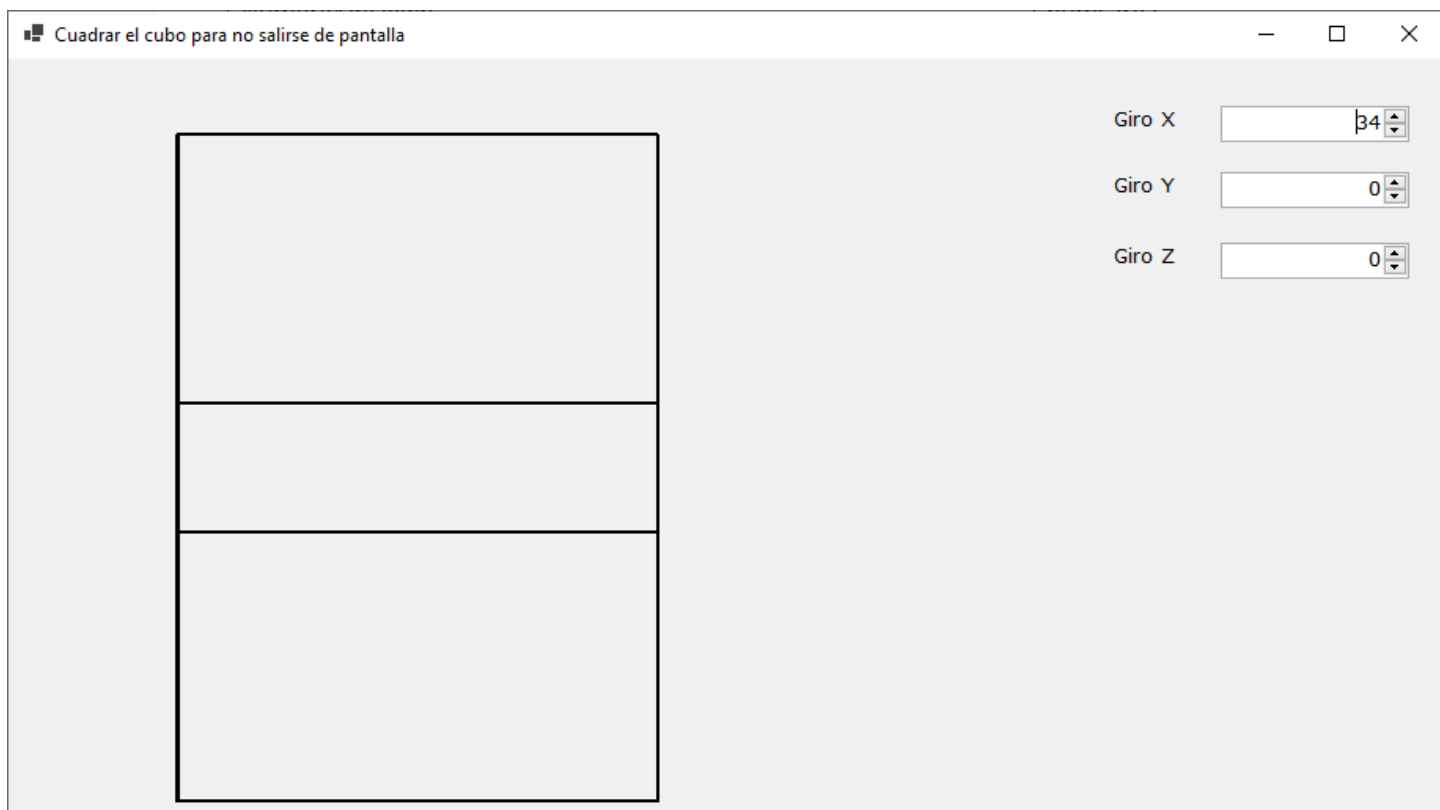


```
int ZPersona = 180;  
Figura3D.Convierte3Da2D(ZPersona);  
Figura3D.CuadraPantalla(20, 20, 500, 500);  
Figura3D.Dibuja(Lienzo, Lapiz);  
}  
}  
}
```

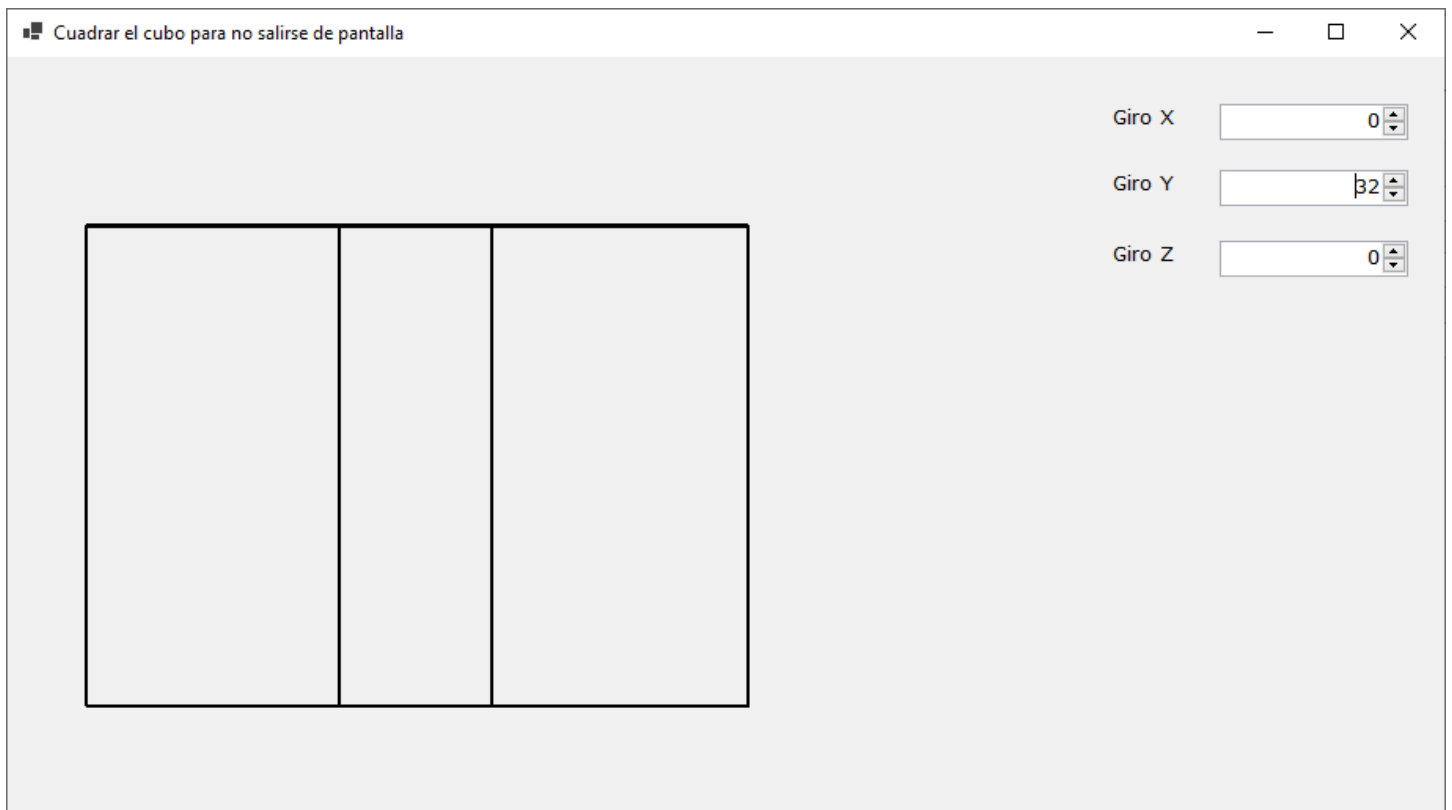
Ejemplo de ejecución:



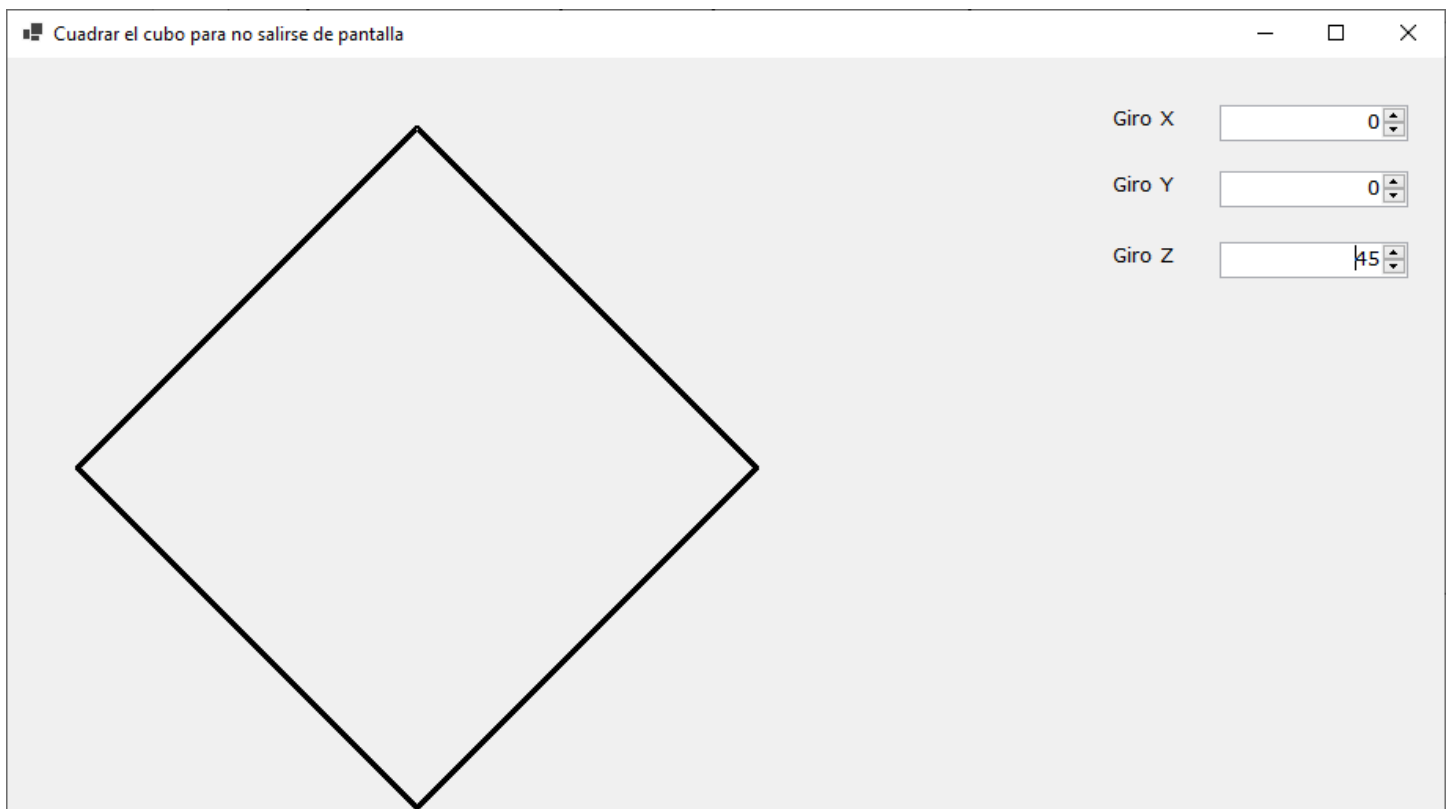
*Ilustración 11: Cubo proyectado por defecto*



*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/005.7z/Cubo.cs

```
using System;
using System.Collections.Generic;

namespace CalculaConstante {
    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 = 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 = new List<double>();
            PlanoX = new List<double>();
            PlanoY = new List<double>();
        }

        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[,] 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 }
};

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 posXg = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2,
0];
    double posYg = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2,
1];
    double posZg = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2,
2];

    //Guarda en la lista de giro
    Giradas.Add(posXg);
    Giradas.Add(posYg);
    Giradas.Add(posZg);
}

//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("Angulo X: " + angX.ToString());
    }

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

```

```

using System;

namespace CalculaConstante {
    internal class Program {
        static void Main(string[] args) {
            Cubo Figura3D = new Cubo();
            Figura3D.CalculaExtremo(5);
            Console.ReadLine();
        }
    }
}

```

```

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:

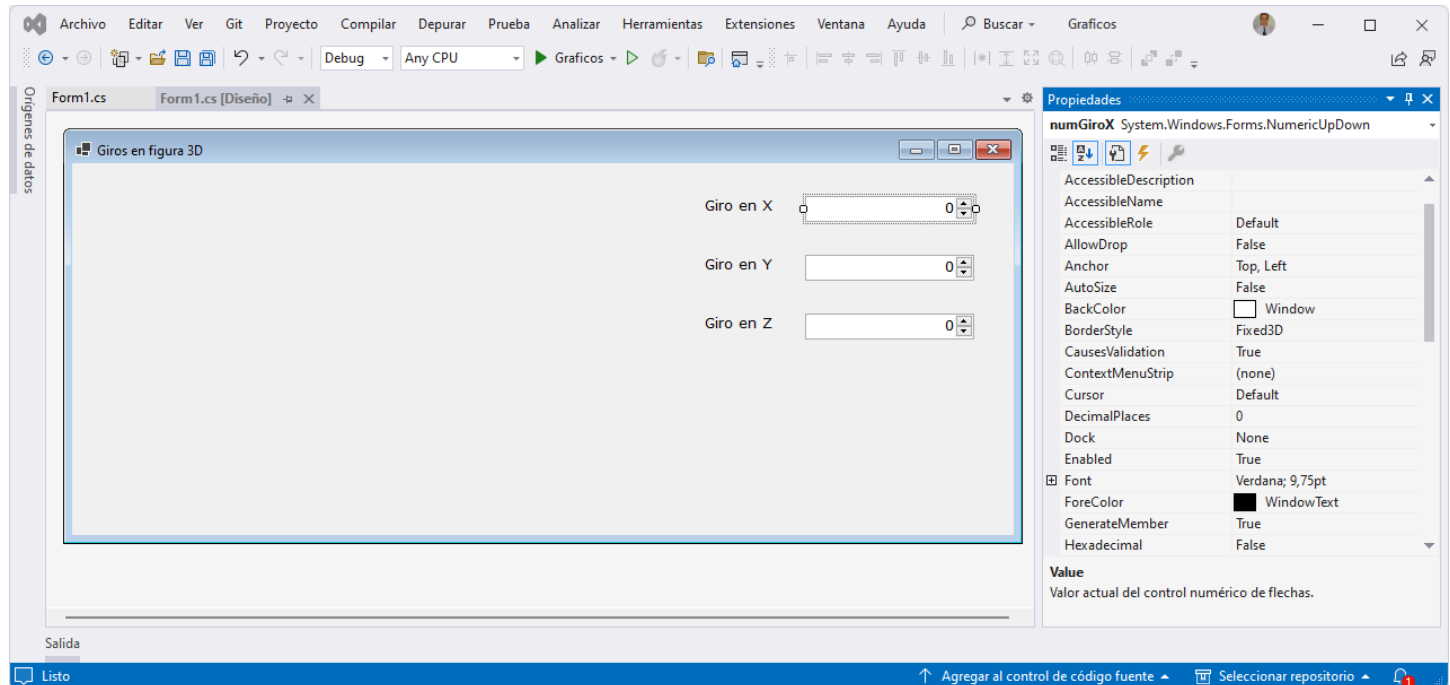


Ilustración 16: Diseño de ventana

Se utilizan los siguientes controles:

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

```

using System;
using System.Collections.Generic;
using System.Drawing;

namespace Graficos {
    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> PantallaX;
        private List<int> PantallaY;

        //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 = new List<double>();
            PlanoX = new List<double>();
            PlanoY = new List<double>();
            PantallaX = new List<int>();
            PantallaY = new List<int>();
        }

        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[,] 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 }
};

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 posXg = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2,
0];
    double posYg = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2,
1];
    double posZg = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2,
2];

    //Guarda en la lista de giro
    Giradas.Add(posXg);
    Giradas.Add(posYg);
    Giradas.Add(posZg);
}

//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 XpantallaIni, int YpantallaIni, int
XpantallaFin, int YpantallaFin) {
    //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 convierteX = (XpantallaFin - XpantallaIni) / (maximoX -
minimoX);
    double convierteY = (YpantallaFin - YpantallaIni) / (maximoY -
minimoY);

    //Deduce las coordenadadas de pantalla
    PantallaX.Clear();
    PantallaY.Clear();
    for (int cont = 0; cont < PlanoX.Count; cont++) {
        int Xpantalla = Convert.ToInt32(convierteX * (PlanoX[cont] -
minimoX) + XpantallaIni);
        int Ypantalla = Convert.ToInt32(convierteY * (PlanoY[cont] -
minimoY) + YpantallaIni);
        PantallaX.Add(Xpantalla);
        PantallaY.Add(Ypantalla);
    }
}

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

```

```

        lienzo.DrawLine(lapiz, PantallaX[4], PantallaY[4], PantallaX[5],
PantallaY[5]);
        lienzo.DrawLine(lapiz, PantallaX[5], PantallaY[5], PantallaX[6],
PantallaY[6]);
        lienzo.DrawLine(lapiz, PantallaX[6], PantallaY[6], PantallaX[7],
PantallaY[7]);
        lienzo.DrawLine(lapiz, PantallaX[7], PantallaY[7], PantallaX[4],
PantallaY[4]);

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

```

M/006.7z/Form1.cs

```

using System;
using System.Drawing;
using System.Windows.Forms;

//Proyección 3D a 2D y giros en los tres ejes simultáneamente
namespace Graficos {
    public partial class Form1 : Form {
        //El cubo que se proyecta y gira
        Cubo Figura3D;

        public Form1() {
            InitializeComponent();

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

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

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

```

```

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

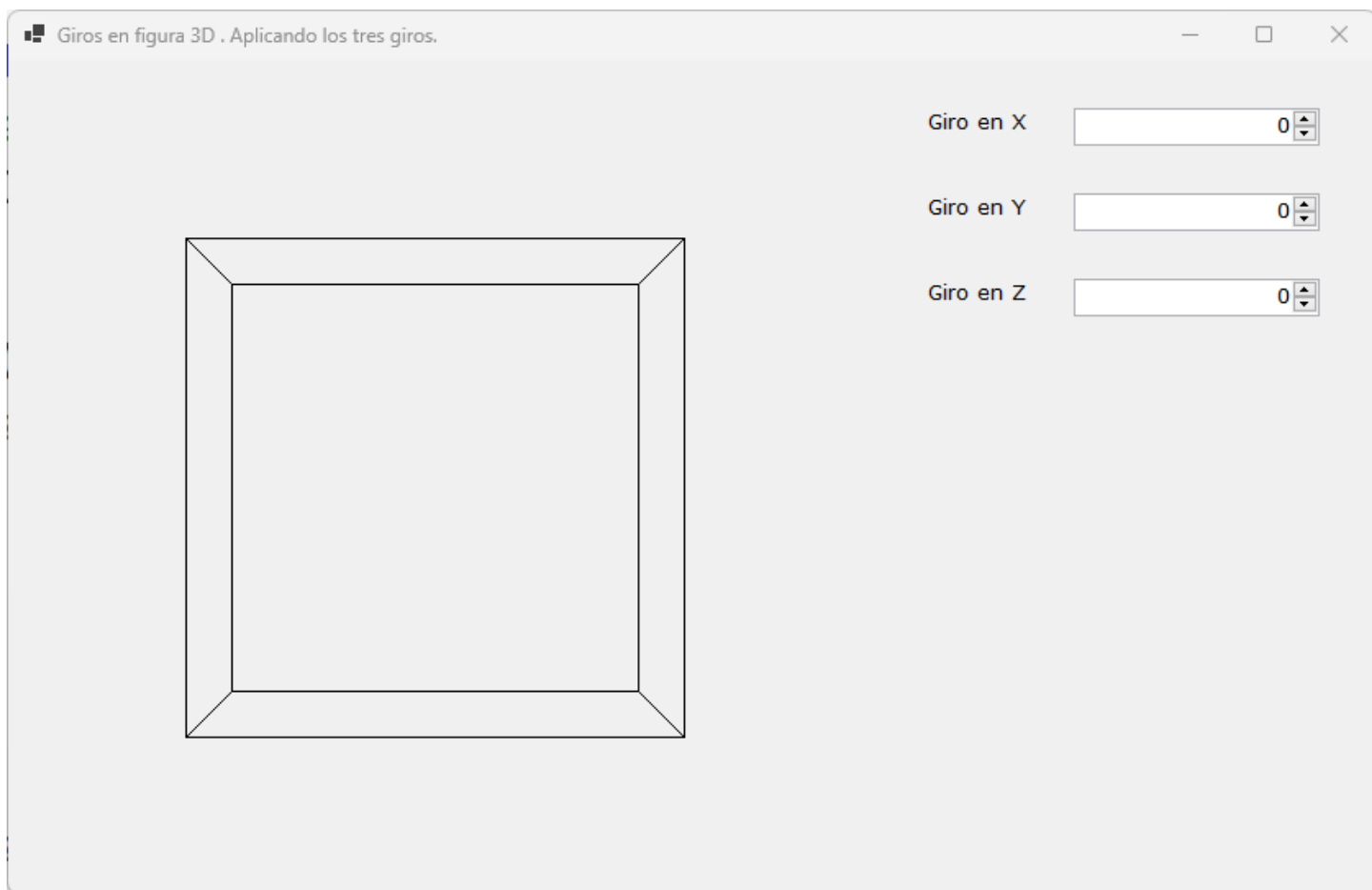
//Pinta la proyección del cubo
private void Form1_Paint(object sender, PaintEventArgs e) {
    Graphics Lienzo = e.Graphics;
    Pen Lapis = new Pen(Color.Black, 1);

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

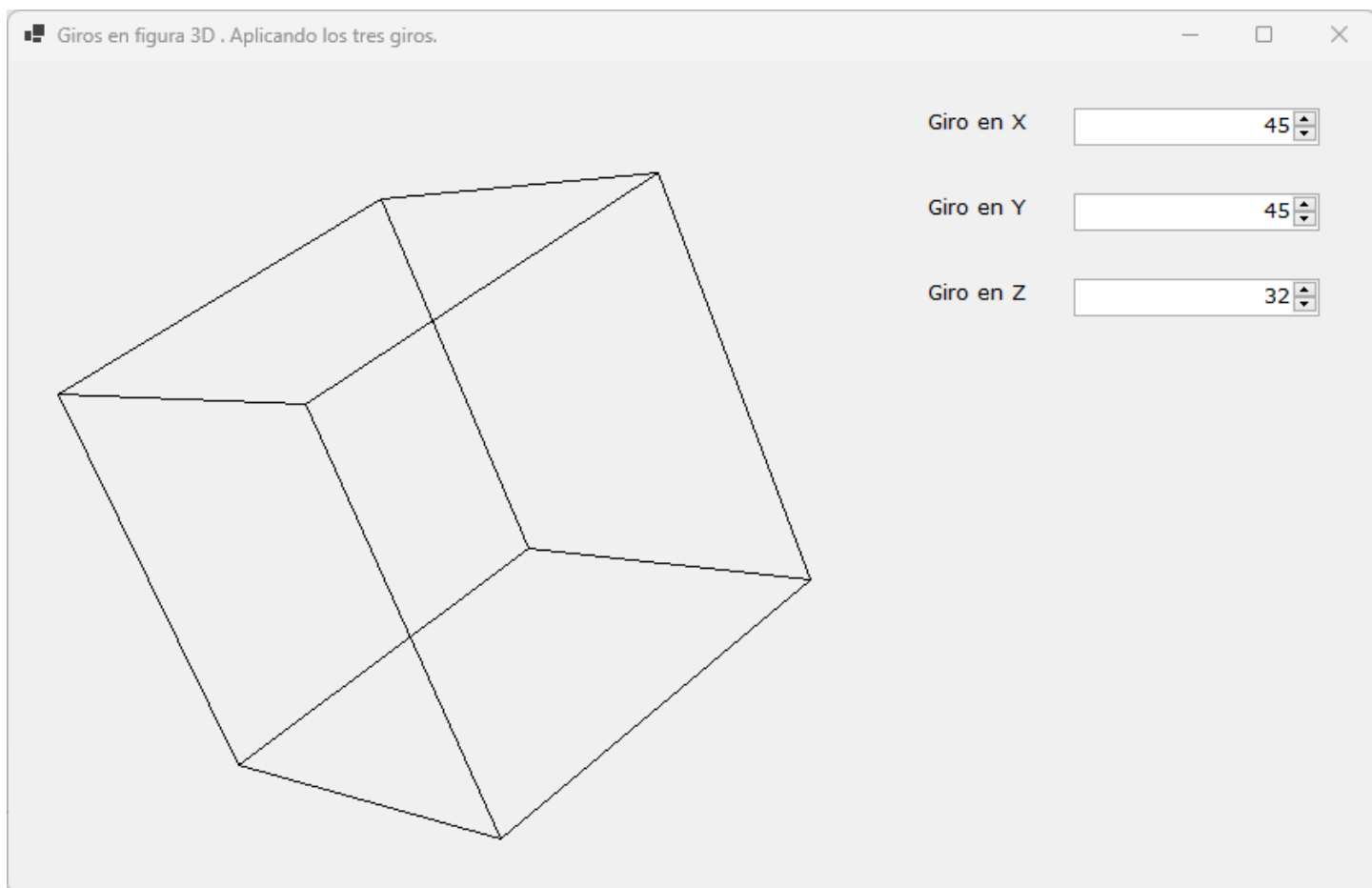
    int ZPersona = 5;
    Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ);
    Figura3D.Convierte3Da2D(ZPersona);
    Figura3D.CuadraPantalla(20, 20, 500, 500);
    Figura3D.Dibuja(Lienzo, Lapis);
}
}
}

```

Ejemplo de ejecución:

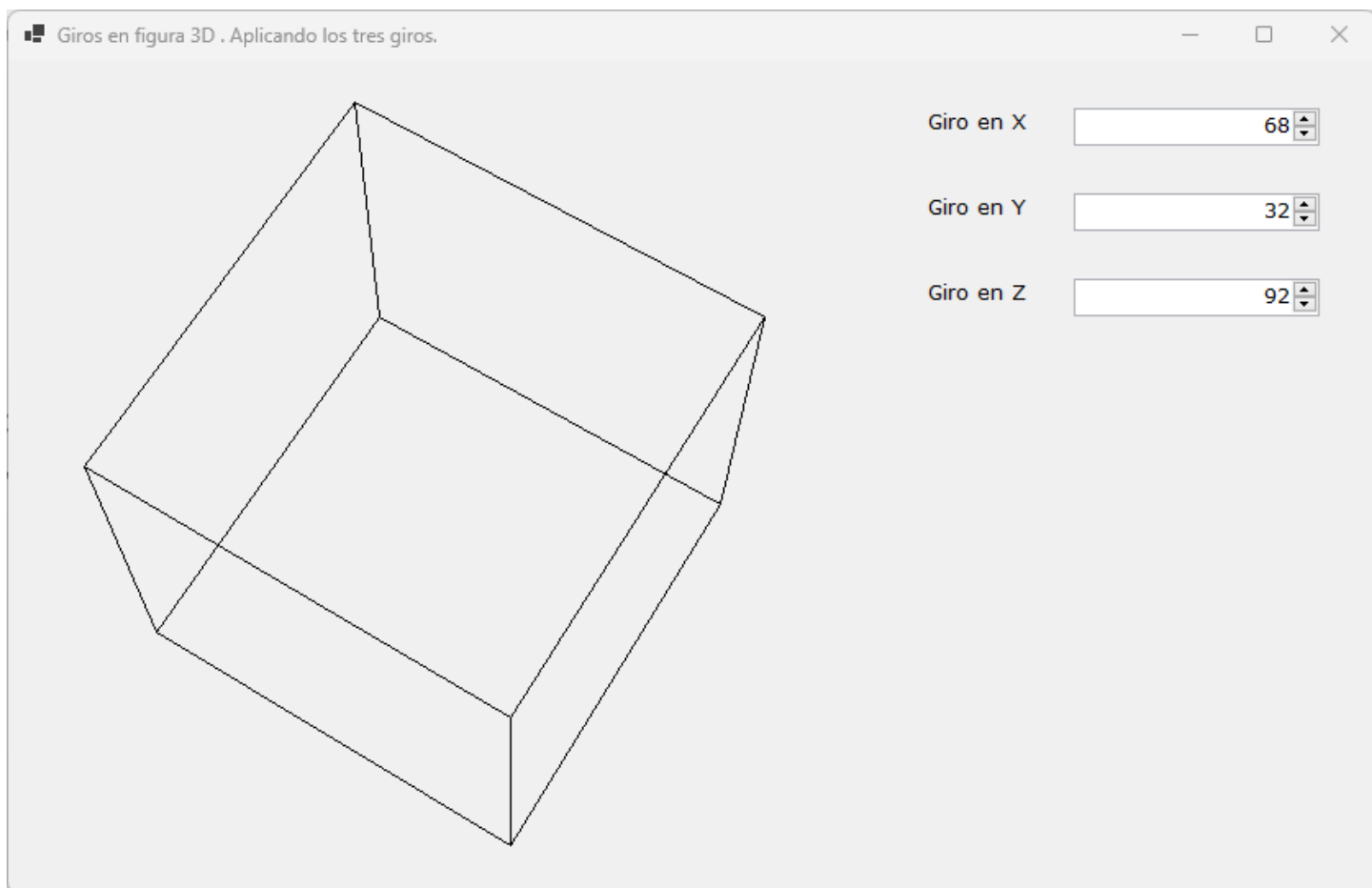


*Ilustración 17: Cubo proyectado, sin giros*



*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/007.7z/Cubo.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;

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

        //Constructor
        public Cubo() {
            poligonos = new List<Poligono>();

            //Polígonos que forman el cubo
            poligonos.Add(new Poligono(-0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5, -0.5,
-0.5, -0.5, -0.5, -0.5));
            poligonos.Add(new Poligono(-0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5,
0.5, -0.5, -0.5, 0.5));

            poligonos.Add(new Poligono(-0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, -
0.5, 0.5, -0.5, -0.5, -0.5));
            poligonos.Add(new Poligono(0.5, 0.5, -0.5, 0.5, 0.5, 0.5, 0.5, -0.5,
0.5, 0.5, -0.5, -0.5));

            poligonos.Add(new Poligono(-0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5, 0.5,
0.5, -0.5, 0.5, 0.5));
            poligonos.Add(new Poligono(-0.5, -0.5, -0.5, 0.5, -0.5, -0.5, 0.5, -
0.5, 0.5, -0.5, -0.5, 0.5));
        }
    }
}
```

```

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[,] 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 }
    };

    //Gira los 8 polígonos
    for (int cont = 0; cont < poligonos.Count; cont++) {
        poligonos[cont].Girar(Matriz);
    }
}

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

//Convierte las coordenadas planas en coordenadas de pantalla
public void CuadraPantalla(int XpantallaIni, int YpantallaIni, int
XpantallaFin, int YpantallaFin) {
    //Los valores extremos de las coordenadas del cubo
    double MaximoX = 0.87931543769177811;
    double MinimoX = -0.87931543769177811;
    double MaximoY = 0.87931543769177811;
    double MinimoY = -0.87931543769177811;

    //Las constantes de transformación

```

```

        double ConstanteX = (XpantallaFin - XpantallaIni) / (MaximoX -
MinimoX);
        double ConstanteY = (YpantallaFin - YpantallaIni) / (MaximoY -
MinimoY);

        for (int cont = 0; cont < poligonos.Count; cont++) {
            poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, MinimoX,
MinimoY, XpantallaIni, YpantallaIni);
        }

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

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

```

M/007.7z/Poligono.cs

```

using System;
using System.Drawing;

namespace Graficos {
    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 PosX1g, PosY1g, PosZ1g, PosX2g, PosY2g, PosZ2g, PosX3g,
PosY3g, PosZ3g, PosX4g, PosY4g, PosZ4g;

        //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 [,] Matriz) {
    PosX1g = X1 * Matriz[0,0] + Y1 * Matriz[1,0] + Z1 * Matriz[2,0];
    PosY1g = X1 * Matriz[0,1] + Y1 * Matriz[1,1] + Z1 * Matriz[2,1];
    PosZ1g = X1 * Matriz[0,2] + Y1 * Matriz[1,2] + Z1 * Matriz[2,2];

    PosX2g = X2 * Matriz[0,0] + Y2 * Matriz[1,0] + Z2 * Matriz[2,0];
    PosY2g = X2 * Matriz[0,1] + Y2 * Matriz[1,1] + Z2 * Matriz[2,1];
    PosZ2g = X2 * Matriz[0,2] + Y2 * Matriz[1,2] + Z2 * Matriz[2,2];

    PosX3g = X3 * Matriz[0,0] + Y3 * Matriz[1,0] + Z3 * Matriz[2,0];
    PosY3g = X3 * Matriz[0,1] + Y3 * Matriz[1,1] + Z3 * Matriz[2,1];
    PosZ3g = X3 * Matriz[0,2] + Y3 * Matriz[1,2] + Z3 * Matriz[2,2];

    PosX4g = X4 * Matriz[0,0] + Y4 * Matriz[1,0] + Z4 * Matriz[2,0];
    PosY4g = X4 * Matriz[0,1] + Y4 * Matriz[1,1] + Z4 * Matriz[2,1];
    PosZ4g = X4 * Matriz[0,2] + Y4 * Matriz[1,2] + Z4 * Matriz[2,2];

    Centro = (PosZ1g + PosZ2g + PosZ3g + PosZ4g) / 4;
}

//Convierte de 3D a 2D (segunda dimensión)
public void Convierte3Da2D(double ZPersona) {
    X1sd = PosX1g * ZPersona / (ZPersona - PosZ1g);
    Y1sd = PosY1g * ZPersona / (ZPersona - PosZ1g);

    X2sd = PosX2g * ZPersona / (ZPersona - PosZ2g);
    Y2sd = PosY2g * ZPersona / (ZPersona - PosZ2g);

    X3sd = PosX3g * ZPersona / (ZPersona - PosZ3g);
    Y3sd = PosY3g * ZPersona / (ZPersona - PosZ3g);

    X4sd = PosX4g * ZPersona / (ZPersona - PosZ4g);
    Y4sd = PosY4g * ZPersona / (ZPersona - PosZ4g);
}

```

```

//Cuadra en pantalla física
public void CuadraPantalla(double ConstanteX, double ConstanteY, double
MinimoX, double MinimoY, int XPantallaIni, int YPantallaIni) {
    X1p = Convert.ToInt32(ConstanteX * (X1sd - MinimoX) + XPantallaIni);
    Y1p = Convert.ToInt32(ConstanteY * (Y1sd - MinimoY) + YPantallaIni);

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

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

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

//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 Point(X1p, Y1p);
    Point Punto2 = new Point(X2p, Y2p);
    Point Punto3 = new Point(X3p, Y3p);
    Point Punto4 = new Point(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;
}
}
}

```

M/007.7z/Form1.cs

```

using System;
using System.Drawing;
using System.Windows.Forms;
//Proyección 3D a 2D y giros en los tres ejes simultáneamente. Líneas
ocultas.

```

```

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

        public Form1() {
            InitializeComponent();

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

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

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

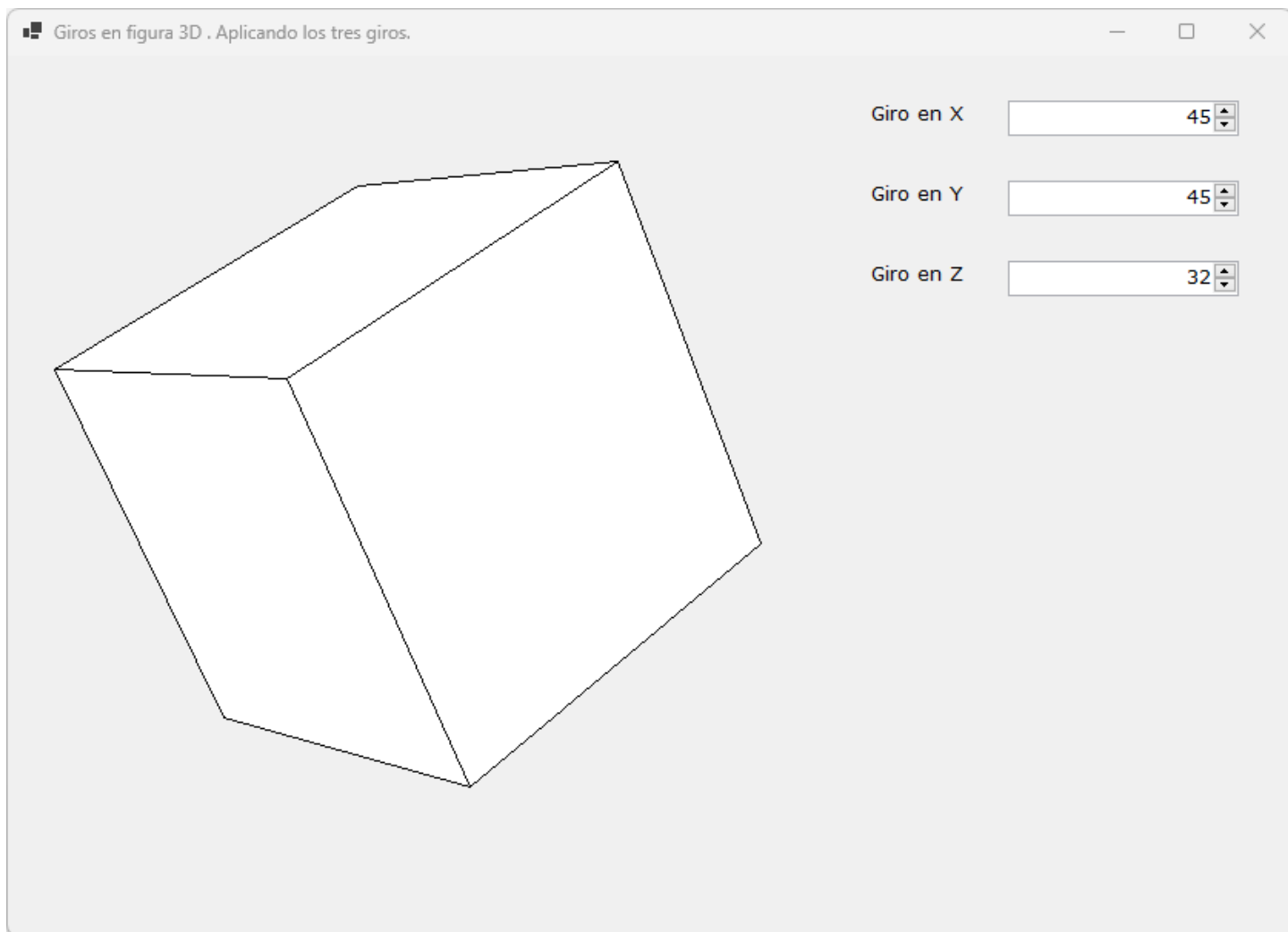
        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            Refresh();
        }

        //Pinta la proyección del cubo
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;
            Pen Lapis = new Pen(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);

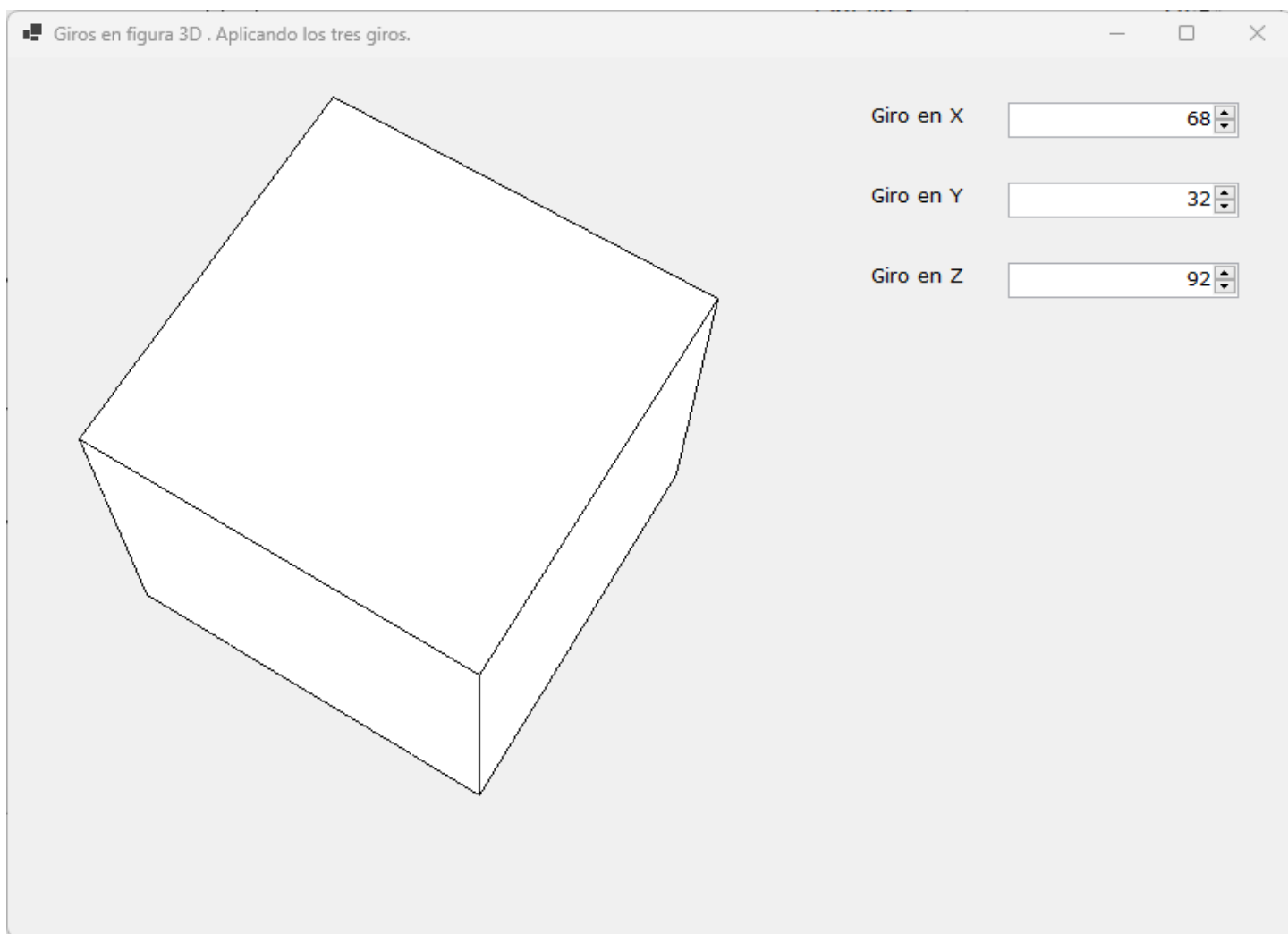
            int ZPersona = 5;
            Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ);
            Figura3D.Convierte3Da2D(ZPersona);
            Figura3D.CuadraPantalla(20, 20, 500, 500);
            Figura3D.Dibuja(Lienzo, Lapis, Relleno);
        }
    }
}

```



*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 mínimo de X (MinX) hasta el valor máximo de X (MaxX). Igual sucede con Y, el valor mínimo de Y (MinY) hasta el valor máximo de Y (MaxY). También se pregunta cuantas líneas tendrá el gráfico (NumLineas), entre más líneas, más detallado será el gráfico.

### Paso 2

Se calcula el avance en X que sería:  $\text{IncrX} = (\text{MaxX} - \text{MinX}) / \text{NumLineas}$

Se calcula el avance en Y que sería:  $\text{IncrY} = (\text{MaxY} - \text{MinY}) / \text{NumLineas}$

El gráfico requiere una matriz bidimensional de objetos. Cada objeto tiene:

- Los puntos X, Y, Z (Z es calculado)
- Los puntos Xg, Yg, Zg (más adelante se calculan)
- Los puntos planoX, planoY (más adelante se calculan)
- Los puntos Xp, Yp (más adelante se calculan)

```
for (int Fila = 0; Fila < NumLineas; Fila++) {  
    X = MinX;  
    for (int Columna = 0; Columna < NumLineas; Columna++) {  
        MtPunto[Fila][Columna].Calcular(X, Y);  
        X += IncrX;  
    }  
    Y += IncrY;  
}
```

### Paso 3

Se normalizan los valores de X, Y, Z de cada objeto, para que queden entre 0 y 1, luego se le resta -0.5 ¿Para qué? Para que los puntos 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  $X_g$ ,  $Y_g$ ,  $Z_g$  para cada objeto.

#### Paso 5

Con  $X_g$ ,  $Y_g$ ,  $Z_g$  se proyecta a la pantalla, obteniéndose el  $\text{planoX}$ ,  $\text{planoY}$  para cada objeto.

#### Paso 6

Con  $\text{planoX}$ ,  $\text{planoY}$  se aplican las constantes

```
//Los valores extremos de las coordenadas del cubo
double MaximoX = 0.87931543769177811;
double MinimoX = -0.87931543769177811;
double MaximoY = 0.87931539875237918;
double MinimoY = -0.87931539875237918;
```

y se obtiene la proyección en pantalla, es decir,  $X_p$  y  $Y_p$  para cada objeto.

#### Paso 7

Se conectan con líneas los puntos  $X_p$ ,  $Y_p$  generando la malla.

La aplicación completa es 008.7z

Se muestra aquí las clases y los controles gráficos:

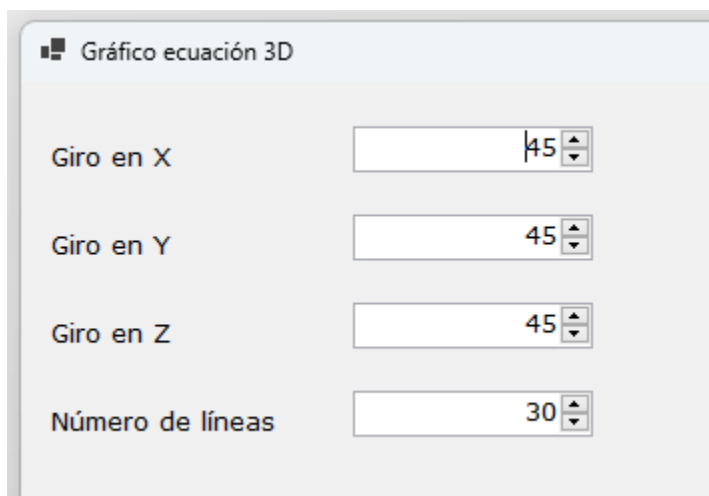
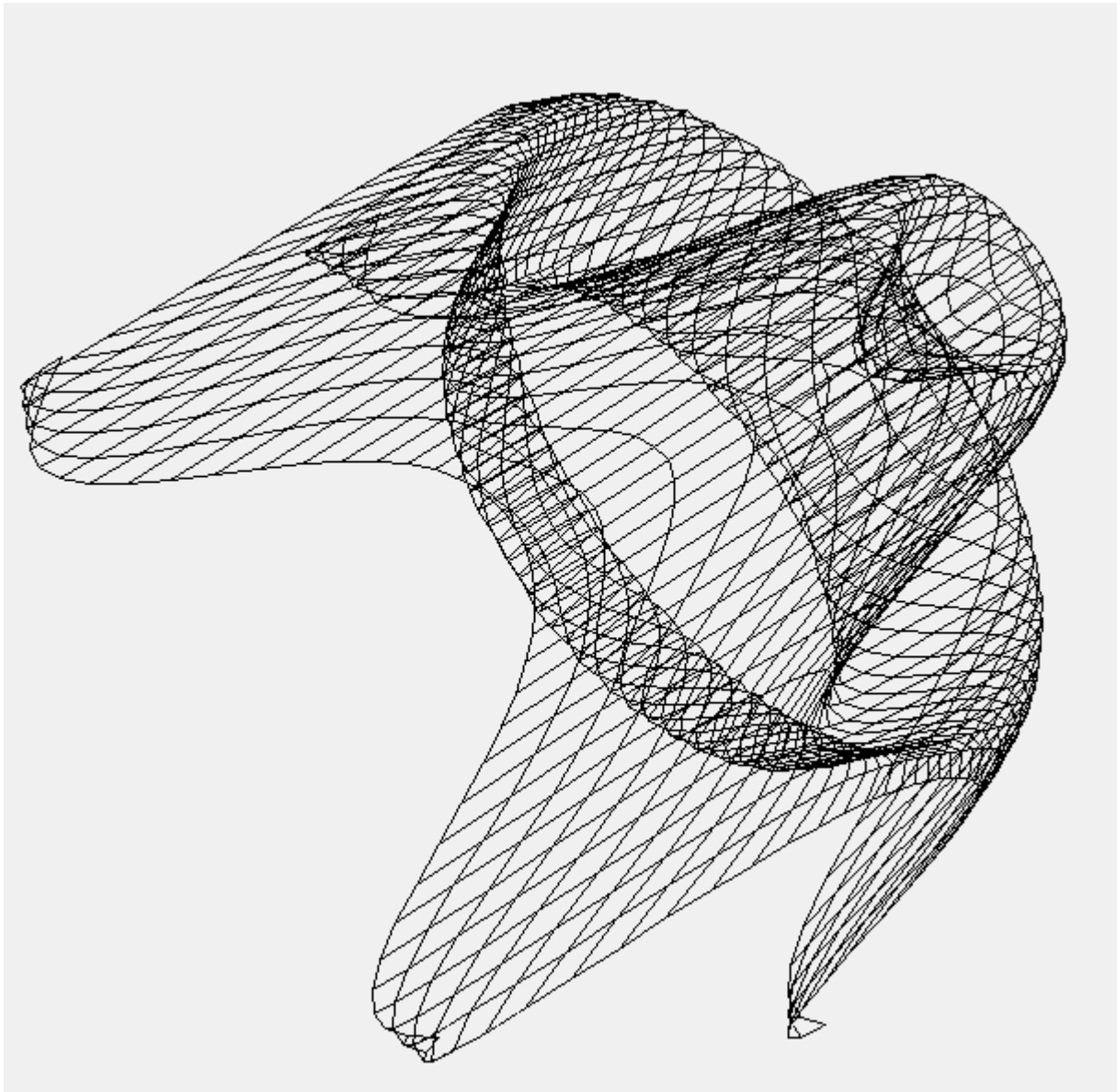


Ilustración 22: Diseño de ventana

Se utilizan los siguientes controles:

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

Ejemplo de ejecución:



*Ilustración 23: Ecuación en 3D proyectada*

Se puede cambiar la ecuación aquí:

Autor: Rafael Alberto Moreno Parra

```

namespace Grafico {
    internal class Ecuacion {
        public double Evaluar(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;
        }
    }
}

```

Este es el objeto punto que hay en cada celda del arreglo bidimensional:

```

namespace Grafico {
    internal class Punto {
        private double X, Y, Z; //Coordenadas originales
        private double Xg, Yg, Zg; //Coordenadas al girar
        private double PlanoX, PlanoY; //Proyección o sombra
        public int Xp, Yp; //En Pantalla

        public void Calcular(double X, double Y) {
            Ecuacion ecuacion = new();
            this.X = X;
            this.Y = Y;
            Z = ecuacion.Evaluar(X, Y);
            if (double.IsNaN(Z) || double.IsInfinity(Z))
                Z = 0;
        }

        //Retorna el valor de Z sin girar
        public double getZ() {
            return Z;
        }

        //Normaliza punto y luego lo ubica entre -0.5 y 0.5
        public void Normaliza(double MinX, double MinY, double MinZ,
            double MaxX, double MaxY, double MaxZ) {
            X = (X - MinX) / (MaxX - MinX) - 0.5;
            Y = (Y - MinY) / (MaxY - MinY) - 0.5;
            Z = (Z - MinZ) / (MaxZ - MinZ) - 0.5;
        }

        //Gira el punto
        public void Giro(double[,] Mt) {
            Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
            Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
        }
    }
}

```

```

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

//Convierte de 3D a 2D (segunda dimensión)
public void Proyecta(double ZPersona) {
    PlanoX = Xg * ZPersona / (ZPersona - Zg);
    PlanoY = Yg * ZPersona / (ZPersona - Zg);
}

//Convierte 2D real a 2D pantalla
public void Pantalla(int XpIni, int YpIni, int XpFin, int YpFin) {

    //Las constantes de transformación
    double conX = (XpFin - XpIni) / 1.75863087538355622;
    double conY = (YpFin - YpIni) / 1.75863087538355622;

    //Cuadra en pantalla física
    Xp = Convert.ToInt32(conX * (PlanoX + 0.87931543769177) + XpIni);
    Yp = Convert.ToInt32(conY * (PlanoY + 0.87931543769177) + YpIni);
}
}
}

```

Este es el objeto Graf3D que forma la malla:

M/008.7z/Graf3D.cs

```

namespace Grafico {
    class Graf3D {
        public Punto[][] MtPunto;

        double MinX, MinY, MaxX, MaxY, MinZ, MaxZ;
        double AnguloX, AnguloY, AnguloZ;
        double ZPersona;
        int NumLineas;
        int XpIni, YpIni, XpFin, YpFin;

        public void Inicializa() {
            //Valores
            MinX = -10;
            MinY = -10;
            MaxX = 10;
            MaxY = 10;
            NumLineas = 40;

            //Angulos de giro
            AnguloX = 45;
            AnguloY = 45;
        }
    }
}

```

```

AnguloZ = 45;

//Distancia de la persona al plano
ZPersona = 5;

//Tamaño de la pantalla
XpIni = 400;
YpIni = 0;
XpFin = 1200;
YpFin = 800;
}

public void setAnguloX(double AnguloX) {
    this.AnguloX = AnguloX;
}

public void setAnguloY(double AnguloY) {
    this.AnguloY = AnguloY;
}

public void setAnguloZ(double AnguloZ) {
    this.AnguloZ = AnguloZ;
}

public void setNumLineas(int NumLineas) {
    this.NumLineas = NumLineas;
}

private void IniciaMatrizPuntos() {
    //Arreglo de arreglos
    MtPunto = new Punto[NumLineas][];
    for (int Fila = 0; Fila < MtPunto.Length; Fila++) {
        MtPunto[Fila] = new Punto[NumLineas];
        for (int Columna = 0; Columna < MtPunto[Fila].Length; Columna++) {
            MtPunto[Fila][Columna] = new Punto();
        }
    }
}

private void CalculaXYZ() {
    //Calcula las coordenadas XYZ originales
    double IncrY = (MaxY - MinY) / NumLineas;
    double IncrX = (MaxX - MinX) / NumLineas;
    double X = MinX;
    double Y = MinY;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        X = MinX;

```

```

        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Calcular(X, Y);
            X += IncrX;
        }
        Y += IncrY;
    }
}

private void ExtremosZ() {
    MinZ = double.MaxValue;
    MaxZ = double.MinValue;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            //Los valores extremos de Z
            if (MtPunto[Fila][Columna].getZ() < MinZ)
                MinZ = MtPunto[Fila][Columna].getZ();

            if (MtPunto[Fila][Columna].getZ() > MaxZ)
                MaxZ = MtPunto[Fila][Columna].getZ();
        }
    }
}

private void NormalizaCalculos() {
    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++)
            MtPunto[Fila][Columna].Normaliza(MinX, MinY, MinZ, MaxX, MaxY,
MaxZ);
}

public void Calcula() {
    IniciaMatrizPuntos();
    CalculaXYZ();
    ExtremosZ();
    NormalizaCalculos();
    GiroProyectaCuadra();
}

public void GiroProyectaCuadra() {
    //Genera la matriz de rotación
    double CosX = Math.Cos(AnguloX * Math.PI / 180);
    double SinX = Math.Sin(AnguloX * Math.PI / 180);
    double CosY = Math.Cos(AnguloY * Math.PI / 180);
    double SinY = Math.Sin(AnguloY * Math.PI / 180);
    double CosZ = Math.Cos(AnguloZ * Math.PI / 180);
    double SinZ = Math.Sin(AnguloZ * Math.PI / 180);
}

```



```

        //Matriz de Rotación

        //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
        double[,] MtRota = 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}
        };

        for (int Fila = 0; Fila < NumLineas; Fila++)
            for (int Columna = 0; Columna < NumLineas; Columna++) {
                MtPunto[Fila][Columna].Giro(MtRota);
                MtPunto[Fila][Columna].Proyecta(ZPersona);
                MtPunto[Fila][Columna].Pantalla(XpIni, YpIni, XpFin, YpFin);
            }
        }
    }
}

```

Y el formulario:

M/008.7z/Form1.cs

```

namespace Grafico {
    public partial class Form1 : Form {
        Graf3D graf3D;

        public Form1() {
            InitializeComponent();
            graf3D = new Graf3D();
            graf3D.Inicializa();
            graf3D.Calcula();
        }

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

            //Hace líneas de un solo sentido
            for (int Fila = 0; Fila < graf3D.MtPunto.Length; Fila++)
                for (int Columna = 0; Columna < graf3D.MtPunto.Length - 1;
Columna++) {
                    int X1 = graf3D.MtPunto[Fila][Columna].Xp;
                    int Y1 = graf3D.MtPunto[Fila][Columna].Yp;

                    int X2 = graf3D.MtPunto[Fila][Columna + 1].Xp;
                    int Y2 = graf3D.MtPunto[Fila][Columna + 1].Yp;

```

```

        Lienzo.DrawLine(Lapiz, X1, Y1, X2, Y2);
    }

    //hace las líneas transversales generando el efecto de trama
    for (int Columna = 0; Columna < graf3D.MtPunto.Length; Columna++)
        for (int Fila = 0; Fila < graf3D.MtPunto.Length - 1; Fila++) {
            int X1 = graf3D.MtPunto[Fila][Columna].Xp;
            int Y1 = graf3D.MtPunto[Fila][Columna].Yp;

            int X2 = graf3D.MtPunto[Fila + 1][Columna].Xp;
            int Y2 = graf3D.MtPunto[Fila + 1][Columna].Yp;

            Lienzo.DrawLine(Lapiz, X1, Y1, X2, Y2);
        }
    }

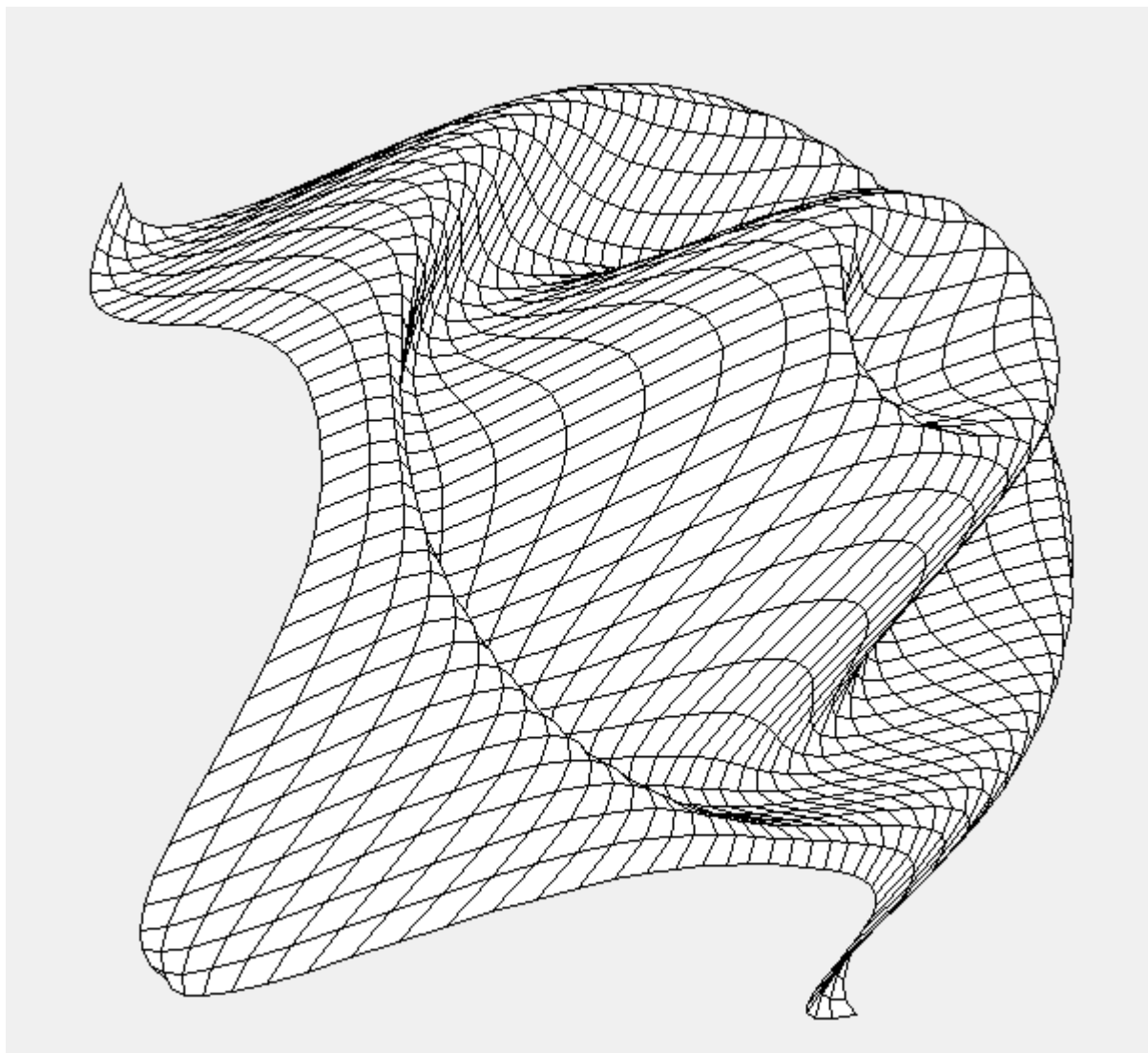
    private void numGiroX_ValueChanged(object sender, EventArgs e) {
        graf3D.setAnguloX((double)numGiroX.Value);
        graf3D.GiroProyectaCuadra();
        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        graf3D.setAnguloY((double)numGiroY.Value);
        graf3D.GiroProyectaCuadra();
        Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        graf3D.setAnguloZ((double)numGiroZ.Value);
        graf3D.GiroProyectaCuadra();
        Refresh();
    }

    private void numLineas_ValueChanged(object sender, EventArgs e) {
        graf3D.setNumLineas((int)numLineas.Value);
        graf3D.Calcula();
        graf3D.GiroProyectaCuadra();
        Refresh();
    }
}
}
}

```



*Ilustración 24: Líneas ocultas*

Para lograr el efecto de ocultar lo que está detrás y dar esa sensación de no transparencia, se hace uso del algoritmo del pintor. El gráfico matemático se elabora no por líneas sino por polígonos de 4 lados. Son los mismos pasos del gráfico anterior, excepto los últimos que serían así:

### **Paso 7**

Con la matriz de puntos ya elaborada, se crea una lista de polígonos de 4 lados, cada polígono se forma tomando un punto de la matriz y los otros tres puntos son los vecinos a ese punto. Además del polígono, se requiere el valor Z girado de esos 4 puntos y se genera un promedio.

### **Paso 8**

Se ordena la lista de polígonos de menor a mayor usando el Z promedio calculado en el paso anterior. Luego se dibuja en orden cada polígono, primero el perímetro y luego se rellena con el color de fondo (algoritmo de pintor).

Se puede cambiar la ecuación aquí:

M/009.7z/Ecuacion.cs

```
namespace Grafico {  
    internal class Ecuacion {  
        public double Evaluar(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;  
        }  
    }  
}
```

M/009.7z/Punto.cs

```
namespace Grafico {  
    internal class Punto {  
        private double X, Y, Z; //Coordenadas originales  
        private double Xg, Yg, Zg; //Coordenadas al girar  
        private double PlanoX, PlanoY; //Proyección o sombra  
        public int Xp, Yp; //En Pantalla  
  
        public void Calcular(double X, double Y) {  
            Ecuacion ecuacion = new();  
            this.X = X;  
            this.Y = Y;  
            Z = ecuacion.Evaluar(X, Y);  
            if (double.IsNaN(Z) || double.IsInfinity(Z))  
                Z = 0;  
        }  
  
        //Retorna el valor de Z sin girar  
        public double getZ() {  
            return Z;  
        }  
  
        //Retorna el valor de Z al girar  
        public double getZg() {  
            return Zg;  
        }  
    }  
}
```

```

//Normaliza punto y luego lo ubica entre -0.5 y 0.5
public void Normaliza(double MinX, double MinY, double MinZ,
    double MaxX, double MaxY, double MaxZ) {
    X = (X - MinX) / (MaxX - MinX) - 0.5;
    Y = (Y - MinY) / (MaxY - MinY) - 0.5;
    Z = (Z - MinZ) / (MaxZ - MinZ) - 0.5;
}

//Gira el punto
public void Giro(double[,] Mt) {
    Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
    Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
    Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];
}

//Convierte de 3D a 2D (segunda dimensión)
public void Proyecta(double ZPersona) {
    PlanoX = Xg * ZPersona / (ZPersona - Zg);
    PlanoY = Yg * ZPersona / (ZPersona - Zg);
}

//Convierte 2D real a 2D pantalla
public void Pantalla(int XpIni, int YpIni, int XpFin, int YpFin) {

    //Las constantes de transformación
    double conX = (XpFin - XpIni) / 1.75863087538355622;
    double conY = (YpFin - YpIni) / 1.75863087538355622;

    //Cuadra en pantalla física
    Xp = Convert.ToInt32(conX * (PlanoX + 0.87931543769177) + XpIni);
    Yp = Convert.ToInt32(conY * (PlanoY + 0.87931543769177) + YpIni);
}
}
}

```

M/009.7z/Poligono.cs

```

namespace Grafico {
    internal class Poligono : IComparable {
        //Coordenadas de dibujo del polígono
        Point Punto1, Punto2, Punto3, Punto4;

        //Profundidad media de Z al girarse las coordenadas
        double Centro;
    }
}

```

```

    public Poligono(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4,
int Y4, double Zg1, double Zg2, double Zg3, double Zg4) {
        Punto1 = new(X1, Y1);
        Punto2 = new(X2, Y2);
        Punto3 = new(X3, Y3);
        Punto4 = new(X4, Y4);
        Centro = (Zg1 + Zg2 + Zg3 + Zg4) / 4;
    }

    //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
    }

    //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[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];

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

```

M/009.7z/Graf3D.cs

```

namespace Grafico {
    class Graf3D {
        //Arreglo bidimensional que tiene los puntos
        public Punto[][] MtPunto;

        double MinX, MinY, MaxX, MaxY, MinZ, MaxZ;
        double AnguloX, AnguloY, AnguloZ;
        double ZPersona;
        int NumLineas;
        int XpIni, YpIni, XpFin, YpFin;
    }
}

```

```

//Listado de poligonos
public List<Poligono> poligonos;

public void Inicializa() {
    //Valores
    MinX = -10;
    MinY = -10;
    MaxX = 10;
    MaxY = 10;
    NumLineas = 40;

    //Angulos de giro
    AnguloX = 45;
    AnguloY = 45;
    AnguloZ = 45;

    //Distancia de la persona al plano
    ZPersona = 5;

    //Tamaño de la pantalla
    XpIni = 400;
    YpIni = 0;
    XpFin = 1200;
    YpFin = 800;

    //Inicializa polígonos
    poligonos = new List<Poligono>();
}

public void setAnguloX(double AnguloX) {
    this.AnguloX = AnguloX;
}

public void setAnguloY(double AnguloY) {
    this.AnguloY = AnguloY;
}

public void setAnguloZ(double AnguloZ) {
    this.AnguloZ = AnguloZ;
}

public void setNumLineas(int NumLineas) {
    this.NumLineas = NumLineas;
}

public void Calcula() {
    IniciaMatrizPuntos();
    CalculaXYZ();
}

```

```

ExtremosZ();
NormalizaCalculos();
GiroProyectaCuadra();
}

private void IniciaMatrizPuntos() {
    //Arreglo de arreglos
    MtPunto = new Punto[NumLineas][];
    for (int Fila = 0; Fila < MtPunto.Length; Fila++) {
        MtPunto[Fila] = new Punto[NumLineas];
        for (int Columna = 0; Columna < MtPunto[Fila].Length; Columna++) {
            MtPunto[Fila][Columna] = new Punto();
        }
    }
}

private void CalculaXYZ() {
    //Calcula las coordenadas XYZ originales
    double IncrY = (MaxY - MinY) / NumLineas;
    double IncrX = (MaxX - MinX) / NumLineas;
    double X = MinX;
    double Y = MinY;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        X = MinX;
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Calcular(X, Y);
            X += IncrX;
        }
        Y += IncrY;
    }
}

private void ExtremosZ() {
    MinZ = double.MaxValue;
    MaxZ = double.MinValue;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            //Los valores extremos de Z
            if (MtPunto[Fila][Columna].getZ() < MinZ)
                MinZ = MtPunto[Fila][Columna].getZ();

            if (MtPunto[Fila][Columna].getZ() > MaxZ)
                MaxZ = MtPunto[Fila][Columna].getZ();
        }
    }
}

```



```

private void NormalizaCalculos() {
    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++)
            MtPunto[Fila][Columna].Normaliza(MinX, MinY, MinZ, MaxX, MaxY,
MaxZ);
}

public void GiroProyectaCuadra() {
    //Genera la matriz de rotación
    double CosX = Math.Cos(AnguloX * Math.PI / 180);
    double SinX = Math.Sin(AnguloX * Math.PI / 180);
    double CosY = Math.Cos(AnguloY * Math.PI / 180);
    double SinY = Math.Sin(AnguloY * Math.PI / 180);
    double CosZ = Math.Cos(AnguloZ * Math.PI / 180);
    double SinZ = Math.Sin(AnguloZ * Math.PI / 180);

    //Matriz de Rotación

    //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
    double[,] MtRota = 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}
    };

    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Giro(MtRota);
            MtPunto[Fila][Columna].Proyecta(ZPersona);
            MtPunto[Fila][Columna].Pantalla(XpIni, YpIni, XpFin, YpFin);
        }

    //Genera los poligonos
    GeneraPoligonos();

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

private void GeneraPoligonos() {
    poligonos.Clear();
    for (int Fila = 0; Fila < NumLineas - 1; Fila++)
        for (int Columna = 0; Columna < NumLineas - 1; Columna++) {
            int X1 = MtPunto[Fila][Columna].Xp;
            int Y1 = MtPunto[Fila][Columna].Yp;

```

```

        double Zg1 = MtPunto[Fila][Columna].getZg();

        int X2 = MtPunto[Fila][Columna + 1].Xp;
        int Y2 = MtPunto[Fila][Columna + 1].Yp;
        double Zg2 = MtPunto[Fila][Columna + 1].getZg();

        int X3 = MtPunto[Fila + 1][Columna + 1].Xp;
        int Y3 = MtPunto[Fila + 1][Columna + 1].Yp;
        double Zg3 = MtPunto[Fila + 1][Columna + 1].getZg();

        int X4 = MtPunto[Fila + 1][Columna].Xp;
        int Y4 = MtPunto[Fila + 1][Columna].Yp;
        double Zg4 = MtPunto[Fila + 1][Columna].getZg();

        poligonos.Add(new Poligono(X1, Y1, X2, Y2, X3, Y3, X4, Y4, Zg1,
Zg2, Zg3, Zg4));
    }
}
}
}

```

M/009.7z/Form1.cs

```

namespace Grafico {
    public partial class Form1 : Form {
        Graf3D graf3D;

        public Form1() {
            InitializeComponent();
            graf3D = new Graf3D();
            graf3D.Inicializa();
            graf3D.Calcula();
        }

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

            for (int Cont = 0; Cont < graf3D.poligonos.Count; Cont++)
                graf3D.poligonos[Cont].Dibuja(Lienzo, Lapis, Relleno);
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloX((double)numGiroX.Value);
            graf3D.GiroProyectaCuadra();
        }
    }
}

```

```

        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        graf3D.setAnguloY((double)numGiroY.Value);
        graf3D.GiroProyectaCuadra();
        Refresh();
    }

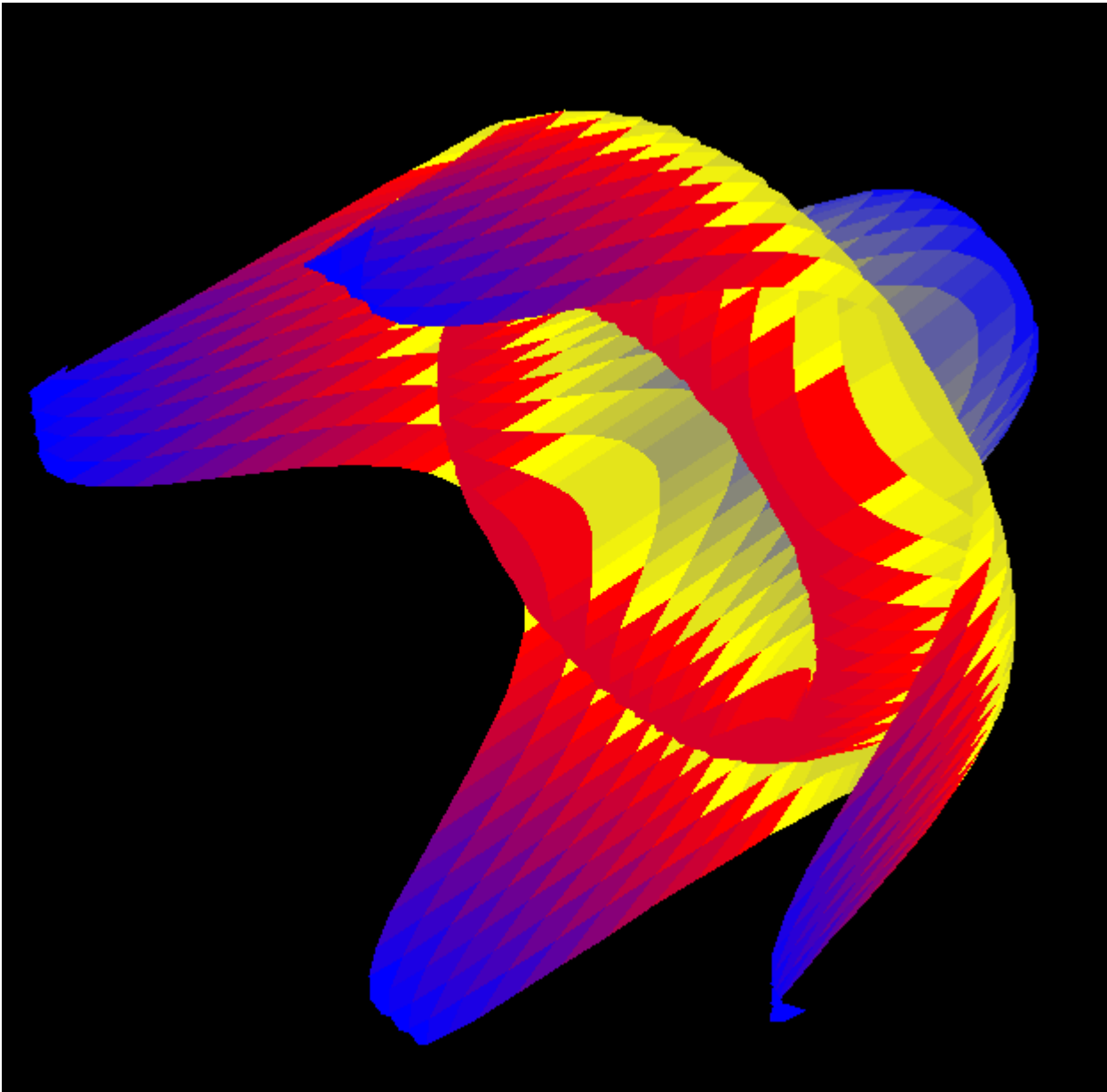
    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        graf3D.setAnguloZ((double)numGiroZ.Value);
        graf3D.GiroProyectaCuadra();
        Refresh();
    }

    private void numLineas_ValueChanged(object sender, EventArgs e) {
        graf3D.setNumLineas((int)numLineas.Value);
        graf3D.Calcula();
        graf3D.GiroProyectaCuadra();
        Refresh();
    }
}
}
}

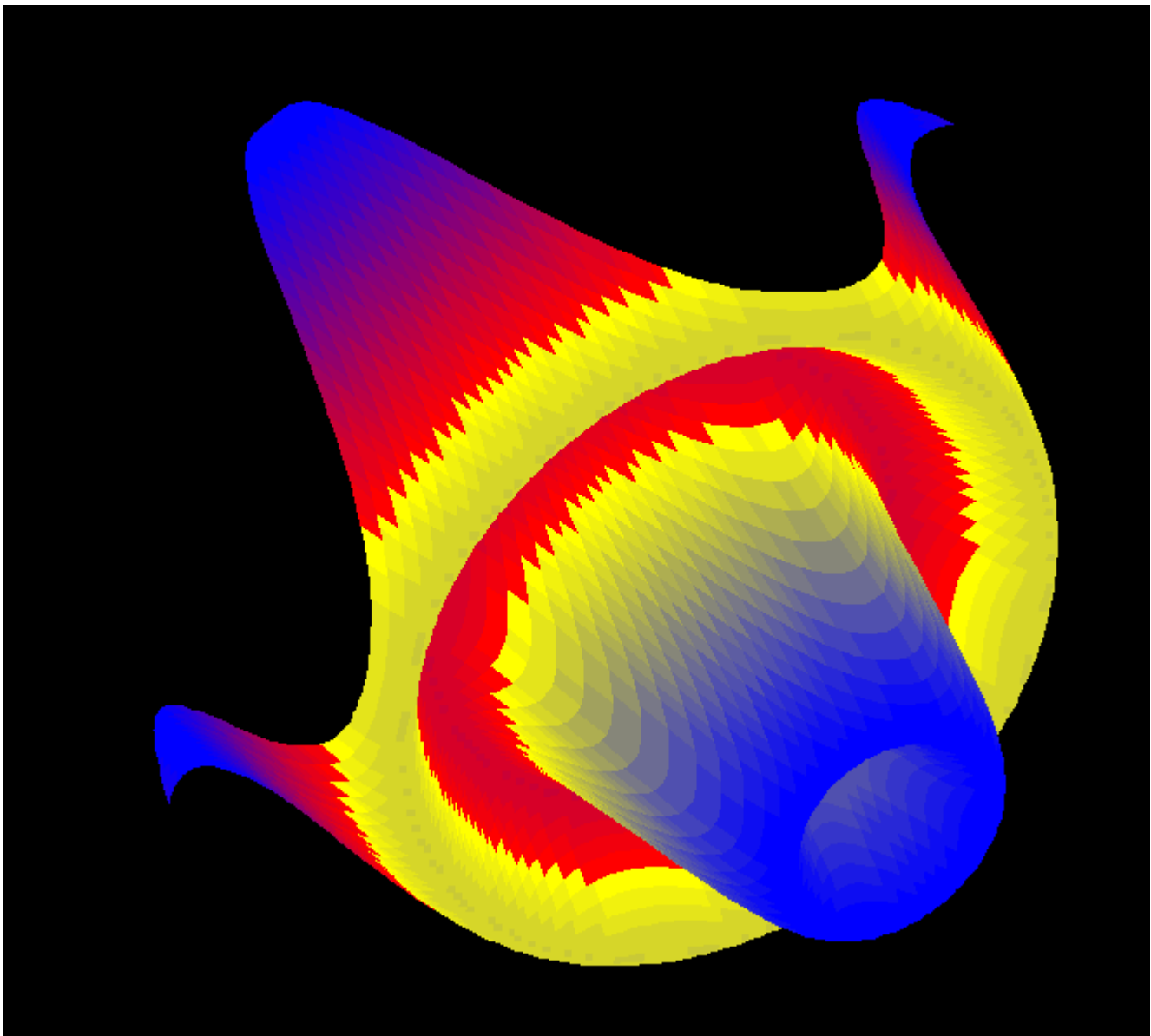
```

## Gráfico Matemático en 3D con colores según altura

En cada polígono a dibujar se tiene en cuenta el valor de  $Z$  original sin girar (promedio de los 4 puntos que conforman al polígono) de la ecuación. Luego se hace un gradiente de color entre los dos extremos valores de  $Z$  y así se dibuja por colores por niveles.



*Ilustración 25: El valor de  $Z$  determina el color*



*Ilustración 26: Inclusive si gira, el valor de Z determina el color*

M/010.7z/Ecuacion.cs

```
namespace Grafico {  
    internal class Ecuacion {  
        public double Evaluar(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;  
        }  
    }  
}
```

```

namespace Grafico {
    internal class Punto {
        private double X, Y, Z; //Coordenadas originales
        private double Xg, Yg, Zg; //Coordenadas al girar
        private double PlanoX, PlanoY; //Proyección o sombra
        public int Xp, Yp; //En Pantalla

        public void Calcular(double X, double Y) {
            Ecuacion ecuacion = new();
            this.X = X;
            this.Y = Y;
            Z = ecuacion.Evaluar(X, Y);
            if (double.IsNaN(Z) || double.IsInfinity(Z))
                Z = 0;
        }

        //Retorna el valor de Z sin girar
        public double getZ() {
            return Z;
        }

        //Retorna el valor de Z al girar
        public double getZg() {
            return Zg;
        }

        //Normaliza punto y luego lo ubica entre -0.5 y 0.5
        public void Normaliza(double MinX, double MinY, double MinZ,
                               double MaxX, double MaxY, double MaxZ) {
            X = (X - MinX) / (MaxX - MinX) - 0.5;
            Y = (Y - MinY) / (MaxY - MinY) - 0.5;
            Z = (Z - MinZ) / (MaxZ - MinZ) - 0.5;
        }

        //Gira el punto
        public void Giro(double[,] Mt) {
            Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
            Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
            Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];
        }

        //Convierte de 3D a 2D (segunda dimensión)
        public void Proyecta(double ZPersona) {
            PlanoX = Xg * ZPersona / (ZPersona - Zg);
            PlanoY = Yg * ZPersona / (ZPersona - Zg);
        }
    }
}

```

```
//Convierte 2D real a 2D pantalla
public void Pantalla(int XpIni, int YpIni, int XpFin, int YpFin) {

    //Las constantes de transformación
    double conX = (XpFin - XpIni) / 1.75863087538355622;
    double conY = (YpFin - YpIni) / 1.75863087538355622;

    //Cuadra en pantalla física
    Xp = Convert.ToInt32(conX * (PlanoX + 0.87931543769177) + XpIni);
    Yp = Convert.ToInt32(conY * (PlanoY + 0.87931543769177) + YpIni);
}
}
```

```

namespace Grafico {
    internal class Poligono : IComparable {
        //Coordenadas de dibujo del polígono
        Point Punto1, Punto2, Punto3, Punto4;

        //Profundidad media de Z al girarse las coordenadas
        double Centro;

        //Color de relleno del polígono
        Color ColorZ;

        public Poligono(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4,
            int Y4, double Zg1, double Zg2, double Zg3, double Zg4, double Z1, double
            Z2, double Z3, double Z4, List<Color> colorList) {
            Punto1 = new(X1, Y1);
            Punto2 = new(X2, Y2);
            Punto3 = new(X3, Y3);
            Punto4 = new(X4, Y4);
            Centro = (Zg1 + Zg2 + Zg3 + Zg4) / 4;

            int TotalColores = colorList.Count;
            double PromedioZ = (Z1 + Z2 + Z3 + Z4 + 2) / 4;
            int ColorEscoge = (int) Math.Floor(TotalColores * PromedioZ);
            ColorZ = colorList[ColorEscoge];
        }

        //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
        }

        //Hace el gráfico del polígono
        public void Dibuja(Graphics Lienzo) {
            //Pone un color de fondo al polígono
            //para borrar lo que hay detrás
            Point[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];

            //Dibuja el polígono relleno y su perímetro
            Brush Relleno = new SolidBrush(ColorZ);

```



```
        Lienzo.FillPolygon(Relleno, ListaPuntos);  
    }  
}  
}
```

```

namespace Grafico {
    class Graf3D {
        //Arreglo bidimensional que tiene los puntos
        public Punto[][] MtPunto;

        double MinX, MinY, MaxX, MaxY, MinZ, MaxZ;
        double AnguloX, AnguloY, AnguloZ;
        double ZPersona;
        int NumLineas;
        int XpIni, YpIni, XpFin, YpFin;

        //Listado de poligonos
        public List<Poligono> Poligonos;

        //Colores para pintar la malla
        List<Color> ListaColores;

        public void Inicializa() {

            //Distancia de la persona al plano
            ZPersona = 5;

            //Tamaño de la pantalla
            XpIni = 400;
            YpIni = 0;
            XpFin = 1200;
            YpFin = 800;

            //Inicializa polígonos
            Poligonos = [];

            int NumColores = 40; // Número de colores a generar
            ListaColores = GeneraGradienteColor(NumColores);
        }

        public void setAnguloX(double AnguloX) {
            this.AnguloX = AnguloX;
        }

        public void setAnguloY(double AnguloY) {
            this.AnguloY = AnguloY;
        }

        public void setAnguloZ(double AnguloZ) {
            this.AnguloZ = AnguloZ;
        }
    }
}

```

```

    }

    public void Calcula(double MinX, double MaxX, double MinY, double MaxY,
double AnguloX, double AnguloY, double AnguloZ, int NumLineas) {
    //Valores del formulario
    this.MinX = MinX;
    this.MaxX = MaxX;
    this.MinY = MinY;
    this.MaxY = MaxY;

    //Angulos de giro
    this.AnguloX = AnguloX;
    this.AnguloY = AnguloY;
    this.AnguloZ = AnguloZ;

    //Número de líneas que forma la malla
    this.NumLineas = NumLineas;

    IniciaMatrizPuntos();
    CalculaXYZ();
    ExtremosZ();
    NormalizaCalculos();
    GiroProyectaCuadra();
}

private void IniciaMatrizPuntos() {
    //Arreglo de arreglos
    MtPunto = new Punto[NumLineas][];
    for (int Fila = 0; Fila < MtPunto.Length; Fila++) {
        MtPunto[Fila] = new Punto[NumLineas];
        for (int Columna = 0; Columna < MtPunto[Fila].Length; Columna++) {
            MtPunto[Fila][Columna] = new Punto();
        }
    }
}

private void CalculaXYZ() {
    //Calcula las coordenadas XYZ originales
    double IncrY = (MaxY - MinY) / NumLineas;
    double IncrX = (MaxX - MinX) / NumLineas;
    double X = MinX;
    double Y = MinY;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        X = MinX;
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Calcular(X, Y);
            X += IncrX;
        }
    }
}

```

```

    }
    Y += IncrY;
}

private void ExtremosZ() {
    MinZ = double.MaxValue;
    MaxZ = double.MinValue;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            //Los valores extremos de Z
            if (MtPunto[Fila][Columna].getZ() < MinZ)
                MinZ = MtPunto[Fila][Columna].getZ();

            if (MtPunto[Fila][Columna].getZ() > MaxZ)
                MaxZ = MtPunto[Fila][Columna].getZ();
        }
    }
}

private void NormalizaCalculos() {
    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++)
            MtPunto[Fila][Columna].Normaliza(MinX, MinY, MinZ, MaxX, MaxY,
MaxZ);
}

public void GiroProyectaCuadra() {
    //Genera la matriz de rotación
    double CosX = Math.Cos(AnguloX * Math.PI / 180);
    double SinX = Math.Sin(AnguloX * Math.PI / 180);
    double CosY = Math.Cos(AnguloY * Math.PI / 180);
    double SinY = Math.Sin(AnguloY * Math.PI / 180);
    double CosZ = Math.Cos(AnguloZ * Math.PI / 180);
    double SinZ = Math.Sin(AnguloZ * Math.PI / 180);

    //Matriz de Rotación

    //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
    double[,] MtRota = 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}
    };

    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++) {

```

```

        MtPunto[Fila][Columna].Giro(MtRota);
        MtPunto[Fila][Columna].Proyecta(ZPersona);
        MtPunto[Fila][Columna].Pantalla(XpIni, YpIni, XpFin, YpFin);
    }

    //Genera los poligonos
    GeneraPoligonos();

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

private void GeneraPoligonos() {
    Poligonos.Clear();
    for (int Fila = 0; Fila < NumLineas - 1; Fila++)
        for (int Columna = 0; Columna < NumLineas - 1; Columna++) {
            int X1 = MtPunto[Fila][Columna].Xp;
            int Y1 = MtPunto[Fila][Columna].Yp;
            double Zg1 = MtPunto[Fila][Columna].getZg();
            double Z1 = MtPunto[Fila][Columna].getZ();

            int X2 = MtPunto[Fila][Columna + 1].Xp;
            int Y2 = MtPunto[Fila][Columna + 1].Yp;
            double Zg2 = MtPunto[Fila][Columna + 1].getZg();
            double Z2 = MtPunto[Fila][Columna + 1].getZ();

            int X3 = MtPunto[Fila + 1][Columna + 1].Xp;
            int Y3 = MtPunto[Fila + 1][Columna + 1].Yp;
            double Zg3 = MtPunto[Fila + 1][Columna + 1].getZg();
            double Z3 = MtPunto[Fila + 1][Columna + 1].getZ();

            int X4 = MtPunto[Fila + 1][Columna].Xp;
            int Y4 = MtPunto[Fila + 1][Columna].Yp;
            double Zg4 = MtPunto[Fila + 1][Columna].getZg();
            double Z4 = MtPunto[Fila + 1][Columna].getZ();

            Poligonos.Add(new Poligono(X1, Y1, X2, Y2, X3, Y3, X4, Y4, Zg1,
Zg2, Zg3, Zg4, Z1, Z2, Z3, Z4, ListaColores));
        }
}

List<Color> GeneraGradienteColor(int NumColores) {
    List<Color> Colores = [];
    int Mitad = NumColores / 2;

    // Gradiente de azul a amarillo

```

```

    for (int Cont = 0; Cont < Mitad; Cont++) {
        int Rojo = (int)(255 * (Cont / (float)(NumColores - Mitad - 1)));
        int Verde = (int)(255 * (Cont / (float)(NumColores - Mitad - 1)));
        int Azul = 255 - (int)(255 * (Cont / (float)(NumColores - Mitad -
1)));
        Colores.Add(Color.FromArgb(Rojo, Verde, Azul));
    }

    // Gradiente de rojo a azul
    for (int Cont = 0; Cont < NumColores - Mitad; Cont++) {
        int Rojo = 255 - (int)(255 * (Cont / (float)(Mitad - 1)));
        int Verde = 0;
        int Azul = (int)(255 * (Cont / (float)(Mitad - 1)));
        Colores.Add(Color.FromArgb(Rojo, Verde, Azul));
    }

    return Colores;
}
}
}

```

```

namespace Grafico {
    public partial class Form1 : Form {
        Graf3D graf3D;

        public Form1() {
            InitializeComponent();
            graf3D = new Graf3D();
            graf3D.Inicializa();
            Recalcula();
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloX((double)numGiroX.Value);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloY((double)numGiroY.Value);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloZ((double)numGiroZ.Value);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void numLineas_ValueChanged(object sender, EventArgs e) {
            Recalcula();
        }

        private void numXminimo_ValueChanged(object sender, EventArgs e) {
            if (numXminimo.Value >= numXmaximo.Value)
                numXminimo.Value = numXmaximo.Value - 1;
            Recalcula();
        }

        private void numXmaximo_ValueChanged(object sender, EventArgs e) {
            if (numXmaximo.Value <= numXminimo.Value)
                numXmaximo.Value = numXminimo.Value + 1;
            Recalcula();
        }
    }
}

```

```

private void numYminimo_ValueChanged(object sender, EventArgs e) {
    if (numYminimo.Value >= numYmaximo.Value)
        numYminimo.Value = numYmaximo.Value - 1;
    Recalcula();
}

private void numYmaximo_ValueChanged(object sender, EventArgs e) {
    if (numYmaximo.Value <= numYminimo.Value)
        numYmaximo.Value = numYminimo.Value + 1;
    Recalcula();
}


private void Recalcula() {
    graf3D.Calcula((double)numXminimo.Value,
        (double)numXmaximo.Value,
        (double)numYminimo.Value,
        (double)numYmaximo.Value,
        (double)numGiroX.Value,
        (double)numGiroY.Value,
        (double)numGiroZ.Value,
        (int)numLineas.Value);
    Refresh();
}

private void Form1_Paint(object sender, PaintEventArgs e) {
    Graphics Lienzo = e.Graphics;

    for (int Cont = 0; Cont < graf3D.Poligonos.Count; Cont++)
        graf3D.Poligonos[Cont].Dibuja(Lienzo);
}
}

```



 Gráfico ecuación 3D

Giro en X

56

Giro en Y

25

Giro en Z

224

Número de líneas

50

Valores X y Y en la ecuación

Valor X mínimo

-6

Valor X máximo

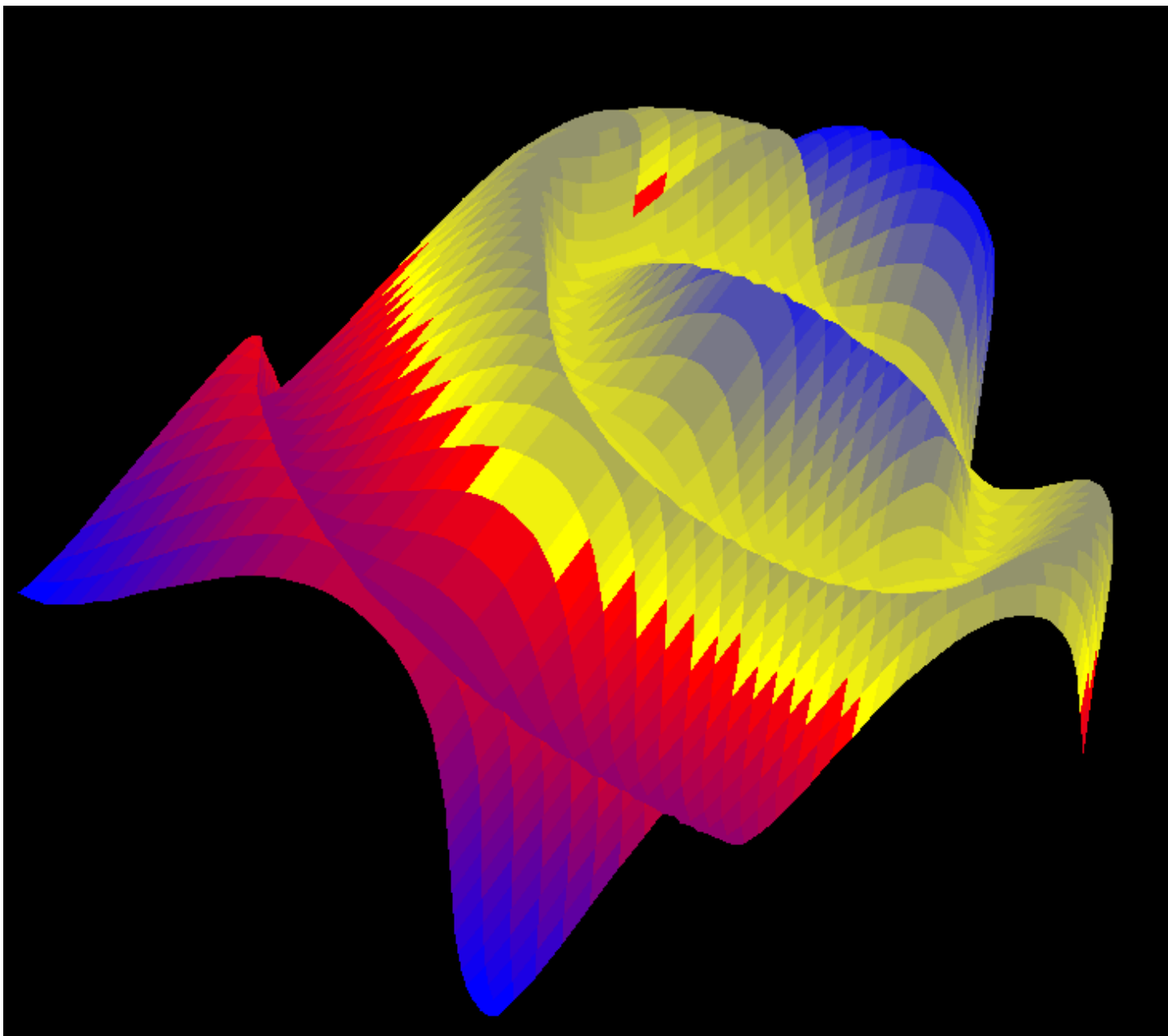
17

Valor Y mínimo

-10

Valor Y máximo

10



*Ilustración 27: Gráfico matemático 3D con colores*

## Gráfico matemático en 3D ingresando la ecuación

El siguiente paso es que el usuario final ingrese la ecuación por el formulario. Para eso, se hace uso del evaluador de expresiones 4.0.

El proyecto completo está en 011.7z

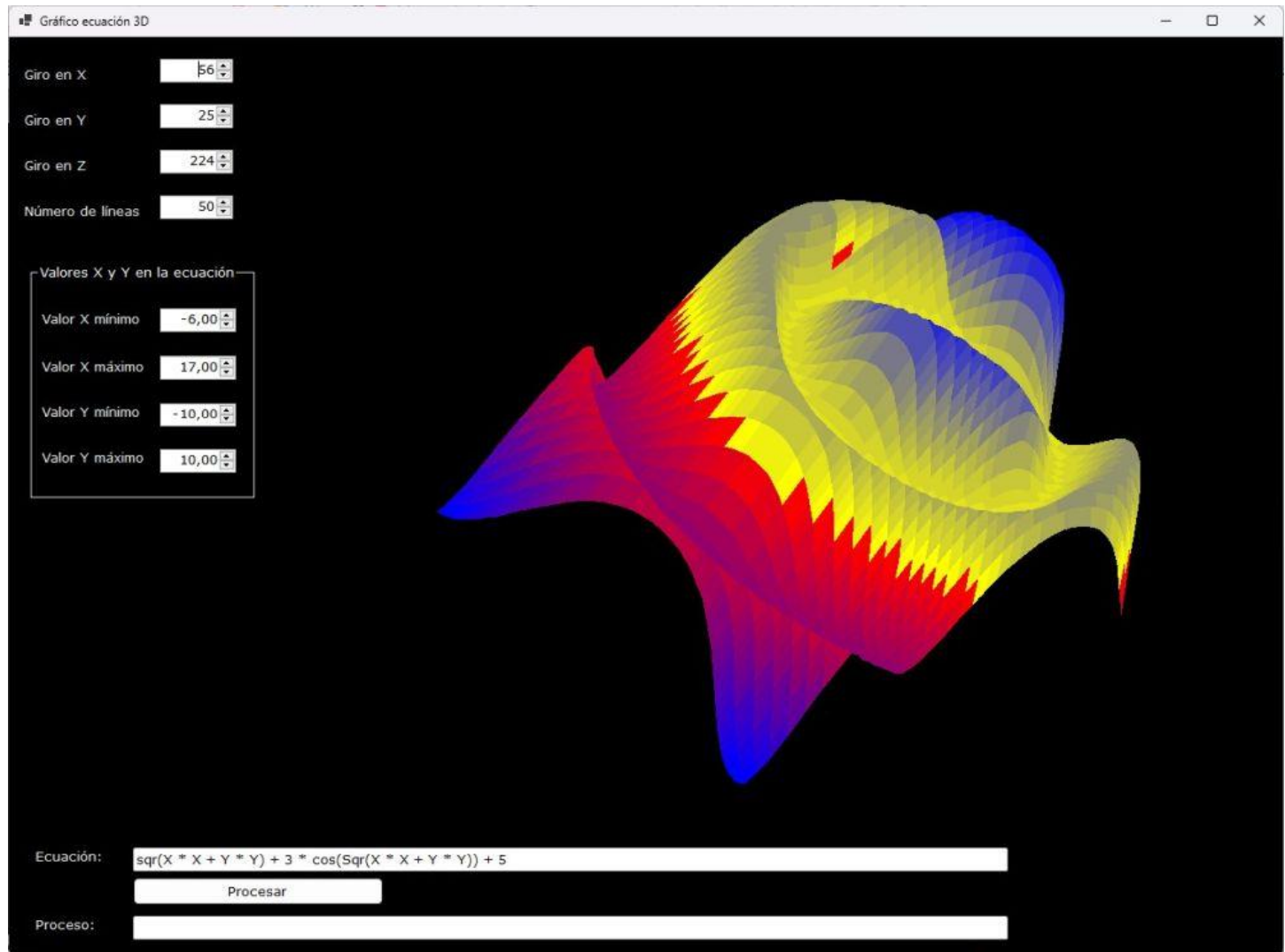


Ilustración 28: Ingresa la ecuación

```

namespace Grafico {
    internal class Punto {
        private double X, Y, Z; //Coordenadas originales
        private double Xg, Yg, Zg; //Coordenadas al girar
        private double PlanoX, PlanoY; //Proyección o sombra
        public int Xp, Yp; //En Pantalla

        public void Calcular(double X, double Y, Evaluador4 Evalua) {
            this.X = X;
            this.Y = Y;
            Evalua.DarValorVariable('x', X);
            Evalua.DarValorVariable('y', Y);
            Z = Evalua.Evaluar();
            if (double.IsNaN(Z) || double.IsInfinity(Z))
                Z = 0;
        }

        //Retorna el valor de Z sin girar
        public double getZ() {
            return Z;
        }

        //Retorna el valor de Z al girar
        public double getZg() {
            return Zg;
        }

        //Normaliza punto y luego lo ubica entre -0.5 y 0.5
        public void Normaliza(double MinX, double MinY, double MinZ,
                               double MaxX, double MaxY, double MaxZ) {
            X = (X - MinX) / (MaxX - MinX) - 0.5;
            Y = (Y - MinY) / (MaxY - MinY) - 0.5;
            Z = (Z - MinZ) / (MaxZ - MinZ) - 0.5;
        }

        //Gira el punto
        public void Giro(double[,] Mt) {
            Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
            Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
            Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];
        }

        //Convierte de 3D a 2D (segunda dimensión)
        public void Proyecta(double ZPersona) {
            PlanoX = Xg * ZPersona / (ZPersona - Zg);
            PlanoY = Yg * ZPersona / (ZPersona - Zg);
        }
    }
}

```

```

//Convierte 2D real a 2D pantalla
public void Pantalla(int XpIni, int YpIni, int XpFin, int YpFin) {

    //Las constantes de transformación
    double conX = (XpFin - XpIni) / 1.75863087538355622;
    double conY = (YpFin - YpIni) / 1.75863087538355622;

    //Cuadra en pantalla física
    Xp = Convert.ToInt32(conX * (PlanoX + 0.87931543769177) + XpIni);
    Yp = Convert.ToInt32(conY * (PlanoY + 0.87931543769177) + YpIni);
}
}
}

```

M/011.7z/Poligono.cs

```

namespace Grafico {
    internal class Poligono : IComparable {
        //Coordenadas de dibujo del polígono
        Point Punto1, Punto2, Punto3, Punto4;

        //Profundidad media de Z al girarse las coordenadas
        double Centro;

        //Color de relleno del polígono
        Color ColorZ;

        public Poligono(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4,
            int Y4, double Zg1, double Zg2, double Zg3, double Zg4, double Z1, double
            Z2, double Z3, double Z4, List<Color> colorList) {
            Punto1 = new(X1, Y1);
            Punto2 = new(X2, Y2);
            Punto3 = new(X3, Y3);
            Punto4 = new(X4, Y4);
            Centro = (Zg1 + Zg2 + Zg3 + Zg4) / 4;

            int TotalColores = colorList.Count;
            double PromedioZ = (Z1 + Z2 + Z3 + Z4 + 2) / 4;
            int ColorEscape = (int) Math.Floor(TotalColores * PromedioZ);
            ColorZ = colorList[ColorEscape];
        }

        //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
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo) {
        //Pone un color de fondo al polígono
        //para borrar lo que hay detrás
        Point[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];

        //Dibuja el polígono relleno y su perímetro
        Brush Relleno = new SolidBrush(ColorZ);
        Lienzo.FillPolygon(Relleno, ListaPuntos);
    }
}
}

```

M/011.7z/Graf3D.cs

```

namespace Grafico {
    class Graf3D {
        //Arreglo bidimensional que tiene los puntos
        public Punto[][] MtPunto;

        double MinX, MinY, MaxX, MaxY, MinZ, MaxZ;
        double AnguloX, AnguloY, AnguloZ;
        double ZPersona;
        int NumLineas;
        int XpIni, YpIni, XpFin, YpFin;

        //Listado de poligonos
        public List<Poligono> Poligonos;

        //Colores para pintar la malla
        List<Color> ListaColores;

        //La ecuación
        Evaluador4 Evalua;

        public void Inicializa() {

            //Distancia de la persona al plano
            ZPersona = 5;

            //Tamaño de la pantalla
            XpIni = 400;

```

```

YpIni = 0;
XpFin = 1200;
YpFin = 800;

//Inicializa polígonos
Poligonos = [];

int NumColores = 40; // Número de colores a generar
ListaColores = GeneraGradienteColor(NumColores);
}

public void setAnguloX(double AnguloX) {
    this.AnguloX = AnguloX;
}

public void setAnguloY(double AnguloY) {
    this.AnguloY = AnguloY;
}

public void setAnguloZ(double AnguloZ) {
    this.AnguloZ = AnguloZ;
}

public void Calcula(double MinX, double MaxX, double MinY, double MaxY,
double AnguloX, double AnguloY, double AnguloZ, int NumLineas, Evaluador4
Evalua) {
    //Valores del formulario
    this.MinX = MinX;
    this.MaxX = MaxX;
    this.MinY = MinY;
    this.MaxY = MaxY;

    //Angulos de giro
    this.AnguloX = AnguloX;
    this.AnguloY = AnguloY;
    this.AnguloZ = AnguloZ;

    //Número de líneas que forma la malla
    this.NumLineas = NumLineas;

    //La ecuación
    this.Evalua = Evalua;

    IniciaMatrizPuntos();
    CalculaXYZ();
    ExtremosZ();
    NormalizaCalculos();
    GiroProyectaCuadra();
}

```

```

}

private void IniciaMatrizPuntos() {
    //Arreglo de arreglos
    MtPunto = new Punto[NumLineas][];
    for (int Fila = 0; Fila < MtPunto.Length; Fila++) {
        MtPunto[Fila] = new Punto[NumLineas];
        for (int Columna = 0; Columna < MtPunto[Fila].Length; Columna++) {
            MtPunto[Fila][Columna] = new Punto();
        }
    }
}

private void CalculaXYZ() {
    //Calcula las coordenadas XYZ originales
    double IncrY = (MaxY - MinY) / NumLineas;
    double IncrX = (MaxX - MinX) / NumLineas;
    double X = MinX;
    double Y = MinY;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        X = MinX;
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Calcular(X, Y, Evalua);
            X += IncrX;
        }
        Y += IncrY;
    }
}

private void ExtremosZ() {
    MinZ = double.MaxValue;
    MaxZ = double.MinValue;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            //Los valores extremos de Z
            if (MtPunto[Fila][Columna].getZ() < MinZ)
                MinZ = MtPunto[Fila][Columna].getZ();

            if (MtPunto[Fila][Columna].getZ() > MaxZ)
                MaxZ = MtPunto[Fila][Columna].getZ();
        }
    }
}

private void NormalizaCalculos() {
    for (int Fila = 0; Fila < NumLineas; Fila++)

```



```

        for (int Columna = 0; Columna < NumLineas; Columna++)
            MtPunto[Fila][Columna].Normaliza(MinX, MinY, MinZ, MaxX, MaxY,
MaxZ);
    }

    public void GiroProyectaCuadra() {
        //Genera la matriz de rotación
        double CosX = Math.Cos(AnguloX * Math.PI / 180);
        double SinX = Math.Sin(AnguloX * Math.PI / 180);
        double CosY = Math.Cos(AnguloY * Math.PI / 180);
        double SinY = Math.Sin(AnguloY * Math.PI / 180);
        double CosZ = Math.Cos(AnguloZ * Math.PI / 180);
        double SinZ = Math.Sin(AnguloZ * Math.PI / 180);

        //Matriz de Rotación

        //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
        double[,] MtRota = 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}
        };

        for (int Fila = 0; Fila < NumLineas; Fila++)
            for (int Columna = 0; Columna < NumLineas; Columna++) {
                MtPunto[Fila][Columna].Giro(MtRota);
                MtPunto[Fila][Columna].Proyecta(ZPersona);
                MtPunto[Fila][Columna].Pantalla(XpIni, YpIni, XpFin, YpFin);
            }

        //Genera los poligonos
        GeneraPoligonos();

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

    private void GeneraPoligonos() {
        Poligonos.Clear();
        for (int Fila = 0; Fila < NumLineas - 1; Fila++)
            for (int Columna = 0; Columna < NumLineas - 1; Columna++) {
                int X1 = MtPunto[Fila][Columna].Xp;
                int Y1 = MtPunto[Fila][Columna].Yp;
                double Zg1 = MtPunto[Fila][Columna].getZg();
                double Z1 = MtPunto[Fila][Columna].getZ();
            }
    }

```

```

        int X2 = MtPunto[Fila][Columna + 1].Xp;
        int Y2 = MtPunto[Fila][Columna + 1].Yp;
        double Zg2 = MtPunto[Fila][Columna + 1].getZg();
        double Z2 = MtPunto[Fila][Columna + 1].getZ();

        int X3 = MtPunto[Fila + 1][Columna + 1].Xp;
        int Y3 = MtPunto[Fila + 1][Columna + 1].Yp;
        double Zg3 = MtPunto[Fila + 1][Columna + 1].getZg();
        double Z3 = MtPunto[Fila + 1][Columna + 1].getZ();

        int X4 = MtPunto[Fila + 1][Columna].Xp;
        int Y4 = MtPunto[Fila + 1][Columna].Yp;
        double Zg4 = MtPunto[Fila + 1][Columna].getZg();
        double Z4 = MtPunto[Fila + 1][Columna].getZ();

        Poligonos.Add(new Poligono(X1, Y1, X2, Y2, X3, Y3, X4, Y4, Zg1,
        Zg2, Zg3, Zg4, Z1, Z2, Z3, Z4, ListaColores));
    }
}

List<Color> GeneraGradienteColor(int NumColores) {
    List<Color> Colores = [];
    int Mitad = NumColores / 2;

    // Gradiente de azul a amarillo
    for (int Cont = 0; Cont < Mitad; Cont++) {
        int Rojo = (int)(255 * (Cont / (float)(NumColores - Mitad - 1)));
        int Verde = (int)(255 * (Cont / (float)(NumColores - Mitad - 1)));
        int Azul = 255 - (int)(255 * (Cont / (float)(NumColores - Mitad -
1)));
        Colores.Add(Color.FromArgb(Rojo, Verde, Azul));
    }

    // Gradiente de rojo a azul
    for (int Cont = 0; Cont < NumColores - Mitad; Cont++) {
        int Rojo = 255 - (int)(255 * (Cont / (float)(Mitad - 1)));
        int Verde = 0;
        int Azul = (int)(255 * (Cont / (float)(Mitad - 1)));
        Colores.Add(Color.FromArgb(Rojo, Verde, Azul));
    }

    return Colores;
}
}
}

```

```

namespace Grafico {
    public partial class Form1 : Form {
        Graf3D graf3D;
        Evaluador4 evaluador;

        public Form1() {
            InitializeComponent();
            evaluador = new Evaluador4();
            evaluador.Analizar(txtEcuacion.Text);
            graf3D = new Graf3D();
            graf3D.Inicializa();
            Recalcula();
        }

        private void numGiroX_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloX((double)numGiroX.Value);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloY((double)numGiroY.Value);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            graf3D.setAnguloZ((double)numGiroZ.Value);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void numLineas_ValueChanged(object sender, EventArgs e) {
            Recalcula();
        }

        private void numXminimo_ValueChanged(object sender, EventArgs e) {
            if (numXminimo.Value >= numXmaximo.Value)
                numXminimo.Value = numXmaximo.Value - 1;
            Recalcula();
        }

        private void numXmaximo_ValueChanged(object sender, EventArgs e) {
            if (numXmaximo.Value <= numXminimo.Value)
                numXmaximo.Value = numXminimo.Value + 1;
        }
    }
}

```

```

    Recalcula();
}

private void numYminimo_ValueChanged(object sender, EventArgs e) {
    if (numYminimo.Value >= numYmaximo.Value)
        numYminimo.Value = numYmaximo.Value - 1;
    Recalcula();
}

private void numYmaximo_ValueChanged(object sender, EventArgs e) {
    if (numYmaximo.Value <= numYminimo.Value)
        numYmaximo.Value = numYminimo.Value + 1;
    Recalcula();
}

private void Recalcula() {
    graf3D.Calcula((double)numXminimo.Value,
        (double)numXmaximo.Value,
        (double)numYminimo.Value,
        (double)numYmaximo.Value,
        (double)numGiroX.Value,
        (double)numGiroY.Value,
        (double)numGiroZ.Value,
        (int)numLineas.Value, evaluador);
    Refresh();
}

private void Form1_Paint(object sender, PaintEventArgs e) {
    Graphics Lienzo = e.Graphics;

    for (int Cont = 0; Cont < graf3D.Poligonos.Count; Cont++)
        graf3D.Poligonos[Cont].Dibuja(Lienzo);
}

private void btnProcesar_Click(object sender, EventArgs e) {
    string Ecuacion = txtEcuacion.Text;
    int Sintaxis = evaluador.Analizar(Ecuacion);

    if (Sintaxis > 0) {
        txtProceso.Text = evaluador.MensajeError(Sintaxis);
    }
    else {
        txtProceso.Text = "Ecuación correcta, procede a dibujar";
        Recalcula();
    }
}
}
}

```

## Gráfico matemático en 3D, control por ratón

Se modifica el programa anterior para que tenga las siguientes funcionalidades:

1. El gráfico matemático aparece en una ventana secundaria permitiendo poder escalarlo.
2. Los giros se controlan con el ratón.

Moviendo el ratón en horizontal (izquierda a derecha o viceversa) ejecuta un giro en el eje Y, moviendo el ratón en vertical (arriba abajo o viceversa) ejecuta un giro en el eje X. Con esos dos giros es más que suficiente, no es requerido en sí, el giro en Z. Luego la matriz de giro se puede optimizar sólo teniendo en cuenta los giros en X y en Y.

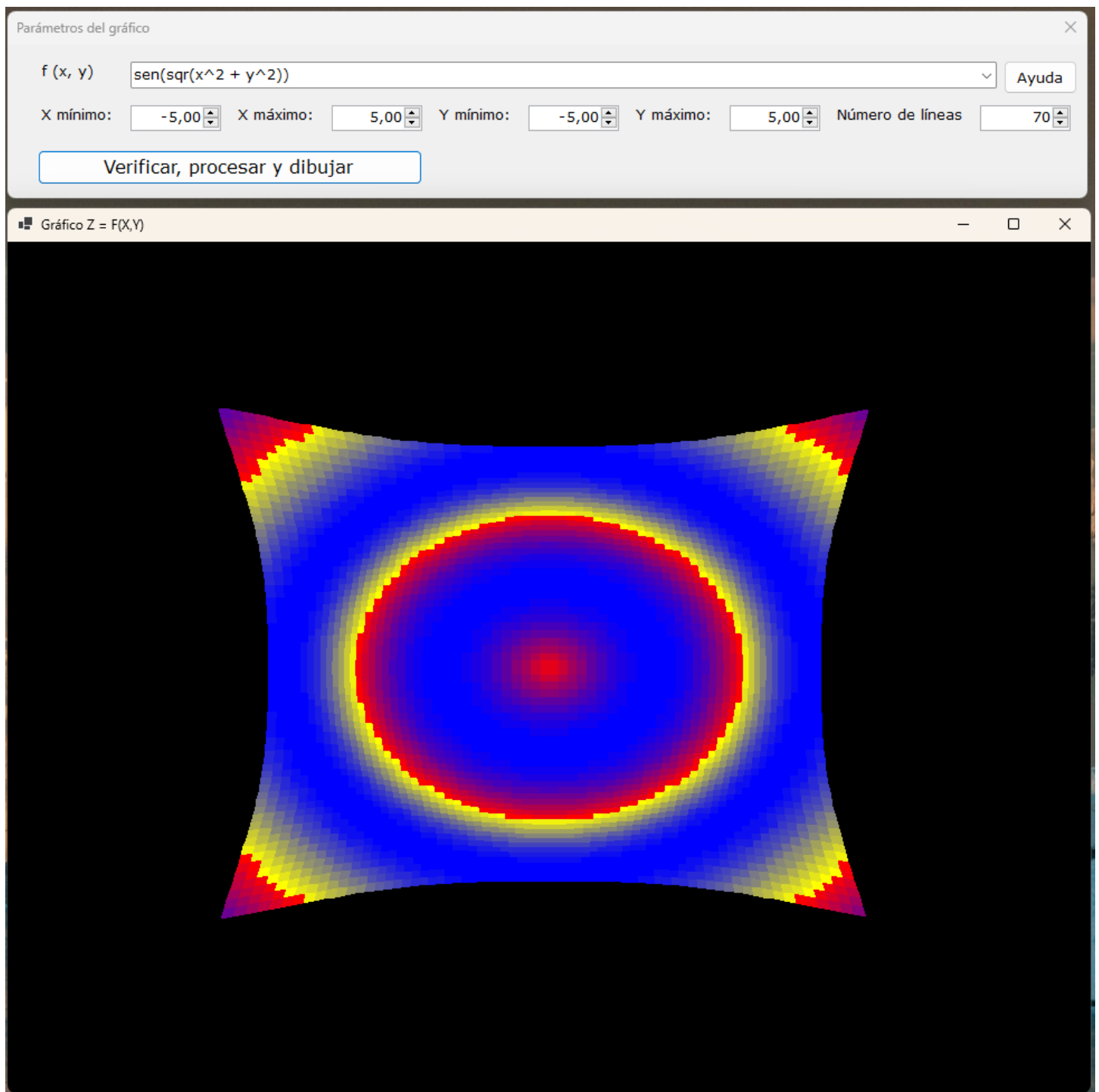


Ilustración 29: Inicio de la aplicación

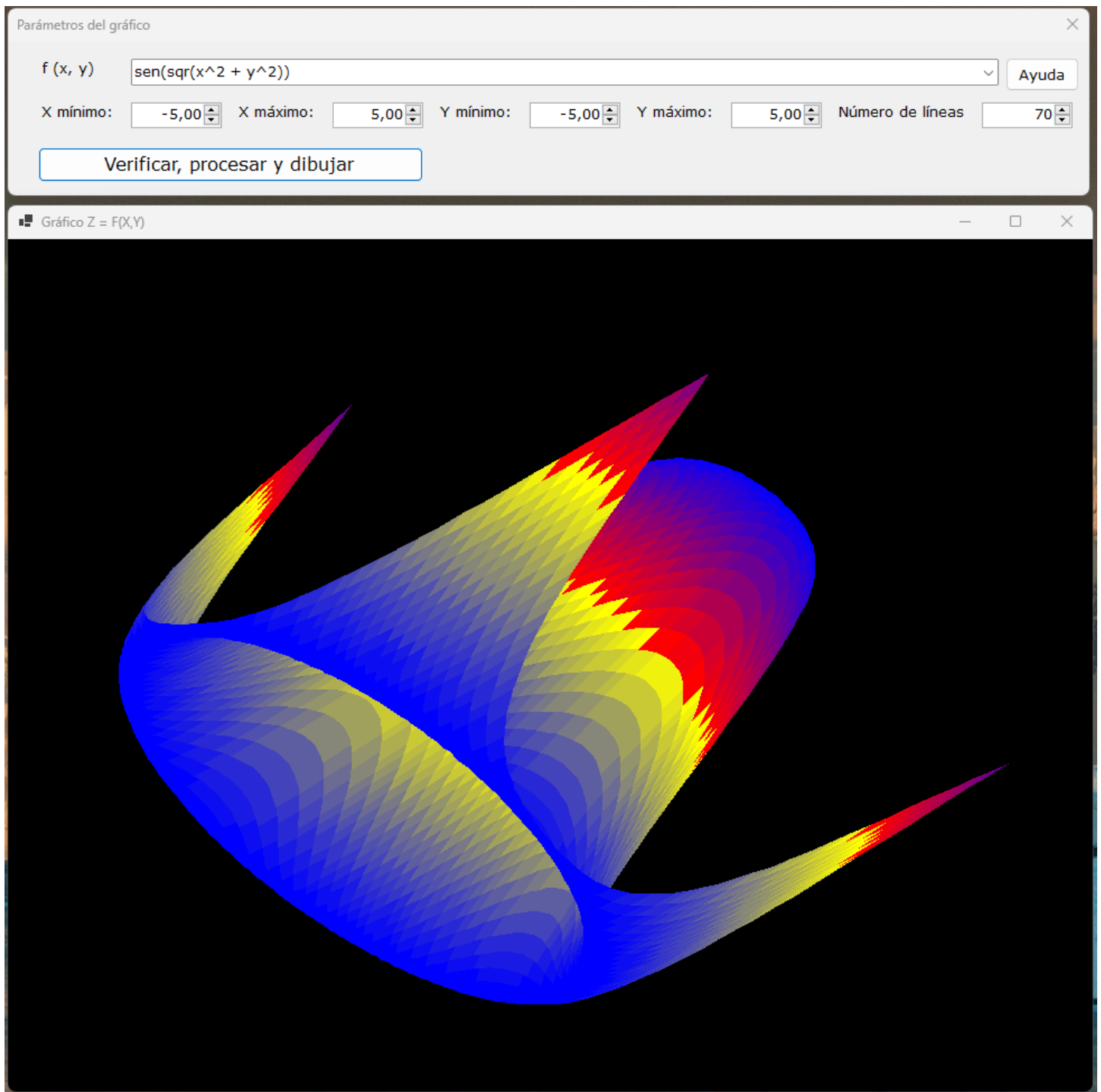


Ilustración 30: Se gira con el ratón

El giro se logra manteniendo presionado el botón izquierdo del ratón y moviendo el ratón.

```

namespace Grafico3D {

    //El gráfico se hace con polígonos
    public class Poligono : IComparable {
        //Coordenadas de dibujo del polígono
        Point Punto1, Punto2, Punto3, Punto4;

        //Profundidad media de Z al girarse las coordenadas
        double Centro;

        //Color de relleno del polígono
        Color ColorZ;

        public Poligono(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4,
            int Y4, double Zg1, double Zg2, double Zg3, double Zg4, double Z1, double
            Z2, double Z3, double Z4, List<Color> colorList) {
            Punto1 = new(X1, Y1);
            Punto2 = new(X2, Y2);
            Punto3 = new(X3, Y3);
            Punto4 = new(X4, Y4);

            //Usado para ordenar los polígonos en la lista
            Centro = (Zg1 + Zg2 + Zg3 + Zg4) / 4;

            //El valor Z sin girar es el que determina
            //el color del polígono
            int TotalColores = colorList.Count;
            double PromedioZ = (Z1 + Z2 + Z3 + Z4 + 2) / 4;
            int ColorEscoge = (int) Math.Floor(TotalColores * PromedioZ);
            ColorZ = colorList[ColorEscoge];
        }

        //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
        }

        //Hace el gráfico del polígono
        public void Dibuja(Graphics Lienzo) {

```



```

//Pone un color de fondo al polígono
//para borrar lo que hay detrás
Point[] ListaPuntos = [Punto1, Punto2, Punto3, Punto4];

//Dibuja el polígono relleno y su perímetro
Brush Relleno = new SolidBrush(ColorZ);
Lienzo.FillPolygon(Relleno, ListaPuntos);
}
}
}

```

M/012.7z/Punto.cs

```

namespace Grafico3D {

//Calcula el punto X,Y,Z de la ecuación
public class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xg, Yg, Zg; //Coordenadas al girar
    private double PlanoX, PlanoY; //Proyección o sombra
    public int Xp, Yp; //En Pantalla

//Calcula el valor de Z, guarda los valores de X, Y
public void Calcular(double X, double Y, Evaluador4 Evalua) {
    this.X = X;
    this.Y = Y;
    Evalua.DarValorVariable('x', X);
    Evalua.DarValorVariable('y', Y);
    Z = Evalua.Evaluar();
    if (double.IsNaN(Z) || double.IsInfinity(Z))
        Z = 0;
}

//Retorna el valor de Z sin girar
public double getZ() {
    return Z;
}

//Retorna el valor de Z al girar
public double getZg() {
    return Zg;
}

//Normaliza punto y luego lo ubica entre -0.5 y 0.5
public void Normaliza(double MinX, double MinY, double MinZ,
    double MaxX, double MaxY, double MaxZ) {

```

```

X = (X - MinX) / (MaxX - MinX) - 0.5;
Y = (Y - MinY) / (MaxY - MinY) - 0.5;
Z = (Z - MinZ) / (MaxZ - MinZ) - 0.5;
}

//Gira el punto
public void Giro(double[,] Mt) {
    Xg = X * Mt[0, 0] + Y * Mt[1, 0] + Z * Mt[2, 0];
    Yg = X * Mt[0, 1] + Y * Mt[1, 1] + Z * Mt[2, 1];
    Zg = X * Mt[0, 2] + Y * Mt[1, 2] + Z * Mt[2, 2];
}

//Convierte de 3D a 2D (segunda dimensión)
public void Proyecta(double ZPersona) {
    PlanoX = Xg * ZPersona / (ZPersona - Zg);
    PlanoY = Yg * ZPersona / (ZPersona - Zg);
}

//Convierte 2D real a 2D pantalla
public void Pantalla(int XpFin, int YpFin) {
    //Constante de transformación, nacida de girar la
    //figura en los ejes X, Y, Z (de 0 a 360 grados),
    //porque está contenida en un cubo de 1*1*1 cuyo centroide
    //está en 0,0,0
    double C = 1.758630875;

    //Cuadra en pantalla física
    Xp = Convert.ToInt32(XpFin / C * (PlanoX + C/2.0));
    Yp = Convert.ToInt32(YpFin / C * (PlanoY + C/2.0));
}
}
}

```

M/012.7z/Graf3D.cs

```

namespace Grafico3D {
    public class Graf3D {
        //Arreglo bidimensional que tiene los puntos
        public Punto[][] MtPunto;

        double MinX, MinY, MaxX, MaxY, MinZ, MaxZ;
        double AnguloX, AnguloY;
        double ZPersona;
        int NumLineas;
        int AnchoVentana, AltoVentana;
    }
}

```

```

//Listado de poligonos
public List<Poligono> Poligonos;

//Colores para pintar la malla
List<Color> ListaColores;

//La ecuación
Evaluador4 Evalua;

public void Inicializa() {
    //Angulos de giro
    this.AnguloX = 0;
    this.AnguloY = 0;

    //Distancia de la persona al plano
    ZPersona = 5;

    //Inicializa polígonos
    Poligonos = [];

    int NumColores = 40; // Número de colores a generar
    ListaColores = GeneraGradienteColor(NumColores);
}

public void setAltoAnchoVentana(int AltoVentana, int AnchoVentana) {
    this.AnchoVentana = AnchoVentana;
    this.AltoVentana = AltoVentana;
}

public void setAnguloX(double AnguloX) {
    this.AnguloX = AnguloX;
}

public void setAnguloY(double AnguloY) {
    this.AnguloY = AnguloY;
}

public double getAnguloX() { return this.AnguloX; }
public double getAnguloY() { return this.AnguloY; }

public void Calcula(double MinX, double MaxX, double MinY, double MaxY,
int NumLineas, Evaluador4 Evalua) {
    //Valores del formulario
    this.MinX = MinX;
    this.MaxX = MaxX;
    this.MinY = MinY;
    this.MaxY = MaxY;
}

```

```

//Número de líneas que forma la malla
this.NumLineas = NumLineas;

//La ecuación
this.Evalua = Evalua;

IniciaMatrizPuntos();
CalculaXYZ();
ExtremosZ();
NormalizaCalculos();
GiroProyectaCuadra();
}

//Guarda los puntos calculados en una matriz bidimensional
private void IniciaMatrizPuntos() {
    //Arreglo de arreglos
    MtPunto = new Punto[NumLineas][];
    for (int Fila = 0; Fila < MtPunto.Length; Fila++) {
        MtPunto[Fila] = new Punto[NumLineas];
        for (int Columna = 0; Columna < MtPunto[Fila].Length; Columna++) {
            MtPunto[Fila][Columna] = new Punto();
        }
    }
}

private void CalculaXYZ() {
    //Calcula las coordenadas XYZ originales
    double IncrY = (MaxY - MinY) / NumLineas;
    double IncrX = (MaxX - MinX) / NumLineas;
    double X = MinX;
    double Y = MinY;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        X = MinX;
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Calcular(X, Y, Evalua);
            X += IncrX;
        }
        Y += IncrY;
    }
}

private void ExtremosZ() {
    MinZ = double.MaxValue;
    MaxZ = double.MinValue;

    for (int Fila = 0; Fila < NumLineas; Fila++) {
        for (int Columna = 0; Columna < NumLineas; Columna++) {

```

```

        //Los valores extremos de Z
        if (MtPunto[Fila][Columna].getZ() < MinZ)
            MinZ = MtPunto[Fila][Columna].getZ();

        if (MtPunto[Fila][Columna].getZ() > MaxZ)
            MaxZ = MtPunto[Fila][Columna].getZ();
    }
}

private void NormalizaCalculos() {
    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++)
            MtPunto[Fila][Columna].Normaliza(MinX, MinY, MinZ, MaxX, MaxY,
MaxZ);
}

public void GiroProyectaCuadra() {
    //Genera la matriz de rotación
    double CosX = Math.Cos(AnguloX * Math.PI / 180);
    double SinX = Math.Sin(AnguloX * Math.PI / 180);
    double CosY = Math.Cos(AnguloY * Math.PI / 180);
    double SinY = Math.Sin(AnguloY * Math.PI / 180);

    //Matriz de Rotación (sólo de X, Y)

    //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
    double[,] MtRota = new double[3, 3] {
        {CosY, 0, SinY},
        {SinX*SinY, CosX, -SinX*CosY},
        {-CosX*SinY, SinX, CosX*CosY}
    };

    for (int Fila = 0; Fila < NumLineas; Fila++)
        for (int Columna = 0; Columna < NumLineas; Columna++) {
            MtPunto[Fila][Columna].Giro(MtRota);
            MtPunto[Fila][Columna].Proyecta(ZPersona);
            MtPunto[Fila][Columna].Pantalla(AnchoVentana, AltoVentana);
        }

    //Genera los poligonos
    GeneraPoligonos();

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

```

```

//Usa 4 coordenadas XY de pantalla para hacer cada polígono
private void GeneraPoligonos() {
    Poligonos.Clear();
    for (int Fila = 0; Fila < NumLineas - 1; Fila++)
        for (int Columna = 0; Columna < NumLineas - 1; Columna++) {
            int X1 = MtPunto[Fila][Columna].Xp;
            int Y1 = MtPunto[Fila][Columna].Yp;
            double Zg1 = MtPunto[Fila][Columna].getZg();
            double Z1 = MtPunto[Fila][Columna].getZ();

            int X2 = MtPunto[Fila][Columna + 1].Xp;
            int Y2 = MtPunto[Fila][Columna + 1].Yp;
            double Zg2 = MtPunto[Fila][Columna + 1].getZg();
            double Z2 = MtPunto[Fila][Columna + 1].getZ();

            int X3 = MtPunto[Fila + 1][Columna + 1].Xp;
            int Y3 = MtPunto[Fila + 1][Columna + 1].Yp;
            double Zg3 = MtPunto[Fila + 1][Columna + 1].getZg();
            double Z3 = MtPunto[Fila + 1][Columna + 1].getZ();

            int X4 = MtPunto[Fila + 1][Columna].Xp;
            int Y4 = MtPunto[Fila + 1][Columna].Yp;
            double Zg4 = MtPunto[Fila + 1][Columna].getZg();
            double Z4 = MtPunto[Fila + 1][Columna].getZ();

            Poligonos.Add(new Poligono(X1, Y1, X2, Y2, X3, Y3, X4, Y4, Zg1,
Zg2, Zg3, Zg4, Z1, Z2, Z3, Z4, ListaColores));
        }
    }

List<Color> GeneraGradienteColor(int NumColores) {
    List<Color> Colores = [];
    int Mitad = NumColores / 2;

    // Gradiente de azul a amarillo
    for (int Cont = 0; Cont < Mitad; Cont++) {
        int Rojo = (int)(255 * (Cont / (float)(NumColores - Mitad - 1)));
        int Verde = (int)(255 * (Cont / (float)(NumColores - Mitad - 1)));
        int Azul = 255 - (int)(255 * (Cont / (float)(NumColores - Mitad -
1)));
        Colores.Add(Color.FromArgb(Rojo, Verde, Azul));
    }

    // Gradiente de rojo a azul
    for (int Cont = 0; Cont < NumColores - Mitad; Cont++) {
        int Rojo = 255 - (int)(255 * (Cont / (float)(Mitad - 1)));
        int Verde = 0;

```

```

        int Azul = (int)(255 * (Cont / (float)(Mitad - 1)));
        Colores.Add(Color.FromArgb(Rojo, Verde, Azul));
    }

    return Colores;
}
}
}

```

M/012.7z/Grafico.cs

```

namespace Grafico3D {
    public partial class Grafico : Form {
        public Graf3D graf3D;
        private bool MousePresionado = false;
        private int MouseAntesX, MouseAntesY;

        public Grafico() {
            InitializeComponent();
        }

        private void Grafico_Paint(object sender, PaintEventArgs e) {
            Graphics Lienzo = e.Graphics;

            for (int Cont = 0; Cont < graf3D.Poligonos.Count; Cont++)
                graf3D.Poligonos[Cont].Dibuja(Lienzo);
        }

        private void Grafico_Resize(object sender, EventArgs e) {
            //Cuadrar en pantalla
            graf3D.setAltoAnchoVentana(this.ClientSize.Height,
            this.ClientSize.Width);
            graf3D.GiroProyectaCuadra();
            Refresh();
        }

        private void Grafico_MouseDown(object sender, MouseEventArgs e) {
            if (e.Button == MouseButtons.Left) {
                MousePresionado = true;
                this.Cursor = Cursors.SizeAll;
                MouseAntesX = e.X;
                MouseAntesY = e.Y;
            }
        }

        private void Grafico_MouseUp(object sender, MouseEventArgs e) {

```

```

        if (e.Button == MouseButton.Left) {
            MousePresionado = false;
            this.Cursor = Cursors.Default;
        }
    }

    private void Grafico_MouseMove(object sender, MouseEventArgs e) {
        if (MousePresionado) {
            if (e.X > MouseAntesX)
                graf3D.setAnguloY(graf3D.getAnguloY() - 2);
            else if (e.X < MouseAntesX)
                graf3D.setAnguloY(graf3D.getAnguloY() + 2);

            if (e.Y > MouseAntesY)
                graf3D.setAnguloX(graf3D.getAnguloX() + 2);
            else if (e.Y < MouseAntesY)
                graf3D.setAnguloX(graf3D.getAnguloX() - 2);

            MouseAntesX = e.X;
            MouseAntesY = e.Y;
            graf3D.GiroProyectaCuadra();
            Refresh();
        }
    }

    private void Grafico_FormClosed(object sender, FormClosedEventArgs e) {
        Application.Exit();
    }
}

```

M/012.7z/Form1.cs

```

namespace Grafico3D {
    public partial class Form1 : Form {
        Graf3D graf3D;
        Evaluador4 Evaluador;
        Grafico Ventana;

        public Form1() {
            InitializeComponent();

            graf3D = new Graf3D();
            graf3D.Inicializa();
        }
    }
}

```



```

private void numXminimo_ValueChanged(object sender, EventArgs e) {
    if (numXminimo.Value >= numXmaximo.Value)
        numXminimo.Value = numXmaximo.Value - 1;
    Recalcula();
}

private void numXmaximo_ValueChanged(object sender, EventArgs e) {
    if (numXmaximo.Value <= numXminimo.Value)
        numXmaximo.Value = numXminimo.Value + 1;
    Recalcula();
}

private void numYminimo_ValueChanged(object sender, EventArgs e) {
    if (numYminimo.Value >= numYmaximo.Value)
        numYminimo.Value = numYmaximo.Value - 1;
    Recalcula();
}

private void numYmaximo_ValueChanged(object sender, EventArgs e) {
    if (numYmaximo.Value <= numYminimo.Value)
        numYmaximo.Value = numYminimo.Value + 1;
    Recalcula();
}

private void numLineas_ValueChanged(object sender, EventArgs e) {
    Recalcula();
}

//Si el usuario ingresa una nueva ecuación
private void btnProcesa_Click(object sender, EventArgs e) {
    string Ecuacion = cboEcuacion.Text;
    int Sintaxis = Evaluador.Analizar(Ecuacion);
    if (Sintaxis > 0) {
        string MensajeError = Evaluador.MensajeError(Sintaxis);
        MessageBox.Show(MensajeError, "Error de sintaxis",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else
        Recalcula();
}

//Si la ecuación ya ha sido analizada, se llama a este método
//que es para calcular cuando cambian los valores mínimos y máximos
//de X y Y
private void Recalcula() {
    graf3D.Calcula((double)numXminimo.Value,
        (double)numXmaximo.Value,
        (double)numYminimo.Value,

```

```

        (double) numYmaximo.Value,
        (int) numLineas.Value, Evaluador);
Ventana.graf3D = graf3D;
Ventana.Refresh();
}

private void Form1_Shown(object sender, EventArgs e) {
    this.Location = new Point(10, 10);

    //Llama la ventana que muestra el gráfico
    Ventana = new Grafico();
    Ventana.Location = new Point(this.Location.X, this.Location.Y +
this.Height);
    Ventana.Show();

    //Cuadrar en pantalla
    graf3D.setAltoAnchoVentana(Ventana.ClientSize.Height,
Ventana.ClientSize.Width);

    //La primera vez calcula la ecuación
    Evaluador = new Evaluador4();
    string Ecuacion = cboEcuacion.Text;
    Evaluador.Analizar(Ecuacion);
    Recalcula();
}

private void btnAyuda_Click(object sender, EventArgs e) {
    string Ayuda = "USE:\r\n\r\nseno: sen( \r\ncoseno: cos( \r\ntangente:
tan( \r\nvalor absoluto: abs( \r\narcoseno: asn( \r\narcocoseno: acs(
\r\narcotangente: atn( \r\nlogaritmo natural: log( \r\nexponencial: exp(
\r\nraiz cuadrada: sqr( \r\n\r\nSuma: + \r\nResta: - \r\nMultiplicación: *
\r\nDivisión: / \r\nPotencia: ^ \r\nParéntesis: ( ) \r\nVariables: x y
";
    MessageBox.Show(Ayuda, "Expresiones matemáticas",
MessageBoxButtons.OK, MessageBoxIcon.Information);
}
}
}

```

## 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](https://en.wikipedia.org/wiki/Spherical_coordinate_system)

Por ejemplo, esta ecuación:

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

Los pasos para dibujar el gráfico polar 3D son:

Paso 1:  $\varphi$  (phi) puede ir de 0 a  $2\pi$ ,  $\theta$  (theta) puede ir de 0 a  $2\pi$  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

La aplicación completa está en M/013.7z

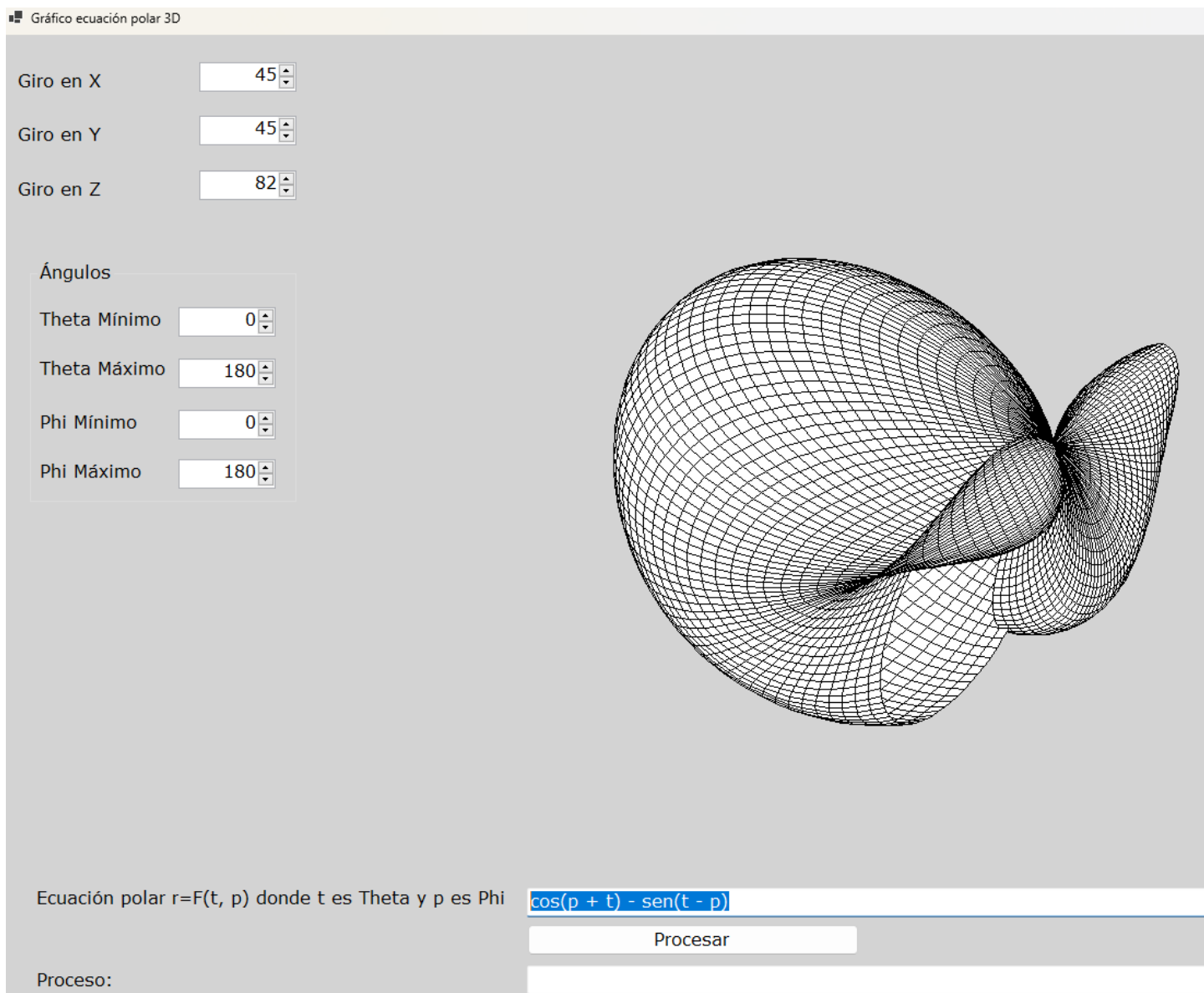


Ilustración 31: Gráfico ecuación polar 3D

Este es el código del gráfico:

```

namespace Grafico {
    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;

        //El valor de Z mas profundo del polígono
        public double ZMasFondo;

        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
ZMasFondo = 100;
if (Z1g < ZMasFondo) ZMasFondo = Z1g;
if (Z2g < ZMasFondo) ZMasFondo = Z2g;
if (Z3g < ZMasFondo) ZMasFondo = Z3g;
if (Z4g < ZMasFondo) ZMasFondo = Z4g;

//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.ZMasFondo < ZMasFondo) return 1;
    if (OrdenCompara.ZMasFondo > ZMasFondo) return -1;
    return 0;
    //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-
a-property-in-the-object
}
}
}

```

M/013.7z/Graf3D.cs

```

namespace Grafico {
    class Graf3D {
        //Donde almacena los poligonos
        private List<Poligono> poligono;

        public void CalculaEcuacion(int numLineas, Evaluador4 Evalua, double
MinTheta, double MaxTheta, double MinPhi, double MaxPhi) {
            //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 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

        //PUNTO 1 DEL POLIGONO
        double Theta1 = Th * Math.PI / 180;
        double Phi1 = Phi * Math.PI / 180;

        Evalua.DarValorVariable('p', Phi1);
        Evalua.DarValorVariable('t', Theta1);
        double R1 = Evalua.Evaluar();
        if (double.IsNaN(R1) || double.IsInfinity(R1)) R1 = 0;

        double X1 = R1 * Math.Cos(Phi1) * Math.Sin(Theta1);
        double Y1 = R1 * Math.Sin(Phi1) * Math.Sin(Theta1);
        double Z1 = R1 * Math.Cos(Theta1);

        //PUNTO 2 DEL POLIGONO
        double Theta2 = (Th + IncTheta) * Math.PI / 180;
        double Phi2 = Phi * Math.PI / 180;

        Evalua.DarValorVariable('p', Phi2);
        Evalua.DarValorVariable('t', Theta2);
        double R2 = Evalua.Evaluar();
        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);

        //PUNTO 3 DEL POLIGONO
        double Theta3 = (Th + IncTheta) * Math.PI / 180;
        double Phi3 = (Phi + IncPhi) * Math.PI / 180;

        Evalua.DarValorVariable('p', Phi3);
        Evalua.DarValorVariable('t', Theta3);
        double R3 = Evalua.Evaluar();
    }

```



```

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);

//PUNTO 4 DEL POLIGONO
double Theta4 = Th * Math.PI / 180;
double Phi4 = (Phi + IncPhi) * Math.PI / 180;

Evalua.DarValorVariable('p', Phi4);
Evalua.DarValorVariable('t', Theta4);
double R4 = Evalua.Evaluar();
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);

//VALORES EXTREMOS
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);
}

public void CalculaGrafico(double AngX, double AngY, double AngZ) {
    int ZPersona = 5;
    int XpIni = 400;
    int YpIni = 0;
    int XpFin = 1200;
    int YpFin = 800;

    //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 Lapiz, Brush Relleno) {
    for (int Cont = 0; Cont < poligono.Count; Cont++)
        poligono[Cont].Dibuja(Lienzo, Lapiz, Relleno);
}
}
}

```

M/013.7z/Form1.cs

```

namespace Grafico {
    public partial class Form1 : Form {
        //El objeto que se encarga de calcular
        //y cuadrar en pantalla la ecuación polar
        Graf3D Grafico3D;

        //Evaluador de expresiones
        Evaluador4 Evaluador;

        //Número de líneas a dibujar
        int NumLineas;

        public Form1() {
            InitializeComponent();
            Grafico3D = new Graf3D();

            Evaluador = new Evaluador4();
            Evaluador.Analizar(txtEcuacion.Text);
        }
    }
}

```

```

        NumLineas = 80;
        Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numThetaMin.Value, (double)numThetaMax.Value,
(double)numPhiMin.Value, (double)numPhiMax.Value);
    }

    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);

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

    private void btnProcesar_Click(object sender, EventArgs e) {
        string Ecuacion = txtEcuacion.Text;
        int Sintaxis = Evaluador.Analizar(Ecuacion);

        if (Sintaxis > 0) {
            txtProceso.Text = Evaluador.MensajeError(Sintaxis);
        }
        else {
            txtProceso.Text = "Ecuación correcta, procede a dibujar";
            Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numThetaMin.Value, (double)numThetaMax.Value,
(double)numPhiMin.Value, (double)numPhiMax.Value);
            Refresh();
        }
    }
}

```

```

private void numThetaMin_ValueChanged(object sender, EventArgs e) {
    if (numThetaMin.Value >= numThetaMax.Value)
        numThetaMin.Value = numThetaMax.Value - 1;
    Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numThetaMin.Value, (double)numThetaMax.Value,
(double)numPhiMin.Value, (double)numPhiMax.Value);
    Refresh();
}

private void numThetaMax_ValueChanged(object sender, EventArgs e) {
    if (numThetaMax.Value <= numThetaMin.Value)
        numThetaMax.Value = numThetaMin.Value + 1;
    Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numThetaMin.Value, (double)numThetaMax.Value,
(double)numPhiMin.Value, (double)numPhiMax.Value);
    Refresh();
}

private void numPhiMin_ValueChanged(object sender, EventArgs e) {
    if (numPhiMin.Value >= numPhiMax.Value)
        numPhiMin.Value = numPhiMax.Value - 1;
    Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numThetaMin.Value, (double)numThetaMax.Value,
(double)numPhiMin.Value, (double)numPhiMax.Value);
    Refresh();
}

private void numPhiMax_ValueChanged(object sender, EventArgs e) {
    if (numPhiMax.Value <= numPhiMin.Value)
        numPhiMax.Value = numPhiMin.Value + 1;
    Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numThetaMin.Value, (double)numThetaMax.Value,
(double)numPhiMin.Value, (double)numPhiMax.Value);
    Refresh();
}
}
}

```

## 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](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 Xg, Yg, Zg

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

Paso 7: Con Xg, Yg, Zg 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.

El proyecto completo está en M/014.7z

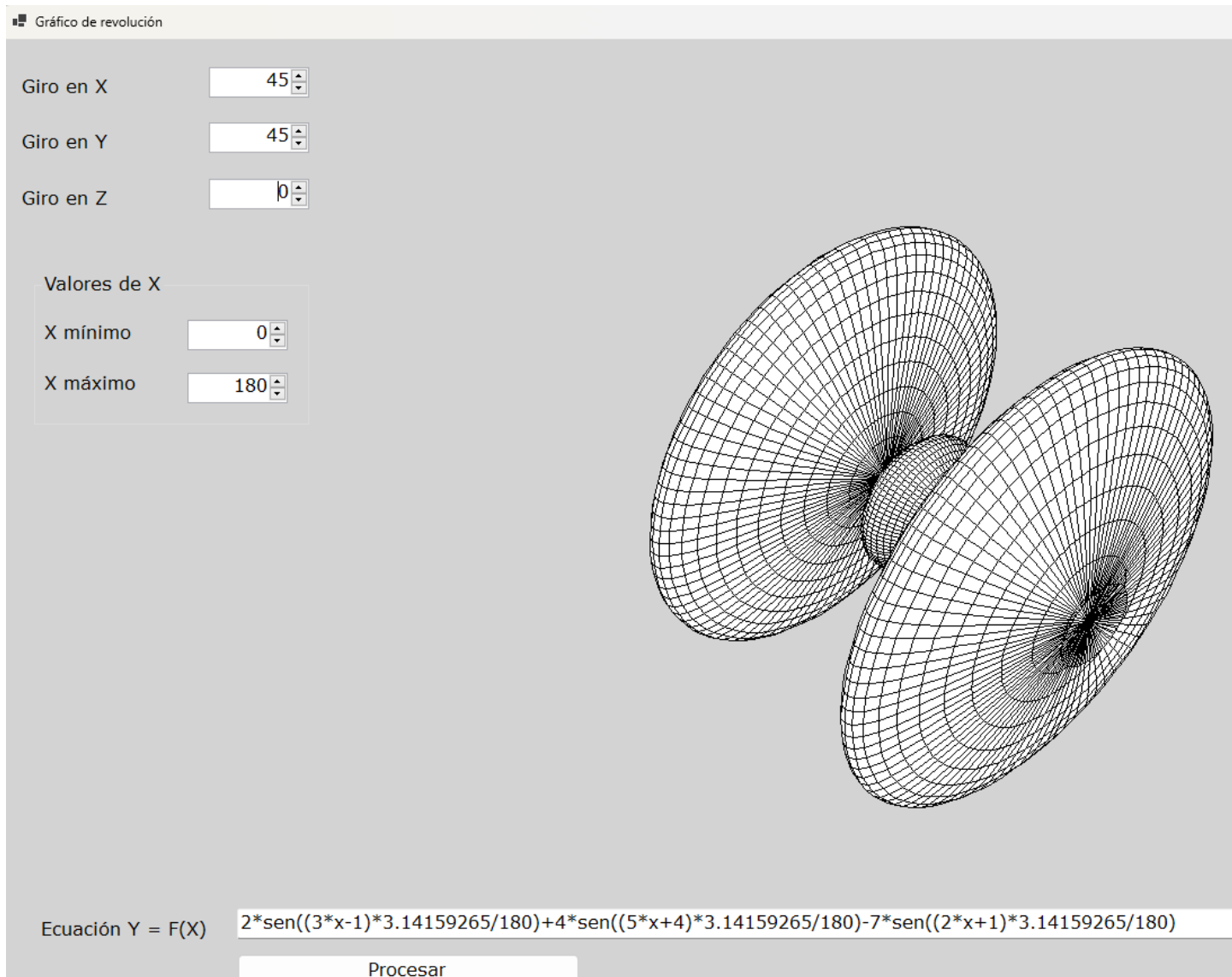


Ilustración 32: Sólido de revolución

```

namespace Grafico {
    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;

        //El valor de Z mas profundo del polígono
        public double ZMasFondo;

        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
/*ZMasFondo = Double.MaxValue;
if (Z1g < ZMasFondo) ZMasFondo = Z1g;
if (Z2g < ZMasFondo) ZMasFondo = Z2g;
if (Z3g < ZMasFondo) ZMasFondo = Z3g;
if (Z4g < ZMasFondo) ZMasFondo = Z4g;*/
ZMasFondo = (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.ZMasFondo < ZMasFondo) return 1;
    if (OrdenCompara.ZMasFondo > ZMasFondo) return -1;
    return 0;
    //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-
    a-property-in-the-object
}
}
}

```

M/014.7z/Graf3D.cs

```

namespace Grafico {
    class Graf3D {
        //Donde almacena los poligonos
        private List<Poligono> poligono;

        public void CalculaEcuacion(int numLineas, Evaluador4 Evalua, double
minX, double maxX) {
            //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;
        Evalua.DarValorVariable('x', X1);
        double Y1 = Evalua.Evaluar();
        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;
        Evalua.DarValorVariable('x', X2);
        double Y2 = Evalua.Evaluar();
        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);
}

public void CalculaGrafico(double AngX, double AngY, double AngZ) {
    int ZPersona = 5;
    int XpIni = 400;
    int YpIni = 0;
    int XpFin = 1200;
    int YpFin = 800;

```

```

//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 Lapiz, Brush Relleno) {
    for (int Cont = 0; Cont < poligono.Count; Cont++)
        poligono[Cont].Dibuja(Lienzo, Lapiz, Relleno);
}
}
}

```

```

namespace Grafico {
    public partial class Form1 : Form {
        //El objeto que se encarga de calcular
        //y cuadrar en pantalla la ecuación polar
        Graf3D Grafico3D;

        //Evaluador de expresiones
        Evaluador4 Evaluador;

        //Número de líneas a dibujar
        int NumLineas;

        public Form1() {
            InitializeComponent();
            Grafico3D = new Graf3D();

            Evaluador = new Evaluador4();
            Evaluador.Analizar(txtEcuacion.Text);

            NumLineas = 80;
            Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numXminimo.Value, (double)numXmaximo.Value);
        }

        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);

```

```

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

private void btnProcesar_Click(object sender, EventArgs e) {
    string Ecuacion = txtEcuacion.Text;
    int Sintaxis = Evaluador.Analizar(Ecuacion);

    if (Sintaxis > 0) {
        txtProceso.Text = Evaluador.MensajeError(Sintaxis);
    }
    else {
        txtProceso.Text = "Ecuación correcta, procede a dibujar";
        Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numXminimo.Value, (double)numXmaximo.Value);
        Refresh();
    }
}

private void numXminimo_ValueChanged(object sender, EventArgs e) {
    if (numXminimo.Value >= numXmaximo.Value)
        numXminimo.Value = numXmaximo.Value - 1;
    Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numXminimo.Value, (double)numXmaximo.Value);
    Refresh();
}

private void numXmaximo_ValueChanged(object sender, EventArgs e) {
    if (numXmaximo.Value <= numXminimo.Value)
        numXmaximo.Value = numXminimo.Value + 1;
    Grafico3D.CalculaEcuacion(NumLineas, Evaluador,
(double)numXminimo.Value, (double)numXmaximo.Value);
    Refresh();
}
}
}
}

```