Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Implementación de una Aplicación de Procesado Digital de la Señal para Analizar Frecuencias de Sonido (Audio) en Lenguaje C en Entorno GNU Linux

Autor: Isabella Bucciarelli Imbrondone

Tutor(a): José Crespo del Arco

Madrid, junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Implementación de una Aplicación de Procesado Digital de la Señal para Analizar Frecuencias de Sonido (Audio) en Lenguaje C en Entorno GNU Linux.
Junio 2024
Autor: Isabella Bucciarelli Imbrondone

Tutor:
José Crespo del Arco
Lenguaje y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El procesamiento de señales digitales es una disciplina que permite manipular una señal para que pueda ser analizada, modificada o mejorada. Esta disciplina se utiliza actualmente en diversos campos, con diferentes tipos de aplicaciones, como las telecomunicaciones, el procesamiento de imágenes, la medicina, entre otros.

Las señales digitales son una representación de una señal analógica convertida a formato digital. Un ejemplo común de señales digitales son los archivos de audio. Mediante el procesamiento de estas señales digitales, se pueden extraer los componentes de frecuencia presentes en el audio. A través del análisis de estos componentes de frecuencia, se puede obtener información valiosa sobre el contenido del audio, como las frecuencias dominantes, patrones, presencia de ruido, entre otros. Una de las formas de analizar las frecuencias presentes en una señal es mediante la visualización de sus componentes de frecuencia a lo largo del tiempo.

En este contexto, se propone el desarrollo de una aplicación en lenguaje C, ejecutada en un entorno GNU/Linux, que sea capaz de procesar señales de audio y permita visualizar sus frecuencias a través de una interfaz de usuario. El objetivo es crear una herramienta que permita a los usuarios analizar las frecuencias de los archivos de audio mediante la implementación de algoritmos de análisis de frecuencias, y que sea configurable a través de la línea de comandos.

Abstract

Digital signal processing is a discipline that allows a signal to be manipulated so that it can be analysed, modified, or enhanced. This discipline is currently used in various fields, with different types of applications, such as telecommunications, image processing, medicine, among others.

Digital signals are a representation of an analogue signal converted to digital format. A common example of digital signals is audio files. By processing these digital signals, the frequency components present in the audio can be extracted. Through the analysis of these frequency components, valuable information about the audio content can be obtained, such as dominant frequencies, patterns, the presence of noise, among others. One of the ways to analyse the frequencies present in a signal is through the visualisation of its frequency components over time.

In this context, the development of an application in the C language, running in a GNU/Linux environment, is proposed. This application should be capable of processing audio signals and allowing their frequencies to be visualised through a user interface. The objective is to create a tool that allows users to analyse the frequencies of audio files through the implementation of frequency analysis algorithms, and that is configurable via the command line.

Tabla de contenidos

1 Introducción

1.1 Motivación y Propósitos

En la actualidad, el procesamiento de señales digitales (DSP por sus siglas en inglés) se ha convertido en una técnica utilizada en diversos campos, incluyendo telecomunicaciones, electrónica, medicina, entretenimiento y aplicaciones modernas como la detección de voces. Esta técnica permite manipular matemáticamente una señal para analizarla, modificarla o incluso mejorarla.

Para comprender adecuadamente cómo funciona el procesamiento de señales digitales, es crucial entender la diferencia entre una señal digital y una señal analógica. Una señal analógica se representa como una función continua de la amplitud en el tiempo, es decir, la señal se puede visualizar como una forma de onda que varía su amplitud a lo largo del tiempo. En cambio, una señal digital se representa como una secuencia de números, donde cada uno determina la amplitud de la señal en un instante determinado del tiempo.

Las señales digitales son una forma de representar una señal analógica que ha sido convertida en una serie de valores discretos y finitos, lo que permite que puedan ser almacenadas y procesadas por un ordenador. Los archivos de audio son ejemplos de señales analógicas que son convertidas en señales digitales a través de un proceso llamado conversión analógico-digital (ADC), que cuantifica las muestras en valores digitales para su almacenamiento. Para la conversión de las señales analógicas a digitales, hay que tomar en cuenta ciertos factores, como la frecuencia de muestreo, el número de canales, el formato, entro otros. Este tipo de características influyen en la transformación de la señal, por lo que, si no se definen bien, se puede obtener como resultado una señal que no es capaz de representar correctamente a la señal original.

Por otra parte, las señales como los archivos de audio normalmente están representados en el dominio del tiempo, esto significa que se tiene el valor de la amplitud para cada instante de tiempo, lo cual es muy útil para su reproducción, pero no mucho para su análisis. Una de las maneras más comunes de analizar los archivos de audio es mediante sus componentes de frecuencia.

El análisis de frecuencias tiene sus inicios gracias a la invención del espectrograma de sonido en 1946 por Walter Koenig, Hugh K. Dunn y Lacy L. Y. [1]. El espectrograma se refiere a una representación visual de las diferentes variaciones de frecuencias y la intensidad del sonido a lo largo del tiempo.

Para poder analizar los componentes de frecuencia presentes en los archivos de audio y extraer información valiosa de los mismos, hay que representar los archivos en el dominio de la frecuencia discreta, lo que es posible mediante un algoritmo denominado Transformada Discreta de Fourier (DFT por sus siglas en inglés) [2]. Esta representación permite descomponer la señal en sus diferentes componentes de frecuencia, desde las más bajas hasta las más altas, lo que facilita la identificación de las frecuencias presentes en la señal.

No obstante, una de las formas más comunes de analizar las frecuencias presentes en los archivos de audio es mediante su visualización a lo largo del tiempo, lo cual permite apreciar las frecuencias presentes en una señal en diferentes instantes de tiempo, así como su intensidad. Por lo tanto, en este trabajo se busca implementar una aplicación en lenguaje C en un entorno GNU/Linux, la cual pueda procesar señales digitales (archivos de audio), y

mostrar los componentes de frecuencia del audio mediante una interfaz de usuario para su visualización y análisis. Además, se busca que la aplicación sea lo más configurable posible a través de la línea de comandos, permitiendo que el usuario pueda personalizar parámetros que influyen en la visualización de frecuencias y procesamiento del audio.

1.2 Objetivos y Planificación

Para realizar este trabajo de fin de grado se han planteado los siguientes objetivos específicos:

- Estudio y análisis de técnicas de procesado de audio.
- Obtención de los componentes de frecuencia de diferentes tipos de audio mediante la aplicación de la Transformada Rápida de Fourier.
- Visualización de los componentes de frecuencia presentes en los audios a partir de una interfaz de usuario desarrollada en lenguaje C.
- Obtención de las características de los archivos de audio.
- Desarrollo de la aplicación de manera configurable, permitiendo al usuario cambiar o personalizar parámetros para la visualización de frecuencias y el procesamiento del audio.

Para abordar y cumplir los objetivos, se realizó un plan de trabajo estableciendo una lista de tareas. Las tareas se han llevado a cabo de la siguiente manera:

Primero, antes de desarrollar la aplicación, se realizó un estudio preliminar sobre diversos aspectos relacionados con el procesamiento de señales de audio. Esto incluye el aprendizaje sobre los formatos de archivos de audio más comunes, la comprensión de las características que componen las señales digitales, y el análisis de los algoritmos utilizados en el procesamiento y análisis de señales como lo es la Transformada Rápida de Fourier (FFT por sus siglas en inglés) [2].

También, se llevó a cabo un análisis de implementaciones previas relacionadas con señales de audio. Esto permitió identificar posibles soluciones y enfoques que puedan ser adaptados y mejorados en la aplicación. Además, se investigó y analizó diversas bibliotecas que sean relevantes para el procesamiento de señales de audio y la creación de interfaces de usuario en lenguaje C. Se evaluaron sus características y capacidades para seleccionar la opción más apropiada para cumplir los objetivos de la aplicación.

Posteriormente, se empezó el desarrollo de la aplicación, empezando por implementar funcionalidades básicas como cargar diferentes formatos de archivos de audio y extraer las muestras correspondientes, es decir, los valores de amplitud en el tiempo. Después, se aplicó la FFT, la cual es una versión más rápida que la original en cuanto a cantidad de cálculos computacionales. Con esta técnica se extrajeron los componentes de frecuencia de las señales de audio.

Una vez extraídos los componentes de frecuencia de las señales, se procedió al diseño e implementación de la interfaz de usuario. Esta interfaz permite la visualización de las frecuencias de manera intuitiva. Se buscó que la interfaz sea lo más configurable posible, adaptándose a las preferencias del usuario, desde la posibilidad de configurar el tamaño de la ventana hasta la configuración de la visualización de las frecuencias de audio.

Continuamente, mediante el desarrollo de la aplicación se realizaron pruebas para verificar el correcto funcionamiento de todas las funcionalidades implementadas.

1.3 Estructura de la memoria

En el capítulo dos se presentan las tecnologías relevantes para el desarrollo del proyecto hasta la fecha. Esto incluye una revisión de los algoritmos utilizados actualmente para el procesamiento de señales, así como una exploración de las bibliotecas disponibles para el desarrollo de la aplicación. A continuación, en el capítulo tres, se discuten las herramientas que se han considerado y descartado para el desarrollo del proyecto. Posteriormente, en el capítulo cuatro, se detalla el proceso de desarrollo del proyecto, abordando cómo se llevó a cabo, las decisiones tomadas, las limitaciones encontradas y las funcionalidades implementadas. Seguidamente, en el capítulo cinco, se analizan los resultados obtenidos y la eficiencia del proyecto, evaluando en qué medida se han cumplido los objetivos establecidos, así como las posibles direcciones futuras del trabajo. Por último, en el capítulo seis, se explora el impacto potencial del proyecto según los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 [3]. Además, se incluye un capítulo para bibliografía y otro para anexos, en el cual se muestra información como el manual de uso de la aplicación, el código de las funciones principales del programa, y la instalación de las bibliotecas utilizadas en el desarrollo de la aplicación.

2 Estado del Arte

2.1 Técnicas para el Procesado de Señales Digitales

2.1.1 La Transformada Discreta de Fourier (DFT) y la Transformada Rápida de Fourier (FFT)

La transformada discreta de Fourier (DFT) fue introducida por el famoso físico Jean-Baptiste Joseph Fourier en su libro llamado "La Teoría Analítica del Calor" publicado en 1822 [4]. Fourier determinó que era posible descomponer una función como la suma de senos y cosenos hasta lograr determinar la función original. De esta manera, la DFT transforma una secuencia finita de puntos, como las muestras de una señal en el dominio del tiempo, en una secuencia de puntos en el dominio de la frecuencia.

La DFT se define matemáticamente como:

$$x_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i*2\pi*kn/N}$$

Donde N es el número de puntos al que se le va a aplicar la DFT, x_n es el valor de la señal antes de calcular la DFT en el índice n, k es el índice de la DFT que se está calculando e i es el número imaginario.

Como se puede observar, para calcular una DFT de N puntos, se tienen que realizar N sumas y N multiplicaciones complejas para cada valor de x_k. Por lo tanto, el número total de operaciones sería N^2 multiplicaciones complejas y N^2 sumas complejas, dando un total de $2 \cdot (N^2)$ operaciones. Cuando se trata de señales muy grandes, esta gran cantidad de operaciones puede tardar demasiado. No obstante, en 1965 Cooley y Tukey presentaron la Rápida Transformada de Fourier (FFT) [5], la cual es capaz de obtener el mismo resultado que la DFT con solo $NlogN$ operaciones, haciendo que el proceso sea mucho más eficiente. La FFT es una forma de realizar la DFT de manera rápida y eficiente mediante una técnica llamada divide y vencerás.

Para que este algoritmo funcione correctamente, se debería tener un número de muestras de entrada de tamaño N, siendo N potencia de dos; de no ser así, puede afectar el rendimiento de la FFT.

De este modo, la idea del algoritmo es inicialmente dividir las muestras de entrada en dos secuencias separadas, una con los elementos de índice par y otra con los elementos de índice impar. Asimismo, estas dos secuencias se subdividen en más secuencias de índice par e impar, lo cual se realizará continuamente hasta obtener secuencias de un solo elemento, siendo la DFT de un solo elemento el elemento en sí. Llegado a ese punto, se irán combinando las secuencias de las más pequeñas a las más grandes, utilizando coeficientes complejos para realizar las combinaciones de las secuencias en cada etapa del algoritmo, este coeficiente se define como:

$$W_N^k = e^{-i \cdot 2\pi \cdot k/N}$$

Donde N es el número de muestras a las que se le quiere aplicar la transformada, k es el índice de la secuencia que se está calculando, i es la unidad imaginaria $i = \sqrt{-1}$, y e es la base del logaritmo natural [6].

Por ejemplo, suponiendo que se tiene una secuencia de muestras de entrada de longitud 8:

$$x = [\, x0, x1, x2, x3, x4, x5\, , x6, x7]$$

Se dividirán los elementos de índice par e índice impar en dos secuencias separadas:

$$x_{par} = [\, x0, x2, x4, x6]$$

$$x_{impar} = [\, x1, x3, x5, x7]$$

Se irá subdividiendo cada secuencia en dos hasta que se tenga una única muestra en cada una:

$$x_{par,par} = [x0, x4]$$

$$x_{par,impar} = [x2, x6]$$

$$x_{impar,par} = [x1, x5]$$

$$x_{impar,impar} = [x3, x7]$$

Una vez se calcule la DFT de cada muestra individualmente, se combinarán las secuencias de menores a mayores:

$$X0 = xo + W_8^0 \cdot x4 \quad X4 = x0 - W_8^0 \cdot x4$$

$$X2 = x2 + W_8^2 \cdot x6 \quad X6 = x6 - W_8^2 \cdot x6$$

$$X1 = x1 + W_8^1 \cdot x5 \quad X5 = x1 - W_8^1 \cdot x5$$

$$X3 = x3 + W_8^3 \cdot x7 \quad X7 = x3 - W_8^3 \cdot x7$$

Como se puede observar, como los elementos X0 y X4 estaban en la misma secuencia, se irán combinando, y lo mismo con el resto de los elementos. Esto se realizará continuamente hasta combinar todas las sucesiones utilizando la siguiente fórmula:

$$X(k) = X_{par}(K) + W_N^k \cdot\ X_{impar}(k)$$

$$X(k + N/2) = X_{par}(K) - W_N^k \cdot\ X_{impar}(k)$$

Debido a que no se sabe la cantidad de muestras que pueden tener los archivos de audio que se analicen en la aplicación y debido a su eficiencia, se consideró la FFT como la opción más adecuada para obtener las frecuencias presentes en los audios.

Por otra parte, de la misma manera que la Transformada Discreta de Fourier (DFT) permite descomponer una señal en sus componentes de frecuencia a partir de sus valores de amplitud, también es posible recuperar los valores de amplitud de una señal a partir de sus componentes de frecuencia. Esto se logra mediante la Transformada Inversa de Fourier. De este modo, la fórmula para calcular la Inversa es:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k \cdot\ e^{i \cdot 2\pi \cdot k \cdot n/N}$$

Donde x_n es el valor de amplitud de la señal en el índice n, x_k es el valor de la señal en el dominio de la frecuencia en el índice k y N es el número total de muestras de la señal [6].

No obstante, se puede destacar que, aunque la FFT fue presentada por Cooley y Tukey en 1965, sus inicios se pueden remontar mucho antes, específicamente en 1805 gracias al famoso físico Carl Friedrich Gauss. En 1805, Gauss trabajó en métodos para la interpolación trigonométrica que implicaba una técnica similar a la descomposición utilizada en la FFT. Sin embargo, sus métodos no fueron publicados hasta mucho después. Por lo que se cree que, gracias a esos avances y métodos desarrollados por Gauss, se sentaron las bases para el desarrollo posterior de la FFT [7].

2.1.1.1 Algoritmos y variantes para calcular la FFT

Existen diferentes tipos de algoritmos para realizar el cálculo de la FFT según las necesidades o el tipo de transformada que se desea calcular [8]. Entre ellos están:

2.1.1.1.1 Radix

Este algoritmo es una versión del algoritmo de Cooley y Tukey, el cual se usa normalmente cuando la longitud de la secuencia a transformar es potencia de dos, basándose en la regla divide y vencerás. Este se subdivide en más algoritmos para calcular FFT de diferentes dimensiones como radix-3 y radix-4.

2.1.1.1.2 Radix Mixto

Es una variante del algoritmo de Cooley y Tukey que permite dividir la secuencia en factores primos pequeños, en lugar de solo potencias de dos, lo cual puede ser más eficiente para cuando se quiere aplicar la transformada a longitudes que no son potencia de dos.

2.1.1.1.3 Split-Radix

Este algoritmo es otra versión del algoritmo de Cooley y Tukey, el cual se basa en dividir la secuencia de longitud N en tres subsecuencias: una de longitud N/2 y dos de longitud N/4. Se calcula la DFT de cada subsecuencia aplicando recursivamente el algoritmo de Split-Radix hasta obtener un solo elemento en cada una.

También existen algunas variantes de la DFT, como la Transformada del seno y del coseno [9], las cuales se utilizan para señales reales, debido a que emplea solo números reales para su cálculo, lo que significa que no hay necesidad de calcular las partes imaginarias que se calculan en la DFT. Por consiguiente, esto reduce a la mitad la cantidad de cálculos necesarios en comparación con la DFT.

Por lo que se puede ver que, dependiendo de la transformada a realizar, existen diferentes implementaciones de algoritmos que realizan un cálculo más eficiente. No obstante, a partir de todas ellas se puede conseguir los componentes de frecuencia de la señal.

2.2 Implementaciones existentes

2.2.1 Musializer

Alexey Kutepov, ha desarrollado una aplicación software llamada Musializer en C [10], esta aplicación permite grabar archivos de audio y visualizar música en tiempo real, por medio de la visualización de los componentes de frecuencia de diferentes formatos de archivos de audio como: WAV, OGG, MP3, QOA, XM, y MOD. Para su implementación, utiliza la biblioteca Raylib para el procesamiento del audio y visualización de frecuencias, y la biblioteca Miniaudio para la grabación y creación de archivos de audio.

Aunque Musializer se centra en la visualización de archivos de música en lugar de su análisis, en su implementación se pueden apreciar técnicas para el procesamiento de audio. Esto incluye la carga de diferentes tipos de formato de audio, las decisiones tomadas para su procesamiento y visualización, y las técnicas utilizadas para el desarrollo de interfaces en C. Por lo tanto, esta aplicación se utilizó para evaluar y mejorar las técnicas de procesamiento de audio en C, y el enfoque del desarrollo de la interfaz en C.

2.3 Tecnologías relevantes

2.3.1 Lenguaje de Programación C

El lenguaje de Programación C es uno de los lenguajes pioneros en la historia de la informática. Este fue desarrollado en la década de 1970 en los Laboratorios Bell por Dennis Ritchie. El lenguaje de C está caracterizado por una mayor flexibilidad y control de hardware en comparación con otros lenguajes de programación, por lo que puede ser de utilidad en diversas aplicaciones. Sin embargo, aunque el lenguaje C ofrece algunas características de lenguajes de alto nivel, no está enfocado en la creación de interfaces o programas más intuitivos a nivel de usuario.

2.3.2 Raylib

Raylib [11], es una biblioteca de código abierto diseñada para el desarrollo de juegos y aplicaciones multimedia en lenguajes como C y C++. Lo que distingue a Raylib es su enfoque en la simplicidad y eficiencia. Esta biblioteca ofrece diferentes funcionalidades como procesamiento de archivos de audio de diferentes formatos como WAV, MP3, OGG, XM y MOD. También permite la creación de interfaces de usuario, la gestión de ventanas y entrada, como la detención y procesamiento de entrada del teclado y ratón, proporciona capacidades de renderizado tanto en 2D y 3D, ofrece soportes para detectar colisiones entre objetos, y es compatible con una variedad de plataformas como Windows, MacOS, Linux, iOS y Android.

En cuanto al procesamiento de audio, la biblioteca ofrece funcionalidades muy básicas como cargar archivos de audio, obtener sus muestras a partir de su reproducción y operaciones como pausar el audio, cambiar su volumen, tono (pitch), etc. Sin embargo, no proporciona funcionalidades como crear nuevos archivos de audio, acceder directamente a las muestras de audio, conversión de formatos de audio o decodificación del audio.

2.3.3 Miniaudio

Miniaudio [12], es una biblioteca enfocada en el procesamiento de audio, esta permite la decodificación de archivos de audio, su reproducción, obtención de muestras, codificación de audios y muchas otras funcionalidades para el manejo del audio. Esta biblioteca se caracteriza por ofrecer un control bajo nivel del audio, permitiendo mayor control sobre el sonido.

Por otra parte, esta biblioteca utiliza otras API de bajo nivel para acceder al hardware del audio, independientemente de la plataforma en que se esté ejecutando. Por ejemplo, utiliza WASAPI en Windows, ALSA en Linux y Core Audio en MacOS.

2.3.4 PortAudio

PortAudio [13], es una biblioteca que ofrece una API para el procesamiento de audio. Provee una interfaz que se conecta con API de audio nativas mediante llamadas internas a las API de audio para su implementación. Algunas de las API utilizadas por esta biblioteca son: WMME, ASIO, DirectSound, WDM/KS, WASAPI, OSS, entre otras.

Esta biblioteca ofrece diversas funcionalidades para la reproducción, grabación y manipulación de audio. Sin embargo, algunas de sus funcionalidades pueden estar limitadas por las otras API de audio nativas que utiliza para su implementación. Por ejemplo, si se quiere convertir la frecuencia de muestreo de un audio, y esta frecuencia de muestreo no es compatible con la API de audio nativa, no se podrá realizar la conversión, o si se quiere abrir más de un dispositivo a la vez, no es posible debido a que la API ASIO no proporciona esta funcionalidad.

2.3.5 FFT GNU GSL

La FFT de GNU GSL (GNU Scientific Library) [14], es una implementación de la transformada rápida de Fourier escrita en C que ofrece la biblioteca GNU GSL. Esta biblioteca incluye funcionalidades para calcular la FFT de datos complejos y datos reales, con longitudes que pueden ser o no potencia de dos.

Para el cálculo de la FFT, esta biblioteca emplea algoritmos como “Radix-2”, utilizado comúnmente cuando la longitud de datos es una potencia de dos, y “Radix Mixto”, utilizado normalmente con FFT donde la longitud de datos no es una potencia de dos.

Por otra parte, la implementación del algoritmo “Radix Mixto” para calcular la FFT en GNU/GSL es una reimplementación del escrito en la biblioteca FFTPACK [15]. FFTPACK es una biblioteca desarrollada por Paul Swarztrauber en el Centro Nacional para la Investigación Atmosférica (NCAR por sus siglas en íngles) en 1982, la cual se basa en un conjunto de subrutinas para calcular la FFT escritas en Fortran [16] (Lenguaje de programación de alto nivel desarrollado por IBM en 1957, especializado en cálculo numérico y computación científica), incluyendo también el cálculo de las transformadas del seno y coseno que no están incluidas en la implementación de FFT de GSL.

2.3.6 FFTW3

La FFTW3 (Fastest Fourier Transform in the West versión 3) [17], es una biblioteca de software libre y de código abierto distribuida por GNU GPL y desarrollada por Matteo Frigo y Steven G. Johnson, para calcular transformadas rápidas de Fourier (FFT) de manera eficiente, adaptándose a la arquitectura hardware de donde se emplea [18]. La FFTW optimiza el algoritmo para calcular la DFT según los detalles del hardware de la máquina para maximizar el rendimiento. Para realizar eso, crea una estructura llamada “planificador” que prueba todas las posibles formas de cálculo de la FFT en la máquina para obtener cuál es la más rápida en términos de ejecución. Para esto, se emplea un algoritmo dinámico que divide la transformada en transformadas más pequeñas y trata de encontrar la mejor combinación según la arquitectura de la máquina. Este planificador produce una estructura de datos llamada “plan” que contendrá la información obtenida por el planificador para realizar el mejor cálculo de la FFT. Una vez creado el “plan” para el cálculo de la FFT, este se

puede utilizar tantas veces como sea necesario. Gracias a esto se pueden calcular varias transformadas del mismo tamaño con solo una inicialización.

Por otra parte, la FFTW es capaz de calcular transformadas de datos complejos y reales con longitud variable, de diferentes dimensiones (unidimensionales para señales temporales, bidimensionales para imágenes, etc.) y transformadas de seno y de coseno.

3 Herramientas Empleadas

3.1 Lenguaje de Programación C.

Como ya se ha mencionado anteriormente en el capítulo “Estado del arte” el lenguaje de programación C es reconocido por su capacidad de ofrecer un control de hardware y software. Sin embargo, debido a que no está orientado específicamente al desarrollo de interfaces de usuario o aplicaciones gráficas, puede hacer un poco más difícil el desarrollo de aplicaciones donde se quiera crear una interacción más visual con el usuario. No obstante, se ha decidido utilizar este lenguaje para el desarrollo de la aplicación debido a que, aunque no es la opción más empleada para el desarrollo de interfaces, puede ser una ventaja significativa en términos de mejora de código y rendimiento.

3.2 Entorno de Desarrollo GNU/Linux

GNU/Linux es un conjunto de sistemas operativos tipo Unix, compuesto por software libre. GNU fue creado por Richard Stallman en 1983, con el objetivo de crear un software libre. Para que el sistema estuviera completo, en 1990 se empezó él desarrolló un kernel llamado HURD, el cual tuvo un desarrollo lento y no se consideraba todavía listo para su uso. Posteriormente, en 1991, Linus Torvalds desarrolló un kernel compatible con Unix llamado Linux. En 1992 se combinó el kernel Linux con el sistema GNU, creando un sistema completo y libre [19].

La elección de desarrollar la aplicación en un entorno GNU/Linux se justifica por diversas razones. En primer lugar, GNU/Linux proporciona un entorno de desarrollo altamente compatible con el lenguaje C, lo que facilita la compilación y depuración del código. Además, la disponibilidad de herramientas de desarrollo de código abierto disponibles en GNU/Linux simplifica el proceso de desarrollo y reduce la probabilidad de problemas relacionados con la compatibilidad de las bibliotecas.

3.3 Bibliotecas Utilizadas y Descartadas

3.3.1 Raylib

Raylib se destaca por su facilidad de uso, especialmente para proyectos más pequeños, lo que la convierte en una opción adecuada para el desarrollo de la interfaz de usuario en la aplicación. Sin embargo, aunque ofrece funcionalidades para el procesamiento de archivos de audio, su alcance en este aspecto es limitado en comparación con otras bibliotecas como Miniaudio o PortAudio.

Por otro lado, existen otras bibliotecas para el desarrollo de interfaces como SFML (Simple and Fast Multimedia Library) y SDL (Simple DirectMedia Layer), las cuales son ampliamente utilizadas en el desarrollo de aplicaciones y videojuegos. No obstante, debido a que Raylib es fácil de usar y ofrece las funcionalidades necesarias para el desarrollo de proyectos pequeños, se optó por su utilización para el desarrollo de la interfaz.

3.3.2 Miniaudio

Inicialmente, se consideró la utilización de Raylib para la implementación del procesamiento del audio en la aplicación. Sin embargo, tras un análisis detallado de las funcionalidades proporcionadas por ambas bibliotecas, se ha tomado la decisión de emplear Miniaudio. Esta elección se basa en la capacidad de Miniaudio para proporcionar un control más profundo y detallado sobre el audio, en comparación con las capacidades limitadas de procesamiento de audio ofrecidas por Raylib.

Por otra parte, también se han considerado otras opciones que ofrecen las mismas capacidades de procesado de audio como PortAudio. Sin embargo, se consideró que Miniaudio es la mejor opción para este trabajo debido a que ofrece una API fácil de entender y usar. También, se puede destacar que su arquitectura ofrece una mayor flexibilidad y control sobre los aspectos técnicos, como lo son la decodificación, creación, reproducción, manipulación y conversión del sonido, lo que resulta fundamental para una aplicación centrada en el análisis de archivos de audio.

3.3.3 FFTW3

Para el cálculo de la Rápida Transformada de Fourier se decidió utilizar la biblioteca FFTW3 de código abierto distribuida por GNU GPL, debido a que en comparación con la FFT de la librería GNU GSL para cálculos a gran escala se recomienda la FFTW. El funcionamiento de esta biblioteca se basa en la inicialización previa de un algoritmo para el cálculo de la FFT denominado “plan”, que contendrá la configuración para calcular la FFT de manera más rápida en cuanto a tiempo de ejecución según el hardware de la plataforma que lo utilice. De esta manera, el cálculo de la FFT será más rápido, ofreciendo un mayor rendimiento. Este plan puede ser continuamente llamado y reutilizado hasta su destrucción.

Por otra parte, la creación de un plan más optimizado según el hardware de la máquina puede llevar tiempo en inicializarse, por lo que la biblioteca también ofrece la posibilidad de especificar las preferencias al crear un plan que permita ejecutar la FFT. De este modo, se pueden crear planes de manera más rápida, pero con menor optimización de cálculo, o crear planes de manera un poco más lenta, que serán más óptimos para el cálculo de la FFT.

4 Desarrollo

4.1 Requisitos

Para llevar a cabo el desarrollo de la aplicación, se han definido los requisitos funcionales y no funcionales que esta debe cumplir. Entre ellos están:

4.1.1 Requisitos Funcionales:

- La aplicación debe permitir la carga de diferentes formatos de archivos de audio comunes (por ejemplo, WAV, MP3, FLAC).
- La aplicación debe extraer muestras de audio (valores de amplitud en el tiempo) de los archivos de audio.
- La aplicación debe aplicar la Transformada Rápida de Fourier (FFT) a las muestras de audio para obtener las componentes de frecuencia presentes.
- La aplicación debe proporcionar una interfaz gráfica que permita visualizar las frecuencias de audio en el tiempo.
- La aplicación debe ser configurable a través de la línea de comandos, permitiendo al usuario ajustar parámetros como el tamaño de ventana de visualización y los rangos de frecuencia mostrados.

4.1.2 Requisitos No Funcionales

- La aplicación debe ser capaz de procesar archivos de audio de manera eficiente, así como también, ser capaz de ejecutar la FFT de grandes cantidades de muestras de audio sin retrasos notables.
- Debe incluir documentación y ayuda para guiar a los usuarios en el uso de la aplicación.

4.2 Archivos de Audio

Como se mencionó anteriormente en la introducción, los archivos de audio son señales analógicas convertidas en señales digitales. La conversión de una señal analógica en una señal digital implica tomar muestras de la señal analógica en intervalos regulares (muestreo), asignar valores numéricos a estas muestras (cuantificación) y codificar estos valores en formato binario. Para llevar a cabo esta conversión de manera correcta, se tienen que tomar en cuenta una serie de características que permitirán obtener una representación equivalente de la señal original. Estas características son:

4.2.1 Frecuencia de Muestreo

Esta determina el número de muestras por segundo que se deben tomar de la señal analógica para convertirla en una señal digital. Según el teorema de Nyquist-Shannon, la frecuencia utilizada para muestrear una señal debe ser al menos el doble de la frecuencia máxima presente en la señal analógica para evitar la pérdida de información durante el proceso de muestreo [20]. En caso de que esto no sea así, se produce un fenómeno denominado Aliasing.

El Aliasing se produce cuando no se toman la cantidad suficiente de muestras de la señal para su recreación. Por ejemplo, si se tiene una señal analógica cuya frecuencia máxima es de 5 kHz, para muestrear la señal de manera adecuada según el teorema de Nyquist-Shannon se necesitaría una frecuencia de muestreo de al menos 10 kHz (el doble de la frecuencia máxima de la señal). No obstante, si se muestrea la señal con una frecuencia de muestreo de 6 kHz, no se estarían tomando suficientes muestras para capturar adecuadamente las frecuencias presentes en la señal original. Como resultado, algunas de estas frecuencias se "aliarán" durante el proceso de muestreo, lo que dará lugar a componentes de frecuencia falsos en la señal digital.

4.2.2 Tamaño de la Muestra

Es el número de bits utilizados para almacenar los valores de amplitud de manera digital. El número de bits puede ser 8, 16, 24 o 32 bits. Cuanto mayor sea el número de bits utilizados para almacenar las muestras de audio, mayor cantidad de valores se podrán representar, y, por ende, mejor será la calidad del audio.

4.2.3 Número de Canales

Es la cantidad de fuentes de audio individuales en las cuales se graba o reproduce el sonido. Cada canal contiene muestras de audio separadas, es decir, que puede contener diferentes sonidos de la misma grabación. Normalmente, los audios suelen ser Mono (un solo canal) y estéreo (dos canales), y para formatos de audio más avanzados puede haber múltiples canales.

4.2.4 Formato del Audio

Define como se almacenan los datos digitalmente, entre los formatos más utilizados se encuentran: WAV, MP3, FLAC, OGG, entre otros.

Por otra parte, es importante destacar que los audios están formados por frames. Un frame se refiere a cada muestra o conjunto de muestras que posee el audio, es decir, si se tiene un audio de tipo estéreo (dos canales), cada frame estará formado por dos muestras, una para el canal izquierdo y otra para el canal derecho, mientras que si se tiene un audio de tipo mono (un canal), cada frame solo contendrá una muestra.

4.3 Procesamiento del Audio

Como se ha mencionado anteriormente, para el procesamiento de audio en la aplicación se optó por utilizar la biblioteca Miniaudio.

Miniaudio soporta varios formatos de tamaño de muestra. Los tamaños de muestra soportados son:

- ma_format_u8: Muestras de 8 bits sin signo.
- ma_format_s16: Muestras de 16 bits con signo.
- ma_format_s24: Muestras de 24 bits con signo.
- ma_format_s32: Muestras de 32 bits con signo.
- ma_format_f32: Muestras de 32 bits en punto flotante.

Por otra parte, a través de Miniaudio también se pueden decodificar archivos de audio para leer directamente sus datos. Los formatos de audios soportados para decodificar son los siguientes: WAV, MP3, FLAC.

Debido a esto, la aplicación solo acepta archivos de audio que tengan un tamaño de muestras de 8, 16, 24, 32 o 32 bits en punto flotante, que sean de tipo WAV, MP3 o FLAC y que sean mono o estéreo.

4.3.1 Decodificación del Audio

Para procesar el audio en la aplicación lo primero que se hace es decodificarlo, lo cual permite obtener las características originales del mismo. En Miniaudio es posible la decodificación del audio mediante la API ma_decoder. A través de la decodificación se pueden leer los parámetros del audio como su frecuencia de muestreo, número de canales, tamaño de muestra y número total de frames, así como también, extraer las muestras presentes en el audio.

4.3.2 Reproducción del Audio

Después de decodificar el audio, se pasa a su reproducción. Para esto se empleó una API de alto nivel de Miniaudio llamada motor (engine). Un motor permite cargar y reproducir sonidos con mayor control y flexibilidad, debido a que aparte de reproducir el sonido, permite realizar modificaciones en tiempo real sobre el sonido. Por ejemplo, permite cambiar el volumen del sonido, cambiar el tono (pitch), e incluso adelantarlo o atrasarlo.

Cuando se crea una engine, se crea automáticamente un dispositivo para la reproducción o captura del audio en la máquina. Sin embargo, en la aplicación se creó un dispositivo utilizando la API ma_devide, debido a que se prefirió configurar manualmente cómo se desean procesar los audios en lugar de utilizar la configuración predeterminada de la engine.

Para configurar el dispositivo, se tienen que establecer parámetros como: El tipo de dispositivo que se desea crear (si es dispositivo de reproducción o grabación), la frecuencia de muestreo con la que se obtendrán o reproducirán las muestras, el tamaño de las muestras, el número de canales con que se quiere leer o reproducir el audio y especificar el nombre de una función denominada “callback”. A través de esta función, Miniaudio envía las muestras extraídas del audio en tiempo real. De esta manera, mediante la reproducción del audio se podrán leer las muestras en tiempo real a través de la función “callback”. Las muestras estarán convertidas al formato especificado en la configuración del dispositivo.

En este caso, se configuró el dispositivo de reproducción utilizando la frecuencia de muestreo del audio original obtenida mediante la decodificación, estableciendo el tamaño de muestra a 32 bits de punto flotante, 2 canales de audio y una función callback denominada “data_callback”. Cuando se reproduce el audio, Miniaudio envía las muestras por la función “data_callback”, las cuales son previamente convertidas en el formato establecido al configurar el dispositivo, es decir, se obtienen las muestras del audio por dos canales y en formato punto flotante.

En el caso de que el número de canales del audio original es uno (mono), Miniaudio duplica el canal, enviando como canal izquierdo y canal derecho los mismos valores de muestras. En el caso de audios con dos canales (esteréo), no

hay modificación, se reciben por la función los dos canales de audio como estaban inicialmente.

Una vez creado el dispositivo, se crea el objeto ma_engine, pasándole el dispositivo ya creado y configurado. Continuamente, se crea un objeto ma_sound, el cual se asocia con la engine y el archivo de audio que se desea reproducir. Creado el objeto ma_sound, se puede iniciar la reproducción del audio.

Cuando se inicia la reproducción del audio, Miniaudio envía las muestras a través de la función callback especificada en el dispositivo. Esta función debe tener el siguiente formato:

```
void callback (ma_device* pDevice, void* pOutput, const void* pInput,
ma_uint32 frameCount)
```

El parámetro frameCount especifica el número de frames que pueden ser escritas en pOutput y leídas en pInput. En este caso, como se configuró el dispositivo a dos canales, se obtienen dos muestras por cada frame, es decir, que el número de muestras que se pueden leer y el tamaño del búfer (espacio de memoria para almacenar datos) pOutput y pInput es el doble que frameCount. Las muestras están ubicadas una al lado de otra, siendo las muestras de índice par correspondientes al canal izquierdo y las de índice impar al canal derecho. Por otra parte, el tamaño en bytes para cada muestra es el mismo que se definió en la configuración del dispositivo, es decir, punto flotante.

Dado que se ha creado un dispositivo de reproducción, para poder obtener las muestras de audio en tiempo real, hay que leer tantas frames como especificadas en frameCount y almacenarlas en pOutput. Esto es posible mediante la función:

```
ma_engine_read_pcm_frames(&pDevice->pUserData, pOutput, frameCount,
NULL);
```

Con las muestras almacenadas en pOutput, es posible acceder a ellas para transformarlas y obtener sus componentes de frecuencia.

4.3.3 Obtención de Frecuencias de Audio

4.3.3.1 Propiedad de la FFT

Como se mencionó antes en el capítulo “Estado del Arte”, la Transformada Rápida de Fourier (FFT) es un algoritmo utilizado para calcular la Transformada Discreta de Fourier (DFT) y su inversa de manera eficiente. La DFT es una herramienta importante en el procesamiento de señales, debido a que transforma una secuencia de valores típicamente en el dominio del tiempo, en una secuencia en el dominio de la frecuencia, representando los componentes de frecuencias de la señal.

Una propiedad clave de la DFT, especialmente cuando se trabaja con señales reales, es la simetría. Esta propiedad implica que, para una señal real, los

componentes de frecuencia negativa son el conjugado complejo de los componentes de frecuencia positiva correspondientes. Matemáticamente, esto se expresa como:

$$X[k] = \overline{X[N-k]}$$

Donde $\overline{X[N-k]}$ representa el conjugado complejo de $X[N-k]$ y N representa el número de puntos calculados. Esto implica que si se realiza una FFT de N puntos, la información de la señal estará contenida en los primeros $(N+1)/2$ valores de $X[k]$.

4.3.3.2 Empleo de la FFT

Para obtener las frecuencias de audio en tiempo real, se utilizan las muestras que envía Miniaudio a través de la función "callback". Como el número de frames que se pasan por la función no está determinado, se implementó un búfer circular donde se almacenan las muestras. Es decir, se creó un búfer con un tamaño predeterminado de 8192, en el cual se escriben las muestras a medida que van llegando por la función "callback". Cuando se llegue al final del búfer, la siguiente muestra se sobrescribe al inicio del búfer y así sucesivamente. Las muestras que se escriben en el búfer corresponden a las muestras de un solo canal, ya sea el izquierdo o el derecho. Por defecto, la aplicación está configurada para leer las muestras del canal izquierdo, pero esto se puede modificar por la línea de comandos antes de ejecutar la aplicación (véase el manual de usuario en el capítulo "Anexos").

Cada vez que el búfer esté lleno, se le aplica una FFT para obtener las frecuencias presentes en ese conjunto de muestras, es decir que, si el tamaño del búfer es 8192, se aplica la FFT a cada 8192 muestras que van llegando por la función "callback". El tamaño del búfer también se puede modificar a través de la línea de comandos antes de ejecutar la aplicación (véase el manual de usuario en el capítulo "Anexos"). Si se cambia el tamaño del búfer, por defecto también se cambiaría el tamaño de la FFT utilizada para obtener las frecuencias. Como resultado, el número de frecuencias obtenidas y visualizadas simultáneamente en la interfaz cambiaría. En el capítulo "Anexos" se puede ver el código correspondiente a la función "callback", donde se guardan las frecuencias de audio y se aplica la FFT.

Por otra parte, antes de empezar a reproducir el audio se inicializa un plan para calcular la FFT. La biblioteca FFTW3 utiliza un sistema de "planes" para optimizar la ejecución de la FFT. Los planes, como ya se explicó en el capítulo "Estado del Arte", son estructuras configuradas según el hardware de la máquina donde se ejecuta, lo que permite ejecutar las transformadas de manera más rápida en cuanto a tiempo de ejecución. Cuando se quiere calcular una FFT, la biblioteca optimiza el cálculo basándose en el hardware de la máquina y almacena la configuración para ejecutarlo en una estructura llamada "plan". Después, para calcular la FFT, lo único que se tiene que hacer es ejecutar el plan creado.

A la hora de crear un plan, se establece un flag que permite especificar las preferencias que se tengan. Por ejemplo, si se usa como flag "FFTW_ESTIMATE", se crea un plan de manera rápida, pero no el más eficiente posible, es decir, con este flag la biblioteca no se enfoca tanto en optimizar el plan según el hardware de la máquina, haciendo que la inicialización del plan sea más rápida, pero su configuración no la más eficiente.

En caso contrario, si se utiliza como flag “FFTW_MEASURE”, la biblioteca se enfocará más en optimizar el plan según el hardware de la computadora, por lo que tardará más en crearse, pero el cálculo de la FFT proporcionado por el plan será más óptimo.

Por consiguiente, dado que las muestras se obtienen en datos reales, se optó por utilizar un plan que reciba como entrada una señal de datos reales y devuelva sus componentes de frecuencia en valores complejos. La FFT que se ha empleado tiene un tamaño igual al tamaño del búfer de entrada de muestras. También, es importante destacar que el plan de la FFT se inicializa antes de empezar a reproducir el audio, y que este se utiliza continuamente cada vez que el búfer donde se guardan las muestras este lleno. Por lo tanto, se optó por utilizar como flag “FFTW_MEASURE”, para obtener el cálculo más eficiente posible.

En concreto, la función de la biblioteca que se utiliza para crear el plan para el cálculo de la FFT es la siguiente:

```
fftw_plan fftw_plan_dft_r2c_1d(int n, double *in, fftw_complex *out,
unsigned flags);
```

Como se puede ver, esta función te crea un plan para el cálculo de la FFT pasándole 4 parámetros. El primero, “n” especifica el tamaño de la FFT que se desea calcular. El segundo, “in”, es un puntero (dirección de memoria en C) a la lista en donde se tendrán las muestras de entrada, es decir, los valores de amplitud de la señal. El tercer parámetro es otro puntero a una lista de salida en donde la biblioteca podrá almacenar las frecuencias calculadas, y el último es el flag que especifica como se desea crear el plan.

Es importante destacar que las listas “in” y “out” deben estar inicializadas antes de la creación del plan y deben tener la capacidad de almacenar “n” valores. Por otra parte, como se puede ver, la variable “out” en la especificación de la función es de un tipo complejo que pertenece a la biblioteca FFTW3. Debido a esto, la biblioteca ofrece funciones propias equivalentes a malloc en C para la inicialización de este tipo de datos. Sin embargo, estos valores de tipo “fftw_complex” son equivalentes a los valores de tipo “double complex” en C, y se puede sustituir “fftw_complex” por “double complex” si se incluye la declaración de la biblioteca “complex.h” antes de la biblioteca “fftw3.h” en el código de C.

Una vez creado el plan, es posible ejecutar la FFT simplemente llamando al plan de la siguiente manera:

```
fftw_execute(plan).
```

Esta función realizará el cálculo de la FFT a partir de los valores que ya deben estar almacenados en la lista “in”. Esta función puede ser llamada múltiples veces para calcular la FFT de varios conjuntos de muestras de tamaño “n” con solo cambiar las muestras escritas en la lista “in”.

Cuando no se quiere utilizar más el plan, solo hace falta destruirlo mediante la función:

```
fftw_destroy_plan(plan).
```

Por otra parte, existen otros flags que permiten una mayor optimización. Sin embargo, los únicos dos que se utilizan en la aplicación son los ya mencionados. Para más información, véase la documentación de la biblioteca [17].

4.4 Desarrollo de la Interfaz de Usuario

4.4.1 Creación de la Ventana de Usuario

Después de procesar correctamente el audio y crear el plan para calcular la Transformada Rápida de Fourier (FFT), se crea una interfaz de usuario utilizando la biblioteca Raylib. Esta interfaz cuenta con un botón en la parte superior izquierda de la ventana para detener o reanudar la reproducción del audio, y otro para recortar el audio (este permitirá guardar las frecuencias de audio en un archivo, lo cual será explicado con mayor detalle en el capítulo "Almacenamiento de Frecuencias de Audio"). En la esquina superior derecha se encuentran cuatro botones: dos de ellos para ajustar el volumen, y los otros dos botones para aumentar y disminuir el tono del audio (pitch).

Estos botones se implementan utilizando las funciones de Raylib para detectar el ratón en la ventana.

Por otra parte, Raylib también ofrece funcionalidades para controlar la entrada a través del teclado, lo que permite ajustar estos valores utilizando el teclado. Con tan solo presionar las teclas "+" y "-" es posible disminuir y bajar el volumen, las teclas de flecha arriba y flecha abajo para aumentar o disminuir el tono, tecla enter para pausar o reanudar el audio y tecla de escape para cerrar la aplicación.

4.4.2 Visualización de Frecuencias

Después de esta configuración inicial, se muestran las frecuencias de audio a través de la ventana. Estas frecuencias se obtienen a partir del búfer de salida después de aplicar la FFT.

Como ya se mencionó antes en la sección "Propiedad de la FFT", la mitad de los valores obtenidos después de realizar la FFT contienen toda la información necesaria. Por lo tanto, en la ventana de usuario solo se muestran la mitad de las frecuencias obtenidas. Es decir, si se realizó una FFT de N puntos, se muestran solo los $(N+1)/2$ primeros puntos. No obstante, antes de ejecutar el programa, el usuario puede ajustar el número de frecuencias mostradas en la ventana, es decir, establecer un rango dentro del intervalo 0 a (N+1) /2, que será el rango de frecuencias a visualizar en la interfaz.

Por otra parte, antes de dibujar las frecuencias se calculan sus valores correspondientes en decibelios (dB). Para esto se utiliza la fórmula que implica calcular la raíz cuadrada de la suma de los cuadrados de los valores reales e imaginarios, seguido por la aplicación de un logaritmo en base 10 y la multiplicación por 20:

$$db = 20 \times log10(\sqrt{real(z)^2 + imag(z)^2})$$

Siendo z una frecuencia de valor complejo, $real(z)$ la parte real de z, $imag(z)$ la parte imaginaria de z y db el valor en decibelios de dicha frecuencia.

Después de obtener los valores de frecuencia en dB, se calcula la frecuencia máxima presente en el conjunto para normalizar los valores de amplitud. El siguiente paso es representar visualmente estas muestras en la ventana. Para esto, primero se calcula el ancho de cada muestra dividiendo el número total de muestras por el ancho de la ventana.

Antes de dibujar las frecuencias en la ventana, se normalizan las frecuencias. Esto significa que se ajustan los valores de frecuencia para que estén dentro de un rango específico y sean proporcionales al tamaño de la ventana. Esta normalización garantiza que las amplitudes se muestren correctamente en la pantalla, independientemente del rango original de valores. Para su normalización, las frecuencias se dividen entre la frecuencia máxima calculada anteriormente.

Una característica de la representación visual es el efecto de suavizado aplicado a los valores de amplitudes de las frecuencias. Este efecto mejora la visualización al ajustar cada valor de amplitud suavemente según la frecuencia anterior dibujada. En términos prácticos, esto significa que los valores de amplitud no cambian bruscamente de un instante a otro, sino que se suavizan gradualmente, lo que proporciona una representación más natural y fluida de las frecuencias de audio.

Para la realización de este suavizado se ha seguido la implementación propuesta por “Musializer” en su visualización de frecuencias [21]. Para esto, se restan los valores representados anteriormente con los valores a representar, así se obtiene la diferencia entre los valores de amplitud, esta diferencia se multiplica por el tiempo que ha pasado desde que Raylib actualizó la ventana de visualización y un índice de suavidad que cuanto menor sea, menor será el cambio brusco entre una trama y otra, este índice de suavidad se estableció a 8 y a este resultado se le suma el valor anterior de amplitud representada.

Por ejemplo, si se están representando las frecuencias en bloques de 4096, se tendrán dos listas, una con los valores de frecuencias visualizados en la iteración anterior y otra con los nuevos valores de frecuencia a representar. Imaginando que los valores de frecuencia que se quieren representar están almacenados en una lista denominada “F” de la siguiente forma:

$$F = [\, x_0, x_1, x_2, x_3, \quad \dots., x_{4096}].$$

$$x_0 = 2, \quad x_1 = 3, \quad x_2 = 2.5\,, \quad x_3 = 1.5, \dots., \quad x_{4096} = 4.$$

Entonces, en la primera iteración, estos valores se suavizarán basándose en los representados en la iteración anterior. Imaginando que los valores anteriores están guardados en una lista denominada “S”. Debido a que no se ha representado ningún valor de frecuencia por ser la primera iteración, todos los valores contenidos en “S” valdrán cero. S estaría formada de la siguiente manera:

$$S = [\, y_0, y_1, y_2, y_3, \quad \dots., y_{4096}].$$

$$y_0 = 0, \quad y_1 = 0, \quad y_2 = 0\,, \quad y_3 = 0, \dots., \quad y_{4096} = 0.$$

Entonces, suponiendo que Raylib tarda 0.1 segundos en actualizar la ventana de visualización, la fórmula de suavizado aplicada a las frecuencias sería:

$$S[y_i] \mathrel{+}= (\, F[x_i] - S[y_i]) \cdot 8 \cdot 0.1.$$

$$S[y_0] \mathrel{+}= (\, 2 - 0) \cdot 8 \cdot 0.1 = 1.6$$

$$S[y_1] += (3 - 0) \cdot 8 \cdot 0.1 = 2.4$$

$$S[y_2] += (2.5 - 0) \cdot 8 \cdot 0.1 = 2$$

$$S[y_3] += (1.5 - 0) \cdot 8 \cdot 0.1 = 1.2$$

……

$$S[y_{4096}] += (4 - 0) \cdot 8 \cdot 0.1 = 3.2$$

Como se puede ver, se guardan en S los valores representados, obteniendo en S la siguiente secuencia:

$$S = [1.6, 2.4, 2, 1.2, \ldots\ldots, 3.2]$$

Ahora, suponiendo que en la segunda iteración se quieren representar los siguientes valores de frecuencia:

$$F = [3, 4.5, 1, 6, \ldots\ldots, 2.4]$$

Se vuelven a normalizar con los representados en la primera iteración:

$$S[y_0] += (3 - 1.6) \cdot 8 \cdot 0.1 = 2.72$$

$$S[y_1] += (4.5 - 2.4) \cdot 8 \cdot 0.1 = 4.08$$

$$S[y_2] += (1 - 2) \cdot 8 \cdot 0.1 = 1.2$$

$$S[y_3] += (6 - 1.2) \cdot 8 \cdot 0.1 = 5.04$$

……..

$$S[y_{4096}] = (2.4 - 3.2) \cdot 8 \cdot 0.1 = 2.56$$

Y así sucesivamente con cada conjunto de frecuencias. Esto permite una visualización más suave, evitando que los valores de amplitud cambien bruscamente en la ventana con respecto a los anteriores.

Finalmente, las amplitudes normalizadas y suavizadas se dibujan en la ventana utilizando un ciclo que recorre el conjunto de frecuencias que se obtienen después del suavizado. Cada muestra se representa como un rectángulo cuya altura corresponde al valor de amplitud de la frecuencia normalizada y suavizada en ese punto, y cuyo color se determina según el índice de la frecuencia. Esto crea una representación visual dinámica y colorida de las frecuencias de audio a lo largo del tiempo. También se incluye en la parte superior de la interfaz los índices de frecuencia representados, es decir que, si se ha hecho una FFT de N puntos, se representarán las frecuencias de índice de 1 a (N+1) /2, estos índices se reflejarán en la parte superior de la interfaz para indicar el rango de índices de frecuencias que se están representando.

En el capítulo de “Anexos” se puede observar la función implementada para dibujar las frecuencias de audio. Un ejemplo de visualización de frecuencias de audio en la aplicación sería:

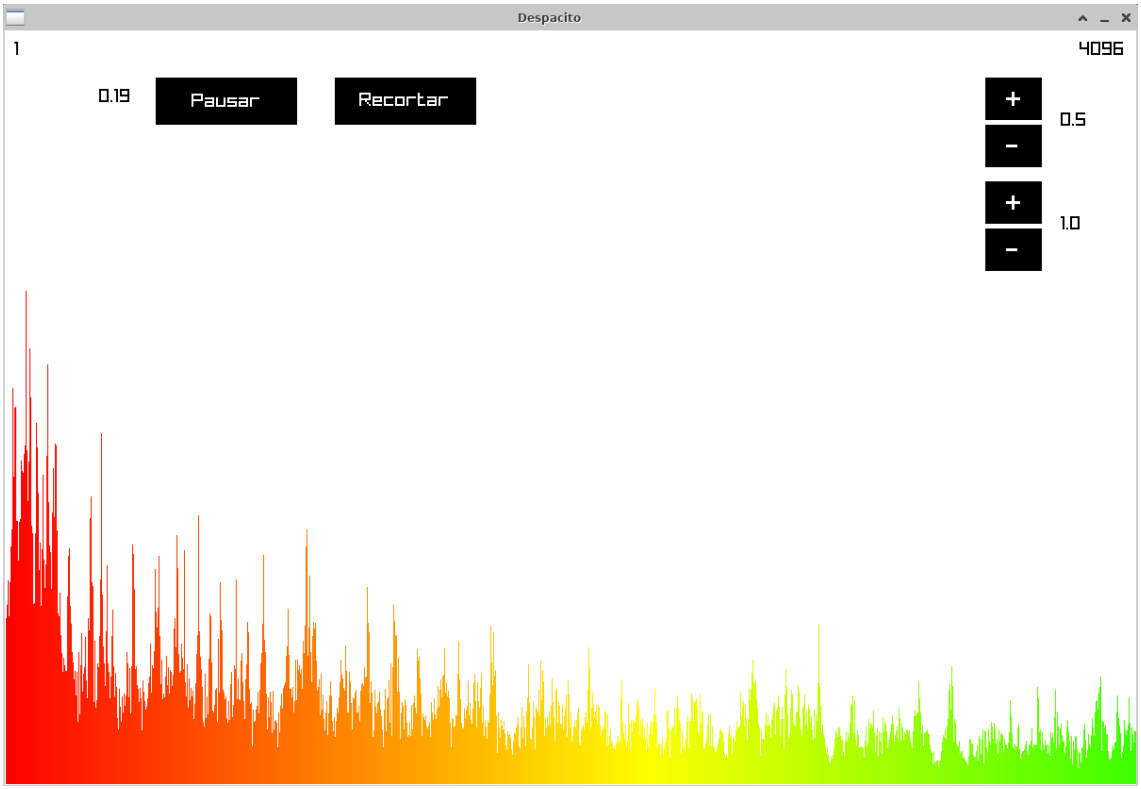

Ilustración 1. Visualización Frecuencias de Audio

En este ejemplo se puede ver la representación de frecuencias de audio. En la parte inferior se muestran los componentes de frecuencia del audio, en la parte superior izquierda se muestra el tiempo transcurrido desde el inicio de la reproducción del audio, en las dos esquinas superiores se muestran los índices de inicio y fin de las frecuencias representadas, en la parte superior derecha se muestra el volumen actual que está a 0.5 y el tono a 1.0, y a la izquierda se encuentran los botones para pausar y recortar el audio.

4.5 Funcionalidades Adicionales

4.5.1 Almacenamiento de Frecuencias de Audio

Para mayor análisis de las frecuencias de audio se implementaron funcionalidades para poder guardar las frecuencias en archivos de tipo RAW. Estos archivos permiten guardar las frecuencias de audio en binario. Esto se implementó de tres diferentes maneras:

Primero, aprovechando el cálculo de la FFT que se va realizando en la función "callback", a medida que se van mostrando las frecuencias, las mismas se pueden escribir en un archivo, obteniendo todos los valores de frecuencias que se han visualizado en la interfaz en un archivo tipo RAW, cuyo nombre se pasaría como parámetro al programa al activar esta funcionalidad (para más información véase el manual de usuarios ubicado en el capítulo "Anexos").

No obstante, hay que tomar en cuenta que las frecuencias obtenidas y escritas en el archivo RAW se calcularían en conjuntos de bloques, por lo que, para obtener las muestras originales a partir de los valores de frecuencia escritos en el archivo, hay que aplicar la FFT inversa a cada conjunto de boque de

frecuencias por separado. También, hay que tomar en cuenta que las muestras utilizadas para calcular la FFT pueden no tener el mismo formato que las del audio original, debido a que las muestras recibidas mediante la función "callback" son transformadas a punto flotante, y las muestras leídas corresponden a un solo canal.

En caso de ir almacenando las frecuencias, cuando se cierre la ventana o se cambie el audio se imprimirá el nombre del archivo donde se almacenaron los valores de frecuencias, y los parámetros de las muestras originales de las cuales se obtuvieron las frecuencias, como el formato que tienen, el número de canales y la frecuencia de muestreo. Esto permite al usuario conocer la información por si se quiere reconstruir el audio.

Otra forma de almacenar los valores de frecuencia es a través de la interfaz de usuario presionando el botón "Recortar". A partir de esto se guardan las frecuencias representadas hasta que se vuelva a presionar el botón. Las frecuencias son almacenadas en un archivo de tipo RAW que tendrá el mismo nombre del audio original con un número adicional. Al igual que el caso anterior, cuando se terminen de almacenar las frecuencias se imprimirá el nombre del archivo donde se almacenaron las frecuencias y el formato de las muestras originales, es decir, tamaño de muestra, frecuencia de muestreo, y número de canales de las muestras utilizadas para obtener las frecuencias.

Para obtener las frecuencias, al presionar el botón "Recortar" se van guardando las muestras en una lista, una vez se vuelva a presionar el botón, se crea el archivo donde se escriben las frecuencias y se crea un plan de FFT que tendrá un tamaño igual a la cantidad de muestras almacenadas desde que se presionó el botón, pasando como entrada la lista con las muestras y obteniendo por salida un puntero "double complex" con las frecuencias a escribir en el archivo. En este caso debido a que el plan se crea a medida que el programa está avanzando, para no retrasar el programa se utilizará como flag "FFTW_ESTIMATE", el cual no busca la opción más óptima para el cálculo de la FFT, pero crea un plan de manera rápida para hacer una FFT eficiente. Después de aplicar la transformada y obtener los componentes de frecuencia se destruye el plan.

Por último, la aplicación también ofrece la posibilidad de especificar por parámetro del programa al momento de ejecución si se quiere guardar un rango de frecuencias de audio. Para esto, se especifica el rango de frames del audio a partir de las cuales se desea obtener las frecuencias. En este caso, las muestras de audio a las cuales se les aplica la FFT son decodificadas con el formato del audio original. Las frecuencias de audio obtenidas son almacenadas en un archivo tipo RAW con el mismo nombre que el audio original. Una vez realizado el recorte se imprimirá por pantalla el formato de las muestras (mismas que el audio original).

En este caso, también se debe crear otra FFT para obtener las frecuencias. Es decir, una vez extraídas las muestras, se creará el plan para hacer la FFT de todo el conjunto de muestras. Este plan tendrá el tamaño de la cantidad de muestras leídas, y se utilizará como flag "FFTW_ESTIMATE" para crearse de manera rápida. Una vez transformadas las muestras, el plan se destruirá.

Se puede destacar que, en este último caso, como las frecuencias obtenidas y almacenadas en el archivo RAW se calculan a partir de las muestras originales del audio sin ningún tipo de conversión del formato del audio, si se desea recrear el audio a partir de estas frecuencias se tiene que usar los parámetros del audio

original para obtener una recreación correcta, es decir, la misma frecuencia de muestreo, el mismo tamaño de muestra y el mismo número de canales.

4.5.2 Recreación del Audio a partir de las Frecuencias

Así como la aplicación es capaz de crear archivos que contienen frecuencias, también es capaz de leer estos archivos y recrear el audio a partir de ellos. Para esto, se pueden pasar archivos de tipo RAW a la aplicación que contengan las frecuencias de audio. En caso de pasar un archivo tipo RAW en lugar de un archivo tipo WAV, MP3 o FLAC, el programa extraerá los componentes de frecuencia del archivo y calculará la transformada inversa de Fourier, la cual permite recuperar las muestras originales del audio a partir de los componentes de frecuencia.

Para que funcione correctamente, el archivo tipo RAW debe contener frecuencias en formato "double complex". A partir del contenido del archivo, se determina su longitud dividiendo el número de bytes que ocupa entre el tamaño de bytes que ocupa una variable de tipo "double complex" en C, de esta manera se obtienen el número de frecuencias contenidas en el archivo.

También, se puede especificar un tamaño de bloque por la línea de comandos para calcular la transformada inversa. Por ejemplo, si se pasa un tamaño de bloque de 8192, la transformada se realizará en bloques a las frecuencias que van de 0 a 8191, de 8192 a 16383, y así sucesivamente. Si no se especifica un tamaño de bloque, se calcula la transformada inversa sobre todo el conjunto de frecuencias contenidas en el archivo.

La transformada inversa es posible calcularla utilizando también la biblioteca FFTW3. En este caso se utilizá la misma función que se había utilizado antes para calcular la FFT, pero de manera inversa, pasando como entrada un conjunto de valores complejos que serían las frecuencias y obteniendo como salida un conjunto de valores reales que corresponderán con las muestras calculadas. La función utilizada en la biblioteca es:

```
fftw_plan fftw_plan_dft_c2r_1d(int n0, fftw_complex *in, double *out,
unsigned flags);
```

Como se puede ver, esta función al igual que la anterior. Se crea con el tamaño de la transformada a realizar "n", y los punteros a las listas que contendrán los valores de entrada y salida. La única diferencia es que en este caso se realiza una transformada que va de complejo a real, pasando como entrada los valores de frecuencia y obteniendo como salida los valores de amplitud de la señal. Al inicializar el plan, se ha utilizado como flag "FFTW_ESTIMATE" debido a que se utiliza una sola vez al crear el audio. Una vez obtenidas las muestras se destruye el plan.

Una vez calculada la transformada inversa y obtenidas las muestras originales, se pasa a crear un archivo de audio tipo WAV utilizando la API ma_encoder para codificar de Miniaudio, la cual es capaz de crear archivos de tipo WAV a partir de un conjunto de muestras. Este archivo se crea por defecto con un solo canal, un tamaño de muestra de 32 bits en punto flotante y una frecuencia de muestreo de 44100 Hz. Sin embargo, estos parámetros se pueden ajustar a las preferencias del usuario por la línea de comandos antes de lanzar el programa. Para esto véase el capítulo "Anexos".

Es importante destacar que, si las muestras de audio originales no tienen el mismo formato con el que se crea el nuevo archivo de audio, el audio resultante no será completamente igual al original. Por ejemplo, si la frecuencia de muestreo original del audio donde se extrajeron las muestras que se utilizaron para obtener los componentes de frecuencias es mayor a la que se utiliza para recrear el audio, el nuevo audio será más lento que el anterior. Además, si el tamaño de la muestra no es el mismo o el número de canales no es el mismo, se puede perder información y afectar al sonido del audio.

Una vez creado el nuevo audio, se crea un dispositivo de reproducción de igual manera que se crea para los audios de tipo WAV, MP3 y FLAC procesados en la aplicación. Sin embargo, en la interfaz se mostrarán las muestras de audio en vez de sus componentes de frecuencia, de las cuales no se podrá hacer ningún tipo de recorte.

Para mostrar las muestras de audio, estas se almacenan en un búfer circular sin aplicarles la FFT. Luego, todas las muestras almacenadas en el búfer se dibujan en la ventana, donde la altura de cada muestra representa su amplitud. La visualización de las muestras de audio se realiza de la siguiente manera:

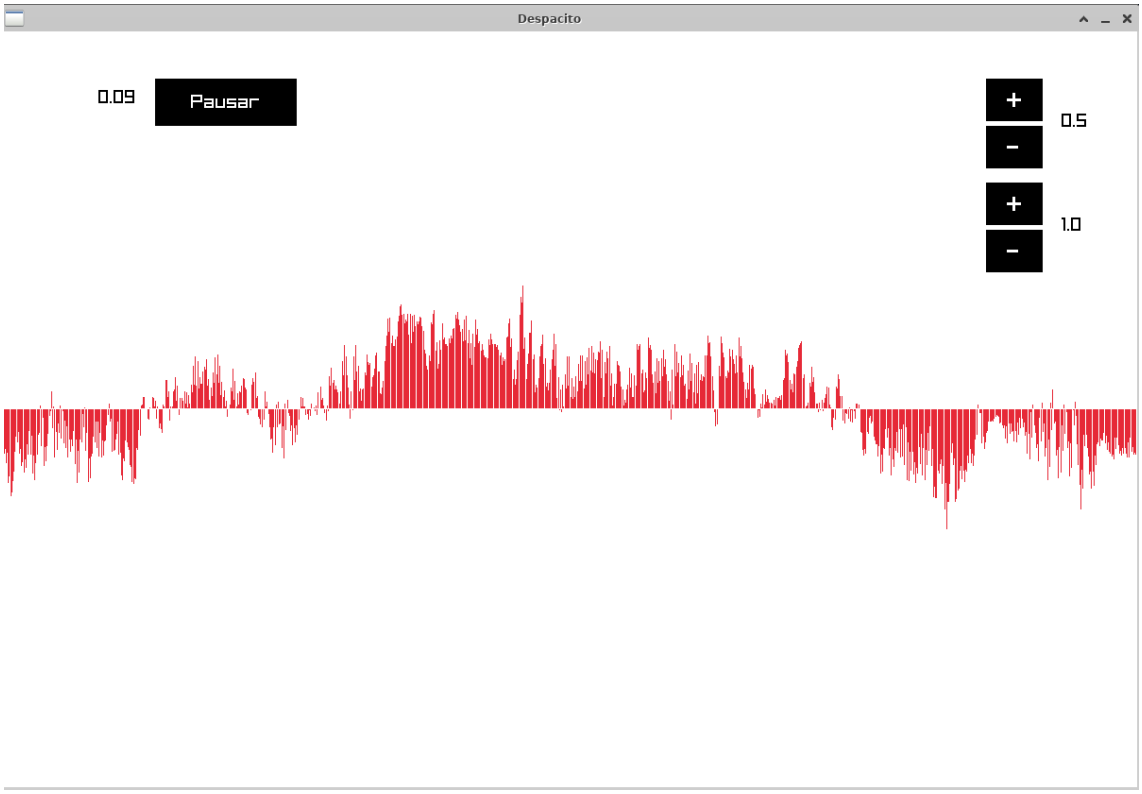

Ilustración 2. Visualización muestras de audio

4.5.3 Cambio de Archivo de Audio en tiempo real

Finalmente, Raylib también proporciona la opción de cargar archivos simplemente arrastrándolos sobre la ventana. De esta manera, es posible cambiar el audio que se está reproduciendo por cualquier otro archivo en formato WAV, MP3, FLAC o RAW. En tal caso, la aplicación detendrá la reproducción del audio actual y restablecerá el estado del programa al inicial,

estableciendo los búferes y las variables utilizadas para el procesamiento del audio a cero, para luego procesar el nuevo audio.

Por otra parte, cuando se termine de reproducir un audio, la interfaz permite arrastrar otro:

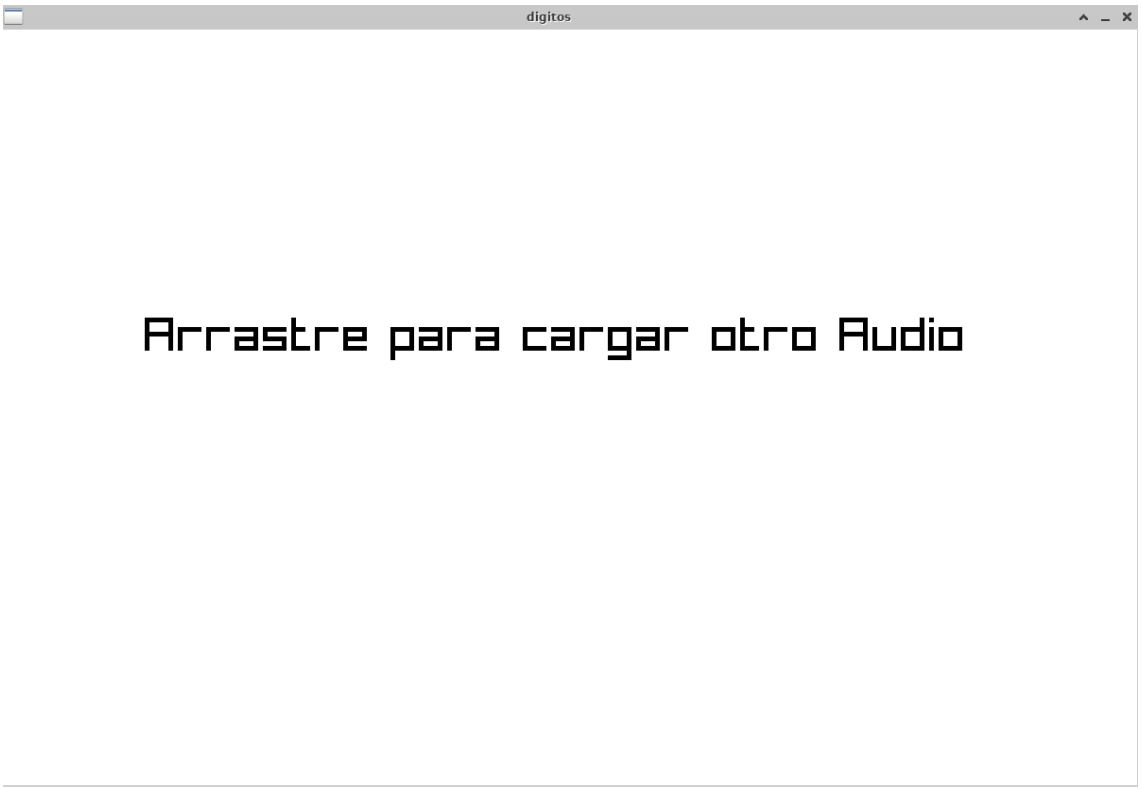

Ilustración 3. Arrastrar nuevo audio

Para cerrar la aplicación, solo se tiene que cerrar la ventana, lo que provoca la destrucción de los dispositivos creados para la reproducción del audio y los planes para el cálculo de la FFT, así como también la liberación de memoria.

La aplicación está disponible en un repositorio de GitHub por si se quiere descargar y usar. Las instrucciones de compilación se encuentran detalladas en el mismo repositorio [22].

4.6 Limitaciones Encontradas

Una de las limitaciones más destacadas de la aplicación es su compatibilidad con formatos de audio. Actualmente, la aplicación solo soporta archivos de audio en formatos WAV, MP3 y FLAC, excluyendo otros formatos populares como OGG y AAC. Esta limitación se debe a las capacidades de la biblioteca Miniaudio, que solo soporta la decodificación de archivos WAV, MP3 y FLAC. Lo que limita la flexibilidad y la variedad de archivos que los usuarios pueden procesar en la aplicación.

Por otra parte, solo se pueden crear audios de tipo WAV, debido a que es el único formato que Miniaudio soporta para codificar, siendo también un límite en la aplicación.

En general, el uso solo de la biblioteca Miniaudio, sin combinarlo con otras bibliotecas para el procesamiento del audio, crea una limitación en cuanto a los formatos y tipos de audios que se pueden procesar.

El rendimiento de la aplicación también puede ser un problema al procesar archivos de audio grandes en tiempo real en equipos menos potentes o más antiguos, debido a que la aplicación puede experimentar ralentizaciones o problemas de rendimiento, lo que podría dificultar su uso eficiente.

Además, la funcionalidad de guardar frecuencias en archivos RAW y luego procesar estos archivos puede presentar limitaciones en cuanto a la precisión de la reconstrucción del audio original. Esto se debe a las diferencias en las características del audio entre el archivo original y el nuevo archivo creado. Si las características del archivo original, como la frecuencia de muestreo, el número de canales y el tamaño de muestra, no se conocen y no se especifican correctamente, el audio reconstruido puede diferir significativamente del original. La falta de información precisa sobre el formato de las muestras originales puede resultar en una reconstrucción inexacta, afectando la calidad del audio final.

5 Resultados y conclusiones

Los resultados obtenidos se pueden evaluar en base a los objetivos planteados inicialmente. Se puede decir que se han logrado cumplir todos los objetivos propuestos.

En cuanto al estudio y análisis del procesamiento de audio, se ha logrado entender qué son las señales digitales, cómo se forman y cuáles son sus características. Además, se ha estudiado la Transformada Rápida de Fourier, como algoritmo para obtener los componentes de frecuencias de la señal, logrando entender su importancia y cómo funciona. Aparte, se han evaluado métodos para la representación visual de frecuencias, que ayudaron a representar una visualización de los componentes de frecuencia de los audios de manera eficiente e intuitiva para el usuario.

En cuanto a la obtención de las frecuencias de los audios, la aplicación desarrollada es capaz de procesar diferentes archivos de audio como WAV, MP3 y FLAC, obteniendo sus valores de amplitud, así como también extrayendo las frecuencias presentes en los audios mediante el cálculo de la FFT. Además, se ha comprobado la correcta extracción de las frecuencias al recrear el audio original a partir de las muestras obtenidas mediante la Transformada Inversa de Fourier aplicada a dichas frecuencias.

En cuanto a la visualización de frecuencias de audio, se ha creado una interfaz de usuario intuitiva que permite visualizar las frecuencias de audio en tiempo real. La visualización permite a los usuarios modificar parámetros como el tamaño de la ventana, los tamaños de bloque que se utilizan para obtener las frecuencias, el rango de frecuencias mostradas en la interfaz y cambiar el tono o el volumen del audio.

Asimismo, la aplicación también ofrece la capacidad de obtener y mostrar las características que componen los archivos de audio que se estén procesando, como la frecuencia de muestreo, el número de canales, el tamaño de las muestras y el número de frames que contiene.

No obstante, la aplicación también es capaz de cumplir con más funcionalidades como la generación de archivos tipo RAW que contienen las frecuencias obtenidas a partir de los audios para su posterior análisis, y de manera inversa es capaz de procesar archivos de tipo RAW, leyendo las frecuencias almacenadas en ellos, calculando la Transformada Inversa de Fourier y creando nuevos archivos de audio tipo WAV, para posteriormente mostrar por la interfaz gráfica los valores de amplitudes de las muestras obtenidas.

Por último, la aplicación es configurable a través de la línea de comandos, permitiendo al usuario cambiar parámetros como volumen, tono, tamaño de la FFT, índices de frecuencias representados en la interfaz, habilitar opciones para obtener en un archivo las frecuencias presentes en un rango de muestras del audio, habilitar opciones para imprimir las características de los audios, así como también la configuración de los parámetros para la creación de audio como la frecuencia de muestreo, el número de canales y el tamaño de la muestra.

En resumen, el desarrollo de esta aplicación ha logrado cumplir con todos los requisitos y objetivos planteados inicialmente. Ofreciendo una interfaz intuitiva

capaz de calcular y mostrar los componentes de frecuencia del audio, y otras funcionalidades en tiempo real y de manera eficiente.

Por otra parte, existen muchas funcionalidades futuras que se pueden desarrollar para mejorar el uso y beneficio de la aplicación. Una mejora importante podría ser la capacidad de procesar una variedad más amplia de formatos de archivo de audio. Esto implicaría la implementación de decodificadores para formatos adicionales.

Además, la aplicación podría ofrecer mayor flexibilidad en la manipulación y configuración del audio procesado. Esto podría incluir opciones para que el usuario pueda ajustar la frecuencia de muestreo, el número de canales y el formato de salida de los audios decodificados para su reproducción y procesado.

En cuanto al análisis de frecuencias, se podrían agregar funcionalidades para mejorar la capacidad de análisis y filtrado de frecuencias. Esto podría incluir la implementación de filtros para eliminar frecuencias no deseadas, la detección de picos de frecuencia y patrones.

Por último, personalmente el desarrollo de este trabajo me ha sido una experiencia formativa. Me ha permitido ampliar mis conocimientos en el campo de las señales y el procesamiento de audio. Gracias a esto, he podido aprender sobre algoritmos que se utilizan diariamente en muchos ámbitos de la tecnología, como lo es la Transformada de Fourier, así como también, me ha permitido abrir mis conocimientos de las aplicaciones que tiene el procesado de señales hoy en día. Por otra parte, el desarrollo de la interfaz en C me ha permitido expandir mis conocimientos en el lenguaje, aprendiendo a poder crear aplicaciones más intuitivas en un lenguaje que no está tan enfocado en el desarrollo de aplicaciones de usuario, logrando el correcto desarrollo de una interfaz y el procesamiento de las señales de audio. Por último, este trabajo me ha ayudado a alcanzar mis objetivos académicos y profesionales, preparándome para abordar desafíos más grandes en mi carrera futura.

6 Análisis de Impacto

Tomando en cuenta los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 [3], se puede decir que el desarrollo de la aplicación se alinea a varios de estos objetivos, específicamente al ODS 4 "Educación de calidad", al ODS 9 "Industria, innovación e infraestructura" y al ODS 12 “Producción y Consumo Responsables”.

El objetivo número cuatro (Educación de calidad), se refiere a promover oportunidades de aprendizaje para todas las personas. La aplicación puede contribuir a este objetivo al proporcionar una herramienta práctica y educativa para personas interesadas en el procesamiento de señales y el análisis de audio. Permite a las personas experimentar directamente con diferentes formatos de audio, aplicar transformadas rápidas de Fourier (FFT) y visualizar las frecuencias en tiempo real. De esta manera, las personas pueden aprender sobre conceptos como el procesamiento de señales digitales y el algoritmo de la Transformada Rápida de Fourier.

El objetivo número 9 (Industria, innovación e infraestructura), se centra en implementar infraestructura para promover la innovación. En este caso, la aplicación puede aportar una ayuda en el ámbito de la investigación y análisis del audio. Debido a que, facilita la realización de estudios sobre los componentes de frecuencia presentes en los audios, pudiendo ser una ayuda para la detección de ruidos, detección de patrones, entre otros.

El objetivo número 12 (Producción y Consumo Responsables), se basa en garantizar modalidades de consumo y producción sostenible. La aplicación se alinea con este objetivo debido a que al desarrollar el software en C y utilizando algoritmos eficientes se puede reducir el consumo de energía.

7 Bibliografía

[1] W. Koenig, H. K. Dunn y L. Y. Lacy, «The sound spectrograph,» *The Journal of the Acoustical Society of America,* vol. 18, nº 1, pp. 19-49, 1946.

[2] L. De la Fraga, « La transformada discreta de Fourier y la transformada rápida de Fourier,» 2001.

[3] A. G. d. l. N. Unidas, «Objetivos de Desarrollo Sostenible (ODS),» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/.

[4] Joseph Fourier, Théorie analytique de la chaleur, 1822.

[5] J. W. Cooley y J. W. Tukey, «An algorithm for the machine calculation of complex Fourier series,» *An algorithm for the machine calculation of complex Fourier series,* vol. 19, nº 90, pp. 297-301., 1965.

[6] H. Pfister, «Discrete-Time Signal Processing,» de *pfister. ee. duke. edu/courses/ece485/dtsp. pdf.*, 2017.

[7] M. T. Heideman, D. H. Johnson y C. S. Burrus, «Gauss and the history of the fast Fourier transform. Archive for history of exact sciences,» *Archive for history of exact sciences,* pp. 265-277, 1985.

[8] M. Soni y P. Kunthe, «A General comparison of FFT algorithms,» *Pioneer Journal Of IT & Management,* 2011.

[9] S. C. Pei y M. H. Yeh, «The discrete fractional cosine and sine transforms,» *IEEE Transactions on Signal Processing,* vol. 49, nº 6, pp. 1198-1207, 2001.

[10] A. Kutepov, «Musializer,» [En línea]. Available: https://github.com/tsoding/musializer.

[11] R. Santamaria, «raylib,» [En línea]. Available: https://www.raylib.com/.

[12] mackron, «miniaudio,» [En línea]. Available: https://miniaud.io/.

[13] R. Bencina y P. Burk, «PortAudio,» [En línea]. Available: https://www.portaudio.com/.

[14] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi y R. Ulerich, «GNU scientific library. Godalming: Network Theory Limited,» de *Fast Fourier Transforms (FFTs)*, 2002, pp. 167-180.

[15] P. N. Swarztrauber, «FFTPACK,» [En línea]. Available: https://netlib.org/fftpack/.

[16] IBM, «Fortran,» [En línea]. Available: https://fortran-lang.org/es/index.

[17] M. Frigo y S. G. Johnson, «FFTW3,» [En línea]. Available: https://fftw.org/.

[18] M. Frigo y S. G. Johnson, «FFTW: An adaptive software architecture for the FFT,» *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181),* vol. 3, pp. 1381-1384, 1998.

[19] R. Stallman, «The GNU project,» 1998.

[20] H. Nyquist, «Certain topics in telegraph transmission theory,» *Transactions of the American Institute of Electrical Engineers,* vol. 47, nº 2, pp. 617-644, 1928.

[21] A. Kutepov, «Musializer,» [En línea]. Available: https://github.com/tsoding/musializer/blob/master/src/plug.c.

[22] I. B. Imbrondone, «AnalizadorFreq,» [En línea]. Available: https://github.com/iimbrobucci/AnalizadorFreq.

8 Anexos

8.1 Manual de Usuario

Parámetro	Descripción
-f <fichero>	Especifica el fichero de audio. (Obligatorio) Solo se permiten audios *tipo = WAV, MP3, FLAC, RAW. *formato = 8 bits, 16 bits, 24 bits, 32 bits, punto flotante. *canales = 1, 2. Los archivos tipo MP3, WAV y FLAC, serán codificados, reproducidos y mostrarán las frecuencias presentes en ellos a través de una interfaz. Los archivos tipo RAW deben obtener solo valores de frecuencias de audio en formato “double complex”. A partir de ellos será posible crear un archivo de audio que luego se reproducirá y mostrará sus muestras a través de una interfaz.
-v <volumen>	Ajusta el volumen del audio (rango [0.0, 1.0]). (Por defecto 0.5).
-t <tono>	Ajusta el tono del audio (rango [1.0, 6.0]). (Por defecto 1.0).
-p	Imprime la información del audio (Formato, número de canales, tamaño de muestra, número total de frames).
-h <alto>	Especifica la altura de la ventana de interfaz de usuario (rango [600, 1400]). (Por defecto 800).
-w <ancho>	Especifica el ancho de la ventana de interfaz de usuario (rango [800, 2400]). (Por defecto 1200).
-n <tamaño>	Especifica el tamaño del búfer de procesamiento de muestras (rango [1024, 8192]). Solo se permiten potencias de dos (Por defecto 8192). Establece el tamaño del búfer que se usara para calcular la Transformada de Fourier de las muestras y, por ende, el número de frecuencias que serán representadas a la vez en la interfaz, es decir, las muestras serán procesadas en tamaños de ventana que pueden variar según (-n).
-i <inicio>	Especifica el índice de inicio de visualización de frecuencias (rango [0, 200]). (Por defecto 1.0).
-e <fin>	Especifica el índice de fin de visualización de frecuencias (rango [64, 4096]). (Por defecto 4096). Si el índice de fin es mayor que la mitad del tamaño de la ventana (-n) se establecerá a ser igual al tamaño de la ventana/2. Si el índice de inicio es mayor que el índice de fin el índice de inicio pasara a ser 1.

-q <canal>	Especifica el canal del audio que se quiere procesar (1 o 2). 1 corresponde al izquierdo y 2 al derecho. Por defecto 1.
-r	Recorta una trama del audio. (Si se habilita esta opción se debe especificar el rango a cortar (-a) (-l).
-a <inicio>	Especifica el índice de frame a partir de la cual se quiere empezar a recortar la trama (>= 0).
-l <fin>	Especifica el índice de la última frame que se quiere recortar de la trama (>= 0).
-s <muestreo>	Especifica la frecuencia de muestreo para la creación de audios (rango [8000, 96000]). (Por defecto 44100).
-c canales	Especifica el número de canales para la creación del audio (1 o 2). (Por defecto 1)
-b bloque	Especifica el tamaño de bloque al que se le debe aplicar la FFT en la creación de audio (rango [1024, 8192]). (Por defecto es el tamaño del archivo).
-m formato	Especifica el formato para la creación del audio (8, 16, 24 o 32 bits). (Por defecto punto flotante)
	-s, -c, -b, -m Solo se utilizan en el caso de pasar un archivo tipo RAW.
-d <nombre>	Activa el almacenamiento de las frecuencias del audio en un fichero con el nombre pasado como argumento.
-u	Muestra la información de uso del programa.

8.2 Función Callback

```
void data_callback(ma_device* pDevice, void* pOutput, const void*
pInput, ma_uint32 frameCount)
{
  (void)pInput;
  if(audio.pausado==1) timeF+=frameCount;
   ma_engine_read_pcm_frames((ma_engine*)pDevice->pUserData,
pOutput, frameCount, NULL);


   float (*samples)[2] = pOutput;
   for (ma_uint32 i = 0; i < frameCount; i++) {
     buffer[buffer_index] = (double)samples[i][canal];
      if (buffer_index == tamanoBuffer - 1 && audio.spectogram == 1)
{
      fftw_execute(plan);
        if (guardarFrecuencias == 0) {
                fwrite(outBuffer,      sizeof(double      complex),
tamanoBuffer, audio.fileF);
        }
      }
      buffer_index = (buffer_index + 1) % tamanoBuffer;
    }


   //Si se quiere recortar una trama se guardan las muestras para
calcular sus componentes de frecuencia
      if(audio.spectogram==1 && r.recortando==0){
      GuardarMuestrasTR(samples, frameCount);
       }
}
```

8.3 Función para visualizar frecuencias

```
void DibujarFrecuencias(){
  double max_amp=1.0;
  int m=0;

  for(int i=vFreq.inicio; i<vFreq.fin; i++){
    double freq=amp(outBuffer[i]);
    if(freq>max_amp) max_amp=freq;
    outLog[m++]=freq;
  }

  int anchoFreq= ventana.ancho/ m;
  if(anchoFreq<1) anchoFreq=1;

  for(int i=vFreq.inicio; i<vFreq.fin; i++){
    double a=outLog[i]/max_amp;
    outSmooth[i] += (a - outSmooth[i]) * vFreq.suavidad * vFreq.dft;
    double t = outSmooth[i];
    double hue = (double)i /m ;
    Color color = ColorFromHSV(hue * 360, 1.0, 1.0);
    DrawRectangle(i * anchoFreq, ventana.alto - ventana.alto *2/3 *
t, anchoFreq, ventana.alto *2/3 * t, color);
  }
}
```

8.4 Instalación de Bibliotecas en entorno GNU/Linux

8.4.1 Raylib

1. Abrir terminal en máquina Linux
2. Actualización de paquetes:

```
sudo apt update
```

3. Instalación de dependencias:

```
sudo apt install git build-essential cmake libasound2-dev libx11-dev libxrandr-dev libxi-dev libgl1-mesa-dev libglu1-mesa-dev libxinerama-dev libxcursor-dev
```

4. Clonación del repositorio:

```
git clone https://github.com/raysan5/raylib.git
```

5. Navegación al directorio de raylib:

```
cd raylib
```

6. Configuración de la biblioteca:

```
cmake .
```

7. Compilación e instalación:

```
sudo make install
```

8.4.2 FFTW3

1. Abrir terminal en máquina Linux.
2. Actualización de paquetes:

```
sudo apt update
```

3. Instalación de biblioteca:

```
sudo apt install libfftw3-dev
```

8.4.3 Bibliotecas para la compilación

1. Instalación de glfw:

```
sudo apt-get install libglfw3-dev
```

2. Instalación de pkg-config:

```
sudo apt-get install pkg-config
```

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 16:29:14 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)