C# Y .NET 9
Parte 14. Animación

2025-09

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/

En memoria de

Snow

Autor: Rafael Alberto Moreno Parra

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

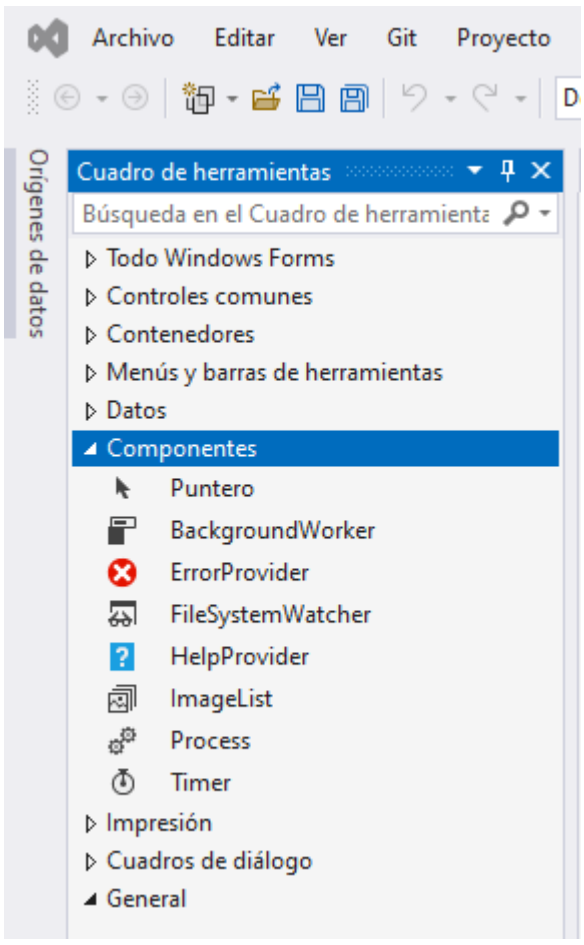

Ilustración 1: Control Timer

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

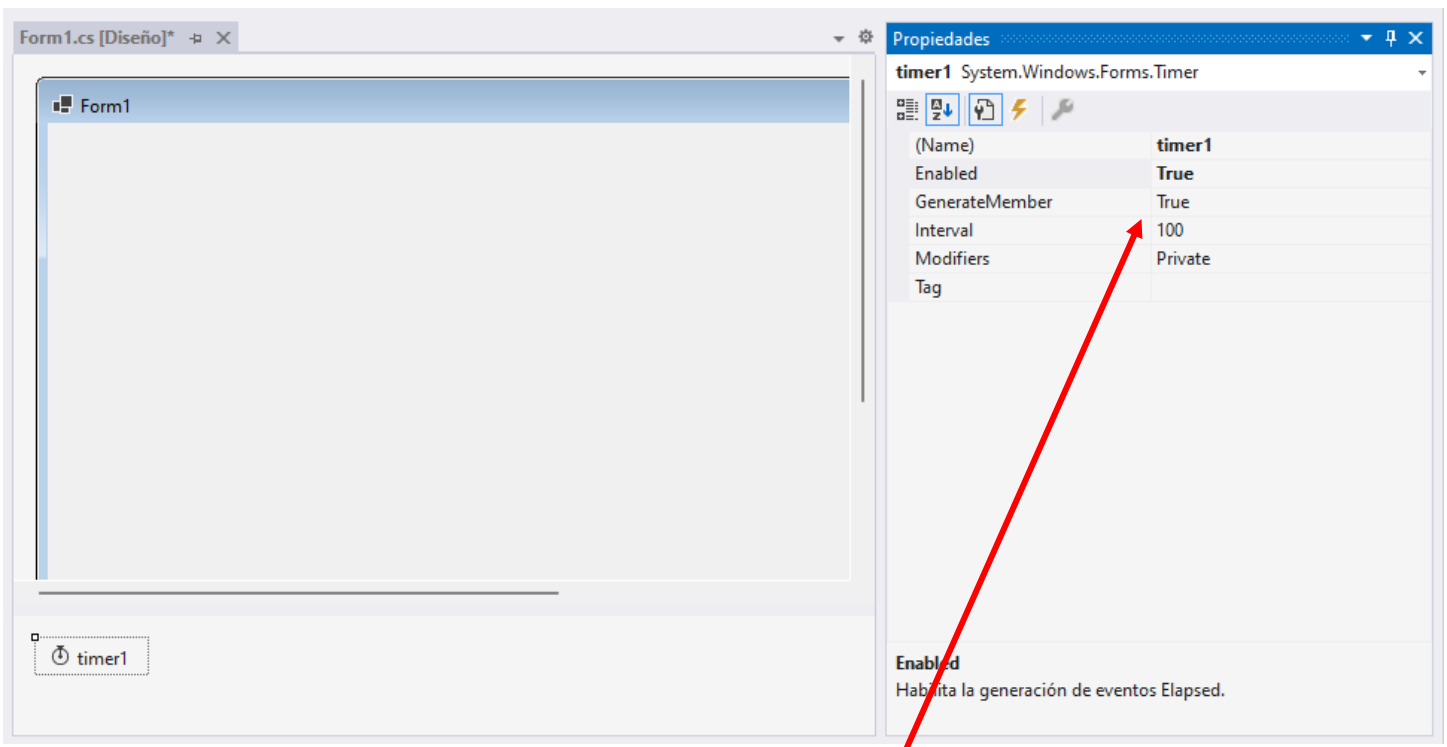

*Ilustración 2: Propiedades del control Timer. Enabled debe estar en **True***

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

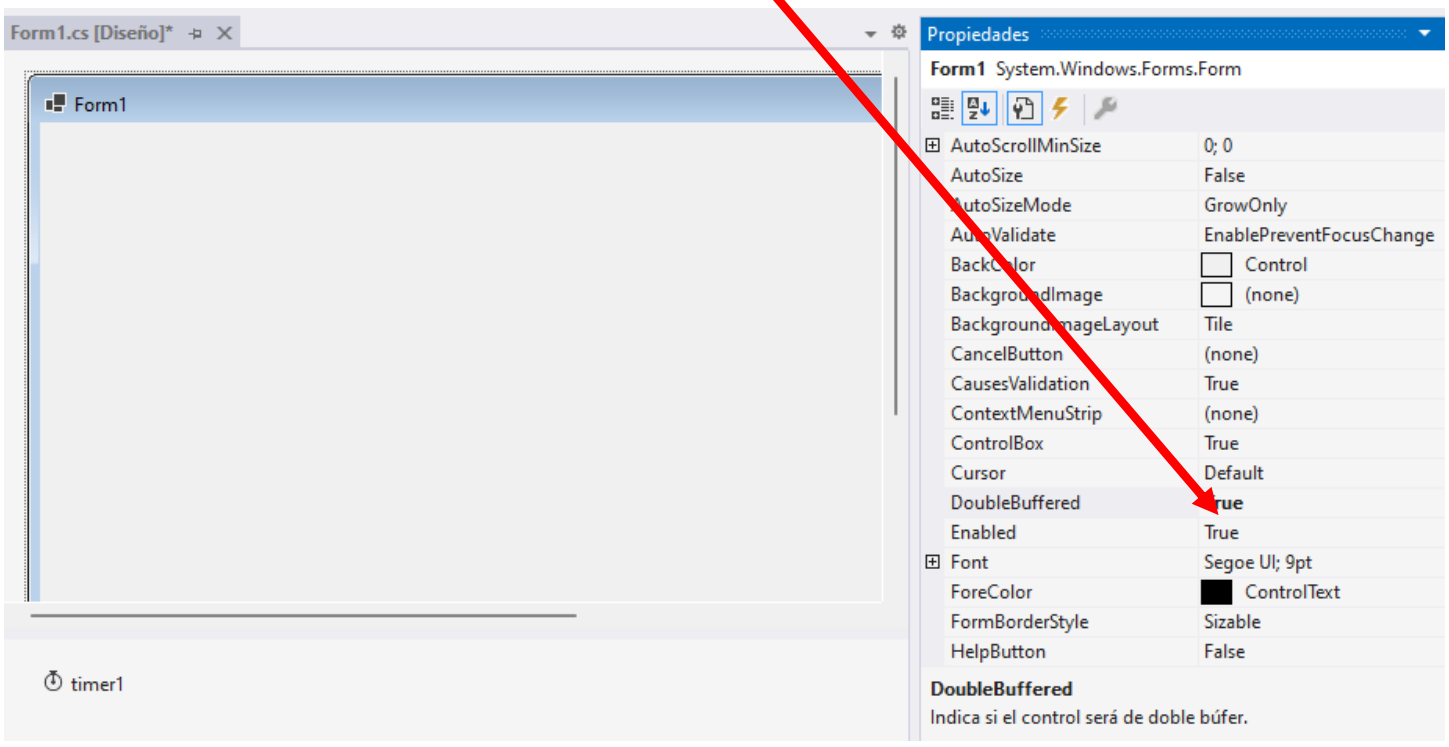

Ilustración 3: Propiedad DoubleBuffered del formulario en True

Ahora se habilita el evento Tick del Timer

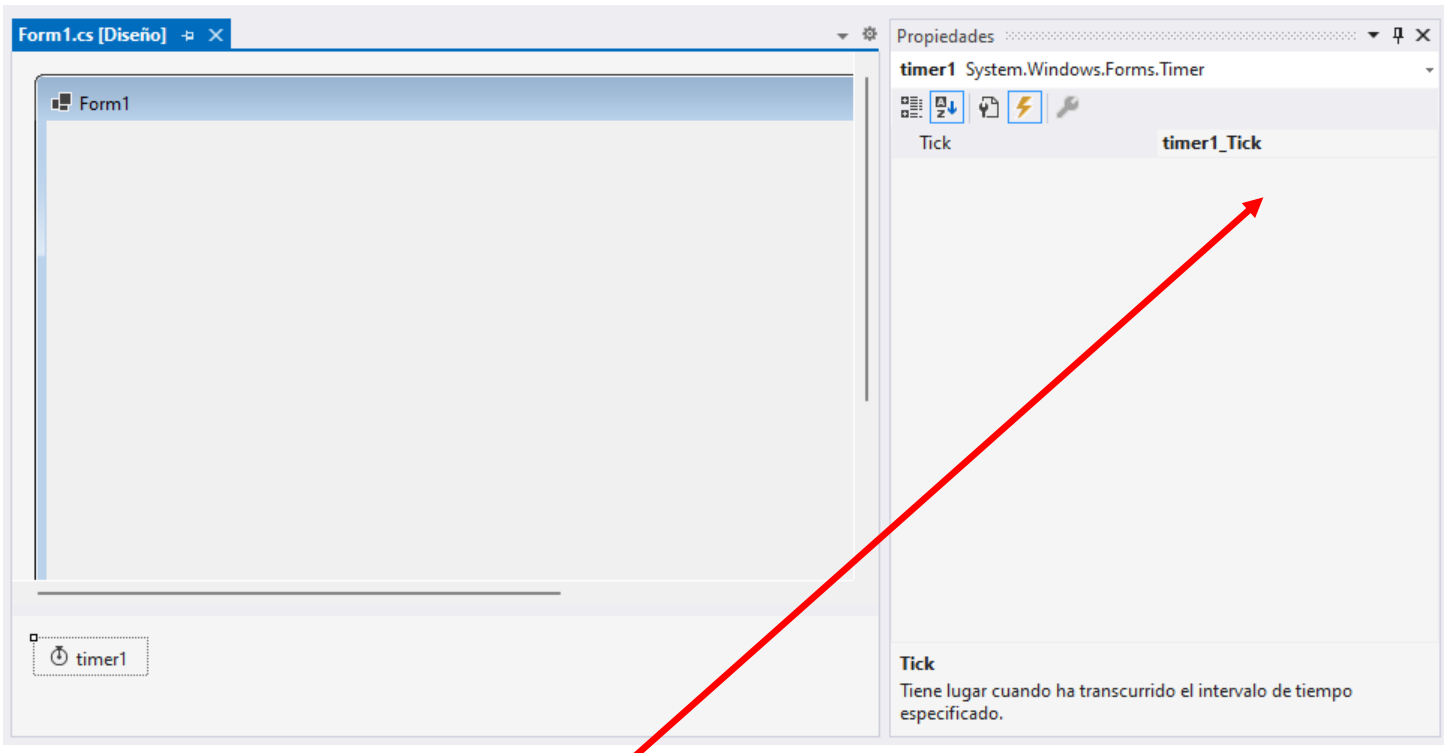

Ilustración 4: Habilitar el evento Tick del Timer

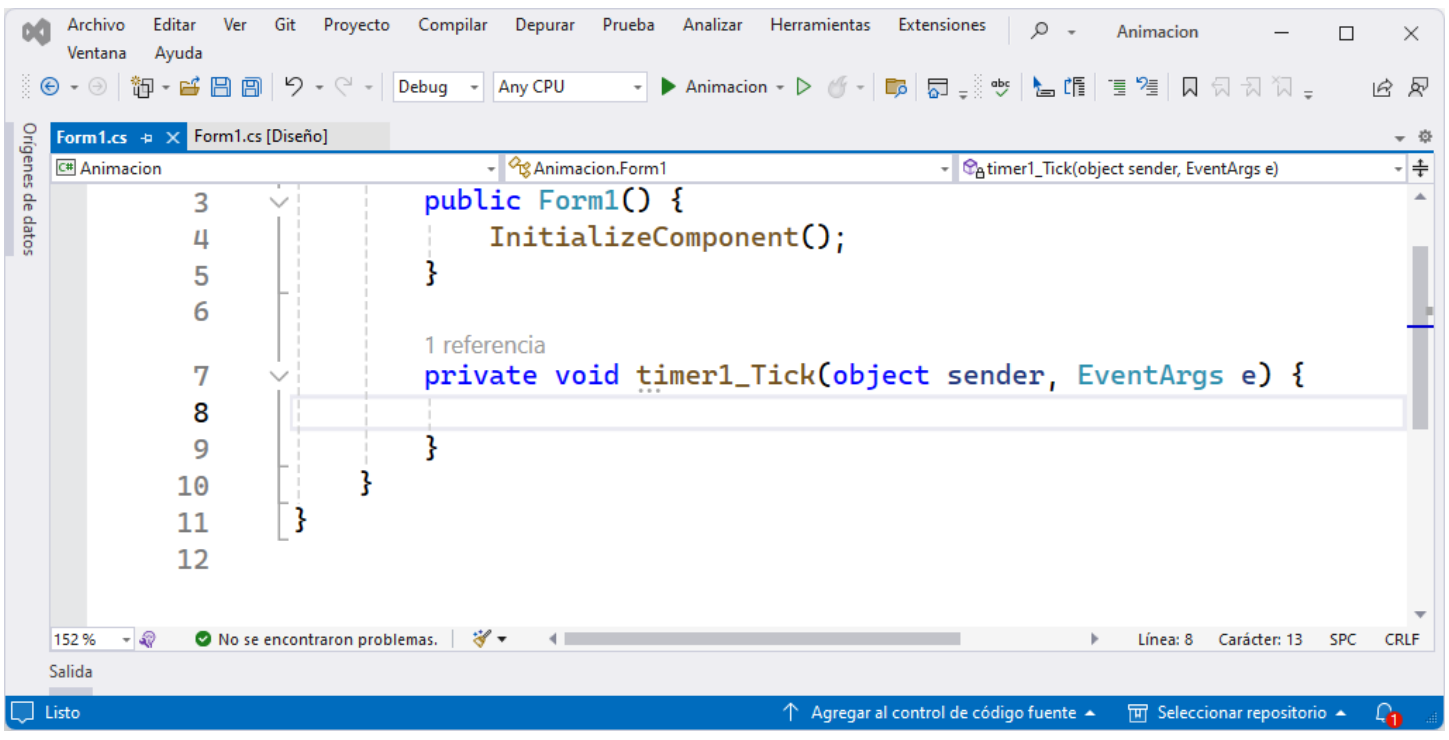

Ilustración 5: Evento para escribir el código

```
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.zip

```
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;

         //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.zip

```
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

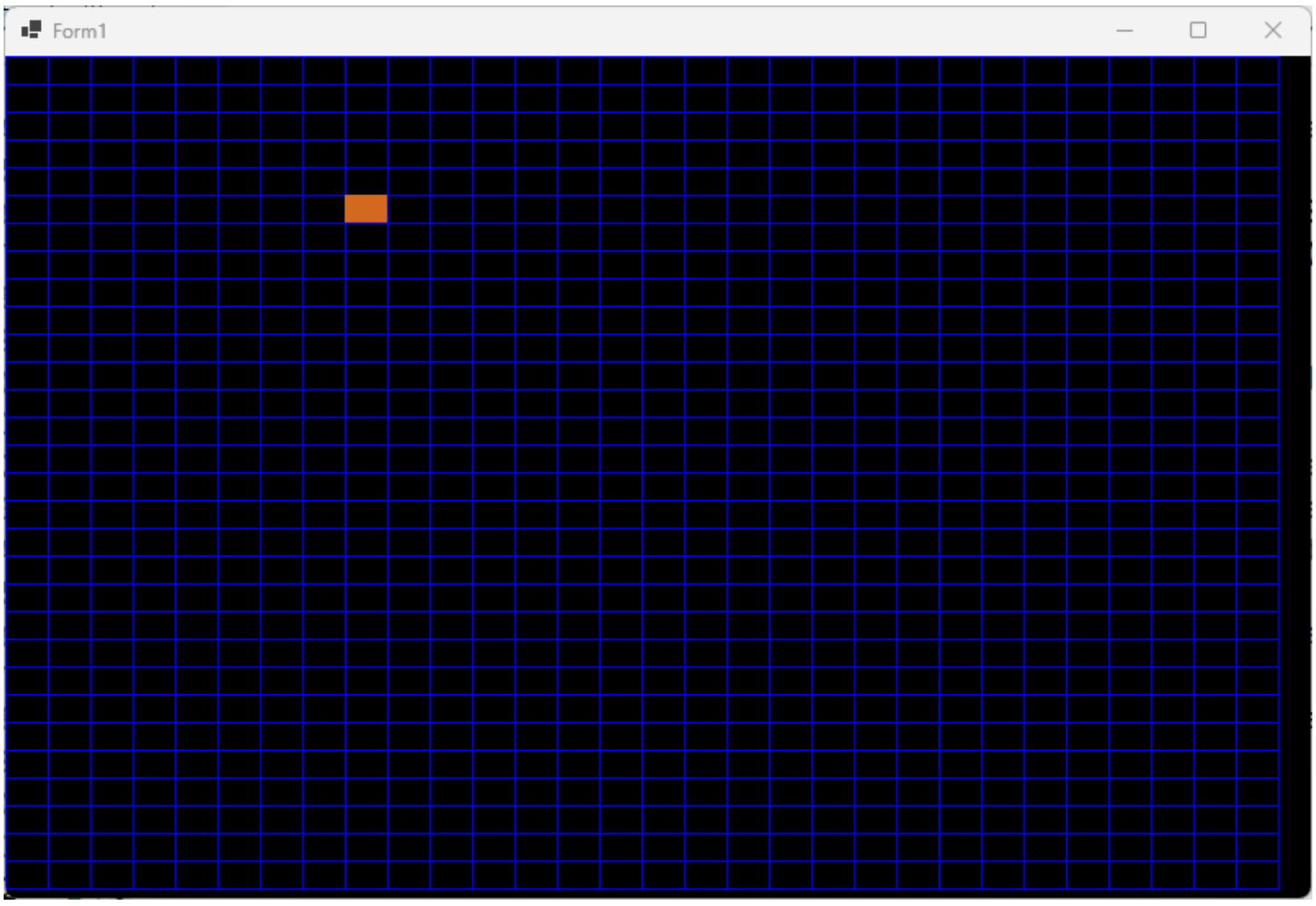

Ilustración 9: Rebote en arreglo bidimensional

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.

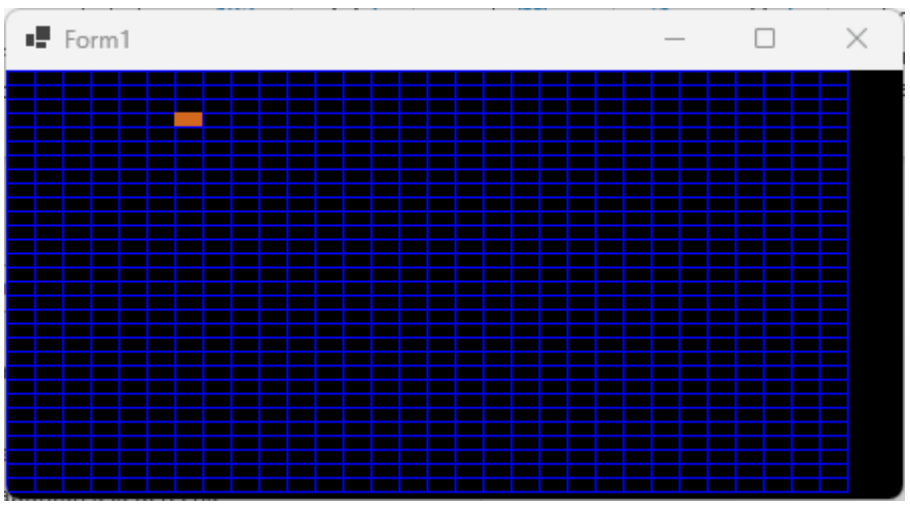

Ilustración 10: Cambiando el tamaño de la ventana

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.zip

```
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;

      //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
```


Autor: Rafael Alberto Moreno Parra

```
      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;
         }
      }
   }
}
```

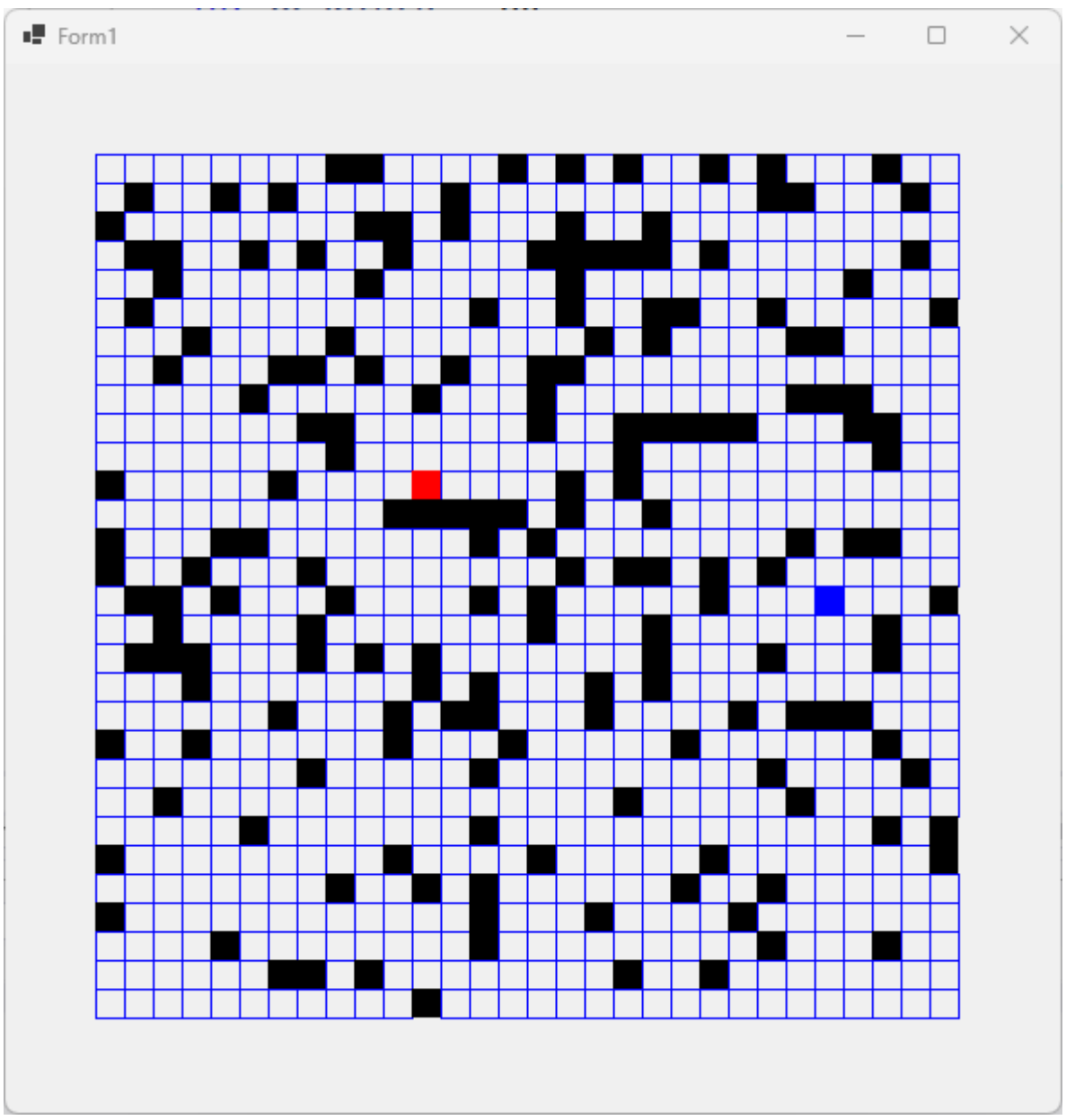

Ilustración 11: Depredador y presa

¡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.zip

```
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();
      }


      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;
            }
         }
      }
   }
}
```

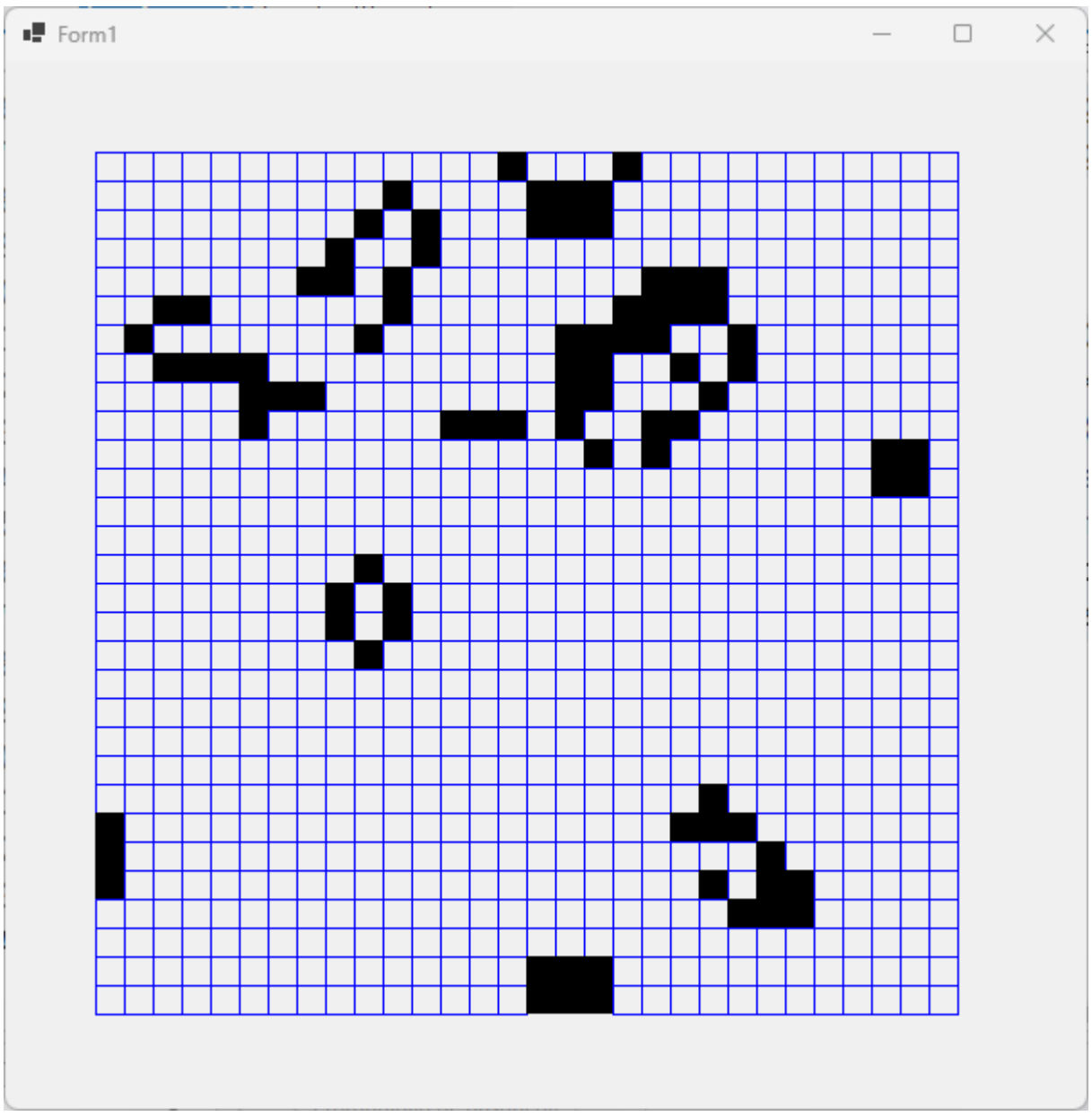

Ilustración 12: Juego de la vida

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.zip

```
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));
         }

         //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++) {
```


Autor: Rafael Alberto Moreno Parra

```
            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;
      }
```

```
    }
}
```

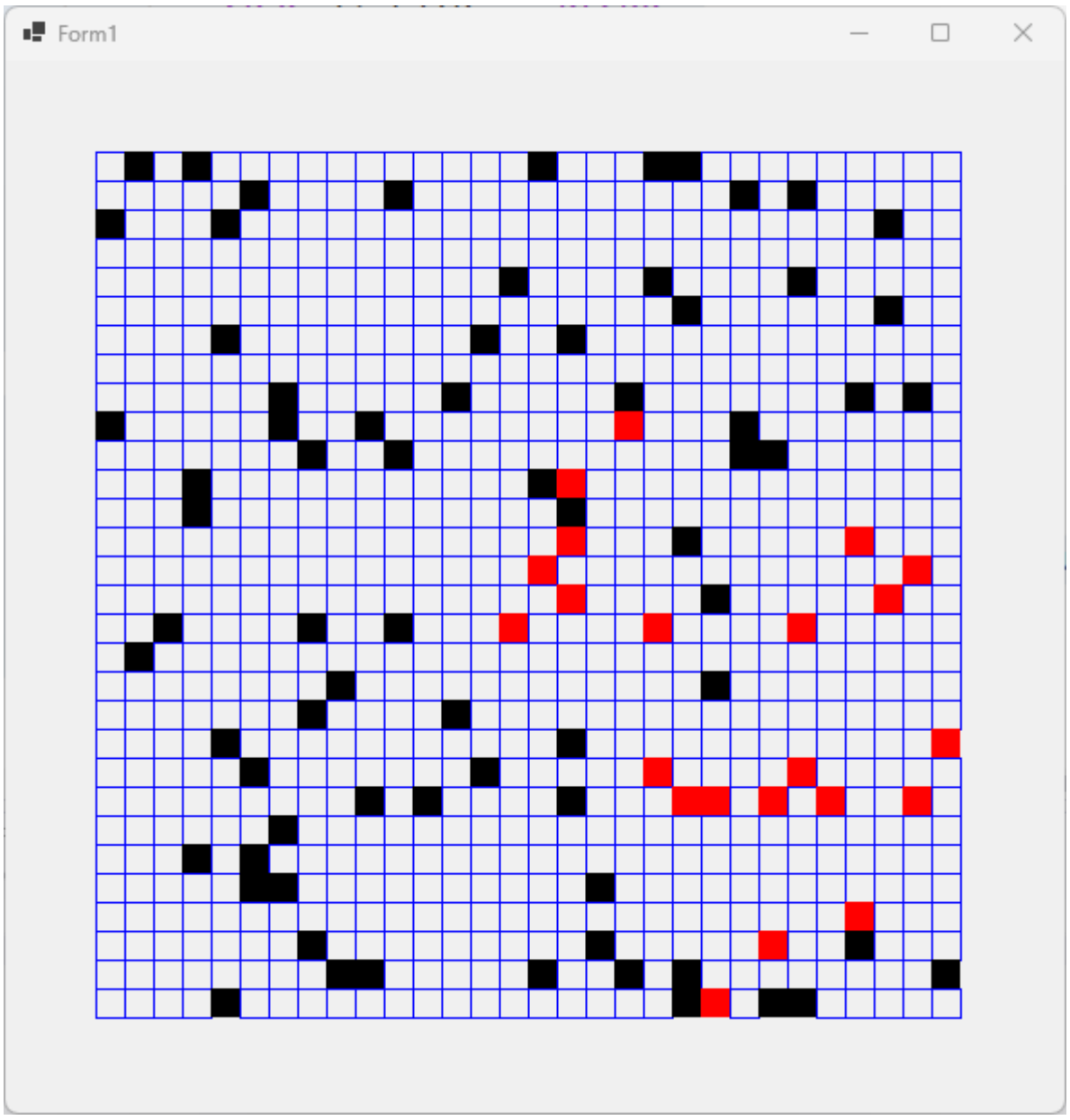

Ilustración 13: Infección propagándose

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.

N/007.zip

```
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;
      }
   }
}
```

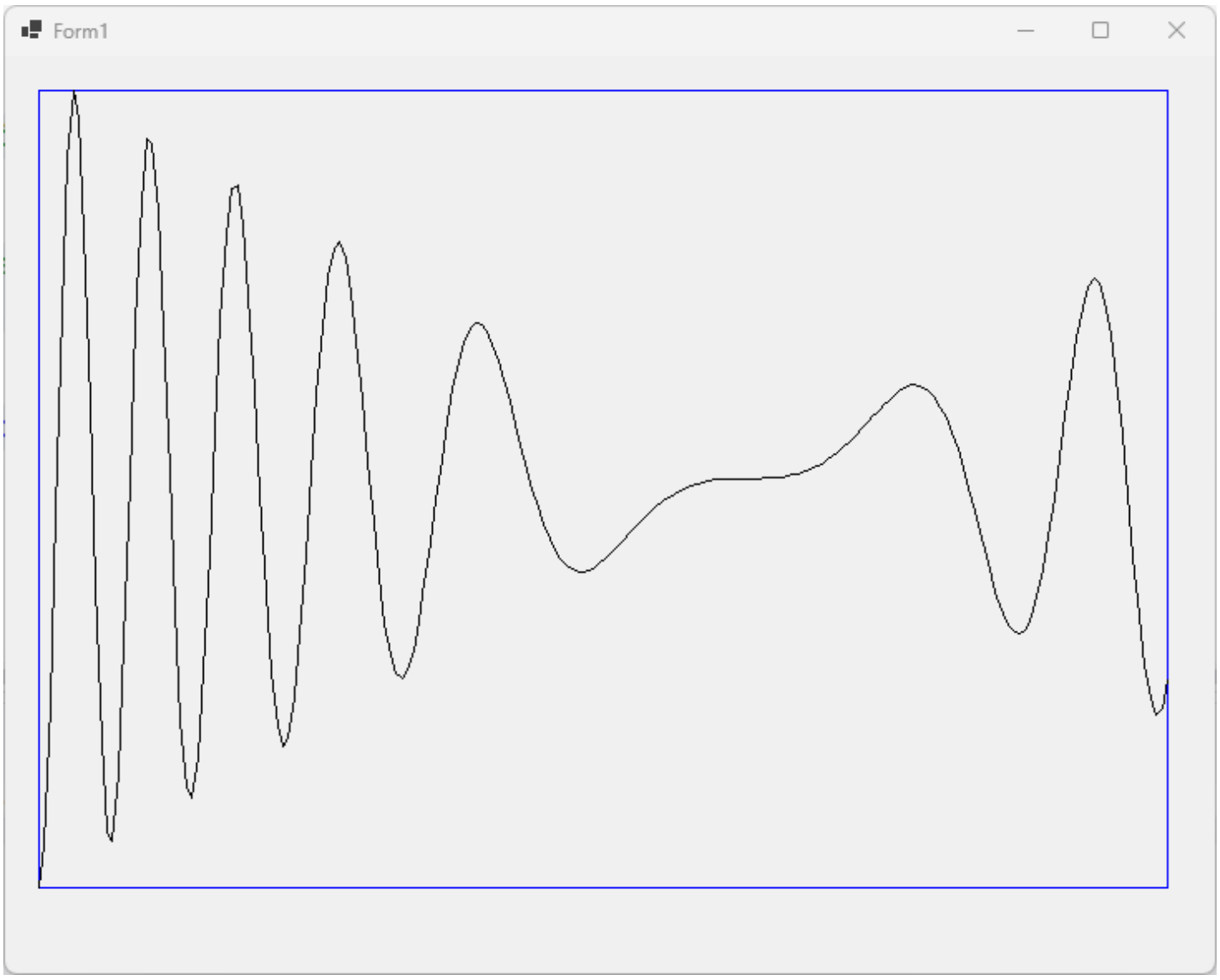

Ilustración 14: Gráfico matemático 2D animado

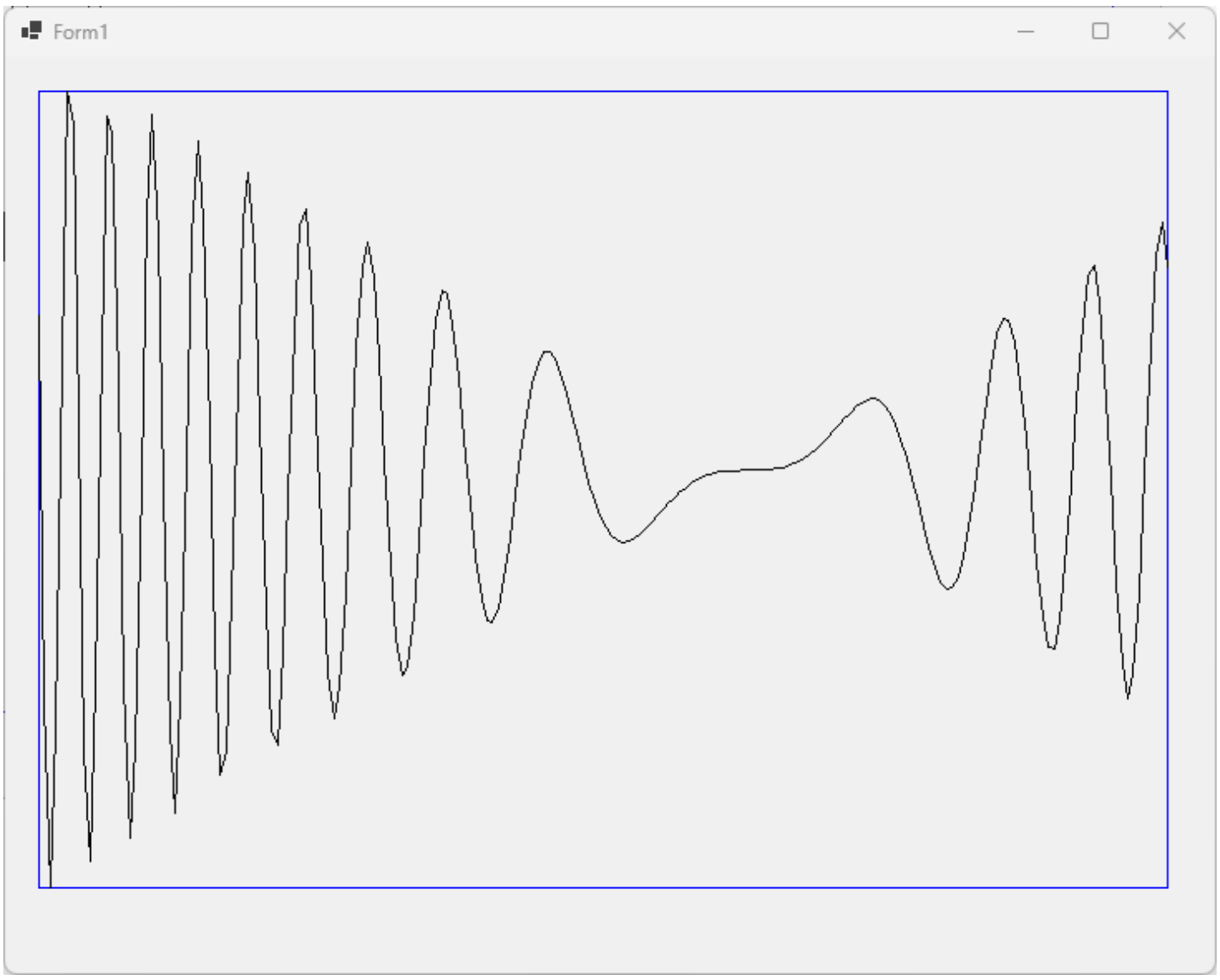

Ilustración 15: Gráfico matemático 2D animado

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.zip

```
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;
      }
   }
}
```

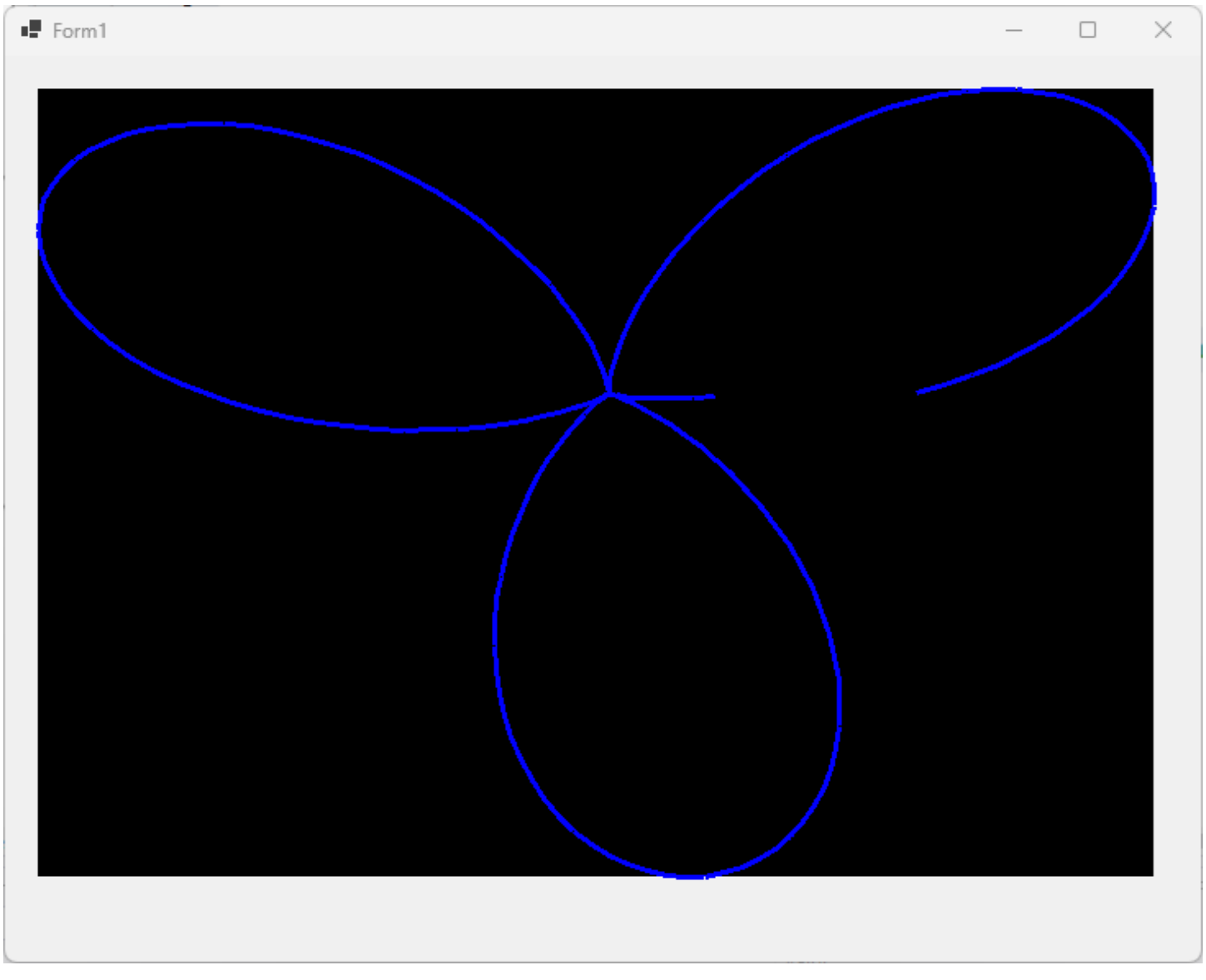

Ilustración 16: Gráfico polar animado

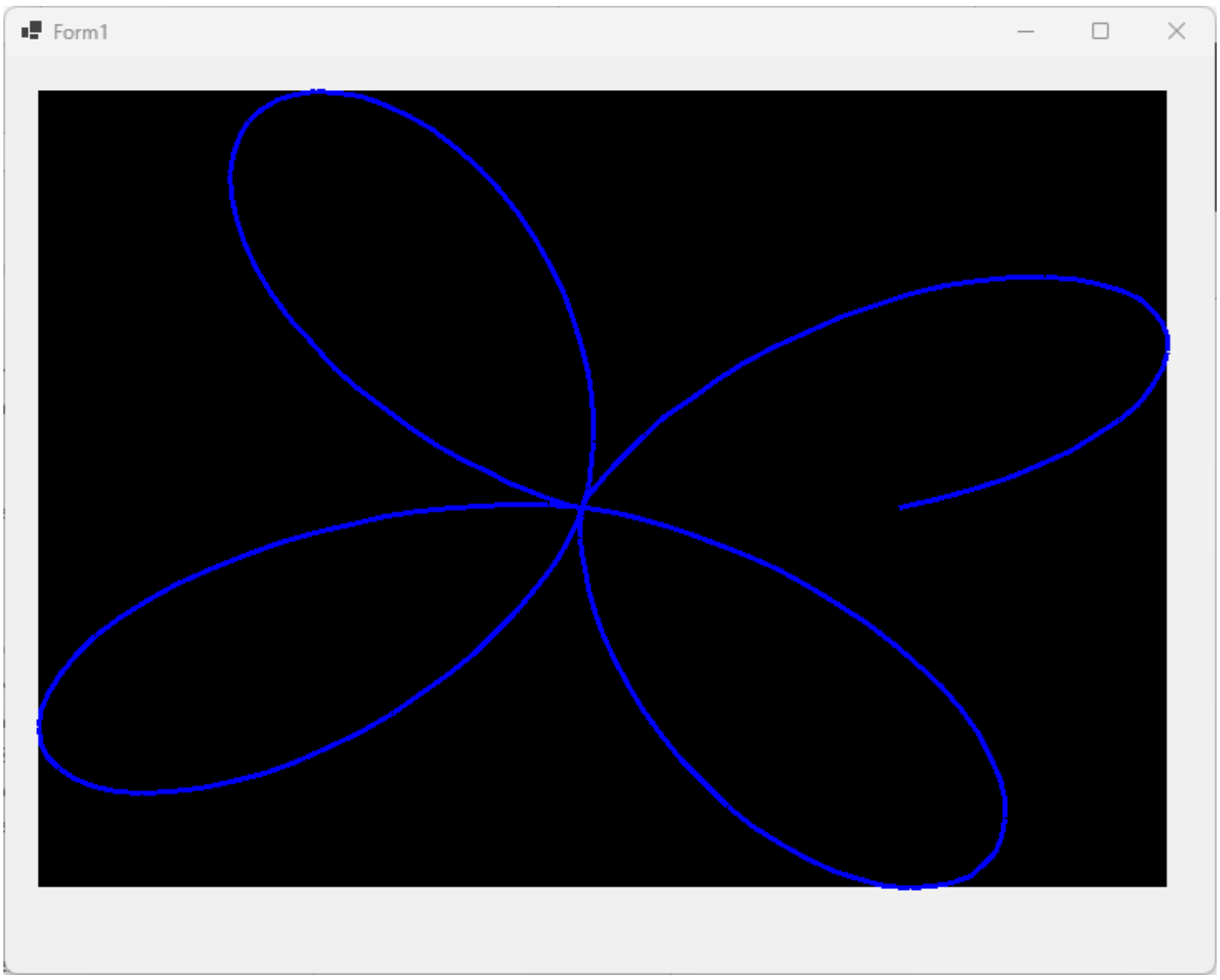

Ilustración 17: Gráfico polar animado

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

N/009.zip

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

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

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

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

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

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

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

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

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

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

      //Color de relleno del polígono
      Color ColorZ;

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

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

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

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

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

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

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

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

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

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

   //Genera listado de colores para el gráfico
   ListaColores = [];
   int Mitad = NumColores / 2;

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

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

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

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

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

   //Valida que la ecuación tenga valores para graficar y de paso calcula
   //las constantes para normalizar
   double Xmin = double.MaxValue;
   double Ymin = double.MaxValue;
   double Zmin = double.MaxValue;
   double Xmax = double.MinValue;
   double Ymax = double.MinValue;
   double Zmax = double.MinValue;
   for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
      for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
         Evaluador.DarValorVariable('x', X);
         Evaluador.DarValorVariable('y', Y);
         double Z = Evaluador.Evaluar();
         if (double.IsNaN(Z) || double.IsInfinity(Z)) Z = 0;

         if (X < Xmin) Xmin = X;
         if (Y < Ymin) Ymin = Y;
         if (Z < Zmin) Zmin = Z;
         if (X > Xmax) Xmax = X;
         if (Y > Ymax) Ymax = Y;
         if (Z > Zmax) Zmax = Z;
```

```
            X += IncrX;
         }
         X = Xini;
         Y += IncrY;
      }

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


      //Coordenadas espaciales X,Y,Z
      X = Xini;
      Y = Yini;
      puntos = [];
      for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
         for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
            Evaluador.DarValorVariable('x', X);
            Evaluador.DarValorVariable('y', Y);
            double Z = Evaluador.Evaluar();
            if (double.IsNaN(Z) || double.IsInfinity(Z)) Z = 0;
            puntos.Add(new Punto(X, Y, Z));
            X += IncrX;
         }
         X = Xini;
         Y += IncrY;
      }

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

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

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

      for (int Cont = 0; Cont < totalPoligonos; Cont++) {
         // Crea un polígono con las coordenadas de los vértices
         poligonos.Add(new Poligono(
            coordenadaActual,
            coordenadaActual + 1,
            coordenadaActual + NumLineas + 1,
            coordenadaActual + NumLineas
         ));

         coordenadaActual++;

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

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

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

```
            {-SinY, SinX * CosY, CosX * CosY }
         };

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

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

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

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

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

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

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

         FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
         FrmGrafico.Yini = Convert.ToDouble((double)numMinimoY.Value);
         FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
         FrmGrafico.Yfin = Convert.ToDouble((double)numMaximoY.Value);
         FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
         FrmGrafico.Ecuacion = txtEcuacion.Text;
         FrmGrafico.ZPersona = 5;

         FrmGrafico.Show();
      }

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

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

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

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

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

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

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

```
         if (numMinimoY.Value >= numMaximoY.Value)
            numMaximoY.Value = numMinimoY.Value + 1;

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

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

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

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

      //Distancia del observador
      public double ZPersona;

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

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

      //Ecuación
      public string Ecuacion;

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


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

         //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);
      }

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Dibuja la figura 3D
         Figura3D.CalcularFigura3D(Ecuacion, Xini, Yini, Xfin, Yfin, NumLineas, AnguloX*Math.PI/180,
AnguloY * Math.PI / 180, AnguloZ * Math.PI / 180, ZPersona, 0, 0, this.ClientSize.Width,
this.ClientSize.Height);
         Figura3D.Dibuja(e.Graphics);
      }

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

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

```
      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();
      }
   }
}
```

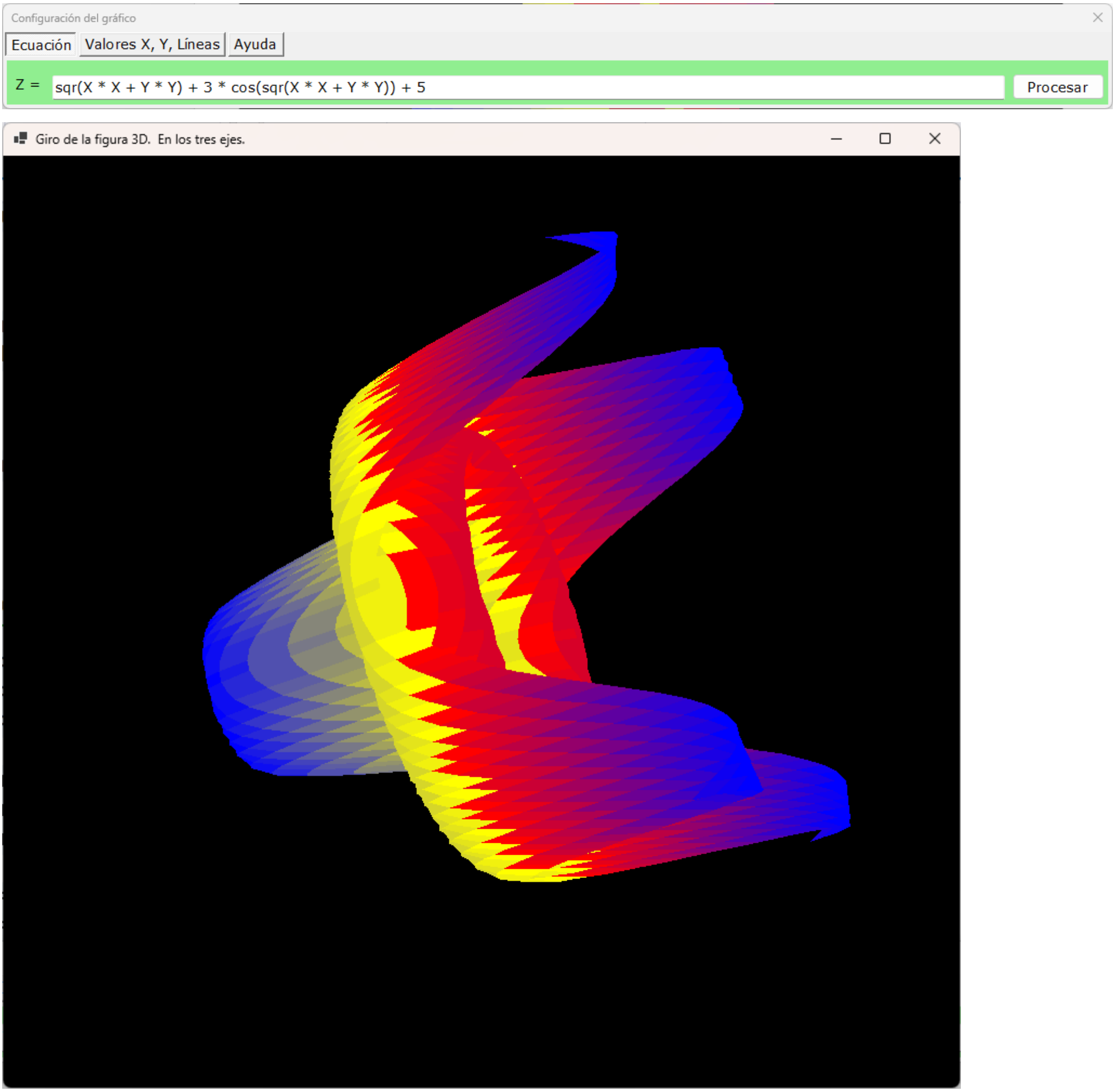

Ilustración 18: Gráfico Matemático en 3D. Giros animados

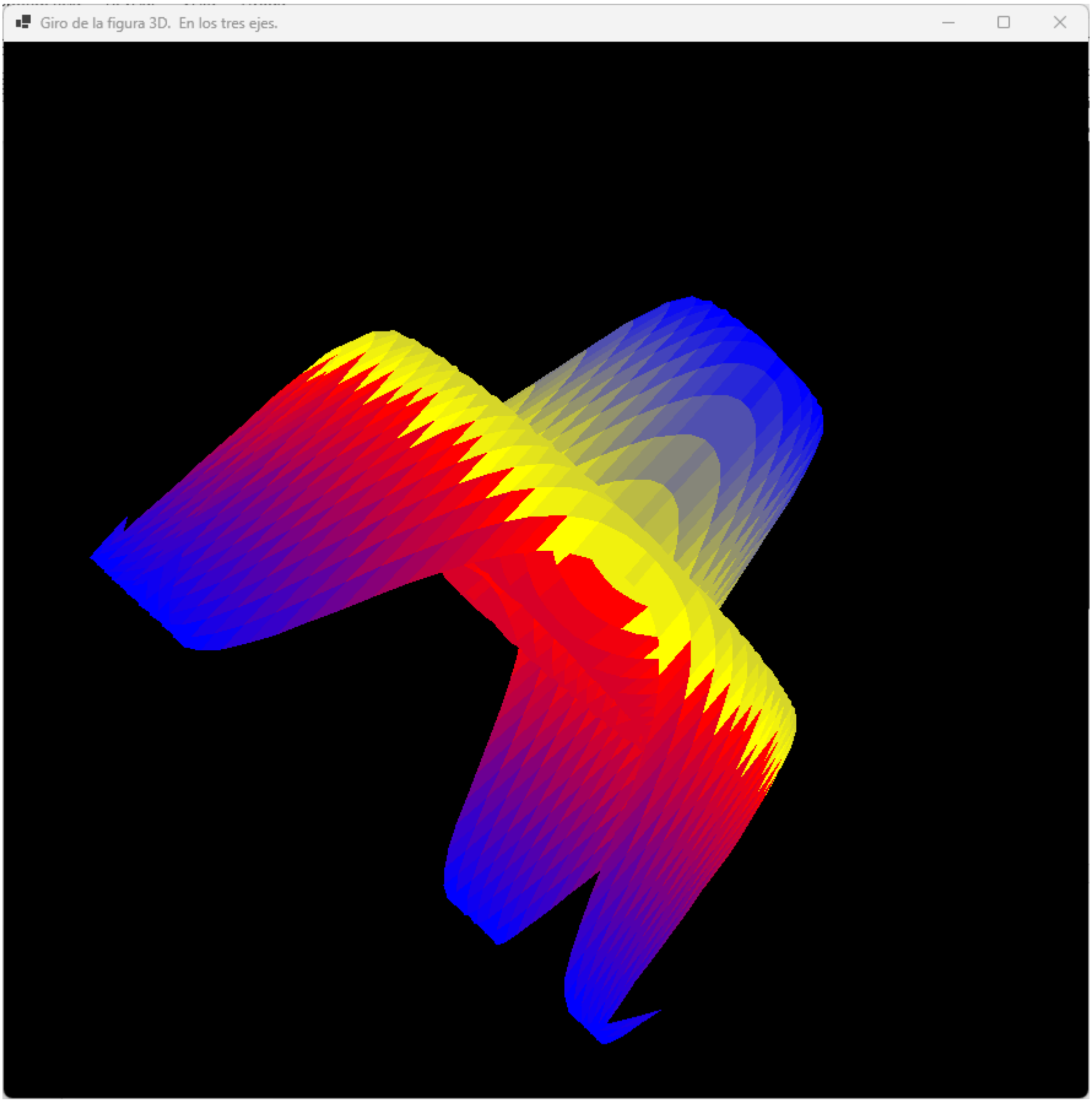

Ilustración 19: Gráfico Matemático en 3D. Giros animados

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

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.zip

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

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

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

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

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

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


Autor: Rafael Alberto Moreno Parra

```
}
```

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

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

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

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

      //Color de relleno del polígono
      Color ColorZ;

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

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

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

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

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

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

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

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

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

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

   //Genera listado de colores para el gráfico
   ListaColores = [];
   int Mitad = NumColores / 2;

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

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

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

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

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

   //Valida que la ecuación tenga valores para graficar y de paso calcula
   //las constantes para normalizar
   double Xmin = double.MaxValue;
   double Ymin = double.MaxValue;
   double Zmin = double.MaxValue;
   double Xmax = double.MinValue;
   double Ymax = double.MinValue;
   double Zmax = double.MinValue;
   for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
      for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
         Evaluador.DarValorVariable('x', X);
         Evaluador.DarValorVariable('y', Y);
         Evaluador.DarValorVariable('t', valorT);
         double Z = Evaluador.Evaluar();
         if (double.IsNaN(Z) || double.IsInfinity(Z)) Z = 0;

         if (X < Xmin) Xmin = X;
         if (Y < Ymin) Ymin = Y;
         if (Z < Zmin) Zmin = Z;
```

```
            if (X > Xmax) Xmax = X;
            if (Y > Ymax) Ymax = Y;
            if (Z > Zmax) Zmax = Z;

            X += IncrX;
         }
         X = Xini;
         Y += IncrY;
      }

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


      //Coordenadas espaciales X,Y,Z
      X = Xini;
      Y = Yini;
      puntos = [];
      for (int EjeY = 1; EjeY <= NumLineas; EjeY++) {
         for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
            Evaluador.DarValorVariable('x', X);
            Evaluador.DarValorVariable('y', Y);
            Evaluador.DarValorVariable('t', valorT);
            double Z = Evaluador.Evaluar();
            if (double.IsNaN(Z) || double.IsInfinity(Z)) Z = 0;
            puntos.Add(new Punto(X, Y, Z));
            X += IncrX;
         }
         X = Xini;
         Y += IncrY;
      }

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

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

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

      for (int Cont = 0; Cont < totalPoligonos; Cont++) {
         // Crea un polígono con las coordenadas de los vértices
         poligonos.Add(new Poligono(
            coordenadaActual,
            coordenadaActual + 1,
            coordenadaActual + NumLineas + 1,
            coordenadaActual + NumLineas
         ));

         coordenadaActual++;

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

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

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

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

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

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

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

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

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

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

         FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
         FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
         FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
         FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
         FrmGrafico.Yini = Convert.ToDouble((double)numMinimoY.Value);
         FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
         FrmGrafico.Yfin = Convert.ToDouble((double)numMaximoY.Value);
         FrmGrafico.Tini = Convert.ToDouble((double)numMinimoT.Value);
         FrmGrafico.Tfin = Convert.ToDouble((double)numMaximoT.Value);
         FrmGrafico.Tavance = Convert.ToDouble((double)numAvanceT.Value);


         FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
         FrmGrafico.Ecuacion = txtEcuacion.Text;
         FrmGrafico.ZPersona = 5;

         FrmGrafico.Show();
      }

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

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

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

```
      FrmGrafico.Refresh();
   }

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

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

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

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

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

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

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

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

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

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

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

      FrmGrafico.Tini = Convert.ToDouble((double)numMinimoT.Value);
      FrmGrafico.valorT = FrmGrafico.Tini + 0.01;
      FrmGrafico.Refresh();
   }

      }

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

         FrmGrafico.Tfin = Convert.ToDouble((double)numMaximoT.Value);
         FrmGrafico.valorT = FrmGrafico.Tini + 0.01;
         FrmGrafico.Refresh();
      }

      private void numAvanceT_ValueChanged(object sender, EventArgs e) {
         FrmGrafico.Tavance = Convert.ToDouble((double)numAvanceT.Value);
         FrmGrafico.Refresh();
      }
   }
}
```

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

```
Objeto3D Figura3D;

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

//Distancia del observador
public double ZPersona;

//Rango de valores
public double Xini, Yini, Xfin, Yfin, Tini, Tfin, Tavance, valorT;

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

//Ecuación
public string Ecuacion;

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

   valorT = Tini+0.01;
}

private void Form1_Paint(object sender, PaintEventArgs e) {

      //Dibuja la figura 3D
      Figura3D.CalcularFigura3D(Ecuacion, Xini, Yini, Xfin, Yfin, NumLineas, valorT, AnguloX, AnguloY,
AnguloZ, ZPersona, 0, 0, this.ClientSize.Width, this.ClientSize.Height);
      Figura3D.Dibuja(e.Graphics);
   }

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

   private void Form1_Resize(object sender, EventArgs e) {
      Refresh();
   }

   private void timer1_Tick(object sender, EventArgs e) {
      valorT += Tavance;
      if (valorT >= Tfin) {
         valorT = Tfin;
         Tavance = -Tavance;
      }

      if (valorT <= Tini) {
         valorT = Tini;
         Tavance = -Tavance;
      }

      Refresh();
   }
}
}
```

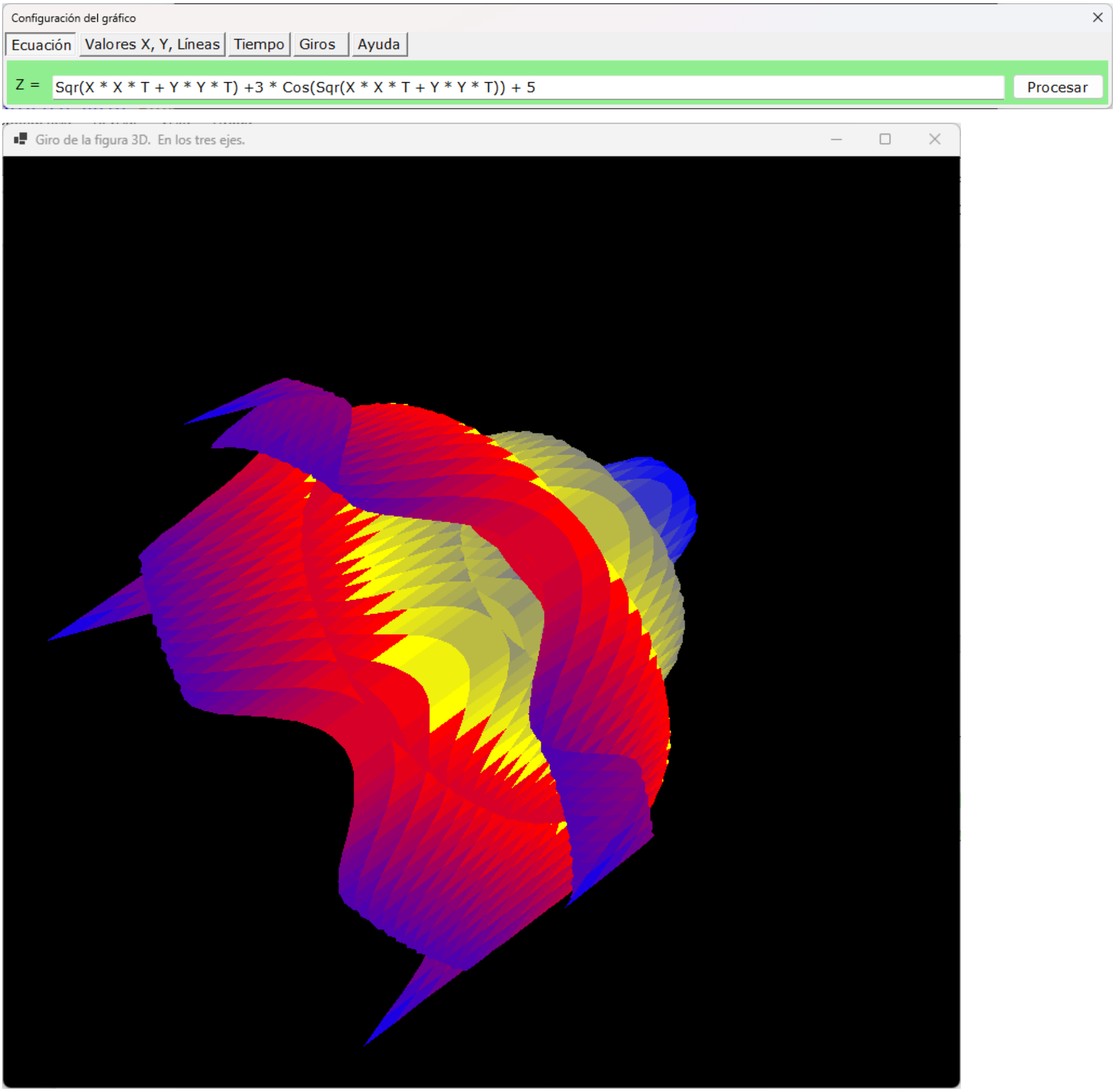

Ilustración 20: Gráfico Matemático en 4D

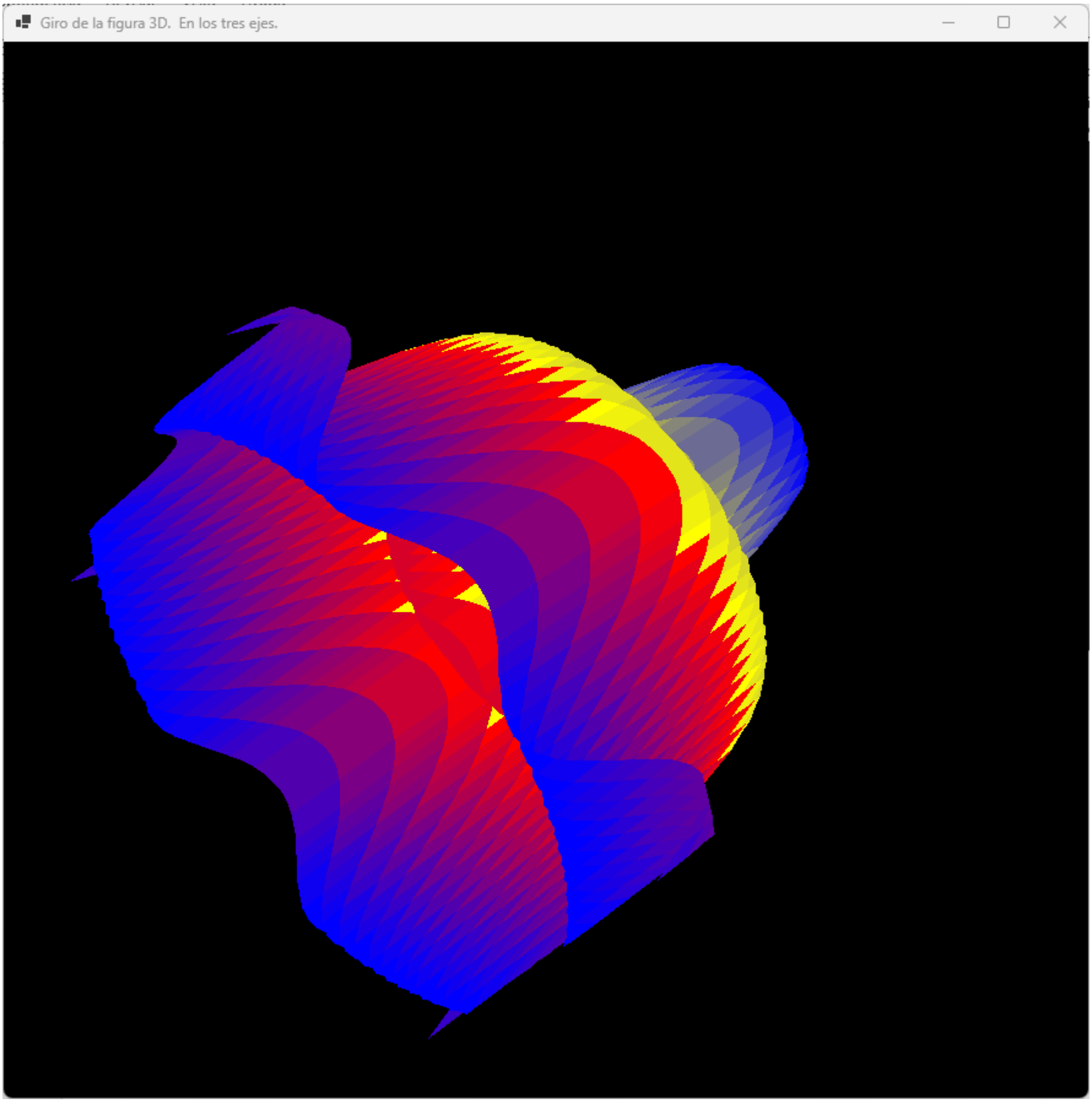

Ilustración 21: Gráfico Matemático en 4D

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.zip

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

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

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

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

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

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

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

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

```
namespace Graficos {
   //Triángulo
   internal class Poligono {
```

```
public int punto1, punto2, punto3, punto4;
public double Centro;

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

//Color de relleno del polígono
Color ColorZ;

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

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

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

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

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

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

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

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

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

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

         //Genera listado de colores para el gráfico
         ListaColores = [];
         int Mitad = NumColores / 2;

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

```
            ListaColores.Add(Color.FromArgb(Rojo, Verde, Azul));
         }

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

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

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

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

         //Valida que la ecuación tenga valores para graficar y de paso calcula
         //las constantes para normalizar
         double Xmin = double.MaxValue;
         double Ymin = double.MaxValue;
         double Zmin = double.MaxValue;
         double Xmax = double.MinValue;
         double Ymax = double.MinValue;
         double Zmax = double.MinValue;
         for (int AngTheta = 1; AngTheta <= NumLineas; AngTheta++) {
            for (int AngPhi = 1; AngPhi <= NumLineas; AngPhi++) {
               double ThetaR = Theta * Math.PI / 180;
               double PhiR = Phi * Math.PI / 180;

               Evaluador.DarValorVariable('p', PhiR);
               Evaluador.DarValorVariable('h', ThetaR);
               Evaluador.DarValorVariable('t', valorT);
               double R = Evaluador.Evaluar();
               if (double.IsNaN(R) || double.IsInfinity(R)) R = 0;

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

               if (X < Xmin) Xmin = X;
               if (Y < Ymin) Ymin = Y;
               if (Z < Zmin) Zmin = Z;
               if (X > Xmax) Xmax = X;
               if (Y > Ymax) Ymax = Y;
               if (Z > Zmax) Zmax = Z;

               Theta += IncrTheta;
            }
            Theta = MinTheta;
            Phi += IncrPhi;
         }

         if ( Math.Abs(Xmin-Xmax) < 0.0001 || Math.Abs(Ymin-Ymax) < 0.0001 || Math.Abs(Zmin - Zmax) <
0.0001) {
```

```
            MessageBox.Show("La ecuación digitada no puede generar un gráfico", "Error de cálculo",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }


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

                Evaluador.DarValorVariable('p', PhiR);
                Evaluador.DarValorVariable('h', ThetaR);
                Evaluador.DarValorVariable('t', valorT);
                double R = Evaluador.Evaluar();
                if (double.IsNaN(R) || double.IsInfinity(R)) R = 0;

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

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

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

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

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

        for (int Cont = 0; Cont < totalPoligonos; Cont++) {
            // Crea un polígono con las coordenadas de los vértices
            poligonos.Add(new Poligono(
                coordenadaActual,
                coordenadaActual + 1,
                coordenadaActual + NumLineas + 1,
                coordenadaActual + NumLineas
            ));

            coordenadaActual++;

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

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

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

```
            { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
            {-SinY, SinX * CosY, CosX * CosY }
        };

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

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

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

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

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

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

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

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

            FrmGrafico.Show();
        }

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

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

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

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

```
            FrmGrafico.Refresh();
        }

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

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

        //Giro de figura
        public double AnguloX, AnguloY, AnguloZ, Tini, Tfin, Tavance, valorT;

        //Distancia del observador
        public double ZPersona;

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

        //Ecuación
        public string Ecuacion;

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

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

            valorT = Tini + 0.01;
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Figura3D.CalcularFigura3D(Ecuacion, NumLineas, AnguloX, AnguloY, AnguloZ, valorT, ZPersona, 0, 0,
this.ClientSize.Width, this.ClientSize.Height);
            Figura3D.Dibuja(e.Graphics);
        }

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

        private void Form1_Resize(object sender, EventArgs e) {
            Refresh();
        }

        private void timer1_Tick(object sender, EventArgs e) {
            valorT += Tavance;
            if (valorT >= Tfin) {
                valorT = Tfin;
                Tavance = -Tavance;
            }

            if (valorT <= Tini) {
                valorT = Tini;
                Tavance = -Tavance;
            }

            Refresh();
        }
    }
}
```

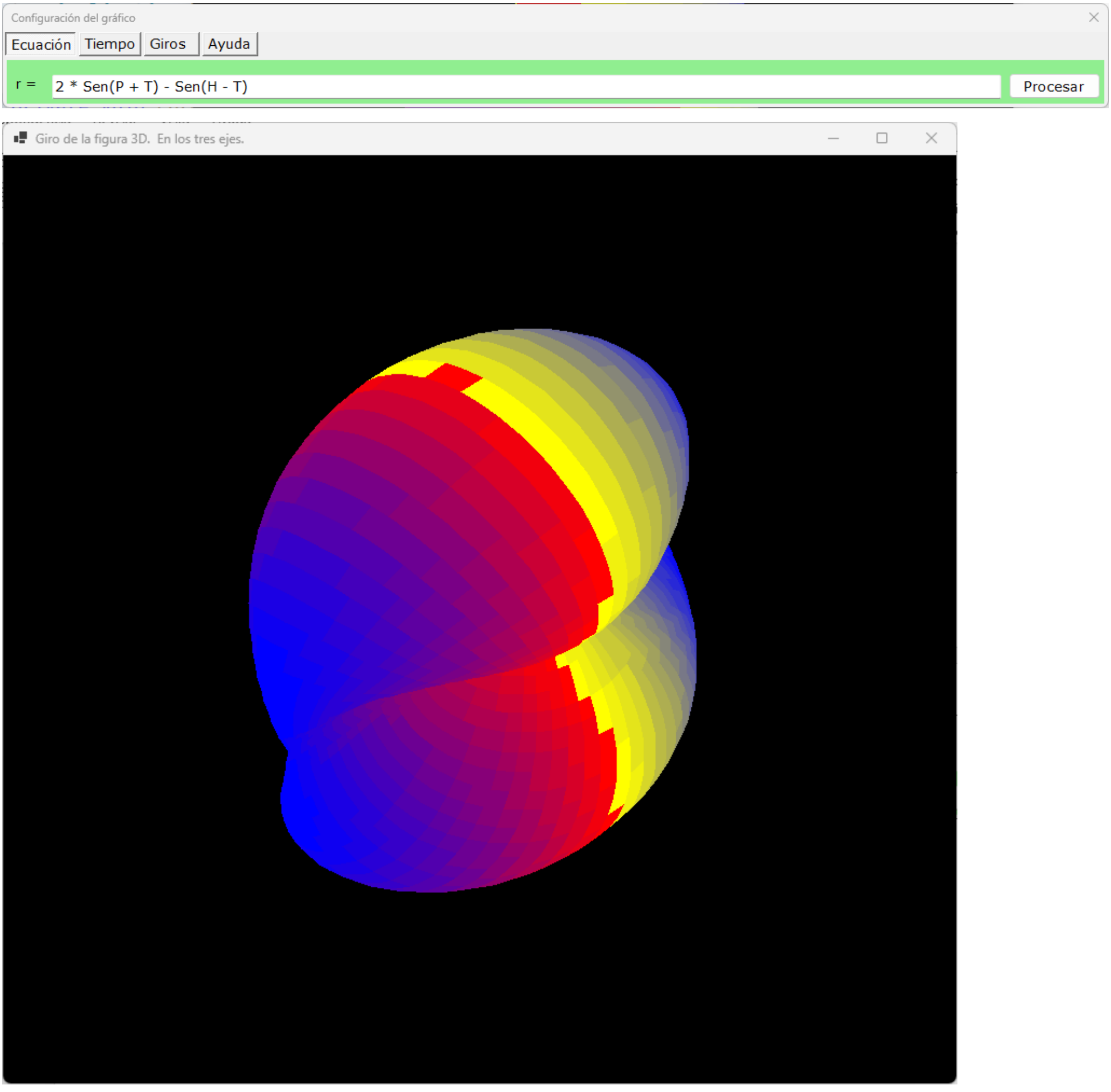

Ilustración 22: Gráfico polar 4D

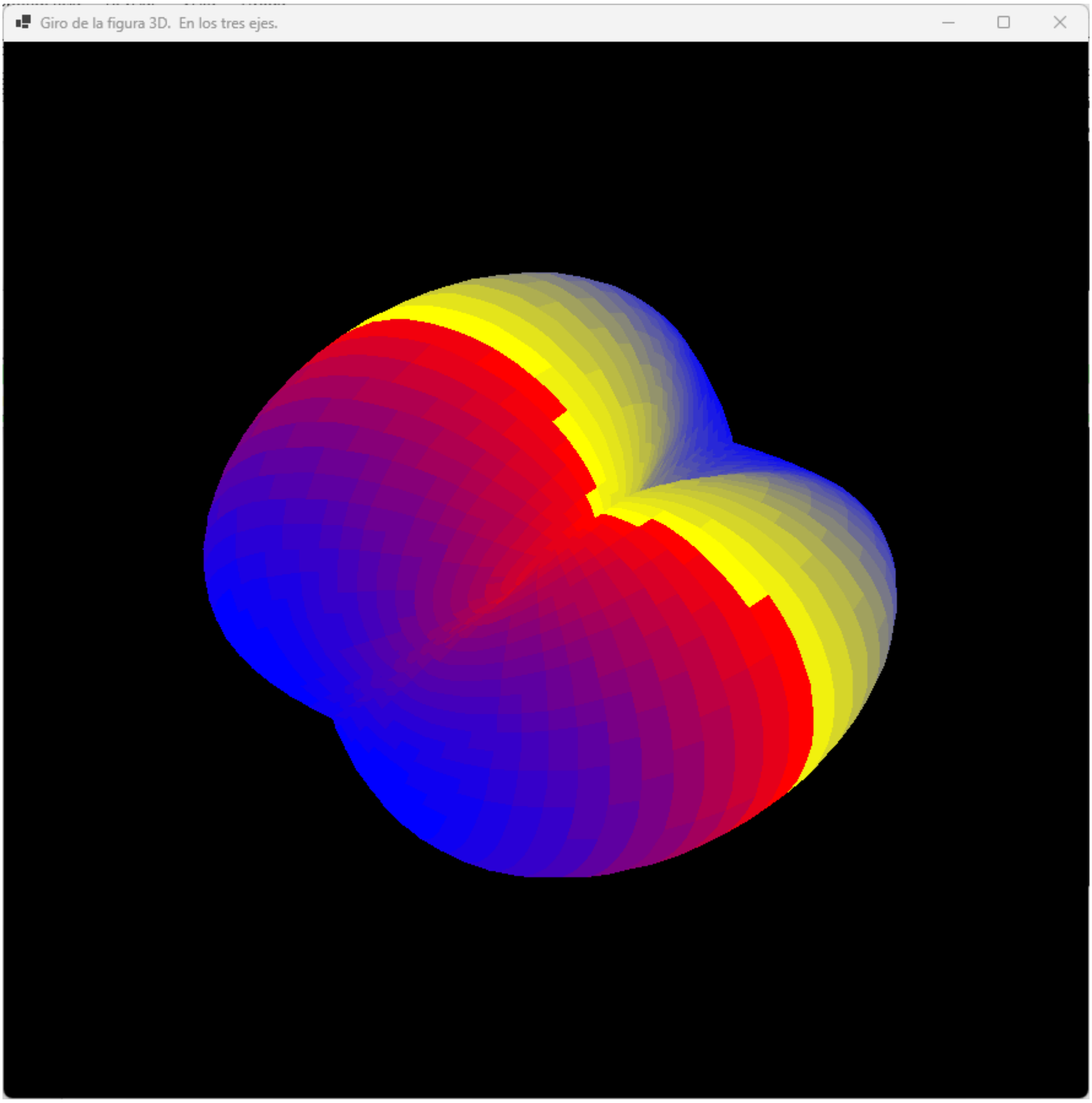

Ilustración 23: Gráfico polar 4D

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.zip

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

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

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

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

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

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

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

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

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

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

      //Color de relleno del polígono
      Color ColorZ;

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

```
      }

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

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

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

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

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

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

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

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

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

         //Genera listado de colores para el gráfico
         ListaColores = [];
         int Mitad = NumColores / 2;

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

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

      //Hace los cálculos de la ecuación Z = F(X,Y)
      //para dibujarla
```

```
     public void CalcularFigura3D(string Ecuacion, double Xini, double Xfin, int NumLineas, double valorT,
double angX, double angY, double angZ, double ZPersona, int XpantallaIni, int YpantallaIni, int
XpantallaFin, int YpantallaFin) {

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

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

        //Valida que la ecuación tenga valores para graficar y de paso calcula
        //las constantes para normalizar
        double Xmin = double.MaxValue;
        double Ymin = double.MaxValue;
        double Zmin = double.MaxValue;
        double Xmax = double.MinValue;
        double Ymax = double.MinValue;
        double Zmax = double.MinValue;

        double AnguloGiro = 0;
        for (int GiroLinea = 1; GiroLinea <= NumLineas + 1; GiroLinea++) {
           for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
              Evaluador.DarValorVariable('x', X);
              Evaluador.DarValorVariable('t', valorT);
              double Y = Evaluador.Evaluar();
              if (double.IsNaN(Y) || double.IsInfinity(Y)) Y = 0;

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

              if (X < Xmin) Xmin = X;
              if (Yg < Ymin) Ymin = Yg;
              if (Zg < Zmin) Zmin = Zg;

              if (X > Xmax) Xmax = X;
              if (Yg > Ymax) Ymax = Yg;
              if (Zg > Zmax) Zmax = Zg;

              X += IncrX;
           }
           AnguloGiro += IncAng;
           X = Xini;
        }

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


        //Coordenadas espaciales X,Y,Z
        X = Xini;
        puntos = [];
        AnguloGiro = 0;
        for (int GiroLinea = 1; GiroLinea <= NumLineas+1; GiroLinea++) {
           for (int EjeX = 1; EjeX <= NumLineas; EjeX++) {
              Evaluador.DarValorVariable('x', X);
              Evaluador.DarValorVariable('t', valorT);
              double Y = Evaluador.Evaluar();
              if (double.IsNaN(Y) || double.IsInfinity(Y)) Y = 0;

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

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

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

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

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

    for (int Cont = 0; Cont < totalPoligonos; Cont++) {
        // Crea un polígono con las coordenadas de los vértices
        poligonos.Add(new Poligono(
            coordenadaActual,
            coordenadaActual + 1,
            coordenadaActual + NumLineas + 1,
            coordenadaActual + NumLineas
        ));

        coordenadaActual++;

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

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

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

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

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

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

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

//Dibuja la figura 3D
public void Dibuja(Graphics lienzo) {
```

```
         for (int Cont = 0; Cont < poligonos.Count; Cont++)
            poligonos[Cont].Dibuja(lienzo);
      }
   }
}
```

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

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

         FrmGrafico.AnguloX = Convert.ToDouble(numGiroX.Value) * Radianes;
         FrmGrafico.AnguloY = Convert.ToDouble(numGiroY.Value) * Radianes;
         FrmGrafico.AnguloZ = Convert.ToDouble(numGiroZ.Value) * Radianes;
         FrmGrafico.Xini = Convert.ToDouble((double)numMinimoX.Value);
         FrmGrafico.Xfin = Convert.ToDouble((double)numMaximoX.Value);
         FrmGrafico.NumLineas = Convert.ToInt32((double)numTotalLineas.Value);
         FrmGrafico.Tini = Convert.ToDouble((double)numMinimoT.Value);
         FrmGrafico.Tfin = Convert.ToDouble((double)numMaximoT.Value);
         FrmGrafico.Tavance = Convert.ToDouble((double)numAvanceT.Value);
         FrmGrafico.Ecuacion = txtEcuacion.Text;
         FrmGrafico.ZPersona = 5;

         FrmGrafico.Show();
      }

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

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

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

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

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

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

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

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

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

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

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

      //Distancia del observador
      public double ZPersona;

      //Rango de valores
      public double Xini, Yini, Xfin, Yfin, Tini, Tfin, Tavance, valorT;

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

      //Ecuación
      public string Ecuacion;

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

         valorT = Tini + 0.01;
      }

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Figura3D.CalcularFigura3D(Ecuacion, Xini, Xfin, NumLineas, valorT, AnguloX, AnguloY, AnguloZ,
ZPersona, 0, 0, this.ClientSize.Width, this.ClientSize.Height);
         Figura3D.Dibuja(e.Graphics);
      }

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

      private void Form1_Resize(object sender, EventArgs e) {
         Refresh();
      }

      private void timer1_Tick(object sender, EventArgs e) {
         valorT += Tavance;
         if (valorT >= Tfin) {
            valorT = Tfin;
            Tavance = -Tavance;
         }

         if (valorT <= Tini) {
            valorT = Tini;
            Tavance = -Tavance;
         }

         Refresh();
      }
   }
}
```

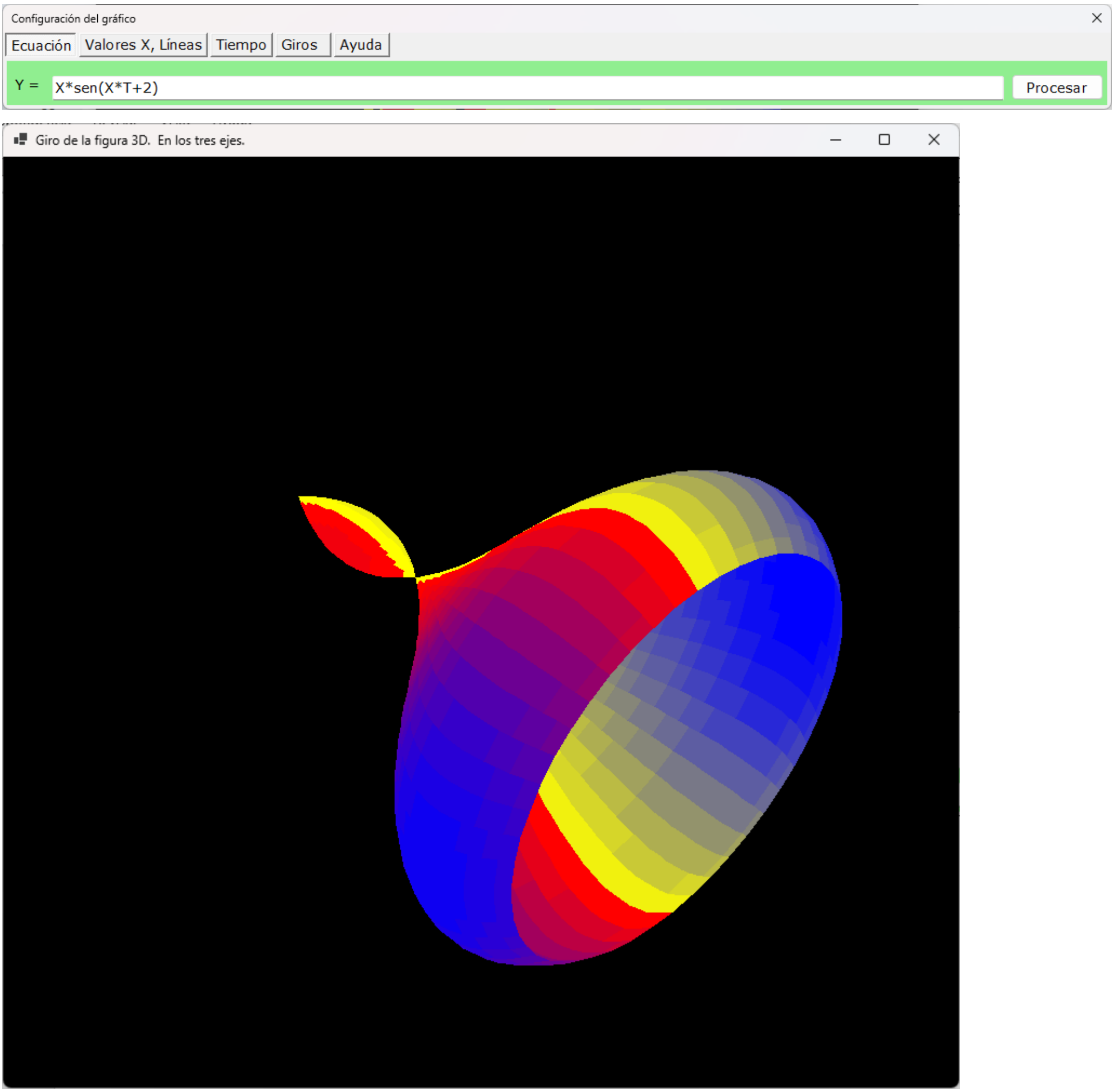

Ilustración 24: Sólido de revolución animado

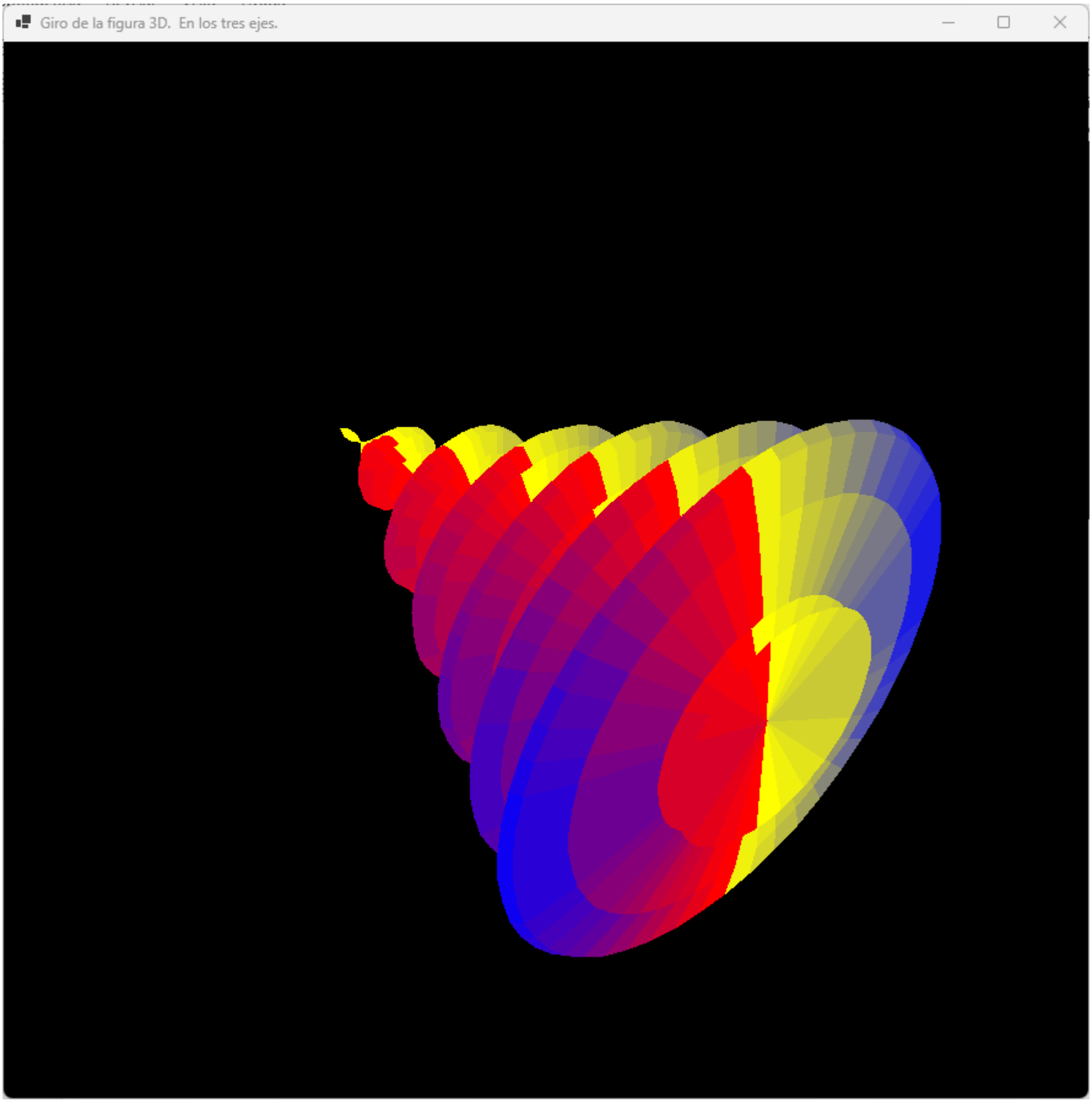

Ilustración 25: Sólido de revolución animado