UNIVERSIDAD DE LAS FUERZAS ARMADAS

Computación Gráfica

Departamento de Ciencias de la Computación

Pentágonos y Polígonos Estrellados de 5 Puntas

Estudiantes:
Daniel Armas, Isaac Escobar, Josué Merino

Docente:
Ing. Darío Morales

Objetivos

- Dado el lado de un pentágono, dibujar la figura geométrica correspondiente y los diferentes polígonos estrellados de 5 puntas, tal y como se muestra en la figura 1.2.1. Se debe considerar que las figuras geométricas se grafican con respecto al punto O(0,0).
- Explicar el algoritmo utilizado para calcular las coordenadas de los vértices del pentágono y los polígonos estrellados.
- Mostrar y explicar el código fuente desarrollado en C# .NET Framework para dibujar las figuras geométricas.
- Presentar capturas de pantalla o representaciones gráficas de las figuras generadas por el programa.

Explicación Matemática

El código realiza diversas operaciones geométricas para dibujar pentágonos y estrellas, utilizando transformaciones y rotaciones. Aquí desglosamos las matemáticas detrás de estas operaciones.

Transformaciones y Rotaciones

Transformaciones consisten en mover, escalar o rotar figuras geométricas en el plano. Rotaciones: La rotación de un punto (x, y) alrededor del origen por un ángulo θ se realiza con las fórmulas:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

Pentágonos

Un pentágono regular tiene 5 lados iguales y los ángulos internos son de 108 grados.

Radio del Círculo Circunscrito: El radio R de un círculo circunscrito alrededor de un pentágono con lado a se calcula como:

$$R = \frac{a}{2\sin(\frac{\pi}{5})}$$

Posición de los Vértices: Los vértices de un pentágono se distribuyen uniformemente en el círculo circunscrito. Los ángulos de los vértices con respecto al eje horizontal son

múltiplos de 72 grados (o $\frac{2\pi}{5}$ radianes). Para un vértice i, las coordenadas (x_i, y_i) son:

$$x_i = R\cos\left(2\pi\frac{i}{5} - \frac{\pi}{2}\right)$$

$$y_i = R\sin\left(2\pi\frac{i}{5} - \frac{\pi}{2}\right)$$

Donde el término $-\frac{\pi}{2}$ ajusta la rotación inicial para que un vértice esté en la parte superior.

Estrellas (Polígonos Estrellados)

Una estrella de 5 puntas se dibuja alternando entre dos radios: uno para los vértices exteriores y otro para los interiores.

Radios Externos R_o y Internos R_i:

$$R_o = \frac{a}{2\sin(\frac{\pi}{5})} \approx 0{,}85065 \cdot a$$

$$R_i = R_o\cos(\frac{\pi}{5}) \approx 0{,}52573 \cdot a$$

Posición de los Vértices: Las estrellas tienen 10 vértices (5 externos y 5 internos), y los ángulos entre ellos son de 36 grados (o $\frac{\pi}{5}$ radianes).

Para un vértice i, las coordenadas (x_i, y_i) alternan entre R_o y R_i:

$$x_i = \begin{cases} R_o\cos\left(\frac{i\pi}{5} - \frac{\pi}{2}\right) & \text{si } i \text{ es par} \\ R_i\cos\left(\frac{i\pi}{5} - \frac{\pi}{2}\right) & \text{si } i \text{ es impar} \end{cases}$$

$$y_i = \begin{cases} R_o\sin\left(\frac{i\pi}{5} - \frac{\pi}{2}\right) & \text{si } i \text{ es par} \\ R_i\sin\left(\frac{i\pi}{5} - \frac{\pi}{2}\right) & \text{si } i \text{ es impar} \end{cases}$$

Dibujar Estrellas Navideñas y Pentagramas

Estas estrellas se dibujan de manera similar a las estrellas regulares, pero con ajustes en los ángulos y radios para obtener las formas deseadas.

Estrella Navideña: Aumenta el radio interior y exterior y ajusta el ángulo inicial $\frac{\pi}{2}$.

$$x_i = R\cos\left(i\frac{\pi}{5} + \frac{\pi}{2} + \frac{235\pi}{180}\right)$$

$$y_i = R \sin\left(i\frac{\pi}{5} + \frac{\pi}{2} + \frac{235\pi}{180}\right)$$

Pentagrama: Usa un radio constante y un ángulo diferente para conectar los puntos de manera no consecutiva, formando un pentagrama.

$$x_i = R \cos\left(i\frac{4\pi}{5} - \frac{\pi}{2}\right)$$

$$y_i = R \sin\left(i\frac{4\pi}{5} - \frac{\pi}{2}\right)$$

Escalado y Rotación de Capas

Para dibujar múltiples capas de pentágonos y estrellas escalados y rotados:

Escalado: Se reduce el tamaño de cada capa sucesiva por un factor constante.

$$s_n = s_0 \cdot (\text{factor de escala})^n$$

Rotación: Cada capa se rota por un ángulo adicional respecto a la capa anterior.

$$\theta_n = \theta_0 + n \cdot (\text{ángulo de rotación})$$

Ejemplo de Cálculo para una Estrella: Supongamos una estrella de 5 puntas con un lado de longitud a:

1. Calcular los radios:

$$R_o = \frac{a}{2\sin(\frac{\pi}{5})} \approx 0{,}85065 \cdot a$$

$$R_i = R_o \cos(\frac{\pi}{5}) \approx 0{,}52573 \cdot a$$

2. Calcular las posiciones de los vértices:

$$\text{Vértice } 0 : x_0 = R_o \cos\left(-\frac{\pi}{2}\right), \quad y_0 = R_o \sin\left(-\frac{\pi}{2}\right)$$

$$\text{Vértice } 1 : x_1 = R_i \cos\left(-\frac{\pi}{2} + \frac{\pi}{5}\right), \quad y_1 = R_i \sin\left(-\frac{\pi}{2} + \frac{\pi}{5}\right)$$

$$\text{Vértice } 2 : x_2 = R_o \cos\left(-\frac{\pi}{2} + \frac{2\pi}{5}\right), \quad y_2 = R_o \sin\left(-\frac{\pi}{2} + \frac{2\pi}{5}\right)$$

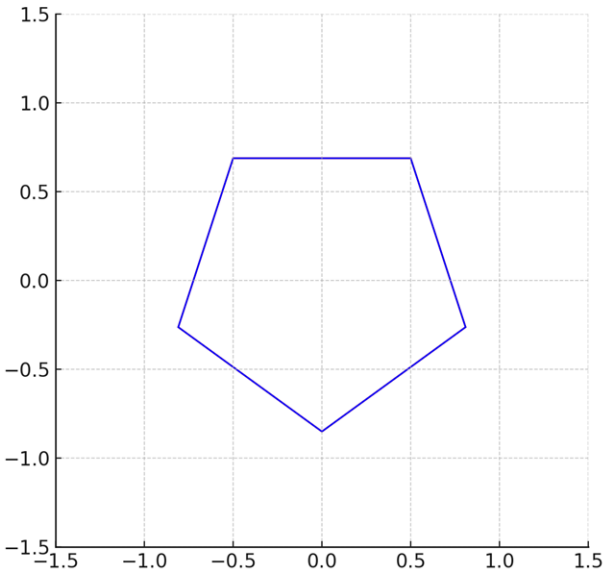

Figura 1: Pentágono

Desarrollo

El código del problema se muestra a continuación:

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace WindowsFormsApp2
{
    public partial class PoligonosEstrellados : Form
    {
        private int rectSize;
        private float angleRotatorio;
        private Point shapePosition; // Posicion de la figura
        private Color[] colors;
        private int colorIndex;

        public PoligonosEstrellados()
        {
            InitializeComponent();
            trackBar1.Maximum = 300;
```

```
            trackBar1.Value = 100;
            rectSize = trackBar1.Value;
            angleRotatorio = 0f;
            shapePosition = new Point(pictureBox1.Width / 2,
                pictureBox1.Height / 2);

            // Definir los colores a alternar
            colors = new Color[] { Color.Red, Color.Green, Color.Blue,
                Color.Orange, Color.Purple, Color.Cyan };
            colorIndex = 0;

            trackBar1.Scroll += trackBar1_Scroll;
            btnderecha.Click += btnderecha_Click;
            btnizquierda.Click += btnizquieda_Click;
            txtSideLength.TextChanged += txtSideLength_TextChanged; //
                Maneja el evento TextChanged del txtSideLength

            this.KeyDown += Form1_KeyDown; // Maneja el evento KeyDown
                del formulario
            this.KeyPreview = true; // Permite que el formulario
                capture eventos de teclado
        }

        private void PoligonosEstrellados_Load(object sender,
            EventArgs e)
        {
            // Cargar datos o inicializaciones adicionales aqu  si es
                necesario
        }

        private void btnDraw_Click(object sender, EventArgs e)
        {
            trackBar1.Value = 100;
            rectSize = trackBar1.Value;
            angleRotatorio = 0f;
            shapePosition = new Point(pictureBox1.Width / 2,
                pictureBox1.Height / 2);
            txtSideLength.Text = rectSize.ToString();

            // Redibujar el gr fico
            pictureBox1.Invalidate();
            txtSideLength.Focus();
        }

        private void pictureBox1_Paint(object sender, PaintEventArgs e
            )
        {
            Graphics g = e.Graphics;
            g.SmoothingMode = SmoothingMode.AntiAlias;
```

```
        if (!float.TryParse(txtSideLength.Text, out float
            sideLength))
        {
            // Manejar el error si el texto no se puede convertir
                en un flotante
            sideLength = rectSize; // Longitud por defecto si
                falla la conversi n
        }

        DrawNestedPolygons(g, sideLength);
        DrawPentagramStar(g, sideLength * 0.27f, shapePosition.X,
            shapePosition.Y, angleRotatorio); // Mantener la
            estrella central sin cambios
        DrawChristmasStar(g, sideLength * 0.27f, shapePosition.X,
            shapePosition.Y, angleRotatorio); // Dibujar estrella
            navide a
        DrawChristmasStar2(g, sideLength * 0.27f, shapePosition.X,
             shapePosition.Y, angleRotatorio); // Dibujar estrella
             navide a
        DrawNormalStar(g, sideLength * 0.27f, shapePosition.X,
            shapePosition.Y, angleRotatorio); // Dibujar estrella
            normal
    }

    private void DrawNestedPolygons(Graphics g, float sideLength)
    {
        int numLayers = 2; // N mero de capas
        float scaleFactor = 0.65f; // Factor de escala aumentado
            para mayor separaci n
        float rotationOffset = 36; //  ngulo  de rotaci n
            aumentado para mayor separaci n
        float initialRotation = angleRotatorio; // Guardar el
             ngulo  inicial para cada capa

        // Dibujar pent gonos y estrellas
        for (int layer = 0; layer < numLayers; layer++)
        {
            float currentScale = (float)Math.Pow(scaleFactor,
                layer);

            // Dibujar estrella
            g.TranslateTransform(shapePosition.X, shapePosition.Y)
                ;
            g.RotateTransform(initialRotation + layer *
                rotationOffset);
            g.TranslateTransform(-shapePosition.X, -shapePosition.
                Y);
            DrawStarPolygon(g, sideLength * currentScale, 5,
```

```
                shapePosition.X, shapePosition.Y);
            g.ResetTransform();

            // Dibujar pent gono
            g.TranslateTransform(shapePosition.X, shapePosition.Y)
                ;
            g.RotateTransform(initialRotation + (layer + 0.5f) *
                rotationOffset);
            g.TranslateTransform(-shapePosition.X, -shapePosition.
                Y);
            DrawPentagon(g, sideLength * currentScale, 5,
                shapePosition.X, shapePosition.Y);
            g.ResetTransform();
        }

        // Dibujar pent gono adicional del tama o de la segunda
            capa
        float secondLayerScale = (float)Math.Pow(scaleFactor, 1);
        DrawRegularPentagon(g, 1.2f * sideLength *
            secondLayerScale, shapePosition.X, shapePosition.Y,
            angleRotatorio);
    }

    private void DrawPentagon(Graphics g, float sideLength, int
        numPoints, float centerX, float centerY)
    {
        PointF[] points = new PointF[numPoints];
        float angleIncrement = (float)(2 * Math.PI / numPoints);
        float radius = sideLength / (2 * (float)Math.Sin(Math.PI /
            numPoints));

        for (int i = 0; i < numPoints; i++)
        {
            float angle = i * angleIncrement - (float)Math.PI / 2;
            points[i] = new PointF(
                centerX + radius * (float)Math.Cos(angle),
                centerY + radius * (float)Math.Sin(angle)
            );
        }

        using (Pen pen = new Pen(GetNextColor(), 2))
        {
            g.DrawPolygon(pen, points);
        }
    }

    private void DrawRegularPentagon(Graphics g, float sideLength,
        float centerX, float centerY, float angleRotatorio)
    {
```

```
            int numPoints = 5;
            PointF[] points = new PointF[numPoints];
            float angleIncrement = (float)(2 * Math.PI / numPoints);
            float radius = sideLength / (2 * (float)Math.Sin(Math.PI /
                numPoints));

            for (int i = 0; i < numPoints; i++)
            {
                float angle = i * angleIncrement - (float)Math.PI / 2;
                points[i] = new PointF(
                    centerX + radius * (float)Math.Cos(angle),
                    centerY + radius * (float)Math.Sin(angle)
                );
            }

            g.TranslateTransform(centerX, centerY);
            g.RotateTransform(angleRotatorio);
            g.TranslateTransform(-centerX, -centerY);

            using (Pen pen = new Pen(GetNextColor(), 2))
            {
                g.DrawPolygon(pen, points);
            }

            g.ResetTransform();
        }

        private void DrawStarPolygon(Graphics g, float sideLength, int
            numPoints, float centerX, float centerY)
        {
            int rotationAngle = 160;
            PointF[] points = new PointF[numPoints * 2];
            float angleIncrement = (float)(Math.PI / numPoints);
            float radiusOuter = sideLength / (2 * (float)Math.Sin(Math
                .PI / numPoints));
            float radiusInner = radiusOuter * (float)Math.Cos(
               angleIncrement);

            for (int i = 0; i < numPoints * 2; i++)
            {
                float radius = (i % 2 == 0) ? radiusOuter :
                   radiusInner;
                float angle = i * angleIncrement - (float)Math.PI / 2
                   - rotationAngle;
                points[i] = new PointF(
                    centerX + (radius) * (float)Math.Cos(angle),
                    centerY + (radius) * (float)Math.Sin(angle)
                );
            }
```

```
        using (Pen pen = new Pen(GetNextColor(), 2))
        {
            g.DrawPolygon(pen, points);
        }
    }

    private void DrawPentagramStar(Graphics g, float sideLength,
        float centerX, float centerY, float angleRotatorio)
    {
        int angleRotation = 0;
        int numPoints = 5; // 5 puntos para la estrella
        PointF[] points = new PointF[numPoints * 2];
        float angleIncrement = (float)(4 * Math.PI / numPoints);
        float radius = sideLength;

        for (int i = 0; i < numPoints * 2; i++)
        {
            float angle = i * angleIncrement - (float)Math.PI / 2
                - angleRotation;
            points[i] = new PointF(
                centerX + radius * (float)Math.Cos(angle),
                centerY + radius * (float)Math.Sin(angle)
            );
        }

        g.TranslateTransform(centerX, centerY);
        g.RotateTransform(angleRotatorio);
        g.TranslateTransform(-centerX, -centerY);

        using (Pen pen = new Pen(GetNextColor(), 2))
        {
            g.DrawPolygon(pen, points);
        }

        g.ResetTransform();
    }

    private void DrawChristmasStar(Graphics g, float sideLength,
        float centerX, float centerY, float angleRotatorio)
    {
        int numPoints = 5;
        PointF[] points = new PointF[numPoints * 2];
        float outerRadius = 2f * sideLength;
        float innerRadius = outerRadius * 0.5f;
        float angleIncrement = (float)(Math.PI / numPoints);

        for (int i = 0; i < numPoints * 2; i++)
        {
```

```
            float radius = (i % 2 == 0) ? outerRadius :
                innerRadius;
            float angle = i * angleIncrement + (float)Math.PI / 2;
                // Punta hacia abajo
            points[i] = new PointF(
                centerX + radius * (float)Math.Cos(angle),
                centerY + radius * (float)Math.Sin(angle)
            );
        }

        g.TranslateTransform(centerX, centerY);
        g.RotateTransform(angleRotatorio);
        g.TranslateTransform(-centerX, -centerY);

        using (Pen pen = new Pen(GetNextColor(), 2))
        {
            g.DrawPolygon(pen, points);
        }

        g.ResetTransform();
    }

    private void DrawChristmasStar2(Graphics g, float sideLength,
        float centerX, float centerY, float angleRotatorio)
    {
        int rotationAngle = 235;
        int numPoints = 5;
        PointF[] points = new PointF[numPoints * 2];
        float outerRadius = 2f * sideLength;
        float innerRadius = outerRadius * 0.7f; // Ajustar el
            tama o de los puntos internos para una estrella m s
            delgada
        float angleIncrement = (float)(Math.PI / numPoints);

        for (int i = 0; i < numPoints * 2; i++)
        {
            float radius = (i % 2 == 0) ? outerRadius :
                innerRadius;
            float angle = i * angleIncrement + rotationAngle + (
                float)Math.PI / 2;
            points[i] = new PointF(
                centerX + radius * (float)Math.Cos(angle),
                centerY + radius * (float)Math.Sin(angle)
            );
        }

        g.TranslateTransform(centerX, centerY);
        g.RotateTransform(angleRotatorio);
        g.TranslateTransform(-centerX, -centerY);
```

```
        using (Pen pen = new Pen(GetNextColor(), 2))
        {
            g.DrawPolygon(pen, points);
        }

        g.ResetTransform();
    }

    private void DrawNormalStar(Graphics g, float sideLength,
       float centerX, float centerY, float angleRotatorio)
    {
        int rotationAngle = 180;
        PointF[] points = new PointF[10];
        double angle = -Math.PI / 2 + rotationAngle * Math.PI /
           180; // Convertir el angulo de rotaci n a radianes
        double angleIncrement = Math.PI / 5;
        float radius = 2f * sideLength;

        for (int i = 0; i < 10; i++)
        {
            float length = (i % 2 == 0) ? radius : radius / 2f;
            points[i] = new PointF(
                centerX + (float)(length * Math.Cos(angle)),
                centerY + (float)(length * Math.Sin(angle))
            );
            angle += angleIncrement;
        }

        g.TranslateTransform(centerX, centerY);
        g.RotateTransform(angleRotatorio);
        g.TranslateTransform(-centerX, -centerY);

        using (Pen pen = new Pen(GetNextColor(), 2))
        {
            g.DrawPolygon(pen, points);
        }

        g.ResetTransform();
    }

    private void trackBar1_Scroll(object sender, EventArgs e)
    {
        rectSize = trackBar1.Value;
        txtSideLength.Text = rectSize.ToString(); // Actualizar el
            valor en txtSideLength
        pictureBox1.Invalidate(); // Redibujar la figura con el
           nuevo tama o
    }
```

```
        private void btnizquieda_Click(object sender, EventArgs e)
        {
            angleRotatorio -= 10f; // Rota 10 grados a la izquierda
            pictureBox1.Invalidate(); // Redibujar la figura con la
                nueva rotacion
            txtSideLength.Focus();
        }

        private void btnderecha_Click(object sender, EventArgs e)
        {
            angleRotatorio += 10f; // Rota 10 grados a la derecha
            pictureBox1.Invalidate(); // Redibujar la figura con la
                nueva rotacion
            txtSideLength.Focus();
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            int moveAmount = 10; // Cantidad de movimiento en pixeles

            switch (e.KeyCode)
            {
                case Keys.Up:
                    shapePosition.Y -= moveAmount;
                    break;
                case Keys.Down:
                    shapePosition.Y += moveAmount;
                    break;
                case Keys.Left:
                    shapePosition.X -= moveAmount;
                    break;
                case Keys.Right:
                    shapePosition.X += moveAmount;
                    break;
            }

            pictureBox1.Invalidate(); // Fuerza el redibujado del
                PictureBox
        }

        private void txtSideLength_TextChanged(object sender,
            EventArgs e)
        {
            if (int.TryParse(txtSideLength.Text, out int newSize) &&
                newSize >= trackBar1.Minimum && newSize <= trackBar1.
                Maximum)
            {
                rectSize = newSize;
```

```
                trackBar1.Value = rectSize; // Actualizar el valor del
                    trackBar
                pictureBox1.Invalidate(); // Redibujar la figura con
                   el nuevo tama o
            }
        }

        private Color GetNextColor()
        {
            Color nextColor = colors[colorIndex];
            colorIndex = (colorIndex + 1) % colors.Length;
            return nextColor;
        }
    }
}
```

Este código en C# define una aplicación de Windows Forms llamada PoligonosEstrellados que permite dibujar varias figuras geométricas, incluyendo polígonos estrellados, pentágonos, y diferentes tipos de estrellas. A continuación se explica el funcionamiento del código en detalle.

El programa comienza con las directivas using necesarias para importar los espacios de nombres que contienen las clases utilizadas en el código. Estos incluyen System, System.Drawing, System.Drawing.Drawing2D, y System.Windows.Forms.

La clase principal PoligonosEstrellados hereda de Form, lo que la convierte en un formulario de Windows. Dentro de esta clase, se declaran varias variables de instancia, incluyendo rectSize (tamaño del rectángulo), angleRotatorio (ángulo de rotación), shapePosition (posición de la figura), colors (array de colores), y colorIndex (índice del color actual).

En el constructor PoligonosEstrellados, se inicializan los componentes del formulario y se establecen los valores iniciales de las variables. Además, se definen los colores a utilizar y se configuran los eventos para los controles del formulario, como el trackBar1 para ajustar el tamaño del rectángulo y los botones para rotar la figura.

El método pictureBox1_Paint es el encargado de dibujar las figuras en el PictureBox. Utiliza el método Graphics para dibujar las figuras con suavizado (anti-aliasing) y llama a varios métodos para dibujar polígonos y estrellas anidadas. Estos métodos incluyen DrawNestedPolygons, DrawPentagon, DrawRegularPentagon, DrawStarPolygon, DrawPentagramStar, DrawChristmasStar, DrawChristmasStar2, y DrawNormalStar. Cada uno de estos métodos dibuja una figura geométrica específica utilizando transformaciones y colores alternos.

El método trackBar1_Scroll maneja el evento de desplazamiento del trackBar1, actuali-

zando el tamaño del rectángulo y redibujando la figura. Los métodos btnizquieda_Click y btnderecha_Click manejan los eventos de clic de los botones para rotar la figura a la izquierda y a la derecha, respectivamente.

El método Form1_KeyDown maneja los eventos de teclado para mover la figura en el PictureBox en respuesta a las teclas de flecha. El método txtSideLength_TextChanged actualiza el tamaño del rectángulo en función del texto ingresado en txtSideLength.

Finalmente, el método GetNextColor devuelve el siguiente color del array de colores, alternando entre los colores definidos.

Resultados

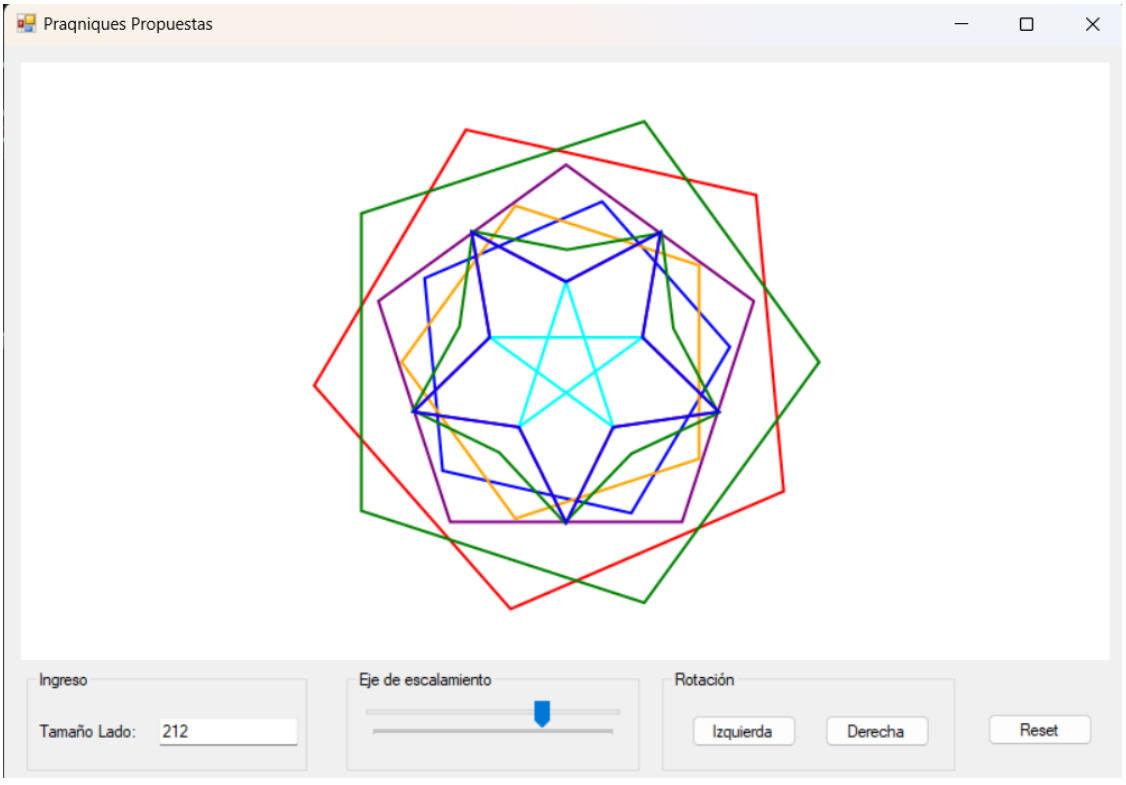

Figura 2: Caption

Conclusiones

- Se logró desarrollar un programa en C# utilizando .NET Framework que dibuja correctamente un pentágono y diferentes polígonos estrellados de 5 puntas, cumpliendo con los objetivos establecidos.

- El uso de C# y .NET Framework resultó ser adecuado para este tipo de problemas gráficos, mostrando la potencia y versatilidad de estas herramientas en el desarrollo de aplicaciones de dibujo geométrico.
- El conocimiento y la experiencia adquirida en este proyecto pueden aplicarse en futuras implementaciones de gráficos geométricos más avanzados y en otros proyectos relacionados con visualización y animación en C#.