UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SOFTWARE DE ORQUESTACIÓN CON POSICIONAMIENTO DE INSTRUMENTOS Y AUDIO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

CLAUDIO ANTONIO ZAMORANO BRUNETTI

PROFESOR GUÍA:
DANIEL CALDERÓN SAAVEDRA

MIEMBROS DE LA COMISIÓN:
JUAN BARRIOS NÚÑEZ
FELIPE BRAVO MÁRQUEZ

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: CLAUDIO ANTONIO ZAMORANO BRUNETTI
FECHA: 2023
PROF. GUÍA: DANIEL CALDERÓN

SOFTWARE DE ORQUESTACIÓN CON POSICIONAMIENTO DE INSTRUMENTOS Y AUDIO

Hoy en día en las tecnologías utilizadas para la industria de la música o acústica no son sencillos de utilizar. Esto se debe principalmente a las complejas funcionalidades que tienen y a las grandes cantidades de recursos como dinero y tiempo de aprendizaje que se deben invertir en ellas y por esta razón tienden a abrumar a los usuarios, quienes terminan utilizando pocas funcionalidades de estas. Es bajo este contexto que desarrollar un *software* de orquestación musical se hace posible. La motivación de este trabajo de título viene con la gran importancia de la música en la vida del autor de este documento y de las grandes ventajas de tener un *software* dedicado específicamente a la orquestación musical brinda. El objetivo principal es el de crear un *software* de orquestación musical con audio 3D y posicionamiento de instrumentos de fácil uso y de código abierto. Otro problema que se tiene es que las tecnologías de la industria de la música tienden a ser muy costosas y esto inhibe la creatividad, ya que sin acceso a ellas la música tiende a ser peor y más complicada de crear. Finalmente, otros *software* que tienen sistemas de audio generalmente no trabajan con archivos MIDI para el sonido. Estos archivos generalmente están reservados para Estaciones de Trabajo de Audio Digital y estas estaciones son muy complejas de aprender y no cuentan con posicionamiento de las fuentes de sonidos.

En este trabajo se diseña e implementa un *software* de orquestación musical en C++ con compilación a través de CMake, el uso de bibliotecas externas sin mayores dependencias externas y un repositorio git alojado en GitHub[1]. La solución es *open source* o de código abierto y tiene licencia MIT para que su uso sea universal. El *software* descrito lee una distribución o configuración en un archivo JSON y renderiza en una ventana una sala estilo teatro con los instrumentos y parlantes de la configuración en el escenario. El usuario puede mover la cámara para escuchar desde distintos ángulos, reproducir el sonido y activar o desactivar el bucle en los sonidos. La configuración se puede cambiar para leer archivos dispuestos por el usuario y cambiar la posición de los objetos del escenario junto con su visualización.

Para validar que la solución es pertinente con el problema, se facilitó el programa a 4 usuarios de prueba que representan los grupos a quienes afectaría esta solución. Estos son, músicos profesionales, estudiantes de música, músicos aficionados y arquitectos. Los resultados fueron aceptables y cumplieron con los objetivos. A pesar de esto, la facilidad de uso del *software* se vio afectada en la validación, ya que la configuración en archivos JSON, que si bien es intuitiva para desarrolladores, no es intuitiva para cualquier usuario del programa, por lo que requiere un detallado manual con instrucciones de como crear una distribución y de como se vera reflejada la configuración en el programa.

Como trabajo futuro se destaca la creación de una interfaz de usuario amigable y con las funcionalidades básicas y desarrollar un enfoque en el cálculo de acústica integrado por el sistema de audio para que la simulación sea lo más cercano a la realidad.

[1] https://github.com/HanPollo/Acousent.git

Para mi madre y mi padre,
sin quienes esto no hubiese sido posible.

Agradecimientos

Primero quiero agradecer a mi profesor guía Daniel Calderón, el cual creyó en mí para poder desarrollar un trabajo de título enfocado en la música y me ayudo con la elección del tema a tratar. Luego quiero agradecer a mis padres, sin quienes no podría haber terminado mi carrera ni hubiese tenido la motivación necesaria para dar lo mejor de mí a lo largo de mi carrera. También quiero agradecer a los 4 usuarios de prueba que me ayudaron con la validación de mi trabajo de título. Estos son Benjamín Zamorano, Eric Mayne-Nicholls, Antonio Duval y Matías Greenhill. También quiero agradecer a Valentina Spoerer, ya que sin su apoyo nunca hubiese terminado esta memoria. Finalmente, quiero agradecer a mi familia y mis amigos que siempre me apoyaron y entendían que no siempre tenía tiempo para ellos a pesar de querer tenerlo y que fueron de mucha ayuda a lo largo de mi carrera.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

Para músicos en todo el mundo, siempre ha sido de gran utilidad el variado *software* que ayuda en la composición, producción y enseñanza musical. No obstante, a menudo surgen situaciones en las que los músicos, ya sean compositores, profesores o simplemente aficionados, no pueden encontrar un *software* específico para sus necesidades. Esto es particularmente cierto en el caso de la orquestación virtual, lo cual afecta principalmente a compositores, directores y profesores de música.

En la actualidad, no existe un *software* o biblioteca que permita a los usuarios sumergirse plenamente en la experiencia de un escenario virtual de manera sencilla. Es decir, escuchar la acústica de los distintos instrumentos o parlantes en el escenario y, al mismo tiempo, visualizar los instrumentos en sus ubicaciones específicas.

Este *software* podrá ser empleado en el ámbito educativo para enseñar a los estudiantes de música cómo posicionar los instrumentos en un escenario y comprender por qué las orquestas siguen arreglos estándar. Además, los directores de orquesta podrán utilizarlo para evaluar cómo se escucha su disposición de instrumentos y los ingenieros de sonido podrán comprobar la calidad del audio en conciertos. Asimismo, arquitectos y constructores podrían emplearlo como base acústica al diseñar teatros y escenarios. Los músicos aficionados también podrán simular fácil e interactivamente cómo sonará su música con diferentes disposiciones instrumentales en el escenario, facilitando la mezcla de audio digital. Por último, este *software* sentará las bases para futuros programadores que deseen crear programas más complejos, que podrán ser utilizados incluso por los nombres más influyentes en la industria musical.

La motivación detrás de este trabajo surge principalmente de la carencia de *software* de uso sencillo para músicos. Esta problemática se agudiza por la falta de conocimientos en computación por parte de muchos músicos, lo que genera una brecha entre aquellos con habilidades computacionales y los que carecen de ellas. Un *software* de fácil manejo, que requiera poca práctica, podría acercar tanto a músicos aficionados como expertos a las ventajas de las herramientas computacionales en el ámbito musical.

1.2. Alternativas Analizadas y Alternativa Escogida

Se analizaron diversas alternativas, las cuales podrían dar inicio a la solución de este problema.

- Plug-in VST[1]:
  - Los plug-ins VST pueden ser usados por distintos DAW[2] pero estos son muchísimos y de capacidades diversas.
  - Los DAW[2] de mayor uso requieren de una gran inversión de tiempo y dinero.
- Biblioteca C++
  - Si bien una biblioteca sería el paso inicial a que se cree un programa apto para tanto músicos como no músicos para orquestación, esto requiere más tiempo o un equipo de mayor tamaño.
  - Esto ya que la biblioteca no solucionaría el problema, sino que delegaría el trabajo a un equipo con mayores recursos para que se pueda resolver el problema.
- Programa creado en un motor de videojuegos
  - Esto podría ser el paso inicial a resolver el problema, pero esto quitaría el control sobre cada aspecto del *software*.
  - También sería una solución con menos flexibilidad y escalabilidad, ya que dependería del desarrollo de dicho motor de videojuegos y no es muy adaptable o escalable a medida que las necesidades evolucionan.
- Programa en C++
  - Esto entregaría una solución inmediata con el uso de configuraciones demo y genéricas.
  - Además, sería un proyecto tanto de desarrollo de nuevas tecnologías como de aprendizaje sobre conceptos y algoritmos utilizados en programas de la industria de la música.
  - Finalmente, se puede garantizar la integración y compatibilidad con sistemas o *hardware* específicos necesarios para el proyecto.

Se escogió la última alternativa, ya que con el análisis de estas se encontró que era la mejor forma de abarcar el problema con las tecnologías estudiadas y el tiempo entregado.

1.3. Objetivos

El objetivo general de este trabajo es el de crear un *software* de fácil uso para ayudar a músicos, profesionales y aficionados a visualizar las posiciones de los instrumentos en una orquesta y escuchar cómo suena el arreglo de posiciones de los instrumentos en una posición particular de la sala con los sonidos pertinentes de cada instrumento.

1.3.1. Objetivos Específicos

1. Implementar un sistema de audio 3D para la reproducción de sonido con fuentes en distintas posiciones.
    - Reproducción de múltiples fuentes de audio simultáneas.
    - Importación de audio en archivos tipo WAVE.
    - Importación de audio en archivos MIDI con su SoundFont respectivo.
2. Implementar un sistema de renderizado 3D para la visualización del teatro y los instrumentos en el escenario.
    - Importación de modelos 3D para ser renderizados en pantalla.
    - Implementación de cámara de usuario.
    - Implementación de sombreadores básicos.
3. Implementar un sistema de configuración del programa para su fácil uso.
    - Importación de archivos JSON.
    - Lectura correcta de los elementos del archivo de configuración.
4. Objetivos de aprendizaje
    - Desarrollo de un proyecto *Open source*.
    - Uso de clases y objetos.
    - Correcto uso de bibliotecas de C++.
    - Planificación y manejo del tiempo.

1.4. Metodología

El trabajo realizado en esta memoria partió con el desarrollo de un programa demo, el cual consistía de un programa con visualización 2D con audio tridimensional. Esto hizo que se partiera familiarizando con todo lo que es OpenAL. C++ y CMake. Una vez completado este demo se partió con el programa principal. Para esto se comenzó adaptando el demo a un ambiente 3D con importación de modelos y mallas para la visualización de la sala y objetos. Se comenzó con los parlantes o altavoces, ya que ya se había trabajado antes con la reproducción de archivos de audio WAVE.

Luego se comenzó a trabajar en la incorporación de una interfaz de usuario. Para esto se decidió utilizar la biblioteca ImGui[3] pero rápidamente se comenzó a tener problemas con una interfaz de usuario suficientemente buena para la resolución del problema. Luego de un tiempo se decidió descartar la posibilidad de una interfaz de usuario y, en cambio, se enfocó en la lectura de archivos y la incorporación de diversas fuentes de sonido para incluir varios instrumentos en la escena. Para esto se trabajó en una posible configuración y esta configuración viene dada por un archivo JSON [1] para su fácil lectura e interpretación.

[1] Los archivos JSON son una colección de pares clave-valor. Estas claves y valores pueden ser casi cualquier cosa, desde números y cadenas hasta arreglos y otros objetos.

Finalmente, se debían incluir archivos Midi[2] y Soundfont[3] para poder decidir que instrumento reproduce que archivo MIDI. Se descubrió que los archivos MIDI tienen una variedad de pistas distintas (hasta 16) y que se debía poder separar el archivo en sus distintas pistas. Para esto se investigó en detalle la documentación de OpenAL y las distintas bibliotecas que podrían ser capaz de ayudar en el proceso de lectura de archivos Midi. Se investigó sobre la biblioteca fluidSynth[4] y sobre el *framework* JUCE[5] los cuales rápidamente se dieron de baja, ya que requerían muchas dependencias externas o requerían de mucha investigación cuando el periodo de investigación ya había terminado y era el momento de desarrollar. Finalmente, se decidió por utilizar la biblioteca TinySoundFont y TinyMidiLoader que funcionan como bibliotecas de encabezado único sin dependencias externas.

Luego se comenzó con el análisis sintáctico correcto de los archivos y su interpretación, lo cual hizo posible tener varios instrumentos en escena con archivos Midi distintos o iguales y un archivo SoundFont asociado a cada uno. Se partió con la reproducción de un archivo Midi completo con un archivo SoundFont completo asignado según los mensajes MIDI.

Finalmente, se crearon las funciones pertinentes para poder dividir el archivo MIDI y el archivo SoundFont para poder mezclar y combinar distintos tipos de instrumentos para distintas pistas del MIDI. Tras esto se trabajó en la creación de distribuciones funcionales y luego se realizó la validación con usuarios finales como músicos profesionales, estudiantes de música, arquitectos y músicos aficionados sin experiencia con *software* complejos.

1.5. Descripción general de la solución

La solución trata de un *software* de orquestación musical que fue desarrollado con código C++ con el proceso de compilación hecho a través de CMake y la incorporación de bibliotecas externas como OpenGL, OpenAL, Assimp, TinySoundFont y otras utilizadas como submódulos del repositorio GitHub.

El *software* consiste en un entorno 3D en donde se encuentra una cámara que representa al usuario y una sala en donde el usuario se encuentra. Una distribución en archivo JSON es utilizada como configuración de las fuentes de sonido en escena. Las fuentes de sonido están divididas en dos tipos, las que tienen un archivo de audio asignado y las que tienen un archivo Midi y un archivo Soundfont asociado. Ambas incluyen el nombre de los archivos a leer, la posición de la fuente en el escenario y el nombre del modelo 3D asignado.

El usuario puede mover y bloquear la cámara, activar el loop de las reproducciones de audio y reproducir y pausar el audio de los instrumentos. Para compilar la solución es necesario tener CMake. Esto se puede realizar de forma sencilla con Visual Studio[6] para Windows.

Finalmente, el usuario puede crear sus propias distribuciones siguiendo el formato JSON de distribution.json que se muestra en el Apéndice C mientras tenga los archivos de audio en la ubicación correcta. Se puede ir probando las posiciones de los instrumentos y los archivos

[2] Un archivo MIDI contiene una serie de mensajes que dictan cómo, cuándo y qué notas deben tocarse, entre otros comandos.

[3] Archivos de audio basados en muestras para notas para poder alterarlas y tener todo el registro del instrumento.

de audio. El ejemplo de distribución incluye 5 instrumentos y 2 parlantes, pero los audios de los parlantes no coinciden con los de los instrumentos, es decir, no aportan a la música que se escucha (aunque la composición musical no tiene que seguir el marco teórico de la música necesariamente).

1.6. Contenidos

El presente documento parte con una introducción al problema abordado en este trabajo de título, el contexto donde este existe y la motivación detrás de solucionarlo. Luego se describen los objetivos que el *software* desarrollado debe cumplir, seguido de la metodología usada durante la implementación de este y una descripción general de la solución realizada.

El segundo capítulo es del estado del arte, en donde se estudian los distintos *software* musicales y funcionalidades de estos. El estado del arte incluye los distintos tipos de *software* musicales y sus funcionalidades y tipos de uso, como se maneja el audio inmersivo en *software* musicales, La poca visualización 3D que existe hoy en día en *software* musicales y porque es necesaria un avance en este ámbito, como funciona la lectura de los distintos tipos de archivos de audio incluyendo archivos Midi y Soundfont y como funciona el cálculo de acústica en los distintos *software* musicales existentes. Finalmente, se explican las bibliotecas disponibles en C++ comúnmente utilizadas para *software* de música.

El tercer capítulo describe en detalle el problema planteado a resolver con este trabajo de título.

Luego, el siguiente capítulo describe la solución en detalle y como funciona cada aspecto y funcionalidad del *software* y en sí como soluciona el problema planteado. Esto incluye la arquitectura del *software*, las bibliotecas externas que se utilizaron y el porqué se utilizaron, el diseño de las clases del programa y porque son de utilidad para la escalabilidad de la solución y de su desarrollo. Finalmente, se menciona como se utiliza el *software* y sus dependencias.

El siguiente capítulo muestra las validaciones que se realizaron para comprobar que la solución del desarrollo del *software* soluciona el problema planteado. Estas validaciones fueron la entrega del *software* a usuarios finales para que puedan opinar sobre las funcionalidades y el uso del programa.

Finalmente, el capítulo 6 describe las conclusiones de este trabajo de título y el posible trabajo futuro que se debe realizar para perfeccionar la solución y hacerla más accesible para todos los posibles usuarios.

Capítulo 2

Estado del Arte

Actualmente, no existe *software* libre (*open source*) que permita al usuario posicionar distintos instrumentos en una sala y hacer que el sonido se distribuya como en una sala de verdad. Aun así, sí existen tipos de *software* que facilitan la vida a los músicos en distintos ámbitos. El ejemplo de *software* más parecido a lo que se plantea sería el VST (*Virtual Studio Technology*, plug-ins de audio que integran sintetizadores y efectos de audio [1]) *VirtualSoundStage* [7] lo cual es una herramienta que se debe utilizar en algún *DAW* (Estación de trabajo de audio digital o *Digital audio workstation*) [2] por lo que hace que sea más complicado su uso. Además de esta complicación, el *software* no es de código abierto e incluso hay que pagar para su uso.

Además de este *software*, hay muchas más aplicaciones que solamente dificultan al usuario en términos de ubicación de instrumentos en una sala, ya que son muy complicadas de utilizar para un novato en la computación. Estos incluyen algunos de los distintos tipos de DAW[2] y algunos programas para acústica de salas (*Odeon Room Acoustics software* [8]). Estos y muchos más programas podrían estar incorporados en un programa de fácil uso y de forma libre para así ayudar a músicos a empezar a familiarizarse con los *software* útiles para músicos.

En las siguientes secciones se describirá el estado del arte de los diversos tipos de *software* musicales y sus elementos. Luego se describirá el estado de arte de cada uno de los elementos más importantes implementados dentro del programa desarrollado en este Trabajo de Título. Cada elemento contiene una descripción de lo que se busca con este elemento y para qué sirve en términos de los *software* musicales. También contiene ejemplos básicos de estos elementos en distintos *software* musicales y sus características. Finalmente, cabe destacar que el programa desarrollado no perfecciona cada uno de estos elementos, sino que propone un inicio a lo que sería el elemento en un *software* musical de fácil uso y apto tanto como para músico como no músicos.

2.1. Tipos de *software* musicales

Hoy en día, en términos prácticos, músicos expertos necesitan ayuda de ingenieros para poder realizar simulaciones que un músico debería poder hacer con facilidad. También músicos aficionados no pueden acceder a estos programas por sus costosas licencias. Además de esto, en las instituciones principiantes de enseñanza musical como colegios y liceos no se utilizan *software* de música para ayudar a los estudiantes a ver, entender y simular lo que es un

escenario y tienen que aprender todo de libros, dificultando el aprendizaje de los estudiantes. Sobre todo considerando el dominio artístico de la música, donde ya es extremadamente difícil retratar lo auditivo en un libro o texto escrito. Con un trabajo novedoso en este ámbito se puede lograr incentivar a nuevas personas a querer entrar en el área de la composición musical.

2.1.1. Digital Audio Workstation DAW

Existen varios tipos de *software* musicales que cubren distintos tipos del mundo de la música. Entre ellos se encuentran los *DAW* (Estación de trabajo de audio digital o *Digital audio workstation*)[2] los cuales son esenciales para componer, producir, editar y manejar audio de la forma más completa posible. Estos incluyen todo lo necesario para la grabación y masterización de audio y creación de audio a través de controladores y archivos MIDI.[9] Estos utilizan una interfaz de usuario común para trabajar, interactuar con el hardware del dispositivo y poder crear archivos de tipo audio para luego ser publicados internacionalmente.

El proceso de creación de música consiste en varios pasos. Primero la grabación, la cual se puede hacer con entradas de audio hacia el hardware y los drivers necesarios para traducir estas señales de audio. También se puede grabar utilizando controladores MIDI o un teclado de computador para crear notas en ciertos tiempos con un cierto sonido para luego transformar esto a audio. Luego se puede editar el audio agregando efectos o editar los mensajes MIDI cambiando las notas. Finalmente, se debe masterizar para que el audio no tenga sonidos no deseados y para que sea más agradable al oyente. Todo este proceso es largo y requiere un gran conocimiento musical, de producción y principalmente de la interfaz de usuario del DAW, ya que cada DAW es distinto. Productores o estudios suelen ser contratados por los compositores para ahorrarse el trabajo de producir, ya que este paso requiere mucho conocimiento de diversos programas complejos de audio.

Hay varios ejemplos de *software* DAW. Entre ellos los más conocidos son 4 por sus diversos casos de uso.

El primero del cual se debe hablar es *Avid Pro tools* [10] de Avid. Pro Tools es una destacada estación de trabajo de audio digital o DAW utilizada ampliamente en la industria musical, de cine y televisión. Pro Tools permite la grabación, edición y mezcla de audio con una interfaz bastante intuitiva. Ofrece una amplia gama de herramientas y características, como la edición precisa de audio, la composición musical, la automatización avanzada y una amplia colección de plug-ins y efectos de alta calidad. El *software* es conocido por su capacidad de trabajar con proyectos complejos y por ser un estándar en estudios de grabación profesionales debido a su calidad de sonido y a su compatibilidad con flujos de trabajo colaborativos. Sin embargo, sigue teniendo como requisito saber mucho de producción musical como todos los DAW y, por lo tanto, no son programas que el usuario puede llegar y utilizar en su fácilmente desde un inicio sin tener conocimientos profundos.

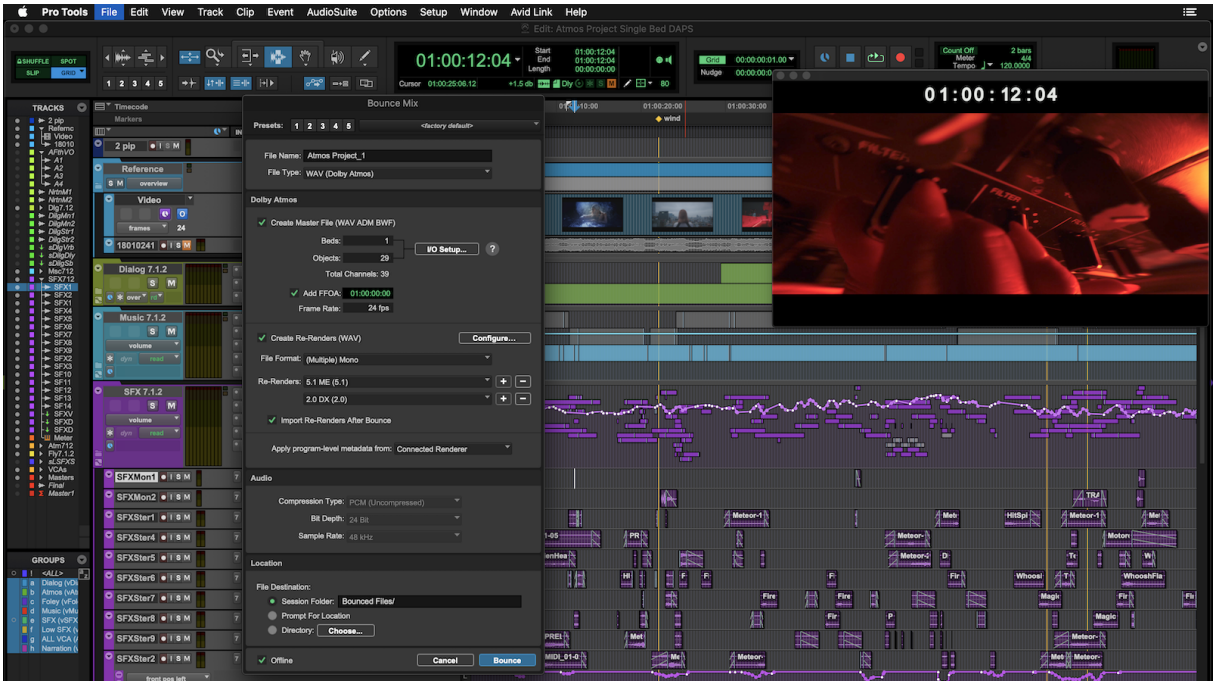

Figura 2.1: Interfaz gráfica de Avid Pro Tools[10]

.

Otro es *Ableton Live*[11] el cual es muy usado ahora mismo por su diversas capacidades y uso general en estudios mundialmente. Ableton se enfoca en adaptarse a todo y a poder dar todas las capacidades necesarias para todo lo que sea audio. Ableton Live, así como la mayoría de los *software* DAW, tiene una complejidad muy alta de uso y para poder utilizarlo en su máxima capacidad se debe tener años de experiencia tanto con *software* DAW como específicamente Ableton Live.

Figura 2.2: Interfaz gráfica de Ableton Live[11]

.

Luego está Logic Pro de Apple[12] que es el DAW más utilizado por hardware con sistema operativo iOS, ya que está hecho por Apple mismo. Este también cuenta con todas las funcionalidades básicas que necesitan los DAW, pero enfocado en el sistema operativo de Apple y sus diversas diferencias con otros sistemas operativos. Nuevamente, es un DAW que ofrece muchas funcionalidades, pero el costo de estas es la dificultad de aprender a utilizar el programa de manera óptima y crea muchas diferencias entre músicos que saben de los programas con los que están recién empezando a producir su música.

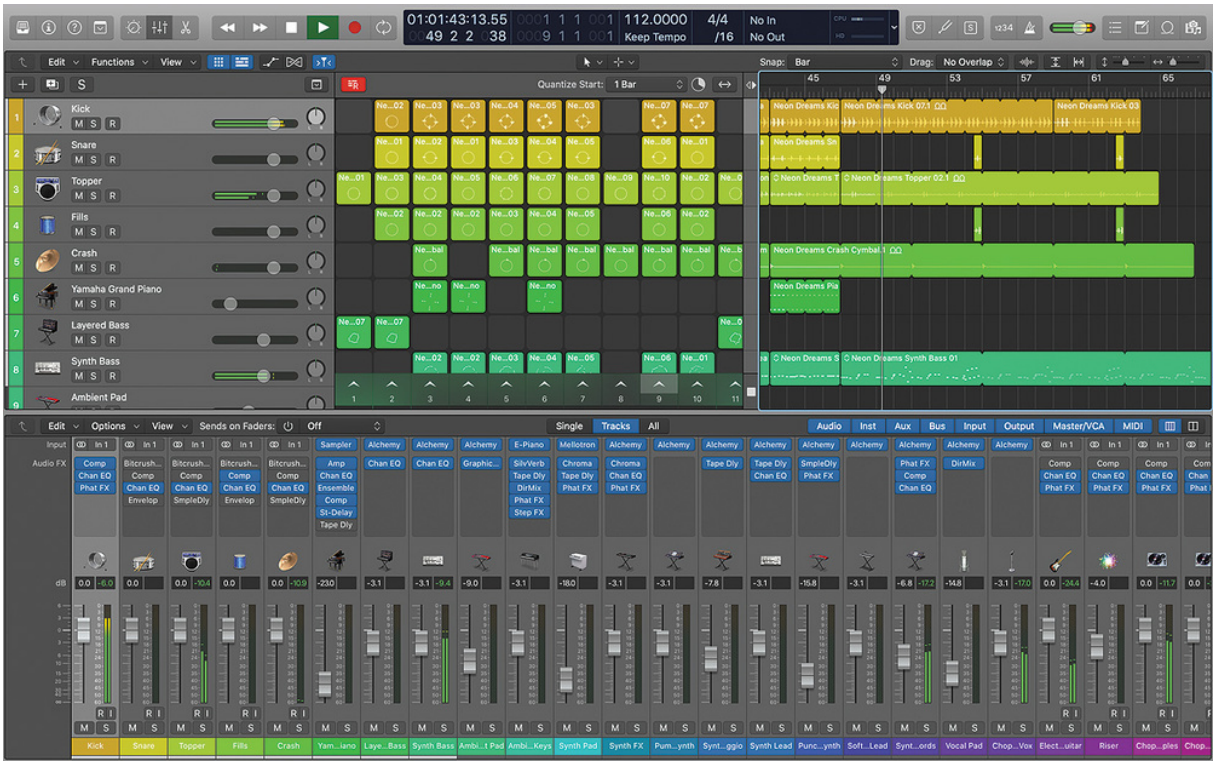

Figura 2.3: Interfaz gráfica de Logic Pro[12]

.

Otro DAW utilizado es Ardour[13] que es Open Source y es el DAW de código abierto más utilizado. Este cuenta con las funcionalidades básicas y complejas de todos los DAW, pero no es tan abiertamente utilizado por diversas razones. Primero, los DAW comerciales a menudo vienen con un conjunto completo de funciones, incluyendo instrumentos virtuales, efectos de alta calidad y capacidades MIDI avanzadas. Si bien los DAW de código abierto como Ardour han avanzado mucho en los últimos años, es posible que no ofrezcan la misma amplitud de funciones o el mismo nivel de integración. Otra razón es que los DAW comerciales tienden a tener una mayor variedad de complementos de terceros, instrumentos virtuales y bibliotecas de sonidos disponibles. Estos pueden mejorar las posibilidades creativas y ofrecer más opciones para el diseño de sonido y la producción musical. Finalmente, utilizar los DAW comerciales puede garantizar la compatibilidad con colaboradores, estudios y otros profesionales.

Figura 2.4: Interfaz gráfica de Ardour[13]

.

Finalmente, hay otros DAW menos utilizados como Cubase [14] de Steinberg, una empresa increíblemente importante en el área de *software* musical, pero se ha quedado un poco atrás en uso, ya que lo que más dicta el DAW de preferencia es la cantidad de artistas y productores que lo están utilizando y Cubase ha ido bajando en su uso. A continuación se muestra un gráfico que muestra una encuesta hecha a usuarios que utilizan algún tipo de programa DAW y muestra la cantidad de uso que tienen[15].

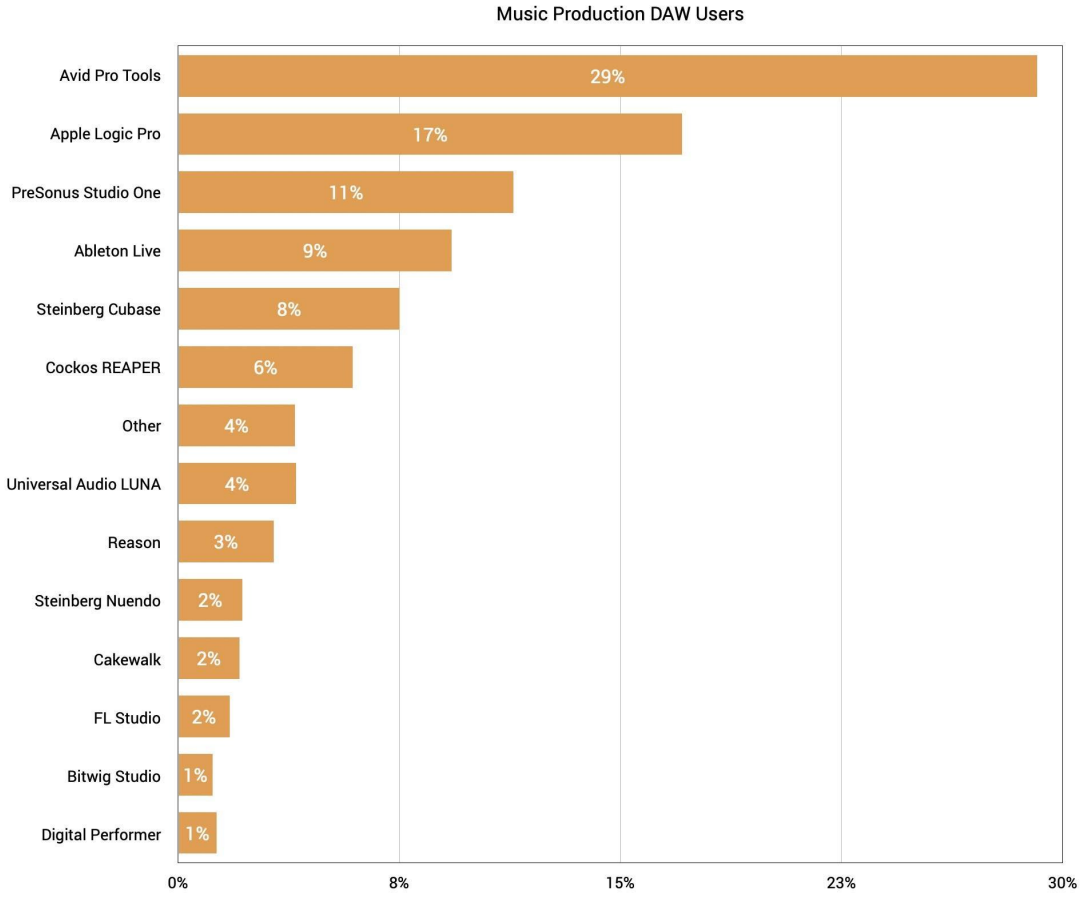

Figura 2.5: Datos de encuesta hecha por Production Expert a sus usuarios para ver las estadísticas de uso de los distintos DAW[15]

.

2.1.2. VST y Plug-ins

Los VST (Virtual Studio Technology)[1] son estándares de plug-ins de *software* utilizados en DAW (estaciones de trabajo de audio digital) para añadir funciones y efectos adicionales a las producciones musicales. Los plug-ins para DAW son componentes de *software* que pueden extender las capacidades del DAW, añadiendo efectos de audio, instrumentos virtuales y procesadores de sonido. Tienen diversos usos, como la creación de efectos de sonido, la modificación de timbres, la manipulación de audio y la creación de ambientes sonoros. Estos suelen tener interfaces gráficas propias para describir los efectos de audio o la masterización que se está realizando con este plug-in.

Algunos ejemplos de plug-ins y plug-ins VST son los siguientes.

- Virtual Sound Stage[7]
  - Un Plug-in de *Parallax Audio* que simula un teatro estándar para poder hacer la masterización del audio de acuerdo con como sonarían los distintos instrumentos en un escenario.

- Native Instruments Kontakt[16]
  - Es un sámpler virtual[17] que permite cargar y manipular una amplia variedad de instrumentos virtuales y bibliotecas de sonidos. Se utiliza para crear instrumentaciones realistas y expresivas.
- Waves SSL G-Master Buss Compressor[18]
  - Este plug-in simula el compresor de la consola de mezcla SSL[19], utilizado en la etapa de masterización musical para unir las mezclas en una experiencia sólida y cohesionada.

Existen muchos otros Plug-ins para todo tipo de creación, edición y grabación de sonido, pero todos se deben usar con DAW u otro tipo de *software* de producción de audio y por esta razón componer música de la manera que lo hacen en la industria es muy costoso y complicado.

Tanto los DAW como los Plug-ins pueden ser utilizados en conjunto, es decir, se pueden utilizar 2 o más DAW para una producción en distintas partes de las pistas de audio y en conjunto con esto se pueden instalar varios plug-ins en distintos DAW para crear sonidos distintos. Esto, si bien complica el arte de componer y producir música y sonido, también logra que cada artista y/o profesional tenga su propio estilo sin incluir su propia creatividad al minuto de componer o producir.

2.2. Audio Inmersivo

El audio inmersivo consiste en la reproducción de audio en una simulación de espacio tridimensional para que el oyente pueda relacionar lo que escucha con la ubicación espacial de la fuente de audio. Los humanos tienen la capacidad de identificar las distintas fuentes de sonido con muy buena precisión. Esto lo pueden hacer interpretando la amplitud del audio, es decir, que tan fuerte se escucha y la diferencia de audio entre las orejas.[20]. Esto produce una mayor inmersión en simulaciones, ya que, si bien en un teatro, auditorio o al aire libre se puede identificar las ubicaciones de las distintas fuentes de sonido, en un ambiente virtual no es el caso necesariamente.

Se puede notar como esto ha sido implementado en distintas tecnologías innovadoras a lo largo de los últimos años. *Apple* ha mejorado sus famosos *AirPods* [21] con un audio espacial. En el cine, el sonido envolvente o *surround sound* ha generado que miles de personas adapten este audio espacial en sus salas de estar.[22] Esto demuestra que las tecnologías avanzan para que el ser humano se sienta cada vez más parte de la experiencia que está teniendo, aunque esto solo sea a través de una pantalla o audífonos.

El enfoque principal por el audio inmersivo es por el gran efecto de inmersión que produce esto en comparación con otras optimizaciones de audio. Esto se puede ver en la importancia del efecto panorámico o *panning* en la masterización de música.[23]

Actualmente, la mejor forma de conseguir inmersión de audio es a través de la función de transferencia relacionada con la cabeza (HRTF) o *Head-related transfer function* [24]. La

HRTF es una función que describe cómo el sonido se transforma a medida que viaja desde una fuente sonora en el espacio hasta los oídos de un oyente, teniendo en cuenta la forma y características físicas de la cabeza y las orejas. Estas transformaciones son fundamentales para la percepción espacial del sonido en humanos y otros animales con audición estereofónica. Cuando un sonido se origina en el espacio, viaja hacia los oídos del oyente y se encuentra con obstáculos como la cabeza y las orejas, el sonido se modifica de varias maneras antes de llegar al tímpano. Estas modificaciones incluyen:

- Atenuación espacial: La cabeza bloquea parcialmente el sonido que llega al oído más alejado de la fuente sonora. Esto significa que el oído más cercano recibe un sonido más fuerte que el oído más lejano, lo que proporciona información sobre la dirección del sonido.
- Retraso temporal: El sonido tarda más tiempo en llegar al oído más alejado que al oído más cercano. Esto crea una diferencia en el tiempo de llegada entre los dos oídos, lo que también es una clave para la percepción de la dirección del sonido.
- Difracción y reflexión: La forma de la cabeza y las orejas puede causar difracción y reflexión del sonido, lo que afecta la intensidad y la fase del sonido que llega a cada oído.

Antes de la HRTF, lograr una experiencia de sonido espacial realista era un desafío considerable. La HRTF ha permitido avances significativos en aplicaciones como la realidad virtual, los videojuegos o la puesta en escena, donde la percepción precisa de la dirección del sonido es esencial para la inmersión del usuario.

2.2.1. Audio Inmersivo en *software*

En los diversos *software* que reproducen audio, tales como videojuegos, aplicaciones web o cualquier tipo de *software* audiovisual, se requiere el procesamiento de audio y la reproducción de este a través del hardware para que el usuario lo oiga. Los distintos *software* que deben reproducir audio de manera realista deben hacer un cálculo acústico muy costoso, pero se puede lograr una gran inmersión con solo utilizar audio espacial y por eso cada día hay más tecnologías que lo adaptan.[25]

Los *software* audiovisuales intentan brindar al usuario con cada vez más realismo en la entrega de lo audiovisual. En los videojuegos, el audio inmersivo es crucial para crear la capacidad de sumergir al jugador en un mundo cautivador y envolvente, donde cada detalle está cuidadosamente diseñado para crear una experiencia que trasciende la pantalla. A medida que se explora este universo, la vista es crucial en la inmersión del usuario, pero si no está acompañada de sonido inmersivo, la experiencia pasa a ser nuevamente la de la vida real y que simplemente es un juego. El alza de la Realidad Virtual hace que los videojuegos o simulaciones deban ser completamente inmersas, por lo que, si bien la vista y la responsividad del aparato VR crean dos tercios de la inmersión, el audio espacial crea el otro tercio.

Los *software* musicales, como se puede ver por su importancia, deben tener audio espacial o inmersión, pero la realidad de hoy en día es que se requiere de mucha práctica y clases con ciertos *software* para poder tener una pieza que incluya la espacialidad. Lo que más va a hacer un compositor para crear tridimensionalidad en su composición es utilizar efectos panorámicos o *panning* [23] lo cual si bien genera una tridimensionalidad horizontal deficiente, esta

no esta dictada por las leyes físicas del sonido, sino que por como ellos desean que suene su música. Los compositores generalmente utilizan DAW [2] para trabajar en las composiciones y generalmente están obligados a contratar a un productor para que se las produzca (el cual estudio al menos 2 años de producción musical y debe tener más de experiencia) los DAW son *software* en 2 dimensiones y trabajan con audio de 2 dimensiones con la excepción de algunos plug-ins que se pueden utilizar, tal como el programa *Dolby Atmos Renderer* [26] pero estos complican más la masterización del audio. Por estas razones en general solo se utilizan las funcionalidades del programa DAW a elección que generalmente no incluyen inmersión del audio.

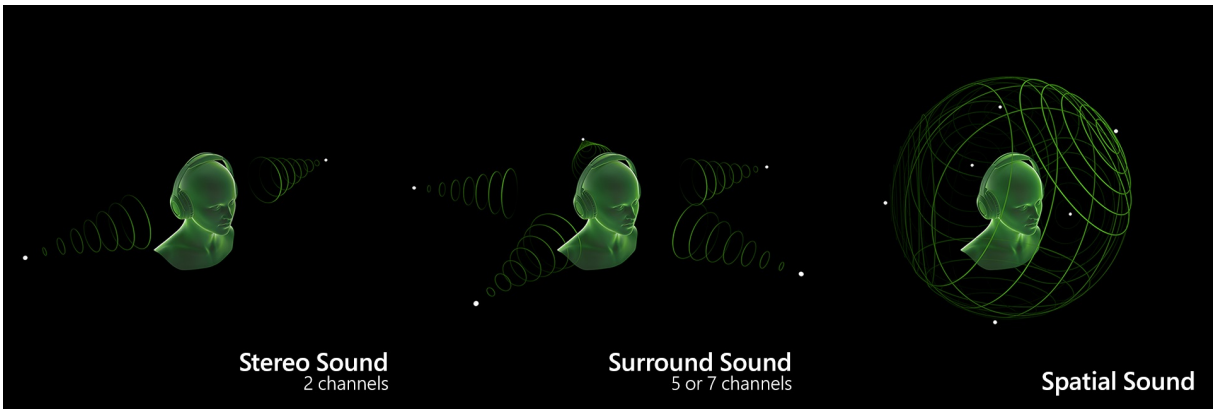

Figura 2.6: Imagen en la cual se ven las diferencias en las dimensiones del sonido extraída de [27]

2.3. Visualización Renderizada de Puesta en Escena

La renderización de gráficos como elemento de visualización de una puesta en escena para orquestación musical o ensamble musical no es un enfoque actual en los *software* musicales y generalmente los únicos que pueden tenerlo son plug-ins de poco uso y poco conocidos. Esto puede estar bien para un músico que se enfoca solamente en el audio y calidad de audio de su música, pero cuando se trata de un no músico que quiere sentir una experiencia de música no puede hacerlo sin ir en vivo a un concierto o interpretación musical en vivo.

A pesar de esto, en el área de los videojuegos se enfoca tanto en lo gráfico como en el audio, por lo que se tienen videojuegos específicos que logran estas simulaciones. Específicamente este año salió el acceso temprano de un videojuego de realidad virtual que tiene la puesta en escena de una orquesta. Este juego es Maestro VR[28] y es un juego de ritmo para plataformas de realidad virtual en la que el jugador es un conductor de orquesta y debe conducir a la orquesta dependiendo de la música. Por ahora esta en acceso temprano, por lo que no se pudo realizar una investigación profunda del juego o de como funciona el tema de renderizado o audio.

Es importante mencionar que, aunque los DAW ofrecen herramientas avanzadas para la producción y edición de audio, la visualización renderizada en 3D de la puesta en escena no es una característica común en estos programas.

2.4. Lectura de Archivos de Audio

Un formato de archivo de audio es un contenedor multimedia que almacena una grabación de audio, ya sea música, voces, entre otros. Estos formatos se diferencian entre sí por sus propiedades, cómo se almacenan los datos, sus capacidades de reproducción y cómo pueden ser utilizados en un sistema de administración de archivos.

El audio digital generalmente se almacena mostrando el voltaje de audio, que al reproducirse, corresponde a un nivel de señal en un canal individual con una cierta resolución, es decir, el número de bits por muestreo en intervalos regulares, creando la frecuencia de muestreo. Estos datos pueden ser almacenados sin comprimir o comprimidos para reducir el tamaño del archivo.

Existen diferentes tipos de formatos según la compresión del audio:

- Formatos de audio sin comprimir: Como WAV, AIFF y AU. WAV y AIFF son formatos flexibles creados para almacenar varias combinaciones de frecuencia de muestreo o tasa de bits, lo que los hace adecuados para archivar grabaciones originales.
- Formatos de audio comprimido sin pérdida (Lossless): Estos formatos, como FLAC, MPEG-4 SLS, MPEG-4 ALS, entre otros, requieren más tiempo de procesamiento que los formatos sin comprimir, pero son más eficientes en cuanto al espacio que ocupan.
- Formatos de audio comprimido con pérdida: En este sistema, como MP3, Vorbis, AAC, entre otros, los datos se comprimen descartando partes de ellos. El proceso intenta minimizar la cantidad de datos que mantiene el archivo, reduciendo su peso y, por lo tanto, su calidad.

Por otro lado, hay otra forma de guardar audio utilizando más de un archivo. MIDI es un protocolo de comunicación serial estándar que permite a los computadores y otros dispositivos musicales electrónicos comunicarse y compartir información para la generación de sonidos. A diferencia de los formatos de audio tradicionales, los archivos MIDI no contienen sonidos reales, sino instrucciones sobre cómo generar esos sonidos. Los archivos MIDI son interpretados por sintetizadores, que producen el sonido real basado en estas instrucciones.

Cuando se trata de sintetizadores y su *software*, estos generalmente utilizan archivos SoundFont. SoundFont es una marca que se refiere tanto a un formato de archivo como a una tecnología asociada que busca cerrar la brecha entre la grabación digital y la síntesis de audio, especialmente para fines de composición musical por computadora. Esta tecnología es una implementación de síntesis de sonido por muestras. Su primer uso fue en la tarjeta de sonido *Sound Blaster AWE32* para soporte MIDI. Un banco SoundFont contiene muestras de base en formato PCM[29] (similar a los archivos WAV) que se asignan a secciones específicas de un teclado musical. Además, un banco SoundFont incluye otros parámetros musicales de síntesis, como loops o bucles, efectos de vibrato y cambios de volumen sensibles a la velocidad. Estos bancos pueden adherirse al estándar del conjunto de sonidos General MIDI o tener su propia definición de conjunto de sonidos[30].

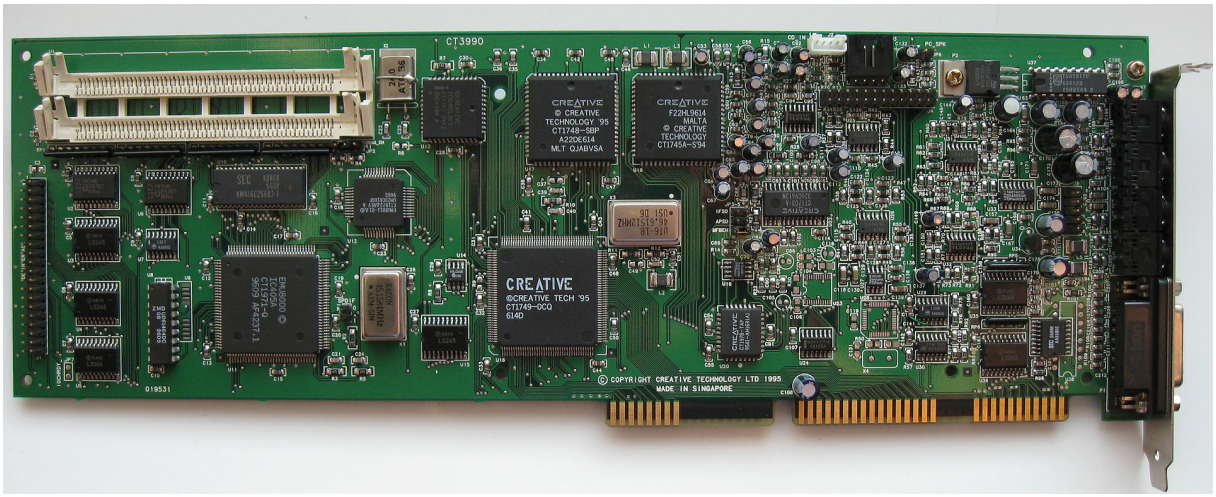

Figura 2.7: La tarjeta de sonido Sound Blaster AWE32[31] utilizo SoundFont por primera vez en 1994

.

Los formatos de archivo de audio, como .wav, .mp3 y .ogg, se utilizan ampliamente en la industria de la música para grabar, editar, mezclar y distribuir música. Los DAW (Digital Audio Workstations) y otros *software* de edición musical leen estos formatos para permitir a los productores y músicos trabajar con ellos. Por otro lado, los archivos MIDI y SoundFont se utilizan para crear y editar composiciones musicales en *software* musical, permitiendo a los músicos y productores manipular notas, ritmos e instrumentos de manera digital.

El análisis sintáctico de estos archivos implica leer y decodificar la información contenida en ellos para que pueda ser procesada o reproducida por *software* o hardware. En el caso de los archivos de audio, el análisis sintáctico implica extraer y procesar la información de audio para su reproducción. Para los archivos MIDI, el análisis sintáctico implica interpretar las instrucciones contenidas en el archivo para generar sonido a través de un sintetizador.

Los archivos MIDI son difíciles de analizar sintácticamente en un *software*, ya que no contienen audio, por esta razón no hay muchos *software* afuera de los DAW que los lean correctamente. A pesar de esto, se están comenzando a incluir en videojuegos y *software* de aprendizaje. Algunos videojuegos que lo utilizan son el Clone Hero[32] o el juego Final Fantasy XIV[33] que introduce la funcionalidad de poder realizar *Performance Actions* [34] las cuales son lectura de señales Midi e incluyen sonidos SoundFont para reproducir el sonido.

2.5. Cálculo de Acústica

El cálculo de acústica en *software* musical ha revolucionado el mundo del diseño sonoro, producción musical y análisis de audio. Esencialmente, este cálculo permite emular de manera precisa el comportamiento del sonido en diferentes ambientes y contextos. Por otro lado, el cálculo preciso de acústica depende de muchas variables, incluyendo pero no limitado a posición de la fuente de sonido, posición del oyente, ángulos de reflexión, factores de reflexión y absorción, materiales de la sala, superficie de la sala, y mucho más. Es por esta razón que el cálculo simulado de acústica esta generalmente reservado para *software* y *hardware* con alta capacidad de procesamiento.

A continuación se explicarán algunos conceptos esenciales del cálculo de acústica:

2.5.1. Reflexión y dispersión de ondas de sonido

La reflexión en ondas se produce cuando las ondas sonoras encuentran una superficie (pared, techo, etc.), parte de la energía sonora se refleja hacia el espacio. La ley básica de la reflexión del sonido establece que el ángulo de incidencia es igual al ángulo de reflexión. Esta característica es esencial al considerar la forma y disposición de las superficies en una sala para evitar ecos y focalizaciones indebidas del sonido.

Una manera de calcular la reflexión es usando el coeficiente de reflexión, que relaciona la energía reflejada con la energía incidente en una superficie. Si E_i es la energía incidente y E_r la energía reflejada, el coeficiente de reflexión R se calcula como:

$$R = \frac{E_r}{E_i} \tag{2.1}$$

En acústica de salas, la dispersión se refiere a la difusión del sonido en varias direcciones después de encontrarse con una superficie. Esto es típicamente deseable para evitar la acumulación de energía sonora y para distribuir el sonido uniformemente en una sala. No hay una fórmula simple para la dispersión, ya que depende de las propiedades de la superficie (forma, tamaño, material) y de la frecuencia de la onda sonora.

2.5.2. Espacios Cerrados vs. Abiertos

La acústica en espacios cerrados se ve muy afectada por el tamaño de la sala, los materiales de las superficies y el mobiliario presente, que afectará las reflexiones y la absorción del sonido. Aquí, se pueden usar modelos matemáticos como la ecuación de Sabine [35] o Eyring [36] para calcular el tiempo de reverberación y otros parámetros acústicos.

Por otro lado, en espacios abiertos las reflexiones son mínimas, y se deben considerar más los efectos del viento, la temperatura y la humedad en la propagación del sonido. El coeficiente de atenuación del aire a 20 ºC con una frecuencia de 1 MHz es 1.64. Este coeficiente se multiplica por la distancia del medio y por la frecuencia. Por esta razón, en estas condiciones ideales la atenuación será de 1.64 dB menos por cm avanzado por parte de la onda de sonido. [37]

2.5.3. Reverberación

Probablemente, el fenómeno acústico más característico en una habitación cerrada es la reverberación: es decir, el hecho de que el sonido producido en la habitación no desaparece inmediatamente después de que se apague la fuente de sonido, sino que permanece audible durante un cierto período de tiempo después, aunque con una disminución constante en el volumen. El proceso físico de la decadencia del sonido en una habitación depende críticamente de la estructura del campo de sonido. Solo se pueden formular leyes simples que describan este proceso cuando el campo de sonido es isotrópico en cualquier punto, lo que significa que

la propagación del sonido es uniforme en todas las direcciones.

Es común en acústica de salas caracterizar la duración de la decadencia del sonido, como el "tiempo de reverberación" o "tiempo de decaimiento", introducido por Sabine. Se define como el intervalo de tiempo en el que el nivel de decaimiento disminuye en 60 dB. Si T es el tiempo de reverberación, V es el volumen de la sala en metros y A es la absorción total en Sabines (la suma del producto de las superficies de todos los materiales por sus respectivos coeficientes de absorción), la ecuación de Sabine es la siguiente:

$$T = 0.161 \frac{V}{A} \tag{2.2}$$

2.5.4. Absorbentes de sonido

Los materiales absorbentes son cruciales para controlar la reverberación y el eco. Funcionan transformando la energía sonora en calor mediante fricción interna cuando el sonido penetra en el material. El coeficiente de absorción es una medida de cuánto sonido absorbe un material en comparación con la cantidad de sonido que se refleja. Se expresa como un número entre 0 (reflexión completa, como un espejo) y 1 (absorción completa, sin reflexión). Si A es la energía absorbida y E_i es la energía incidente, el coeficiente de absorción 'α' se calcula como:

$$\alpha = \frac{A}{E_i} \tag{2.3}$$

2.5.5. Cálculo de ondas de sonido

Desde el punto de vista matemático, el comportamiento del sonido se describe generalmente a través de ecuaciones diferenciales parciales, como la ecuación de onda. Estas ecuaciones modelan la propagación del sonido a través de un medio, teniendo en cuenta factores como la densidad del medio, la velocidad del sonido y la presión. Las soluciones a estas ecuaciones nos proporcionan una representación del campo sonoro en un espacio dado. A partir de estas soluciones, se pueden calcular características clave, como la respuesta al impulso de una sala, que es esencial para recrear cómo un sonido se percibiría en un entorno específico.

Además, el cálculo de transformadas de Fourier es esencial para analizar y procesar sonidos en el dominio de la frecuencia. Esta técnica permite descomponer un sonido en sus componentes de frecuencia básicos, facilitando la manipulación específica de ciertas frecuencias y ofreciendo herramientas como la ecualización.

La Figura 2.8 es el resultado de aplicar una transformada de Q constante (una transformada relacionada con Fourier) a la forma de onda de un acorde de piano de do mayor. Los tres primeros picos de la izquierda corresponden a las frecuencias de la frecuencia fundamental del acorde (C, E, G). Los picos más pequeños restantes son sobretonos de frecuencia más alta de los tonos fundamentales. Un algoritmo de detección de tono podría usar la intensidad

relativa de estos picos para inferir qué teclas presionó el pianista.

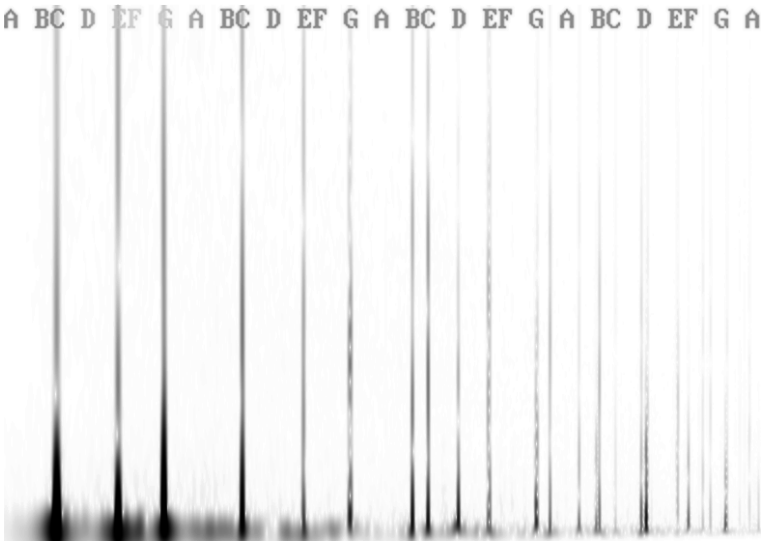

Figura 2.8: Un ejemplo de aplicación de la transformada de Fourier es determinar los tonos constituyentes en una forma de onda musical.

Los datos sobre cálculo de acústica, específicamente sobre acústica de salas, se pueden encontrar con su cálculo detallado en el libro de Heinrich Kuttruff, Room Acoustics [38].

2.5.6. Cálculo de acústica en *software*

Desde la perspectiva del *software*, el cálculo acústico se maneja mediante una combinación de técnicas de programación y algoritmos especializados. Los algoritmos de convolución, por ejemplo, se utilizan para aplicar respuestas al impulso a señales de audio, emulando la reverberación de una sala o el sonido de un amplificador específico. Otros algoritmos, como el modelado físico, recrean instrumentos musicales mediante el cálculo de cómo las ondas sonoras interactúan con diferentes materiales y estructuras, como las cuerdas de una guitarra o la piel de un tambor.

OpenAL (Open Audio Library) es una biblioteca para el manejo de sonido. Es una API de audio espacializada y tridimensional, permitiendo a los desarrolladores situar fuentes de sonido en un espacio tridimensional y recrear efectos como la atenuación, doppler, y reverberación. Para recrear estos efectos, OpenAL utiliza distorsiones y efectos en los sonidos, pero no hace el cálculo de acústica en sí. El desarrollador debe entregar todos los coeficientes y datos de los elementos para que el cálculo se acerque a la realidad. Para utilizar los efectos de OpenAL se debe utilizar la extensión de efectos de OpenAL que en sí presenta nuevos elementos como *Auxiliary Effect Slots* que son contenedores de efectos y *Effects* que son los efectos. Estos efectos pueden ser de distintos tipos, pero dependen de las constantes del medio de propagación y materiales del espacio.

En términos del cálculo de acústica en una sala modelada en 3D se requieren los coeficientes de materiales, específicamente los de reflexión y absorción para cada vértice y material.

2.6. Tecnologías externas en C++ para *software* de música

Existen diversas bibliotecas en C++ que ayudan con el manejo de funciones necesarias para crear un *software* de música.

2.6.1. Herramientas para renderizado

2.6.1.1. OpenGL

OpenGL (Open Graphics Library)[39] es una especificación estándar para la renderización de gráficos 2D y 3D en una computadora. Es una API [4] que proporciona una amplia colección de funciones para desarrollar aplicaciones que produzcan gráficos interactivos de alta calidad. OpenGL es el corazón del renderizado en las clases del *software*. Proporciona todas las herramientas necesarias para dibujar gráficos en la pantalla, desde simples primitivas hasta complejas mallas 3D con iluminación y texturas.

OpenGL esta encargada del renderizado, ya que proporciona funciones para dibujar primitivas como puntos, líneas y triángulos. También se encarga de las transformaciones que permiten la transformación de vértices utilizando matrices de transformación para operaciones como traslación, rotación y escalado. Finalmente, se encarga de la texturización que posibilita mapear imágenes (texturas) sobre primitivas e iluminación que ofrece técnicas para simular la iluminación en una escena, incluyendo luces puntuales, direccionales y de foco.

2.6.1.2. Assimp

Assimp (Open Asset Import Library)[40] es una biblioteca que permite importar y procesar modelos geométricos en diferentes formatos. Su propósito principal es convertir toda la geometría y la información de un modelo en una estructura de datos común, independientemente del formato de archivo original. Assimp actúa como el puente entre los archivos de modelo en disco y las estructuras de datos en memoria que OpenGL necesita para renderizar. Convierte una variedad de formatos de archivo en una representación común que las clases del *software* pueden utilizar fácilmente.

Assimp esta encargada de la importación al ser capaz de leer una amplia variedad de formatos de archivo populares, como glTF, OBJ, FBX, COLLADA, entre otros. También se encarga del procesamiento de estos archivos, ya que proporciona herramientas para operaciones post-importación, como la triangulación, la generación de normales, y la eliminación de duplicados. Finalmente, Assimp puede optimizar las mallas para mejorar el rendimiento de renderizado.

2.6.2. Herramientas para gestión de Audio

[4] Interfaz de Programación de Aplicaciones

2.6.2.1. OpenAL

OpenAL, que significa "Open Audio Library", es una interfaz de programación de aplicaciones (API) multiplataforma destinada a la gestión y reproducción de audio en aplicaciones interactivas. Su diseño está inspirado en OpenGL, un estándar para gráficos, lo que facilita su comprensión y adopción para aquellos familiarizados con este último. Aunque es ampliamente utilizado en videojuegos para proporcionar una experiencia de audio tridimensional, OpenAL también es útil en muchas otras aplicaciones que requieren capacidades avanzadas de procesamiento de audio.

Una de las características distintivas de OpenAL es que ofrece una abstracción de alto nivel del hardware de audio y los drivers. Esto permite a los desarrolladores centrarse en la creación de experiencias de audio envolvente sin preocuparse por las especificidades del hardware subyacente o los drivers de audio. Además, al ser una API multiplataforma, garantiza que el audio funcione de manera coherente en una variedad de sistemas y dispositivos.

Dentro de OpenAL, tres objetos principales trabajan en conjunto para proporcionar esta abstracción: el Dispositivo (Device), la Fuente (Source) y el Buffer.

El objeto Dispositivo es la representación de OpenAL del hardware de audio o del driver. Cuando una aplicación desea iniciar la reproducción de audio a través de OpenAL, primero debe abrir un dispositivo. Este acto de abrir un dispositivo es en realidad una solicitud para que OpenAL establezca una conexión con el hardware de audio o con el driver del sistema operativo. Una vez que esta conexión está establecida, la aplicación puede comenzar a enviarle datos de audio para su reproducción. Es importante notar que, en la mayoría de los casos, el Dispositivo actúa como un intermediario, tomando el audio que se le envía, procesándolo según sea necesario y luego enviándolo al hardware o driver adecuado para su reproducción.

La Fuente en OpenAL representa un punto de emisión de audio en un espacio tridimensional. Se puede imaginar como un altavoz virtual en un mundo virtual. Las fuentes tienen propiedades que determinan cómo se reproduce el sonido, como la posición, la velocidad y la dirección. Estas propiedades permiten a OpenAL proporcionar efectos como el audio 3D posicional y el efecto Doppler. Al asociar una fuente con un buffer (que contiene datos de audio), le decimos a OpenAL desde dónde queremos que se reproduzca ese audio y cómo queremos que suene en relación con el oyente.

El Buffer es donde se almacenan los datos de audio que se quieren reproducir. Se puede comparar como un contenedor para una muestra de audio. Una vez que tienes un buffer lleno de datos de audio, puedes asociarlo a una o más fuentes para su reproducción. Es importante destacar que los buffers son simplemente almacenamientos de datos y no tienen propiedades espaciales por sí mismos; es la fuente la que determina cómo se reproduce el audio contenido en el buffer.

En cuanto a cómo OpenAL conecta el *software* con el hardware y los drivers, el proceso es bastante transparente para el desarrollador. Cuando una aplicación envía datos de audio a OpenAL (generalmente a través de un buffer), OpenAL se encarga de procesar esos datos, aplicar cualquier efecto o transformación necesaria y luego enviarlos al dispositivo adecuado para su reproducción. Esto podría implicar la conversión de formatos de audio, la mezcla de

múltiples fuentes de audio en un solo flujo o la aplicación de efectos 3D. Una vez que el audio ha sido procesado, OpenAL se comunica con el hardware de audio o el driver del sistema operativo para garantizar que el audio se reproduzca correctamente.

Para el uso de OpenAL se puede acudir a la documentación en el PDF *OpenAL Programmer's Guide*[41] y con un estudio profundo de este documento se puede entender todo el funcionamiento de OpenAL.

2.6.3. Bibliotecas externas para lectura de archivos

Se utilizan diversas bibliotecas externas para la lectura de los archivos. Se hablará de 4 de ellas para distintos tipos de archivos. Assimp[40], dr_libs[42], json[43] y TinySoundFont[44]. Estas serán detalladas a continuación. Como el funcionamiento de la biblioteca Assimp ya se detalló anteriormente, en la sección Assimp no se explicara nuevamente.

2.6.3.1. DR_libs

La biblioteca dr_wav forma parte de la colección DR_Libs y está específicamente diseñada para facilitar operaciones de lectura y escritura de archivos en formato WAV en aplicaciones escritas en C++. Es una solución sin dependencias externas que ofrece un enfoque simplificado y eficiente para el manejo de archivos WAV.

Sus características incluyen que dr_wav es una biblioteca independiente, lo que significa que no requiere ninguna otra biblioteca o *software* externo para funcionar. Esta característica facilita su integración en diversos proyectos. También tiene un enfoque en la simplicidad, dr_wav permite a los desarrolladores leer y escribir archivos WAV con facilidad, sin tener que lidiar con complicaciones innecesarias. Además, La biblioteca es capaz de manejar archivos WAV con múltiples canales, lo que la hace adecuada para aplicaciones de audio profesional y de alta fidelidad.

Otras características incluyen que dr_wav es capaz de convertir automáticamente entre diferentes formatos de muestra, lo que facilita el trabajo con diferentes calidades y profundidades de bit de audio. Además de las operaciones de alto nivel para leer y escribir archivos completos, dr_wav proporciona funciones de bajo nivel para operaciones más detalladas y específicas.

dr_wav está diseñado para ser intuitivo y fácil de usar. Para leer un archivo WAV, simplemente se utiliza la función de apertura, y luego se pueden leer las muestras directamente. La escritura es igualmente sencilla, permitiendo a los desarrolladores especificar la configuración deseada y escribir muestras en el archivo.

2.6.3.2. JSON

La biblioteca *nlohmann/json* es una solución moderna y completa para el manejo de JSON en C++. Diseñada por Niels Lohmann, esta biblioteca se ha convertido en una herramienta esencial para muchos desarrolladores debido a su simplicidad, eficiencia y rica funcionalidad.

Una de las características destacadas de nlohmann/json es su sencillez. Con solo unas pocas líneas de código, los desarrolladores pueden parsear y serializar datos JSON con facilidad. Esta biblioteca permite tratar los datos JSON como si fueran objetos y arreglos nativos de

C++, eliminando gran parte de la complejidad asociada al manejo de JSON. Otra característica es que la biblioteca es altamente personalizable. Los desarrolladores pueden definir sus propias clases y estructuras y convertirlas fácilmente en formatos JSON y viceversa. A pesar de su simplicidad, esta biblioteca no compromete el rendimiento. Es eficiente tanto en términos de tiempo de ejecución como de memoria, lo que la hace adecuada para aplicaciones de alto rendimiento. Finalmente, está diseñada para ser compatible con una amplia variedad de compiladores y plataformas y además tiene una extensa documentación y una comunidad activa.

El uso de la biblioteca es muy simple, ya que, proporciona un tipo json que permite representar cualquier valor JSON en C++. Los valores pueden ser manipulados de manera similar a los contenedores nativos de C++, como std::vector o std::map.

2.6.3.3. TinySoundFont

TinySoundFont (TSF) y TinyMidiLoader (tml) son bibliotecas complementarias en C diseñadas para la síntesis de audio utilizando el formato SoundFont y para cargar archivos MIDI respectivamente. Ambas bibliotecas ofrecen un enfoque en la simplicidad y eficiencia, proporcionando a los desarrolladores herramientas poderosas y ligeras para integrar la reproducción de SoundFont y MIDI en sus aplicaciones sin la necesidad de dependencias externas.

Tanto TSF como tml han sido diseñados para ser lo más ligeros posible, facilitando su integración en aplicaciones donde el tamaño del código y el rendimiento son cruciales. Otra de las fortalezas de estas bibliotecas es que no requieren ninguna biblioteca o *software* externo para funcionar. Esto simplifica la integración y evita posibles conflictos o problemas de compatibilidad. TSF se centra en ofrecer una síntesis de alta calidad de archivos SoundFont (SF2). Los desarrolladores pueden cargar, procesar y reproducir archivos SoundFont con facilidad, aprovechando las capacidades de síntesis de la biblioteca. Mientras que TSF se encarga de la síntesis de SoundFont, tml proporciona las herramientas necesarias para cargar archivos MIDI. Esto permite a los desarrolladores trabajar con secuencias de notas y eventos MIDI y combinarlas con la síntesis de SoundFont para una reproducción completa.

La combinación de TSF y tml permite una integración fluida de la reproducción de MIDI y SoundFont. Para la inicialización y carga con TSF se puede inicializar una instancia de TSF y cargar un archivo SoundFont directamente con las funciones de la biblioteca. De forma similar, los archivos MIDI se pueden cargar usando tml. Luego se pueden leer los mensajes MIDI e interpretarlos con TSF o con otras funciones del *software* dependiendo del tipo de mensaje MIDI.

Capítulo 3

Problema

El presente trabajo de memoria de título se centra en la resolución de un problema crítico en el ámbito musical: la ausencia de una herramienta accesible para la orquestación musical y la experiencia de sonido en entornos virtuales. Hoy en día, para simular como suena un conjunto de fuentes de sonido en el espacio, se debe ir presencialmente al lugar en donde se desea hacer sonar el conjunto de instrumentos y modificar la salida del audio con mezcladores a partir de lo que se escucha en distintos lugares de este espacio. Virtualmente, se pueden realizar simulaciones, pero estas son hechas en programas o *software* con dificultades de uso por parte de músicos e incluso productores.

Este vacío en la industria musical se traduce en la carencia de *software* open source que logre integrar de manera eficaz y amigable la orquestación, el renderizado de modelos y la acústica del escenario, así como la reproducción y mezcla de archivos MIDI y audio en un entorno tridimensional. Esto produce que para los músicos, la única forma de simular una puesta en escena con sus composiciones es probando presencialmente o aprendiendo el uso de diversas tecnologías de alta dificultad.

La relevancia de abordar este problema radica en las múltiples aplicaciones y beneficios que una solución de este tipo podría ofrecer a músicos, compositores, directores de orquesta, profesores, ingenieros de sonido, sonidistas, arquitectos y entusiastas de la música. En la actualidad, la falta de herramientas accesibles y comprensibles limita la capacidad de los músicos para explorar y experimentar con arreglos instrumentales, la disposición en el escenario y la acústica en entornos virtuales. Esta carencia también obstaculiza la enseñanza y el aprendizaje de conceptos fundamentales relacionados con la orquestación y la acústica en el ámbito musical.

La solución a construir se basará en un *software* que combine una serie de características esenciales. En primer lugar, permitirá la representación visual y auditiva de instrumentos y escenarios mediante renderizado de modelos en tres dimensiones. Esto incluirá la capacidad de modelar teatros, asientos, instrumentos y parlantes de manera realista, ofreciendo a los usuarios una experiencia inmersiva y envolvente.

Además, la solución permitirá la división de archivos MIDI en sus pistas individuales, si se requiere, asignando a cada pista sonidos específicos provenientes de uno o varios archivos Soundfont. Asimismo, posibilitará la reproducción simultánea de múltiples archivos MIDI y audio, permitiendo la creación de arreglos complejos y ricos en capas sonoras.

En términos de características de calidad, la solución aspira a ser intuitiva y fácil de usar, eliminando la complejidad asociada con los *software* de música tradicionales. Los usuarios, independientemente de su nivel de experiencia en el ámbito musical o tecnológico, deberían poder aprovechar plenamente las funcionalidades del *software*. Además, se podrá visualizar la orquestación musical para crear más inmersión y en conjunto con el audio 3D logrará crear una simulación parecida a la realidad para así evitar la necesidad de aprender a utilizar diferentes *software* musicales para poder escuchar una puesta en escena para términos de conciertos o interpretaciones musicales.

Los criterios de aceptación de una solución al problema se basarán en la capacidad del *software* para proporcionar una experiencia auditiva y visual envolvente en entornos virtuales, facilitar la creación y reproducción de arreglos musicales complejos y brindar una interfaz amigable que reduzca las barreras de entrada para músicos de todos los niveles.

El problema que se aborda es la dificultad que se presenta al minuto de simular una interpretación musical o concierto para los músicos en términos del uso de tecnologías y la necesidad de combinar diversas de estas para conseguir un resultado realista. La solución propuesta aspira aportar con la simulación de estos diversos escenarios, proporcionando una plataforma que ayude a músicos y entusiastas de la música a mejorar la comprensión y apreciación de la orquestación y la acústica en el ámbito musical.

Capítulo 4

Solución

La solución para el problema consiste en un *software* hecho en C++ con CMake para compilación y ejecución. Este *software* es un programa en el que el usuario entrega una configuración simple en un archivo JSON y una vez ejecutado el programa el usuario se ve dentro de un auditorio, teatro o sala en el cual la configuración asignó instrumentos o parlantes en el escenario con un archivo de audio también especificado en la configuración. Los instrumentos y parlantes suenan al mismo tiempo y el usuario puede mover la cámara para ir escuchando el audio de forma tridimensional y las diferencias de audio con la posición del oyente. Si se requiere cambiar de configuración, el usuario puede cerrar el programa con la tecla *esc* y tras cambiar la configuración puede ejecutar el programa de nuevo y habrá cambiado la configuración. Los archivos de audio deben estar guardados en un directorio específico. Al *software* se le llamó Acousent que es un nombre proveniente de las palabras *acoustics* y *sensation*, ya que es un *software* de audio y visual que se enfoca en el sentido del oír para crear simulaciones inmersivas para el usuario y demuestra las capacidades del ser humano a poder orientarse con el oído.

4.1. Arquitectura del *software*

El diagrama 4.1 muestra la arquitectura del *software* de orquestación musical. Esta arquitectura muestra como los elementos del *software* interactúan entre sí para poder dar lugar al programa en sí. Las zonas más bajas representan el hardware, el sistema operativo y los drivers.

Luego viene la capa de las bibliotecas externas utilizadas para el funcionamiento correcto del programa. Estas bibliotecas son:

- GLFW: Consiste en una biblioteca externa que se encarga de crear una ventana de visualización de gráficos y de la interacción con dispositivos periféricos como el ratón y teclado.
- GLAD: Esta biblioteca se encarga de poder utilizar las funciones de OpenGL. OpenGL es una de muchas especificaciones que permiten controlar la GPU [45] y es parte íntegra de los drivers de las tarjetas de video modernas.
- OpenAL: Es una interfaz de software para hardware de audio. La interfaz consta de una serie de funciones que permiten a un programador especificar los objetos y operaciones

para producir una salida de audio de alta calidad, específicamente salida multicanal de arreglos 3D de fuentes de sonido alrededor de un oyente.

- Assimp: *Open Asset Import Library* es una biblioteca para cargar diversos formatos de archivos 3D en un formato inmediato compartido en memoria.
- dr_libs: En este caso se utilizó dr_wav la cual es una biblioteca para cargar y escribir archivos de audio tipo WAVE.
- TinySoundFont: Biblioteca que incluye TinySoundFont y TinyMidiLoader las cuales se encargan del análisis sintáctico de los archivos SoundFont y Midi respectivamente para crear sonido con la interpretación de estos archivos.
- glm: Biblioteca encargada del álgebra lineal mínima requerida por OpenGL.
- stb: Se utilizó stb_image para cargar imágenes para ser utilizadas como texturas.
- JSON: Biblioteca para el análisis sintáctico de archivos JSON.
- rtMidi: Conecta las señales MIDI de un controlador MIDI con el software, facilitando la interacción entre dispositivos MIDI y el programa.

La capa siguiente tiene los sistemas básicos para poder interactuar con el programa como usuario. Después se tienen los tres sistemas principales que contienen clases y archivos y se conectan entre ellos para hacer posible todas las funcionalidades del programa. A continuación viene el sistema de objetos del *software*. Este sistema imita lo que son *game objects* en un videojuego, es decir, son los elementos fundamentales que componen el entorno del *software*. Estos objetos representan cualquier tipo de entidad dentro del programa, como los personajes, escenarios, cámaras, luces, y otros elementos interactivos o no interactivos. Esta capa depende de los tres sistemas previamente mencionados asignados para poder tener una visualización, una fuente de audio y un archivo de audio asignado para así poder reproducir el sonido en su posición y que visualmente aparezca en esa posición. Finalmente, se tienen los ejecutables que consisten en los distintos demos de funcionalidades, un ejecutable para tener información de archivos para poder tener una configuración correcta y finalmente el programa *main* que es el ejecutable principal del programa.

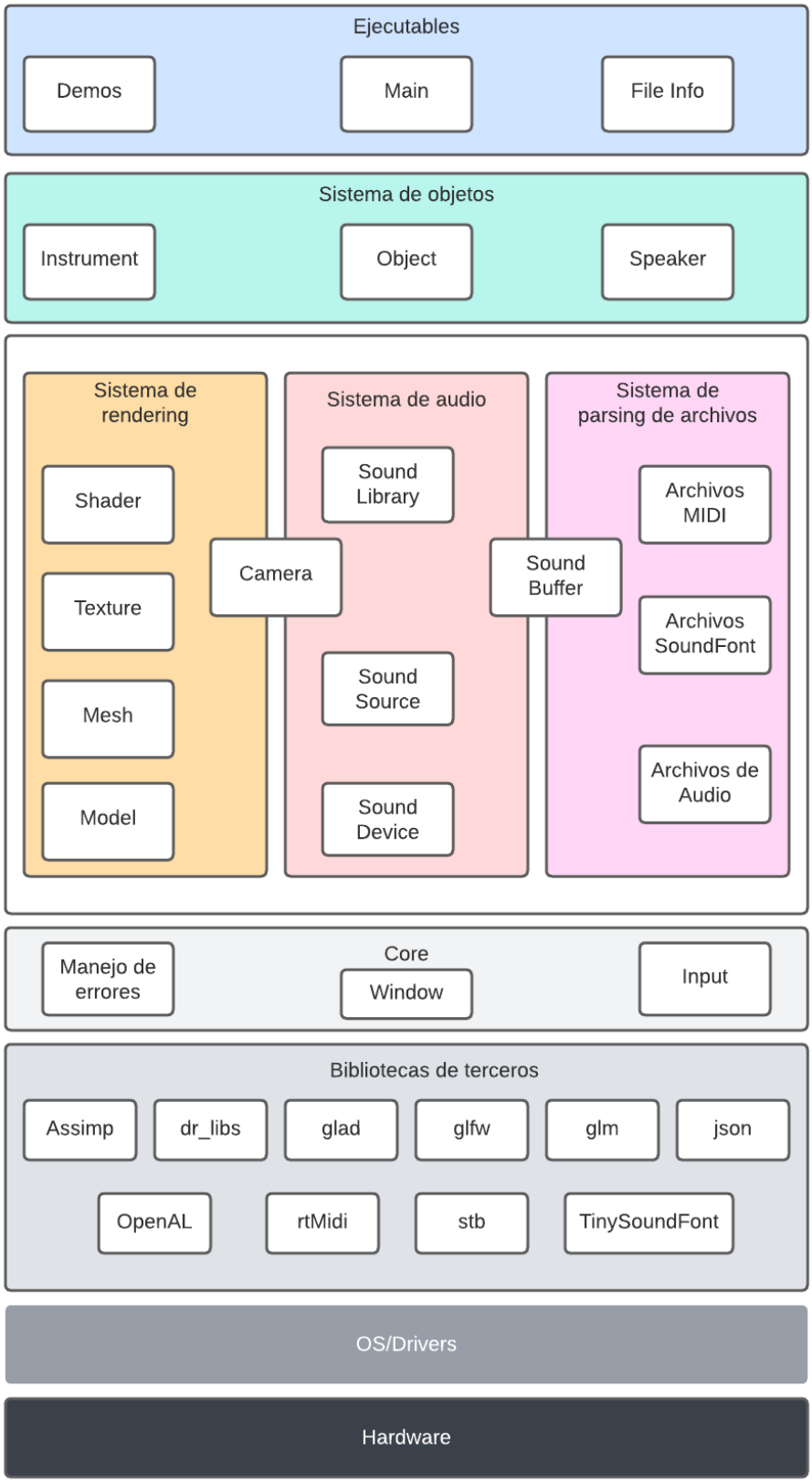

Figura 4.1: Diagrama de la arquitectura del *software*

.

4.2. Bibliotecas externas

Para la creación del *software* fueron necesarias diversas bibliotecas externas.

- Core:
  - GLFW: Manejo de input y de ventanas.
  - glm: Manejo y uso de funciones algebraicas.
  - rtMIDI: Manejo de input desde controlador Midi.
- Computación gráfica:
  - GLAD: El uso de funciones de OpenGL para la visualización 3D.
  - stb image: carga de imágenes para texturas.
  - OpenMeshCore: Crea mallas de matrices de vectores e índices.
  - Assimp: Crea modelos de mallas y les asigna la textura.
- Audio:
  - OpenAL: Manejo de dispositivos, fuentes y audio para poder reproducir audio a través del *software* y con salida hacia el hardware.
- Manejo de archivos:
  - dr libs: Análisis sintáctico de archivos de audio.
  - json: Análisis sintáctico de archivos json.
  - TinySoundFont (tsf): Análisis sintáctico de archivos MIDI y SoundFonts (sf2).

4.3. Diseño de Clases

Para el diseño de clases se siguió la arquitectura previamente mencionada y se explican brevemente a continuación:

4.3.1. Clases para el manejo de Audio

:

4.3.1.1. SoundDevice

La clase SoundDevice es responsable de conectar todo lo que se reproducirá como audio con el hardware del sistema en cuestión. Esta clase detecta el dispositivo de salida de audio predeterminado del hardware y guarda la dirección hacia la salida de audio para poder acceder a ella. Esta clase también guarda el contexto con el cual se esta trabajando y maneja todos los datos de posición del oyente. Actúa como oyente y como contexto general de audio.

4.3.1.2. SoundSource

La clase SoundSource maneja todo lo que implican las fuentes de audio virtuales. La fuente de audio interactúa directamente con el dispositivo o SoundDevice para poder hacer los cálculos de acústica necesarios para conseguir un audio espacial. Es esta clase como objeto la que se guarda en los instrumentos o parlantes para que la posición de estos sea la posición de la fuente de audio. También es en esta clase en donde se guarda la cola de audio que se reproducirá.

4.3.1.3. SoundLibrary

La clase SoundLibrary maneja todo lo que es creación e identificación de sonidos para poder ser reproducidos por las fuentes de audio.

4.3.2. Clases para el manejo de gráficas

4.3.2.1. Shader

La clase Shader maneja la lectura y manejo de los sombreadores. Esto es crear un sombreador o *shader* a partir del sombreador de vértices o *vertex shader* y sombreador de píxel o *fragment shader* y poder cambiar los valores algebraicos dentro de ellos.

4.3.2.2. Mesh, Vertex y Texture

La clase Mesh contiene la estructura Vertex y Texture y junto con el shader que se le entrega puede crear mallas geométricas a partir de matrices de vectores (Vextex) y asignarle vectores como normales para luego asignarles una textura o color para poder renderizarla en pantalla.

4.3.2.3. Model

La clase Model se encarga de hacer el análisis sintáctico de modelos 3D en diversos tipos de archivos y crea una malla a partir de estos. Guarda todas las mallas creadas y las junta en un modelo 3D.

4.3.2.4. ModelManager

La clase ModelManager se encarga de guardar los modelos 3D cargados en un diccionario con un nombre para poder acceder a ellos mientras se ejecuta el programa y no tener que cargarlos nuevamente.

4.3.3. Clases de interfaz de usuario

4.3.3.1. Camera

La clase Camera se encarga de ser los ojos del usuario. Guarda la posición, orientación y más vectores de posición del usuario para que interactúe con el audio y movimiento como lo haría una persona.

4.3.3.2. Window

La clase Window maneja la ventana del *software* por donde se verá todo el programa. También recibe las señales de input del usuario para hacer cambios con la ventana.

4.3.4. Clases de objetos en escenario

4.3.4.1. Intrument

La clase Instrument se encarga de guardar toda la información del instrumento como su modelo 3D, su fuente de audio y su cola de audio. También tiene la posición del instrumento en el escenario y desde esta posición suena el audio.

4.3.4.2. Speaker

La clase Speaker se encarga de guardar toda la información del parlante como su modelo 3D, su fuente de audio y su cola de audio. También tiene la posición del instrumento en el escenario y desde acá suena el audio.

4.4. Diseño de la Interfaz de Usuario

Para el diseño de la interfaz de usuario se consideró utilizar la biblioteca ImGui[3] docking para crear ventanas en donde el usuario pueda configurar los instrumentos y parlantes en el escenario y al mismo tiempo sea una interfaz fácil de usar y estéticamente agradable. Esto causo muchos problemas, ya que ImGui docking es una rama que aun esta en prueba para la biblioteca de ImGui. Por esta razón, en el programa principal se decidió no incluir ImGui o una interfaz de usuario por temas de que un programa con GUI[46] interactiva y complicada requiere de mucho tiempo y de un equipo más grande para dividir tareas.

A pesar de esto, el usuario tiene que poder interactuar con el programa de alguna forma, por lo que se decidió incluir un archivo de configuración en formato JSON para elegir la ubicación y los archivos de audio que tienen los instrumentos o parlantes. Con esto se le entrega personalización al usuario para que pueda probar distintas distribuciones de instrumentos y parlantes y pueda cambiar los instrumentos y el audio a reproducir.

Si bien la interfaz gráfica no se realizó, la solución principal no depende de una interfaz gráfica, ya que uno de los problemas con los *software* de audio actuales es la complejidad de uso y mientras más interacción pueda tener el usuario con el *software* mayor dificultad y para el objetivo de simular el audio tridimensional en un teatro, auditorio o sala no es necesaria la interfaz gráfica.

4.5. Procesos y flujo

A continuación se muestran diagramas de flujo para tanto el *software* en sí como para el flujo que el usuario realiza al utilizar este *software*.

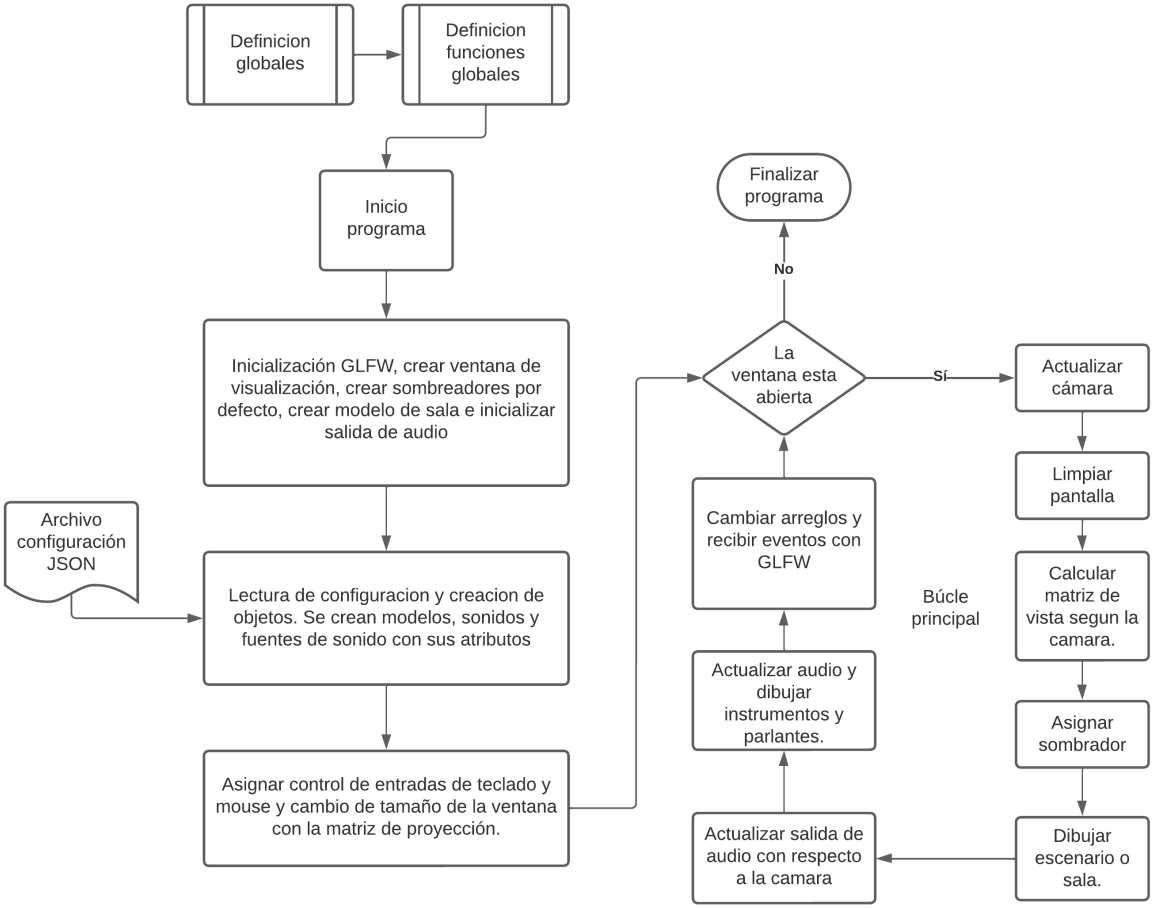

Figura 4.2: Diagrama de flujo del funcionamiento del *software*.

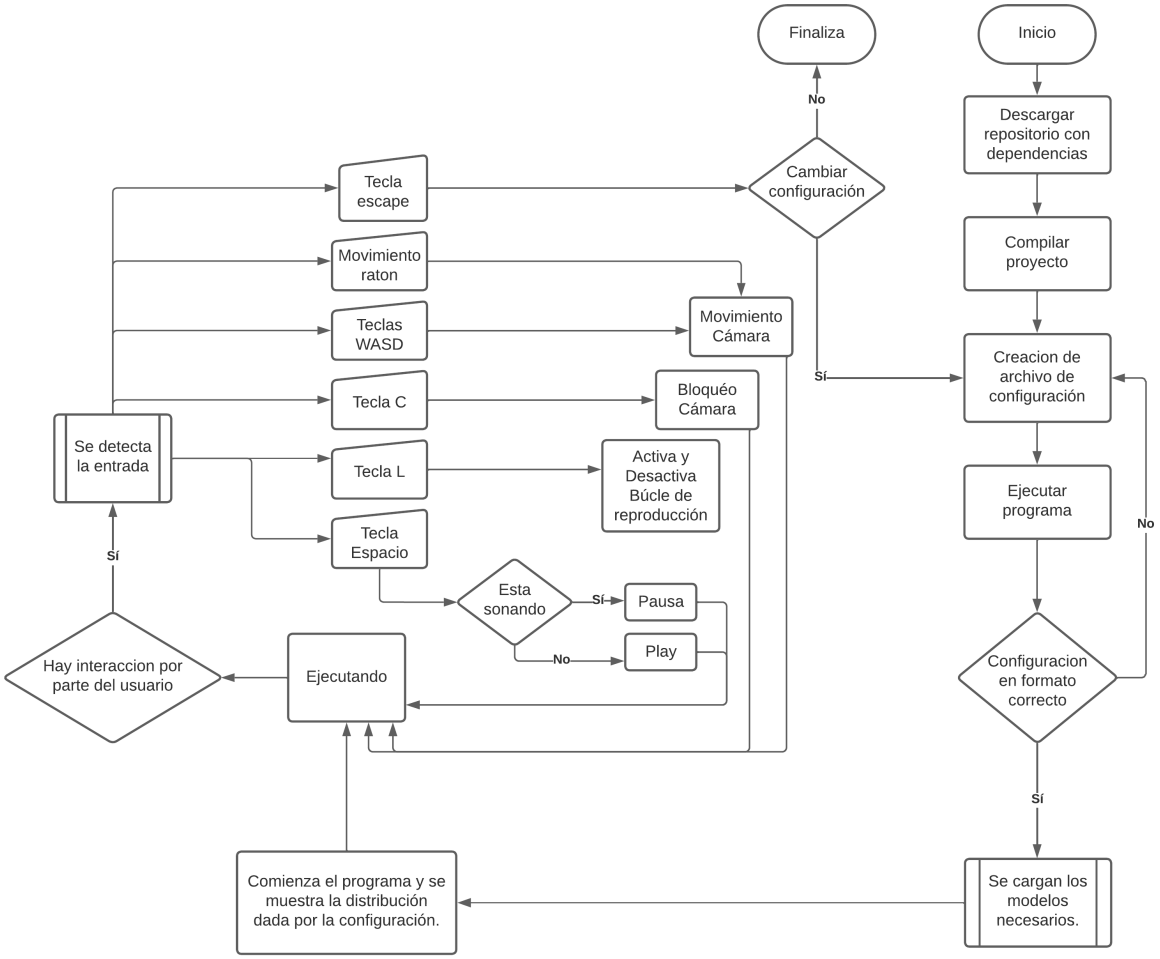

Figura 4.3: Diagrama de flujo del uso del *software* por parte del usuario.

Como se puede ver en la Figura 4.3, el usuario es responsable de crear la configuración del programa. Esto consta de un archivo JSON, el cual contiene dos llaves principales en cualquier orden. Una de estas son las fuentes de sonido que leen archivos de audio WAVE llamadas “audio_instruments” y cada elemento de esta llave representa un instrumento o parlante con su posición en el escenario, su nombre que representa el modelo 3D que utilizará y el archivo de audio que reproducirá. La otra llave representa las fuentes de sonido que leen archivos Midi y SoundFont llamadas “midi_instruments” y cada elemento de esta llave es otro instrumento con su posición en el escenario, su nombre que representa el modelo 3D que utilizará, sus archivos Midi y SoundFont, el número de la pista del archivo Midi que reproducirá (si no tiene reproducirá el archivo con todas sus pistas) y el índice del sonido específico o preestablecido que tiene el archivo Soundfont (si no tiene este valor utilizará el predeterminado por el archivo Midi).

4.6. Sistema de Rendering

4.6.1. Descripción General

El proceso de renderizado en la solución se basa en la representación de modelos 3D dentro de un espacio virtual, utilizando técnicas avanzadas de sombreado para simular una apariencia visual realista. La arquitectura del sistema se ha diseñado para ser modular y eficiente, aprovechando las capacidades de las modernas GPU y las bibliotecas de *software* especializadas.

El corazón de este sistema es la clase Model, que sirve como contenedor principal para los datos tridimensionales. Un modelo, en este contexto, es una representación digital de un objeto o entidad, compuesto por una serie de mallas. Cada malla, gestionada por la clase Mesh, es una colección de vértices, índices y texturas que definen una porción del objeto en 3D. Estas mallas se renderizan en pantalla a través de programas de sombreado, que son manejados por la clase Shader. Estos programas determinan cómo se presentan los píxeles individuales en la pantalla, permitiendo efectos como iluminación, sombreado y texturización.

Para cargar estos modelos desde archivos en diferentes formatos (como .glTF, .obj, .fbx, etc.), el *software* utiliza Assimp[40], una biblioteca de importación de activos. Esta herramienta convierte una variedad de formatos de archivo en una estructura de datos común que el sistema puede procesar y renderizar.

Una vez que los datos están cargados en memoria, OpenGL[39] entra en acción. Esta API de gráficos se encarga de la tarea de renderizado, convirtiendo las estructuras de datos en imágenes visuales en la pantalla. Todo el proceso, desde la carga hasta el renderizado, se ha optimizado para garantizar un rendimiento fluido y una alta calidad visual.

A continuación se explican las clases y bibliotecas en detalle.

4.6.2. Cámara

La clase Camera es esencial en cualquier sistema de renderizado, ya que define desde qué perspectiva se visualizará y renderizará una escena 3D. En este *software* también define donde esta el oyente para la reproducción de audio. Funciona como los ojos del usuario, determinando qué parte de la escena se ve, cómo se ve y en este caso como se escucha. Los detalles de la cámara para efectos de audio serán descritos en la sección de sistema de audio.

La cámara cuenta con diversos atributos que en conjunto determinan donde esta la cámara o los ojos del usuario, la orientación de la cámara, movimiento de la cámara, velocidades de movimiento y proyección.

La cámara tiene una posición en el espacio 3D y una orientación que determina hacia dónde está mirando. Estos se definen con los vectores *position, front, up, right y world up*[5] en combinación con los ángulos *yaw y pitch*[6] para poder realizar el movimiento. La clase Camera también proporciona métodos para mover la cámara en respuesta a la entrada del usuario,

[5] Vector posición, delante, arriba, derecha y mundo que representan la posición de la cámara, hacia donde mira, hacia arriba (90 grados con front), derecha (producto cruz negativo con up y front) y finalmente la normal hacia arriba del espacio 3D.

[6] Ángulos en el plano horizontal y vertical respectivamente.

esto lo hace con atributos *moving right, left, forward y backward*[7]. Esto se hizo para que el movimiento de la cámara pueda ser en más de un eje y si moving right y moving forward están encendidos, entonces la cámara se moverá diagonal hacia delante y derecha (noreste si se ve desde arriba y el norte es hacia delante de la cámara). La cámara no se mueve para arriba o abajo, ya que esto se hace con la combinación del movimiento y la rotación que responde a movimientos del ratón.

La proyección define cómo se mapea la escena 3D en una imagen 2D en la pantalla. Las cámaras suelen soportar proyecciones en perspectiva u ortográficas. En este caso, la proyección es en perspectiva y simula la forma en que el ojo humano ve el mundo, con objetos más grandes cuando están cerca y más pequeños cuando están lejos.

La cámara es fundamental para determinar cómo se verá la escena renderizada. Cuando se renderiza la escena, se utiliza la matriz de vista de la cámara, que se deriva de su posición y orientación, y la matriz de proyección, que se basa en el tipo de proyección elegido y en parámetros como el ángulo de visión, la relación de aspecto, y los planos cercano y lejano. La matriz resultante se pasa al programa de sombreado (a través de la clase Shader) para transformar los vértices de la escena desde el espacio del mundo al espacio de la cámara y, finalmente, al espacio de pantalla.

4.6.3. Shader

La clase Shader es una abstracción que representa un programa de sombreado o sombreador en el contexto de gráficos OpenGL. Un sombreador o *shader* es esencialmente una serie de instrucciones que se ejecutan en la GPU para manipular la apariencia visual de los objetos en una escena. Estos programas son escritos en un lenguaje especializado conocido como GLSL (OpenGL Shading Language).

La clase Shader se inicializa mediante el constructor, el cual acepta rutas a archivos que contienen el código fuente para el sombreador de vértices (vertex shader) y el sombreador de fragmentos o píxel (fragment shader). Estos archivos se leen, compilan y enlazan para formar un programa de sombreado único, identificado por su ID. Para este *software* se utilizaron *vertex y fragment shaders* simples por defecto y utilizan el modelo de iluminación de luz difusa, ya que la solución al problema no requiere tanta importancia a la gráfica o iluminación, por lo que solo se necesita cargar modelos. A pesar de esto, al *software* se le puede proporcionar shaders propios del usuario para ir mejorando las gráficas de este. Los archivos de los shaders están en el Apéndice A. Una vez que el programa de sombreado ha sido compilado y enlazado con éxito, puede ser activado para su uso mediante el método use(). Esto es necesario antes de dibujar objetos en la escena utilizando dicho shader.

Para que la clase shader sepa qué hacer con el código GLSL se requiere la configuración de uniformes. Los "uniformes" son variables globales en los shaders que pueden ser modificadas desde el código C++ para cambiar el comportamiento del sombreador. La clase Shader ofrece una serie de métodos para establecer el valor de estas variables uniformes, como setBool, setInt, setFloat, setVec2, setVec3, setVec4, setMat2, setMat3 y setMat4. Para las operacio-

[7] Movimiento a la derecha, izquierda, delante y atrás respectivamente.

nes de álgebra lineal se utilizó la biblioteca glm[47] y se logra que Estos métodos faciliten la interacción entre el código C++ y los shaders, permitiendo una gran flexibilidad en la manipulación de la apariencia visual.

Para buenas prácticas también se implementaron métodos destructores[8] y de gestión de errores. La función checkCompileErrors se encarga de verificar errores tanto en la compilación de los shaders individuales como en el enlace del programa completo. En caso de detectar un problema, se imprime un mensaje de error detallado. Por otro lado, para gestionar adecuadamente los recursos, la clase Shader proporciona un método *delete* que elimina el programa de sombreado del contexto OpenGL, liberando así los recursos asociados.

El uso de la clase Shader simplifica enormemente la tarea de trabajar con shaders en OpenGL. Al encapsular la lógica de lectura, compilación, enlace y gestión de uniformes en una clase, el desarrollador puede centrarse en la lógica del programa y en la creación de efectos visuales, sin tener que preocuparse por los detalles de bajo nivel de la API de OpenGL y le entrega al producto una mayor escalabilidad al *software*.

4.6.4. Mesh

La clase Mesh es una estructura fundamental en gráficos por computadora y representa un conjunto de vértices, índices y texturas que juntos forman un objeto tridimensional en una escena. Estos objetos pueden ser tan simples como un triángulo o tan complejos como un modelo de personaje. Mesh hace uso extensivo de OpenGL para configurar y renderizar mallas utilizando VAOs[9], VBOs[10] y EBOs[11].

La clase mesh contiene dos estructuras para poder tener las mallas completas y para facilitar la creación de estas. Estas estructuras podrían ser clases por sí solas, pero para la solución no es necesaria la separación de estas. Estas son las estructuras de Vertex y de Texture. La estructura Vertex representa un vértice específico de una malla o modelo. Cada vértice tiene información esencial como su posición (Position), normal (Normal) y coordenadas de textura (TexCoords). Esta información es crucial para renderizar la malla y para aplicar efectos como la iluminación y el mapeo de texturas. La estructura Texture Representa una textura que se puede aplicar a la malla. Contiene un ID, un tipo (como difuso o especular) y una ruta al archivo de textura.

La malla contiene una lista de vértices (vértices), una lista de índices (índices) que define qué vértices forman cada triángulo y una lista de texturas (textures) que se aplican a la malla. Además, la malla tiene un VAO (Vertex Array Object) que representa el objeto de malla en OpenGL.

El constructor de Mesh acepta listas de vértices, índices y texturas y se encarga de confi-

[8] Estos son necesarios por si se requiere dejar de utilizar un shader (cambiar el sombreado de los objetos) se pueda eliminar sin necesidad de cerrar el programa

[9] El Objeto de Arreglo de Vértices o VAO guarda referencias de configuraciones de atributos de vértices e índices, no datos directos. Requiere que los arreglos estén creados previamente para referenciarlos.

[10] El Objeto de Búfer de Vértices o VBO es una colección de bytes, almacena datos que la GPU interpreta según las instrucciones proporcionadas. Su vinculación afecta y modifica el estado del VAO relacionado.

[11] Los Objetos de Búfer de Elementos o EBOs, similares a los VBOs, se usan específicamente para índices en dibujos de elementos, indicando a la GPU de dónde sacar dichos índices.

gurar los buffers y arreglos necesarios para renderizar la malla en OpenGL. Por otra parte, la función privada setupMesh() se encarga de configurar el VAO, VBO (Vertex Buffer Object) y EBO (Element Buffer Object) para que la malla esté lista para ser renderizada. Para renderizar la función Draw se encarga de vincular las texturas adecuadas y renderizar la malla utilizando un programa de sombreado (representado por la clase Shader). Aquí, se activan las texturas correctas, se configura el programa de sombreado y se dibuja la malla.

La clase Mesh actúa como un contenedor para la información geométrica y de textura de un objeto y proporciona las herramientas necesarias para renderizar ese objeto en una escena. Al encapsular esta funcionalidad en una clase, se facilita la tarea de trabajar con múltiples objetos en una escena y se abstrae la complejidad de la configuración y renderizado en OpenGL.

4.6.5. Model

Finalmente, se complementan todas las clases de renderizado en la clase Model, la cual sirve como contenedor para una colección de mallas (clase Mesh) y proporciona las herramientas necesarias para cargar, procesar y renderizar un modelo 3D en una escena. Los datos del modelo consisten en los siguientes:

- textures_loaded: Una lista de texturas que ya han sido cargadas para evitar cargar texturas duplicadas.
- meshes: Una lista de mallas que componen el modelo.
- directory: La ruta del directorio donde se encuentra el modelo.

El constructor de la clase acepta una ruta a un archivo de modelo y utiliza la biblioteca Assimp para cargar el modelo y procesar sus mallas y texturas. Luego *loadModel* se encarga de cargar el modelo utilizando Assimp y procesa todos sus nodos y mallas. Dentro de esta función se ejecuta *processNode* la cual procesa un nodo del modelo y sus hijos de manera recursiva utilizando *processMesh*, la cual procesa una malla individual y recopila sus vértices, índices y texturas. Para las texturas se ejecuta *loadMaterialTextures* que carga las texturas asociadas con un material dado. Finalmente, para renderizar el modelo se llama a la función *Draw* la cual renderiza cada malla en la lista meshes, utilizando un programa de sombreado (clase Shader).

La clase Model es una abstracción de alto nivel que permite al desarrollador trabajar con modelos 3D complejos sin tener que preocuparse por los detalles de bajo nivel de la carga, procesamiento y renderizado. Al integrarse con las clases Shader y Mesh, la clase Model proporciona una representación completa de un objeto 3D que puede ser fácilmente renderizado en una escena OpenGL. Esto también facilita la creación de modelos para los instrumentos, parlantes y teatro o sala, ya que se pueden crear en programas como *Blender*[48] para crear los modelos y además da la posibilidad de descargar modelos (respetando las licencias) para su uso en el *software*.

4.7. Sistema de Audio

4.7.1. Descripción General

El sistema de audio de la aplicación es una mezcla sofisticada y altamente modular que combina la potencia y flexibilidad de la API de OpenAL con una serie de clases personalizadas diseñadas para facilitar la gestión y reproducción de audio en el entorno de la aplicación. Su arquitectura ha sido cuidadosamente diseñada para proporcionar una experiencia de audio inmersiva, manteniendo al mismo tiempo un alto grado de eficiencia y rendimiento.

En el corazón de este sistema se encuentra OpenAL, una API multiplataforma que permite la reproducción de audio en un espacio tridimensional. A través de OpenAL, la aplicación es capaz de simular fuentes de sonido posicionales, creando la ilusión de sonidos que provienen de direcciones específicas en un espacio virtual. Esto es especialmente crucial para aplicaciones interactivas, como lo es la solución entregada, donde la posición y movimiento del sonido pueden ser tan críticos como los gráficos en la creación de una experiencia envolvente.

Sin embargo, interactuar directamente con OpenAL puede ser complejo. Es aquí donde entra en juego la suite de clases personalizadas que actúan como capas de abstracción. La clase SoundDevice, por ejemplo, representa el hardware de sonido y facilita la comunicación entre la aplicación y el dispositivo físico o driver del sistema operativo. Por otro lado, la clase SoundSource actúa como un altavoz virtual, permitiendo al desarrollador especificar de dónde proviene un sonido, su dirección y otras propiedades.

Para gestionar los sonidos que se han cargado en la memoria, la aplicación utiliza la clase SoundLibrary. Esta clase actúa como un repositorio central, permitiendo a los desarrolladores cargar, acceder y descargar sonidos de forma eficiente. La capacidad de cargar diferentes tipos de archivos, desde simples clips de audio hasta archivos MIDI complejos, se gestiona a través de esta biblioteca.

El diseño modular del sistema permite una gran flexibilidad. Las clases pueden ser extendidas o modificadas según sea necesario, permitiendo la incorporación de características adicionales o adaptaciones específicas para hardware o plataformas particulares. Además, la estructura del sistema garantiza que los recursos, como la memoria y el procesamiento, se utilicen de manera eficiente, minimizando el impacto en el rendimiento general de la aplicación.

A continuación se explicarán todas las clases y herramientas que hacen posible el sistema de audio.

4.7.2. SoundDevice

La clase SoundDevice es una representación de alto nivel del dispositivo de sonido en una aplicación que utiliza la API de OpenAL para la gestión de audio. Esta clase es particularmente interesante porque sigue el patrón de diseño *Singleton*, lo que garantiza que solo exista una instancia de la clase en cualquier momento durante la ejecución del programa. Esta singularidad es esencial cuando se trata de gestionar recursos como el hardware de sonido, donde múltiples instancias podrían conducir a conflictos o comportamientos indeseados.

El encabezado del archivo revela que SoundDevice proporciona una serie de métodos tanto

estáticos como no estáticos. Los métodos estáticos, Get() e Init(), se utilizan para obtener la instancia única de la clase y para inicializarla, respectivamente. Una vez inicializado, el objeto SoundDevice actúa como una interfaz entre la aplicación y el hardware de sonido, permitiendo al programador obtener y establecer propiedades del "oyente", que es esencialmente una representación virtual del usuario en el espacio de audio 3D.

El oyente tiene una posición en el espacio, una orientación y un nivel de ganancia (volumen). Los métodos GetLocation, GetOrientation y GetGain permiten al programador obtener estos valores, mientras que los métodos SetLocation, SetOrientation y SetGain se utilizan para establecerlos. Estos métodos interactúan directamente con la API de OpenAL para obtener o establecer los valores correspondientes del oyente en el hardware o driver subyacente.

Además, hay un método interesante llamado SetAttunation, que permite al programador establecer el modelo de atenuación para el sonido. La atenuación se refiere a cómo disminuye la intensidad del sonido a medida que nos alejamos de la fuente. OpenAL proporciona varios modelos de atenuación, y este método permite al programador seleccionar uno de ellos pasando una clave específica.

Internamente, la clase SoundDevice mantiene dos punteros importantes: p_ALCDevice y p_ALCContext. Estos punteros son esenciales para la interacción con OpenAL. El primero representa el dispositivo de sonido, mientras que el segundo representa un contexto, que es esencialmente un entorno en el que OpenAL opera. Ambos son inicializados cuando se crea la instancia de SoundDevice.

El archivo de implementación contiene la lógica detrás de cada método declarado en el encabezado. Cada vez que se realiza una operación en OpenAL, como obtener o establecer una propiedad del oyente, se realiza una comprobación de errores usando AL_CheckAndThrow(). Esto asegura que cualquier problema o conflicto con el hardware o driver subyacente se detecte y maneje adecuadamente.

4.7.3. SoundSource

La clase SoundSource representa una fuente de sonido en la aplicación que en sí utiliza la API de OpenAL. Una fuente de sonido, en el contexto de audio 3D, puede imaginarse como un altavoz virtual en un espacio tridimensional. Esta fuente puede tener un conjunto de propiedades como posición, dirección y el buffer de sonido que está reproduciendo. La clase SoundSource encapsula todos estos detalles y proporciona métodos para gestionar y manipular estas propiedades.

La clase comienza con dos constructores: uno por defecto y otro que acepta un número y un vector de fuentes de sonido. El constructor por defecto se encarga de generar una nueva fuente de sonido utilizando la función alGenSources de OpenAL. El segundo constructor, aunque un tanto inusual, genera múltiples instancias de SoundSource y las añade al vector proporcionado. Este patrón es útil para pre generar un conjunto de fuentes que se utilizarán posteriormente en la aplicación.

A continuación, hay una serie de métodos para controlar la reproducción de la fuente. Estos incluyen Play, Stop, Pause y Resume. El método Play es particularmente interesante,

ya que verifica si el buffer que se le pasa es diferente del buffer actualmente asociado a la fuente. Si es así, lo asocia antes de iniciar la reproducción. Esto permite una fácil transición entre diferentes clips de sonido en caso de que sea necesario.

Además de los controles de reproducción, la clase SoundSource proporciona una serie de métodos setter para modificar las propiedades de la fuente. Estos incluyen SetBufferToPlay para establecer el buffer de sonido, SetLooping para determinar si el sonido debe repetirse, SetPosition para definir la posición de la fuente en el espacio 3D y SetDirection para establecer la dirección en la que la fuente está apuntando.

En cuanto a los métodos *getter*[12], tenemos GetBufferPlaying, que devuelve el buffer de sonido actualmente asociado a la fuente, y GetPosition, que devuelve la posición de la fuente en el espacio. Además, hay un método isPlaying que verifica si la fuente está actualmente reproduciendo un sonido y getSourceID que devuelve el identificador único de la fuente.

Internamente, la clase mantiene dos variables principales: p_Source y p_Buffer. p_Source es el identificador de la fuente de sonido en OpenAL, mientras que p_Buffer es el buffer de sonido que se está reproduciendo o que se va a reproducir.

4.7.4. SoundLibrary

La clase SoundLibrary actúa como un repositorio central para gestionar y mantener los sonidos que se han cargado en la memoria para su reproducción. Esta clase sigue el patrón de diseño Singleton, lo que garantiza una única instancia en cualquier momento durante la ejecución del programa. Tal singularidad es esencial en la gestión de sonidos, donde mantener múltiples bibliotecas podría conducir a redundancias y a un uso ineficiente de la memoria.

SoundLibrary proporciona una serie de métodos públicos para interactuar con la biblioteca. El método Load permite cargar un archivo de sonido en la biblioteca y devuelve un identificador único (de tipo ALuint) que puede ser usado posteriormente para hacer referencia a ese sonido. Hay también una función especializada LoadMIDI que se encarga de cargar archivos MIDI. Esta función tiene una serie de parámetros adicionales que permiten especificar detalles como si se debe cargar una única pista del archivo MIDI, el número de pistas a cargar, y el número de sonidos. El proceso exacto de cómo se realiza el análisis sintáctico de los archivos MIDI y Soundfont no se detallará aquí, y se abordará en profundidad en la sección del sistema de archivos.

Para eliminar un sonido de la memoria, la clase proporciona el método UnLoad, que acepta el identificador único del sonido y lo elimina de la biblioteca. Es importante notar que una vez que un sonido ha sido descargado, su identificador ya no será válido y no será utilizado nuevamente.

Internamente, SoundLibrary mantiene una lista de buffers de efectos de sonido, representada por el vector p_SoundEffectBuffers. Cada vez que se carga un nuevo sonido, se añade a esta lista, y cuando se descarga un sonido, se elimina de la misma.

[12] Métodos obtenedores, es decir, para obtener atributos de la clase.

Además, hay una constante sampleRate definida en la clase, que establece la tasa de muestreo a 44.1 kHz. Esta tasa de muestreo es típica para la mayoría de las aplicaciones de audio, ya que es la misma tasa utilizada en los CD de audio. Sin embargo, la clase permite la flexibilidad de cambiar esta tasa si es necesario, por ejemplo, para mejorar la calidad del audio o adaptarse a las especificaciones de hardware específicas.

4.8. Instrumentos y parlantes

4.8.1. Descripción General

El sistema de objetos del *software* se ha diseñado para proporcionar una estructura coherente y modular que permita la representación y manipulación de entidades en un espacio tridimensional. Este sistema es esencial para crear una experiencia auditiva envolvente, ya que permite simular la posición y movimiento de fuentes de sonido en un entorno virtual.

La base de este sistema es la clase Object. Esta clase representa una entidad genérica en el espacio 3D y encapsula una serie de características y funcionalidades esenciales. En primer lugar, un objeto tiene un modelo asociado, que define su apariencia visual en el entorno. Además, un objeto puede tener una fuente de sonido asociada, permitiendo que emita sonido en el espacio virtual. Estos sonidos pueden ser controlados mediante métodos que permiten reproducir, pausar o detener el audio. También se proporcionan métodos para manipular la posición, rotación y escala del objeto en el espacio 3D. La posición del objeto es particularmente importante cuando se asocia con una fuente de sonido, ya que determina el punto de origen del sonido en el espacio virtual.

Construido sobre esta base, se encuentran las clases Speaker e Instrument. Ambas clases heredan de Object y, por lo tanto, comparten todas las características mencionadas anteriormente. Sin embargo, introducen algunas especializaciones y diferencias clave.

La clase Speaker es una representación especializada de una fuente de sonido que puede emitir audio en el espacio virtual. Aunque comparte muchas similitudes con la clase Object, introduce un método específico llamado ProcessSpeakers. Esta función es esencial para procesar y actualizar la posición y orientación del altavoz en relación con otros objetos o el oyente en el espacio virtual.

Por otro lado, la clase Instrument representa un instrumento musical en el espacio 3D. Aunque un instrumento también es una fuente de sonido, su naturaleza y comportamiento son diferentes de un altavoz. Por ejemplo, esta clase introduce el método ProcessInstruments, que, al igual que su contra parte en Speaker, es esencial para procesar y actualizar el instrumento en relación con el espacio virtual utilizando archivos Midi y Soundfont para crear sonido.

4.8.2. Object

La clase Object sirve como base fundamental para la representación de las entidades del escenario como distintos instrumentos o altavoces en el espacio 3D dentro del *software*. Es una clase multifacética que encapsula tanto las propiedades visuales como las auditivas de una entidad. Por esta razón, en el caso de requerir más objetos como lo podrían ser los asientos del teatro para su interacción con el usuario, estos deben heredar de Object, ya que

representa objetos en la vida real y tiene las funciones necesarias para su visualización e interacción con otras entidades del *software*.

Desde el punto de vista visual, un objeto en el sistema está asociado a un modelo 3D, representado por la variable miembro object_model. Esta variable es una referencia a un objeto de tipo Model, que define la apariencia visual del objeto en el entorno. Para establecer y obtener este modelo, la clase proporciona los métodos setModel y Draw, respectivamente. Además, cada objeto tiene un shader asociado, representado por la variable object_shader. Los shaders son programas que determinan cómo se renderiza un objeto en la pantalla, y esta asociación permite que cada objeto tenga su propia representación visual única. Los métodos setShader y getShader se utilizan para establecer y obtener el shader asociado, respectivamente.

Desde el punto de vista auditivo, un objeto puede estar asociado a una fuente de sonido, representada por la variable object_Asource. Esta fuente de sonido permite que el objeto emita audio en el espacio virtual. Para controlar la reproducción del sonido, la clase proporciona métodos como Play, que inicia la reproducción del sonido, e isPlaying, que verifica si el sonido está siendo reproducido actualmente. Además, el método addAudioSource permite asociar una fuente de sonido al objeto.

Uno de los aspectos más críticos de la clase Object es su capacidad para manipular la posición y orientación de un objeto en el espacio 3D. La clase proporciona una serie de métodos para realizar transformaciones en el objeto, como Rotate, Translate y Scale[13]. Estas transformaciones se reflejan en la matriz transform_model, que determina la posición final, orientación y escala del objeto en el espacio virtual. Por ejemplo, la función Rotate permite girar el objeto alrededor de uno de sus ejes, mientras que Translate mueve el objeto a una nueva posición. La variable position mantiene la posición actual del objeto en el espacio 3D.

Además, la clase Object tiene la capacidad de mantener una lista de sonidos asociados, representada por la variable *sounds* o sonidos. Esta lista permite que un objeto tenga múltiples sonidos asociados que pueden ser reproducidos según sea necesario.

4.8.3. Diferencias entre Speaker e Instrument

En la arquitectura del sistema, tanto Speaker como Instrument son representaciones especializadas de entidades en un espacio 3D capaces de emitir sonido. Sin embargo, a pesar de sus similitudes estructurales, estas clases tienen diferencias fundamentales en términos de funcionalidad, representación y propósito.

4.8.3.1. Procesamiento de Audio

La diferencia más significativa entre estas dos clases radica en la forma en que procesan y emiten sonido:

La clase Speaker está diseñada para trabajar con archivos de audio directos. Estos archivos pueden estar en varios formatos, como WAV o MP3. Cuando se configura un Speaker, se carga un archivo de audio específico, y este archivo se reproduce directamente, emulando el

[13] Rotar, trasladar y escalar respectivamente.

sonido que emitiría un altavoz real en un entorno 3D.

A diferencia de Speaker, la clase Instrument no reproduce directamente archivos de audio. En su lugar, interpreta archivos MIDI, que son esencialmente secuencias de comandos que dictan qué notas deben tocarse, cuándo y cómo. Estos archivos MIDI no contienen sonidos en sí mismos. En su lugar, los sonidos se generan utilizando archivos SoundFont (sf2)[49]. Un archivo SoundFont contiene una colección de muestras de audio para diferentes instrumentos. Cuando se interpreta un archivo MIDI, cada nota o comando se traduce en un sonido específico utilizando las muestras del archivo SoundFont. Por lo tanto, el Instrument actúa como un intérprete virtual, transformando las instrucciones del archivo MIDI en música audible utilizando las muestras del SoundFont.

Estas diferencias en el procesamiento de audio también se reflejan en las funciones específicas de cada clase. Mientras que Speaker utiliza la función Load para cargar y procesar archivos de audio directamente, Instrument utiliza la función LoadMIDI, que integra la interpretación de archivos MIDI con la generación de sonido a partir de archivos SoundFont.

4.9. Sistema de Archivos

Una de las funciones más importantes del *software* es la capacidad de leer distintos tipos de archivo. Para estas lecturas y su correcto analizado sintáctico se utilizaron diversas bibliotecas de externos. Estas bibliotecas son: Assimp, dr_libs, nlohmann_json y TinySoundFont que incluye tsf (TinySoundFont) y tml (TinyMidiLoader). Estas bibliotecas son para la lectura de distintos archivos que incluyen los archivos glTF[50], OBJ[51], WAVE[52], json[53], MIDI[9] y SFZ[54] o SoundFont.

Como se menciona anteriormente, las distintas clases y el archivo main tienen las funciones necesarias para la lectura, análisis sintáctico y principalmente uso correcto de estos archivos.

4.9.1. Tipos de archivos utilizados

Como se menciona antes, la solución puede realizar la lectura, parseo y utilización de estos tipos de archivos. Los archivos y qué información tienen están explicados a continuación.

4.9.1.1. Archivos JSON

JSON, que significa "JavaScript Object Notation", es un formato ligero de intercambio de datos. Aunque originalmente surgió como un subconjunto de la programación de JavaScript, su fácil legibilidad y simplicidad lo han convertido en un estándar para la transferencia de datos en la web y más allá.

La notación JSON tiene sus raíces en la programación de JavaScript. Fue introducido por Douglas Crockford a principios de los años 2000 como una alternativa a XML, que hasta entonces era el principal formato de intercambio de datos. Crockford reconoció que, aunque XML era poderoso, a menudo era demasiado complejo para muchos de los casos de uso comunes en la web. JSON, con su estructura más sencilla y su similitud con el código literal de objetos en JavaScript, pronto se convirtió en una opción atractiva para los desarrolladores.

JSON es esencialmente una colección de pares clave-valor. Estas claves y valores pueden

ser casi cualquier cosa, desde números y cadenas hasta arreglos y otros objetos. En JSON, un objeto es una colección desordenada de pares clave-valor, rodeada por llaves . Por ejemplo: “nombre”: “Claudio”, “edad”: 25. Un arreglo es una colección ordenada de valores, rodeada por corchetes []. Por ejemplo: [“manzana”, “banana”, “cereza”]. Finalmente, los valores en JSON pueden ser cadenas, números, objetos, arreglos, verdadero, falso o nulo.

Las ventajas de JSON sobre otros formatos son varias. Primero, una de las grandes ventajas de JSON es su legibilidad. Tanto para máquinas como para humanos, es fácil de interpretar y escribir. Por otra parte, JSON tiende a ser más conciso que otros formatos de intercambio de datos, como XML. Otra ventaja es su versatilidad, ya que, a pesar de su simplicidad, JSON puede representar estructuras de datos complejas, lo que lo hace adecuado para una amplia variedad de aplicaciones. Finalmente, otra ventaja entre varias que tiene el formato JSON es que dada su amplia adopción, casi todos los lenguajes de programación modernos tienen bibliotecas que facilitan la codificación y decodificación de datos JSON.

4.9.1.2. Archivos glTF

Los archivos glTF, que significa “Graphics Language Transmission Format”[50], es a menudo referido como el “JPEG de la gráfica 3D”. Fue desarrollado por el grupo Khronos y está diseñado para ser un formato eficiente, inter operable y extensible para la entrega y carga de activos 3D.

Los archivos glTF tienen un formato basado en JSON, lo que significa que utiliza la estructura de notación de objetos de JavaScript para definir los detalles del modelo. Puede contener información sobre mallas, cámaras, materiales, animaciones y más. Además, permite la inclusión de binarios, imágenes y shaders, lo que lo hace completo para representar escenas 3D completas.

Una de las principales ventajas de glTF es su eficiencia en la transmisión. Está diseñado para minimizar tanto el tamaño de los archivos 3D como el tiempo de carga de los modelos, lo que lo hace ideal para aplicaciones web y móviles. Además, su estructura basada en JSON facilita su manipulación y lectura.

4.9.1.3. Archivos OBJ

OBJ[51], desarrollado por *Wavefront Technologies*, es uno de los formatos de archivo 3D más antiguos y ampliamente adoptados. Es conocido por su simplicidad y capacidad para representar geometría 3D.

El formato OBJ se centra principalmente en la representación de geometrías mediante vértices, normales y coordenadas de textura. Un archivo OBJ típico tendrá listas de vértices (puntos en el espacio 3D), normales (vectores que apuntan perpendicularmente a una superficie) y coordenadas de textura (que mapean imágenes 2D en la superficie 3D). Estos elementos se combinan para formar caras, que juntas construyen el modelo 3D.

La simplicidad del formato OBJ es tanto su fortaleza como su debilidad. Es fácil de leer y escribir, y muchos programas de diseño 3D lo soportan. Sin embargo, no soporta animaciones ni algunas características avanzadas presentes en formatos más modernos.

4.9.1.4. Archivos WAVE

WAVE, comúnmente conocido como WAV debido a su extensión de archivo, es un formato de archivo de audio desarrollado por Microsoft e IBM. Es uno de los formatos de audio más antiguos y ampliamente utilizados, especialmente en el mundo de la producción profesional de audio y música, debido a su calidad sin pérdida y simplicidad.

Un archivo WAV encapsula el audio en "trozos" o "chunks". El más importante de estos es el "chunk" de formato, que describe las propiedades fundamentales del audio, y el "chunk" de datos, que contiene el audio mismo. Una de las características definitorias del WAV es que es un formato sin pérdida. Esto significa que el audio se almacena en su forma original sin ninguna compresión que degrade la calidad. Como resultado, los archivos WAV tienden a ser grandes.

WAV utiliza modulación por código de pulso (PCM)[29] para codificar el audio. En PCM, la amplitud del sonido se muestrea regularmente a intervalos uniformes y luego se cuantifica a una serie de valores en una escala definida. Aunque el propósito principal de un archivo WAV es almacenar audio, también puede contener metadatos, como el nombre del artista, el título de la canción y otros detalles.

A continuación se hace una pequeña comparación entre WAV y otros formatos de audio:

- WAV vs. MP3[55]: Mientras que WAV es un formato sin pérdida, MP3 es un formato con pérdida. MP3 utiliza técnicas de compresión para reducir el tamaño del archivo, pero esto también resulta en una pérdida de calidad de audio. MP3 es preferido para la distribución debido a su tamaño de archivo reducido, mientras que WAV es a menudo la elección para la producción profesional debido a su calidad prístina.
- WAV vs. FLAC[56]: FLAC, al igual que WAV, es un formato sin pérdida. Sin embargo, a diferencia de WAV, FLAC utiliza la compresión sin pérdida, lo que significa que mientras el archivo es más pequeño que un WAV equivalente, aún puede ser descomprimido para recuperar el audio original sin ninguna pérdida de calidad.
- WAV vs. AAC[57]: AAC es otro formato con pérdida, similar en concepto a MP3 pero más moderno y generalmente con una mejor relación tamaño/calidad. Es el formato predeterminado utilizado por plataformas como iTunes[58].

Los archivos WAV se utilizan en una amplia variedad de aplicaciones, desde la edición y producción profesional de audio hasta su uso en sistemas operativos para sonidos de notificación. Su naturaleza sin pérdida los hace ideales para situaciones en las que la calidad del audio es de suma importancia, como en la masterización de música o en la postproducción de audio para video.

4.9.1.5. Archivos MIDI

MIDI[9], que significa Interfaz Digital de Instrumentos Musicales (por sus siglas en inglés "Musical Instrument Digital Interface"), es un protocolo de comunicación estándar utilizado para interconectar dispositivos musicales electrónicos y computadoras. Aunque se introdujo por primera vez en la década de 1980, MIDI sigue siendo una herramienta esencial en la producción musical y la actuación en vivo. Sin embargo, es fundamental comprender que

MIDI no es audio en sí; en cambio, se puede pensar en él como una serie de mensajes que indican cómo se debe generar o manipular el sonido.

Un archivo MIDI contiene una serie de mensajes que dictan cómo, cuándo y qué notas deben tocarse, entre otros comandos. Estos mensajes pueden incluir:

- Mensajes de Nota: Estos mensajes indican cuándo se debe comenzar a tocar una nota (Nota Encendida) y cuándo debe cesar (Nota Apagada). También especifican la "velocidad" con la que se toca una nota, que puede interpretarse como la intensidad o el volumen de la nota.
- Mensajes de Control: Estos mensajes pueden afectar diferentes parámetros del sonido, como el volumen, el balance, la resonancia, entre otros.
- Mensajes de Programa: Estos mensajes indican un cambio de instrumento. Por ejemplo, un mensaje de programa podría indicar un cambio de un sonido de piano a un sonido de flauta.
- Mensajes de Sistema: Estos incluyen comandos para sincronizar varios dispositivos MIDI y otros mensajes especiales.

MIDI admite hasta 16 canales diferentes en una única transmisión o archivo. Esto significa que puede enviar información para hasta 16 instrumentos diferentes simultáneamente. Por ejemplo, el canal 1 podría contener información para un piano, mientras que el canal 2 podría contener información para una guitarra.

Cuando un archivo MIDI se reproduce, no se genera audio directamente desde el archivo. En cambio, los mensajes MIDI se envían a un sintetizador o a un módulo de sonido, que luego produce el sonido real basándose en esos mensajes. Es el sintetizador o el módulo de sonido de un *software* el que realmente "interpreta" los mensajes MIDI y los convierte en música audible. El sonido exacto que produce el *software* o sintetizador depende de su propia configuración y las muestras de sonido que tiene almacenadas. Esto es lo que permite que los archivos MIDI sean tan versátiles: el mismo archivo MIDI podría sonar como un cuarteto de cuerdas en un sintetizador o *software* y como un conjunto de sintetizadores electrónicos en otro.

MIDI es una herramienta esencial en la composición y arreglos. Permite a los compositores y productores "escribir" música en *software* de estación de trabajo de audio digital (DAW) y escucharla en tiempo real. Además, los artistas pueden usar controladores MIDI, como teclados o *Pads*, para enviar sonidos en tiempo real durante una actuación. Otro uso común de MIDI es que los *software* de notación musical a menudo utiliza MIDI para reproducir partituras escritas. Finalmente, muchos videojuegos antiguos utilizaban música en formato MIDI debido a su pequeño tamaño de archivo y su capacidad para ser manipulado en tiempo real. Esto es porque un archivo de audio es mucho más pesado que un archivo MIDI. Por esto los Videojuegos antiguos tienen un timbre en común, llamado *8-bit* generalmente, ya que utilizaban archivos MIDI y un sonido simple para reproducirlo.

4.9.1.6. Archivos SoundFont

SoundFont es una marca registrada que se refiere a un formato de archivo que contiene muestras de audio utilizadas para la síntesis de música y efectos de sonido en computadoras

y sistemas de generación de sonido. Estos archivos, especialmente en sus variantes más conocidas, sf2 y sfz, actúan como bibliotecas de instrumentos y sonidos que pueden ser utilizadas por sintetizadores de *software* y hardware para recrear música y efectos de sonido.

Un archivo SoundFont encapsula una variedad de información que lo hace excepcionalmente versátil para la reproducción y modificación de sonido:

- Muestras: En el núcleo de un archivo SoundFont se encuentran las muestras de audio, que son grabaciones digitales de sonidos reales. Estas pueden ser desde una única nota tocada en un instrumento hasta complejas grabaciones ambientales.
- Instrumentos: Las muestras se agrupan y se asignan a instrumentos específicos. Un instrumento puede contener varias muestras, permitiendo, por ejemplo, que diferentes notas o diferentes velocidades de una tecla en un teclado produzcan diferentes sonidos.
- Presets o Preestablecido: Los instrumentos, a su vez, se agrupan en presets. Un preset puede ser pensado como un instrumento final que se presenta al usuario en un sintetizador. Un preset podría, por ejemplo, combinar muestras de varias cuerdas de una guitarra para crear el sonido completo de una guitarra.
- Moduladores y Generadores: Estos componentes definen cómo se deben alterar y procesar las muestras para producir sonido. Pueden afectar el tono, la envolvente, los filtros, y otros aspectos del sonido, permitiendo a los diseñadores de SoundFont modelar sonidos con gran detalle.

En este *software* se utiliza sf2 el cual es el formato clásico de SoundFont. Es un formato binario que encapsula todas las muestras, instrumentos, presets, y otros datos en un solo archivo. Fue popularizado en gran parte por las tarjetas de sonido *Creative Labs' Sound Blaster*, y ha sido un estándar en la síntesis de sonido desde entonces.

Existe otro tipo de SoundFont llamado sfz que es una variación más moderna y abierta del formato SoundFont. En lugar de ser un archivo binario monolítico como sf2, sfz utiliza una combinación de archivos de texto (que describen las muestras y cómo deben ser utilizadas) y las propias muestras de audio, que suelen estar en formatos estándar como WAV o FLAC. Esto lo hace más flexible y fácilmente editable, pero también puede resultar en conjuntos de archivos más grandes y dispersos.

Los archivos SoundFont son ampliamente utilizados en la producción musical, especialmente en la música MIDI. Al asignar diferentes instrumentos SoundFont a diferentes canales MIDI, los compositores y productores pueden generar piezas musicales ricas y detalladas sin necesidad de grabaciones en vivo. También son populares en videojuegos y aplicaciones multimedia, donde el espacio puede ser limitado y la síntesis en tiempo real es esencial.

4.9.2. Uso de los archivos y bibliotecas en el *software*

En el *software* de solución de este trabajo de título implementa un conjunto de funciones especializadas diseñadas para leer y procesar información de archivos de diferentes formatos. Estas funciones son cruciales para cargar configuraciones, modelos 3D, sonidos y otros recur-

sos en la aplicación. A continuación, se describe en detalle el propósito y funcionamiento de estas funciones.

4.9.2.1. Funciones para Leer Archivos JSON

Como se mencionó anteriormente, el formato JSON es una estructura de datos ampliamente utilizada para representar información estructurada en una forma legible por humanos. En el contexto de esta aplicación, los archivos JSON se utilizan para almacenar configuraciones y metadatos relacionados con los objetos y sonidos en la escena.

La función readJsonFromFile es responsable de leer un archivo JSON y devolver su contenido como un objeto JSON. Esta función comienza abriendo el archivo especificado. Si no se puede abrir, lanza una excepción para notificar el error. Una vez que el archivo está abierto, la función procede a parsear su contenido utilizando la biblioteca *nlohmann:json* y devuelve el objeto JSON resultante.

Además, hay otra función, getAllJSONKeys, que, como su nombre indica, recupera todas las claves presentes en un objeto JSON. Esta función es útil para explorar la estructura de un archivo JSON y determinar qué información contiene.

4.9.2.2. Funciones para Leer Archivos OBJ

Los archivos OBJ son uno de los muchos archivos que la biblioteca Assimp puede leer e interpretar. Las funciones ProcessInstruments y ProcessSpeakers ambas leen el nombre de la fuente de sonido y crean el modelo que tiene ese nombre. Esto lo hace con la función createModel, la cual se encarga de encontrar el modelo en archivo OBJ indicado por su nombre. Los modelos de fuentes de sonidos están en la carpeta Resources/Models/Instruments/*nombre de la fuente de sonido*/instruments.obj y createModel se encarga de que se cree el modelo asociado y se guarde en ModelManager para su uso en el programa. Si el modelo ya fue creado, no se crea nuevamente, ya que esto aumentaría el tiempo de ejecución al programa. Finalmente, si el modelo con el nombre asociado no existe, se le asigna un modelo por defecto que representa un sintetizador, ya que los sintetizadores tienden a poder reproducir distintos tipos de sonido.

Figura 4.4: Imagen sacada del *software* realizado para este trabajo de título conteniendo todos los modelos predeterminados como se puede ver en la distribucion2.json

.

4.9.2.3. Funciones para leer archivos WAVE

Como se mencionó en la sección Instrumentos y parlantes los archivos WAVE son para darle un audio buffer a los Speakers o parlantes. Para esto primero la función processSpeakers es responsable de procesar la configuración JSON relacionada con las fuentes con archivos de audio. Esta función examina el objeto JSON proporcionado en busca de configuraciones relacionadas con los parlantes. Si detecta configuraciones de parlantes, crea objetos Speaker correspondientes y configura sus propiedades según lo especificado en el JSON. Una vez que las fuentes de sonido están configuradas, la función utiliza el método Load para cargar y procesar los archivos de audio asociados. Este método, a su vez, utiliza la biblioteca dr_libs para el parseo de archivos de audio.

Dentro de la clase SoundLibrary, la función load desempeña un papel crucial en el procesamiento y manejo de archivos de audio. Su función principal es cargar y procesar archivos de audio, preparándolos para su posterior reproducción o manipulación.

Al inicio de la función, se declara una variable drwav que se utiliza para abrir y leer el archivo WAV especificado. La función utiliza la biblioteca dr_wav para este propósito, lo que simplifica significativamente la tarea de leer y procesar archivos en formato WAV.

Una vez que el archivo se ha abierto correctamente, la función procede a reservar memoria

para almacenar las muestras de audio que se leerán del archivo. Esta reserva de memoria es esencial para garantizar que haya suficiente espacio para almacenar todas las muestras de audio que se extraerán del archivo.

A continuación, la función lee las muestras de audio del archivo WAV y las almacena en la memoria reservada previamente. La cantidad de muestras leídas se determina por la cantidad de cuadros PCM (Pulse-Code Modulation)[29] que contiene el archivo. Es importante destacar que la lectura se realiza en formato de punto flotante, lo que permite una mayor precisión en la representación de las muestras de audio.

Una vez que todas las muestras han sido leídas y almacenadas, la función procede a cerrar el archivo WAV usando las herramientas proporcionadas por la biblioteca dr_wav. Es crucial cerrar el archivo para liberar recursos y garantizar que no haya bloqueos o problemas al acceder al archivo en el futuro.

Con el archivo cerrado y las muestras de audio almacenadas en memoria, la función finalmente crea y devuelve un objeto Sound que encapsula la información del audio cargado. Este objeto contiene las muestras de audio, así como metadatos sobre el audio, como la tasa de muestreo y el número de canales.

4.9.2.4. Funciones para leer archivos MIDI y SoundFont

Como también se mencionó en la sección Instrumentos y parlantes los archivos WAVE son para darle un audio buffer a los Speakers o parlantes. Para esto primero la función ProcessInstruments tiene un propósito similar a ProcessSpeakers, pero se centra en los instrumentos musicales. Esta función examina la configuración JSON en busca de definiciones de instrumentos. Si detecta tales definiciones, crea objetos Instrument y configura sus propiedades según lo especificado en el JSON. Lo más importante de esta función es su capacidad para procesar archivos MIDI en conjunción con archivos SoundFont (sf2). Para ello, la función utiliza el método LoadMIDI. Todas las funciones mencionadas a continuación se pueden encontrar en el Apéndice B.

Dentro de la clase SoundLibrary, la función loadMIDI es esencial para la manipulación y procesamiento de archivos MIDI. Está diseñada para cargar, interpretar y preparar estos archivos para su uso posterior en síntesis y reproducción.

Al comienzo de la función, se utiliza la biblioteca tinyMIDI Loader (tml) para cargar el archivo MIDI especificado. Esta biblioteca simplifica la tarea de interpretar la estructura y el contenido de un archivo MIDI, transformando este formato complejo en una secuencia de mensajes más manejable.

Una vez que se ha cargado el archivo MIDI, la función procede a iterar sobre cada mensaje MIDI en la secuencia. Estos mensajes pueden representar una variedad de eventos, desde notas que se deben tocar hasta cambios en el control y otros comandos de interpretación. Es esencial interpretar y procesar adecuadamente cada uno de estos mensajes para garantizar una reproducción precisa del archivo MIDI.

Mientras se procesan los mensajes MIDI, la función también utiliza la biblioteca Tiny-

SoundFont (TSF) para la síntesis de sonido. Esta biblioteca convierte los mensajes MIDI en sonido audible utilizando un archivo SoundFont, que proporciona las muestras de instrumento necesarias. Cada vez que se encuentra un mensaje de nota en la secuencia MIDI, loadMIDI utiliza TSF para generar el sonido correspondiente.

A medida que se procesan los mensajes y se genera el sonido, la función loadMIDI también mantiene un registro del tiempo. Esto es crucial para garantizar que los eventos MIDI se reproduzcan en el orden y tiempo correctos. Los archivos MIDI, después de todo, no solo indican qué notas se deben tocar, sino también cuándo y durante cuánto tiempo.

Dentro de la clase SoundLibrary, además de la función loadMIDI, existen otras dos funciones clave para el procesamiento de archivos MIDI: ProcessMIDIAndRender y ProcessTrackAndRender.

Mientras que la función loadMIDI se encarga de cargar y preparar el archivo MIDI para su interpretación, estas dos funciones adicionales se encargan de procesar y sintetizar el audio de diferentes maneras, tomando ventaja de las capacidades de la biblioteca TinySoundFont (TSF).

La función ProcessMIDIAndRender procesa el archivo MIDI en su totalidad, interpretando cada mensaje en la secuencia MIDI y utilizando TSF para generar el sonido correspondiente. Al abordar el archivo MIDI completo, esta función asegura que todos los eventos, instrumentos y matices del archivo original se reproduzcan con precisión. Los mensajes MIDI, que pueden representar desde notas hasta cambios de control y otros comandos de interpretación, se manejan en su orden y tiempo originales, lo que garantiza una reproducción fiel del archivo. Para utilizar esta función es crucial que el archivo MIDI y el archivo SoundFont coincidan con lo que el usuario quiere reproducir, ya que, esta función utilizara los presets del archivo sf2 que son dictados por el mensaje de programa de cada mensaje Midi del archivo MIDI.

A diferencia de ProcessMIDIAndRender, esta función se centra en un enfoque más específico y detallado. En lugar de procesar todo el archivo MIDI, ProcessTrackAndRender se centra en una sola pista o track del archivo. Además, en lugar de utilizar todos los presets disponibles en el archivo SoundFont, esta función utiliza un único preset específico para sintetizar el audio. Esta capacidad de centrarse en una pista y preset específicos permite una mayor flexibilidad y precisión en la reproducción, lo que es especialmente útil en aplicaciones donde se desea aislar o destacar un instrumento o parte específica de la composición. Además, es crucial cuando se tiene un archivo MIDI, pero se quiere ir cambiando el instrumento o sonido que se le asigna a las distintas partes de la composición MIDI.

Después de procesar todos los mensajes en la secuencia MIDI y generar el audio correspondiente, la función finalmente devuelve un objeto Sound. Este objeto encapsula el audio generado, junto con metadatos adicionales, como la tasa de muestreo y el número de canales.

4.10. Funcionamiento del *software*

El código fuente del *software* desarrollado en este trabajo de título fue subido a un repositorio en GitHub[14] bajo la licencia MIT[59]. Dentro del repositorio creado, la estructura de las principales carpetas es la siguiente:

- **Resources:** esta carpeta contiene todos los recursos utilizados para implementar los ejemplos. Esta dividido las sub carpetas:
  - Audio: Contiene los archivos de audio, Midi y SoundFont utilizados en el programa (y aquí se agregarían más archivos WAV, Midi o SoundFont para ser utilizados con el programa).
  - Config: Aquí se encuentran 2 archivos json de configuración demo para utilizar de ejemplo. En esta carpeta debe ir la configuración final llamada distribution.json.
  - Font: Aquí se encuentran archivos de tipografías para ImGui, ya que un ejemplo utiliza ImGui y en el futuro si se avanza hacia un *software* con UI de ImGui se necesitarán tipografías.
  - Models: Aquí están los modelos 3D que se utilizan. La sub-carpeta Instruments contiene los modelos de las fuentes de sonido en formato OBJ. Si se requieren más modelos se pueden crear modelos en formato OBJ y ponerlos en la carpeta Instruments/*nombre del modelo* y al seleccionarlo en la configuración se utilizará este modelo para la fuente de sonido deseada. Es esperable que los nuevos modelos agregados tengan una escala parecida a los demás modelos que vienen por defecto.
  - Shaders: Aquí están los programas GLSL de shaders que se utilizan en el programa. Se utilizan los shaders default.vs y default.fs, pero se podría administrar otros shaders si es necesario.
- **Examples:** esta carpeta contiene el código fuentes de los demos de funcionalidades del *software*. Estos demos no son parte del programa, pero sirven como ejemplos por si se quiere editar la aplicación. También aquí se encuentra el archivo files_info.cpp que se utiliza para el programa que tras leer un archivo Midi y uno SoundFont entrega la información pertinente para decidir como hacer la configuración.
- **source:** esta es la carpeta más importante, ya que es la que contiene el código fuente del *software*. Dentro de esta carpeta existen otras, una por cada sistema del *software*. Aquí también esta el archivo main.cpp que es el que crea el ejecutable de la aplicación.
- **third_party:** esta carpeta contiene las bibliotecas externas creadas por terceros usadas para la implementación del *software*. El principal criterio para escoger estas bibliotecas fue que estas fuesen de código abierto, multiplataforma y sin más dependencias externas.
- **Memoria:** esta carpeta contiene tanto el presente documento como un archivo comprimido con el proyecto de LaTeX a partir del cual se generó el presente documento.

[14] https://github.com/HanPollo/Acousent

4.10.1. Compilación

Para controlar el proceso de compilación se utilizó la herramienta CMake[15], la cual mediante el uso de archivos de configuración multiplataforma permite generar proyectos de Visual Studio en Windows. Para hacer uso de los archivos de configuración de este repositorio es necesario tener instalada la versión 3.5 o superior.

Para la compilación en Windows primero se debe clonar el repositorio GitHub a la carpeta deseada. Para realizar esto se debe utilizar un terminal que funcione con el comando git e ir a la carpeta o directorio donde se quiere guardar el *software*. Generalmente, para esto se utiliza el terminal de git bash que funciona perfecto para este caso. En el terminal se debe insertar el comando siguiente:

git clone –recursive https://github.com/HanPollo/Acousent.git

Luego se deberían descargar todos los archivos del repositorio, incluyendo las dependencias. Si no se descargaron las dependencias se debe probar con el comando:

git submodule update –init –recursive

Para asegurar que se descargaron las dependencias.

Luego para utilizar CMake se debe ir a algún terminal que pueda utilizar el comando cmake e ir al directorio Acousent que fue recién descargado. En este directorio se debe seguir los siguientes comandos.

mkdir build
cd build
cmake ..
cmake –build .
./source/Debug/main.exe

Alternativamente, se puede utilizar Visual Studio para la compilación y basta con realizar cmake .. y luego se creara la solución en archivo .sln en la carpeta build dentro de la carpeta Acousent.

El ejecutable se creará en Acousent/build/source/Debug/main.exe.

[15] https://cmake.org/

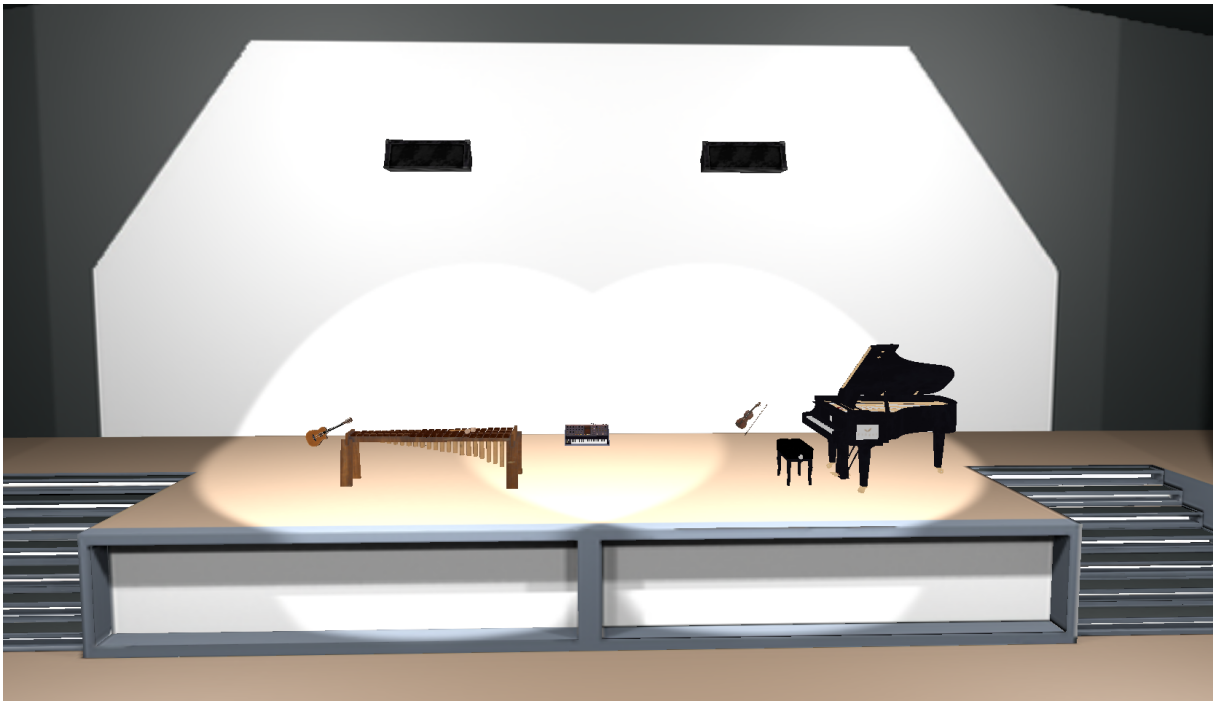

Figura 4.5: Imagen sacada del *software* realizado para este trabajo de título cuando se ejecuta el programa

.

4.10.2. Agregando modelos 3D

Se puede personalizar la experiencia entregando modelos 3D propios. Hay 2 tipos de modelos que se pueden utilizar. Primero son modelos para las fuentes de sonido o instrumentos y también se puede proporcionar con un modelo nuevo para la sala o escenario.

4.10.2.1. Importar modelos para la sala

Si se requiere un nuevo modelo 3D para la sala, este tiene que estar en formato glTF y debe llamarse scene.gltf. Debe estar en la carpeta Resources/Models/Auditorium/ y es esperable que sea de un tamaño parecido al proporcionado previamente en esta misma carpeta.

4.10.2.2. Importar modelos para fuentes de sonido

Si se requieren más modelos para los instrumentos o fuentes de sonido, estos deben estar en formato OBJ, llamarse instruments.obj y deben estar posicionados en la carpeta Resources/Models/Instruments/*nombre del modelo*/. Nuevamente, deben estar en una escala parecida a los proporcionados. Luego se pueden elegir estos modelos nuevos dentro de la configuración.

4.10.3. Utilizando el Programa

Para utilizar el programa es fundamental el archivo distribution.json ubicado en Acousent/Resources/Config/distribution.json. Se incluye un archivo de demostración y un archivo distribution2.json como otro ejemplo. En el distribution.json se encuentran listadas 2 fuentes de audio y 5 fuentes Midi en posiciones específicas. Se puede cambiar esta distribución como requiera el usuario sea la posición, el sonido que se desea reproducir y el nombre del modelo

a utilizar. Los archivos de distribución de demo y de ejemplo están en el Apéndice C.

Se puede cambiar esta distribución antes de ejecutar el programa main para cambiar la configuración de los instrumentos en escena y en sí cambiar como se ve y se escucha el programa. Para los archivos de audio, estos tienen que estar en las ubicaciones correctas. Si es un archivo tipo WAVE o .wav debe estar en la carpeta Resources/Audio/Wav. Si es un archivo MIDI o .mid debe estar en la carpeta Resources/Audio/MIDI. Finalmente, si es un archivo SoundFont o .sf2 debe estar en la carpeta Resources/Audio/SoundFonts. Solo con estas ubicaciones funcionarán los nombres de los archivos en la configuración o distribución.

Una vez que se tiene la configuración deseada se debe ejecutar el programa main como se mencionó anteriormente. Esto ejecutará el programa y en la consola se mostrarán los parlantes e instrumentos con sus archivos de audio correspondientes y su posición. Luego en la ventana principal se mostrara la sala con los instrumentos en el escenario. El usuario puede mover la cámara y el oyente utilizando las teclas WASD del teclado.

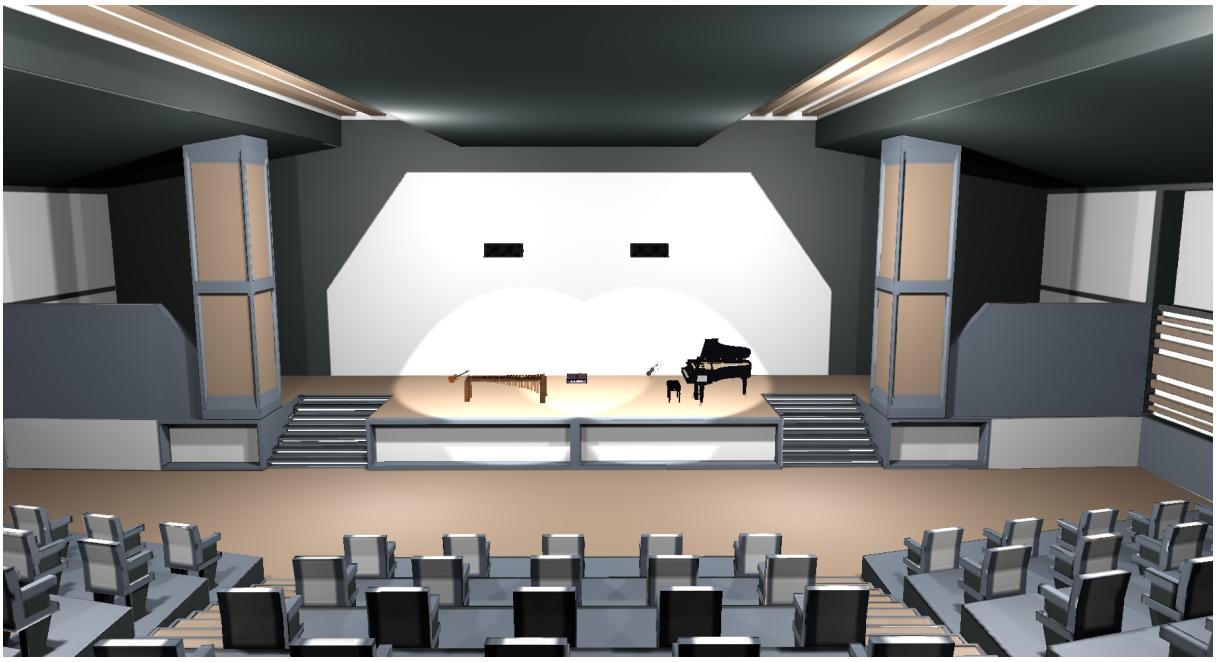

Figura 4.6: Imagen sacada del *software* realizado para este trabajo de título cuando se ejecuta el programa y la cámara se mueve un poco hacia atrás

.

Se puede rotar la cámara con el movimiento del ratón y se puede bloquear la cámara con la tecla ‘C’ del teclado. Para reproducir el sonido se debe apretar la tecla de espacio. Si se aprieta nuevamente la tecla de espacio se pausará el audio. Si se quiere activar el bucle o loop del sonido se puede apretar la tecla ‘L’ del teclado. Con la tecla de escape se cierra el programa.

Figura 4.7: Imagen sacada del *software* realizado para este trabajo de título cuando se ejecuta el programa y la cámara se rota un poco hacia la izquierda

.

Como se puede ver en la Figura 4.6 y la Figura 4.7 la rotación es muy importante para el uso del programa, ya que, con la rotación y el movimiento de la cámara, se puede llegar a todos los puntos de la sala.

Figura 4.8: Imagen sacada del *software* realizado para este trabajo de título cuando se ejecuta el programa y la cámara se rota un poco hacia la derecha

.

La mejor forma de utilizar el *software* es con un archivo Midi dividido en varias pistas que se le asignan a distintos instrumentos del ensamble en el *software*. Si se tiene un archivo Midi con un Soundfont que no se sabe que pistas o presets se recomienda utilizar, se puede ejecutar el demo files_info.exe en la carpeta Examples y poner el nombre del archivo MIDI y SoundFont correspondientes. La información aparecerá en pantalla como los nombres de los preset del Soundfont y la cantidad de pistas o tracks, las pistas que tienen notas y su preset por defecto del SoundFont seleccionado para cada pista. Esto se puede probar con los archivos demo de Midi y SoundFont que aparecen en la configuración demo.

El archivo MIDI demo llamado venture.mid fue sacado de los ejemplos de la biblioteca TinySoundFont y la información es la siguiente: Venture (Original WIP) by Ximon https://musescore.com/user/2391686/scores/841451 License: Creative Commons copyright waiver (CC0). El archivo SoundFont también fue sacado de los ejemplos de la biblioteca TinySoundFont, pero no tiene licencia o link. Los otros archivos en los recursos de audio fueron creados por el desarrollador y autor de este trabajo de título.

Capítulo 5

Validación

La validación de este trabajo de título se realizó con la entrega del *software* a posible usuarios finales y realizar una entrevista presencial con el usuario utilizando el *software* y con ayudas personales de compilación y uso del desarrollador en caso de ser necesarias.

5.1. Selección y Justificación de Usuarios de Prueba

Como se detalla en el capítulo 4 la solución consiste en el desarrollo de un *software* de orquestación musical, el cual permite la distribución de los instrumentos en el espacio 3D de una sala y la reproducción de audio de estos instrumentos junto con la interacción de un usuario oyente controlable para escuchar y analizar las diferencias de audio entre distribuciones distintas.

Por el diseño de la solución, la mejor forma de validar los resultados era proporcionando el *software* a usuarios de prueba y realizar una encuesta a estos tras el uso del *software*. Los criterios de selección del muestreo son diversos, ya que la solución abarca usuarios de distintos ámbitos en la tecnología e industria musical.

Primero se debe contar con la prueba realizada por un músico profesional para que compare el uso de la solución con otros *software* de música que ha utilizado antes. Otro posible usuario es un estudiante de música, ya que a diferencia de un músico profesional esta aprendiendo sobre las tecnologías en el área musical y por esta razón esta familiarizado con el uso de estas tecnologías, pero al mismo tiempo ha tenido que probar alternativas para poder comparar las distintas funcionalidades que proporcionan los distintos *software* musicales. Otro posible usuario tiene que estar enfocado en el rubro de la arquitectura, ya que, como en la música, los *software* utilizados en la industria de la arquitectura y construcción tienen a ser complejos. La solución permitiría ir acercándose a lo que sería un *software* de renderizado de salas, teatros, auditorios y más para su posible construcción con una acústica específica para esos tipos de salas. Finalmente, la solución busca ser una herramienta esencial para músicos aficionados sin experiencia con tecnología o *software* musicales para que puedan probar sus configuraciones deseadas en un entorno principiante, como un *show de talentos* o tocatas musicales en vivo para pocas personas.

Con esto en mente se seleccionaron 4 usuarios finales de prueba para la solución:

- Eric Mayne-Nicholls: músico profesional y estudiante de Berklee College of Music[60] en

Boston, MA, EE.UU. de nacionalidad chilena. Eric es el artista detrás de Mother Sun & The Rainbow Band y si bien esta comenzando sus estudios en el ámbito de la música, ha tenido experiencia como productor de música y su música se encuentra disponible en Spotify[61] y ya le ha dado remuneración positiva.

- Benjamín Zamorano: Estudiante de Berklee College of Music[60] en Boston, MA, EE.UU. con un enfoque en la producción de *film scoring*[62] que se enfoca en composición y producción musical para películas y videojuegos. En esta área de la música la solución proporciona avances enormes, ya que la inmersión de audio es fundamental en estos productos.

- Antonio Duval: Arquitecto profesional, cofundador de la empresa CentralCorp[63] y de Duvify[64] por lo que aparte de ser un arquitecto, esta constantemente en busca de nuevas tecnologías para utilizar con su estudio de arquitectura y emprendimiento tecnológico de la renta.

- Matías Greenhill: músico aficionado y odontólogo profesional, sin conocimientos profundos de *software* ni tecnologías asociadas a estos. Además, no tiene conocimientos de *software* musicales, pero tiene un gran talento y enfoque en la música como hobby.

5.2. Preguntas hechas a los usuarios finales

La solución se le entregó presencialmente para poder ayudar con temas de compilación y otros usos cuando el usuario no esta familiarizado con herramientas como Visual Studio[6] o CMake[65] y así puedan utilizar el *software* sin problemas de compilación. Luego se les dejó a solas con el programa con 3 distribuciones distintas y la posibilidad de que editen las distribuciones a su manera. Luego de aproximadamente 30 minutos de uso se les entrevistó realizando principalmente preguntas con enfoque en 4 áreas. Estas áreas son usabilidad y diseño, Características de Audio 3D e Inmersión, Aplicabilidad en la industria y sugerencias, mejoras y preguntas generales. Las preguntas fueron las siguientes:

- Usabilidad y Diseño:
  - ¿Cuál fue tu primera impresión al abrir el programa?
  - ¿Fue fácil encontrar y usar todas las funciones que necesitabas?
  - ¿Hubo alguna función o herramienta que te pareció innecesaria o redundante?
  - ¿Experimentaste algún problema o dificultad durante el uso del programa?

- Características de Audio 3D e Inmersión:
  - ¿Sentiste que la experiencia de audio 3D fue verdaderamente inmersiva?
  - ¿Cómo compararías la calidad y experiencia de este audio 3D con otros programas o sistemas que hayas utilizado anteriormente?
  - ¿En qué tipo de entornos o situaciones consideras que el audio 3D de este *software* brilla particularmente?

- Aplicabilidad en la Industria:

- ¿Crees que este programa podría revolucionar la forma en que trabajas trabajan actualmente? ¿Por qué?
- ¿Qué ventajas ves en este *software* en comparación con otras herramientas que hayas utilizado anteriormente?
- ¿En qué áreas específicas (música, arquitectura, diseño de sonido, etc.) ves un mayor potencial para este *software*?

• Sugerencias, mejoras y preguntas generales:

- ¿Hay características específicas que te gustaría que se incorporaran en futuras versiones del *software*?
- ¿Qué consejo le darías a alguien que esté considerando usar este programa por primera vez?
- ¿Consideras que el *software* satisface tus necesidades profesionales o creativas?
- Si pudieras cambiar una sola cosa del programa, ¿cuál sería?
- ¿Ves potencial en este programa para transformar la forma en que las personas experimentan y crean música u otras artes?

Las preguntas se realizaron en un ambiente seguro, sin manipulación del entrevistador y sin forzar respuestas positivas.

5.3. Resultados

Los resultados de las encuestas o entrevistas se dan a continuación.

5.3.1. Entrevista a Eric Mayne-Nicholls

Eric utilizó el *software* de solución con ayuda del memorista para la compilación. Tras el uso del *software* se le entrevistó y dio las siguientes respuestas a las preguntas mencionadas anteriormente:

• Usabilidad y Diseño:

- ¿Cuál fue tu primera impresión al abrir el programa?

  Respuesta: "*Me pareció muy real la sala y los instrumentos y altavoces en el escenario y al apretar el espacio pude escuchar la reproducción de los audios como si estuviera realmente en la sala a través de mis audífonos.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA).
- ¿Fue fácil encontrar y usar todas las funciones que necesitabas?

  Respuesta: "*Las funciones de movimiento, loop, play y pausa del audio son intuitivas, lo que sí es que la forma de configurar el archivo de configuración es un poco complicada, ya que tuve errores por no saber escribir bien el archivo de distribución*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- ¿Hubo alguna función o herramienta que te pareció innecesaria o redundante?

  Respuesta: "*No había ninguna funcionalidad innecesaria*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- ¿Experimentaste algún problema o dificultad durante el uso del programa?

  Respuesta: "*Como dije antes me costó un poco entender el archivo de distribución, pero una vez que lo entendí pude hacer cambios en la distribución y en los audios a reproducir*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

• Características de Audio 3D e Inmersión:

- ¿Sentiste que la experiencia de audio 3D fue verdaderamente inmersiva?

  Respuesta: "*Completamente, lo que sí me hubiese gustado que la pantalla fuera completa por defecto, ya que es un poco chica y cuando se agranda los instrumentos se alargan un poco*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- ¿Cómo compararías la calidad y experiencia de este audio 3D con otros programas o sistemas que hayas utilizado anteriormente?

  Respuesta: "*Estoy acostumbrado a utilizar programas con demasiadas funcionalidades y muy complicadas de usar, por esto nunca he utilizado programas que incluyen audio 3D, pero me parece que este programa es una buena forma de implementarlo en la industria.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- ¿En qué tipo de entornos o situaciones consideras que el audio 3D de este *software* brilla particularmente?

  Respuesta: "*Si se pudiera utilizar realidad virtual y que los instrumentos fueran animados o un video, se podría usar este programa para ver conciertos en vivo desde la casa. En producción musical no es tan necesario el audio 3D, aunque sería interesante ver como se puede utilizar el programa para trabajar el panning de una producción al masterizarla.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

• Aplicabilidad en la Industria:

- ¿Crees que este programa podría revolucionar la forma en que trabajas actualmente? ¿Por qué?

  Respuesta: "*En mi área este programa puede revolucionar el panning dando opciones de panning tridimensionales en vez de solo izquierda y derecha para la remasterización de una canción*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- ¿Qué ventajas ves en este *software* en comparación con otras herramientas que hayas utilizado anteriormente?

  Respuesta: "*Por ahora no ofrece ventajas comparativas con los DAW, que es lo que usamos los productores y músicos generalmente. Lo que sí podría ser un buen plug-in para DAW para quienes lo necesiten al momento de masterizar su música.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- ¿En qué áreas específicas (música, arquitectura, diseño de sonido, etc.) ves un mayor potencial para este *software*?

  Respuesta: "*Sería bien útil para planificación de música en vivo en donde se tienen muchos parlantes y se envía distinto audio a cada salida. En la arquitectura me imagino que serviría para tener en cuenta la acústica de las salas.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

- Sugerencias, mejoras y preguntas generales:

  - ¿Hay características específicas que te gustaría que se incorporaran en futuras versiones del *software*?

    Respuesta: "*Principalmente un instalador para el programa en distintos computadores y poder cambiar las distribuciones mientras se usa el programa para así entender realmente el cambio que provoca la distribución musical.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

  - ¿Qué consejo le darías a alguien que esté considerando usar este programa por primera vez?

    Respuesta: "*Tienes que usarlo con Claudio, ya que no lo pude correr en mi computador. Una vez que ya te funcione, el programa intentaría cambiar de a poco las distribuciones para realmente escuchar la diferencia.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

  - ¿Consideras que el *software* satisface tus necesidades profesionales o creativas?

    Respuesta: "*Para la masterización, que es un proceso muy creativo para cada músico, serviría bastante para ver con qué distribución queda mejor el sonido. En otras áreas creo que me enfocaré con el uso de DAW a menos que sea necesario un audio inmersivo en alguna de mis composiciones.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

  - Si pudieras cambiar una sola cosa del programa, ¿cuál sería?

    Respuesta: "*Poder cambiar la distribución mientras se usa el programa o en vivo.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

  - ¿Ves potencial en este programa para transformar la forma en que las personas experimentan y crean música u otras artes?

Respuesta: "*Para efectos de teatro o cine le veo mucho potencial. En la creación de música yo creo que los DAW ofrecen las herramientas necesarias (y las innecesarias también) para crear música.*" (Eric Mayne-Nicholls, comunicación personal, 7 de agosto de 2023, Boston, MA)

5.3.2. Entrevista a Benjamín Zamorano

Benjamín utilizó el *software* de solución con ayuda del memorista para la compilación. Tras el uso del *software* se le entrevistó y dio las siguientes respuestas a las preguntas mencionadas anteriormente:

- Usabilidad y Diseño:
  - ¿Cuál fue tu primera impresión al abrir el programa?

    Respuesta: "*El programa va directo a su objetivo. Al abrir el programa uno ya se encuentra con el escenario y con el simple espacio puede comenzar a escuchar los pianos y parlantes. Si uno sabe para qué está usando el programa, y se sabe los controles, es muy fácil llegar y trabajar sin perder tiempo. El diseño gráfico del escenario podría ser mejor y más realista, pero entiendo que es un programa en una etapa básica y conceptual.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA).
  - ¿Fue fácil encontrar y usar todas las funciones que necesitabas?

    Respuesta: "*Los controles dentro del programa son muy fáciles de entender y usar. Las cosas que uno tiene que configurar fuera de la aplicación misma, por ejemplo los tracks y sonidos de los pianos, la canción misma, etc. Son difíciles de cambiar y entender, ya que se debe hacer fuera del programa.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
  - ¿Hubo alguna función o herramienta que te pareció innecesaria o redundante?

    Respuesta: "*Encuentro que todas las herramientas que ya tiene funcionan y son clave para el programa, de hecho, diría que le faltan varias herramientas para lo que parece ser su propósito. Está todo lo esencial para lo que es el programa, y tiene mucho potencial, solo que le faltan más herramientas de personalización para poder trabajar.* " (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
  - ¿Experimentaste algún problema o dificultad durante el uso del programa?

    Respuesta: "*El ratón y control del personaje, es complicado, ya que uno suele salirse del marco y no tiene mucha movilidad 360.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
- Características de Audio 3D e Inmersión:
  - ¿Sentiste que la experiencia de audio 3D fue verdaderamente inmersiva?

Respuesta: "*Sí. Se nota que el principal enfoque es cómo interactúa el personaje, el sonido y el lugar del escenario en el cual uno está. El sonido 3D es su mayor fuerte.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

- ¿Cómo compararías la calidad y experiencia de este audio 3D con otros programas o sistemas que hayas utilizado anteriormente?

  Respuesta: "*Personalmente no tengo muchos programas como estos con que compararlos, pero comparándolo con pannings de DAWs que uso al trabajar en mi música, funciona bastante parecido, solo que este programa tiene la ventaja que es interactivo, entonces hace el programa automático.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

- ¿En qué tipo de entornos o situaciones consideras que el audio 3D de este *software* brilla particularmente?

  Respuesta: "*Yo veo que este programa podría funcionar perfectamente en paralelo como una herramienta de un DAW. Yo todavía no veo ni un DAW que tenga una herramienta para escuchar mi música como si se estuviera tocando en vivo. Esta herramienta me sería muy útil en el caso de que si estoy componiendo una pieza que eventualmente se tocará en un escenario, usaría este programa como una herramienta dentro del DAW para escuchar mi canción en simulación de escenario.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

- Aplicabilidad en la Industria:

  - ¿Crees que este programa podría revolucionar la forma en que trabajas actualmente? ¿Por qué?

    Respuesta: "*Creo que definitivamente me ayudaría un montón en mi trabajo. Como mencioné, el programa lo veo funcionando muy bien en paralelo con un DAW. Al trabajar en mi música en el DAW, podría usar el programa, mientras compongo, para ir viendo como va sonando y como sonaría en vivo. El programa también me podría ayudar un montón en la preparación de una función en vivo de mi música. Podría usarlo para ver si tengo que cambiar las dinámicas de los sonidos y partituras dependiendo del escenario y como suena la simulación.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

  - ¿Qué ventajas ves en este *software* en comparación con otras herramientas que hayas utilizado anteriormente?

    Respuesta: "*La mayor ventaja que veo es el hecho de que es interactivo y la simulación es en vivo. No me he encontrado ni un programa que haga lo que hace este en esa área. Pero como ya mencioné veo que para este programa más que reemplazar herramientas que uso, las complementarias y usaría en conjunto o paralelo.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

  - ¿En qué áreas específicas (música, arquitectura, diseño de sonido, etc.) ves un mayor potencial para este *software*?

Respuesta: "*Personalmente, como soy músico, veo el mayor potencial en esa área, como una herramienta extra para trabajar y experimentar con mi arte. Pero con más desarrollo al programa también veo cómo podría ser usado en la arquitectura de un escenario o la administración y preparación de una función en vivo.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

- Sugerencias, mejoras y preguntas generales:
  - ¿Hay características específicas que te gustaría que se incorporaran en futuras versiones del *software*?

    Respuesta: "*Definitivamente creo que le faltan varias características y desarrollo al programa. Específicamente le incorporaría más formas de personalizar el programa dentro del programa mismo. Por ejemplo, con más desarrollo se podría hacer que hay una ventana dentro del programa para cambiar todos tracks y audios de los pianos. También cambiaría los Soundfonts a samples de mayor calidad y más realísticos, pero entiendo que la versión de ahora es muy conceptual y básica y los samples de calidad hay que invertir mucho dinero.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
  - ¿Qué consejo le darías a alguien que esté considerando usar este programa por primera vez?

    Respuesta: "*Le diría que es muy complicado lo de la distribución y personalizaron del programa. Y que para sacarle el provecho hay que entenderlo muy bien.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
  - ¿Consideras que el *software* satisface tus necesidades profesionales o creativas?

    Respuesta: "*Para lo que usaría el software, si creo que satisface mis necesidades creativas. Como ya he mencionado varias veces, yo lo usaría para experimentar y escuchar mis composiciones mientras las creo y hacer modificaciones de acuerdo a lo aprendido por el programa. Cómo está el programa ahora, no creo que satisfaga completamente mi objetivo, pero con más desarrollo veo que en el futuro podría funcionar exactamente como me gustaría y podría satisfacerme completamente.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
  - Si pudieras cambiar una sola cosa del programa, ¿cuál sería?

    Respuesta: "*La personalizaron de los tracks y sonidos. Ahora mismo es muy poco intuitivo todo lo que es personalizar el programa para que lo pueda usar de la forma que quiero.*" (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)
  - ¿Ves potencial en este programa para transformar la forma en que las personas experimentan y crean música u otras artes?

    Respuesta: "*Le veo mucho potencial a este programa, ya que nunca he visto algo así. Sin embargo, le falta mucho desarrollo y especificación. Si el programa se desarrolla por el rumbo de la música, y como herramienta de composición y simulación, si veo*

que podría aportar mucho a la creación de mis composiciones, y a la preparación de funciones." (Benjamín Zamorano, comunicación personal, 6 de agosto de 2023, Boston, MA)

5.3.3. Entrevista a Antonio Duval

Antonio utilizó el *software* de solución con ayuda del memorista para la compilación. Tras el uso del *software* se le entrevistó y dio las siguientes respuestas a las preguntas mencionadas anteriormente:

- Usabilidad y Diseño:
  - ¿Cuál fue tu primera impresión al abrir el programa?

    Respuesta: "*Están bastante buenos los rénderes que hace la aplicación.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
  - ¿Fue fácil encontrar y usar todas las funciones que necesitabas?

    Respuesta: "*El movimiento y reproducción sí, aunque, pondría en algún lugar las teclas que hay que apretar. La distribución es bastante fácil de usar, aunque es la parte menos intuitiva del programa.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
  - ¿Hubo alguna función o herramienta que te pareció innecesaria o redundante?

    Respuesta: "*La verdad no. Quizás la distribución se puede manejar de otra forma, pero funciona como está*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
  - ¿Experimentaste algún problema o dificultad durante el uso del programa?

    Respuesta: "*La compilación es muy difícil y sería bueno que no se necesitase ayuda del desarrollador mismo para poder compilar el programa.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
- Características de Audio 3D e Inmersión:
  - ¿Sentiste que la experiencia de audio 3D fue verdaderamente inmersiva?

    Respuesta: "*Sí, la experiencia de audio fue inmersiva y se siente como si se tuviera audio surround. Lo único es que se necesitan o audífonos o home theater para de verdad diferenciar las ubicaciones de las fuentes de audio.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
  - ¿Cómo compararías la calidad y experiencia de este audio 3D con otros programas o sistemas que hayas utilizado anteriormente?

    Respuesta: "*No he utilizado otros programas con audio 3D, pero sí con rénders visualización 3D. En el ámbito de audio se puede comparar con ir a ver una película al cine en vez de en la casa, pero con respecto a la visualización no se compara aún*

con otros programas de renderizado." (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

- ¿En qué tipo de entornos o situaciones consideras que el audio 3D de este *software* brilla particularmente?

  Respuesta: "*Cuando se tienen instrumentos con diferencias de sonido claras y en posiciones opuestas.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

- Aplicabilidad en la Industria:
  - ¿Crees que este programa podría revolucionar la forma en que trabajas actualmente? ¿Por qué?

    Respuesta: "*Le falta mucho para revolucionar el área de la arquitectura, pero va en buen camino. Con un cálculo de acústica de medios y rebote en materiales con cálculo de ángulos, este programa podría ser una herramienta muy buena para arquitectos modelando un teatro, auditorio u otra sala que requiera audio espacial.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
  - ¿Qué ventajas ves en este *software* en comparación con otras herramientas que hayas utilizado anteriormente?

    Respuesta: "*Esta aplicación no tiene ventajas en comparación con los programas actuales utilizados en la arquitectura. Aun así, para personas que no estudiaron arquitectura y desean diseñar su sala de estar o de películas, esta aplicación le ayudaría a decidir donde poner las salidas de audio. Espero que en el futuro este programa pueda compararse con otros en la industria.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).
  - ¿En qué áreas específicas (música, arquitectura, diseño de sonido, etc.) ves un mayor potencial para este *software*?

    Respuesta: "*Yo creo que en el diseño de salas para cine como home theater puede ser muy útil. También en el posicionamiento de parlantes en un evento social. En la arquitectura la acústica requiere muchos más cálculos, pero para uso personal de donde colocar parlantes en alguna sala con este programa modificado tiene mucho potencial.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

- Sugerencias, mejoras y preguntas generales:
  - ¿Hay características específicas que te gustaría que se incorporaran en futuras versiones del *software*?

    Respuesta: "*Una interfaz de usuario y la posibilidad de cambiar la distribución en vivo.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

– ¿Qué consejo le darías a alguien que esté considerando usar este programa por primera vez?

Respuesta: "*Prueba distintas distribuciones y principalmente prueba cambiando el sonido a una pista de audio con el preset de la distribución. Esto es bien fácil para ser un cambio tan radical.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

– ¿Consideras que el *software* satisface tus necesidades profesionales o creativas?

Respuesta: "*En este minuto no. No me enfoco en la acústica de salas, aunque de vez en cuando las oficinas tienen una pésima acústica y hay que mejorarla, pero no utilizaría esta aplicación, ya que es más real ir a la oficina y escuchar como suena. Si tuviera un mejor cálculo de acústica, se podría simular cambios en la oficina para reducir los problemas de acústica.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

– Si pudieras cambiar una sola cosa del programa, ¿cuál sería?

Respuesta: "*Tener un cálculo de acústica real para poder simular salas y sonidos emergentes de distintos lados.*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

– ¿Ves potencial en este programa para transformar la forma en que las personas experimentan y crean música u otras artes?

Respuesta: "*Yo creo que en el ámbito de la música sí puede cambiar. Pero no tengo mucha experiencia con la industria musical*" (Antonio Duval, comunicación personal, 1ro de agosto de 2023, Santiago, Chile).

5.3.4. Entrevista a Matías Greenhill

Matías utilizó el *software* de solución con ayuda del memorista para la compilación. Tras el uso del *software* se le entrevistó y dio las siguientes respuestas a las preguntas mencionadas anteriormente:

- Usabilidad y Diseño:

  – ¿Cuál fue tu primera impresión al abrir el programa?

  Respuesta: "*Primero que nada no lo entendí muy bien. Me dieron muchas instrucciones para su uso y tuve que utilizarlo con su programador a mi lado. Aparte de eso, cuando hice funcionar el programa me pareció muy profesional y comparable con otros programas o videojuegos en calidad.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

  – ¿Fue fácil encontrar y usar todas las funciones que necesitabas?

  Respuesta: "*Todo lo que es control de la aplicación es muy intuitivo, lo de como personalizar la distribución de los instrumentos no lo entendí muy bien y tuve que*

acudir a ayuda." (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿Hubo alguna función o herramienta que te pareció innecesaria o redundante?

  Respuesta: "*No, pero si es complicada la personalización del programa y debería ser más intuitiva.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿Experimentaste algún problema o dificultad durante el uso del programa?

  Respuesta: "*Tuve que pedirle a Claudio ayuda y termino utilizando el programa, el siguiendo mis instrucciones, por lo que tendría que practicar antes de utilizarla para mi uso personal.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- Características de Audio 3D e Inmersión:
  - ¿Sentiste que la experiencia de audio 3D fue verdaderamente inmersiva?

    Respuesta: "*Sí, es muy notoria la diferencia de posiciones de los instrumentos y el audio que reproducen.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

  - ¿Cómo compararías la calidad y experiencia de este audio 3D con otros programas o sistemas que hayas utilizado anteriormente?

    Respuesta: "*No utilizo muchos programas o sistemas como este. En general cuando teníamos que tocar con mi banda en vivo o teníamos un sonidista que nos decía donde posicionarnos o lo improvisábamos en el momento.* " (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

  - ¿En qué tipo de entornos o situaciones consideras que el audio 3D de este *software* brilla particularmente?

    Respuesta: "*Cuando se tiene música que se enfoca en la sensación auditiva como la música inmersiva o música 4D como le llaman. En este programa esa música se escucharía increíblemente inmersiva*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- Aplicabilidad en la Industria:
  - ¿Crees que este programa podría revolucionar la forma en que trabajas actualmente? ¿Por qué?

    Respuesta: "*Yo no trabajo personalmente en la industria de la música u otra que podría utilizar este programa, pero sí como músico aficionado podría utilizarlo para poder realizar posicionamiento de la banda en funciones en vivo cuando no hay sonidista a cargo.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿Qué ventajas ves en este *software* en comparación con otras herramientas que hayas utilizado anteriormente?

  Respuesta: "*Como no he utilizado programas que se acerquen a esto, no puedo responder de forma informada, pero nunca había visto ni oído hablar de un programa así.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿En qué áreas específicas (música, arquitectura, diseño de sonido, etc.) ves un mayor potencial para este *software*?

  Respuesta: "*Yo creo que para funciones en vivo puede ser muy bueno y me imagino que en el proceso de producción musical también aporta si esa pieza se quiere tocar en vivo o hacer como si fuera música en vivo.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

• Sugerencias, mejoras y preguntas generales:

- ¿Hay características específicas que te gustaría que se incorporaran en futuras versiones del *software*?

  Respuesta: "*Principalmente facilidad de uso. Si se quiere que músicos principiantes puedan utilizar este programa, se tiene que asumir que también son principiantes en el uso de programas de este estilo.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿Qué consejo le darías a alguien que esté considerando usar este programa por primera vez?

  Respuesta: "*Tienes que usarlo con Claudio y el te puede mostrar las mejores formas de utilizar el programa.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿Consideras que el *software* satisface tus necesidades profesionales o creativas?

  Respuesta: "*En el proceso creativo yo soy guitarrista y no hago mucha producción, pero un contexto que me toque producir o ayudar en la producción de una de mis canciones utilizaría el programa para revisar que la guitarra se escuche bien en cualquier posición de la sala.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- Si pudieras cambiar una sola cosa del programa, ¿cuál sería?

  Respuesta: "*Hacerlo más intuitivo y poder cambiar las distribuciones sin tener que salir del programa.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

- ¿Ves potencial en este programa para transformar la forma en que las personas experimentan y crean música u otras artes?

Respuesta: "*Yo creo que sí. Puede ser de mucha utilidad para funciones en vivo y para que la gente del público pueda disfrutar la música en vivo de la mejor manera posible.*" (Matías Greenhill, comunicación personal, 2 de agosto de 2023, Santiago, Chile).

5.3.5. Resumen de Resultados

Como se puede notar de las entrevistas a usuarios finales, el *software* es una solución al problema planteado, pero sí requiere de más desarrollo. Principalmente, para las personas sin haber estudiado computación, los archivos json pueden ser complicados y se requiere un buen manual de uso y más ejemplos de configuraciones. Otro avance de desarrollo que requiere el *software* es la posibilidad de incluir cálculo de acústica y una interfaz de usuario amigable. Aparte de esto, los usuarios se veían sorprendidos con el *software* y principalmente creen que si se sigue trabajando en esto se puede llegar a algo que de verdad sea utilizado mundialmente en la industria de la música, arquitectura e incluso videojuegos.

Capítulo 6

Conclusiones

6.1. Resultados y Reflexiones

Durante el semestre de trabajo que se dedicó a este trabajo de memoria de título se creó un *software* de orquestación musical con audio 3D para resolver el problema de que los *software* musicales son muy complejos de utilizar, caros y principalmente no incluyen visualización del espacio 3D ni la inmersión del audio en este. La solución es un *software* que deja al usuario introducirse a un teatro y escuchar la función musical configurada de forma inmersiva y cambiante desde su computador.

El objetivo principal del trabajo de título se cumplió parcialmente, ya que se desarrolló un *software* funcional para ayudar a músicos tanto profesionales como aficionados a visualizar las posiciones de los instrumentos en una orquesta y escuchar cómo suena el arreglo de posiciones de los instrumentos en una posición particular de la sala con los sonidos pertinentes de cada instrumento. Esto fue parcialmente logrado, ya que la facilidad de uso se intenta lograr a través de la configuración, pero el manejo de los archivos JSON demostró ser muy complicado para usuarios sin conocimiento previo sobre este tipo de archivos. Aun así, los usuarios pudieron utilizar el programa de manera efectiva cuando se tenían las instrucciones claras.

El objetivo específico 1 de implementar un sistema de audio 3D para la reproducción de sonido con fuentes en distintas posiciones se logró completamente. El sistema de audio espacial funciona correctamente y deja al usuario completamente inmerso en este audio. La lectura de archivos es el fuerte del *software*, ya que puede leer archivos MIDI e interpretarlos con archivos SoundFont correspondientes. Aun así, los archivos SoundFont de mejor calidad para de verdad simular un instrumento real interpretado por un maestro musical están protegidos por licencias y precio, por lo que se debe invertir para crear una simulación más parecida a la vida real.

El objetivo específico 2 de implementar un sistema de renderizado 3D para la visualización del teatro y los instrumentos en el escenario se cumplió. El sistema de visualización o renderizado es básico y ofrece lo necesario para ayudar con la inmersión del usuario en el programa. La importación de modelos 3D por el usuario son un poco complejas, ya que requieren de un modelo escalado correctamente para que se visualice de forma realista en el *software*. Pero con la adición de modelos distintos en el programa base se puede personalizar

más la experiencia.

El objetivo específico 3 de implementar un sistema de configuración del programa para su fácil uso no se cumplió en su totalidad. Se cumplieron los objetivos de la importación de archivos JSON y su lectura correcta para la configuración del programa, pero esto no hizo intuitivo el uso del programa para los usuarios. Esto es porque el usuario final no cuenta con un buen manual de uso para las configuraciones JSON del *software* y, al tener poco conocimiento de los archivos JSON, una lista de objetos JSON no es intuitiva. Con una interfaz de usuario se podría mejorar este aspecto, pero la implementación de una interfaz de usuario es muy compleja y se debe hacer de forma correcta y esto llevaría mucho tiempo de desarrollo, especialmente para solamente un desarrollador.

El objetivo específico 4 sobre los objetivos de aprendizaje. Se cumplieron mayoritariamente. El único objetivo de aprendizaje que no se cumplió, pero sí se fue mejorando a lo largo del desarrollo, fue el de planificación y manejo del tiempo. Al inicio del desarrollo se perdió mucho tiempo en escoger las bibliotecas más aptas y a aprender a utilizarlas. Con respecto a la planificación, los resultados no fueron buenos, ya que al minuto de desarrollar no se contaba con los posibles errores que iban a aparecer y el orden de desarrollo de sistemas para el software produjo que la mayoría de las veces se tuviera que volver a algún sistema ya implementado y cambiarlo para así hacer que funcione correctamente con el sistema nuevo recién implementado.

El uso de bibliotecas externas ayudo mucho con el desarrollo, pero también creo complicaciones dentro del trabajo. Esto principalmente por el uso de CMake y la importancia de que las bibliotecas utilizadas no tengan dependencias externas que el *software* de solución no las tenga, ya que esto hace que sea escalable y apto para cualquier dispositivo eventualmente.

La arquitectura del *software* funciona de manera correcta, pero hay ámbitos que se pueden mejorar. Para la configuración solo se puede tener un archivo de configuración llamado distribution.json, pero esto hace que sea más complicado cambiar drásticamente las configuraciones. Sería ideal poder elegir el nombre del archivo de configuración al ejecutar el programa y así poder guardar distintas distribuciones de instrumentos en la misma carpeta sin tener que cambiar el nombre y el contenido del archivo en sí.

Como análisis critico, aun cuando se cumplieron los objetivos, la facilidad de uso no es un fuerte de la solución. Esto se debe principalmente a que no se creó un manual claro sobre como utilizar el *software* y esto causo que muchos usuarios del programa tuvieran dificultades al minuto de querer hacer cambios a la distribución de instrumentos o configuración.

El trabajo realizado puede causar un gran impacto en la industria de la música o de la arquitectura y construcción si se desarrolla más con enfoque a la factibilidad de uso del usuario final y a la exactitud del sonido en comparación con la acústica real. Se debe crear un ambiente de simulación con más leyes físicas, pero para esto se requiere más tiempo de investigación y principalmente de desarrollo. La exactitud del sonido no es tan diferente, pero sí hay elementos como reverberación y reflexión que deben ser desarrolladas para simular con más exactitud la acústica de una sala.

Con el trabajo realizado durante este tiempo se aprendió a manejar bibliotecas externas con CMake y como realmente o se analizan los archivos de distintos tipos. También se aprendió a utilizar bibliotecas y especificaciones para poder crear gráficos en un programa y utilizar la GPU, de manera que el programa pueda mostrar mucho detalle gráficamente sin saturar los otros procesos que tiene que realizar un computador constantemente. Asimismo, se logró crear un complejo sistema de audio y aprender las diferencias entre salidas de audio y fuentes de audio junto con la verdadera definición de un sistema de audio. Finalmente, se pudo corroborar que cada programa, aplicación o *software* es un conjunto de muchos sistemas que interactúan entre sí, desde el más alto nivel, como el usuario que interactúa con un teclado, hasta el más bajo nivel, que al pasar por una serie de instrucciones, desarrolladas por distintos equipos y personas, envían una señal al computador a través de puertas lógicas tras todo el recorrido.

Con el desarrollo del proyecto se descubrió que las interfaces de usuario que vemos en los programas son muy difíciles de crear y requieren un equipo grande para poder crear algo que como humanos tecnológicos vemos día a día en los distintos *software* que utilizamos. También se descubrió que el cálculo de acústica es un proceso muy complicado y trabajar con modelos 3D y simular algún efecto de sonido que produce la sala y objetos en ella en cuestión es esencial para tener la verdadera capacidad de simular la acústica en distintas salas.

Con este trabajo el memorista se motivó mucho a trabajar con sistemas de bajo nivel en programación para poder entender realmente todo el trabajo que existe por detrás de simples aplicaciones o programas que hoy creemos simples, pero no es el caso. El área tecnológica de la industria musical es de suma complejidad y al mismo tiempo muy interesante y desafiante. Se espera poder aportar a esta área con este trabajo e ir descubriendo nuevas tecnologías para aportar a la industria musical.

6.2. Trabajo Futuro

Es evidente que el trabajo futuro de este proyecto va en dirección la interactividad con el usuario. A continuación se listan algunas mejoras principales que se le debería realizar al *software* para que de verdad pueda ser utilizado por músicos y demás que lo necesiten:

- Interfaz gráfica: Se puede implementar una interfaz gráfica con la biblioteca ImGui u otra para brindarle al usuario una forma más intuitiva de reproducir el audio, elegir que instrumentos suenan y cuáles están en bucle, cambiar el volumen de las fuentes de sonido y principalmente cambiar la configuración o distribución de los instrumentos y parlantes en el escenario.

- Sistema de configuración adaptable: Actualmente, el *software* no permite bien cambiar la configuración sin editar un archivo o cambiar el nombre de otro, ya que la configuración deseada se recibe de manera estática por parte del*software*. Esto se podría cambiar simplemente pidiendo un nombre de archivo de configuración al ejecutar el programa y utilizar esa entrada del usuario.

- Cálculo de acústica: Con un buen cálculo de acústica, este *software* puede pasar a ser de gran utilidad para la arquitectura y construcción, ya que se puede importar el modelo de una sala y revisar la acústica antes de construir. Esto evitaría el gasto innecesario

de recursos para la construcción de una sala que acústicamente no es agradable. Esto serviría principalmente en la construcción de teatros, auditorios o salas de reunión o de eventos sociales.

- Controles del usuario: Los controles de la cámara no son perfectos y en muchos casos no son intuitivos para los usuarios. Esto puede mejorar utilizando la sensibilidad del ratón y ajustando los controles para parecer reales cuando se trata del movimiento humano. Esto ayudaría más aún con la inmersión del usuario en el programa.

- Física de colisiones: El *software* no detecta colisiones, por lo que la cámara puede atravesar los modelos y los modelos pueden atravesar otros modelos. Para crear una simulación más real de una función en vivo se puede agregar colisión y así tener leyes físicas más reales en la simulación.

Finalmente, se puede seguir desarrollando este *software* para poder utilizarlo como herramienta de enseñanza sobre inmersión de audio o por ejemplo educar sobre falta de visión, ya que con un *software* como este entendemos realmente que para las personas ciegas el sonido es crucial para poder vivir. Con el *software* se demuestra que el humano tiene la capacidad de orientarse solo a base de sonido y para las personas con poca visión o sin visión alguna el sonido los guía en la vida. Para poder utilizarlo como *software* de enseñanza se debe trabajar mucho más en la interfaz de usuario y en la facilidad de uso.

Bibliografía

[1] "Virtual studio technology.", https://en.wikipedia.org/wiki/Virtual_Studio_Technology.

[2] "Daw.", https://es.wikipedia.org/wiki/EstaciÃşn_de_trabajo_de_audio_digital.

[3] ocornut, "Github - ocornut/imgui: Dear imgui: Bloat-free graphical user interface for c++ with minimal dependencies," 2023, https://github.com/ocornut/imgui.

[4] FluidSynth, "Github - fluidsynth/fluidsynth: Software synthesizer based on the soundfont 2 specifications," 2023, https://github.com/FluidSynth/fluidsynth.

[5] "Juce framework," 2023, https://juce.com/.

[6] "Visual studio 2022," 2020, https://visualstudio.microsoft.com/es/.

[7] Gabriel Heinrich, P.-A., "Virtual sound stage.", https://www.parallax-audio.com/.

[8] A/S, O., "Odeon room acoustics software.", https://odeon.dk/.

[9] "Midifile.", https://github.com/craigsapp/midifile.

[10] "Avid pro tools," 2023, https://www.avid.com/es/pro-tools.

[11] "Ableton live.", https://www.ableton.com/en/live/.

[12] Apple, "Logic pro.", https://www.apple.com/logic-pro/.

[13] "Ardour.", https://ardour.org/.

[14] Steinberg, "Cubase.", https://www.steinberg.net/cubase/.

[15] by Production Expert, P., "Daw usage statistigs survey 2023," 2023,

[16] "Native instruments kontak," 2022, https://www.native-instruments.com/en/products/komplete/samplers/kontakt-7/.

[17] "Sampler.", https://en.wikipedia.org/wiki/Sampler_(musical_instrument).

[18] "Waves ssl g-master buss compressor," 2019, https://www.waves.com/plugins/ssl-g-master-buss-compressor.

[19] Contributors, W., "Solid state logic," 2023, https://en.wikipedia.org/wiki/Solid_State_Logic.

[20] knowingneurons, "How does the brain locate sound sources? - knowing neurons," 2013, https://knowingneurons.com/blog/2013/03/15/how-does-the-brain-locate-sound-sources/.

[21] 2015, https://www.apple.com/airpods/compare/?modelList=airpods-2nd-gen,airpods-3rd-gen.

[22] Contributors, W., "Surround sound," 2023, https://en.wikipedia.org/wiki/Surround_sound.

[23] Contributors, W., "Panning (audio)," 2021, https://en.wikipedia.org/wiki/Panning_(audio).

[24] Contributors, W., "Head-related transfer function," 2023,

[25] Mitchelle, A., "Immersive audio is on the rise thanks to tech developments and creative minds," 2022, https://www.inavateonthenet.net/features/article/immersive-audio-is-on-the-rise-thanks-to-tech-developments-and-creative-minds.

[26] 2023, https://professional.dolby.com/product/dolby-atmos-content-creation/dolby-atmos-renderer/#gref.

[27] Shields, J., "Experience your games in full audio immersion with windows sonic and dolby atmos spatial sound," 2017, https://news.xbox.com/en-us/2017/11/13/xbox-one-x-windows-sonic-dolby-atmos-feature/.

[28] "Maestro vr," 2023,

[29] Contributors, W., "Pulse-code modulation," 2023,

[30] "Soundfont.", https://en.wikipedia.org/wiki/SoundFont.

[31] Contributors, W., "Sound blaster awe32," 2023, https://en.wikipedia.org/wiki/Sound_Blaster_AWE32.

[32] Contributors, W., "Clone hero," 2023, https://en.wikipedia.org/wiki/Clone_Hero.

[33] Contributors, W., "Final fantasy xiv (2010 video game)," 2023, https://en.wikipedia.org/wiki/Final_Fantasy_XIV_(2010_video_game).

[34] "Performance actions final fantasy xiv," 2023, https://ffxiv.consolegameswiki.com/wiki/Performance_Actions.

[35] de, C., "Wallace clement sabine," 2007, https://es.wikipedia.org/wiki/Wallace_Clement_Sabine.

[36] Contributors, W., "Carl f. eyring," 2023, https://en.wikipedia.org/wiki/Carl_F._Eyring.

[37] Contributors, W., "Attenuation," 2023, https://en.wikipedia.org/wiki/Attenuation.

[38] Kuttruff, H., Room Acoustics. CRC Press, 6 ed., 2016, doi:10.1201/9781315372150.

[39] "Open graphics library.", https://www.opengl.org/.

[40] "Open asset import library (assimp).", https://github.com/assimp/assimp.

[41] Hiebert, G., "Openal programmer's guide," 2007, https://www.openal.org/documentation/OpenAL_Programmers_Guide.pdf. https://www.openal.org/documentation/OpenAL_Programmers_Guide.pdf.

[42] "Dr libs.", https://github.com/mackron/dr_libs.

[43] "Json for modern c++.", https://github.com/nlohmann/json.git.

[44] "Tinysoundfont.", https://github.com/schellingb/TinySoundFont.git.

[45] Contributors, W., "Graphics processing unit," 2023, https://en.wikipedia.org/wiki/Graphics_processing_unit.

[46] “Graphical user interface.”, https://en.wikipedia.org/wiki/Graphical_user_interface.

[47] “glm.”, https://glm.g-truc.net/0.9.9/index.html.

[48] Foundation, B., “blender.org - home of the blender project - free and open 3d creation software,” 2023, https://www.blender.org/.

[49] Contributors, W., “Soundfont,” 2023, https://en.wikipedia.org/wiki/SoundFont.

[50] Contributors, W., “gltf,” 2023, https://en.wikipedia.org/wiki/GlTF.

[51] Contributors, W., “Wavefront .obj file,” 2023, https://en.wikipedia.org/wiki/Wavefront_.obj_file.

[52] Contributors, W., “Wav,” 2023,

[53] Contributors, W., “Json,” 2023, https://en.wikipedia.org/wiki/JSON.

[54] “sfz format.”, https://sfzformat.com/.

[55] de, C., “Mp3,” 2003, https://es.wikipedia.org/wiki/MP3.

[56] de, C., “Free lossless audio codec,” 2004, https://es.wikipedia.org/wiki/FLAC.

[57] de, C., “Advanced audio coding,” 2004, https://es.wikipedia.org/wiki/Advanced_Audio_Coding.

[58] “itunes,” 2018, https://www.apple.com/itunes/.

[59] Contributors, W., “Mit license,” 2023, https://en.wikipedia.org/wiki/MIT_License.

[60] “Berklee college of music,” 2019, https://www.berklee.edu/.

[61] de, C., “Spotify,” 2009, https://es.wikipedia.org/wiki/Spotify.

[62] Contributors, W., “Film score,” 2023, https://en.wikipedia.org/wiki/Film_score.

[63] de, H., “Centralcorp - habilitación de oficinas,” 2014, https://centralcorp.cl/.

[64] “Duvify | el ecosistema de la renta,” 2019, https://duvify.com/.

[65] “Cmake,” 2023, https://cmake.org/.

ANEXOS

Anexo A

Código fuente GLSL para Shaders

A.1. Vertex Shader

El vertex shader llamado default.vs para los modelos del software está ubicado en Acousent/Resources/Shaders/default.vs y es el siguiente:

Código A.1: Vertex Shader de los modelos

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    TexCoords = aTexCoords;
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}

```

A.2. Fragment Shader

El fragment shader llamado default.fs para los modelos del software está ubicado en Acousent/Resources/Shaders/default.vs y es el siguiente:

Código A.2: Fragment Shader para las texturas de los modelos

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoords;

uniform sampler2D texture_diffuse1;

void main()
{
    FragColor = texture(texture_diffuse1, TexCoords);
}

```

Anexo B

Código de Lectura de Archivos de Audio, MIDI y SoundFont

B.1. Función Load de la clase SoundLibrary

La función Load se encarga de recibir un archivo de audio del tipo WAVE y asignarlo a un parlante como un buffer de OpenAL.

Código B.1: Función Load de SoundLibrary

```
    ALuint SoundLibrary::Load(const char* filename)
{
  while (loading) {
    // busy waiting
  }
  loading = true;

  ALenum err;
  ALuint buffer;
  unsigned int channels;
  unsigned int sample_rate;
  drwav_uint64 frame_count;

  drwav_int16* data = drwav_open_file_and_read_pcm_frames_s16(
    filename,
    &channels,
```

```
    &sample_rate,
    &frame_count,
    nullptr);

  if (!data) {
    throw std::runtime_error(std::string("Failed to open file: ") + filename);
  }
  enum class Format {
    Mono8 = AL_FORMAT_MONO8,
    Mono16 = AL_FORMAT_MONO16,
    Stereo8 = AL_FORMAT_STEREO8,
    Stereo16 = AL_FORMAT_STEREO16
  };
  const auto fmt = channels == 1 ? Format::Mono16 : Format::Stereo16;
  const auto size = sizeof(drwav_int16) * frame_count;

  /* Buffer the audio data into a new buffer object, then free the data and
   * close the file.
   */
  buffer = 0;
  alGenBuffers(1, &buffer);
  alBufferData(buffer, static_cast<ALenum>(fmt), data, size, sample_rate);

  drwav_free(data, nullptr);


  //Hasta Aca

  /* Check if an error occured, and clean up if so. */
  err = alGetError();
  if (err != AL_NO_ERROR)
  {
    fprintf(stderr, "OpenAL Error: %s\n", alGetString(err));
    if (buffer && alIsBuffer(buffer))
      alDeleteBuffers(1, &buffer);
    return 0;
  }

  p_SoundEffectBuffers.push_back(buffer);  // add to the list of known buffers

  loading = false;
  return buffer;
}
```

B.2. Función LoadMIDI de la clase SoundLibrary

La función LoadMIDI se encarga de recibir un archivo MIDI y un archivo SoundFont y asignarlo a un parlante como un buffer de OpenAL.

Código B.2: Función LoadMIDI de SoundLibrary

```
ALuint SoundLibrary::LoadMIDI(const char* midi, const char* sf2, bool singleTrack, int
    ↪ trackNum, int soundNum)
{
  if (!loading) {
    loading = true;
    ALenum err;
    tml_message* TinyMidiLoader = nullptr;
    tsf* tsf = nullptr;

    // Cargar el soundfont desde el archivo.
    tsf = tsf_load_filename(sf2);
    if (!tsf)
    {
      fprintf(stderr, "Could not load SoundFont\n");
      return 1;
    }

    // Inicializa el preset en el canal especial numero 10 del MIDI para utilizar el banco
    ↪ de sonidos de percucion del soundfont (128) si lo tiene
    tsf_channel_set_bank_preset(tsf, 9, 128, 0);

    // Le indicamos a tsf los parametros de rendering del audio.
    tsf_set_output(tsf, TSF_MONO, sampleRate, 0.0f); // Hay que

    // Cargamos el archivo MIDI
    TinyMidiLoader = tml_load_filename(midi);
    if (!TinyMidiLoader)
    {
      fprintf(stderr, "Could not load MIDI file\n");
      return 1;
    }

    // Inicializamos el midi loader global en el primer mensaje de nuestro archivo
    ↪ cargado.
    // Esto para que la funcion pueda utilizar el midi correcto.
    //g_MidiMessage = TinyMidiLoader;

    // Calculamos el largo necesario del buffer de audio.
    auto streamLength = CalculateStreamLength(sampleRate, TinyMidiLoader);
    // Alocamos la memoria necesaria para el buffer completo en forma de arreglo.
    float* audioStream = new float[streamLength / sizeof(float)];

    // Procesamos el midi con el soundfont elegido y lo guardamos en audioStream
    if (!singleTrack)
      ProcessMIDIAndRender(audioStream, streamLength, TinyMidiLoader, tsf);
    else
      ProcessTrackAndRender(audioStream, streamLength, trackNum, soundNum,
    ↪ TinyMidiLoader, tsf);


    // TML Info Variables
```

```
48      int channels;
49      int out_programs;
50      int out_total_notes;
51      unsigned int out_first_note;
52      unsigned int time_length;
53      tml_get_info(TinyMidiLoader, &channels, &out_programs, &out_total_notes, &
        ↪ out_first_note, &time_length);
54
55      cout << "Channels: " << channels << endl;
56      cout << "Programs: " << out_programs << endl;
57      cout << "Total notes: " << out_total_notes << endl;
58      cout << "First note: " << out_first_note << endl;
59      cout << "Time length: " << time_length << endl;
60
61      // Creamos el buffer de OpenAL utilizando el buffer audioStream para la
        ↪ informacion del audio.
62      ALuint buffer;
63      alGenBuffers(1, &buffer);
64      alBufferData(buffer, AL_FORMAT_MONO_FLOAT32, audioStream, static_cast<
        ↪ ALsizei>(streamLength), sampleRate);
65
66      p_SoundEffectBuffers.push_back(buffer); // Add to the list of known buffers
67
68      //Cleanup
69      delete[] audioStream;
70      tsf_close(tsf);
71      tml_free(TinyMidiLoader);
72
73      loading = false;
74      return buffer;
75    }
76  }
```

B.3. Función ProcessMIDIAndRender de SoundLibrary

La función ProcessMIDIAndRender se encarga de recibir un archivo MIDI y un archivo SoundFont y transformarlo a formato SoundBuffer de OpenAL.

Código B.3: Función ProcessMIDIAndRender

```
1  // Esta es la función principal para la lectura de archivos MIDI. Lee los archivos y crea
       ↪ un buffer que lo guarda en el puntero audioStream.
2  void ProcessMIDIAndRender(float* audioStream, int streamLength, tml_message* tml,
       ↪ tsf* tsf)
3  {
4    double g_Msec = 0;
5    // El archivo MIDI se tiene que leer por bloques.
6    int sampleBlock, sampleCount = streamLength / (1 * sizeof(float)); // La
       ↪ multiplicacion por 1 es por la cantidad de canales (MONO en este caso)
```

```
7
8   // Iteracion por bloque
9   for (sampleBlock = TSF_RENDER_EFFECTSAMPLEBLOCK; sampleCount > 0;
      ↪ sampleCount -= sampleBlock)
10  {
11    // Esto para que cuando el bloque sea mayor a los samples restantes no solo utilice
      ↪ un bloque del tamano de los restantes.
12    if (sampleBlock > sampleCount)
13      sampleBlock = sampleCount;
14
15    // Se procesa el MIDI con los parametros necesarios y para cada senal que entrega
      ↪ el MIDI el soundfont debe realizar una accion.
16    for (g_Msec += sampleBlock * (1000.0 / 44100.0); tml && g_Msec >= tml->time;
      ↪ tml = tml->next)
17    {
18      switch (tml->type)
19      {
20      case TML_PROGRAM_CHANGE:
21        // Si el MIDI tiene algun numero de preset especifico para elegir instrumento se
      ↪  detecta aca y se cambia el preset del soundfont
22        tsf_channel_set_presetnumber(tsf, tml->channel, tml->program, (tml->channel
      ↪ == 9));
23        break;
24      case TML_NOTE_ON:
25        // Nota ON
26        tsf_channel_note_on(tsf, tml->channel, tml->key, tml->velocity / 127.0f);
27        break;
28      case TML_NOTE_OFF:
29        // Nota OFF
30        tsf_channel_note_off(tsf, tml->channel, tml->key);
31        break;
32      case TML_PITCH_BEND:
33        // Si hay cambios de tonos se deben ajustar en el soundfont para tener la nota
      ↪ correcta para esa nota en el MIDI.
34        tsf_channel_set_pitchwheel(tsf, tml->channel, tml->pitch_bend);
35        break;
36      case TML_CONTROL_CHANGE:
37        // La libreria tiny sound font recibe los mensajes de controllador MIDI si esque
      ↪  hay cambios.
38        tsf_channel_midi_control(tsf, tml->channel, tml->control, tml->control_value);
39        break;
40      }
41    }
42
43    // La siguiente línea de código crea el audio a partir de valores float y los guarda en
      ↪  el puntero audioStream.
44    // Esto lo realiza.
45    tsf_render_float(tsf, audioStream, sampleBlock, 0);
46    // Contador para efectos de Debug
47    test_int++;
48    if (test_int > 2267) {
49      //cout << "Test_int es maypr a 2267" << endl;
```

```
50        //auto float_size = sizeof(float);
51        //cout << "Float size: " << float_size << endl;
52      }
53      // Se deben calcular cuantos bytes se procesaron para asi avanzar el puntero al
        ↪ siguiente lugar en memoria donde se continuara la cancion.
54      int blockSizeBytes = sampleBlock * (1 * sizeof(float));
55
56      // Incrementar el puntero por la cantidad de samples procesados.
57      audioStream += blockSizeBytes / sizeof(unsigned int);
58
59    }
60  }
```

B.4. Función ProcessTrackAndRender de SoundLibrary

La función ProcessTrackAndRender se encarga de recibir un archivo MIDI y un archivo SoundFont y transformarlo a formato SoundBuffer de OpenAL.

Código B.4: Función ProcessTrackAndRender

```
1  // Esta es la función principal para la lectura de tracks específicos archivos MIDI. Lee
       ↪ los archivos y crea un buffer que lo guarda en el puntero audioStream.
2  void ProcessTrackAndRender(float* audioStream, int streamLength, int trackNumber,
       ↪ int presetNumber, tml_message* tml, tsf* tsf)
3  {
4    double g_Msec = 0;
5    // El archivo MIDI se tiene que leer por bloques.
6    int sampleBlock, sampleCount = streamLength / (1 * sizeof(float)); // La
       ↪ multiplicacion por 1 es por la cantidad de canales (MONO en este caso)
7
8    // Iteracion por bloque
9    for (sampleBlock = TSF_RENDER_EFFECTSAMPLEBLOCK; sampleCount > 0;
       ↪ sampleCount -= sampleBlock)
10   {
11     // Esto para que cuando el bloque sea mayor a los samples restantes no solo utilice
       ↪ un bloque del tamano de los restantes.
12     if (sampleBlock > sampleCount)
13       sampleBlock = sampleCount;
14
15     // Se procesa el MIDI con los parametros necesarios y para cada senal que entrega
       ↪ el MIDI el soundfont debe realizar una accion.
16     for (g_Msec += sampleBlock * (1000.0 / 44100.0); tml && g_Msec >= tml->time;
       ↪ tml = tml->next)
17     {
18       tsf_channel_set_presetindex(tsf, trackNumber, presetNumber);
19       // Revisa si el mensaje midi es el que se desea reproducir.
20       if (tml->channel == trackNumber)
21       {
22         switch (tml->type)
```

```
23          {
24          case TML_PROGRAM_CHANGE:
25            // No hay que revisar la pista de percucion pero se mantiene estos cambios
    ↪ de programa por si se encuentran en el mismo track. Aunque no se cambie el
    ↪ preset.
26            tsf_channel_set_presetnumber(tsf, tml->channel, presetNumber, (tml->
    ↪ channel == 9));
27            break;
28          case TML_NOTE_ON:
29            // Nota ON
30            tsf_channel_note_on(tsf, tml->channel, tml->key, tml->velocity / 127.0f);
31            break;
32          case TML_NOTE_OFF:
33            // Nota OFF
34            tsf_channel_note_off(tsf, tml->channel, tml->key);
35            break;
36          case TML_PITCH_BEND:
37            // Si hay cambios de tonos se deben ajustar en el soundfont para tener la
    ↪ nota correcta para esa nota en el MIDI.
38            tsf_channel_set_pitchwheel(tsf, tml->channel, tml->pitch_bend);
39            break;
40          case TML_CONTROL_CHANGE:
41            // La libreria tiny sound font recibe los mensajes de controllador MIDI si
    ↪ esque hay cambios.
42            tsf_channel_midi_control(tsf, tml->channel, tml->control, tml->
    ↪ control_value);
43            break;
44          }
45        }
46      }
47
48      // La siguiente linea de codigo crea el audio a partir de valores float y los guarda en
    ↪  el puntero audioStream.
49      // Esto lo realiza.
50      tsf_render_float(tsf, audioStream, sampleBlock, 0);
51      // Contador para efectos de Debug
52      test_int++;
53      if (test_int > 2267) {
54        //cout << "Test_int es maypr a 2267" << endl;
55        //auto float_size = sizeof(float);
56        //cout << "Float size: " << float_size << endl;
57      }
58      // Se deben calcular cuantos bytes se procesaron para asi avanzar el puntero al
    ↪ siguiente lugar en memoria donde se continuara la cancion.
59      int blockSizeBytes = sampleBlock * (1 * sizeof(float));
60
61      // Incrementar el puntero por la cantidad de samples procesados.
62      audioStream += blockSizeBytes / sizeof(unsigned int);
63
64    }
65  }
```

Anexo C

Archivos de Configuracion

C.1. Archivo Principal distribution.json

El archivo distribution.json es el que tiene la configuración del programa. Si se requiere cambiar, debe quedar con el mismo nombre en la ubicación en la que se encuentra.

Código C.1: Distribución de Demostración

```
{
  "midi_instruments": [
    {
      "id": 0,
      "midi_file": "venture.mid",
      "name": "Guitar",
      "midi_track": 0,
      "position": {
        "x": -5.0,
        "y": 0.0,
        "z": 0.0
      },
      "soundfont_file": "florestan-subset.sf2",
      "soundfont_preset": 5
    },
    {
      "id": 1,
      "midi_file": "venture.mid",
      "name": "Violin",
      "midi_track": 1,
      "position": {
        "x": 3.0,
        "y": 0.0,
        "z": 0.0
      },
      "soundfont_file": "florestan-subset.sf2",
      "soundfont_preset": 7
    },
    {
      "id": 2,
      "midi_file": "venture.mid",
```

```
      "name": "Piano",
      "midi_track": 4,
      "position": {
        "x": 5.0,
        "y": 0.0,
        "z": 0.0
      },
      "soundfont_file": "florestan-subset.sf2",
      "soundfont_preset": 0
    },
    {
      "id": 3,
      "midi_file": "venture.mid",
      "name": "Marimba",
      "midi_track": 5,
      "position": {
        "x": -3.0,
        "y": 0.0,
        "z": 0.0
      },
      "soundfont_file": "florestan-subset.sf2",
      "soundfont_preset": 2
    },
    {
      "id": 4,
      "midi_file": "venture.mid",
      "name": "cualquier_cosa",
      "midi_track": 9,
      "position": {
        "x": 0.0,
        "y": 0.0,
        "z": 0.0
      },
      "soundfont_file": "florestan-subset.sf2",
      "soundfont_preset": 13
    }
  ],
  "audio_instruments": [
    {
      "audio_file": "Sound1_R.wav",
      "name": "Speaker",
      "id": 2,
      "position": {
        "x": -3.0,
        "y": 6.0,
        "z": 0.0
      }
    },
    {
      "audio_file": "Sound1_R.wav",
      "name": "Speaker",
      "id": 2,
```

```
    "position": {
      "x": 3.0,
      "y": 6.0,
      "z": 0.0
    }
  }
 ]
}
```

C.2. Otra distribución

distribution2.json no se utiliza en el programa, pero es un ejemplo distinto.

Código C.2: Distribución 2 de ejemplo con todos los modelos

```
{
 "audio_instruments": [
  {
   "audio_file": "Sound1_R.wav",
   "name": "Accordion",
   "id": 2,
   "position": {
    "x": -6.0,
    "y": 8.0,
    "z": 0.0
   }
  },
  {
   "audio_file": "Sound1_R.wav",
   "name": "Bagpipes",
   "id": 2,
   "position": {
    "x": -3.0,
    "y": 8.0,
    "z": 0.0
   }
  },
  {
   "audio_file": "Sound1_R.wav",
   "name": "BassGuitar",
   "id": 2,
   "position": {
    "x": 0.0,
    "y": 8.0,
    "z": 0.0
   }
  },
  {
   "audio_file": "Sound1_R.wav",
```

```
      "name": "ElectricGuitar",
      "id": 2,
      "position": {
        "x": 3.0,
        "y": 8.0,
        "z": 0.0
      }
    },
    {
      "audio_file": "Sound1_R.wav",
      "name": "Guitar",
      "id": 2,
      "position": {
        "x": 6.0,
        "y": 8.0,
        "z": 0.0
      }
    },
    {
      "audio_file": "Sound1_R.wav",
      "name": "Speaker",
      "id": 2,
      "position": {
        "x": -4.5,
        "y": 4.0,
        "z": 0.0
      }
    },
    {
      "audio_file": "Sound1_R.wav",
      "name": "Synth",
      "id": 2,
      "position": {
        "x": 0.0,
        "y": 4.0,
        "z": 0.0
      }
    },
    {
      "audio_file": "Sound1_R.wav",
      "name": "Violin",
      "id": 2,
      "position": {
        "x": 4.5,
        "y": 4.0,
        "z": 0.0
      }
    },
    {
      "audio_file": "Sound1_R.wav",
      "name": "Drums",
      "id": 2,
```

```
    "position": {
      "x": -4.5,
      "y": 0.0,
      "z": 0.0
    }
   },
   {
    "audio_file": "Sound1_R.wav",
    "name": "Piano",
    "id": 2,
    "position": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.0
    }
   },
   {
    "audio_file": "Sound1_R.wav",
    "name": "Marimba",
    "id": 2,
    "position": {
      "x": 4.5,
      "y": 0.0,
      "z": 0.0
    }
   }
  ]
}
```