C# Y .NET 8
Parte 14. Animación

2024-12

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 2022 ® Enlace: https://visualstudio.microsoft.com/es/vs/

Dedicatoria

A mis padres, a mi hermana….

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

Haciendo uso del Paint()

C# tiene un sistema fácil para hacer animaciones. Sin embargo, no es para hacer animaciones muy complejas

Es necesario agregar el control “Timer” al formulario

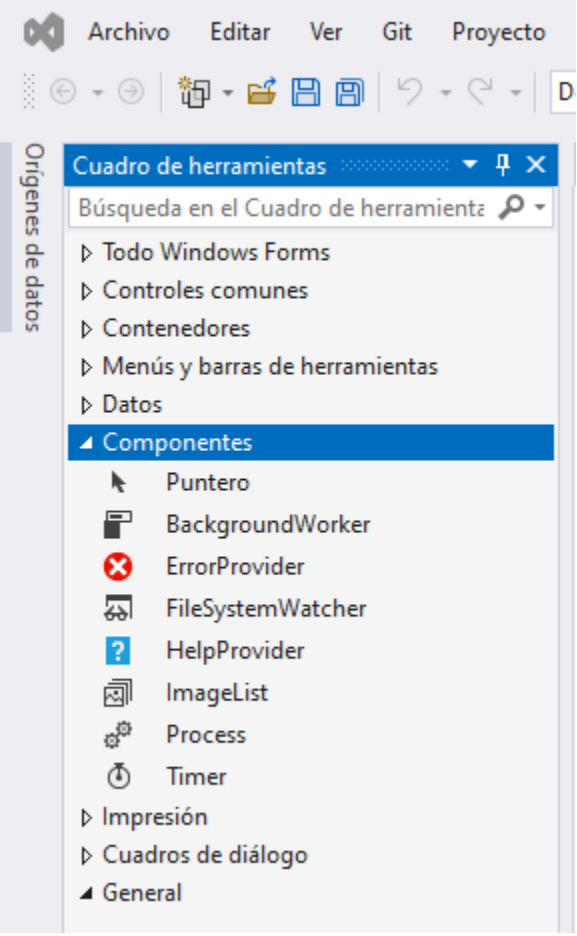

Ilustración 1: Control Timer

Se ajustan las propiedades de este control: (Name) y Enabled que debe estar en **True**

Autor: Rafael Alberto Moreno Parra

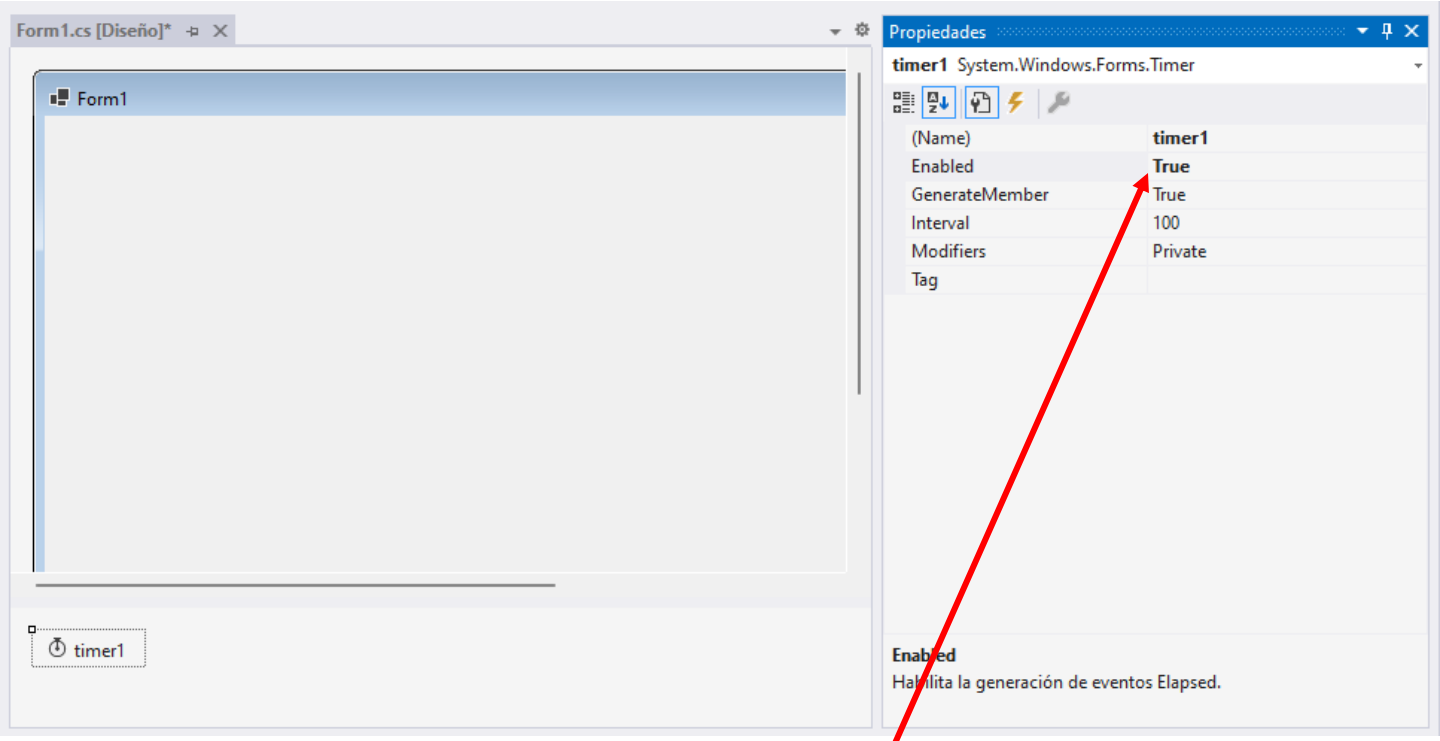

*Ilustración 2: Propiedades del control Timer. Enabled debe estar en **True***

¡OJO!: Hay que poner en **True** la propiedad DoubleBuffered del formulario

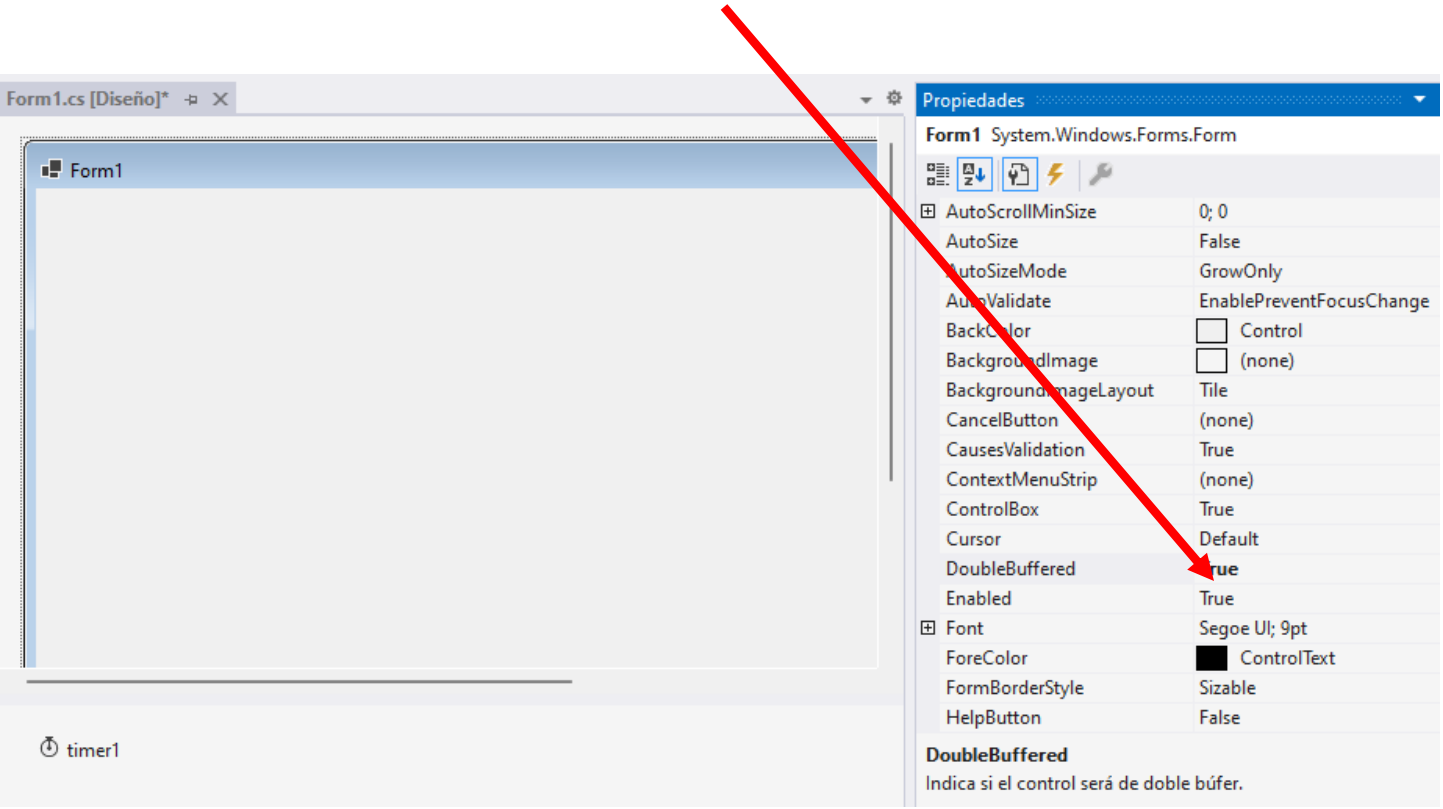

Ilustración 3: Propiedad DoubleBuffered del formulario en True

Ahora se habilita el evento Tick del Timer

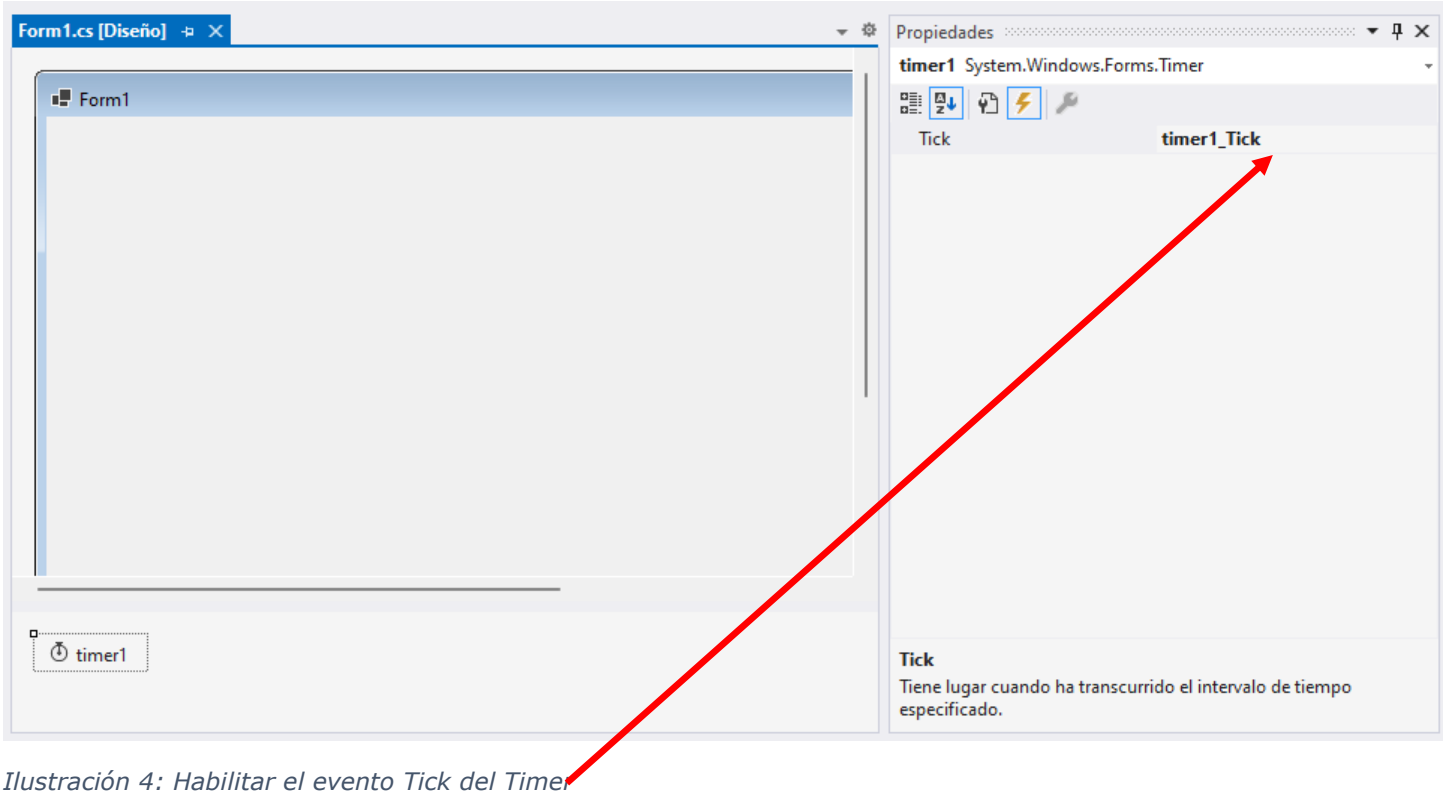

Ilustración 4: Habilitar el evento Tick del Timer

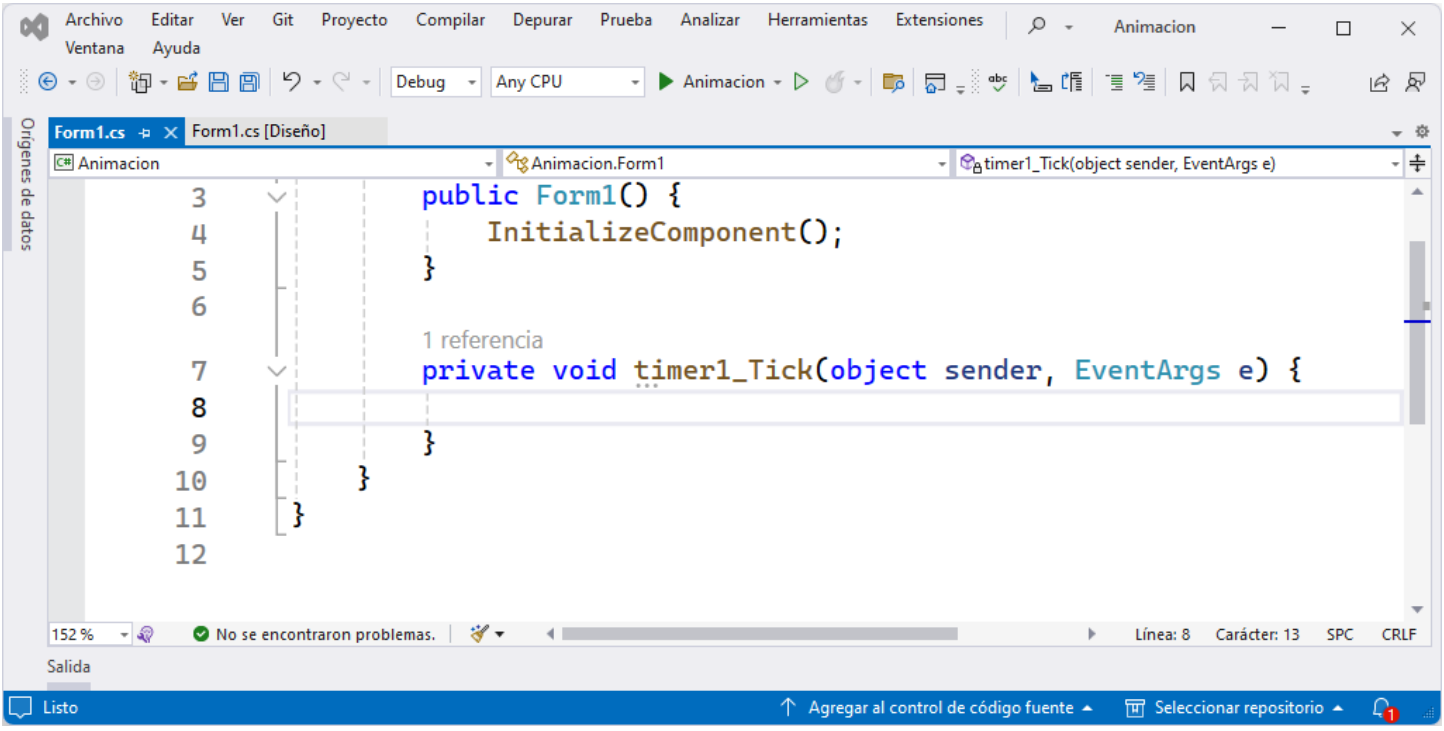

Ilustración 5: Evento para escribir el código

```
namespace Animacion {
   public partial class Form1 : Form {
      int PosX, PosY; //Coordenadas del cuadrado relleno
      public Form1() {
         InitializeComponent();

         //Inicializa las posiciones
         PosX = 10;
         PosY = 20;
      }

      private void timer1_Tick(object sender, EventArgs e) {
         //Por cada tick, incrementa en 10 el valor de
         //la posición X del cuadrado relleno
         PosX += 10;
         Refresh(); //Refresca el formulario y llama a Paint()
      }

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics Lienzo = e.Graphics;
         SolidBrush Relleno = new(Color.Chocolate);

         //================================
         //Rectángulo: Xpos, Ypos, ancho, alto
         //================================
         Lienzo.FillRectangle(Relleno, PosX, PosY, 40, 40);
      }
   }
}
```

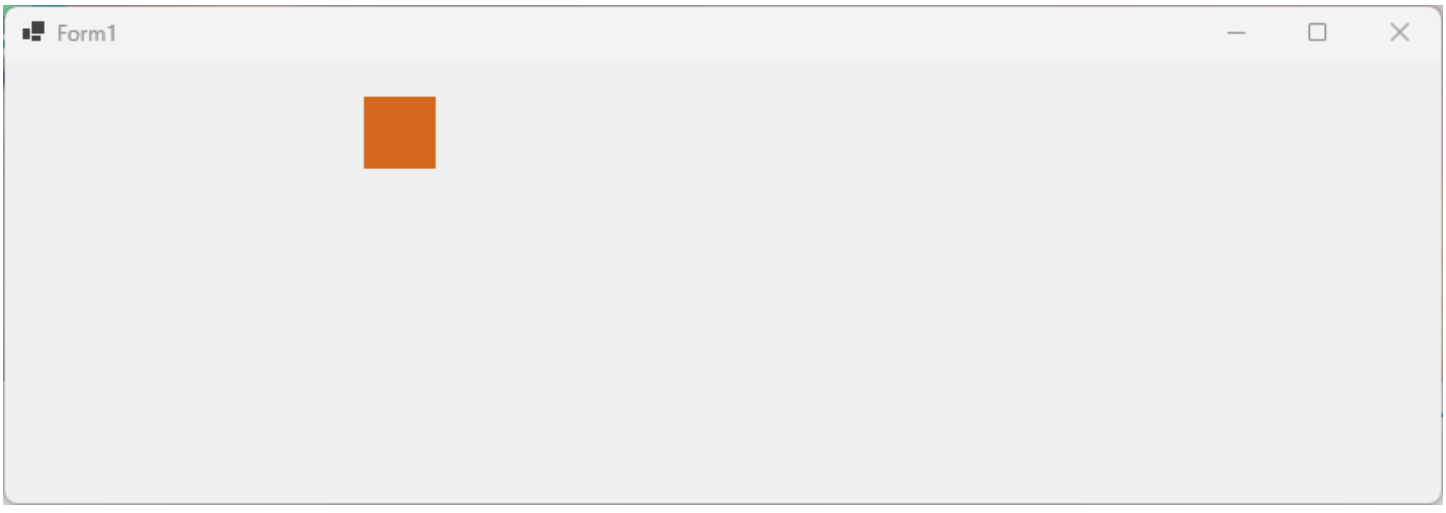

Ilustración 6: Cuadro desplazándose

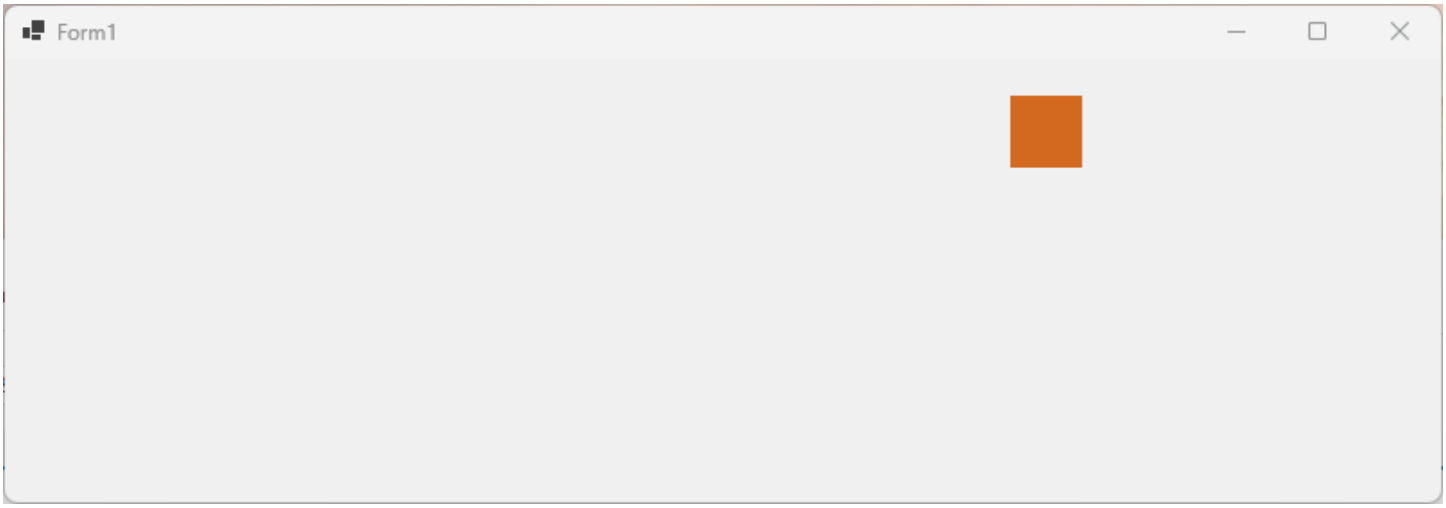

Ilustración 7: Cuadro desplazándose

¡OJO! La animación no será suave, es una limitante que tiene Winforms. En algunos foros recomiendan cambiar la instrucción Refresh(); por Invalidate(); . Ambas instrucciones redibujan, pero operan de distinta forma:

Invalidate() : Envía el mensaje al sistema operativo para que redibuje, pero no lo hace inmediatamente.

Refresh() : Fuerza a redibujar inmediatamente.

Rebotando en pantalla

El siguiente ejemplo, muestra un cuadrado relleno rebotando en las paredes de la ventana.

N/002.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      int PosX, PosY; //Coordenadas del cuadrado relleno
      int Tamano; //Tamaño del lado del cuadrado
      int IncrementoX, IncrementoY; //Desplazamiento del cuadrado relleno

      public Form1() {
         InitializeComponent();

         //Inicializa la posición y tamaño del cuadrado relleno
         PosX = 10;
         PosY = 20;
         Tamano = 40;

         //Velocidad con que se desplaza el cuadrado relleno
         IncrementoX = 5;
         IncrementoY = 5;
      }

      private void timer1_Tick(object sender, System.EventArgs e) {
         Logica(); //Lógica de la animación
         Refresh(); //Visual de la animación
      }

      void Logica() {
         //Si colisiona con alguna pared cambia el incremento
         if (PosX + Tamano > this.ClientSize.Width || PosX < 0)
            IncrementoX *= -1;

         if (PosY + Tamano > this.ClientSize.Height || PosY < 0)
            IncrementoY *= -1;

         //Cambia la posición de X y Y
         PosX += IncrementoX;
         PosY += IncrementoY;
      }

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


Autor: Rafael Alberto Moreno Parra

```
            //Fondo de la ventana
            Rectangle rect = new Rectangle(0, 0, this.Width, this.Height);
            Lienzo.FillRectangle(Brushes.Black, rect);

            //Gráfico a animar
            Lienzo.FillRectangle(Brushes.Red, PosX, PosY, Tamano, Tamano);
        }
    }
}
```

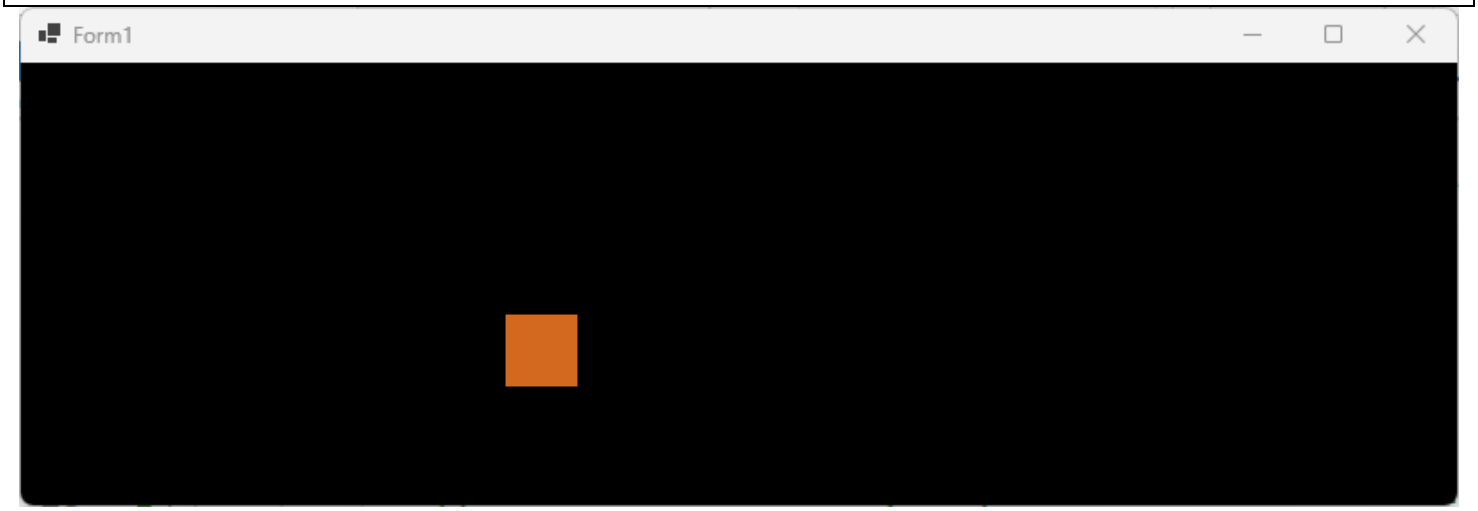

Ilustración 8: Rebote

Lógica y representación

Es muy recomendable que la parte lógica de la aplicación gráfica se concentre sobre un arreglo bidimensional y la parte visual es reflejar lo que sucede en ese arreglo bidimensional en pantalla.

Se reescribe el programa de rebote anterior, haciendo uso de un arreglo bidimensional.

N/003.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      int[,] Plano; //Dónde ocurre realmente la acción
      int PosX, PosY; //Coordenadas del cuadrado relleno
      int IncrX, IncrY; //Desplazamiento del cuadrado relleno

      public Form1() {
         InitializeComponent();

         //Inicializa el tablero
         Plano = new int[30, 30];

         //Inicializa la posición del cuadrado relleno
         Random azar = new();
         PosX = azar.Next(0, Plano.GetLength(0));
         PosY = azar.Next(0, Plano.GetLength(1));

         //Desplaza el cuadrado relleno
         IncrX = 1;
         IncrY = 1;
      }

      private void timer1_Tick(object sender, System.EventArgs e) {
         Logica(); //Lógica de la animación
         Refresh(); //Visual de la animación
      }

      void Logica() {
         //Borra la posición anterior
         Plano[PosX, PosY] = 0;

         //Si colisiona con alguna pared cambia el incremento
         if (PosX + IncrX >= Plano.GetLength(0) || PosX + IncrX < 0)
            IncrX *= -1;

         if (PosY + IncrY >= Plano.GetLength(1) || PosY + IncrY < 0)
            IncrY *= -1;
```

```
            //Cambia la posición de X y Y
            PosX += IncrX;
            PosY += IncrY;

            //Nueva posición
            Plano[PosX, PosY] = 1;
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen Lapiz = new(Color.Blue, 1);
            Brush Llena = new SolidBrush(Color.Red);

            //Tamaño de cada celda
            int tX = ClientSize.Width / Plano.GetLength(0);
            int tY = ClientSize.Height / Plano.GetLength(1);

            //Fondo de la ventana
            Rectangle rect = new(0, 0, this.Width, this.Height);
            lienzo.FillRectangle(Brushes.Black, rect);

            //Dibuja la malla y la posición del rectángulo relleno
            for (int Fil = 0; Fil < Plano.GetLength(0); Fil++) {
                for (int Col = 0; Col < Plano.GetLength(1); Col++)
                    if (Plano[Fil, Col] == 0)
                        lienzo.DrawRectangle(Lapiz, Fil * tX, Col * tY, tX, tY);
                    else
                        lienzo.FillRectangle(Llena, Fil * tX, Col * tY, tX, tY);
            }
        }
    }
}
```

Así ejecuta

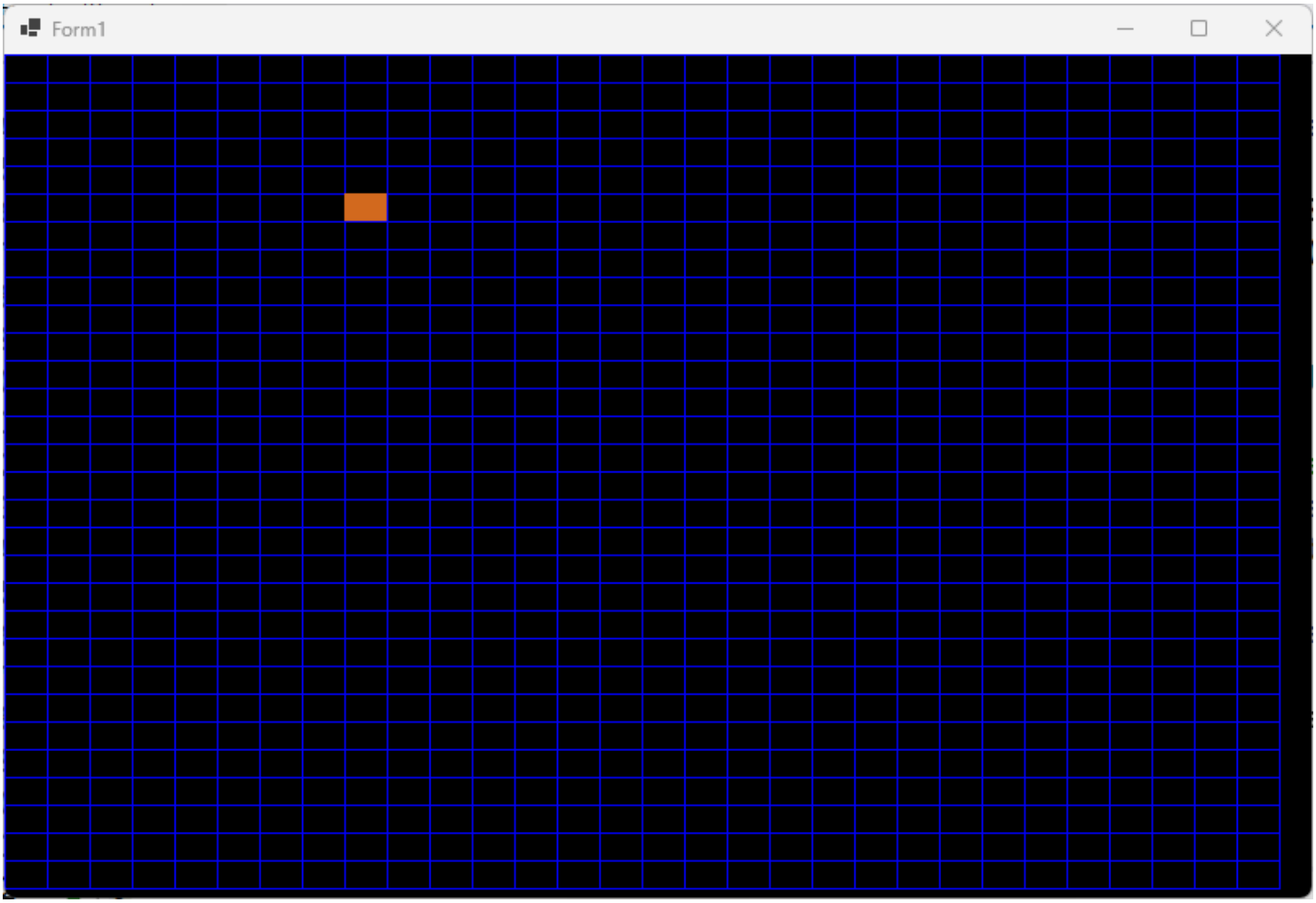

Ilustración 9: Rebote en arreglo bidimensional

Es mucho mejor así, porque hay pleno control del tamaño del tablero, cómo se desplaza, dónde rebota. En el código de ejemplo, si se cambia el tamaño de la ventana, el programa reacciona al nuevo tamaño.

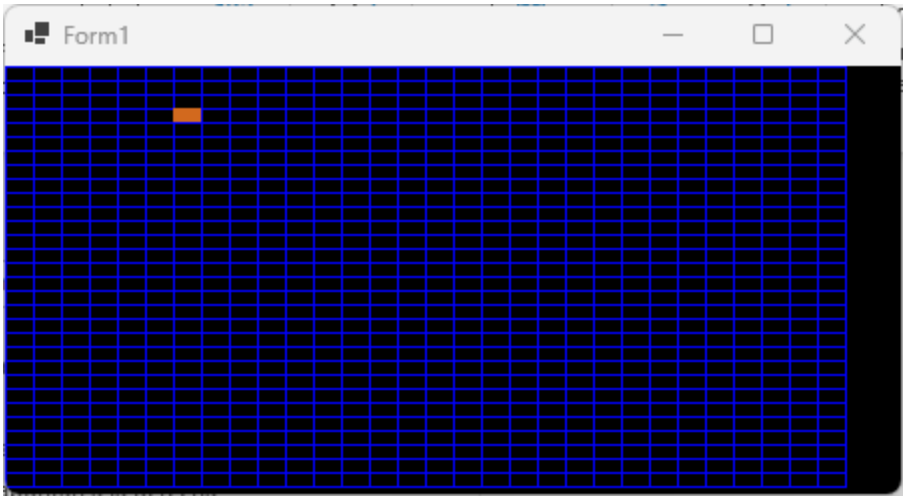

Ilustración 10: Cambiando el tamaño de la ventana

Cazador y Presa

Dado un arreglo bidimensional de 30*30, se ubica en una casilla (seleccionada al azar) un cazador y en otra casilla (seleccionada al azar) una presa. El objetivo es hacer que el cazador salte de casilla en casilla, este salto puede ser horizontal o vertical o diagonal hasta alcanzar la casilla de la presa. El tamaño del salto del depredador es de una casilla cada vez. La presa se mantiene inmóvil.

Se deben agregar obstáculos al azar a este tablero bidimensional (casillas en las cuales el cazador **NO** puede saltar). Y allí está el problema, porque el cazador se puede topar con una pared que le impide llegar a la presa. ¿Y qué se hace en ese caso? Una solución planteada es generar al azar alguna coordenada temporal e instruir al cazador para que olvide la presa y se dirija a esa ubicación temporal, una vez el cazador llega a esa ubicación temporal, de nuevo se le instruye para que se dirija hacia la presa con la esperanza que ahora si encuentre un camino libre hasta la presa.

N/004.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Constantes para calificar cada celda del Plano
      private const int CAMINO = 0;
      private const int OBSTACULO = 1;
      private const int CAZADOR = 2;
      private const int PRESA = 3;

      //Dónde ocurre realmente la acción
      int[,] Plano;

      //Coordenadas del cazador
      int CazaX, CazaY;

      //Desplazamiento del cazador
      int CazaMX, CazaMY;

      //Coordenadas de la presa
      int PresaX, PresaY;

      //Coordenadas temporales para desatorar
      int tmpX, tmpY;

      //Alterna entre buscar la presa real o
      //ir a la coordenada temporal
      bool BuscaTmp;
```


Autor: Rafael Alberto Moreno Parra

```
//Único generador de números aleatorios.
Random Azar;

public Form1() {
   InitializeComponent();
   Azar = new Random();
   IniciarParametros();
}

public void IniciarParametros() {
   //Inicializa el Plano.
   Plano = new int[30, 30];

   //Total de paredes dentro del plano
   int Obstaculos = 150;
   for (int cont = 1; cont <= Obstaculos; cont++) {
      int obstaculoX, obstaculoY;
      do {
         obstaculoX = Azar.Next(0, Plano.GetLength(0));
         obstaculoY = Azar.Next(0, Plano.GetLength(1));
      } while (Plano[obstaculoX, obstaculoY] != 0);
      Plano[obstaculoX, obstaculoY] = OBSTACULO;
   }

   //Inicializa la posición del cazador
   do {
      CazaX = Azar.Next(0, Plano.GetLength(0));
      CazaY = Azar.Next(0, Plano.GetLength(1));
   } while (Plano[CazaX, CazaY] != 0);
   Plano[CazaX, CazaY] = CAZADOR;

   //Inicializa la posición de la presa
   do {
      PresaX = Azar.Next(0, Plano.GetLength(0));
      PresaY = Azar.Next(0, Plano.GetLength(1));
   } while (Plano[PresaX, PresaY] != 0);
   Plano[PresaX, PresaY] = PRESA;

   //Desplazamiento del cazador
   CazaMX = 1;
   CazaMY = 1;

   timer1.Start();
}


private void timer1_Tick(object sender, System.EventArgs e) {
   Logica(); //Lógica de la animación
```

```
        Refresh(); //Visual de la animación
    }

    public void Logica() {
        Plano[CazaX, CazaY] = CAMINO;

        if (BuscaTmp == false) {
            //Esta buscando la presa
            if (CazaX > PresaX) CazaMX = -1;
            else if (CazaX < PresaX) CazaMX = 1;
            else CazaMX = 0;

            if (CazaY > PresaY) CazaMY = -1;
            else if (CazaY < PresaY) CazaMY = 1;
            else CazaMY = 0;

            //Verifica si ya llegó a la presa
            if (CazaX == PresaX && CazaY == PresaY) {
                timer1.Stop();
                MessageBox.Show("El cazador alcanzó a la presa");
                return;
            }
            //Si no, verifica si puede desplazarse a la nueva ubicación
            else if (Plano[CazaX + CazaMX, CazaY + CazaMY] == CAMINO ||
                  Plano[CazaX + CazaMX, CazaY + CazaMY] == PRESA) {
                CazaX += CazaMX;
                CazaY += CazaMY;
            }
            //Si no, entonces está atorado con los obstáculos.
            //Luego genera ubicación temporal para ir allí
            else {
                do {
                    tmpX = Azar.Next(0, Plano.GetLength(0));
                    tmpY = Azar.Next(0, Plano.GetLength(1));
                } while (Plano[tmpX, tmpY] != CAMINO);
                BuscaTmp = true;
            }
        }
        else { //Está yendo a la ubicación temporal
            if (CazaX > tmpX) CazaMX = -1;
            else if (CazaX < tmpX) CazaMX = 1;
            else CazaMX = 0;

            if (CazaY > tmpY) CazaMY = -1;
            else if (CazaY < tmpY) CazaMY = 1;
            else CazaMY = 0;

            //Si ha llegado a la ubicación temporal o se queda atorado
```

```
            //deja de ir a esa ubicación temporal
            if (CazaX == tmpX && CazaY == tmpY ||
               Plano[CazaX + CazaMX, CazaY + CazaMY] == OBSTACULO)
               BuscaTmp = false;
            else {
               CazaX += CazaMX;
               CazaY += CazaMY;
            }
         }

         Plano[CazaX, CazaY] = CAZADOR;
      }

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics lienzo = e.Graphics;
         Pen Lapiz = new(Color.Blue, 1);
         Brush Llena1 = new SolidBrush(Color.Black);
         Brush Llena2 = new SolidBrush(Color.Red);
         Brush Llena3 = new SolidBrush(Color.Blue);

         //Tamaño de cada celda
         int tX = 500 / Plano.GetLength(0);
         int tY = 500 / Plano.GetLength(1);
         int desplaza = 50;

         //Dibuja el arreglo bidimensional
         for (int Fil = 0; Fil < Plano.GetLength(0); Fil++) {
            for (int Col = 0; Col < Plano.GetLength(1); Col++) {
               int uX = Fil * tX + desplaza;
               int uY = Col * tY + desplaza;
               switch (Plano[Fil, Col]) {
                  case CAMINO:
                     lienzo.DrawRectangle(Lapiz, uX, uY, tX, tY);
                     break;
                  case OBSTACULO:
                     lienzo.FillRectangle(Llena1, uX, uY, tX, tY);
                     break;
                  case CAZADOR:
                     lienzo.FillRectangle(Llena2, uX, uY, tX, tY);
                     break;
                  case PRESA:
                     lienzo.FillRectangle(Llena3, uX, uY, tX, tY);
                     break;
               }
            }
         }
      }
   }
```

```
}
```

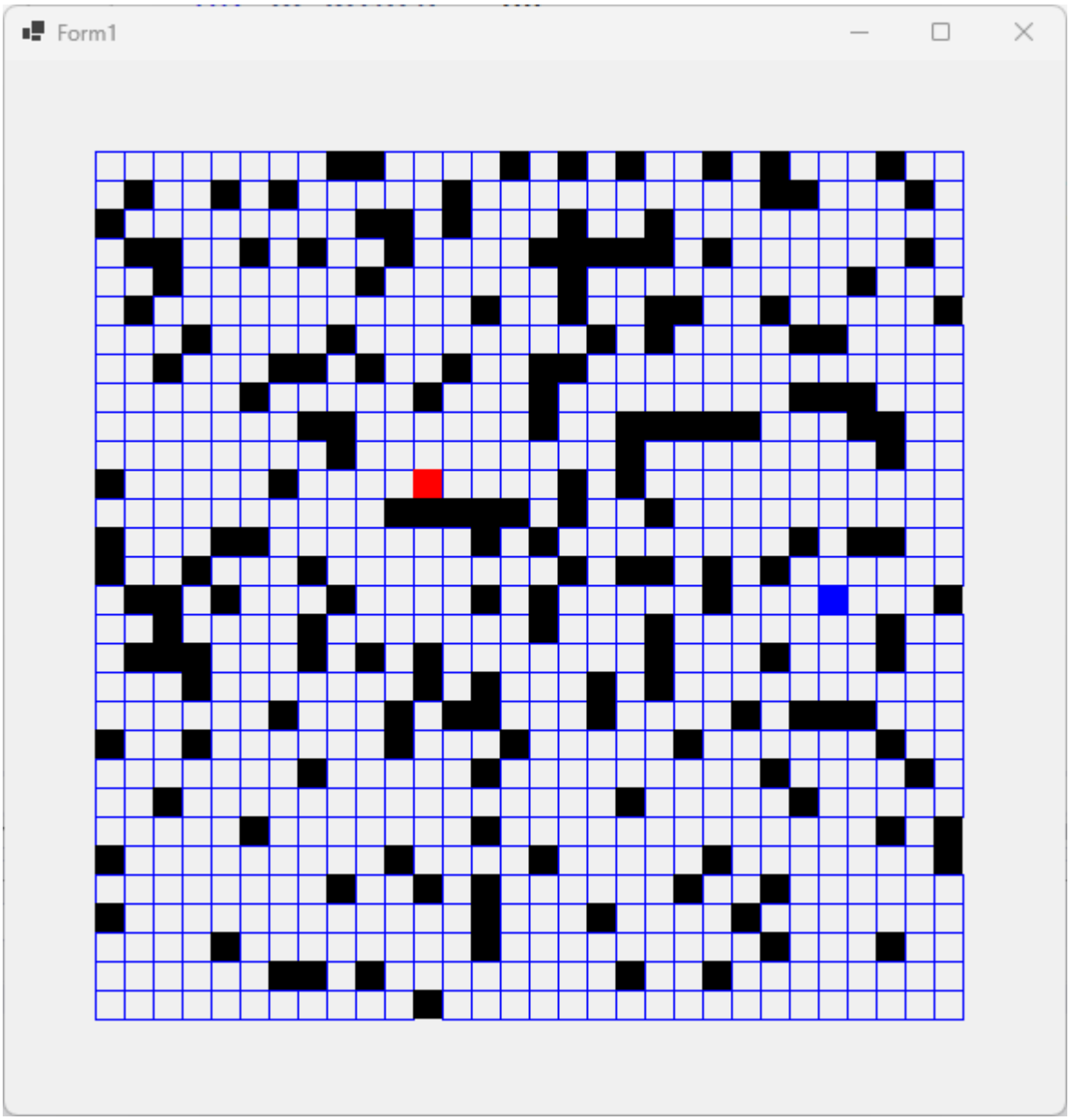

Ilustración 11: Depredador y presa

¡OJO! El algoritmo de búsqueda de la presa es bastante ineficiente en su cometido: El cazador puede tardar un tiempo considerable en encontrar la presa, además este algoritmo no valida que exista un camino viable por lo que se puede quedar operando en forma infinita buscando como llegar a la presa.

El juego de la vida

Este juego https://es.wikipedia.org/wiki/Juego_de_la_vida inventado por John Horton Conway, ha llamado mucho la atención por como dada una serie de unas reglas muy sencillas, genera patrones complejos.

N/005.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Constantes para calificar cada celda del tablero
      private const int INACTIVA = 0;
      private const int ACTIVA = 1;

      int[,] Plano; //Dónde ocurre realmente la acción
      Random Azar; //Único generador de números aleatorios.

      public Form1() {
         InitializeComponent();
         Azar = new Random();
         IniciarParametros();
      }

      private void mnuReiniciar_Click(object sender, EventArgs e) {
         IniciarParametros();
      }

      public void IniciarParametros() {
         //Inicializa el tablero.
         Plano = new int[40, 40];

         //Habrán celdas activas en el 20%
         int Activos = 200;
         for (int cont = 1; cont <= Activos; cont++) {
            int posX, posY;
            do {
               posX = Azar.Next(0, Plano.GetLength(0));
               posY = Azar.Next(0, Plano.GetLength(1));
            } while (Plano[posX, posY] != 0);
            Plano[posX, posY] = ACTIVA;
         }

         timer1.Start();
      }
```


Autor: Rafael Alberto Moreno Parra

```
private void timer1_Tick(object sender, System.EventArgs e) {
   Logica(); //Lógica de la animación
   Refresh(); //Visual de la animación
}

public void Logica() {
   //Copia el tablero a uno temporal
   int[,] PlanoTmp = Plano.Clone() as int[,];

   //Va de celda en celda
   for (int posX = 0; posX < Plano.GetLength(0); posX++) {
      for (int posY = 0; posY < Plano.GetLength(1); posY++) {

         //Determina el número de vecinos activos
         int BordeXizq;
         if (posX - 1 < 0)
            BordeXizq = Plano.GetLength(0) - 1;
         else
            BordeXizq = posX - 1;

         int BordeYarr;
         if (posY - 1 < 0)
            BordeYarr = Plano.GetLength(1) - 1;
         else
            BordeYarr = posY - 1;

         int BordeXder;
         if (posX + 1 >= Plano.GetLength(0))
            BordeXder = 0;
         else
            BordeXder = posX + 1;


         int BordeYaba;
         if (posY + 1 >= Plano.GetLength(1))
            BordeYaba = 0;
         else
            BordeYaba = posY + 1;

         int Activos = Plano[BordeXizq, BordeYarr];
         Activos += Plano[posX, BordeYarr];
         Activos += Plano[BordeXder, BordeYarr];

         Activos += Plano[BordeXizq, posY];
         Activos += Plano[BordeXder, posY];

         Activos += Plano[BordeXizq, BordeYaba];
         Activos += Plano[posX, BordeYaba];
```

```
            Activos += Plano[BordeXder, BordeYaba];

            //Los cambios se registran en el tablero temporal

            //Si la celda está inactiva y tiene 3 celdas activas
            //alrededor, entonces la celda se activa
            if (Plano[posX, posY] == INACTIVA && Activos == 3)
               PlanoTmp[posX, posY] = ACTIVA;

            //Si la celda está activa y tiene menos de 2 celdas o
            //más de 3 celdas, entonces la celda se inactiva
            if (Plano[posX, posY] == ACTIVA &&
               (Activos < 2 || Activos > 3))
               PlanoTmp[posX, posY] = INACTIVA;
         }
      }

      //El cambio en el tablero temporal se copia sobre el tablero
      Plano = PlanoTmp.Clone() as int[,];
   }

   private void Form1_Paint(object sender, PaintEventArgs e) {
      Graphics lienzo = e.Graphics;
      Pen Lapiz = new(Color.Blue, 1);
      Brush Llena = new SolidBrush(Color.Black);

      //Tamaño de cada celda
      int tX = 600 / Plano.GetLength(0);
      int tY = 600 / Plano.GetLength(1);
      int desplaza = 50;

      //Dibuja el arreglo bidimensional
      for (int Fil = 0; Fil < Plano.GetLength(0); Fil++) {
         for (int Col = 0; Col < Plano.GetLength(1); Col++) {
            int uX = Fil * tX + desplaza;
            int uY = Col * tY + desplaza;
            switch (Plano[Fil, Col]) {
               case INACTIVA:
                  lienzo.DrawRectangle(Lapiz, uX, uY, tX, tY);
                  break;
               case ACTIVA:
                  lienzo.FillRectangle(Llena, uX, uY, tX, tY);
                  break;
            }
         }
      }
   }
}
```

```
}
```

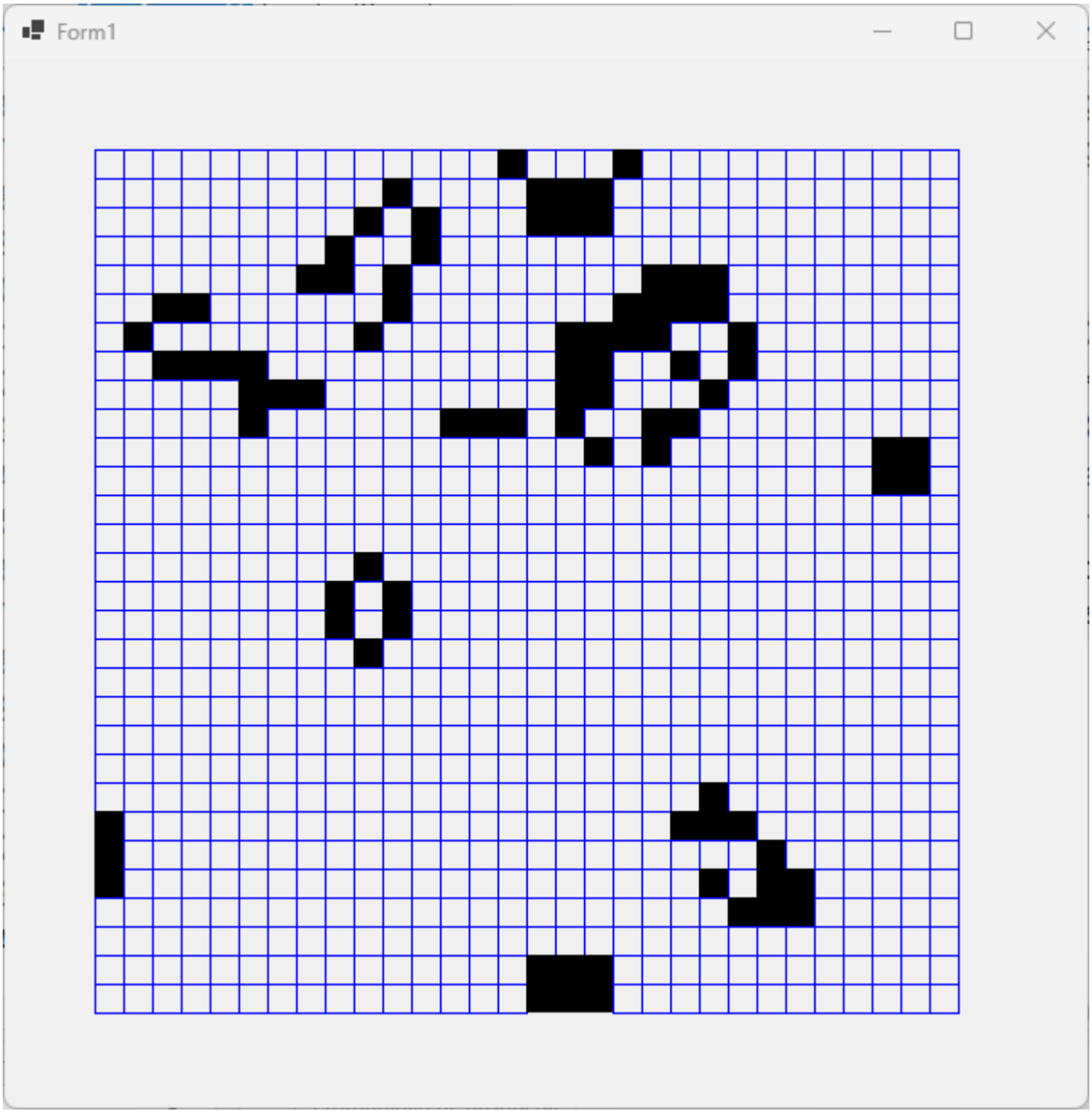

Ilustración 12: Juego de la vida

Población e infección

Simula una población de individuos en la cual uno tiene una infección y como esta se va esparciendo por la población. Los individuos se mueven al azar y si un infectado (en rojo) coincide en la misma casilla que uno sano (en negro), el individuo sano se infecta.

N/006.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Constantes para calificar cada celda del tablero
      private const int VACIO = 0;
      private const int SANO = 1;
      private const int INFECTADO = 2;

      int[,] Plano; //Dónde ocurre realmente la acción

      Random azar;

      //Tamaño de cada celda
      int tamF, tamC, desplaza;

      //Población
      List<Individuo> Pobl;

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

      public void IniciarParametros() {
         Plano = new int[30, 30]; //Inicializa el tablero.
         azar = new Random();

         //Inicializa la población
         Pobl = new List<Individuo>();
         int NumIndividuos = 100;
         for (int cont = 1; cont <= NumIndividuos; cont++) {
            int Fila, Columna;
            do {
               Fila = azar.Next(Plano.GetLength(0));
               Columna = azar.Next(Plano.GetLength(1));
            } while (Plano[Fila, Columna] != VACIO);
            Plano[Fila, Columna] = SANO;
            Pobl.Add(new Individuo(Fila, Columna, SANO));
         }
```


Autor: Rafael Alberto Moreno Parra

```
        //Inicia con un individuo infectado
        Pobl[azar.Next(NumIndividuos)].Estado = INFECTADO;

        timer1.Start();
    }


    private void timer1_Tick(object sender, System.EventArgs e) {
        Logica();
        Refresh(); //Visual de la animación
    }

    //Lógica de la población
    private void Logica() {
        int NumFil = Plano.GetLength(0);
        int NumCol = Plano.GetLength(1);

        //Mueve los individuos
        for (int cont = 0; cont < Pobl.Count; cont++)
            Pobl[cont].Mover(azar.Next(8), NumFil, NumCol);

        //Limpia el tablero
        for (int Fil = 0; Fil < NumFil; Fil++)
            for (int Col = 0; Col < NumCol; Col++)
                Plano[Fil, Col] = VACIO;

        //Refleja ese movimiento en el tablero
        for (int cont = 0; cont < Pobl.Count; cont++) {
            int Fila = Pobl[cont].Fila;
            int Columna = Pobl[cont].Columna;
            Plano[Fila, Columna] = Pobl[cont].Estado;
        }

        //Chequea si un individuo infectado coincide con un
        //individuo sano en la misma casilla para infectarlo
        for (int cont = 0; cont < Pobl.Count; cont++) {
            if (Pobl[cont].Estado == INFECTADO) {
                for (int busca = 0; busca < Pobl.Count; busca++) {
                    if (Pobl[cont].Fila == Pobl[busca].Fila &&
                        Pobl[cont].Columna == Pobl[busca].Columna)
                        Pobl[busca].Estado = INFECTADO;
                }
            }
        }
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {
```

```
        Graphics lienzo = e.Graphics;
        Pen Lapiz = new(Color.Blue, 1);
        Brush Llena1 = new SolidBrush(Color.Black);
        Brush Llena2 = new SolidBrush(Color.Red);

        //Tamaño de cada celda
        tamF = 500 / Plano.GetLength(0);
        tamC = 500 / Plano.GetLength(1);
        desplaza = 50;

        //Dibuja el arreglo bidimensional
        for (int Fil = 0; Fil < Plano.GetLength(0); Fil++) {
           for (int Col = 0; Col < Plano.GetLength(1); Col++) {
              int uF = Fil * tamF + desplaza;
              int uC = Col * tamC + desplaza;
              switch (Plano[Fil, Col]) {
                 case VACIO:
                    lienzo.DrawRectangle(Lapiz, uF, uC, tamF, tamC);
                    break;
                 case SANO:
                    lienzo.FillRectangle(Llena1, uF, uC, tamF, tamC);
                    break;
                 case INFECTADO:
                    lienzo.FillRectangle(Llena2, uF, uC, tamF, tamC);
                    break;
              }
           }
        }
     }
  }

  internal class Individuo {
     public int Fila, Columna, Estado;

     public Individuo(int Fila, int Columna, int Estado) {
        this.Fila = Fila;
        this.Columna = Columna;
        this.Estado = Estado;
     }

     public void Mover(int direccion, int NumFilas, int NumColumnas) {
        switch (direccion) {
           case 0: Fila--; Columna--; break;
           case 1: Fila--; break;
           case 2: Fila--; Columna++; break;
           case 3: Columna--; break;
           case 4: Columna++; break;
           case 5: Fila++; Columna--; break;
```

```
            case 6: Fila++; break;
            case 7: Fila++; Columna++; break;
         }

         if (Fila < 0) Fila = 0;
         if (Columna < 0) Columna = 0;
         if (Fila >= NumFilas) Fila = NumFilas - 1;
         if (Columna >= NumColumnas) Columna = NumColumnas - 1;
      }
   }
}
```

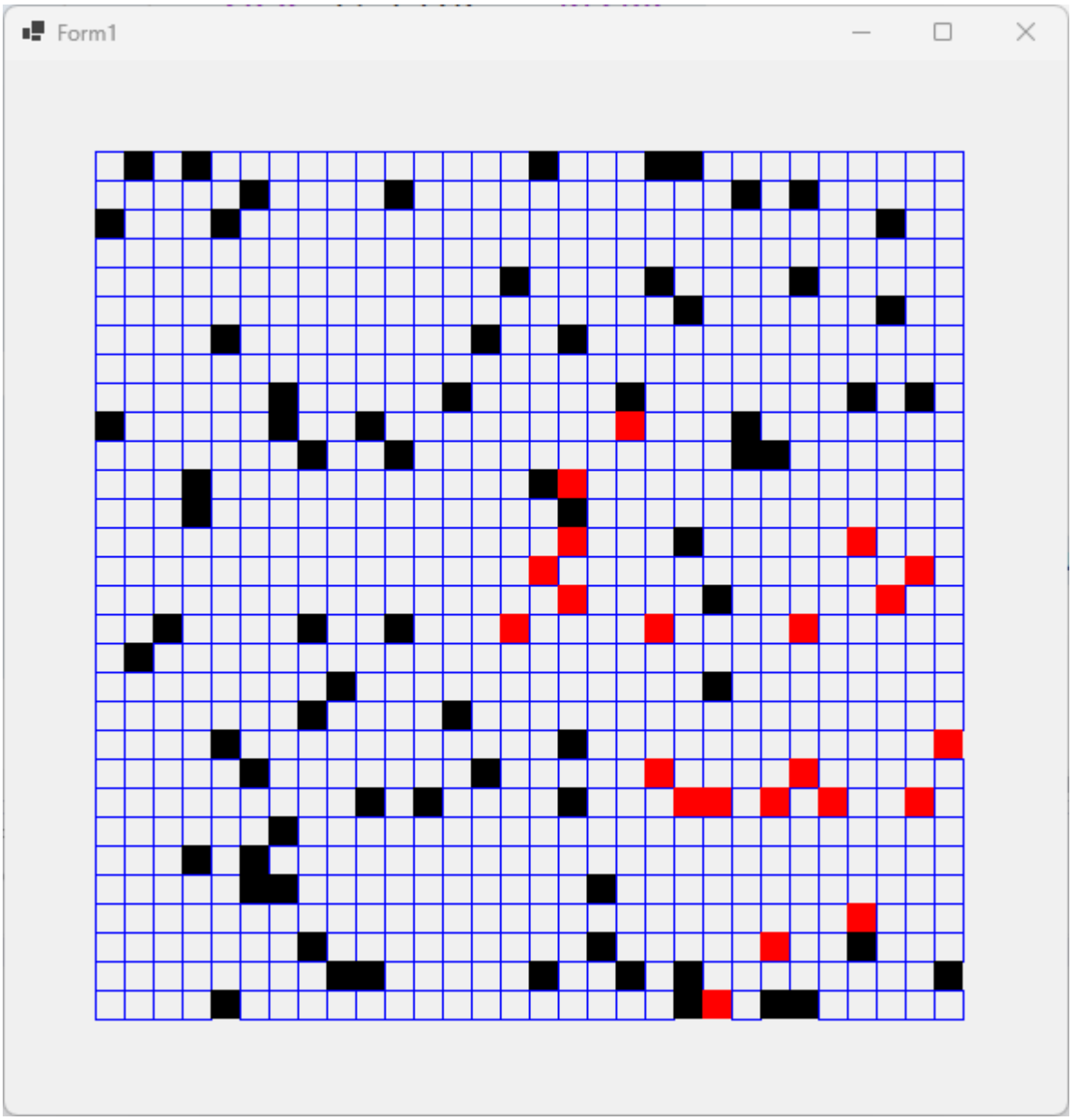

Ilustración 13: Infección propagándose

Gráfico matemático en 2D y animado

Dada una ecuación del tipo $Y = F(X,T)$, donde X es la variable independiente y T es el tiempo, por ejemplo:

$$Y = X * (T + 1) * seno(X^2 * T^2)$$

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de T y el valor final de T. Se llamarán Tminimo y Tmaximo respectivamente.
2. Saber la tasa de incremento del tiempo, para ir desde Tminimo hasta Tmaximo, una vez se alcance Tmaximo se devuelve decrementando a la misma tasa hasta Tminimo y vuelve a empezar.
3. Saber el valor de inicio de X y el valor final de X. Se llamarán Xini y Xfin respectivamente.
4. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
5. Con un control Timer, en cada tick se da un valor a T y se hacen los siguientes cálculos:
    a. Con X desde Xini hasta Xfin, se calculan los valores de Y. Se almacena el par X, Y
    b. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
    c. Se requieren cuatro datos:
        i. El mínimo valor de X. Es el mismo Xini.
        ii. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
        iii. El mínimo valor de Y obtenido. Se llamará Ymin.
        iv. El máximo valor de Y obtenido. Se llamará Ymax.
    d. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
        i. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
        ii. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
    e. Se calculan unas constantes de conversión con estas fórmulas:
        i. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
        ii. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
    f. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
        i. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
        ii. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
    g. Se grafican esos puntos y se unen con líneas.
    h. Se borra la pantalla y se da otro valor de T, eso da la sensación de animación.

Autor: Rafael Alberto Moreno Parra

```
namespace Animacion {
   public partial class Form1 : Form {
      //Para la variable temporal
      private double TiempoValor, Tminimo, Tmaximo, Tincrementa;

      //Datos de la ecuación (desde donde inicia y termina X)
      double minX;
      double maxX;
      int totalPuntos;

      //Donde almacena los puntos
      List<PuntosGrafico> punto;

      //Datos de la pantalla
      int XpIni, YpIni, XpFin, YpFin;

      private void timer1_Tick(object sender, EventArgs e) {
         TiempoValor += Tincrementa;
         if (TiempoValor <= Tminimo || TiempoValor >= Tmaximo)
            Tincrementa = -Tincrementa;

         Logica();
         Refresh();
      }

      public Form1() {
         InitializeComponent();
         punto = new List<PuntosGrafico>();

         //Área dónde se dibujará el gráfico matemático
         XpIni = 20;
         YpIni = 20;
         XpFin = 700;
         YpFin = 500;

         //Inicia el tiempo
         Tminimo = 0;
         Tmaximo = 2;
         Tincrementa = 0.05;
         TiempoValor = Tminimo;

         //Datos de la ecuación
         minX = -5; //Valor X mínimo
         maxX = 5; //Valor X máximo
         totalPuntos = 300;
      }
```

```
public void Logica() {
   //Calcula los puntos de la ecuación a graficar
   double pasoX = (maxX - minX) / totalPuntos;
   double Ymin = double.MaxValue; //El mínimo valor de Y obtenido
   double Ymax = double.MinValue; //El máximo valor de Y obtenido
   double maximoXreal = double.MinValue; //El máximo valor de X

   punto.Clear();
   for (double X = minX; X <= maxX; X += pasoX) {
      //Se invierte el valor porque el eje Y aumenta hacia abajo
      double valY = -1 * Ecuacion(X, TiempoValor);
      if (valY > Ymax) Ymax = valY;
      if (valY < Ymin) Ymin = valY;
      if (X > maximoXreal) maximoXreal = X;
      punto.Add(new PuntosGrafico(X, valY));
   }
   //¡OJO! X puede que no llegue a ser Xfin, por lo que
   //la variable maximoXreal almacena el valor máximo de X

   //Calcula los puntos a poner en la pantalla
   double conX = (XpFin - XpIni) / (maximoXreal - minX);
   double conY = (YpFin - YpIni) / (Ymax - Ymin);

   for (int cont = 0; cont < punto.Count; cont++) {
      double Xr = conX * (punto[cont].X - minX) + XpIni;
      double Yr = conY * (punto[cont].Y - Ymin) + YpIni;
      punto[cont].pX = Convert.ToInt32(Xr);
      punto[cont].pY = Convert.ToInt32(Yr);
   }
}

//Aquí está la ecuación que se desee graficar con variable
//independiente X y T
public double Ecuacion(double X, double T) {
   return X * (T + 1) * Math.Sin(X * X * T * T);
}

//Pinta la ecuación
private void Form1_Paint(object sender, PaintEventArgs e) {
   Graphics lienzo = e.Graphics;
   Pen lapiz = new(Color.Blue, 1);

   //Un recuadro para ver el área del gráfico
   int Xini = XpIni;
   int Yini = YpIni;
   int Xfin = XpFin - XpIni;
   int Yfin = YpFin - YpIni;
   lienzo.DrawRectangle(lapiz, Xini, Yini, Xfin, Yfin);
```

```
            //Dibuja el gráfico matemático
            for (int cont = 0; cont < punto.Count - 1; cont++) {
               Xini = punto[cont].pX;
               Yini = punto[cont].pY;
               Xfin = punto[cont + 1].pX;
               Yfin = punto[cont + 1].pY;
               lienzo.DrawLine(Pens.Black, Xini, Yini, Xfin, Yfin);
            }
         }
      }

      internal class PuntosGrafico {
         //Valor X, Y reales de la ecuación
         public double X, Y;

         //Puntos convertidos a coordenadas enteras de pantalla
         public int pX, pY;

         public PuntosGrafico(double X, double Y) {
            this.X = X;
            this.Y = Y;
         }
      }
}
```

Ilustración 14: Gráfico matemático 2D animado

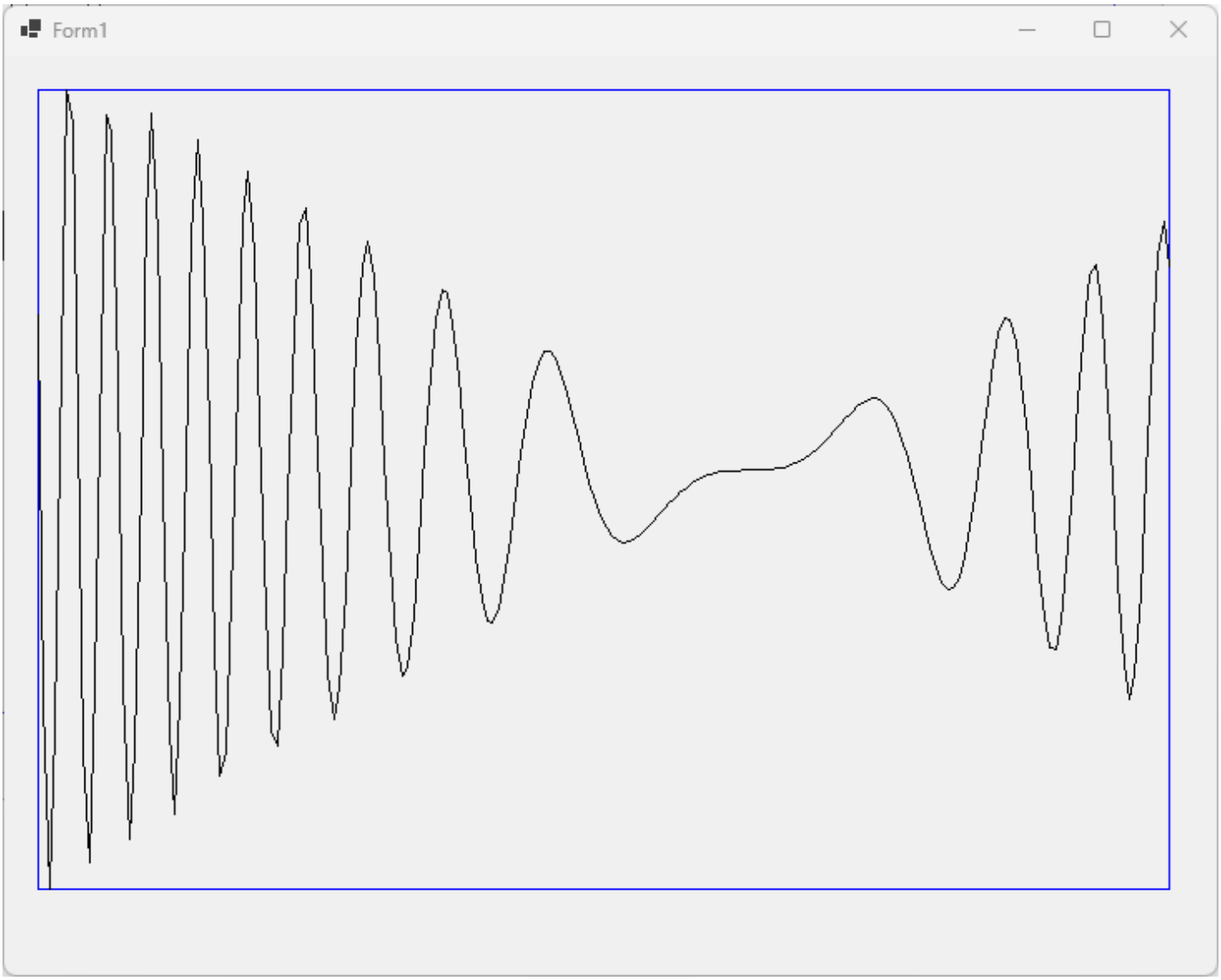

Ilustración 15: Gráfico matemático 2D animado

Gráfico polar animado

Dada una ecuación del tipo r=F(Θ, T), donde Θ es el ángulo, r el radio y T el tiempo, por ejemplo:

$$r = 2 * seno(4 * \theta * T * \pi/180)$$

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de T y el valor final de T. Se llamarán Tminimo y Tmaximo respectivamente.
2. Saber la tasa de incremento del tiempo, para ir desde Tminimo hasta Tmaximo, una vez se alcance Tmaximo se devuelve decrementando a la misma tasa hasta Tminimo y vuelve a empezar.
3. Saber el valor de inicio de Θ y el valor final de Θ. Se llamarán minTheta y maxTheta respectivamente.
4. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
5. Con un control Timer, en cada tick se da un valor a T y se hacen los siguientes cálculos:
    a. Con Θ desde minTheta y maxTheta se calculan los valores de r. Luego se hace esta conversión:

        r = Ecuacion(theta, T);

        X = r * Math.Cos(theta * Math.PI / 180);

        Y = r * Math.Sin(theta * Math.PI / 180);

    b. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
    c. Se requieren cuatro datos:
        i. El mínimo valor de X. Es el mismo Xini.
        ii. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
        iii. El mínimo valor de Y obtenido. Se llamará Ymin.
        iv. El máximo valor de Y obtenido. Se llamará Ymax.
    d. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
        i. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
        ii. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
    e. Se calculan unas constantes de conversión con estas fórmulas:
        i. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
        ii. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)

f. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
    i. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
    ii. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
g. Se grafican esos puntos y se unen con líneas.

Se borra la pantalla y se da otro valor de T, eso da la sensación de animación.

N/008.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Para la variable temporal
      private double TiempoValor, Tminimo, Tmaximo, Tincrementa;

      //Datos de la ecuación (desde donde inicia y termina Theta)
      double minTheta;
      double maxTheta;
      int numPuntos;

      //Donde almacena los puntos
      List<Puntos> punto;

      //Datos de la pantalla
      int XpIni, YpIni, XpFin, YpFin;

      public Form1() {
         InitializeComponent();
         punto = [];

         //Área dónde se dibujará el gráfico matemático
         XpIni = 20;
         YpIni = 20;
         XpFin = 700;
         YpFin = 500;

         //Inicia el tiempo
         Tminimo = 0;
         Tmaximo = 5;
         Tincrementa = 0.05;
         TiempoValor = Tminimo;

         //Datos de la ecuación
         minTheta = 0;
         maxTheta = 360;
         numPuntos = 200;
      }

      private void timer1_Tick(object sender, EventArgs e) {
```

```
      TiempoValor += Tincrementa;
      if (TiempoValor <= Tminimo || TiempoValor >= Tmaximo)
         Tincrementa = -Tincrementa;

      Logica();
      Refresh();
   }

   public void Logica() {
      //Calcula los puntos de la ecuación a graficar
      double paso = (maxTheta - minTheta) / numPuntos;
      double Ymin = double.MaxValue; //El mínimo valor de Y
      double Ymax = double.MinValue; //El máximo valor de Y
      double Xmin = double.MaxValue; //El máximo valor de X
      double Xmax = double.MinValue; //El máximo valor de X

      punto.Clear();
      for (double theta = minTheta; theta <= maxTheta; theta += paso) {
         double valorR = Ecuacion(theta, TiempoValor);
         double X = valorR * Math.Cos(theta * Math.PI / 180);
         double Y = -1 * valorR * Math.Sin(theta * Math.PI / 180);

         if (Y > Ymax) Ymax = Y;
         if (Y < Ymin) Ymin = Y;
         if (X > Xmax) Xmax = X;
         if (X < Xmin) Xmin = X;
         punto.Add(new Puntos(X, Y));
      }

      //Calcula los puntos a poner en la pantalla
      double conX = (XpFin - XpIni) / (Xmax - Xmin);
      double conY = (YpFin - YpIni) / (Ymax - Ymin);

      for (int cont = 0; cont < punto.Count; cont++) {
         double Xr = conX * (punto[cont].X - Xmin) + XpIni;
         double Yr = conY * (punto[cont].Y - Ymin) + YpIni;
         punto[cont].pX = Convert.ToInt32(Xr);
         punto[cont].pY = Convert.ToInt32(Yr);
      }
   }

   //Aquí está la ecuación que se desee graficar
   //con variable Theta y T (tiempo)
   public double Ecuacion(double Theta, double T) {
      return 2 * (1 + Math.Sin(Theta * T * Math.PI / 180));
   }

   //Pinta la ecuación
```

```
      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics lienzo = e.Graphics;
         Pen lapiz = new(Color.Blue, 1);

         //Un recuadro para ver el área del gráfico
         int Xini = XpIni;
         int Yini = YpIni;
         int Xfin = XpFin - XpIni;
         int Yfin = YpFin - YpIni;
         lienzo.DrawRectangle(lapiz, Xini, Yini, Xfin, Yfin);

         //Dibuja el gráfico matemático
         for (int cont = 0; cont < punto.Count - 1; cont++) {
            Xini = punto[cont].pX;
            Yini = punto[cont].pY;
            Xfin = punto[cont + 1].pX;
            Yfin = punto[cont + 1].pY;
            lienzo.DrawLine(Pens.Black, Xini, Yini, Xfin, Yfin);
         }
      }
   }

   internal class Puntos {
      //Valor X, Y reales de la ecuación
      public double X, Y;

      //Puntos convertidos a coordenadas enteras de pantalla
      public int pX, pY;

      public Puntos(double X, double Y) {
         this.X = X;
         this.Y = Y;
      }
   }
}
```

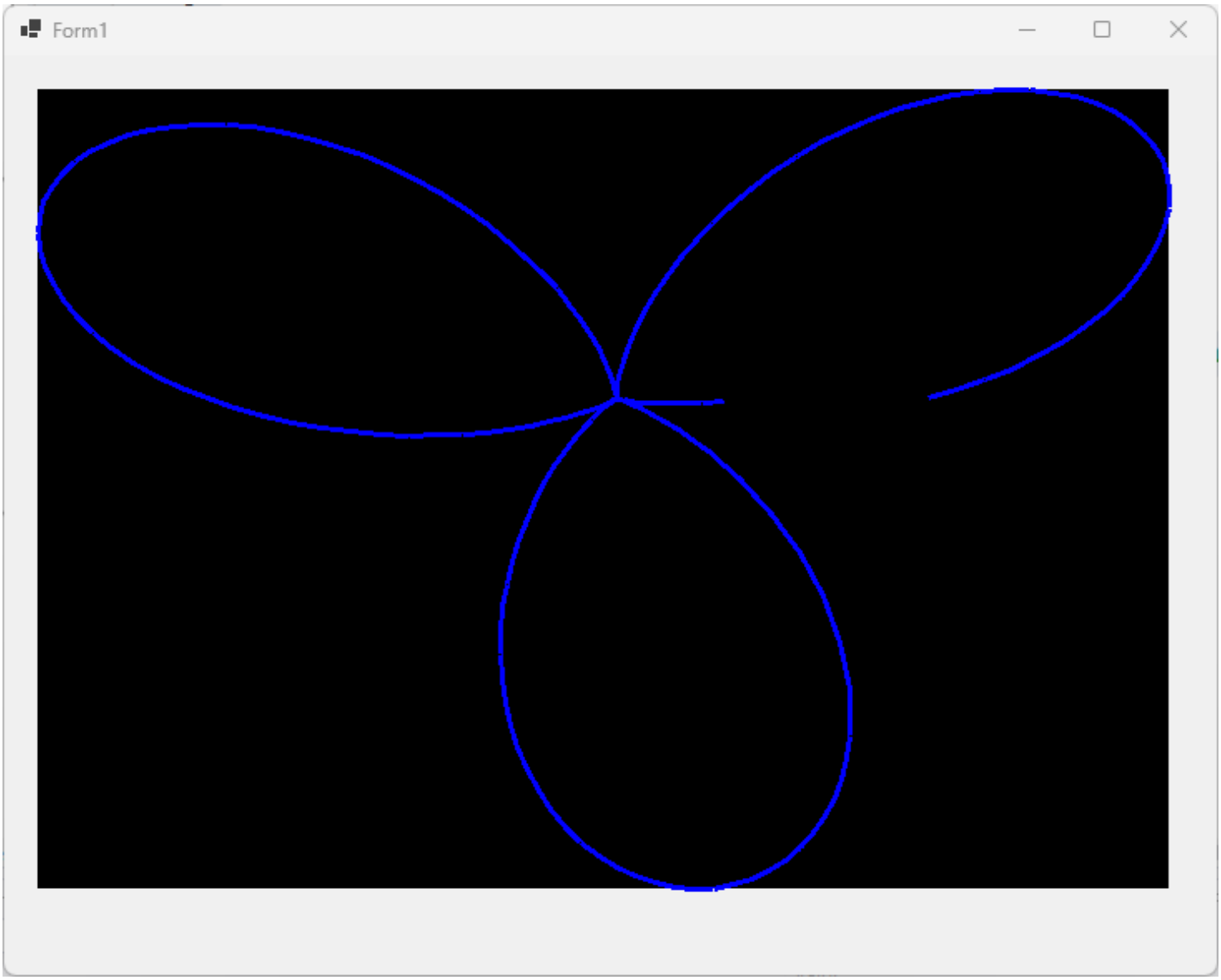

Ilustración 16: Gráfico polar animado

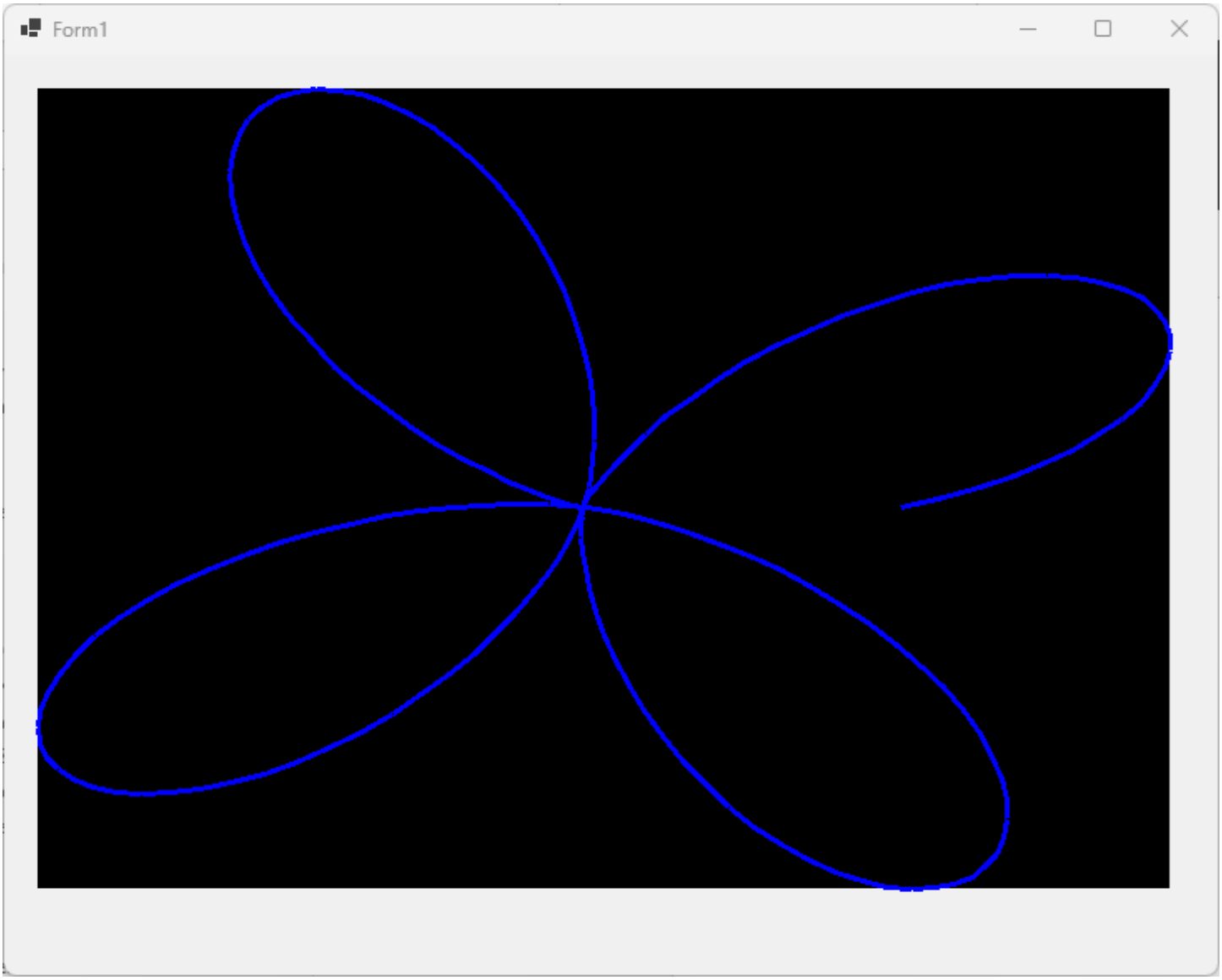

Ilustración 17: Gráfico polar animado

Gráfico Matemático en 3D. Giros animados.

Cuando tenemos una ecuación del tipo $Z = F(X,Y)$, tenemos una ecuación con dos variables independientes y una variable dependiente. Se van a hacer giros en los tres ejes de forma aleatoria y continua por lo que da la sensación de una animación. Por ejemplo:

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

Los pasos para hacer el gráfico son parecidos al anterior:

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

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

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

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

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

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

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

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

Paso 9: Se calcula un nuevo giro en los tres ángulos. Y se vuelve al Paso 4.

Nota 1: El gráfico se ve animado por lo que es necesario que el formulario tenga en “true” la propiedad DoubleBuffered

Nota 2: El gráfico animado está contenido dentro de un cubo invisible de lado = 1. Eso se logra con la normalización. Sin importar el valor que tome Z, el gráfico estará contenido dentro de ese cubo, luego si Z crece mucho, el resultado visual es que el gráfico se empequeñece para que se ajuste todo a ese cubo

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

      //Para el movimiento aleatorio del gráfico 3D
      private int AnguloX, AnguloY, AnguloZ;
      private int IncrAngX, IncrAngY, IncrAngZ;
      private int AnguloXNuevo, AnguloYNuevo, AnguloZNuevo;
      Random Azar;

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

         //Hace los cálculos de la ecuación.
         double MinimoX = -9;
         double MinimoY = -9;
         double MaximoX = 9;
         double MaximoY = 9;
         int LineasGrafico = 30;
         Grafico3D.CalculaEcuacion(MinimoX, MinimoY,
                           MaximoX, MaximoY,
                           LineasGrafico);

         //Ángulos aleatorios para simular animación
         Azar = new Random();
         AnguloX = Azar.Next(0, 360);
         AnguloY = Azar.Next(0, 360);
         AnguloZ = Azar.Next(0, 360);
         AnguloXNuevo = Azar.Next(0, 360);
         AnguloYNuevo = Azar.Next(0, 360);
         AnguloZNuevo = Azar.Next(0, 360);
      }

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

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

```
        //Aplica giros, conversión a 2D y cuadrar en pantalla
        Grafico3D.CalculaGrafico(AnguloX, AnguloY, AnguloZ, ZPersona,
                                 XPantallaIni, YPantallaIni,
                                 XPantallaFin, YPantallaFin);
        Grafico3D.Dibuja(Lienzo, Lapiz, Relleno);
    }

    private void timer1_Tick(object sender, EventArgs e) {
        //Cambio de los ángulos de giro
        if (AnguloX > AnguloXNuevo) IncrAngX = -1; else IncrAngX = 1;
        if (AnguloY > AnguloYNuevo) IncrAngY = -1; else IncrAngY = 1;
        if (AnguloZ > AnguloZNuevo) IncrAngZ = -1; else IncrAngZ = 1;

        AnguloX += IncrAngX;
        AnguloY += IncrAngY;
        AnguloZ += IncrAngZ;

        if (AnguloX == AnguloXNuevo) AnguloXNuevo = Azar.Next(0, 360);
        if (AnguloY == AnguloYNuevo) AnguloYNuevo = Azar.Next(0, 360);
        if (AnguloZ == AnguloZNuevo) AnguloZNuevo = Azar.Next(0, 360);
        Refresh();
    }
}

internal class Grafico {

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

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

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

        //Calcula cada polígono dependiendo de la ecuación
        double IncrX = (maxX - minX) / numLineas;
        double IncrY = (maxY - minY) / numLineas;

        for (double ValX = minX; ValX <= maxX; ValX += IncrX)
            for (double ValY = minY; ValY <= maxY; ValY += IncrY) {

                //Calcula los 4 valores del eje Z del polígono
```

```
            double Z1 = Ecuacion(ValX, ValY);
            double Z2 = Ecuacion(ValX + IncrX, ValY);
            double Z3 = Ecuacion(ValX + IncrX, ValY + IncrY);
            double Z4 = Ecuacion(ValX, ValY + IncrY);

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

            //Captura el mínimo valor de Z de todos los polígonos
            if (Z1 < MinimoZ) MinimoZ = Z1;
            if (Z2 < MinimoZ) MinimoZ = Z2;
            if (Z3 < MinimoZ) MinimoZ = Z3;
            if (Z4 < MinimoZ) MinimoZ = Z4;

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

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

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

   }

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

   public void CalculaGrafico(double AngX, double AngY, double AngZ,
                              int ZPersona,
```

```
                        int XpIni, int YpIni,
                        int XpFin, int YpFin) {

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

        //Matriz de Rotación

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

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

        //Las constantes de transformación
        double conX = (XpFin - XpIni) / (MaxX - MinX);
        double conY = (YpFin - YpIni) / (MaxY - MinY);

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

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

     //Dibuja el polígono
     public void Dibuja(Graphics Lienzo, Pen Lapiz, Brush Relleno) {
```

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


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

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

   //Centro del polígono
   public double Centro;

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

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

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

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

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


   //Gira en XYZ, convierte a 2D y cuadra en pantalla
```

```
public void CalculoPantalla(double[,] Giro, double ZPersona,
                    double conX, double conY,
                    double MinimoX, double MinimoY,
                    int XpIni, int YpIni) {
   double X1g = X1 * Giro[0, 0] + Y1 * Giro[1, 0] + Z1 * Giro[2, 0];
   double Y1g = X1 * Giro[0, 1] + Y1 * Giro[1, 1] + Z1 * Giro[2, 1];
   double Z1g = X1 * Giro[0, 2] + Y1 * Giro[1, 2] + Z1 * Giro[2, 2];

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

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

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

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

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

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

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

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

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

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

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

   X4p = Convert.ToInt32(conX * (X4sd - MinimoX) + XpIni);
```

```
            Y4p = Convert.ToInt32(conY * (Y4sd - MinimoY) + YpIni);
        }

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

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

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

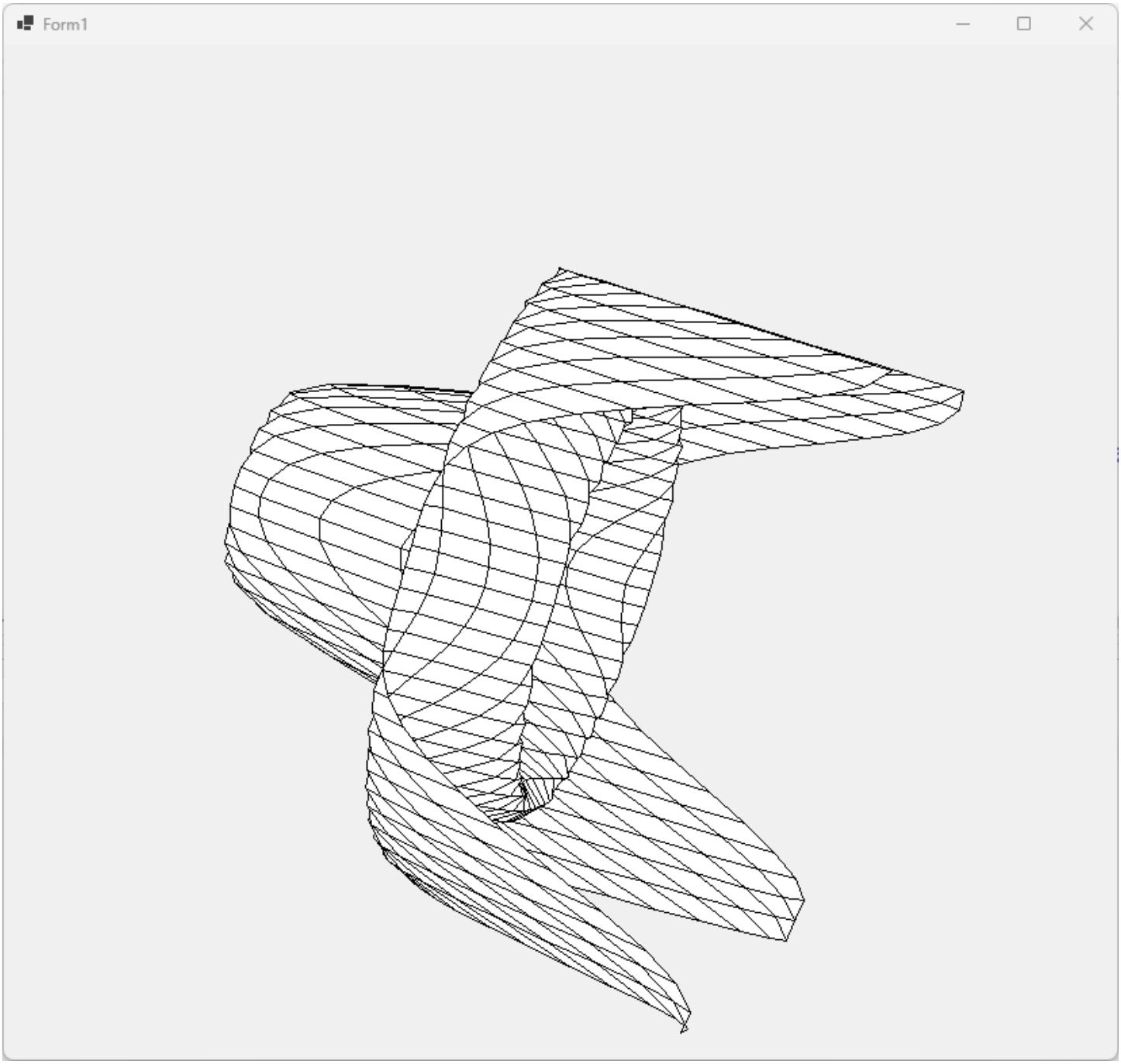

Ilustración 18: Gráfico Matemático en 3D. Giros animados

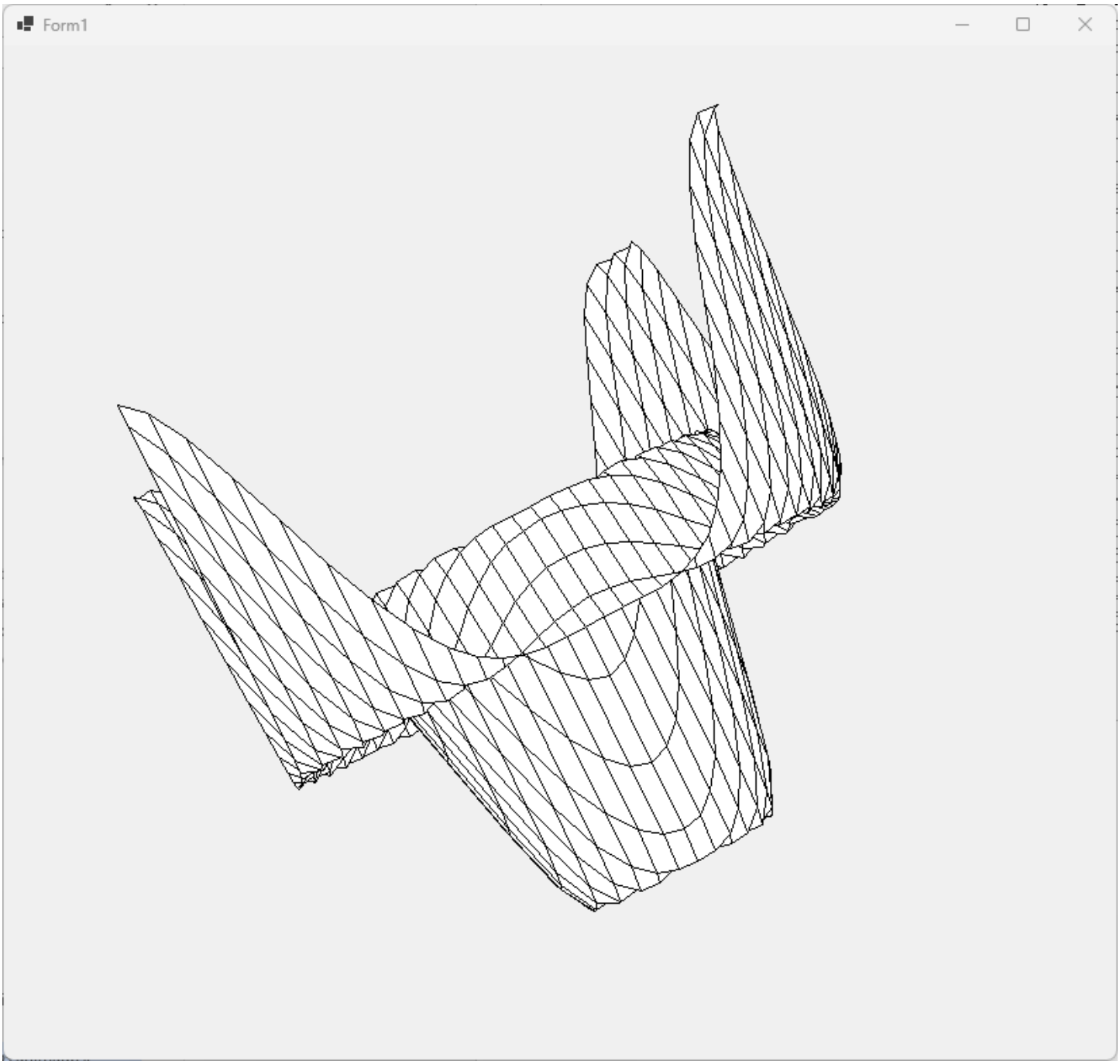

Ilustración 19: Gráfico Matemático en 3D. Giros animados

Gráfico Matemático en 4D

Cuando tenemos una ecuación del tipo $Z = F(X, Y, T)$, tenemos una ecuación con tres variables independientes (entre ellas, la del tiempo por lo que se conoce como 4D) y una variable dependiente. Su representación es 3D animado. Por ejemplo:

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

Los pasos para hacer el gráfico son los siguientes:

Paso 0: Saber el valor del tiempo mínimo y el valor del tiempo máximo. Se hace uso de un control "timer" que cada vez que se dispara el evento "Tick" varía el valor de T desde el mínimo hasta el máximo paso a paso, una vez alcanzado el máximo se decrementa paso a paso hasta el mínimo y repite el ciclo. De resto es igual a hacer un gráfico de ecuación tipo $Z = F(X, Y)$.

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

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

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

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

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

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

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

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

Nota 1: El gráfico se ve animado por lo que es necesario que el formulario tenga en "true" la propiedad DoubleBuffered

Autor: Rafael Alberto Moreno Parra

Nota 2: El gráfico animado está contenido dentro de un cubo invisible de lado = 1. Eso se logra con la normalización. Sin importar el valor que tome Z, el gráfico estará contenido dentro de ese cubo, luego si Z crece mucho, el resultado visual es que el gráfico se empequeñece para que se ajuste todo a ese cubo.

N/010.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Para la variable temporal
      private double Tminimo, Tmaximo, Tincrementa, Tiempo;

      //El objeto que se encarga de calcular y
      //cuadrar en pantalla la ecuación Z = F(X,Y)
      Grafico Grafico3D;

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

         //Variable Tiempo
         Tminimo = 0.2;
         Tmaximo = 3.0;
         Tincrementa = 0.01;
         Tiempo = Tminimo;
      }

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

         //Hace los cálculos de la ecuación primero.
         double minX = -10;
         double minY = -10;
         double maxX = 10;
         double maxY = 10;
         int numLineas = 40;
         Grafico3D.CalculaEcuacion(minX, minY, maxX, maxY, numLineas,
Tiempo);

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

```
        double AnguloX = 45;
        double AnguloY = 45;
        double AnguloZ = 60;

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

     private void timer1_Tick(object sender, EventArgs e) {
        //Cambia el valor de la variable tiempo
        Tiempo += Tincrementa;
        if (Tiempo <= Tminimo || Tiempo >= Tmaximo) Tincrementa = -
Tincrementa;

        Refresh();
     }
  }

  internal class Grafico {

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

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

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

        //Calcula cada polígono dependiendo de la ecuación
        double IncrX = (maxX - minX) / numLineas;
        double IncrY = (maxY - minY) / numLineas;

        for (double ValX = minX; ValX <= maxX; ValX += IncrX)
           for (double ValY = minY; ValY <= maxY; ValY += IncrY) {

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

```
            double Z4 = Ecuacion(ValX, ValY + IncrY, Tiempo);

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

            //Captura el mínimo valor de Z de todos los polígonos
            if (Z1 < MinimoZ) MinimoZ = Z1;
            if (Z2 < MinimoZ) MinimoZ = Z2;
            if (Z3 < MinimoZ) MinimoZ = Z3;
            if (Z4 < MinimoZ) MinimoZ = Z4;

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

            //Añade un polígono a la lista
            if (Math.Abs(MaximoZ - MinimoZ) > 0.00001)
               poligono.Add(new Poligono(ValX, ValY, Z1,
                                ValX + IncrX, ValY, Z2,
                                ValX + IncrX, ValY + IncrY, Z3,
                                ValX, ValY + IncrY, Z4));
         }

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

   }

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

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

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

        //Matriz de Rotación

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

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

        //Las constantes de transformación
        double conX = (XpFin - XpIni) / (MaxX - MinX);
        double conY = (YpFin - YpIni) / (MaxY - MinY);

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

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

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

```
    }
}


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

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

    //Centro del polígono
    public double Centro;

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

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

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

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

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


    //Gira en XYZ, convierte a 2D y cuadra en pantalla
    public void CalculoPantalla(double[,] Giro, double ZPersona,
                                double conX, double conY,
```

```
                    double MinimoX, double MinimoY,
                    int XpIni, int YpIni) {
        double X1g = X1 * Giro[0, 0] + Y1 * Giro[1, 0] + Z1 * Giro[2, 0];
        double Y1g = X1 * Giro[0, 1] + Y1 * Giro[1, 1] + Z1 * Giro[2, 1];
        double Z1g = X1 * Giro[0, 2] + Y1 * Giro[1, 2] + Z1 * Giro[2, 2];

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

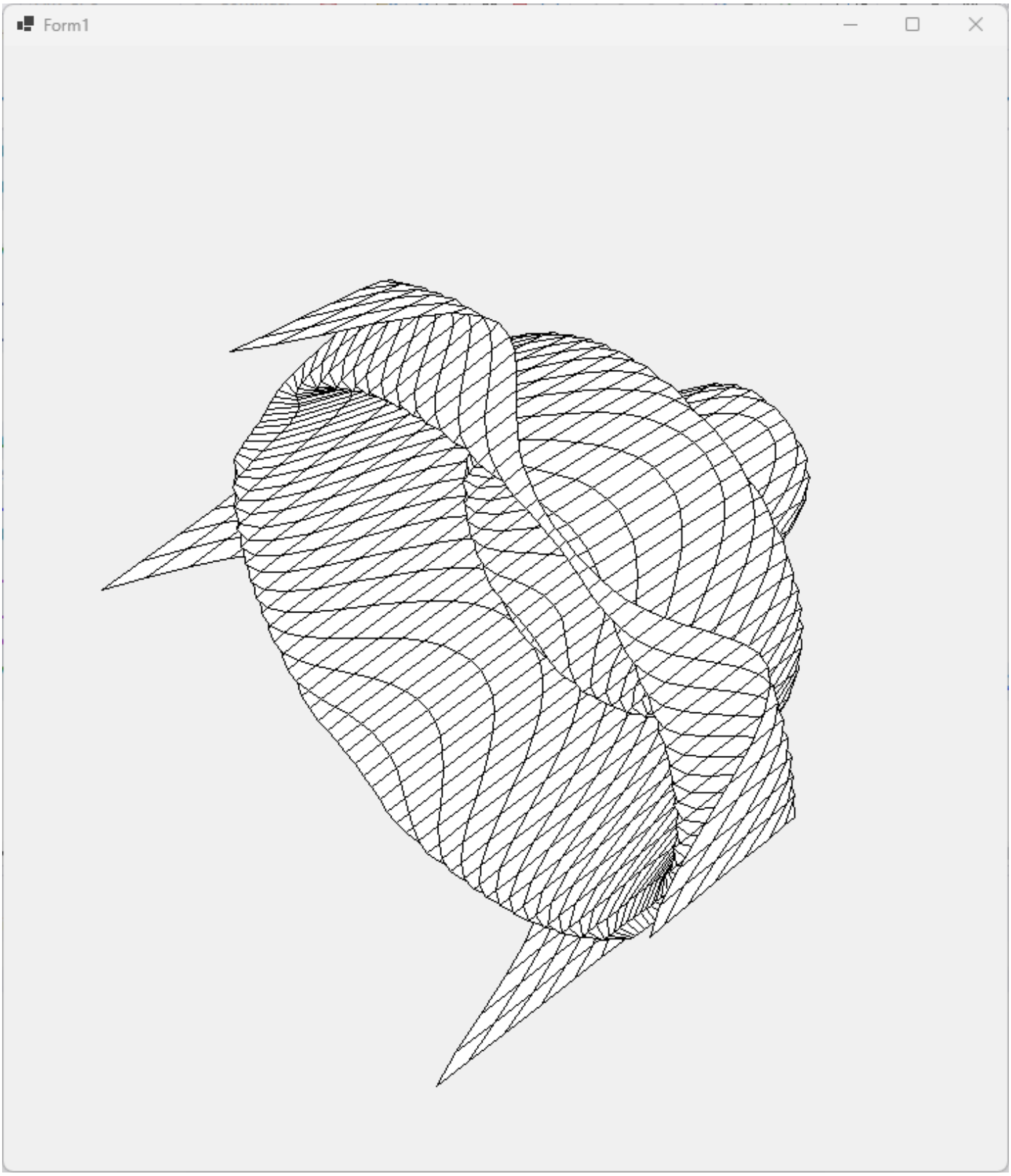

Ilustración 20: Gráfico Matemático en 4D

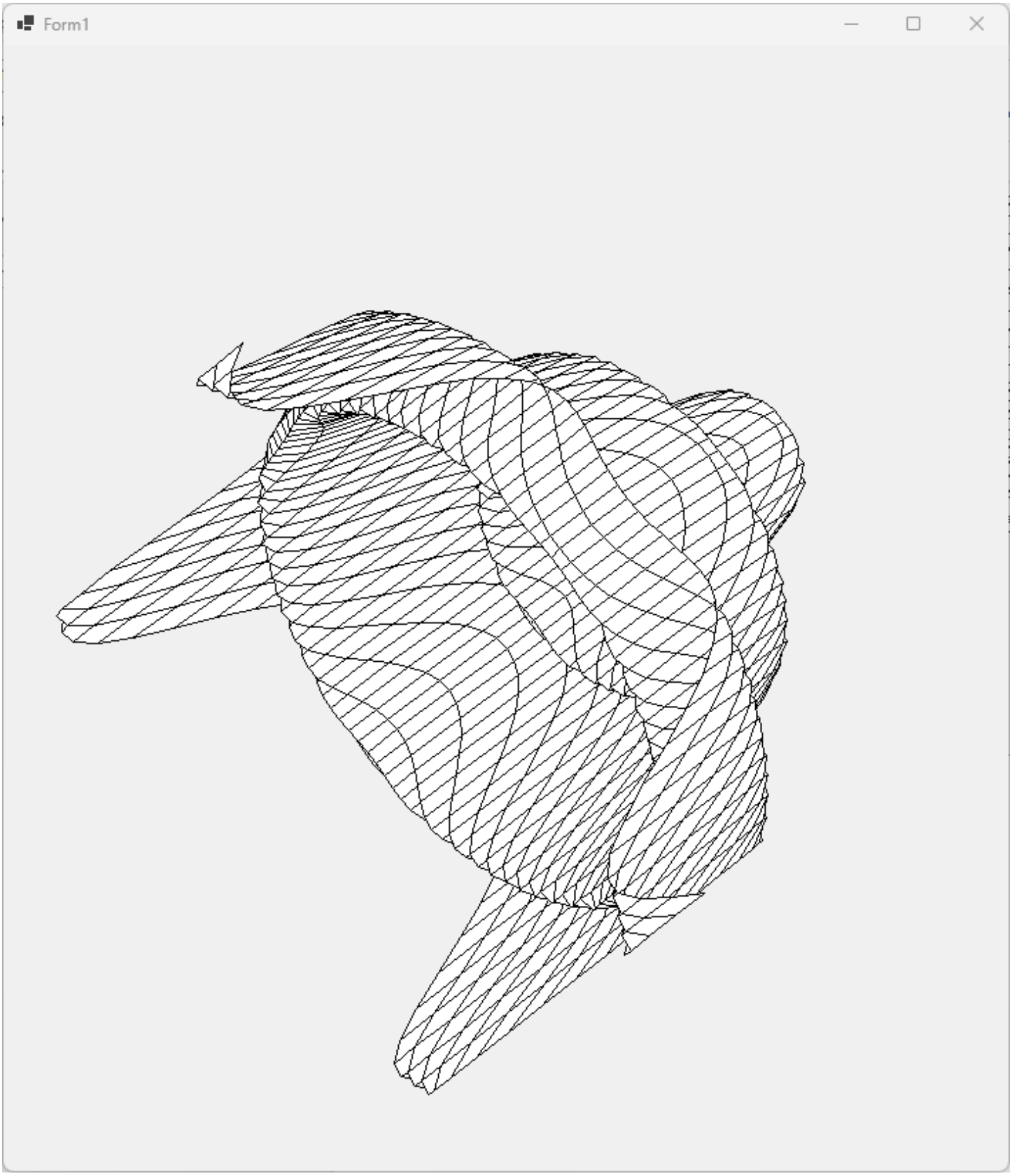

Ilustración 21: Gráfico Matemático en 4D

Gráfico Polar en 4D

Cuando tenemos una ecuación polar del tipo $r = F(\theta, \varphi, t)$, tenemos una ecuación con tres variables independientes (una de ellas es el tiempo) y una variable dependiente. Su representación es en 3D animada. Un ejemplo de este tipo de ecuación:

$$r = Cos(\varphi * t) + Sen(\theta * t)$$

Los pasos para hacer el gráfico son los siguientes:

Paso 0: Saber el valor del tiempo mínimo y el valor del tiempo máximo. Se hace uso de un control "timer" que cada vez que se dispara el evento "Tick" varía el valor de T desde el mínimo hasta el máximo paso a paso, una vez alcanzado el máximo se decrementa paso a paso hasta el mínimo y repite el ciclo. De resto es igual a hacer un gráfico de ecuación tipo $r = F(\theta, \varphi)$.

En este código se cambia además los ángulos de giro.

N/011.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Para la variable temporal
      private double Tminimo, Tmaximo, Tincrementa, Tiempo;

      //El objeto que se encarga de calcular y
      //cuadrar en pantalla la ecuación polar
      Grafico Grafico3D;

      //Para el movimiento aleatorio del gráfico 3D
      private int AnguloX, AnguloY, AnguloZ;
      private int IncrAngX, IncrAngY, IncrAngZ;

      private int AnguloXNuevo, AnguloYNuevo, AnguloZNuevo;
      Random Azar;

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

         //Ángulos aleatorios para simular animación
         Azar = new Random();
         AnguloX = Azar.Next(0, 360);
```

```
        AnguloY = Azar.Next(0, 360);
        AnguloZ = Azar.Next(0, 360);
        AnguloXNuevo = Azar.Next(0, 360);
        AnguloYNuevo = Azar.Next(0, 360);
        AnguloZNuevo = Azar.Next(0, 360);

        //Variable Tiempo
        Tminimo = 0.2;
        Tmaximo = 1.2;
        Tincrementa = 0.05;
        Tiempo = Tminimo;
    }

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

        int NumeroLineasGrafico = 70;
        Grafico3D.CalculaEcuacion(NumeroLineasGrafico, Tiempo);

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

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

    private void timer1_Tick(object sender, EventArgs e) {
        //Cambia los ángulos de giro
        if (AnguloX > AnguloXNuevo) IncrAngX = -1; else IncrAngX = 1;
        if (AnguloY > AnguloYNuevo) IncrAngY = -1; else IncrAngY = 1;
        if (AnguloZ > AnguloZNuevo) IncrAngZ = -1; else IncrAngZ = 1;

        AnguloX += IncrAngX;
        AnguloY += IncrAngY;
        AnguloZ += IncrAngZ;

        if (AnguloX == AnguloXNuevo) AnguloXNuevo = Azar.Next(0, 360);
        if (AnguloY == AnguloYNuevo) AnguloYNuevo = Azar.Next(0, 360);
```

```
        if (AnguloZ == AnguloZNuevo) AnguloZNuevo = Azar.Next(0, 360);

        //Cambia el valor de la variable tiempo
        Tiempo += Tincrementa;
        if (Tiempo <= Tminimo || Tiempo >= Tmaximo)
           Tincrementa = -Tincrementa;

        Refresh();
     }
  }

  internal class Grafico {

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

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

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

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

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

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

              //Calcula los 4 valores del eje Z del polígono
              double Theta1 = Theta * Math.PI / 180;
              double Phi1 = Phi * Math.PI / 180;
              double R1 = Ecuacion(Theta1, Phi1, Tiempo);
              if (double.IsNaN(R1) || double.IsInfinity(R1)) R1 = 0;
              double X1 = R1 * Math.Cos(Phi1) * Math.Sin(Theta1);
              double Y1 = R1 * Math.Sin(Phi1) * Math.Sin(Theta1);
              double Z1 = R1 * Math.Cos(Theta1);

              double Theta2 = (Theta + IncTheta) * Math.PI / 180;
```

```
double Phi2 = Phi * Math.PI / 180;
double R2 = Ecuacion(Theta2, Phi2, Tiempo);
if (double.IsNaN(R2) || double.IsInfinity(R2)) R2 = 0;
double X2 = R2 * Math.Cos(Phi2) * Math.Sin(Theta2);
double Y2 = R2 * Math.Sin(Phi2) * Math.Sin(Theta2);
double Z2 = R2 * Math.Cos(Theta2);

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

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

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

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

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

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

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

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

            //Añade un polígono a la lista
            if (Math.Abs(MaximoZ - MinimoZ) > 0.00001)
               poligono.Add(new Poligono(X1, Y1, Z1,
                                         X2, Y2, Z2,
                                         X3, Y3, Z3,
                                         X4, Y4, Z4));
         }

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

   }

   //Aquí está la ecuación polar 3D que se desee graficar
   //con variable Theta, PHI y Tiempo
   public double Ecuacion(double Theta, double Phi, double Tiempo) {
      return 2 * Math.Sin(Phi + Tiempo) - Math.Sin(Theta - Tiempo);
   }

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

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

      //Matriz de Rotación

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

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

      //Las constantes de transformación
      double conX = (XpFin - XpIni) / (MaxX - MinX);
      double conY = (YpFin - YpIni) / (MaxY - MinY);

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

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

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


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

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

   //Centro del polígono
   public double Centro;

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

```
        this.X2 = X2; this.Y2 = Y2; this.Z2 = Z2;
        this.X3 = X3; this.Y3 = Y3; this.Z3 = Z3;
        this.X4 = X4; this.Y4 = Y4; this.Z4 = Z4;
    }

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

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

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

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


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

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

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

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

        //Usado para ordenar los polígonos
```

```
        //del más lejano al más cercano
        Centro = (Z1g + Z2g + Z3g + Z4g) / 4;

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

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

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

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

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

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

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

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

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

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

    //Usado para ordenar los polígonos
    //del más lejano al más cercano
    public int CompareTo(object obj) {
```

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

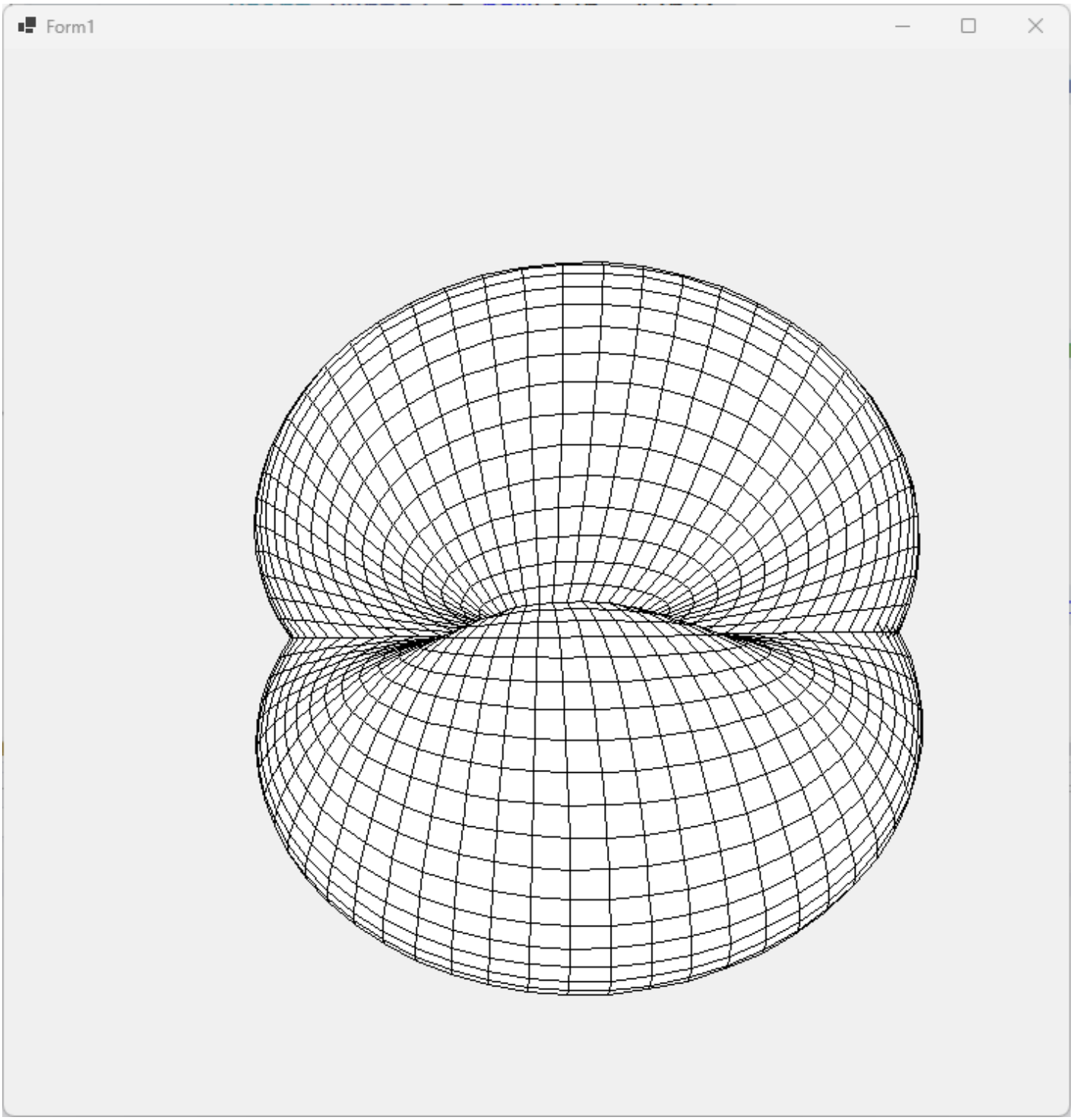

Ilustración 22: Gráfico polar 4D

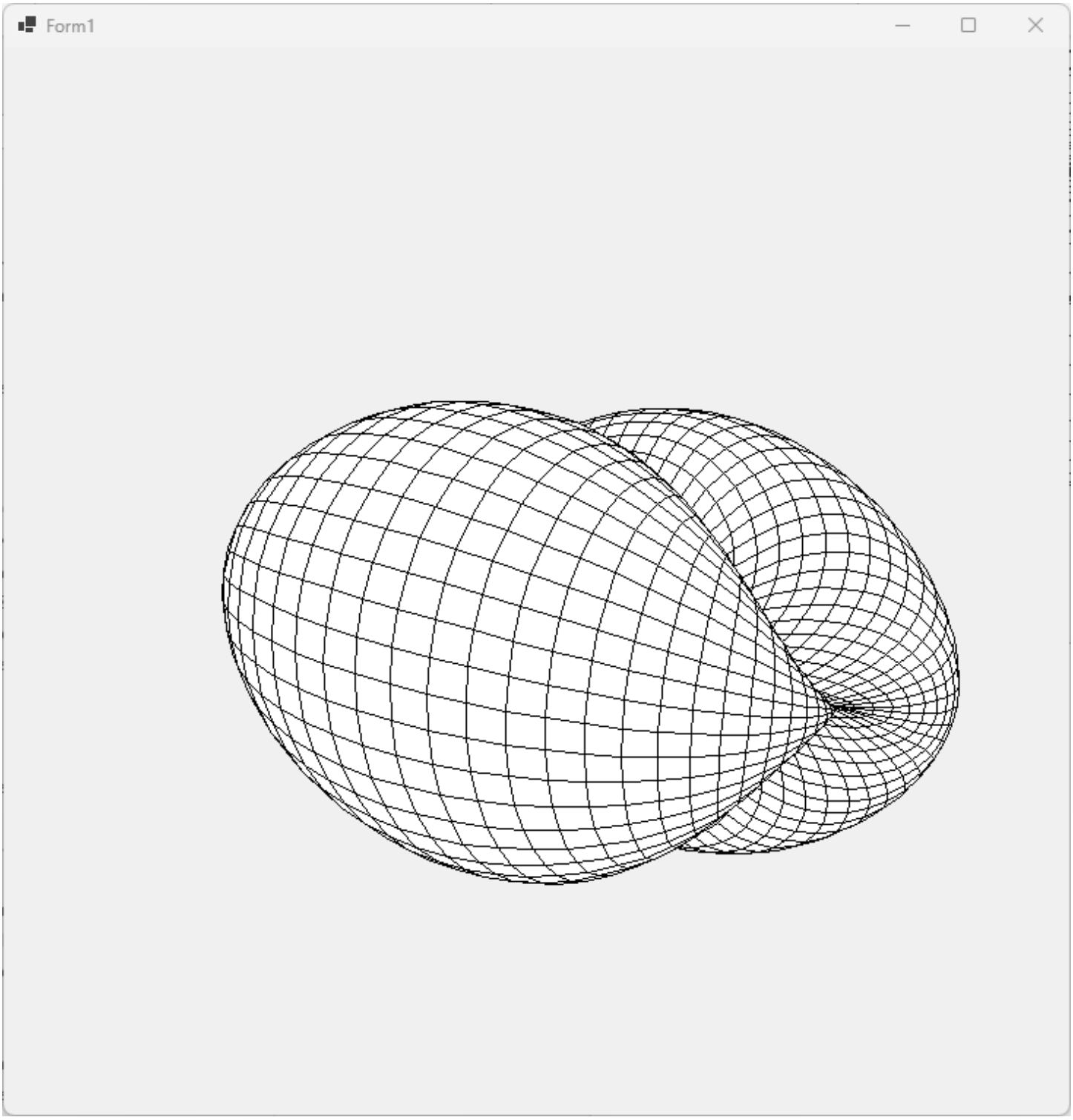

Ilustración 23: Gráfico polar 4D

Sólido de revolución animado

Una ecuación del tipo $Y = F(X,t)$ donde t es el tiempo. Y similar al anterior, con esa ecuación al girarla en el eje X se obtiene el sólido de revolución. Por cada t se cambia el gráfico. Un control timer se encarga de cambiar t y luego evaluar toda la ecuación con ese valor de t en particular, generar el sólido de revolución y mostrarlo en pantalla.

N/012.7z

```
namespace Animacion {
   public partial class Form1 : Form {
      //Para la variable temporal
      private double Tminimo, Tmaximo, Tincrementa, Tiempo;

      //El objeto que se encarga de calcular y
      //cuadrar en pantalla la ecuación que
      //genera el sólido de revolución
      Grafico Grafico3D;

      //Para el movimiento aleatorio del gráfico 3D
      private int AnguloX, AnguloY, AnguloZ;
      private int IncrAngX, IncrAngY, IncrAngZ;

      private int AnguloXNuevo, AnguloYNuevo, AnguloZNuevo;
      Random Azar;

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

         //Ángulos aleatorios para simular animación
         Azar = new Random();
         AnguloX = Azar.Next(0, 360);
         AnguloY = Azar.Next(0, 360);
         AnguloZ = Azar.Next(0, 360);
         AnguloXNuevo = Azar.Next(0, 360);
         AnguloYNuevo = Azar.Next(0, 360);
         AnguloZNuevo = Azar.Next(0, 360);

         //Variable Tiempo
         Tminimo = 0.2;
         Tmaximo = 1.2;
         Tincrementa = 0.05;
         Tiempo = Tminimo;
      }

      //Pinta el gráfico generado por la ecuación
      private void Form1_Paint(object sender, PaintEventArgs e) {
```

```
        Graphics Lienzo = e.Graphics;
        Pen Lapiz = new(Color.Black, 1);
        Brush Relleno = new SolidBrush(Color.White);

        double MinXreal = 0;
        double MaxXreal = 400;
        int numLineas = 40;
        Grafico3D.CalculaEcuacion(MinXreal, MaxXreal, numLineas, Tiempo);

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

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

    private void timer1_Tick(object sender, EventArgs e) {
        //Cambia los ángulos de giro
        if (AnguloX > AnguloXNuevo) IncrAngX = -1; else IncrAngX = 1;
        if (AnguloY > AnguloYNuevo) IncrAngY = -1; else IncrAngY = 1;
        if (AnguloZ > AnguloZNuevo) IncrAngZ = -1; else IncrAngZ = 1;

        AnguloX += IncrAngX;
        AnguloY += IncrAngY;
        AnguloZ += IncrAngZ;

        if (AnguloX == AnguloXNuevo) AnguloXNuevo = Azar.Next(0, 360);
        if (AnguloY == AnguloYNuevo) AnguloYNuevo = Azar.Next(0, 360);
        if (AnguloZ == AnguloZNuevo) AnguloZNuevo = Azar.Next(0, 360);

        //Cambia el valor de la variable tiempo
        Tiempo += Tincrementa;
        if (Tiempo <= Tminimo || Tiempo >= Tmaximo)
            Tincrementa = -Tincrementa;

        Refresh();
    }
}

internal class Grafico {
```

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

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

   double incAng = 360 / numLineas;
   double incrX = (maxXreal - minXreal) / numLineas;

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

   for (double ang = 0; ang < 360; ang += incAng)
      for (double X = minXreal; X <= maxXreal; X += incrX) {

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

         //Hace giro
         double X1g = X1;
         double Y1g = Y1 * Math.Cos(ang * Math.PI / 180);
         double Z1g = Y1 * -Math.Sin(ang * Math.PI / 180);

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

         //Hace giro
         double X2g = X2;
         double Y2g = Y2 * Math.Cos(ang * Math.PI / 180);
         double Z2g = Y2 * -Math.Sin(ang * Math.PI / 180);

         //Tercer punto ya girado
         double X3g = X2;
         double Y3g = Y2 * Math.Cos((ang + incAng) * Math.PI / 180);
         double Z3g = Y2 * -Math.Sin((ang + incAng) * Math.PI / 180);

         //Cuarto punto ya girado
```

```
            double X4g = X1;
            double Y4g = Y1 * Math.Cos((ang + incAng) * Math.PI / 180);
            double Z4g = Y1 * -Math.Sin((ang + incAng) * Math.PI / 180);

            //Obtener los valores extremos para poder normalizar
            if (X1g < MinimoX) MinimoX = X1g;
            if (X2g < MinimoX) MinimoX = X2g;
            if (X3g < MinimoX) MinimoX = X3g;
            if (X4g < MinimoX) MinimoX = X4g;

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

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

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

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

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

            if (Math.Abs(MaximoZ - MinimoZ) > 0.00001)
               poligono.Add(new Poligono(X1g, Y1g, Z1g,
                                         X2g, Y2g, Z2g,
                                         X3g, Y3g, Z3g,
                                         X4g, Y4g, Z4g));
        }

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

```
    }

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

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

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

       //Matriz de Rotación

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

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

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

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

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

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


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

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

   //Centro del polígono
   public double Centro;

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

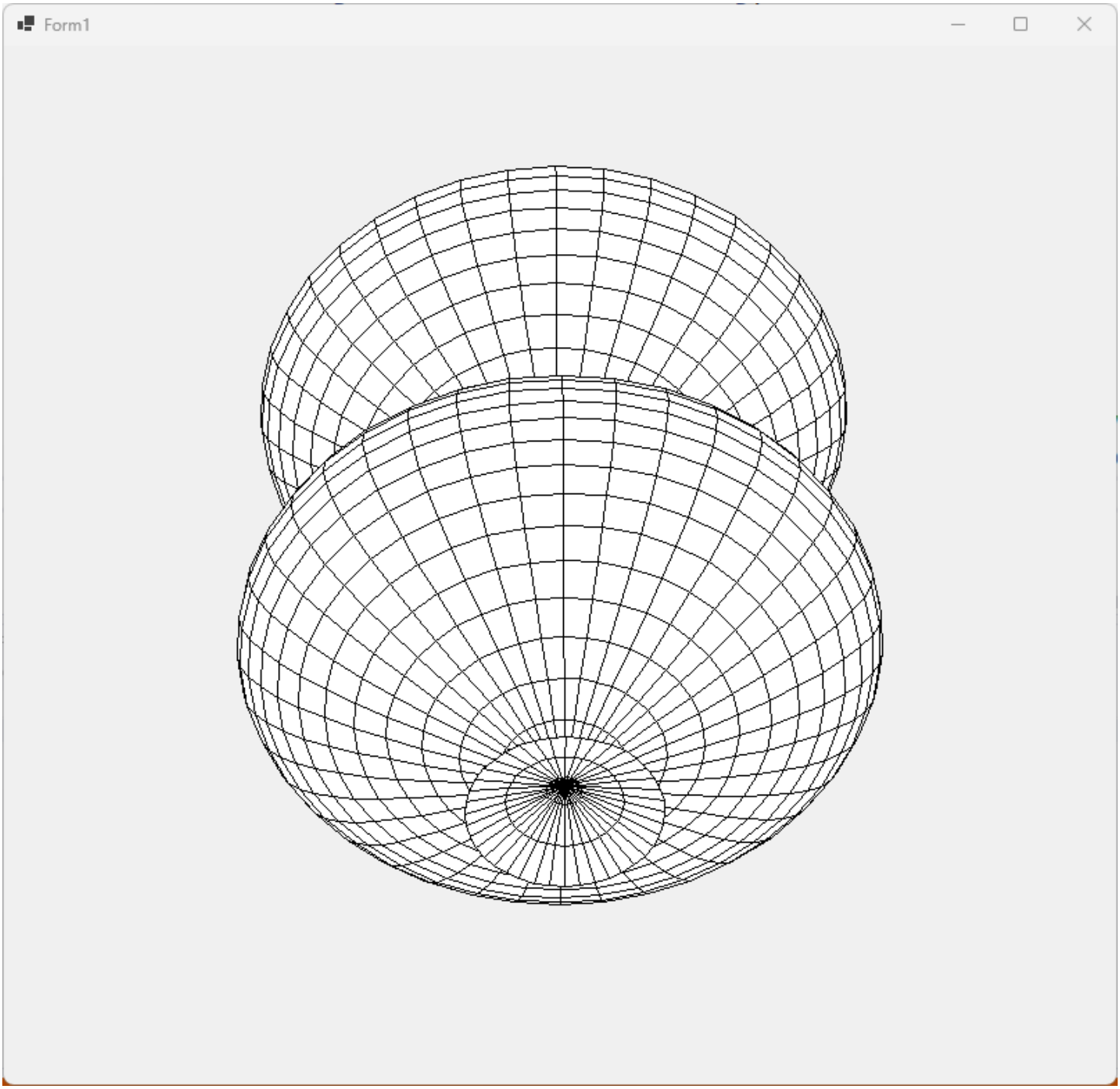

Ilustración 24: Sólido de revolución animado

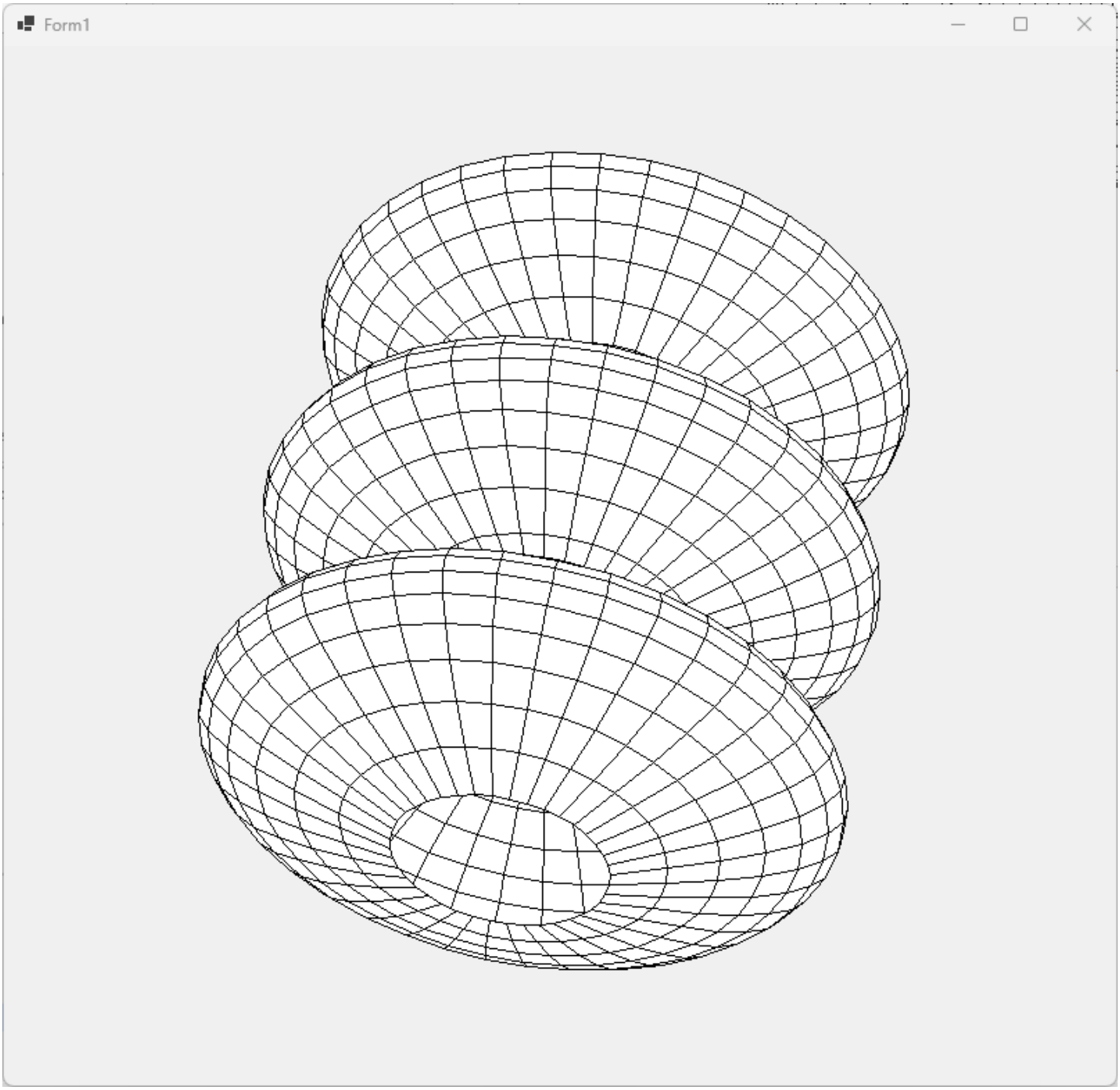

Ilustración 25: Sólido de revolución animado