Universidad de Las fuerzas Armadas - ESPE

Departamento de Ciencias de la Computación (DCCO)

Carrera de Ingeniería de Software

Computación Gráfica - NRC23463

Tema: Desarrollo de una “Simulación del Reproductor de Windows Media”

Grupo Número:

Miembros:
García Mateo
Parra Sebastián
Sagnay Luis
Yepez Jefferson

Docente:
Ing. Morales Caiza Dario Javier

Fecha:
2025-05-23

Link del repositorio:

https://github.com/Cotbert2/WindowsMediaPlayerVisualizer.git

Link de presentación:

https://www.canva.com/design/DAGoVfD9uFo/O9MvUJ2mqqsium9UJlbb4Q/edit?utm_content=DAGoVfD9uFo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Introducción

El siguiente proyecto consiste en el desarrollo de un reproductor de música con efectos visuales dinámicos sincronizados con la música que se reproduce. Se utilizará tecnologías como C#, WinForms, la biblioteca de audio NAudio y componentes personalizados para representar diferentes tipos de visualizaciones, inspiradas en reproductores clásicos como Windows Media Player.

Objetivo General

Desarrollar una aplicación de escritorio en C# que funcione como un reproductor de audio capaz de mostrar visualizaciones dinámicas e interactivas en tiempo real, implementadas con figuras geométricas básicas y basadas en el análisis del volumen y frecuencia del audio.

Objetivos Específicos

- Crear un reproductor de audio con controles básicos como por ejemplo reproducir, pausar, adelantar y retroceder.
- Implementar diferentes tipos de visualizadores interactivos que reaccionan a la frecuencia y la amplitud de la música.
- Permitir la rotación automática entre diferentes visualizadores con base en un temporizador, enriqueciendo la experiencia del usuario.

Tecnologías utilizadas

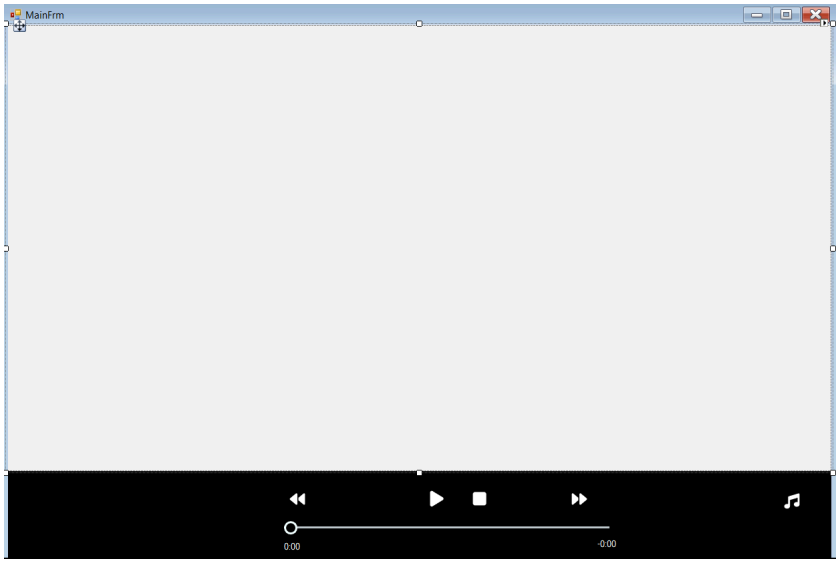
- **Lenguaje:** C# (.NET)
- **Interfaz gráfica:** WinForms
- **Audio:** NAudio
- **Visuales:** GDI+(Graphics, Drawing2D)
- **Librerías adicionales:**
    - FontAwesome.Sharp
    - Guna.UI2.WinForms (para dar estética a los controles)

Estructura del proyecto

1. Views

Interfaz de usuario principal:

- **MainFrm.cs :** Ventana principal donde se mostrará al usuario los eventos de control del reproductor y los visualizadores.



2. Controllers

- **Player.cs :** Lógica donde está el reproductor multimedia, el cargar la canción, control de reproducción, visualizadores.

```
public class Player
{
    private IWavePlayer waveOut;
    private AudioFileReader audioFileReader;
    private ISampleProvider sampleProvider;

    private bool isPlaying = false;
    private Label lblArtist;
    private Label lblSong;
    private Guna2TrackBar barPlayer;
    private IconButton btnPlay;
    private Timer timer1;
    private Label lblCounter;
    private Label lblCountdown;
    private Panel canvas;

    //Visualizers
    private Visualizer1 visualizer1;
    private Visualizer2 visualizer2;
    private Visualizer3 visualizer3;
    private Visualizer4 visualizer4;
    private Visualizer5 visualizer5;
    private Visualizer6 visualizer6;


    private int currentVisualizerIndex = 0;
    private double visualizerTimerSeconds = 0;
    private const double visualizerDuration = 30;

    1 referencia
    public Player(Label lblArtist, Label lblSong, Guna2TrackBar barPlayer, IconButton btnPlay,
        Timer timer1, Label lblCounter, Label lblCountdown, Panel canvas)
    {
        this.lblArtist = lblArtist;
        this.lblSong = lblSong;
        this.barPlayer = barPlayer;
        this.btnPlay = btnPlay;
        this.timer1 = timer1;
        this.lblCounter = lblCounter;
        this.lblCountdown = lblCountdown;
        this.canvas = canvas;

        //setup visualizers

        visualizer1 = new Visualizer1(canvas);
        visualizer2 = new Visualizer2(canvas);
        visualizer3 = new Visualizer3(canvas);
        visualizer4 = new Visualizer4(canvas);
        visualizer5 = new Visualizer5(canvas);
        visualizer6 = new Visualizer6(canvas);

        currentVisualizer = visualizer1;
        canvas.Paint += visualizer1.Canvas_Paint;
    }
```

En la siguiente imagen se muestra la inicialización del reproductor con los controles gráficos y crea las instancias de los visualizadores. Establece el **Visualizer1** como visualizador inicial y conecta el método **Paint** al lienzo.

```
private void setSong(string path)
{
    var file = TagLib.File.Create(path);

    lblArtist.Text = file.Tag.FirstPerformer ?? "Unknown Artist";
    lblSong.Text = file.Tag.Title ?? "Unknown Song";

    audioFileReader = new AudioFileReader(path);
    var sampleChannel = new SampleChannel(audioFileReader, true);
    var meteringProvider = new MeteringSampleProvider(sampleChannel);

    meteringProvider.StreamVolume += (s, a) =>
    {
        currentVolume = a.MaxSampleValues.Average();
    };

    fftSampleProvider = new FFTSampleProvider(meteringProvider, 64);
    sampleProvider = fftSampleProvider;

    waveOut = new WaveOutEvent();
    waveOut.Init(sampleProvider);
    barPlayer.Value = 0;
    isPlaying = false;
    btnPlay.IconChar = IconChar.Play;
}


1 referencia
public void play()
{
    if (audioFileReader == null || waveOut == null)
    {
        MessageBox.Show("No audio file loaded.");
        return;
    }

    if (!isPlaying)
    {
        waveOut.Play();
        isPlaying = true;
        btnPlay.IconChar = IconChar.Pause;
        timer1.Start();
        return;
    }

    waveOut.Pause();
    isPlaying = false;
    btnPlay.IconChar = IconChar.Play;
    timer1.Stop();
}
```

En la siguiente imagen se muestran las funciones **setSong(string path)** la cual carga un archivo de audio desde la ruta especificada, se extraen el artista y el título de la canción usando **TagLib,** configura el flujo del audio y annalisa la frecuencia, dejando todo listo para reproducir

También está la función **play()** la cual controla la reproducción del audio, si no está sonando, inicia la reproducción y cambia el ícono del botón a “Pause”.

```
2 referencias
public void stop()
{
    if (waveOut == null || audioFileReader == null) return;

    waveOut.Stop();
    audioFileReader.Position = 0;
    btnPlay.IconChar = IconChar.Play;
    isPlaying = false;

}

1 referencia
public void backward() {
    if (audioFileReader == null) return;

    var newTime = audioFileReader.CurrentTime - TimeSpan.FromSeconds(5);

    if (newTime < TimeSpan.Zero)
        newTime = TimeSpan.Zero;

    audioFileReader.CurrentTime = newTime;
}


1 referencia
public void forward()
{
    if (audioFileReader == null) return;

    var newTime = audioFileReader.CurrentTime + TimeSpan.FromSeconds(5);

    if (newTime < TimeSpan.Zero)
        newTime = TimeSpan.Zero;

    audioFileReader.CurrentTime = newTime;
}
1 referencia
public void hanldePosition() {
    if (audioFileReader != null)
    {
        double total = audioFileReader.TotalTime.TotalSeconds;
        double newPosition = (barPlayer.Value / 100.0) * total;
        audioFileReader.CurrentTime = TimeSpan.FromSeconds(newPosition);
    }
}

float currentVolume = 0f;
float volumeSmoothing = 0.2f;
private FFTSampleProvider fftSampleProvider;
```

La función **stop()** detiene la reproducción del audio y reinicia su posición al inicio.

La función **backward()** retrocede 5 segundos en la reproducción actual y la función **fordward()** hace lo contrario, que es avanzar 5 segundos en la canción.

La función **handlePosition()** actualiza la posición del archivo de audio según el valor actual del **TrackBar.**

```
public void tickAction() {
    tickCounter++;
    if (tickCounter % 120 == 0)
    {
        SwitchVisualizer();
    }

    if (currentVisualizer is Visualizer1 v1 && fftSampleProvider != null)
        v1.Update(fftSampleProvider.GetFrequencies());
    else if (currentVisualizer is Visualizer2 v2)
        v2.Update(currentVolume, volumeSmoothing);
    else if (currentVisualizer is Visualizer3 v3)
        v3.Update();
    else if (currentVisualizer is Visualizer4 v4)
        v4.Update();
    else if (currentVisualizer is Visualizer5 v5)
        v5.Update(fftSampleProvider.GetFrequencies());
    else if (currentVisualizer is Visualizer6 v6)
        v6.Update(fftSampleProvider.GetFrequencies());
    else if (currentVisualizer is Visualizer11 v11)
        v11.Update(fftSampleProvider.GetFrequencies(), currentVolume);
    if (audioFileReader != null && audioFileReader.TotalTime.TotalSeconds > 0)
    {
        double current = audioFileReader.CurrentTime.TotalSeconds;
        double total = audioFileReader.TotalTime.TotalSeconds;
        double remaining = total - current;


        int progress = (int)((current / total) * 100);
        barPlayer.Value = Math.Min(progress, 100);

        TimeSpan currentTime = TimeSpan.FromSeconds(current);
        TimeSpan remainingTime = TimeSpan.FromSeconds(remaining);

        lblCounter.Text = $"{currentTime.Minutes}:{zeroFormat(currentTime.Seconds)}";
        lblCountdown.Text = $"-{remainingTime.Minutes}:{zeroFormat(remainingTime.Seconds)}";

        if (audioFileReader.CurrentTime.TotalSeconds > audioFileReader.TotalTime.TotalSeconds) stop();


    }
}

2 referencias
private string zeroFormat(int time)
{
    return (time <= 9) ? $"0{time}" : $"{time}";
}
```

En la siguiente imagen se muestra la función **tickAction(),** ésta se ejecuta cada vez que el **timer1** hace un tick, por ejemplo cada segundo, aquí el visualizador cambia cada 120 ticks, llama al método **Update()** del visualizador activo, pasando el volumen o frecuencias según corresponda, actualiza la barra de progreso del reproductor y las etiquetas de tiempo transcurrido y restante.

En la función **zeroFormat(int time)** formatea un número entero como texto para mostrarlo con dos dígitos.

```
public void selectFile() {
    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.Filter = "Audio File|*.mp3;*.wav;*.wma";
        openFileDialog.Title = "Chose an audio file";

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            string path = openFileDialog.FileName;

            try
            {
                waveOut?.Stop();
                audioFileReader?.Dispose();
                waveOut?.Dispose();
                setSong(path);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al cargar el archivo: " + ex.Message);
            }
        }
    }
}

private int tickCounter = 0;
private object currentVisualizer;

1 referencia
private void DetachCurrentVisualizer()
{
    if (currentVisualizer is Visualizer1 v1) canvas.Paint -= v1.Canvas_Paint;
    else if (currentVisualizer is Visualizer2 v2) canvas.Paint -= v2.Canvas_Paint;
    else if (currentVisualizer is Visualizer3 v3) canvas.Paint -= v3.Canvas_Paint;
    else if (currentVisualizer is Visualizer4 v4) canvas.Paint -= v4.Canvas_Paint;
    else if (currentVisualizer is Visualizer5 v5) canvas.Paint -= v5.Canvas_Paint;
    else if (currentVisualizer is Visualizer6 v6) canvas.Paint -= v6.Canvas_Paint;
    else if (currentVisualizer is Visualizer11 v11) canvas.Paint -= v11.Canvas_Paint;
}
```

Aquí tenemos dos funciones, la de **selectFile()** que hace que el usuario seleccione un archivo de audio y la de **DetachCurrentVisualizer()** que hace que se evite que varios visualizadores se dibujen a la vez.

```
private void SwitchVisualizer()
{
    DetachCurrentVisualizer();

    currentVisualizerIndex = (currentVisualizerIndex + 1) % 7;

    switch (currentVisualizerIndex)
    {
        case 0:
            currentVisualizer = visualizer1;
            break;
        case 1:
            currentVisualizer = visualizer2;
            break;
        case 2:
            currentVisualizer = visualizer3;
            break;
        case 3:
            currentVisualizer = visualizer4;
            break;
        case 4:
            currentVisualizer = visualizer5;
            break;
        case 5:
            currentVisualizer = visualizer6;
            break;
        case 6:
            currentVisualizer = visualizer11;
            break;
    }

    if (currentVisualizer is Visualizer1 v1) canvas.Paint += v1.Canvas_Paint;
    else if (currentVisualizer is Visualizer2 v2) canvas.Paint += v2.Canvas_Paint;
    else if (currentVisualizer is Visualizer3 v3) canvas.Paint += v3.Canvas_Paint;
    else if (currentVisualizer is Visualizer4 v4) canvas.Paint += v4.Canvas_Paint;
    else if (currentVisualizer is Visualizer5 v5) canvas.Paint += v5.Canvas_Paint;
    else if (currentVisualizer is Visualizer6 v6) canvas.Paint += v6.Canvas_Paint;
    else if (currentVisualizer is Visualizer11 v11) canvas.Paint += v11.Canvas_Paint;

    canvas.Invalidate();
}
```

Aquí, en la función de **SwitchVisualizer()** hace que cambie al siguiente visualizador en la lista circularmente y conecta su método **Paint** al lienzo, también hace que se redibuje con el nuevo visualizador.

- **FFTSampleProvider.cs** : Aquí es donde se da la captura de las frecuencias en tiempo real mediante FFT.

```
public int Read(float[] buffer, int offset, int count)
{
    int samplesRead = source.Read(buffer, offset, count);
    lock (lockObj)
    {
        for (int i = 0; i < samplesRead; i += channels)
        {
            float sample = buffer[offset + i];

            fftBuffer[bufferPos].X = (float)(sample * FastFourierTransform.HammingWindow(bufferPos, fftLength));
            fftBuffer[bufferPos].Y = 0;
            bufferPos++;

            if (bufferPos >= fftLength)
            {
                FastFourierTransform.FFT(true, (int)Math.Log(fftLength, 2.0), fftBuffer);

                for (int j = 0; j < result.Length; j++)
                {
                    result[j] = (float)Math.Sqrt(fftBuffer[j].X * fftBuffer[j].X + fftBuffer[j].Y * fftBuffer[j].Y);
                }

                bufferPos = 0;
            }
        }
    }

    return samplesRead;
}
```

Este método **Read,** lo que hace es leer muestras de audio, aplica una ventana de Hamming a cada muestra, se ejecuta la **FFT** usando **FastFourierTransform.FFT(...)** y calcula magnitud de cada frecuencia resultante y las guarda en **result[]**

- **VisualizerCanvas.cs :** Panel con doble búfer para renderizado suave.
- **Visualizer1 al Visualizer6:** Las diferentes clases en donde se implementan los diferentes estilos visuales.

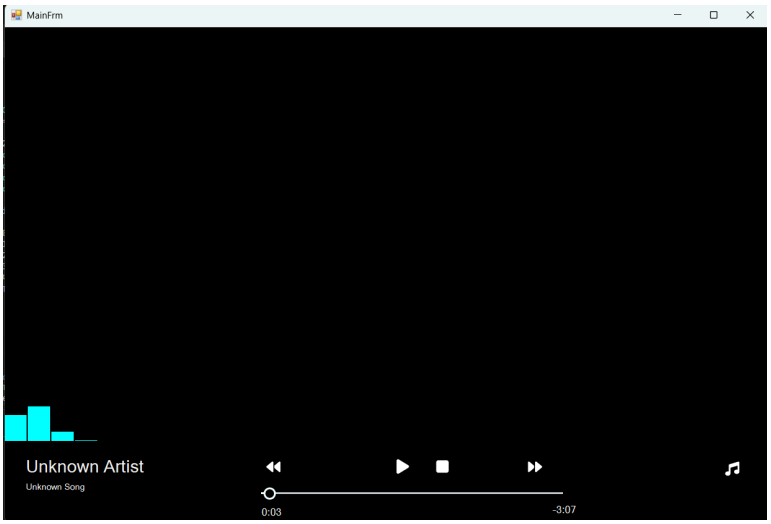
3. Components
Contiene componentes visuales reutilizables:

- VisualizerCanvas.cs: Panel personalizado con double buffering para renderizado suave

4. Utils
Utilidades matemáticas y de procesamiento:

- FFTSampleProvider.cs: Implementación de FFT para análisis de frecuencias
- Operations.cs: Operaciones matemáticas para la rotación
- PolygonBuilder.cs: Generación y manipulación de formas geométricas
- VisualizerHelpers.cs: Funciones auxiliares para visualizaciones

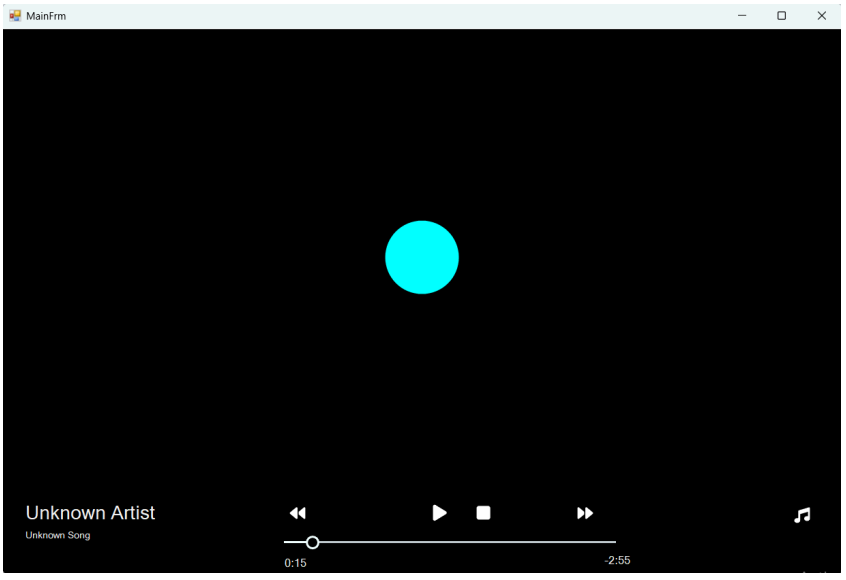
Descripción de los visualizadores
Visualizer1:



```
public class Visualizer1
{
    private Panel canvas;

    1 referencia
    public Visualizer1(Panel canvas)
    {
        this.canvas = canvas;
        this.canvas.Paint += Canvas_Paint;
    }

    4 referencias
    public void Canvas_Paint(object sender, PaintEventArgs e)
    {

    }

    1 referencia
    public void Update(float[] frequencyAmplitudes)
    {
        Graphics g = canvas.CreateGraphics();
        g.Clear(Color.Black);

        int numBars = frequencyAmplitudes.Length;
        int barWidth = canvas.Width / numBars;

        for (int i = 0; i < numBars; i++)
        {
            float amplitude = frequencyAmplitudes[i];

            amplitude = Math.Min(amplitude * 5f, 1f);

            int height = (int)(amplitude * canvas.Height);
            Rectangle bar = new Rectangle(i * barWidth, canvas.Height - height, barWidth - 2, height);
            g.FillRectangle(Brushes.Cyan, bar);
        }
    }


}
```

Se puede observar que en la función **Update(...)** recibe un arreglo de amplitudes de frecuencia. Bora el canvas y dibuja barras verticales proporcionalmente a cada amplitud. Cada barra representa una banda de frecuencia.

Visualizer2:



```
public class Visualizer2
{
    private Panel canvas;
    private float currentVolume;
    private float volumeSmoothing;
    private int flashAlpha = 0;
    private const float volumeThreshold = 0.9f;
    private Timer fadeTimer;
    private float currentAngle = 0f;


    1 referencia
    public Visualizer2(Panel canvas)
    {
        this.canvas = canvas;
        this.canvas.Paint += Canvas_Paint;

        fadeTimer = new Timer();
        fadeTimer.Interval = 50;
        fadeTimer.Tick += (s, e) =>
        {
            if (flashAlpha > 0)
            {
                flashAlpha -= 15;
                if (flashAlpha < 0) flashAlpha = 0;
                canvas.Invalidate();
            }
        };
        fadeTimer.Start();


        points[0] = new PointF(0, 0);
        points[1] = new PointF(canvas.Width, canvas.Height);
        points[2] = new PointF(canvas.Width, 0);
        points[3] = new PointF(0, canvas.Height);
    }
```

En el **Visualizer2** primero se configura el panel, activa el temporizador de desvanecimiento del flash y establece coordenadas para rayos zigzag. En la función **Update(...)** actualiza el volumen y, si supera el umbral, activa un destello blanco y rayos zigzag.

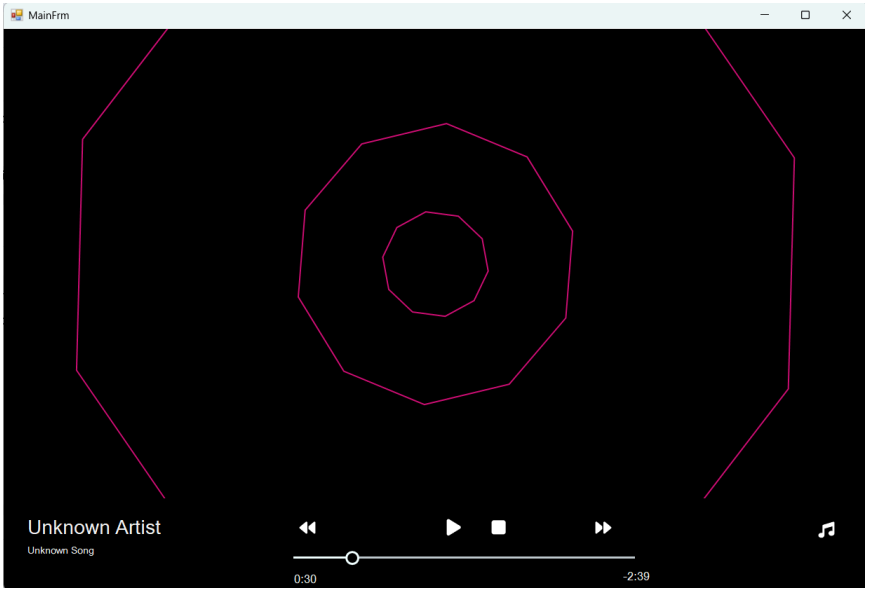
```
public void Canvas_Paint(object sender, PaintEventArgs e)
{
    Draw(e.Graphics);
}

PointF[] points = new PointF[4];

1 referencia
private void Draw(Graphics graphics)
{
    graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
    graphics.Clear(Color.Black);

    float centerX = canvas.Width / 2f;
    float centerY = canvas.Height / 2f;

    int minRadius = 30;
    int maxRadius = 160;
    int radius = minRadius + (int)((maxRadius - minRadius) * currentVolume);

    Rectangle circle = new Rectangle(
        (int)(centerX - radius),
        (int)(centerY - radius),
        radius * 2,
        radius * 2
    );

    if (flashAlpha > 0)
    {
        float len = Math.Max(canvas.Width, canvas.Height);
        Color flashColor = Color.FromArgb(flashAlpha, Color.White);
        using (Pen pen = new Pen(flashColor, 2))
        {
            for (int i = 0; i < 8; i++)
            {
                float angle = currentAngle + i * 45; // 8 rayos cada 45°
                DrawZigZagRay(graphics, pen, centerX, centerY, len, angle, 10);
            }
        }
    }
```

Para la función **Draw(...)** dibuja el círculo central pulsante y si el volumen es alto, también dibuja 8 rayos en zigzag giratorios.

```
private void DrawZigZagRay(Graphics g, Pen pen, float cx, float cy, float length, float angleDegrees, int segments)
{
    float angleRadians = angleDegrees * (float)(Math.PI / 180);
    float dx = (float)Math.Cos(angleRadians);
    float dy = (float)Math.Sin(angleRadians);

    float segmentLength = length / segments;
    float zigzagOffset = 10f;

    PointF[] points = new PointF[segments + 1];
    for (int i = 0; i <= segments; i++)
    {
        float x = cx + i * segmentLength * dx;
        float y = cy + i * segmentLength * dy;

        if (i % 2 == 1)
        {
            float px = -dy * zigzagOffset;
            float py = dx * zigzagOffset;
            x += px;
            y += py;
        }

        points[i] = new PointF(x, y);
    }

    g.DrawLines(pen, points);
}
```

En la función **DrawZigZagRay(...)** dibuja una linea en zigzag desde el centro hacia afuera en una dirección específica.

Visualizer3:



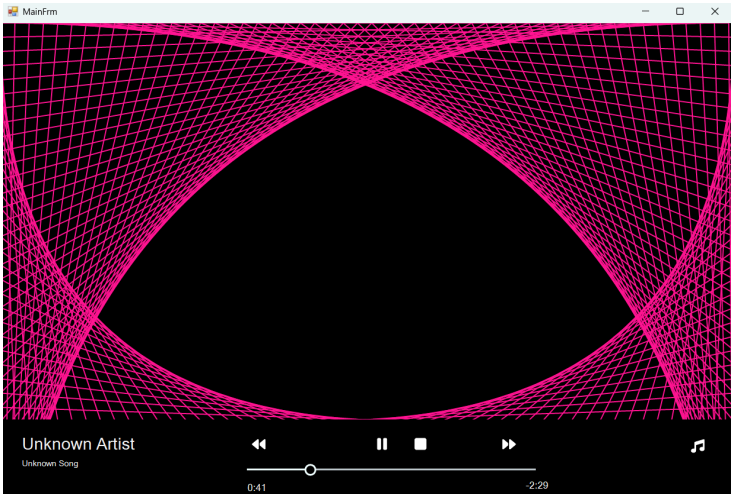

```
public class Visualizer3
{
    private Panel canvas;
    private List<PointF[]> polygons;
    private float alphaChannel = 255;
    private float scaleFactor = 1.05f;
    private int frameCount = 0;

    1 referencia
    public Visualizer3(Panel canvas)
    {
        this.canvas = canvas;
        this.canvas.Paint += Canvas_Paint;
        this.polygons = new List<PointF[]>();

        // Primer decágono en el centro
        polygons.Add(
            PolygonBuilder.centerPolygon(
                PolygonBuilder.buildPolygon(10, 30),
                canvas
            ).ToArray()
        );
    }

    1 referencia
    public void Update()
    {
        frameCount++;

        if (frameCount % 10 == 0)
        {
            polygons.Add(
                PolygonBuilder.centerPolygon(
                    PolygonBuilder.buildPolygon(10, 30),
                    canvas
                ).ToArray()
            );
        }

        canvas.Invalidate();
    }
```

En este visualizer se inicializa con un decágono centrado y en la función **Update()** cada 10 frames añade un decágono al centro. Escala y rota los existentes

```
public void Canvas_Paint(object sender, PaintEventArgs e)
{
    Draw(e.Graphics);
}


1 referencia
public void Draw(Graphics graphics)
{
    graphics.Clear(Color.Black);

    using (Pen pen = new Pen(Color.FromArgb((int)alphaChannel, 255, 20, 147), 2))
    {
        for (int i = 0; i < polygons.Count; i++)
        {
            if (polygons[i][0].X - polygons[i][1].X > canvas.Width)
            {
                polygons.RemoveAt(i);
                i--;
                continue;
            }
            graphics.DrawPolygon(pen, polygons[i]);

            polygons[i] = PolygonBuilder.scaleFromCenter(polygons[i].ToList(), scaleFactor).ToArray();

            float angle = (i % 2 == 0) ? 1.5f : -1.5f;
            polygons[i] = PolygonBuilder.rotate(polygons[i].ToList(), angle).ToArray();
        }

        alphaChannel -= 0.25f;
        if (alphaChannel < 50) alphaChannel = 255;
    }
}
```

Aquí se llama a la función **Draw()** en la función **Canvas_Paint(..)** y en la función llamada, se escala y rota cada polígono. Dibuja todos con color magenta translúcido.

Visualizer4:



```
public class Visualizer4
{
    private Panel canvas;

    public List<PointF>[] points = new List<PointF>[4];

    1 referencia
    public Visualizer4(Panel canvas)
    {
        this.canvas = canvas;
        canvas.Paint += Canvas_Paint;

        float distance = canvas.Width / 50;

        for (int i = 0; i < 4; i ++)
        {
            points[i] = new List<PointF>();
        }


        for (int i = 0; i * distance < canvas.Width; i++)
        {
            points[0].Add(
                new PointF(0, i * distance)
                );
            points[1].Add(
                new PointF(i * distance, canvas.Height)
                );
            points[2].Add(
                new PointF(canvas.Width, i * distance)
                );
            points[3].Add(
                new PointF(i * distance, 0)
                );
        }
    }
```

Aquí se crean 4 conjuntos de puntos en los bordes del panel que seran conectados para formar una figura cerrada.

```
public void Update()
{
    scalePhase += 0.0005f;
    float scale = 1f + 0.05f * (float)Math.Sin(scalePhase);

    for (int i = 0; i < points.Length; i++)
    {
        points[i] = ScalePointsFromCenter(points[i], scale);
    }

    canvas.Invalidate();
}

private float scalePhase = 0f;


1 referencia
public void  Draw(Graphics graphics)
{
    using (System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Color.FromArgb(255, 20, 147), 2))
    {
        graphics.Clear(System.Drawing.Color.Black);
        int count = Math.Min(points[0].Count, Math.Min(points[1].Count, Math.Min(points[2].Count, points[3].Count)));

        for (int i = 0; i < count - 1; i++)
        {
            graphics.DrawLine(pen, points[0][i], points[1][i]);
            graphics.DrawLine(pen, points[1][i], points[2][count - 1 - i]);
            graphics.DrawLine(pen, points[2][count - 1 - i], points[3][count - 1 - i]);
            graphics.DrawLine(pen, points[3][count - 1 - i], points[0][i]);
        }



    }
}
```

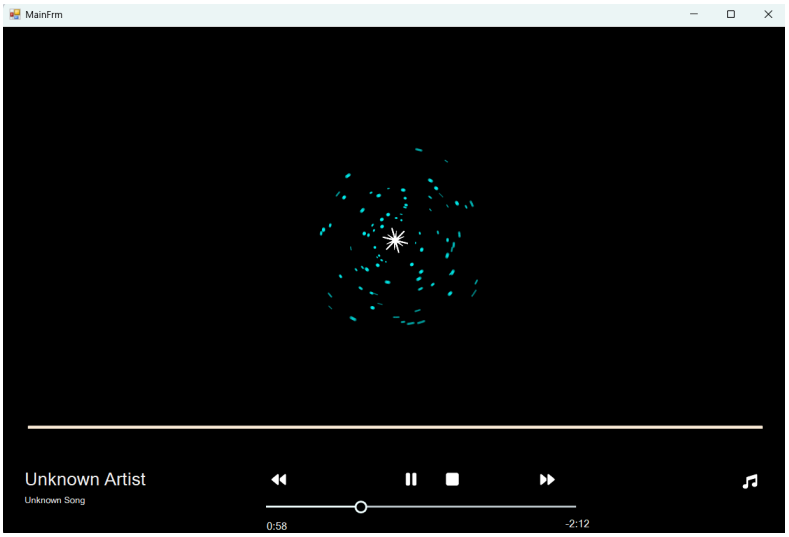
En la función **Update()** se aplica una transformación de escala sinusoidal a los puntos, creando un efecto de contracción y expansión desde el centro.

En la función **Draw(...)** se conectan los puntos de los bordes en la secuencia [0, 1, 2, 3, 0] formando una figura tipo lazo cerrado.

```
private List<PointF> ScalePointsFromCenter(List<PointF> points, float scale)
{
    float centerX = canvas.Width / 2f;
    float centerY = canvas.Height / 2f;

    List<PointF> scaled = new List<PointF>();

    foreach (PointF p in points)
    {
        float translatedX = p.X - centerX;
        float translatedY = p.Y - centerY;

        float scaledX = translatedX * scale;
        float scaledY = translatedY * scale;

        scaled.Add(new PointF(scaledX + centerX, scaledY + centerY));
    }

    return scaled;
}
```

En esta función **ScalePointFromCenter(...)** se aplica un escalado a los puntos desde el centro del panel, usado para dar efecto de pulsación.

Visualizer5:



```
internal class Visualizer5
{
    private Panel canvas;
    private float[] frequencies = new float[64];
    private float angle = 0;

    3 referencias
    private class Particle
    {
        public float x, y;
        public float radius;
        public float angle;
        public float speed;
        public float baseSize;
        public Color color;
    }

    private List<Particle> particles = new List<Particle>();
    private Random rand = new Random();


    1 referencia
    public Visualizer5(Panel canvas)
    {
        this.canvas = canvas;
        canvas.Paint += Canvas_Paint;

        for (int i = 0; i < 80; i++)
        {
            float radius = rand.Next(30, 150);
            float angle = (float)(rand.NextDouble() * Math.PI * 2);

            particles.Add(new Particle
            {
                radius = radius,
                angle = angle,
                speed = (float)(rand.NextDouble() * 0.01 + 0.005),
                baseSize = rand.Next(2, 6),
                color = Color.FromArgb(100, Color.Cyan)
            });
        }

    }
```

En este visualizer se inicializa una lista de partículas con valores aleatorios. Conecta el método **Paint.**

```
public void Update(float[] freqs)
{
    frequencies = freqs;
    angle += 0.05f;
    canvas.Invalidate();
}

3 referencias
public void Canvas_Paint(object sender, PaintEventArgs e)
{
    Draw(e.Graphics);
}

1 referencia
private void Draw(Graphics g)
{
    g.SmoothingMode = SmoothingMode.AntiAlias;
    g.Clear(Color.Black);

    float cx = canvas.Width / 2f;
    float cy = canvas.Height / 2f;
    int numWaves = 6;
    float maxRadius = Math.Min(cx, cy) - 0.2f;

    for (int i = 0; i < numWaves; i++)
    {
        double rotation = (2 * Math.PI / numWaves) * i + angle;
        DrawWave(g, cx, cy, rotation, maxRadius);
        DrawBounceLine(g);
        DrawParticles(g);

    }


}
```

En esta imagen se muestran las funciones **Update()** que es la que almacena las frecuencias y actualiza el ángulo. Solicita redibujar, la función **Draw(...)** dibuja múltiples ondas circulares basadas en las frecuencias, una línea horizontal que se mueve según la energía promedio del sonido, y partículas que orbitan alrededor del centro y aumentan de tamaño según el volumen.

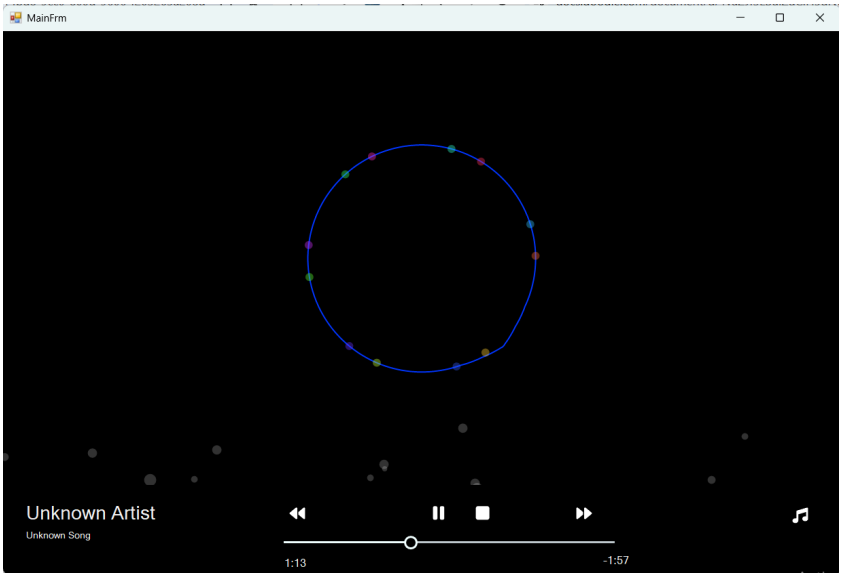
```
private void DrawWave(Graphics g, float cx, float cy, double rotation, float maxRadius)
{

    PointF[] points = new PointF[frequencies.Length];
    for (int i = 0; i < frequencies.Length; i++)
    {
        float amplitude = Math.Min(frequencies[i] * 4f, 3f);
        float radius = maxRadius * amplitude;

        double angle = (i / (double)frequencies.Length) * Math.PI * 2;

        float x = cx + (float)(Math.Cos(angle + rotation) * radius);
        float y = cy + (float)(Math.Sin(angle + rotation) * radius);
        points[i] = new PointF(x, y);
    }
    for (int j = 1; j <= 3; j++)
    {
        PointF[] blurred = points.Select(p =>
            new PointF(
                cx + (p.X - cx) * (1f + j * 0.05f),
                cy + (p.Y - cy) * (1f + j * 0.05f))
            ).ToArray();

        using (Pen auraPen = new Pen(Color.FromArgb(255, Color.White), 2f))
        {
            g.DrawPolygon(auraPen, blurred);
        }
    }
}

1 referencia
private void DrawBounceLine(Graphics g)
{
    float avg = frequencies.Average();
    float height = Math.Min(avg * 2000f, 2000f);

    float baseY = canvas.Height - 40;
    float lineY = baseY - height;

    using (Pen pen = new Pen(Color.AntiqueWhite, 4f))
    {
        g.DrawLine(pen, 40, lineY, canvas.Width - 40, lineY);
    }
}
```

En estas funciones, en la de **DrawWave(...)** se dibuja una onda circular con un efecto de aura blanco para dar sensación de brillo y en el método **DrawBonceLine(...)** se dibuja una linea que sube y baja con el volumen promedio.

```
private void DrawParticles(Graphics g)
{
    float cx = canvas.Width / 2f;
    float cy = canvas.Height / 2f;

    float volume = frequencies.Average();

    foreach (var p in particles)
    {
        p.angle += p.speed;
        float x = cx + (float)(Math.Cos(p.angle) * p.radius);
        float y = cy + (float)(Math.Sin(p.angle) * p.radius);
        float size = p.baseSize + (volume * 20f);

        using (Brush brush = new SolidBrush(p.color))
        {
            g.FillEllipse(brush, x - size / 2f, y - size / 2f, size, size);
        }
    }
}
```

En esta función **DrawParticles(...)** se dibujan partículas que se mueven en círculos alrededor del centro.

Visualizer6:



```
internal class Visualizer6
{
    private Panel canvas;
    private float[] frequencies = new float[64];
    private float angle = 0;
    private List<FloatingBubble> bubbles = new List<FloatingBubble>();
    private Random random = new Random();

    1 referencia
    public Visualizer6(Panel canvas)
    {
        this.canvas = canvas;
        canvas.Paint += Canvas_Paint;
    }

    1 referencia
    public void Update(float[] freqs)
    {
        frequencies = freqs;
        angle += 0.01f;


        if (random.NextDouble() < 0.2)
        {
            bubbles.Add(new FloatingBubble(
                x: random.Next(canvas.Width),
                y: canvas.Height + 10,
                radius: random.Next(4, 10),
                speed: (float)(random.NextDouble() * 1.5 + 0.5),
                color: Color.FromArgb(50, Color.White)
            ));
        }


        for (int i = bubbles.Count - 1; i >= 0; i--)
        {
            bubbles[i].Y -= bubbles[i].Speed;
            if (bubbles[i].Y + bubbles[i].Radius < 0)
                bubbles.RemoveAt(i);
        }

        canvas.Invalidate();
    }
```

En la siguiente imagen, se puede observar cómo se le asigna el canvas y se suscribe al evento **Paint** el visualizer, en la función **Update(...)** se guarda la frecuencia y actualiza el ángulo, se crean nuevas burbujas blancas que flotan desde abajo y elimina las que salen del canvas.

```
private void Draw(Graphics g)
{
    g.SmoothingMode = SmoothingMode.AntiAlias;
    g.Clear(Color.Black);

    foreach (var b in bubbles)
    {
        using (Brush brush = new SolidBrush(b.Color))
        {
            g.FillEllipse(brush, b.X - b.Radius, b.Y - b.Radius, b.Radius * 2, b.Radius * 2);
        }
    }

    float cx = canvas.Width / 2f; float cy = canvas.Height / 2f;
    float baseRadius = Math.Min(cx, cy) * 0.5f;

    int detail = 256;
    PointF[] points = new PointF[detail];

    for (int i = 0; i < detail; i++)
    {
        float interpIndex = (i / (float)detail) * frequencies.Length;
        int idx = (int)Math.Floor(interpIndex);
        int next = (idx + 1) % frequencies.Length;
        float t = interpIndex - idx;

        float amp = 0;
        if (idx < frequencies.Length && next < frequencies.Length)
        {
            float interp = frequencies[idx] * (1 - t) + frequencies[next] * t;
            amp = Math.Min(interp * 12f, 1.2f);
        }

        float radius = baseRadius + amp * 50f;
        double theta = (i / (double)detail) * Math.PI * 2;
        float x = cx + (float)(Math.Cos(theta + angle) * radius);
        float y = cy + (float)(Math.Sin(theta + angle) * radius);
        points[i] = new PointF(x, y);
    }

    float hue = (float)(angle % 1.0);
    Color dynamicColor = ColorFromHSV(hue * 360, 1, 1);
```

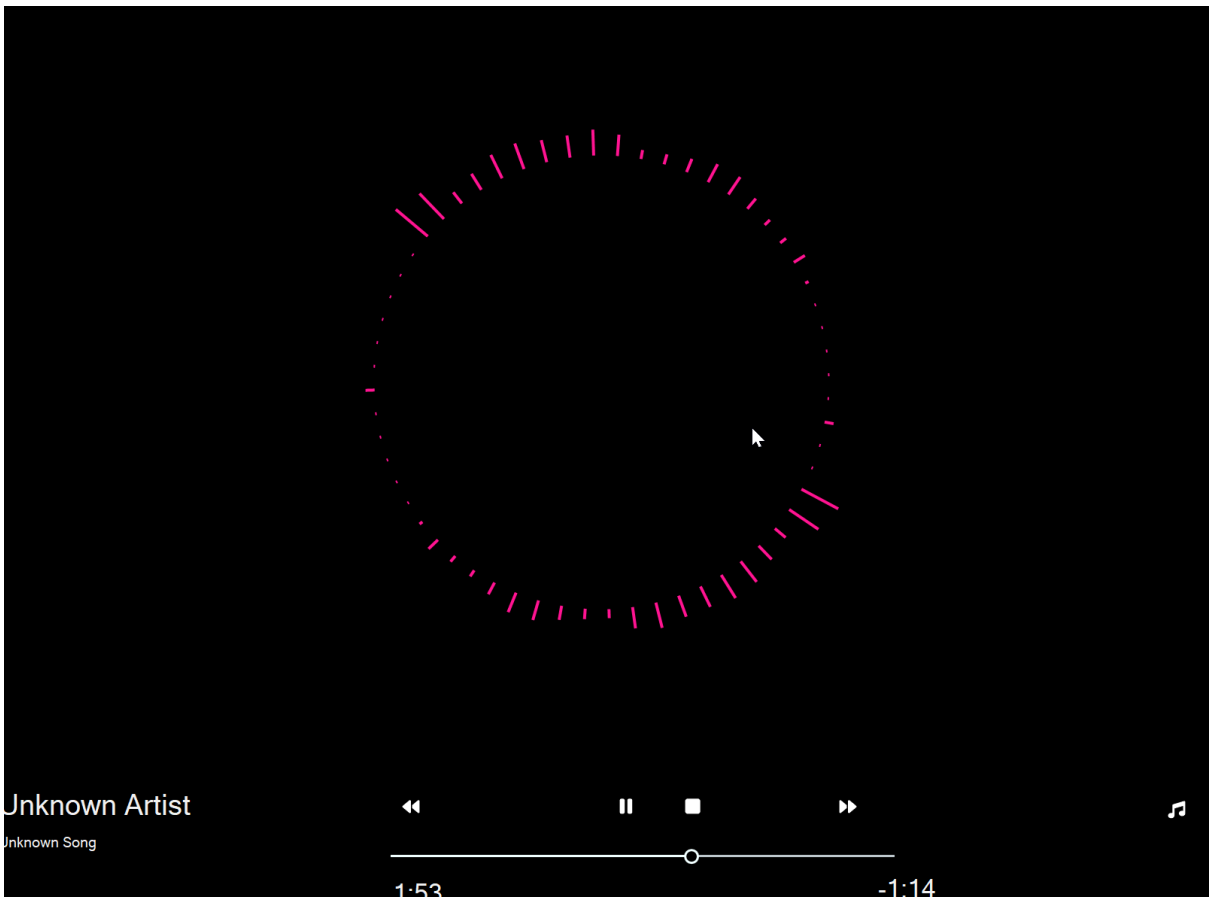
```
    using (Pen pen = new Pen(dynamicColor, 2f))
    {
        g.DrawClosedCurve(pen, points, 1.0f, FillMode.Alternate);
    }

    for (int i = 0; i < 12; i++)
    {
        int band = random.Next(frequencies.Length);
        float amplitude = frequencies[band] * 80f;

        float angleOffset = angle * 2 + i;
        float r = baseRadius + amplitude;

        float x = cx + (float)Math.Cos(angleOffset) * r;
        float y = cy + (float)Math.Sin(angleOffset) * r;

        Color bubbleColor = ColorFromHSV((hue * 360 + i * 30) % 360, 0.8, 1);
        using (Brush b = new SolidBrush(Color.FromArgb(100, bubbleColor)))
        {
            g.FillEllipse(b, x - 6, y - 6, 12, 12);
        }
    }
}
```

En la función **Draw(...)** se dibuja el fondo con burbujas blancas flotando, se dibuja el anillo circular deformado por las frecuencias y se añade pequeñas burbujas de colores que giran alrededor del anillo, con colores dinámicos en función del ángulo.

```
private Color ColorFromHSV(double hue, double saturation, double value)
{
    int hi = Convert.ToInt32(Math.Floor(hue / 60)) % 6;
    double f = hue / 60 - Math.Floor(hue / 60);

    value *= 255;
    int v = Convert.ToInt32(value);
    int p = Convert.ToInt32(value * (1 - saturation));
    int q = Convert.ToInt32(value * (1 - f * saturation));
    int t = Convert.ToInt32(value * (1 - (1 - f) * saturation));

    switch (hi)
    {
        case 0: return Color.FromArgb(v, t, p);
        case 1: return Color.FromArgb(q, v, p);
        case 2: return Color.FromArgb(p, v, t);
        case 3: return Color.FromArgb(p, q, v);
        case 4: return Color.FromArgb(t, p, v);
        default: return Color.FromArgb(v, p, q);
    }
}


4 referencias
private class FloatingBubble
{
    public float X, Y, Radius, Speed;
    public Color Color;

    1 referencia
    public FloatingBubble(float x, float y, float radius, float speed, Color color)
    {
        X = x;
        Y = y;
        Radius = radius;
        Speed = speed;
        Color = color;
    }
}
```

En las siguientes 2 funciones, se convierten valores de tono, saturación y brillo con un color RGB y se representa una burbuja flotante con posición, tamaño, velocidad y color respectivamente.

Visualizer7:


Unknown Artist
Unknown Song
1:53
-1:14

```
6 references
public class Visualizer7
{
    private Panel canvas;
    private float[] amplitudes;
    private float angleStep;
    private float rotation = 0;

    1 reference
    public Visualizer7(Panel canvas)
    {
        this.canvas = canvas;
        this.canvas.Paint += Canvas_Paint;

        amplitudes = new float[60];
        angleStep = 360f / amplitudes.Length;
    }
    1 reference
    public void Update(float[] freqs)
    {
        int sourceLen = freqs.Length;
        int targetLen = amplitudes.Length;

        for (int i = 0; i < targetLen; i++)
        {
            float freq = freqs[i % sourceLen];
            float db = 20 * (float)Math.Log10(freq + 1e-6f);
            amplitudes[i] = Math.Max((db + 60) * 1.5f, 2f);
        }

        rotation += 0.01f;
        canvas.Invalidate();
    }

    3 references
    public void Canvas_Paint(object sender, PaintEventArgs e)
    {
        Draw(e.Graphics);
    }
```

Constructor:

- Guarda la referencia al panel recibido.
- Suscribe el método Canvas_Paint al evento Paint del panel, para que se dibuje automáticamente cuando sea necesario.
- Inicializa el arreglo amplitudes con 60 elementos (bandas de frecuencia).
- Calcula el ángulo entre cada barra del visualizador circular (angleStep).

Update():

- Calcula la cantidad de frecuencias de entrada y la cantidad de bandas del visualizador.
- Para cada banda, toma una frecuencia (cíclicamente si hay menos frecuencias que bandas).
- Convierte la frecuencia a decibelios y la escala para obtener la amplitud visual.
- Actualiza el ángulo de rotación para animar el visualizador.
- Llama a canvas.Invalidate() para forzar el repintado del panel.

Canvas_Paint():

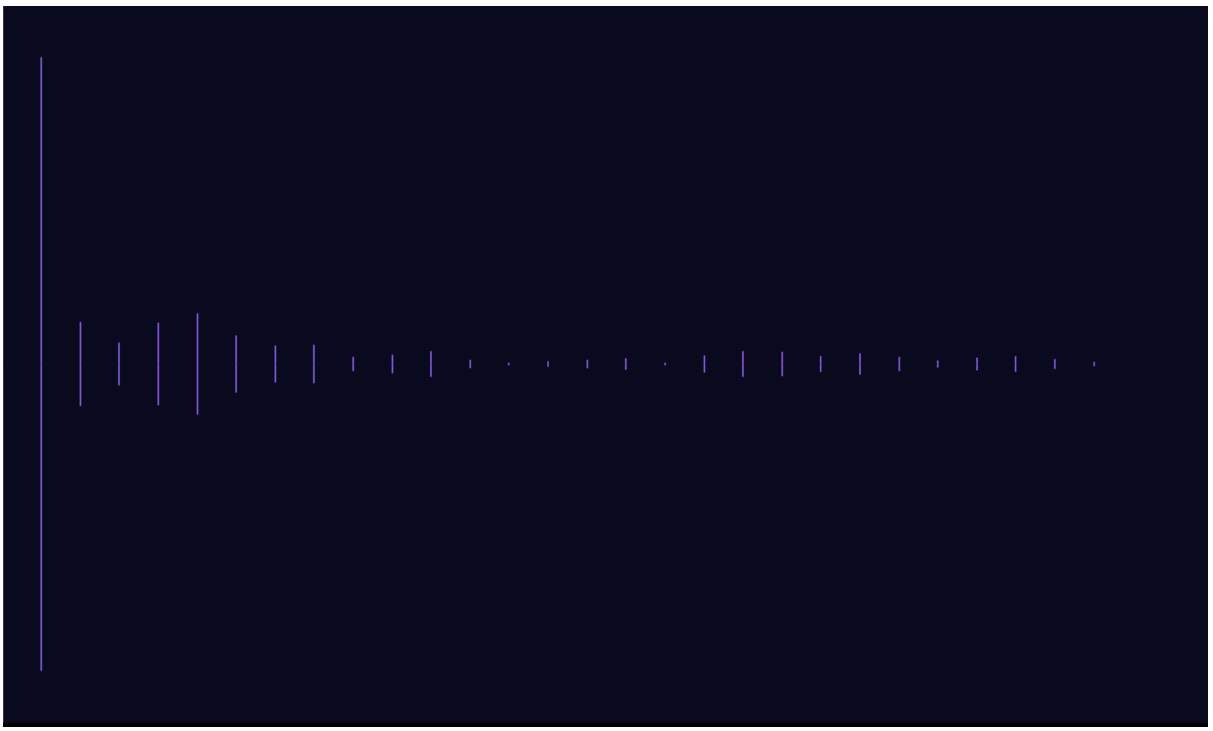
- Llama al método Draw, pasando el objeto Graphics para dibujar el visualizador.

```
1 reference
private void Draw(Graphics g)
{
    g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
    g.Clear(Color.Black);

    float centerX = canvas.Width / 2f;
    float centerY = canvas.Height / 2f;
    float radius = Math.Min(centerX, centerY) * 0.6f;

    using (Pen pen = new Pen(Color.DeepPink, 4))
    {
        for (int i = 0; i < amplitudes.Length; i++)
        {
            float angle = (i * angleStep + rotation * 180f / (float)Math.PI) * (float)Math.PI / 180f;
            float x1 = centerX + radius * (float)Math.Cos(angle);
            float y1 = centerY + radius * (float)Math.Sin(angle);
            float x2 = centerX + (radius + amplitudes[i]) * (float)Math.Cos(angle);
            float y2 = centerY + (radius + amplitudes[i]) * (float)Math.Sin(angle);

            g.DrawLine(pen, x1, y1, x2, y2);
        }
    }
}
```

Draw():

- Activa el suavizado de gráficos y limpia el fondo a negro.
- Calcula el centro y el radio del círculo base.
- Crea un Pen color rosa fuerte para dibujar las líneas.
- Para cada banda de amplitud:
    - Calcula el ángulo correspondiente (incluyendo la rotación animada).
    - Calcula el punto de inicio (en el círculo base) y el punto final (en el círculo expandido por la amplitud).
    - Dibuja una línea desde el centro hacia afuera, representando la energía de esa banda de frecuencia.

Visualizer8:



```
public class Visualizer8
{
    private Panel canvas;
    private float[] amplitudes = new float[0];

    1 reference
    public Visualizer8(Panel canvas)
    {
        this.canvas = canvas;
        this.canvas.Paint += Canvas_Paint;
    }

    1 reference
    public void Update(float[] frequencyAmplitudes)
    {
        this.amplitudes = frequencyAmplitudes;
        canvas.Invalidate();
    }
```

Constructor:

- Guarda la referencia al panel recibido.
- Suscribe el método Canvas_Paint al evento Paint del panel, para que se dibuje automáticamente cuando sea necesario.
- Inicializa el arreglo amplitudes como vacío.

Update():

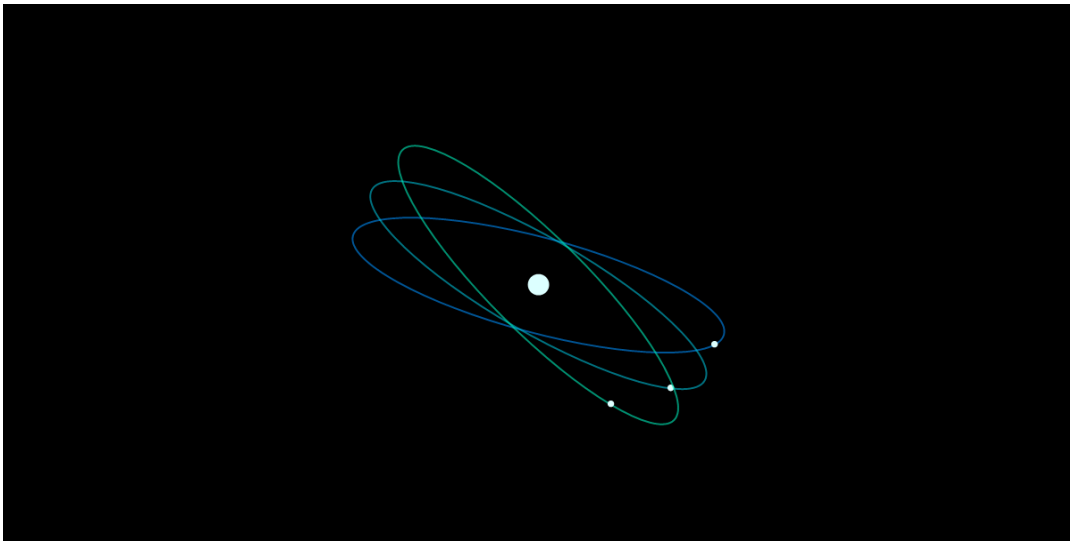
- Asigna el arreglo recibido a la variable amplitudes.
- Llama a canvas.Invalidate() para forzar el repintado del panel, lo que provoca que se ejecute el método Canvas_Paint.

```
3 references
public void Canvas_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.Clear(Color.FromArgb(0, 0, 0));

    if (amplitudes == null || amplitudes.Length == 0)
        return;

    int numBars = amplitudes.Length;
    int barWidth = 2;

    int totalBarWidth = barWidth * numBars;
    int availableSpace = canvas.Width - totalBarWidth;
    float spacing = (numBars > 1) ? (float)availableSpace / (numBars - 1) : 0;

    int centerY = canvas.Height / 2;

    using (Brush barBrush = new SolidBrush(Color.FromArgb(150, 100, 255)))
    {
        for (int i = 0; i < numBars; i++)
        {
            float amplitude = Math.Min(amplitudes[i] * 5f, 1f);
            int barHeight = (int)(amplitude * (canvas.Height / 2));

            int x = (int)(i * (barWidth + spacing));
            Rectangle topBar = new Rectangle(x, centerY - barHeight, barWidth, barHeight);
            Rectangle bottomBar = new Rectangle(x, centerY, barWidth, barHeight);

            g.FillRectangle(barBrush, topBar);
            g.FillRectangle(barBrush, bottomBar);
        }
    }
}
```

Canvas_Paint():

- Obtiene el objeto Graphics para dibujar.
- Limpia el fondo del panel con color negro.
- Si no hay amplitudes, termina la función.
- Calcula el número de barras, el ancho de cada barra y el espacio disponible para distribuirlas uniformemente.
- Calcula la posición vertical central del panel.
- Crea un pincel (Brush) de color violeta claro y semitransparente.
- Para cada barra:
    - Calcula la altura de la barra según la amplitud (escalada y limitada).
    - Calcula la posición horizontal de la barra.
    - Dibuja dos rectángulos: uno hacia arriba y otro hacia abajo desde el centro, creando un efecto de barras simétricas.

Visualizer9:



El Visualizer9 representa un concepto visual inspirado en un modelo atómico, donde el centro de la pantalla funciona como núcleo, y las órbitas que giran a su alrededor simulan el comportamiento de los electrones.

```
public void Update(float[] frequencyAmplitudes, float volume)
{
    this.audioSamples = frequencyAmplitudes;
    this.currentVolume = volume;
    this.rotationAngle = (rotationAngle + 0.5f) % 360;
    canvas.Invalidate();
}

public void Canvas_Paint(object sender, PaintEventArgs e)
{

    e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
    e.Graphics.Clear(Color.Black);
    if (audioSamples == null || audioSamples.Length == 0) return;
    DrawUniformOrbits(e.Graphics);
}

```

Update(float[] frequencyAmplitudes, float volume)

Se llama en cada frame desde el controlador principal de visualizadores, posteriormente actualiza el volumen y la rotación y finalmente llama a Invalidate() para redibujar el canvas.

Canvas_Paint(...)

Se activa cuando el panel necesita redibujarse y llama a DrawUniformOrbits para representar gráficamente las órbitas.

```
private void DrawUniformOrbits(Graphics g)
{
    int centerX = canvas.Width / 2;
    int centerY = canvas.Height / 2;

    float scale = 1.0f + currentVolume * 0.5f;
    int width = (int)(BaseEllipseWidth * scale);
    int height = (int)(BaseEllipseHeight * scale);

    for (int i = 0; i < OrbitCount; i++)
    {
        float orbitRotation = rotationAngle * (i + 1) / 2;

        using (Pen orbitPen = new Pen(GetOrbitColor(i), 2))
        {
            g.TranslateTransform(centerX, centerY);
            g.RotateTransform(orbitRotation);
            g.TranslateTransform(-centerX, -centerY);

            g.DrawEllipse(orbitPen,
                centerX - width / 2,
                centerY - height / 2,
                width,
                height);

            DrawElectron(g, centerX, centerY, width, height, orbitRotation);

            g.ResetTransform();
        }
    }

    DrawNucleus(g, centerX, centerY);
}
```

DrawUniformOrbits(Graphics g)

Dibuja 3 órbitas elípticas centradas, en donde cada órbita tiene:

- Tiene un color distinto.
- Rota suavemente con base en rotationAngle.
- Contiene un electrón que gira en su trayectoria elíptica.

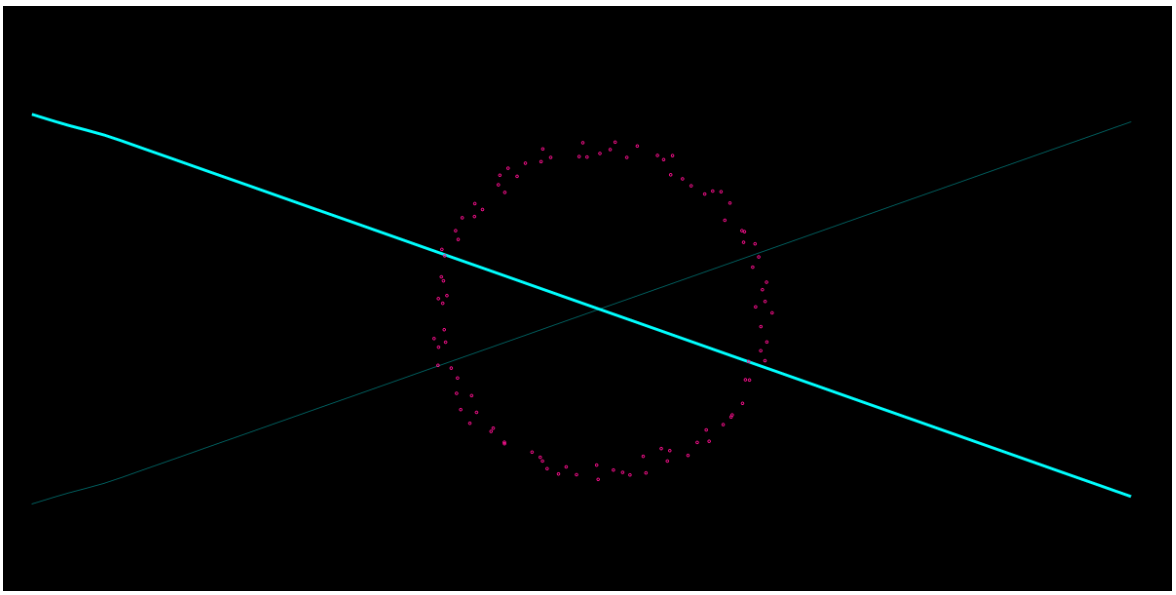
DrawElectron(...)

Calcula la posición del electrón en la órbita elíptica con funciones trigonométricas, en donde el tamaño del electrón depende del volumen actual y luego se dibuja con un color claro para resaltar el movimiento.

DrawNucleus(...)

Representa el núcleo central del átomo, es un círculo ligeramente más grande que los electrones y su tamaño también se ve influenciado por el volumen, dando la sensación de "pulsación" con el audio.

Visualizer10:



El Visualizer10 ofrece una representación visual dual: una onda central que rota lentamente sobre sí misma, acompañada de puntos estáticos distribuidos en un patrón circular.

```
public void Update(float[] frequencyAmplitudes, float volume)
{
    this.audioSamples = frequencyAmplitudes;
    this.currentVolume = volume;

    rotationAngle = (rotationAngle + 0.2f) % 360; // Rotación por frame

    canvas.Invalidate();
}

public void Canvas_Paint(object sender, PaintEventArgs e)
{
    DrawWaveform(e.Graphics);
    DrawPoints(e.Graphics);
}
```

Update(frequencyAmplitudes, volume)

Se llama por cada frame de audio, en donde actualiza los valores de volumen y ángulo de rotación y luego llama a Invalidate() para refrescar el canvas.

Canvas_Paint(...)

Se dispara cuando el panel necesita redibujarse, para ello llama a:

- DrawWaveform: para dibujar la onda de audio y su reflejo.
- DrawPoints: para pintar los puntos estáticos alrededor.

```
private void DrawWaveform(Graphics g)
{
    if (audioSamples == null || audioSamples.Length == 0) return;

    g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
    g.Clear(Color.Black);

    int width = canvas.Width;
    int height = canvas.Height;
    int centerY = height / 2;
    float scale = height * 0.4f * (0.5f + currentVolume * 0.5f);

    PointF[] points = new PointF[audioSamples.Length];
    for (int i = 0; i < audioSamples.Length; i++)
    {
        float x = i * width / (float)audioSamples.Length;
        float y = centerY + audioSamples[i] * scale;
        points[i] = new PointF(x, y);
    }

    PointF[] rotatedPoints = RotatePoints(points, new PointF(width / 2, centerY), rotationAngle);

    using (Pen wavePen = new Pen(Color.Cyan, 3))
    {
        g.DrawCurve(wavePen, rotatedPoints, 0.5f);
    }

    using (Pen reflectionPen = new Pen(Color.FromArgb(100, 0, 255, 255), 1))
    {
        PointF[] reflectionPoints = new PointF[rotatedPoints.Length];
        for (int i = 0; i < rotatedPoints.Length; i++)
        {
            reflectionPoints[i] = new PointF(
                rotatedPoints[i].X,
                centerY + (centerY - rotatedPoints[i].Y)
            );
        }
        g.DrawCurve(reflectionPen, reflectionPoints, 0.5f);
    }
}

private void DrawPoints(Graphics g)
{
    if (audioSamples == null) return;
    float pointSize = 2 + currentVolume * 3;


    foreach (var point in staticPoints)
    {
        int blueValue = 150 + (int)(currentVolume * 105);
        using (Pen circlePen = new Pen(Color.FromArgb(255, 20, 147), 1))
        {
            g.DrawEllipse(circlePen, point.X, point.Y, pointSize, pointSize);
        }
    }
}
```

DrawWaveform(Graphics g)

Dibuja una curva suavizada basada en los valores de audioSamples, en donde la altura de la onda se escala con el volumen (scale) y por último aquí se aplica una rotación usando RotatePoints para crear una sensación dinámica.

DrawPoints(Graphics g)

Dibuja pequeños círculos fijos en una disposición circular, generados al inicio del visualizador. El tamaño de cada punto varía levemente con el volumen del audio, y mantiene un color rosa fuerte.

Conclusiones:

- La aplicación que se desarrolló cumple con la simulación de un reproductor multimedia, integrando visualizadores gráficos en tiempo real.
- El uso de estructuras modulares facilitó la implementación y comprensión del funcionamiento interno del sistema
- Los efectos gráficos responden de forma adecuada a los cambios de volumen y frecuencia, ofreciendo al usuario una experiencia visual inmersiva y representativa del sonido.

Recomendaciones:

- Incluir opciones para que el usuario elija o configure su visualizador preferido.
- Optimizar el rendimiento en visualizadores complejos para asegurar fluidez en equipos con menores recursos.
- Permitir ingresar múltiples audios y que estos se queden guardados en una lista para su fácil acceso y navegación entre audios.