UN PLUGIN DE EFECTO DE AUDIO BASADO EN TRANSFORMACIÓN DE VELOCIDAD PARA ESTACIONES DE TRABAJO DE AUDIO DIGITAL

A SPEED TRANSFORMATION AUDIO EFFECT PLUGIN FOR DIGITAL AUDIO WORKSTATIONS

TRABAJO FIN DE GRADO
CURSO 2020 - 2021

AUTOR
HÉCTOR ULLATE CATALÁN

DIRECTOR
MIGUEL GÓMEZ-ZAMALLOA GIL

GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

UN PLUGIN DE EFECTO DE AUDIO BASADO EN TRANSFORMACIÓN DE VELOCIDAD PARA ESTACIONES DE TRABAJO DE AUDIO DIGITAL

A SPEED TRANSFORMATION AUDIO EFFECT PLUGIN FOR DIGITAL AUDIO WORKSTATIONS

TRABAJO DE FIN DE GRADO EN INGENIERÍA DEL SOFTWARE
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

AUTOR
HÉCTOR ULLATE CATALÁN

DIRECTOR
MIGUEL GÓMEZ-ZAMALLOA GIL

CONVOCATORIA: SEPTIEMBRE 2021
CALIFICACIÓN:

GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

21 DE SEPTIEMBRE DE 2021

RESUMEN

Un plugin de efecto de audio basado en transformación de velocidad para estaciones de trabajo de audio digital

El lanzamiento comercial de los ordenadores personales supuso una revolución en muchos sectores de la sociedad, entre los cuales se encuentra el de la industria musical. Lejos queda aquella época en la que realizar una grabación musical era un proceso tedioso y costoso, reservado a unos pocos elegidos. La llegada del ordenador a los hogares del público general ha desembocado en una democratización de la industria musical, permitiendo que hoy en día cualquier persona pueda producir música con un sonido profesional y una inversión mínima.

Todo esto es gracias a las estaciones de trabajo de audio digital, aplicaciones software que incluyen todo lo necesario para realizar una producción profesional desde cualquier ordenador. Uno de sus puntos fuertes es la posibilidad de ampliar sus funcionalidades mediante programas software externos denominados plugins, compatibles con todas las estaciones de trabajo del mercado. El propósito de este trabajo es el de desarrollar un plugin de efecto que tome una onda de audio entrante y que, a base de manipular la velocidad de ciertos fragmentos, genere una salida alternativa que encaje bien musicalmente, tanto a nivel rítmico como melódico, aportando variedad a la producción.

Palabras clave

plugin, halfspeed, VST, música, DAW, audio digital, JUCE, estación de trabajo de audio digital

ABSTRACT

A speed transformation audio effect plugin for digital audio workstations

The commercial launch of personal computers revolutionised many sectors of society, including the music industry. Gone are the days when making a music record was a tedious and costly process, reserved for a select few. The arrival of the computer to the homes of the general public has led to a democratisation of the music industry, allowing anyone to produce professional sounding music with minimal investment.

All this is thanks to digital audio workstations, software applications that include everything necessary for professional production from any computer. One of their strong points is the possibility of extending their functionalities by means of external software programmes called plugins, which are compatible with all the workstations on the market. The purpose of this project is to develop an effect plugin that takes an incoming audio waveform and, by manipulating the speed of certain fragments, generates an alternative output that fits well musically, both rhythmically and melodically, bringing variety to the production.

Keywords

plugin, halfspeed, VST, music, DAW, digital audio, JUCE, digital audio workstation

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 1 - Introducción

1.1 Motivación

1.1.1 Audio digital

1.1.1.1 Historia del audio digital

La idea de utilizar la tecnología para crear música lleva existiendo desde hace muchos años. El primer hecho del que se tiene constancia se remonta a 1842, cuando Ada Lovelace escribió acerca de la máquina analítica de Charles Babbage [1], considerada la primera computadora de la historia. Lovelace vio el enorme potencial de esa máquina antes incluso de que fuese construida, y afirmó que cualquier cosa que pudiese ser expresada a través de la ciencia abstracta de las operaciones (por ejemplo, la música) podría entonces ser sometida a la potencia computacional de la máquina con increíbles resultados. En palabras de Ada Lovelace:

"The Analytical Engine, on the contrary, is not merely adapted for *tabulating* the results of one particular function and of no other, but for *developing and tabulating* any function whatever. In fact the engine may be described as being the material expression of any indefinite function of any degree of generality and complexity.

Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent."

Sin embargo, la aplicación práctica de estas ideas sobre el mundo de la música tardó 140 años más en llegar. En la década de 1980, Yamaha lanzó al mercado sus sintetizadores digitales de la serie DX, los cuales fueron pioneros en la utilización de circuitos digitales para generar sonidos en el instrumento. El más popular de todos ellos fue el Yamaha DX7, convirtiéndose en uno de los sintetizadores mas vendidos de la historia, en una época en la que el mercado estaba dominado por sintetizadores analógicos.

En esa misma década fue también notable el lanzamiento comercial del disco compacto de audio digital o CD, por sus siglas en inglés, en agosto del año 1982. Fue desarrollado en conjunto por las compañías Philips y Sony, y permitió comenzar a almacenar y reproducir grabaciones de audio digital.

1.1.1.2 Conversión analógica-digital

El audio digital es la representación digital de una señal eléctrica analógica, utilizando un conversor analógico a digital, con el objetivo de que esa señal pueda ser grabada, editada o reproducida utilizando un ordenador, dispositivos de reproducción de audio, u otro tipo de herramientas digitales.

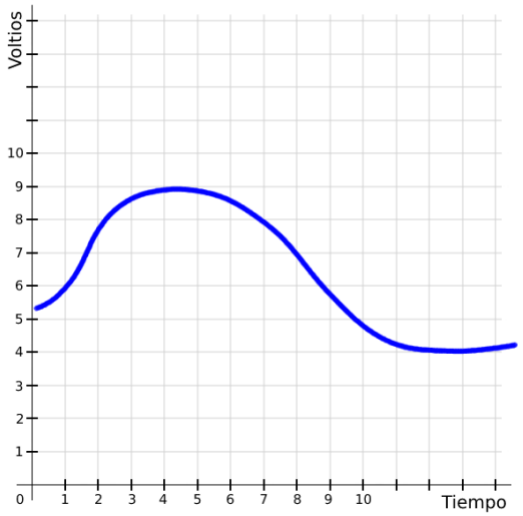

Figura 1-1. Variación del voltaje de una onda analógica a lo largo del tiempo

El proceso de conversión consta de tres fases fundamentales:

- **Muestreo**: durante esta primera fase, como se puede observar en la Figura 1-2, se tomarán muestras de la señal analógica (continua) entrante con el objetivo de obtener una señal digital (discreta). La frecuencia de muestreo define el número de veces por segundo que se toman esas muestras, y será una variable importante a la hora de desarrollar el proyecto, como se explicará más adelante. Para que la onda no pierda información durante el proceso de digitalización, la frecuencia de muestreo debe ser superior al doble de la frecuencia máxima de dicha señal, pero debemos controlar que tampoco sea demasiado alta; el número de muestras que tomemos

determinará la cantidad de información que debemos codificar por segundo para digitalizar la señal, y por tanto el ancho de banda necesario para su transmisión. Para conocer más información sobre este proceso, se recomienda leer acerca del teorema de muestreo de Nyquist-Shannon.

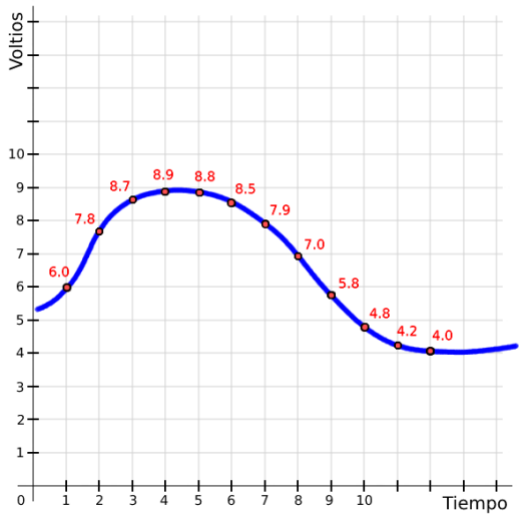

Figura 1-2. Fase de muestreo de la onda analógica dibujada en la Figura 1-1.

– **Cuantificación**: en la fase de cuantificación, se realizarán aproximaciones de las muestras tomadas en el paso anterior (en función de la precisión estipulada por el sistema) con el objetivo de cuantificar con bits estos valores. En el ejemplo mostrado en la Figura 1-3, los valores se redondean a la unidad, tal y como se puede observar en el área coloreada en verde.

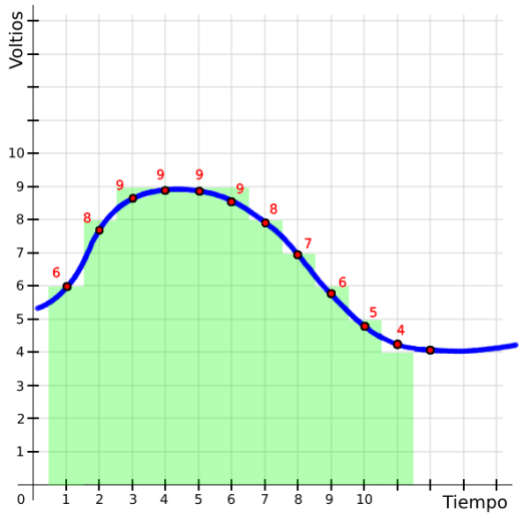

Figura 1-3. Fase de cuantificación de la onda analógica dibujada en la Figura 1-1.

- **Codificación**: por último, una vez cuantificado con bits el valor de cada una de las muestras, la fase de codificación se encargará de convertir esos valores a código binario con el objetivo de que puedan ser procesados por un sistema digital (Ver Figura 1-4).

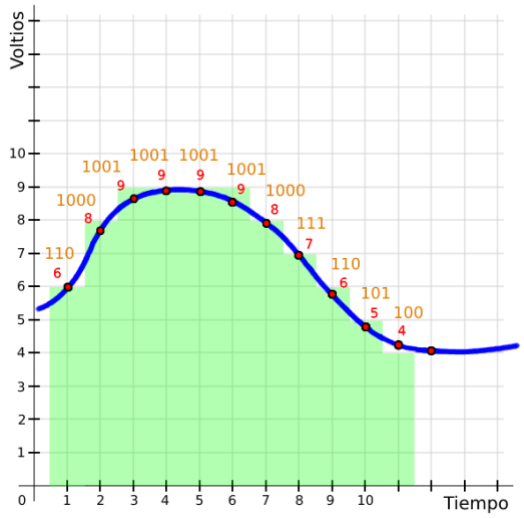

Figura 1-4. Fase de codificación de la onda analógica dibujada en la Figura 1-1.

Finalmente, tras completar el proceso de conversión analógica a digital obtendremos como resultado una secuencia de unos y ceros (bits) que representan de manera digital la señal analógica original. Tal y como se ha explicado dentro de esta sección, la calidad de la señal obtenida dependerá de la frecuencia de muestreo escogida y del número de bits por muestra, es decir, de la precisión de la cuantificación y codificación. En el caso de la señal analógica de la Figura 1-1, el resultado digitalizado sería el siguiente: 0110 1000 1001 1001 1001 1001 1000 1111 1010 1100.

1.1.2 Estaciones de Trabajo de Audio Digital

Actualmente, la música y el sonido digital se encuentran en un momento álgido, en el que cualquier artista o productor amateur puede obtener un sonido óptimo sin tener que acceder a estudios de música profesionales ni invertir grandes cantidades de dinero. Gracias a las Estaciones de Trabajo de Audio Digital o DAWs, por sus siglas en inglés, crear música es más accesible que nunca.

Una Estación de Trabajo de Audio Digital es un sistema que permite la grabación, edición y producción de audio digital, y puede estar formada tanto por herramientas software como herramientas hardware (micrófono, monitores, interfaz de sonido, etc.).

A lo largo de los años 70 y 80, se lanzaron al mercado las primeras estaciones de trabajo, formadas puramente por hardware junto a un software dedicado, debido a limitaciones tecnológicas de la época, tales como el alto coste de almacenamiento o la lenta velocidad de procesamiento. Con los avances en potencia de cómputo de los ordenadores personales e incluso de los dispositivos móviles, hoy en día las estaciones de trabajo son básicamente aplicaciones software que requieren únicamente de una interfaz de audio para introducir audio y MIDI en la máquina y obtener una salida de audio hacia dispositivos de reproducción.

Este tipo de programas suelen incluir funcionalidades como filtros de audio, secuenciadores, mezcladores virtuales y herramientas de gestión y organización de ficheros, con el objetivo de facilitar la producción de canciones, radio, podcasts, bandas sonoras, efectos de sonido y cualquier otro tipo de música que requiera de cierta complejidad.

1.1.2.1 Funcionalidades básicas de una estación de trabajo

A continuación, se explican las funcionalidades básicas de una estación de trabajo de audio digital tomando como ejemplo la estación Logic Pro X [2]. (Figura 1-5).

Figura 1-5. Interfaz de usuario de Logic Pro X

Tal y como hemos explicado previamente, las estaciones de trabajo han evolucionado desde rudimentarios sistemas hardware hasta los complejos sistemas software existentes hoy en día. Estos sistemas son utilizados en estudios profesionales alrededor de todo el mundo debido a que simplifican la tarea de trabajar con audio digital, gracias a una serie de características entre las que se encuentran:

- **Entorno multi-pista**, permitiendo trabajar con múltiples pistas independientes en cada proyecto, dependiendo de las especificaciones del sistema donde estemos trabajando. El término pista es una reminiscencia de la época previa al software musical, cuando cada instrumento podía ser grabado de manera independiente utilizando una cinta multi-pista.
- **Extensa duración de proyecto**, superior a varias horas de duración en función de la frecuencia de muestreo seleccionada. En el caso de Logic Pro X, la extensión es superior a 6 horas cuando utilizamos una frecuencia de muestreo de 96 kHz, y superior a 13 horas en caso de que la frecuencia de muestreo utilizada sea de 44,1 kHz.
- **Extensiones incluidas de gran calidad**, entre las que se encuentran instrumentos software (sintetizadores, samplers, etc.) y efectos de audio (compresores, reverbs, etc.). En el apartado 1.1.3 se explicará qué son estas extensiones o plugins, y qué funcionalidad aportan.

1.1.2.2 Estructura de una estación de trabajo

Dentro de la compleja interfaz que se muestra al poner en marcha una estación de trabajo, son fundamentales los siguientes elementos; comprendiendo su función, tendremos las nociones básicas para realizar una producción musical utilizando una estación de trabajo:

1. **Línea de tiempo**: es la parte principal de la estación de trabajo, ya que a lo largo de ella se colocarán los diferentes bloques de audio o MIDI que conforman la producción. En el eje horizontal está separada en pequeñas partes o compases que dividen el proyecto en pequeñas secciones, y en el eje vertical, se separa en filas independientes en lo que se conoce como pistas. Como veremos más adelante en este apartado, cada pista es

independiente, pudiéndose aplicar por separado a cada una de ellas efectos de audio, volumen, etc.

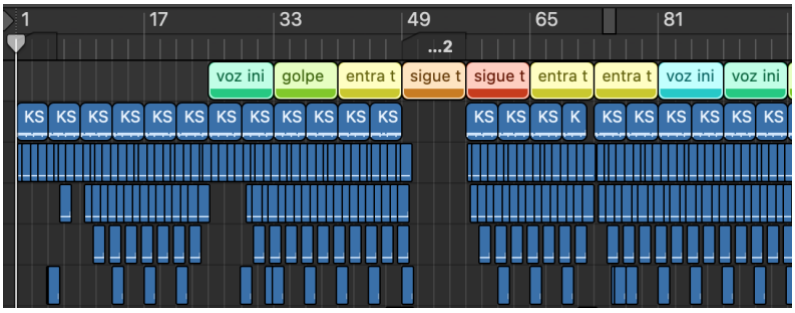

Figura 1-6. Línea de tiempo en Logic Pro X

2. **Centro de control**: en la parte superior de la estación de trabajo encontramos generalmente el centro de control de nuestro proyecto, desde donde podemos iniciar o pausar la reproducción del mismo, o grabar una nueva secuencia. Además, en su interior también se muestra la información que determina las características de nuestra producción, como el compás musical escogido o los pulsos por minuto (término utilizado en el ámbito musical para indicar la velocidad de reproducción de una pieza). Esta información podrá ser modificada en cualquier momento por el usuario de la estación de trabajo para ajustar el proyecto a sus objetivos.

Figura 1-7. Centro de control en Logic Pro X

3. **Vista individual de pista**: tras seleccionar una pista en la vista general de la línea de tiempo se mostrará su vista individual, dentro de la cual podremos realizar todo tipo de modificaciones a dicha pista, como aplicar efectos de audio, modificar su volumen, silenciarla con el botón "M", escucharla aislada con el botón "S", o modificar su posición de "Pan" (la posición de "Pan" cobra relevancia en caso de que dispongamos de audio estéreo, ya que permitirá que configuremos la pista para que el audio de salida se escuche con mayor presencia por el canal izquierdo o el canal derecho).

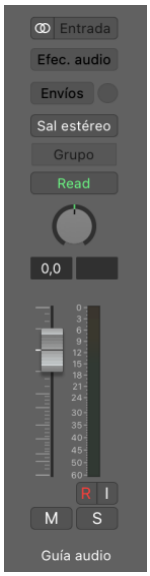

Figura 1-8. Vista individual de pista en Logic Pro X

4. **Editor MIDI**: MIDI, bajo las siglas en inglés de Musical Instrument Digital Interface, es un estándar tecnológico que describe un protocolo de comunicación que permite que ordenadores e instrumentos electrónicos se conecten y se comuniquen entre sí. El sistema MIDI transporta eventos MIDI que contienen información como la notación musical, el tono o la intensidad de las notas musicales, información que puede ser utilizada posteriormente por otros dispositivos o programas para generar una salida de audio, como introduciremos más adelante.

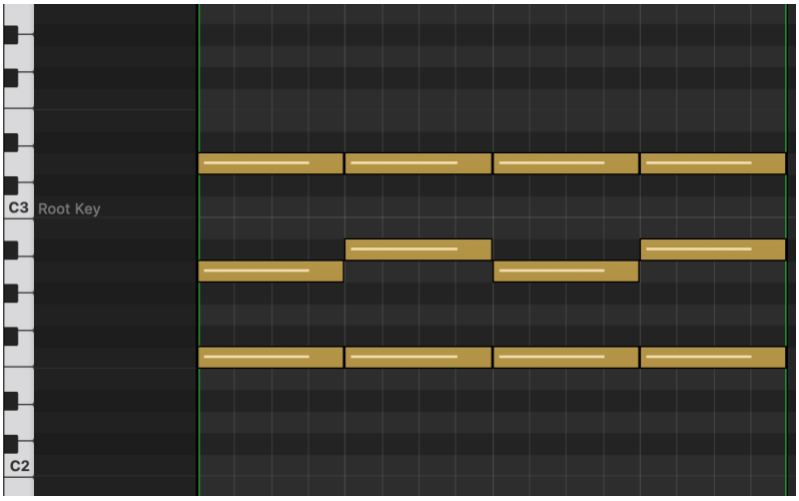

Figura 1-9. Editor MIDI en Logic Pro X

El editor MIDI de nuestra estación de trabajo nos permitirá crear eventos MIDI a lo largo de la línea de tiempo del proyecto, de manera que podamos crear fácilmente melodías utilizando el ratón del ordenador o dispositivos externos como los teclados MIDI. Además, gracias a las características de dicha tecnología, podremos modificar diferentes aspectos de las notas musicales como puede ser su velocidad o su panoramización.

5. **Mezclador**: por último, el mezclador ofrece una visión general de todas las pistas del proyecto, con la finalidad de facilitar el proceso de mezcla, pudiendo manipular el estado de cada una de las pistas (con una vista similar a la de la Figura 1-8) desde un mismo punto.

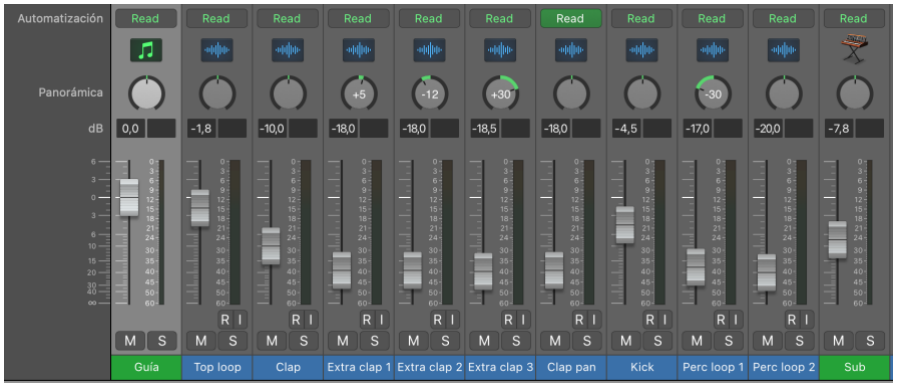

Figura 1-10. Mezclador en Logic Pro X

1.1.3 ¿Qué es un plugin?

Las estaciones de trabajo de audio digital han sido y son desde su creación muy importantes dentro del sector de la producción musical debido a su versatilidad, puesto que es posible ampliar su rango de posibilidades utilizando diferentes tipos de extensiones que añaden funcionalidad extra, entre los que se encuentran los plugins VST.

VST (de las siglas en inglés Virtual Studio Technology) [3] es un estándar desarrollado por la compañía alemana Steinberg con el objetivo de ofrecer una plataforma que permita desarrollar plugins de audio compatibles con las estaciones de trabajo de audio digital. Existen otros estándares con la misma función, como AAX de la compañía Avid Audio o AU de la compañía Apple, pero a día de hoy VST es el más popular de entre todos ellos.

Un plugin VST es una aplicación software desarrollada para ser utilizada dentro de las estaciones de trabajo de audio digital, cuya finalidad es la de añadir funcionalidades extra o mejorar las ya existentes en la estación de trabajo. Un factor diferenciador e importante de este tipo de aplicaciones es que es posible utilizarlas en cualquier estación de trabajo, por lo que, con un solo desarrollo, se amplían las posibilidades de todas las estaciones de trabajo del mercado. Además, generalmente funcionan en tiempo real, es decir, no funcionan a partir de la introducción de un fichero de audio, si no que trabajan sobre la propia pista de manera instantánea, permitiendo al usuario escuchar los cambios realizados sobre la música al momento. Existen tres tipos fundamentales de plugins:

1. **Instrumentos virtuales**: este tipo de plugins generan audio de salida a partir de eventos musicales simbólicos (información MIDI), emulando el sonido de instrumentos físicos como pueden ser sintetizadores, guitarras o pianos, proporcionando al usuario una alternativa económica a los precios de dichos instrumentos.
2. **Efectos de audio**: en este caso, estos plugins toman una entrada de audio ya existente (no lo generan) y la modifican de diferentes maneras. El plugin a desarrollar en este proyecto entra dentro de esta categoría.
3. **Efectos MIDI**: esta clase de plugin es similar a los plugins de efecto de audio, con la diferencia de que toman MIDI como entrada. El estándar MIDI es un estándar tecnológico que permite que ordenadores, instrumentos musicales y hardware se comuniquen entre sí, como ya hemos mencionado anteriormente.

Dentro de la categoría de efectos de audio, existen diferentes tipos de plugins relacionados con la transformación de velocidad, como pueden los efectos de trémolo, cambio de tono o delay, que serán tratados posteriormente en el estado de la cuestión. Tomando como referencia dichos efectos, estudiaremos su comportamiento para entender como funciona su lógica interior, con el objetivo de extrapolarla a nuestros intereses para crear un efecto de transformación de velocidad que amplíe las posibilidades ya existentes.

1.2 Objetivos

El objetivo de este proyecto es el de crear un efecto de audio único a partir de conceptos ya existentes, abriendo al usuario un abanico de posibilidades y permitiéndole obtener muchas variaciones de un mismo sonido con pocos pasos. Implementando este efecto de audio como un plugin VST, se pretende crear una extensión accesible desde cualquier estación de trabajo de audio digital que facilite la utilización y amplíe las posibilidades de los efectos de transformación de velocidad a cualquier usuario.

En particular, se plantean los siguientes subobjetivos:

- Ofrecer al usuario una interfaz profesional y sencilla de entender, obteniendo resultados funcionales desde el momento de encender el plugin.
- Permitir variar el intervalo de aplicación del plugin; por defecto, el plugin funcionará en intervalos de un compás, pero el usuario tendrá hasta ocho posibilidades diferentes.
- Ofrecer al usuario un panel general de control desde el que pueda ajustar al detalle los distintos aspectos de la salida generada. Este panel incluirá:
    - Controlador de transición entre secciones, cuya función será la de suavizar o acentuar la transición entre un intervalo de aplicación y el siguiente.
    - Mezclador que permita combinar en tiempo real el sonido entrante con el sonido resultante, generando una salida que será el resultado de ambos.
    - Controlador de ganancia, para que el usuario pueda ajustar el volumen de salida en caso de que este sea elevado.
- Posibilidad de filtrar la salida de audio del plugin, permitiendo que el usuario elimine las frecuencias graves o agudas del efecto sin necesidad de utilizar otros plugins de ecualización.

- Ofrecer la posibilidad de tener dos iteraciones independientes del efecto trabajando simultáneamente, con el objetivo de crear sonidos innovadores, permitiendo que el usuario mezcle y aplique paneo a cada una de dichas iteraciones de manera independiente.

A nivel de aprendizaje personal, los objetivos son los siguientes:

- Aprender a utilizar herramientas nuevas (como el framework JUCE, del que hablaremos mas adelante) que faciliten el desarrollo y la implementación de nuestras ideas.
- Enfrentarnos a lo largo del desarrollo con problemas y conceptos que no hayamos trabajado durante el grado, mejorando y ampliando así nuestras competencias.

1.3 Plan de trabajo

A continuación, tras haber expuesto la motivación y los objetivos a cumplir en este proyecto, describiremos cual ha sido el proceso seguido para la consecución de dichos objetivos.

En primer lugar, se comenzó este trabajo investigando y tratando de familiarizarse con el framework JUCE, el cual proporciona un entorno y las herramientas necesarias para desarrollar plugins VST utilizando el lenguaje C++. Para afianzar los conocimientos, se desarrolló un primer plugin cuya función era únicamente la de subir y bajar el volumen de la onda de audio que recibiera como entrada, en función de la variable controlada por el usuario.

El siguiente paso fue, siguiendo los consejos del tutor, el desarrollo de una versión simplificada de un plugin de trémolo, debido a que comparten cierta similitud en su comportamiento. Un plugin de tremolo es un efecto de modulación de amplitud que varía de forma periódica el volumen de la onda de audio entrante; esto significa que dividirá el audio en secciones, y aplicará una bajada de volumen en alguna de ellas siguiendo un patrón. Este comportamiento es similar al del plugin objeto de este trabajo, puesto que se dividirá el audio entrante en secciones, aplicando la transformación de velocidad únicamente en la mitad de ellas; la otra mitad se desechará.

Una vez construido este plugin, se reutilizó parte de su código para comenzar el desarrollo del proyecto final. A lo largo de las diferentes versiones, se implementaron y pulieron los elementos básicos del plugin: un controlador principal que permite al usuario modificar la duración del intervalo de efecto, junto al filtro de ecualización mencionado arriba y el panel general de control para gestionar la transición, mezcla y ganancia de la salida de audio. Además, se dedicó tiempo a modificar la apariencia por defecto de cada uno de los elementos gráficos, con el objetivo de profesionalizar el plugin.

Finalmente, se implementó la posibilidad de tener dos iteraciones independientes del efecto trabajando simultáneamente, permitiendo al usuario mezclar y panear cada una de ellas de manera individual.

Capítulo 2 - Estado de la cuestión

En un principio, comprender la utilidad y repercusión que puede tener un efecto de estas características dentro del sector de la producción musical puede ser complicado; por ello, comenzaremos explicando el contexto en el que se encuentran este tipo de efectos actualmente.

La categoría de los efectos de audio relacionados con la transformación de velocidad abarca un amplio espectro de utilidades, entre las que se encuentran por ejemplo los efectos de trémolo o cambio de tono. A modo introductorio, analizaremos los más relevantes explicando sus funcionalidades:

- **Cambio de tono (Pitch shift)**: es capaz de subir o bajar el tono de una señal de audio en un intervalo preestablecido. Para ello, modificará la forma de la señal, apretándola o estirándola por los lados, con el objetivo de variar su frecuencia. Como resultado, la onda resultante será más corta si hemos modificado el tono hacia una frecuencia más alta, y más larga en caso de haberlo modificado hacia una frecuencia más baja. En la Figura 2-1 encontramos una versión de este tipo de efecto realizada por la compañía Waves Audio, de gran renombre dentro de la industria musical.

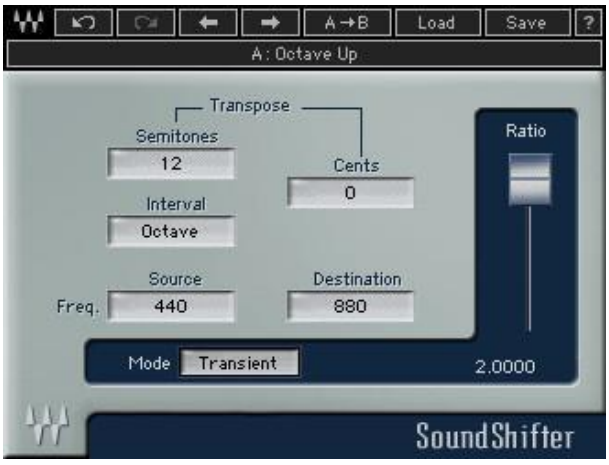

Figura 2-1. Plugin 'SoundShifter' de la compañía Waves Audio

- **Trémolo**: su funcionamiento se basa en provocar una oscilación en la amplitud de onda, de manera que en nuestros oídos escuchemos una oscilación de volumen en el fragmento de audio entrante de manera

periódica. Este tipo de efecto ha sido mencionado previamente en la sección donde se explica el plan de trabajo, puesto que la división del audio en segmentos guarda cierta similitud con la realizada en el efecto objeto de este proyecto.

- **Corrección de tono (Pitch correction)**: es una variación de los efectos de pitch-shift (cambio de tono) mencionados arriba. Su función es la de modificar los diferentes tonos existentes dentro de una señal de audio, con el objetivo de que todos los tonos estén afinados ajustándose a la tonalidad seleccionada por el usuario. Este tipo de efectos son de gran utilidad en el campo de la grabación de voces, consiguiendo que, aunque la afinación del usuario grabado no esté perfectamente ajustada, el resultado final sí lo esté. Este efecto, del que existen diferentes versiones, ha estado utilizándose desde el año 1998, y actualmente es utilizado en muchas de las producciones comerciales que se lanzan al mercado. Su versión más extendida es 'Autotune', de la compañía Antares Audio Technologies (Figura 2-2), siendo su popularidad tal que ha provocado que gran parte de los usuarios utilice su nombre para referirse a este tipo de efectos en general.

Figura 2-2. Plugin 'Autotune Pro' de la compañía Antares Audio Technologies

- **Delay**: almacena una onda de audio entrante con el objetivo de reproducirla una cantidad determinada de veces durante un periodo de tiempo configurado por el usuario, consiguiendo emular así un efecto de eco. Para completar el efecto, el volumen del fragmento almacenado se

irá reduciendo en cada repetición hasta que dicho audio deje de escucharse, tal y como ocurre en el mundo real. A día de hoy, la categoría de efectos de delay es una de las más populares; comprendiendo su funcionamiento, este tipo de efectos pueden transformar una producción amateur en profesional.

Hoy en día, muchos de estos efectos, y entre ellos los efectos de transformación de velocidad, se han vuelto muy populares dentro de la escena Hip-hop. La producción de este estilo de música se caracteriza por tener instrumentales formadas por bucles que se repiten a lo largo de la canción, agregando o quitando elementos a los mismos. A la hora de crear melodías para dichas instrumentales, es muy popular entre los productores utilizar la técnica del "sampleo" [4], consistente en extraer fragmentos o muestras (en inglés, "samples") de otras canciones para reutilizar sus melodías. Esta técnica permite ahorrar tiempo y evitar complicaciones, puesto que cualquier persona es capaz de obtener una melodía sin tener ningún tipo de conocimiento sobre teoría musical.

En relación a la utilización de samples de melodías surge la importancia de los plugins de transformación de velocidad. A la hora de producir la instrumental de una canción es importante tener en cuenta las diferentes secciones de su estructura; por ejemplo, debemos distinguir las secciones del estribillo separándolas de las del verso, de manera que cualquier persona que escuche la instrumental sea capaz de diferenciarlas claramente. Utilizando un plugin de transformación de velocidad sobre nuestra melodía sampleada, seremos capaces de obtener diferentes melodías alternativas a la melodía principal, pero en su misma tonalidad y BPM, lo que lo hace atractivo tanto a productores con conocimientos teóricos de músico como a productores sin ellos.

Actualmente, solo existe un plugin en el mercado que se enfoque exclusivamente en esta función: HalfTime [5]. Desarrollado por la empresa alemana Cableguys, fue lanzado en el año 2017 y esta teniendo una gran aceptación dentro del mundo de la producción musical, llegando a ser utilizado por artistas de renombre como David Guetta, y siendo clave en la producción de canciones con millones de reproducciones.

Figura 2-3. Plugin 'HalfTime' de la compañía Cableguys

Entre sus funciones, destaca la posibilidad de aplicar el efecto sobre intervalos de distinta longitud (1/4 compás, 1/2 compás, 1 compás, etc.) pudiendo aplicar un fundido de audio entre ellos, posibilidad de filtrar los graves y agudos de la salida de audio, o incluso combinar el audio entrante con el audio modificado mediante el controlador de Mix.

A lo largo de este Trabajo de Fin de Grado, se estudiará el comportamiento de este plugin con el objetivo de comprender el funcionamiento de la transformación de velocidad, pudiendo entonces aplicarla en la construcción de nuestro propio plugin de efecto para ampliar sus posibilidades actuales.

Capítulo 3 - Manual de uso del plugin

En este capítulo se detallará el proceso necesario para comenzar a utilizar el plugin dentro de nuestra estación de trabajo de audio digital, y se explicará como utilizar sus diferentes funcionalidades. Se recomienda preparar un archivo de audio con antelación, de manera que podamos comenzar a experimentar con las posibilidades del plugin tras su instalación.

3.1 Instalación y puesta en marcha

En primer lugar, descargaremos los ficheros de instalación del plugin utilizando el enlace a continuación: [6]. Este enlace contiene el plugin en formato VST3 y AU, de manera que el usuario pueda elegir el formato óptimo para su estación de trabajo. Una vez se hayan descargado dichos ficheros, los ubicaremos en el directorio pertinente, en función del formato elegido por el usuario:

- Para instalar el plugin bajo el formato **VST3**, la ruta de instalación será "C:\Archivos de programa\Common Files\VST3" en caso de utilizar el sistema operativo Windows, y "Almacenamiento/Biblioteca/Audio/Plug-Ins/VST3" en caso de utilizar OS X.
- En caso de escoger el formato **AU**, formato disponible únicamente para OS X, la ruta de instalación será "Almacenamiento/Biblioteca/Audio/Plug-Ins/Components".

Tras ubicar los ficheros a utilizar, el plugin ya estará listo para ser utilizado, aunque es posible que sea necesario reiniciar el ordenador para que la estación de trabajo lo detecte. Para comenzar a utilizarlo, abriremos un proyecto dentro de nuestra estación de trabajo y seleccionaremos cualquier pista de audio existente. Dentro de su vista individual, pulsaremos dentro de la sección "Efectos de audio", que mostrará un menú desplegable con todos los plugins instalados en el sistema.

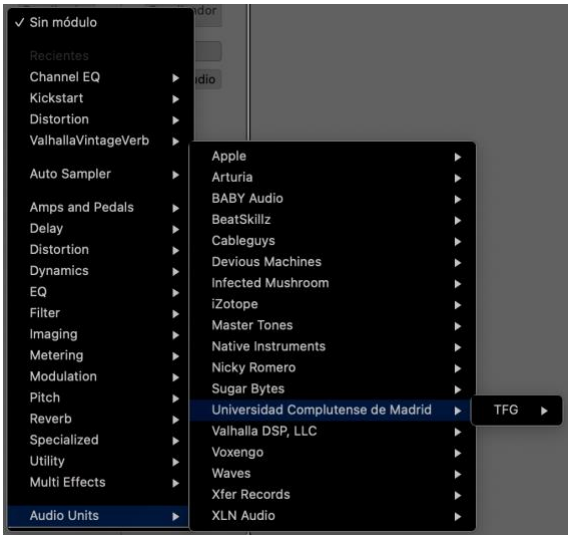

Figura 3-1. Selector de efectos de audio en Logic Pro X

Dentro del apartado titulado "Universidad Complutense de Madrid", encontraremos el desplegable "TFG", el cual deberemos seleccionar para activar nuestro efecto. Una vez tengamos el plugin activado sobre nuestra pista de sonido, dicho efecto se aplicará en tiempo real sobre la salida de audio, permitiendo al usuario escuchar en directo como afectan las diferentes configuraciones del plugin al audio resultante.

3.2 Estructura del plugin

Una vez hemos instalado y activado el plugin, tenemos todo lo necesario para comenzar a utilizarlo. A continuación, se describirán las funciones de cada uno de los componentes del plugin, con el objetivo de que cualquier usuario sea capaz de utilizarlo correctamente.

3.2.1 Sección principal

Figura 3-2. Sección principal del plugin

Situada en la parte izquierda del plugin, es la encargada de gestionar los parámetros fundamentales del efecto de transformación de velocidad. Consta de tres partes, todas ellas relacionadas entre sí:

- **Pestañas o "Tabs"**: situadas en la parte superior, sirven para manejar cada una de las dos iteraciones independientes del efecto. Por defecto, solo estará activada la pestaña principal ("Main Tab"), mientras que la pestaña adicional ("Aux Tab") se podrá encender mediante el botón situado junto a ella.
- **Selector de intervalo**: este controlador sirve para variar el intervalo de aplicación del efecto, permitiendo hasta ocho posibilidades diferentes: 1/16 compás, 1/8 compás, 1/4 compás, 1/2 compás, 1 compás, 2 compases, 4 compases y 8 compases. El intervalo de aplicación establece la duración de un ciclo de efecto, es decir, durante ese periodo de tiempo, el plugin almacenará la mitad del fragmento de audio, y aplicará la transformación de velocidad en tiempo real provocando que la duración de dicha mitad se prolongue ocupando lo que ocupaba inicialmente el fragmento completo, de manera que el usuario escuche el audio original modificado por dicho efecto. Por defecto, "1 compás"

será la opción seleccionada, pero deslizando con el ratón hacia arriba o hacia abajo podremos variar dicha selección.

- **Controlador de pestañas o "Tabs"**: ubicado en la parte inferior de la sección principal, amplía las posibilidades de las dos iteraciones del efecto mediante la inclusión de las funcionalidades explicadas a continuación:
    - **"Tab Mix"**: este controlador permitirá gestionar y mezclar las dos iteraciones independientes del efecto, de manera que la onda de salida sea una combinación de ambas. A mayor porcentaje, mayor será la presencia de la iteración principal gestionada por la "Main Tab".
    - **"Main Pan"**: permite aplicar paneo lateral a la iteración principal gestionada por la "Main Tab", provocando que dicha iteración se escuche más hacia la izquierda o hacia la derecha del oyente.
    - **"Aux Pan"**: permite aplicar paneo lateral a la iteración adicional gestionada por la "Aux Tab".

3.2.2 Filtro de audio

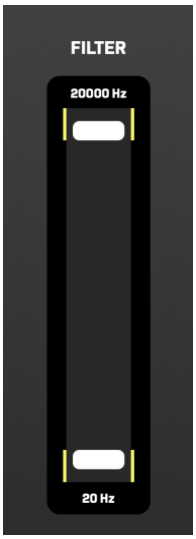

Figura 3-3. Filtro del plugin

Junto a la sección principal nos encontramos con el ecualizador, que contiene un filtro de paso bajo y un filtro de paso alto, ambos implementados en el controlador

vertical de dos selectores, con una frecuencia de operación mínima de 20 Hz y una máxima de 20000 Hz. La razón detrás de escoger dichas frecuencias es que son la mínima y máxima frecuencia respectivamente del espectro audible por el ser humano. El objetivo de este apartado es que el usuario sea capaz de eliminar graves o agudos de la onda de salida sin necesidad de utilizar plugins externos. Para recortar las frecuencias agudas, deberemos deslizar hacia abajo el selector situado en la parte superior, y para recortar las frecuencias graves, deslizaremos hacia arriba el selector inferior.

3.2.3 Control de "Smooth"

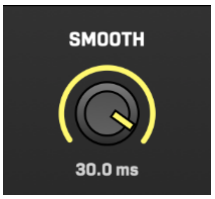

Figura 3-4. Control de Smooth en el plugin

Al comenzar a utilizar el efecto de transformación de velocidad, notaremos que entre los intervalos de aplicación aparecerá un pequeño ruido conocido como "clipping", provocado por las discontinuidades que genera el plugin en la onda de salida. Para disimular estas discontinuidades y resolver el problema, el plugin se encarga de realizar fundidos de audio entrantes y salientes en cada intervalo, consiguiendo solucionar el ruido a través de estas disminuciones de volumen.

Este controlador nos permitirá variar la longitud de cada fundido, de manera que, cuanto mayor sea el valor escogido en milisegundos, mayor será la duración del fundido. En caso de que necesitemos desactivar o reducir el fundido de audio, como se puede dar en ocasiones al manipular audio que requiera de cierta fuerza en su inicio (como el bombo de una batería), simplemente desplazaremos el controlador hasta la posición de 0 ms para desactivar dicho fundido.

3.2.4 Control de "Mix"

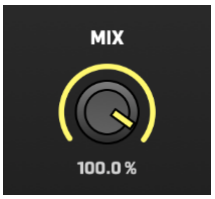

Figura 3-5. Control de Mix en el plugin

Este controlador nos permite mezclar la onda de audio saliente, afectada por el efecto de transformación de velocidad, con la onda de audio entrante, la cual se mantiene intacta. A mayor porcentaje seleccionado, mayor será la presencia de la onda de audio modificada en el resultado final. Su comportamiento es similar al mencionado previamente con el controlador "Tab Mix".

3.2.5 Control de "Gain"

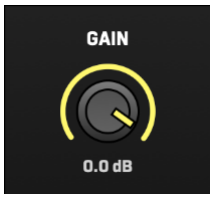

Figura 3-6. Control de Gain en el plugin

En ocasiones, es posible que el audio de salida tenga un volumen elevado. Para poder gestionar este problema desde dentro del propio plugin, se ha implementado este controlador. Deslizando dicho controlador hacia la izquierda, podremos reducir el volumen (o ganancia) de la onda de audio saliente hasta 60 decibelios, haciéndolo prácticamente inaudible.

3.3 Ejemplo de uso

Para finalizar este capítulo, se han incluido dos enlaces que contienen audios de muestra, de manera que el lector de esta memoria, al escuchar dichos fragmentos de audio, pueda comprender de manera clara como afecta la transformación de velocidad a un sample musical.

- En primer lugar, el siguiente enlace mostrará un pequeño fragmento de audio que contiene una grabación de guitarra clásica: [7].

- A continuación, aplicaremos la transformación de velocidad sobre dicho audio seleccionando la configuración de 1 compás en una sola iteración. Se aplicará un fundido de audio entre secciones utilizando el controlador de Smooth, obteniendo como resultado el audio que puede ser escuchado a través del siguiente enlace: [8].

Capítulo 4 - Implementación

En este capítulo se explicarán los conceptos teóricos detrás del algoritmo de transformación de velocidad. Hablaremos de las tecnologías empleadas para el desarrollo del plugin, así como del código escrito para implementar las distintas funcionalidades. A lo largo del desarrollo del proyecto, se ha utilizado el software Git para llevar a cabo un control de versiones del plugin. El código que conforma este proyecto puede encontrarse en el siguiente repositorio de Github: [9]

4.1 Conceptos teóricos

Comenzaremos este capítulo explicando de manera teórica cómo funciona el algoritmo principal de este plugin. Una vez activamos el plugin en cualquier pista de audio o MIDI de la estación de trabajo, este comenzará a recibir una onda de audio entrante. En función del intervalo de aplicación escogido por el usuario, el algoritmo calculará cuantos samples de dicho audio conforman un intervalo, con el objetivo de poder dividir el audio en secciones. Para aplicar la transformación de velocidad, esencialmente se duplicarán todos los samples de la primera mitad del intervalo realizando interpolación, de manera que esa primera mitad se extienda a lo largo de todo el intervalo, obteniendo así el efecto como resultado final.

A modo de ejemplo, en las siguientes figuras se ha representado gráficamente los cambios que sufre una onda de audio tras introducirse en el plugin desarrollado. En primer lugar, en la Figura 4-1 tenemos una onda de audio cuyos valores de ganancia (entre 1 y -1) se representan a lo largo del tiempo, representado en unidades de compás.

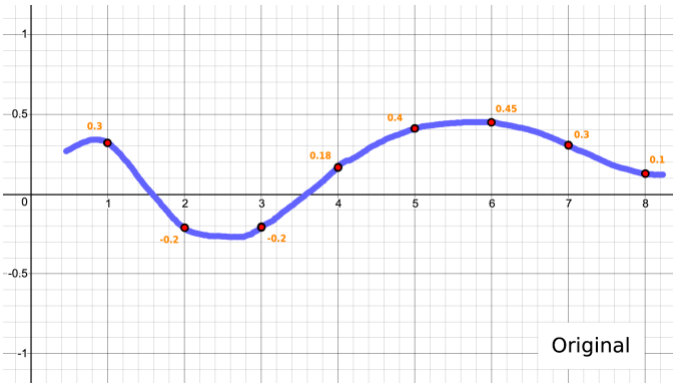

Figura 4-1. Onda de audio antes de ser modificada

Sobre dicha onda se va a aplicar la transformación de velocidad bajo la configuración de 8 compases, de manera que duplicará la extensión de su primera mitad provocando que ocupe ocho compases. Como podemos observar en la Figura 4-1 (onda original), los valores 0.3, -0.2, -0.2, 0.18 de los primero cuatro compases, se ubicarán en las posiciones 2, 4, 6 y 8 de la Figura 4-2 (onda modificada). Para aplicar el efecto de manera exitosa, deberá utilizarse la técnica de interpolación en las posiciones 1, 3, 5 y 7, calculando la media entre el valor anterior y el posterior; en caso de no existir un valor previo, se tomará el 0. De esta manera, en lugar de simplemente duplicar los valores a extender, utilizando la interpolación obtendremos una transición más fluida entre los valores alargados.

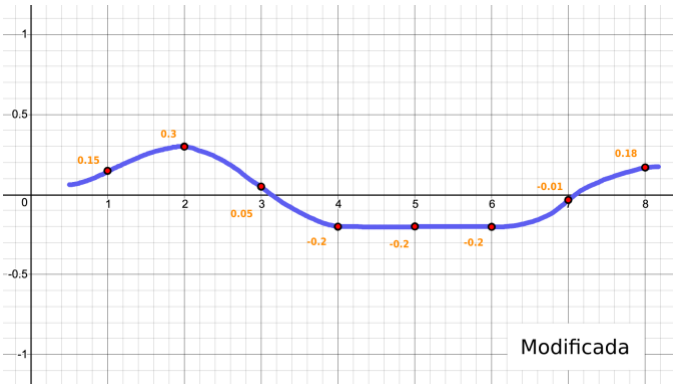

Figura 4-2. Onda de audio tras aplicar transformación de velocidad

A continuación, en la Figura 4-3 encontramos un esquema ejemplificativo de como se comportaría el plugin en caso de aplicarse sobre un bucle de audio de 4 compases, bajo un intervalo de aplicación de 1 compás.

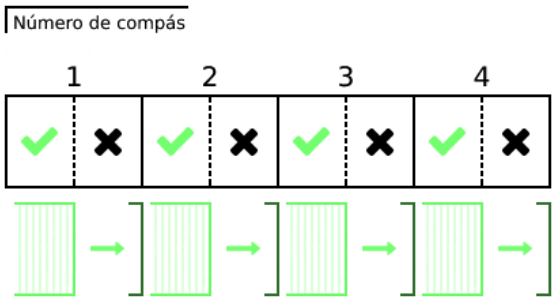

Figura 4-3. Algoritmo funcionando bajo un intervalo de aplicación de 1 compás

Tal y como podemos observar, el algoritmo almacena la primera mitad de todos los intervalos, eliminando su parte restante. La mitad almacenada será interpolada, como hemos mencionado previamente, extendiéndose a lo largo de todo el intervalo.

En un principio, el concepto teórico de este algoritmo no es complejo. La dificultad reside en su implementación, como veremos más adelante, debido a que no recibiremos la onda de audio entrante al completo ya que al utilizar el plugin estamos reproduciendo el audio en tiempo real. En realidad, la onda entrará dividida en pequeños bloques, en función del tamaño de buffer seleccionado dentro de la estación de trabajo.

A modo de demostración, utilizaré la configuración actual de mi estación de trabajo para ejemplificar como se trabajaría con los bloques de audio. Suponiendo un tamaño de bloque de 1024 samples, y una frecuencia de muestreo de 44100 Hz (utilizada en dispositivos como el CD de audio), obtenemos que cada bloque de samples contiene audio de una longitud aproximada de 0,023 segundos. Como podemos observar, deberemos almacenar muchos bloques para obtener las mitades de las que hemos hablado anteriormente, lo cual complica bastante el proceso de transformación.

4.2 JUCE

De entre todas las posibilidades existentes, JUCE [6] ha sido el framework escogido a la hora de implementar este proyecto por varios motivos. En primer lugar, es un framework de código abierto y multiplataforma cuya licencia de uso es gratuita para proyectos no comerciales de uso personal. La herramienta dispone de versiones funcionales en los principales sistemas operativos del mercado y permite crear productos orientados tanto a dispositivos móviles como a ordenadores personales,

siendo compatible con los estándares VST, AU y AAX, mencionados previamente. Además, cuenta con una documentación bien organizada y una amplia comunidad activa, factor realmente útil a la hora de familiarizarse con el framework y solucionar problemas. Por último, su uso se basa en la utilización de C++, lenguaje que ha sido utilizado regularmente a lo largo del grado y con el cual me siento cómodo.

4.2.1 Projucer

A la hora de comenzar un nuevo proyecto con el framework JUCE, tendremos que operar utilizando su herramienta de gestión: Projucer [7]. Esta aplicación es la encargada de realizar la creación y configuración de los proyectos JUCE, ofreciendo hasta nueve plantillas diferentes que sean de ayuda a la hora de crear un nuevo proyecto; cada una de estas plantillas estará orientada a un tipo de proyecto en específico, e inicializará el código y los ficheros necesarios para su funcionamiento.

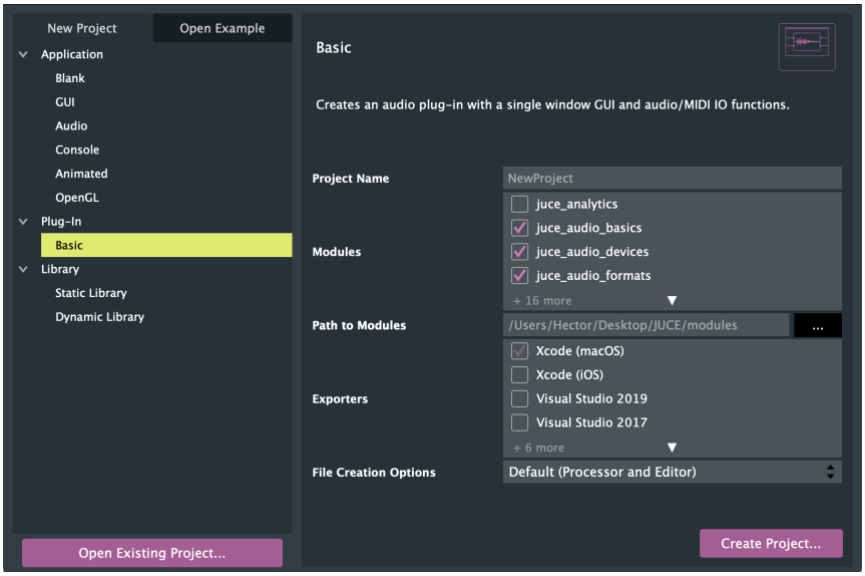

Figura 4-4. Creación de un nuevo proyecto en la herramienta Projucer

En nuestro caso, escogeremos la opción de "Basic Plugin", puesto que creará y configurará un nuevo proyecto listo para construir un plugin de audio con la estructura adecuada.

4.2.2 Estructura de un plugin de audio en JUCE

Al abrir nuestro recién creado proyecto, nos encontraremos con sus dos clases fundamentales, creadas por la plantilla elegida en el Projucer: "PluginEditor" y "PluginProcessor".

La clase "PluginEditor" será la encargada de gestionar todo lo relacionado con la interfaz de usuario del plugin: aspecto gráfico de los componentes, interacción usuario-plugin, etc. Hereda de la clase "AudioProcessorEditor", la cual a su vez lo hace de "Component" [8], clase base de todos los objetos que tengan relación con la interfaz de usuario del proyecto JUCE. Como aspecto a destacar, dentro de la citada clase "Component", existen dos métodos virtuales fundamentales para configurar la interfaz de nuestro plugin, que serán llamados automáticamente por JUCE debido a su estructura dirigida por eventos. Dichos métodos deberán ser implementados dentro de nuestra clase "PluginEditor", y son los siguientes:

- paint(): este método es llamado en el momento que una región de un componente necesita ser redibujada. Dentro del plugin del proyecto se ha utilizado su implementación dentro de la clase "PluginEditor" para dibujar elementos como el título del plugin, los separadores gráficos entre secciones o el fondo de cada una de las secciones.
- resized(): este método es llamado cada vez que las dimensiones del componente (en este caso, la clase "PluginEditor") varían. Dentro del plugin del proyecto se ha utilizado su implementación para ubicar cada uno de los subcomponentes dentro de la interfaz del plugin.

Por otro lado, la clase "PluginProcessor" se encargará de gestionar todo lo relacionado con el procesamiento de los bloques de audio del plugin. Hereda de la clase "AudioProcessor", clase base preparada para llevar a cabo el procesamiento de audio del plugin. Como hemos mencionado previamente, el audio nos llegará separado en pequeños bloques, los cuales seguirán un ciclo determinado de funciones dentro de la clase "PluginProcessor". Los dos métodos fundamentales del ciclo de procesamiento de un bloque son:

- prepareToPlay(): este método es llamado antes de que comience la reproducción del propio audio, de manera que podamos inicializar todas las variables necesarias para su procesado.
- processBlock(): una vez ha comenzado a reproducirse el audio, se invocará al método processBlock(), que traerá consigo el buffer de audio que contiene el bloque actual. Para obtener los diferentes efectos buscados (transformación de velocidad, ecualización, etc.) deberemos modificar el contenido de dicho buffer.

4.3 Algoritmo principal

A continuación, explicaremos de manera técnica el proceso que sigue el plugin para realizar la transformación de velocidad.

En primer lugar, dentro de la función "prepareToPlay()" almacenaremos la frecuencia de muestreo seleccionada en nuestra estación de trabajo. Tras esto, empezará el procesamiento del bloque en la función "processBlock()", en donde comenzaremos inicializando el objeto de tipo "AudioPlayHead" [9]; este objeto contiene información sobre el estado y la posición del cursor de reproducción. A través del mismo, podremos saber en todo momento si el audio se esta reproduciendo, los BPM seleccionados en el proyecto o la posición actual en número de samples, información fundamental para poder realizar las divisiones de audio y por tanto fundamental para implementar el efecto.

Una vez hecho esto, utilizaremos dicha información dentro de "processBlock()" para saber si el audio esta en reproducción o en pausa. En caso de que esté en reproducción, dentro de la función "calculateRemainingWaitingCycles()" calcularemos si podemos comenzar a aplicar el efecto, o debemos esperar, silenciando la salida de audio. Esto es debido a que, como hemos observado anteriormente en la Figura 4-1, las divisiones de audio se realizan en puntos concretos que varían en función de los BPM seleccionados; por tanto, si comenzamos a reproducir el audio fuera de esos puntos, debemos esperar a que el audio llegue a uno de ellos para comenzar a realizar dichas divisiones de manera correcta. Por el contrario, si no esperásemos a dichos puntos para aplicar el efecto, cada vez que reprodujésemos el audio obtendríamos resultados diferentes y fuera de tiempo.

El funcionamiento de la función "calculateRemainingWaitingCycles()" es el siguiente. En primer lugar, calculará la duración total del intervalo de aplicación en número de samples, para lo cual tendrá en cuenta variables como el BPM, el intervalo de aplicación y la frecuencia de muestreo actual. Después, calcularemos el número de samples restantes para acabar el ciclo utilizando la duración total que acabamos de calcular, y la posición actual en número de samples ofreciada por el objeto "AudioPlayHead" que hemos calculado previamente. Finalmente, en función del resultado obtenido, calculará el número de bloques que no debemos procesar; este resultado será utilizado por la función "processBlock()" para realizar un conteo descendente mientras silencia la salida de audio.

Una vez llegue el momento de procesar el audio, entrará en funcionamiento la función "halfspeed()"; como su nombre indica, se encargará de hacer posible el efecto de transformación de velocidad. Esta función será invocada al menos dos veces en cada iteración de la función "processBlock()" para gestionar el canal de audio izquierdo y el canal de audio derecho; en caso de que el usuario haya activado la iteración auxiliar, esta función se ejecutará dos veces más para gestionarla.

Para el correcto funcionamiento de esta función, necesitamos tener un buffer de escritura independiente para cada canal. El audio va entrando al plugin en pequeños bloques, como hemos mencionado previamente, por lo que no podemos simplemente dividirlo en dos partes y modificar una de ellas. Este buffer independiente servirá para ir almacenando samples hasta que determinemos que hemos llegado a la mitad del intervalo de aplicación; por cada iteración de la función "processBlock()", almacenaremos un bloque completo dentro del buffer y leeremos hacia salida solamente medio bloque, interpolándolo de manera que dupliquemos dicha mitad almacenada para conseguir el efecto deseado, debido a lo cual necesitaremos tener dos punteros independientes de lectura y escritura.

Al comienzo de la función "halfspeed()", almacenaremos en el buffer de escritura "writeBuffer" el bloque de audio que nos llega desde la función "processBlock()" y actualizaremos el valor del puntero de escritura asociado. Tras esto, obtendremos un puntero de escritura al buffer de la salida de audio mediante la función "getWritePointer()", y calcularemos el valor del último sample escrito anteriormente (en caso de que lo haya) para poder comenzar a realizar la interpolación. Finalmente, el

bucle while se encargará de generar la salida interpolada a partir de cada valor almacenado previamente en el "writeBuffer", e irá actualizando el puntero de lectura. En caso de que finalice el "writeBuffer" significará que el intervalo de aplicación ha terminado, y por tanto resetearemos dicho buffer y los punteros de lectura y escritura.

Como añadido extra, el algoritmo también tendrá en cuenta que, a pesar de haber finalizado el buffer de escritura, es posible que queden samples restantes en el bloque actual de audio, los cuales serán necesarios para suavizar la transición con el siguiente intervalo. Tras efectuar el vaciado del "writeBuffer", obtendremos la cuenta de samples restantes a partir de la variable "remaining" y leeremos del buffer de entrada los valores para guardarlos dentro del propio "writeBuffer", de manera que el algoritmo pueda aplicar la transformación de velocidad sobre dichos samples restantes.

4.4 Controles de "Pan"

Los controles de panoramización ubicados en la sección principal del plugin tendrán como función modificar la posición estéreo de las dos iteraciones del efecto de manera independiente, como ya hemos mencionado previamente. Para implementar dicha funcionalidad de manera correcta, se ha utilizado la regla de panoramización de potencia constante [10]. Este método permite que el volumen de la señal sea el mismo independientemente de si está orientada hacia el centro o hacia los laterales, lo cual es una ventaja con respecto a otro tipo de reglas, como por ejemplo la regla lineal, en la cual el volumen es mayor en los laterales que en el centro, tal y como se explica dentro de la referencia citada en este párrafo.

La implementación de este algoritmo dentro de JUCE se ha realizado dentro de la función "handlePan()", dentro de la cual comprobamos en primer lugar si el usuario está utilizando dos canales; en caso de que estuviese operando en mono, no tendría sentido aplicar panoramización puesto que sería imposible escucharla. Una vez satisfecha esta condición, calculamos el parámetro de pan (p') establecido por la regla de potencia constante, que será necesario para calcular el porcentaje de panoramización en cada uno de los dos canales de salida. Este parámetro depende del valor seleccionado por el usuario (p), y se calcula utilizando la siguiente fórmula:

$$p' = \frac{\pi(p+1)}{4}$$

Una vez calculado este valor, se modificará el volumen de los samples de cada canal siguiendo el procedimiento marcado por la regla de potencia constante. Para ello, se multiplicarán los samples del canal izquierdo por el coseno de p' y los samples del canal derecho por el seno de p', respectivamente, obteniendo como resultado la señal de audio panoramizada a la izquierda, derecha o centro en función de la configuración escogida por el usuario.

4.5 Filtro de audio

En el proceso de implementación del filtro de ecualización se ha utilizado uno de los módulos incluidos por JUCE por defecto: el módulo DSP (Digital Signal Processing) [10]. Este módulo, tal y como indican sus siglas en inglés, tiene como objetivo facilitar el desempeño de ciertas tareas relacionadas con el procesamiento de señales digitales. En este caso, sus clases y funciones nos serán de gran utilidad a la hora de implementar los filtros, como detallaremos a continuación.

En primer lugar, nos ubicamos de nuevo en la clase "PluginProcessor", donde crearemos dos instancias de la clase "ProcessorChain" que gestionarán cada uno de los dos canales estéreo del filtro. Dentro de la función "prepareToPlay()", debemos inicializar una estructura de tipo "ProcessSpec", con información como la frecuencia de muestreo o el número de samples que caben en un bloque. Esta estructura será pasada por parámetro al método "prepare()" de las dos instancias de la clase "ProcessorChain" que hemos creado previamente.

Después, se inicializará un objeto del tipo "ParameterLayout", perteneciente a la clase "AudioProcessorValueTreeState", encargado de almacenar los diferentes parámetros que configuran los dos filtros de paso bajo y paso alto. Se establecerá una frecuencia mínima de 20 Hz y una máxima de 20000 Hz.

Una vez terminada toda esta configuración, el plugin estará listo para comenzar a ecualizar el audio entrante en el método "processBlock()", tarea que realizará la función "handleFilters()". Dentro de ella, comenzaremos por actualizar los valores actuales que guardan la frecuencia máxima y la frecuencia mínima, tarea desempeñada por la función "updateFilters()", de manera que al procesar el nuevo bloque de audio podamos aplicar dichos cambios. Tras esto, obtendremos los bloques de audio correspondientes a cada canal mediante la función

"getSingleChannelBlock()", que nos devolverá dos objetos de tipo "AudioBlock". Por último, utilizaremos las dos instancias de la clase "ProcessorChain" inicializadas previamente para ejecutar su método "process()", el cual, a través de los bloques de audio obtenidos en el paso anterior, modificará la salida de audio aplicando el filtro de ecualización correspondiente.

4.6 Control de "Smooth"

Una de las partes fundamentales de este efecto de audio es el control de "Smooth", gracias al cual el usuario será capaz de eliminar el clipping que aparece como consecuencia del funcionamiento del efecto. Durante la implementación del plugin, observamos que entre cada una de las secciones de aplicación del efecto aparece un ruido (clipping) que provoca que el audio de salida no suene correctamente. Es por ello que, para solucionar este problema, se han implementado fundidos de audio entrantes y salientes entre cada sección, de manera que el ruido desaparezca.

La implementación de dichos fundidos se ha realizado dentro de la función "halfspeed()", la cual, tal y como hemos explicado anteriormente, alberga el código del algoritmo principal. Para comenzar, se calculará el número de samples de duración del fundido de audio en función del valor en milisegundos escogido por el usuario; conociendo el valor de la frecuencia de muestreo actual, almacenado previamente en la función "prepareToPlay()", realizaremos una regla de tres con el valor escogido por el usuario sabiendo que la frecuencia de muestreo equivale a los samples guardados en 1000 milisegundos. El valor resultante habrá que dividirlo entre 2, puesto que los fundidos se realizarán sobre "writeBuffer" antes de realizar la interpolación de samples que provoca su duplicación.

Una vez obtenido este valor, dentro del bucle donde se realiza la interpolación crearemos la variable "fadeFactor", la cual multiplicará a los samples del buffer de salida para aplicar el fundido de audio cuando sea necesario. Por norma general, el valor de este factor será 1, y se modificará únicamente cuando haya que realizar un fundido de audio. Por ejemplo, en el caso del fundido de audio entrante, se realizarán dos pasos para calcular el factor de fundido; primero, el factor se igualará al valor de "readBufferPosition", el cual irá aumentando progresivamente desde 0 (como es el

fundido de audio entrante nos encontramos al comienzo del buffer). Tras esto, dividiremos el valor actual del factor entre la longitud en número de samples del fundido, de manera que, iteración tras iteración, el factor irá tomando valores desde el 0 hasta el 1 (cuando la variable "readBufferPosition" sea igual a la longitud del fundido). En el caso del fundido de audio saliente, tendremos que realizar el mismo proceso de manera inversa, puesto que los requisitos son los mismos pero el fundido deberá ser descendente.

4.7 Controles de "Mix" y "Tab Mix"

En este caso, volveremos a utilizar la librería DSP de JUCE para implementar la mezcla de señales con y sin efecto. En primer lugar, para implementar el control de "Mix", inicializaremos un objeto de la clase "DryWetMixer" [11], incluida en la librería DSP, cuya función es la de simplificar la tarea de mezclar dos señales de audio diferentes. Tras esta inicialización, utilizaremos el método "setMixingRule()" para establecer la manera en que las dos señales serán mezcladas; la opción "linear" escogida determinará que el volumen de la señal original será igual a uno menos el volumen de la señal modificada. Al igual que en el caso del filtro de ecualización, debemos inicializar un objeto de la clase "ProcessSpec" para pasarlo como parámetro en la función "prepare()" de la instancia de "DryWetMixer".

Una vez configurado este objeto, podremos hacer uso de sus funciones dentro de la función "processBlock()" para conseguir la mezcla. En primer lugar, utilizaremos la función "setWetMixProportion()" para especificar la proporción de señal modificada que debe tener la salida final, información que tomaremos del selector gráfico manipulado por el usuario. A continuación, guardaremos el buffer de audio sin modificar mediante el uso del método "pushDrySamples()". Finalmente, tras haber realizado todo el procesamiento de audio necesario, utilizaremos la función "mixWetSamples()" pasándole como parámetro el buffer modificado, de manera que dicha función pueda sobrescribirlo con la mezcla entre ambas señales.

El procedimiento para implementar el control de "Tab Mix" será prácticamente igual, con la diferencia de que en lugar de guardar la señal original y la señal modificada, guardaremos dos iteraciones independientes de señales modificadas que funcionan simultáneamente. Con el mezclador al 100% se escuchará únicamente la señal principal, y a medida que lo reduzcamos, ganará presencia la señal auxiliar.

4.8 Control de “Gain”

Para concluir con este capítulo, explicaremos el procedimiento utilizado para gestionar el control de volumen de la onda de audio saliente. De este proceso se encargará la función “handleOutputGain()”, la cual se llamará en la parte final del método “processBlock()” con el objetivo de que disponga de la señal con todas las modificaciones añadidas.

Dentro de dicha función, utilizaremos un bucle “for” para iterar sobre los canales de salida existentes. Tras recuperar el valor de volumen en decibelios escogido por el usuario, debemos calcular su equivalente en valores de ganancia para poder utilizarlo con el buffer de audio. Para ello, utilizaremos la función “decibelsToGain()” de la librería “Decibels” de JUCE. Una vez transformado dicho valor, modificaremos el buffer de audio en ambos canales sustituyéndolo por su valor de ganancia correcto. Como resultado, obtendremos la señal de salida modificada ajustada al volumen deseado por el usuario.

Capítulo 5 - Conclusiones y trabajo futuro

En este capítulo sintetizaremos las ideas y soluciones expuestas a lo largo del desarrollo de este trabajo, con el objetivo de extraer una conclusión sólida y general acerca del mismo, y estudiar las posibles ampliaciones y mejoras que se podrían llevar a cabo de cara al futuro.

5.1 Conclusiones

El objetivo principal de este trabajo ha sido el estudio y la elaboración de un plugin de efecto de audio basado en transformación de velocidad; tras haber finalizado el proyecto, hemos obtenido como resultado una utilidad completamente funcional que desempeña dicha tarea. El plugin desarrollado ofrece una interfaz de usuario accesible, proporcionando resultados desde la propia puesta en marcha. Además, permite variar con facilidad el intervalo de aplicación del plugin, con hasta ocho posibilidades diferentes (1/4 compás, 1/2 compás, 1 compás, etc.), y contiene un panel de control desde el cual ajustar al detalle los distintos aspectos de la salida generada.

En adición a las características básicas mencionadas arriba, también se ha trabajado en la implementación de funcionalidades adicionales de cara a mejorar el producto final. En primer lugar, se ha conseguido implementar con éxito un filtro de ecualización con la finalidad de que el usuario sea capaz de eliminar las frecuencias graves o agudas de la salida de audio. Además, un punto a favor de este plugin es la posibilidad de tener dos iteraciones independientes del efecto trabajando simultáneamente, ampliando las posibilidades existentes en el mercado ahora mismo. El usuario tendrá a su disposición controles para mezclar y aplicar paneo a cada una de dichas iteraciones de manera independiente.

Por otra parte, en lo que respecta al desarrollo del plugin, este proyecto ha sido algo totalmente nuevo para mí. Nunca antes había tratado de llevar a cabo un desarrollo relacionado con el sector del audio, ni tenía ningún conocimiento previo relacionado con él. Durante el transcurso de este proyecto, he investigado y estudiado acerca de este sector de desarrollo, aprendiendo a aprovechar el potencial del

framework JUCE, el cual, pese a no tener experiencia previa, ha facilitado la tarea de desarrollo e implementación del proyecto, tanto en el ámbito del procesamiento de audio como en el aspecto gráfico y visual del plugin.

5.2 Trabajo futuro

Teniendo en consideración el hecho de que se han cumplido los objetivos principales dispuestos para este trabajo de fin de grado, a continuación se pretende destacar las posibles extensiones que se podrían llevar a cabo para enriquecer este proyecto en el futuro.

Por una parte, la ausencia de la posibilidad de guardado y recuperación del estado del plugin provoca que una vez cerremos el proyecto en el que nos encontramos dentro de la estación de trabajo perdamos cualquier tipo de configuración que hayamos realizado. Esto se puede resolver exportando como audio la salida resultante del efecto, pero no es una solución efectiva a largo plazo, por lo que implementar esta funcionalidad sería de gran utilidad.

Otra característica valiosa que mejoraría el aspecto final del proyecto es la representación visual de la onda de audio, dibujándola en tiempo real, de manera que se muestre de forma gráfica como afectan los cambios de configuración en el efecto al sonido final. Además de la mejora estética que supondría para el plugin aportándole profesionalidad, podría llegar a ser de gran ayuda para los usuarios facilitando encontrar ruido en la onda resultante para corregirlo posteriormente.

Por último, sería reseñable permitir al usuario modificar la afinación del audio de salida. En ocasiones, al aplicar la transformación de velocidad, se obtienen resultados demasiado graves que no son agradables al oído, haciendo necesaria una transposición a una tonalidad más aguda con herramientas externas. De igual forma, la implementación de diferentes modos de velocidad sería otra mejora a destacar. Multiplicando las secciones ralentizadas por cantidades alternativas a 2 se obtendrían resultados totalmente diferentes que ampliarían el rango de posibilidades ofrecidas por este plugin.

Capítulo 6 - Introduction

6.1 Motivation

6.1.1 Digital audio

6.1.1.1 Digital audio history

The idea of using technology to create music has been around for many years. The first recorded fact dates back to 1842, when Ada Lovelace wrote about Charles Babbage's analytical machine, considered to be the first computer in history. Lovelace saw the enormous potential of that machine before it was even built, and claimed that anything that could be expressed through the abstract science of operations (e.g., music) could then be subjected to the computational power of the machine obtaining incredible results. In the words of Ada Lovelace:

"The Analytical Engine, on the contrary, is not merely adapted for *tabulating* the results of one particular function and of no other, but for *developing and tabulating* any function whatever. In fact the engine may be described as being the material expression of any indefinite function of any degree of generality and complexity.

Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent."

However, the practical application of these ideas in the world of music took another 140 years to arrive. In the 1980s, Yamaha launched its DX digital synthesizers series, which pioneered the use of digital circuitry to generate sounds on the instrument. The most popular of these was the Yamaha DX7, which became one of the best-selling synthesizers in history, at a time when the market was dominated by analogue synthesizers.

Also notable in the same decade was the commercial launch of the digital audio compact disc, or CD, in August 1982. It was developed jointly by Philips and Sony, and allowed digital audio recordings to begin to be stored and played back.

6.1.1.2 Analogue-to-digital conversion

Digital audio is the digital representation of an analogue electrical signal, using an analogue-to-digital converter, so that the signal can be recorded, edited or played back using a computer, audio playback devices, or other digital tools.

The conversion process consists of three fundamental phases:

- **Sampling**: during this first phase, as can be seen in Figure 1-2, samples will be taken from the incoming analogue (continuous) signal in order to obtain a digital (discrete) signal. The sampling frequency defines the number of times per second that these samples are taken, and will be an important variable in the development of the project as will be explained later. In order to avoid losing signal information during the digitisation process, the sampling frequency must be higher than twice the maximum frequency of the signal, but we must control that it is not too high either; the number of samples we take will determine the amount of information we must encode per second to digitise the signal, and therefore the bandwidth necessary for its transmission. To learn more about this process, it is recommended to read about the Nyquist-Shannon sampling theorem.
- **Quantisation**: in the quantisation phase, approximations of the samples taken in the previous step will be made (depending on the precision stipulated by the system) with the aim of quantifying these values with bits. In the example shown in Figure 1-3, the values are rounded to unity, as can be seen in the area coloured in green.
- **Encoding**: finally, once the value of each of the samples has been quantified with bits, the encoding phase is responsible for converting these values to binary code so that they can be processed by a digital system (see Figure 1-4).

Finally, after completing the analogue-to-digital conversion process, the result is a sequence of ones and zeros (bits) that digitally represent the original analogue signal. As explained in this section, the quality of the signal obtained will depend on the sampling frequency chosen and the number of bits per sample, i.e. the precision of the

quantisation and coding. In the case of the analogue signal in Figure 1-1, the digitised result would be as follows: 0110 1000 1001 1001 1001 1001 1000 1111 1010 1100.

6.1.2 Digital Audio Workstations

Today, music and digital sound is at an all-time high, where any amateur artist or producer can achieve optimal sound without having to access professional music studios or invest large amounts of money. Thanks to Digital Audio Workstations or DAWs, creating music is more accessible than ever.

A Digital Audio Workstation is a system that allows the recording, editing and production of digital audio, and can consist of both software and hardware tools (microphone, monitors, sound interface, etc.).

Throughout the 1970s and 1980s, the first workstations were launched to the market, consisting purely of hardware with dedicated software, due to technological limitations of the time, such as the high cost of storage or the slow processing speed. With the increases in computing power of personal computers and even mobile devices, workstations today are basically software applications that require only an audio interface to bring audio and MIDI into the machine and output audio to playback devices.

These types of programmes usually include functionalities such as audio filters, sequencers, virtual mixers and file management and organisation tools, with the aim of facilitating the production of songs, radio, podcasts, soundtracks, sound effects and any other type of music that requires a certain degree of complexity.

6.1.2.1 Basic functionalities of a digital audio workstation

The basic functionalities of a digital audio workstation are explained using Logic Pro X as an example in the following lines. (Figure 1-5).

As previously explained, workstations have evolved from rudimentary hardware systems to the complex software systems that exist today. These systems are used in professional studios around the world because they simplify the task of working with digital audio, due to a large number of features including:

- **Multi-track environment**, allowing the user to work with multiple independent tracks in each project, depending on the specifications of the system you are working on. The term track stems from the days before the existence of music software, when each instrument could be recorded independently using a multi-track tape.
- **Extensive project duration**, up to several hours in length depending on the selected sample rate. In the case of Logic Pro X, the extension is over 6 hours when using a 96 kHz sample rate, and over 13 hours if the sample rate used is 44.1 kHz.
- **High quality extensions are included**, including software instruments (synthesizers, samplers, etc.) and audio effects (compressors, reverbs, etc.). Section 1.1.3 will explain what these extensions or plugins are, and what functionality they provide.

6.1.2.2 Structure of a digital audio workstation

Within the complex interface that is displayed when starting up a workstation, the following elements are fundamental; by understanding their function, we will have the basic notions to carry out a music production using a workstation:

1. **Timeline**: this is the main part of the workstation, since the different audio or MIDI blocks that make up the production will be placed along it. On the horizontal axis, it is separated into small parts or bars that divide the project into small sections, and on the vertical axis, it is separated into independent rows in what are known as tracks. As we will see later in this section, each track is independent, and audio effects, volume, etc. can be applied separately to each of them.
2. **Control centre**: at the top of the workstation we generally find the control centre of our project, from where we can start or pause the playback of our project, or record a new sequence. In addition, it also displays the information that determines the characteristics of our production, such as the beat chosen or the beats per minute (a term used in the musical field to indicate the playback speed of a piece). This information can be

modified at any time by the user of the workstation in order to adjust the project to his purposes.

3. **Individual track view**: after selecting a track in the general view of the timeline, its individual view will be displayed, within which we will be able to make all kinds of modifications to that track, such as applying audio effects, modifying its volume, muting it with the "M" button, listening to it isolated with the "S" button, or modifying its "Pan" position (the "Pan" position becomes relevant in case we have stereo audio, as it will allow us to configure the track so that the output audio is heard with greater presence through the left channel or the right channel).

4. **MIDI Editor**: MIDI, which stands for Musical Instrument Digital Interface, is a technology standard that describes a communication protocol that allows computers and electronic instruments to connect and communicate with each other. The MIDI system carries MIDI events that contain information such as musical notation, pitch or intensity of musical notes, information that can later be used by other devices or programmes to generate an audio output, as we will introduce later.

   The MIDI editor of our workstation will allow us to create MIDI events along the project timeline, so that we can easily create melodies using the computer mouse or external devices such as MIDI keyboards. In addition, thanks to the features of this technology, we can modify different aspects of the musical notes such as velocity or panning.

5. **Mixer**: finally, the mixer offers an overview of all the tracks of the project, in order to facilitate the mixing process, being able to manipulate the current status of each track (with a view similar to the one in Figure 1-8) from a single point.

6.1.3 What is an audio plugin?

Digital audio workstations have been and are since their creation very important in the music production industry due to their versatility, since it is possible to extend their

range of possibilities by using different types of extensions that add extra functionality, among which are the VST plugins.

VST (Virtual Studio Technology) is a standard developed by the German company Steinberg to provide a platform for developing audio plug-ins compatible with digital audio workstations. There are other standards available with the same function, such as AAX from Avid Audio or AU from Apple, but VST is currently the most popular of them all.

A VST plugin is a software application developed to be used within digital audio workstations, whose purpose is to add extra functionalities or improve the existing ones in the workstation. An important differentiating factor of this type of application is that it is possible to use it on any workstation, so that, with a single development, the possibilities of all workstations on the market are extended. In addition, they generally work in real time, that is, they do not need to get an audio file inserted by the user, but instead they work on the track itself instantly, allowing the user to hear the changes made to the music at the moment. There are three main types of plugins:

1. **Virtual instruments**: this type of plugins generate output audio from symbolic musical events (MIDI information), emulating the sound of physical instruments such as synthesizers, guitars or pianos, providing the user with an economical alternative to the prices of these instruments.
2. **Audio effects**: in this case, these plugins take an existing audio input (they do not generate it) and modify it in different ways. The plugin to be developed in this project belongs to this category.
3. **MIDI effects**: this kind of plugin is similar to audio effect plugins, with the difference that they take MIDI as input. The MIDI standard is a technological standard that allows computers, musical instruments and hardware to communicate with each other, as we have mentioned before.

Within the category of audio effects, there are different types of plugins related to speed transformation, such as tremolo, pitch-shift or delay effects, which will be discussed later in the state of the art. Taking these effects as a reference, we will study their behaviour to understand how their inner logic works, with the aim of extrapolating it to our interests in order to create a speed transformation effect that expands the existing possibilities.

6.2 Objectives

The aim of this project is to create a unique audio effect from existing ideas, opening up a range of possibilities for the user and allowing him to obtain different variations of the same sound in a few steps. By implementing this audio effect as a VST plugin, the aim is to create an extension accessible from any digital audio workstation that facilitates the use and extends the possibilities of speed transformation effects to any user.

In particular, the following sub-objectives have been proposed:

- To offer the user a professional and easy to understand interface, obtaining functional results from the moment the plug-in is turned on.
- To allow varying the application interval of the plugin; by default, the plugin will work in intervals of one bar, but the user will have up to eight different possibilities.
- Provide the user with a general control panel from which he can fine-tune the various aspects of the generated output. This panel will include:
    - Transition controller between sections, whose function will be to smooth or accentuate the transition between one application interval and the next.
    - A mixer that allows the incoming sound to be combined in real time with the resulting sound, generating an output that will be the result of both.
    - Gain controller, so that the user can adjust the output volume if it is too high.
- Possibility of filtering the audio output of the plugin, allowing the user to delete the low or high frequencies of the effect from the signal without the need to use other equalisation plugins.
- Offer the possibility of having two independent iterations of the effect working simultaneously, with the aim of creating innovative sounds, allowing the user to mix and pan each of these iterations independently.

On a personal learning level, the objectives are as follows:

- To learn how to use new tools (such as the JUCE framework, which we will talk about later) to facilitate the development and implementation of our ideas.
- Throughout the development process, we will face problems and concepts that we have not worked on during the degree, thus improving and broadening our competences.

6.3 Work plan

After explaining the motivation and objectives of this project, we will now describe the process followed to achieve all these objectives.

First of all, we started this project by researching and trying to become familiar with the JUCE framework, which provides an environment and the necessary tools to develop VST plugins using the C++ language. In order to consolidate the knowledge, a first plugin was developed whose function was only to increase and decrease the volume of the audio waveform received as input, depending on the variable controlled by the user.

The next step was, following the advice of the tutor, the development of a simplified version of a tremolo plugin, as it shares certain behaviours with the plugin subject of this project. A tremolo plugin is an amplitude modulation effect that periodically varies the volume of the incoming audio waveform; this means that it will split the audio into sections, and apply a volume decrease in some of them following a pattern. This behaviour is related to what our plugin is doing, since it will divide the incoming audio into sections, applying the speed transformation only in half of them; the other half will be discarded.

Once this plugin was built, part of its code was reused to start the development of the final project. Throughout the different versions, the basic elements of the plugin were implemented and polished: a main controller that allows the user to modify the duration of the effect range, together with the EQ filter mentioned above and the general control panel to manage the transition, mix and gain of the audio output. In addition, the default

appearance of each of the graphical elements was also improved, with the aim of professionalising the plugin.

Finally, the possibility of having two independent iterations of the effect working simultaneously was implemented, allowing the user to mix and pan each of them individually.

Capítulo 7 - Conclusions and future work

In this chapter we will synthesise the ideas and solutions presented throughout the development of this project, with the aim of giving a solid and general conclusion about it, and to study the possible extensions and improvements that could be carried out in the future.

7.1 Conclusions

The main objective of this work has been the study and development of a speed transformation audio effect plugin for digital audio workstations; after having completed the project, we have obtained as a result a fully functional utility that performs this task. The developed plugin offers an accessible user interface, providing results from the very beginning. Furthermore, it allows to easily modify the interval of application of the plugin, with up to eight different possibilities (1/4 bar, 1/2 bar, 1 bar, etc.), and contains a control panel from which to adjust in detail the different aspects of the generated output.

In addition to the basic features mentioned above, work has also been done to implement additional functionalities in order to improve the final product. First of all, an equalisation filter has been successfully implemented in order to allow the user to remove low or high frequencies from the audio output. On top of that, a positive aspect of this plugin is the possibility to have two independent iterations of the effect working simultaneously, extending the possibilities available on the market right now. The user will have at his disposal controls to mix and pan each of these iterations independently.

On the other hand, as far as the development of the plugin is concerned, this project has been something completely new for me. I had never tried to do any development related to the audio industry before, nor did I have any previous knowledge related to it. During the course of this project, I have researched and studied about this development sector, learning to take advantage of the potential of the JUCE framework, which, despite having no previous experience, has facilitated the task of development and implementation of the project, both in the field of audio processing and in the graphic and visual aspect of the plugin.

7.2 Future work

Taking into consideration the fact that the main objectives set out for this final project have been met, the following is intended to highlight the possible extensions that could be carried out to enrich this project in the future.

On the one hand, the absence of the possibility of saving and recovering the state of the plugin means that once we close the project in which we are working inside the workstation, we will lose any type of configuration that we have made. This can be solved by exporting the output of the effect as an audio file, but it is not an effective long-term solution, so implementing this feature would be very useful.

Another valuable feature that would improve the final appearance of the project is the visual representation of the audio waveform, drawing it in real time, in order to show graphically how the configuration changes in the effect affect the final sound. In addition to the aesthetic improvement that it would bring to the plugin, making it more professional, it could be of great help to users by making it easier to find noise in the resulting waveform in order to correct it later.

Finally, it would be remarkable to allow the user to modify the tuning of the output audio. Sometimes, when applying the speed transformation effect, the resulting audio frequencies are too low and are not pleasant to the listener's ear, making it necessary to transpose the audio to a higher pitch with external tools. Similarly, the implementation of different speed modes would be another improvement. Multiplying the slowed down sections by amounts different from 2 would give totally different results that would widen the range of possibilities offered by this plugin.

BIBLIOGRAFÍA

[1] «The Analytical Engine,» Museum of imaginary musical instruments, [En línea]. Available: http://imaginaryinstruments.org/lovelace-analytical-engine/. [Último acceso: 01 09 2021].

[2] «Logic Pro X,» Apple, [En línea]. Available: https://www.apple.com/es/logic-pro/. [Último acceso: 01 09 2021].

[3] «Virtual Studio Technology,» Wikipedia, [En línea]. Available: https://en.wikipedia.org/wiki/Virtual_Studio_Technology. [Último acceso: 01 09 2021].

[4] «Sampling,» Wikipedia, [En línea]. Available: https://en.wikipedia.org/wiki/Sampling_(music). [Último acceso: 01 09 2021].

[5] «Halftime,» Cableguys, [En línea]. Available: https://www.cableguys.com/halftime.html. [Último acceso: 01 09 2021].

[6] «Ficheros de instalación,» Dropbox, [En línea]. Available: https://www.dropbox.com/s/6woaabmoqxe9ry2/Ficheros_de_instalacion.zip?dl=0. [Último acceso: 18 09 2021].

[7] «Audio demostrativo original,» Dropbox, [En línea]. Available: https://www.dropbox.com/s/2juc8owoaxna3w2/demo_original.wav?dl=0. [Último acceso: 16 09 2021].

[8] «Audio demostrativo modificado,» Dropbox, [En línea]. Available: https://www.dropbox.com/s/63hik6tzbwmynb6/demo_halftime.wav?dl=0. [Último acceso: 16 09 2021].

[9] «Repositorio del código,» Github, [En línea]. Available: https://github.com/hues0s/Trabajo-de-Fin-de-Grado. [Último acceso: 21 09 2021].

[10] «JUCE Framework,» JUCE, [En línea]. Available: https://juce.com/. [Último acceso: 01 09 2021].

[11] «Getting started with the Projucer,» JUCE, [En línea]. Available: https://docs.juce.com/master/tutorial_new_projucer_project.html. [Último acceso: 01 09 2021].

[12] «Component reference,» JUCE, [En línea]. Available: https://docs.juce.com/master/classComponent.html. [Último acceso: 01 09 2021].

[13] «AudioPlayHead reference,» JUCE, [En línea]. Available: https://docs.juce.com/master/classAudioPlayHead.html. [Último acceso: 01 09 2021].

[14] «Pan rules,» RS-MET, [En línea]. Available: http://www.rs-met.com/documents/tutorials/PanRules.pdf. [Último acceso: 16 09 2021].

[15] «Introduction to DSP,» JUCE, [En línea]. Available: https://docs.juce.com/master/tutorial_dsp_introduction.html. [Último acceso: 01 09 2021].

[16] «DryWetMixer reference,» JUCE, [En línea]. Available: https://docs.juce.com/master/classdsp_1_1DryWetMixer.html. [Último acceso: 01 09 2021].