C# Y .NET 10
Parte 13. Gráficos 3D

2026-01

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelife@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 2026 ® Enlace: https://visualstudio.microsoft.com/es/vs/

En memoria de mi gata Capuchina.

Autor: Rafael Alberto Moreno Parra

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 (Xplano, Yplano). Hay que considerar la distancia de la persona u observador a ese objeto (Zpersona). Si el observador 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 Xplano, Yplano considerando la distancia del observador (ZPersona) es:

$$Xplano = (ZPersona * X) / (ZPersona - Z)$$

$$Yplano = (ZPersona * Y) / (ZPersona - Z)$$

Demostración:

Paso 1

Se pone el valor de X = 0 para ver sólo el comportamiento de Y, este sería el gráfico

Ilustración 2: Esquema de proyección

Paso 2

Es conocido entonces los valores de Y, Z y Zpersona. Se necesita hallar el valor de ¿? Que sería Yplano, ¿Cómo? La respuesta está en el ángulo Θ, más concretamente en la tangente que sería así:

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

Pero también se sabe que:

$$\tan\theta = \frac{¿?}{ZPersona}$$

Igualando:

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

Despejando:

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

Luego:

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

Igualmente, para Xplano sería:

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

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private int X, Y, Z; //Coordenadas originales
    public int PlanoX, PlanoY; //Proyección o sombra

    public Punto(int X, int Y, int Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Convierte de 3D a 2D (segunda dimensión)
    public void Proyecta(int ZPersona) {
      PlanoX = X * ZPersona / (ZPersona - Z);
      PlanoY = Y * ZPersona / (ZPersona - Z);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

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

    //Un cubo tiene 12 líneas de conexión
    private List<Conexion> conexiones;

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

      //Las 12 líneas para dibujar el cubo
      //punto inicial ---> punto final
      conexiones = new List<Conexion> {
        new Conexion(0, 1),
        new Conexion(1, 2),
        new Conexion(2, 3),
        new Conexion(3, 0),
        new Conexion(4, 5),
        new Conexion(5, 6),
        new Conexion(6, 7),
        new Conexion(7, 4),
        new Conexion(0, 4),
        new Conexion(1, 5),
        new Conexion(2, 6),
        new Conexion(3, 7)
      };

      //Convierte de 3D a 2D
      for (int Cont = 0; Cont < puntos.Count; Cont++)
        puntos[Cont].Proyecta(ZPersona);
    }


    //Dibuja el cubo
    public void Dibuja(Graphics lienzo, Pen lapiz) {
      for (int Cont = 0; Cont < conexiones.Count; Cont++) {
        int Inicio = conexiones[Cont].punto1;
        int Final = conexiones[Cont].punto2;
        lienzo.DrawLine(lapiz, puntos[Inicio].PlanoX, puntos[Inicio].PlanoY,
                  puntos[Final].PlanoX, puntos[Final].PlanoY);
      }
    }
  }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D de una figura tridimensional: un cubo
  public partial class Form1 : Form {
    public Form1() {
      InitializeComponent();
    }

    //Pinta la proyección
    private void Form1_Paint(object sender, PaintEventArgs e) {
      //Distancia de la persona (observador) al plano donde se
      //proyecta la "sombra" del cubo
      int ZPersona = 180;

      //Crea el objeto tridimensional
      Cubo Figura3D = new Cubo(ZPersona);

      //Dibuja el objeto tridimensional
      Graphics Lienzo = e.Graphics;
      Pen Lapiz = new Pen(Color.Blue, 3);
      Figura3D.Dibuja(Lienzo, Lapiz);
    }
  }
}
```

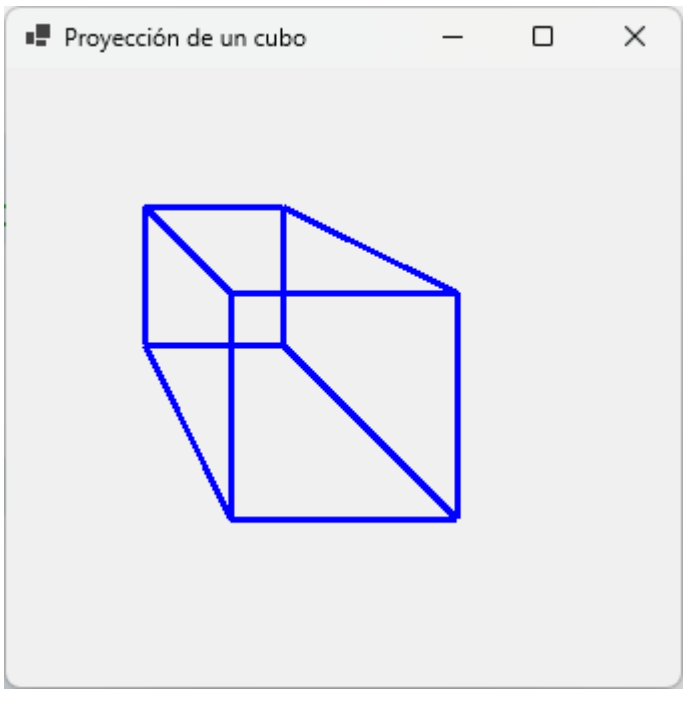


Ilustración 3: Proyección 3D a 2D de una figura tridimensional: un cubo

Girando un objeto 3D y mostrarlo en 2D

Dada una coordenada tridimensional (X, Y, Z) se quiere aplicar un giro en un determinado ángulo. Ese ángulo puede ser en X o en Y o en Z. Para aplicar el giro, se requiere una matriz de transformación. Estas serían las matrices:

Matriz de Giro en X

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = [X \quad Y \quad Z] * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(angX) & sen(angX) \\ 0 & -sen(angX) & \cos(angX) \end{bmatrix}$$

Aplicando:

$$Xgiro = X * 1 + Y * 0 + Z * 0$$

$$Ygiro = X * 0 + \mathrm{Y} * \cos(angX) - Z * sen(angX)$$

$$Zgiro = X * 0 + Y * sen(angX) + Z * \cos(angX)$$

Simplificando:

$$Xgiro = X$$

$$Ygiro = \mathrm{Y} * \cos(angX) - Z * sen(angX)$$

$$Zgiro = Y * sen(angX) + Z * \cos(angX)$$

Matriz de Giro en Y

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = [X \quad Y \quad Z] * \begin{bmatrix} \cos(angY) & 0 & -sen(angY) \\ 0 & 1 & 0 \\ sen(angY) & 0 & \cos(angY) \end{bmatrix}$$

Aplicando:

$$Xgiro = X * \cos(angY) + Y * 0 + Z * sen(angY)$$

$$Ygiro = X * 0 + Y * 1 + Z * 0$$

$$Zgiro = -X * sen(angY) + Y * 0 + Z * \cos(angY)$$

Simplificando:

$$Xgiro = X * \cos(angY) + Z * sen(angY)$$

$$Ygiro = Y$$

$$Zgiro = -X * sen(angY) + Z * \cos(angY)$$

Matriz de Giro en Z

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = [X \quad Y \quad Z] * \begin{bmatrix} \cos(angZ) & sen(angZ) & 0 \\ -sen(angZ) & \cos(\text{angZ}) & 0 \\ 0 & 0 & 1) \end{bmatrix}$$

Aplicando:

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

$$Ygiro = X * sen(angZ) + Y * \cos(angZ) + Z * 0$$

$$Zgiro = X * 0 + Y * 0 + Z * 1$$

Simplificando:

$$Xgiro = \text{X} * \cos(angZ) - Y * sen(angZ)$$

$$Ygiro = X * sen(angZ) + Y * \cos(angZ)$$

$$Zgiro = Z$$

Más información en: https://www.cs.buap.mx/~iolmos/graficacion/5_Transformaciones_geometricas_3D.pdf

M/002.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xg, Yg, Zg; //Coordenadas al girar
    public int PlanoX, PlanoY; //Proyección o sombra

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Convierte de 3D a 2D (segunda dimensión)
    public void Proyecta(double ZPersona) {
      PlanoX = Convert.ToInt32(Xg * ZPersona / (ZPersona - Zg));
      PlanoY = Convert.ToInt32(Yg * ZPersona / (ZPersona - Zg));
    }

    //Gira en X
    public void GiroX(double angulo) {
      double[,] Mt = new double[3, 3] {
        {1, 0, 0},
        {0, Math.Cos(angulo), Math.Sin(angulo)},
        {0, -Math.Sin(angulo), Math.Cos(angulo) }
      };

      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];
    }

    //Gira en Y
    public void GiroY(double angulo) {
      double[,] Mt = new double[3, 3] {
        {Math.Cos(angulo), 0, -Math.Sin(angulo)},
        {0, 1, 0},
        {Math.Sin(angulo), 0, Math.Cos(angulo) }
      };

      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];
    }

    //Gira en Z
    public void GiroZ(double angulo) {
      double[,] Mt = new double[3, 3] {
        {Math.Cos(angulo), Math.Sin(angulo), 0},
        {-Math.Sin(angulo), Math.Cos(angulo), 0},
        {0, 0, 1 }
      };

      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];
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  internal class Cubo {
    //Un cubo tiene 8 coordenadas espaciales X, Y, Z
    public List<Punto> puntos;

    //Un cubo tiene 12 líneas de conexión
    private List<Conexion> conexiones;

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

      //Las 12 líneas para dibujar el cubo
      //punto inicial ---> punto final
      conexiones = [
        new Conexion(0, 1),
        new Conexion(1, 2),
        new Conexion(2, 3),
        new Conexion(3, 0),
        new Conexion(4, 5),
        new Conexion(5, 6),
        new Conexion(6, 7),
        new Conexion(7, 4),
        new Conexion(0, 4),
        new Conexion(1, 5),
        new Conexion(2, 6),
        new Conexion(3, 7)
      ];
    }

    public void AplicaGiro(int CualAnguloGira, int ValorAngulo, int ZPersona) {
      double ang = ValorAngulo * Math.PI / 180;

      //Dependiendo de cuál ángulo gira
      for (int Cont = 0; Cont < puntos.Count; Cont++) {
        switch (CualAnguloGira) {
          case 0: puntos[Cont].GiroX(ang); break;
          case 1: puntos[Cont].GiroY(ang); break;
          case 2: puntos[Cont].GiroZ(ang); break;
        }
        puntos[Cont].Proyecta(ZPersona);
      }
    }

    //Dibuja el cubo
    public void Dibuja(Graphics lienzo, Pen lapiz) {
      for (int Cont = 0; Cont < conexiones.Count; Cont++) {
        int Inicio = conexiones[Cont].punto1;
        int Final = conexiones[Cont].punto2;
        lienzo.DrawLine(lapiz, puntos[Inicio].PlanoX, puntos[Inicio].PlanoY,
```

```
                puntos[Final].PlanoX, puntos[Final].PlanoY);
      }
    }
  }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D. Uso de giros en X, Y, Z pero tan sólo en uno

  public partial class Form1 : Form {
    //El cubo que se proyecta y gira
    Cubo Figura3D;
    int ZPersona = 180;

    public Form1() {
      InitializeComponent();

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

    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, ZPersona); //O 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, ZPersona); //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, ZPersona); //2 es giro en Z
      Refresh();
    }

    //Pinta la proyección
    private void Form1_Paint(object sender, PaintEventArgs e) {
      Graphics Lienzo = e.Graphics;
      Pen Lapiz = new(Color.Blue, 3);
      Figura3D.Dibuja(Lienzo, Lapiz);

      //Dibuja la línea de referencia del giro que parte de 0,0
      Pen Lapiz2 = new(Color.Red, 3);
      Lienzo.DrawLine(Lapiz2, 0, 0, Figura3D.puntos[0].PlanoX, Figura3D.puntos[0].PlanoY);
    }
  }
}
```

Ejemplo de ejecución:

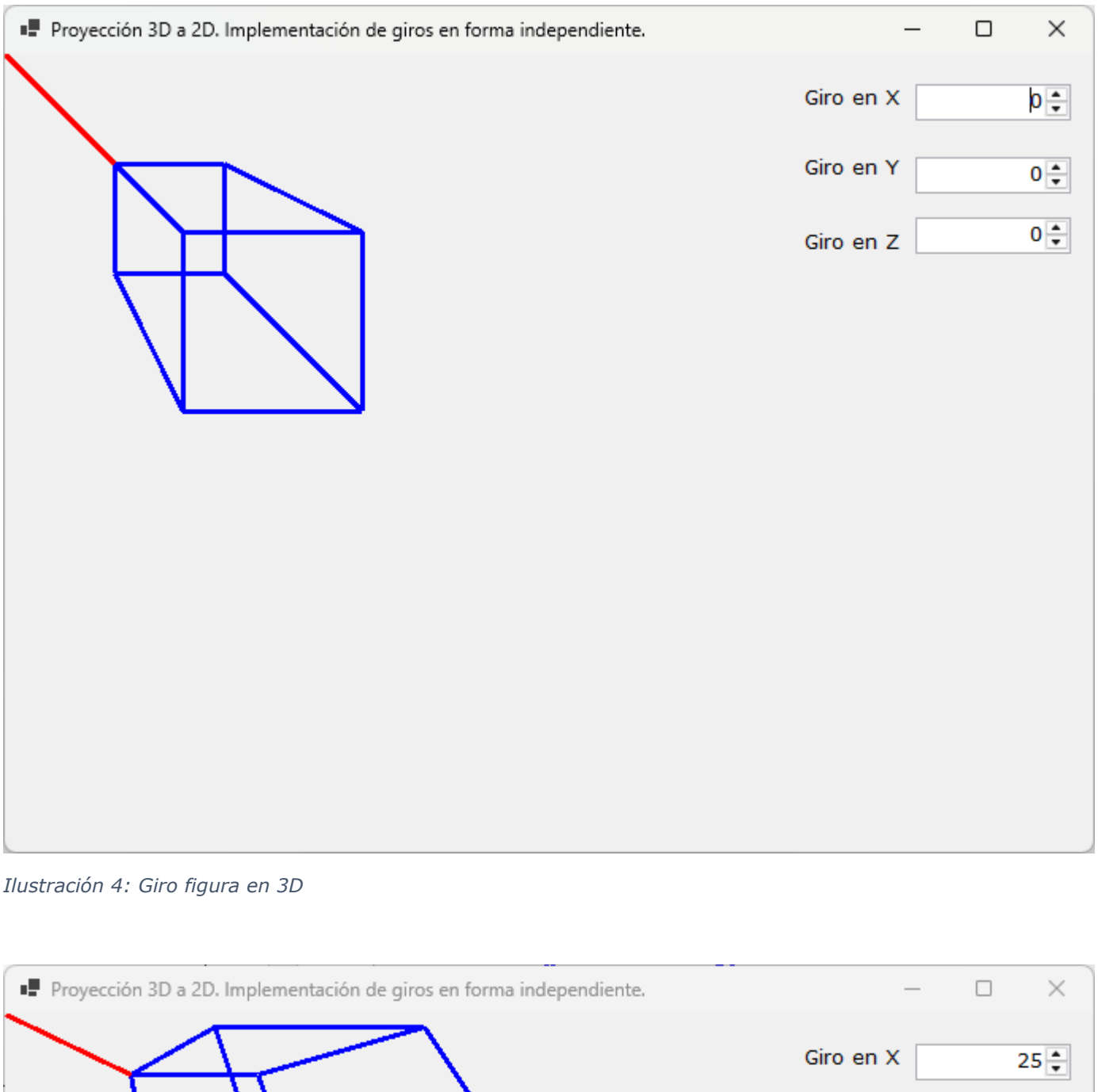


Ilustración 4: Giro figura en 3D

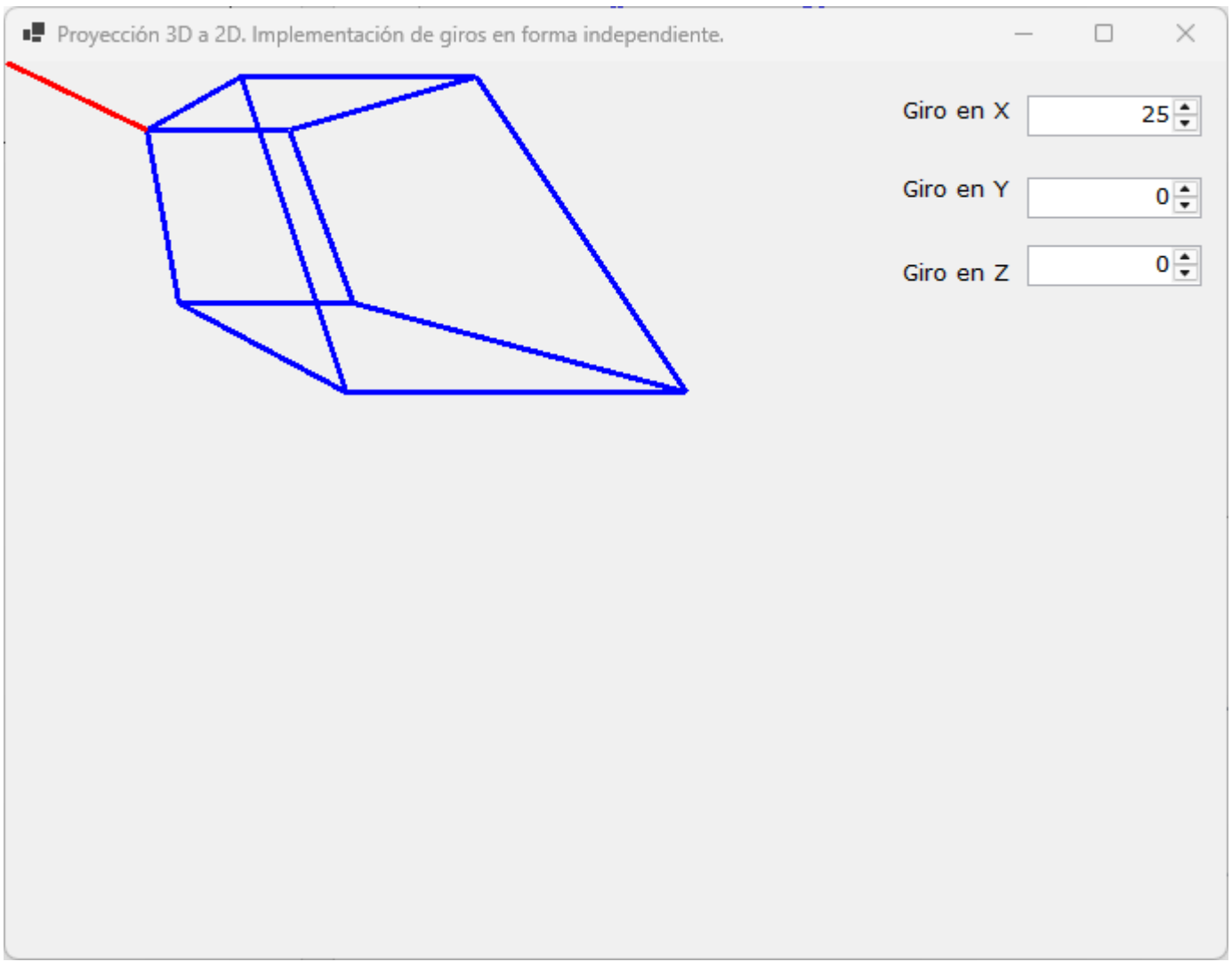


Ilustración 5: Giro en el eje X de la figura 3D. No hay giro en los otros ejes.

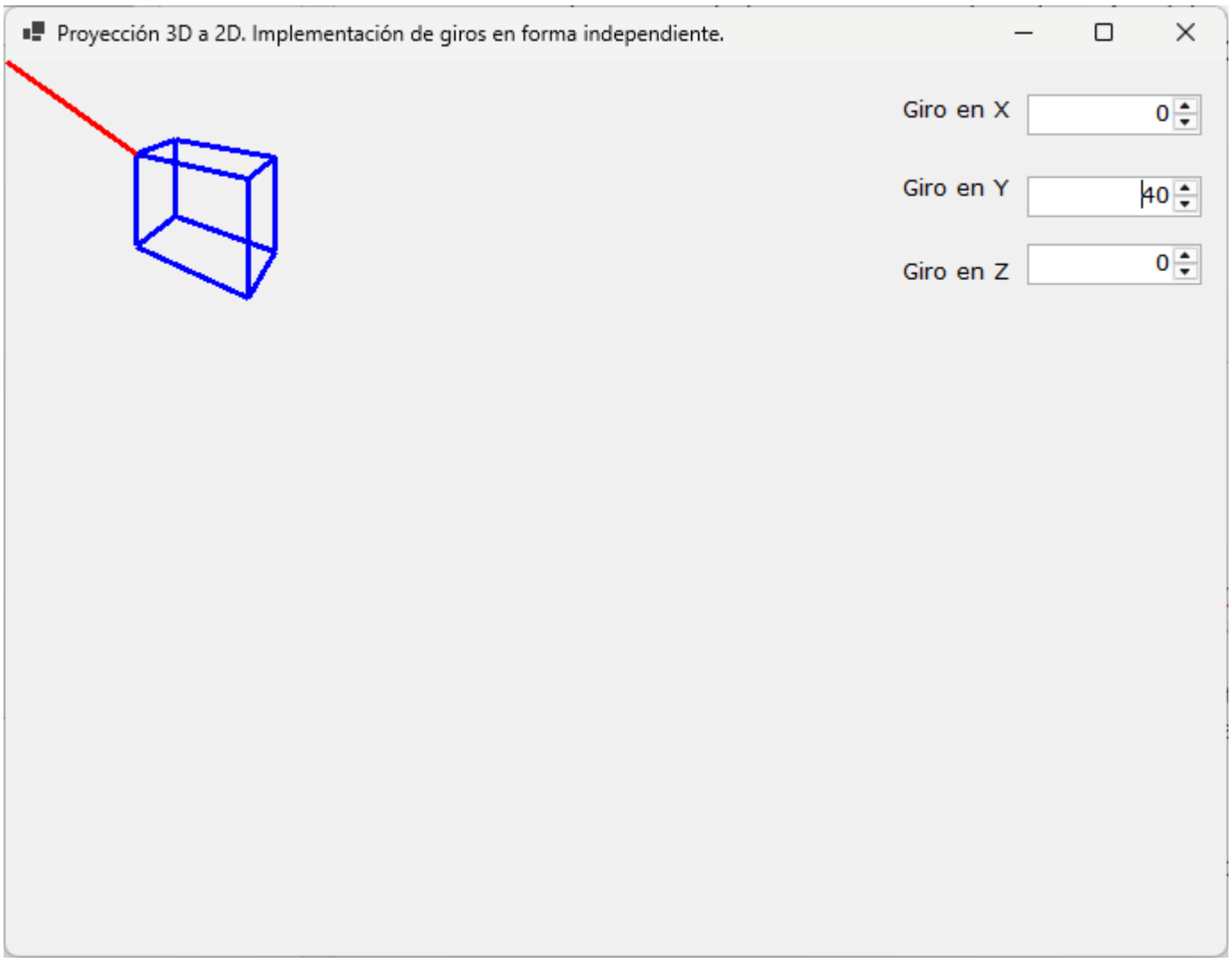


Ilustración 6: Giro en el eje Y de la figura 3D. No hay giro en los otros ejes.

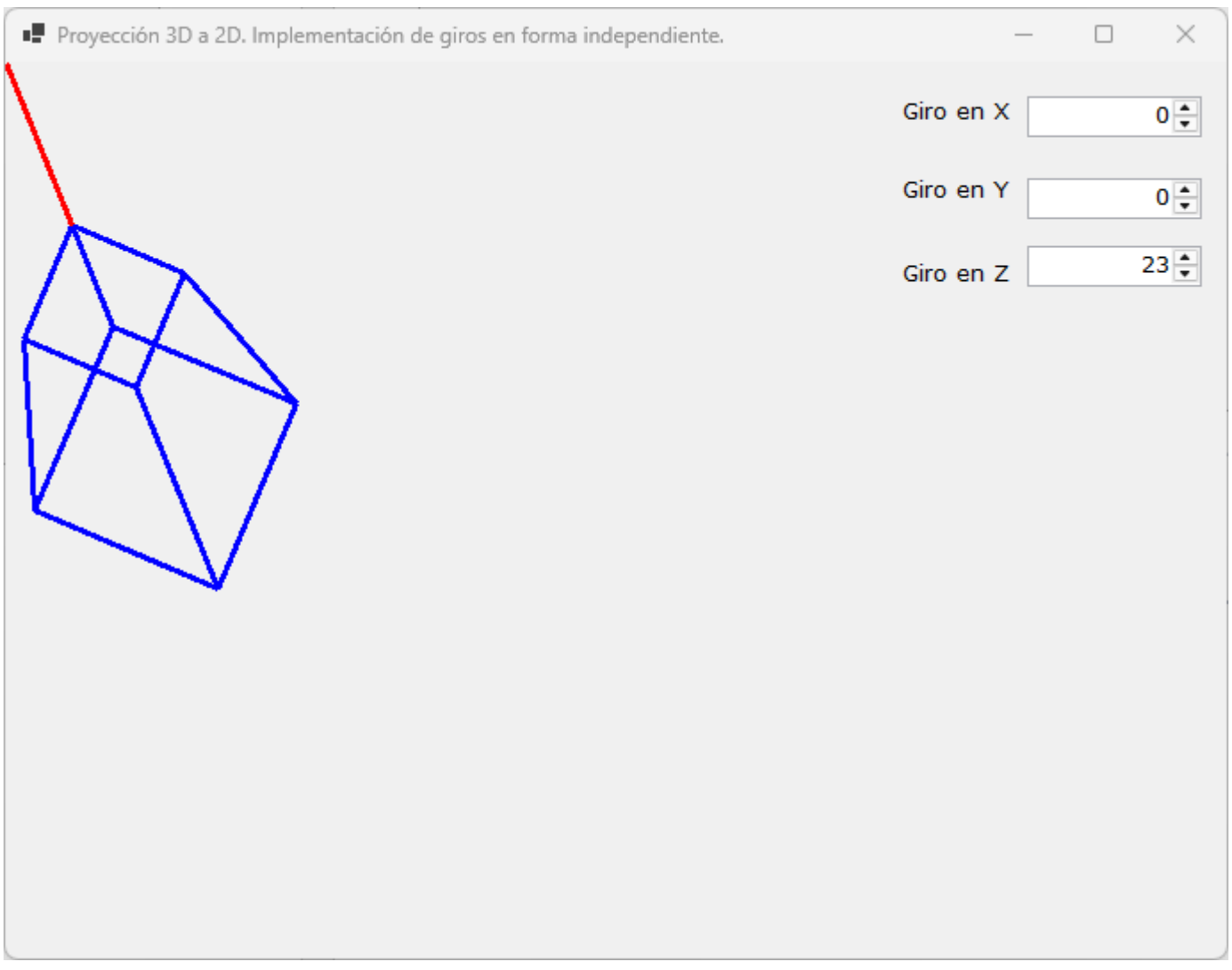


Ilustración 7: Giro en el eje Z de la figura 3D. No hay giro en los otros ejes.

La línea roja representa desde donde se hace el giro, en el punto (0,0,0). Por ese motivo, la figura no solo gira sino que se desplaza.

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.

Pero aparece un problema: al girar el cubo, en ciertos ángulos, su proyección queda por fuera de la ventana. ¿Cómo evitar esto? Habría que calcular para cada punto del cubo, su proyección girando desde 0 a 360 grados, tanto en X como en Y como en Z.

Se sabe que el cubo tiene coordenadas de -0.5 a 0.5 (su lado mide 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 X plano mínimo, Y plano mínimo, X plano máximo, Y plano máximo

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

Desde ángulo en Z = 0 grados hasta 360 grados ➔ Calcule 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 plano máximo. Estos valores son del plano matemático.

M/003.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    public double PlanoX, PlanoY; //Proyección o sombra

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Gira en X
    public 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
    public 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
    public 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);
    }
```

```
    //Aplica la matriz de giro
    private void AplicaMatrizGiro(double[,] Matriz) {
      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

    //Convierte de 3D a 2D (segunda dimensión)
    public void Proyecta(int ZPersona, ref double minimoX, ref double maximoX, ref double minimoY, ref
double maximoY) {
      PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      if (PlanoX < minimoX) minimoX = PlanoX;
      if (PlanoX > maximoX) maximoX = PlanoX;
      if (PlanoY < minimoY) minimoY = PlanoY;
      if (PlanoY > maximoY) maximoY = PlanoY;
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

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

    //Un cubo tiene 12 líneas de conexión
    private List<Conexion> conexiones;

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

      //Las 12 líneas para dibujar el cubo
      //punto inicial ---> punto final
      conexiones = [
        new Conexion(0, 1),
        new Conexion(1, 2),
        new Conexion(2, 3),
        new Conexion(3, 0),
        new Conexion(4, 5),
        new Conexion(5, 6),
        new Conexion(6, 7),
        new Conexion(7, 4),
        new Conexion(0, 4),
        new Conexion(1, 5),
        new Conexion(2, 6),
```

```
    new Conexion(3, 7)
  ];
}

//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;

  //Va de punto en punto
  for (int Cont = 0; Cont < puntos.Count; Cont++) {

    //Va de angulo en angulo
    for (double angulo = 0; angulo <= 360; angulo++) {
      puntos[Cont].GiroX(angulo);
      puntos[Cont].Proyecta(ZPersona, ref minimoX, ref maximoX, ref minimoY, ref maximoY);

      puntos[Cont].GiroY(angulo);
      puntos[Cont].Proyecta(ZPersona, ref minimoX, ref maximoX, ref minimoY, ref maximoY);

      puntos[Cont].GiroZ(angulo);
      puntos[Cont].Proyecta(ZPersona, ref minimoX, ref maximoX, ref minimoY, ref maximoY);
    }
  }

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

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

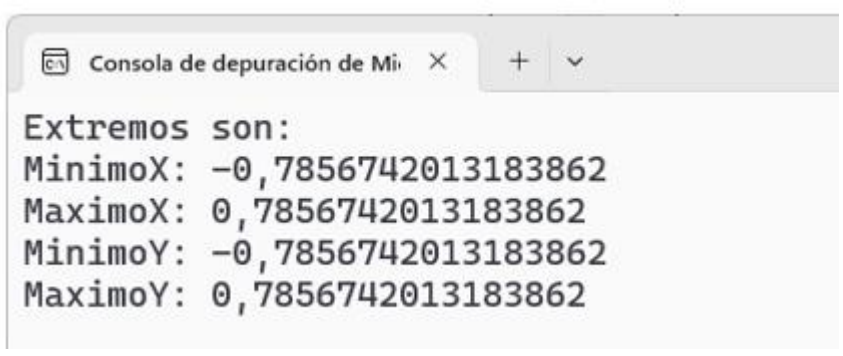


Ilustración 8: Constantes para después calcular los puntos en pantalla

Volviendo a las ecuaciones:

convierteX = (XpantallaFin – XpantallaIni) / (MaximoX - MinimoX)

convierteY = (YpantallaFin – YpantallaIni) / (MaximoY – MinimoY)

Entonces:

convierteX = (XpantallaFin – XpantallaIni) / (0,7856742013183862 – -0,7856742013183862)

convierteY = (YpantallaFin – YpantallaIni) / (0,7856742013183862 – -0,7856742013183862)

Es decir:

convierteX = (XpantallaFin – XpantallaIni) / 1,57134840263677

convierteY = (YpantallaFin – YpantallaIni) / 1,57134840263677

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

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

Nota 2: Esas constantes se usarán así:

```
//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);
```

Giro centrado: Uso de las constantes calculadas previamente

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

M/004.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    private double PlanoX, PlanoY; //Proyección o sombra
    public int Xpantalla, Ypantalla; //Coordenadas en la pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Gira en X
    public void GiroX(double AnguloRad) {
      double[,] Matriz = new double[3, 3] {
        {1, 0, 0},
        {0, Math.Cos(AnguloRad), Math.Sin(AnguloRad)},
        {0, -Math.Sin(AnguloRad), Math.Cos(AnguloRad) }
      };

      AplicaMatrizGiro(Matriz);
    }

    //Gira en Y
    public void GiroY(double AnguloRad) {
      double[,] Matriz = new double[3, 3] {
        {Math.Cos(AnguloRad), 0, -Math.Sin(AnguloRad)},
        {0, 1, 0},
        {Math.Sin(AnguloRad), 0, Math.Cos(AnguloRad) }
      };

      AplicaMatrizGiro(Matriz);
    }

    //Gira en Z
    public void GiroZ(double AnguloRad) {
      double[,] Matriz = new double[3, 3] {
        {Math.Cos(AnguloRad), Math.Sin(AnguloRad), 0},
        {-Math.Sin(AnguloRad), Math.Cos(AnguloRad), 0},
        {0, 0, 1 }
      };

      AplicaMatrizGiro(Matriz);
    }

    //Aplica la matriz de giro
    private void AplicaMatrizGiro(double[,] Matriz) {
      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

    //Convierte de 3D a 2D (segunda dimensión)
    public void Proyecta(int ZPersona) {
      PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);
    }

    //Cuadra en pantalla
    public void CuadraPantalla(double convierteX, double convierteY,
                double minimoX, double minimoY,
                int XpantallaIni, int XpantallaFin,
                int YpantallaIni, int YpantallaFin) {
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX - minimoX) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY - minimoY) + YpantallaIni);
    }
  }
```

```
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

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

    //Un cubo tiene 12 líneas de conexión
    private List<Conexion> conexiones;

    //Valores extremos
    private double minimoX, maximoX, minimoY, maximoY, convierteX, convierteY;

    //Despliegue en pantalla
    private int XpantallaIni, XpantallaFin, YpantallaIni, YpantallaFin;

    //Distancia del observador
    private int ZPersona;

    //Constructor
    public Cubo(int XpantallaIni, int XpantallaFin, int YpantallaIni, int YpantallaFin) {
      //Ejemplo de coordenadas espaciales X,Y,Z
      puntos = [
        new Punto(-0.5, -0.5, -0.5),
        new Punto(0.5, -0.5, -0.5),
        new Punto(0.5, 0.5, -0.5),
        new Punto(-0.5, 0.5, -0.5),
        new Punto(-0.5, -0.5, 0.5),
        new Punto(0.5, -0.5, 0.5),
        new Punto(0.5, 0.5, 0.5),
        new Punto(-0.5, 0.5, 0.5)
      ];

      //Las 12 líneas para dibujar el cubo
      //punto inicial ---> punto final
      conexiones = [
        new Conexion(0, 1),
        new Conexion(1, 2),
        new Conexion(2, 3),
        new Conexion(3, 0),
        new Conexion(4, 5),
        new Conexion(5, 6),
        new Conexion(6, 7),
        new Conexion(7, 4),
        new Conexion(0, 4),
        new Conexion(1, 5),
        new Conexion(2, 6),
        new Conexion(3, 7)
      ];

      this.XpantallaIni = XpantallaIni;
      this.XpantallaFin = XpantallaFin;
      this.YpantallaIni = YpantallaIni;
      this.YpantallaFin = YpantallaFin;
      this.ZPersona = 5;

      //Los valores extremos de las coordenadas del cubo
      this.maximoX = 0.785674201318386;
      this.minimoX = -0.785674201318386;
```

```
    this.maximoY = 0.785674201318386;
    this.minimoY = -0.785674201318386;

    //Las constantes de transformación
    convierteX = (XpantallaFin - XpantallaIni) / (maximoX - minimoX);
    convierteY = (YpantallaFin - YpantallaIni) / (maximoY - minimoY);
  }

  public void GiroX(int Angulo) {
    double Radian = Angulo * Math.PI / 180;
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].GiroX(Radian);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(convierteX, convierteY,
              minimoX, minimoY,
              XpantallaIni, XpantallaFin,
              YpantallaIni, YpantallaFin);
    }
  }

  public void GiroY(int Angulo) {
    double Radian = Angulo * Math.PI / 180;
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].GiroY(Radian);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(convierteX, convierteY,
              minimoX, minimoY,
              XpantallaIni, XpantallaFin,
              YpantallaIni, YpantallaFin);
    }
  }

  public void GiroZ(int Angulo) {
    double Radian = Angulo * Math.PI / 180;
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].GiroZ(Radian);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(convierteX, convierteY,
              minimoX, minimoY,
              XpantallaIni, XpantallaFin,
              YpantallaIni, YpantallaFin);
    }
  }

  //Dibuja el cubo
  public void Dibuja(Graphics lienzo, Pen lapiz) {
    for (int Cont = 0; Cont < conexiones.Count; Cont++) {
      int Inicio = conexiones[Cont].punto1;
      int Final = conexiones[Cont].punto2;
      lienzo.DrawLine(lapiz, puntos[Inicio].Xpantalla, puntos[Inicio].Ypantalla,
                puntos[Final].Xpantalla, puntos[Final].Ypantalla);
    }
  }
 }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D y giros. Figura centrada.
  public partial class Form1 : Form {
    //El cubo que se proyecta y gira
    Cubo Figura3D;

    public Form1() {
      InitializeComponent();

      int XpantallaIni = 30, XpantallaFin = 700, YpantallaIni = 30, YpantallaFin = 700;
      Figura3D = new Cubo(XpantallaIni, XpantallaFin, YpantallaIni, YpantallaFin);
      Figura3D.GiroX(0);
    }

    private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
      numGiroY.Value = 0;
      numGiroZ.Value = 0;
      int AnguloX = Convert.ToInt32(numGiroX.Value);
```

```
      Figura3D.GiroX(AnguloX);
      Refresh();
    }

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

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

    //Pinta la proyección
    private void Form1_Paint(object sender, PaintEventArgs e) {
      Graphics Lienzo = e.Graphics;
      Pen Lapiz = new(Color.Blue, 3);
      Figura3D.Dibuja(Lienzo, Lapiz);
    }
  }
}
```

Ejemplo de ejecución:

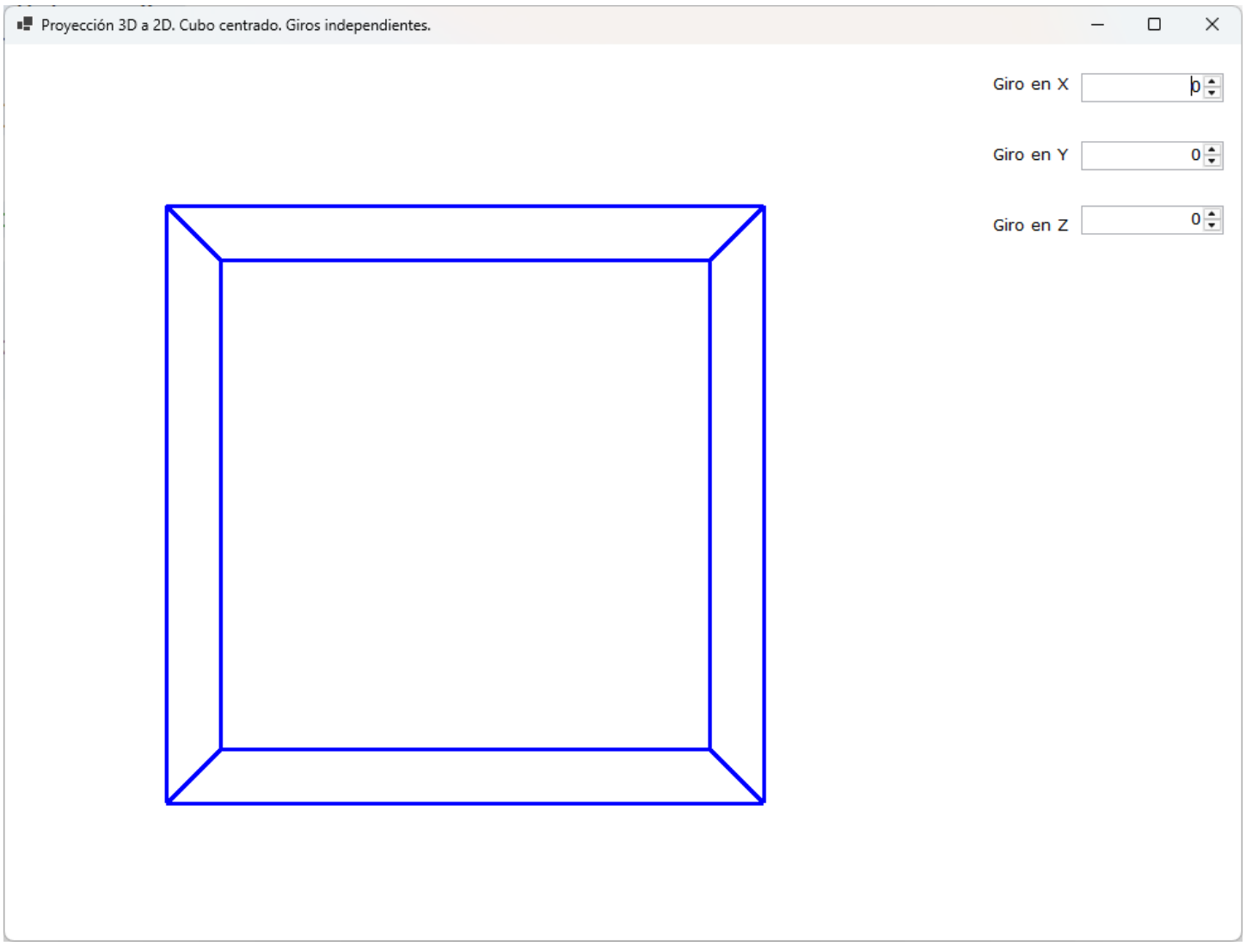


Ilustración 9: Cubo proyectado por defecto

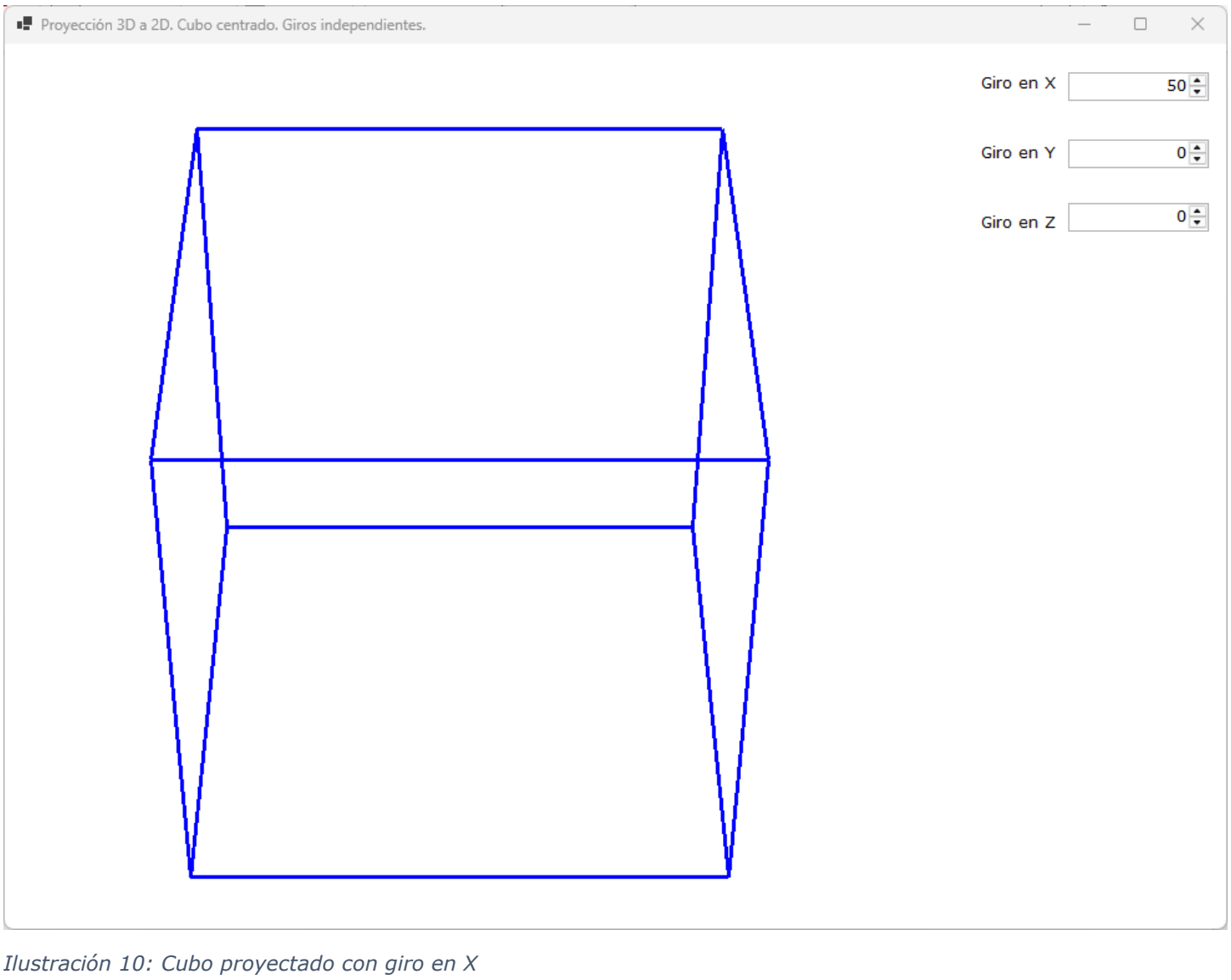


Ilustración 10: Cubo proyectado con giro en X

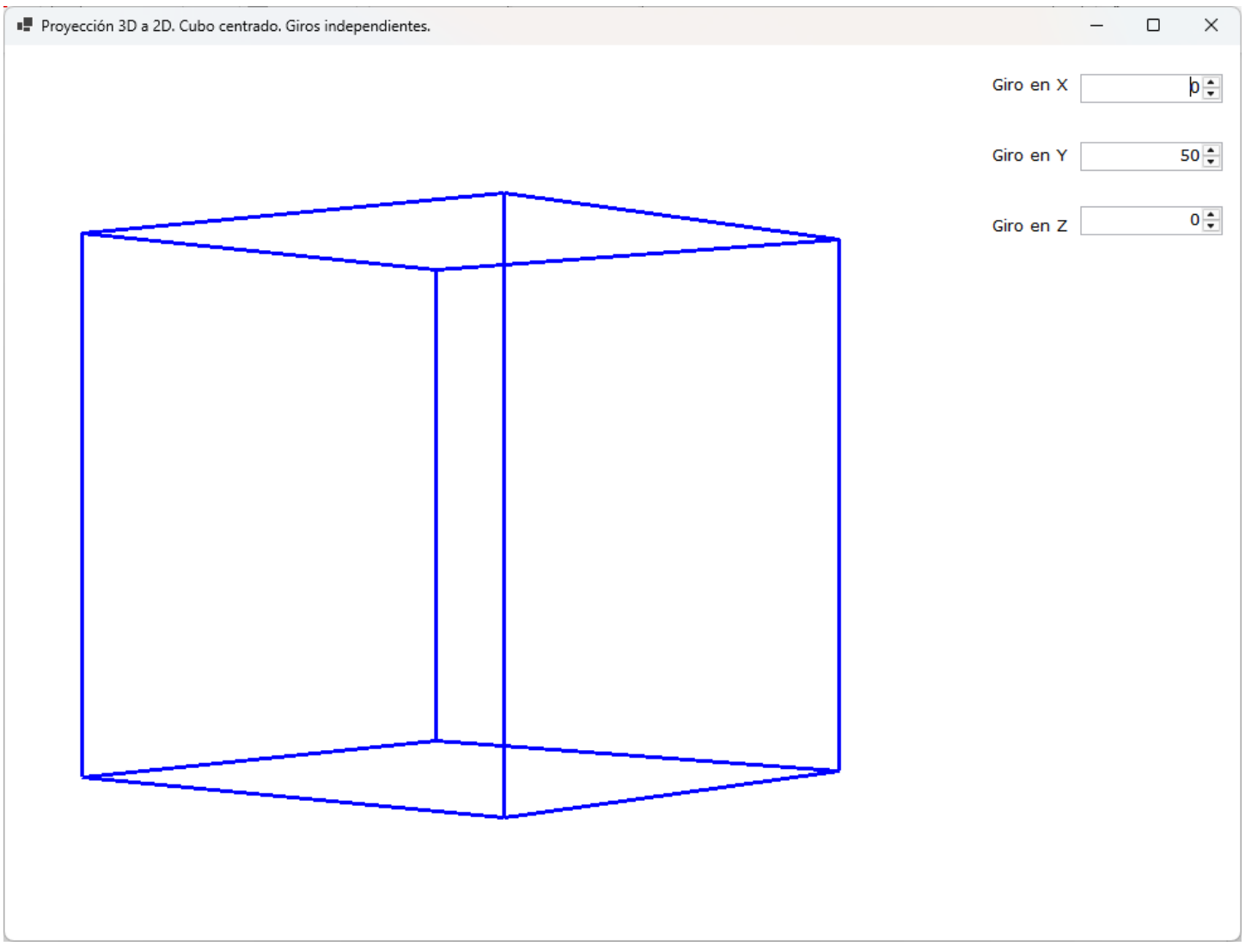


Ilustración 11: Cubo proyectado con giro en Y

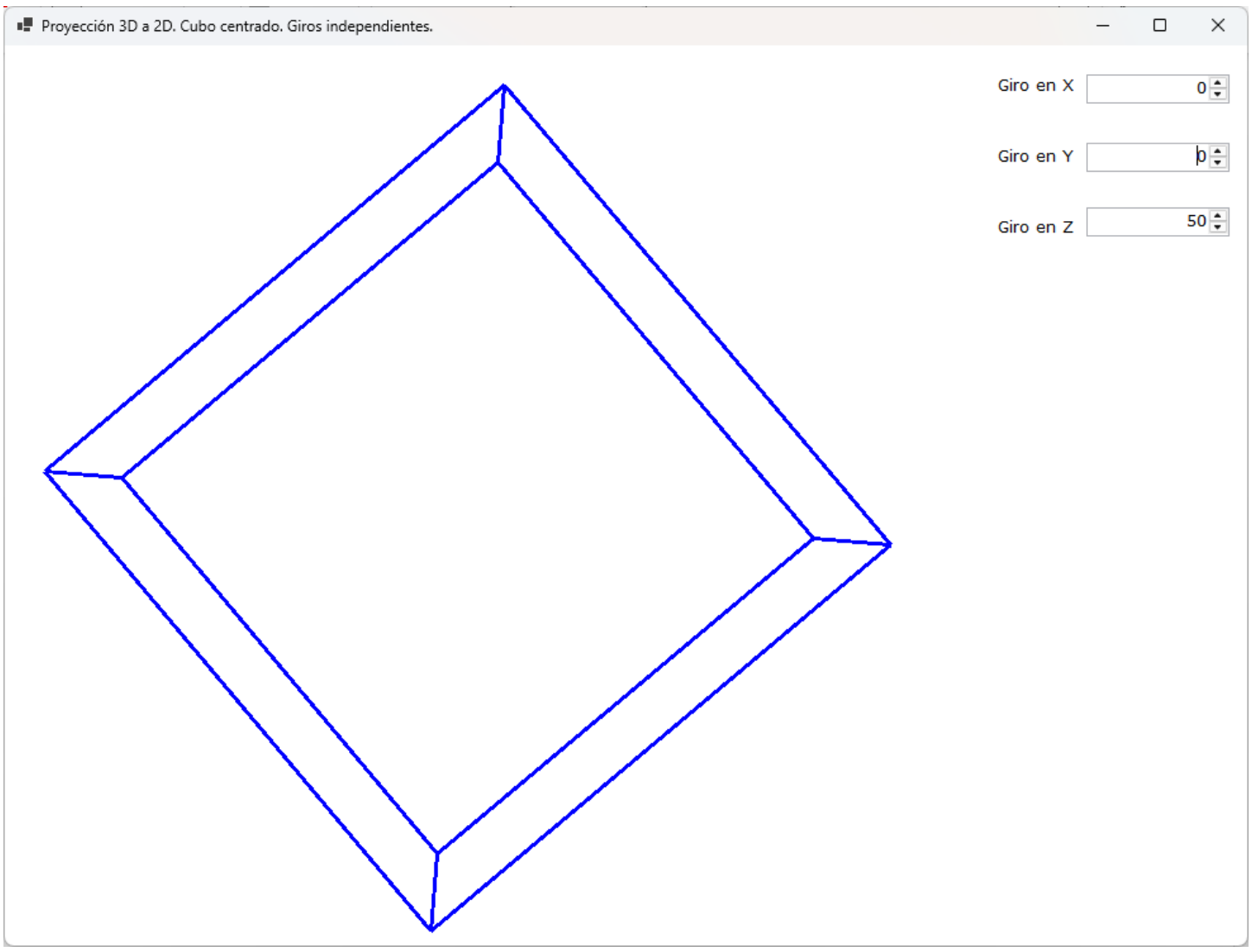


Ilustración 12: Cubo proyectado con giro en Z

Giro centrado: Icosaedro

El programa cambia a usar un icosaedro. Esta figura está dentro de ese cubo de lado = 1.

M/005.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    private double PlanoX, PlanoY; //Proyección o sombra
    public int Xpantalla, Ypantalla; //Coordenadas en la pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Gira en X
    public void GiroX(double AnguloRad) {
      double[,] Matriz = new double[3, 3] {
        {1, 0, 0},
        {0, Math.Cos(AnguloRad), Math.Sin(AnguloRad)},
        {0, -Math.Sin(AnguloRad), Math.Cos(AnguloRad) }
      };

      AplicaMatrizGiro(Matriz);
    }

    //Gira en Y
    public void GiroY(double AnguloRad) {
      double[,] Matriz = new double[3, 3] {
        {Math.Cos(AnguloRad), 0, -Math.Sin(AnguloRad)},
        {0, 1, 0},
        {Math.Sin(AnguloRad), 0, Math.Cos(AnguloRad) }
      };

      AplicaMatrizGiro(Matriz);
    }

    //Gira en Z
    public void GiroZ(double AnguloRad) {
      double[,] Matriz = new double[3, 3] {
        {Math.Cos(AnguloRad), Math.Sin(AnguloRad), 0},
        {-Math.Sin(AnguloRad), Math.Cos(AnguloRad), 0},
        {0, 0, 1 }
      };

      AplicaMatrizGiro(Matriz);
    }

    //Aplica la matriz de giro
    private void AplicaMatrizGiro(double[,] Matriz) {
      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

    //Convierte de 3D a 2D (segunda dimensión)
    public void Proyecta(int ZPersona) {
      PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);
    }

    //Cuadra en pantalla
    public void CuadraPantalla(double convierteX, double convierteY,
                double minimoX, double minimoY,
                int XpantallaIni, int XpantallaFin,
                int YpantallaIni, int YpantallaFin) {
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX - minimoX) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY - minimoY) + YpantallaIni);
    }
  }
```

```
}
```

```
namespace Graficos {
  //Triángulo
  internal class Poligono {
    public int punto1, punto2, punto3;

    public Poligono(int punto1, int punto2, int punto3) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

    //Valores extremos
    private double minimoX, maximoX, minimoY, maximoY, convierteX, convierteY;

    //Despliegue en pantalla
    private int XpantallaIni, XpantallaFin, YpantallaIni, YpantallaFin;

    //Distancia del observador
    private int ZPersona;

    //Constructor
    public Objeto3D(int XpantallaIni, int XpantallaFin, int YpantallaIni, int YpantallaFin) {
      //Ejemplo de coordenadas espaciales X,Y,Z
      puntos = [
        new Punto( 0.00000000000,  0.361803,  0.5),
        new Punto( 0.00000000000,  0.361803, -0.5),
        new Punto( 0.00000000000, -0.361803,  0.5),
        new Punto( 0.00000000000, -0.361803, -0.5),
        new Punto( 0.361803,  0.5,  0.00000000000),
        new Punto( 0.361803, -0.5,  0.00000000000),
        new Punto(-0.361803,  0.5,  0.00000000000),
        new Punto(-0.361803, -0.5,  0.00000000000),
        new Punto( 0.5,  0.00000000000,  0.361803),
        new Punto( 0.5,  0.00000000000, -0.361803),
        new Punto(-0.5,  0.00000000000,  0.361803),
        new Punto(-0.5,  0.00000000000, -0.361803)
       ];

      //Los polígonos que dibujan el icosaedro
      poligonos = [
         new Poligono(0, 6, 4),
         new Poligono(0, 4, 8),
         new Poligono(0, 8, 2),
         new Poligono(0, 2, 10),
         new Poligono(0, 10, 6),
         new Poligono(1, 9, 4),
         new Poligono(1, 4, 6),
         new Poligono(1, 6, 11),
         new Poligono(1, 11, 3),
         new Poligono(1, 3, 9),
         new Poligono(2, 8, 5),
         new Poligono(2, 5, 7),
         new Poligono(2, 7, 10),
         new Poligono(3, 11, 7),
         new Poligono(3, 7, 5),
         new Poligono(3, 5, 9),
         new Poligono(6, 10, 11),
         new Poligono(5, 8, 9),
```

```
      new Poligono(7, 11, 10),
      new Poligono(9, 5, 8)
    ];

    this.XpantallaIni = XpantallaIni;
    this.XpantallaFin = XpantallaFin;
    this.YpantallaIni = YpantallaIni;
    this.YpantallaFin = YpantallaFin;
    this.ZPersona = 5;

    //Los valores extremos
    this.maximoX = 0.785674201318386;
    this.minimoX = -0.785674201318386;
    this.maximoY = 0.785674201318386;
    this.minimoY = -0.785674201318386;

    //Las constantes de transformación
    convierteX = (XpantallaFin - XpantallaIni) / (maximoX - minimoX);
    convierteY = (YpantallaFin - YpantallaIni) / (maximoY - minimoY);
  }

  public void GiroX(int Angulo) {
    double Radian = Angulo * Math.PI / 180;
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].GiroX(Radian);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(convierteX, convierteY,
              minimoX, minimoY,
              XpantallaIni, XpantallaFin,
              YpantallaIni, YpantallaFin);
    }
  }

  public void GiroY(int Angulo) {
    double Radian = Angulo * Math.PI / 180;
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].GiroY(Radian);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(convierteX, convierteY,
              minimoX, minimoY,
              XpantallaIni, XpantallaFin,
              YpantallaIni, YpantallaFin);
    }
  }

  public void GiroZ(int Angulo) {
    double Radian = Angulo * Math.PI / 180;
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].GiroZ(Radian);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(convierteX, convierteY,
              minimoX, minimoY,
              XpantallaIni, XpantallaFin,
              YpantallaIni, YpantallaFin);
    }
  }

  //Dibuja el icosaedro
  public void Dibuja(Graphics lienzo, Pen lapiz) {
    for (int Cont = 0; Cont < poligonos.Count; Cont++) {
      int puntoA = poligonos[Cont].punto1;
      int puntoB = poligonos[Cont].punto2;
      int puntoC = poligonos[Cont].punto3;
      lienzo.DrawLine(lapiz, puntos[puntoA].Xpantalla, puntos[puntoA].Ypantalla,
              puntos[puntoB].Xpantalla, puntos[puntoB].Ypantalla);
      lienzo.DrawLine(lapiz, puntos[puntoA].Xpantalla, puntos[puntoA].Ypantalla,
              puntos[puntoC].Xpantalla, puntos[puntoC].Ypantalla);
      lienzo.DrawLine(lapiz, puntos[puntoC].Xpantalla, puntos[puntoC].Ypantalla,
              puntos[puntoB].Xpantalla, puntos[puntoB].Ypantalla);
    }
  }
}
```

```
namespace Graficos {
```

```
//Proyección 3D a 2D y giros. Figura centrada.
public partial class Form1 : Form {
  //El icosaedro que se proyecta y gira
  Objeto3D Figura3D;

  public Form1() {
    InitializeComponent();

    int XpantallaIni = 0, XpantallaFin = 700, YpantallaIni = 0, YpantallaFin = 700;
    Figura3D = new Objeto3D(XpantallaIni, XpantallaFin, YpantallaIni, YpantallaFin);
    Figura3D.GiroX(0);
  }

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

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

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

   //Pinta la proyección
   private void Form1_Paint(object sender, PaintEventArgs e) {
    Graphics Lienzo = e.Graphics;
    Pen Lapiz = new(Color.Blue, 3);
    Figura3D.Dibuja(Lienzo, Lapiz);
   }
  }
}
```

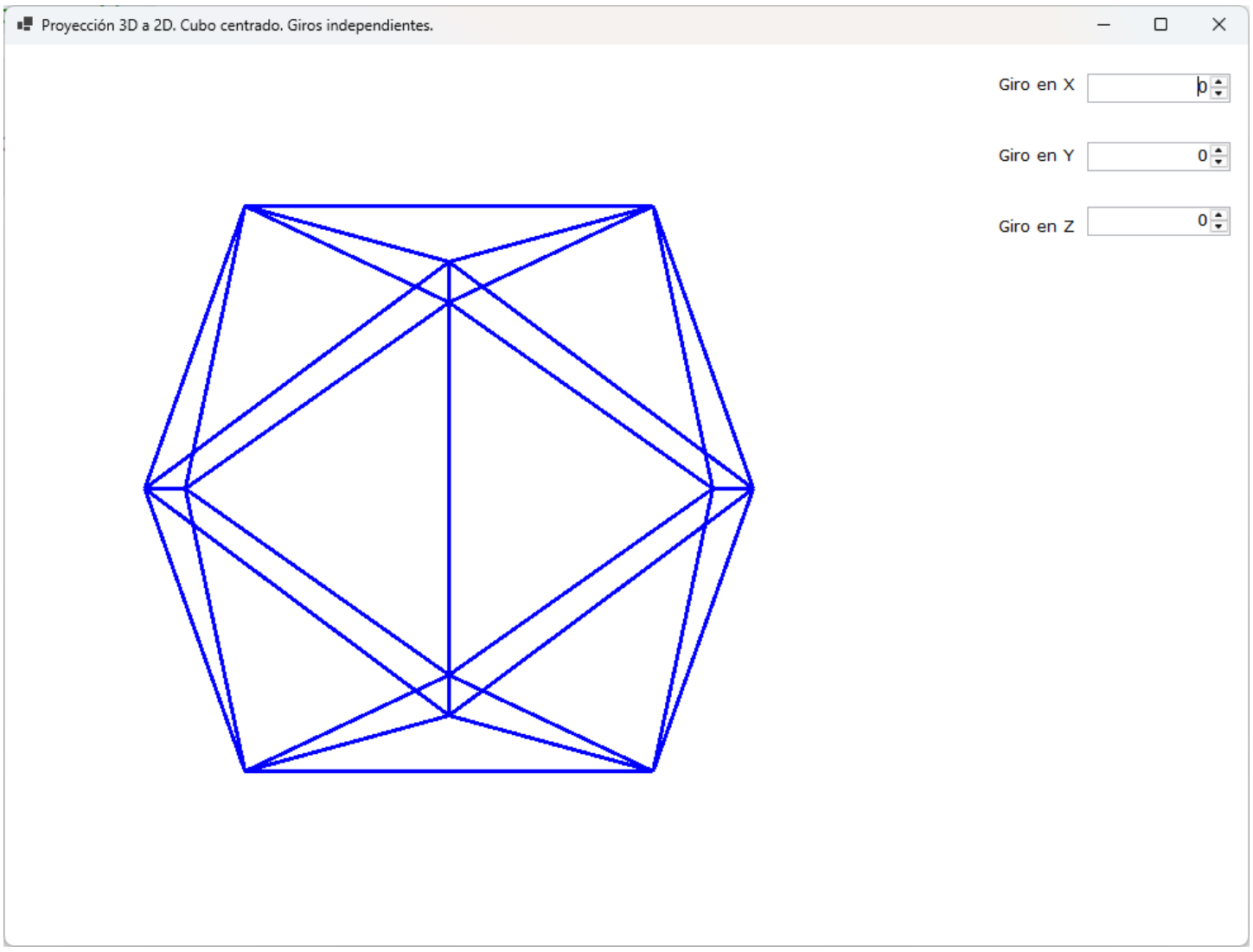


Ilustración 13: Icosaedro por defecto

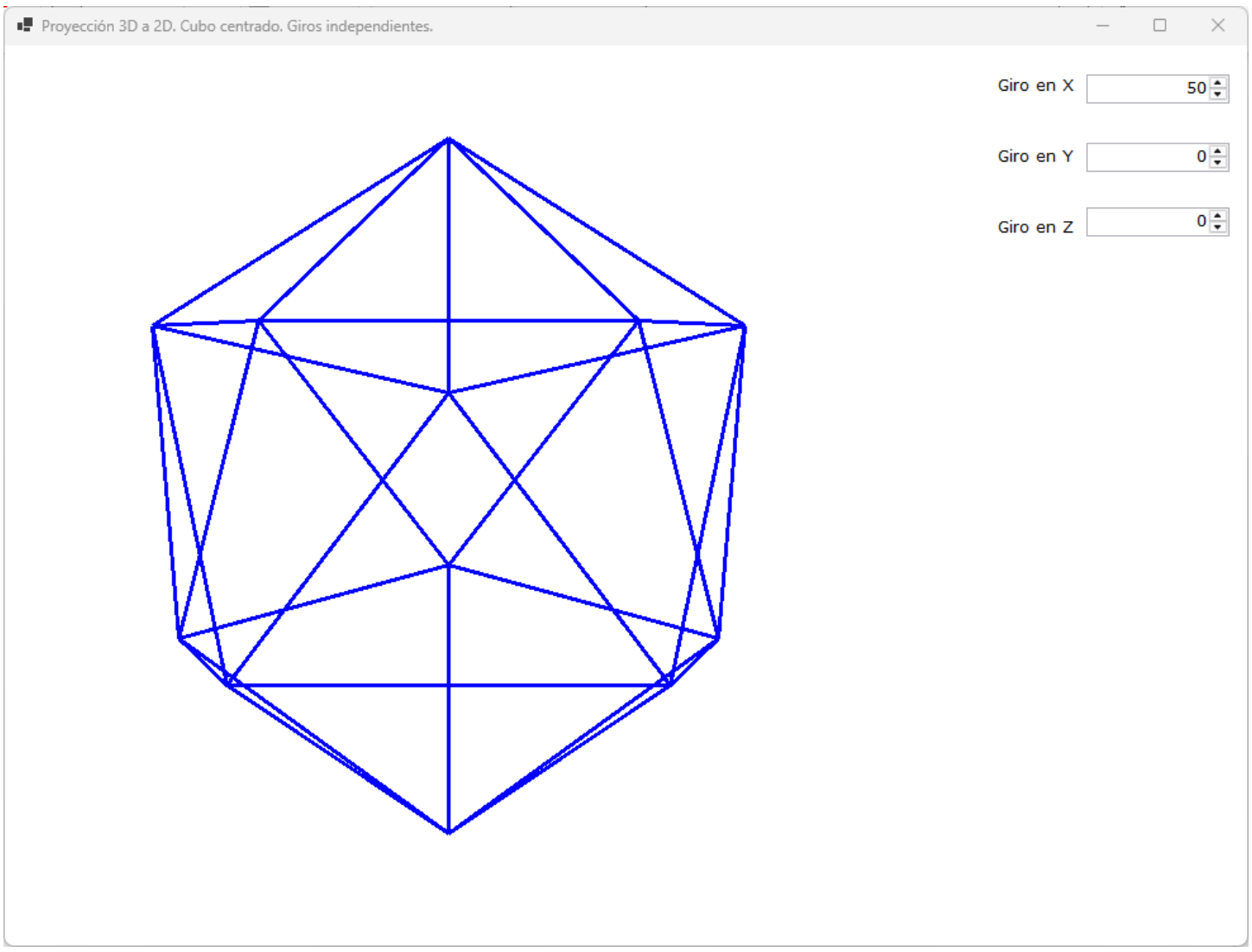


Ilustración 14: Giro en X

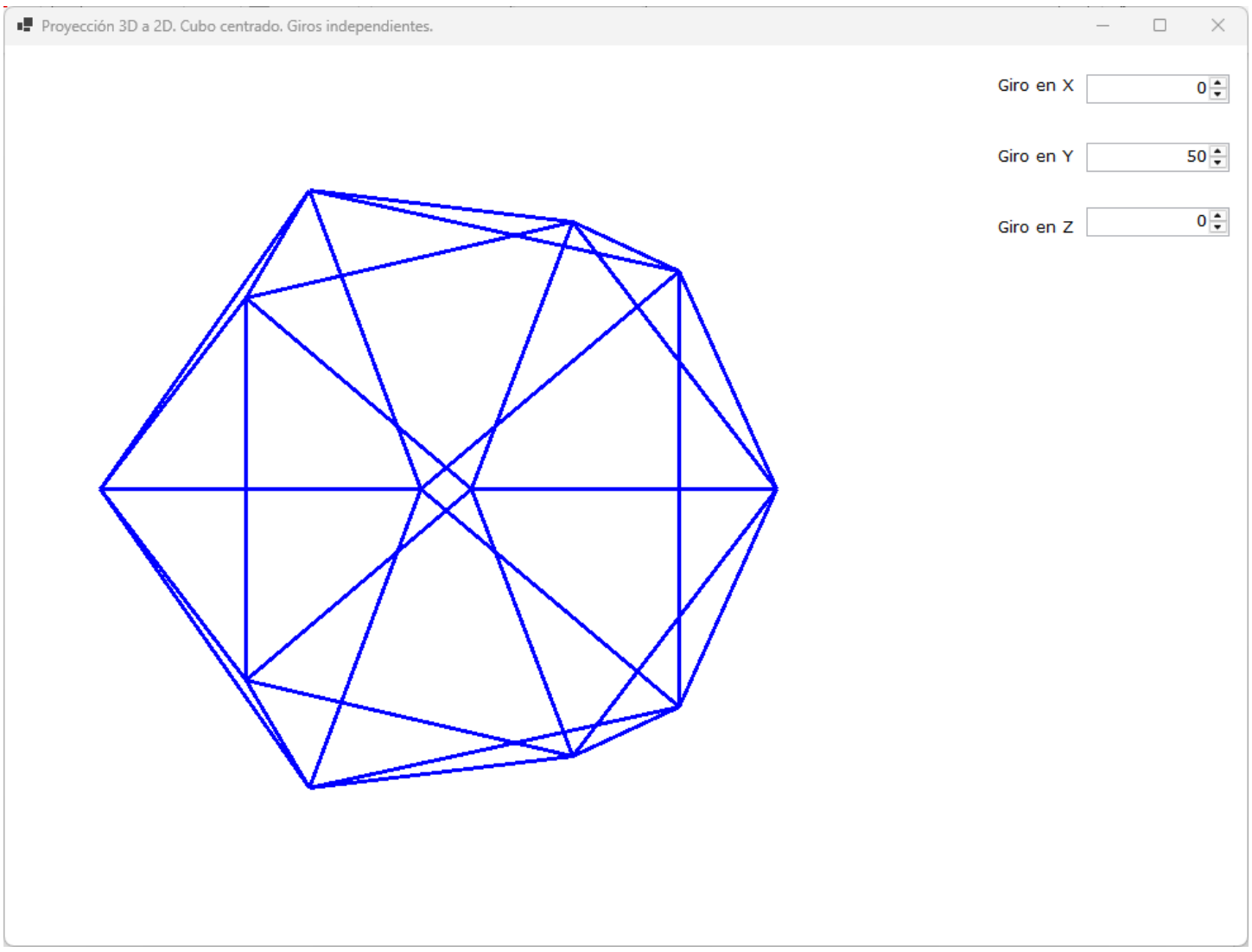


Ilustración 15: Giro en Y

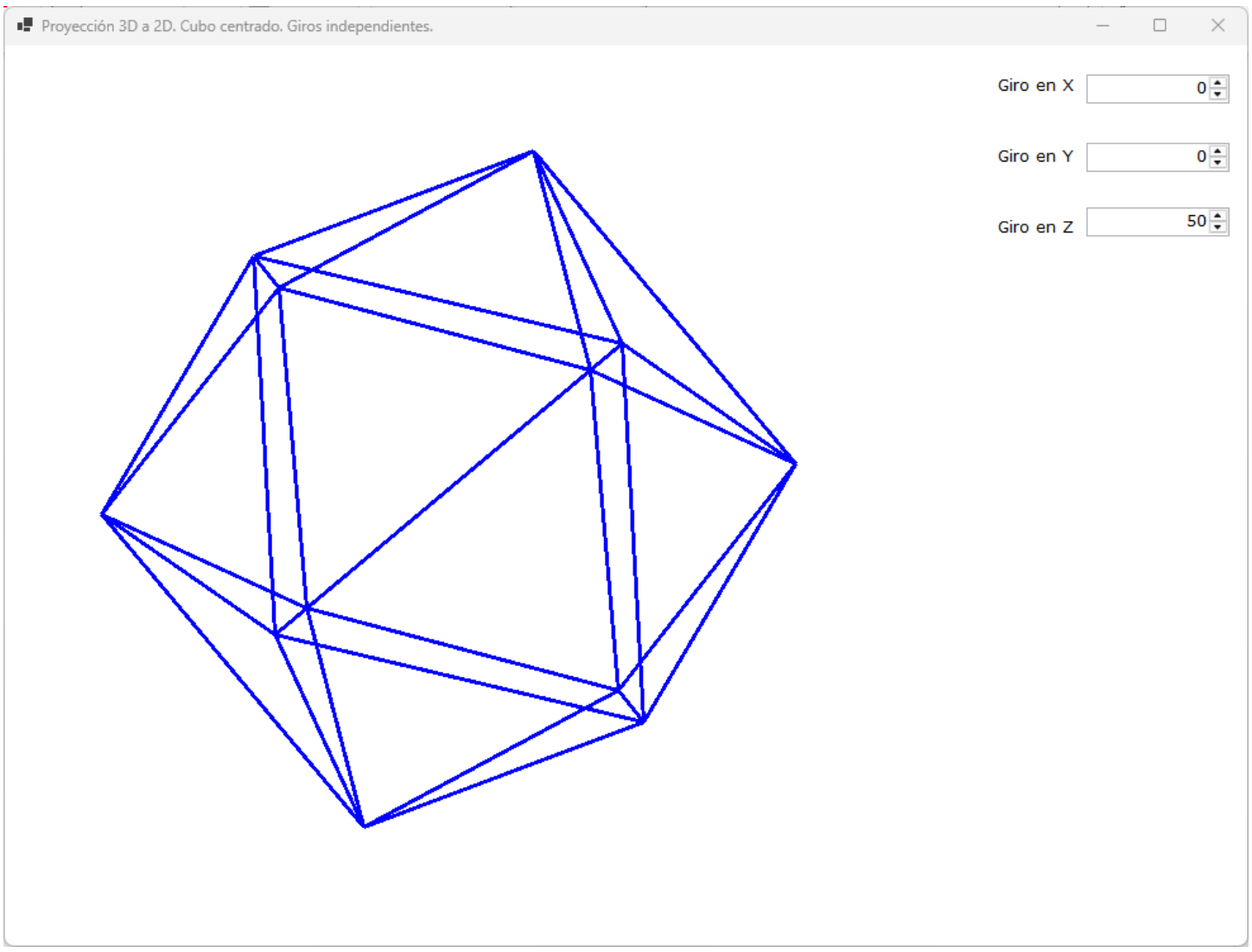


Ilustración 16: Giro en Z

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.

$$CosX = \cos(angX)$$
$$SenX = sen(angX)$$
$$CosY = \cos(angY)$$
$$SenY = sen(angY)$$
$$CosZ = \cos(angZ)$$
$$SenZ = sen(angZ)$$

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = \begin{bmatrix} \text{CosY} * \text{CosZ} & -\text{CosX} * \text{SinZ} + \text{SinX} * \text{SinY} * \text{CosZ} & \text{SinX} * \text{SinZ} + \text{CosX} * \text{SinY} * \text{CosZ} \\ \text{CosY} * \text{SinZ} & \text{CosX} * \text{CosZ} + \text{SinX} * \text{SinY} * \text{SinZ} & -\text{SinX} * \text{CosZ} + \text{CosX} * \text{SinY} * \text{SinZ} \\ -\text{SinY} & \text{SinX} * \text{CosY} & \text{CosX} * \text{CosY} \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

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. Pero aquí se tiene un problema: como se pueden combinar los tres giros, hay que recalcular de nuevo las constantes. 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*361*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.

M/006.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    public double PlanoX, PlanoY; //Proyección o sombra

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    public void Giro(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 }
      };
```

```
      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

    //Convierte de 3D a 2D (segunda dimensión)
    public void Proyecta(int ZPersona, ref double minimoX, ref double maximoX, ref double minimoY, ref
double maximoY) {
      PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      if (PlanoX < minimoX) minimoX = PlanoX;
      if (PlanoX > maximoX) maximoX = PlanoX;
      if (PlanoY < minimoY) minimoY = PlanoY;
      if (PlanoY > maximoY) maximoY = PlanoY;
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

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

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

    //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;

      //Va de punto en punto
      for (int Cont = 0; Cont < puntos.Count; Cont++) {

        //Va de angulo en angulo
        for (double angX = 0; angX <= 360; angX++)
          for (double angY = 0; angY <= 360; angY++)
            for (double angZ = 0; angZ <= 360; angZ++) {
              puntos[Cont].Giro(angX, angY, angZ);
              puntos[Cont].Proyecta(ZPersona, ref minimoX, ref maximoX, ref minimoY, ref maximoY);
            }
      }
```

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

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

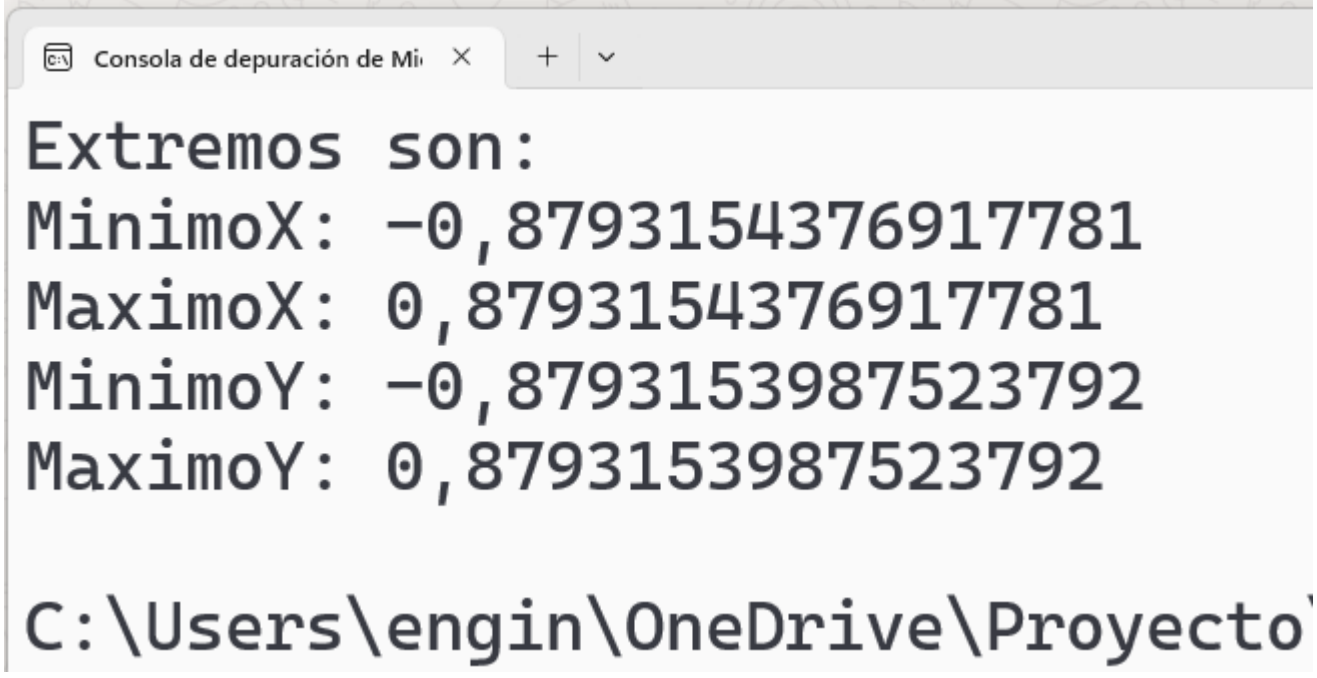


Ilustración 17: Constantes para cuadrar el cubo en la pantalla

Combinando los tres giros: Uso de constantes

M/007.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    private double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    public double PlanoX, PlanoY; //Proyección o sombra
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Giro en los tres ejes
    public void Giro(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 }
      };

      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

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

    //Cuadra en pantalla
    public void CuadraPantalla(int XpantallaIni, int YpantallaIni, int XpantallaFin, int YpantallaFin) {
      //Los valores extremos de las coordenadas del cubo
      double maximoX = 0.879315437691778;
      double minimoX = -0.879315437691778;
      double maximoY = 0.879315398752379;
      double minimoY = -0.879315437691778;

      //Las constantes de transformación
      double convierteX = (XpantallaFin - XpantallaIni) / (maximoX - minimoX);
      double convierteY = (YpantallaFin - YpantallaIni) / (maximoY - minimoY);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX - minimoX) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY - minimoY) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

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

    //Un cubo tiene 12 líneas de conexión
    private List<Conexion> conexiones;

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

      //Las 12 líneas para dibujar el cubo
      //punto inicial ---> punto final
      conexiones = [
        new Conexion(0, 1),
        new Conexion(1, 2),
        new Conexion(2, 3),
        new Conexion(3, 0),
        new Conexion(4, 5),
        new Conexion(5, 6),
        new Conexion(6, 7),
        new Conexion(7, 4),
        new Conexion(0, 4),
        new Conexion(1, 5),
        new Conexion(2, 6),
        new Conexion(3, 7)
      ];
    }

    public void GirarFigura(double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int
YpantallaIni, int XpantallaFin, int YpantallaFin) {
      //Gira los 8 puntos
      for (int cont = 0; cont < puntos.Count; cont++) {
        puntos[cont].Giro(angX, angY, angZ);
        puntos[cont].Proyecta(ZPersona);
        puntos[cont].CuadraPantalla(XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin);
      }
    }

    //Dibuja el cubo
    public void Dibuja(Graphics lienzo, Pen lapiz) {
      for (int Cont = 0; Cont < conexiones.Count; Cont++) {
        int Inicio = conexiones[Cont].punto1;
        int Final = conexiones[Cont].punto2;
        lienzo.DrawLine(lapiz, puntos[Inicio].Xpantalla, puntos[Inicio].Ypantalla,
                  puntos[Final].Xpantalla, puntos[Final].Ypantalla);
      }
    }
  }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D y giros. Figura centrada.
  public partial class Form1 : Form {

    //El cubo que se proyecta y gira
    Cubo Figura3D;

    //Tamaño de pantalla
    double ZPersona;
    int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

    public Form1() {
      InitializeComponent();

      //Distancia del observador
      ZPersona = 5;

      //Tamaño de pantalla
      XpantallaIni = 20;
      YpantallaIni = 20;
      XpantallaFin = 500;
      YpantallaFin = 500;

      Figura3D = new Cubo();

      //Gira, proyecta y cuadra cada punto de la figura 3D
      Figura3D.GirarFigura(0, 0, 0, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin);
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {
      double AnguloX = Convert.ToDouble(numGiroX.Value);
      double AnguloY = Convert.ToDouble(numGiroY.Value);
      double AnguloZ = Convert.ToDouble(numGiroZ.Value);

      Graphics Lienzo = e.Graphics;

      //Dibuja el rectángulo que contiene la figura 3D
      Pen LapizMarco = new(Color.Blue, 1);
      Lienzo.DrawRectangle(LapizMarco, XpantallaIni, YpantallaIni, XpantallaFin - XpantallaIni, YpantallaFin
- YpantallaIni);

      //Gira, proyecta y cuadra cada punto de la figura 3D
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);

      //Dibuja la figura 3D
      Pen Lapiz3D = new(Color.Black, 1);
      Figura3D.Dibuja(Lienzo, Lapiz3D);
    }

    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();
    }
  }
}
```

Ejemplo de ejecución:

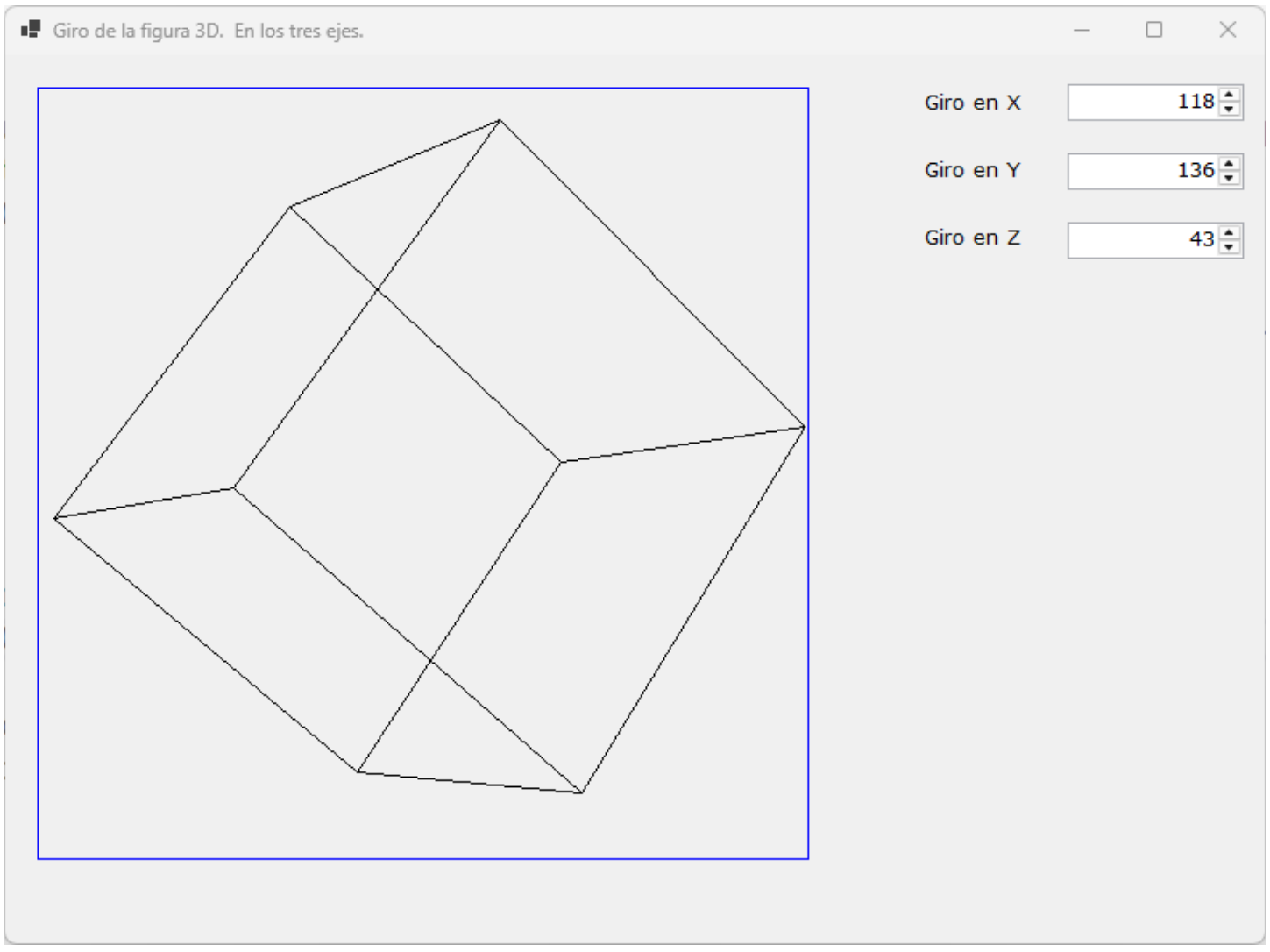


Ilustración 18: Cubo proyectado con giros

Al interior de un cubo

Las figuras 3D se dibujan al interior de un cubo “imaginario” de lado con valor uno(1) de esa manera los giros ya están precalculados.

M/008.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    public double X, Y, Z; //Coordenadas originales
    private double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    public double PlanoX, PlanoY; //Proyección o sombra
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Giro en los tres ejes
    public void Giro(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 }
      };

      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

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

    //Cuadra en pantalla
    public void CuadraPantalla(int XpantallaIni, int YpantallaIni, int XpantallaFin, int YpantallaFin) {
      //Los valores extremos de las coordenadas del cubo
      double maximoX = 0.879315437691778;
      double minimoX = -0.879315437691778;
      double maximoY = 0.879315398752379;
      double minimoY = -0.879315437691778;

      //Las constantes de transformación
      double convierteX = (XpantallaFin - XpantallaIni) / (maximoX - minimoX);
      double convierteY = (YpantallaFin - YpantallaIni) / (maximoY - minimoY);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX - minimoX) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY - minimoY) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
```

```
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Triángulo
  internal class Poligono {
    public int punto1, punto2, punto3;

    public Poligono(int punto1, int punto2, int punto3) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

    //Un cubo tiene 12 líneas de conexión
    private List<Conexion> conexiones;

    //Constructor
    public Objeto3D() {
      //Ejemplo de coordenadas espaciales X,Y,Z
      puntos = [
        new Punto( 0.00000000000,  0.361803,  0.5),
        new Punto( 0.00000000000,  0.361803, -0.5),
        new Punto( 0.00000000000, -0.361803,  0.5),
        new Punto( 0.00000000000, -0.361803, -0.5),
        new Punto( 0.361803,  0.5,  0.00000000000),
        new Punto( 0.361803, -0.5,  0.00000000000),
        new Punto(-0.361803,  0.5,  0.00000000000),
        new Punto(-0.361803, -0.5,  0.00000000000),
        new Punto( 0.5,  0.00000000000,  0.361803),
        new Punto( 0.5,  0.00000000000, -0.361803),
        new Punto(-0.5,  0.00000000000,  0.361803),
        new Punto(-0.5,  0.00000000000, -0.361803),

        //Coordenadas del cubo que contiene al icosaedro
        new Punto(-0.5, -0.5, -0.5),
        new Punto(0.5, -0.5, -0.5),
        new Punto(0.5, 0.5, -0.5),
        new Punto(-0.5, 0.5, -0.5),
        new Punto(-0.5, -0.5, 0.5),
        new Punto(0.5, -0.5, 0.5),
        new Punto(0.5, 0.5, 0.5),
        new Punto(-0.5, 0.5, 0.5)
       ];


    //Los polígonos que dibujan el icosaedro
    poligonos = [
     new Poligono(0, 6, 4),
      new Poligono(0, 4, 8),
      new Poligono(0, 8, 2),
      new Poligono(0, 2, 10),
      new Poligono(0, 10, 6),
      new Poligono(1, 9, 4),
```

```
      new Poligono(1, 4, 6),
      new Poligono(1, 6, 11),
      new Poligono(1, 11, 3),
      new Poligono(1, 3, 9),
      new Poligono(2, 8, 5),
      new Poligono(2, 5, 7),
      new Poligono(2, 7, 10),
      new Poligono(3, 11, 7),
      new Poligono(3, 7, 5),
      new Poligono(3, 5, 9),
      new Poligono(6, 10, 11),
      new Poligono(5, 8, 9),
      new Poligono(7, 11, 10),
      new Poligono(4, 9, 8)
    ];

    //Las 12 líneas para dibujar el cubo
    //punto inicial ---> punto final
    conexiones = [
      new Conexion(0+12, 1+12),
      new Conexion(1+12, 2+12),
      new Conexion(2+12, 3+12),
      new Conexion(3+12, 0+12),
      new Conexion(4+12, 5+12),
      new Conexion(5+12, 6+12),
      new Conexion(6+12, 7+12),
      new Conexion(7+12, 4+12),
      new Conexion(0+12, 4+12),
      new Conexion(1+12, 5+12),
      new Conexion(2+12, 6+12),
      new Conexion(3+12, 7+12)
    ];
  }

  public void GirarFigura(double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int
YpantallaIni, int XpantallaFin, int YpantallaFin) {
    //Gira los 8 puntos
    for (int cont = 0; cont < puntos.Count; cont++) {
      puntos[cont].Giro(angX, angY, angZ);
      puntos[cont].Proyecta(ZPersona);
      puntos[cont].CuadraPantalla(XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin);
    }
  }

  //Dibuja el icosaedro
  public void Dibuja(Graphics lienzo, Pen lapizIcosaedro, Pen lapizCubo) {
    for (int Cont = 0; Cont < poligonos.Count; Cont++) {
      int puntoA = poligonos[Cont].punto1;
      int puntoB = poligonos[Cont].punto2;
      int puntoC = poligonos[Cont].punto3;
      lienzo.DrawLine(lapizIcosaedro, puntos[puntoA].Xpantalla, puntos[puntoA].Ypantalla,
                puntos[puntoB].Xpantalla, puntos[puntoB].Ypantalla);
      lienzo.DrawLine(lapizIcosaedro, puntos[puntoA].Xpantalla, puntos[puntoA].Ypantalla,
                puntos[puntoC].Xpantalla, puntos[puntoC].Ypantalla);
      lienzo.DrawLine(lapizIcosaedro, puntos[puntoC].Xpantalla, puntos[puntoC].Ypantalla,
                puntos[puntoB].Xpantalla, puntos[puntoB].Ypantalla);
    }

    for (int Cont = 0; Cont < conexiones.Count; Cont++) {
      int Inicio = conexiones[Cont].punto1;
      int Final = conexiones[Cont].punto2;
      lienzo.DrawLine(lapizCubo, puntos[Inicio].Xpantalla, puntos[Inicio].Ypantalla,
                puntos[Final].Xpantalla, puntos[Final].Ypantalla);
    }
  }
 }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D y giros. Figura centrada.
  public partial class Form1 : Form {

    //El icosaedro que se proyecta y gira
    Objeto3D Figura3D;

    //Tamaño de pantalla
    double ZPersona;
    int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

    public Form1() {
      InitializeComponent();

      //Distancia del observador
      ZPersona = 5;

      //Tamaño de pantalla
      XpantallaIni = 20;
      YpantallaIni = 20;
      XpantallaFin = 500;
      YpantallaFin = 500;

      Figura3D = new Objeto3D();

      //Gira, proyecta y cuadra cada punto de la figura 3D
      Figura3D.GirarFigura(0, 0, 0, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin);
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {
      double AnguloX = Convert.ToDouble(numGiroX.Value);
      double AnguloY = Convert.ToDouble(numGiroY.Value);
      double AnguloZ = Convert.ToDouble(numGiroZ.Value);

      Graphics Lienzo = e.Graphics;

      //Dibuja el rectángulo que contiene la figura 3D
      Pen LapizMarco = new(Color.Blue, 1);
      Lienzo.DrawRectangle(LapizMarco, XpantallaIni, YpantallaIni, XpantallaFin - XpantallaIni, YpantallaFin
- YpantallaIni);

      //Gira, proyecta y cuadra cada punto de la figura 3D
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);

      //Dibuja la figura 3D
      Pen LapizIcosaedro = new(Color.Black, 1);
      Pen LapizCubo = new(Color.Red, 1);
      Figura3D.Dibuja(Lienzo, LapizIcosaedro, LapizCubo);
    }

    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();
    }
  }
}
```

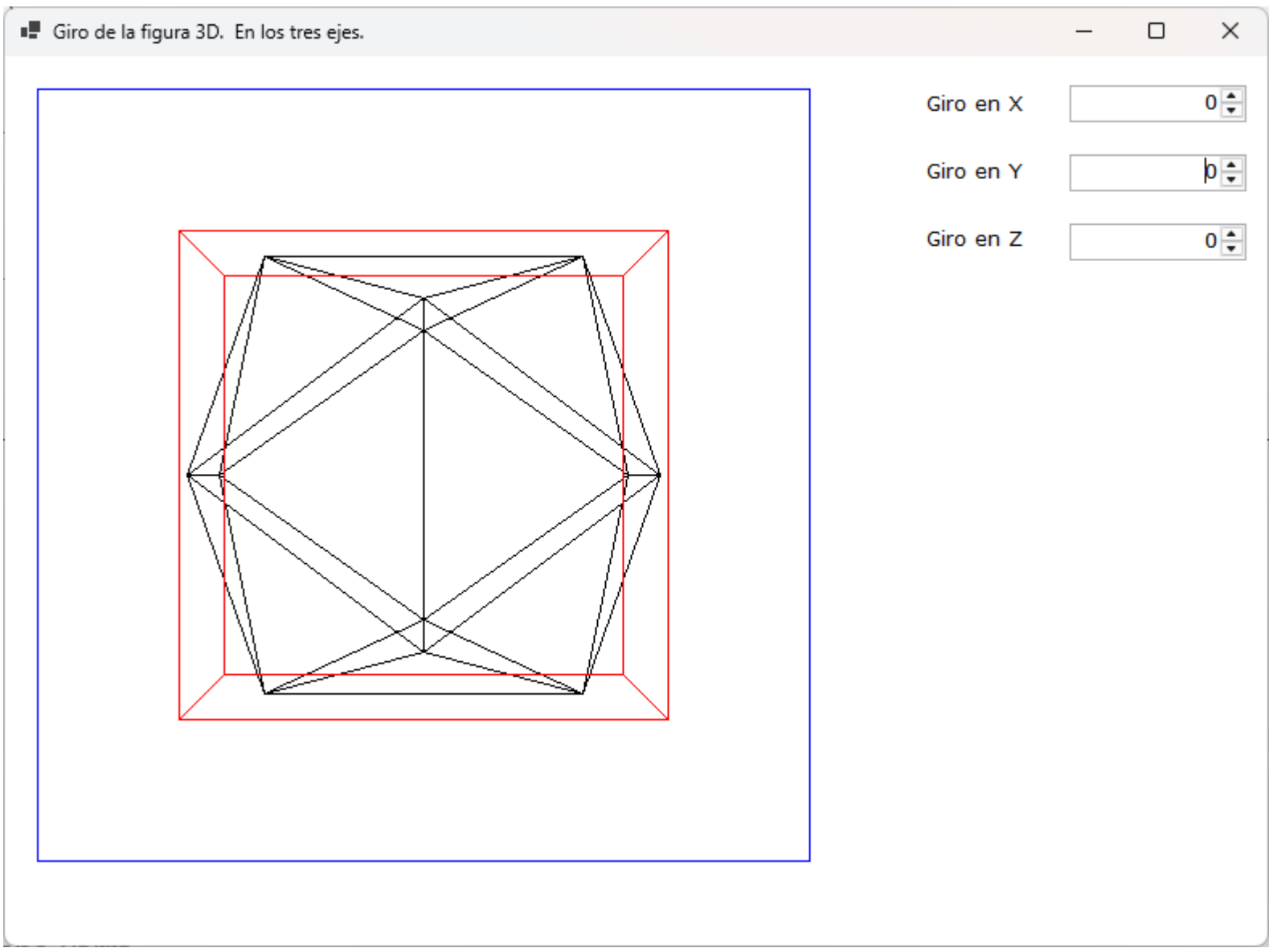


Ilustración 19: El icosaedro está dentro del cubo

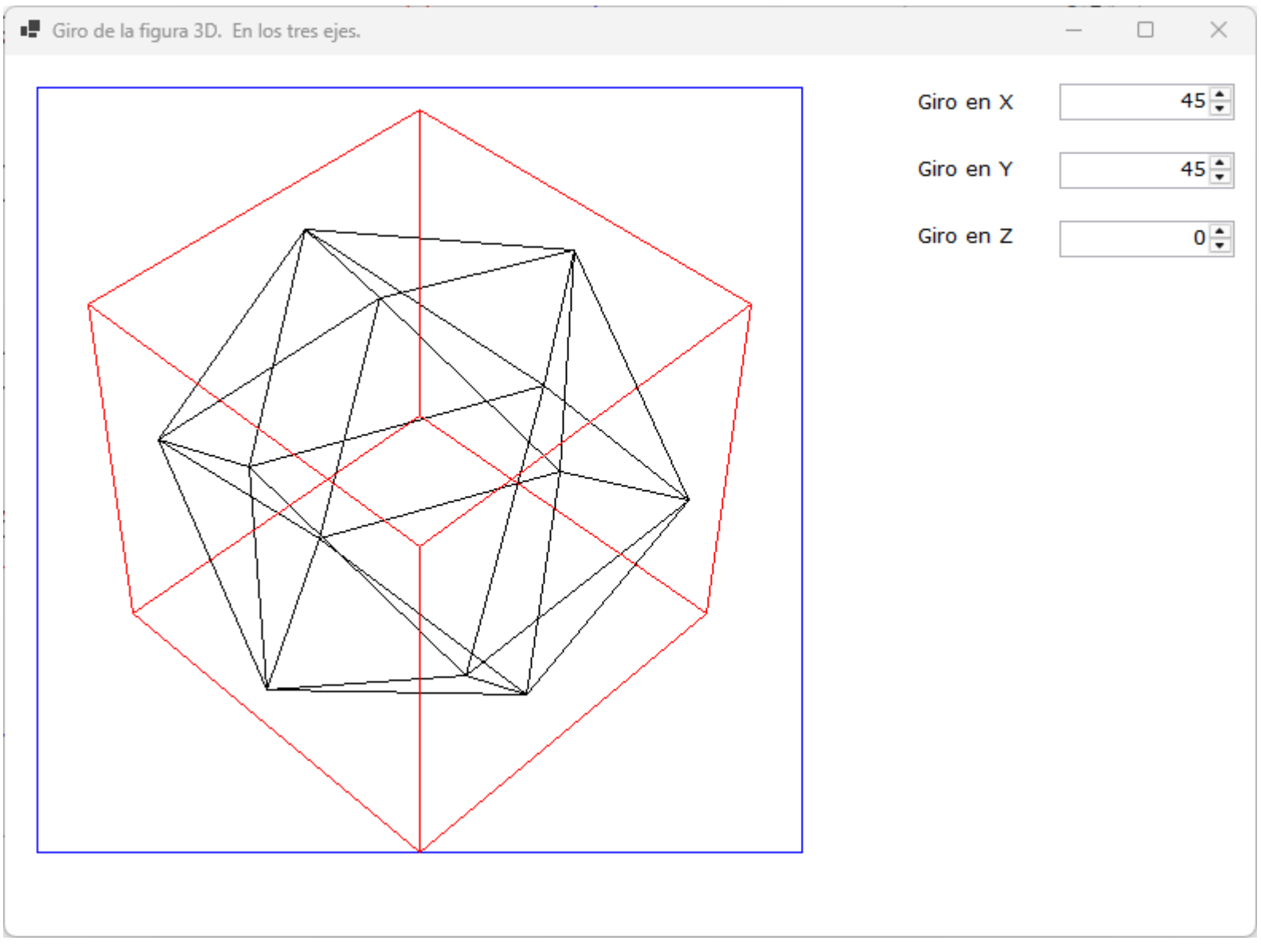


Ilustración 20: Se gira el cubo, entonces gira el icosaedro. Todo pre-calculado.

Líneas ocultas

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

1. Escribir las tres coordenadas XYZ que servirán para componer cada uno de los veinte 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 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/009.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    public double X, Y, Z; //Coordenadas originales
    public double Xgiro, Ygiro, Zgiro; //Al girar los puntos
    public double PlanoX, PlanoY; //Proyección o sombra
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Giro en los tres ejes
    public void Giro(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 }
      };

      //Hace el giro
      Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];
    }

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

    //Cuadra en pantalla
    public void CuadraPantalla(int XpantallaIni, int YpantallaIni, int XpantallaFin, int YpantallaFin) {
      //Los valores extremos de las coordenadas del cubo
      double maximoX = 0.879315437691778;
      double minimoX = -0.879315437691778;
      double maximoY = 0.879315398752379;
      double minimoY = -0.879315437691778;

      //Las constantes de transformación
      double convierteX = (XpantallaFin - XpantallaIni) / (maximoX - minimoX);
      double convierteY = (YpantallaFin - YpantallaIni) / (maximoY - minimoY);
```

```
      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX - minimoX) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY - minimoY) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Triángulo
  internal class Poligono {
    public int punto1, punto2, punto3;
    public double Centro;

    //Coordenadas de dibujo del polígono
    Point Punto1, Punto2, Punto3;

    public Poligono(int punto1, int punto2, int punto3) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
    }

    //Calcula la profundidad y crea los vértices del polígo
    public void CalculaProfundidad(List<Punto> puntos) {
      double Z1 = puntos[punto1].Zgiro;
      double Z2 = puntos[punto2].Zgiro;
      double Z3 = puntos[punto3].Zgiro;
      Centro = (Z1+ Z2 +Z3) / 3;

      Punto1 = new(puntos[punto1].Xpantalla, puntos[punto1].Ypantalla);
      Punto2 = new(puntos[punto2].Xpantalla, puntos[punto2].Ypantalla);
      Punto3 = new(puntos[punto3].Xpantalla, puntos[punto3].Ypantalla);
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
      Point[] ListaPuntos = [Punto1, Punto2, Punto3];

      //Dibuja el polígono relleno y eso borra los polígonos que no se ven
      Lienzo.FillPolygon(Relleno, ListaPuntos);

      //Dibuja el perímetro del polígono haciéndolo visible.
      Lienzo.DrawPolygon(Lapiz, ListaPuntos);
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

    //Constructor
    public Objeto3D() {
```

```
    //Ejemplo de coordenadas espaciales X,Y,Z
    puntos = [
      new Punto( 0.00000000000,  0.361803,  0.5),
      new Punto( 0.00000000000,  0.361803, -0.5),
      new Punto( 0.00000000000, -0.361803,  0.5),
      new Punto( 0.00000000000, -0.361803, -0.5),
      new Punto( 0.361803,  0.5,  0.00000000000),
      new Punto( 0.361803, -0.5,  0.00000000000),
      new Punto(-0.361803,  0.5,  0.00000000000),
      new Punto(-0.361803, -0.5,  0.00000000000),
      new Punto( 0.5,  0.00000000000,  0.361803),
      new Punto( 0.5,  0.00000000000, -0.361803),
      new Punto(-0.5,  0.00000000000,  0.361803),
      new Punto(-0.5,  0.00000000000, -0.361803),
     ];


    //Los polígonos que dibujan el icosaedro
    poligonos = [
       new Poligono(0, 6, 4),
       new Poligono(0, 4, 8),
       new Poligono(0, 8, 2),
       new Poligono(0, 2, 10),
       new Poligono(0, 10, 6),
       new Poligono(1, 9, 4),
       new Poligono(1, 4, 6),
       new Poligono(1, 6, 11),
       new Poligono(1, 11, 3),
       new Poligono(1, 3, 9),
       new Poligono(2, 8, 5),
       new Poligono(2, 5, 7),
       new Poligono(2, 7, 10),
       new Poligono(3, 11, 7),
       new Poligono(3, 7, 5),
       new Poligono(3, 5, 9),
       new Poligono(6, 10, 11),
       new Poligono(5, 8, 9),
       new Poligono(7, 11, 10),
       new Poligono(4, 9, 8)
     ];
   }

   public void GirarFigura(double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int
YpantallaIni, int XpantallaFin, int YpantallaFin) {
     //Gira los 8 puntos
     for (int cont = 0; cont < puntos.Count; cont++) {
       puntos[cont].Giro(angX, angY, angZ);
       puntos[cont].Proyecta(ZPersona);
       puntos[cont].CuadraPantalla(XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin);
     }

     //Calcula la profundidad promedio
     for (int Cont = 0; Cont < poligonos.Count; Cont++)
       poligonos[Cont].CalculaProfundidad(puntos);

     //Ordena del polígono más alejado al más cercano,
     //de esa manera los polígonos de adelante son
     //visibles y los de atrás son borrados.
     //Algoritmo de pintor.
     poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
   }

   //Dibuja el icosaedro
   public void Dibuja(Graphics lienzo, Pen lapizIcosaedro, Brush relleno) {
     for (int Cont = 0; Cont < poligonos.Count; Cont++)
       poligonos[Cont].Dibuja(lienzo, lapizIcosaedro, relleno);
   }
  }
}
```

```
using System.Diagnostics;

namespace Graficos
{
  //Proyección 3D a 2D y giros. Figura centrada.
  public partial class Form1 : Form {

    //Para calcular los FPS (Frames Per Second)
    private Stopwatch cronometro = new Stopwatch();
    private int fps = 0;
    private int contadorFrames = 0;
    private long tiempoAcumulado = 0;

    //El icosaedro que se proyecta y gira
    Objeto3D Figura3D;

    //Tamaño de pantalla
    double ZPersona;
    int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

    public Form1() {
      InitializeComponent();

      //Para calcular los FPS (Frames Per Second)
      cronometro.Start();
      System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();
      timer.Interval = 16; // ~60 FPS
      timer.Tick += (s, e) => this.Invalidate(); // Forzar redibujo
      timer.Start();

      //Distancia del observador
      ZPersona = 5;

      //Tamaño de pantalla
      XpantallaIni = 20;
      YpantallaIni = 20;
      XpantallaFin = 500;
      YpantallaFin = 500;

      Figura3D = new Objeto3D();
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {
      double AnguloX = Convert.ToDouble(numGiroX.Value);
      double AnguloY = Convert.ToDouble(numGiroY.Value);
      double AnguloZ = Convert.ToDouble(numGiroZ.Value);

      //Gira, proyecta y cuadra cada punto de la figura 3D
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);

      //Dibuja la figura 3D
      Graphics Lienzo = e.Graphics;
      Pen LapizIcosaedro = new(Color.Black, 1);
      Brush Relleno = new SolidBrush(Color.White);
      Figura3D.Dibuja(Lienzo, LapizIcosaedro, Relleno);

      //Calcular los FPS
      contadorFrames++;
      tiempoAcumulado += cronometro.ElapsedMilliseconds;
      cronometro.Restart();

      if (tiempoAcumulado >= 1000) // cada segundo
      {
        fps = contadorFrames;
        contadorFrames = 0;
        tiempoAcumulado = 0;
      }

      // Dibujar texto con los FPS
      Lienzo.DrawString($"FPS: {fps}", this.Font, Brushes.Red, 10, 10);
    }

    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();
    }
  }
}
```

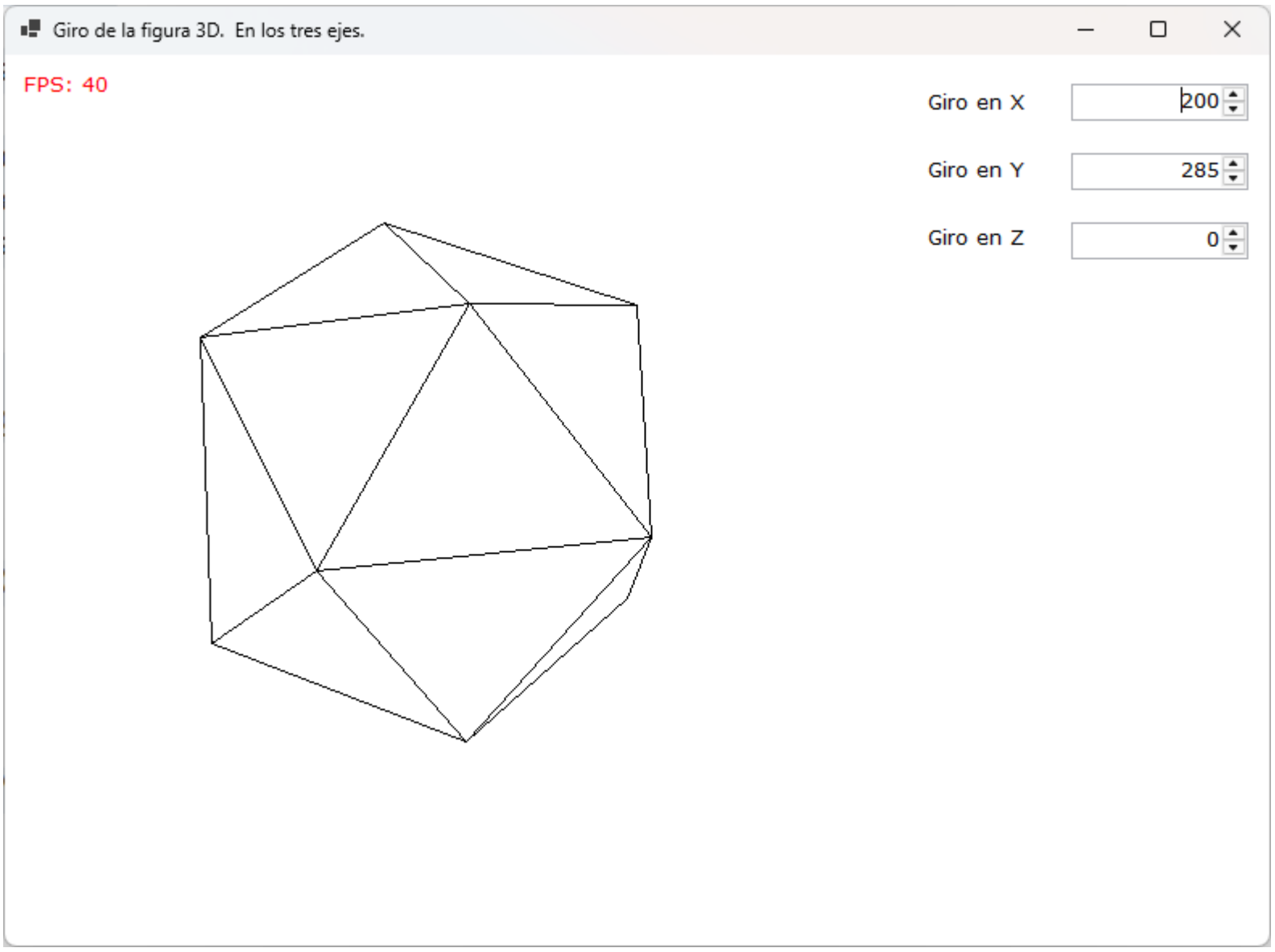


Ilustración 21: Icosaedro pintado con líneas ocultas

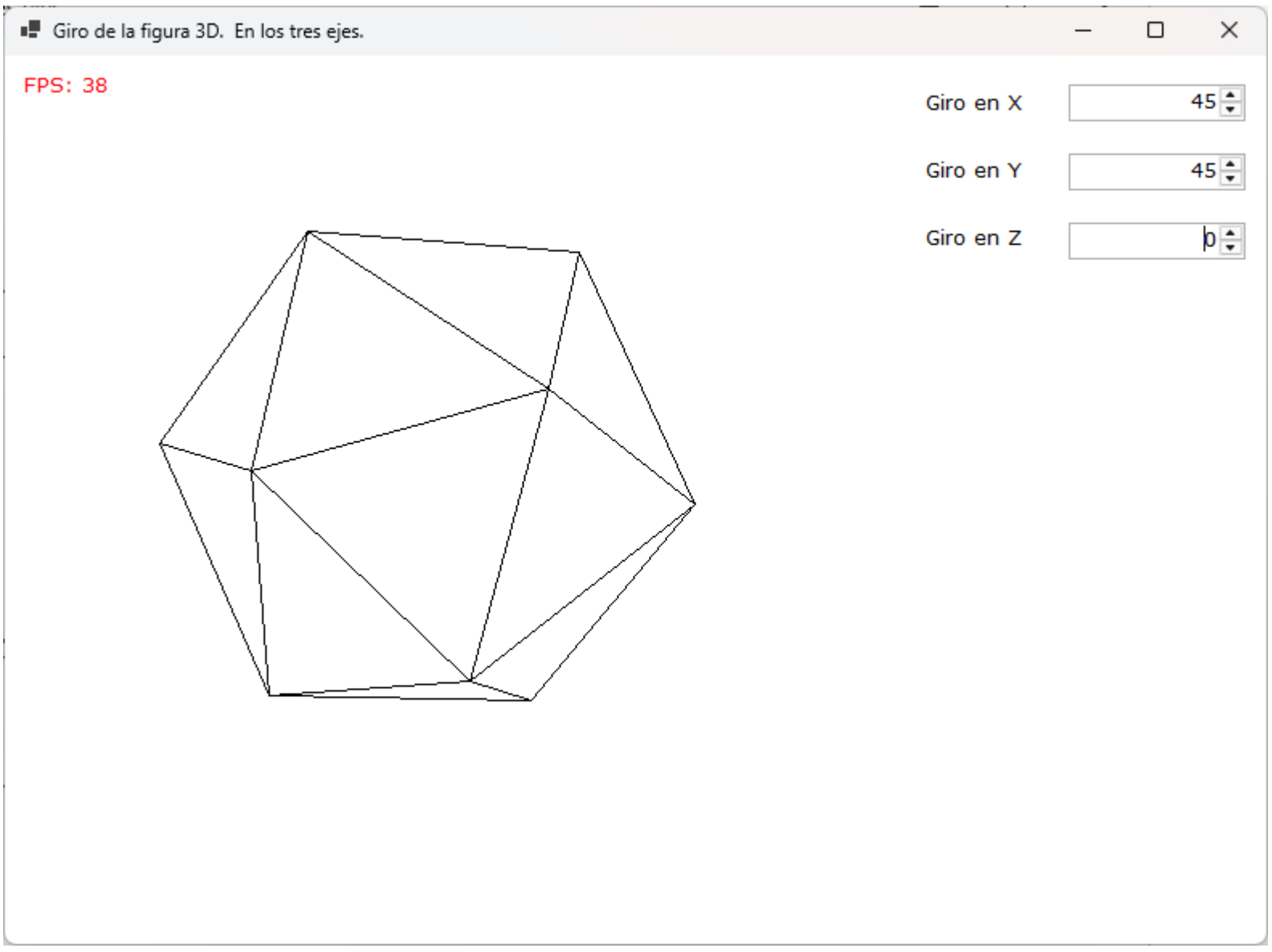


Ilustración 22: Icosaedro pintado con líneas ocultas

Líneas ocultas. Optimización del código.

Algunas operaciones no son necesarias repetirlas cuando el icosaedro gira. Luego se analiza el código buscando como mejorar su velocidad.

M/010.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    private double X, Y, Z; //Coordenadas originales
    public double Zgiro; //Al girar los puntos
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      this.Z = Z;
    }

    //Giro en los tres ejes
    public void Giro(double[,] Matriz, double ZPersona, int XpantallaIni, int YpantallaIni, double
convierteX, double convierteY) {

      //Hace el giro
      double Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      double Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];

      //Convierte de 3D a 2D (segunda dimensión)
      double PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      double PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX + 0.879315437691778) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY + 0.879315437691778) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Triángulo
  internal class Poligono {
    public int punto1, punto2, punto3;
    public double Centro;

    //Coordenadas de dibujo del polígono
    private Point Polig1, Polig2, Polig3;
    private Point[] ListaPuntos;

    public Poligono(int punto1, int punto2, int punto3) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
    }

    //Calcula la profundidad y crea los vértices del polígono
    public void ProfundidadFigura(List<Punto> puntos) {
      double Z1 = puntos[punto1].Zgiro;
```

```
      double Z2 = puntos[punto2].Zgiro;
      double Z3 = puntos[punto3].Zgiro;
      Centro = (Z1 + Z2 + Z3) / 3;

      Polig1 = new(puntos[punto1].Xpantalla, puntos[punto1].Ypantalla);
      Polig2 = new(puntos[punto2].Xpantalla, puntos[punto2].Ypantalla);
      Polig3 = new(puntos[punto3].Xpantalla, puntos[punto3].Ypantalla);
      ListaPuntos = [Polig1, Polig2, Polig3];
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
      //Dibuja el polígono relleno y eso borra los polígonos que no se ven
      Lienzo.FillPolygon(Relleno, ListaPuntos);

      //Dibuja el perímetro del polígono haciéndolo visible.
      Lienzo.DrawPolygon(Lapiz, ListaPuntos);
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

    //Constructor
    public Objeto3D() {
      //Ejemplo de coordenadas espaciales X,Y,Z
      puntos = [
        new Punto( 0.00000000000,  0.361803,  0.5),
        new Punto( 0.00000000000,  0.361803, -0.5),
        new Punto( 0.00000000000, -0.361803,  0.5),
        new Punto( 0.00000000000, -0.361803, -0.5),
        new Punto( 0.361803,  0.5,  0.00000000000),
        new Punto( 0.361803, -0.5,  0.00000000000),
        new Punto(-0.361803,  0.5,  0.00000000000),
        new Punto(-0.361803, -0.5,  0.00000000000),
        new Punto( 0.5,  0.00000000000,  0.361803),
        new Punto( 0.5,  0.00000000000, -0.361803),
        new Punto(-0.5,  0.00000000000,  0.361803),
        new Punto(-0.5,  0.00000000000, -0.361803),
       ];


      //Los polígonos que dibujan el icosaedro
      poligonos = [
         new Poligono(0, 6, 4),
         new Poligono(0, 4, 8),
         new Poligono(0, 8, 2),
         new Poligono(0, 2, 10),
         new Poligono(0, 10, 6),
         new Poligono(1, 9, 4),
         new Poligono(1, 4, 6),
         new Poligono(1, 6, 11),
         new Poligono(1, 11, 3),
         new Poligono(1, 3, 9),
         new Poligono(2, 8, 5),
         new Poligono(2, 5, 7),
         new Poligono(2, 7, 10),
         new Poligono(3, 11, 7),
         new Poligono(3, 7, 5),
         new Poligono(3, 5, 9),
         new Poligono(6, 10, 11),
         new Poligono(5, 8, 9),
         new Poligono(7, 11, 10),
         new Poligono(4, 9, 8)
      ];
    }
```

```
   public void GirarFigura(double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int
YpantallaIni, int XpantallaFin, int YpantallaFin) {
     //Para la matriz de rotación
     double CosX = Math.Cos(angX);
     double SinX = Math.Sin(angX);
     double CosY = Math.Cos(angY);
     double SinY = Math.Sin(angY);
     double CosZ = Math.Cos(angZ);
     double SinZ = Math.Sin(angZ);

     //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 }
     };

     //Las constantes de transformación para cuadrar en pantalla
     double convierteX = (XpantallaFin - XpantallaIni) / 1.758630875383556;
     double convierteY = (YpantallaFin - YpantallaIni) / 1.758630875383556;

     //Gira los 8 puntos
     for (int cont = 0; cont < puntos.Count; cont++)
       puntos[cont].Giro(Matriz, ZPersona, XpantallaIni, YpantallaIni, convierteX, convierteY);

     //Calcula la profundidad y forma el polígono
     for (int Cont = 0; Cont < poligonos.Count; Cont++)
       poligonos[Cont].ProfundidadFigura(puntos);

     //Algoritmo de pintor.
     //Ordena del polígono más alejado al más cercano,
     //los polígonos de adelante son visibles y los de atrás son borrados.
     poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
   }

   //Dibuja el icosaedro
   public void Dibuja(Graphics lienzo, Pen lapizIcosaedro, Brush relleno) {
     for (int Cont = 0; Cont < poligonos.Count; Cont++)
       poligonos[Cont].Dibuja(lienzo, lapizIcosaedro, relleno);
   }
  }
}
```

```
using System.Diagnostics;

namespace Graficos {
  //Proyección 3D a 2D y giros. Figura centrada. Optimización.
  public partial class Form1 : Form {

    //Para calcular los FPS (Frames Per Second)
    private Stopwatch cronometro = new Stopwatch();
    private int fps = 0;
    private int contadorFrames = 0;
    private long tiempoAcumulado = 0;

    //El icosaedro que se proyecta y gira
    Objeto3D Figura3D;

    //Tamaño de pantalla
    double AnguloX, AnguloY, AnguloZ, ZPersona;
    const double Radianes = Math.PI / 180;
    int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

    //Para dibujar
    Pen LapizIcosaedro = new(Color.Black, 1);
    Brush Relleno = new SolidBrush(Color.White);

    public Form1() {
      InitializeComponent();

      //Para calcular los FPS (Frames Per Second)
      cronometro.Start();
```

```
      System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();
      timer.Interval = 15; // ~60 FPS
      timer.Tick += (s, e) => this.Invalidate(); // Forzar redibujo
      timer.Start();

      //Distancia del observador
      ZPersona = 5;

      //Tamaño de pantalla
      XpantallaIni = 20;
      YpantallaIni = 20;
      XpantallaFin = 500;
      YpantallaFin = 500;

      Figura3D = new Objeto3D();

      //Gira, proyecta y cuadra cada punto de la figura 3D
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {

      //Dibuja la figura 3D
      Figura3D.Dibuja(e.Graphics, LapizIcosaedro, Relleno);

      //Calcular los FPS
      contadorFrames++;
      tiempoAcumulado += cronometro.ElapsedMilliseconds;
      cronometro.Restart();

      if (tiempoAcumulado >= 1000) {  // cada segundo
        fps = contadorFrames;
        contadorFrames = 0;
        tiempoAcumulado = 0;
      }

      // Dibujar texto con los FPS
      e.Graphics.DrawString($"FPS: {fps}", this.Font, Brushes.Red, 10, 10);
    }

    private void numGiroX_ValueChanged(object sender, EventArgs e) {
      AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
      Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
      AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
      Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
      AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
      Refresh();
    }
  }
}
```

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:

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

Los pasos para hacer el gráfico son los siguientes (implementación en M/011):

Paso 1

Saber el valor mínimo de X (Xini) hasta el valor máximo de X (Xfin). Igual sucede con Y, el valor mínimo de Y (Yini) hasta el valor máximo de Y (Yfin). También se pregunta cuantas líneas tendrá el gráfico (NumLineas), entre más líneas, más detallado será el gráfico.

```
    double Xini = -10;
    double Yini = -10;
    double Xfin = 10;
    double Yfin = 10;
    int NumLineas = 40;
```

Paso 2

Se calcula el avance en X que sería: IncrX = (Xfin - Xini) / (NumLineas-1)

Se calcula el avance en Y que sería: IncrY = (Yfin - Yini) / (NumLineas-1)

```
    double IncrX = (Xfin - Xini) / (NumLineas-1);
    double IncrY = (Yfin - Yini) / (NumLineas-1);
    double X = Xini, Y = Yini;
```

Paso 3

El gráfico requiere una lista de objetos tipo punto. Cada objeto tiene:

a. Los puntos X, Y, Z (Z es calculado). Si el valor de Z no es real (infinito o NaN), hace que el punto no sea válido, luego no se usará más adelante.
b. Los puntos Xg, Yg, Zg (más adelante se calculan)
c. Los puntos planoX, planoY (más adelante se calculan)
d. Los puntos Xp, Yp (más adelante se calculan)

```
    //Coordenadas espaciales X,Y,Z
    puntos = [];
    for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
      for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
        double Z = Ecuacion(X, Y);
        puntos.Add(new Punto(X, Y, Z));
        X += IncrX;
      }
      X = Xini;
      Y += IncrY;
    }
```

Paso 4

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.

```
    //Debe normalizar las coordenadas y ponerlas entre -0.5 y 0.5
    double Xmin = double.MaxValue;
    double Ymin = double.MaxValue;
    double Zmin = double.MaxValue;
    double Xmax = double.MinValue;
    double Ymax = double.MinValue;
    double Zmax = double.MinValue;
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      if (puntos[Cont].X < Xmin) Xmin = puntos[Cont].X;
      if (puntos[Cont].Y < Ymin) Ymin = puntos[Cont].Y;
      if (puntos[Cont].Z < Zmin) Zmin = puntos[Cont].Z;
      if (puntos[Cont].X > Xmax) Xmax = puntos[Cont].X;
```

```
      if (puntos[Cont].Y > Ymax) Ymax = puntos[Cont].Y;
      if (puntos[Cont].Z > Zmax) Zmax = puntos[Cont].Z;
    }

    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      puntos[Cont].X = (puntos[Cont].X - Xmin) / (Xmax - Xmin) - 0.5;
      puntos[Cont].Y = (puntos[Cont].Y - Ymin) / (Ymax - Ymin) - 0.5;
      puntos[Cont].Z = (puntos[Cont].Z - Zmin) / (Zmax - Zmin) - 0.5;
    }
```

Paso 5

Se forman los polígonos con cuatro puntos

```
    // Inicializa la lista de polígonos
    poligonos = [];

    int coordenadaActual = 0;
    int filaActual = 1;
    int totalPoligonos = (NumLineas - 1) * (NumLineas - 1);

    for (int Cont = 0; Cont < totalPoligonos; Cont++) {
      // Crea un polígono con las coordenadas de los vértices
      // siempre y cuando los vértices sean válidos
      if (puntos[coordenadaActual].Valido &&
        puntos[coordenadaActual + 1].Valido &&
        puntos[coordenadaActual + NumLineas + 1].Valido &&
        puntos[coordenadaActual + NumLineas].Valido) {

        poligonos.Add(new Poligono(
          coordenadaActual,
          coordenadaActual + 1,
          coordenadaActual + NumLineas + 1,
          coordenadaActual + NumLineas
        ));
      }

      coordenadaActual++;

      // Salta al inicio de la siguiente fila si se alcanza el final de la actual
      if (coordenadaActual == NumLineas * filaActual - 1) {
        coordenadaActual++;
        filaActual++;
      }
    }
  }
```

Paso 6

Se gira cada polígono. En el código es sencillo: cada polígono tiene 4 puntos donde cada punto está en una lista, luego en vez de girar polígono a polígono, se gira toda la lista de puntos.

```
   public void GirarFigura(double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int
YpantallaIni, int XpantallaFin, int YpantallaFin) {
    //Para la matriz de rotación
    double CosX = Math.Cos(angX);
    double SinX = Math.Sin(angX);
    double CosY = Math.Cos(angY);
    double SinY = Math.Sin(angY);
    double CosZ = Math.Cos(angZ);
    double SinZ = Math.Sin(angZ);

    //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 }
    };

    //Las constantes de transformación para cuadrar en pantalla
    double convierteX = (XpantallaFin - XpantallaIni) / 1.758630875383556;
```

```
    double convierteY = (YpantallaFin - YpantallaIni) / 1.758630875383556;

    //Gira los puntos
    for (int cont = 0; cont < puntos.Count; cont++)
      puntos[cont].Giro(Matriz, ZPersona, XpantallaIni, YpantallaIni, convierteX, convierteY);

    //Calcula la profundidad y forma el polígono
    for (int Cont = 0; Cont < poligonos.Count; Cont++)
      poligonos[Cont].ProfundidadFigura(puntos);

    //Algoritmo de pintor.
    //Ordena del polígono más alejado al más cercano,
    //los polígonos de adelante son visibles y los de atrás son borrados.
    poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
  }
```

Y así se cuadra cada punto en pantalla

```
  //Giro en los tres ejes
  public void Giro(double[,] Matriz, double ZPersona, int XpantallaIni, int YpantallaIni, double
convierteX, double convierteY) {

    //Hace el giro
    double Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
    double Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
    Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];

    //Convierte de 3D a 2D (segunda dimensión)
    double PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
    double PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

    //Deduce las coordenadadas de pantalla
    Xpantalla = Convert.ToInt32(convierteX * (PlanoX + 0.879315437691778) + XpantallaIni);
    Ypantalla = Convert.ToInt32(convierteY * (PlanoY + 0.879315437691778) + YpantallaIni);
  }
```

Paso 7

Se dibuja cada polígono

```
  //Dibuja la figura 3D
  public void Dibuja(Graphics lienzo, Pen lapizIcosaedro, Brush relleno) {
    for (int Cont = 0; Cont < poligonos.Count; Cont++)
      poligonos[Cont].Dibuja(lienzo, lapizIcosaedro, relleno);
  }
```

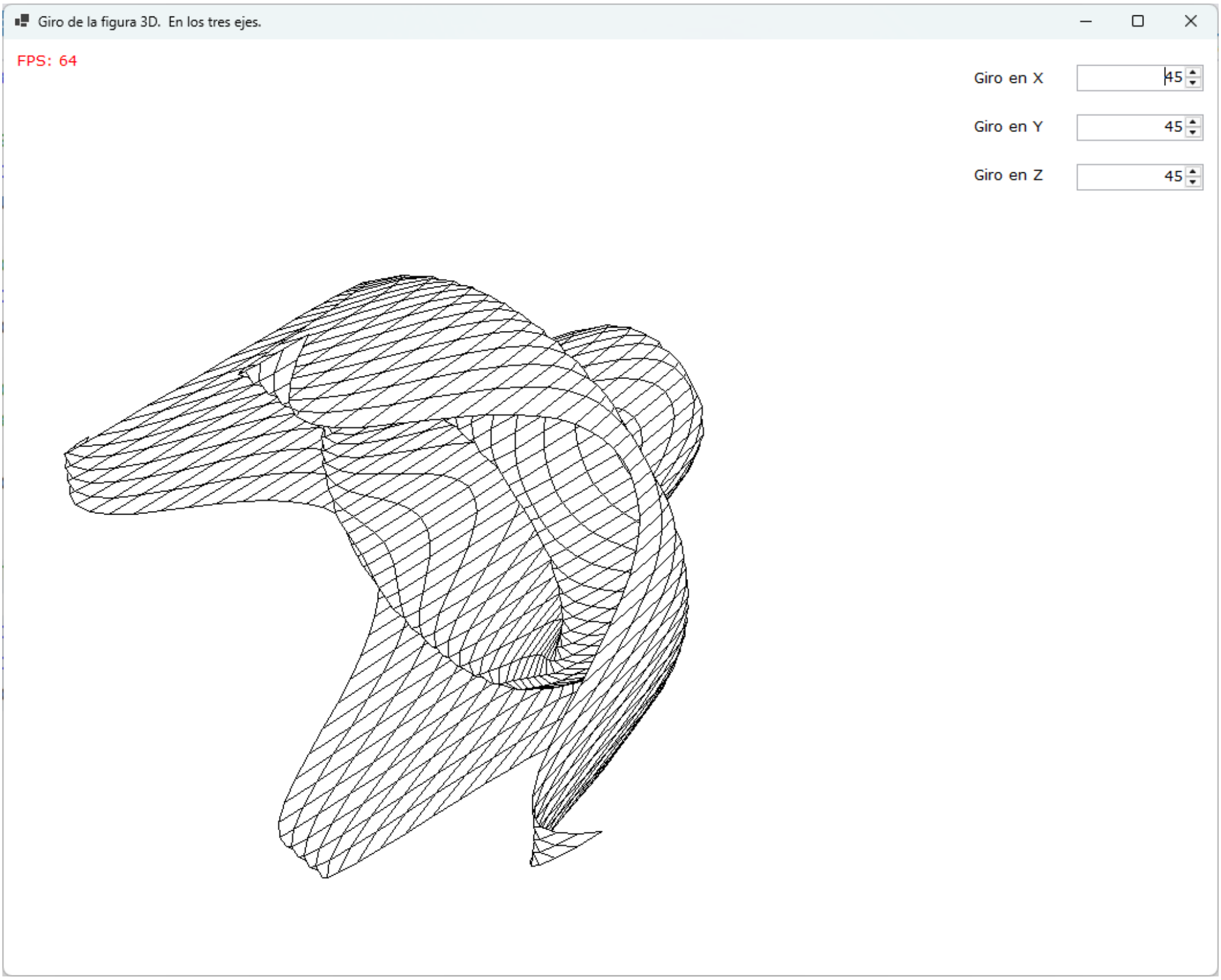


Ilustración 23: Ecuación en 3D proyectada

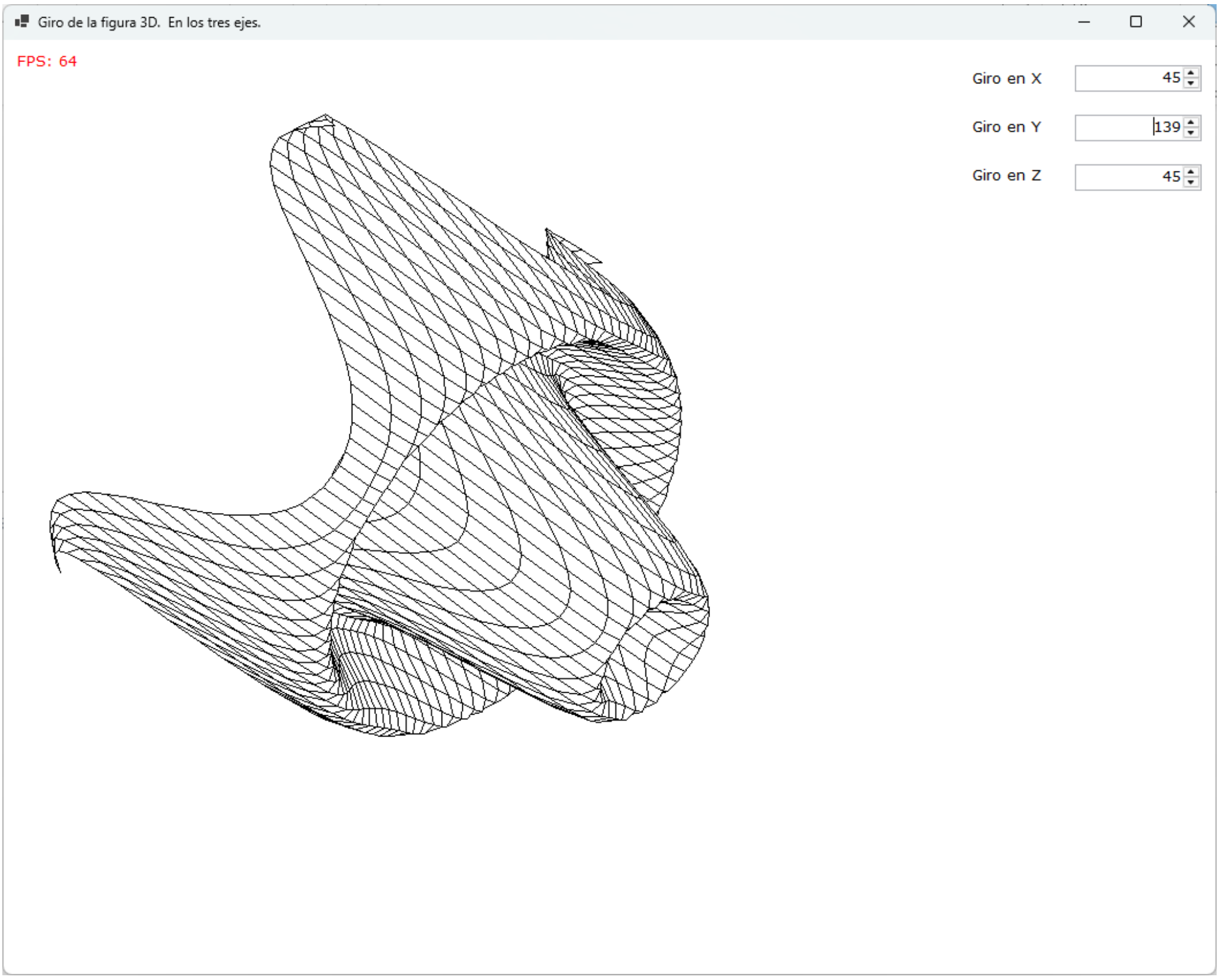


Ilustración 24: Ecuación en 3D proyectada

M/011.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    public bool Valido; //Si el punto es válido
    public double X, Y, Z; //Coordenadas originales
    public double Zgiro; //Al girar los puntos
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      Valido = true;
      if (double.IsFinite(Z)) {
        this.Z = Z;
      } else {
        Valido = false;
      }
    }

    //Giro en los tres ejes
    public void Giro(double[,] Matriz, double ZPersona, int XpantallaIni, int YpantallaIni, double
convierteX, double convierteY) {
      //Si el punto no es válido, no hace nada
      if (Valido == false) return;

      //Hace el giro
      double Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      double Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
```

```
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];

      //Convierte de 3D a 2D (segunda dimensión)
      double PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      double PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX + 0.879315437691778) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY + 0.879315437691778) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Triángulo
  internal class Poligono {
    public int punto1, punto2, punto3, punto4;
    public double Centro;

    //Coordenadas de dibujo del polígono
    private Point Polig1, Polig2, Polig3, Polig4;
    private Point[] ListaPuntos;

    public Poligono(int punto1, int punto2, int punto3, int punto4) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
      this.punto4 = punto4;
    }

    //Calcula la profundidad y crea los vértices del polígono
    public void ProfundidadFigura(List<Punto> puntos) {
      double Z1 = puntos[punto1].Zgiro;
      double Z2 = puntos[punto2].Zgiro;
      double Z3 = puntos[punto3].Zgiro;
      double Z4 = puntos[punto4].Zgiro;
      Centro = (Z1 + Z2 + Z3 + Z4) / 4;

      Polig1 = new(puntos[punto1].Xpantalla, puntos[punto1].Ypantalla);
      Polig2 = new(puntos[punto2].Xpantalla, puntos[punto2].Ypantalla);
      Polig3 = new(puntos[punto3].Xpantalla, puntos[punto3].Ypantalla);
      Polig4 = new(puntos[punto4].Xpantalla, puntos[punto4].Ypantalla);
      ListaPuntos = [Polig1, Polig2, Polig3, Polig4];
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
      //Dibuja el polígono relleno y eso borra los polígonos que no se ven
      Lienzo.FillPolygon(Relleno, ListaPuntos);

      //Dibuja el perímetro del polígono haciéndolo visible.
      Lienzo.DrawPolygon(Lapiz, ListaPuntos);
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
```

```
private List<Poligono> poligonos;

//Constructor
public Objeto3D() {
  double Xini = -10;
  double Yini = -10;
  double Xfin = 10;
  double Yfin = 10;
  int NumLineas = 40;

  double IncrX = (Xfin - Xini) / (NumLineas-1);
  double IncrY = (Yfin - Yini) / (NumLineas-1);
  double X = Xini, Y = Yini;

  //Coordenadas espaciales X,Y,Z
  puntos = [];
  for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
    for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
      double Z = Ecuacion(X, Y);
      puntos.Add(new Punto(X, Y, Z));
      X += IncrX;
    }
    X = Xini;
    Y += IncrY;
  }

  //Debe normalizar las coordenadas y ponerlas entre -0.5 y 0.5
  double Xmin = double.MaxValue;
  double Ymin = double.MaxValue;
  double Zmin = double.MaxValue;
  double Xmax = double.MinValue;
  double Ymax = double.MinValue;
  double Zmax = double.MinValue;
  for (int Cont = 0; Cont < puntos.Count; Cont++) {
    if (puntos[Cont].Valido == false) continue;
    if (puntos[Cont].X < Xmin) Xmin = puntos[Cont].X;
    if (puntos[Cont].Y < Ymin) Ymin = puntos[Cont].Y;
    if (puntos[Cont].Z < Zmin) Zmin = puntos[Cont].Z;
    if (puntos[Cont].X > Xmax) Xmax = puntos[Cont].X;
    if (puntos[Cont].Y > Ymax) Ymax = puntos[Cont].Y;
    if (puntos[Cont].Z > Zmax) Zmax = puntos[Cont].Z;
  }

  for (int Cont = 0; Cont < puntos.Count; Cont++) {
    if (puntos[Cont].Valido == false) continue;
    puntos[Cont].X = (puntos[Cont].X - Xmin) / (Xmax - Xmin) - 0.5;
    puntos[Cont].Y = (puntos[Cont].Y - Ymin) / (Ymax - Ymin) - 0.5;
    puntos[Cont].Z = (puntos[Cont].Z - Zmin) / (Zmax - Zmin) - 0.5;
  }

  // Inicializa la lista de polígonos
  poligonos = [];

  int coordenadaActual = 0;
  int filaActual = 1;
  int totalPoligonos = (NumLineas - 1) * (NumLineas - 1);

  for (int Cont = 0; Cont < totalPoligonos; Cont++) {
    // Crea un polígono con las coordenadas de los vértices
    // siempre y cuando los vértices sean válidos
    if (puntos[coordenadaActual].Valido &&
      puntos[coordenadaActual + 1].Valido &&
      puntos[coordenadaActual + NumLineas + 1].Valido &&
      puntos[coordenadaActual + NumLineas].Valido) {

      poligonos.Add(new Poligono(
        coordenadaActual,
        coordenadaActual + 1,
        coordenadaActual + NumLineas + 1,
        coordenadaActual + NumLineas
      ));
    }

    coordenadaActual++;

    // Salta al inicio de la siguiente fila si se alcanza el final de la actual
```

```
      if (coordenadaActual == NumLineas * filaActual - 1) {
        coordenadaActual++;
        filaActual++;
      }
    }
  }

  public double Ecuacion(double X, double Y) {
    double Z = Math.Sqrt(X*X + Y*Y) + 3 * Math.Cos(Math.Sqrt(X * X + Y * Y)) + 5;
    return Z;
  }

  public void GirarFigura(double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int
YpantallaIni, int XpantallaFin, int YpantallaFin) {
    //Para la matriz de rotación
    double CosX = Math.Cos(angX);
    double SinX = Math.Sin(angX);
    double CosY = Math.Cos(angY);
    double SinY = Math.Sin(angY);
    double CosZ = Math.Cos(angZ);
    double SinZ = Math.Sin(angZ);

    //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 }
    };

    //Las constantes de transformación para cuadrar en pantalla
    double convierteX = (XpantallaFin - XpantallaIni) / 1.758630875383556;
    double convierteY = (YpantallaFin - YpantallaIni) / 1.758630875383556;

    //Gira los puntos
    for (int cont = 0; cont < puntos.Count; cont++)
      puntos[cont].Giro(Matriz, ZPersona, XpantallaIni, YpantallaIni, convierteX, convierteY);

    //Calcula la profundidad y forma el polígono
    for (int Cont = 0; Cont < poligonos.Count; Cont++)
      poligonos[Cont].ProfundidadFigura(puntos);

    //Algoritmo de pintor.
    //Ordena del polígono más alejado al más cercano,
    //los polígonos de adelante son visibles y los de atrás son borrados.
    poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
  }

  //Dibuja la figura 3D
  public void Dibuja(Graphics lienzo, Pen lapizIcosaedro, Brush relleno) {
    for (int Cont = 0; Cont < poligonos.Count; Cont++)
      poligonos[Cont].Dibuja(lienzo, lapizIcosaedro, relleno);
  }
 }
}
```

```
using System.Diagnostics;

namespace Graficos {
 //Proyección 3D a 2D y giros. Figura centrada. Optimización.
 public partial class Form1 : Form {

  //Para calcular los FPS (Frames Per Second)
  private Stopwatch cronometro = new Stopwatch();
  private int fps = 0;
  private int contadorFrames = 0;
  private long tiempoAcumulado = 0;

  //El icosaedro que se proyecta y gira
  Objeto3D Figura3D;

  //Tamaño de pantalla
  double AnguloX, AnguloY, AnguloZ, ZPersona;
  const double Radianes = Math.PI / 180;
  int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;
```

```
//Para dibujar
Pen LapizIcosaedro = new(Color.Black, 1);
Brush Relleno = new SolidBrush(Color.White);

public Form1() {
  InitializeComponent();

  //Para calcular los FPS (Frames Per Second)
  cronometro.Start();
  System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();
  timer.Interval = 15; // ~60 FPS
  timer.Tick += (s, e) => this.Invalidate(); // Forzar redibujo
  timer.Start();

  //Distancia del observador
  ZPersona = 5;

  //Tamaño de pantalla
  XpantallaIni = 20;
  YpantallaIni = 20;
  XpantallaFin = 800;
  YpantallaFin = 800;

  Figura3D = new Objeto3D();

  //Gira, proyecta y cuadra cada punto de la figura 3D
  AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
  AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
  AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
  Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
 }

 private void Form1_Paint(object sender, PaintEventArgs e) {

  //Dibuja la figura 3D
  Figura3D.Dibuja(e.Graphics, LapizIcosaedro, Relleno);

  //Calcular los FPS
  contadorFrames++;
  tiempoAcumulado += cronometro.ElapsedMilliseconds;
  cronometro.Restart();

  if (tiempoAcumulado >= 1000) {  // cada segundo
   fps = contadorFrames;
   contadorFrames = 0;
   tiempoAcumulado = 0;
  }

  // Dibujar texto con los FPS
  e.Graphics.DrawString($"FPS: {fps}", this.Font, Brushes.Red, 10, 10);
 }

 private void numGiroX_ValueChanged(object sender, EventArgs e) {
  AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
  Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
  Refresh();
 }

 private void numGiroY_ValueChanged(object sender, EventArgs e) {
  AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
  Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
  Refresh();
 }

 private void numGiroZ_ValueChanged(object sender, EventArgs e) {
  AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
  Figura3D.GirarFigura(AnguloX, AnguloY, AnguloZ, ZPersona, XpantallaIni, YpantallaIni, XpantallaFin,
YpantallaFin);
  Refresh();
 }
 }
}
```

Gráfico Matemático en 3D. Colores según altura y evaluador de expresiones.

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.

La ecuación no es fija sino que el usuario la ingresa y es evaluada por el programa.

M/012.zip

```
using System.Globalization;
using System.Text;

/* Evaluador de expresiones versión 4 (enero de 2024)
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com ;  enginelife@hotmail.com
 * URL: http://darwin.50webs.com
 * GitHub: https://github.com/ramsoftware
 * */

namespace Graficos {
  internal class ParteEvl4 {
    /* Constantes de los diferentes tipos
     * de datos que tendrán las piezas */
    private const int ESFUNCION = 1;
    private const int ESPARABRE = 2;
    private const int ESOPERADOR = 4;
    private const int ESNUMERO = 5;
    private const int ESVARIABLE = 6;

    /* Acumulador, función, paréntesis que abre,
     * paréntesis que cierra, operador,
     * número, variable */
    public int Tipo;

    /* Código de la función 0:seno, 1:coseno, 2:tangente,
     * 3: valor absoluto, 4: arcoseno, 5: arcocoseno,
     * 6: arcotangente, 7: logaritmo natural, 8: exponencial
     * 9: raíz cuadrada */
    public int Funcion;

    /* + suma - resta * multiplicación / división ^ potencia */
    public int Operador;

    /* Posición en lista de valores del número literal
     * por ejemplo: 3.141592 */
    public int posNumero;

    /* Posición en lista de valores del
     * valor de la variable algebraica */
    public int posVariable;

    /* Posición en lista de valores del valor de la pieza.
     * Por ejemplo:
       3 + 2 / 5  se convierte así:
       |3| |+| |2| |/| |5|
       |3| |+| |A|  A es un identificador de acumulador */
    public int posAcumula;

    public ParteEvl4(int TipoParte, int Valor) {
      this.Tipo = TipoParte;
      switch (TipoParte) {
        case ESFUNCION: this.Funcion = Valor; break;
        case ESNUMERO: this.posNumero = Valor; break;
        case ESVARIABLE: this.posVariable = Valor; break;
        case ESPARABRE: this.Funcion = -1; break;
      }
    }

    public ParteEvl4(char Operador) {
      this.Tipo = ESOPERADOR;
      switch (Operador) {
        case '+': this.Operador = 0; break;
        case '-': this.Operador = 1; break;
        case '*': this.Operador = 2; break;
        case '/': this.Operador = 3; break;
        case '^': this.Operador = 4; break;
```

```
    }
  }
}


internal class PiezaEvl4 {
  /* Posición donde se almacena el valor que genera
   * la pieza al evaluarse */
  public int PosResultado;

  /* Código de la función 0:seno, 1:coseno, 2:tangente,
   * 3: valor absoluto, 4: arcoseno, 5: arcocoseno,
   * 6: arcotangente, 7: logaritmo natural, 8: valor tope,
   * 9: exponencial, 10: raíz cuadrada */
  public int Funcion;

  /* Posición donde se almacena la primera parte de la pieza */
  public int pA;

  /* 0 suma 1 resta 2 multiplicación 3 división 4 potencia */
  public int Operador;

  /* Posición donde se almacena la segunda parte de la pieza */
  public int pB;
}

internal class Evaluador4 {
  /* Constantes de los diferentes tipos
   * de datos que tendrán las piezas */
  private const int ESFUNCION = 1;
  private const int ESPARABRE = 2;
  private const int ESPARCIERRA = 3;
  private const int ESOPERADOR = 4;
  private const int ESNUMERO = 5;
  private const int ESVARIABLE = 6;
  private const int ESACUMULA = 7;

  /* Expresión algebraica convertida de la
   * original dada por el usuario */
  private StringBuilder Analizada = new();

  /* Donde guarda los valores de variables, constantes y piezas */
  private List<double> Valores = new List<double>();

  /* Listado de partes en que se divide la expresión
     Toma una expresión, por ejemplo:
     1.68 + sen( 3 / x ) * ( 2.92 - 9 )
     y la divide en partes así:
     [1.68] [+] [sen(] [3] [/] [x] [)] [*] [(] [2.92] [-] [9] [)]
     Cada parte puede tener un número, un operador, una función,
     un paréntesis que abre o un paréntesis que cierra.
     En esta versión 4.0, las constantes, piezas y variables guardan
     sus valores en la lista Valores.
     En partes, se almacena la posición en Valores */
  private List<ParteEvl4> Partes = new List<ParteEvl4>();

  /* Listado de piezas que ejecutan
    Toma las partes y las divide en piezas con
    la siguiente estructura:

    acumula = funcion valor operador valor

    donde valor puede ser un número, una variable o
    un acumulador

    Siguiendo el ejemplo anterior sería:
    A =  2.92 - 9
    B =  sen( 3 / x )
    C =  B * A
    D =  1.68 + C

     Esas piezas se evalúan de arriba a abajo y así
  se interpreta la ecuación */
  private List<PiezaEvl4> Piezas = new List<PiezaEvl4>();

  /* Analiza la expresión */
```

```
public int Analizar(string ExpresionOriginal) {
  Partes.Clear();
  Piezas.Clear();
  Valores.Clear();

  /* Hace espacio para las 26 variables que
   * puede manejar el evaluador */
  for (int Variables = 1; Variables <= 26; Variables++)
    Valores.Add(0);

  int Sintaxis = ChequeaSintaxis(ExpresionOriginal);
  if (Sintaxis == 0) {
    CrearPartes();
    CrearPiezas();
  }
  return Sintaxis;
}

/* Retorna mensaje de error sintáctico */
public string MensajeError(int CodigoError) {
  string Msj = "";
  switch (CodigoError) {
    case 1:
      Msj = "1. Número seguido de letra";
      break;

    case 2:
      Msj = "2. Número seguido de paréntesis que abre";
      break;

    case 3:
      Msj = "3. Doble punto seguido";
      break;

    case 4:
      Msj = "4. Punto seguido de operador";
      break;

    case 5:
      Msj = "5. Punto y sigue una letra";
      break;

    case 6:
      Msj = "6. Punto seguido de paréntesis que abre";
      break;

    case 7:
      Msj = "7. Punto seguido de paréntesis que cierra";
      break;

    case 8:
      Msj = "8. Operador seguido de un punto";
      break;

    case 9:
      Msj = "9. Dos operadores estén seguidos";
      break;

    case 10:
      Msj = "10. Operador seguido de paréntesis que cierra";
      break;

    case 11:
      Msj = "11. Letra seguida de número";
      break;

    case 12:
      Msj = "12. Letra seguida de punto";
      break;

    case 13:
      Msj = "13. Letra seguida de otra letra";
      break;

    case 14:
      Msj = "14. Letra seguida de paréntesis que abre";
```

```
      break;

    case 15:
      Msj = "15. Paréntesis que abre seguido de punto";
      break;

    case 16:
      Msj = "16. Paréntesis que abre y sigue operador";
      break;

    case 17:
      Msj = "17. Paréntesis que abre y luego cierra";
      break;

    case 18:
      Msj = "18. Paréntesis que cierra y sigue número";
      break;

    case 19:
      Msj = "19. Paréntesis que cierra y sigue punto";
      break;

    case 20:
      Msj = "20. Paréntesis que cierra y sigue letra";
      break;

    case 21:
      Msj = "21. Paréntesis que cierra y luego abre";
      break;

    case 22:
      Msj = "22. Inicia con operador";
      break;

    case 23:
      Msj = "23. Finaliza con operador";
      break;

    case 24:
      Msj = "24. No hay correspondencia entre paréntesis";
      break;

    case 25:
      Msj = "25. Paréntesis desbalanceados";
      break;

    case 26:
      Msj = "26. Dos o más puntos en número real";
      break;

    case 27:
      Msj = "27. Expresión vacía";
      break;
  }
  return Msj;
}

/* Da valor a las variables que tendrá
 * la expresión algebraica */
public void DarValorVariable(char varAlgebra, double Valor) {
  Valores[varAlgebra - 'a'] = Valor;
}

/* Evalúa la expresión convertida en piezas */
public double Evaluar() {
  double Resulta = 0;
  PiezaEvl4 tmp;

  /* Va de pieza en pieza */
  for (int Posicion = 0; Posicion < Piezas.Count; Posicion++) {
    tmp = Piezas[Posicion];

    switch (tmp.Operador) {
      case 0:
        Resulta = Valores[tmp.pA] + Valores[tmp.pB];
        break;
```

```
      case 1:
        Resulta = Valores[tmp.pA] - Valores[tmp.pB];
        break;
      case 2:
        Resulta = Valores[tmp.pA] * Valores[tmp.pB];
        break;
      case 3:
        Resulta = Valores[tmp.pA] / Valores[tmp.pB];
        break;
      default:
        Resulta = Math.Pow(Valores[tmp.pA], Valores[tmp.pB]);
        break;
    }

    switch (tmp.Funcion) {
      case 0: Resulta = Math.Sin(Resulta); break;
      case 1: Resulta = Math.Cos(Resulta); break;
      case 2: Resulta = Math.Tan(Resulta); break;
      case 3: Resulta = Math.Abs(Resulta); break;
      case 4: Resulta = Math.Asin(Resulta); break;
      case 5: Resulta = Math.Acos(Resulta); break;
      case 6: Resulta = Math.Atan(Resulta); break;
      case 7: Resulta = Math.Log(Resulta); break;
      case 8: Resulta = Math.Exp(Resulta); break;
      case 9: Resulta = Math.Sqrt(Resulta); break;
    }

      Valores[tmp.PosResultado] = Resulta;
    }
    return Resulta;
  }

  /* Divide la expresión en partes, yendo de caracter en caracter */
  private void CrearPartes() {
    StringBuilder Numero = new StringBuilder();
    for (int Pos = 0; Pos < this.Analizada.Length; Pos++) {
      char Letra = this.Analizada[Pos];
      switch (Letra) {
        case '.':
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':   /* Si es un digito o un punto
             * va acumulando el número */
          Numero.Append(Letra); break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
          /* Si es un operador matemático entonces verifica
           * si hay un número que se ha acumulado */
          if (Numero.Length > 0) {
            Valores.Add(Convierte(Numero));
            Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
            Numero.Clear();
          }
          /* Agregar el operador matemático */
          Partes.Add(new ParteEvl4(Letra));
          break;

        case '(':   /* Es paréntesis que abre */
          Partes.Add(new ParteEvl4(ESPARABRE, 0));
          break;

        case ')':
          /* Si es un paréntesis que cierra entonces verifica
           * si hay un número que se ha acumulado */
          if (Numero.Length > 0) {
            Valores.Add(Convierte(Numero));
```

```
        Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
        Numero.Clear();
      }

      /* Si sólo había un número o variable
       * dentro del paréntesis le agrega + 0
       * (por ejemplo:  sen(x) o 3*(2) ) */
      if (Partes[Partes.Count - 2].Tipo == ESPARABRE ||
        Partes[Partes.Count - 2].Tipo == ESFUNCION) {
        Partes.Add(new ParteEvl4(ESOPERADOR, 0));
        Valores.Add(0);
        Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
      }

      /* Adiciona el paréntesis que cierra */
      Partes.Add(new ParteEvl4(ESPARCIERRA, 0));
      break;
    case 'A':   /* Seno */
    case 'B':   /* Coseno */
    case 'C':   /* Tangente */
    case 'D':   /* Valor absoluto */
    case 'E':   /* Arcoseno */
    case 'F':   /* Arcocoseno */
    case 'G':   /* Arcotangente */
    case 'H':   /* Logaritmo natural */
    case 'I':   /* Exponencial */
    case 'J':   /* Raíz cuadrada */
      Partes.Add(new ParteEvl4(ESFUNCION, Letra - 'A'));
      Pos++;
      break;
    default:
      Partes.Add(new ParteEvl4(ESVARIABLE, Letra - 'a'));
      break;
  }
 }
}

private double Convierte(StringBuilder Numero) {
  string Cad = Numero.ToString();
  return double.Parse(Cad, CultureInfo.InvariantCulture);
}

/* Convierte las partes en las piezas finales de ejecución */
private void CrearPiezas() {
  int Contador = Partes.Count - 1;
  do {
    ParteEvl4 tmpParte = Partes[Contador];
    if (tmpParte.Tipo == ESPARABRE || tmpParte.Tipo == ESFUNCION) {

      /* Evalúa las potencias */
      GeneraPiezaOpera(4, 4, Contador);

      /* Luego evalúa multiplicar y dividir */
      GeneraPiezaOpera(2, 3, Contador);

      /* Finalmente evalúa sumar y restar */
      GeneraPiezaOpera(0, 1, Contador);

      /* Agrega la función a la última pieza */
      if (tmpParte.Tipo == ESFUNCION) {
        Piezas[Piezas.Count - 1].Funcion = tmpParte.Funcion;
      }

      /* Quita el paréntesis/función que abre y
       * el que cierra, dejando el centro */
      Partes.RemoveAt(Contador);
      Partes.RemoveAt(Contador + 1);
    }
    Contador--;
  } while (Contador >= 0);
}

/* Genera las piezas buscando determinado operador */
private void GeneraPiezaOpera(int OperA, int OperB, int Inicia) {
  int Contador = Inicia + 1;
  do {
```

```
ParteEvl4 tmpParte = Partes[Contador];
if (tmpParte.Tipo == ESOPERADOR &&
  (tmpParte.Operador == OperA || tmpParte.Operador == OperB)) {
  ParteEvl4 tmpParteIzq = Partes[Contador - 1];
  ParteEvl4 tmpParteDer = Partes[Contador + 1];

  /* Crea Pieza */
  PiezaEvl4 NuevaPieza = new PiezaEvl4();
  NuevaPieza.Funcion = -1;

  switch (tmpParteIzq.Tipo) {
    case ESNUMERO:
      NuevaPieza.pA = tmpParteIzq.posNumero;
      break;

    case ESVARIABLE:
      NuevaPieza.pA = tmpParteIzq.posVariable;
      break;

    default:
      NuevaPieza.pA = tmpParteIzq.posAcumula;
      break;
  }

  NuevaPieza.Operador = tmpParte.Operador;

switch (tmpParteDer.Tipo) {
  case ESNUMERO:
    NuevaPieza.pB = tmpParteDer.posNumero;
    break;

  case ESVARIABLE:
    NuevaPieza.pB = tmpParteDer.posVariable;
    break;

  default:
    NuevaPieza.pB = tmpParteDer.posAcumula;
    break;
}

/* Añade a lista de piezas y crea una
 * nueva posición en Valores */
Valores.Add(0);
NuevaPieza.PosResultado = Valores.Count - 1;
Piezas.Add(NuevaPieza);

/* Elimina la parte del operador y la siguiente */
Partes.RemoveAt(Contador);
Partes.RemoveAt(Contador);

/* Retorna el contador en uno para tomar
 * la siguiente operación */
Contador--;

      /* Cambia la parte anterior por parte que acumula */
      tmpParteIzq.Tipo = ESACUMULA;
      tmpParteIzq.posAcumula = NuevaPieza.PosResultado;
    }
    Contador++;
  } while (Partes[Contador].Tipo != ESPARCIERRA);
}

private int ChequeaSintaxis(string ExpOrig) {
  /* Primero a minúsculas */
  StringBuilder Minusculas = new StringBuilder(ExpOrig.ToLower());

  /* Sólo los caracteres válidos */
  string Valido = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
  HashSet<char> Permite = new HashSet<char>(Valido);
  StringBuilder ConLetrasValidas = new StringBuilder("(");
  for (int Cont = 0; Cont < Minusculas.Length; Cont++)
    if (Permite.Contains(Minusculas[Cont]))
      ConLetrasValidas.Append(Minusculas[Cont]);
  ConLetrasValidas.Append(')');

  /* Agrega +0) donde exista )) porque es
```

```
 * necesario para crear las piezas */
string nuevo = ConLetrasValidas.ToString();
if (nuevo.Length == 2) return 27;
while (nuevo.IndexOf("))") != -1)
  nuevo = nuevo.Replace("))", ")+0)");
ConLetrasValidas = new StringBuilder(nuevo);

/* Validación de sintaxis, se genera una copia
 * y allí se reemplaza las funciones por a+( */
StringBuilder sbSintax = new StringBuilder();
sbSintax.Append(ConLetrasValidas);
sbSintax.Replace("sen(", "a+(");
sbSintax.Replace("cos(", "a+(");
sbSintax.Replace("tan(", "a+(");
sbSintax.Replace("abs(", "a+(");
sbSintax.Replace("asn(", "a+(");
sbSintax.Replace("acs(", "a+(");
sbSintax.Replace("atn(", "a+(");
sbSintax.Replace("log(", "a+(");
sbSintax.Replace("exp(", "a+(");
sbSintax.Replace("sqr(", "a+(");

for (int Cont = 1; Cont < sbSintax.Length - 2; Cont++) {
  char cA = sbSintax[Cont];
  char cB = sbSintax[Cont + 1];

  if (Char.IsDigit(cA) && Char.IsLower(cB)) return 1;
  if (Char.IsDigit(cA) && cB == '(') return 2;
  if (cA == '.' && cB == '.') return 3;
  if (cA == '.' && EsUnOperador(cB)) return 4;
  if (cA == '.' && Char.IsLower(cB)) return 5;
  if (cA == '.' && cB == '(') return 6;
  if (cA == '.' && cB == ')') return 7;
  if (EsUnOperador(cA) && cB == '.') return 8;
  if (EsUnOperador(cA) && EsUnOperador(cB)) return 9;
  if (EsUnOperador(cA) && cB == ')') return 10;
  if (Char.IsLower(cA) && Char.IsDigit(cB)) return 11;
  if (Char.IsLower(cA) && cB == '.') return 12;
  if (Char.IsLower(cA) && Char.IsLower(cB)) return 13;
  if (Char.IsLower(cA) && cB == '(') return 14;
  if (cA == '(' && cB == '.') return 15;
  if (cA == '(' && EsUnOperador(cB)) return 16;
  if (cA == '(' && cB == ')') return 17;
  if (cA == ')' && Char.IsDigit(cB)) return 18;
  if (cA == ')' && cB == '.') return 19;
  if (cA == ')' && Char.IsLower(cB)) return 20;
  if (cA == ')' && cB == '(') return 21;
}

/* Valida el inicio y fin de la expresión */
if (EsUnOperador(sbSintax[1])) return 22;
if (EsUnOperador(sbSintax[sbSintax.Length - 2])) return 23;

/* Valida balance de paréntesis */
int ParentesisAbre = 0; /* Contador de paréntesis que abre */
int ParentesisCierra = 0; /* Contador de paréntesis que cierra */
for (int Cont = 1; Cont < sbSintax.Length - 1; Cont++) {
  switch (sbSintax[Cont]) {
    case '(': ParentesisAbre++; break;
    case ')': ParentesisCierra++; break;
  }
  if (ParentesisCierra > ParentesisAbre) return 24;
}
if (ParentesisAbre != ParentesisCierra) return 25;

/* Validar los puntos decimales de un número real */
int TotalPuntos = 0;
for (int Cont = 0; Cont < sbSintax.Length; Cont++) {
  if (EsUnOperador(sbSintax[Cont])) TotalPuntos = 0;
  if (sbSintax[Cont] == '.') TotalPuntos++;
  if (TotalPuntos > 1) return 26;
}

/* Deja la expresión para ser analizada.
 * Reemplaza las funciones de tres letras
 * por una letra mayúscula. Cambia los )) 
```

```
   * por )+0) porque es requerido al crear las piezas */
  this.Analizada.Length = 0;
  this.Analizada.Append(ConLetrasValidas);
  this.Analizada.Replace("sen", "A");
  this.Analizada.Replace("cos", "B");
  this.Analizada.Replace("tan", "C");
  this.Analizada.Replace("abs", "D");
  this.Analizada.Replace("asn", "E");
  this.Analizada.Replace("acs", "F");
  this.Analizada.Replace("atn", "G");
  this.Analizada.Replace("log", "H");
  this.Analizada.Replace("exp", "I");
  this.Analizada.Replace("sqr", "J");

  /* Sintaxis correcta */
  return 0;
}

/* Retorna si el Caracter es un operador matemático */
private static bool EsUnOperador(char Caracter) {
  switch (Caracter) {
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
      return true;
    default:
      return false;
  }
 }
}
```

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    public bool Valido; //Si el punto es válido
    public double X, Y, Z; //Coordenadas originales
    public double Zgiro; //Al girar los puntos
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      Valido = true;
      if (double.IsFinite(Z)) {
        this.Z = Z;
      }
      else {
        Valido = false;
      }
    }

    //Giro en los tres ejes
    public void Giro(double[,] Matriz, double ZPersona, int XpantallaIni, int YpantallaIni, double
convierteX, double convierteY) {
      //Si el punto no es válido, no hace nada
      if (Valido == false) return;

      //Hace el giro
      double Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      double Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];

      //Convierte de 3D a 2D (segunda dimensión)
      double PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      double PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX + 0.879315437691778) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY + 0.879315437691778) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Triángulo
  internal class Poligono {
    public int punto1, punto2, punto3, punto4;
    public double Centro;

    //Coordenadas de dibujo del polígono
    private Point Polig1, Polig2, Polig3, Polig4;
    private Point[] ListaPuntos;

    //Color de relleno del polígono
    Color ColorZ;

    public Poligono(int punto1, int punto2, int punto3, int punto4) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
      this.punto4 = punto4;
    }

    //Calcula la profundidad y crea los vértices del polígono
    public void ProfundidadFigura(List<Punto> puntos, List<Color> colorList) {
      double Z1 = puntos[punto1].Zgiro;
      double Z2 = puntos[punto2].Zgiro;
      double Z3 = puntos[punto3].Zgiro;
      double Z4 = puntos[punto4].Zgiro;
      Centro = (Z1 + Z2 + Z3 + Z4) / 4;

      Polig1 = new(puntos[punto1].Xpantalla, puntos[punto1].Ypantalla);
      Polig2 = new(puntos[punto2].Xpantalla, puntos[punto2].Ypantalla);
      Polig3 = new(puntos[punto3].Xpantalla, puntos[punto3].Ypantalla);
      Polig4 = new(puntos[punto4].Xpantalla, puntos[punto4].Ypantalla);
      ListaPuntos = [Polig1, Polig2, Polig3, Polig4];

      int TotalColores = colorList.Count;
      double PromedioZ = (puntos[punto1].Z + puntos[punto2].Z + puntos[punto3].Z + puntos[punto4].Z + 2) /
4;
      int ColorEscoge = (int)Math.Floor(TotalColores * PromedioZ);
      ColorZ = colorList[ColorEscoge];
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo) {
      //Dibuja el polígono relleno
      Brush Relleno2 = new SolidBrush(ColorZ);
      Lienzo.FillPolygon(Relleno2, ListaPuntos);
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

    //Colores para pintar la malla
```

```
List<Color> ListaColores;

//Ecuación
string EcuacionAnterior = "";
Evaluador4 Evaluador = new();

public Objeto3D() {
  int NumColores = 40; // Número de colores a generar

  //Genera listado de colores para el gráfico
  ListaColores = [];
  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)));
    ListaColores.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)));
    ListaColores.Add(Color.FromArgb(Rojo, Verde, Azul));
  }
 }

 //Hace los cálculos de la ecuación Z = F(X,Y)
 //para dibujarla
 public void CalcularFigura3D(string Ecuacion, double Xini, double Yini, double Xfin, double Yfin, int
NumLineas, double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int YpantallaIni, int
XpantallaFin, int YpantallaFin) {

  //Evalúa la ecuación. Si es nueva, la analiza.
  if (Ecuacion.Equals(EcuacionAnterior) == false) {
    int Sintaxis = Evaluador.Analizar(Ecuacion);
    if (Sintaxis > 0) { //Tiene un error de sintaxis
      string MensajeError = Evaluador.MensajeError(Sintaxis);
      MessageBox.Show(MensajeError, "Error de sintaxis",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }
  }
  EcuacionAnterior = Ecuacion;

  //Incrementos X, Y
  double IncrX = (Xfin - Xini) / (NumLineas - 1);
  double IncrY = (Yfin - Yini) / (NumLineas - 1);
  double X = Xini, Y = Yini;

  //Coordenadas espaciales X,Y,Z
  X = Xini;
  Y = Yini;
  puntos = [];
  for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
    for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
      Evaluador.DarValorVariable('x', X);
      Evaluador.DarValorVariable('y', Y);
      double Z = Evaluador.Evaluar();
      puntos.Add(new Punto(X, Y, Z));
      X += IncrX;
    }
    X = Xini;
    Y += IncrY;
  }

  //Debe normalizar las coordenadas y ponerlas entre -0.5 y 0.5
  double Xmin = double.MaxValue;
  double Ymin = double.MaxValue;
  double Zmin = double.MaxValue;
  double Xmax = double.MinValue;
  double Ymax = double.MinValue;
  double Zmax = double.MinValue;
```

```
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      if (puntos[Cont].X < Xmin) Xmin = puntos[Cont].X;
      if (puntos[Cont].Y < Ymin) Ymin = puntos[Cont].Y;
      if (puntos[Cont].Z < Zmin) Zmin = puntos[Cont].Z;
      if (puntos[Cont].X > Xmax) Xmax = puntos[Cont].X;
      if (puntos[Cont].Y > Ymax) Ymax = puntos[Cont].Y;
      if (puntos[Cont].Z > Zmax) Zmax = puntos[Cont].Z;
    }

    if (Math.Abs(Xmin - Xmax) < 0.0001 || Math.Abs(Ymin - Ymax) < 0.0001 || Math.Abs(Zmin - Zmax) <
0.0001) {
      MessageBox.Show("La ecuación digitada no puede generar un gráfico", "Error de cálculo",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }

    //Normaliza
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      puntos[Cont].X = (puntos[Cont].X - Xmin) / (Xmax - Xmin) - 0.5;
      puntos[Cont].Y = (puntos[Cont].Y - Ymin) / (Ymax - Ymin) - 0.5;
      puntos[Cont].Z = (puntos[Cont].Z - Zmin) / (Zmax - Zmin) - 0.5;
    }

    // Inicializa la lista de polígonos
    poligonos = [];

    int coordenadaActual = 0;
    int filaActual = 1;
    int totalPoligonos = (NumLineas - 1) * (NumLineas - 1);

    for (int Cont = 0; Cont < totalPoligonos; Cont++) {
      // Crea un polígono con las coordenadas de los vértices
      // siempre y cuando los vértices sean válidos
      if (puntos[coordenadaActual].Valido &&
        puntos[coordenadaActual + 1].Valido &&
        puntos[coordenadaActual + NumLineas + 1].Valido &&
        puntos[coordenadaActual + NumLineas].Valido) {

        poligonos.Add(new Poligono(
          coordenadaActual,
          coordenadaActual + 1,
          coordenadaActual + NumLineas + 1,
          coordenadaActual + NumLineas
        ));
      }

      coordenadaActual++;

      // Salta al inicio de la siguiente fila si se alcanza el final de la actual
      if (coordenadaActual == NumLineas * filaActual - 1) {
        coordenadaActual++;
        filaActual++;
      }
    }

    //Para la matriz de rotación
    double CosX = Math.Cos(angX);
    double SinX = Math.Sin(angX);
    double CosY = Math.Cos(angY);
    double SinY = Math.Sin(angY);
    double CosZ = Math.Cos(angZ);
    double SinZ = Math.Sin(angZ);

    //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 }
    };

    //Las constantes de transformación para cuadrar en pantalla
    double convierteX = (XpantallaFin - XpantallaIni) / 1.758630875383556;
    double convierteY = (YpantallaFin - YpantallaIni) / 1.758630875383556;
```

```
      //Gira los puntos válidos
      for (int cont = 0; cont < puntos.Count; cont++)
         puntos[cont].Giro(Matriz, ZPersona, XpantallaIni, YpantallaIni, convierteX, convierteY);

      //Calcula la profundidad y forma el polígono
      for (int Cont = 0; Cont < poligonos.Count; Cont++)
        poligonos[Cont].ProfundidadFigura(puntos, ListaColores);

      //Algoritmo de pintor.
      //Ordena del polígono más alejado al más cercano,
      //los polígonos de adelante son visibles y los de atrás son borrados.
      poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
    }

    //Dibuja la figura 3D
    public void Dibuja(Graphics lienzo) {
      for (int Cont = 0; Cont < poligonos.Count; Cont++)
        poligonos[Cont].Dibuja(lienzo);
    }
  }
}
```

```
namespace Graficos {
  public partial class Valores : Form {

    Form1 FrmGrafico = new();
    const double Radianes = Math.PI / 180;
    public Valores() {
      InitializeComponent();

      FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
      FrmGrafico.Yini = Convert.ToDouble((double)numMinimoY.Value);
      FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
      FrmGrafico.Yfin = Convert.ToDouble((double)numMaximoY.Value);
      FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
      FrmGrafico.Ecuacion = txtEcuacion.Text;
      FrmGrafico.ZPersona = 5;
      FrmGrafico.HuboCambio = true;

      FrmGrafico.Show();
    }

    private void numGiroX_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numMinimoX_ValueChanged(object sender, EventArgs e) {
      if (numMinimoX.Value >= numMaximoX.Value)
        numMinimoX.Value = numMaximoX.Value - 1;

      FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numMinimoY_ValueChanged(object sender, EventArgs e) {
      if (numMinimoY.Value >= numMaximoY.Value)
```

```
    numMinimoY.Value = numMaximoY.Value - 1;

  FrmGrafico.Yini = Convert.ToDouble((double)numMinimoY.Value);
  FrmGrafico.HuboCambio = true;
  FrmGrafico.Refresh();
}

private void numMaximoX_ValueChanged(object sender, EventArgs e) {
  if (numMinimoX.Value >= numMaximoX.Value)
    numMaximoX.Value = numMinimoX.Value + 1;

  FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
  FrmGrafico.HuboCambio = true;
  FrmGrafico.Refresh();
}

private void numMaximoY_ValueChanged(object sender, EventArgs e) {
  if (numMinimoY.Value >= numMaximoY.Value)
    numMaximoY.Value = numMinimoY.Value + 1;

  FrmGrafico.Yfin = Convert.ToDouble((double)numMaximoY.Value);
  FrmGrafico.HuboCambio = true;
  FrmGrafico.Refresh();
}

private void numTotalLineas_ValueChanged(object sender, EventArgs e) {
  FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
  FrmGrafico.HuboCambio = true;
  FrmGrafico.Refresh();
 }

 private void btnProcesar_Click(object sender, EventArgs e) {
  FrmGrafico.Ecuacion = txtEcuacion.Text;
  FrmGrafico.HuboCambio = true;
  FrmGrafico.Refresh();
 }
 }
}
```

```
namespace Graficos {
 //Proyección 3D a 2D y giros. Figura centrada. Optimización.
 public partial class Form1 : Form {
  //La figura que se proyecta y gira
  Objeto3D Figura3D;

  //Giro de figura
  public double AnguloX, AnguloY, AnguloZ;

  //Distancia del observador
  public double ZPersona;

  //Rango de valores
  public double Xini, Yini, Xfin, Yfin;

  //Número de líneas que tendrá el gráfico
  public int NumLineas;

  //Ecuación
  public string Ecuacion;

  //En caso de que el usuario haya hecho algún cambio a los valores
  //se recalcula todo de nuevo.
  public bool HuboCambio;

  public Form1() {
   InitializeComponent();
   Figura3D = new Objeto3D();
  }

  private void Form1_Paint(object sender, PaintEventArgs e) {

   //Dibuja la figura 3D
   if (HuboCambio) {
```

```
      Figura3D.CalcularFigura3D(Ecuacion, Xini, Yini, Xfin, Yfin, NumLineas, AnguloX, AnguloY, AnguloZ,
ZPersona, 0, 0, this.ClientSize.Width, this.ClientSize.Height);
      HuboCambio = false;
     }

     Figura3D.Dibuja(e.Graphics);
    }

   private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
     Application.Exit();
    }

   private void Form1_Resize(object sender, EventArgs e) {
     HuboCambio = true;
     Refresh();
    }
  }
}
```

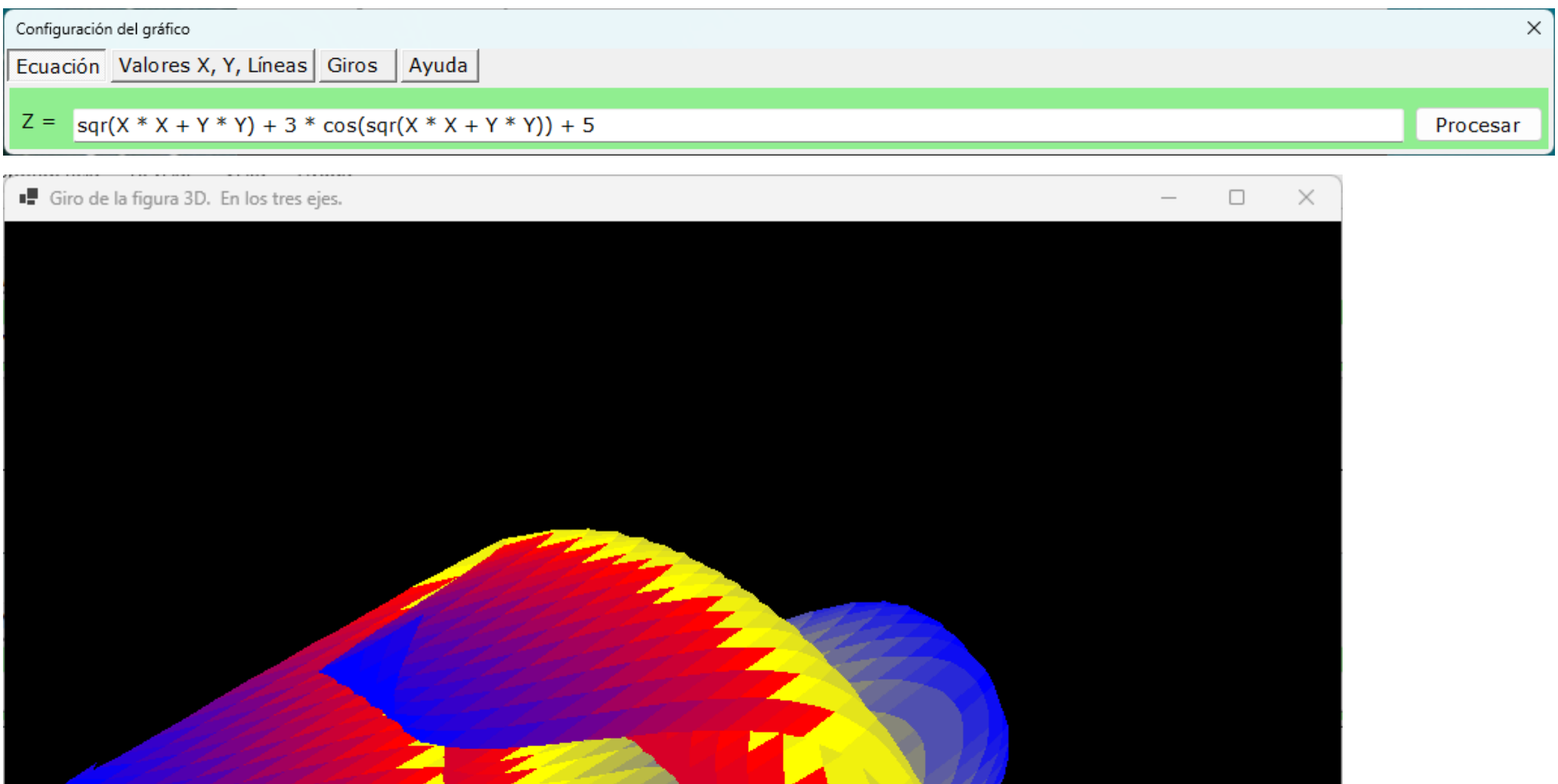


Ilustración 25: Gráfico ecuación 3D con colores según altura

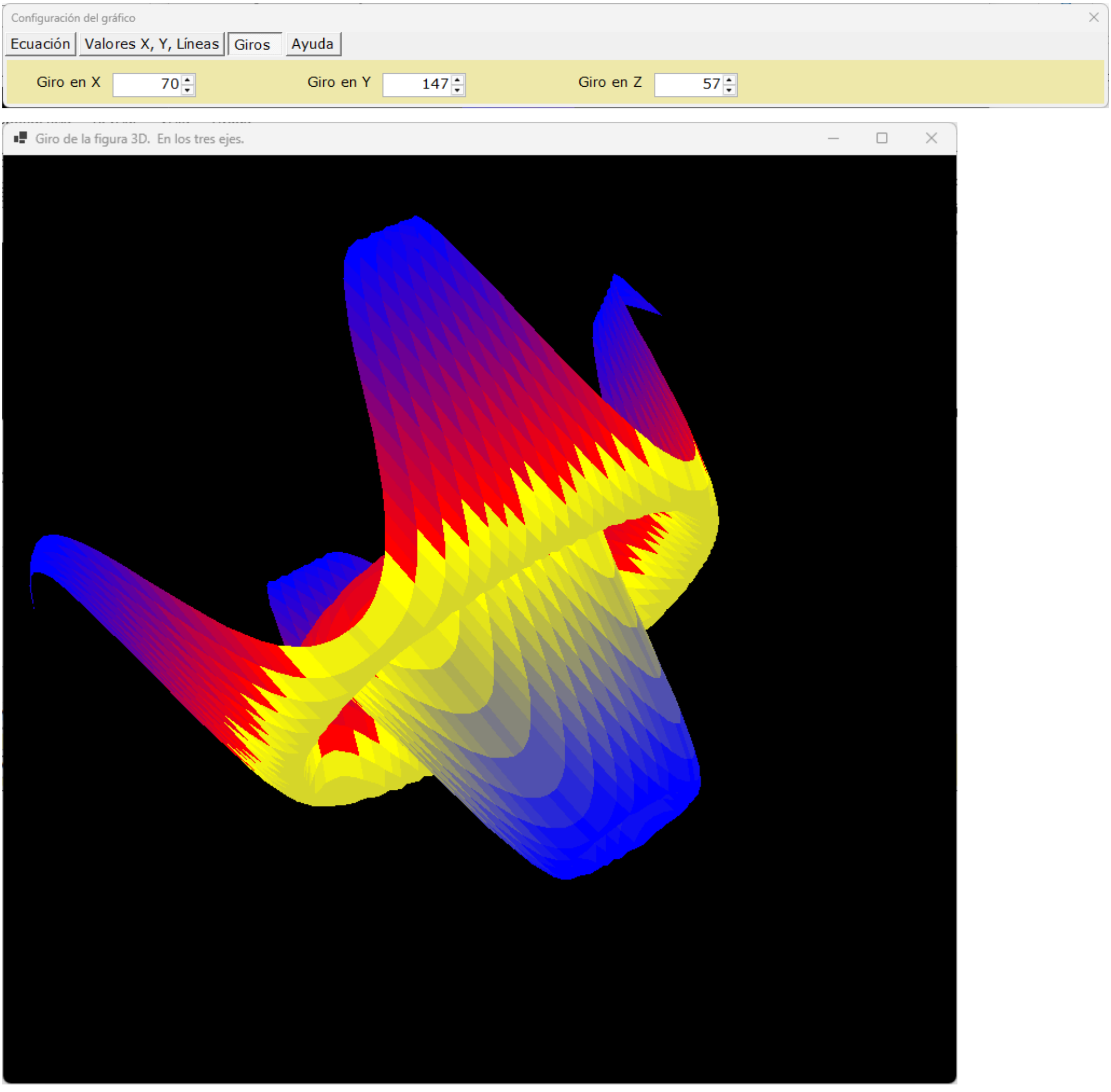


Ilustración 26: Gráfico ecuación 3D con colores según altura

Gráfico Polar en 3D

Los gráficos polares en 3D trabajan con dos ángulos: Theta y Phi. Ver: https://mathematica.stackexchange.com/questions/83867/how-to-make-a-3d-plot-using-polar-coordinates o https://stackoverflow.com/questions/55031161/polar3d-plot-with-theta-phi-and-radius o https://mathworld.wolfram.com/SphericalCoordinates.html o https://en.wikipedia.org/wiki/Spherical_coordinate_system

Por ejemplo, esta ecuación:

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

Los pasos para dibujar el gráfico polar 3D son:

Paso 1: φ (phi) va de 0 a 2π, Θ (theta) va de 0 a 2π y con eso se obtienen los valores de r

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

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

$$y = r * Sen(\varphi) * Sen(\theta)$$

$$z = r * Cos(\theta)$$

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

M/013.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    public bool Valido; //Si el punto es válido
    public double X, Y, Z; //Coordenadas originales
    public double Zgiro; //Al girar los puntos
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      Valido = true;
      if (double.IsFinite(Z)) {
        this.Z = Z;
      }
      else {
        Valido = false;
      }
    }

    //Giro en los tres ejes
    public void Giro(double[,] Matriz, double ZPersona, int XpantallaIni, int YpantallaIni, double
convierteX, double convierteY) {
      //Si el punto no es válido, no hace nada
      if (Valido == false) return;

      //Hace el giro
      double Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      double Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];

      //Convierte de 3D a 2D (segunda dimensión)
      double PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      double PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX + 0.879315437691778) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY + 0.879315437691778) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
  internal class Conexion {
```

```
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Polígono
  internal class Poligono {
    public int punto1, punto2, punto3, punto4;
    public double Centro;

    //Coordenadas de dibujo del polígono
    private Point Polig1, Polig2, Polig3, Polig4;
    private Point[] ListaPuntos;

    //Color de relleno del polígono
    Color ColorZ;

    public Poligono(int punto1, int punto2, int punto3, int punto4) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
      this.punto4 = punto4;
    }

    //Calcula la profundidad y crea los vértices del polígono
    public void ProfundidadFigura(List<Punto> puntos, List<Color> colorList) {
      double Z1 = puntos[punto1].Zgiro;
      double Z2 = puntos[punto2].Zgiro;
      double Z3 = puntos[punto3].Zgiro;
      double Z4 = puntos[punto4].Zgiro;
      Centro = (Z1 + Z2 + Z3 + Z4) / 4;

      Polig1 = new(puntos[punto1].Xpantalla, puntos[punto1].Ypantalla);
      Polig2 = new(puntos[punto2].Xpantalla, puntos[punto2].Ypantalla);
      Polig3 = new(puntos[punto3].Xpantalla, puntos[punto3].Ypantalla);
      Polig4 = new(puntos[punto4].Xpantalla, puntos[punto4].Ypantalla);
      ListaPuntos = [Polig1, Polig2, Polig3, Polig4];

      int TotalColores = colorList.Count;
      double PromedioZ = (puntos[punto1].Z + puntos[punto2].Z + puntos[punto3].Z + puntos[punto4].Z + 2) /
4;
      int ColorEscoge = (int)Math.Floor(TotalColores * PromedioZ);
      ColorZ = colorList[ColorEscoge];
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo) {
      //Dibuja el polígono relleno
      Brush Relleno2 = new SolidBrush(ColorZ);
      Lienzo.FillPolygon(Relleno2, ListaPuntos);
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

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

```
//Ecuación
string EcuacionAnterior = "";
Evaluador4 Evaluador = new();

public Objeto3D() {
  int NumColores = 40; // Número de colores a generar

  //Genera listado de colores para el gráfico
  ListaColores = [];
  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)));
    ListaColores.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)));
    ListaColores.Add(Color.FromArgb(Rojo, Verde, Azul));
  }
 }

 //Hace los cálculos de la ecuación Z = F(X,Y)
 //para dibujarla
 public void CalcularFigura3D(string Ecuacion, int NumLineas, double angX, double angY, double angZ,
double ZPersona, int XpantallaIni, int YpantallaIni, int XpantallaFin, int YpantallaFin) {

  //Evalúa la ecuación. Si es nueva, la analiza.
  if (Ecuacion.Equals(EcuacionAnterior) == false) {
    int Sintaxis = Evaluador.Analizar(Ecuacion);
    if (Sintaxis > 0) { //Tiene un error de sintaxis
      string MensajeError = Evaluador.MensajeError(Sintaxis);
      MessageBox.Show(MensajeError, "Error de sintaxis",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }
  }
  EcuacionAnterior = Ecuacion;

  //Incrementos X, Y
  double MinTheta = 0;
  double MinPhi = 0;
  double MaxTheta = 360;
  double MaxPhi = 360;
  double IncrTheta = (MaxTheta - MinTheta) / (NumLineas - 1);
  double IncrPhi = (MaxPhi - MinPhi) / (NumLineas - 1);
  double Theta = MinTheta, Phi = MinPhi;

  //Coordenadas espaciales X,Y,Z
  Theta = MinTheta;
  Phi = MinPhi;
  puntos = [];
  for (int AngTheta = 1; AngTheta <= NumLineas; AngTheta++) {
    for (int AngPhi = 1; AngPhi <= NumLineas; AngPhi++) {
      double ThetaR = Theta * Math.PI / 180;
      double PhiR = Phi * Math.PI / 180;

      Evaluador.DarValorVariable('t', ThetaR);
      Evaluador.DarValorVariable('p', PhiR);
      double R = Evaluador.Evaluar();

      double X = R * Math.Cos(PhiR) * Math.Sin(ThetaR);
      double Y = R * Math.Sin(PhiR) * Math.Sin(ThetaR);
      double Z = R * Math.Cos(ThetaR);

      puntos.Add(new Punto(X, Y, Z));
      Theta += IncrTheta;
    }
    Theta = MinTheta;
    Phi += IncrPhi;
```

```
    }

    //Debe normalizar las coordenadas y ponerlas entre -0.5 y 0.5
    double Xmin = double.MaxValue;
    double Ymin = double.MaxValue;
    double Zmin = double.MaxValue;
    double Xmax = double.MinValue;
    double Ymax = double.MinValue;
    double Zmax = double.MinValue;
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      if (puntos[Cont].X < Xmin) Xmin = puntos[Cont].X;
      if (puntos[Cont].Y < Ymin) Ymin = puntos[Cont].Y;
      if (puntos[Cont].Z < Zmin) Zmin = puntos[Cont].Z;
      if (puntos[Cont].X > Xmax) Xmax = puntos[Cont].X;
      if (puntos[Cont].Y > Ymax) Ymax = puntos[Cont].Y;
      if (puntos[Cont].Z > Zmax) Zmax = puntos[Cont].Z;
    }

    if (Math.Abs(Xmin - Xmax) < 0.0001 || Math.Abs(Ymin - Ymax) < 0.0001 || Math.Abs(Zmin - Zmax) <
0.0001) {
      MessageBox.Show("La ecuación digitada no puede generar un gráfico", "Error de cálculo",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }

    //Normaliza
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      puntos[Cont].X = (puntos[Cont].X - Xmin) / (Xmax - Xmin) - 0.5;
      puntos[Cont].Y = (puntos[Cont].Y - Ymin) / (Ymax - Ymin) - 0.5;
      puntos[Cont].Z = (puntos[Cont].Z - Zmin) / (Zmax - Zmin) - 0.5;
    }

    // Inicializa la lista de polígonos
    poligonos = [];

    int coordenadaActual = 0;
    int filaActual = 1;
    int totalPoligonos = (NumLineas - 1) * (NumLineas - 1);

    for (int Cont = 0; Cont < totalPoligonos; Cont++) {
      // Crea un polígono con las coordenadas de los vértices
      // siempre y cuando los vértices sean válidos
      if (puntos[coordenadaActual].Valido &&
        puntos[coordenadaActual + 1].Valido &&
        puntos[coordenadaActual + NumLineas + 1].Valido &&
        puntos[coordenadaActual + NumLineas].Valido) {

        poligonos.Add(new Poligono(
          coordenadaActual,
          coordenadaActual + 1,
          coordenadaActual + NumLineas + 1,
          coordenadaActual + NumLineas
        ));
      }

      coordenadaActual++;

      // Salta al inicio de la siguiente fila si se alcanza el final de la actual
      if (coordenadaActual == NumLineas * filaActual - 1) {
        coordenadaActual++;
        filaActual++;
      }
    }

    //Para la matriz de rotación
    double CosX = Math.Cos(angX);
    double SinX = Math.Sin(angX);
    double CosY = Math.Cos(angY);
    double SinY = Math.Sin(angY);
    double CosZ = Math.Cos(angZ);
    double SinZ = Math.Sin(angZ);

    //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 }
    };

    //Las constantes de transformación para cuadrar en pantalla
    double convierteX = (XpantallaFin - XpantallaIni) / 1.758630875383556;
    double convierteY = (YpantallaFin - YpantallaIni) / 1.758630875383556;

    //Gira los puntos
    for (int cont = 0; cont < puntos.Count; cont++)
      puntos[cont].Giro(Matriz, ZPersona, XpantallaIni, YpantallaIni, convierteX, convierteY);

    //Calcula la profundidad y forma el polígono
    for (int Cont = 0; Cont < poligonos.Count; Cont++)
      poligonos[Cont].ProfundidadFigura(puntos, ListaColores);

    //Algoritmo de pintor.
    //Ordena del polígono más alejado al más cercano,
    //los polígonos de adelante son visibles y los de atrás son borrados.
    poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
  }

  //Dibuja la figura 3D
  public void Dibuja(Graphics lienzo) {
    for (int Cont = 0; Cont < poligonos.Count; Cont++)
      poligonos[Cont].Dibuja(lienzo);
    }
  }
}
```

```
namespace Graficos {
  public partial class Valores : Form {

    Form1 FrmGrafico = new();
    const double Radianes = Math.PI / 180;
    public Valores() {
      InitializeComponent();

      FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
      FrmGrafico.Ecuacion = txtEcuacion.Text;
      FrmGrafico.ZPersona = 5;
      FrmGrafico.HuboCambio = true;

      FrmGrafico.Show();
    }

    private void numGiroX_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numMinimoX_ValueChanged(object sender, EventArgs e) {

    }
```

```
    private void numMinimoY_ValueChanged(object sender, EventArgs e) {

    }

    private void numMaximoX_ValueChanged(object sender, EventArgs e) {
    }

    private void numMaximoY_ValueChanged(object sender, EventArgs e) {
    }

    private void numTotalLineas_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void btnProcesar_Click(object sender, EventArgs e) {
      FrmGrafico.Ecuacion = txtEcuacion.Text;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }
  }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D y giros. Figura centrada. Optimización.
  public partial class Form1 : Form {
    //La figura que se proyecta y gira
    Objeto3D Figura3D;

    //Giro de figura
    public double AnguloX, AnguloY, AnguloZ;

    //Distancia del observador
    public double ZPersona;

    //Número de líneas que tendrá el gráfico
    public int NumLineas;

    //Ecuación
    public string Ecuacion;

    //En caso de que el usuario haya hecho algún cambio a los valores
    //se recalcula todo de nuevo.
    public bool HuboCambio;

    public Form1() {
      InitializeComponent();
      Figura3D = new Objeto3D();
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {

      //Dibuja la figura 3D
      if (HuboCambio) {
        Figura3D.CalcularFigura3D(Ecuacion, NumLineas, AnguloX, AnguloY, AnguloZ, ZPersona, 0, 0,
this.ClientSize.Width, this.ClientSize.Height);
        HuboCambio = false;
      }

      Figura3D.Dibuja(e.Graphics);
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
      Application.Exit();
    }

    private void Form1_Resize(object sender, EventArgs e) {
      HuboCambio = true;
      Refresh();
    }
  }
}
```

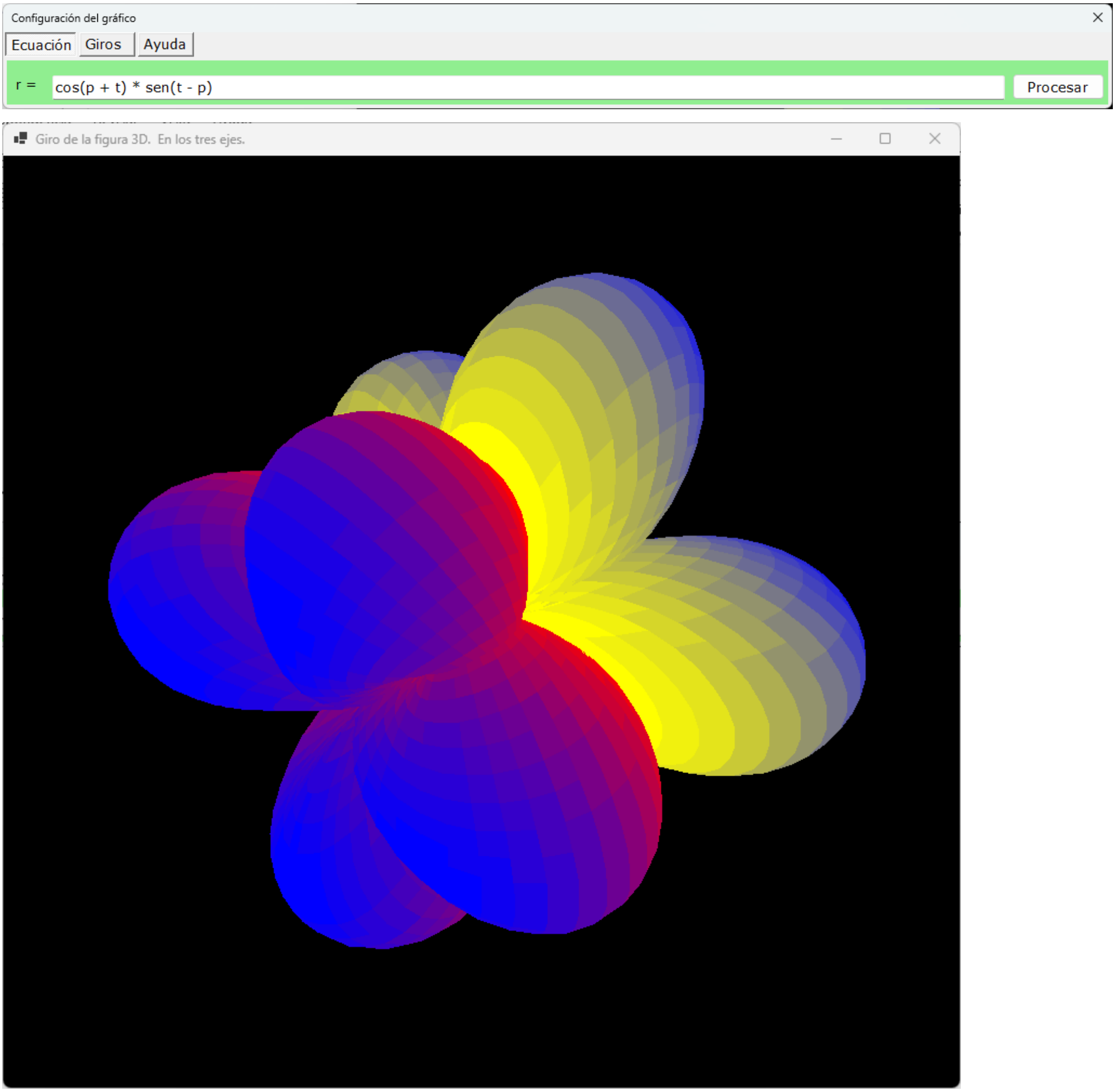


Ilustración 27: Gráfico polar 3D

Gráfico de sólido de revolución

Un sólido de revolución puede nacer de tomar una ecuación del tipo Y=F(X), y luego poner a girar el gráfico resultante en el eje X. Ver más en: https://es.wikipedia.org/wiki/S%C3%B3lido_de_revoluci%C3%B3n

Los pasos para hacer el gráfico son los siguientes:

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

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

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

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

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

M/014.zip

```
namespace Graficos {
  //Cada punto espacial es almacenado y convertido
  internal class Punto {
    public bool Valido; //Si el punto es válido
    public double X, Y, Z; //Coordenadas originales
    public double Zgiro; //Al girar los puntos
    public int Xpantalla, Ypantalla; //Puntos en pantalla

    public Punto(double X, double Y, double Z) {
      this.X = X;
      this.Y = Y;
      Valido = true;
      if (double.IsFinite(Z)) {
        this.Z = Z;
      }
      else {
        Valido = false;
      }
    }

    //Giro en los tres ejes
    public void Giro(double[,] Matriz, double ZPersona, int XpantallaIni, int YpantallaIni, double
convierteX, double convierteY) {
      //Si el punto no es válido, no hace nada
      if (Valido == false) return;

      //Hace el giro
      double Xgiro = X * Matriz[0, 0] + Y * Matriz[1, 0] + Z * Matriz[2, 0];
      double Ygiro = X * Matriz[0, 1] + Y * Matriz[1, 1] + Z * Matriz[2, 1];
      Zgiro = X * Matriz[0, 2] + Y * Matriz[1, 2] + Z * Matriz[2, 2];

      //Convierte de 3D a 2D (segunda dimensión)
      double PlanoX = Xgiro * ZPersona / (ZPersona - Zgiro);
      double PlanoY = Ygiro * ZPersona / (ZPersona - Zgiro);

      //Deduce las coordenadadas de pantalla
      Xpantalla = Convert.ToInt32(convierteX * (PlanoX + 0.879315437691778) + XpantallaIni);
      Ypantalla = Convert.ToInt32(convierteY * (PlanoY + 0.879315437691778) + YpantallaIni);
    }
  }
}
```

```
namespace Graficos {
  //Conecta con una línea recta un punto con otro
```

```
  internal class Conexion {
    public int punto1, punto2;

    public Conexion(int punto1, int punto2) {
      this.punto1 = punto1;
      this.punto2 = punto2;
    }
  }
}
```

```
namespace Graficos {
  //Polígono
  internal class Poligono {
    public int punto1, punto2, punto3, punto4;
    public double Centro;

    //Coordenadas de dibujo del polígono
    private Point Polig1, Polig2, Polig3, Polig4;
    private Point[] ListaPuntos;

    //Color de relleno del polígono
    Color ColorZ;

    public Poligono(int punto1, int punto2, int punto3, int punto4) {
      this.punto1 = punto1;
      this.punto2 = punto2;
      this.punto3 = punto3;
      this.punto4 = punto4;
    }

    //Calcula la profundidad y crea los vértices del polígono
    public void ProfundidadFigura(List<Punto> puntos, List<Color> colorList) {
      double Z1 = puntos[punto1].Zgiro;
      double Z2 = puntos[punto2].Zgiro;
      double Z3 = puntos[punto3].Zgiro;
      double Z4 = puntos[punto4].Zgiro;
      Centro = (Z1 + Z2 + Z3 + Z4) / 4;

      Polig1 = new(puntos[punto1].Xpantalla, puntos[punto1].Ypantalla);
      Polig2 = new(puntos[punto2].Xpantalla, puntos[punto2].Ypantalla);
      Polig3 = new(puntos[punto3].Xpantalla, puntos[punto3].Ypantalla);
      Polig4 = new(puntos[punto4].Xpantalla, puntos[punto4].Ypantalla);
      ListaPuntos = [Polig1, Polig2, Polig3, Polig4];

      int TotalColores = colorList.Count;
      double PromedioZ = (puntos[punto1].Z + puntos[punto2].Z + puntos[punto3].Z + puntos[punto4].Z + 2) /
4;
      int ColorEscoge = (int)Math.Floor((TotalColores-1) * PromedioZ);
      ColorZ = colorList[ColorEscoge];
    }

    //Hace el gráfico del polígono
    public void Dibuja(Graphics Lienzo) {
      //Dibuja el polígono relleno
      Brush Relleno2 = new SolidBrush(ColorZ);
      Lienzo.FillPolygon(Relleno2, ListaPuntos);
    }
  }
}
```

```
namespace Graficos {
  internal class Objeto3D {
    //Coordenadas espaciales X, Y, Z
    private List<Punto> puntos;

    //Coordenadas del polígono (triángulo)
    private List<Poligono> poligonos;

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

```
//Ecuación
string EcuacionAnterior = "";
Evaluador4 Evaluador = new();

public Objeto3D() {
  int NumColores = 40; // Número de colores a generar

  //Genera listado de colores para el gráfico
  ListaColores = [];
  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)));
    ListaColores.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)));
    ListaColores.Add(Color.FromArgb(Rojo, Verde, Azul));
  }
 }

 //Hace los cálculos de la ecuación Z = F(X,Y)
 //para dibujarla
 public void CalcularFigura3D(string Ecuacion, double Xini, double Xfin, int NumLineas, double angX,
double angY, double angZ, double ZPersona, int XpantallaIni, int YpantallaIni, int XpantallaFin, int
YpantallaFin) {

   //Evalúa la ecuación. Si es nueva, la analiza.
   if (Ecuacion.Equals(EcuacionAnterior) == false) {
     int Sintaxis = Evaluador.Analizar(Ecuacion);
     if (Sintaxis > 0) { //Tiene un error de sintaxis
       string MensajeError = Evaluador.MensajeError(Sintaxis);
       MessageBox.Show(MensajeError, "Error de sintaxis",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
       return;
     }
   }
   EcuacionAnterior = Ecuacion;

   //Incrementos X, Y
   double IncrX = (Xfin - Xini) / (NumLineas - 1);
   double IncAng = (double) 360 / NumLineas;
   double X = Xini;
   double AnguloGiro = 0;

   //Coordenadas espaciales X,Y,Z
   X = Xini;
   puntos = [];
   AnguloGiro = 0;
   for (int GiroLinea = 1; GiroLinea <= NumLineas+1; GiroLinea++) {
     for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
       Evaluador.DarValorVariable('x', X);
       double Y = Evaluador.Evaluar();

       double Yg = Y * Math.Cos(AnguloGiro * Math.PI / 180);
       double Zg = Y * Math.Sin(AnguloGiro * Math.PI / 180);

       puntos.Add(new Punto(X, Yg, Zg));
       X += IncrX;
     }
     AnguloGiro += IncAng;
     X = Xini;
   }

   //Debe normalizar las coordenadas y ponerlas entre -0.5 y 0.5
   double Xmin = double.MaxValue;
   double Ymin = double.MaxValue;
   double Zmin = double.MaxValue;
   double Xmax = double.MinValue;
```

```
    double Ymax = double.MinValue;
    double Zmax = double.MinValue;
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      if (puntos[Cont].X < Xmin) Xmin = puntos[Cont].X;
      if (puntos[Cont].Y < Ymin) Ymin = puntos[Cont].Y;
      if (puntos[Cont].Z < Zmin) Zmin = puntos[Cont].Z;
      if (puntos[Cont].X > Xmax) Xmax = puntos[Cont].X;
      if (puntos[Cont].Y > Ymax) Ymax = puntos[Cont].Y;
      if (puntos[Cont].Z > Zmax) Zmax = puntos[Cont].Z;
    }

    if (Math.Abs(Xmin - Xmax) < 0.0001 || Math.Abs(Ymin - Ymax) < 0.0001 || Math.Abs(Zmin - Zmax) <
0.0001) {
      MessageBox.Show("La ecuación digitada no puede generar un gráfico", "Error de cálculo",
 MessageBoxButtons.OK, MessageBoxIcon.Error);
      return;
    }

    //Normaliza
    for (int Cont = 0; Cont < puntos.Count; Cont++) {
      if (puntos[Cont].Valido == false) continue;
      puntos[Cont].X = (puntos[Cont].X - Xmin) / (Xmax - Xmin) - 0.5;
      puntos[Cont].Y = (puntos[Cont].Y - Ymin) / (Ymax - Ymin) - 0.5;
      puntos[Cont].Z = (puntos[Cont].Z - Zmin) / (Zmax - Zmin) - 0.5;
    }

    // Inicializa la lista de polígonos
    poligonos = [];

    int coordenadaActual = 0;
    int filaActual = 1;
    int totalPoligonos = (NumLineas - 1) * (NumLineas - 1) + (NumLineas - 1);

    for (int Cont = 0; Cont < totalPoligonos; Cont++) {
      // Crea un polígono con las coordenadas de los vértices
      // siempre y cuando los vértices sean válidos
      if (puntos[coordenadaActual].Valido &&
        puntos[coordenadaActual + 1].Valido &&
        puntos[coordenadaActual + NumLineas + 1].Valido &&
        puntos[coordenadaActual + NumLineas].Valido) {

        poligonos.Add(new Poligono(
          coordenadaActual,
          coordenadaActual + 1,
          coordenadaActual + NumLineas + 1,
          coordenadaActual + NumLineas
        ));
      }

      coordenadaActual++;

      // Salta al inicio de la siguiente fila si se alcanza el final de la actual
      if (coordenadaActual == NumLineas * filaActual - 1) {
        coordenadaActual++;
        filaActual++;
      }
    }

    //Para la matriz de rotación
    double CosX = Math.Cos(angX);
    double SinX = Math.Sin(angX);
    double CosY = Math.Cos(angY);
    double SinY = Math.Sin(angY);
    double CosZ = Math.Cos(angZ);
    double SinZ = Math.Sin(angZ);

    //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 }
    };

    //Las constantes de transformación para cuadrar en pantalla
```

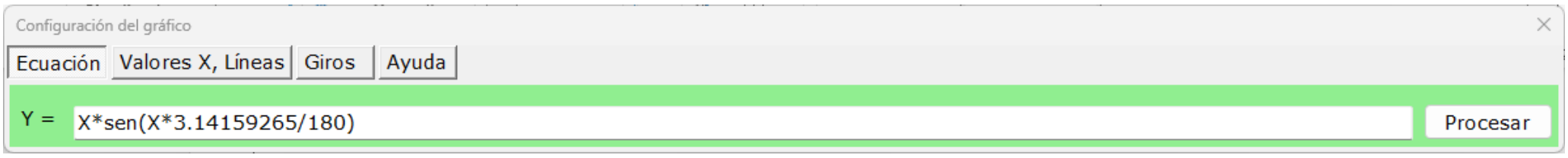
```
      double convierteX = (XpantallaFin - XpantallaIni) / 1.758630875383556;
      double convierteY = (YpantallaFin - YpantallaIni) / 1.758630875383556;

      //Gira los puntos
      for (int cont = 0; cont < puntos.Count; cont++)
        puntos[cont].Giro(Matriz, ZPersona, XpantallaIni, YpantallaIni, convierteX, convierteY);

      //Calcula la profundidad y forma el polígono
      for (int Cont = 0; Cont < poligonos.Count; Cont++)
        poligonos[Cont].ProfundidadFigura(puntos, ListaColores);

      //Algoritmo de pintor.
      //Ordena del polígono más alejado al más cercano,
      //los polígonos de adelante son visibles y los de atrás son borrados.
      poligonos.Sort((p1, p2) => p1.Centro.CompareTo(p2.Centro));
    }

    //Dibuja la figura 3D
    public void Dibuja(Graphics lienzo) {
      for (int Cont = 0; Cont < poligonos.Count; Cont++)
        poligonos[Cont].Dibuja(lienzo);
    }
  }
}
```

```
namespace Graficos {
  public partial class Valores : Form {

    Form1 FrmGrafico = new();
    const double Radianes = Math.PI / 180;
    public Valores() {
      InitializeComponent();

      FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
      FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
      FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
      FrmGrafico.Ecuacion = txtEcuacion.Text;
      FrmGrafico.ZPersona = 5;
      FrmGrafico.HuboCambio = true;

      FrmGrafico.Show();
    }

    private void numGiroX_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numMinimoX_ValueChanged(object sender, EventArgs e) {
      if (numMinimoX.Value >= numMaximoX.Value)
        numMinimoX.Value = numMaximoX.Value - 1;

      FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }
```

```
    private void numMaximoX_ValueChanged(object sender, EventArgs e) {
      if (numMinimoX.Value >= numMaximoX.Value)
        numMaximoX.Value = numMinimoX.Value + 1;

      FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void numTotalLineas_ValueChanged(object sender, EventArgs e) {
      FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }

    private void btnProcesar_Click(object sender, EventArgs e) {
      FrmGrafico.Ecuacion = txtEcuacion.Text;
      FrmGrafico.HuboCambio = true;
      FrmGrafico.Refresh();
    }
  }
}
```

```
namespace Graficos {
  //Proyección 3D a 2D y giros. Figura centrada. Optimización.
  public partial class Form1 : Form {
    //La figura que se proyecta y gira
    Objeto3D Figura3D;

    //Giro de figura
    public double AnguloX, AnguloY, AnguloZ;

    //Distancia del observador
    public double ZPersona;

    //Rango de valores
    public double Xini, Yini, Xfin, Yfin;

    //Número de líneas que tendrá el gráfico
    public int NumLineas;

    //Ecuación
    public string Ecuacion;

    //En caso de que el usuario haya hecho algún cambio a los valores
    //se recalcula todo de nuevo.
    public bool HuboCambio;

    public Form1() {
      InitializeComponent();
      Figura3D = new Objeto3D();
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {

      //Dibuja la figura 3D
      if (HuboCambio) {
        Figura3D.CalcularFigura3D(Ecuacion, Xini, Xfin, NumLineas, AnguloX, AnguloY, AnguloZ, ZPersona, 0,
0, this.ClientSize.Width, this.ClientSize.Height);
        HuboCambio = false;
      }

      Figura3D.Dibuja(e.Graphics);
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
      Application.Exit();
    }

    private void Form1_Resize(object sender, EventArgs e) {
      HuboCambio = true;
      Refresh();
    }
```

```
    }
}
```



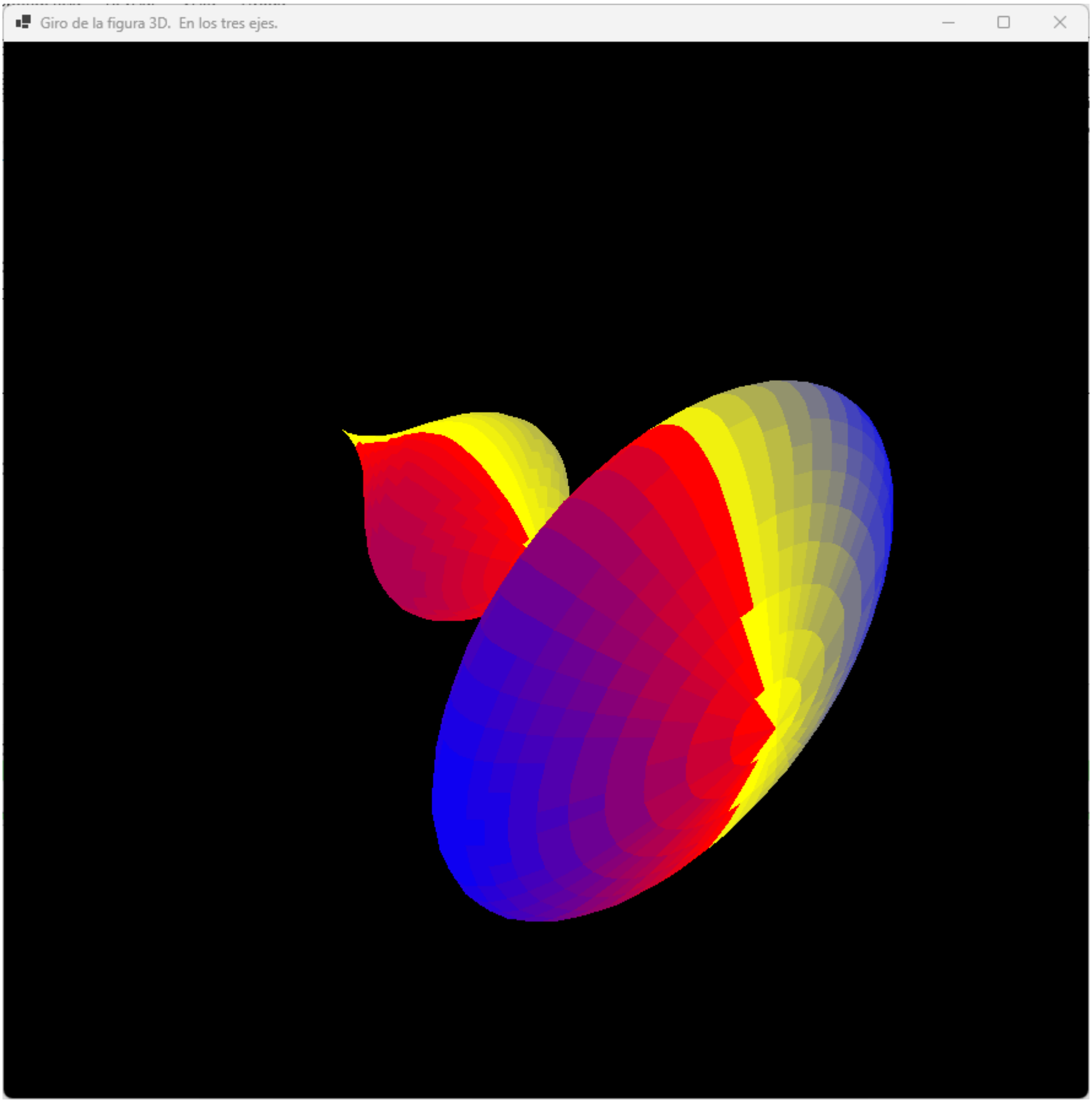


Ilustración 28: Gráfico de sólido de revolución