C# Y .NET 9
Parte 12. Gráficos 2D

2025-06

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

Sally

Autor: Rafael Alberto Moreno Parra

Primitivas Gráficas

Para trabajar con gráficos, se debe crear un proyecto de escritorio gráfico

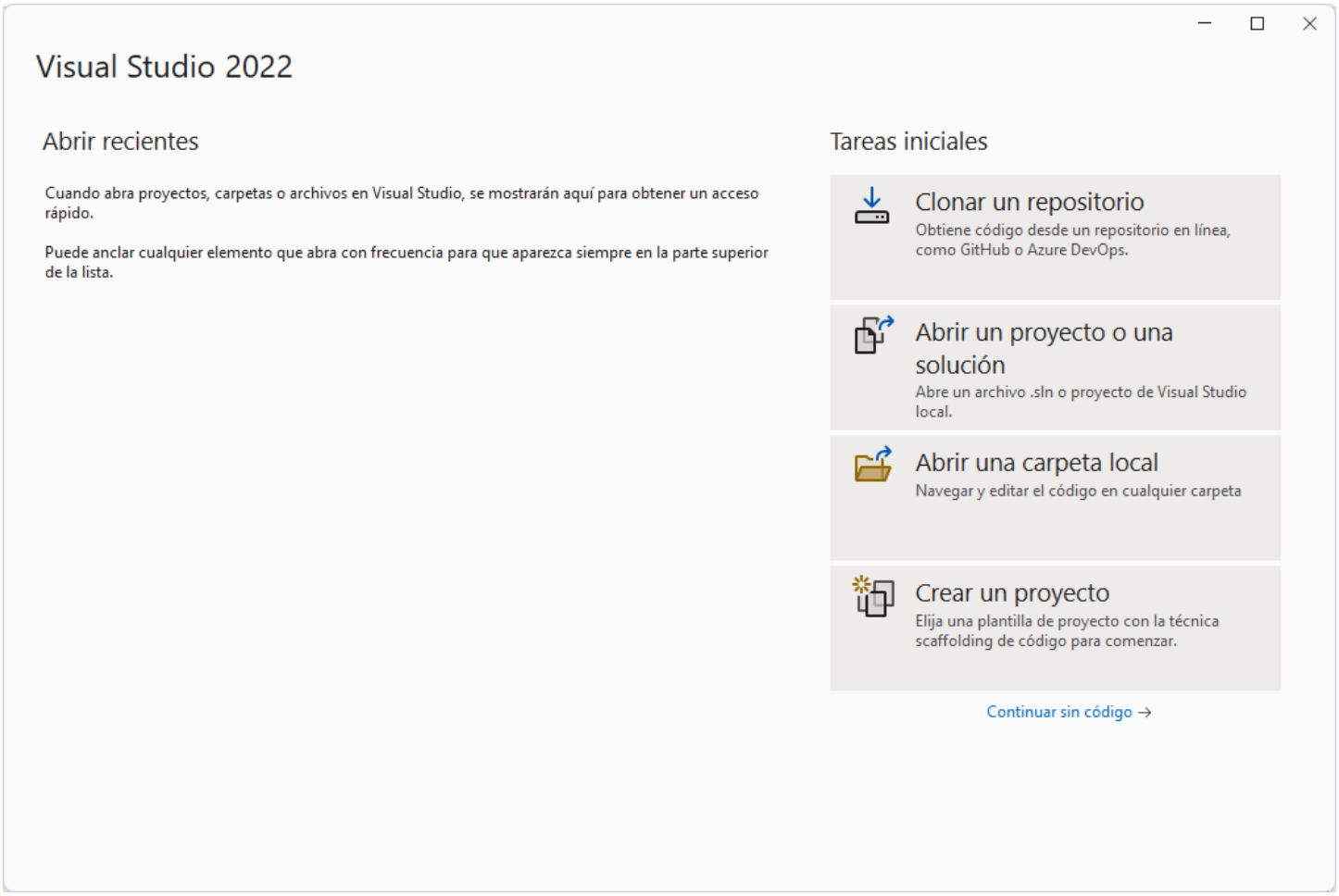

Ilustración 1: Inicia proyecto en Visual Studio 2022

Autor: Rafael Alberto Moreno Parra

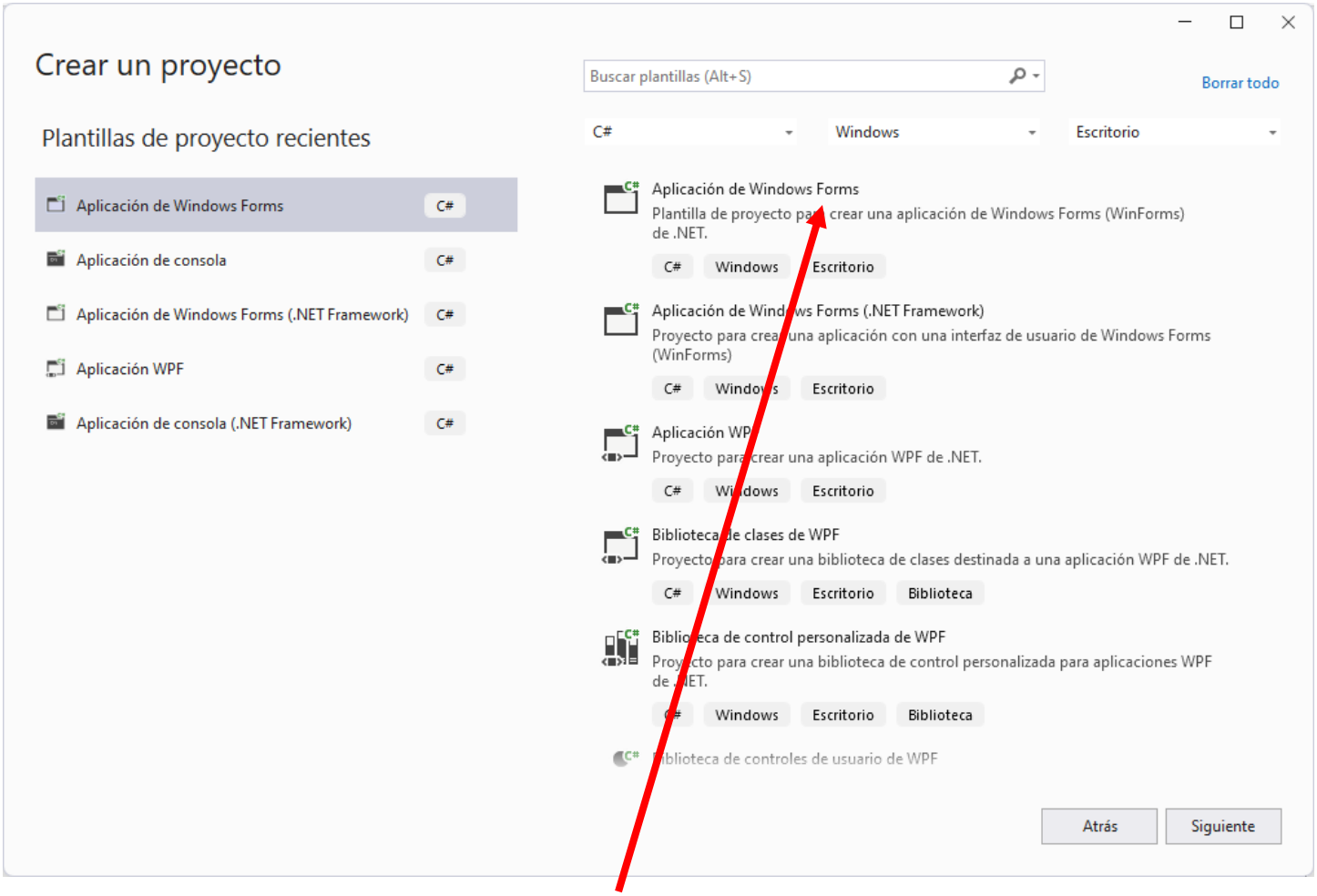

Ilustración 2: Selecciona "Aplicación de Windows Forms"

Una aplicación de tipo Windows Forms

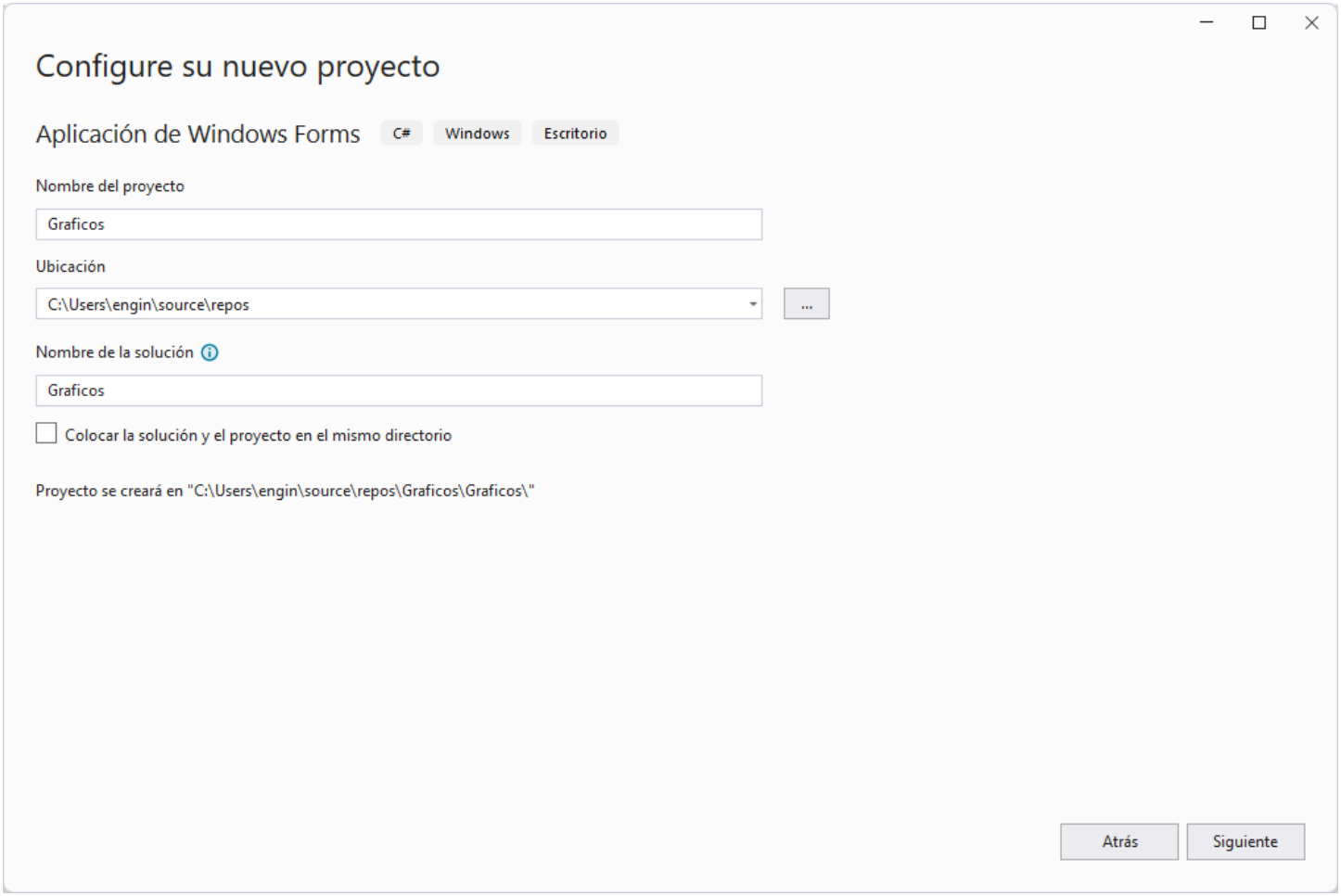

Ilustración 3: Pone nombre al proyecto, para este libro es "Graficos" (sin tilde)

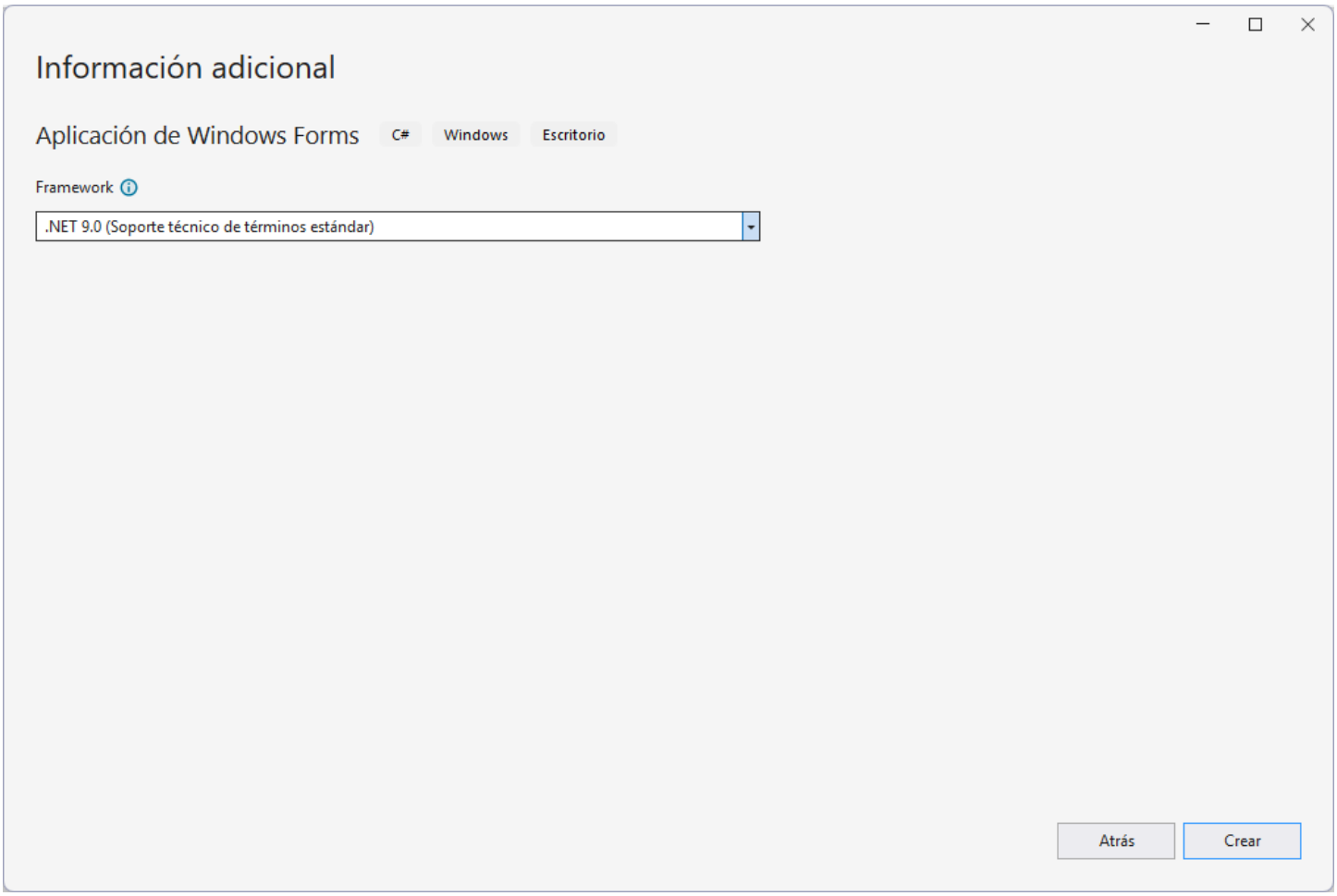

Ilustración 4: Selecciona .NET 9.0

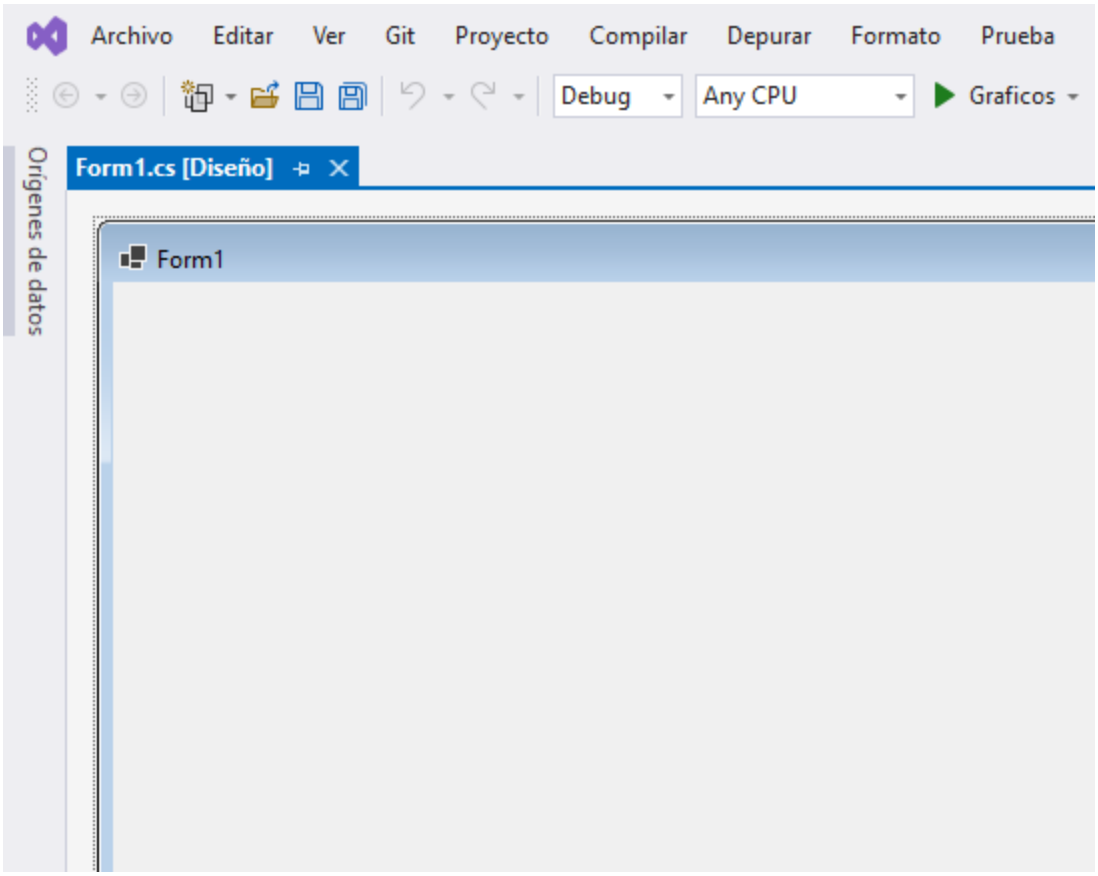

Ilustración 5: Una aplicación de escritorio típica

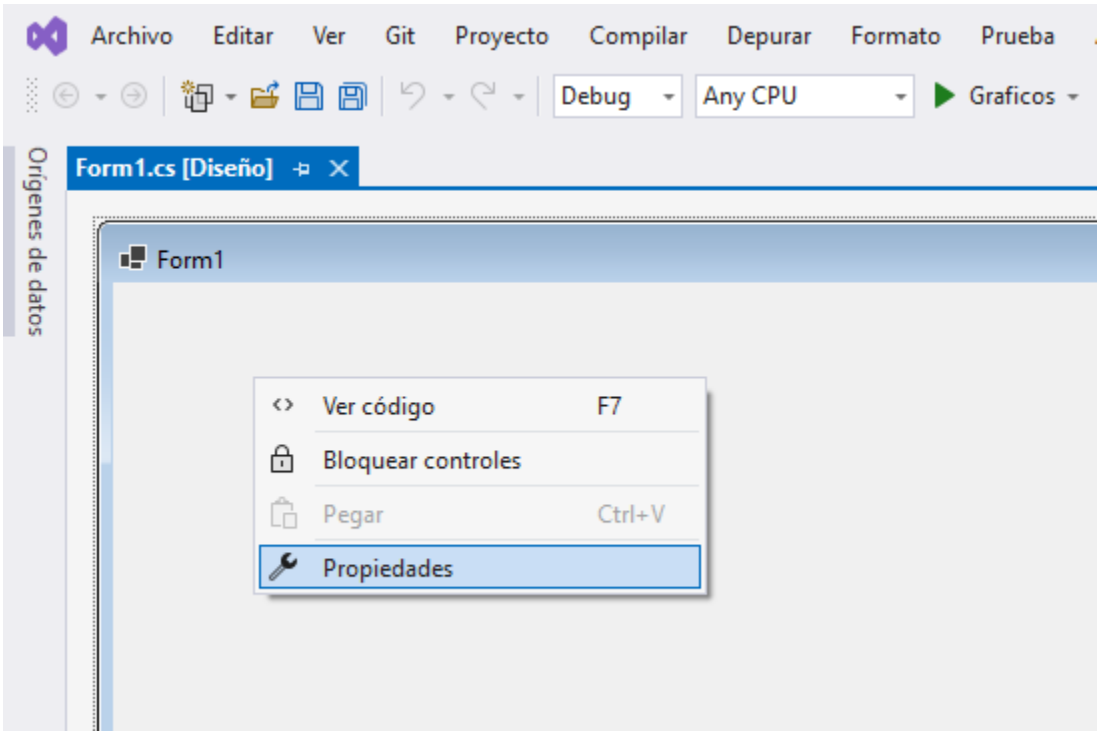

Ilustración 6: Clic botón derecho sobre la ventana y selecciona "Propiedades"

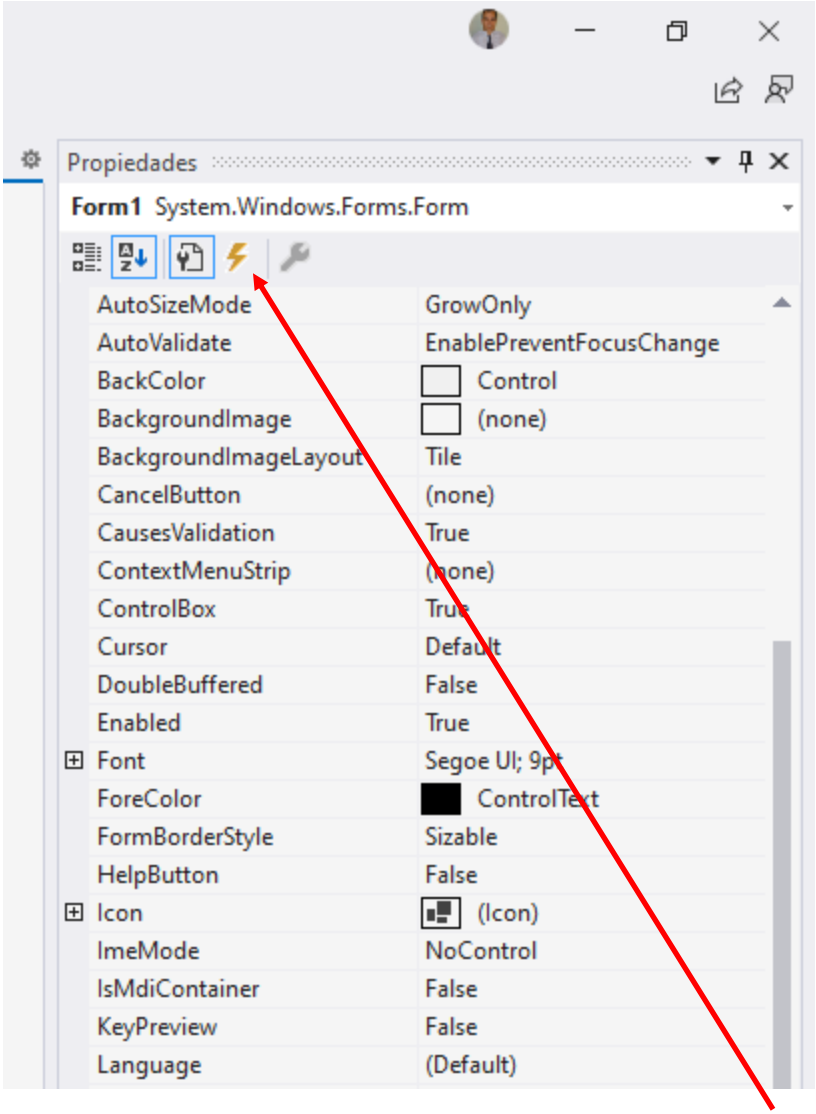

Ilustración 7: En la ventana de "Propiedades", selecciona "Eventos"

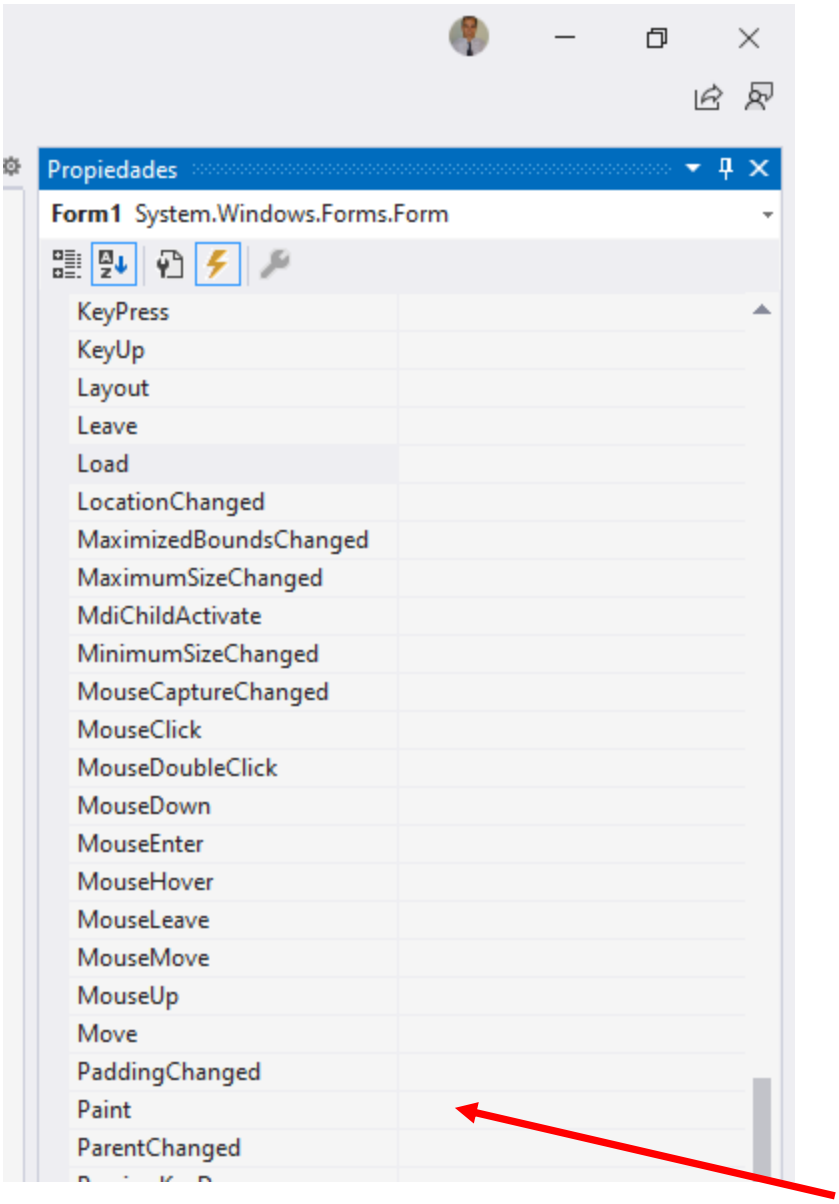

Ilustración 8: En "Eventos" se da doble clic al frente del evento "Paint"

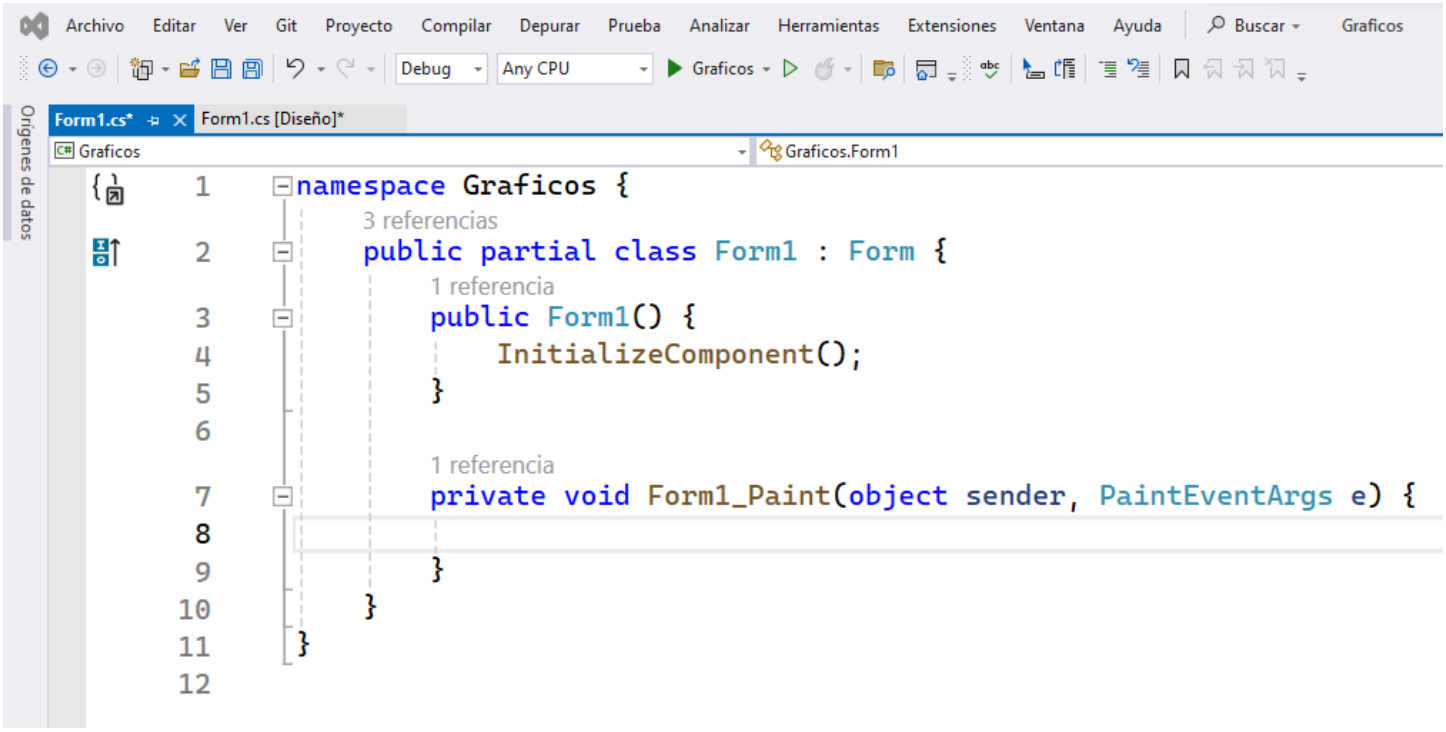

Ilustración 9: Esta es el código que se genera automáticamente

Línea

L/001.cs

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

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new(Color.Blue, 2);

            //============================
            //Línea: Xini, Yini, Xfin, Yfin
            //============================
            lienzo.DrawLine(lapiz, 10, 10, 130, 140);
        }
    }
}
```

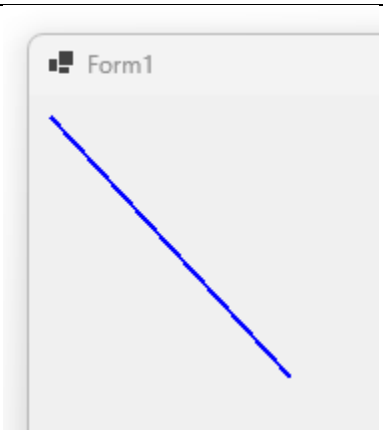

Ilustración 10: Línea

Cuando se usa el objeto "Graphics" proporcionado por el evento "Paint", se está dibujando directamente en la pantalla. Esto tiene algunas limitaciones:

- No se puede acceder ni modificar píxeles individuales fácilmente.

- No se puede conservar lo que se ha dibujado entre repintados (por ejemplo, al minimizar/restaurar la ventana).

- No se puede aplicar lógica de profundidad (Z-buffer) fácilmente, porque no se tiene control sobre el orden de los píxeles una vez dibujados.

Arco

L/002.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //============================================================
         //Arco: Xpos, Ypos, Ancho, Alto, Ángulo Inicial, Ángulo Final
         //============================================================
         lienzo.DrawArc(lapiz, 10, 90, 160, 170, 0, 270);
      }
   }
}
```

Ilustración 11: Arco

Rectángulo

L/003.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //====================================
         //Rectángulo: Xpos, Ypos, Ancho, Alto
         //====================================
         lienzo.DrawRectangle(lapiz, 100, 100, 200, 150);
      }
   }
}
```

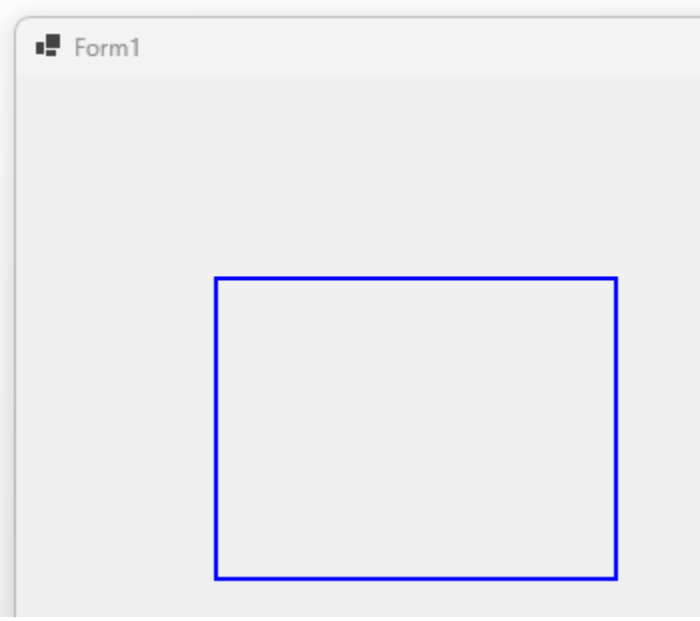

Ilustración 12: Rectángulo

Rectángulo relleno

L/004.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Con qué color se rellenan las figuras
         SolidBrush Relleno = new SolidBrush(Color.Red);

         //====================================
         //Rectángulo: Xpos, Ypos, Ancho, Alto
         //====================================
         lienzo.FillRectangle(Relleno, 100, 100, 200, 150);
      }
   }
}
```

Ilustración 13: Rectángulo relleno

L/005.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //============================================
         //Dibujar una Curva de Bézier
         //============================================
         Point P0 = new Point(100, 180);
         Point P1 = new Point(200, 10);
         Point P2 = new Point(350, 50);
         Point P3 = new Point(500, 180);
         lienzo.DrawBezier(lapiz, P0, P1, P2, P3);
      }
   }
}
```

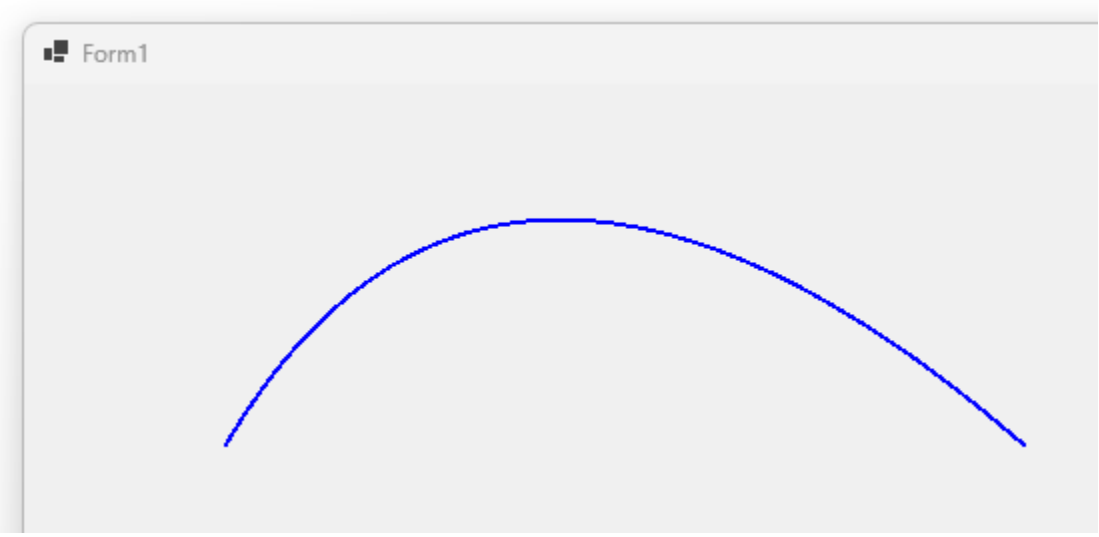

Ilustración 14: Curva Bézier

Varias líneas rectas

L/006.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //Conjunto de puntos
         PointF p1 = new PointF(150.0F, 150.0F);
         PointF p2 = new PointF(200.0F, 50.0F);
         PointF p3 = new PointF(300.0F, 140.0F);
         PointF p4 = new PointF(400.0F, 200.0F);
         PointF p5 = new PointF(450.0F, 300.0F);
         PointF p6 = new PointF(350.0F, 350.0F);
         PointF p7 = new PointF(300.0F, 150.0F);
         PointF[] Puntos = { p1, p2, p3, p4, p5, p6, p7 };

         //Dibuja líneas rectas para unir los puntos
         lienzo.DrawLines(lapiz, Puntos);
      }
   }
}
```

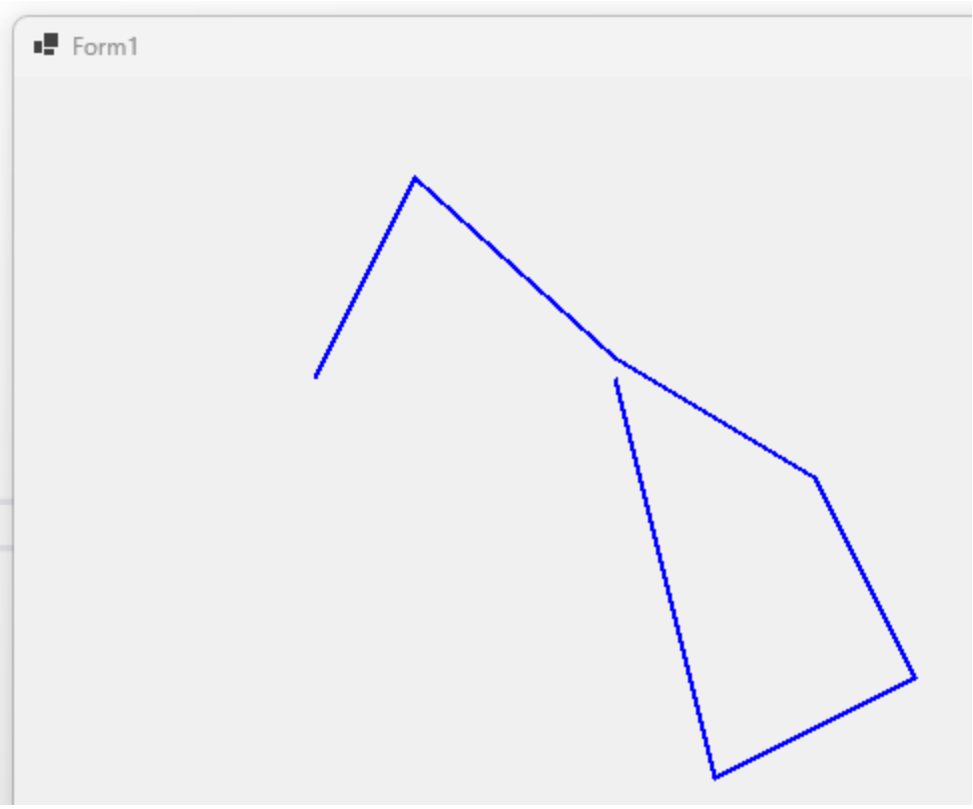

Ilustración 15: Varías líneas rectas

Polígono

L/007.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //Conjunto de puntos
         PointF p1 = new PointF(150.0F, 150.0F);
         PointF p2 = new PointF(200.0F, 50.0F);
         PointF p3 = new PointF(300.0F, 140.0F);
         PointF p4 = new PointF(400.0F, 200.0F);
         PointF p5 = new PointF(450.0F, 300.0F);
         PointF p6 = new PointF(350.0F, 350.0F);
         PointF p7 = new PointF(300.0F, 150.0F);
         PointF[] Puntos = { p1, p2, p3, p4, p5, p6, p7 };

         //Dibuja el polígono
         lienzo.DrawPolygon(lapiz, Puntos);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

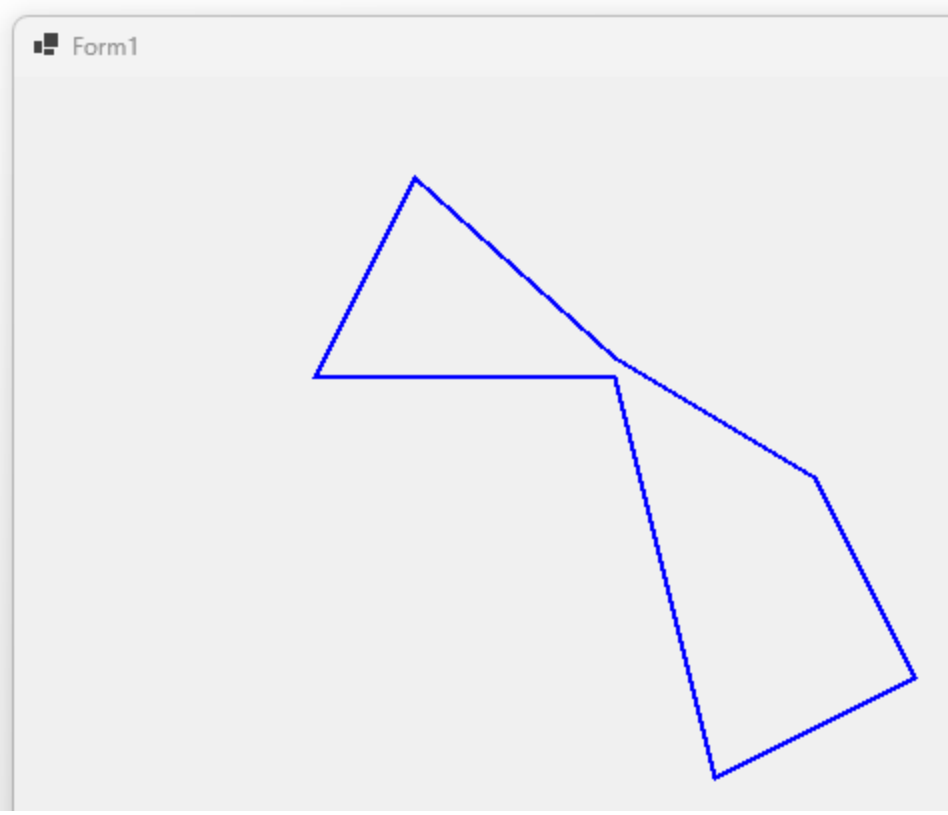

Ilustración 16: Polígono

Polígono relleno

L/008.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Con qué color se rellenan las figuras
         SolidBrush Relleno = new SolidBrush(Color.Red);

         //Conjunto de puntos
         PointF p1 = new PointF(150.0F, 150.0F);
         PointF p2 = new PointF(200.0F, 50.0F);
         PointF p3 = new PointF(300.0F, 140.0F);
         PointF p4 = new PointF(400.0F, 200.0F);
         PointF p5 = new PointF(450.0F, 300.0F);
         PointF p6 = new PointF(350.0F, 350.0F);
         PointF p7 = new PointF(300.0F, 150.0F);
         PointF[] Puntos = { p1, p2, p3, p4, p5, p6, p7 };

         //Dibuja el polígono
         lienzo.FillPolygon(Relleno, Puntos);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

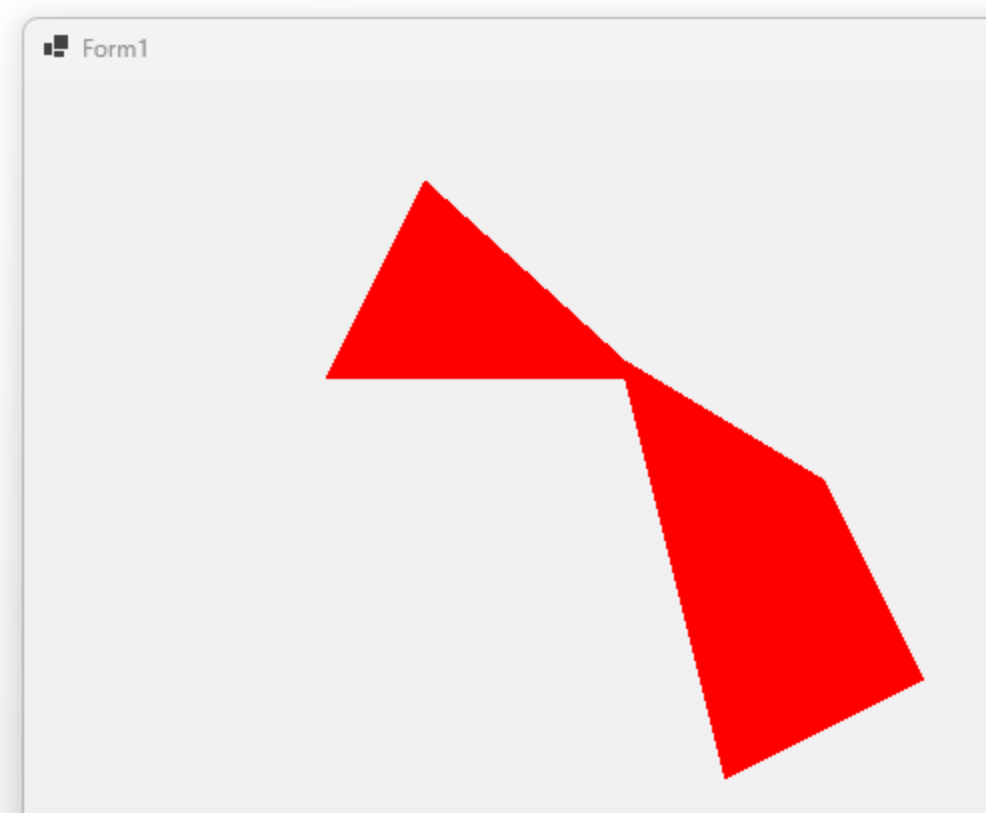

Ilustración 17: Polígono relleno

Curva cerrada

L/009.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //Conjunto de Puntos
         PointF p1 = new PointF(150.0F, 150.0F);
         PointF p2 = new PointF(200.0F, 50.0F);
         PointF p3 = new PointF(300.0F, 140.0F);
         PointF p4 = new PointF(400.0F, 200.0F);
         PointF p5 = new PointF(450.0F, 300.0F);
         PointF p6 = new PointF(350.0F, 350.0F);
         PointF p7 = new PointF(300.0F, 150.0F);
         PointF[] Puntos = { p1, p2, p3, p4, p5, p6, p7 };

         //Dibuja una curva cerrada que une esos puntos
         lienzo.DrawClosedCurve(lapiz, Puntos);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

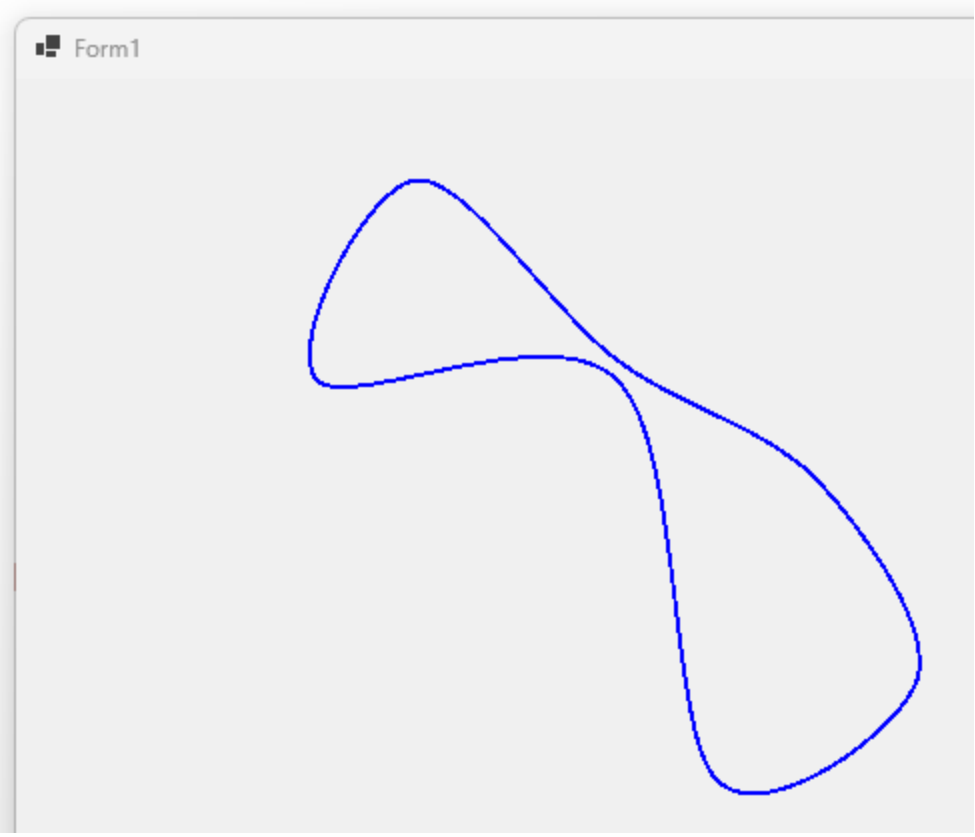

Ilustración 18: Curva cerrada

Curva cerrada rellena

L/010.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Relleno
         SolidBrush Relleno = new SolidBrush(Color.Gray);

         //Conjunto de Puntos
         PointF p1 = new PointF(150.0F, 150.0F);
         PointF p2 = new PointF(200.0F, 50.0F);
         PointF p3 = new PointF(300.0F, 140.0F);
         PointF p4 = new PointF(400.0F, 200.0F);
         PointF p5 = new PointF(450.0F, 300.0F);
         PointF p6 = new PointF(350.0F, 350.0F);
         PointF p7 = new PointF(300.0F, 150.0F);
         PointF[] Puntos = { p1, p2, p3, p4, p5, p6, p7 };

         //Dibuja una curva cerrada rellena que une esos puntos
         lienzo.FillClosedCurve(Relleno, Puntos);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

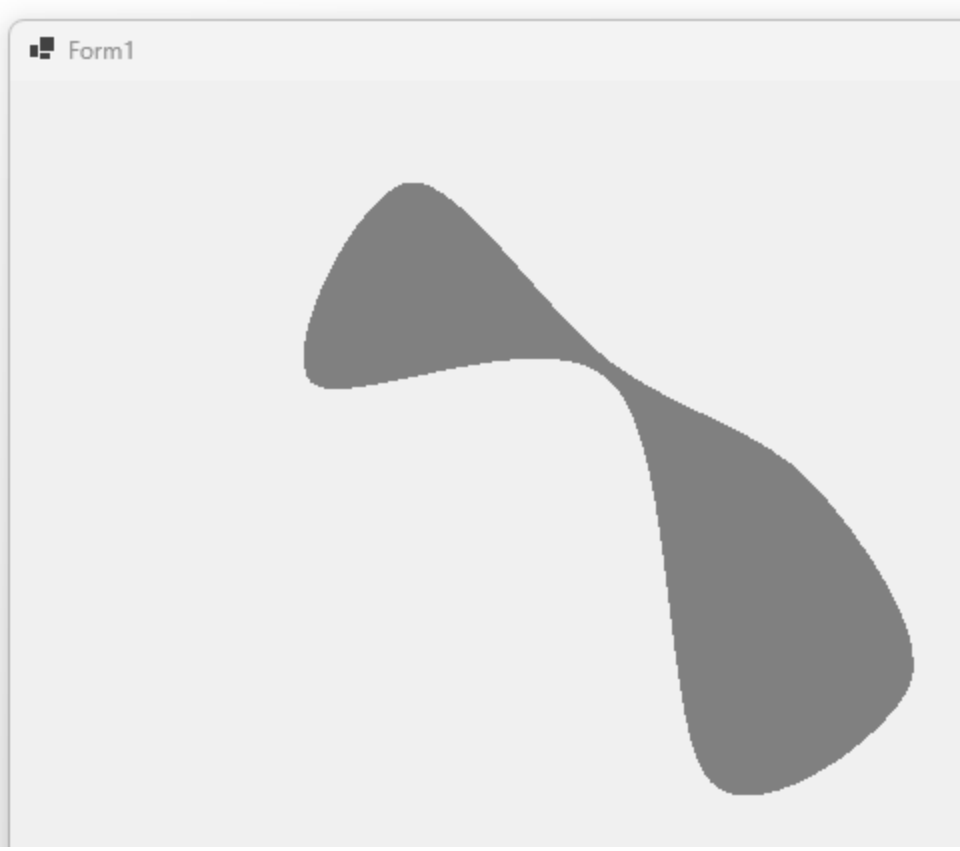

Ilustración 19: Curva cerrada rellena

Curva que une varios puntos

L/011.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //Conjunto de Puntos
         PointF p1 = new PointF(150.0F, 150.0F);
         PointF p2 = new PointF(200.0F, 50.0F);
         PointF p3 = new PointF(300.0F, 140.0F);
         PointF p4 = new PointF(400.0F, 200.0F);
         PointF p5 = new PointF(450.0F, 300.0F);
         PointF p6 = new PointF(350.0F, 350.0F);
         PointF p7 = new PointF(300.0F, 150.0F);
         PointF[] Puntos = { p1, p2, p3, p4, p5, p6, p7 };

         //Dibuja la curva que une esos puntos
         lienzo.DrawCurve(lapiz, Puntos);
      }
   }
}
```

Ilustración 20: Curva que une varios puntos

Elipse

L/012.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //================================
         //Elipse: Xpos, Ypos, ancho, alto
         //================================
         lienzo.DrawEllipse(lapiz, 200, 30, 250, 90);
      }
   }
}
```

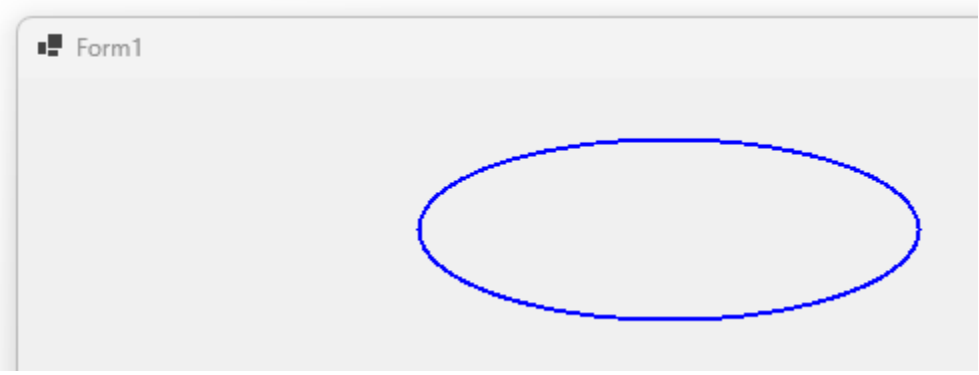

Ilustración 21: Elipse

Elipse rellena

L/013.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Relleno
         SolidBrush Relleno = new SolidBrush(Color.Chocolate);

         //================================
         //Elipse: Xpos, Ypos, ancho, alto
         //================================
         lienzo.FillEllipse(Relleno, 200, 30, 250, 90);
      }
   }
}
```

Ilustración 22: Elipse rellena

Letras

L/014.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //==============
         //Letras
         //==============
         string Cadena = "Esta es una prueba";

         //Fuente y la brocha con que se pinta.
         Font Fuente = new Font("Tahoma", 16);
         SolidBrush Brocha = new SolidBrush(Color.Black);

         //Punto arriba a la izquierda para pintar la cadena
         PointF PuntoCadena = new PointF(100.0F, 140.0F);

         //Formato para dibujar
         StringFormat Formato = new StringFormat();
         FormatoDibuja.FormatFlags = StringFormatFlags.DirectionVertical;

         //Dibuja la cadena
         lienzo.DrawString(Cadena, Fuente, Brocha, PuntoCadena, Formato);
      }
   }
}
```

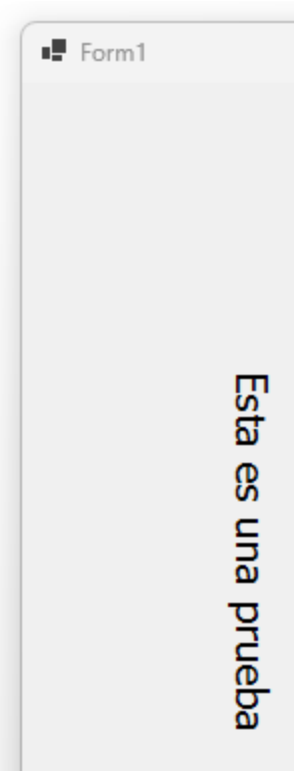

Ilustración 23: Letras

Diagrama de pastel

L/015.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Lápiz con que dibuja. Color, grosor
         Pen lapiz = new Pen(Color.Blue, 2);

         //================
         //Dibuja un pastel
         //================

         //Crea un rectángulo para la elipse que dibujará el pastel
         Rectangle rectangulo = new Rectangle(10, 10, 400, 400);

         //Pastel: rectángulo que contiene:
         //ángulo inicial, ángulo de apertura
         lienzo.DrawPie(lapiz, rectangulo, 0F, 45F);
         lienzo.DrawPie(lapiz, rectangulo, 90F, 45F);
         lienzo.DrawPie(lapiz, rectangulo, 150F, 45F);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

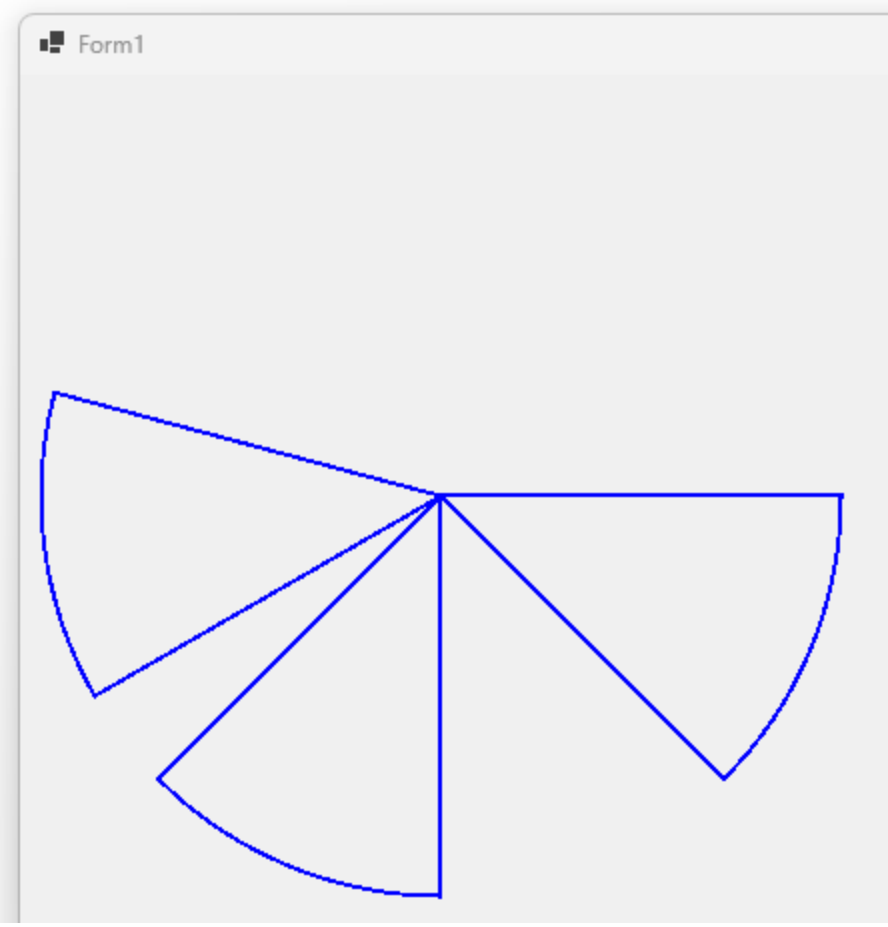

Ilustración 24: Diagrama de pastel

Diagrama de pastel relleno

L/016.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo donde va a hacer el gráfico
         Graphics lienzo = e.Graphics;

         //Rellenos
         SolidBrush Relleno1 = new SolidBrush(Color.Chocolate);
         SolidBrush Relleno2 = new SolidBrush(Color.Red);
         SolidBrush Relleno3 = new SolidBrush(Color.Blue);

         //================
         //Dibuja un pastel
         //================

         //Crea un rectángulo para la elipse que dibujará el pastel
         Rectangle rectangulo = new Rectangle(10, 10, 400, 400);

         //Pastel: rectángulo que contiene:
         //ángulo inicial, ángulo de apertura
         lienzo.FillPie(Relleno1, rectangulo, 0F, 45F);
         lienzo.FillPie(Relleno2, rectangulo, 90F, 45F);
         lienzo.FillPie(Relleno3, rectangulo, 150F, 45F);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

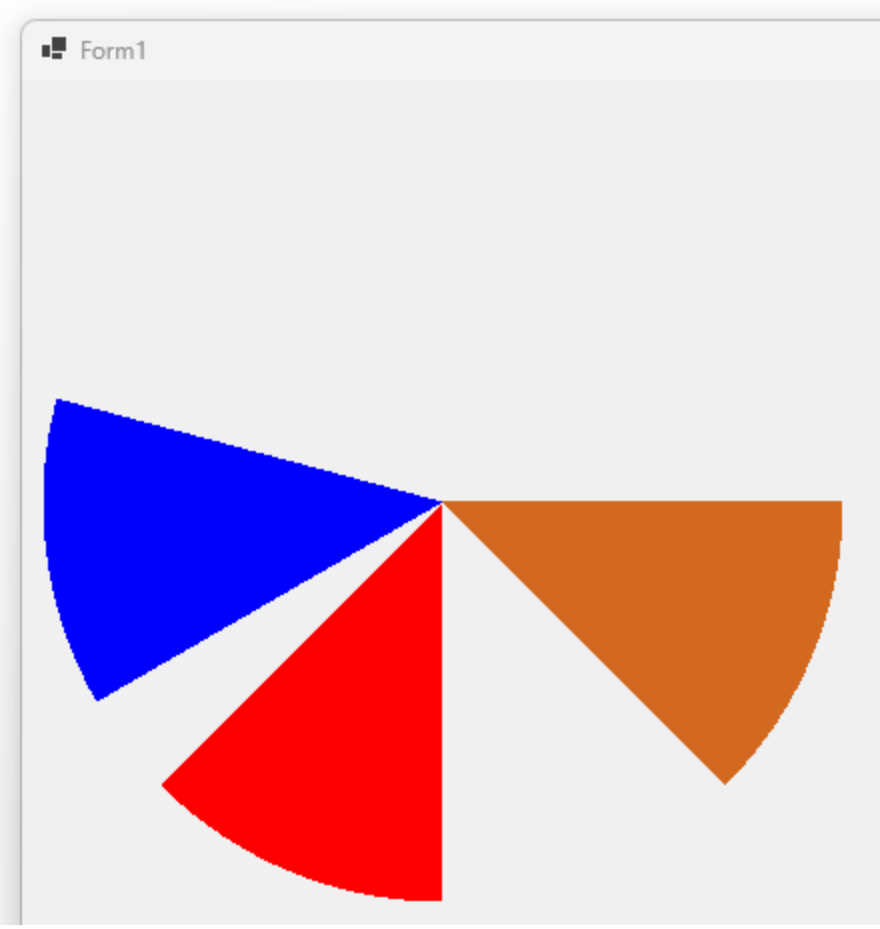

Ilustración 25: Diagrama de pastel relleno

Líneas horizontales

L/017.cs

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

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

         //Primer gráfico
         Pen lapiz = new Pen(Color.Blue, 2);
         int contador = 10;
         int incremento = 2;

         do {
            grafico.DrawLine(lapiz, 10, contador, 300, contador);
            incremento++;
            //Incrementa el espacio entre línea y línea
            contador += incremento;
         } while (contador <= 602);
         contador -= incremento;

         //Segundo gráfico
         Pen lapiz2 = new Pen(Color.Red, 2);
         incremento = 2;
         do {
            grafico.DrawLine(lapiz2, 300, contador, 590, contador);
            incremento++;
            contador -= incremento;
         } while (contador >= 10);
      }
   }
}
```

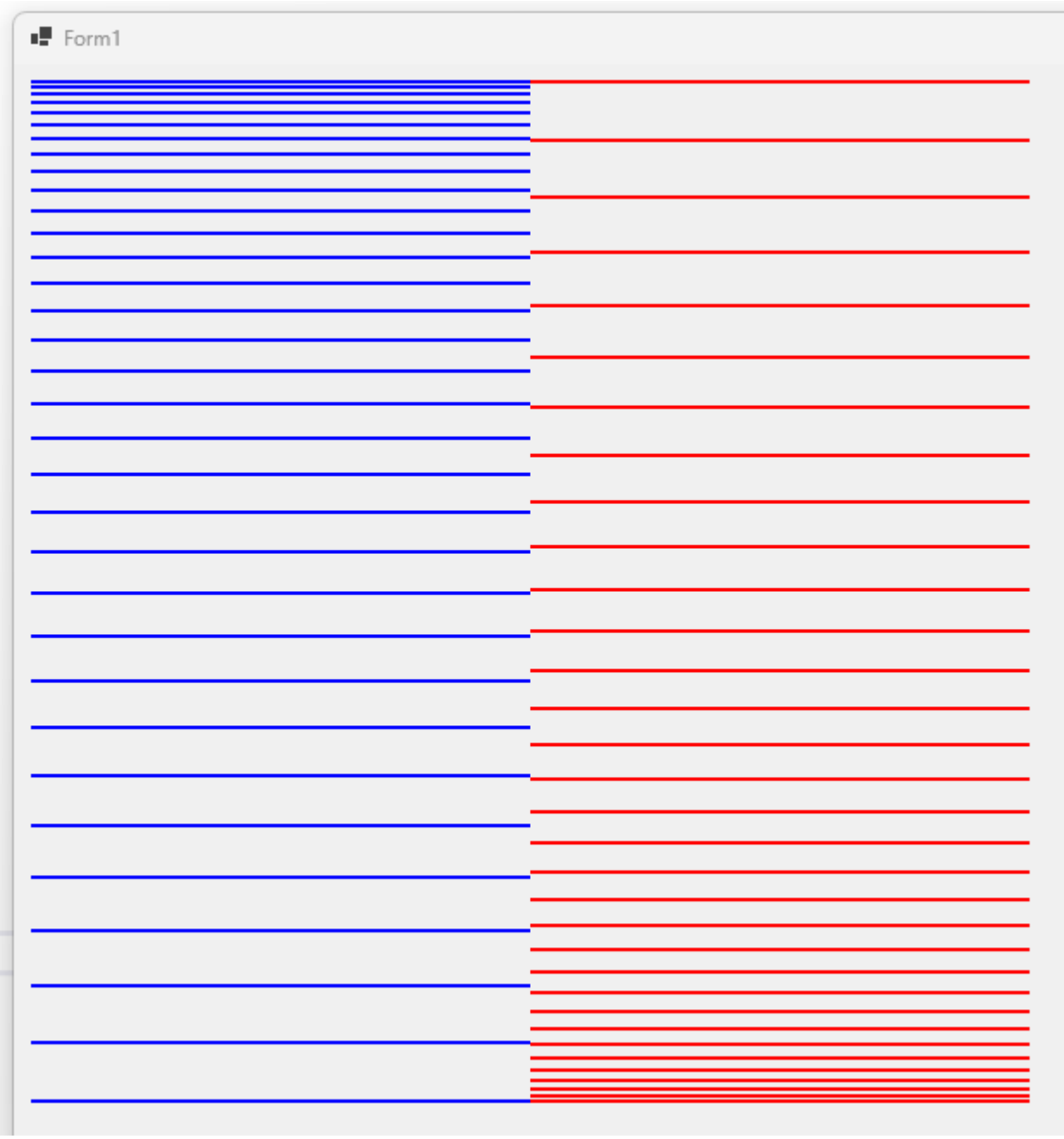

Ilustración 26: Líneas horizontales

Líneas verticales

L/018.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Lienzo
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         int contador = 10;
         int incremento = 2;

         //Primer gráfico
         do {
            grafico.DrawLine(lapiz, contador, 10, contador, 200);
            incremento++;
            //Incrementa el espacio entre línea y línea
            contador += incremento;
         }
         while (contador <= 602);
         contador -= incremento;

         //Segundo gráfico
         Pen lapiz2 = new Pen(Color.Red, 2);
         incremento = 2;
         do {
            grafico.DrawLine(lapiz2, contador, 200, contador, 380);
            incremento++;
            contador -= incremento;
         }
         while (contador >= 10);
      }
   }
}
```

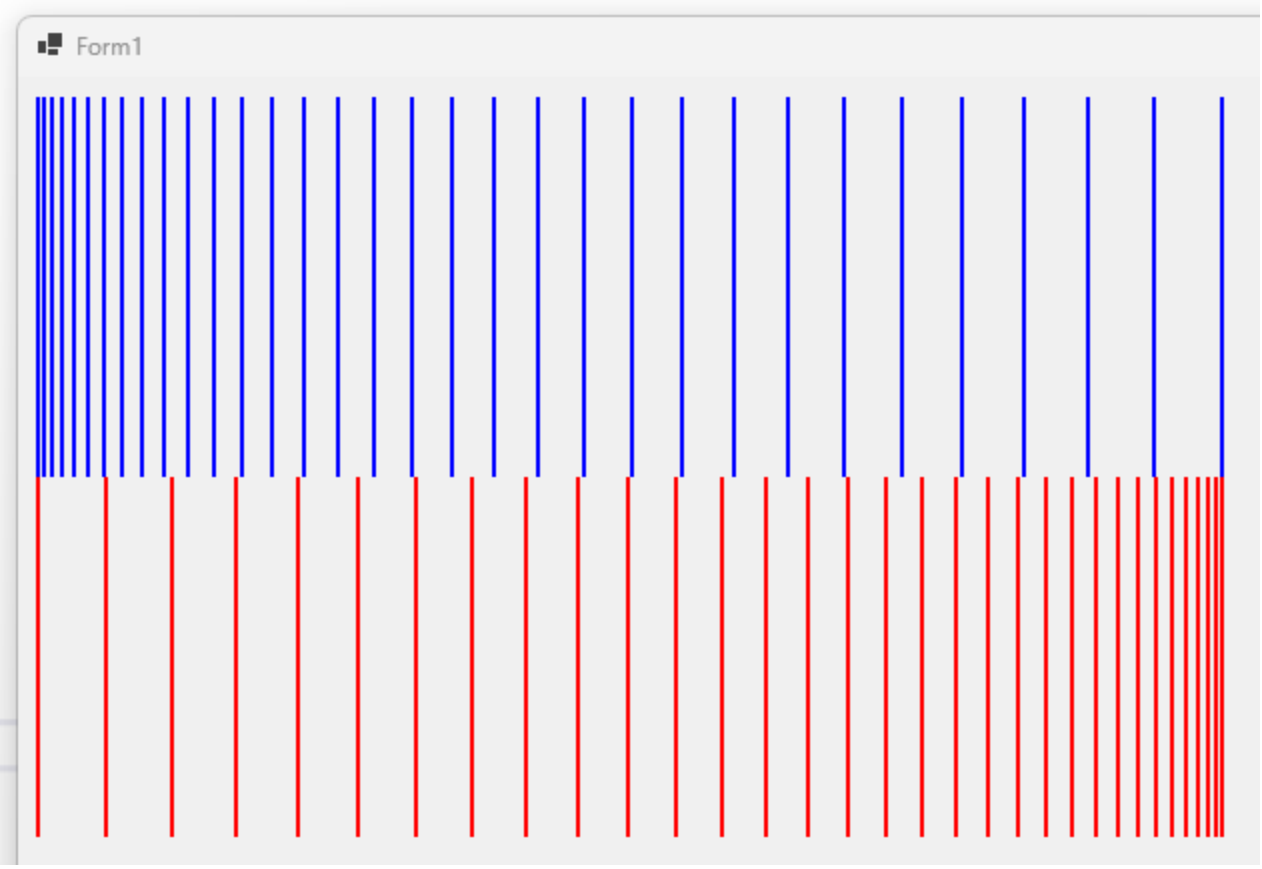

Ilustración 27: Líneas verticales

Líneas más juntas en el centro

L/019.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics lienzo = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int lim = 500;
         int cont = lim;
         int incremento = 2;

         do {
            int comun = lim - (cont - lim);
            lienzo.DrawLine(lapiz, cont, 10, cont, 200);
            lienzo.DrawLine(lapiz, comun, 10, comun, 200);
            lienzo.DrawLine(lapiz2, cont, 200, cont, 380);
            lienzo.DrawLine(lapiz2, comun, 200, comun, 380);
            incremento++;
            cont += incremento;
         }
         while (cont <= 2 * lim);
      }
   }
}
```

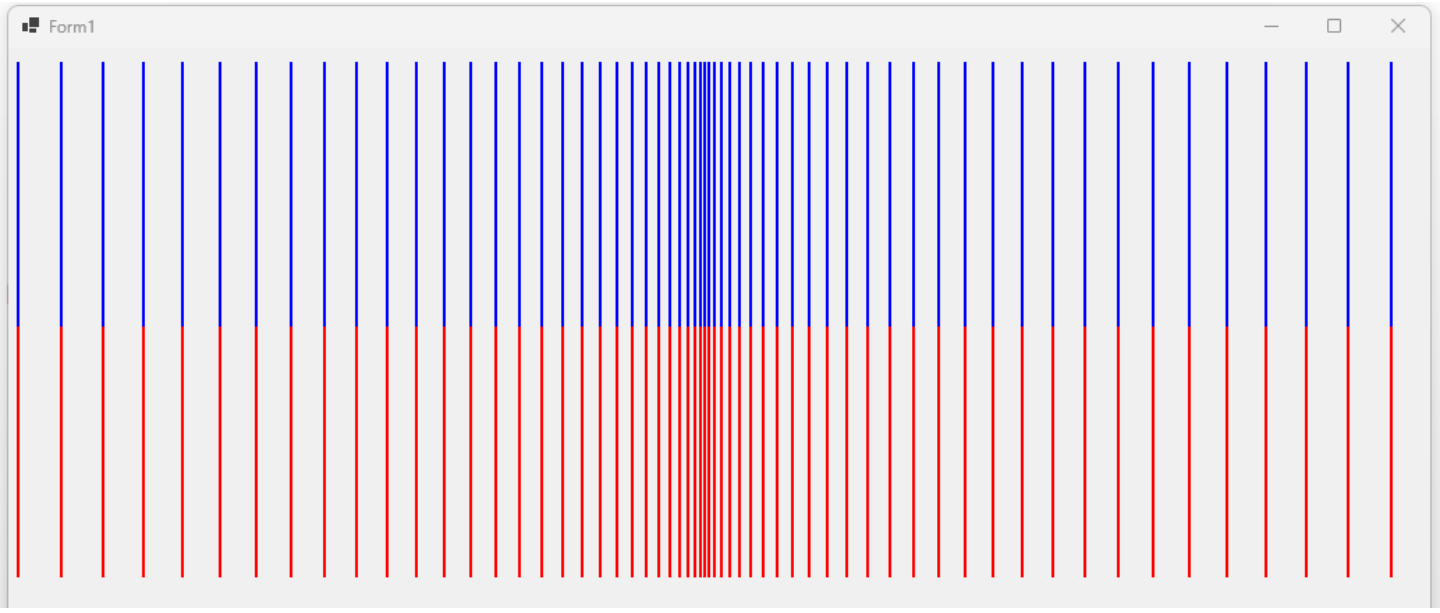

Ilustración 28: Líneas más juntas en el centro

Líneas generan rombo

L/020.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int xval = 10;
         int yval = 5;

         do {
            grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
            xval += 5;
            yval += 5;
         }
         while (yval < 300);

         do {
            grafico.DrawLine(lapiz2, xval, 300 - yval, xval, 300 + yval);
            xval += 5;
            yval -= 5;
         }
         while (yval > 0);
      }
   }
}
```

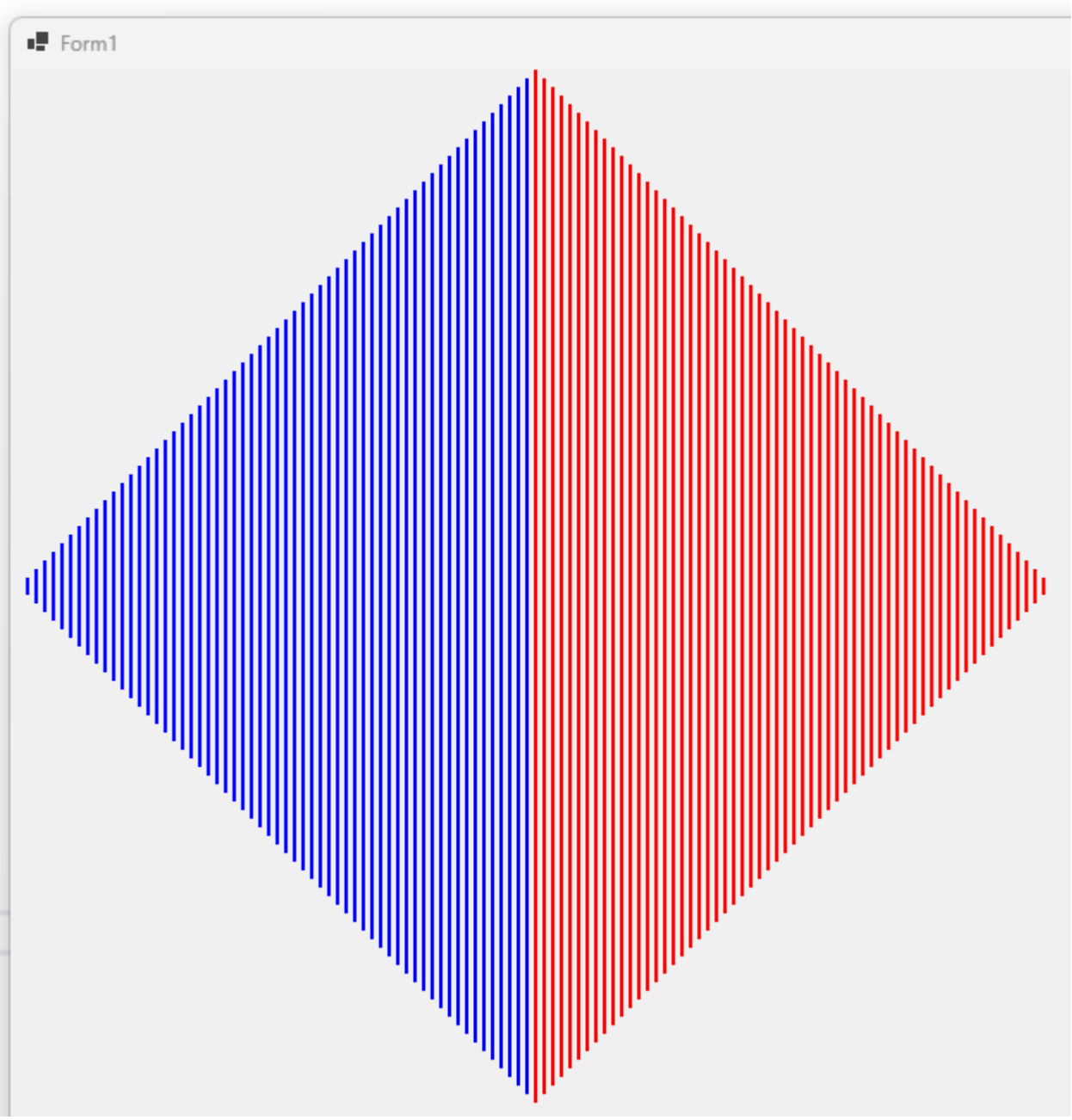

Ilustración 29: Líneas generan rombo

Líneas en esquina superior derecha

L/021.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int xval = 600;
         int yval = 300;

         do {
            grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
            xval -= 5;
            yval -= 5;
         }
         while (yval >= 0);

         xval = 300;
         yval = 0;
         do {
            grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
            xval -= 5;
            yval += 5;
         }
         while (yval <= 300);
      }
   }
}
```

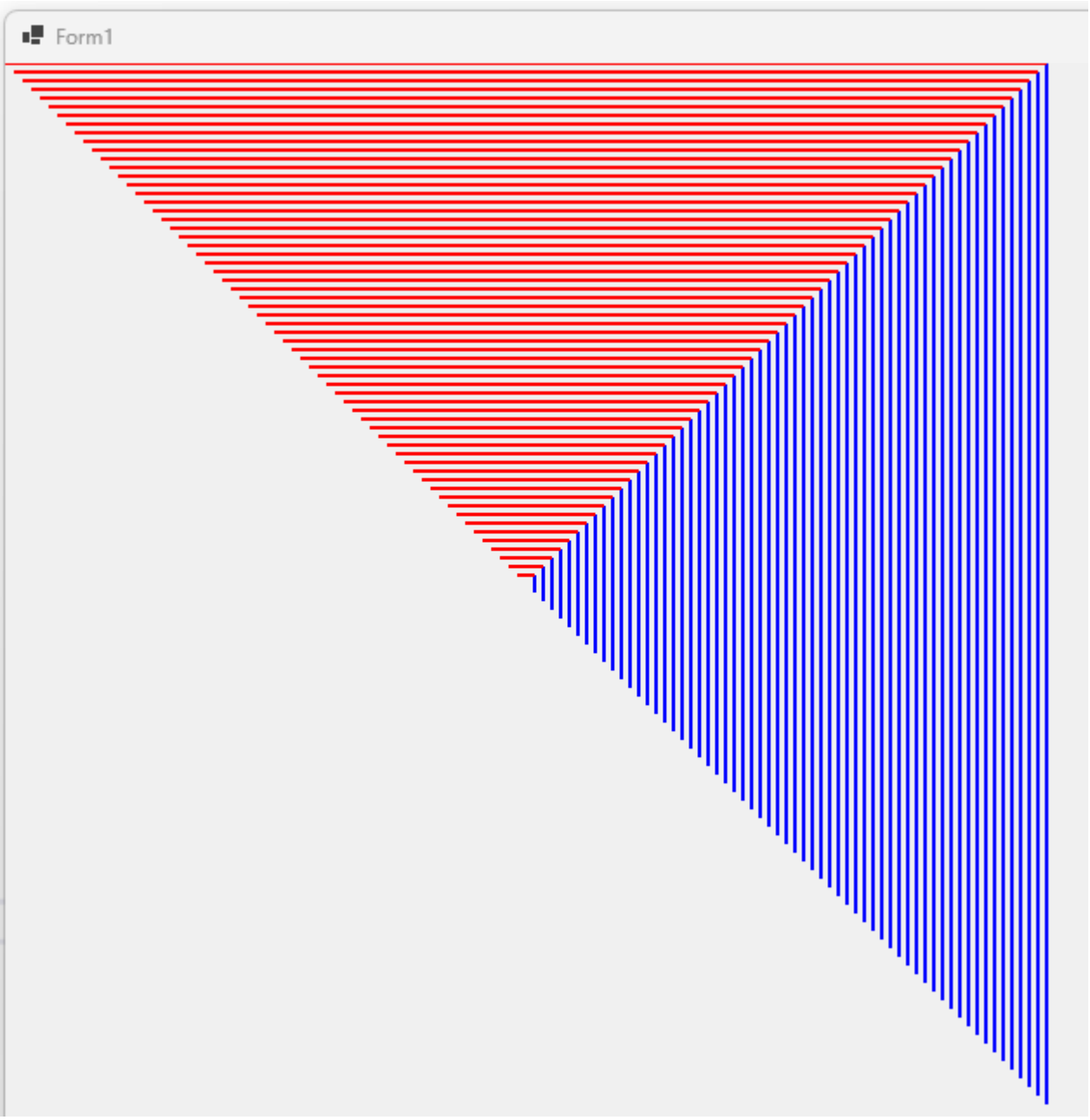

Ilustración 30: Líneas en esquina superior derecha

Líneas en esquina inferior derecha

L/022.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int xval = 600;
         int yval = 300;

         do {
            grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
            xval -= 5;
            yval -= 5;
         }
         while (yval >= 0);

         xval = 300;
         yval = 600;
         do {
            grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
            xval -= 5;
            yval -= 5;
         }
         while (yval >= 300);
      }
   }
}
```

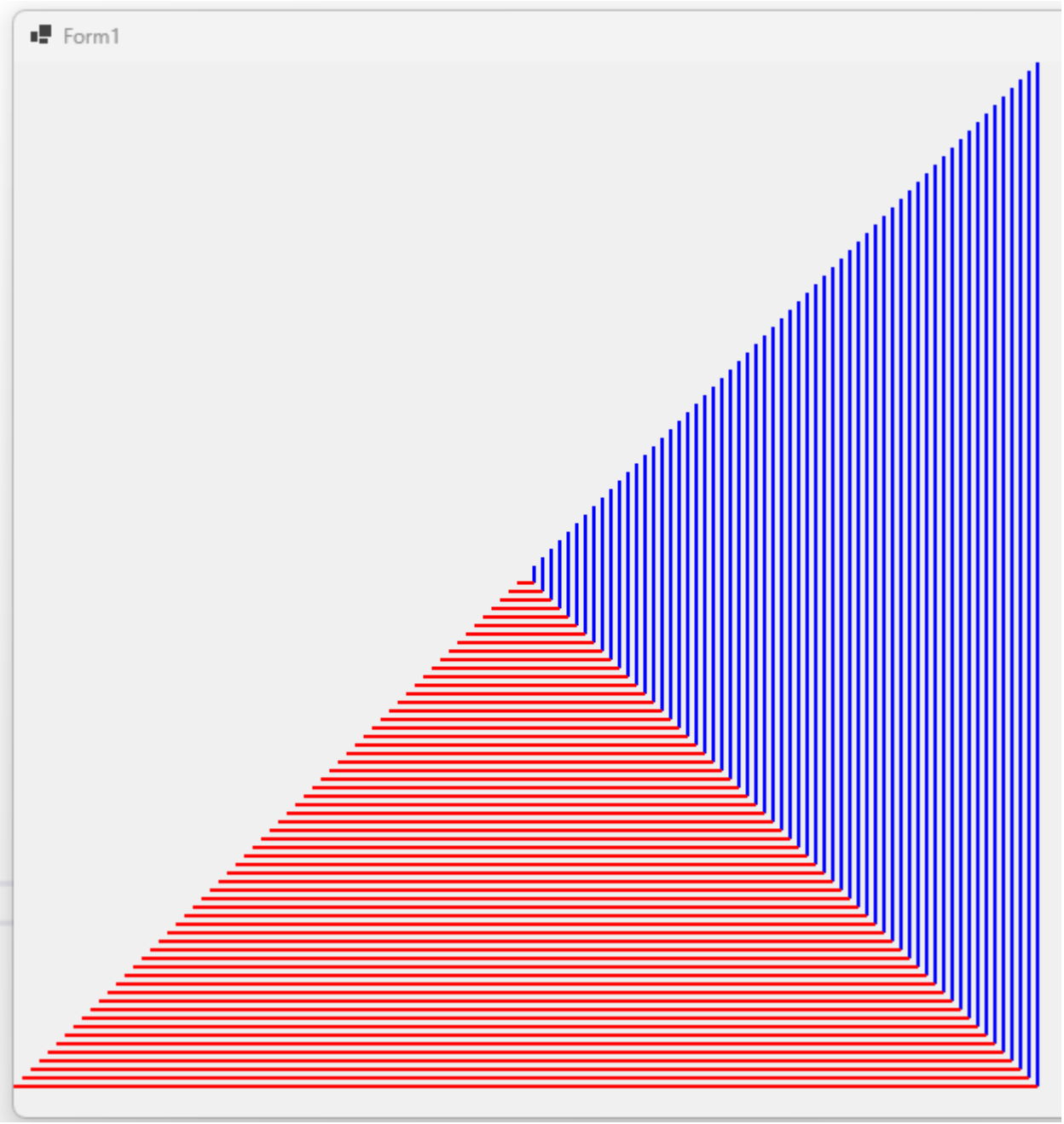

Ilustración 31: Líneas en esquina inferior derecha

Líneas en esquina inferior izquierda

L/023.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int xval = 0;
         int yval = 300;

         do {
            grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
            xval += 5;
            yval -= 5;
         }
         while (yval >= 0);

         xval = 300;
         yval = 600;
         do {
            grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
            xval -= 5;
            yval -= 5;
         }
         while (yval >= 300);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

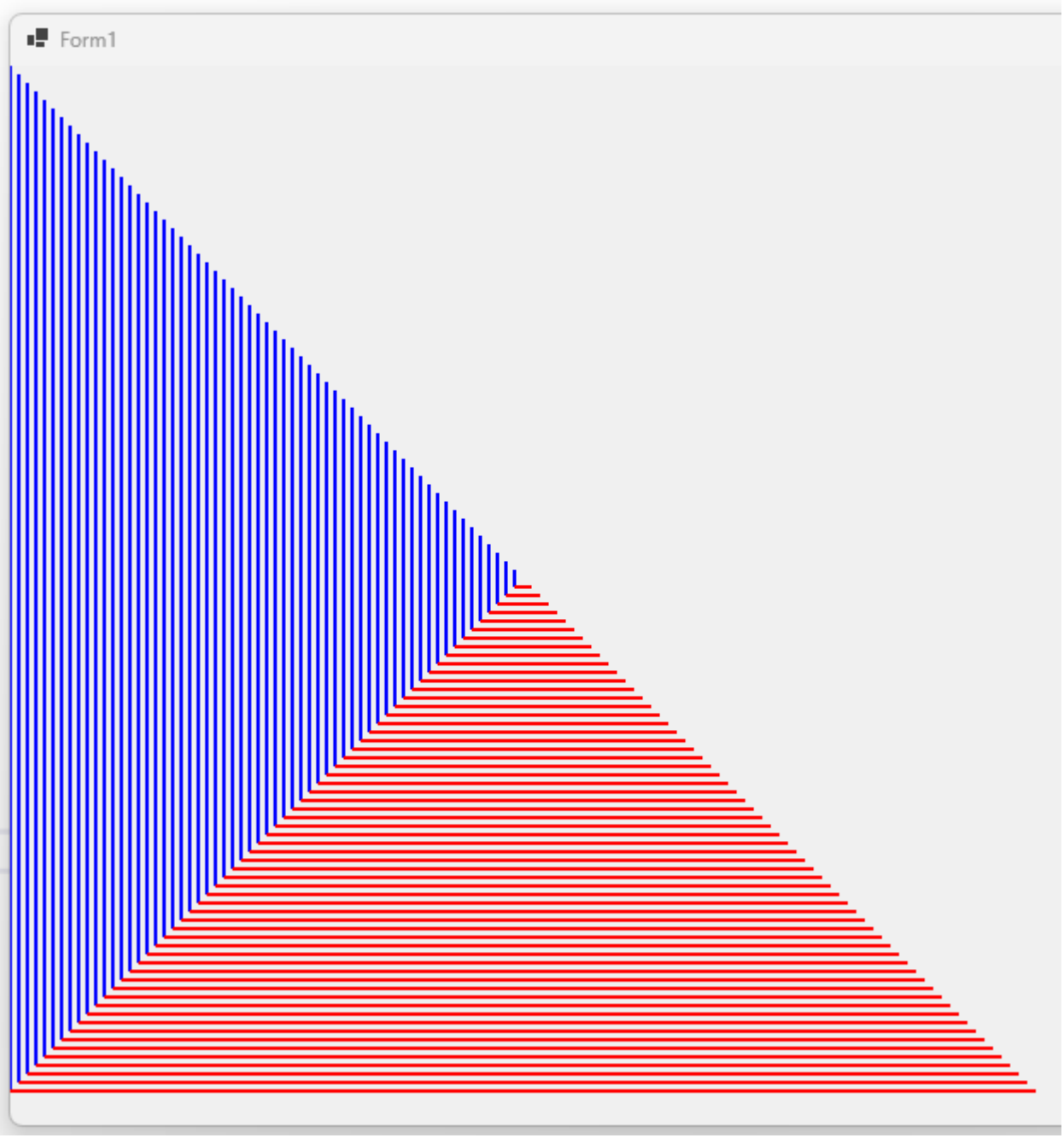

Ilustración 32: Líneas en esquina inferior izquierda

Triángulos apuntan a la derecha

L/024.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);

         int pX1, pY1, pX2, pY2, pX3, pY3, constante;
         pX1 = 420;
         pY1 = 360;
         pX2 = 480;
         pY2 = 400;
         pX3 = 420;
         pY3 = 440;
         constante = 20;
         do {
            grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
            grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
            grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
            pX1 -= constante;
            pY1 -= constante;
            pX2 += constante;
            pX3 -= constante;
            pY3 += constante;
         }
         while (pY1 >= 0);
      }
   }
}
```

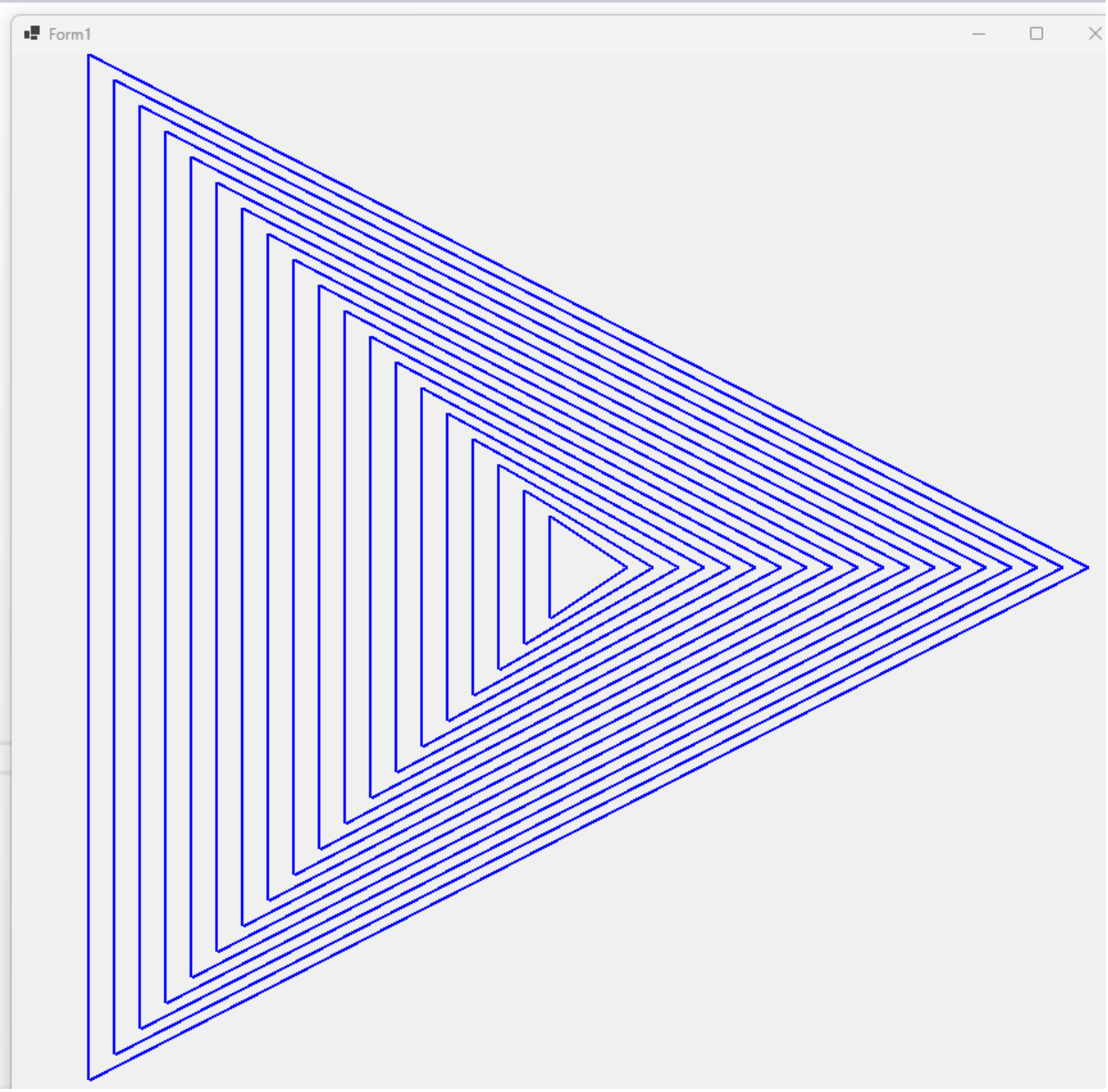

Ilustración 33: Triángulos apuntan a la derecha

Triángulos apuntan a la izquierda

L/025.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);

         int pX1, pY1, pX2, pY2, pX3, pY3, constante;
         pX1 = 420;
         pY1 = 360;
         pX2 = 360;
         pY2 = 400;
         pX3 = 420;
         pY3 = 440;
         constante = 20;
         do {
            grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
            grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
            grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
            pX1 += constante;
            pY1 -= constante;
            pX2 -= constante;
            pX3 += constante;
            pY3 += constante;
         }
         while (pY1 >= 0);
      }
   }
}
```

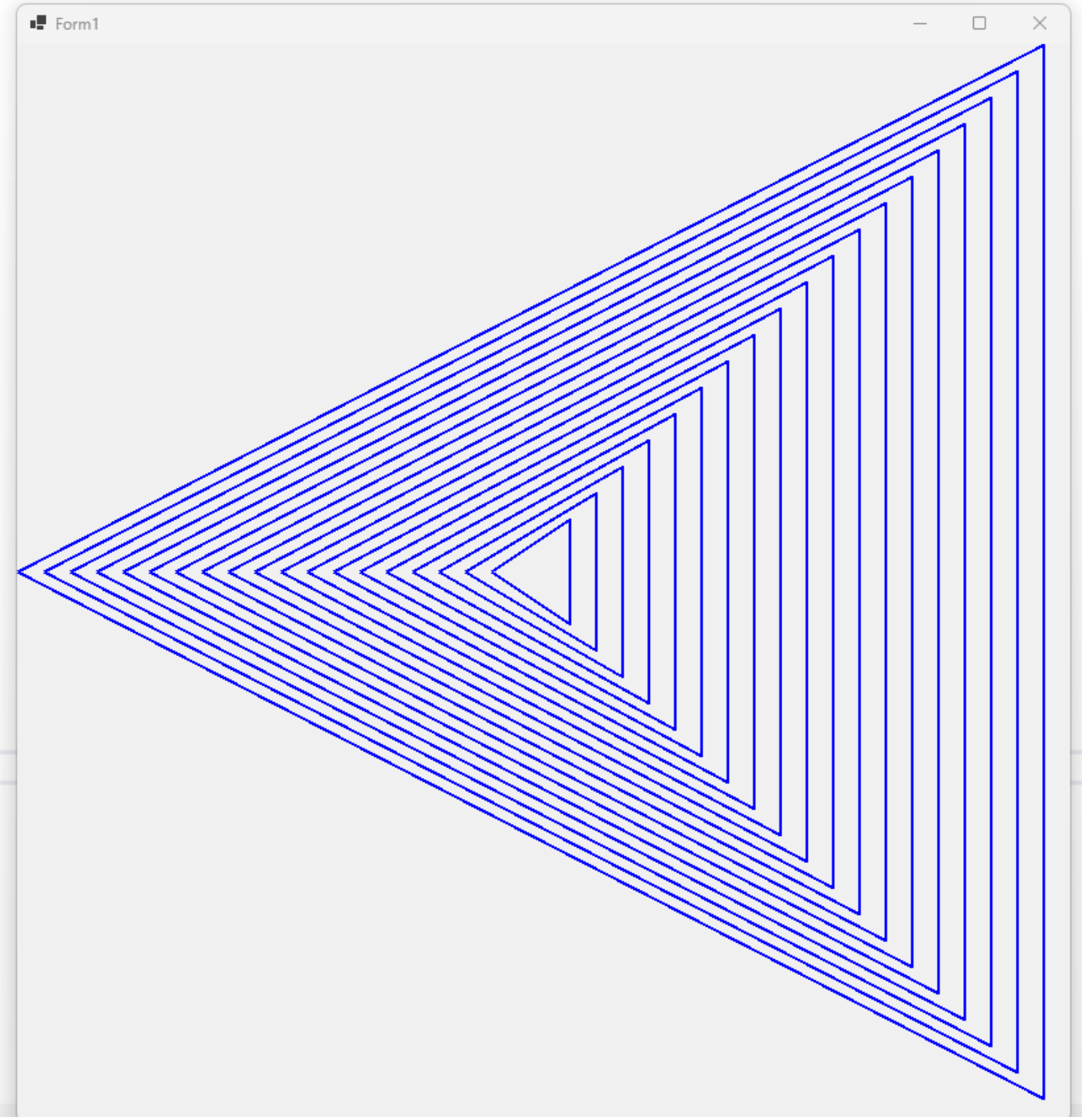

Ilustración 34: Triángulos apuntan a la izquierda

Triángulos apuntan a la izquierda alternando colores

L/026.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz;
         Pen lapizRojo = new Pen(Color.Red, 2);
         Pen lapizAzul = new Pen(Color.Blue, 2);

         int pX1, pY1, pX2, pY2, pX3, pY3, constante, cambia;
         pX1 = 220;
         pY1 = 360;
         pX2 = 160;
         pY2 = 400;
         pX3 = 220;
         pY3 = 440;
         constante = 30;
         cambia = 1;
         do {
            if (cambia == 1) {
               cambia = 2;
               lapiz = lapizAzul;
            }
            else {
               cambia = 1;
               lapiz = lapizRojo;
            }

            grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
            grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
            grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
            pX1 += constante;
            pY1 -= constante;
            pX2 += constante;
            pX3 += constante;
            pY3 += constante;
         }
         while (pY1 >= 0);
      }
   }
}
```

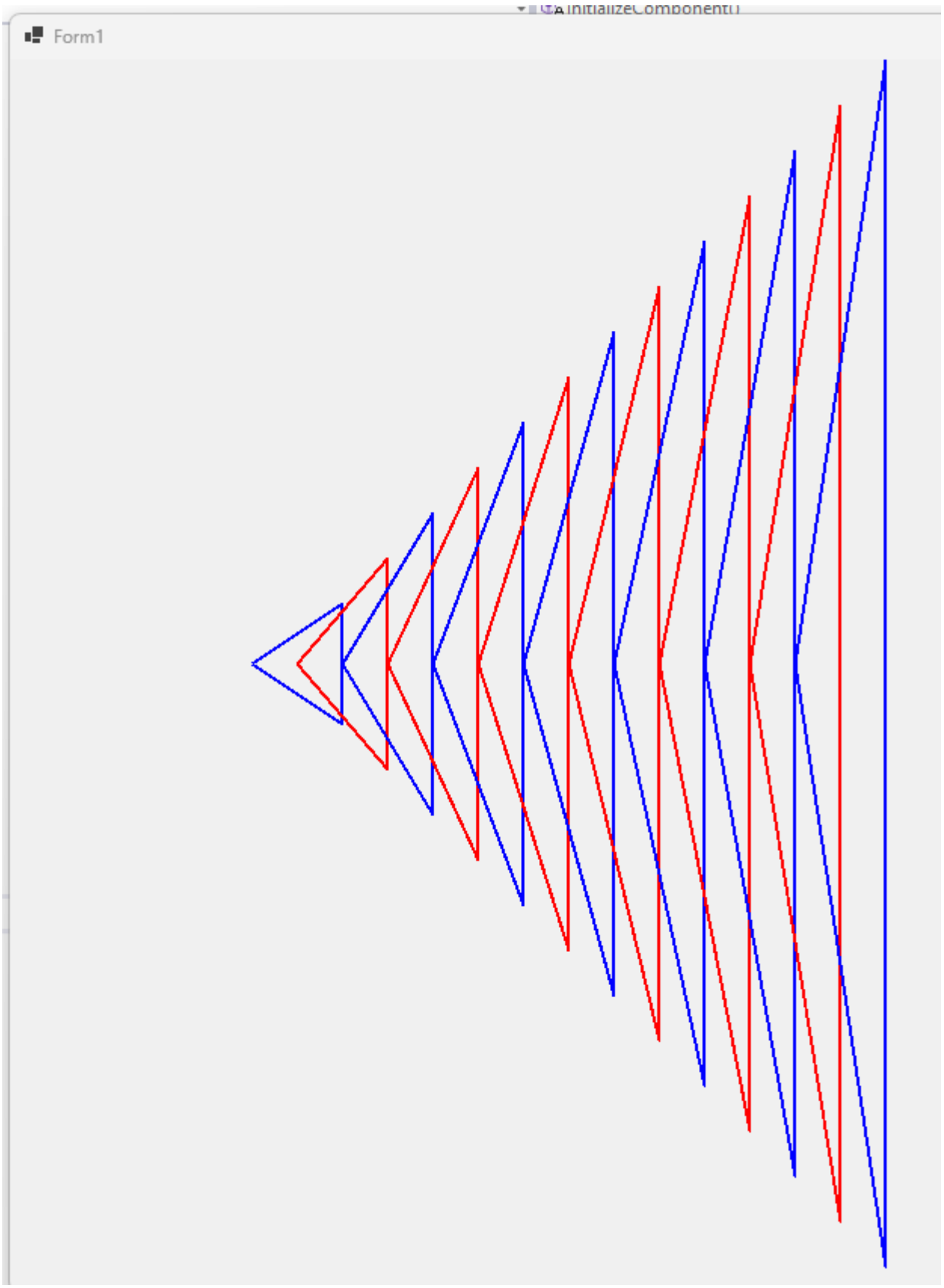

Ilustración 35: Triángulos apuntan a la izquierda alternando colores

Rectángulos concéntricos

L/027.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapizA = new Pen(Color.Blue, 2);
         Pen lapizB = new Pen(Color.Red, 2);
         Pen lapizC = new Pen(Color.Chocolate, 2);
         Pen lapizD = new Pen(Color.DarkGreen, 2);

         int variar = 0;
         int centro = 380;

         for (var cont = 1; cont <= 30; cont += 1) {
            int valA = centro - variar;
            int valB = centro + variar;
            grafico.DrawLine(lapizA, valA, valA, valB, valA);
            grafico.DrawLine(lapizB, valA, valA, valA, valB);
            grafico.DrawLine(lapizC, valB, valA, valB, valB);
            grafico.DrawLine(lapizD, valA, valB, valB, valB);
            variar += 10;
         }
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

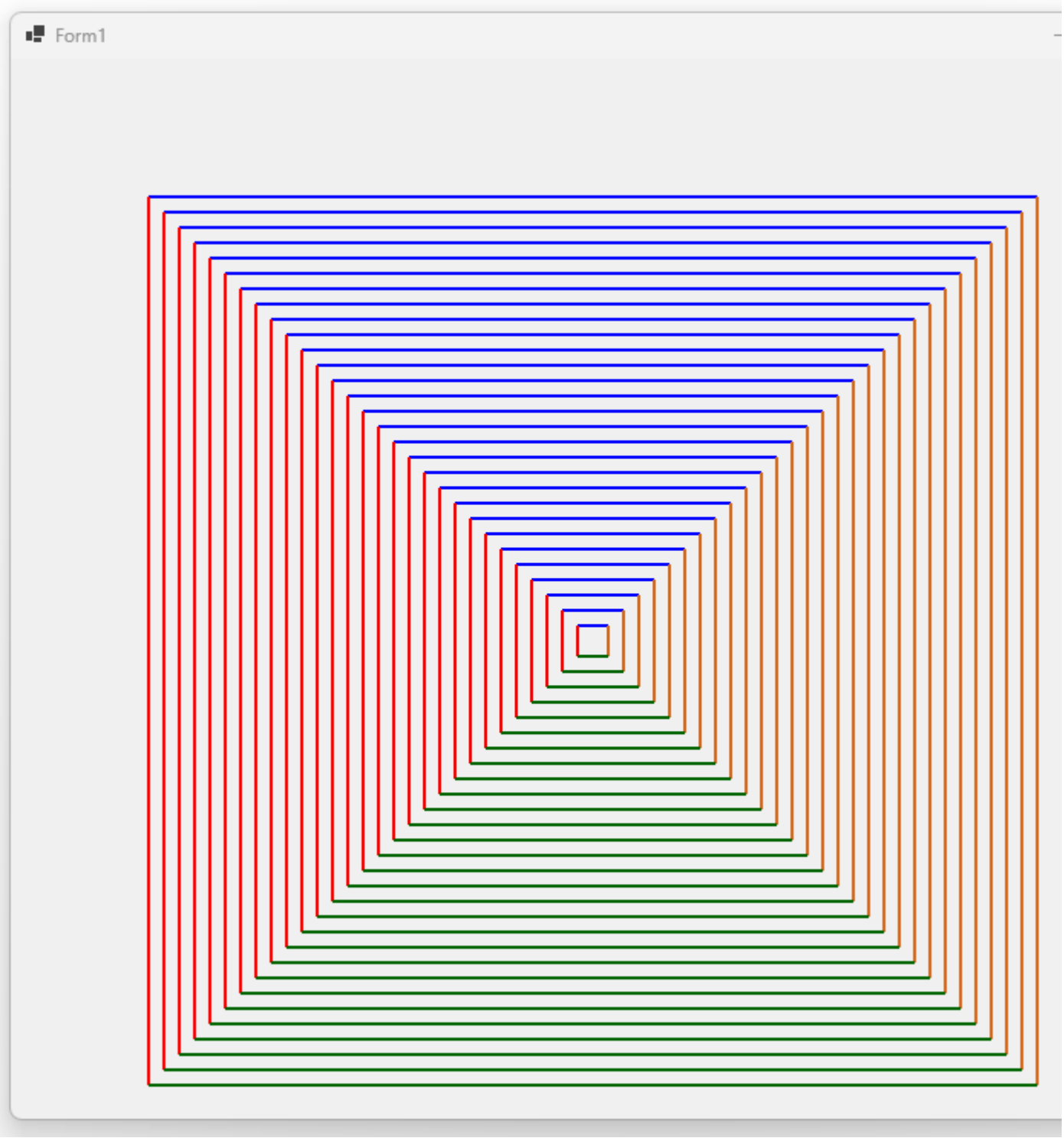

Ilustración 36: Rectángulos concéntricos

Dibuja formas que semejan cuadrados a las que les falta el techo o la base

L/028.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics lienzo = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int variar = 0;

         for (var cont = 1; cont <= 18; cont += 1) {
            int valA = 380 - variar;
            int valB = 380 + variar;
            lienzo.DrawLine(lapiz, valA, valA, valB, valA);
            lienzo.DrawLine(lapiz, valA, valA, valA, valB);
            lienzo.DrawLine(lapiz, valB, valA, valB, valB);
            variar += 10;

            lienzo.DrawLine(lapiz2, valA, valB, valB, valB);
            lienzo.DrawLine(lapiz2, valA, valA, valA, valB);
            lienzo.DrawLine(lapiz2, valB, valA, valB, valB);
            variar += 10;
         }
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

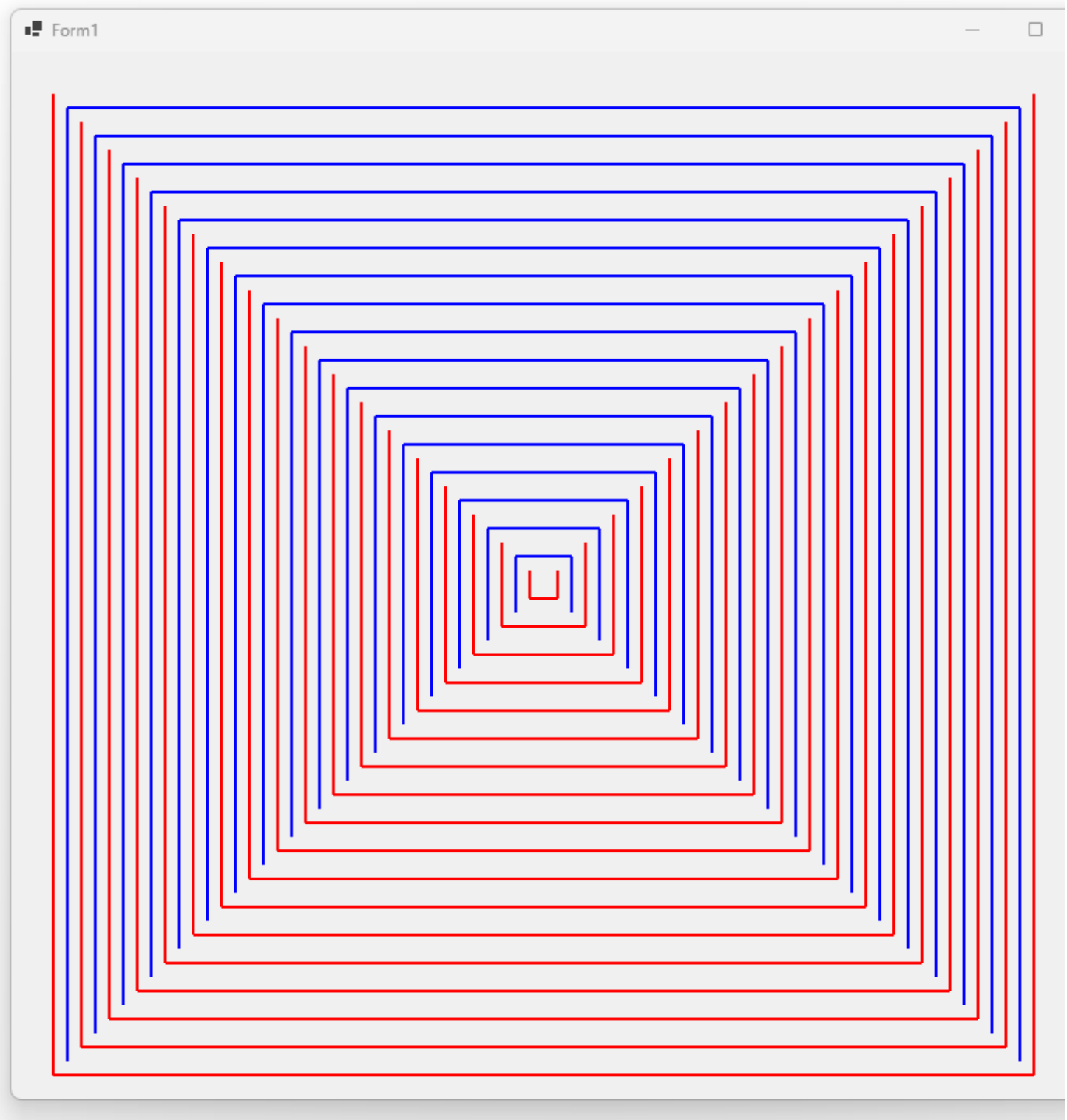

Ilustración 37: Dibuja formas que semejan cuadrados a las que les falta el techo o la base

Dibuja ángulos rectos

L/029.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics lienzo = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         int variar = 0;

         for (var cont = 1; cont <= 18; cont += 1) {
            int valA = 380 - variar;
            int valB = 380 + variar;
            lienzo.DrawLine(lapiz, valA, valA, valB, valA);
            lienzo.DrawLine(lapiz, valA, valA, valA, valB);
            variar += 10;

            lienzo.DrawLine(lapiz2, valA, valB, valB, valB);
            lienzo.DrawLine(lapiz2, valA, valA, valA, valB);
            variar += 10;
         }
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

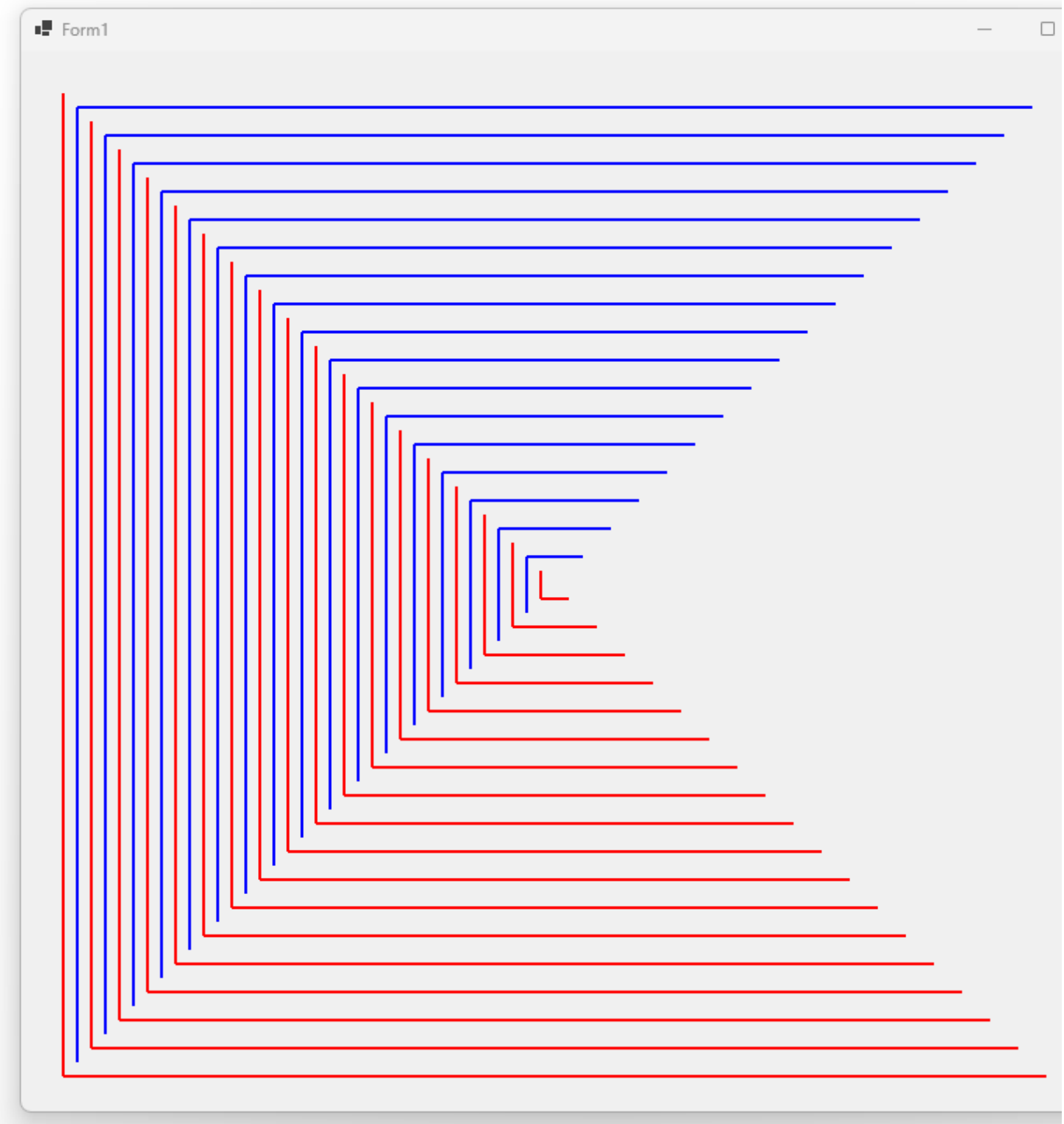

Ilustración 38: Dibuja ángulos rectos

Líneas diagonales cruzadas

L/030.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics grafico = e.Graphics;
         Pen lapiz = new Pen(Color.Blue, 2);
         Pen lapiz2 = new Pen(Color.Red, 2);
         Pen lapiz3 = new Pen(Color.Green, 2);
         Pen lapiz4 = new Pen(Color.Violet, 2);
         int cont = 0;

         do {
            grafico.DrawLine(lapiz, cont, 600, 600, cont);
            grafico.DrawLine(lapiz2, 1200 - cont, 600, 600, cont);
            grafico.DrawLine(lapiz3, 1200 - cont, 0, 600, 600 - cont);
            grafico.DrawLine(lapiz4, cont, 0, 600, 600 - cont);
            cont += 20;
         }
         while (cont <= 600);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

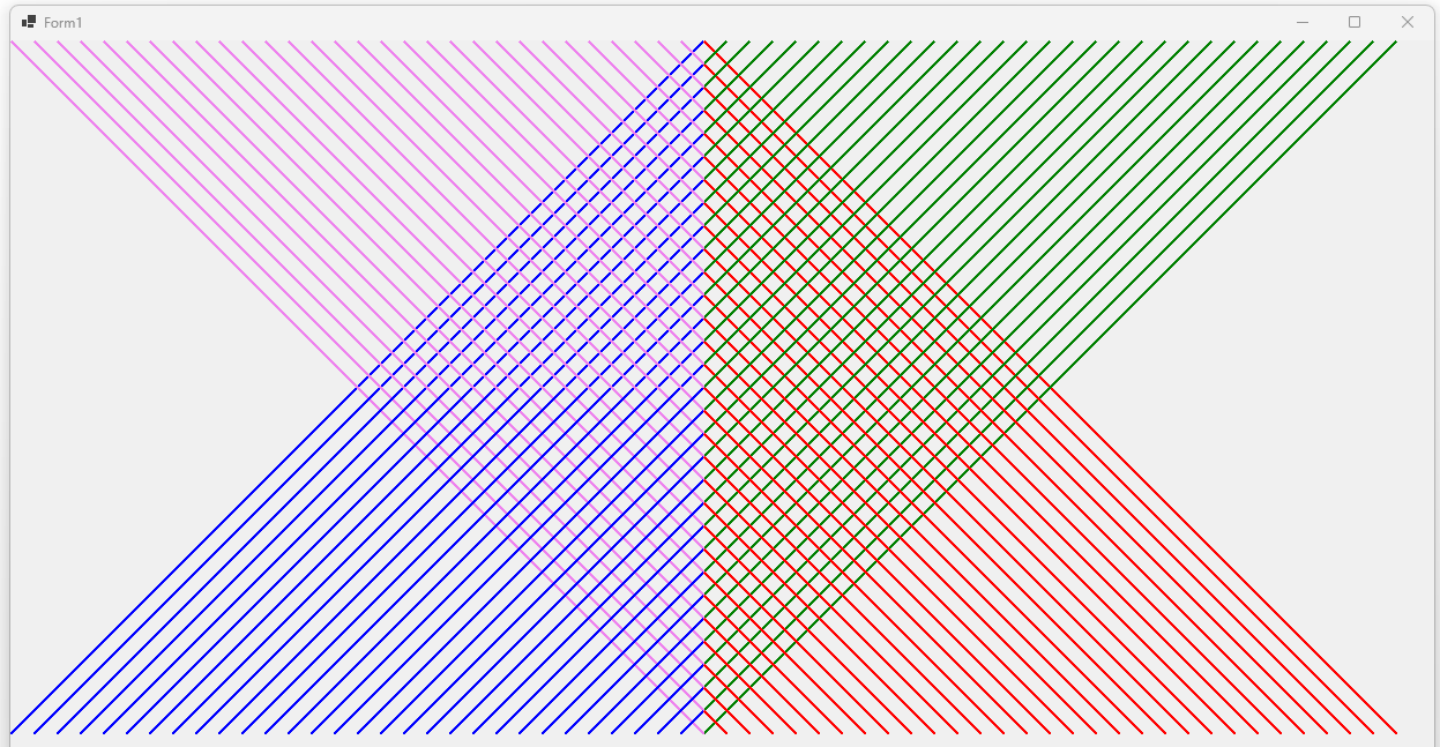

Ilustración 39: Líneas diagonales cruzadas

Celdas

L/031.cs

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

      private void Form1_Paint(object sender, PaintEventArgs e) {
         // Necesario para los gráficos GDI+
         Graphics grafico = e.Graphics;

         // Lápiz para el color de la línea y con un ancho de 1
         Pen lapiz = new Pen(Color.Blue, 1);

         for (int Fila = 0; Fil <= 10; Fil++) {
            for (int Col = 0; Col <= 10; Col++){
               int valA = Fil * 35 + 80;
               int valB = Col * 35 + 40;
               grafico.DrawRectangle(lapiz, valA, valB, 30, 30);
            }
         }
      }
   }
}
```

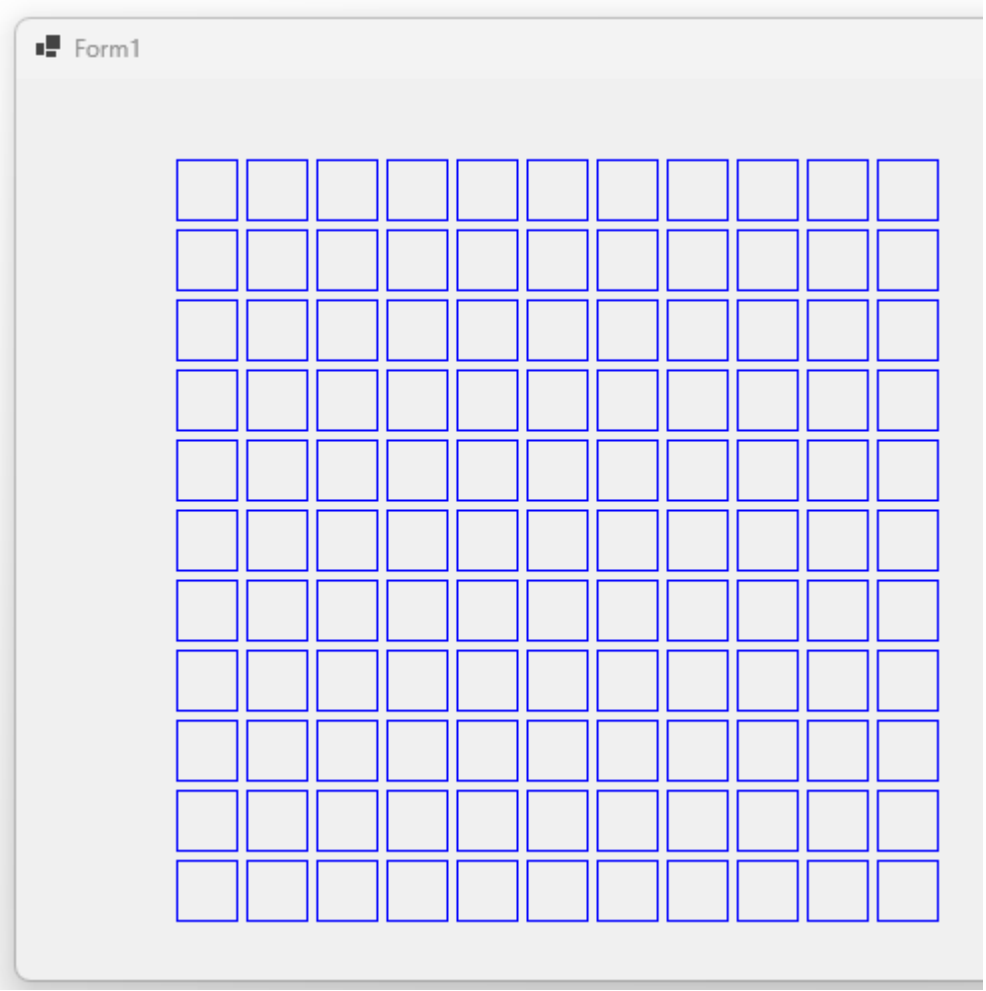

Ilustración 40: Celdas

Líneas rectas simulan curvas

L/032.cs

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

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

         // Lápiz para el color de la línea y con un ancho de 1
         Pen lapiz = new Pen(Color.Blue, 1);

         // Parte superior
         for (int Posicion = 0; Posicion <= 200; Posicion += 20) {
            // Línea requiere:  Lapiz, X1, Y1, X2, Y2
            grafico.DrawLine(lapiz, 200, Posicion, Posicion + 200, 200);
            grafico.DrawLine(lapiz, 200, Posicion, 200 - Posicion, 200);
         }

         //Parte inferior
         for (int Posicion = 400; Posicion >= 200; Posicion += -20) {
            // Línea requiere:  Lapiz, X1, Y1, X2, Y2
            grafico.DrawLine(lapiz, 200, Posicion, 600 - Posicion, 200);
            grafico.DrawLine(lapiz, 200, Posicion, Posicion - 200, 200);
         }
      }
   }
}
```

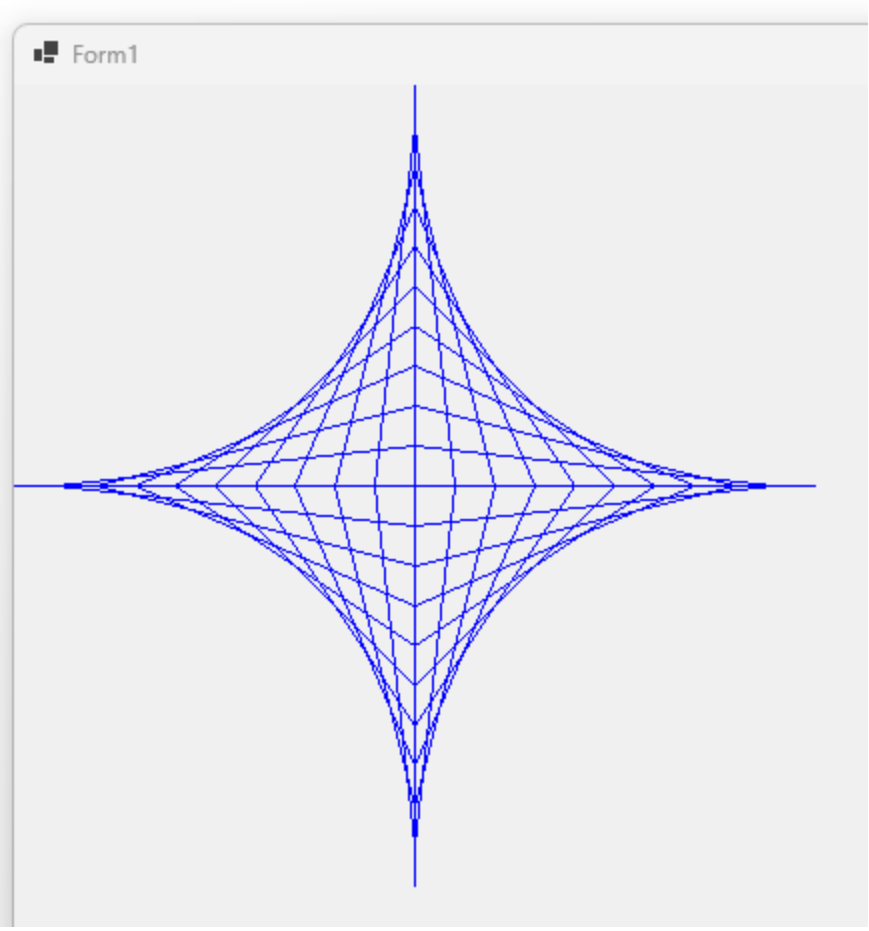

Ilustración 41: Líneas rectas simulan curvas

Algoritmo de Bresenham para dibujar líneas

Este algoritmo dibuja líneas rectas dadas las coordenadas inicial y final. Lo interesante es que no hace operaciones de punto flotante (que sería lento) sino operaciones de enteros (más rápido). Ese es el algoritmo de la función DrawLine

L/033.cs

```
//Algoritmo de Bresenham, para dibujar líneas rectas rápidamente
namespace Graficos {
   public partial class Form1 : Form {

      public Form1() {
         InitializeComponent();
      }

      public void Linea(Graphics lienzo,
                        int iniX, int iniY,
                        int finX, int finY) {
         int Contador, Distancia;
         int Xerror=0, Yerror=0;
         int CambioX, CambioY, incrementoX, incrementoY;

         CambioX = finX - iniX;
         CambioY = finY - iniY;

         if (CambioX > 0) incrementoX = 1;
         else if (CambioX == 0) incrementoX = 0;
         else incrementoX = -1;

         if (CambioY > 0) incrementoY = 1;
         else if (CambioY == 0) incrementoY = 0;
         else incrementoY = -1;

         if (CambioX < 0) CambioX *= -1;
         if (CambioY < 0) CambioY *= -1;

         if (CambioX > CambioY)
            Distancia = CambioX;
         else
            Distancia = CambioY;

         for (Contador = 0; Contador <= Distancia + 1; Contador++) {
            lienzo.FillRectangle(Brushes.Black, iniX, iniY, 1, 1);
            Xerror += CambioX;
            Yerror += CambioY;
            if (Xerror > Distancia) {
               Xerror -= Distancia;
```

```
            iniX += incrementoX;
         }

         if (Yerror > Distancia) {
            Yerror -= Distancia;
            iniY += incrementoY;
         }
      }
   }

   private void Form1_Paint(object sender, PaintEventArgs e) {
      //Ejemplo
      Linea(e.Graphics, 16, 83, 197, 206);
   }
}
}
```

Ilustración 42: Algoritmo de Bresenham para dibujar líneas

Transformaciones en 2D

Traslado de figuras en un plano

Mover la figura en el eje X y en el eje Y, es simplemente sumar o restar a los valores de las coordenadas:

NuevaPosicionX = PosicionOriginalX + MueveX

NuevaPosicionY = PosicionOriginalY + MueveY

L/034.cs

```
//Traslado de figuras en un plano
namespace Graficos {
   public partial class Form1 : Form {

      public Form1() {
         InitializeComponent();
      }

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

         //Traslado horizontal
         int mueveX = 0;

         //Traslado vertical
         int mueveY = 0;

         //Nuevas coordenadas
         int posX1 = 10 + mueveX;
         int posX2 = 400 + mueveX;
         int posY1 = 10 + mueveY;
         int posY2 = 250 + mueveY;

         //Dibuja un rectángulo con cuatro líneas
         lienzo.DrawLine(Pens.Black, posX1, posY1, posX2, posY1);
         lienzo.DrawLine(Pens.Black, posX1, posY2, posX2, posY2);
         lienzo.DrawLine(Pens.Black, posX2, posY1, posX2, posY2);
         lienzo.DrawLine(Pens.Black, posX1, posY1, posX1, posY2);
      }
   }
}
```


Autor: Rafael Alberto Moreno Parra

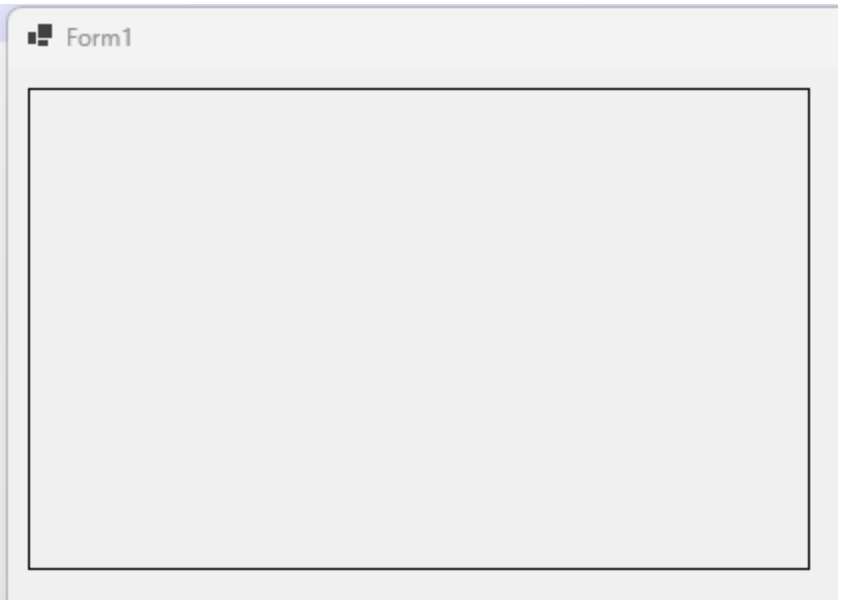

Ilustración 43: Traslado de figuras en un plano

Ilustración 44: Traslado de figuras en un plano mueveX = 70

Ilustración 45: Traslado de figuras en un plano mueveY = 70

Ilustración 46: Traslado de figuras en un plano mueveX = 70 y mueveY = 70

Giro de figuras en un plano

Para el giro en determinado ángulo, se usa la siguiente fórmula para una coordenada plana:

PosicionNuevaX = PosicionOriginalX * coseno(Angulo) - PosicionOriginalY * seno(Angulo)

PosicionNuevaY = PosicionOriginalX * seno(Angulo) + PosicionOriginalY * coseno(Angulo)

L/035.cs

```
//Giro de figuras en un plano
namespace Graficos {
   public partial class Form1 : Form {

      public Form1() {
         InitializeComponent();
      }

      private void Form1_Paint(object sender, PaintEventArgs e) {
         //Coordenadas de la figura
         int posXa, posYa, posXb, posYb, posXc, posYc;

         posXa = 80;
         posYa = 80;
         posXb = 400;
         posYb = 300;
         posXc = 500;
         posYc = 200;

         //Dibuja un triángulo con tres líneas
         e.Graphics.DrawLine(Pens.Black, posXa, posYa, posXb, posYb);
         e.Graphics.DrawLine(Pens.Black, posXb, posYb, posXc, posYc);
         e.Graphics.DrawLine(Pens.Black, posXc, posYc, posXa, posYa);

         //Ángulo de giro
         int AnguloGiro = 15;
         double AnguloRadianes = AnguloGiro * Math.PI / 180;
         double CosA = Math.Cos(AnguloRadianes);
         double SinA = Math.Sin(AnguloRadianes);

         //Cálcula el giro
         int posXga = Convert.ToInt32(posXa * CosA - posYa * SinA);
         int posYga = Convert.ToInt32(posXa * SinA + posYa * CosA);

         int posXgb = Convert.ToInt32(posXb * CosA - posYb * SinA);
         int posYgb = Convert.ToInt32(posXb * SinA + posYb * CosA);

         int posXgc = Convert.ToInt32(posXc * CosA - posYc * SinA);
```

```
            int posYgc = Convert.ToInt32(posXc * SinA + posYc * CosA);

            //Dibuja el triángulo con el giro
            e.Graphics.DrawLine(Pens.Red, posXga, posYga, posXgb, posYgb);
            e.Graphics.DrawLine(Pens.Red, posXgb, posYgb, posXgc, posYgc);
            e.Graphics.DrawLine(Pens.Red, posXgc, posYgc, posXga, posYga);
        }
    }
}
```

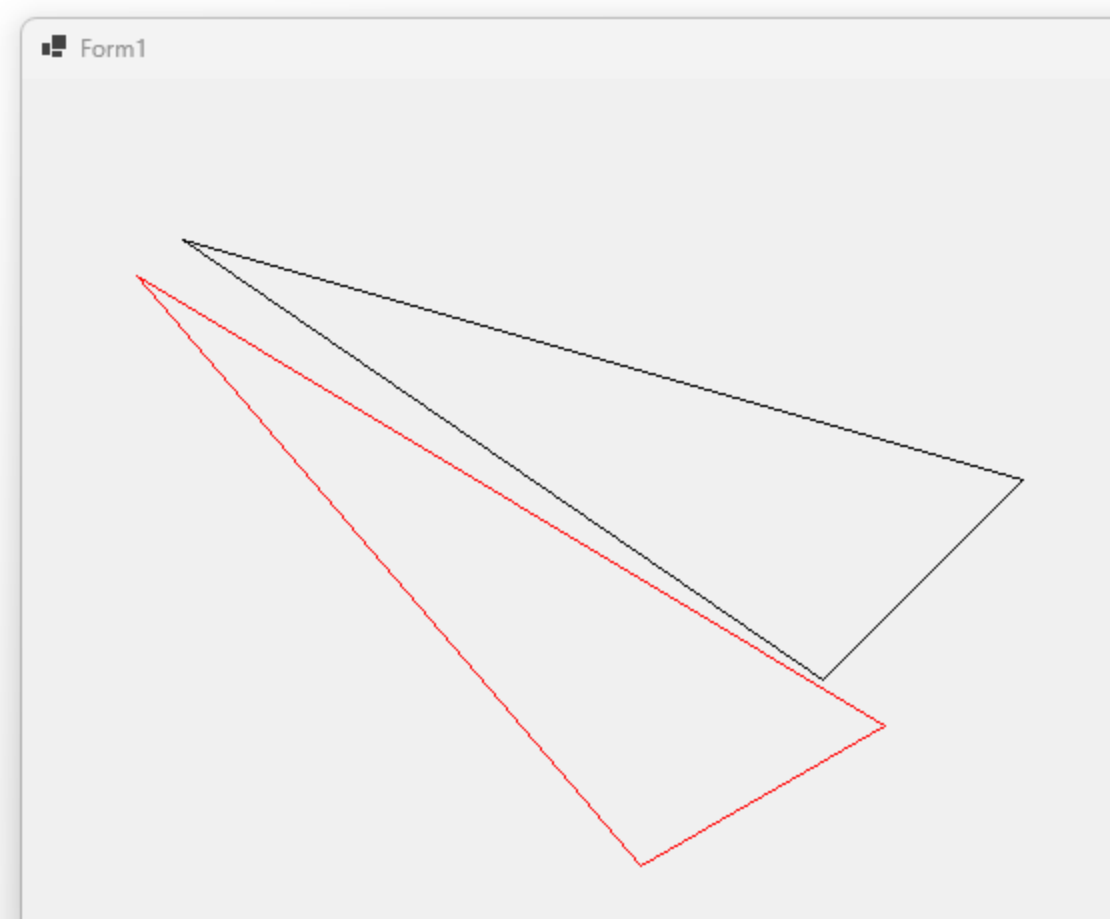

Ilustración 47: Giro de figuras en un plano

En el ejemplo, se usó un giro de 5 grados (que se deben convertir en radianes antes de hacer uso de las funciones de seno y coseno). El ángulo se aplica desde la posición 0,0 de la ventana.

Giro de figuras en un plano calculando el centroide

Se requiere calcular el centroide de la figura (fácil si es un rectángulo), se aplica el giro y se compara cuánto se desplazó en X y Y con respecto a la posición original. Una vez obtenidos esos datos, se restaura para toda la figura y se obtiene el giro sobre sí misma.

L/036.cs

```
//Giro de figuras en un plano calculando el centroide
namespace Graficos {
   public partial class Form1 : Form {

      public Form1() {
         InitializeComponent();
      }

      private void Form1_Paint(object sender, PaintEventArgs e) {
         Graphics lienzo = e.Graphics;
         Pen lapizA = new(Color.Black, 1);
         pen lapizB = new(Color.Red, 2);

         //Datos del rectángulo
         int Xa, Ya, Largo, Alto;
         Xa = 80;
         Ya = 80;
         Largo = 400;
         Alto = 180;

         //Deduce las otras tres coordenadas del rectángulo
         int Xb, Yb, Xc, Yc, Xd, Yd;
         Xb = Xa + Largo;
         Yb = Ya;
         Xc = Xa + Largo;
         Yc = Ya + Alto;
         Xd = Xa;
         Yd = Ya + Alto;

         //Dibuja un rectángulo con cuatro líneas
         lienzo.DrawLine(lapizA, Xa, Ya, Xb, Yb);
         lienzo.DrawLine(lapizA, Xb, Yb, Xc, Yc);
         lienzo.DrawLine(lapizA, Xc, Yc, Xd, Yd);
         lienzo.DrawLine(lapizA, Xd, Yd, Xa, Ya);

         //Ángulo de giro
         int AnguloGiro = 15;
         double AnguloRadianes = AnguloGiro * Math.PI / 180;
         double CosA = Math.Cos(AnguloRadianes);
         double SinA = Math.Sin(AnguloRadianes);
```

```
        //Centroide
        int Xcentro = Xa + Largo / 2;
        int Ycentro = Ya + Alto / 2;

        //Cálcula el giro
        int Xga = Convert.ToInt32(Xa * CosA - Ya * SinA);
        int Yga = Convert.ToInt32(Xa * SinA + Ya * CosA);

        int Xgb = Convert.ToInt32(Xb * CosA - Yb * SinA);
        int Ygb = Convert.ToInt32(Xb * SinA + Yb * CosA);

        int Xgc = Convert.ToInt32(Xc * CosA - Yc * SinA);
        int Ygc = Convert.ToInt32(Xc * SinA + Yc * CosA);

        int Xgd = Convert.ToInt32(Xd * CosA - Yd * SinA);
        int Ygd = Convert.ToInt32(Xd * SinA + Yd * CosA);

        //Giro del centroide
        int Xgcentro = Convert.ToInt32(Xcentro * CosA - Ycentro * SinA);
        int Ygcentro = Convert.ToInt32(Xcentro * SinA + Ycentro * CosA);

        //¿Cuánto se desplazo el centroide?
        int dX = Xgcentro - Xcentro;
        int dY = Ygcentro - Ycentro;

        //Dibuja el triángulo con el giro
        lienzo.DrawLine(lapizB, Xga - dX, Yga - dY, Xgb - dX, Ygb - dY);
        lienzo.DrawLine(lapizB, Xgb - dX, Ygb - dY, Xgc - dX, Ygc - dY);
        lienzo.DrawLine(lapizB, Xgc - dX, Ygc - dY, Xgd - dX, Ygd - dY);
        lienzo.DrawLine(lapizB, Xga - dX, Yga - dY, Xgd - dX, Ygd - dY);
    }
  }
}
```

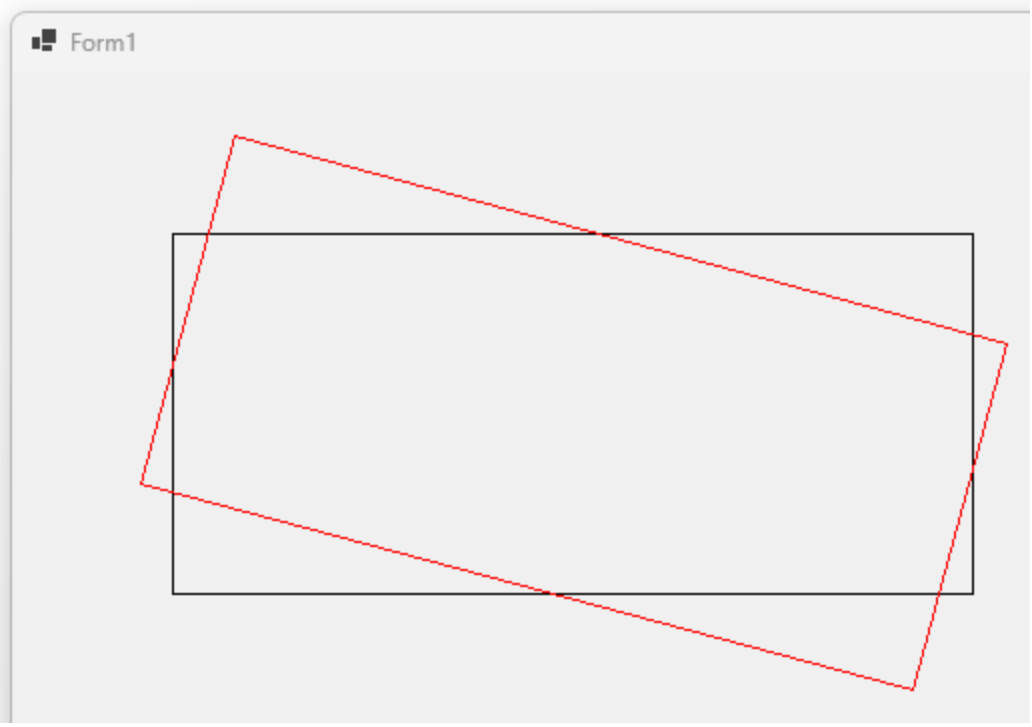

Ilustración 48: Giro de figuras en un plano calculando el centroide

Gráfico matemático en 2D

Dada una ecuación del tipo Y=F(X), por ejemplo:

$$Y = 0.1X^6 + 0.6X^5 - 0.7X^4 - 5.7X^3 + 2X^2 + 2X + 1$$

Para graficarla, se deben hacer estos pasos:

1. Saber el valor de inicio de X y el valor final de X. Se llamarán Xini y Xfin respectivamente.
2. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
3. Con esos datos, se hace uso de un ciclo que calcule los valores de Y. Se almacena el par X, Y
4. 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
5. Se requieren cuatro datos:
    a. El mínimo valor de X. Es el mismo Xini.
    b. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
    c. El mínimo valor de Y obtenido. Se llamará Ymin.
    d. El máximo valor de Y obtenido. Se llamará Ymax.
6. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
    a. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
    b. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
7. Se calculan unas constantes de conversión con estas fórmulas:
    a. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
    b. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
8. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
    a. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
    b. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
9. Se grafican esos puntos y se unen con líneas.

La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

```
namespace Graficos {
   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;
      }
   }

   public partial class Form1 : Form {
      List<Puntos> punto;
      int XpIni, YpIni, XpFin, YpFin;

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

         //Área dónde se dibujará el gráfico matemático
         XpIni = 20;
         YpIni = 20;
         XpFin = 600;
         YpFin = 400;

         //Algoritmo que calcula los puntos del gráfico
         double minX = -5;
         double maxX = 3;
         int numPuntos = 80;
         Logica(minX, maxX, numPuntos);
      }

      public void Logica(double Xini, double Xfin, int numPuntos) {
         //Calcula los puntos de la ecuación a graficar
         double pasoX = (Xfin - Xini) / numPuntos;
         double Ymin = double.MaxValue; //El mínimo valor de Y obtenido
         double Ymax = double.MinValue; //El máximo valor de Y obtenido

         //El máximo valor de X (difiere de Xfin)
         double maximoXreal = double.MinValue;

         punto.Clear();
         for (double X = Xini; X <= Xfin; X += pasoX) {
            //Se invierte el valor porque el eje Y
```

```
            //aumenta hacia abajo
            double valY = -1 * Ecuacion(X);
            if (valY > Ymax) Ymax = valY;
            if (valY < Ymin) Ymin = valY;
            if (X > maximoXreal) maximoXreal = X;
            punto.Add(new Puntos(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 - Xini);
        double conY = (YpFin - YpIni) / (Ymax - Ymin);

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

    //Aquí está la ecuación que se desee graficar
    //con variable independiente X
    public double Ecuacion(double X) {
        double Y = 0.1 * Math.Pow(X, 6) + 0.6 * Math.Pow(X, 5);
        Y -= 0.7 * Math.Pow(X, 4) - 5.7 * Math.Pow(X, 3);
        Y += 2 * X * X + 2 * X + 1;
        return Y;
    }

    //Pinta la ecuación
    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics lienzo = e.Graphics;
        Pen lapiz = new (Color.Blue, 3);
        Brush llena = new SolidBrush(Color.Black);

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

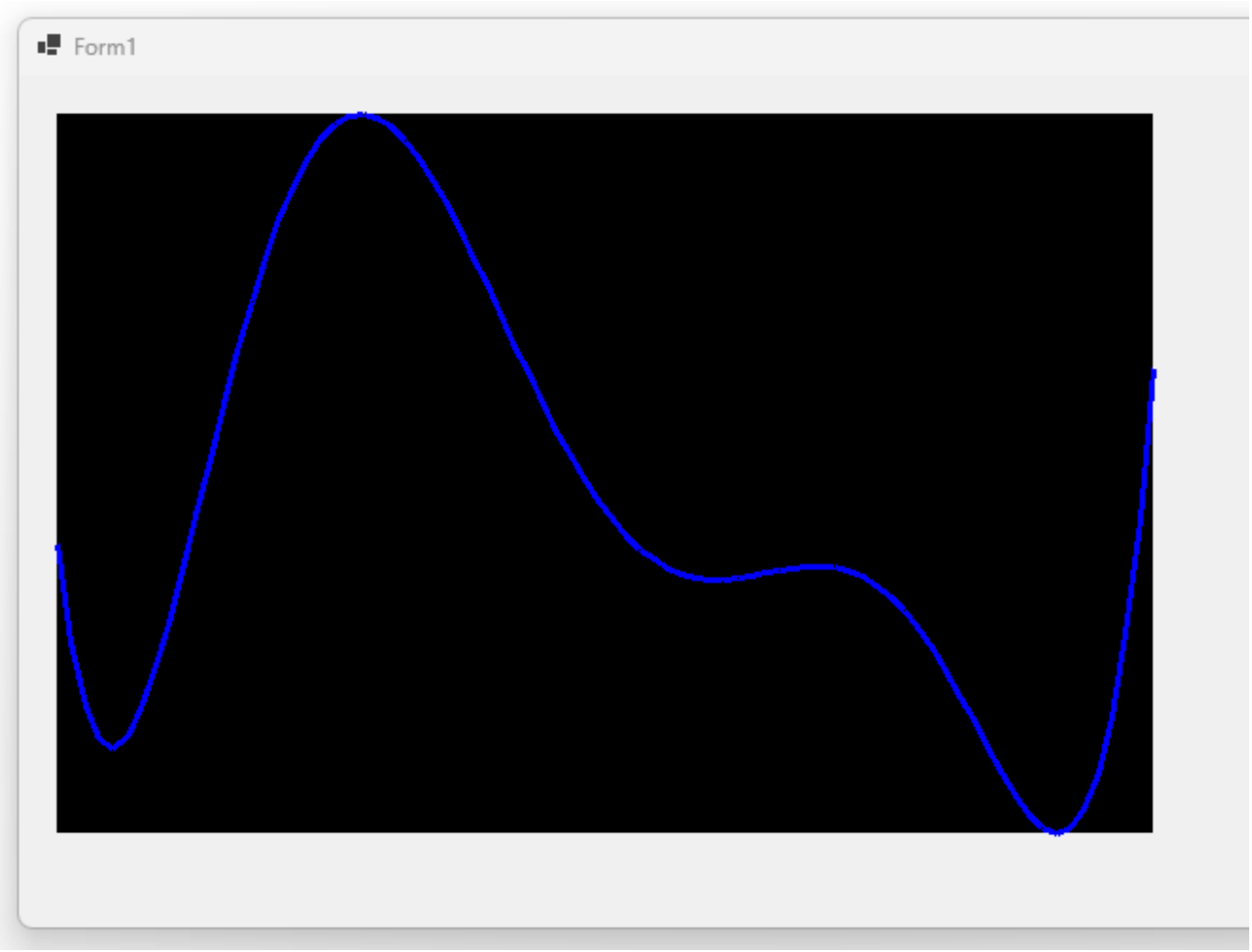

Ilustración 49: Gráfico matemático en 2D

Gráfico polar

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

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

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de Θ y el valor final de Θ. Se llamarán minTheta y maxTheta respectivamente.
2. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
3. Con esos datos, se hace uso de un ciclo que calcule los valores de r. Luego se hace esta conversión:

```
r = Ecuacion(theta);

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

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

4. 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
5. Se requieren cuatro datos:
    a. El mínimo valor de X. Es el mismo Xinia.
    b. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
    c. El mínimo valor de Y obtenido. Se llamará Ymin.
    d. El máximo valor de Y obtenido. Se llamará Ymax.
6. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
    a. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
    b. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
7. Se calculan unas constantes de conversión con estas fórmulas:
    a. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
    b. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
8. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
    a. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
    b. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
9. Se grafican esos puntos y se unen con líneas.

Autor: Rafael Alberto Moreno Parra

La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

L/038.cs

```
namespace Graficos {
   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;
      }
   }

   public partial class Form1 : Form {
      List<Puntos> punto;
      int XpIni, YpIni, XpFin, YpFin;

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

         //Área dónde se dibujará el gráfico matemático
         XpIni = 20;
         YpIni = 20;
         XpFin = 600;
         YpFin = 400;

         //Algoritmo que calcula los puntos del gráfico
         double minTheta = 0;
         double maxTheta = 360;
         int numPuntos = 200;
         Logica(minTheta, maxTheta, numPuntos);
      }

      public void Logica(double thIni, double thFin, int numPuntos) {
         //Calcula los puntos de la ecuación a graficar
         double pasoT = (thFin - thIni) / 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 = thIni; theta <= thFin; theta += pasoT) {
         double valorR = Ecuacion(theta);
         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 pX = conX * (punto[cont].X - Xmin) + XpIni;
         double pY = conY * (punto[cont].Y - Ymin) + YpIni;
         punto[cont].pX = Convert.ToInt32(pX);
         punto[cont].pY = Convert.ToInt32(pY);
      }
   }

   //Aquí está la ecuación polar que se desee
   //graficar con variable Theta
   public double Ecuacion(double Theta) {
      return 2 * Math.Sin(4 * (Theta * Math.PI / 180));
   }

   //Pinta la ecuación
   private void Form1_Paint(object sender, PaintEventArgs e) {
      Graphics lienzo = e.Graphics;
      Pen lapiz = new(Color.Blue, 3);
      Brush llena = new SolidBrush(Color.Black);

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

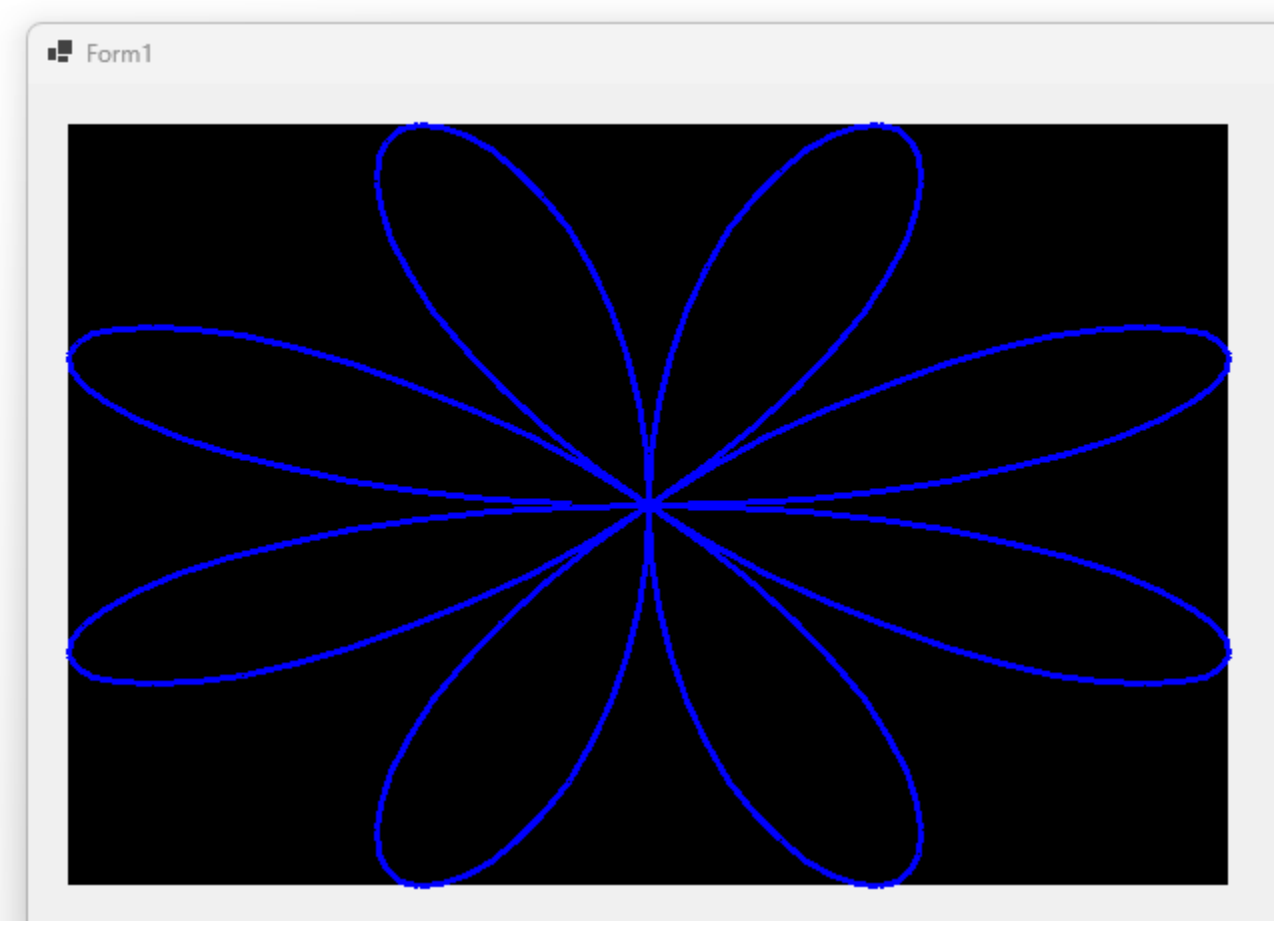

Ilustración 50: Gráfico polar

Uso de Bitmap para control a nivel de pixel

C# trae Bitmap para control sobre el pixel, estas son sus características:

Control total del píxel

1. Se puede usar "SetPixel(x, y, color)" para modificar cada píxel individualmente.
2. Es Ideal para algoritmos como Z-buffer, ray tracing, rasterización manual, etc.

Persistencia del dibujo

1. El contenido del "Bitmap" se mantiene en memoria.
2. Se puede redibujar la imagen fácilmente en cada "Paint" sin recalcular todo.

Separación de lógica y presentación

1. Se puede hacer todo el cálculo y renderizado en memoria.
2. Luego simplemente es mostrar el resultado con "Graphics.DrawImage()".

Mayor eficiencia en renderizados complejos

1. Aunque `SetPixel` no es muy rápido, se puede optimizar usando `LockBits` si es necesario.
2. Se evita redibujar todo desde cero en cada evento `Paint`.

L/039.cs

```
namespace Graficos {
   public partial class Form1 : Form {
      private Bitmap Lienzo;

      public Form1() {
         InitializeComponent();
         Lienzo = new Bitmap(400, 300);
      }

      //Pintar
      private void Form1_Paint(object sender, PaintEventArgs e) {
         DibujarLinea(50, 50, 300, 200, Color.Blue); // Línea de ejemplo
         e.Graphics.DrawImage(Lienzo, 0, 0);
      }

      //Algoritmo de Bresenham para líneas
      private void DibujarLinea(int Xini, int Yini, int Xfin, int Yfin,
Color color) {
```

```
            int dx = Math.Abs(Xfin - Xini), sx = Xini < Xfin ? 1 : -1;
            int dy = -Math.Abs(Yfin - Yini), sy = Yini < Yfin ? 1 : -1;
            int error = dx + dy, e2;

            while (true) {
               if (Xini >= 0 && Xini < Lienzo.Width && Yini >= 0 && Yini <
Lienzo.Height)
                  Lienzo.SetPixel(Xini, Yini, color); //Dibuja el pixel

               if (Xini == Xfin && Yini == Yfin) break;
               e2 = 2 * error;
               if (e2 >= dy) { error += dy; Xini += sx; }
               if (e2 <= dx) { error += dx; Yini += sy; }
            }
         }
      }
   }
```

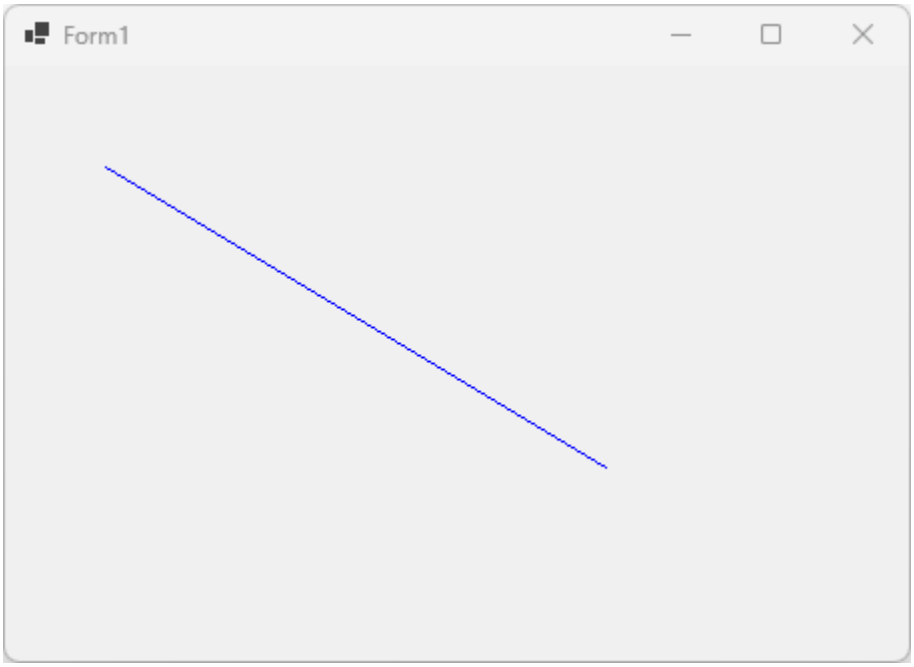

Ilustración 51: Línea dibujada a nivel de pixel

Uso de Bitmap y Graphics

Se puede hacer uso de la API de Graphics dentro de Bitmap.

L/040.cs

```
namespace Graficos {

   public partial class Form1 : Form {
      private Bitmap Lienzo;

      public Form1() {
         InitializeComponent();

         //Uso de bitmap para hacer gráficos
         Lienzo = new Bitmap(200, 200);
         Graphics Grafico = Graphics.FromImage(Lienzo);
         Grafico.Clear(Color.White);
         Pen Lapiz = new(Color.Green, 4);
         Grafico.DrawEllipse(Lapiz, 25, 25, 150, 150); // Círculo
      }

      //Pintar
      private void Form1_Paint(object sender, PaintEventArgs e) {
         e.Graphics.DrawImage(Lienzo, 50, 50);
      }
   }
}
```

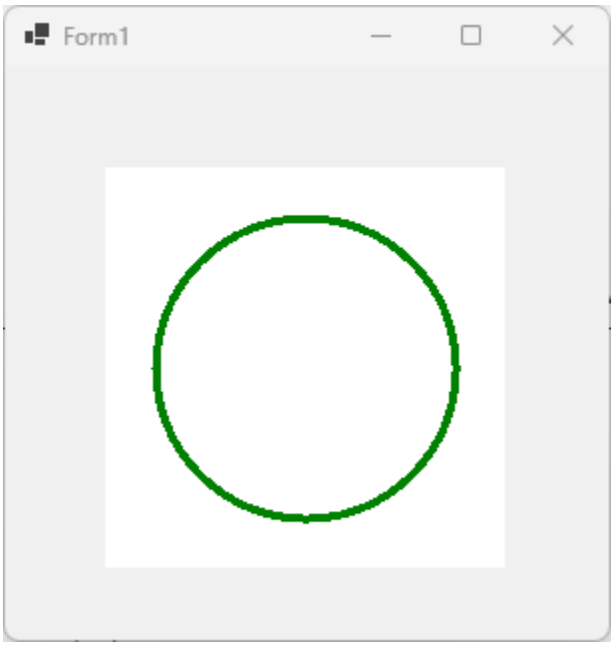

Ilustración 52: Uso de Bitmap y Graphics

Uso de LockBits para mayor velocidad

El código es más complejo y largo pero trae como beneficio la velocidad.

L/041.cs

```
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace Graficos {

   public partial class Form1 : Form {
      private Bitmap Lienzo;

      public Form1() {
         Lienzo = new Bitmap(400, 300, PixelFormat.Format24bppRgb);
         InitializeComponent();
      }

      //Pintar
      private void Form1_Paint(object sender, PaintEventArgs e) {
         DibujaLineaUsandoLockBits(100, 100, 301, 201, Color.Red);
         e.Graphics.DrawImage(Lienzo, 0, 0);
      }

      /*
       * LockBits bloquea una parte del Bitmap en memoria para que pueda
acceder directamente a los datos de píxeles
       * mediante punteros o arrays. Esto es mucho más rápido que usar
GetPixel y SetPixel, que son lentos porque
       * implican muchas llamadas a funciones nativas.
       *
       * Usa LockBits cuando:
       * Es necesario modificar muchos píxeles rápidamente (por ejemplo,
filtros de imagen, efectos, procesamiento de imágenes).
       * Se está haciendo animaciones por píxel o manipulación de texturas.
       * Se desea leer o escribir datos de píxeles en bloque.
       * */
      private void DibujaLineaUsandoLockBits(int Xini, int Yini, int Xfin,
int Yfin, Color color) {
         Rectangle Rectangulo = new(0, 0, Lienzo.Width, Lienzo.Height);
         BitmapData Datos = Lienzo.LockBits(Rectangulo,
ImageLockMode.ReadWrite, Lienzo.PixelFormat);

         int bytesPorPixel = Image.GetPixelFormatSize(Lienzo.PixelFormat) /
8;
```

```
            int Paso = Datos.Stride;
            IntPtr ptr = Datos.Scan0;
            int Alto = Lienzo.Height;
            int Ancho = Lienzo.Width;

            byte[] Pixeles = new byte[Paso * Alto];
            Marshal.Copy(ptr, Pixeles, 0, Pixeles.Length);

            // Algoritmo de Bresenham
            int dx = Math.Abs(Xfin - Xini), sx = Xini < Xfin ? 1 : -1;
            int dy = -Math.Abs(Yfin - Yini), sy = Yini < Yfin ? 1 : -1;
            int Error = dx + dy, Error2;

            while (true) {
               if (Xini >= 0 && Xini < Ancho && Yini >= 0 && Yini < Alto) {
                  int Indice = Yini * Paso + Xini * bytesPorPixel;
                  Pixeles[Indice + 0] = color.B;
                  Pixeles[Indice + 1] = color.G;
                  Pixeles[Indice + 2] = color.R;
               }

               if (Xini == Xfin && Yini == Yfin) break;
               Error2 = 2 * Error;
               if (Error2 >= dy) { Error += dy; Xini += sx; }
               if (Error2 <= dx) { Error += dx; Yini += sy; }
            }

            Marshal.Copy(Pixeles, 0, ptr, Pixeles.Length);
            Lienzo.UnlockBits(Datos);
         }
      }
   }
```

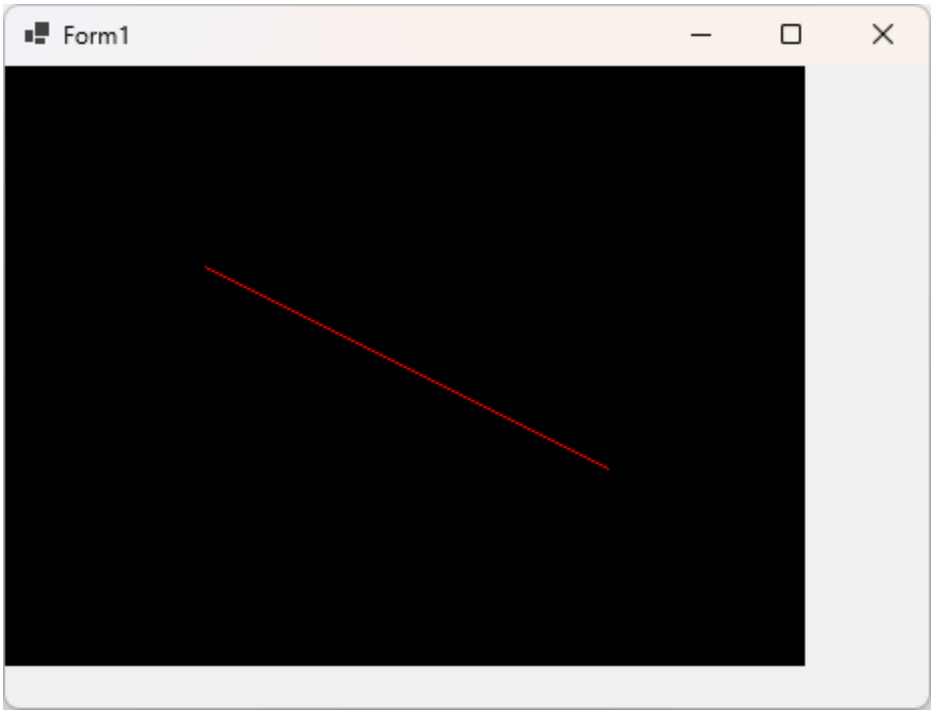

Ilustración 53: Uso de LockBits

LockBits dibujando un círculo

L/042.cs

```
/*
Si se está dibujando formas geométricas, usar Graphics.
Es más simple, legible y suficientemente rápido para
la mayoría de los casos.

Si se necesita modificar píxeles directamente
(como en procesamiento de imágenes), entonces
usar LockBits.
*/
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace Graficos {
    public partial class Form1 : Form {
        private Bitmap Lienzo;

        public Form1() {
            InitializeComponent();

            // Crear el bitmap
            Lienzo = new Bitmap(200, 200, PixelFormat.Format24bppRgb);

            // Usar LockBits para dibujar un círculo
            Rectangle rectangulo = new Rectangle(0, 0, Lienzo.Width, Lienzo.Height);
            BitmapData bmpDato = Lienzo.LockBits(rectangulo, ImageLockMode.ReadWrite,
Lienzo.PixelFormat);

            int paso = bmpDato.Stride;
            IntPtr ptr = bmpDato.Scan0;
            int bytes = Math.Abs(paso) * Lienzo.Height;
            byte[] rgbValues = new byte[bytes];

            Marshal.Copy(ptr, rgbValues, 0, bytes);

            int centroX = 100;
            int centroY = 100;
            int radio = 75;

            for (int y = 0; y < Lienzo.Height; y++) {
                for (int x = 0; x < Lienzo.Width; x++) {
                    int dx = x - centroX;
                    int dy = y - centroY;
                    int distancia = dx * dx + dy * dy;
```

```
                    // Dibujar solo el borde del círculo (ancho de 4 píxeles)
                    if (distancia >= radio * radio - 4 * radio && distancia <= radio * radio + 4
* radio) {
                        int indice = y * paso + x * 3;
                        rgbValues[indice] = 0;      // Blue
                        rgbValues[indice + 1] = 255; // Green
                        rgbValues[indice + 2] = 0;   // Red
                    }
                    else {
                        int indice = y * paso + x * 3;
                        rgbValues[indice] = 255;
                        rgbValues[indice + 1] = 255;
                        rgbValues[indice + 2] = 255;
                    }
                }
            }

            Marshal.Copy(rgbValues, 0, ptr, bytes);
            Lienzo.UnlockBits(bmpDato);
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            e.Graphics.DrawImage(Lienzo, 50, 50);
        }
    }
}
```

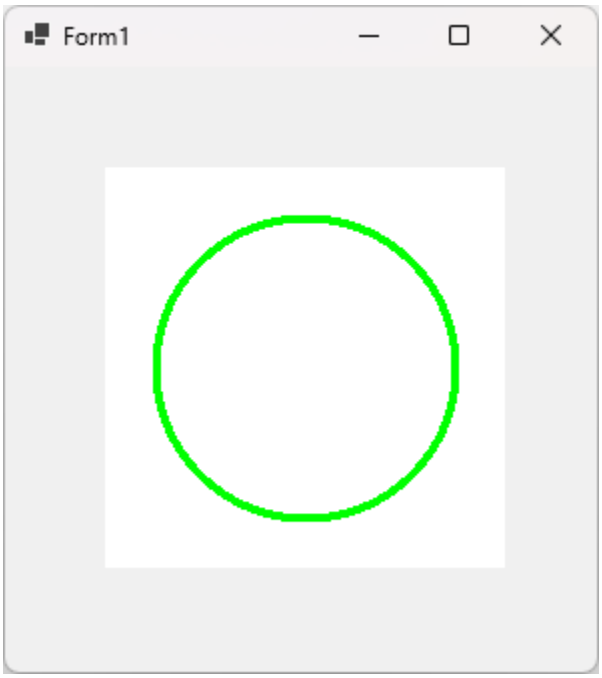

Ilustración 54: Dibujando un círculo con LockBits