Daniel Ortega Domínguez

05 TRABAJO FIN DE GRADO

INDUSTRIALES

TRABAJO FIN DE GRADO

RECICLAJE DE OSCILOSCOPIOS ANALÓGICOS PARA LA GENERACIÓN DE SONIDO Y ANIMACIONES MEDIANTE GRÁFICOS VECTORIALES.

TRABAJO FIN DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE **GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES**

FEBRERO 2025

Daniel Ortega Dominguez

DIRECTOR DEL TRABAJO FIN DE GRADO:

Yago Torroja Fungairiño

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

A mi padre y a mi madre.

Por el sacrificio, el amor y la paciencia.

A todos los docentes, compañeros y amigos que me han acompañado en el camino.

A Yago Torroja Fungairiño.

Por darme un soplo de aire fresco cuando más lo necesitaba.

“Por eso han aprendido a cultivar flores y a cantar bien sus penas,
y han inventado las mejores obras y los mejores instrumentos.
Por eso entienden de arte y saben encontrarlo donde lo haya,
aunque no lo haya, que siempre lo hay.”

- Ana Isabel García Llorente

RESUMEN.

Este proyecto se plantea con la ambición de alcanzar un punto de encuentro entre el arte y la ingeniería. Se trata de una propuesta que combina herramientas tecnológicas con una intención artística, ofreciendo una perspectiva en la que la creatividad y la técnica dialogan de manera activa.

El arte se enfoca en la creatividad y la libre expresión, utilizando medios visuales y sonoros, entre otros, para transmitir mensajes que invitan a la reflexión y a la interpretación subjetiva.

La ingeniería como disciplina debe resolver problemas complejos a través de la creatividad y la innovación, siempre apoyándose en un profundo conocimiento teórico y práctico y aplicando los principios físicos a su alcance para poder hacer más fácil la vida de las personas.

Este proyecto se sitúa en la intersección de ambos mundos, con el objetivo de explorar las posibilidades creativas que ofrece la ingeniería más allá de sus aplicaciones utilitarias. Se pretende desarrollar un dispositivo que pueda ser utilizado como medio artístico y que, además, pueda ser reutilizado por otras personas.

Para llevar a cabo este propósito, se aprovecha uno de los instrumentos más representativos en la formación de cualquier ingeniero electrónico: el osciloscopio. Este dispositivo, normalmente utilizado para visualizar señales eléctricas, se convierte aquí en un lienzo donde se proyectan animaciones gráficas vectoriales, generando una experiencia visual y sonora.

El proyecto se centra en la transformación de animaciones vectoriales SVG *(Scalable Vector Graphics)* en señales de audio, que luego se reproducen en un osciloscopio analógico y en unos altavoces externos. A través de diferentes técnicas que se desarrollan a lo largo de esta memoria, las animaciones SVG se transforman y adecuan a un formato que permite la generación y reproducción de estas señales de audio en dichos dispositivos. Al tratarse de la misma señal, es posible reproducir las animaciones en la pantalla del osciloscopio mientras se genera una experiencia sonora en paralelo. El sistema, además, provee al usuario de una entrada interactiva a través de un teclado MIDI (*Musical Instrument Digital Interface)*, con el que es posible aplicar efectos y modificar distintos parámetros de la reproducción.

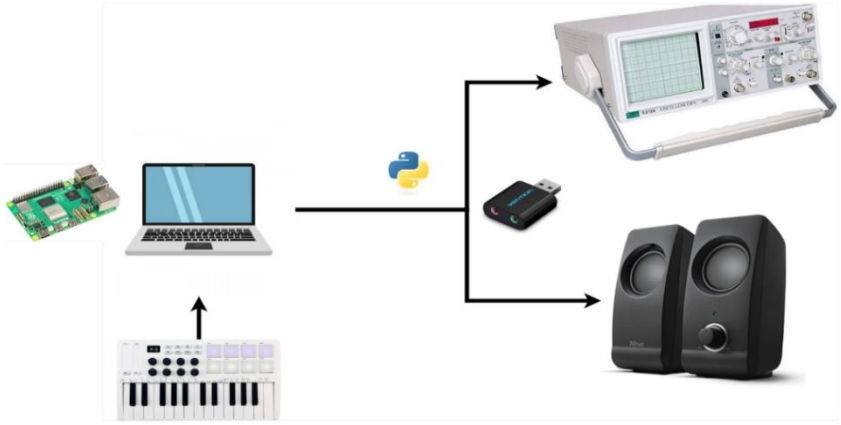


Figura 1: Esquema general del dispositivo.

La lógica del programa se desarrolla en Python, donde un sistema multihilo gestiona de manera eficiente tanto la reproducción de las animaciones como la generación de las señales de audio correspondientes, además de la interacción del usuario en tiempo real.

El uso de gráficos SVG en este contexto aporta una flexibilidad considerable, ya que este formato permite definir las animaciones de manera precisa mediante coordenadas y vectores, facilitando su manipulación para la conversión en señales de audio y posterior visualización en el osciloscopio. A lo largo del código, se implementa un proceso de preprocesado de estas animaciones, de manera que el coste de computación del dispositivo sea menor.

Figura 2. Esquema general del software implementado.

El proyecto ha sido diseñado no solo para cumplir con las funciones básicas propuestas, sino también para ser una plataforma abierta a la expansión. Es escalable y modular, permitiendo futuras mejoras y expansiones según las necesidades o la imaginación de quienes interactúen con él, ya sea para hacer música, visualizar animaciones o aportar nuevas funcionalidades al dispositivo.

En su forma actual, el sistema permite la interacción en tiempo real mediante el ajuste de parámetros de reproducción, sin embargo, las posibilidades de crecimiento son amplias, ya que es posible añadir más efectos sonoros y visuales, así como aumentar la complejidad de las animaciones SVG.

Se puede imaginar, por ejemplo, la incorporación de más instrumentos de entrada, como sensores de movimiento o cámaras, que permitirían modificar las animaciones en función de la actividad física del usuario. También, podría ser utilizado en instalaciones artísticas interactivas, donde el público manipula las animaciones y el sonido en tiempo real, creando una experiencia inmersiva o podría tener aplicaciones educativas, ayudando a los estudiantes de electrónica a comprender mejor los principios de las señales y su representación gráfica. La capacidad de combinar arte y tecnología en un solo sistema presenta una herramienta poderosa para la enseñanza y la exploración creativa.

Otro aspecto importante es el reaprovechamiento de osciloscopios analógicos que se encuentran en desuso. Muchos de estos dispositivos, a pesar de estar obsoletos en algunos laboratorios modernos, conservan su funcionalidad y pueden ser utilizados de manera creativa en proyectos artísticos y educativos.

No solo se rescatan equipos que de otro modo quedarían relegados, sino que se promueve un enfoque sostenible y de aprovechamiento de recursos ya existentes. La naturaleza de los osciloscopios analógicos, con su respuesta directa y sin procesamientos digitales que puedan alterar la señal, les confiere un carácter único en este tipo de aplicaciones, generando imágenes y sonidos que son difíciles de replicar en dispositivos modernos. Este enfoque añade valor al proyecto en términos de sostenibilidad y ofrece una oportunidad para revalorizar tecnologías pasadas desde una nueva perspectiva creativa.

En definitiva, se propone una mirada creativa hacia el uso de herramientas técnicas para crear experiencias artísticas. Al convertir un dispositivo como el osciloscopio en un medio de expresión visual y sonoro, se demuestra que la ingeniería no solo resuelve problemas, sino que también puede inspirar y abrir nuevas vías para la expresión a través del arte.

Por otro lado, se incluye en este documento un análisis en base a la experiencia del autor sobre el papel de las nuevas tecnologías involucradas, como la implementación de la inteligencia artificial en el proceso de aprendizaje o en la programación. A lo largo del desarrollo, se aplican herramientas basadas en inteligencia artificial para optimizar el diseño del sistema y facilitar la toma de decisiones técnicas.

Estas tecnologías permiten acelerar ciertos procesos de programación y fueron clave para el aprendizaje del autor en áreas complejas como la generación de señales. Se ofrecen reflexiones personales sobre cómo la inteligencia artificial puede ser una herramienta valiosa para ingenieros en formación y profesionales que busquen automatizar procesos y mejorar la eficiencia de sus proyectos.

Códigos UNESCO.

- **3307** - Tecnología Electrónica.
- **1203** - Inteligencia Artificial.
- **1205** - Procesamiento de Señales.
- **1206** - Gráficos por Ordenador.
- **2203** - Arte Electrónico y Digital.
- **3311** - Instrumentación y Medición Electrónica.
- **1201** - Computación e Informática.
- **6203** - Estética y Teoría del Arte.
- **6202** - Historia y Filosofía de la Tecnología.
- **5312** - Educación en Ciencias e Ingeniería.

ÍNDICE

1. INTRODUCCIÓN.

Los ingenieros no solo desarrollan soluciones técnicas eficientes, sino que también toman decisiones que afectan al medio ambiente, la economía y la sociedad en general. Es una disciplina que en ciertos puntos puede ser interpretada como utilitarista, carente de diversión o 'cruda'. Nada más lejos de la realidad.

Este proyecto aspira a construir un espacio de convergencia entre el arte y la ingeniería. Es una iniciativa que integra tecnologías avanzadas con un enfoque artístico, brindando una visión en la que la creatividad y el rigor técnico interactúan de forma dinámica.

Se plantea una aproximación al empleo de herramientas técnicas para generar experiencias artísticas, de manera que, al transformar un dispositivo como el osciloscopio en un canal de expresión visual y auditiva, se evidencia que la ingeniería no solo ofrece soluciones prácticas, sino que también puede inspirar y ampliar las posibilidades de expresión en el ámbito artístico.

La idea inicial es concebida en colaboración con el director del proyecto, focalizando su propósito en la reutilización de osciloscopios analógicos en desuso. La naturaleza de los osciloscopios analógicos, con su respuesta directa y sin procesamientos digitales, les confiere un carácter único en este tipo de aplicaciones, generando imágenes que son difíciles de replicar en dispositivos modernos.

Dichos osciloscopios se encuentran en desuso en el Centro de Electrónica Industrial (CEI) de la Escuela Técnica Superior de Ingenieros Industriales (ETSII) de la Universidad Politécnica de Madrid (UPM).

Inspirado en diversas obras artísticas del mismo tipo, principalmente en el software *OsciRender* del autor (Ball, 2024), el proyecto se orienta hacia el desarrollo de un software propio para la visualización de imágenes en osciloscopios. *OsciRender*, definido por su autor como: *"Un plugin de audio sintetizador para crear música dibujando objetos, texto e imágenes en un osciloscopio"* es una fuente de inspiración importante. Sin embargo, el proyecto se diferencia al buscar la reproducción de animaciones en movimiento, además de imágenes estáticas.

Inicialmente, se estudia la posibilidad de crear un dispositivo con varios osciloscopios conectados en serie, permitiendo la división de la animación en partes o la creación de efectos visuales complejos en los que las animaciones transitan entre varios osciloscopios. Sin embargo, esta idea se desestima en el curso del proyecto debido a que su complejidad técnica y limitaciones temporales quedan fuera del alcance del mismo. También se plantea la posibilidad de procesar el programa utilizando un microprocesador SBC (*Single Board Computer*), como una *Raspberry Pi*. Aunque esta opción no ha sido probada, se considera a lo largo de todo el desarrollo del proyecto.

Finalmente, se decide desarrollar una tecnología capaz de visualizar animaciones en un solo osciloscopio, con la posibilidad de que el usuario interactúe con el dispositivo para explorar diferentes posibilidades visuales y acústicas. Todo ello se mueve desde un ordenador portátil.

Las animaciones se generan a través del software de diseño *Blender,* aprovechando que el autor cuenta con experiencia en el uso de este programa. A partir de animaciones FBX *(Filmbox)* se generan animaciones SVG, con un formato fácil de desglosar y trabajar. Con estas animaciones SVG, a través de distintas técnicas de formateo de datos, se consigue generar ondas de sonido que son reproducidas en paralelo en un osciloscopio analógico y en unos altavoces externos. De esta manera, se consigue una experiencia en la que el usuario puede visualizar y escuchar la misma onda de sonido, la cual ha sido tratada para poder representar la animación original en la pantalla del osciloscopio.

A lo largo del texto se exponen los conceptos que son necesarios para el desarrollo del proyecto, la metodología utilizada, explicaciones detalladas de la tecnología implementada, el código y las pruebas a las que ha sido sometido, junto con un contexto histórico relevante.

También se incluyen reflexiones sobre la utilización de nuevas tecnologías como la inteligencia artificial y su implementación en el ámbito educativo en base a la experiencia del autor, y por último, se incluye una planificación temporal y un presupuesto económico, que sirve como referencia para estudiar la viabilidad temporal y económica de un proyecto de características similares.

1.1. EL OSCILOCOPIO.

Los primeros osciloscopios fueron inventado a finales del siglo XIX, en la década de 1890 por el ingeniero y físico francés André Blondel. Este aparato consistía en un oscilógrafo electromecánico que, con ayuda de un péndulo con tinta conectado a una bobina, permitía registrar en una cinta de papel los valores de la señal eléctrica, aunque su utilidad era limitada.

Figura 3: Osciloscopio analógico.

No fue hasta 1897, con la invención de los rayos catódicos (CRT), que el científico alemán Karl Ferdinand Braun inventara el primer osciloscopio de rayos catódicos, lo que supuso una verdadera evolución en el campo de la medición de las señales eléctricas. En 1931 la compañía británica A.C. Cossor pudo adaptar esta tecnología y presentó el primer osciloscopio utilizable fuera de un entorno de laboratorio. Con el término de la Segunda Guerra Mundial los aparatos de medida, incluyendo los osciloscopios, se desarrollaron y se convirtieron en herramientas esenciales para la ingeniería y la ciencia. En 1946 Howard Vollum y Melvin Murdock fundaron la compañía Tektronix, que muy pronto se convirtió en el líder mundial en oscilografía.

1.2. OSCILOSCOPIOS ANALÓGICOS.

Los osciloscopios analógicos constan de un tubo de rayos catódicos que tiene 3 partes fundamentales encerradas en un tubo de vidrio con vacío en su interior:

- Cañón de electrones.
- Dispositivo de desviación.
- Pantalla.

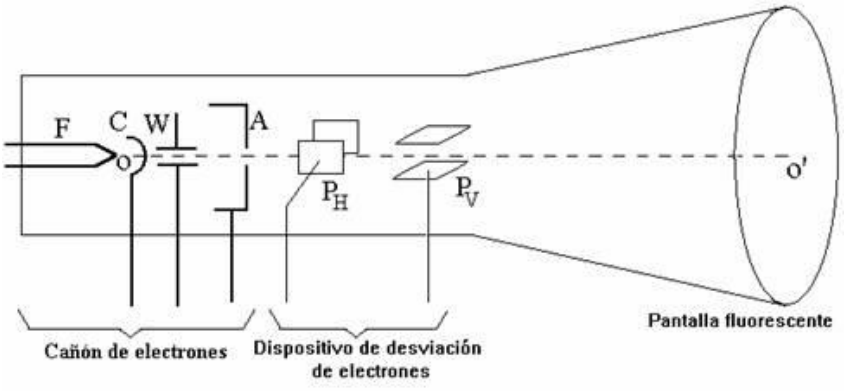


Figura 4: Esquema de un CRT

El cañón da lugar a un haz de electrones que es desviado por los pares de placas horizontales y verticales que crean un campo eléctrico (algunos emplean campos magnéticos). La pantalla está recubierta en su interior por una sustancia fluorescente que se ilumina cuando inciden en ella los electrones proyectados.

La naturaleza del osciloscopio analógico se considera ideal para el propósito de este proyecto, debido a su capacidad para producir una representación más fluida y natural de las señales gráficas en comparación con los osciloscopios digitales, cuya representación discreta afecta a la precisión y calidad de la visualización.

1.3. OSCILOSCOPIOS DIGITALES.

Los osciloscopios digitales son una evolución de los tradicionales osciloscopios analógicos y supusieron un avance en la forma en que los ingenieros y científicos miden y analizan señales eléctricas, al desarrollarse equipos más compactos, cómodos y versátiles. Convierten las señales analógicas en valores digitales a través de un proceso de muestreo, almacenan estos datos en memoria y los representan gráficamente en una pantalla digital. El desarrollo de los osciloscopios digitales comenzó en la década de 1980, impulsado por los avances en la tecnología de semiconductores y la digitalización de señales, con la llegada de los convertidores analógico-digitales (DAC) y la capacidad de los sistemas electrónicos para procesar grandes volúmenes de datos en tiempo real.

Figura 5: Osciloscopio digital.

1.4. GRÁFICOS VECTORIALES.

Una imagen vectorial es un tipo de imagen digital formada por objetos geométricos como segmentos, polígonos, arcos, y curvas, cada uno de ellos definidos por atributos matemáticos. Su principal ventaja es la capacidad de ampliar el tamaño de la imagen sin sufrir pérdidas de calidad, en contraste con las imágenes de formato de mapa de bits (formadas por píxeles), que pierden definición al aumentar su tamaño. Además, las imágenes vectoriales permiten aplicar transformaciones geométricas de forma sencilla mediante operaciones matemáticas, tales como mover, rotar, estirar o deformar los objetos.

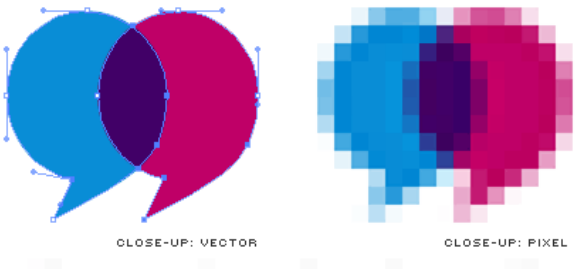


Figura 6: Imagen vectorial y sus atributos matemáticos.

La tecnología de gráficos vectoriales supuso un avance en el campo de la representación visual digital, ya que permite crear imágenes infinitamente escalables, lo cual es fundamental para aplicaciones técnicas y artísticas.

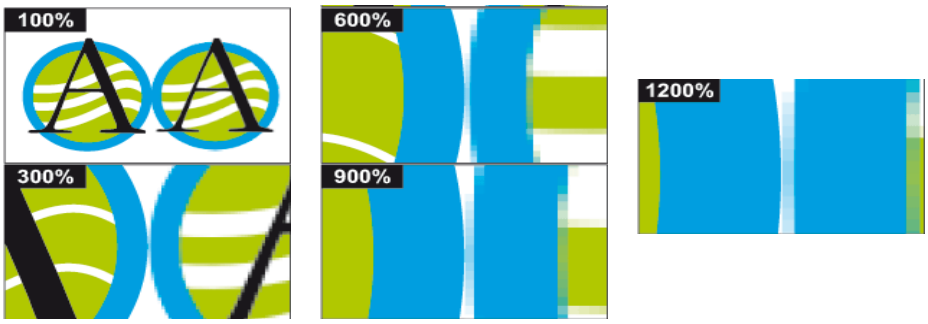


Figura 7: Escalabilidad de una imagen vectorial y una de mapa de bits.

En los primeros días de la computación, entre las décadas de 1950 y 1980, se utilizaba un sistema de generación de gráficos vectoriales conocido como sistema caligráfico. Este sistema empleaba un rayo electrónico del tubo de rayos catódicos que era dirigido para trazar formas directamente en la pantalla. Al repetirse este proceso a gran velocidad, se conseguía una imagen continua, casi libre de intermitencias. Estos dispositivos, conocidos como monitores X-Y, trazaban gráficos vectoriales, lo que ofrecía una mayor precisión en las curvas y líneas en comparación con los sistemas existentes hasta el momento.

Con el tiempo, las tecnologías evolucionaron hacia dispositivos como la Tektronix 4010, un terminal gráfico vectorial que fue pionero en la representación gráfica en tiempo real. Estos sistemas permitían el trazado preciso de líneas y gráficos, lo que los hacía ideales para aplicaciones técnicas de representación y medición.

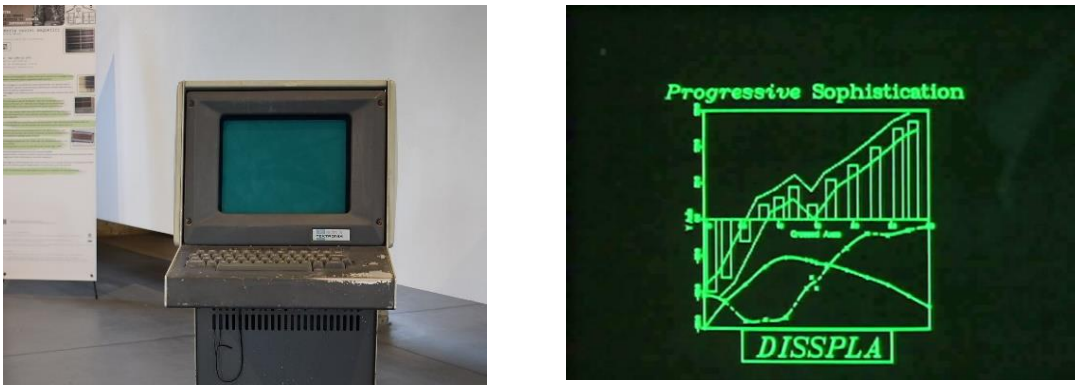


Figura 8: Tektronix 4010 y representación en pantalla de gráfico vectorial.

Con el tiempo los dispositivos rasterizados, que dividen la pantalla en una cuadrícula de píxeles, pasaron a dominar la industria. Estos dispositivos permiten representar imágenes con gran detalle y colores variados, lo que los hace ideales para monitores y pantallas modernas como las pantallas LCD y OLED gracias a la alta resolución que ofrecen.

1.5. APLICACIÓN EN EL PROYECTO.

A pesar del dominio de los sistemas rasterizados, los osciloscopios analógicos, aunque no fueron desarrollados como dispositivos de visualización gráfica, comparten principios fundamentales con los sistemas vectoriales. Estos dispositivos trazan gráficas de señales eléctricas en tiempo real mediante coordenadas X-Y, de manera similar a los monitores caligráficos. Al mostrar las variaciones de voltaje como formas de onda, los osciloscopios proporcionan una representación continua de los datos.

En este proyecto se logra la convergencia de ambas tecnologías. Las animaciones vectoriales, que permiten manipular y transformar geometrías de manera precisa, se integran con las capacidades del osciloscopio para trazar estas señales gráficas como ondas sonoras.

Esta combinación de tecnologías refleja cómo herramientas antiguas, como el osciloscopio analógico, pueden ser reutilizadas en aplicaciones creativas modernas, ampliando sus horizontes más allá de sus usos tradicionales.

2. OBJETIVOS

El objetivo principal del proyecto consiste en reutilizar osciloscopios analógicos en desuso para reproducir animaciones vectoriales en ellos, combinando representaciones gráficas con señales de sonido y aportar diferentes posibilidades interactivas al usuario.

En las fases tempranas del proyecto, el objetivo se establece en la creación de un dispositivo formado por varios osciloscopios unidos en *'rack'* de manera que se obtengan efectos visuales en los que las animaciones transiten de uno a otro o se dividan entre las distintas pantallas de los osciloscopios. Para ello se pretende desarrollar un software procesado y gestionado por un microprocesador SBC como una *Raspberry Pi*. Este objetivo se desestima en el transcurso del proyecto por quedar fuera del alcance del mismo.

El objetivo finalmente se fija en garantizar la reproducción de animaciones *SVG* con una resolución adecuada y una fluidez robusta, permitiendo al usuario interactuar en tiempo real con la pantalla de un osciloscopio analógico. El software desarrollado se procesa directamente desde un ordenador portátil. Se busca que las modificaciones realizadas por el usuario se reflejen de forma inmediata en la visualización y en el sonido, integrando eficazmente los sentidos de la vista y el oído.

Además de este objetivo principal, se establecen los siguientes objetivos específicos:

- **Modularidad del programa:** El software debe ser altamente modularizable, con componentes que puedan ser reutilizados en otros proyectos similares. Esto incluye la posibilidad de añadir nuevas funcionalidades sin comprometer el rendimiento.
- **Eficiencia del sistema:** Se plantea la utilización de diferentes técnicas de ahorro de consumo de memoria, de manera que sea posible su ejecución tanto en un ordenador portátil o un microprocesador SBC. Esto último no ha sido testado.
- **Interactividad mediante MIDI:** Integración de un teclado MIDI, o cualquier otro sistema de control genérico, que permita al usuario controlar parámetros en tiempo real, como la frecuencia de reproducción o la aplicación de efectos visuales y sonoros.
- **Escalabilidad:** El sistema debe ser escalable, permitiendo la incorporación de nuevas animaciones, efectos y dispositivos de entrada.

Los resultados obtenidos han demostrado que es posible generar señales de audio que puedan ser reproducidas en la pantalla de un osciloscopio y unos altavoces, creando un entorno interactivo en el que el usuario puede modificar los efectos en tiempo real, percibiendo los cambios visual y auditivamente. El proyecto ha alcanzado su objetivo inicial de crear una interfaz artística y tecnológica.

3. METODOLOGÍA.

La metodología empleada en este proyecto se diseña para facilitar la estructuración y el avance progresivo desde la planificación inicial hasta la implementación completa del sistema. Dada la complejidad de la propuesta, se divide el desarrollo en varias fases principales, cada una orientada a un conjunto específico de objetivos y desafíos.

Este capítulo describe en detalle cada fase metodológica y su contribución al éxito del proyecto, junto con los logros específicos alcanzados en cada una, destacando los enfoques de trabajo que resultaron ser más efectivos.

Se divide en tres fases principales:

1. Fase previa.
2. Preparación y aprendizaje del lenguaje de programación elegido.
3. Desarrollo del proyecto.

3.1. FASE PREVIA.

En la fase inicial del proyecto, se establecen las directrices generales y se define la concepción del sistema en función de los objetivos planteados. La elección recae en un osciloscopio analógico de tubo de rayos catódicos, principalmente por su disponibilidad en el laboratorio. No obstante, este dispositivo también resulta ventajoso para el proyecto, ya que proporciona una representación fluida y continua de las animaciones, superando en este aspecto a los osciloscopios digitales.

Durante esta etapa, se definen los requisitos técnicos, anticipan posibles desafíos y estructura la carga de trabajo. También se identifica y selecciona el hardware y software adecuados, sentando una base sólida para las etapas posteriores de desarrollo.

3.2. PREPARACIÓN Y APRENDIZAJE DEL LENGUAJE DE PROGRAMACIÓN.

El lenguaje de programación seleccionado para el desarrollo del proyecto es Python, debido a su simplicidad, amplia gama de bibliotecas especializadas y su capacidad para gestionar grandes volúmenes de datos, además del interés por adquirir competencia en este lenguaje.

Durante la preparación se emplean diversos recursos de aprendizaje incluyendo cursos en línea, libros especializados y documentación oficial de Python. Además, el uso de la inteligencia artificial resulta una herramienta clave en el proceso de aprendizaje, ya que facilita el desarrollo mediante explicaciones simplificadas y ejemplos personalizados básicos. Los conceptos más relevantes adquiridos durante el proyecto abarcan:

1. Manipulación de arrays y datos numéricos con la biblioteca ‘NumPy’, así como la utilización de las listas nativas de Python.
2. Preprocesado y optimización de datos.
3. Programación multihilo (*multithreading*).
4. Manejo y procesamiento de archivos SVG y lenguaje XML *(Extensible Markup Language)*.
5. Generación y manipulación de audio en tiempo real.
6. Control MIDI.
7. Automatización de tareas repetitivas.

Esta lista de conceptos se numera para servir como referencia a cualquier persona interesada en desarrollar un proyecto similar en Python, proporcionando una orientación sobre qué orden de conceptos resulta más relevante aprender primero, en base a la experiencia del autor.

3.3. DESARROLLO DEL PROYECTO.

La fase de desarrollo se estructura en diversas etapas técnicas que permiten integrar y optimizar los componentes necesarios para cumplir los objetivos del proyecto. Se emplea una metodología que avanza gradualmente, abordando cada proceso de forma modular y asegurando una implementación eficiente. Para ello, el desarrollo se divide en áreas clave, entre las que se encuentran:

- Tratamiento y adecuación de datos para la reproducción de sonido.
- Generación de audio.
- Programación multihilo.
- Interactividad mediante MIDI.
- Depuración y pruebas.

Este enfoque facilita la construcción progresiva del sistema, permitiendo primero la reproducción de un solo fotograma en el osciloscopio para, posteriormente, ampliar el alcance a la reproducción continua de una animación completa. Cada semana, se evalúan los avances en cada etapa, se planifican los pasos siguientes y se realizan los ajustes técnicos según los resultados obtenidos. De este modo, cada componente se optimiza de manera independiente y se consolida en el sistema final para asegurar un rendimiento coherente y efectivo en la reproducción de las animaciones.

4. DESARROLLO.

El desarrollo del proyecto se estructura en dos partes principales, organizadas de manera que cada etapa contribuya progresivamente a la construcción de un sistema que permita reproducir animaciones en un osciloscopio de forma continua y precisa. Estas fases abarcan los estudios previos y el desarrollo del proyecto propiamente dicho.

4.1. ESTUDIOS PREVIOS.

En esta primera parte, se realiza un análisis de las tecnologías disponibles, considerando tanto el hardware como el software necesario para alcanzar los objetivos del proyecto. Posteriormente, se definen los modelos y conceptos técnicos que sustentan el diseño del sistema, incluyendo el modelado y las herramientas específicas que facilitarán la gestión de las animaciones y su transformación en señales sonoras que reproducir en altavoces y pantalla del osciloscopio, así como el fundamento en el que se basa el código desarrollado en Python.

4.1.1. Análisis de las tecnologías disponibles.

Las tecnologías necesarias para alcanzar los objetivos del proyecto, tanto en el ámbito del hardware como del software son:

- **Hardware**: Un osciloscopio analógico (PROMAX OD-416 de 20 MHz), un dispositivo para ejecutar el programa, una tarjeta de sonido externa para enviar la señal de audio tanto al osciloscopio como a los altavoces, los altavoces, y un dispositivo de entrada interactiva, decidiéndose un teclado *MIDI* (modelo WORDLE TUNA Mini), por su facilidad de uso, tamaño compacto y disponibilidad del autor.
- **Software**: Tras evaluar distintos lenguajes de programación como Processing, Java, Python o C++, se selecciona Python por su simplicidad y ventajas técnicas, aunque el motivo de más peso es la amplia gama de bibliotecas especializadas para la generación de audio y el tratamiento de datos, así como el deseo del autor de ganar competencia en este lenguaje. Python facilita el desarrollo rápido, la integración con hardware y tecnologías externas y el manejo de grandes cantidades de datos. Entre las bibliotecas utilizadas destacan *NumPy*, *sounddevice*, *mido* o *xml.etree.ElementTree,* entre otras.

Las ventajas técnicas y prácticas de Python se pueden resumir en:

- Simplicidad y facilidad de uso: Sintaxis simple y legible, lo que facilita tanto el desarrollo como la comprensión del código.
- Amplia gama de bibliotecas especializadas, lo que permite abordar diferentes aspectos del proyecto aprovechando dichas bibliotecas, así como a optimización para el manejo eficiente de grandes cantidades de datos, permitiendo realizar operaciones complejas con arrays multidimensionales, así como el uso de las listas nativas de Python.
- Lenguaje multiplataforma, pudiendo ejecutarse en diferentes sistemas operativos.
- Procesamiento multihilo, facilitando la implementación de procesos concurrentes donde se realizan múltiples tareas al mismo tiempo, permitiendo la comunicación entre hilos en tiempo real y manteniendo una eficiencia del programa destacable.
- Comunidad activa y soporte, con documentación extensa y accesible, además de recursos y ejemplos en línea.
- Integración con tecnologías externas, interactuando fácilmente con otros lenguajes como C o C++ si se quisieran añadir componentes más intensivos en términos de procesamiento.
- Flexibilidad y escalabilidad.
- Herramientas para visualización y análisis, así como la posibilidad de interfaces gráficas de usuario (GUI).

4.1.2. Modelado.

Como ya se ha mencionado, inicialmente se plantea utilizar varios osciloscopios conjuntamente para crear efectos visuales complejos. Sin embargo, esta idea se descarta por cuestiones de alcance del proyecto. Finalmente, se decide utilizar un solo osciloscopio y emplear un sistema de preprocesado de animaciones *SVG* para optimizar el rendimiento.

El esquema adoptado incluye:

- Un ordenador portátil como dispositivo de desarrollo, con la posibilidad de migrar a un microprocesador SBC en el futuro.
- Una tarjeta de sonido externa modificada para eliminar condensadores de desacoplo, lo que evita la recolocación constante de las animaciones. La tarjeta permite desdoblar cómodamente la señal de audio, enviándola tanto al osciloscopio como a los altavoces.
- Osciloscopio y altavoces como salidas.
- Un teclado MIDI como entrada interactiva.

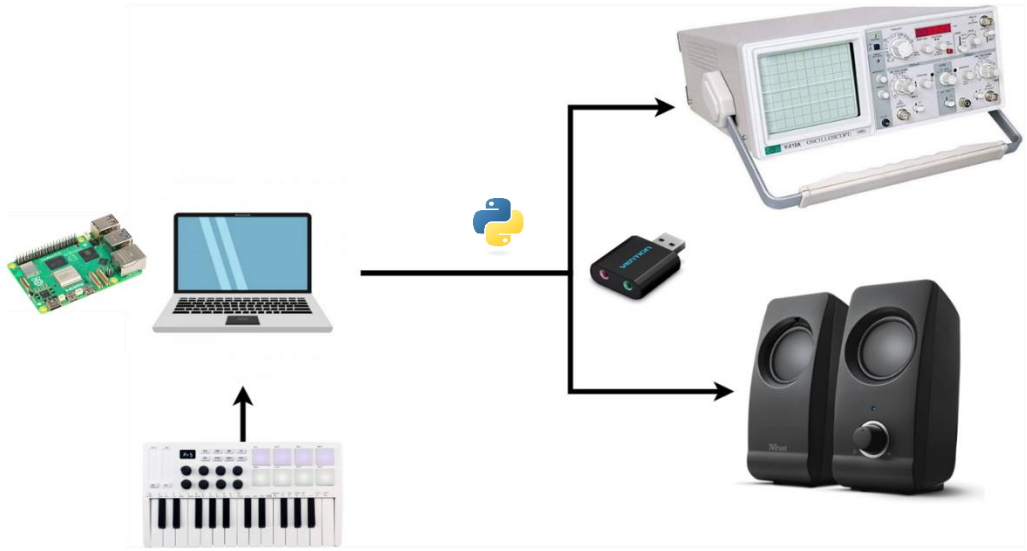


Figura 9: Esquema general del dispositivo.

4.1.3. Procedimiento de generación de las señales.

El procedimiento para generar señales de audio en este proyecto utiliza una técnica conocida como Síntesis Digital Directa (DDS, por sus siglas en inglés). La DDS es una técnica común y ampliamente aplicable para generar formas de onda de manera digital, y puede implementarse tanto en hardware como en software. En este contexto, la DDS permite crear ondas de frecuencia y fase controladas con precisión, eliminando la necesidad de resamplear constantemente los datos y optimizando el rendimiento del sistema.

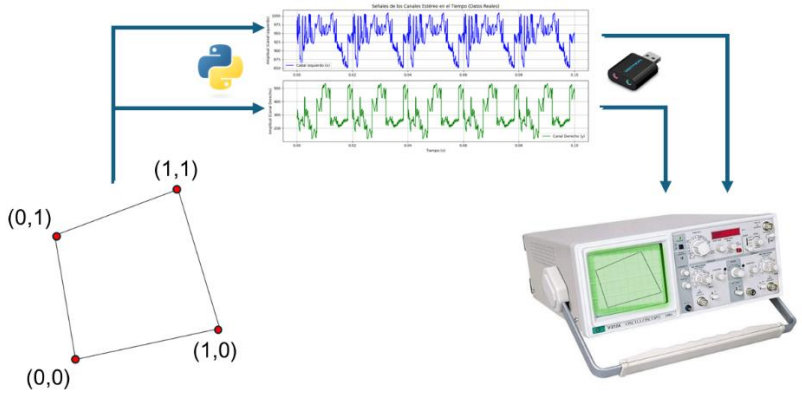


Figura 10: Esquema del flujo de tratamiento de una animación SVG en el proyecto.

Un aspecto fundamental es que la señal de audio se genera en formato estéreo, donde cada canal representa una de las coordenadas de la animación en un plano bidimensional. El canal izquierdo se corresponde con el eje *x* y el canal derecho con el eje *y*, de modo que, al enviarse a un osciloscopio analógico, los dos canales producen una representación visual de la animación original. Esta configuración permite generar una señal vectorial en la pantalla del osciloscopio, representando las trayectorias bidimensionales de la animación.

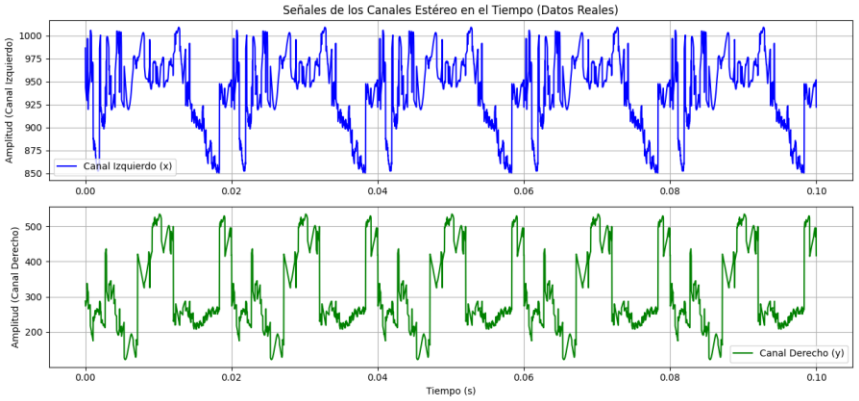


Figura 11: Señales independientes que conforman la señal estéreo.

Otro elemento clave en la DDS es el Oscilador Controlado Numéricamente (NCO), cuyo componente principal es el Acumulador de Fase. Este acumulador funciona como un contador binario de alta precisión que recorre continuamente una tabla de datos en la que se almacena la forma de onda deseada (en este caso, los puntos de una animación SVG). Cuando el contador alcanza su valor máximo, se desborda y vuelve a cero, permitiendo un ciclo continuo de la forma de onda.

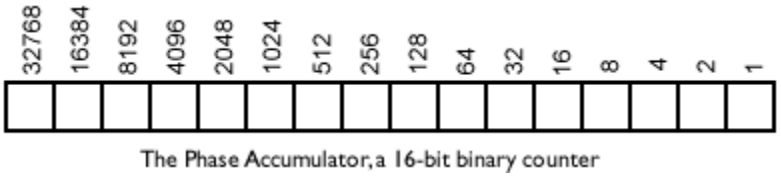


Figura 12: Acumulador de fase.

El valor de incremento de fase, también conocido como Frecuencia de Incremento, determina la velocidad con la que el acumulador recorre la tabla de datos. Cuanto mayor es el valor de incremento, más rápido el acumulador completa un ciclo y, por tanto, mayor es la frecuencia de la onda generada. En el proyecto, este incremento se ajusta en función de la frecuencia de muestreo del sistema y la frecuencia específica que se desea reproducir. De esta manera, al aumentar el incremento, la frecuencia de salida también aumenta, permitiendo una amplia gama de frecuencias ajustables de manera precisa.

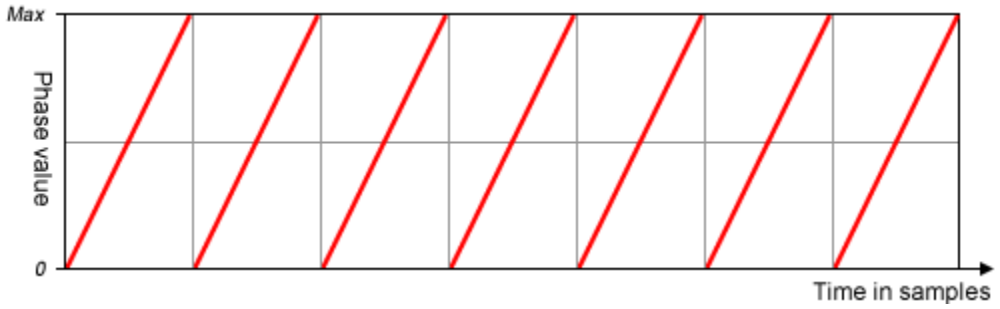


Figura 13: Acumulador de fase en el tiempo.

Para obtener un mayor rango de frecuencias, se aumenta la precisión del acumulador de fase mediante el uso de contadores de mayor tamaño (por ejemplo, pasando de 16 a 32 bits) y se ajustan los incrementos de fase en consecuencia, asegurando una precisión suficiente en los valores de frecuencia.

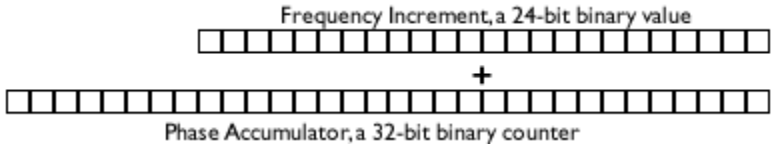


Figura 14: Aumento en la precisión del contador binario.

La ventaja fundamental del DDS es su capacidad para generar cualquier forma de onda. El valor del acumulador de fase se utiliza como dirección de acceso en una tabla de forma de onda almacenada en memoria. Esta tabla contiene los valores de la forma de onda para cada fase de entrada, permitiendo obtener una señal de salida a partir de cualquier forma de onda arbitraria almacenada en la tabla. En este proyecto, se almacenan las coordenadas de la animación SVG, que se recorren con el acumulador de fase para producir la señal deseada.

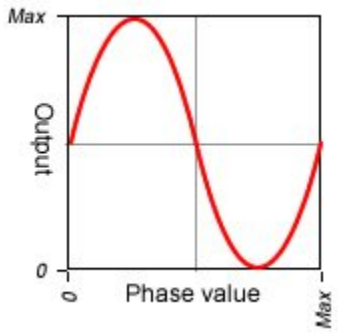


Figura 15: Señal senoidal en función de los valores del acumulador de fase.

La salida puede enviarse a un convertidor digital a analógico (DAC) para obtener una señal de audio o mantenerse en el dominio digital para su procesamiento.

$$\text{incremento} = \left\lfloor 2^{32} \times \frac{\text{frecuencia}}{F_{\text{muestra}}} \right\rfloor$$

Figura 16: Fórmula de la frecuencia de incremento.

En la práctica, el sistema no usa el valor completo del acumulador de fase como dirección en la tabla de forma de onda. En su lugar, se utilizan los bits más significativos del acumulador para acceder a la tabla, que en este caso, tiene un tamaño reducido (4096 valores). De este modo, mientras el incremento de frecuencia no sea tan grande como para alterar significativamente los bits más altos del acumulador, el sistema evita saltarse muestras en la tabla de forma de onda.

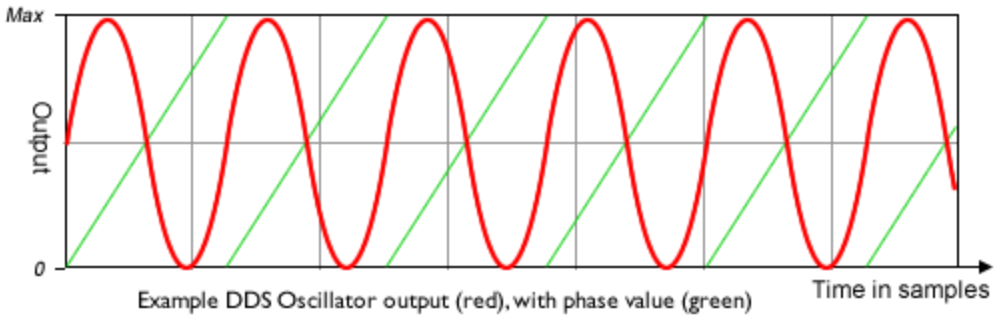


Figura 17: Representación de una onda senoidal (rojo) y del acumulador de fase en función del tiempo (verde).

Este método es eficiente y permite mantener la precisión en la generación de la señal de audio. Sin embargo, al generar frecuencias superiores a la frecuencia de Nyquist, pueden aparecer problemas de *aliasing*. El *aliasing* ocurre cuando los componentes de frecuencia en la señal generada exceden la mitad de la frecuencia de muestreo (frecuencia de Nyquist). Esto provoca que las altas frecuencias se "plieguen" o "reflejen" en el espectro audible como distorsiones no deseadas, generando tonos adicionales que no forman parte de la señal original. Para mitigarlo, es necesario mantener la frecuencia de muestreo suficientemente alta o aplicar filtros y técnicas de prefiltrado, especialmente al utilizar formas de onda que contienen componentes de frecuencia superiores a Nyquist.

Esta técnica garantiza una reproducción eficiente de la señal sonora en tiempo real, al optimizar el cálculo de incrementos del fasor en función de la frecuencia deseada. Gracias a ello, se minimiza la carga computacional durante la generación continua de la señal, lo que permite no solo una respuesta rápida, sino también una interacción fluida entre el sistema y el usuario. Además, esta optimización es fundamental para mantener una latencia baja en aplicaciones interactivas, asegurando que los cambios de frecuencia o efectos aplicados se reflejen de inmediato en el audio reproducido, proporcionando una experiencia auditiva sin interrupciones.

4.1.4. Teoría del *multithreading.*

El procesamiento multihilo o *multithreading* permite que varias tareas se ejecuten simultáneamente, evitando que una operación bloquee a las demás. En este proyecto, se utiliza el módulo *threading* de Python para gestionar de manera eficiente la carga de datos, la reproducción de audio y la interacción en tiempo real. Los hilos son útiles para tareas dependientes de E/S, como la carga de datos y la generación y reproducción de audio.

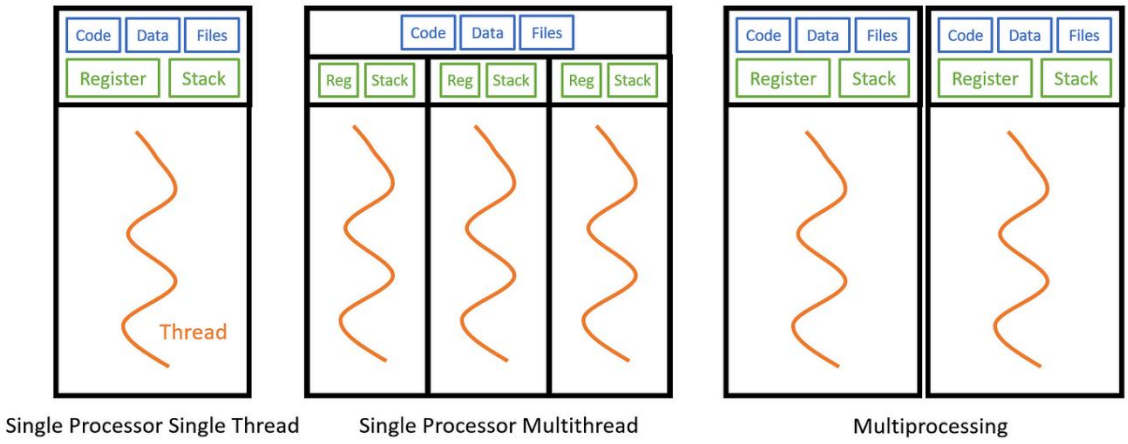


Figura 18: Comparación entre sistemas multihilo y sistemas multiproceso.

Los conceptos clave incluyen:

- **Hilos (*threads*):** Un hilo es la unidad básica de procesamiento que puede gestionar un sistema operativo. A diferencia de los procesos, que tienen su propio espacio de memoria, los hilos dentro de un mismo proceso comparten los mismos recursos y datos. Esto hace que los hilos sean más ligeros y eficientes en términos de uso de memoria y velocidad de ejecución.
- ***Multithreading vs. Multiprocessing*:** Aunque ambos conceptos permiten la ejecución de múltiples tareas simultáneamente, el *multithreading* utiliza varios hilos dentro de un solo proceso, mientras que el *multiprocessing* crea múltiples procesos, cada uno con su propio espacio de memoria. El *multithreading* es más adecuado cuando las tareas necesitan compartir datos de manera eficiente, mientras que el *multiprocessing* es preferible cuando se busca mayor independencia entre las tareas.
- **Contexto de uso:** Este enfoque es especialmente útil en sistemas que requieren ejecutar múltiples tareas a la vez. En el caso de un proyecto que involucra la manipulación y reproducción de animaciones SVG, el *multithreading* permite que el procesamiento de una animación ocurra en segundo plano mientras se reproduce otra, mejorando la interactividad y la fluidez del sistema.
- **Tipos de hilos:** Existen principalmente dos tipos de hilos en Python:
  - **Hilos de usuario:** Son gestionados directamente el programa y son los más comunes en el uso del módulo *threading*. Necesitan ser iniciados y normalmente gestionados mediante el comando *join()* para poder sincronizarlos con el resto de los hilos que haya ejecutándose.
  - **Hilos '*daemon*':** Estos son hilos especiales que se ejecutan en segundo plano y no bloquean la terminación del programa principal. Cuando todos los hilos *no-daemon* han terminado, el programa finaliza, incluso si los hilos *daemon* siguen activos. Esto los hace útiles para tareas que deben ejecutarse en el fondo, como la monitorización o la escucha de eventos de E/S, pero cuya finalización no es crítica para el cierre correcto del programa. En este proyecto, el hilo de escucha de teclado se establece como hilo *deamon* por su carácter de escucha en segundo plano, así como la gestión MIDI.

Desafíos del *multithreading*:

- **Condiciones de carrera** (*race conditions*): Se producen cuando varios hilos intentan acceder y modificar una variable compartida al mismo tiempo, lo que puede generar comportamientos impredecibles. Para evitar estos problemas, se utilizan mecanismos de sincronización como *locks* (bloqueos) o semáforos, que controlan el acceso a los recursos compartidos.
- ***Deadlocks***: Un *deadlock* ocurre cuando dos o más hilos se bloquean mutuamente al esperar recursos que están siendo utilizados por otro hilo, lo que puede detener la ejecución del programa. El diseño adecuado del flujo de control y la gestión eficiente de los recursos compartidos son esenciales para prevenir estos bloqueos.
- **Dificultad de depuración**: Debido a que los hilos se ejecutan de manera concurrente, los errores pueden ser difíciles de identificar y depurar, ya que no siempre ocurren de manera predecible o reproducible.

4.2. DESARROLLO DEL PROYECTO. EXPLICACIÓN DEL CÓDIGO.

Para reproducir animaciones *SVG* en un osciloscopio analógico, generando ondas sonoras derivadas de las mismas animaciones, se divide el código en dos partes: Primero se preprocesan estas animaciones para adecuarlas a un formato óptimo para las técnicas de reproducción con DDS, y posteriormente, la gestión del programa para la generación de audio y reproducción de las animaciones, englobado bajo lo que se denomina postprocesado.

4.2.1. Esquemático.

Figura 19: Esquema general del software implementado.

El programa consta de dos partes fundamentales:

- **Preprocesado**: Incluye la generación de animaciones y el análisis y transformación de datos para adecuarlos al formato necesario para la reproducción.
- **Postprocesado**: En este bloque se ejecutan los procesos necesarios para el control, reproducción y actualización de las ondas de sonido generadas a partir de los datos obtenidos del bloque de preprocesado.

4.2.2. Bloque de preprocesado.

El bloque de preprocesado se encarga de transformar las animaciones SVG en datos numéricos optimizados para la reproducción en un osciloscopio y la generación de ondas de audio. Este preprocesado se realiza una única vez para cada animación, y los resultados se guardan en archivos comprimidos (.npz) que permiten una carga rápida y eficiente en la fase de reproducción.

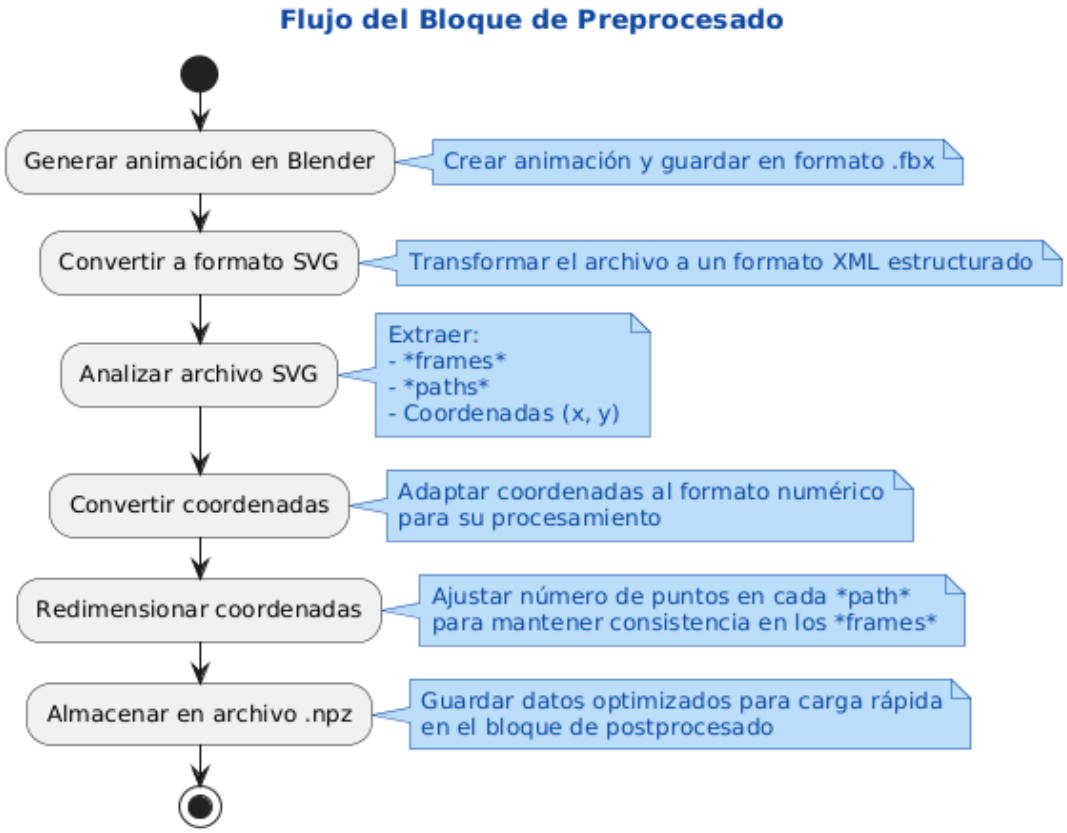


Figura 20: Diagrama de flujo del bloque de preprocesado.

La estructura de este bloque se divide en cinco etapas principales:

- Generación de las animaciones.
- Análisis de los archivos SVG.
- Redimensionamiento de coordenadas.
- Concatenación de trayectorias y ajuste de longitud total
- Almacenamiento en formato comprimido.

4.2.2.1. Generación de las animaciones.

El primer paso implica crear las animaciones en Blender, un software de diseño 3D de código abierto ampliamente utilizado para la creación de gráficos animados. En este proyecto, Blender se emplea para generar las trayectorias de las animaciones en un formato intermedio (FBX), que luego se convierte en formato SVG.

El formato SVG es un estándar para gráficos vectoriales que almacena las animaciones en una estructura jerárquica basada en XML. Esta estructura divide la animación en elementos como fotogramas (*frames*), trayectorias (*paths*) y coordenadas individuales (x, y) para cada punto en la trayectoria. La elección del formato SVG es clave, ya que facilita la manipulación de los datos y permite organizar la información de manera estructurada, lo que resulta ventajoso para el procesamiento posterior.

En la Figura 21, se puede apreciar el primer conjunto de coordenadas dentro de una trayectoria y a su vez, dentro del primer fotograma de la animación.

```
<?xml version='1.0' encoding='ascii'?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
version="1.1" width="1920" height="1080">
    <g id="ViewLayer_LineSet" inkscape:groupmode="lineset" inkscape:label="ViewLayer_LineSet">
        <g id="frame_0001" inkscape:groupmode="frame" inkscape:label="frame_0001">
            <g inkscape:groupmode="layer" id="strokes" inkscape:label="strokes">
                <path fill="none" stroke-width="3.0" stroke-linecap="butt" stroke-opacity="1.0"
stroke="rgb(0, 0, 0)" stroke-linejoin="round" d=" M 988.642, 357.608 985.509, 356.346 985.177, 356.221
984.475, 355.958 " />
```

Figura 21: Estructura de árbol en archivo XML.

4.2.2.2. Análisis del archivo SVG.

Una vez que se generan las animaciones y se guardan en formato SVG, el siguiente paso consiste en analizar estos archivos para extraer sus componentes clave. El análisis del archivo SVG tiene como objetivo desglosar la información jerárquica en elementos procesables: fotogramas, trayectorias y las coordenadas de cada punto de la trayectoria.

Figura 22: Fotograma de una de las animaciones SVG creadas.

El formato XML del archivo SVG organiza estos elementos en un árbol jerárquico, donde los nodos representan fotogramas y trayectorias, y las hojas del árbol contienen las coordenadas (x, y) de cada punto. Para acceder a estos datos, se desarrollan funciones específicas que recorren esta estructura en busca de los datos relevantes. Durante el proceso de análisis, se extraen las coordenadas en una lista para cada fotograma, facilitando así su manipulación en los pasos siguientes.

Además, es necesario realizar una conversión de formato para adaptar las coordenadas al uso en Python, ya que los datos SVG incluyen caracteres especiales (como el carácter 'M' al inicio de cada serie de coordenadas) y están separados por espacios. Estas adaptaciones son fundamentales para obtener un formato numérico accesible que pueda ser procesado directamente en las etapas posteriores.

- **Funciones clave en esta etapa**
  - `obtener_frames(nombre)`**:** Esta función toma el nombre de un archivo SVG y utiliza la biblioteca *xml.etree.ElementTree* para analizar su estructura. Navega por el árbol XML, buscando los elementos correspondientes a cada fotograma y sus trayectorias, y descompone la animación en una lista de listas donde cada sublista representa las trayectorias de un fotograma. Para cada trayectoria, se extrae el atributo ‘d’ del SVG, que contiene la secuencia de coordenadas.

- `string_a_lista(coordenadas_str)`: Las coordenadas extraídas de cada trayectoria se encuentran en formato de cadena, con un carácter 'M' inicial y separadas por espacios. Esta función convierte esta cadena en una lista de pares de coordenadas (x, y), eliminando el carácter 'M' y separando cada par para crear un array numérico que puede ser manipulado en las siguientes etapas. El array resultante se devuelve en tipo *Numpy* para un manejo eficiente de los datos.

Estas funciones preparan los datos de cada fotograma para el siguiente paso, donde se ajusta la cantidad de puntos en cada trayectoria.

4.2.2.3. Redimensionamiento del vector

El último paso en el preprocesado es el redimensionamiento de las coordenadas. Debido a que los fotogramas extraídos pueden tener un número variable de puntos en cada trayectoria, es importante normalizar esta cantidad para asegurar una reproducción visual coherente y un rendimiento optimizado en el osciloscopio.

El proceso de redimensionamiento consiste en ajustar cada trayectoria para que tenga una cantidad fija de puntos, manteniendo la forma visual de la trayectoria original. Para ello, se calculan las distancias entre los puntos y se distribuyen los puntos adicionales de manera proporcional, según estas distancias. Esto garantiza que la densidad de puntos sea uniforme y que la forma original de la trayectoria no se vea alterada.

Este ajuste se realiza mediante funciones específicas que toman como entrada una lista de puntos de la trayectoria y la cantidad deseada de puntos. El resultado es un conjunto de trayectorias con un número homogéneo de puntos en cada fotograma, lo que facilita una reproducción sincronizada y sin desajustes.

- **Funciones clave en esta etapa:**
  - `calcular_distancias(path_list)`: Esta función toma una lista de trayectorias y calcula las distancias entre cada par de puntos consecutivos en cada trayectoria. Las distancias calculadas permiten distribuir los puntos de manera uniforme en el paso de redimensionamiento, preservando la forma visual de cada trayectoria. La función devuelve una lista de arrays que representan las distancias de cada trayectoria.

- `redimensiona(vector, nueva_longitud, distancias)`: Basándose en las distancias calculadas, esta función redimensiona cada trayectoria a una longitud específica definida por `nueva_longitud`. Si la trayectoria tiene menos puntos que la longitud deseada, la función añade puntos intermedios, distribuyéndolos de acuerdo con las distancias entre los puntos originales. Este enfoque asegura que los puntos adicionales se distribuyan proporcionalmente a lo largo de la trayectoria, manteniendo la integridad visual de la animación.

Estas funciones son fundamentales para normalizar las trayectorias en cada fotograma y asegurar una representación visual y sonora coherente.

4.2.2.4. Concatenación de trayectorias y ajuste de longitud total.

Una vez redimensionadas, las trayectorias deben concatenarse en una única estructura que represente el fotograma completo. Dado que cada fotograma tiene un número uniforme de puntos, el proceso de concatenación garantiza que todos los fotogramas mantengan una estructura similar.

- **Función clave en esta etapa:**
  - `redimensiona_y_concatena(path_list, nueva_longitud)`: Esta función toma una lista de trayectorias redimensionadas y las concatena para formar un único array que representa el fotograma completo. Calcula la longitud deseada para cada trayectoria en función de la distancia total de cada una y ajusta el tamaño final del fotograma, asegurando que contenga exactamente `nueva_longitud` puntos. Este ajuste es clave para mantener la consistencia en todos los fotogramas de la animación.

4.2.2.5. Almacenamiento en archivos .npz

Finalmente, los datos procesados se almacenan en archivos comprimidos con extensión .npz. Estos archivos contienen las trayectorias y coordenadas en un formato optimizado, lo que permite cargarlos rápidamente en el sistema de reproducción. Al utilizar archivos .npz, el sistema evita recalcular las coordenadas de cada animación cada vez que se ejecuta el programa, reduciendo significativamente el tiempo de carga y permitiendo que el sistema se concentre en la reproducción en tiempo real.

- **Función clave en esta etapa:**
  - `procesa_multiples_animaciones(archivos_svg, nueva_longitud)`**:** Esta función actúa como el núcleo del preprocesado, aplicando todas las funciones anteriores para cada archivo SVG en *`archivos_svg`*. Para cada fotograma de cada animación, se extraen y redimensionan las trayectorias y luego se almacenan en un archivo .npz. Cada archivo .npz contiene los datos optimizados de todos los fotogramas de la animación, permitiendo su acceso rápido y eficiente en la fase de postprocesado.

Este enfoque permite liberar recursos del sistema para otras tareas, como la sincronización de audio y el control en tiempo real mediante MIDI, asegurando una experiencia de usuario fluida. Además, la modularidad de las funciones de preprocesado y el uso de bibliotecas como NumPy y ElementTree contribuyen a la flexibilidad y rendimiento del sistema, permitiendo una ejecución optimizada y adaptable.

4.2.3. Bloque de postprocesado.

El bloque de postprocesado realiza la generación y reproducción de animaciones visuales en el osciloscopio y en los altavoces, aprovechando varias estructuras de datos e hilos que operan de manera concurrente. Esta organización en hilos permite que el sistema reproduzca las animaciones en tiempo real y responda de forma fluida a la interacción del usuario. Se estructura en tres secciones principales: diccionarios e inicializaciones, funciones declaradas y estructura de hilos.

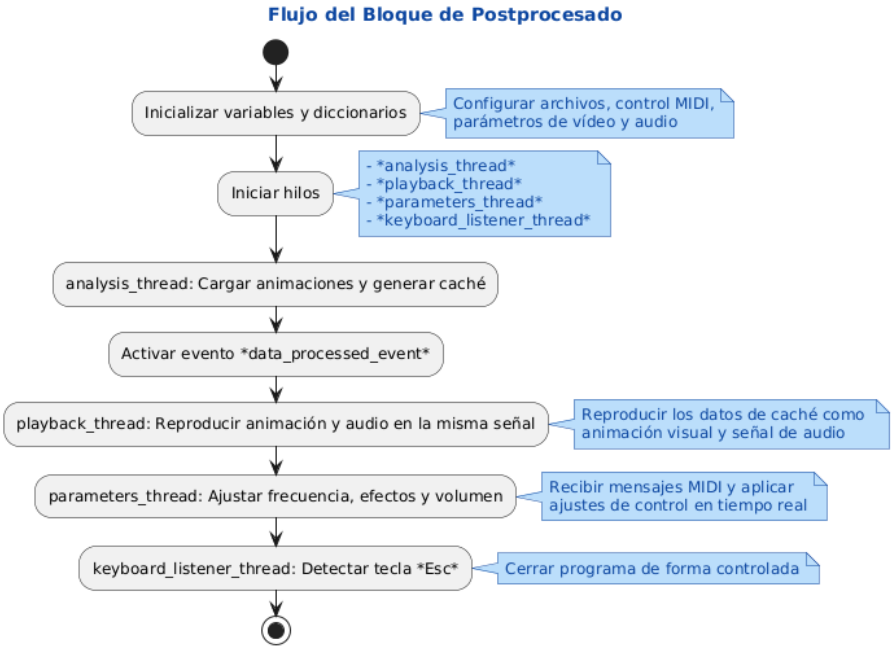


Figura 23: Diagrama de flujo del bloque de postprocesado.

4.2.3.1. Diccionarios, inicializaciones y eventos de hilos

Los diccionarios y variables globales permiten definir los parámetros de configuración y gestionar el estado del sistema. Además, se incluyen eventos que facilitan la coordinación entre los hilos:

- **Diccionarios de archivos y parámetros de control**
  - `files_npz`: Asocia nombres de animaciones a los archivos .npz preprocesados que contienen los datos para su reproducción. Esta estructura permite un acceso rápido a los datos almacenados en caché, lo que mejora la eficiencia del sistema.
  - `midi_parameters`: Contiene parámetros de configuración del audio, como la frecuencia de muestreo y el volumen. Estos valores iniciales controlan la reproducción de sonido generada a partir de las coordenadas de las animaciones. También define los parámetros de frecuencia para el control mediante MIDI, además de los efectos visuales que se aplicarán a las animaciones.
  - `video_parameters`: Incluye los valores iniciales de reproducción de animación, como la velocidad en fotogramas por segundo (*fps*), la animación seleccionada y el efecto aplicado.
- **Variables de control de audio y animación**
  - Variables como `phasor`, `frame_idx`, `rotation`, `increment`, y `scale` regulan la reproducción de audio y la transformación de la animación. `phasor` actúa como contador para la tabla de datos de la animación, mientras que `frame_idx` rastrea el fotograma actual.
  - `current_wave`: Tabla para almacenar las coordenadas de la animación en reproducción, normalizadas para facilitar la salida de audio.
- **Eventos de coordinación**
  - `data_processed_event`: Indica que la caché de animaciones ha sido creada y los datos están listos para ser utilizados por los hilos de reproducción y control.
  - `data_playback_event`: Señala el inicio de la reproducción de las animaciones, activándose una vez que todos los datos necesarios están listos.

4.2.3.2. Funciones declaradas

Las funciones en este bloque son esenciales para diversas tareas que van desde la conversión de entradas MIDI y la aplicación de efectos visuales y de audio, hasta la reproducción precisa de las animaciones mediante técnicas de síntesis digital directa (DDS). Estas funciones no solo convierten señales MIDI en valores de frecuencia y ajustan parámetros visuales y sonoros, sino que también son responsables de coordinar la reproducción fluida de las animaciones en el osciloscopio, asegurando que los datos de las coordenadas se traduzcan en señales de audio. Las principales funciones en este bloque son las siguientes:

- **Funciones de conversión y ajuste de parámetros**: Estas funciones permiten adaptar las entradas MIDI y ajustar en tiempo real los parámetros de la animación y el audio.
    - `midi_note_to_frequency(note)`: Convierte una nota MIDI en la frecuencia correspondiente, necesaria para ajustar la frecuencia de reproducción de audio según las notas tocadas en el teclado MIDI. Esta función permite que las animaciones se sincronicen con los cambios en frecuencia impuestos por el usuario, recreando variaciones tonales en tiempo real.
    - `handle_control_change(control, value)`: Ajusta parámetros en respuesta a mensajes de control MIDI, aplicando efectos en la animación y el audio, como el volumen o efectos visuales de rotación y redondeo. Esto da al usuario control directo sobre las características visuales y sonoras a través de perillas y deslizadores, logrando una experiencia de interacción dinámica.
- **Funciones de procesamiento de señales y reproducción de audio**: Estas funciones son clave en el manejo de la técnica *DDS*, generando ondas de sonido directamente a partir de las coordenadas (x, y) de la animación.
    - `adjust_volume(value)`: Modifica el volumen general de la salida de audio en función del valor de entrada recibido, lo que permite cambios en tiempo real desde la interfaz MIDI.
    - `scale_and_rotate(frame, scale_factor, rotation_degrees)`: Aplica transformaciones a las coordenadas de cada fotograma, ajustando el tamaño y la orientación de la animación en el osciloscopio para representar visualmente los efectos solicitados por el usuario.
    - `normalize(frame)`: Escala las coordenadas de un fotograma a un rango estándar, asegurando que las señales de audio generadas sean proporcionales y no provoquen distorsiones visuales ni auditivas durante la reproducción.

  - `get_audio_buffer_from_wave(bits_idx,incr,tabla_datos_xy)`: Genera el *buffer* de audio llenándolo con las coordenadas de la animación, donde el fasor, basado en la técnica de *DDS*, recorre los datos de la tabla de ondas. Este enfoque garantiza que la señal se genere a la velocidad correcta en función de la frecuencia configurada, manteniendo la sincronización entre audio y visualización.
  - `callback(outdata, frames, time, status)`: Función de *callback* que actualiza el *buffer* de salida del flujo de audio en cada iteración, ejecutándose durante toda la reproducción. Esta función asegura que, si el programa se cierra, el *buffer* se rellene con ceros, deteniendo el audio sin generar picos o sonidos abruptos.
- **Funciones de control de flujo y finalización**: Estas funciones regulan la ejecución general del sistema y permiten que el programa finalice correctamente al recibir una instrucción de cierre.
  - `compute_incremento(freq)`: Calcula el incremento de fasor necesario para recorrer los datos de la tabla a la velocidad de reproducción deseada, permitiendo que el sistema mantenga la frecuencia de salida de audio sincronizada con la animación en el osciloscopio.
  - `stop_program()`: Detiene todos los hilos de ejecución y finaliza el programa de manera controlada, lo cual es esencial para cerrar el sistema de manera ordenada, sin dejar procesos activos.

4.2.3.3. Estructura de hilos.

Para gestionar la reproducción y control de las animaciones y el audio, el sistema implementa una serie de hilos que operan de manera independiente y coordinada.

- **Hilo de Análisis** (`analysis_thread`): Carga los archivos `.npz` generados en el preprocesado y crea una caché en `animation_cache`. Esto asegura que los datos de las animaciones estén disponibles en memoria para un acceso rápido, mejorando la eficiencia de reproducción. Cuando la caché está completa, el hilo activa `data_processed_event`, permitiendo que otros hilos comiencen a utilizar los datos.
- **Hilo de Reproducción** `(playback_thread)`**:** Maneja la reproducción visual de las animaciones y la salida de audio en el osciloscopio. Utiliza los datos de la caché para actualizar los fotogramas de la animación y generar señales de audio. Este hilo controla la velocidad de reproducción y aplica efectos visuales como rotación y escalado mediante las funciones de ajuste mencionadas. La función de *callback* garantiza una salida de audio continua y controlada.

- **Hilo de Parámetros** `(parameters_thread)`**:** Recibe los mensajes MIDI para permitir ajustes en tiempo real. Este hilo gestiona las entradas de frecuencia y efectos visuales, aplicando cambios que reflejan la interacción del usuario en tiempo real. Por ejemplo, cuando se toca una nota en el teclado MIDI, el sistema ajusta la frecuencia de reproducción; también se pueden modificar el volumen y los efectos visuales con los controles de perillas y deslizadores.
- **Hilo de Escucha de Teclado** `(keyboard_listener_thread)`**:** Detecta la pulsación de la tecla de salida (Esc) para detener el programa de manera controlada. Este hilo permanece activo hasta que el usuario desea finalizar el programa y asegura que todos los hilos se detengan correctamente.
- **Función Principal** `(main())`**:** Inicia los hilos descritos y coordina la ejecución concurrente del sistema. Se asegura de que cada hilo esté en funcionamiento y supervisa el estado general del programa hasta su finalización.

Esta organización permite que el bloque de Postprocesado gestione la reproducción de animaciones y audio de manera sincronizada y eficiente, con controles en tiempo real y la capacidad de adaptarse a la interacción del usuario.

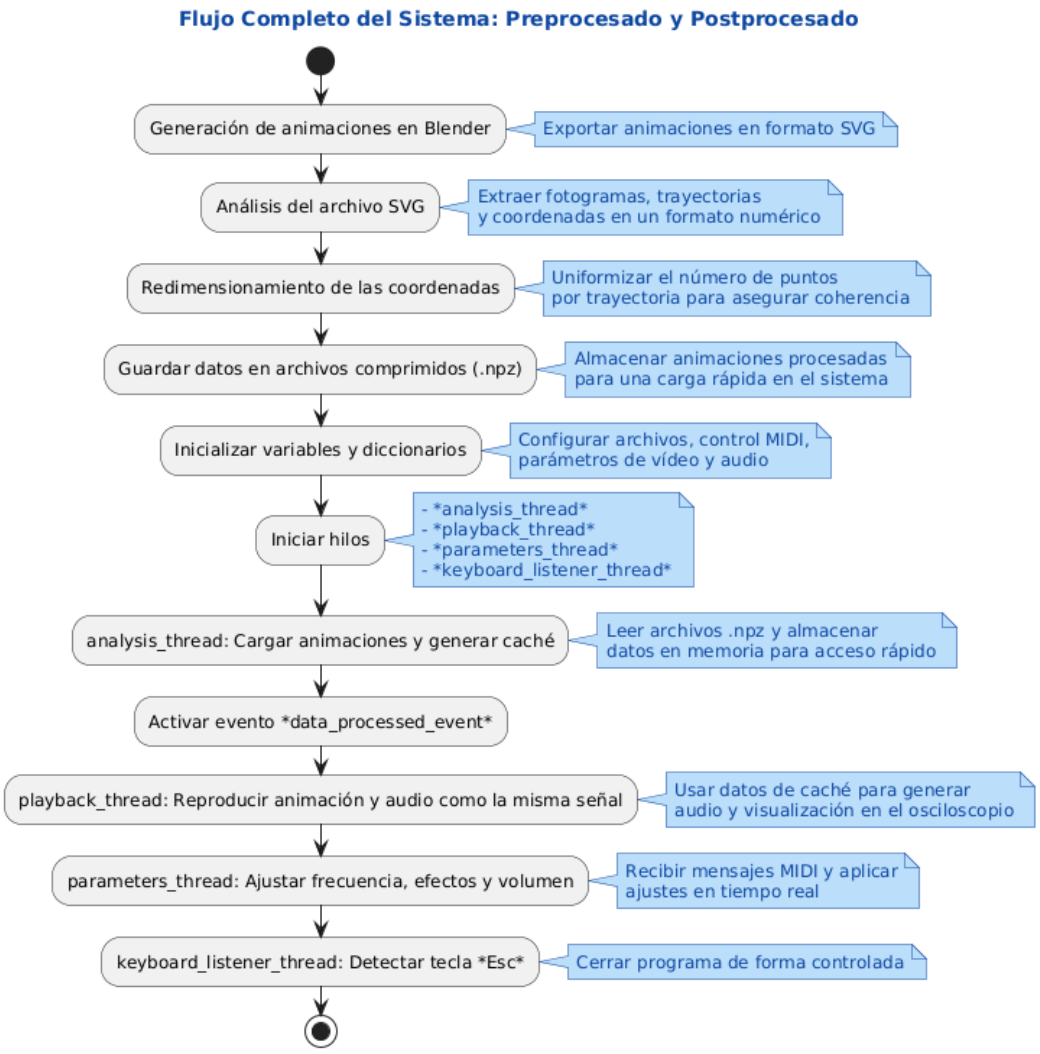


Figura 24: Diagrama de flujo del sistema completo.

5. RESULTADOS Y DISCUSIÓN.

5.1. BANCO DE PRUEBAS.

Para respaldar los resultados mencionados en el apartado anterior, se llevan a cabo diversos bancos de pruebas con el objetivo de analizar y verificar la robustez del código. Estas pruebas son de distinta naturaleza y se detallan a continuación:

- **Prueba de procesamiento de audio.**

Esta prueba tiene como objetivo evaluar el tiempo necesario para procesar los datos de audio generados a partir de animaciones SVG preprocesadas. Se mide el tiempo requerido para procesar cada fotograma de las animaciones cargadas desde archivos .npz. Los resultados incluyen el tiempo total de procesamiento por animación, así como el promedio y la desviación estándar para un análisis detallado del rendimiento.

Iteración	Animación	Tamaño del Archivo (bytes)	Frames Procesados	Tiempo Total de Procesamiento (s)	Estado
1	baile1	2393582	73	0.22134113311767578	Éxito
1	baile2	3987159	117	0.2706751823425293	Éxito
1	break1	3032474	106	0.2953832149505615	Éxito
1	break2	6799275	203	0.3727684020996094	Éxito
2	baile1	2393582	73	0.1679389476776123	Éxito
2	baile2	3987159	117	0.21997356414794922	Éxito
2	break1	3032474	106	0.1865220069885254	Éxito
2	break2	6799275	203	0.43979382514953613	Éxito
3	baile1	2393582	73	0.13170075416564941	Éxito
3	baile2	3987159	117	0.24022126197814941	Éxito
3	break1	3032474	106	0.20274901390075684	Éxito
3	break2	6799275	203	0.5713624954223633	Éxito
4	baile1	2393582	73	0.2879042625427246	Éxito
4	baile2	3987159	117	0.4447050094604492	Éxito
4	break1	3032474	106	0.3090023994445801	Éxito
4	break2	6799275	203	0.5493638515472412	Éxito
5	baile1	2393582	73	0.2815377712249756	Éxito
5	baile2	3987159	117	0.3653090000152588	Éxito
5	break1	3032474	106	0.3874039649963379	Éxito
5	break2	6799275	203	0.8569269180297852	Éxito
Promedio	**baile1**			0.21808457374572754	
Desviación Estándar	**baile1**			0.061462916448863245	
Promedio	**baile2**			0.3081768035888672	
Desviación Estándar	**baile2**			0.08450819788233396	
Promedio	**break1**			0.27621212005615237	
Desviación Estándar	**break1**			0.0738207342568556	
Promedio	**break2**			0.558043098449707	
Desviación Estándar	**break2**			0.16606858644122027	

Tabla 1: Resultados de la prueba de procesamiento de audio.

- **Prueba de ejecución concurrente.**

Se verifica la capacidad del sistema para ejecutar hilos de manera concurrente, abarcando el análisis de animaciones, la reproducción de audio y la escucha de mensajes MIDI. Durante esta prueba, se mide el tiempo total de ejecución de todos los hilos y se registra el estado final de cada uno, demostrando la robustez del sistema al manejar múltiples tareas simultáneamente sin errores.

Iteration	Duración Analysis (s)	Duración Keyboard Listener (s)	Duración MIDI Listener (s)	Duración Playback (s)
1	0,621243477	4,924374819	4,944147825	5,05784893
2	0,892325401	7,569397688	7,507690907	7,677013636
3	1,17035675	4,654243946	4,57445097	4,763260841
4	0,91290164	4,98423028	5,005237818	5,143358231
5	0,976148367	4,256570816	4,263954639	4,450495243
Promedio	**0,914595127**	**5,27776351**	**5,259096432**	**5,418395376**
Desviación estándar	**0,176497483**	**1,174243901**	**1,155756076**	**1,155251275**

Tabla 2: Resultados de la prueba de ejecución concurrente.

- **Prueba de uso de *CPU*.**

El objetivo de esta prueba es analizar el uso de la CPU durante la ejecución de las tareas principales del proyecto. Se registra el porcentaje de uso de la CPU y el tiempo total de CPU consumido por los hilos principales. Los resultados muestran el uso promedio de la CPU y su desviación estándar, proporcionando una visión clara del impacto del proyecto en los recursos del sistema.

Timestamp (s)	CPU Usage (%)
0.0	19
1.0	21,8
2.0	10,9
3.0	4,3
4.0	3,5
5.0	3,9
6.01	3,5
7.01	5,3
8.01	4,1
9.01	6,2
10.01	2,1
11.01	1,8
12.01	2,9
13.01	0,8
14.01	2,1
15.01	1,6
16.02	7,9
17.02	1,4
18.02	1
19.02	1,7
20.02	2,5
21.02	1,6
22.02	1,8
23.02	1,2
24.02	1,2
25.03	1,4
26.03	1
27.03	0,4
28.03	1,4
29.03	1,5
Promedio	**3,993333333**

Tabla 3: Resultado de la prueba de uso de CPU.

- **Prueba de carga de animaciones.**

Esta prueba evalúa el tiempo necesario para cargar las animaciones preprocesadas desde archivos .npz. Se mide el tiempo de carga para cada archivo individualmente, y los resultados incluyen el tiempo promedio de carga y la variabilidad, lo que permite evaluar la eficiencia del proceso de carga de datos.

Iteración	Archivo	Tiempo de Carga (s)
1	baile1_redimensionado.npz	0.17244505882263184
1	baile2_redimensionado.npz	0.2446889877319336
1	break1_redimensionado.npz	0.21349096298217773
1	break2_redimensionado.npz	0.28992390632629395
2	baile1_redimensionado.npz	0.09793233871459961
2	baile2_redimensionado.npz	0.21640467643737793
2	break1_redimensionado.npz	0.18709135055541992
2	break2_redimensionado.npz	0.3018808364868164
3	baile1_redimensionado.npz	0.14786982536315918
3	baile2_redimensionado.npz	0.151289701461792
3	break1_redimensionado.npz	0.11517477035522461
3	break2_redimensionado.npz	0.29056572914123535
4	baile1_redimensionado.npz	0.11003279685974121
4	baile2_redimensionado.npz	0.2266404628753662
4	break1_redimensionado.npz	0.18921828269958496
4	break2_redimensionado.npz	0.2810049057006836
5	baile1_redimensionado.npz	0.08914709091186523
5	baile2_redimensionado.npz	0.18084931373596191
5	break1_redimensionado.npz	0.1696302890777588
5	break2_redimensionado.npz	0.2388477325439453
Promedio	Todos	0.19570645093917846
Desviación Estándar	Todos	0.0643993075519293

Tabla 4: Resultado de la prueba de carga de animaciones

- **Prueba de interacción MIDI.**

Evalúa la capacidad del sistema para procesar entradas MIDI. Se mide el tiempo de respuesta a cada mensaje MIDI, proporcionando un análisis detallado del rendimiento del sistema en términos de latencia. Los resultados incluyen el tiempo promedio de respuesta y su variación, asegurando una interacción fluida con dispositivos MIDI.

Iteración	Mensaje MIDI	Tiempo de Respuesta (s)
1	0	0.0
1	1	0.0
1	2	0.0
1	3	0.0
1	4	0.0
1	5	0.0
1	6	0.0
1	7	0.0
1	8	0.0
1	9	0.0
2	0	0.0
2	1	0.0
2	2	0.0
2	3	0.008507728576660156
2	4	0.0
2	5	0.0
2	6	0.0
2	7	0.0
2	8	0.0
2	9	0.0
3	0	0.0
3	1	0.0
3	2	0.0009002685546875
3	3	0.0
3	4	0.0
3	5	0.0
3	6	0.0
3	7	0.0
3	8	0.0
3	9	0.0
Promedio	Todos	0.0003135999043782552
Desviación Estándar	Todos	0.001530158837113513

Tabla 5: Resultados de la prueba de interacción MIDI.

- **Prueba de preprocesado de animaciones.**

Esta prueba tiene como objetivo evaluar la eficiencia del preprocesado de animaciones SVG. Se mide el tiempo de preprocesado necesario para convertir y redimensionar las animaciones, además de verificar la correcta generación de los archivos .npz. Los resultados incluyen el tiempo promedio de preprocesado y el éxito en la generación de los archivos, asegurando la preparación adecuada de los datos para su posterior uso.

Iteración	Tiempo de Preprocesado (s)	Archivos Generados Correctamente	Archivos Faltantes
1	87,989	4	0
2	84,508	4	0
3	105,221	4	0
4	198,138	4	0
5	196,400	4	0
Promedio	134,451		
Desviación Estándar	51,771		

Tabla 6: Resultados de la prueba de preprocesado

5.2. DISCUSIÓN DE RESULTADOS.

Los resultados obtenidos en este proyecto han demostrado obtener un dispositivo robusto y eficiente, cumpliendo con las expectativas en cuanto a la tasa de refresco y actualización establecida en los objetivos iniciales. La implementación de la estructura multihilo, junto con el preprocesado de las animaciones y la jerarquía optimizada del código, ha permitido que las animaciones se visualicen de manera casi instantánea al ejecutar el programa. La tasa predeterminada de 25 fotogramas por segundo garantiza una visualización fluida en el osciloscopio, aunque es posible modificar este valor para adaptarlo a diferentes efectos visuales. Un aumento en los fotogramas por segundo (más de 30) produce una sensación de 'cámara rápida', mientras que una reducción por debajo de los 25 genera un efecto de 'cámara lenta'. Sin embargo, al bajar de este umbral, la fluidez del vídeo disminuye, haciendo que las animaciones se perciban como una sucesión de imágenes estáticas en lugar de un flujo continuo, como era de esperar.

En cuanto a la generación de señales de audio, se ha logrado una reproducción sin interrupciones ni interferencias. El cambio en el tono de la señal, causado por la modificación de la frecuencia al pulsar las teclas del teclado *MIDI*, ocurre de forma inmediata, brindando una experiencia interactiva muy satisfactoria. A frecuencias bajas (menos de 100 Hz), se genera un efecto visual interesante en el osciloscopio, con un rastro que sigue al rayo catódico en pantalla. A frecuencias más altas (superiores a 120 Hz), las animaciones tienden a deformarse debido a la velocidad con la que el rayo recorre las trayectorias de los fotogramas, generando una distorsión visual entre los puntos finales e iniciales de las trayectorias. Para mitigar este problema, se plantea la posibilidad de implementar una nueva tarjeta de sonido que permita controlar el tercer canal de los osciloscopios analógicos, el cual es clave para gestionar el rayo catódico.

En relación a los efectos visuales, se ha conseguido una buena respuesta de las perillas y deslizadores del teclado *MIDI*, aunque la calidad básica del hardware utilizado (Worlde Tuna MINI) limita la precisión de los ajustes. Con la adquisición de un teclado *MIDI* de mayor calidad, la experiencia interactiva con los efectos podría verse significativamente mejorada.

En resumen, el proyecto ha logrado cumplir con los objetivos técnicos y artísticos propuestos. Se ha implementado un sistema que ofrece una reproducción fluida entre de las animaciones y el audio, con una alta capacidad de respuesta en tiempo real. Aunque existen áreas de mejora, especialmente en términos de *hardware* (como el control del rayo catódico y un teclado *MIDI* más avanzado), los resultados obtenidos demuestran el éxito del diseño y plantean un camino claro para futuras mejoras y expansiones del proyecto.

6. CONCLUSIONES.

6.1. CONCLUSIONES TÉCNICAS.

El proyecto demuestra la viabilidad de reutilizar osciloscopios analógicos para fines artísticos, transformándolos en plataformas visuales y sonoras mediante la generación y reproducción de señales de audio derivadas de animaciones vectoriales. A través de la implementación de técnicas avanzadas como la Síntesis Digital Directa (DDS) y un sistema multihilo, se logra una visualización precisa en el osciloscopio y una reproducción sonora en los altavoces.

El preprocesado de las animaciones se revela como un componente esencial para optimizar el rendimiento del sistema, permitiendo una carga rápida y eficiente de datos en tiempo real. Los resultados obtenidos reflejan la estabilidad y eficiencia del sistema, destacando su capacidad para procesar y reproducir animaciones con un uso reducido de recursos, lo que facilita la interacción en tiempo real mediante dispositivos MIDI.

El diseño modular del software garantiza la escalabilidad y adaptabilidad del sistema, permitiendo la incorporación futura de nuevos efectos y dispositivos de entrada. La reutilización de osciloscopios analógicos subraya un enfoque sostenible, revitalizando equipos que, de otro modo, quedarían obsoletos.

Aspectos clave de aprendizaje:

1. **Integración de tecnologías analógicas y digitales:** Se destaca la integración efectiva de tecnologías analógicas y digitales, lo que permite maximizar el potencial visual y sonoro de las animaciones.
2. **Optimización de recursos en tiempo real:** La preparación previa de datos a través del preprocesado reduce significativamente la carga computacional durante la ejecución, mejorando la fluidez del sistema.
3. **Programación multihilo eficiente:** La utilización de programación multihilo permite la ejecución concurrente de tareas, asegurando una sincronización precisa entre las señales visuales y sonoras.
4. **Uso creativo de dispositivos MIDI:** La incorporación de dispositivos MIDI como interfaces de control en tiempo real amplía las posibilidades de interacción, enriqueciendo la experiencia del usuario en el contexto visual y sonoro.
5. **Escalabilidad y diseño modular:** Un diseño modular facilita tanto el mantenimiento del sistema como su expansión futura, permitiendo agregar funcionalidades adicionales sin comprometer la estructura existente.

Estos aspectos técnicos destacan la importancia de la innovación en la reutilización de tecnología existente para nuevos propósitos creativos, contribuyendo a la evolución de proyectos que combinan elementos analógicos y digitales.

6.2. LA INTELIGENCIA ARTIFICIAL COMO HERRAMIENTA DE PROGRAMACIÓN.

La inteligencia artificial ha sido una herramienta clave en este proyecto, permitiendo optimizar el proceso de desarrollo y mejorar la eficiencia en la programación. Este apartado analiza cómo la IA ha facilitado la automatización de tareas, la depuración del código y la integración de funcionalidades complejas, destacando su impacto en la ejecución del proyecto.

Un aspecto clave en el uso de inteligencia artificial durante este proyecto es la implementación de ***prompts*** para la generación y optimización de código. Un *prompt* es una instrucción detallada que se proporciona a un modelo de lenguaje para obtener respuestas específicas o resolver problemas técnicos. Esta técnica se emplea en varias etapas del desarrollo, agilizando significativamente la implementación y mejora del sistema. La clave para utilizar bien esta herramienta consiste en diseccionar en partes sencillas la funcionalidad y objetivos que se quieren conseguir, de manera que la inteligencia artificial asimile los conceptos poco a poco para así poder modificar los aspectos del proyecto completo de manera efectiva.

Ejemplo Concreto: Generación de Código para el Manejo de Interacciones MIDI

Uno de los retos del proyecto fue integrar de manera efectiva un teclado MIDI para permitir la interacción en tiempo real con las animaciones y el audio. Para abordar esto, se utilizó el siguiente prompt:

***Prompt* utilizado.**

"Necesito un fragmento de código en Python que procese mensajes MIDI para ajustar la frecuencia de reproducción de audio en tiempo real. El código debe convertir la nota MIDI en frecuencia y manejar los cambios de control para ajustar otros parámetros como el volumen y efectos visuales."

Respuesta generada por el modelo de IA:

```
import mido

def midi_note_to_frequency(note):
    """Convierte una nota MIDI a frecuencia en Hz."""
    A440 = 440.0  # Frecuencia de la nota A4
    return A440 * (2 ** ((note - 69) / 12.0))
```

```
def handle_control_change(control, value):
    """Maneja los cambios de control MIDI para ajustar parámetros."""
    if control == 1:  # Ejemplo: controlador 1 para volumen
        adjust_volume(value)
    elif control == 2:  # Ejemplo: controlador 2 para un efecto visual
        apply_visual_effect(value)

def process_midi_message(msg):
    """Procesa mensajes MIDI entrantes."""
    if msg.type == 'note_on':
        frequency = midi_note_to_frequency(msg.note)
        print(f"Nota MIDI: {msg.note}, Frecuencia: {frequency} Hz")
    elif msg.type == 'control_change':
        handle_control_change(msg.control, msg.value)

# Simulación de mensajes MIDI
test_msg = mido.Message('note_on', note=60)  # Nota C4
process_midi_message(test_msg)
```

Aplicación y resultados.

Este fragmento de código sirvió como base para integrar el procesamiento de mensajes MIDI en el proyecto. A partir de este punto de partida, se ajustaron las funciones para que se adaptaran a las necesidades específicas del sistema, como la aplicación de efectos en las animaciones y la generación de señales de audio. El uso de *prompts* permitie no solo acelerar la implementación de esta funcionalidad crítica, sino también obtener una solución inicial que fue iterada y refinada para el contexto particular del proyecto.

La inteligencia artificial está transformando el ámbito académico y social, optimizando procesos y abriendo nuevas oportunidades.

En el ámbito social, la IA encuentra aplicaciones en salud, transporte, comunicación etc., con un potencial futuro significativo en la resolución de problemas globales. A pesar de sus ventajas, persiste un estigma sobre su utilización, percibiéndose a veces como un "atajo" que resta autenticidad al esfuerzo humano.

En el contexto académico, la IA facilita el aprendizaje, permite la personalización de tareas y mejora la eficiencia en proyectos complejos. Sin embargo, plantea desafíos éticos, como la dependencia excesiva y la necesidad de directrices claras para su uso responsable.

Superar este estigma requiere promover la IA como una herramienta complementaria, no sustitutiva, del intelecto humano. Una adopción ética y consciente permitirá maximizar sus beneficios, asegurando un equilibrio entre innovación y responsabilidad social.

7. LÍNEAS FUTURAS.

En el desarrollo futuro del proyecto, se identifican varias líneas de mejora que permitirán expandir sus capacidades. Una de las prioridades es ampliar el número de efectos aplicables a las animaciones y señales de sonido, buscando que los usuarios dispongan de una mayor variedad de transformaciones sobre las formas de onda generadas.

Además, se prevé la adaptación del programa para trabajar con otros formatos de animaciones tridimensionales, como *FBX* u *OBJ (Object)*, con el objetivo de incrementar la compatibilidad con formatos estándar de la industria de la animación y gráficos 3D. Este avance permitiría utilizar modelos complejos y mejorar la versatilidad del sistema en entornos gráficos profesionales.

Otro aspecto clave es la mejora de la representación en el osciloscopio. Se busca implementar un mecanismo que apague el rayo catódico entre los trazos, evitando la visualización de líneas sobrantes y optimizando la calidad de las animaciones. Esta mejora técnica permitirá que las animaciones sean más limpias y visualmente más precisas en dispositivos analógicos.

En cuanto a la generación de sonido, se pretende aproximar el sistema a la reproducción de notas de canciones de manera automática, implementando una funcionalidad en la que las frecuencias coincidan con los *'Note ON'* y *'Note OFF'* del lenguaje MIDI. Para ello, se explorará la posibilidad de mantener una frecuencia base que se reproduzca continuamente, y que las teclas del teclado MIDI actúen como modificadores temporales de esa frecuencia, permitiendo una transición fluida entre notas y un retorno automático a la frecuencia original después de unos segundos. Este comportamiento reproducirá de manera más fiel el funcionamiento de un piano real, mejorando la experiencia sonora del usuario.

Se prevé también añadir un modo de pausa y reproducción que permita al usuario detener y reanudar las animaciones mediante un botón del teclado MIDI, mejorando la interactividad y el control sobre el flujo de las animaciones.

Asimismo, se considera mejorar la estructura del código, aislando cada parte aún más y aplicando una sintaxis más modular, aprovechando al máximo las capacidades que ofrece Python para una mayor escalabilidad y legibilidad del código. Este enfoque permitirá una mejor mantenibilidad del sistema y facilitará la implementación de nuevas funcionalidades en el futuro.

Finalmente, se busca implementar soporte para múltiples dispositivos de salida, permitiendo la interacción con varios dispositivos simultáneamente, como osciloscopios, altavoces y pantallas, mejorando la versatilidad del sistema en diferentes contextos.

8. BIBLIOGRAFÍA.

Python y Bibliotecas Utilizadas:

- Python Software Foundation. (2023). *Python Language Reference, version 3.9*. Disponible en https://www.python.org
- **Sounddevice:** Burock, S., Matveev, Y., & Heller, R. (2023). *Python-sounddevice Documentation*. Disponible en https://python-sounddevice.readthedocs.io
- **XML:** Python Software Foundation. (2023). *The ElementTree XML API*. Disponible en https://docs.python.org/3/library/xml.etree.elementtree.html
- **Threading**: Python Software Foundation. (2023). *Threading – Thread-based parallelism*. En *Python 3.9 documentation*. Recuperado de https://docs.python.org/3/library/threading.html
- **Time**: Python Software Foundation. (2023). *Time – Time access and conversions*. En *Python 3.9 documentation*. Recuperado de https://docs.python.org/3/library/time.html
- **NumPy**: Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). *Array programming with NumPy*. *Nature*, 585, 357–362. Disponible en https://numpy.org
- **Keyboard**: Burock, P. (2023). *Keyboard: Hook and simulate global keyboard events on Windows and Linux*. Recuperado de https://pypi.org/project/keyboard/
- **Mido**: MIDI Developers. (2023). *Mido - MIDI Messages for Python*. Recuperado de https://mido.readthedocs.io

Herramientas y Conceptos Clave:

- Blender Foundation. (2023). *Blender - Open Source 3D Creation*. Disponible en https://www.blender.org
- World Wide Web Consortium (W3C). (2022). *Scalable Vector Graphics (SVG) 2.0: W3C Recommendation*. Disponible en https://www.w3.org/TR/SVG2/
- Rauber, T., & Rünger, G. (2013). *Parallel programming: for multicore and cluster systems*. Springer Science & Business Media.
- Osci-render. (n.d.). *Osci-render*. Disponible en https://osci-render.com/

Especificaciones MIDI y DDS:

- Analog Devices. (2019). *Direct Digital Synthesis (DDS) Theory and Applications*. Disponible en https://www.analog.com
- Electric Druid. (s.f.). *Direct Digital Synthesis (DDS)*. Electric Druid. Direct Digital Synthesis (DDS) and Numerically Controlled Oscillators (NCOs). Disponible en https://electricdruid.net/direct-digital-synthesis/

- **Cornell University - ECE 4760 Final Project**. (2021). *Oscilloscope Music*. https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2021/sec322_edk52_rbm244/sec322_edk52_rbm244/oscilloscope_music.html#results

9. PLANIFICACIÓN TEMPORAL Y PRESUPUESTO.

9.1. PLANIFICACIÓN TEMPORAL.

La planificación temporal, se dividió en varias fases para asegurar el desarrollo organizado del proyecto, siguiendo una estructura en pirámide.

A continuación, se detalla el cronograma del trabajo establecido:

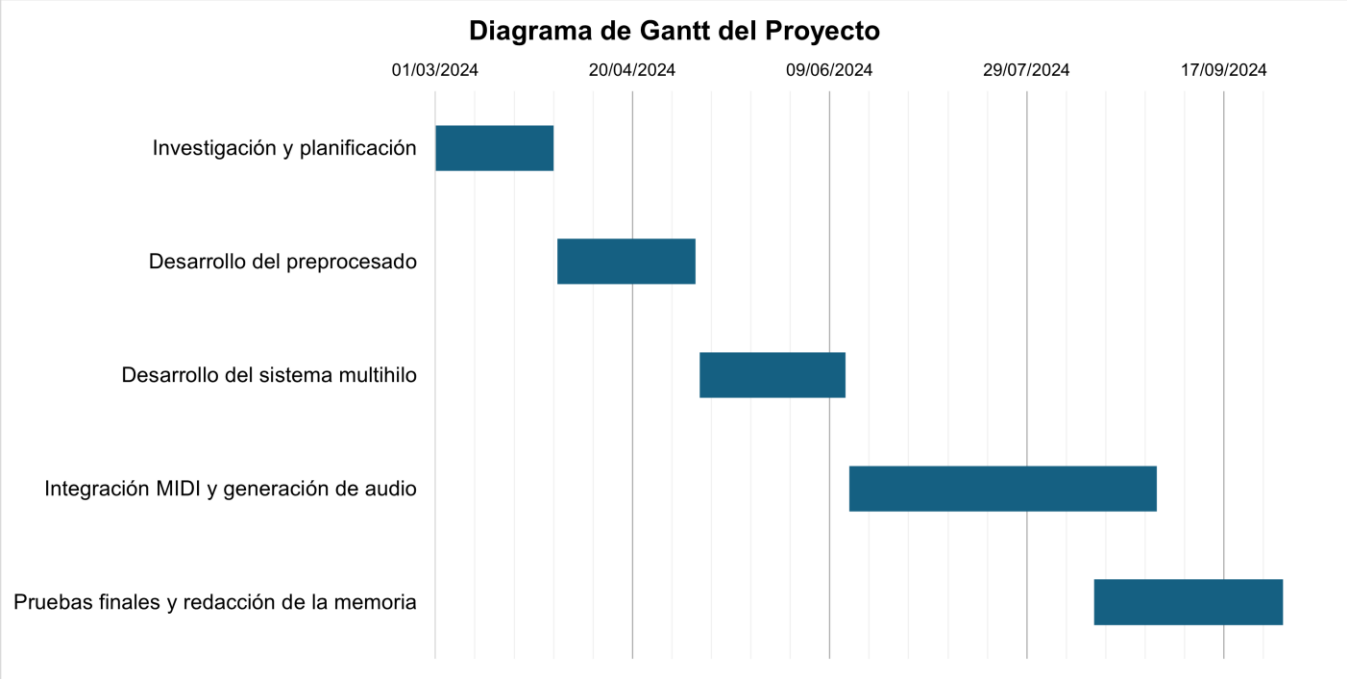
- **Fase 1: Investigación y planificación inicial (4 semanas)**
  - **Objetivo**: Establecer la viabilidad del proyecto, definir los objetivos, y estudiar las tecnologías que se emplearían.
  - **Tareas**:
    - Investigación sobre gráficos vectoriales y tecnologías de visualización.
    - Análisis de sistemas de procesamiento de señales.
    - Selección del entorno de desarrollo (Python y bibliotecas asociadas).
    - Planificación de la estructura general del sistema.
- **Fase 2: Desarrollo del preprocesado de animaciones (5 semanas)**
  - **Objetivo**: Implementar el sistema de preprocesado para transformar archivos SVG en datos manipulables por el sistema.
  - **Tareas**:
    - Análisis y redimensionado de animaciones.
    - Implementación de la conversión de archivos SVG a coordenadas vectoriales.
    - Pruebas de rendimiento del preprocesado y optimización del código.
- **Fase 3: Desarrollo del sistema multihilo (5 semanas)**
  - **Objetivo**: Crear la arquitectura multihilo para la reproducción de audio y visualización de animaciones.
  - **Tareas**:
    - Implementación de los hilos de análisis, reproducción y parámetros.
    - Coordinación de los hilos para la interacción fluida en tiempo real.
    - Pruebas de sincronización y ajuste de parámetros de reproducción.

- **Fase 4: Integración MIDI y generación de audio (10 semanas)**
  - **Objetivo**: Incorporar la interacción con el teclado MIDI para controlar la animación y los efectos sonoros.
  - **Tareas**:
    - Implementación de la conversión de señales MIDI a frecuencias sonoras.
    - Generación de señales de audio a partir de las animaciones.
    - Pruebas de interacción y ajuste de parámetros en tiempo real.
- **Fase 5: Pruebas finales y redacción de la memoria (7 semanas)**
  - **Objetivo**: Realizar pruebas intensivas de estabilidad, rendimiento y optimización.
  - **Tareas**:
    - Pruebas de uso de CPU, memoria y estabilidad a largo plazo.
    - Ajustes en la sincronización de audio y visualización.
    - Documentación del proyecto.

Fase	Duración	Objetivo	Tareas
Fase 1: Investigación y planificación inicial	4 semanas	Establecer la viabilidad del proyecto. Definir los objetivos. Estudiar las tecnologías que se emplearían.	Investigación sobre gráficos vectoriales y tecnologías de visualización. Análisis de sistemas de procesamiento de señales. Selección del entorno de desarrollo (Python y bibliotecas asociadas). Planificación de la estructura general del sistema.
Fase 2: Desarrollo del preprocesado de animaciones	5 semanas	Implementar el sistema de preprocesado para transformar archivos SVG en datos manipulables por el sistema.	Análisis y redimensionado de animaciones. Implementación de la conversión de archivos SVG a coordenadas vectoriales. Pruebas de rendimiento del preprocesado y optimización del código.
Fase 3: Desarrollo del sistema multihilo	5 semanas	Crear la arquitectura multihilo para la reproducción de audio y visualización de animaciones.	Implementación de los hilos de análisis, reproducción y parámetros. Coordinación de los hilos para la interacción fluida en tiempo real. Pruebas de sincronización y ajuste de parámetros de reproducción.
Fase 4: Integración MIDI y generación de audio	10 semanas	Incorporar la interacción con el teclado MIDI para controlar la animación y los efectos sonoros.	Implementación de la conversión de señales MIDI a frecuencias sonoras. Generación de señales de audio a partir de las animaciones. Pruebas de interacción y ajuste de parámetros en tiempo real.
Fase 5: Pruebas finales y redacción de la memoria	7 semanas	Realizar pruebas intensivas de estabilidad, rendimiento y optimización.	Pruebas de uso de CPU, memoria y estabilidad a largo plazo. Ajustes en la sincronización de audio y visualización. Documentación del proyecto.

Tabla 7: Resumen de la planificación temporal del proyecto.

9.2. DIAGRAMA DE GANTT.



9.3. PRESUPUESTO.

El proyecto requirió una serie de recursos tanto de hardware como de software, además del tiempo invertido en su desarrollo. A continuación, se desglosan los principales costes asociados:

1. Hardware:
    - **Tarjeta de sonido externa Vantec USB External 7.1**: 45 EUR. Utilizada para enviar la señal de audio tanto al osciloscopio como a los altavoces.
    - **Osciloscopio analógico PROMAX OD-416 (20 MHz)**: Disponible en laboratorio. Utilizado para la visualización de las animaciones en tiempo real.
    - **Teclado MIDI Worlde MINI TUNA**: 75 EUR. Empleado para la interacción en tiempo real con las animaciones y los parámetros sonoros.
    - **PC portátil (amortización)**: 240 EUR. Utilizado para el desarrollo del código, pruebas y reproducción.
2. Software:
    - **Python (lenguaje de programación y bibliotecas)**: Gratuito (código abierto). Herramienta principal utilizada para el desarrollo del sistema, incluyendo bibliotecas como numpy, sounddevice, mido, y xml.etree.ElementTree.
3. Costes de Desarrollo:
    - **Tiempo invertido por el autor**: 300 horas estimadas, con un coste de entre 2400 y 4500 EUR, dependiendo de la tarifa por hora aplicable.
4. Costes Totales Aproximados:
    - **Hardware**: 360 EUR.
    - **Costes de desarrollo**: 2400 - 4500 EUR.
    - **Coste total aproximado**: 2760 - 4860 EUR.

Categoría	Componente/Descripción	Cantidad	Coste (EUR)	Descripción
Hardware	Tarjeta de sonido externa Vantec USB External 7.1	1	45	Utilizada para enviar la señal de audio tanto al osciloscopio como a los altavoces.
	Osciloscopio analógico PROMAX OD-416 (20 MHz)	-	Disponible en laboratorio	Osciloscopio utilizado para la visualización de las animaciones en tiempo real.
	Teclado MIDI Worlde MINI TUNA	1	75	Utilizado para la interacción en tiempo real con la animación y los parámetros sonoros.
	PC portátil (amortización)	1	240	Ordenador personal del autor para desarrollo del código, pruebas y reproducción
Software	Python (lenguaje de programación y bibliotecas)	-	Gratuito (Open Source)	Herramienta principal utilizada para el desarrollo del sistema, incluyendo bibliotecas como numpy, sounddevice, mido y xml.etree.ElementTree.
Costes de desarrollo	Tiempo invertido por el autor (Estimación)	300 h	2400 - 4500	Dependiendo de la tarifa por hora aplicable.
Costes totales aproximados			**2760 - 4860**	Suma de todos los costes de hardware y desarrollo.

Tabla 8: Resumen del presupuesto del proyecto.

10. ÍNDICE DE FIGURAS.

11. ÍNDICE DE TABLAS.

12. ABREVIATURAS, UNIDADES Y ACRÓNIMOS.

- **ADC**: *Analog to Digital Converter* (Convertidor de Analógico a Digital).
- **CRT**: *Cathode Ray Tube* (Tubo de Rayos Catódicos).
- **CPU**: *Central Processing Unit* (Unidad Central de Procesamiento).
- **DDS**: *Direct Digital Synthesis* (Síntesis Digital Directa).
- **E/S**: Entrada y Salida.
- **FBX**: *Filmbox* (Formato de archivo de animación 3D).
- **FPS**: *Frames per second* (Fotogramas por segundo).
- **Hz**: Hertz (Unidad de frecuencia).
- **LCD**: *Liquid Crystal Display* (Pantalla de Cristal Líquido).
- **MIDI**: *Musical Instrument Digital Interface* (Interfaz Digital de Instrumentos Musicales).
- **npz**: Formato de archivo comprimido de *NumPy*.
- **RAM**: *Random Access Memory* (Memoria de Acceso Aleatorio).
- **SBC**: *Single Board Computer* (Ordenador de Placa Única).
- **SVG**: *Scalable Vector Graphics* (Gráficos Vectoriales Escalables).
- **XML**: *Extensible Markup Language* (Lenguaje de Marcado Extensible).

13. GLOSARIO.

- **Acumulador de Fase**: Componente en un oscilador controlado numéricamente (*NCO*) que determina la posición en la tabla de ondas para generar señales.
- **Alias:** Fenómeno en el procesamiento de señales donde las frecuencias superiores a la mitad de la frecuencia de muestreo (frecuencia de Nyquist) se reflejan en el espectro, provocando distorsiones y la aparición de componentes de frecuencia no deseadas en el rango audible o visual.
- **Interpolación**: Técnica para estimar nuevos puntos de datos dentro del rango de un conjunto discreto de puntos ya conocidos.
- ***Multithreading***: Técnica de programación que permite la ejecución simultánea de múltiples hilos dentro de un solo proceso para mejorar el rendimiento.
- **Osciloscopio**: Instrumento de medición que muestra representaciones gráficas de señales eléctricas variables en el tiempo.
- **Pasos de Interpolación**: Proceso de ajuste de los datos para crear una transición suave entre puntos en una animación o señal.
- ***Phasor* (Fasor)**: Representación compleja de señales sinusoidales, utilizada para simplificar el análisis de sistemas lineales y tiempo invariante.
- **Preprocesado**: Proceso de transformación de datos crudos en un formato optimizado para su procesamiento posterior.
- **Síntesis Digital Directa (DDS)**: Método de generación de señales de forma digital utilizando tablas de onda y osciladores controlados numéricamente.
- **Vectorial**: Relativo a gráficos definidos por coordenadas matemáticas, permitiendo escalabilidad sin pérdida de calidad.

14. ANEXO.

En este anexo se incluye el código fuente del proyecto en el estado correspondiente a la entrega de esta memoria. Las versiones futuras y el código actualizado estarán disponibles en el repositorio de GitHub del autor.

14.1. CÓDIGO.

14.1.1. Preprocesado.

```
import numpy as np
import xml.etree.ElementTree as ET
import os

# Convierte una cadena de texto con coordenadas en una lista de pares de
coordenadas numéricas (x, y)
def string_a_lista(coordenadas_str):
    coordenadas_lista = coordenadas_str.split()
    coordenadas = []
    for i in range(0, len(coordenadas_lista), 2):
        coordenada_x = float(coordenadas_lista[i].split(',')[0])
        coordenada_y = float(coordenadas_lista[i+1])
        coordenadas.append([coordenada_x, coordenada_y])
    return np.array(coordenadas, dtype=np.float32)

# Extrae los frames de un archivo SVG, retornando los paths de cada frame como
listas de coordenadas
def obtener_frames(nombre):
    lista_total = []
    tree = ET.parse(nombre)
    root = tree.getroot()
    namespaces = {
        'svg': 'http://www.w3.org/2000/svg',
        'inkscape': 'http://www.inkscape.org/namespaces/inkscape'
    }

    # Identifica los frames en el archivo SVG que contienen grupos de paths
    frames = root.findall(".//svg:g[@inkscape:groupmode='frame']", namespaces)
    for f in frames:
        lista_de_paths = []
        paths = f.findall(".//svg:path", namespaces)
        for p in paths:
            path = p.get('d')
```

```
            path_sinM = path[3:]  # Elimina el primer comando 'M' para aislar
las coordenadas
            lista_de_coordenadas = string_a_lista(path_sinM)
            lista_de_paths.append(lista_de_coordenadas)
        lista_total.append(lista_de_paths)

    return lista_total

# Calcula las distancias entre puntos consecutivos en cada path, retornando
una lista de distancias para cada path
def calcular_distancias(path_list):
    distancias_list = []
    for path in path_list:
        path = np.array(path)
        num_puntos = len(path)
        distancias = np.zeros(num_puntos - 1)
        for i in range(num_puntos - 1):
            distancia = np.sqrt((path[i + 1][0] - path[i][0])**2 + (path[i +
1][1] - path[i][1])**2)
            distancias[i] = distancia
        distancias_list.append(distancias)
    return distancias_list

# Redimensiona un vector a una longitud específica mediante interpolación
basada en las distancias entre puntos
def redimensiona(vector, nueva_longitud, distancias):
    if nueva_longitud <= len(vector):
        return vector

    puntos = nueva_longitud - len(vector)
    distancia_total = np.sum(distancias)
    fracciones = distancias / distancia_total
    redimensionado = []
    resto = 0
    num_puntos = np.zeros_like(distancias)
    num_puntos_real = fracciones * puntos

    for i in range(len(num_puntos) - 1):
        npt = num_puntos_real[i]
        npr = round(npt)
        resto += npt - npr
        if resto >= 1:
            npr += 1
            resto -= 1
```

```
        elif resto <= -1:
            npr -= 1
            resto += 1
        num_puntos[i] = max(npr + 1, 1)

    num_puntos[-1] = nueva_longitud - (num_puntos.sum() - num_puntos[-1])
    num_puntos[-1] = max(num_puntos[-1], 1)

    for i in range(len(vector) - 1):
        punto1 = vector[i]
        punto2 = vector[i + 1]
        intermedios = np.linspace(punto1, punto2, int(num_puntos[i]),
endpoint=False)
        redimensionado = np.concatenate((redimensionado, intermedios))

    redimensionado = np.concatenate((redimensionado, [vector[-1]]))

    return redimensionado

# Redimensiona y concatena todos los paths en un frame a una longitud total
especificada
def redimensiona_y_concatena(path_list, nueva_longitud):
    distancias_list = calcular_distancias(path_list)
    distancia_total = 0

    for distancias in distancias_list:
        distancia_total += np.sum(distancias)

    total_points = 0
    all_points = []
    for path, distancias in zip(path_list, distancias_list):
        path = np.array(path)
        n = int(round(nueva_longitud * np.sum(distancias) / distancia_total))
        concatenado_x = redimensiona(path[:, 0], n, distancias)
        concatenado_y = redimensiona(path[:, 1], n, distancias)
        narr = np.column_stack((concatenado_x, concatenado_y))
        all_points.append(narr)
        total_points += len(narr)

    if total_points > nueva_longitud:
        redimensionado = np.vstack(all_points)[:nueva_longitud]
    else:
        redimensionado = np.vstack(all_points)
```

```
        indices_originales = np.linspace(0, len(redimensionado) - 1,
len(redimensionado))
        indices_nuevos = np.linspace(0, len(redimensionado) - 1,
nueva_longitud)
        redimensionado_x = np.interp(indices_nuevos, indices_originales,
redimensionado[:, 0])
        redimensionado_y = np.interp(indices_nuevos, indices_originales,
redimensionado[:, 1])
        redimensionado = np.column_stack((redimensionado_x, redimensionado_y))

    return redimensionado

# Procesa cada archivo SVG de animación, redimensionando y guardando los
frames con una longitud de puntos especificada
def procesa_multiples_animaciones(archivos_svg, nueva_longitud,
verbose=False):
    for archivo in archivos_svg:
        if verbose:
            print(f"Procesando archivo: {archivo}")
        frame_list = obtener_frames(archivo)

        frames_dict = {}

        for frame_idx, path_list in enumerate(frame_list):
            redimensionado = redimensiona_y_concatena(path_list,
nueva_longitud)

            frames_dict[f"frame_{frame_idx+1}"] = redimensionado

            if verbose:
                print(f"Frame {frame_idx + 1} redimensionado con
{len(redimensionado)} puntos.")

        nombre_archivo = f"{os.path.splitext(archivo)[0]}_redimensionado.npz"
        np.savez_compressed(nombre_archivo, **frames_dict)

        if verbose:
            print(f"Archivo guardado: {nombre_archivo}")
```

14.1.2. Postprocesado.

```
import numpy as np
import xml.etree.ElementTree as ET
import os

# Convierte una cadena de texto con coordenadas en una lista de pares de
coordenadas numéricas (x, y)
def string_a_lista(coordenadas_str):
    coordenadas_lista = coordenadas_str.split()
    coordenadas = []
    for i in range(0, len(coordenadas_lista), 2):
        coordenada_x = float(coordenadas_lista[i].split(',')[0])
        coordenada_y = float(coordenadas_lista[i+1])
        coordenadas.append([coordenada_x, coordenada_y])
    return np.array(coordenadas, dtype=np.float32)

# Extrae los frames de un archivo SVG, retornando los paths de cada frame como
listas de coordenadas
def obtener_frames(nombre):
    lista_total = []
    tree = ET.parse(nombre)
    root = tree.getroot()
    namespaces = {
        'svg': 'http://www.w3.org/2000/svg',
        'inkscape': 'http://www.inkscape.org/namespaces/inkscape'
    }

    # Identifica los frames en el archivo SVG que contienen grupos de paths
    frames = root.findall(".//svg:g[@inkscape:groupmode='frame']", namespaces)
    for f in frames:
        lista_de_paths = []
        paths = f.findall(".//svg:path", namespaces)
        for p in paths:
            path = p.get('d')
            path_sinM = path[3:]  # Elimina el primer comando 'M' para aislar
las coordenadas
            lista_de_coordenadas = string_a_lista(path_sinM)
            lista_de_paths.append(lista_de_coordenadas)
        lista_total.append(lista_de_paths)

    return lista_total
```

```
# Calcula las distancias entre puntos consecutivos en cada path, retornando
una lista de distancias para cada path
def calcular_distancias(path_list):
    distancias_list = []
    for path in path_list:
        path = np.array(path)
        num_puntos = len(path)
        distancias = np.zeros(num_puntos - 1)
        for i in range(num_puntos - 1):
            distancia = np.sqrt((path[i + 1][0] - path[i][0])**2 + (path[i +
1][1] - path[i][1])**2)
            distancias[i] = distancia
        distancias_list.append(distancias)
    return distancias_list

# Redimensiona un vector a una longitud específica mediante interpolación
basada en las distancias entre puntos
def redimensiona(vector, nueva_longitud, distancias):
    if nueva_longitud <= len(vector):
        return vector

    puntos = nueva_longitud - len(vector)
    distancia_total = np.sum(distancias)
    fracciones = distancias / distancia_total
    redimensionado = []
    resto = 0
    num_puntos = np.zeros_like(distancias)
    num_puntos_real = fracciones * puntos

    for i in range(len(num_puntos) - 1):
        npt = num_puntos_real[i]
        npr = round(npt)
        resto += npt - npr
        if resto >= 1:
            npr += 1
            resto -= 1
        elif resto <= -1:
            npr -= 1
            resto += 1
        num_puntos[i] = max(npr + 1, 1)

    num_puntos[-1] = nueva_longitud - (num_puntos.sum() - num_puntos[-1])
    num_puntos[-1] = max(num_puntos[-1], 1)
```

```
    for i in range(len(vector) - 1):
        punto1 = vector[i]
        punto2 = vector[i + 1]
        intermedios = np.linspace(punto1, punto2, int(num_puntos[i]),
endpoint=False)
        redimensionado = np.concatenate((redimensionado, intermedios))

    redimensionado = np.concatenate((redimensionado, [vector[-1]]))

    return redimensionado

# Redimensiona y concatena todos los paths en un frame a una longitud total
especificada
def redimensiona_y_concatena(path_list, nueva_longitud):
    distancias_list = calcular_distancias(path_list)
    distancia_total = 0

    for distancias in distancias_list:
        distancia_total += np.sum(distancias)

    total_points = 0
    all_points = []
    for path, distancias in zip(path_list, distancias_list):
        path = np.array(path)
        n = int(round(nueva_longitud * np.sum(distancias) / distancia_total))
        concatenado_x = redimensiona(path[:, 0], n, distancias)
        concatenado_y = redimensiona(path[:, 1], n, distancias)
        narr = np.column_stack((concatenado_x, concatenado_y))
        all_points.append(narr)
        total_points += len(narr)

    if total_points > nueva_longitud:
        redimensionado = np.vstack(all_points)[:nueva_longitud]
    else:
        redimensionado = np.vstack(all_points)
        indices_originales = np.linspace(0, len(redimensionado) - 1,
len(redimensionado))
        indices_nuevos = np.linspace(0, len(redimensionado) - 1,
nueva_longitud)
        redimensionado_x = np.interp(indices_nuevos, indices_originales,
redimensionado[:, 0])
        redimensionado_y = np.interp(indices_nuevos, indices_originales,
redimensionado[:, 1])
        redimensionado = np.column_stack((redimensionado_x, redimensionado_y))
```

```
    return redimensionado

# Procesa cada archivo SVG de animación, redimensionando y guardando los
frames con una longitud de puntos especificada
def procesa_multiples_animaciones(archivos_svg, nueva_longitud,
verbose=False):
    for archivo in archivos_svg:
        if verbose:
            print(f"Procesando archivo: {archivo}")
        frame_list = obtener_frames(archivo)

        frames_dict = {}

        for frame_idx, path_list in enumerate(frame_list):
            redimensionado = redimensiona_y_concatena(path_list,
nueva_longitud)

            frames_dict[f"frame_{frame_idx+1}"] = redimensionado

            if verbose:
                print(f"Frame {frame_idx + 1} redimensionado con
{len(redimensionado)} puntos.")

        nombre_archivo = f"{os.path.splitext(archivo)[0]}_redimensionado.npz"
        np.savez_compressed(nombre_archivo, **frames_dict)

        if verbose:
            print(f"Archivo guardado: {nombre_archivo}")
```

14.2. BANCO DE PRUEBAS.

14.2.1. Procesamiento de audio.

```
import time
import os
import csv
import numpy as np
from Osci_main import load_animation, get_audio_buffer_from_wave,
audio_parameters, files_npz

def test_audio_processing_time(iterations=5, detailed=False):
    results = []
    animation_times = {name: [] for name in files_npz}  # Diccionario para
almacenar los tiempos de cada animación

    for i in range(iterations):
        print(f"\n================= Iteración {i + 1} =================")

        for name, file in files_npz.items():
            try:
                # Registrar el tiempo de inicio de la animación
                animation_start_time = time.time()

                # Cargar y procesar la animación
                animation_data = load_animation(file)
                frame_count = len(animation_data)

                # Obtener el tamaño del archivo con manejo de excepciones
                try:
                    file_size = os.path.getsize(file)
                except Exception as e:
                    print(f"[ERROR] No se pudo obtener el tamaño del archivo
'{file}': {e}")
                    file_size = "Desconocido"

                # Procesar frames y registrar el tiempo final
                for frame_data in animation_data.values():
                    get_audio_buffer_from_wave(audio_parameters['n_bits_phasor
'], 1, frame_data)

                processing_end_time = time.time()
                elapsed_time = processing_end_time - animation_start_time  #
Calcular diferencia de tiempo
```

```
                # Guardar el tiempo en el diccionario y en los resultados
                animation_times[name].append(elapsed_time)
                results.append({
                    "Iteración": i + 1,
                    "Animación": name,
                    "Archivo": file,
                    "Tamaño del Archivo (bytes)": file_size,
                    "Frames Procesados": frame_count,
                    "Tiempo Total de Procesamiento (s)": elapsed_time,
                    "Estado": "Éxito"
                })

                print(f"Animación '{name}' procesada en {elapsed_time:.4f}
segundos con {frame_count} frames.")

            except Exception as e:
                print(f"[ERROR] Error procesando '{name}': {e}")
                results.append({
                    "Iteración": i + 1,
                    "Animación": name,
                    "Archivo": file,
                    "Tamaño del Archivo (bytes)": "",
                    "Frames Procesados": "",
                    "Tiempo Total de Procesamiento (s)": "Error",
                    "Estado": f"Error: {e}"
                })

    # Calcular promedio y desviación estándar para cada animación
    for name, times in animation_times.items():
        if times:
            avg_time = np.mean(times)
            std_dev_time = np.std(times)
            results.append({
                "Iteración": "Promedio",
                "Animación": name,
                "Archivo": "",
                "Tamaño del Archivo (bytes)": "",
                "Frames Procesados": "",
                "Tiempo Total de Procesamiento (s)": avg_time,
                "Estado": "Promedio"
            })
            results.append({
                "Iteración": "Desviación Estándar",
```

```
                "Animación": name,
                "Archivo": "",
                "Tamaño del Archivo (bytes)": "",
                "Frames Procesados": "",
                "Tiempo Total de Procesamiento (s)": std_dev_time,
                "Estado": "Desviación Estándar"
            })

    # Guardar en CSV
    with open("audio_processing_results_detailed.csv", "w", newline='') as f:
        writer = csv.DictWriter(f, fieldnames=["Iteración", "Animación",
"Archivo", "Tamaño del Archivo (bytes)", "Frames Procesados", "Tiempo Total de
Procesamiento (s)", "Estado"])
        writer.writeheader()
        writer.writerows(results)

if __name__ == "__main__":
    test_audio_processing_time(iterations=5, detailed=True)
```

14.2.2. Ejecución concurrente.

```
import time
import threading
import csv
import numpy as np
from Osci_main import analysis_thread, playback_thread, parameters_thread,
keyboard_listener_thread, files_npz

def test_concurrent_execution(iterations=5):
    results = []
    execution_times = []

    for i in range(iterations):
        iteration_start_time = time.time()  # Tiempo de inicio de la iteración
        print(f"\n================= Iteración {i + 1} =================")

        successful_threads = 0
        try:
            # Iniciar hilos
            analysis = threading.Thread(target=analysis_thread,
args=(files_npz,))
            playback = threading.Thread(target=playback_thread)
            midi_listener = threading.Thread(target=parameters_thread,
args=('WORLDE    0',), daemon=True)
            keyboard_listener =
threading.Thread(target=keyboard_listener_thread, daemon=True)

            threads = {
                "analysis_thread": analysis,
                "playback_thread": playback,
                "midi_listener_thread": midi_listener,
                "keyboard_listener_thread": keyboard_listener
            }

            for name, thread in threads.items():
                start_time = time.time()
                thread.start()
                results.append({
                    "Iteración": i + 1,
                    "Hilo": name,
                    "Tiempo de Inicio": start_time,
                    "Tiempo de Fin": "",
                    "Estado": "Iniciado"
```

```
                })

            # Finalizar hilos
            for name, thread in threads.items():
                thread.join()
                end_time = time.time()
                successful_threads += 1
                results.append({
                    "Iteración": i + 1,
                    "Hilo": name,
                    "Tiempo de Inicio": "",
                    "Tiempo de Fin": end_time,
                    "Estado": "Finalizado"
                })

            iteration_end_time = time.time()  # Tiempo de finalización de la
iteración
            iteration_elapsed = iteration_end_time - iteration_start_time  #
Tiempo transcurrido
            execution_times.append(iteration_elapsed)
            print(f"Tiempo total de la iteración {i + 1}:
{iteration_elapsed:.4f} segundos")

        except Exception as e:
            print(f"[ERROR] Error en la iteración {i + 1}: {e}")
            results.append({
                "Iteración": i + 1,
                "Hilo": "Error",
                "Tiempo de Inicio": "",
                "Tiempo de Fin": "",
                "Estado": f"Error: {e}"
            })

    # Calcular promedio y desviación estándar del tiempo total de ejecución
    if execution_times:
        avg_execution_time = np.mean(execution_times)
        std_dev_execution_time = np.std(execution_times)
        results.append({
            "Iteración": "Promedio",
            "Hilo": "Total",
            "Tiempo de Inicio": "",
            "Tiempo de Fin": "",
            "Estado": f"{avg_execution_time:.4f} segundos (Promedio)"
        })
```

```
        results.append({
            "Iteración": "Desviación Estándar",
            "Hilo": "Total",
            "Tiempo de Inicio": "",
            "Tiempo de Fin": "",
            "Estado": f"{std_dev_execution_time:.4f} segundos (Desviación
Estándar)"
        })

    # Guardar en CSV
    with open("concurrent_execution_results_detailed.csv", "w", newline='') as
f:
        writer = csv.DictWriter(f, fieldnames=["Iteración", "Hilo", "Tiempo de
Inicio", "Tiempo de Fin", "Estado"])
        writer.writeheader()
        writer.writerows(results)

if __name__ == "__main__":
    test_concurrent_execution(iterations=5)
```

14.2.3. Uso de CPU.

```
import psutil
import time
import threading
import csv
import numpy as np
from Osci_main import analysis_thread, playback_thread, parameters_thread,
keyboard_listener_thread, files_npz

def test_cpu_usage(iterations=5):
    results = []
    cpu_usages = []
    total_cpu_times = []

    for i in range(iterations):
        iteration_start_time = time.time()
        process = psutil.Process()
        print(f"\n================= Iteración {i + 1} =================")

        # Iniciar hilos y medir uso de CPU
        analysis = threading.Thread(target=analysis_thread, args=(files_npz,))
        playback = threading.Thread(target=playback_thread)
        midi_listener = threading.Thread(target=parameters_thread,
args=('WORLDE    0',), daemon=True)
        keyboard_listener = threading.Thread(target=keyboard_listener_thread,
daemon=True)

        threads = [analysis, playback, midi_listener, keyboard_listener]

        for thread in threads:
            thread.start()

        # Registrar el uso de CPU después de iniciar los hilos
        cpu_usage = process.cpu_percent(interval=5)
        cpu_time_total = process.cpu_times().user + process.cpu_times().system

        cpu_usages.append(cpu_usage)
        total_cpu_times.append(cpu_time_total)

        results.append({
            "Iteración": i + 1,
            "Uso de CPU (%)": cpu_usage,
            "Tiempo Total de CPU (s)": cpu_time_total,
```

```
            "Número de Hilos": len(threads)
        })

        # Esperar a que los hilos terminen
        for thread in threads:
            thread.join()

        print(f"Uso de CPU en la iteración {i + 1}: {cpu_usage}% con tiempo de
CPU total {cpu_time_total:.4f} segundos")

    # Calcular promedio y desviación estándar del uso de CPU
    if cpu_usages:
        avg_cpu_usage = np.mean(cpu_usages)
        std_dev_cpu_usage = np.std(cpu_usages)
        results.append({
            "Iteración": "Promedio",
            "Uso de CPU (%)": avg_cpu_usage,
            "Tiempo Total de CPU (s)": "",
            "Número de Hilos": ""
        })
        results.append({
            "Iteración": "Desviación Estándar",
            "Uso de CPU (%)": std_dev_cpu_usage,
            "Tiempo Total de CPU (s)": "",
            "Número de Hilos": ""
        })

    # Guardar en CSV
    with open("cpu_usage_results_detailed.csv", "w", newline='') as f:
        writer = csv.DictWriter(f, fieldnames=["Iteración", "Uso de CPU (%)",
"Tiempo Total de CPU (s)", "Número de Hilos"])
        writer.writeheader()
        writer.writerows(results)

if __name__ == "__main__":
    test_cpu_usage(iterations=5)
```

14.2.4. Carga de archivos.

```
import time
import csv
import numpy as np
from Osci_main import load_animation, files_npz

def test_load_time(iterations=5):
    results = []
    load_times = []

    for i in range(iterations):
        print(f"\n================= Iteración {i + 1} =================")

        for name, file in files_npz.items():
            start_time = time.time()
            load_animation(file)
            end_time = time.time()
            elapsed_time = end_time - start_time  # Diferencia de tiempo

            load_times.append(elapsed_time)
            results.append({
                "Iteración": i + 1,
                "Archivo": file,
                "Tiempo de Carga (s)": elapsed_time
            })
            print(f"Archivo '{file}' cargado en {elapsed_time:.4f} segundos.")

    # Calcular promedio y desviación estándar del tiempo de carga
    if load_times:
        avg_load_time = np.mean(load_times)
        std_dev_load_time = np.std(load_times)
        results.append({
            "Iteración": "Promedio",
            "Archivo": "Todos",
            "Tiempo de Carga (s)": avg_load_time
        })
        results.append({
            "Iteración": "Desviación Estándar",
            "Archivo": "Todos",
            "Tiempo de Carga (s)": std_dev_load_time
        })

    # Guardar en CSV
```

```
    with open("load_time_results_detailed.csv", "w", newline='') as f:
        writer = csv.DictWriter(f, fieldnames=["Iteración", "Archivo", "Tiempo
de Carga (s)"])
        writer.writeheader()
        writer.writerows(results)

if __name__ == "__main__":
    test_load_time(iterations=5)
```

14.2.5. Interacción MIDI.

```
import time
import csv
import numpy as np
from Osci_main import process_midi_message

def test_midi_interaction(iterations=5, messages_per_iteration=10):
    results = []
    response_times = []

    for i in range(iterations):
        print(f"\n================= Iteración {i + 1} =================")

        for msg_id in range(messages_per_iteration):
            try:
                start_time = time.time()

                # Procesar un mensaje MIDI simulado
                process_midi_message(msg_id, 64)  # Número de mensaje y valor
arbitrario

                end_time = time.time()
                response_time = end_time - start_time  # Tiempo de respuesta
                response_times.append(response_time)

                results.append({
                    "Iteración": i + 1,
                    "Mensaje MIDI": msg_id,
                    "Tiempo de Respuesta (s)": response_time
                })

                print(f"Mensaje MIDI {msg_id} procesado en {response_time:.4f}
segundos")

            except Exception as e:
                print(f"[ERROR] Error procesando mensaje MIDI {msg_id}: {e}")
                results.append({
                    "Iteración": i + 1,
                    "Mensaje MIDI": msg_id,
                    "Tiempo de Respuesta (s)": "Error"
                })

    # Calcular promedio y desviación estándar del tiempo de respuesta
```

```
    if response_times:
        avg_response_time = np.mean(response_times)
        std_dev_response_time = np.std(response_times)
        results.append({
            "Iteración": "Promedio",
            "Mensaje MIDI": "Todos",
            "Tiempo de Respuesta (s)": avg_response_time
        })
        results.append({
            "Iteración": "Desviación Estándar",
            "Mensaje MIDI": "Todos",
            "Tiempo de Respuesta (s)": std_dev_response_time
        })

    # Guardar en CSV
    with open("midi_interaction_results_detailed.csv", "w", newline='') as f:
        writer = csv.DictWriter(f, fieldnames=["Iteración", "Mensaje MIDI",
"Tiempo de Respuesta (s)"])
        writer.writeheader()
        writer.writerows(results)

if __name__ == "__main__":
    test_midi_interaction(iterations=5)
```

14.2.6. Preprocesado de animaciones.

```
import time
import os
import csv
import numpy as np
from preprocesado_animaciones import procesa_multiples_animaciones

# Lista de archivos SVG a procesar
archivos_svg = ['baile1.svg', 'baile2.svg', 'break2.svg', 'break1.svg']

def test_preprocessing(iterations=5, nueva_longitud=4096):
    results = []
    preprocessing_times = []

    for i in range(iterations):
        start_time = time.time()  # Tiempo de inicio de la iteración

        # Procesar las animaciones
        procesa_multiples_animaciones(archivos_svg, nueva_longitud)

        end_time = time.time()  # Tiempo de finalización de la iteración
        elapsed_time = end_time - start_time  # Tiempo transcurrido
        preprocessing_times.append(elapsed_time)

        # Verificar archivos generados
        files_generated = 0
        missing_files = []
        for archivo in archivos_svg:
            nombre_archivo =
f"{os.path.splitext(archivo)[0]}_redimensionado.npz"
            if os.path.exists(nombre_archivo):
                files_generated += 1
            else:
                missing_files.append(nombre_archivo)

        results.append({
            "Iteración": i + 1,
            "Tiempo de Preprocesado (s)": f"{elapsed_time:.4f}",  # Limitar
decimales en el tiempo
            "Archivos Generados Correctamente": files_generated,
            "Archivos Faltantes": ", ".join(missing_files)
        })
```

```
        print(f"Iteración {i + 1}: Tiempo de preprocesado: {elapsed_time:.4f}
segundos, Archivos generados: {files_generated}")

    # Calcular promedio y desviación estándar del tiempo de preprocesado
    if preprocessing_times:
        avg_time = np.mean(preprocessing_times)
        std_dev_time = np.std(preprocessing_times)
        results.append({
            "Iteración": "Promedio",
            "Tiempo de Preprocesado (s)": f"{avg_time:.4f}",  # Limitar
decimales en el promedio
            "Archivos Generados Correctamente": "",
            "Archivos Faltantes": ""
        })
        results.append({
            "Iteración": "Desviación Estándar",
            "Tiempo de Preprocesado (s)": f"{std_dev_time:.4f}",  # Limitar
decimales en la desviación estándar
            "Archivos Generados Correctamente": "",
            "Archivos Faltantes": ""
        })

    # Guardar en CSV
    with open("preprocessing_results_detailed.csv", "w", newline='') as f:
        writer = csv.DictWriter(f, fieldnames=["Iteración", "Tiempo de
Preprocesado (s)", "Archivos Generados Correctamente", "Archivos Faltantes"])
        writer.writeheader()
        writer.writerows(results)

if __name__ == "__main__":
    test_preprocessing(iterations=5)
```