UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Software

Asignatura: Computación Gráfica – NRC28467

Informe Técnico – Reproductor de Música

Integrantes:

Báez, Gabriel
Calvache, Gabriel
Valencia, Yuliana

Profesor: Ing. Dario Javier Morales Caiza

Fecha: 9 de noviembre de 2025

Sangolquí – Ecuador

Índice

1. Justificación del Diseño Gráfico

El diseño del reproductor de música se basa en una interfaz limpia, moderna y funcional, pensada para ofrecer una experiencia visual atractiva y una operación intuitiva. Se emplea una paleta de colores oscuros con acentos en tonos violetas y rosados (como `MediumVioletRed`), buscando un contraste adecuado entre los controles y el fondo, además de transmitir una estética moderna similar a reproductores contemporáneos.

Los botones principales (Play, Pause, Stop, Siguiente, Anterior) están dispuestos de forma central para facilitar el acceso rápido, mientras que la barra de progreso y el visualizador de audio animado proporcionan retroalimentación visual sobre la reproducción. La inclusión de un `PictureBox` como lienzo de dibujo permite representar las ondas o picos del sonido mediante animaciones, dando al usuario una sensación de dinamismo y respuesta visual al ritmo de la música.

2. Descripción de las transformaciones implementadas

El análisis espectral (FFT) constituye el punto de partida del sistema de visualización. Cuando está disponible, el componente `AudioAnalyzer` procesa la señal de audio y devuelve magnitudes separadas por bandas de frecuencia, las cuales sirven como datos de entrada para generar los efectos visuales. En caso de que no exista un análisis real, se implementa una simulación que utiliza funciones trigonométricas y ruido aleatorio para producir valores plausibles, garantizando así una respuesta visual continua y coherente incluso sin datos de audio reales.

El suavizado temporal (EMA) se aplica a las magnitudes espectrales mediante un promedio móvil exponencial controlado por la variable `spectrumSmoothing`. Este proceso atenúa las fluctuaciones abruptas entre cuadros consecutivos, proporcionando una animación más fluida y agradable a la vista.

El escalado no lineal de amplitud introduce una corrección perceptiva en los valores obtenidos del análisis. A través de la función `Math.Pow(amp, 0.6)`, se comprimen las dinámicas del sonido, haciendo visibles los niveles bajos sin saturar los picos más intensos. Este ajuste mejora la legibilidad visual de las frecuencias y equilibra la representación entre sonidos suaves y potentes.

En cuanto a la detección de beats y pulsos, el sistema analiza la variación promedio de amplitudes entre cuadros para identificar incrementos significativos. Cuando la energía supera un umbral predefinido, se genera un pulso (`beatPulse`) que altera temporalmente la escala y el brillo de los elementos visuales, además de activar la creación de partículas que refuerzan el impacto del ritmo musical.

La emisión y física de partículas se desencadena en cada pulso detectado. En ese momento, se generan partículas con propiedades físicas como velocidad, fricción y decaimiento de opacidad (`alpha`). Estas partículas se actualizan en cada cuadro y se dibujan como destellos radiales que se disipan gradualmente, añadiendo dinamismo y realismo a la experiencia visual.

Las transformaciones radiales se utilizan para efectos de anillos, destellos y ondas. Se emplea la conversión de coordenadas polares a cartesianas (ángulo, radio) para posicionar los puntos y líneas que componen las figuras radiales.

En la deformación poligonal, cada vértice del polígono está vinculado a una banda del espectro de frecuencias. La magnitud de esa banda modifica la distancia radial del vértice respecto al centro, provocando deformaciones animadas del contorno.

El uso de gradientes y mezcla de color aporta riqueza cromática a la visualización. Se emplean herramientas como `LinearGradientBrush` y `ColorBlend` para generar degradados verticales en las barras espectrales, además de una función personalizada `BlendColors` que combina tonos coherentes

con el tema seleccionado.

La conversión HSL→RGB se implementa mediante la función `ColorFromHue`, que transforma valores de tono, saturación y luminosidad en colores RGB vibrantes. Esto permite crear paletas dinámicas que varían por capa y se adaptan al ritmo de la música.

Finalmente, la aceleración y amortiguación visual se logran al hacer que parámetros como `beatPulse` y el tamaño de las partículas decaigan exponencialmente en cada cuadro, evitando efectos permanentes y simulando un comportamiento físico realista.

3. Capturas de Pantalla

Figura 1: Vista del proyecto en Visual Studio 2022 como área de trabajo.

Figura 2: Vista previa de las funcionalidades del formulario `frmMediaPlayer.cs`.

Figura 3: Vista previa del código en C# que implementa la funcionalidad principal del programa.

Figura 4: Interfaz principal del reproductor en ejecución, mostrando controles, barra de progreso y área de visualización animada.

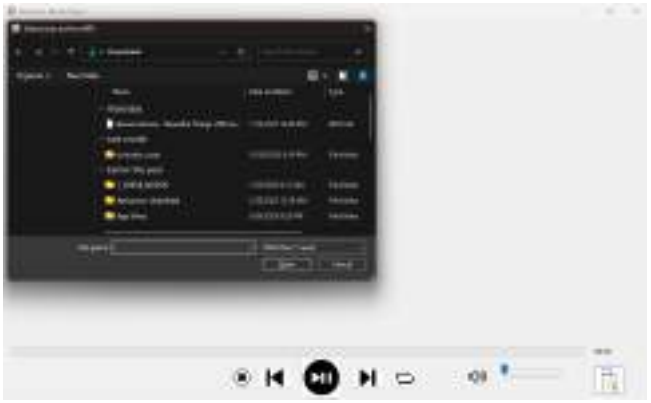

Figura 5: Diálogo de selección de archivos para cargar audio (`.mp3`) y ejemplo de ruta seleccionada.

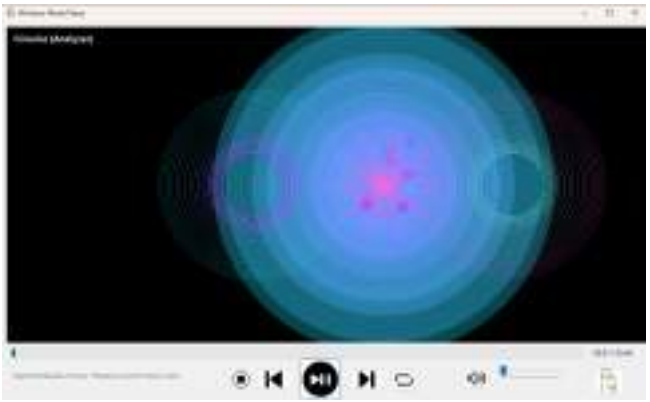

Figura 6: Visualización dinámica en forma de círculos que responde en tiempo real a la energía del espectro de audio.

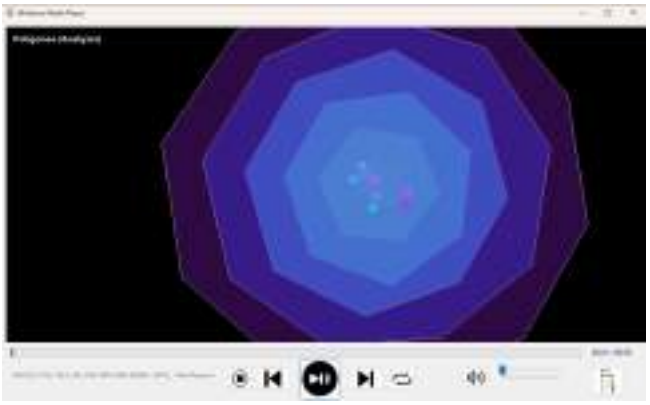

Figura 7: Visualización mediante elipses deformadas, sincronizadas con las bandas de frecuencia y la amplitud del sonido.

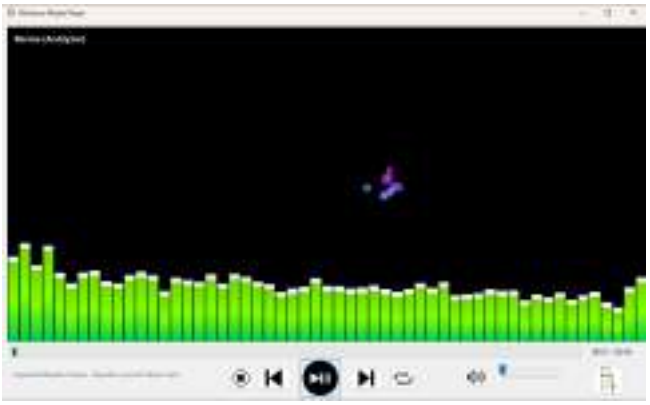

Figura 8: Representación en barras espectrales mostrando las magnitudes por banda obtenidas por el análisis FFT durante la reproducción.

4. Código Fuente Relevante Comentado

Archivo: `CMediaPlayer.cs`

```
class CMediaPlayer
{
    private AxWMPLib.AxWindowsMediaPlayer player;   // Control principal para reproducci n de audio.
    private Timer animationTimer;                   // Temporizador que controla la animaci n del ecualizador.
    private int progress;                           // Variable que almacena el avance del dibujo animado.
    private PictureBox canvas;                      // Lienzo gr fico donde se dibujan las ondas.
    private Pen pen = new Pen(Color.MediumVioletRed, 2); // Define color y grosor de la l nea.
    private int offset = 0;                         // Desfase para el movimiento de la onda.
}
```

Listing 1: Definición de la clase CMediaPlayer

Este bloque define los atributos principales del reproductor, combinando el control de Windows Media Player con un sistema de animación propio basado en GDI+.

```
public CMediaPlayer(AxWMPLib.AxWindowsMediaPlayer player, PictureBox canvas)
{
    this.player = player;
    this.canvas = canvas;
    animationTimer = new Timer();
    animationTimer.Interval = 50; // Actualiza la animaci n cada 50 ms.
    animationTimer.Tick += AnimationTimer_Tick;
}
```

Listing 2: Constructor de la clase CMediaPlayer

En este constructor se inicializa el reproductor y se configura el temporizador de animación, asegurando una actualización fluida del gráfico. El uso de un intervalo de 50 milisegundos proporciona una frecuencia aproximada de 20 cuadros por segundo.

```
private void AnimationTimer_Tick(object sender, EventArgs e)
{
    offset += 5; // Desplaza la onda para generar movimiento.
    canvas.Invalidate(); // Redibuja el PictureBox.
}
```

Listing 3: Método de actualización del temporizador

Aquí se genera el movimiento continuo de la animación. El incremento del valor `offset` produce un desplazamiento horizontal en la función seno, y el método `Invalidate()` provoca el redibujado del lienzo para reflejar el nuevo cuadro.

```
public void DrawWave(Graphics g)
{
    int midY = canvas.Height / 2;
    for (int x = 0; x < canvas.Width; x += 5)
    {
        int y = (int)(midY + 30 * Math.Sin((x + offset) * Math.PI / 180));
        g.DrawLine(pen, x, midY, x, y);
    }
}
```

Listing 4: Función de dibujo de la onda sinusoidal

Esta función implementa el cálculo trigonométrico que simula las ondas de sonido. Cada línea vertical representa una muestra visual, y la ecuación basada en la función seno permite alternar alturas de manera periódica, creando la ilusión de un ecualizador dinámico.

Archivo: `frmMusicPlayer.cs`

```
public partial class frmMusicPlayer : Form
{
    private CMediaPlayer mediaPlayer;

    public frmMusicPlayer()
    {
        InitializeComponent();
```

```
        mediaPlayer = new CMediaPlayer(axWindowsMediaPlayer1, pictureBoxCanvas);
    }
}
```

Listing 5: Inicialización del formulario principal

El formulario principal crea una instancia del objeto `CMediaPlayer`, pasando el control de reproducción (`AxWindowsMediaPlayer`) y el lienzo gráfico (`PictureBox`). De esta forma, la lógica del reproductor se mantiene modular, separando la interfaz de usuario de la funcionalidad de animación.

```
private void btnPlay_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "Archivos de audio|*.mp3;*.wav";
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        axWindowsMediaPlayer1.URL = openFileDialog1.FileName;
        axWindowsMediaPlayer1.Ctlcontrols.play();
    }
}
```

Listing 6: Evento para la reproducción de archivos

Este método gestiona la reproducción de archivos de audio. Permite al usuario seleccionar archivos con extensiones `.mp3` o `.wav`, y los reproduce mediante el control de Windows Media Player. La instrucción `Ctlcontrols.play()` inicia la reproducción, mientras que el filtro del cuadro de diálogo restringe la selección a tipos de audio válidos.

5. Historial de commits

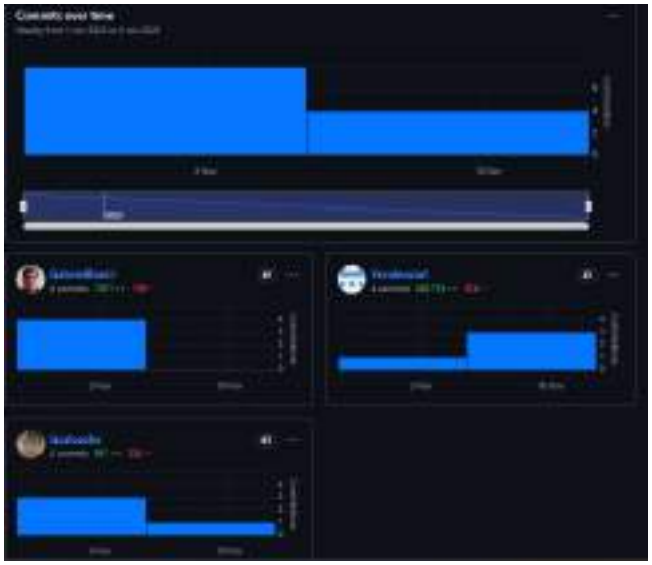

Figura 9: Historial de commits para trabajo del proyecto

6. Conclusiones

- El desarrollo del reproductor de música permitió integrar de forma efectiva los conceptos fundamentales de computación gráfica y programación orientada a eventos en C. La implementación

de transformaciones visuales sincronizadas con la reproducción de audio demostró cómo los principios matemáticos pueden aplicarse en la generación de animaciones dinámicas y atractivas.

- El uso del componente AxWindowsMediaPlayer facilitó la gestión de archivos de audio, mientras que el empleo de GDI+ proporcionó control sobre el dibujo y renderizado de elementos gráficos, garantizando fluidez en las animaciones. Además, la modularización mediante la clase CMediaPlayer promovió una estructura de código clara, reutilizable y fácil de mantener.
- El proyecto logró cumplir su objetivo de ofrecer una interfaz moderna, funcional y visualmente atractiva, capaz de combinar la reproducción de música con efectos visuales que mejoran la experiencia del usuario. Asimismo, se comprobó la importancia del manejo eficiente de los recursos gráficos y del control de excepciones para mantener la estabilidad de la aplicación.

7. Recomendaciones

- Incorporar una biblioteca que permita vincular directamente las visualizaciones con las frecuencias del sonido, logrando una respuesta visual más precisa.
- Añadir soporte para redimensionamiento dinámico y temas personalizables, con el fin de mejorar la accesibilidad y la experiencia de usuario.
- Integrar un sistema de playlist y una función aleatorio aumentaría la funcionalidad y el atractivo del reproductor.

Bibliografía

- Kennedy, E. D. (2017). *Color in UI Design: A (Practical) Framework*. Medium. Recuperado de https://medium.com/@erikdkennedy/color-in-ui-design-a-practical-framework-e18cacd97f9e
- Li, X., & Zhang, Y. (2018). *Study on Application of Audio Visualization in New Media Art*. ResearchGate. Recuperado de https://www.researchgate.net/publication/328235993_Study_on_Application_of_Audio_Visualization_in_New_Media_Art
- Microsoft. (2023). *Graphics in Windows Forms Applications (GDI+)*. Microsoft Learn. Recuperado de https://learn.microsoft.com/en-us/dotnet/desktop/winforms/advanced/graphics-and-drawin
- Smith, J. O. (2020). *The Fast Fourier Transform (FFT): Theory and Applications*. Stanford University. Recuperado de https://ccrma.stanford.edu/~jos/mdft/
- Wikipedia contributors. (2025). *Aesthetic–usability effect*. Wikipedia. Recuperado el 10 de noviembre de 2025 de https://en.wikipedia.org/wiki/Aesthetic%E2%80%93usability_effect