Escuela Politécnica Superior de Linares

Grado en I.T. de las Telecomunicaciones

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

Banco de efectos musicales para guitarra eléctrica sobre dispositivos IPAD

Alumno: Pablo López Santaella

Tutores: Prof. D. Pedro Vera Candeas
Prof. D. Francisco Jesús Cañadas Quesada

Dpto: Ingeniería de Telecomunicación

Septiembre, 2016

Gracias a mis padres, hermana y familia, y a mi ayuda idónea Elisabeth por ayudarme y apoyarme siempre.

ÍNDICE DE CONTENIDOS

1. RESUMEN 6

2. INTRODUCCIÓN 6

2.1 Que es el sonido 7
2.2 Conceptos básicos del sonido 8
2.3 Los parámetros del sonido 9

3. OBJETIVOS 11

4. ESTADO DEL ARTE 12

4.1 Clasificación de los efectos 12
4.1.1. Efectos de rango dinámico 12
4.1.2. Distorsiones 12
4.1.3. Ecualizadores 13
4.1.4. Efectos de modulación 13
4.1.5. Efectos de transposición de tono 14
4.1.6. Efectos basados en retardos 14
4.1.7. Efectos simuladores 14
4.1.8. Reductores de ruido 14

5. ANÁLISIS Y DESARROLLO DE EFECTOS 15

5.1 Compresor 18
5.1.1. Introducción 18
5.1.2. Implementación 18
5.1.3. Adaptación a tiempo real 22

5.2 Expansor 23
5.2.1. Introducción 23
5.2.2. Implementación 23
5.2.3. Adaptación a tiempo real 26

5.3 Fuzz 27
5.3.1. Introducción 27
5.3.2. Implementación 27
5.3.3. Adaptación a tiempo real 29

5.4 Distorsión 30
5.4.1. Introducción 30
5.4.2. Implementación 30
5.4.3. Adaptación a tiempo real 33

5.5 Chorus 34
5.5.1. Introducción 34
5.5.2. Implementación 34
5.5.3. Adaptación a tiempo real 40

5.6 Flanger 42
5.6.1. Introducción 42
5.6.2. Implementación 43
5.6.3. Adaptación a tiempo real 46

5.7 Tremolo 48
5.7.1. Introducción 48
5.7.2. Implementación 49
5.7.3. Adaptación a tiempo real 52

5.8 Vibrato 53
5.8.1. Introducción 53
5.8.2. Implementación 53
5.8.3. Adaptación a tiempo real 57

5.9 Delay 59
5.9.1. Introducción 59
5.9.2. Implementación 59
5.9.3. Adaptación a tiempo real 63

5.10 Reverberación 64
5.10.1. Introducción 64
5.10.2. Implementación 65
5.10.3. Adaptación a tiempo real 69

5.11 Noise Gate 71
5.11.1. Introducción 71
5.11.2. Implementación 71
5.11.3. Adaptación a tiempo real 75

6. MATERIALES Y METODOS 77

6.1 Interfaz de guitarra y conexionado 77

7. RESULTADOS Y DISCUSIÓN 79

7.1 Trabajo final obtenido 79
7.2 Realización para dispositivo Mac OSX 82
7.3 Líneas de futuro 83

8. CONCLUSIONES 84

9. PLIEGO DE CONDICIONES 85

9.1 Requisitos de la aplicación 85
9.2 Características del dispositivo de desarrollo 85

10. ANEXOS 88

10.1 Anexo I. Manual de usuario 88
10.2 Anexo II. Documentación técnica 92

11. BIBLIOGRAFÍA 99

1. RESUMEN

El presente proyecto consiste en el diseño de una aplicación que posea un banco de efectos para guitarra eléctrica, la cual pueda ejecutarse en dispositivos de Apple, concretamente sobre IPAD.

A lo largo de esta memoria se explicará la naturaleza del sonido, conceptos básicos de este y sus parámetros mas importantes. Se describirá la naturaleza de cada uno de los efectos utilizados y su adaptación a una ejecución en tiempo real.
Se detallará el análisis de cada uno de los efectos facilitados en MATLAB e implementados en tiempo real en C++ mediante "*openFrameworks"*, y se describirán los componentes físicos necesarios para el buen funcionamiento de nuestra aplicación. En este caso, se entrará en detalle en la interfaz utilizada para conectar nuestra guitarra con el dispositivo IPAD.

Analizaremos los resultados y detallaremos los métodos utilizados en el TFG para alcanzar el objetivo de este trabajo. Finalmente se harán unas conclusiones y se detallaran los requisitos de la aplicación y las características de nuestros dispositivos en el pliego de condiciones.

Tener en cuenta en todo momento que se ha partido de un PFC anterior cuyo autor es Iván Palomares Alguacil, el cual nos ha servido de punto de partida de nuestro TFG, puesto que estos 11 efectos de los que hemos partido ya fueron realizados por este alumno citado y gran parte de la información referente a la realización y programación de estos efectos, forma parte del trabajo de este alumno adaptado en nuestro TFG, y en esta memoria, con el objetivo de realizar estos efectos en tiempo real y en otro lenguaje de programación.

2. INTRODUCCIÓN

La aplicación de las técnicas de procesamiento digital de señales al tratamiento de las señales de audio se ha convertido en el conjunto de herramientas mas óptimas para la transformación y análisis del sonido, y sus aplicaciones en la música.

Gracias al desarrollo y expansión de la tecnología informática, estas herramientas son cada vez más accesibles a compositores, músicos e ingenieros de sonido, pudiendo ser implementadas hasta en nuestros dispositivos móviles. Hoy en día estas técnicas son ampliamente utilizadas en todo tipo de aplicaciones, que van desde la composición electroacústica hasta la post-producción fonográfica [20].

2.1 Qué es el sonido

Una definición que describe la naturaleza senoidal del sonido es la siguiente:

"Vibración transmitida a través de un material elástico, con frecuencias en el rango aproximado de 20 a 20000 Hercios" [1].

Gracias a esta definición los físicos y los ingenieros utilizan el "sonido" para referirse a una onda de presión que se propaga por el aire, algo que puede ser fácilmente medido, digitalizado y analizado.

Por otra parte, desde un punto de vista perceptual del sonido puede referirse como:

"Sensación producida en el órgano del oído por el movimiento vibratorio de los cuerpos, transmitido por un medio elástico, como el aire" [2].

A diferencia de la primera, esta definición se refiere a la percepción o sensación que ocurre dentro del oído, algo que es mucho más difícil de cuantificar.

Basándonos entonces en la primera definición, las ondas sonoras se pueden representar como se muestra en la Figura 1, donde las zonas de altas presiones aparecen por encima de la "presión nominal" y las zonas de bajas presiones se muestran debajo. Esta forma de onda particular (onda senoidal) se puede caracterizar por tres términos matemáticos: frecuencia, amplitud y fase.

La frecuencia de una sinusoide es el número de oscilaciones completas que suceden en un segundo. Así si una onda con frecuencia de 100 Hz oscila 100 veces cada segundo. En lo correspondiente a una onda sonora las moléculas de aire se comprimen y estiran 100 veces en cada segundo [16].

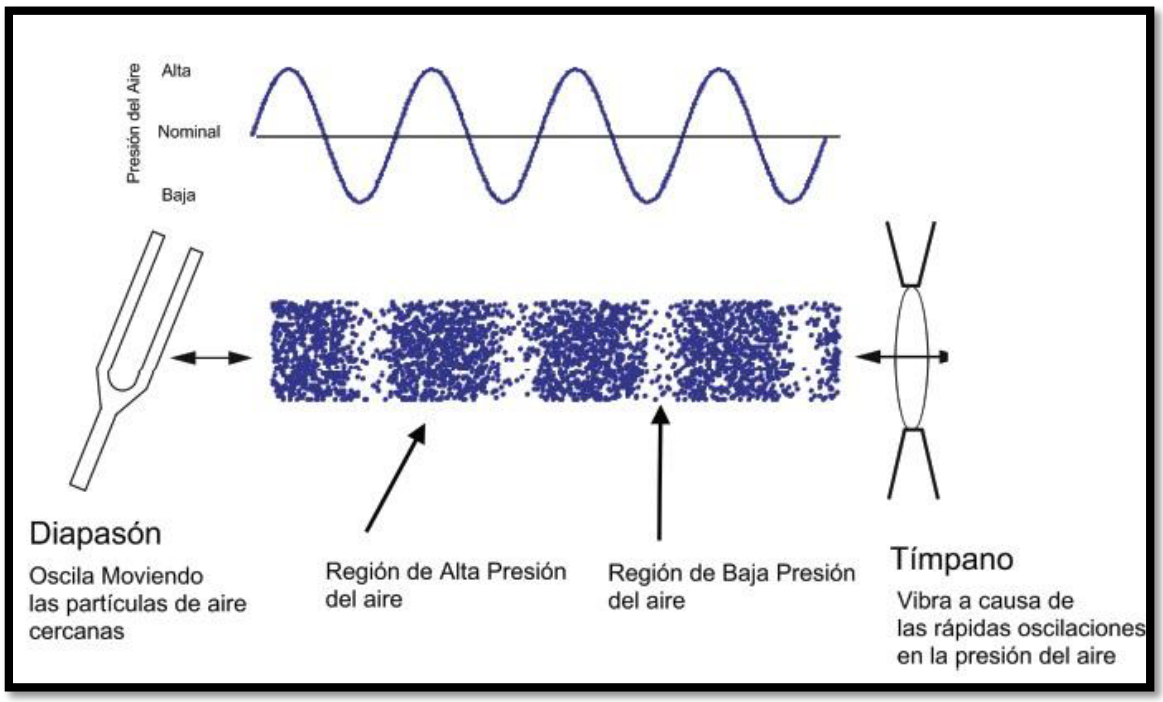


Figura 1: El sonido como una onda de presión. [3]

2.2 Conceptos básicos del sonido

Como onda, el sonido responde a las siguientes características:

1. Es una onda mecánica

Las ondas mecánicas no pueden desplazarse en el vacío, necesitan hacerlo a través de un medio material (aire, agua, cuerpo sólido).
Además, de que exista un medio material, se requiere que éste sea elástico. Un medio rígido no permite la transmisión del sonido, porque no permite las vibraciones.
La propagación de la perturbación se produce por la compresión y expansión del medio por el que se propagan. La elasticidad del medio permite que cada partícula transmita la perturbación a la partícula adyacente, dando origen a un movimiento en cadena.

2. Es una onda longitudinal

El movimiento de las partículas que transporta la onda se desplaza en la misma dirección de propagación de la onda.

3. Es una onda esférica

Las ondas sonoras son ondas tridimensionales, es decir, se desplazan en tres direcciones y sus frentes de ondas son esferas radiales que salen de la fuente de perturbación en todas las direcciones. El principio de Huygens afirma que cada uno de los puntos de un frente de ondas esféricas puede ser

considerado como un nuevo foco emisor de ondas secundarias también esféricas, que como la originaria, avanzarán en el sentido de la perturbación con la misma velocidad y frecuencia que la onda primaria [19].

2.3 Los parámetros del sonido

Existen 6 propiedades o atributos que pueden describir lo que un ser humano experimenta al escuchar un sonido:

- Tono o frecuencia

Se refiere a la rapidez de ocurrencia de las vibraciones que componen un tono. Cuando la frecuencia aumenta, también lo hace el tono.

- Sonoridad

Indica cuan fuerte o suave se escucha un sonido, y se refiere a la amplitud de la onda vibratoria. La cantidad de compresión que sucede en las moléculas de aire, cuando el sonido viaje a través de ellas, no implica la rapidez a la que se desplazan, sino cuanta energía está almacenada en ellas.

- Timbre

Se refiere a la calidad del sonido. Tanto como los instrumentos musicales y la voz humana producen sonidos que son ricos en vibraciones de diferente frecuencia y que son percibidos al mismo tiempo. La combinación de todas esas vibraciones, con diferentes amplitudes también, es lo que se percibe como timbre.

Una forma sencilla de comprender este concepto es imaginar una flauta y la voz de una persona emitiendo un tono de la misma frecuencia. Fácilmente puede notarse la diferencia entre ambos sonidos. Colectivamente, el cerebro escucha lo anterior como un cambio en el color del sonido.

El timbre es uno de los atributos del sonido más interesantes y complejos de comprender. Pero es en el timbre donde radica la naturaleza del diseño digital del sonido, pues tiene una gran facilidad para cambiar el color y el timbre del sonido rápidamente.

Para hacer lo anterior en el mundo acústico, se tendría que cambiar algo del instrumento para cambiar el timbre. Gracias al diseño digital del sonido, la mayor parte de la música que actualmente escuchamos ha pasado por un proceso digital de refinación.

- Percepción de duración

La duración no es un elemento fijo, sino algo que se percibe. Un ser humano puede escuchar un sonido, proveniente de cualquier lado, que dura unas pocas milésimas de segundo, o bien, varios minutos. Cuando se habla

de lo que se está experimentando, cualquiera puede decir que percibió un tiempo lento o un tiempo rápido.

Si se escucha una canción, ¿cómo se determina si la canción tiene un ritmo lento o rápido? Para ello, existe un punto estándar relativo, como referencia del tiempo, basado en los latidos del corazón.

Los latidos del corazón rondan entre los 60, 70, y a veces hasta 80 latidos por minuto si es nervioso. Un sonido se percibirá como de tiempo rápido si va más rápido que los latidos y de tiempo lento, si va más lento. Así que cuando se habla de tiempo rápido y de tiempo lento, el reloj interno del cuerpo es el encargado de proveer la información.

- Envolvente o articulación

Muestra la forma del sonido en el dominio del tiempo basada en la forma en que se interpreta un instrumento musical, incluyendo a la voz humana.

La articulación se refiere a lo que sucede en los primeros milisegundos de un sonido, y a la cantidad de energía que se aplica a la nota. Por ejemplo, con un violín se puede aplicar una gran cantidad de energía al tocar una nota, si se ejerce una presión inicial mayor con el arco sobre la cuerda. Luego, se libera la mayor parte de la presión haciendo que el sonido se estabilice hasta que suavemente desaparece.

Con un piano, la articulación es diferente, pues inicialmente se escucha un martillazo, y luego, la nota que suavemente desaparece. Así, el piano tiene una articulación percusiva, que gráficamente, representa un impulso inmediato. La envolvente es importante para el diseño digital del sonido. Pues conociendo las envolventes de cada instrumento, se pueden imitar los mismos. Y entonces, por medio de una computadora, se puede tener todos los instrumentos musicales.

- Difusión

Se refiere a capacidad del cerebro para localizar espacialmente la fuente de un sonido. El cerebro tiene la capacidad de procesar simultáneamente múltiples sonidos de diferentes fuentes y, ubicar espacialmente los mismos consciente o inconscientemente.

Por ejemplo, si se está en la calle hablando con un amigo, uno le pone mayor atención a la conversación. Cuando se llega a una intersección, inconscientemente se procesan sonidos de ciertos automóviles provenientes de ciertas direcciones. Y en ése momento, se está teniendo la conversación, se están ubicando las fuentes de sonido de los coches, y se está determinando si los mismos se están acercando o alejando.

La difusión es un atributo importante en el diseño digital del sonido, por ejemplo, cuando se está escuchando una película en el cine. La experiencia

auditiva resulta ser más real en el cine por que el sonido proviene de diferentes fuentes. Esto hace que una persona sienta que forma parte de la acción, y que sienta que forma parte de la película.
Aquí, el sonido es representado por una configuración de canales (2.1, 5.1, 7.1). Una configuración 5.1 implica la existencia de 6 fuentes de sonido; y una de 2.1, sólo 3 fuentes. Ésta última configuración es mejor conocida como sonido estéreo (altavoces izquierdo, derecho y subwoofer). [18]

3. OBJETIVOS

El objetivo de este Trabajo Fin de Grado es realizar una aplicación para dispositivos IPAD que permita disponer de un banco de diferentes algoritmos que modelen los principales efectos para guitarra eléctrica. Para tal fin, se deberá seleccionar el sistema hardware que permita la captura de audio procedente de la guitarra y su correcta manipulación en el dispositivo IPAD. Seguidamente, se implementará un banco de algoritmos que modelen conocidos efectos musicales donde el usuario podrá configurarlos de manera dinámica. Para finalizar, se desarrollará una aplicación para dispositivos IPAD para la utilización del software realizado.

Metodología a desarrollar:

1) Selección del sistema hardware guitarra-IPAD (utilización de la interfaz *"iRig"* para la interconexión de el IPAD con la guitarra).

2) Implementación del sistema de captura y manipulación de audio a través del sistema hardware

3) Implementación del conjunto de algoritmos que componen el banco de efectos musicales para guitarra eléctrica. Análisis, en lenguaje de programación MATLAB, de los algoritmos facilitados que se han considerado interesantes para la implementación de la aplicación. Posteriormente sobre este mismo lenguaje se realizará la adaptación para su ejecución en tiempo real.

4) Desarrollo de la aplicación para dispositivo IPAD: utilización de Xcode para el desarrollo y compilación de nuestra aplicación. Implementación, en lenguaje de programación C++, mediante la herramienta de programación "*openFrameworks",* un código abierto de C++, el cual es un conjunto de herramientas diseñadas para facilitar el proceso creativo, proporcionando un marco sencillo e intuitivo para la experimentación sobre IOS.

5) Evaluación: la selección de algoritmos será variada y cubrirá los distintos enfoques de efectos a utilizar para procesar audio. Los algoritmos seleccionados serán los elegidos para implementar, a la vez que se evaluaran los resultados obtenidos para ver si el efecto implementado se adapta a la realidad

4. ESTADO DEL ARTE

En este apartado se pretende que el usuario empiece a familiarizarse con los efectos antes de empezar a leer más sobre ellos. De tal manera que con una breve descripción pueda ser capaz de clasificarlos y ver sus diferencias características respecto a otros efectos.

También se detallaran en este apartado algunos conceptos que van ligados a la implementación de los efectos ya que algunos efectos para poder quedar terminados como tal necesitan alguna pequeña ayuda, modulación, filtrado, etc. Para poder obtener el efecto deseado.

4.1 Clasificación de los efectos

Un efecto de guitarra es un circuito analógico o digital que audiblemente modifica una señal de entrada producida por un instrumento para producir una salida deseada [16].

Hay innumerables tipos de efectos, algunos de ellos derivados de otros o de la combinación de otros tantos. [6]

Los efectos en general se pueden clasificar en categorías específicas [13]:

- Efectos basados en el rango dinámico
- Distorsiones
- Ecualizadores
- Efectos de Modulación
- Efectos de Transposición de tono
- Efectos de retrasos
- Simuladores
- Reductores de ruido

4.1.1. Efectos de rango dinámico

El rango dinámico de un equipo de sonido, es la diferencia existente entre los sonidos más potentes y menos potentes que el equipo es capaz de reproducir. Los efectos tales como compresor y expansor son efectos basados en el rango dinámico [6, 7].

4.1.2. Distorsiones

Cuando hablamos de la distorsión de un amplificador o una pedalera nos referimos a la distorsión armónica.
La “distorsión armónica” es un parámetro técnico utilizado para definir la señal de audio que sale de un sistema.
Dicha distorsión se produce cuando la señal de salida no equivale a la señal que entro en él. Esta falta de linealidad afecta a la forma de onda, porque el equipo introduce armónicos que no estaban en la señal de entrada. Puesto que son armónicos, es decir múltiplos de la señal de entrada esta distorsión no es tan disonante y a veces es más difícil de detectar. Efectos como

distorsiones pueden ser tales como el Fuzz, Bitcrusher, PhaserDistorsión, distorsión, así como otros tipos de distorsiones derivados es una distorsión simple, aunque este tipo de distorsiones suelen ser más específicas para el tipo de sonido que se quiera buscar [16].

4.1.3. Ecualizadores

Dentro de los ecualizadores tenemos dos tipos, ecualizadores activos y pasivos.

Los ecualizadores pasivos son los que se pueden encontrar en los amplificadores. Donde con pocos parámetros son capaces de tratar la señal.
Los graves dan cuerpo a la señal, es decir dan peso, para una guitarra los graves nos es una frecuencia que nos deba preocupar excepto si tenemos ausencia de una señal de bajo [11].

Los agudos nos dan brillo a la señal, es decir, cuanta más distorsión tengamos más destacaran los agudos. Por el contrario cuanta menos distorsión tengamos menos debemos de preocuparnos por los agudos [10].
Los medios es la frecuencia que más afecta al sonido de una guitarra. Para el sonido de una guitarra cuantos menos medios tengamos obtendremos un sonido más suave, más redondo, menos ruidoso, aunque se necesitaran más vatios en un equipo de sonido para poder oírlos con claridad.
Por el contrario si contamos con más medios el sonido sonará como más “duro” y algo más ronco a la vez que nos facilitara la mezcla de ese sonido con otros.

Efectos basados en ecualización los más conocidos son el “wha-wha” y su derivado el “auto-wha” [16].

4.1.4. Efectos de modulación

Modulación es un efecto no lineal en el cual varias señales interactúan unas con otras para producir nuevas señales con frecuencias que no estaban presentes en las señales originales.

Hay muchas formas de modulación, pero para este caso solo utilizaremos dos de ellas, la modulación de frecuencia y de amplitud.
La modulación de frecuencia (FM) es la variación en frecuencia de una señal, debido a la influencia de otra señal, generalmente de frecuencia más baja.
La modulación de amplitud (AM) es un tipo de modulación lineal que consiste en hacer variar la amplitud de la señal portadora de forma que esta cambie de acuerdo con las variaciones de nivel de la señal que contiene la información que se desea transmitir, llamada señal moduladora o modulante. [7]

También hay que indicar que los efectos de modulaciones hacen uso de un LFO, Low Frecuency Oscilator (Oscilador de baja frecuencia). Los LFO's son primariamente usados como moduladores para otras señales, y no para generar audio en sí [13].

En el mundo de los osciladores se considera "*baja frecuencia*" cualquier frecuencia por debajo de los 20 Hz, que es la frecuencia más baja que puede escuchar un humano. Aunque los LFO's no están destinados para ser usados como fuente de sonido, en algunos sintetizadores pueden operar a frecuencias más allá de los 20 Hz.

Efectos basados en modulación nos encontramos con efectos como el Chorus, el Flanger, el Phaser, el Tremolo o el Vibrato [9].

4.1.5. Efectos de transposición de tono

Para transponer el tono, la señal que entra en un equipo se muestrea y posteriormente se reproduce más deprisa para subir el tono o más despacio para bajarlo. Efectos basados en la transposición tenemos el Octavador que lo que hace es añadir a la señal octavas por arriba o por abajo. El Armonizador el cual añade a la señal su copia transpuesta o el Detuner el cual desafina la señal y la añade a la señal original [10].
Hay que indicar que estos efectos no son de los más utilizados debido a que se necesitan grandes niveles de composición y armonización para poder usarlos correctamente.

4.1.6. Efectos basados en retardos

Este tipo de efectos están basados en el uso de muestras y/o salidas pasadas para generar la salida en el momento actual.
Hay que señalar que muchos efectos aunque no se encuentre dentro de esta clasificación también usan retardos como puede ser el Chorus. Los efectos más puros basados en retardos son el Delay, el Eco, el Flanger y el Reverb o Reverberación [10].

4.1.7. Efectos simuladores

Los efectos basados en simuladores no son de gran interés, ya que simplemente emulan tipos de amplificadores, altavoces, tipos de guitarras o de pastillas. Son especialmente utilizados en pedaleras digitales. Estos efectos carecen de gran interés de ahí su poca utilización y venta de este tipo de pedaleras [13].

4.1.8. Reductores de ruido

Los efectos de reducción de ruido “cierran” el paso de toda señal que no supere un determinado umbral fijado por el usuario. Son muy útiles en situaciones de “directo” en las que hay multitud de micrófonos que pueden captar lo mismo que el principal, y tratamos de que la señal sólo entre por el principal. También nos ayudan a “recortar” todos aquellos ruidos no deseados que se han colado en una grabación (toses, respiraciones, rozamientos de ropas, ruidos de ambiente), siempre que no se mezclen con la señal principal. El efecto más conocido de este tipo es el Noise Gate o puerta de ruido, el cuale es muy utilizado para tocar en directos [13].

Rango Dinámico	Distorsiones	Efectos de Modulación	Efectos de Retardos	Reductores de Ruido
Compresor	Fuzz	Chorus	Delay	Noise Gate
Expansor	Distorsión	Flanger	Reverb	
		Tremolo		
		VIbrato		

Figura 2: Cuadro con los efectos a implementar en el TFG.

5. ANÁLISIS Y DESARROLLO DE EFECTOS

En este apartado se analizarán los efectos facilitados para la realización del proyecto y se detallará el desarrollo de los mismos.
Posteriormente se reflejará la implementación de los efectos facilitados y el diagrama de flujo de estos.
Finalmente se observarán los cambios realizados en los efectos originales para poder ejecutarlos en tiempo real, puesto que a la hora de transcribir este código a C++ será necesario anteriormente simular en MATLAB un entorno en tiempo real y comprobar el funcionamiento de todos los efectos. En la simulación en tiempo real, siempre debemos de tener en cuenta que desconocemos el futuro de la señal, para ello la captación de audio y todo el procesado se realiza en base a buffers de muestras.

En cada uno de los efectos facilitados se han realizado las modificaciones oportunas para poder simular esos efectos en un entorno de tiempo real.

El procedimiento a seguir en los efectos “sin memoria” (COMPRESOR, EXPANSOR, FUZZ, DISTORSION Y NOISEGATE) ha sido similar en los cinco.

En primer lugar se ha analizado cada efecto facilitado para comprender su funcionamiento, y posteriormente se ha pasado a simular un entorno en tiempo real en MATLAB. El efecto original no ha sido modificado drásticamente, pero si hay que tener en cuenta las modificaciones en la señal de entrada y salida, ya que aunque nosotros cargamos una señal de audio en MATLAB, simulamos una división de esta señal de entrada en buffers de 256 muestras, y posteriormente reconstruimos la señal.

En el caso de los 6 efectos restantes (FLANGER, CHORUS, TREMOLO, VIBRATO, REVERB Y DELAY) los cuales se pueden englobar en efectos “con memoria” (esto quiere decir que necesitaré acceder a muestras del pasado), se ha necesitado implementar un buffer de retardo que permita realizar el acceso a muestras pasadas tal y como se hace en el efecto original.

Así pues, estos últimos efectos si que han recibido una mayor modificación, además de que se han considerado varias opciones a la hora

de crear estos buffer que permiten acceder a muestras pasadas, de tal manera que se han considerado dos tipos de buffer:

- Buffer Lineal:

  El código utilizado para implementar el buffer en MATLAB, es el siguiente (anteriormente hemos inicializado el buffer en referencia al retraso requerido en el algoritmo):

```
bret(1:end-1)=bret:end);
bret(end)=senal_ent(j);
```

  El funcionamiento de este buffer, lo podemos explicar a través de el siguiente gráfico, imaginando que mi vector de retrasos tiene solo cuatro posiciones:

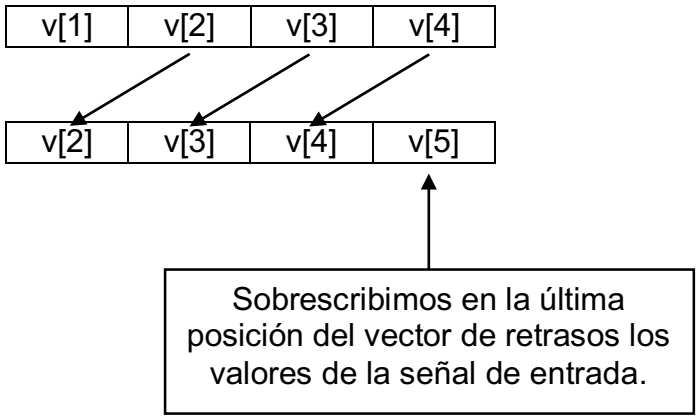


Figura 3.1: Funcionamiento del vector de retrasos lineal "bret".

Así que como podemos observar si deseamos acceder a posiciones del pasado, en nuestro caso, accederíamos a las posiciones de "bret" que me permitirán reflejar el retraso en los algoritmos que necesiten de este fenómeno acústico.

Cabe destacar que este buffer se ha estado utlizando en varios de los efectos denominado "con memoria", y su funcionamiento en MATLAB ha sido óptimo.
Como desventaja hay que decir que este buffer no ha podido ser implementado posteriormente en C++, puesto que se podría producir un colapso de la CPU debido a lo sobrescritura, de ahí la necesidad de trabajar con bufferes circulares.

- Buffer Circular:

El funcionamiento de este buffer, lo podemos explicar a través de el siguiente gráfico, imaginando que mi vector de retrasos tiene solo cuatro posiciones:

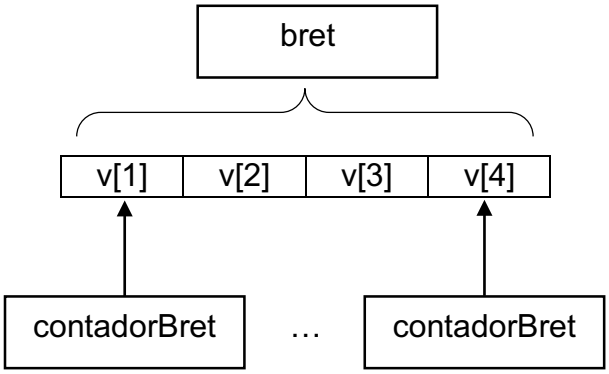


Figura 3.2: Funcionamiento del vector de retrasos circular "bret".

El funcionamiento del buffer circular es mucho mas sencillo, puesto que el buffer de retrasos se va llenando de información de la señal de entrada, hasta que llegue al final del vector de retrasos.

Al principio de la ejecución *contadorBret* se inicializa a 0, por lo tanto el contador estará en la primera posición de mi vector. El contador se irá incrementando en cada llamada a la función. El vector se irá rellenando de los valores que tome la señal de entrada (*bret(contadorBret) = seña_entrada(j)*) hasta que *contadorBret* llegue a la ultima posición de mi vector de retrasos, cuando alcance la última posición volverá a inicializarse a 0. Por tanto *contadorBret* me irá diciendo por que posición de mi vector de retrasos voy.

Para acceder al vector circular de retrasos, siempre tengo que tener en cuenta "por donde voy", por ello la necesidad del contador, el cual me facilitará el acceder a la posición adecuada para realizar el retraso.

Cabe destacar que este es el buffer utilizado en C++ para la realización de los efectos "con memoria". La necesidad de utilizar este buffer y no el anterior citado es debido a que este último no utiliza tantos recursos en la memoria de mi dispositivo, ya que en efectos en los que tenga que reservar bastante memoria será muy costoso computacionalmente utilizar un buffer lineal, de ahí la necesidad y la prioridad de usar este buffer.

De esta manera, y tras todas estas modificaciones, conseguimos simular el efecto en tiempo real, y esto nos facilitará el paso de un lenguaje a otro, en nuestro caso desde MATLAB a C++, teniendo en cuenta las librerías y el funcionamiento de openFrameworks que en sus funciones de introducción y salida de audio utilizan buffers cuyo tamaño puede ser definido y que en nuestro caso son de 256 muestras para la ejecución en tiempo real, de ahí el hecho de que simulemos en MATLAB la división de la señal de entrada cada 256 valores.

También tendremos en cuenta que tanto en lenguaje de programación MATLAB como en C++ siempre tendremos presente la condición de transformación de ESTEREO a MONO. Es decir, si mi señal esta en ESTEREO, será necesaria su transformación en MONO para aplicar el efecto. Posteriormente la señal procesada en MONO se volverá a transformar en ESTEREO para devolver la señal en dos canales, obteniendo así la señal procesada.

En cada efecto se detallará el código facilitado, el cual es el original, y posteriormente se detallará el código modificado en lenguaje de programación MATLAB, desde el cual realizamos la transcripción a C++.

Las diferencias a la hora de transcribir cada uno de los efectos modificados serán muy pequeñas, por ello la importancia de la simulación y modificación previa en MATLAB.

5.1 Compresor

5.1.1. Introducción

Como el nombre indica, la compresión reduce el rango dinámico de una señal. Se utiliza extensivamente en la grabación de audio, trabajo de producción, reducción de ruido, y en aplicaciones de actuaciones en vivo, pero debe ser utilizada con cuidado.
Suele ser un efecto interesante sobre todo con sonidos limpios jazz o en alguna ocasión una guitarra en limpio, puede resultar chillona y molesta así se corrige ese error [6].

5.1.2. Implementación

Un compresor es básicamente un dispositivo de ganancia variable, donde la cantidad de ganancia utilizada depende del nivel de la entrada. En este caso, la ganancia se reducirá cuando el nivel de la señal es alta lo que hará más fuerte pasajes suaves, reduciendo el rango dinámico [16].

El esquema de un compresor se muestra en la Figura 4:

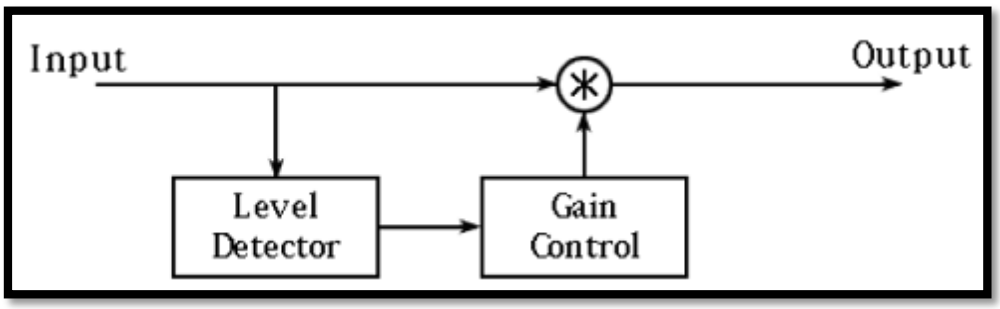


Figura 4: Esquema básico de un compresor. [6]

La relación entrada / salida de un compresor a menudo es descrita por un simple gráfico donde el eje horizontal corresponde a la señal de entrada, y el eje vertical es el nivel de salida (ambos medidos en decibelios). Una línea a 45 grados corresponde a una ganancia de uno, cualquier nivel de entrada se asigna a exactamente el mismo nivel de salida. El compresor cambia la pendiente (hace que sea más horizontal) de la línea por encima de un valor umbral.

En la figura 5 se muestra la relación entre la entrada y la salida.

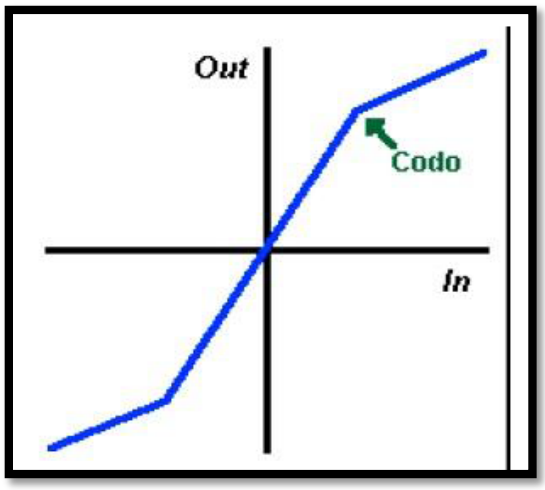


Figura 5: Relación entrada salida del compresor o función de transferencia. [6]

Para su implementación solamente deberemos contemplar dos parámetros básico como el umbral y la nueva pendiente una vez superado el umbral, como se mostrara en las ecuaciones siguientes basta simplemente con realizar un plan de comprobaciones y en función del resultado de las misma o no se realizara nada o se efectuara un productor y dos sumas correspondientes de la resta.

Las ecuaciones que corresponde a un compresor son las siguientes:

Si señal_de_entrada(n)> + umbral

Señal_de_salida(n)=+umbral+pendiente[señal_de_entrada(n)-(+umbral)]*

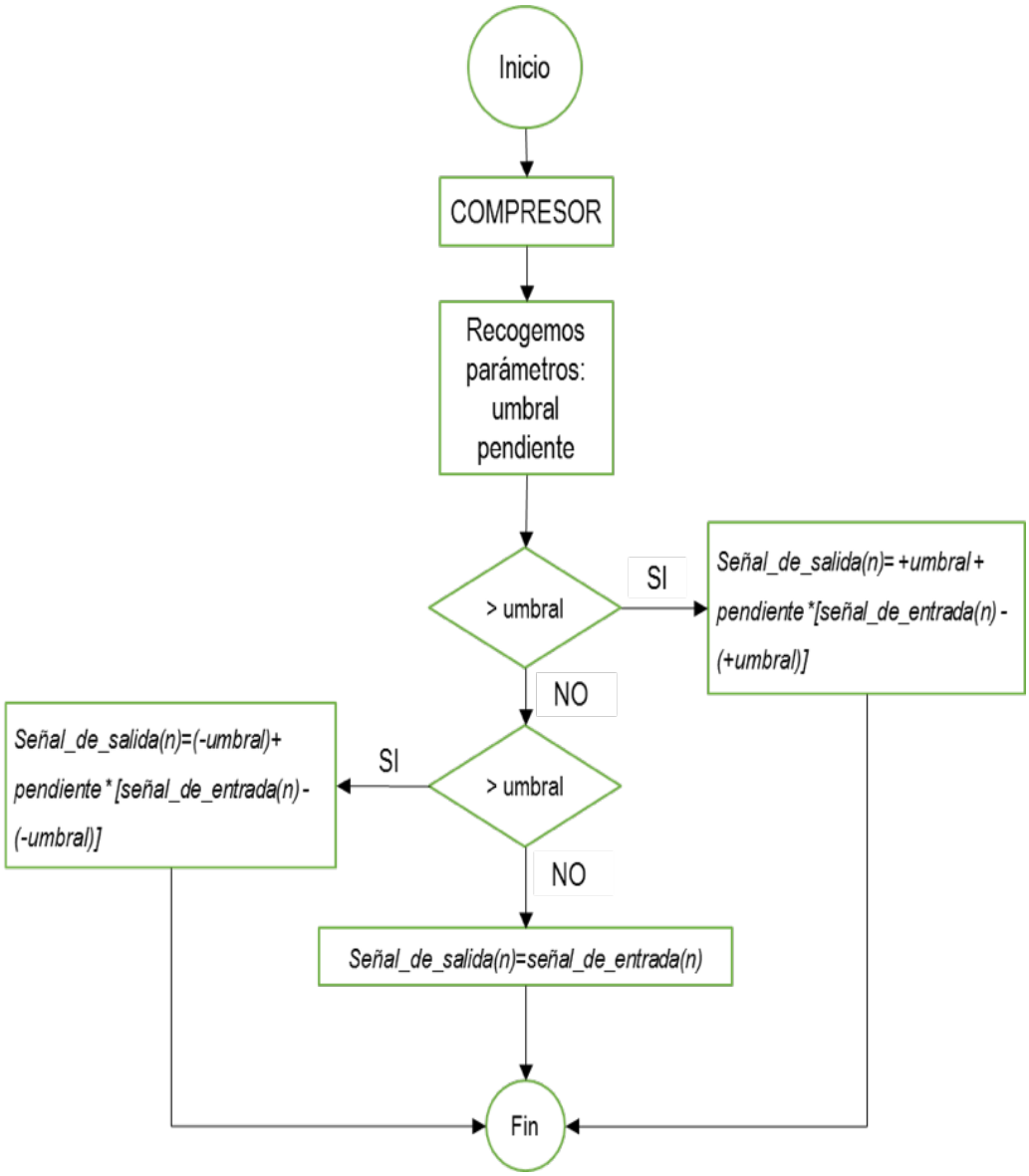
Si señal_de_entrada(n) < -umbral

Señal_de_salida(n)=(-umbral)+pendiente[señal_de_entrada(n)-(-umbral)]*

Si no se aplicó ninguna de las anteriores

Señal_de_salida(n)=señal_de_entrada(n)

Donde en las ecuaciones anteriores pendiente es la nueva pendiente que tomara la señal y dicho valor estará limitado a 1. Y el umbral de la señal será el umbral introducido convertido a un tanto por cierto del valor absoluto del máximo de nuestra señal [16].

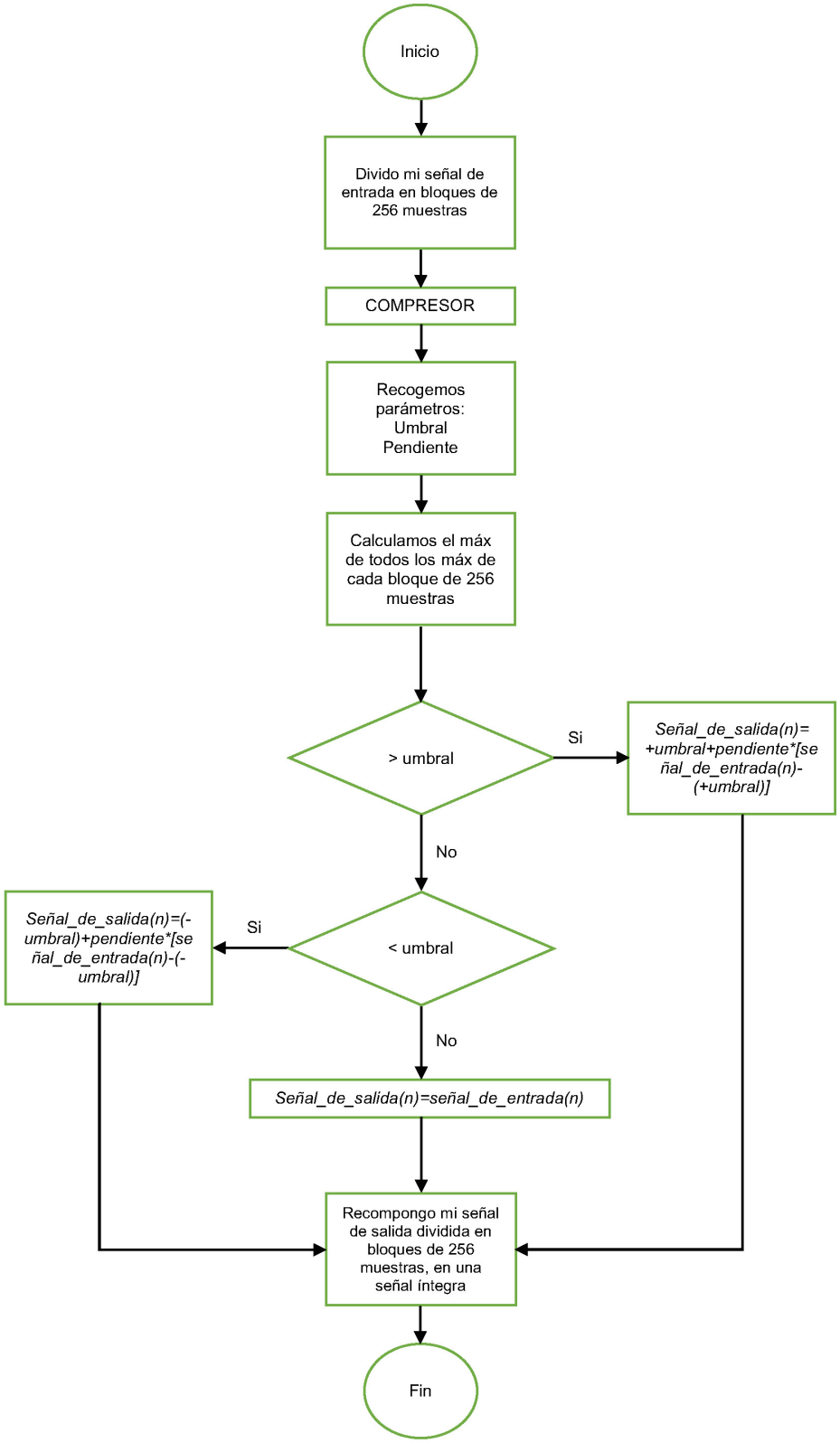


Con lo que los únicos parámetros a manejar por el usuario serán:

- Umbral
- Pendiente

5.1.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto COMPRESOR para su adaptación a un entorno de tiempo real:



En estos efectos (FUZZ, EXPANSOR, COMPRESOR, DISTORSION y NOISE GATE) si que debemos de destacar la presencia de una variable que me permite calcular el máximo de cada uno de los bloques de 256 muestras, para posteriormente buscar el máximo absoluto de todos los bloques anteriores que conforman mi señal, dicho de otra manera, calculo el máximo de los máximos de toda la señal dividida en bloques (habrá un bloque que tendrá el absoluto), obteniendo así el valor deseado.

5.2 Expansor

5.2.1. Introducción

Este efecto es otro efecto más basado en el rango dinámico. Dicho efecto es totalmente el efecto contrario al compresor, pues bien como su nombre indica expande el rango dinámico de la señal, de manera que las señales débiles son atenuadas mientras que las más fuertes no se ven afectadas apenas. Dicho efecto suele ser muy útil en la restauración de viejas grabaciones [6].

5.2.2. Implementación

Un expansor es básicamente un dispositivo de ganancia variable, donde la cantidad de ganancia utilizada depende del nivel de la entrada. Dicha ganancia nunca será mayor que uno, en este caso cuando el nivel de la señal de entrada es alto el expansor presentara una ganancia unidad, y cuando el nivel de esta decaiga la ganancia decrecerá haciendo a la señal aparecer atenuada a la salida [16].

El diagrama de un expansor es el mostrado en la figura 6.

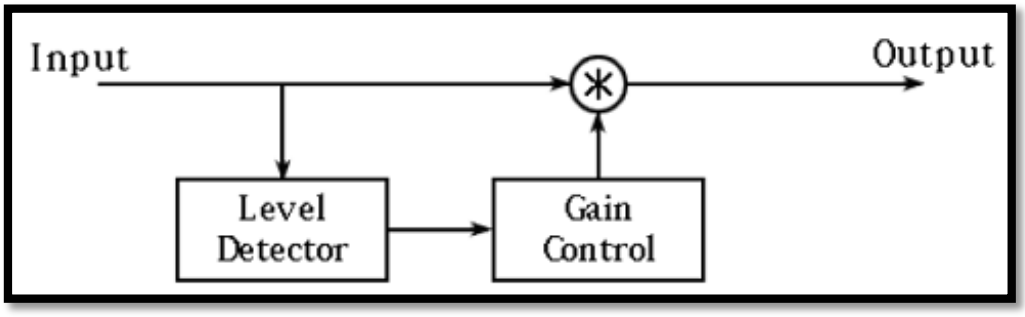


Figura 6: Esquema básico de un expansor.[6]

Como podemos apreciar el esquema es igual que el de un compresor con la única diferencia es el funcionamiento de forma análoga.
Como se puede observar en la figura 10, donde es mostrada la función de transferencia de un expansor la pendiente de 45º corresponde a una ganancia unidad, así todo lo que esté por debajo del umbral marcado

el expansor cambiara la pendiente haciéndola menor que uno y expandiendo así los sonidos, de ahí que el decremento de la ganancia expanda el rango dinámico de la señal.

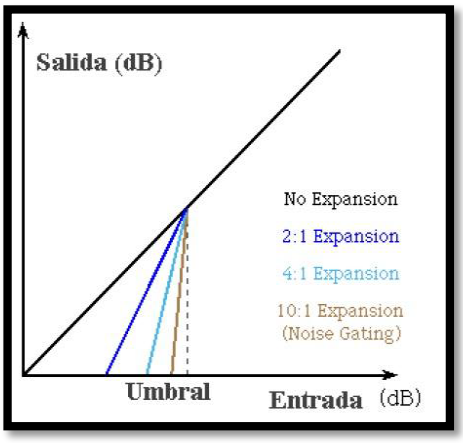


Figura 7: Función de transferencia del expansor. [6]

Como se puede observar en la función de transferencia, el expansor funciona de igual forma pero análoga al compresor, donde en este caso la expansión se produce por debajo del umbral no por encima y en vez de comprimirla las expande.

Para su implementación los únicos parámetros de control son el umbral, el cual es indicado por el usuario a partir de un porcentaje, el cual será el valor absoluto del valor máximo de nuestra señal, aunque esto es totalmente transparente para el usuario. Con lo que este parámetro estará comprendido entre 0 y 1 pero perfectamente adaptado. El otro parámetro que maneja la señal será la nueva pendiente que tendrá la señal si esta entre el rango de los umbrales, donde esta estará comprendida como una pendiente entre 0 y 1.

Una vez conocidos los parámetros de umbral y pendiente, el expansor se define digitalmente en su modelo más simple teniendo solamente en cuenta estos dos parámetros [16]:

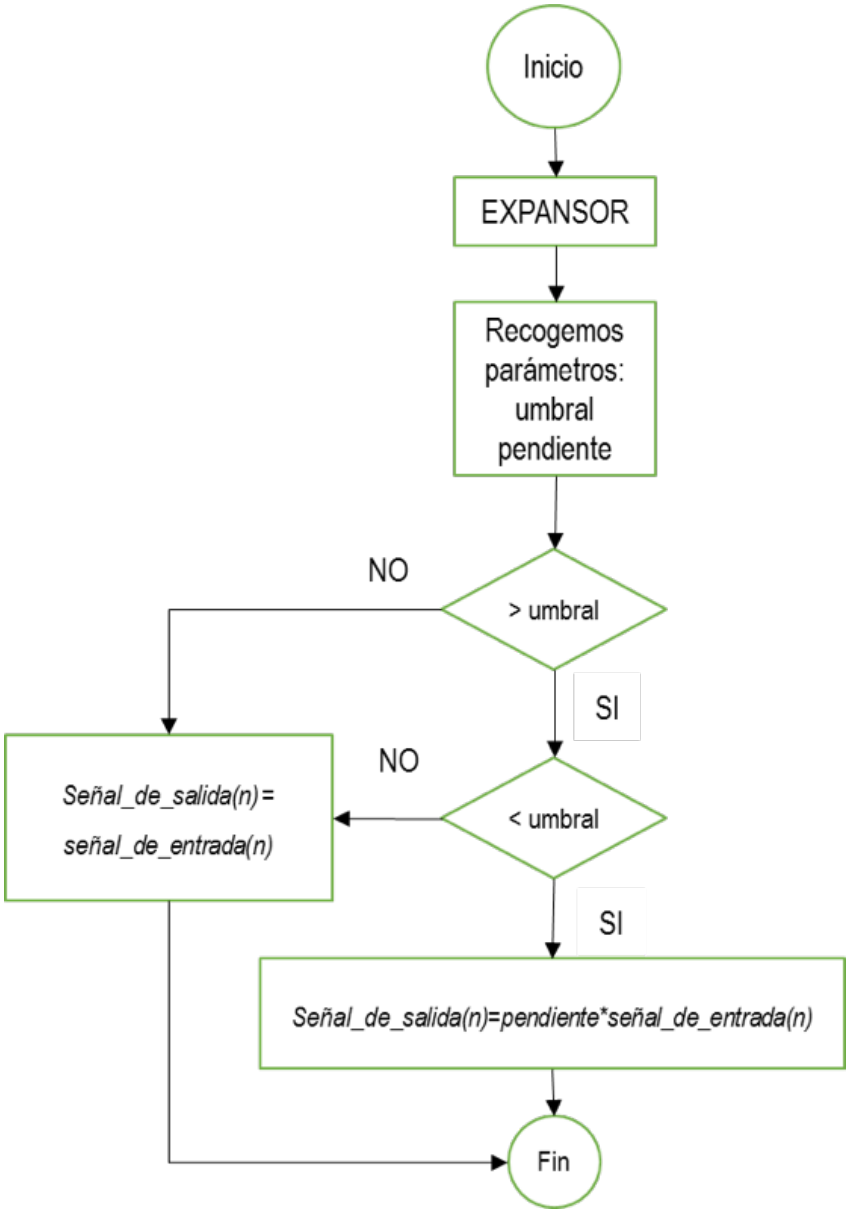
Si señal_de_entrada(n) < + umbral && señal_de_entrada(n)> -umbral.

*Señal_de_salida(n)=pendiente*señal_de_entrada(n)*

Si no se cumplió la condición anterior

Señal_de_salida(n)=señal_de_entrada(n)

Una vez conocidos los parámetros de umbral y pendiente, el expansor se define digitalmente en su modelo más simple teniendo solamente en cuenta estos dos parámetros [16]:

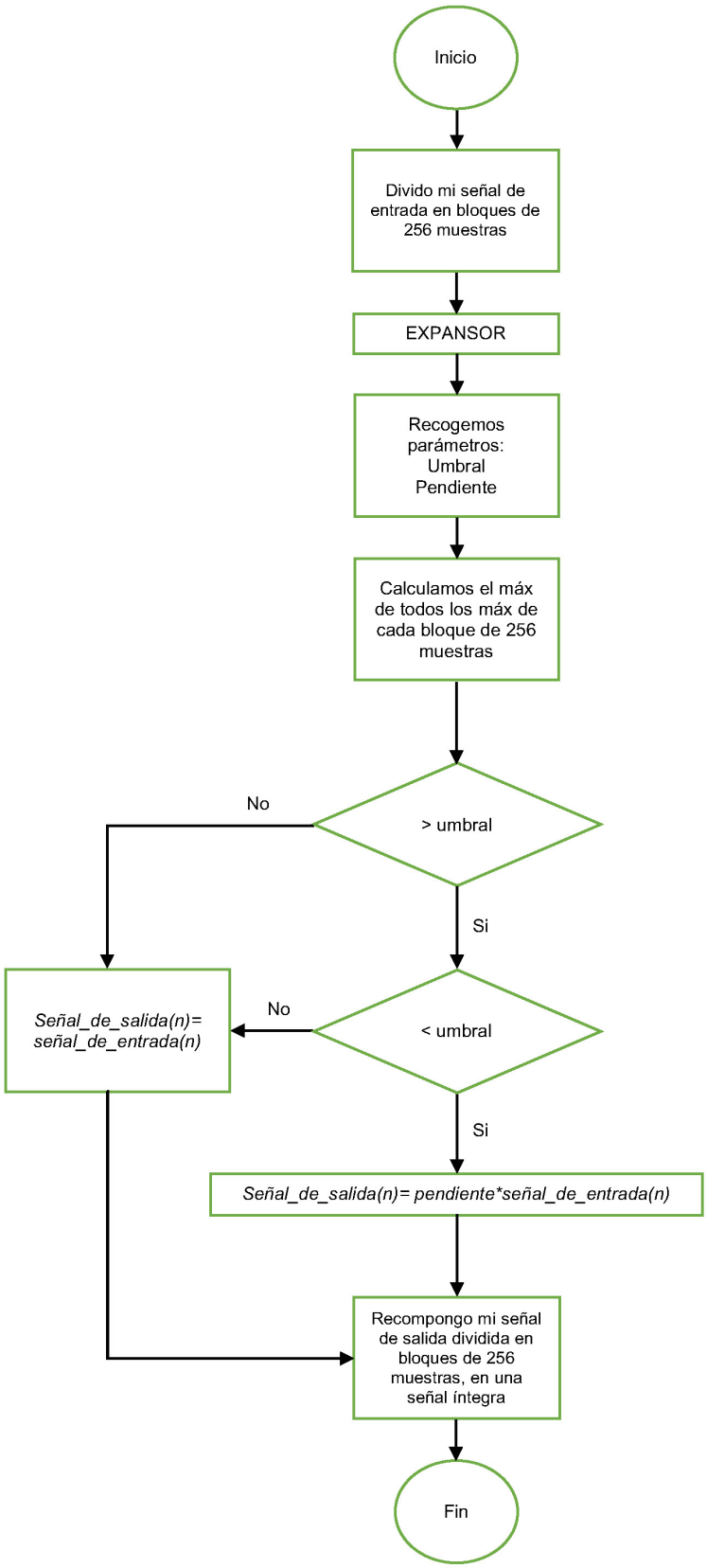


Con lo que los únicos parámetros a manejar por el usuario serán:

- Umbral
- Pendiente

5.2.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto EXPANSOR para su adaptación a un entorno de tiempo real:



En estos efectos (FUZZ, EXPANSOR, COMPRESOR, DISTORSION y NOISE GATE) si que debemos de destacar la presencia de una variable que me permite calcular el máximo de cada uno de los bloques de 256 muestras, para posteriormente buscar el máximo absoluto de todos los bloques que conforman mi señal, dicho de otra manera, calculo el máximo de los máximos de cada bloque, obteniendo así el valor deseado.

5.3 Fuzz

5.3.1. Introducción

Una guitarra eléctrica distorsionada es esencial en la música rock. El Fuzz es un efecto clave en este sentido.

El Fuzz fue uno de los primeros efectos en aparecer, usado por músicos como Eric Clapton o Jimmy Hendrix.
Dicho efecto trabaja sobre la ganancia de la señal y se caracteriza por tener un sonido suave y un poco más áspero en comparación con una distorsión, es muy similar al sonido que pude producir una guitarra eléctrica sonando por un amplificador saturado con los altavoces rotos [8].

5.3.2. Implementación

La implementación de este efecto se basa simplemente en recortes a partir de un umbral indicado. Consiguiendo así convertir una señal de entrada sinusoidal en una forma de onda mucho más parecido a una onda cuadrada creando así un efecto áspero en el sonido, creando un efecto Fuzz clásico. Dándole a la canción un sonido más sintético que una distorsión normal.

La amplitud de la señal de entrada se recorta por arriba y por debajo a partir de cierto umbral establecido, y posteriormente automáticamente será amplificada para compensar cualquier perdida de la señal. Dicho umbral de la señal estará comprendido entre 0-1 al ser introducido por el usuario, pero realmente representa un porcentaje de valor absoluto del máximo de nuestra señal.

La figura 8 muestra el esquema de un efecto Fuzz.

La amplitud la cual se amplificara la señal vendrá dada como la inversa de nuestro umbral menos uno. Asegurando así que nuestra amplificación estará comprendida entre 0.99 y -0.99. Quedando así como únicos parámetros a introducir por el usuario el umbral de la señal [16].

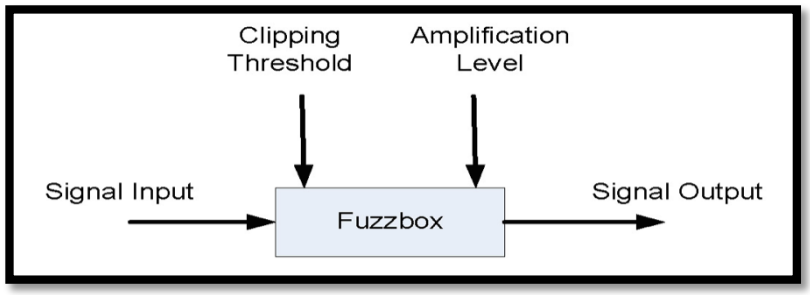


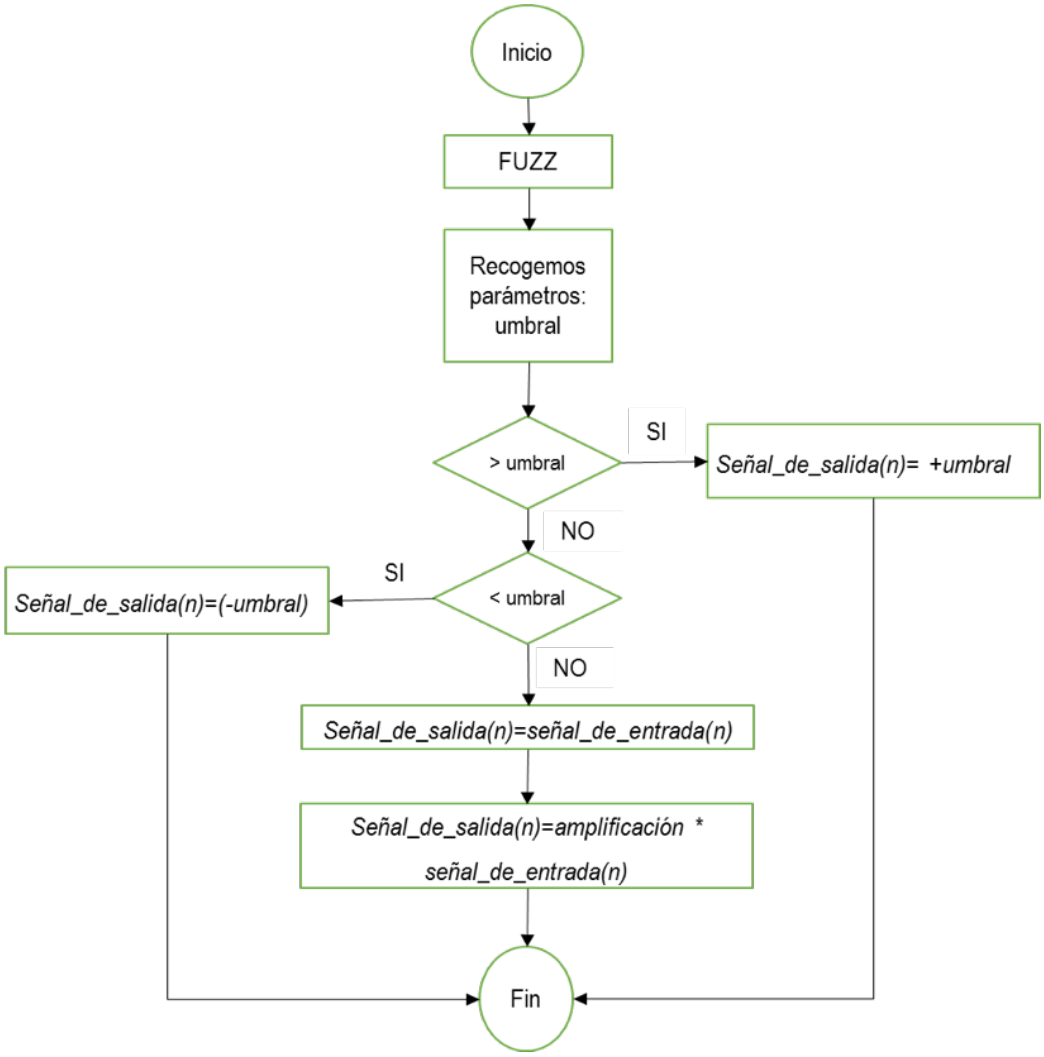
Figura 8. Diagrama de un efecto Fuzz [7].

Con lo que las ecuaciones que definen nuestro efecto Fuzz serán [16]:

Si señal_de_entrada(n) > + umbral
señal_de_salida(n) =+ umbral.
Si señal_de_entrada(n) < -umbral
señal_de_salida(n) =-umbral.

Tras esto deberemos aplicar nuestra amplificación a la señal [16].

*señal_de_salida(n) = amplificación * señal_de_salida(n).*

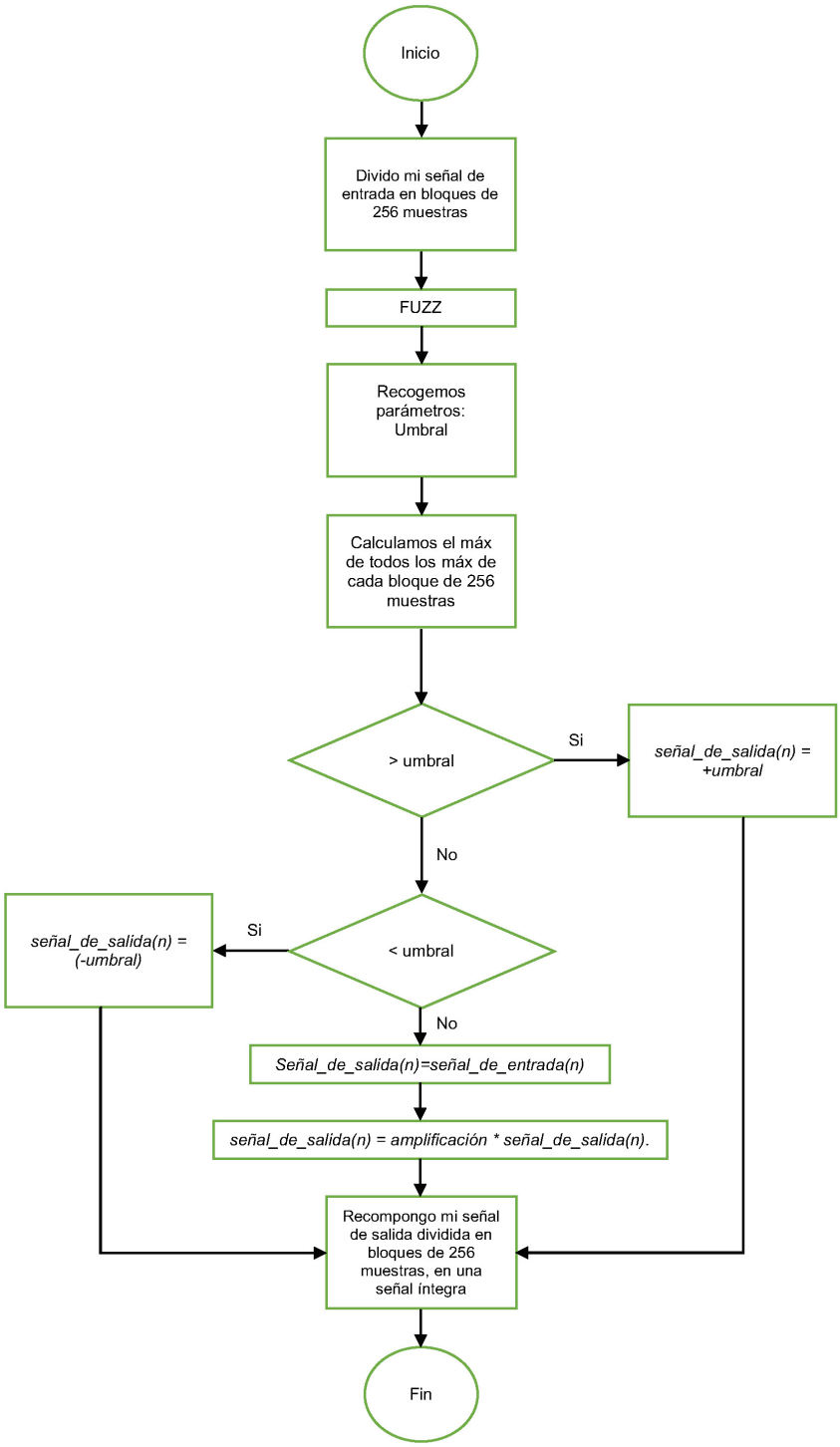


Con lo que los únicos parámetros a manejar por el usuario serán:

- Umbral

5.3.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto FUZZ para su adaptación a un entorno de tiempo real:



En estos efectos (FUZZ, EXPANSOR, COMPRESOR, DISTORSION y NOISE GATE) si que debemos de destacar la presencia de una variable que me permite calcular el máximo de cada uno de los bloques de 256 muestras, para posteriormente buscar el máximo absoluto de todos los bloques que conforman mi señal, dicho de otra manera, calculo el máximo de los máximos de cada bloque, obteniendo así el valor deseado.

5.4 Distorsión

5.4.1. Introducción

La distorsión es uno de los efectos más populares y más conocidos así como uno de los primeros en aparecer. Se entiende por distorsión como concepto general a la modificación de un sonido general, de esta manera cualquier efecto de audio se puede calificar como distorsión. Pero centrándonos más en la distorsión como efecto, la distorsión es aquel efecto que modifica la amplitud de un sonido sin el uso de ninguna información temporal. El efecto Fuzz es otro ejemplo de distorsión pero mucho más leve. Esto es debido a que la distorsión modifica da muestra de manera independiente a las demás. [6]

La distorsión es usada para simular el sonido de un amplificador que está saturado.

5.4.2. Implementación

La distorsión puede ser implementada básicamente con un par de comparaciones respecto al umbral. Es decir, con +umbral y –umbral y si se cumplen las condición convertir inmediatamente la señal al nivel de saturación indicado. Si dicho umbral no es superado la señal no se modificara, esto es un modelo muy sencillo prácticamente similar al efecto Fuzz. La siguiente figura muestra un ejemplo sencillo de distorsión con un umbral=10 y una saturación=20 [16].

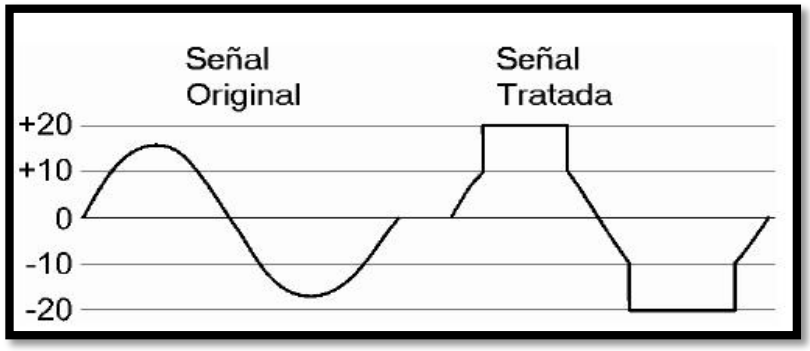


Figura 9. Distorsión Básica [6].

Sin embargo por su similitud con el efecto Fuzz se decidió implementar un modelo más completo que el anterior el cual consiste en tomar un porcentaje de señal de la señal original de entrada, el cual está adaptado para que coja el porcentaje del valor absoluto del valor máximo de nuestra señal. Todo esto es sumado con otro porcentaje del nivel de saturación elegido para generar el efecto. Esto nos mantiene que la señal permanezca igual que la original si no se supera el umbral indicado.

Cuando la señal supera el umbral es tratada según las ecuaciones:

Si señal_de_entrada(n) > + umbral
señal_de_salida(n) = a señal_de_entrada(n) + b*sat*
Si señal_de_entrada(n) < -umbral
señal_de_salida(n) = a señal_de_entrada(n) + b*(-sat)*

Si no se aplicó ningún caso anterior:
señal_de_salida(n) = señal_de_entrada(n)

Para que no haya lugar a confusión la variable “a” corresponde con el tanto por ciento de señal que queremos mezclar, mientras que “b” se corresponde con el tanto por cierto del nivel de saturación que queremos añadir a la señal y la variable “sat” será el nivel de saturación escalado a 1. Dichos parámetros deberán de ser introducidos manualmente por el usuario [16].

La siguiente figura muestra el ejemplo de un distorsionador si se superara el umbral:

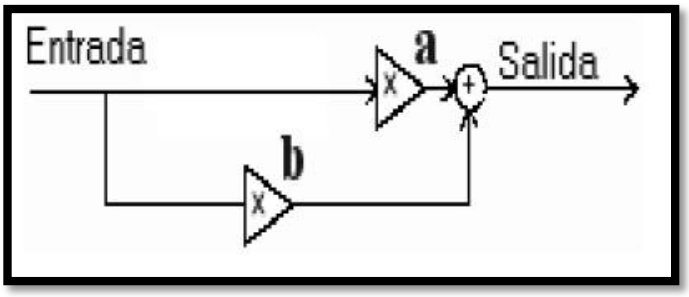
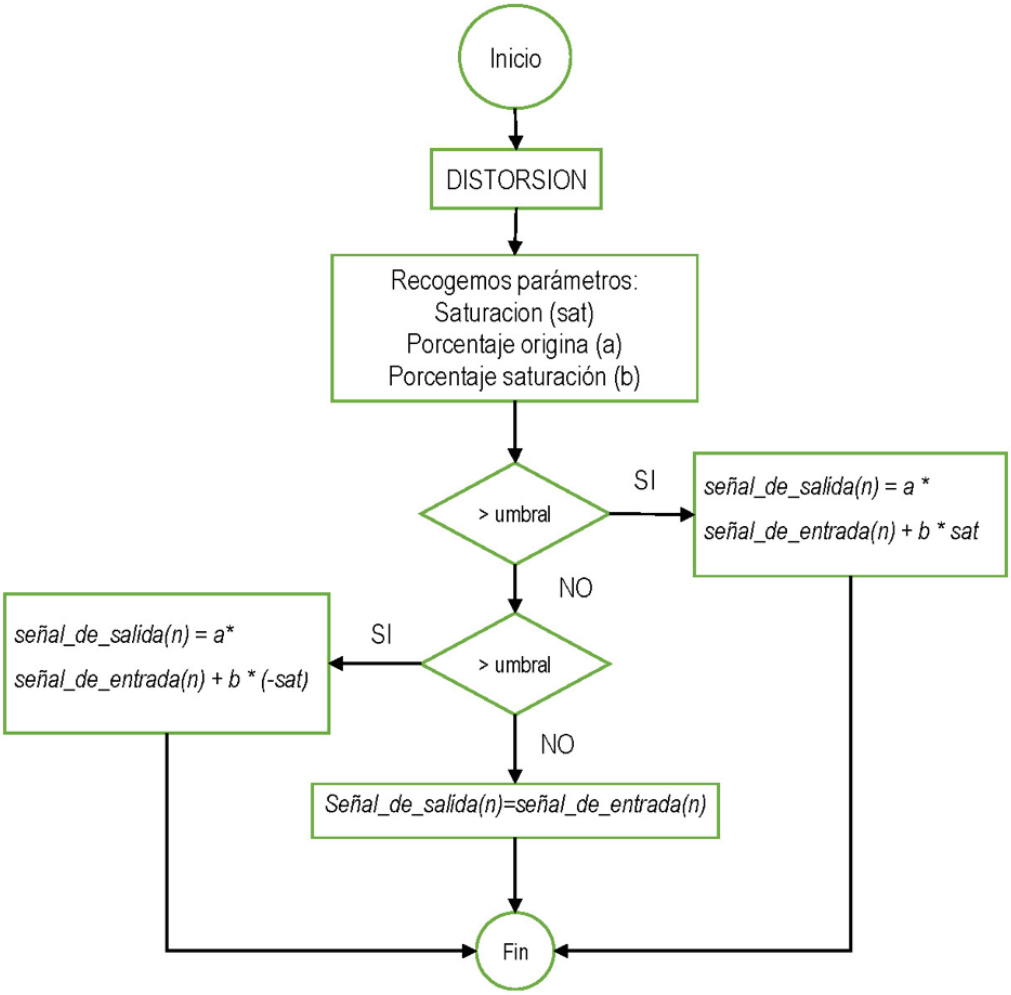


Figura 10. Distorsionador si se supera el umbral [6].

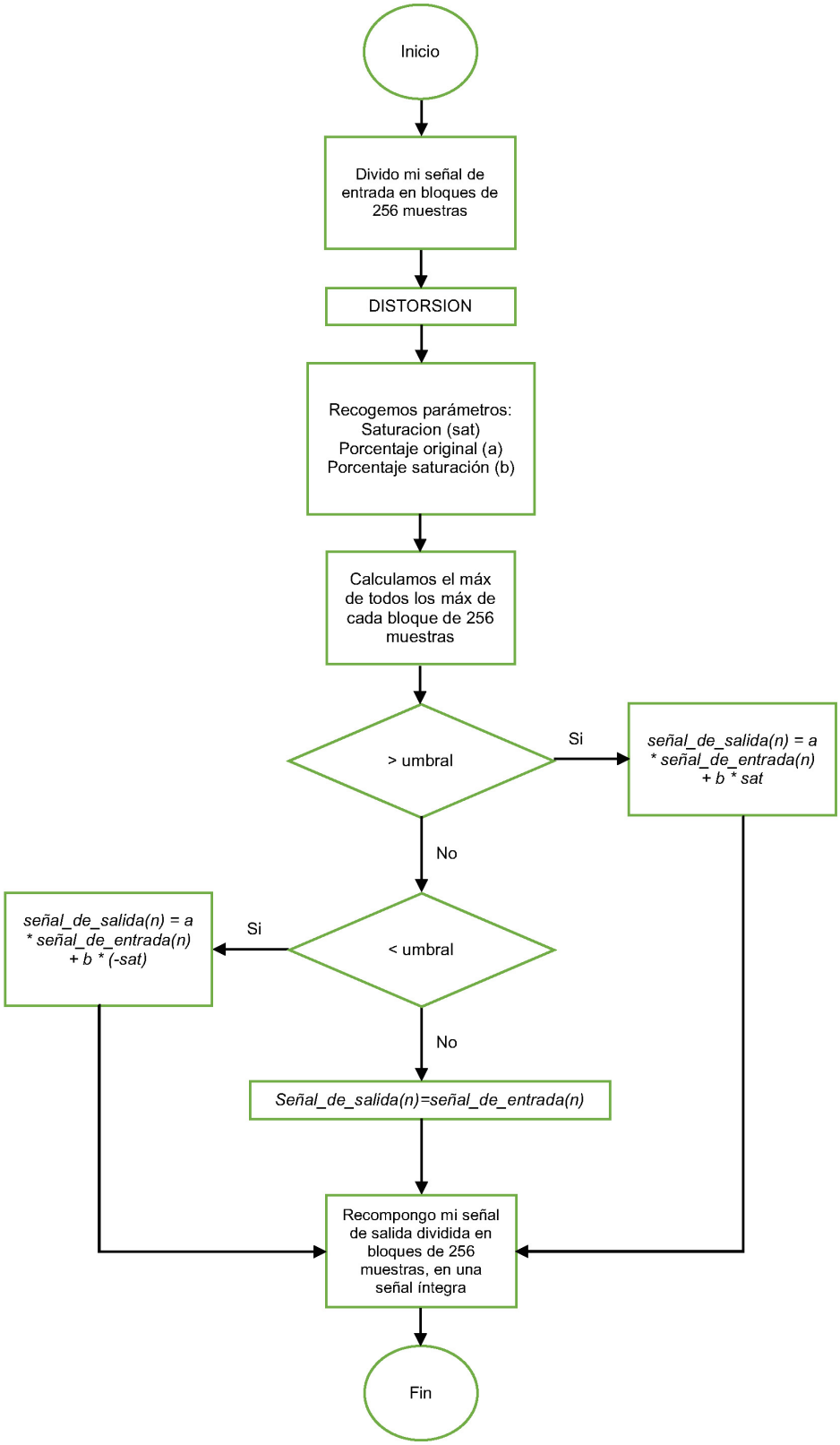


Con lo que los únicos parámetros a manejar por el usuario serán:

- Saturación o umbral (sat)
- Cantidad de señal original(a).
- Cantidad de saturación(b).

5.4.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto DISTORSION para su adaptación a un entorno de tiempo real:



En estos efectos (FUZZ, EXPANSOR, COMPRESOR, DISTORSION y NOISE GATE) si que debemos de destacar la presencia de una variable que me permite calcular el máximo de cada uno de los bloques de 256 muestras, para posteriormente buscar el máximo absoluto de todos los bloques que conforman mi señal, dicho de otra manera, calculo el máximo de los máximos de cada bloque, obteniendo así el valor deseado.

5.5 Chorus

5.5.1. Introducción

El efecto chorus consigue hacer que un único instrumento suene como si hubiera varios instrumentos sonado al unísono, añadiendo algo de grosor al sonido.
Suele ser un efecto muy interesante para usar con sus controles al mínimo. De esta forma genera un movimiento casi imperceptible pero muy interesante para usar en arpegios y rítmicas [6].
Lo que básicamente hace un chorus es retrasar la señal original. El tiempo de retardo se modula con un LFO. La señal retrasada y modulada se mezcla con la señal original, seca [9].

5.5.2. Implementación

Para la implementación nos basamos prácticamente, en la teoría de que cuando dos músicos tocan sus instrumentos juntos realmente no lo hacen perfectamente sincronizados, esto hace que al sumarse los sonidos de ambos instrumentos se detecten algunos retardos entre un instrumento y otro.

De esta forma un simple retraso puede implementarse como un efecto Delay más básico conocido, pero aparte el chorus deberá crear un pequeño efecto de desafinación para que pueda haber diferencia en el tono del instrumento. Para ello un simple Delay será convertido en una línea de retraso variable, haciendo así que de esta forma el retraso cambiará con el tiempo para crear una modulación de tono.

Para comprender como el tono es modificado, pensemos que el bloque de delay es un dispositivo de grabación. Éste almacena una copia exacta de la señal de entrada conforme llega, como si fuera un casette y después lleva a la salida la misma copia un tiempo después. Para incrementar la cantidad de retraso, se precisa un segmento mayor de la señal que esté almacenada en la unidad antes de que se reproduzca. Para lograr esto, leemos la señal retrasada de la línea de la línea de retraso a una tasa menor que la de escritura (la tasa de escritura-grabación no se altera). Así, leer a una tasa menor es como arrastrar los dedos en las ruedas del casette, que como se sabe, produce una bajada del tono.

Por otro lado, para reducir el tiempo de delay, podemos leer a una tasa mayor que la escritura, efecto similar al de aumentar la velocidad del casette, cuyo resultado es un aumento del tono en la grabación.

En la siguiente figura se puede observar un Chorus básico donde tiene la misma estructura que un efecto Delay, sin realimentación y con un tiempo de retraso típicamente entre 20-30 milisegundos para poder logar la superposición del sonido consigo mismo. Como el efecto Delay no consigue las variaciones de tono que el efecto requiere es necesario añadir un segundo elemento.

Este segundo elemento es el encargado de variar el tiempo de retardo para conseguir la modulación del tono. En general cualquier forma de onda periódica se podría usar, siendo ejemplo de ello una sinusoidal que es la que se utilizó en nuestro caso. Esta forma de onda debe cambiar lentamente (3 Hz o inferior) y es lo que ya se definió como LFO (Oscilador de baja frecuencia) [16].

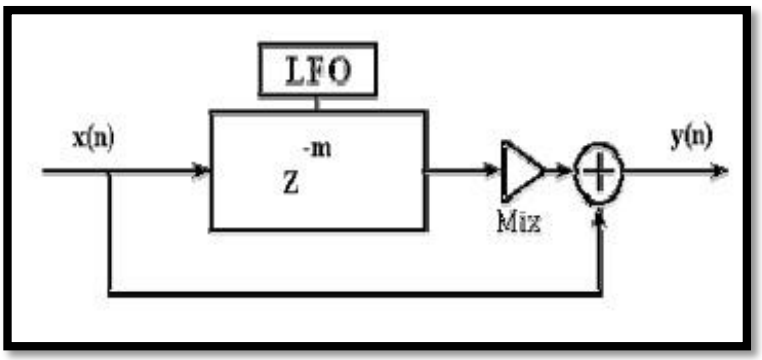


Figura 11. Modelo básico del efecto Chorus [8].

Cabe indicar que para mayor complejidad se decidió implementar un Chorus más complejo se decidió implementar un chorus multi-voz, donde a nivel de funcionamiento es exactamente igual con la diferencia de llevar añadida una copia más de la señal.

La figura 12 muestra el ejemplo del chorus que se implementó, donde a pesar de no venir indicado en el diagrama las dos voces que son retrasadas copia de la señal original vienen también controladas por el mismo LFO [16].

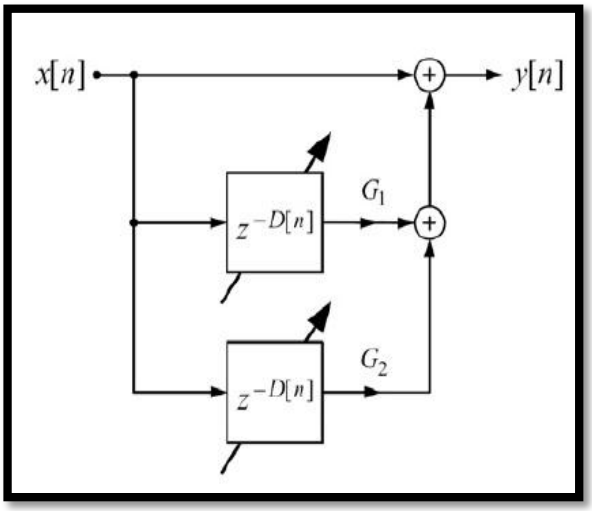


Figura 12. Diagrama de bloque de un chorus multi-voz [8].

Como se puede apreciar en los diagramas de bloques a pesar de haber cierta diferencia representan lo mismo, fíjese en la figura 19 el retraso viene dado en función de “m”, esto no es más que el número de muestras que debemos retrasar nuestra señal, para calcularlo deberemos atender al tiempo de retraso (ms) introducido por el usuario y multiplicarlo por la frecuencia de muestreo de nuestra señal, obteniendo así que “$m=t_{retraso}*fs$”.

Debemos señalar que para el numero de retrasos de la segunda voz el proceso será totalmente idéntico al de la primera pero en este caso el número de retrasos será multiplicado por un factor aleatoria entre 0-1, haciendo así que se note una pequeña diferencia en la segunda voz copiada respecto de la primera.

Como podemos observar en la figura 12 nuestras señales están retrasadas mediante una función, esta no es más que la función obtenida mediante el LFO, definiéndose la función como:

$$D[n] = \frac{m}{2} * (1 - \cos(2\pi * Fd * n))$$

Dicha función es la que nos proporcionara la modulación de tono, debido a que está controlada con un coseno, en su valor máximo el coseno será igual a la unidad, con lo que para ese intervalo de tiempo no se producirá ningún retraso a nuestro señal mientras que el nivel de retraso será máximo cuando el valor del coseno sea los más cercano a su valor nulo [16].

Por otro lado la frecuencia que controla al coseno no es más que una frecuencia obtenida entre nuestra “fs” y la frecuencia de nuestro LFO, definiéndose Fd como:

$$Fd = \frac{2\pi}{\frac{Fs}{FLFO}}$$

De esta manera queda definida la frecuencia para nuestras señales desfasadoras, siendo FLFO la frecuencia de nuestro LFO, hay que indicar que para la frecuencia de la segunda señal desfasadora la frecuencia del LFO deberá de ir multiplicada por nuestro factor aleatorio produciendo así una pequeña diferencia entre nuestras funciones.
Por otro lado para no dejar ningún detalle de lado si nos volvemos a fijar en la figura 20 donde se nos muestra G1 y G2, no son más que una ganancia de amplificación para nuestras voces duplicadas la cual estará comprendida entre 0-1.

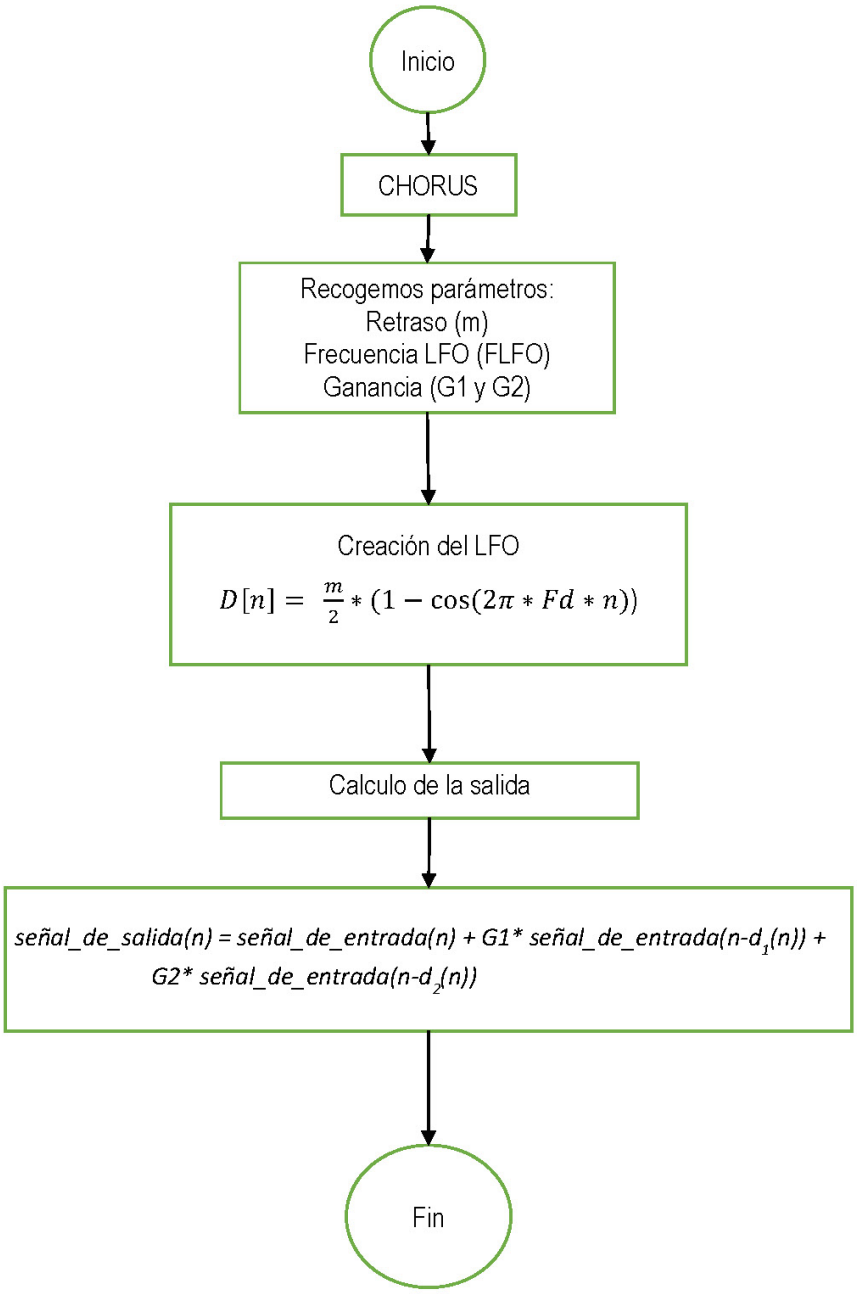
Una vez conocido esto se puede mostrar la señal que define completamente el efecto chorus, la cual queda definida como:

*señal_de_salida(n)=señal_de_entrada(n)+G1*señal_de_entrada(n-d1(n))+G2* señal_de_entrada(n-d2(n))*

O de manera simplificada basándonos en el diagrama de la figura 12:

Y(n)=x(n)+G1x(n-d1(n))+G2x(n-d2(n)).

Quedando así definido completamente las ecuaciones del chorus [16].



Para su implementación los valores a introducir por el usuario deben de ser el tiempo de retraso, frecuencia del LFO y ganancia de las repeticiones.

Una vez hecho esto se procederá a calcular el número de retrasos como ya se vio anteriormente, además a la hora de implementarlos los retrasos fueron redondeados para que se pudiera trabajar como números enteros.

Con los retrasos calculados se definen las Fd de nuestras señales atendiendo a la formula vista anteriormente, las cuales también fueron redondeadas para obtener números enteros.

Finalmente se definen nuestras señales desfasadoras D[n] según se vio en su respectiva ecuación, sin olvidar que el valor que nos den están funciones para un determinado intervalo de tiempo también deberá ser redondeado.

Hágase notar la importancia de redondear ciertos valores para obtener valores enteros, esto es así debido a que tenemos que desplazar nuestra señal, no podemos desplazarla muestras decimales, solamente se podrá desplazar muestras que sean valores enteros.

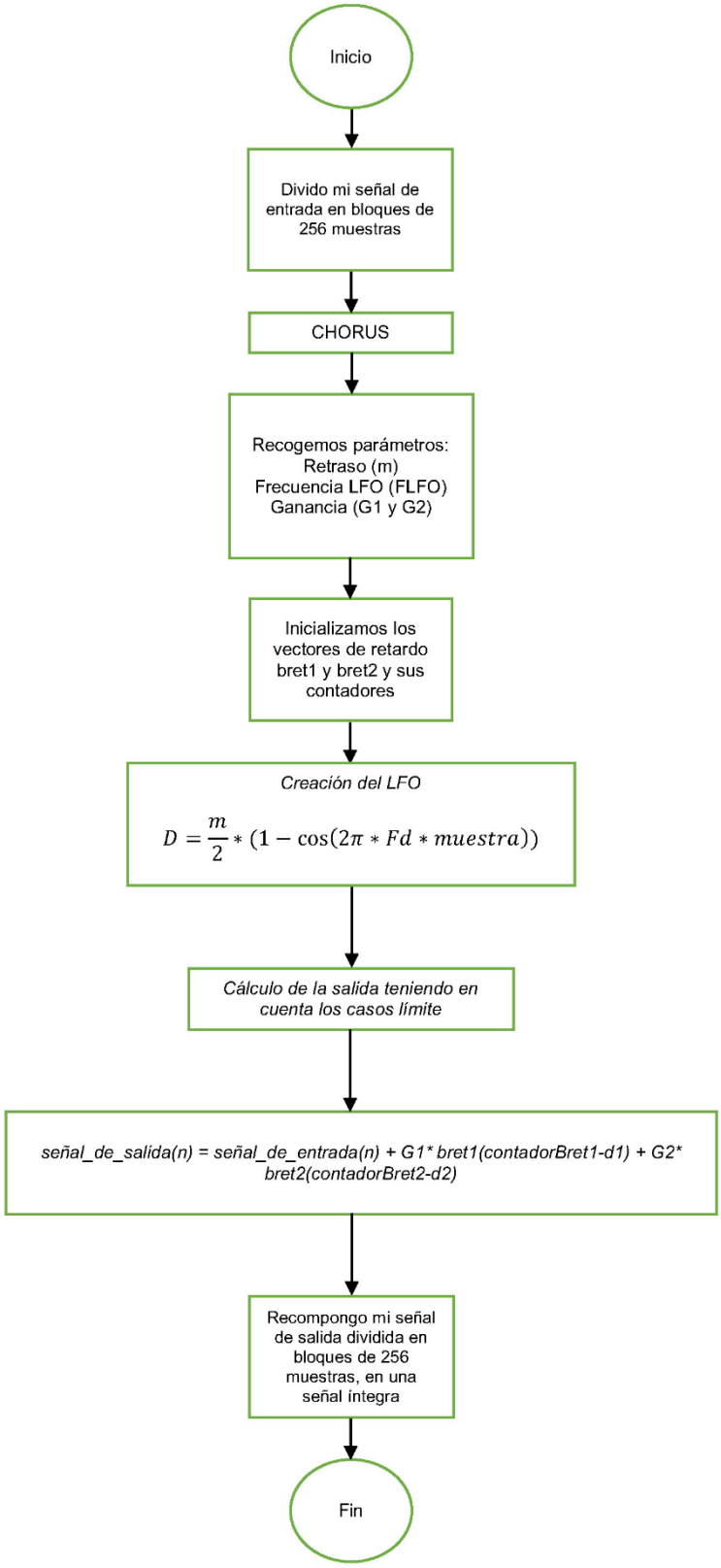
Una vez tengamos nuestros valores y parámetros preparados solamente tendremos que aplicar la ecuación que define el chorus y habremos conseguido el efecto.

A pesar de parecer complejo el chorus los únicos parámetros a introducir por el usuario serán:

- Tiempo de retraso de las voces (m)
- Frecuencia del LFO (FLFO)
- Ganancia de las voces repetidas. (G1 y G2) [16]

5.5.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto CHORUS para su adaptación a un entorno de tiempo real:

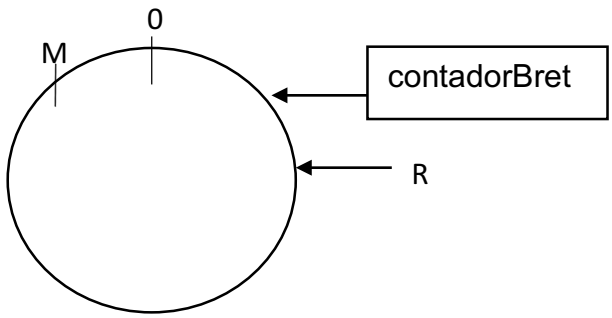


Tendremos que tener en cuenta la creación e inicialización de mis vectores de retardo (bret1 y bret2), cuyo tamaño vendrá dado por el retraso convertido a muestras introducido por el usuario.

A la hora de crear mi LFO, tendremos que tener en cuenta que no tenemos la señal de entrada en su totalidad para crear el LFO con el mismo tamaño que la señal de entrada, pero si que podemos calcular la muestra por la que vamos dentro de los bucles donde realizamos el efecto, así, podremos calcular el valor del LFO por cada muestra calculada, y posteriormente acceder a mi vector de retardos con el valor obtenido en el LFO, dicho de otra manera accedo a la posición de mi vector de retardos definida por el valor del LFO (teniendo presente siempre el contador de mi vector de retardos).

La necesidad de crear el buffer circular genera la problemática de tener en cuenta una serie de casos límite que deben de ser presentados para acercarnos cuanto más podamos al efecto original consiguiendo así un error relativamente pequeño a pesar de movernos en un entorno de tiempo real.

En el siguiente esquema podemos observar la problemática del buffer circular (suponiendo que mi buffer tiene un tamaño M y el retardo calculado es R):



Esquema representando buffer circular “bret”

Como podemos observar hay representado una de las problemáticas (casos límite) de hacer uso del buffer circular y que está presente de manera común en los efectos que hacen uso de este buffer. En el esquema vemos que el buffer tiene un tamaño M, y el 0 representa la primera posición de mi vector. Como vemos contadorBret está por debajo del retardo calculado, esto generará un problema puesto que se querrá acceder a una posición negativa (teniendo en cuenta que accedemos a la posición *contadoBret – R* de nuestro vector bret), dicho de otra manera el retardo R calculado valdrá 125 cuando el contador va mas retrasado, p.ej va por el valor 45. Esto se dará de manera repetida en cada iteración y por tanto será necesario estos cambios. Existirán mas casos límite, todos ellos están expuestos en el pseudocódigo que sigue a continuación, debemos de tener en cuenta que los casos limites se verán reducidos en otros efectos, pero aun así se deberán de contemplar en nuestro código.

Puesto que estamos utilizando bufferes circulares (bret y contadores), habrá que tener en cuenta además los siguientes casos:

Si d1==0

tmp1 = señal_de_entrada(n)

Si ((contadorBret1-d1) == 0

tmp = bret1((retraso1-1)+(contadorBret1-d1))

Si ((contadorBret1-d1) < 0

tmp = bret1(retraso1+(contadorBret1-d1))

Si no se aplicó ningún caso anterior:

tmp = bret1(contadorBret1-d1)

(Lo mismo haríamos con el segundo vector de retardos (bret2) → tmp2)

Finalmente aplicaríamos las condiciones a la ecuación general:

*señal_de_salida(n) = señal_de_entrada(n) + amplitud*tmp1 + amplitud*aleatorio*tmp2*

5.6 Flanger

5.6.1. Introducción

El flanger es un efecto de audio digital que se produce al mezclar una señal con una copia de ella ligeramente retrasada, a pesar de que el tiempo de retraso de la señal retrasada cambia constantemente [6].

El efecto de flanger se caracteriza porque crea un grupo de muescas igualmente espaciadas en el espectro de audio que se desplazarán armónicamente por él filtrando la señal de forma que se produce tal efecto [7].

El efecto flanger se obtiene duplicando la onda sonora original; una de las ondas se mantiene limpia de procesado, la segunda se desfasa moduladamente, aumentando o disminuyendo su retraso con una oscilación determinada [8].

5.6.2. Implementación

El modelado del efecto de flanger es muy similar al del chorus, como puede verse en la figura 13, pero con tiempos de retardo menores y con el uso de una forma de onda típicamente triangular para el LFO. A pesar de utilizar típicamente un LFO triangular, finalmente nuestro flanger fue implementado con un LFO sinusoidal por problemas acústicos [16].

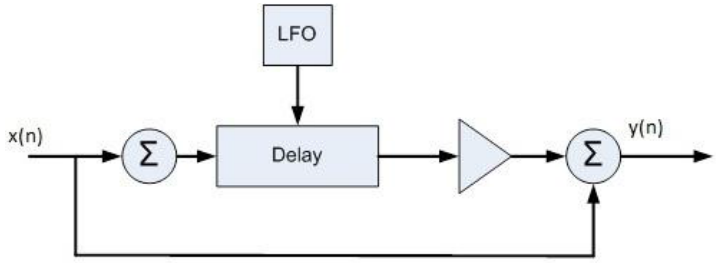


Figura 13. Diagrama de bloques del Flanger [8].

A pesar de tener una copia de la señal de entrada retrasada no se oye como un eco, esto es debido a que el valor máximo del retraso del flanger es de unos 20 ms (el oído humano percibe un eco si el retraso es de más de 50 a 70 milisegundos). El flanger tiene un efecto de filtro en lugar de efecto de eco. Este efecto de filtrado crea muescas en la respuesta en frecuencia como puede verse en la figura 14.

Así en los puntos donde la respuesta en frecuencia tiende a cero los sonidos de las frecuencias medias se eliminan dejando pasar así otras frecuencias. Este tipo de respuesta en frecuencia es denominado efecto peine.

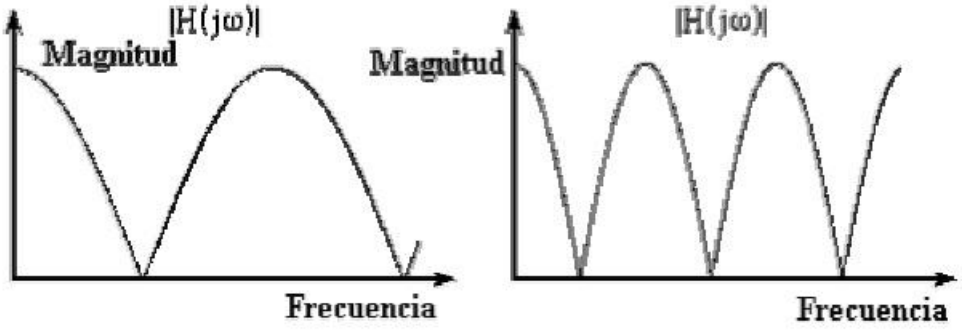
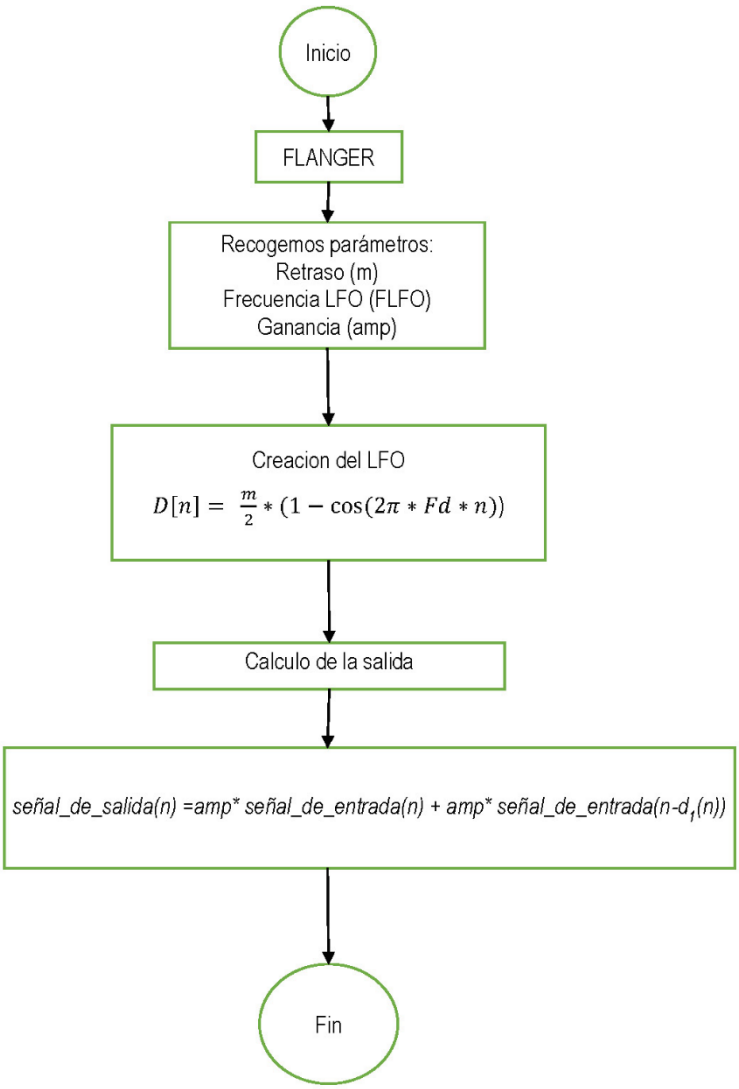


Figura 14. Respuesta en frecuencia del Flanger con diferentes tipos de retraso [8].

Así, este efecto es modelado mezclando una señal con una copia levemente retrasada, donde la longitud del retardo está constantemente cambiando debido a la modulación del LFO.



Para su implementación se parte de una unidad de delay básica, modulada mediante un LFO, esto produce esta serie de saltos en la respuesta en frecuencia. Esto es producido porque para algunas frecuencias el retraso de fase introducido es igual a 180º, que es equivalente a añadir su entrada pero negativa.

Así cuando se mezcla esta señal con la entrada, las frecuencias que sufrieron el retraso de 180º se cancelaran exactamente con las componentes originales de la entrada (interferencia destructiva), originando las muescas para dichas frecuencias. El caso contrario se dará cuando es desfase producido sea de 360º en cuyo caso las señales originales y retrasada se sumarán doblando su valor (interferencia constructiva).

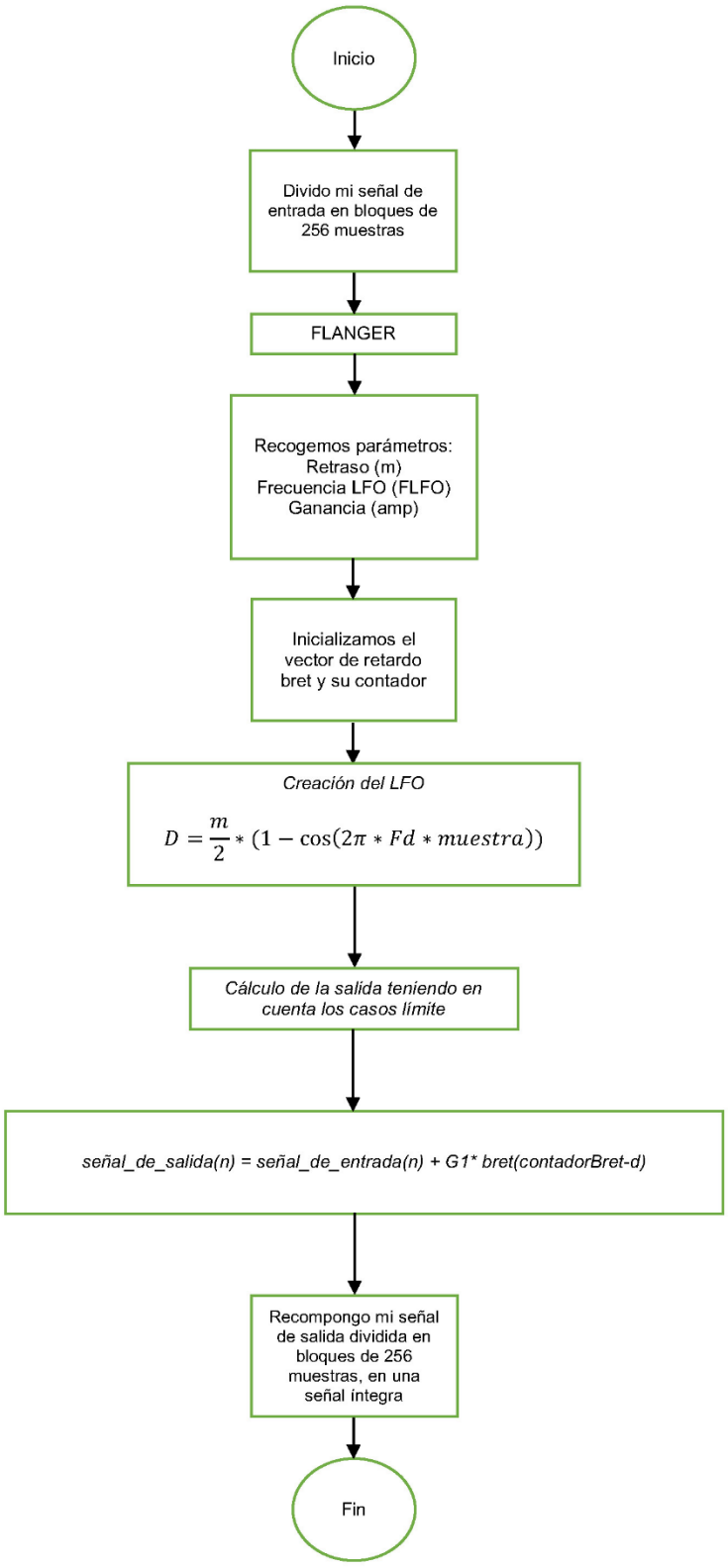
La acción de estos saltos llevada a cabo de forma continua son las que provocan la variación de continua del retardo, debido a esta modulación de tono es creado el efecto flanger. Pues su sonido tan característico resulta pues cuando las muescas de la respuesta van

recorriendo el eje de frecuencia a lo largo del tiempo como resultado de la variación del retraso producido por es LFO a unas frecuencias entre 0-1.5Hz. Haciendo así que cuando el retraso se incremente las muescas se comprimen hacia las bajas frecuencias y cuando éste disminuye, dichas muescas se expandan hacia las altas frecuencias.
En su implementación en el microprocesador el tiempo de retraso introducido por el usuario será convertido a muestras multiplicándolo por la frecuencia de muestreo.

Tras esto recorreremos nuestra señal por todas sus muestras a partir de la muestra obtenida por el tiempo de retraso.
Cabe indicar que para conseguir el filtro peine solamente hemos tomado los valores de nuestro LFO, es decir hemos tomado el modulo, haciendo así que su valor este entre 0-1. Y redondeándolo al entero más próximo, tras esto ya tenemos el número de muestras para retrasar nuestra copia de la señal de entrada, que antes de ser mezclada con la original será amplificada o atenuada, según lo indique el usuario. Añadiendo esta finalmente a la señal original para conseguir el efecto de flanger [16].

5.6.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto FLANGER para su adaptación a un entorno de tiempo real:

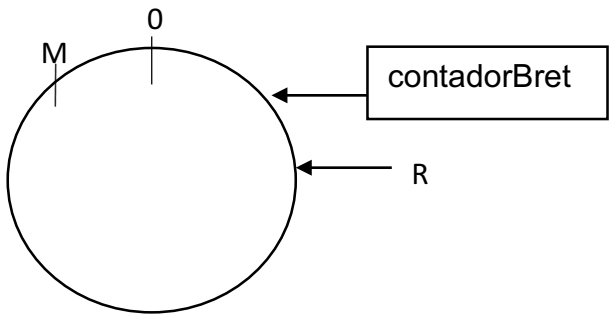


Tendremos que tener en cuenta la creación e inicialización de mi vector de retardo (bret), cuyo tamaño vendrá dado por el retraso convertido a muestras introducido por el usuario.

A la hora de crear mi LFO, tendremos que tener en cuenta que no tenemos la señal de entrada en su totalidad para crear el LFO con el mismo tamaño que la señal de entrada, pero si que podemos calcular la muestra por la que vamos dentro de los bucles donde realizamos el efecto, así, podremos calcular el valor del LFO por cada muestra calculada, y posteriormente acceder a mi vector de retardos con el valor obtenido en el LFO, dicho de otra manera accedo a la posición de mi vector de retardos definida por el valor del LFO (teniendo presente siempre el contador de mi vector de retardos).

La necesidad de crear el buffer circular genera la problemática de tener en cuenta una serie de casos límite que deben de ser presentados para acercarnos cuanto más podamos al efecto original consiguiendo así un error relativamente pequeño a pesar de movernos en un entorno de tiempo real.

En el siguiente esquema podemos observar la problemática del buffer circular (suponiendo que mi buffer tiene un tamaño M y el retardo calculado es R):



Esquema representando buffer circular “bret”

Como podemos observar hay representado una de las problemáticas (casos límite) de hacer uso del buffer circular y que está presente de manera común en los efectos que hacen uso de este buffer. En el esquema vemos que el buffer tiene un tamaño M, y el 0 representa la primera posición de mi vector. Como vemos contadorBret está por debajo del retardo calculado, esto generará un problema puesto que se querrá acceder a una posición negativa (teniendo en cuenta que accedemos a la posición *contadoBret – R* de nuestro vector bret), dicho de otra manera el retardo R calculado valdrá 125 cuando el contador va mas retrasado, p.ej va por el valor 45. Esto se dará de manera repetida en cada iteración y por tanto será necesario estos cambios. Existirán mas casos límite, todos ellos están expuestos en el pseudocódigo que sigue a continuación, debemos de tener en cuenta que los casos limites se verán reducidos en otros efectos, pero aun así se deberán de contemplar en nuestro código.

En este efecto en concreto uno de los casos límites que cabe destacar sobre los demás es que el efecto se empezará a aplicar a partir de la muestra definida por retrasos, dicho de otra manera si el retraso en muestras vale 441, el efecto empezara a aplicarse a partir de la muestra 441 de mi señal de entrada.

Puesto que estamos utilizando bufferes circulares (bret y contadores), habrá que tener en cuenta además los siguientes casos:

Si muestra<retraso

señal_de_salida(n) = 0

Si retraso == 0

*señal_de_salida(n) = amplitud*señal_de_entrada(n) + amplitud*señal_de_entrada(n)*

Si (contadorBret-retr)==0

*señal_de_salida(n) = amplitud*señal_de_entrada(n) + amplitud*(bret((retrasos-1)+(contadorBret-retr)))*

Si (contadorBret-retr)<0

*señal_de_salida(n) = amplitud*señal_de_entrada(n) + amplitud*(bret(retrasos+(contadorBret-retr)))*

Si no se aplicó ningún caso anterior:

*señal_de_salida(n) = amplitud*señal_de_entrada(n) + amplitud*(bret(contadorBret-retr))*

(Siendo retr el valor redondeado de mi LFO para cada muestra)

5.7 Tremolo

5.7.1. Introducción

El efecto tremolo es simplemente una fluctuación periódica de la amplitud de la señal con el tiempo. Debido a estas variaciones se puede considerar como una modulación de la amplitud, producida por la aplicación de un LFO. Lo que genera que la señal original vaya variando según la amplitud de la señal del LFO [8].

El termino Tremolo a menudo se utiliza indistintamente como vibrato. Sin embargo existen definiciones estrictas que explican los efectos; el

vibrato es una variación periódica de la frecuencia del sonido mientras que el Tremolo es una variación periódica de la amplitud como ya dijimos. La figura 15 ilustra la diferencia entre los espectrogramas de los dos efectos.

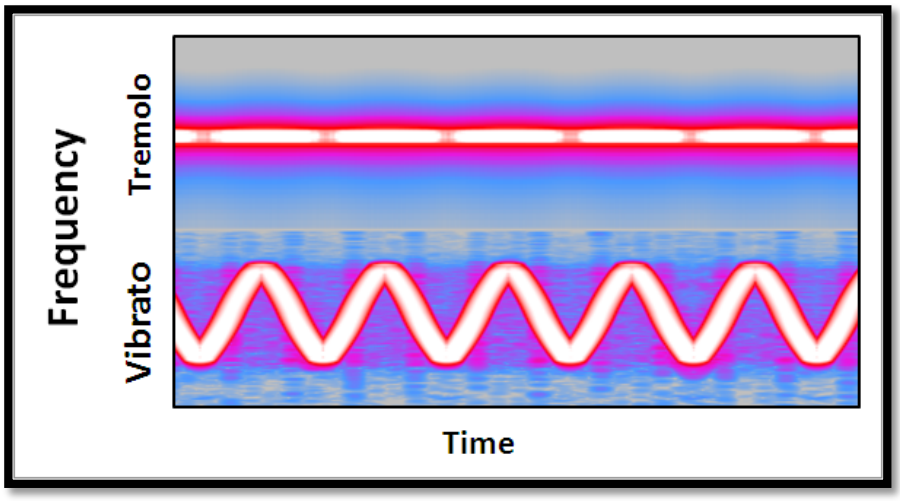


Figura 15. Espectrograma del tremolo y vibrato [8].

5.7.2. Implementación

La modulación utilizada para el efecto tremolo generalmente viene dada por un oscilado de baja frecuencia (LFO), la amplitud de la señal de audio varía según la amplitud del LFO. La figura 16 muestra el esquema de un efecto tremolo.

Figura 16. Diagrama de un tremolo [8].

Tal y como dijimos anteriormente el efecto tremolo es un efecto de modulación donde la señal de entrada es la portadora y la señal generada por el LFO es la moduladora.

Así en la figura 17 se puede ver un ejemplo de esta modulación, donde se utilizó un LFO de forma sinusoidal, el resultado obtenido es muy parecido al de una modulación AM, donde se tienen máximo y mínimos en amplitud. Demostrando esto que el tremolo no es más otra modulación más en amplitud [16].

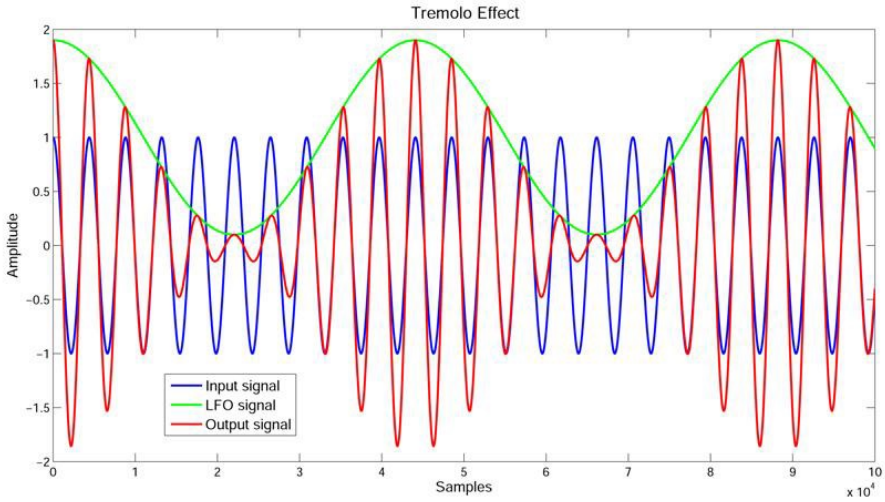
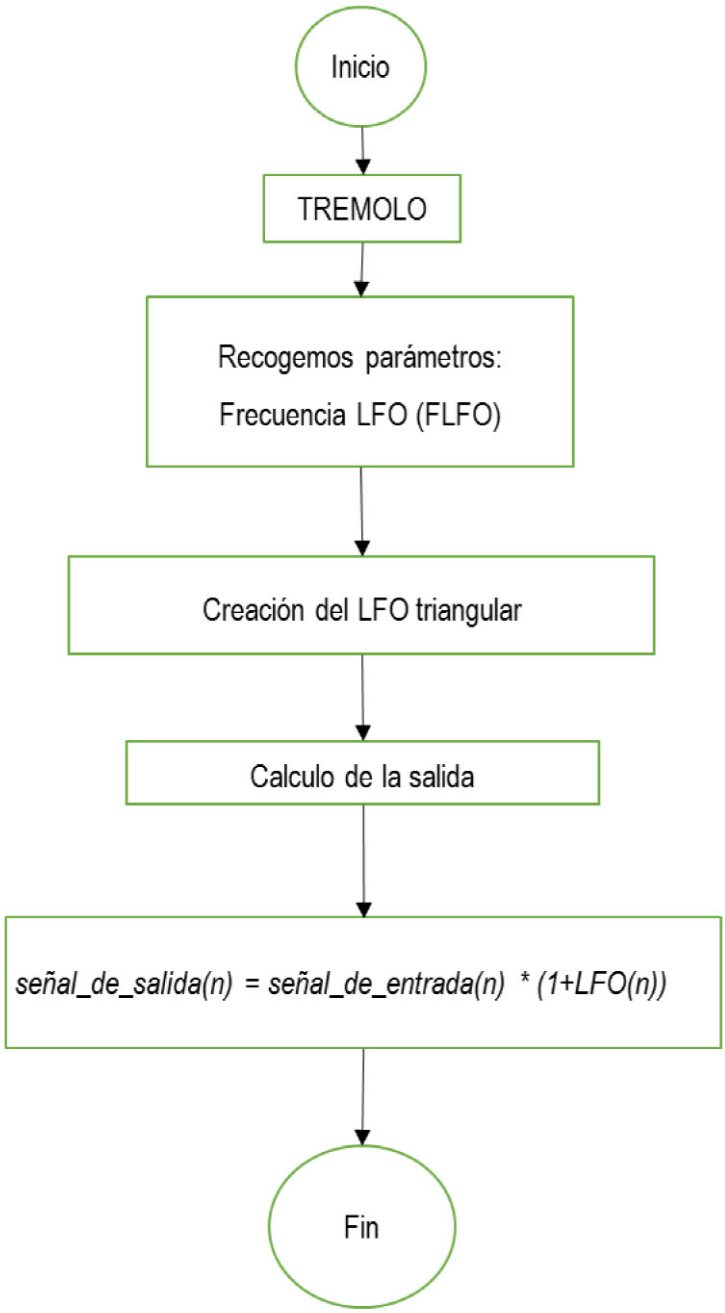


Figura 17. Tremolo aplicado con una señal sinusoidal [8].

Tras esto definir que las ecuaciones que definen al tremolo no tienen gran complejidad, ya que la ecuación que lo define es la siguiente:

*señal_de_salida(n) = señal_de_entrada(n) * (1+LFO(n))*



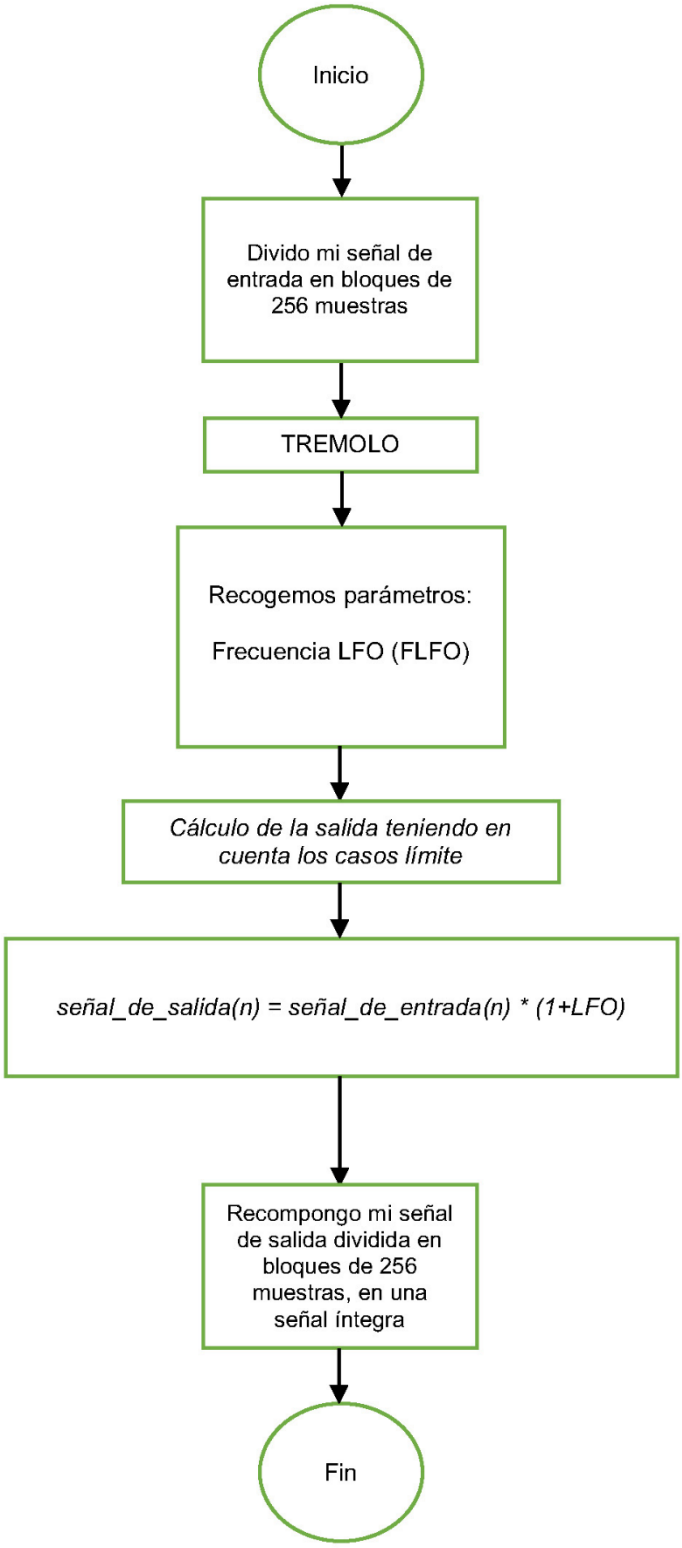
Dándonos esta ecuación como único parámetro a introducir por el usuario la frecuencia del LFO.
Y una vez conocida la frecuencia del LFO se creara una señal triangular para modular el efecto y aplicar la ecuación mencionada anteriormente [16].

Con lo que los parámetros para manejar el tremolo serán:

- Frecuencia del LFO

5.7.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto TREMOLO para su adaptación a un entorno de tiempo real:



A la hora de implementar este efecto, tendremos que desarrollar el LFO de forma triangular y posteriormente ir accediendo a las posiciones del LFO para que se simule ese efecto acústico.
Esta posición se obtiene mediante el resto de la división entera del redondeo de la frecuencia del LFO por la muestra por la que vamos y el tamaño del LFO creado (que es un vector de n posiciones).

Solo tendremos presente un caso límite:

Si lfo(freq)==0

señal_de_salida(n) = señal_de_entrada(n)

Si no se aplicó ningún caso anterior:

señal_de_salida(n) = senal_de_entrada(n)(1+lfo(freq))*

(Donde freq es el resto de la división entera del redondeo de la frecuencia del LFO por la muestra por la que vamos y el tamaño del LFO creado (que es un vector de n posiciones)).

5.8 Vibrato

5.8.1. Introducción

El vibrato puede describirse como una fluctuación periódica de la frecuencia de la señal, lo que produce fluctuación audible claramente en la señal procesada. Como ya se mostró en la figura 15, se puede ver claramente la diferencia entre un tremolo y un vibrato. El ejemplo más básico de un vibrato podría ser el producido por un guitarrista si al tocar una nota, con la mano que pulsa la nota la desliza hacia arriba y abajo provocando fluctuaciones en el sonido [8].

5.8.2. Implementación

La implementación de un efecto vibrato no tiene mayor complicación, ya que prácticamente consiste en aplicar una serie de retardos modulados mediante un LFO.

Los retardos a aplicar a la señal de entrada al ser modulados mediante un LFO unas veces serán mayores que otras incluso llegando a ser cero, haciendo esto que para algunos instantes de tiempo no se llegue a aplicar retraso a la señal haciendo así que esta coincida con la señal original.

Al igual que ocurría con los demás efectos implementado la forma de onda del LFO puede ser cualquiera, habiendo elegido para nuestra implementación una forma de onda sinusoidal para que la variación fuera más suave y no estropeara tanto la señal a tratar [16].

En la figura 18 se puede ver el diagrama de bloques de un efecto vibrato.

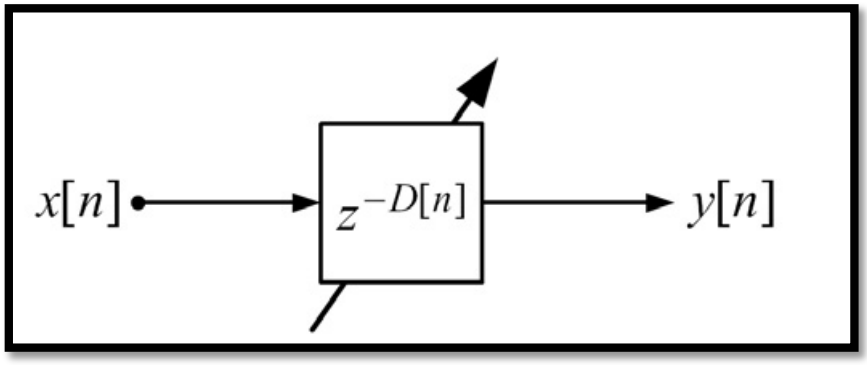


Figura 18. Diagrama de bloques del vibrato [8]

Como puede verse y ya se comentó anteriormente los retrasos a aplicar a nuestra señal vienen definidos como una función. Dicha función no es más que un LFO y queda definida como ya se mencionó anteriormente en otros efectos como:

$$D[n] = \frac{m}{2} * (1 - \cos(2\pi * Fd * n))$$

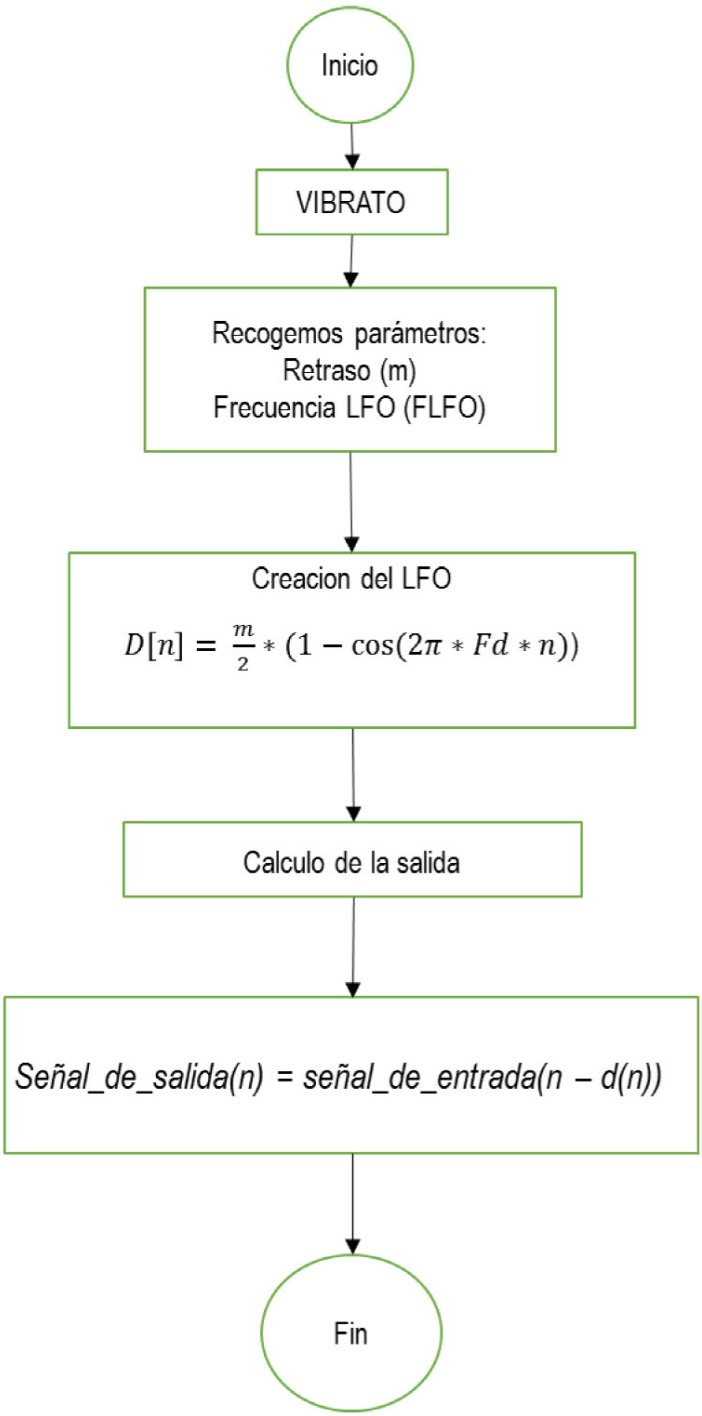
Donde el parámetro "m" de la ecuación no es más que el tiempo de delay introducido por el usuario, el cual será convertido a muestras y redondeado al entero más próximo para poder acceder perfectamente a la muestra deseada [16].

Y donde el parámetro "F_d" es la frecuencia del LFO, la cual para un vibrato suele estar comprendida entre 4 y 15 Hz, la cual también deberá ser introducida por el usuario. Y modelada mediante la siguiente ecuación [16]:

$$Fd = \frac{2\pi}{\frac{Fs}{FLFO}}$$

Una vez que ya conocemos los parámetros para nuestro efecto solamente nos quedaría definir la ecuación que define al vibrato como tal:

Señal_de_salida(n) = señal_de_entrada(n –d(n))



Cabe destacar que nuestra señal D(n) al ser una sinusoidal periódica tendrá sus valores máximos y en otras ocasiones su valor será cero, en este caso no se producirá retraso alguno sobre la señal de entrada [16].

La siguiente figura muestra claramente esto, como la señal de salida se encuentra desfasada respecto a la de entrada y como luego vuelve a coincidir de nuevo con la señal de entrada.

Cabe destacar que lo que se muestra en la siguiente figura es un ejemplo probado con un tono, lógicamente esto no se verá tan claro con señales más complejas [16].

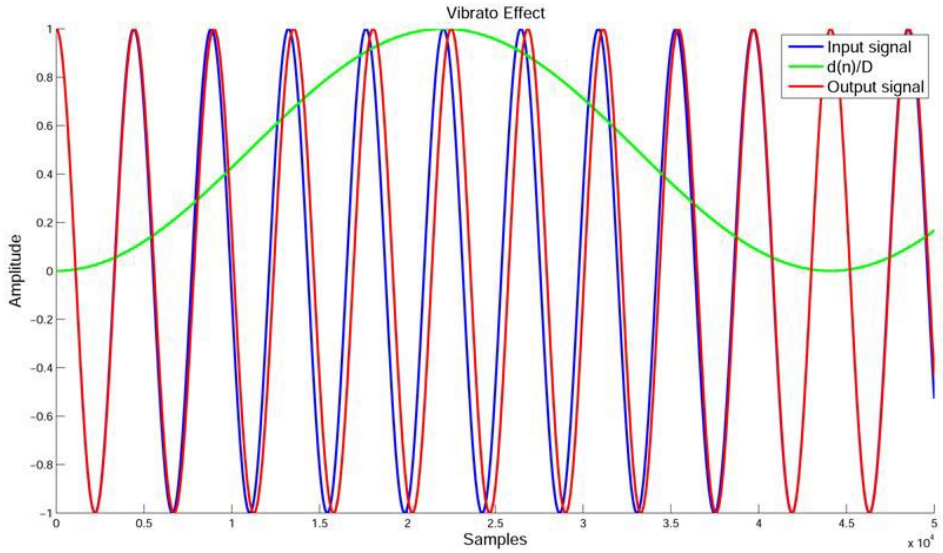


Figura 19. Vibrato aplicado a una señal sinusoidal [8].

Tras estos indicar que el retraso para un vibrato suele estar comprendido entre 5-10 ms con lo cual la única función que hará será darle valor a la amplitud de la señal del LFO para modular los retrasos a aplicar a nuestras señales.

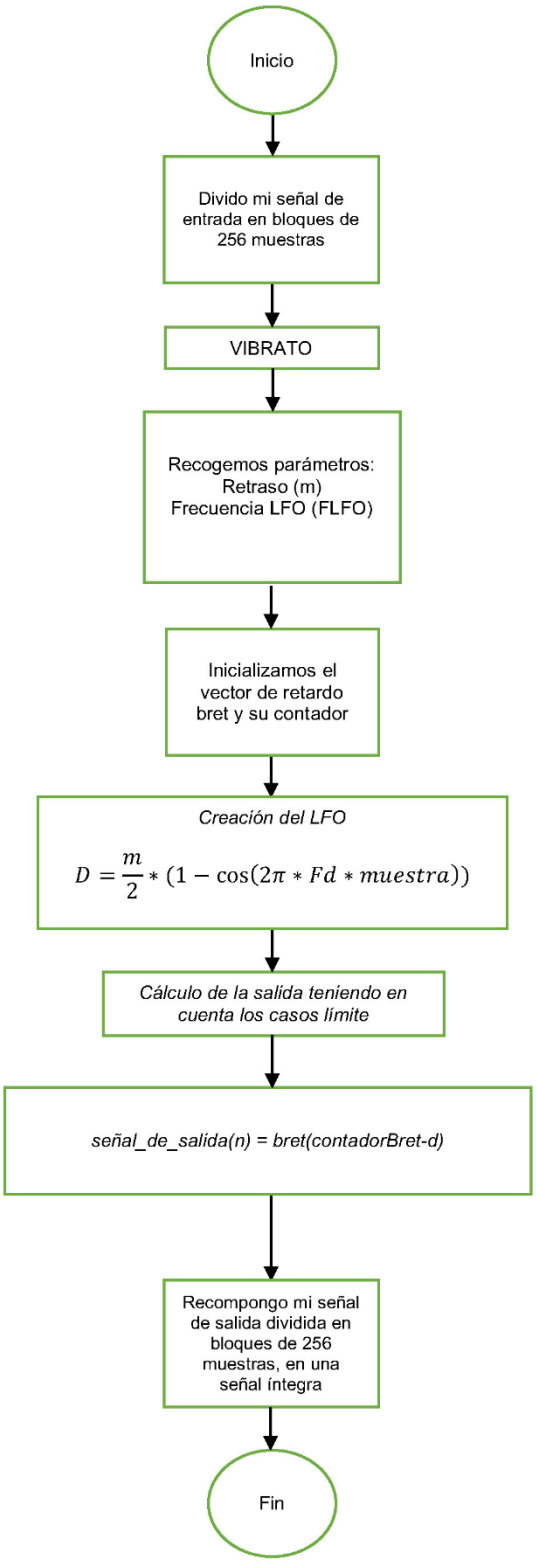
Mientras que la frecuencia del oscilador local estará comprendida como ya se dijo anteriormente entre 4 y los 15 Hz.

Quedando así de esta manera definido nuestro efecto vibrato y sus parámetros a introducir por el usuario serán los siguientes:

- Tiempo de retraso (m)
- Frecuencia del LFO [16]

5.8.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto VIBRATO para su adaptación a un entorno de tiempo real:

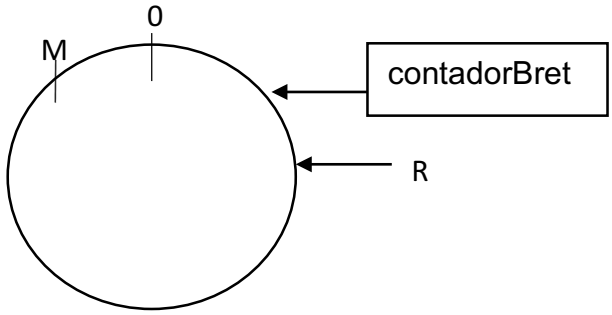


Tendremos que tener en cuenta la creación e inicialización de mi vector de retardo (bret), cuyo tamaño vendrá dado por el retraso convertido a muestras introducido por el usuario, en este caso el tamaño vendrá dado por la fs, es decir el tamaño del buffer es de 44100 posiciones.

A la hora de crear mi LFO, tendremos que tener en cuenta que no tenemos la señal de entrada en su totalidad para crear el LFO con el mismo tamaño que la señal de entrada, pero si que podemos calcular la muestra por la que vamos dentro de los bucles donde realizamos el efecto, así, podremos calcular el valor del LFO por cada muestra calculada, y posteriormente acceder a mi vector de retardos con el valor obtenido en el LFO, dicho de otra manera accedo a la posición de mi vector de retardos definida por el valor del LFO (teniendo presente siempre el contador de mi vector de retardos).

La necesidad de crear el buffer circular genera la problemática de tener en cuenta una serie de casos límite que deben de ser presentados para acercarnos cuanto más podamos al efecto original consiguiendo así un error relativamente pequeño a pesar de movernos en un entorno de tiempo real.

En el siguiente esquema podemos observar la problemática del buffer circular (suponiendo que mi buffer tiene un tamaño M y el retardo calculado es R):



Esquema representando buffer circular “bret”

Como podemos observar hay representado una de las problemáticas (casos límite) de hacer uso del buffer circular y que está presente de manera común en los efectos que hacen uso de este buffer. En el esquema vemos que el buffer tiene un tamaño M, y el 0 representa la primera posición de mi vector. Como vemos contadorBret está por debajo del retardo calculado, esto generará un problema puesto que se querrá acceder a una posición negativa (teniendo en cuenta que accedemos a la posición *contadoBret – R* de nuestro vector bret), dicho de otra manera el retardo R calculado valdrá 125 cuando el contador va mas retrasado, p.ej va por el valor 45. Esto se dará de manera repetida en cada iteración y por tanto será necesario estos cambios. Existirán mas casos límite, todos ellos están expuestos en el pseudocódigo que sigue a continuación, debemos de tener en cuenta

que los casos limites se verán reducidos en otros efectos, pero aun así se deberán de contemplar en nuestro código.

Tendremos en cuenta los siguientes casos límite:

Si (contadorBret-d)<0

señal_de_salida(n) = (bret(retrasos+(contadorBret-d)))

Si no se aplicó ningún caso anterior:

señal_de_salida(n) = bret(contadorBret-d))

(Siendo d el valor redondeado de mi LFO para cada muestra y retrasos = fs = 44100)

5.9 Delay

5.9.1. Introducción

El delay es el efecto más básico y simple de los efectos basados en retardos, siendo uno de los efectos más clásicos y comunes en muchas aplicaciones de audio, estando presenta hasta en la producción musical.

Se utiliza en todo tipo de música desde música electrónica hasta electroacústica, así como en los sistemas de refuerzo de sonido. Pudiendo ofrecer gran variedad de aplicaciones que van desde fines puramente artísticos hasta otras aplicaciones de mejora del sonido [6].

A pesar de ser un efecto muy simple es un efecto muy importante, no solo como efecto sino como elemento potencial para otros efectos como el chorus o el flanger como se vio anteriormente así como para la reverberación [7].

5.9.2. Implementación

Para la implementación del delayno tiene mayor complicación, ya que básicamente se trata de coger una muestra se la señal original desplazada y escalada y sumada a la señal original.

La figura 19 muestra el diagrama de bloques de un efecto Delay.

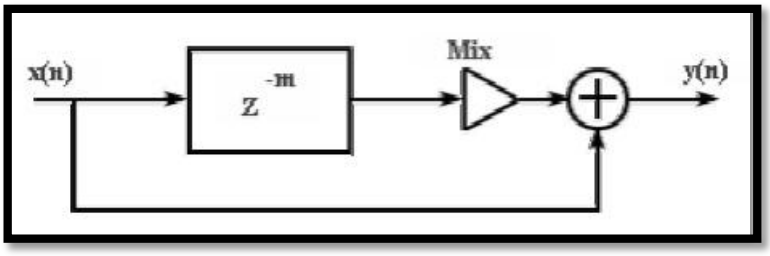


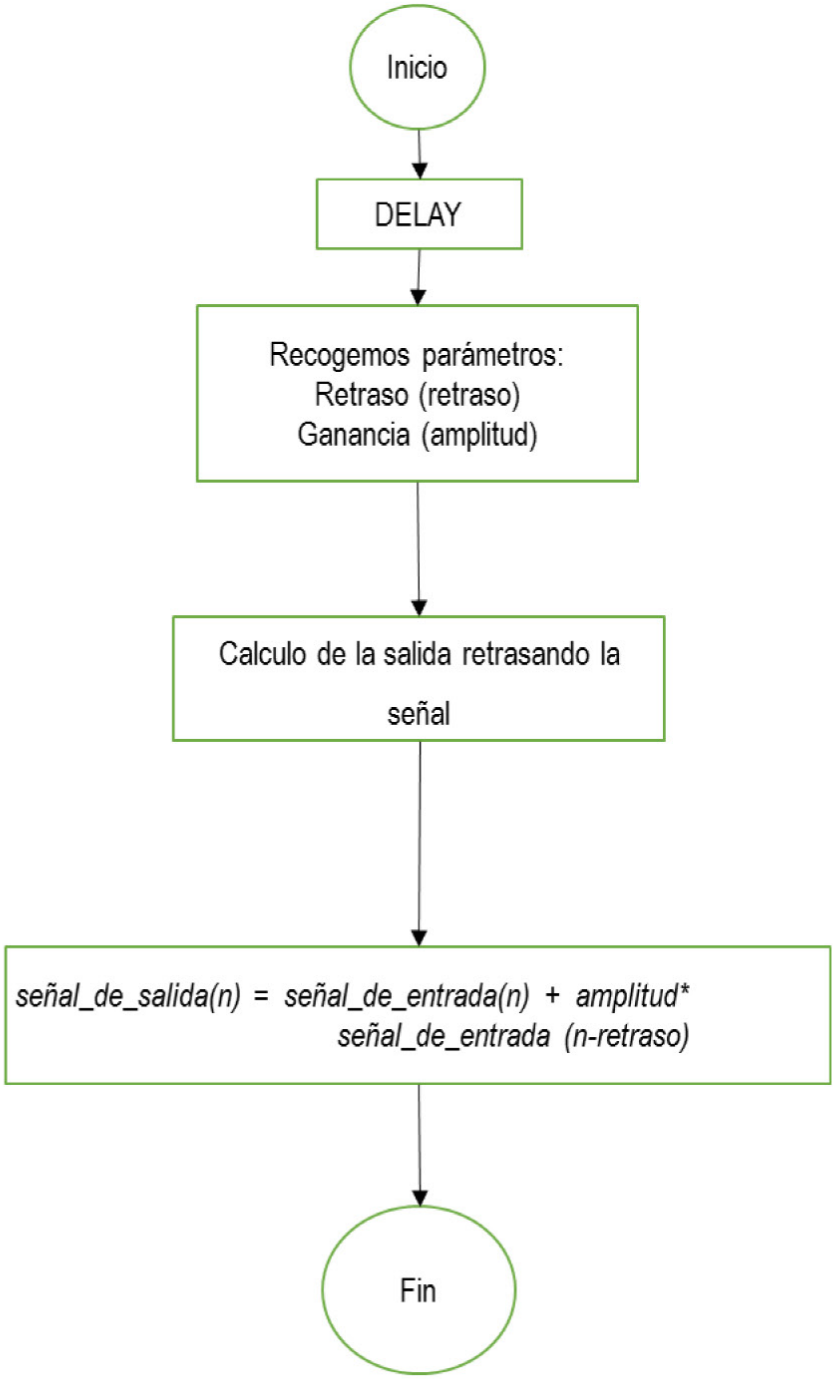
Figura 20. Diagrama de bloques del delay [6].

Como puede apreciarse en el diagrama de bloques a la señal retrasada se le aplica un amplificación, normalmente suele ser para atenuar un poco el sonido de la señal haciéndose notar más o menos la intensidad de las muestras repetidas.

El tiempo de delay para retrasar nuestra señal puede variar desde unos pocos milisegundos hasta varios segundos. Dependiendo del tiempo de retraso elegido el efecto delay sonara de una manera u otra, esto es con un tiempo de retraso de unos 40-80 ms el efecto delay sonara como un efecto de coro poco elaborado. Es con un tiempo de retraso entre 100 y 120 ms cuando sonara como un efecto delay.
También indicar que para tiempos mayores a 1 segundo permite al músico tocar melodías sobre sí mismo.

Tras esto podemos decir que el efecto delay toma una señal de audio y la reproduce de nuevo después de un tiempo de delay, escalada y mezclada con la señal actual. Siendo las ecuaciones que definen al delay [16]:

*señal_de_salida(n) = señal_de_entrada(n) + amplitud * señal_de_entrada(n-retraso)*



Donde la amplitud será para controlar la cantidad de eco que se percibe y será indicado por el usuario estando su valor comprendido entre 0-1.

Por otra parte los retrasos serán introducidos por el usuario como un tiempo de delay, dicho tiempo será convertido a muestras multiplicado por la frecuencia de muestreo de nuestra señal.
Tras aplicar el algoritmo de nuestro efecto delay a nuestra señal, deberemos normalizarla, es decir, la señal será divida entre el modulo

del valor absoluto de la señal para que no exceda de nuestro rango dinámico.

Esto se ve más claramente de la siguiente manera, imaginemos una serie de muestras de amplitud igual a la unidad, si aplicamos el delay con un tiempo de retraso igual a una muestra, el delay cogerá la primera muestra de amplitud 1 la retrasará una muestra y la añadirá a la muestra anterior que también tenía amplitud 1, haciendo así que la amplitud de esta segunda muestra sea igual a 2 escapándose así de nuestro rango dinámico comprendido entre ±1 [16].

En el siguiente ejemplo vemos cómo reacciona el delay si la entrada es un impulso unitario, donde la ganancia de las repeticiones es de 0.9 y el tiempo de retraso es igual a 1 [16].

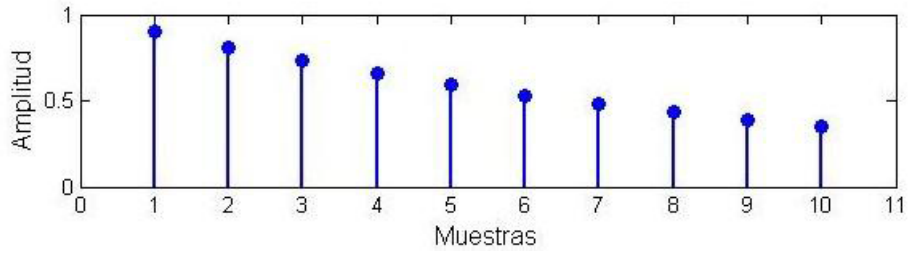


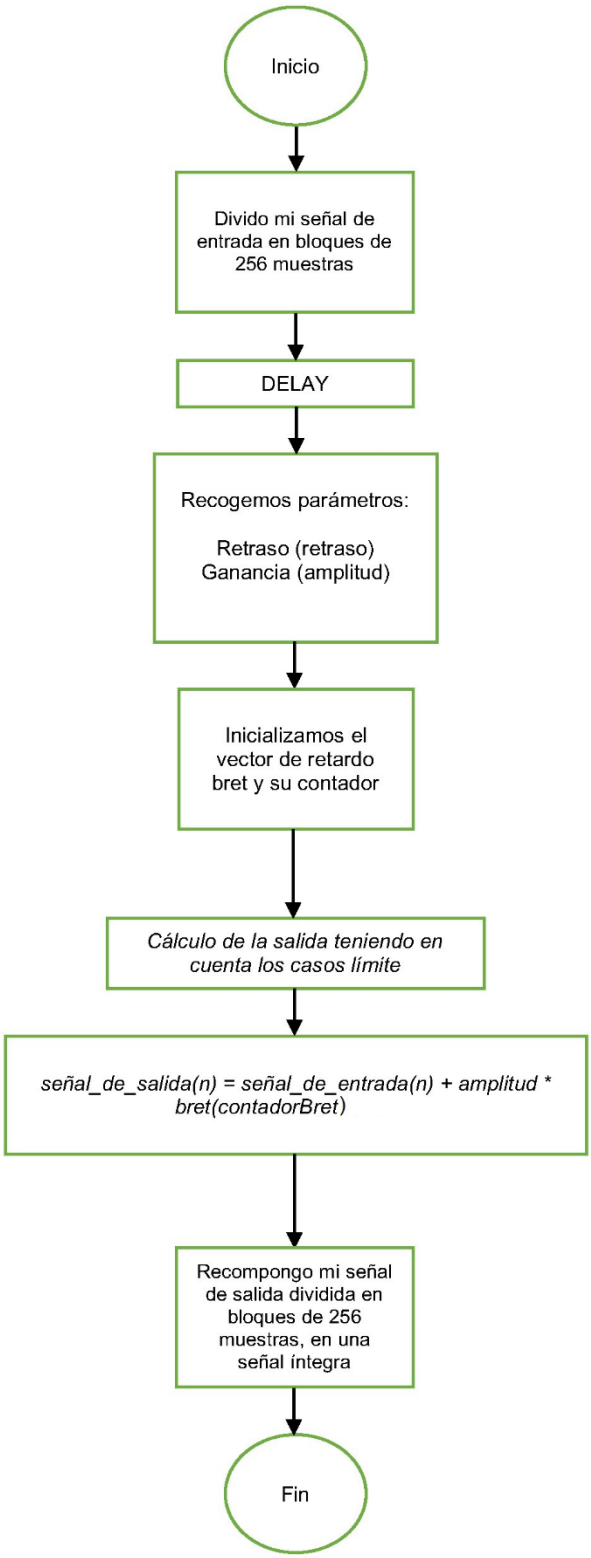
Figura 21. Respuesta al impulso unitario con amplitud=0.9 y retraso=1 muestra.

De esta manera los parámetros a controlar por el usuario serán:

- Tiempo de retardo (retraso)
- Amplitud de las repeticiones (amplitud)

5.9.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto DELAY para su adaptación a un entorno de tiempo real:



Tendremos que tener en cuenta la creación e inicialización de mi vector de retardo (bret), cuyo tamaño vendrá dado por el retraso convertido a muestras introducido por el usuario.

En este efecto solo tendremos presente en la salida:

señal_de_salida(n) = señal_de_entrada(n) + amplitud(bret(contadorBret))*

5.10 Reverberación

5.10.1. Introducción

La reverberación, también conocida como “reverb” es uno de los efectos más utilizados hoy día en la música.

La reverberación se basa prácticamente en retardos, ya que es la persistencia del sonido en un espacio particular después de que el sonido original se haya eliminado. Dicho efecto es algo que suele ocurrir continuamente en la vida cotidiana, debido a las reflexiones del sonido que toma una señal al reflejarse en distintos puntos. Estas múltiples reflexiones de ondas crean una serie de ecos que se atenúan lentamente debido a que el sonido es absorbido por el aire y las paredes [7].

Es decir la reverberación no es más que el resultado de varias reflexiones del sonido que ocurren en una habitación, así desde cualquier fuente de sonido hay un camino directo que las ondas recorren hasta alcanzar al oyente. Y otra parte del sonido tomara un camino más largo a través del rebote en las paredes, tal y como se muestra en la siguiente figura [16].

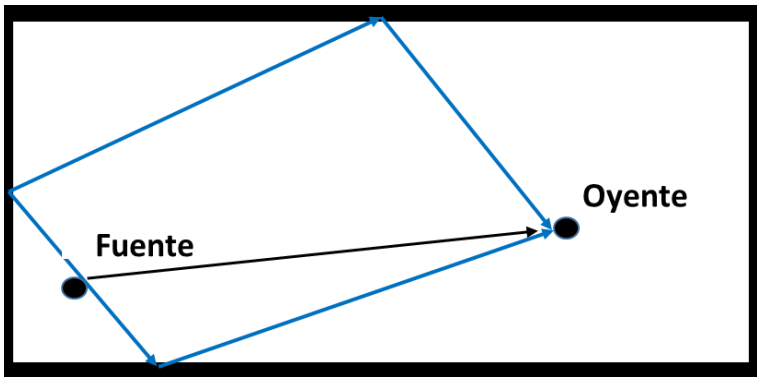


Figura 22. Camino del sonido.

5.10.2. Implementación

Como ya se ha comentado anteriormente la onda de sonido reflejada llegar al oído con cierto retardo respecto a la onda directa, pues esta viaja una mayor distancia. Además es generalmente más débil que la onda directa, pues las paredes y otras superficies de la habitación, así como el propio aire absorben parte del sonido.

Claramente la onda reflejada puede rebotar varias veces en distintos lugares antes de llegar a nuestros oídos. Esta serie de rebotes es lo que le dará a nuestra reverberación una sensación espacial en una habitación.

Una media utilizada para caracterizar la reverberación en una habitación es el denominado tiempo de reverberación. Que técnicamente es el tiempo que le lleva al nivel de presión del sonido en decaer 60 dB de su valor original.

Pues así tiempos de reverberación largos indican que la energía del sonido se mantiene dentro de la habitación por mayor tiempo antes de ser absorbida. El tiempo de reverberación se encuentra fuertemente ligado a dos factores, las superficies de la habitación y el tamaño de la misma. Así las superficies de la habitación determinan cuanta energía se pierde en cada reflexión. De esta manera algunos materiales altamente reflectantes como ladrillos y ventas incrementan este tiempo de reflexión. Todo lo contrario de otros materiales absorbentes como cortinas, las cuales reducirán el tiempo de reverberación.

Cabe indicar que también existe absorción por parte del aire, donde en los microprocesadores a la hora de simular esta absorción suelen incorporar un filtro paso bajo [16].

Teniendo en cuenta todo lo expuesto, la figura 23, muestra el modelo de reverberación a implementar, donde se incluye un filtro Butterworth para cumplir la funcionalidad de absorción del aire.

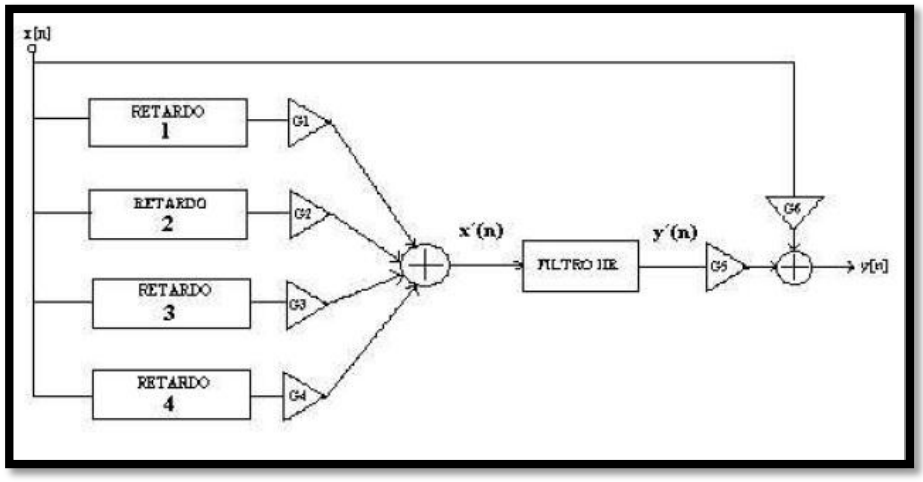
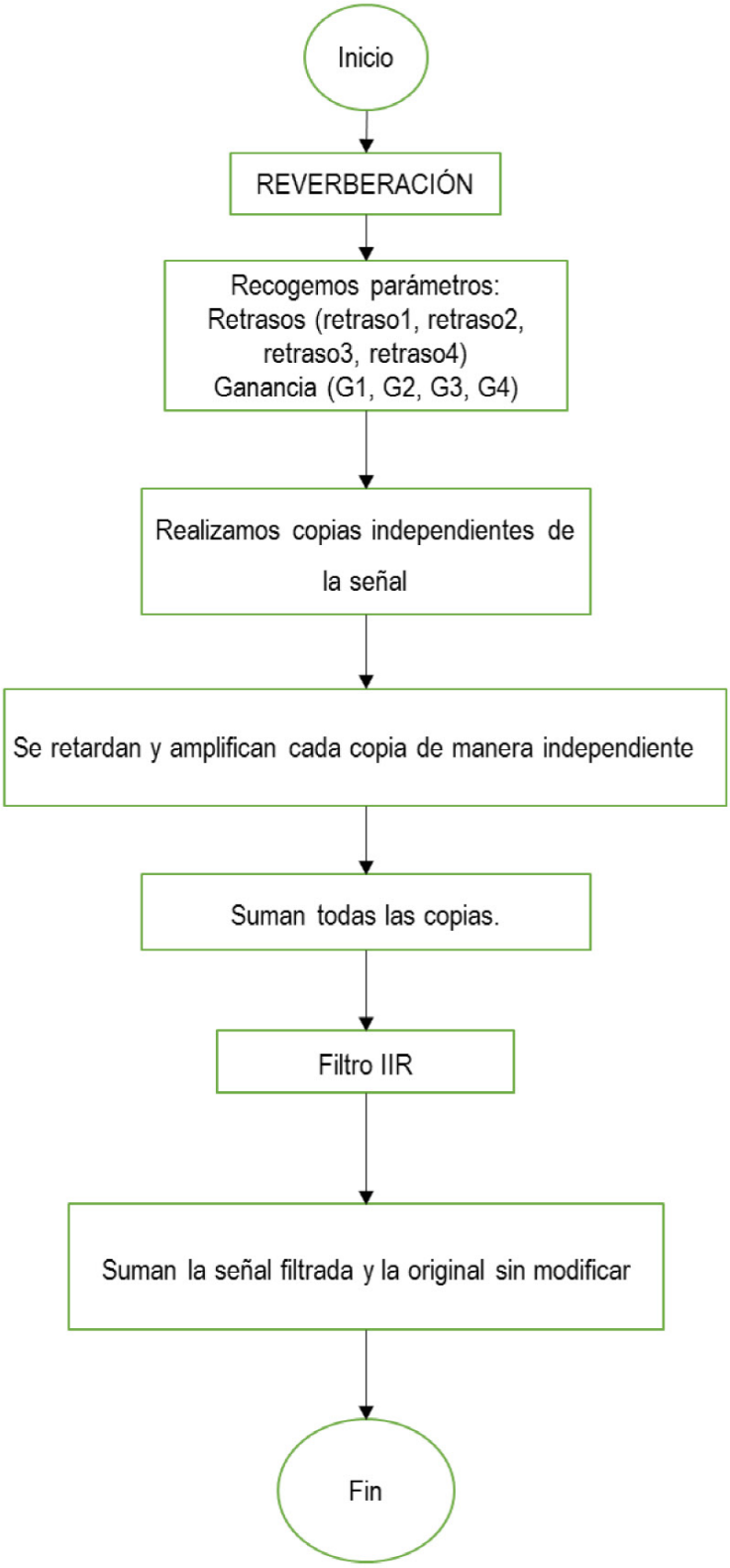


Figura 23. Diagrama de bloques de un reverberador [6].


Inicio
REVERBERACIÓN
Recogemos parámetros:
Retrasos (retraso1, retraso2,
retraso3, retraso4)
Ganancia (G1, G2, G3, G4)
Realizamos copias independientes de la señal
Se retardan y amplifican cada copia de manera independiente
Suman todas las copias.
Filtro IIR
Suman la señal filtrada y la original sin modificar
Fin

Lo que se puede apreciar en la figura 25 como filtro IIR no es más que un filtro pasó bajo de tercer orden del tipo Butterworth, para el cual su función de transferencia es la siguiente [16]:

$$y'(n) = 0.4y'(n-1) - 0.2499y'(n-2) + 0.0441y'(n-3) + 0.5814x'(n-1) + 0.2142x'(n-2)$$

Como puede apreciarse en la figura anterior, nuestra reverberación se compone de cuatro reflexiones o cuatro copias de la señal original, las cuales serán atenuadas por un coeficiente de ganancia entre 0-1 y tras ser sumadas y filtradas se añadirán a la señal original.

Para implementar esto en un microprocesador, el usuario deberá de introducir cada uno de los 4 tiempos de retraso, conviene que sean tiempos diferentes entre sí para que el efecto quede más realista.

Tras esto se deberán de introducir las ganancias de cada señal retardada, indicando que cuanto mayor sea la atenuación es como si proviniera de un lugar más lejano. Esto hace que el tiempo de retardo de cada muestra y la ganancia vayan ligeramente ligadas, es decir, conviene que para que el efecto quede más realista si retardo1<retardo2<retardo3<retardo4 la ganancia de dichas copias sea g1<g2<g3<g4. Consiguiendo así un efecto más real. Aunque no tiene por qué ser así estrictamente.

Tras esto estas cuatro señales serán sumadas en una misma señal, la cual ira directamente al filtro, donde serán filtradas según la ecuación expuesta anteriormente del filtro paso bajo, y tras esto se le aplicara de nuevamente otra atenuación, esto es así para que suene más tenue que la señal original, la cual como se muestra en la figura va atenuada.

Dicha atenuación para la señal original se decidió omitirla de tal manera que se apreciara más la diferencia entre señal original y señal reflejada.

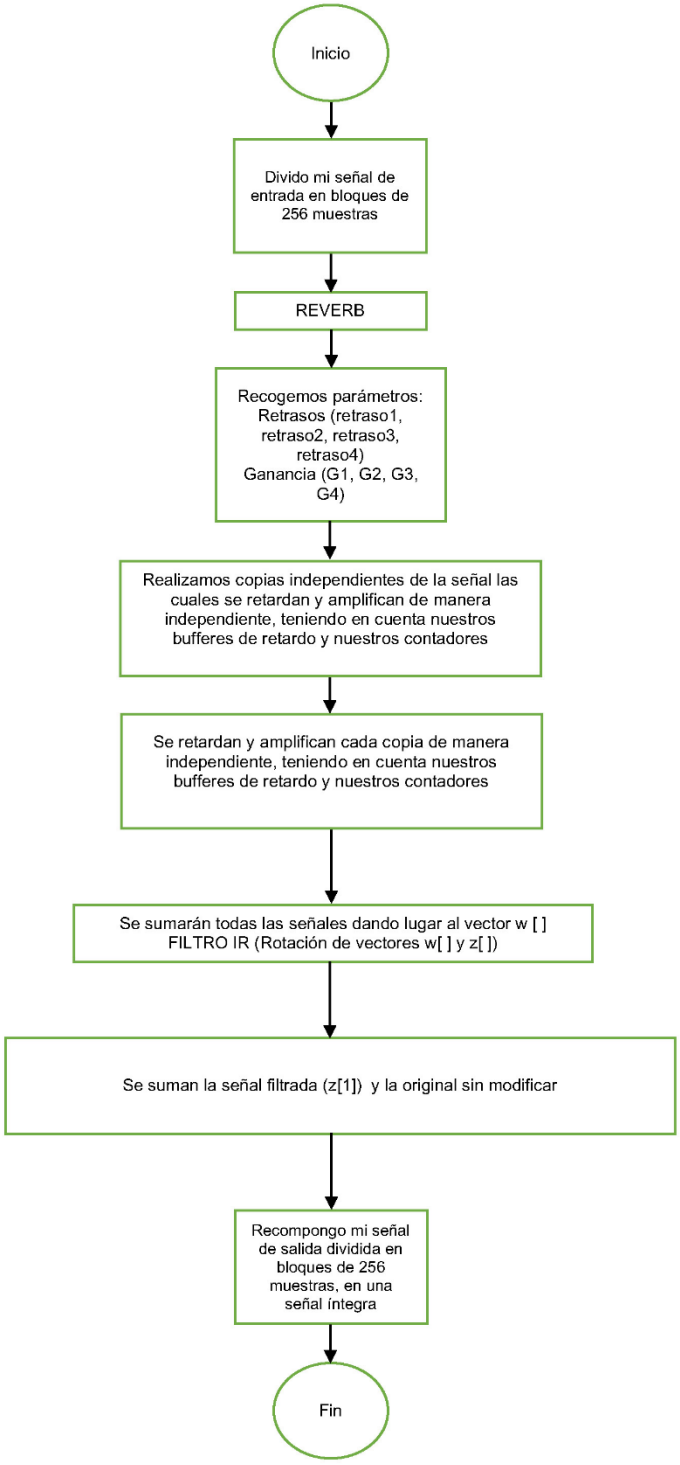
También indicar que el tiempo de atenuación de cada señal como ya se ha comentado otra varias veces anteriormente deberá ser convertido a muestras multiplicándolo por la frecuencia de muestreo de nuestra señal [16].

Quedando así nuestro efecto de reverberación completamente definido, siendo los parámetros a introducir por el usuario:

- Tiempo de retraso de la copia 1.
- Tiempo de retraso de la copia 2.
- Tiempo de retraso de la copia 3.
- Tiempo de retraso de la copia 4.
- Ganancia de la copia 1(G1).
- Ganancia de la copia 2(G2).
- Ganancia de la copia 3(G3).
- Ganancia de la copia 4(G4).

5.10.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto REVERB para su adaptación a un entorno de tiempo real:



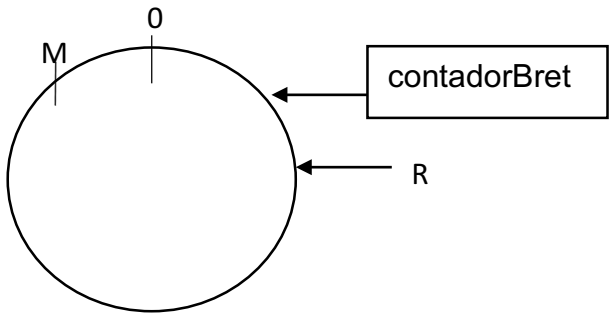
Tendremos que tener en cuenta la creación e inicialización de mi vector de retardo (bret), cuyo tamaño vendrá dado por el retraso convertido a muestras introducido por el usuario.

El retraso utilizado para implementar el vector de retardos será calculado como el máximo de los retardos de cada copia de señal retardada.

También será importante destacar los casos límite presentes en el código original, y que por tanto tendrán que ser implementados también en nuestro código modificado para la adaptación a tiempo real que me facilitará la posterior exportación a C++.

La necesidad de crear el buffer circular genera la problemática de tener en cuenta una serie de casos límite que deben de ser presentados para acercarnos cuanto más podamos al efecto original consiguiendo así un error relativamente pequeño a pesar de movernos en un entorno de tiempo real.

En el siguiente esquema podemos observar la problemática del buffer circular (suponiendo que mi buffer tiene un tamaño M y el retardo calculado es R):



Esquema representando buffer circular “bret”

Como podemos observar hay representado una de las problemáticas (casos límite) de hacer uso del buffer circular y que está presente de manera común en los efectos que hacen uso de este buffer. En el esquema vemos que el buffer tiene un tamaño M, y el 0 representa la primera posición de mi vector. Como vemos contadorBret está por debajo del retardo calculado, esto generará un problema puesto que se querrá acceder a una posición negativa (teniendo en cuenta que accedemos a la posición *contadoBret* – *R* de nuestro vector bret), dicho de otra manera el retardo R calculado valdrá 125 cuando el contador va mas retrasado, p.ej va por el valor 45. Esto se dará de manera repetida en cada iteración y por tanto será necesario estos cambios. Existirán mas casos límite, todos ellos están expuestos en el pseudocódigo que sigue a continuación, debemos de tener en cuenta que los casos limites se verán reducidos en otros efectos, pero aun así se deberán de contemplar en nuestro código.

Los casos límite surgirán de la necesidad de utilizar el buffer circular, por ello, en las cuatro copias de señal retardadas se realizarán los mismo casos limite. Posteriormente se rotaran los vectores w[ ] y z[ ] que son los que me permitirán implementar el filtro necesario para realizar la reverberación.

Detallamos el seudocódigo de reverberación:

Si (contadorBret-delay1)==0

señal_de_salida(n) = (bret((retardo-1)+(contadorBret-delay1)))

Si (contadorBret-delay1)<0

señal_de_salida(n) = (bret(retardo+(contadorBret-delay1)))

Si no se aplicó ningún caso anterior:

señal_de_salida(n) = (bret(contadorBret-delay1))

(Se realiza el mismo procedimiento con cada una de las cuatro copias de señal)

Finalmente se realizaría el filtro y se sumaría la señal filtrada y la original sin modificar.

5.11 Noise Gate

5.11.1. Introducción

El noise gate será el único efecto que se implemente de los conocidos como reductores de ruido, aunque también sea considerado como un efecto de distorsión, a pesar de funcionar de manera totalmente contradictoria a una distorsión, es decir el efecto ignora o anula partes de una señal por debajo de un umbral indicado, totalmente lo contrario a cómo funcionaba la distorsión, donde la señal tratada era la comprendida por encima del umbral [6].

Esto hace así que los sonidos más fuertes o partes de un sonido fuerte que sean los que únicamente puedan pasar a través de la unidad de distorsión [16].

5.11.2. Implementación

Como ya se ha mencionado anteriormente el noise gate se basa en la eliminación de algunos sonidos, atendiendo a ello a un valor umbral indicado por el usuario. Para verlo más claramente la figura 24

muestra un ejemplo de cómo los sonidos más fuertes pasarían mientras que los sonidos débiles serian eliminados.

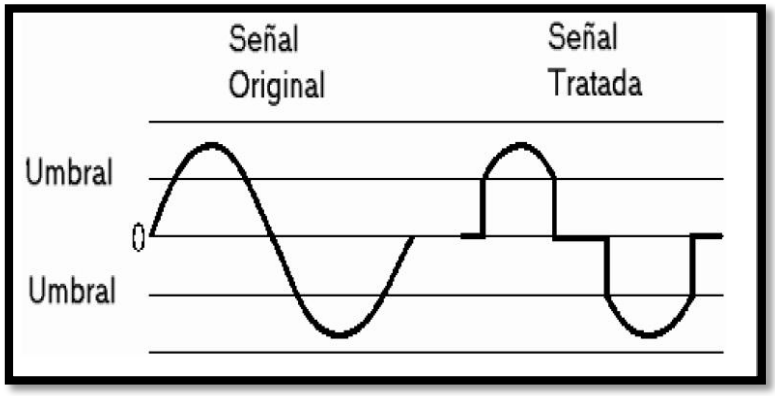


Figura 24. Ejemplo de noise gate con una señal que pasa el umbral frecuentemente [6].

Observando la figura anterior podemos ver como los valores por debajo del umbral o sonidos débiles son suprimidos, mientras que los valores por encima del umbral se dejan pasar de manera inalterable.

Por el contrario la figura 25 muestra como reaccionaria la puerta de ruido ante una señal donde todo fueran sonidos débiles y se encontraran por debajo del umbral.

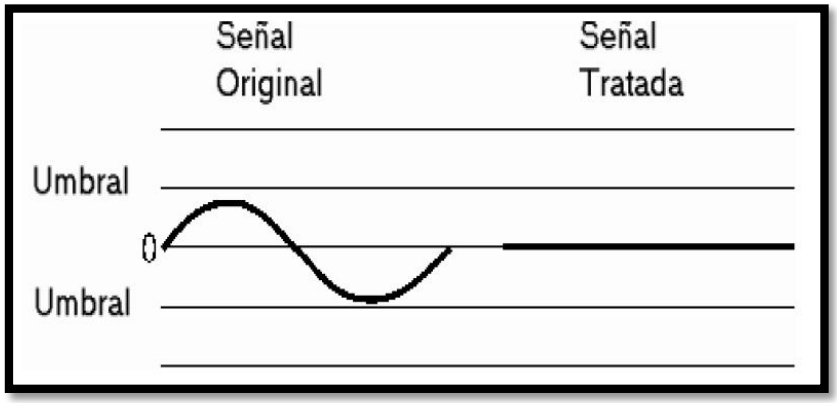
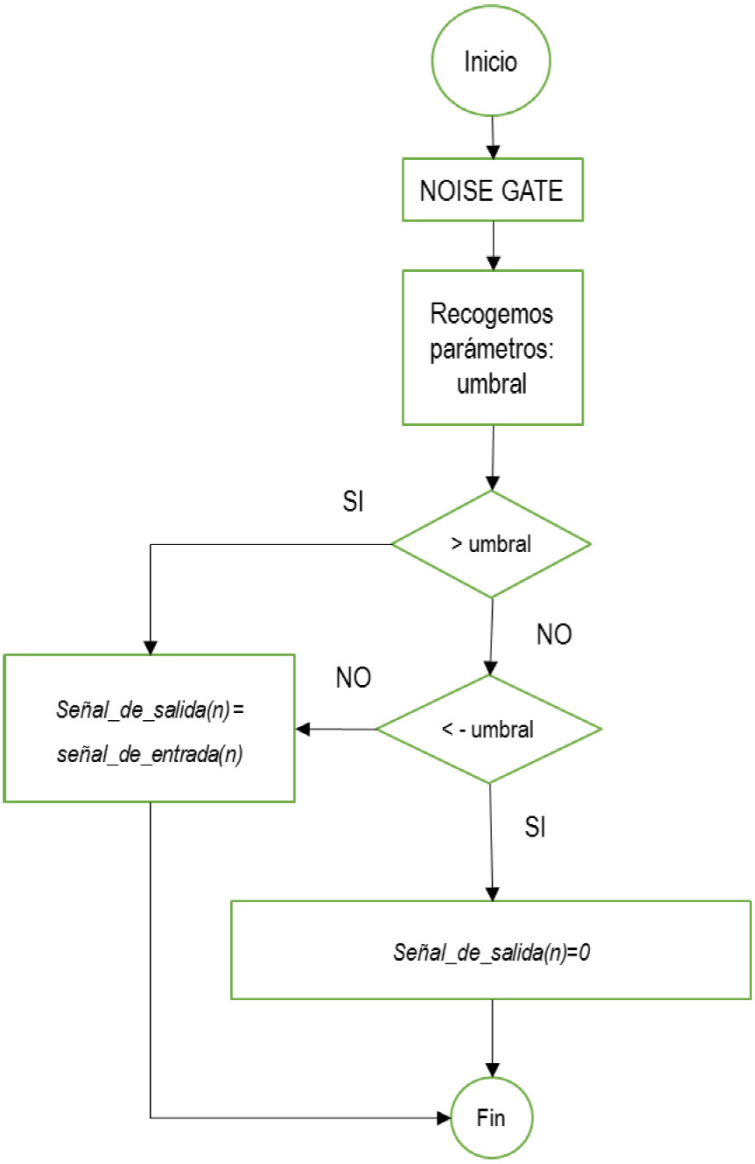


Figura 25. Ejemplo de noise gate con una señal débil [6].



Así pues una vez el usuario introduzca el valor umbral, dicho umbral será adaptado para que sea el modulo del valor máximo de la señal, tras esto el noise gate deberá cumplir las siguientes ecuaciones [16]:

Si +umbral > señal_de_entrada(n) < -umbral

Señal_de_salida(n) = 0.

Si no secumple la condición de que la señal no está comprendida entre el umbral entonces se aplicara [16]:

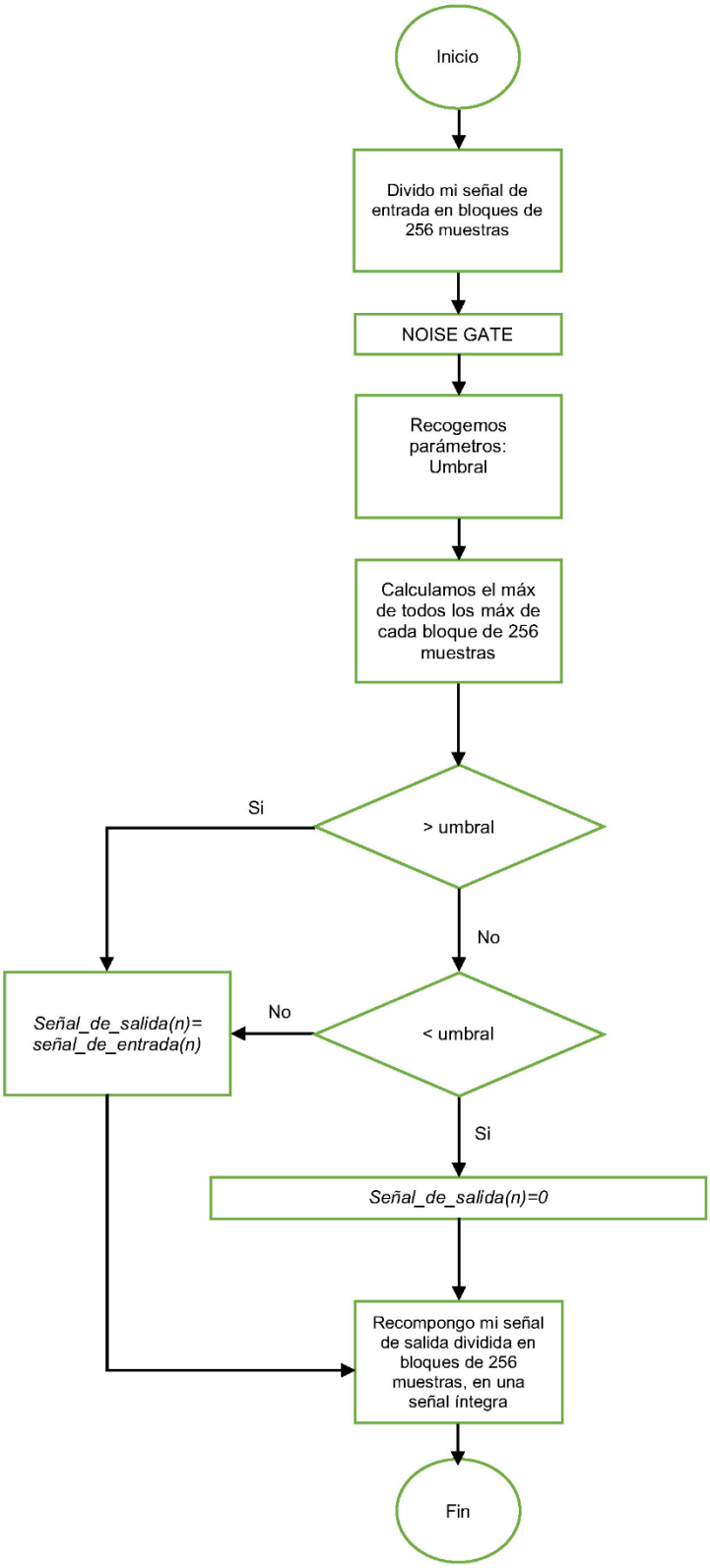
Señal_de_salida(n) = señal_de_entrada(n).

Siendo los parámetros a introducir por el usuario:

- Umbral

5.11.3. Adaptación a tiempo real

Detallamos el diagrama de flujo modificado del efecto REVERB para su adaptación a un entorno de tiempo real:



En estos efectos (FUZZ, EXPANSOR, COMPRESOR, DISTORSION y NOISE GATE) si que debemos de destacar la presencia de una variable que me permite calcular el máximo de cada uno de los bloques de 256 muestras, para posteriormente buscar el máximo absoluto de todos los bloques que conforman mi señal, dicho de otra manera, calculo el máximo de los máximos de cada bloque, obteniendo así el valor deseado.

6. MATERIALES Y METODOS

6.1 Interfaz de guitarra y conexionado

Para poder realizar el conexionado de mi guitarra eléctrica a nuestro dispositivo IPAD, tendremos que hacer uso de una interfaz que me permita introducir por el canal externo de mi dispositivo la señal de la guitarra eléctrica.

Para ello hacemos uso de la interfaz de audio para guitarra iRig.

iRig nos permitirá conectar nuestra guitarra/bajo a un dispositivo iOS. Para usar iRig, simplemente colocaremos el conector TRRS de 3,5 mm en la toma de auriculares de nuestro dispositivo iOS. Luego conectaremos nuestra guitarra a la toma de 1/4 pulg. usando un cable de guitarra estándar. Para monitorear el sonido, podremos conectar unos auriculares/línea de salida a la toma de auriculares de 3,5 mm en iRig.

Figura 26. Conexionado mediante iRig a dispositivos iOS [14].

Así, obtendré en el IPAD una señal limpia de la guitarra, como podremos apreciar en la aplicación realizada, en la cual hay una función que me permite sacar el sonido limpio de la guitarra sin aplicar ningún filtro ni efecto.

Como también podemos apreciar, el proprio fabricante (IK Multimedia) nos recomienda utilizar un software para iOS denominado “Amplitube”.
Este software realizará el mismo procedimiento que nuestra aplicación puesto que se basa en la aplicación de varios efectos implementados de la misma manera que en nuestra aplicación.

A la hora de realizar el TFG, hemos necesitado conectar la interfaz de guitarra a un ordenador Mac Book Pro. De esta manera se ha ido comprobando efecto por efecto la correcta implementación y el buen funcionamiento de la aplicación antes de pasarla a nuestro dispositivo IPAD.

Finalmente el proyecto ha sido realizado tanto para Mac como para ejecución en IPAD, siendo una aplicación propia para estos dispositivos.

Figura 27. Conexionado mediante iRig a Mac Book [14].

Finalmente realizaremos un resumen de las características de nuestro adaptador de interfaz de guitarra e instrumentos para iPhone/iPod touch/iPad:

- Adaptador de interfaz de instrumentos + software móvil de tonos de guitarra y bajo
- Funciona en iPhone/iPod touch/iPad
- Toma de entrada compacta hembra de 1/4 pulg. a adaptador de interfaz de audio de toma de salida hembra de 1/8 pulg.
- Auricular/Amplificador/Altavoz de potencia de SALIDA
- Conecta la guitarra eléctrica y el bajo
- También funciona con señales de nivel de línea de sintetizadores, teclados, mezcladoras
- Grabación de efectos múltiples en tiempo real para guitarra y bajo + aplicación para práctica

7. RESULTADOS Y DISCUSIÓN

7.1 Trabajo final obtenido

El trabajo final obtenido será una aplicación en IPAD por medio de la cual capte (mediante una interfaz de guitarra) la señal limpia de mi guitarra, y sobre esta pueda aplicar efectos, modificando así la señal original de tal manera que se pueda obtener el efecto deseado a través de algoritmos facilitados al principio del desarrollo de este proyecto.

Como hemos mencionado anteriormente, partimos de 11 efectos facilitados al principio del desarrollo de este proyecto, los cuales fueron implementados en totalidad en el lenguaje de programación MATLAB.
Debemos de tener en cuenta que los efectos facilitados estaban implementados bajo el requerimiento de conocer a priori la señal de entrada. Es aquí, donde surge la dificultad principal del proyecto, puesto que se deben implementar estos mismos efectos pero con las modificaciones oportunas para que se puedan ejecutar en tiempo real.

En un principio, realizamos una simulación en MATLAB, realizando las modificaciones oportunas para simular un entorno en tiempo real. Debíamos de tener en cuenta que en el entorno de openFrameworks, hacemos uso de funciones de entrada y salida de audio en tiempo real, con bufferes de 256 muestras, esto quiere decir que nuestras funciones captan y devuelven la señal acústica en bloques de 256 en 256.

Así pues, tuvimos que desarrollar un entorno en MATLAB que simulara esta división de la señal de entrada en bloques y la posterior reconstrucción de la señal de salida. Además teniendo en cuenta este procedimiento, el algoritmo original se veía modificado en mayor grado en los efectos con memoria (DELAY, CHORUS, FLANGER, TREMOLO, VIBRATO y REBERB) y en un menor grado en los efectos sin memoria (COMPRESOR, EXPANSOR, FUZZ, DISTORSION y NOISE GATE).

Por tanto los resultados reflejados en esta memoria, serán simplemente la comparativa de alguno de los efectos con memoria, que como hemos visto son los más modificados, para examinar meticulosamente la diferencia entre los efectos implementados originalmente, y los posteriormente modificados.

Tenemos que aclarar, que el hecho de realizar este entorno de tiempo real en MATLAB, ha facilitado en gran manera la realización de los efectos en openFrameworks, en concreto bajo C++.

El efecto que reflejáramos gráficamente para observar las diferencias entre el efecto original y el modificado será un efecto con memoria, puesto que los efectos sin memoria, apenas sufren modificaciones y por lo tanto si los representáramos gráficamente no habría ninguna diferencia.

Por tanto representamos gráficamente el efecto FLANGER, para observar las diferencias más llamativas y la magnitud del error que se produce al aplicar esos cambios en el efecto original:

En primer lugar representaremos en una misma gráfica ambas señales (efecto original y efecto en tiempo real), la señal en rojo será el efecto original y la señal coloreada en azul será el efecto simulado en un entorno de tiempo real y con las modificaciones oportunas:

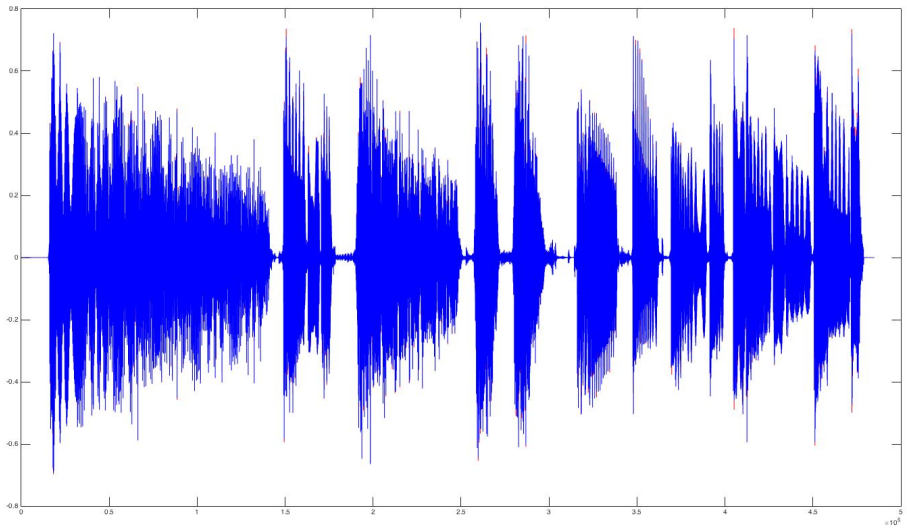


Figura 28. Efecto FLANGER (Original y modificado)

Como podemos observar la diferencia entre las dos señales es casi inapreciable, tanto es así que hemos tenido que ampliar una zona de la las dos señales para apreciar los cambios:

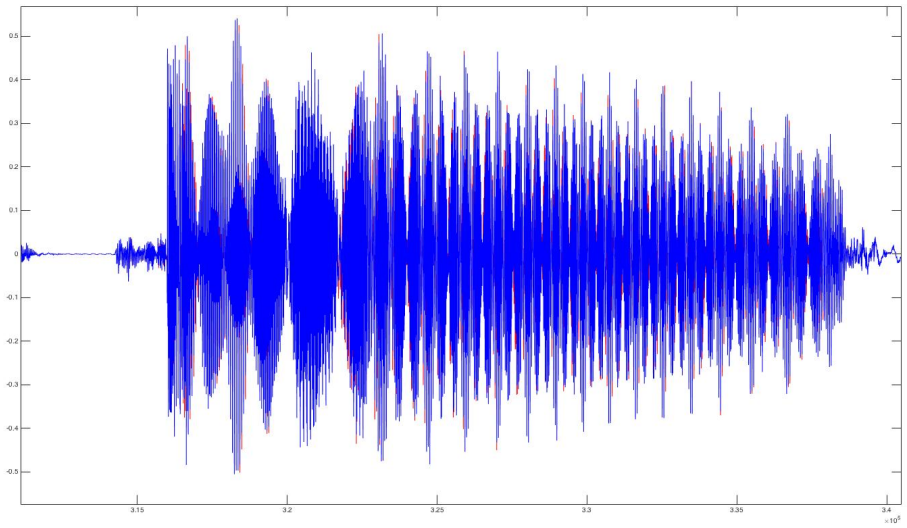


Figura 29. Efecto FLANGER (Original y modificado) ampliado

En la figura 29 si que podremos apreciar unas diferencias aunque pequeñas pero mas notables, sobre todo en los valores de amplitud, esto es debido a los casos límite que podemos analizar en el efecto en sí y que fueron detallados anteriormente en la memoria para cada efecto.

No todos los casos límite que se pueden observar en el efecto original pueden transcribirse en el efecto modificado, pues de esta manera incumpliríamos las premisas necesarias para simular el efecto en tiempo real, dicho de otra manera el hecho de que simulemos un entrono en tiempo real obligatoriamente nos hace modificar nuestros efectos y por lo tanto generar esa diferencia.

El error que obtenemos en los efectos adaptado a tiempo real, en conclusión se genera por la obtención de máximos (efectos SIN MEMORIA) y la utilización de un buffer circular (efectos CON MEMORIA). Como hemos dicho anteriormente el hecho de hacer uso de este buffer, implícitamente genera la necesidad de presentar una serie de casos limite para que el efecto se adapte lo mejor posible a tiempo real sin perder la originalidad acústica que podemos escuchar en el efecto original. Esto producirá este pequeño error que podemos ver reflejado en la gráfica.

Finalmente graficamos la diferencia de ambos efectos, donde podemos observar la magnitud del error:

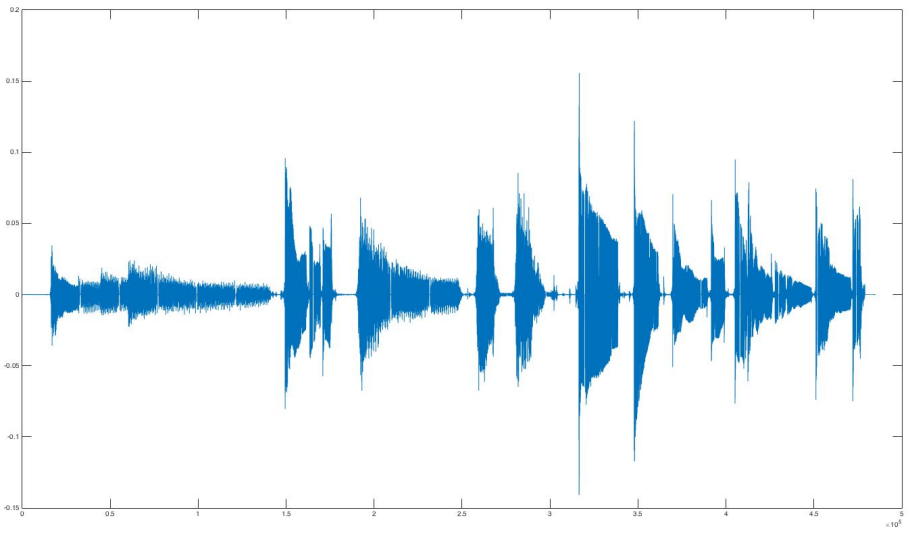


Figura 30. Diferencia entre FLANGER original y FLANGER modificado

Como podemos observar la diferencia es bastante pequeña, siendo el error de una magnitud muy minoritaria, y reflejando así el hecho de que en todos mis efectos con memoria, existirán diferencias, pero reducidas al máximo, para que acústicamente el efecto suene similar al original, aunque analíticamente si existan pequeñas diferencias.

La magnitud del error de los efectos con memoria restantes no será muy diferente al presente en este efecto, por ello el hecho de utilizar este único ejemplo para detallar el resultado analítico de aquellos efectos que si sufrirán alguna modificación (CHORUS, FLANGER, REVERB, DELAY, TREMOLO y VIBRATO).

7.2 Realización para dispositivo Mac OSX

Desde el comienzo del proyecto nos informamos del entorno necesario para ejecutar la aplicación, y elegimos la librería de openFrameworks para OSX. Por tanto una de las dos aplicaciones que obtendremos será para la ejecución en OSX y posteriormente hicimos los cambios oportunos para la ejecución en IPAD (iOS).

La implementación de los efectos en sí no cambiaría, aunque si habría otras variaciones en la interfaz y en las funciones para captar y sacar audio al utilizar dos librerías diferentes: la de desarrollo para aplicaciones en OSX y la de desarrollo para iOS.

Por tanto obtendremos una aplicación para sistemas operativos OSX, realizando los requisitos pedidos en el proyecto y además la aplicación modificada para la realización del banco de efectos sobre dispositivos IPAD.

Se realizará una interfaz gráfica para poder seleccionar cada uno de los efectos y a su vez poder modificar los parámetros oportunos para cada efecto, obteniendo así una aplicación en tiempo real que realice el procesado de señal dando como resultado el correcto funcionamiento de estos efectos. Cabe destacar que habrá una diferencia mínima entre interfaz para iOS y de OSX, por eso representemos en la figura 31 una de ellas.

En la siguiente figura mostramos la interfaz gráfica de la aplicación sobre dispositivos Mac con sistema operativo OSX:

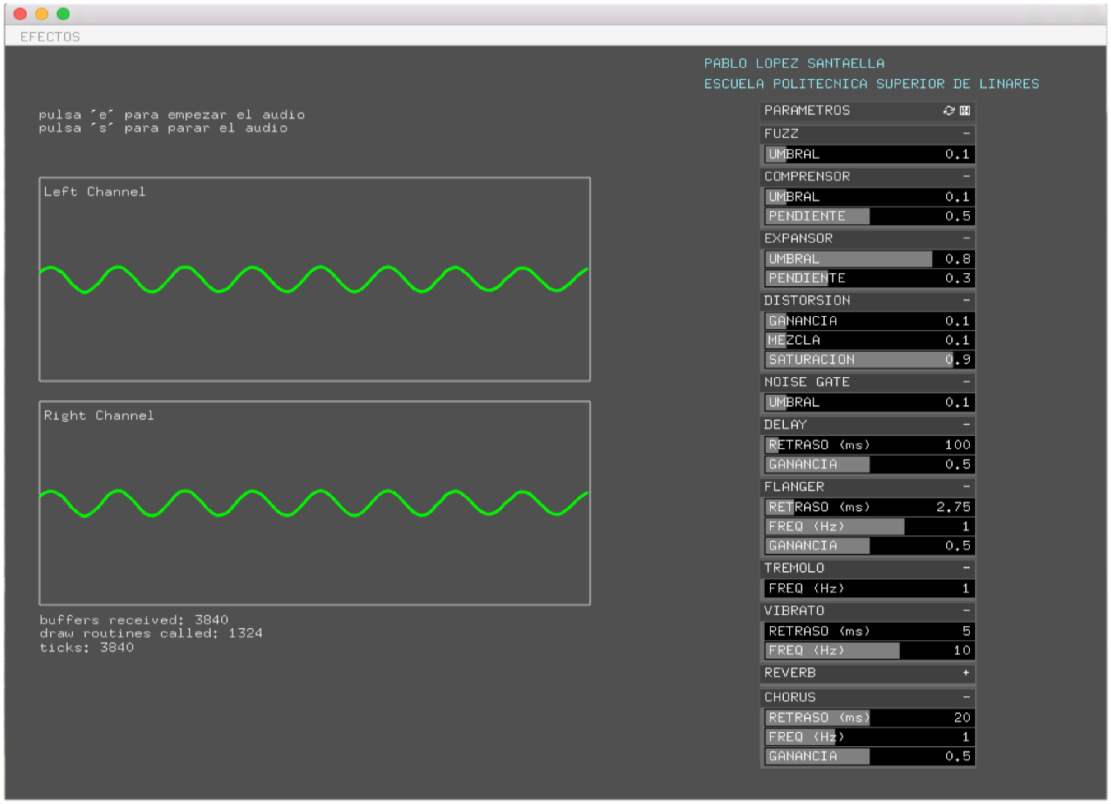


Figura 31. Interfaz gráfica

7.3 Líneas de futuro

Algunas de las líneas de futuro son:

- La posibilidad de ejecutar varios efectos de manera simultánea, puesto que en la práctica podemos comprobar que los guitarristas hacen uso de varias combinaciones de pedales, tales como COMPRESOR + FUZZ, DELAY + CHORUS, y otras combinaciones usuales para conseguir el efecto acústico deseado.

- Controlador de volumen MASTER.

- Posibilidad de grabar fracciones acústicas (LOOP).

- Implementación de efectos sin memoria como OVERLAY u otros tipos de distorsiones.

- Implementación de metrónomo para la ejecución simultánea a los efectos.

- Posibilidad de guardar un efecto modificado para su posterior utilización.

8. CONCLUSIONES

- Como hemos mencionado anteriormente, la aplicación final obtenida será un banco de efectos, en total 11 efectos que se aplicarán a la señal de guitarra limpia, y que se ejecutará en un ordenador Mac con sistema operativo OSX preferentemente y sobre dispositivos IPAD.

- La implementación en tiempo real partiendo de los 11 efectos ya facilitados, y el desarrollo de cada uno de estos efectos para su correcto funcionamiento.

- Se facilitará la modificación y acceso a cada uno de los efectos mediante la interfaz gráfica implementada, en conclusión la sencillez de la interfaz para un correcto funcionamiento de la aplicación en tiempo real.

- Además de los factores comunes en el procesado digital de sonido, se dieron a conocer otros factores que se pueden aprovechar al utilizar openFrameworks y se demostró la facilidad con que se pueden ejecutar algoritmos que, en cualquier otro lenguaje, serían mucho más complicados de implementar o presentar similar eficiencia, como son los algoritmos de entrada y salida de audio y la detallada librería de procesado de audio que nos facilita este desarrollador.

9. PLIEGO DE CONDICIONES

9.1 Requisitos de la aplicación

El trabajo de fin de grado desarrollado, necesita de un dispositivo (Mac e IPAD) que soporte la plataforma Mac OS y iOS la cual debe de presentar una versión reciente de dicho sistema operativo para su correcto funcionamiento.

Como se ha nombrado anteriormente, la aplicación ha sido probada a lo largo de todo el proceso en un Mac Book Pro 10.10.5, con una velocidad de CPU de 2,7 GHz. Se ha podido probar para IPAD (modelos diversos) mediante el simulador que nos facilita el entrono de programación de Xcode. Por tanto el funcionamiento es correcto para ambas plataformas.

9.2 Características del dispositivo de desarrollo

A continuación se muestran las características aportadas por el fabricante del dispositivo utilizado para la realización de todo el desarrollo de la aplicación. Se ha trabajado con un Mac OS 10.10.5:

Procesador	Turbo Boost hasta 3,1 Ghz Intel Core i5
Velocidad de procesador	2,7 GHz
Tipo de nivel	3
Sistema operativo	OS X Yosemite
Tipo de pantalla	Pantalla Retina: retroiluminada por LED con tecnología IPS
Tamaño de la pantalla	13,3 "
Resolución	2500x1600 ppp
Memoria Ram	8 GB LPDDR3
Ampliación de memoria	16 GB
Capacidad de disco duro	128 GB SSD
Tarjeta gráfica	Intel Iris Graphics 6100
Tarjeta gráfica dedicada	

	Gb
Tipo de altavoces	
	Estéreo
Entradas / salidas de audio	
	1 Toma para auriculares
Lan inalámbrica	
	Sí
Tipos de Lan inalámbrica	
	IEEE 802.11 a,b,g,n,ac
Bluetooth	
	Bluetooth 4.0
Puertos entrada / salida	
	1 toma de corriente MagSafe 2 2 Thunderbolt 2 2 USB 3.0 1 HDMI
Tipo de teclado	
	Retroiluminado de tamaño estándar y sensor de luz ambiental
Número de teclas del teclado	
	79
Teclas multimedia	
	12 teclas de función 4 de flecha (dispuestas en forma de T invertida)
Tipo de ratón	
	Trackpad Force Touch
Webcam	
	Sí
Resolución de la webcam	
	720 p
Alimentación	
	Adaptador de corriente MagSafe 2 de 60 W
Especificación de baterías	
	Polímero de litio
Tipo de baterías	
	Batería
Audio	
	Compatible con los auriculares con mando y micro de Apple para el iPhone. Compatible con salida de audio (digital y analógica).
Gráficos y compatibilidad de vídeo	
	Doble monitor y vídeo en espejo: admite simultáneamente la resolución nativa completa en la pantalla integrada y hasta

	3840 por 2160 píxeles en hasta dos monitores externos, ambos con capacidad para millones de colores.
Cámara	
	Cámara FaceTime HD 720p.
Requisitos eléctricos y de funcionamiento	
	Tensión: de 100 a 240V de CA. Frecuencia: de 50 a 60 Hz. Temperatura de funcionamiento: de 10 a 35ºC. Temperatura de almacenamiento: de -25 a 45ºC. Humedad relativa: del 0 al 90% sin condensación. Altitud de funcionamiento: probado hasta 3000m. Altitud máxima de almacenamiento: 4500m. Altitud máxima de transporte: 10500m.

10. ANEXOS

10.1 Anexo I. Manual de usuario

Este manual explica al usuario como utilizar la aplicación para un correcto funcionamiento. No es necesario que el usuario sea experimentado en manejo de pedales o efectos de guitarra eléctrica, puesto que se ha implementado una interfaz que sea lo suficientemente práctica y sencilla para cambiar de efecto y modificar los valores deseados de ellos sin dificultad. El funcionamiento para dispositivos IPAD será prácticamente similar a la explicación detallada para Mac OS, así mismo la interfaz tendrá el mismo funcionamiento salvo en la descripción de las teclas las cuales no estarán presentes.

Descripción de las teclas (key pressed):

‘e’ : Si se marcó, el audio sigue reproduciéndose y en el caso en que lo hayamos parado, comienza de nuevo a reproducirse.
‘s’ : Si se marcó, el audio cesa de reproducirse.
‘h’ : Si se marcó, muestra o esconde los sliders.

Como se detalló en anteriores puntos de la memoria, hemos hecho uso de la interfaz de audio iRig para conectar mi dispositivo (Mac en nuestro caso) a la guitarra electrivca, y de esta manera conseguir tener señal en nuestra aplicación.

a. Así que el primer paso sería simplemente conectar el iRig a nuestra guitarra y posteriormente a nuestro Mac.

b. Iniciaríamos nuestra aplicación y observaríamos como la señal presente es una señal limpia de nuestra guitarra eléctrica:

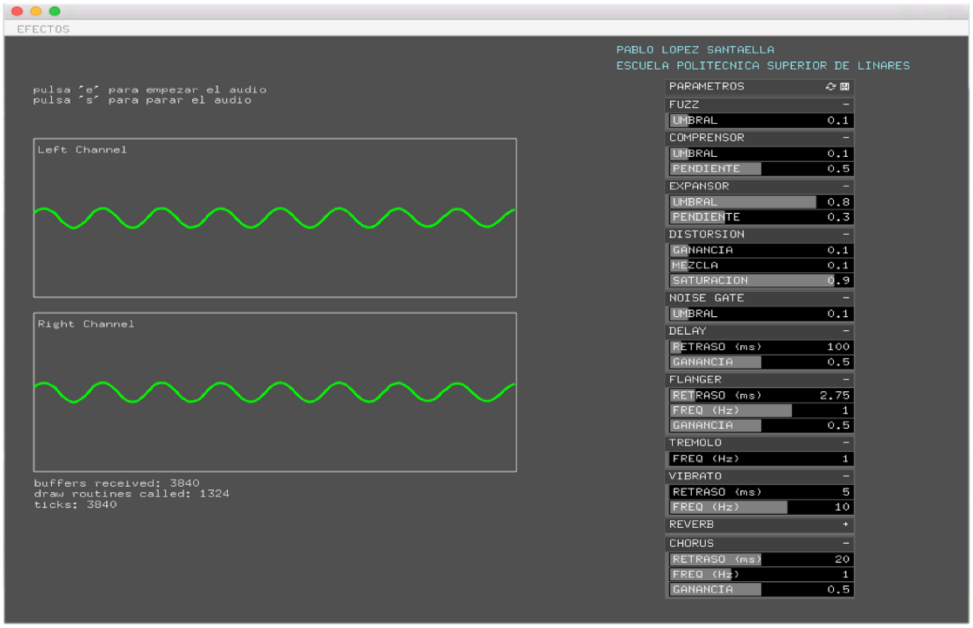


Figura 32. Interfaz de la aplicación (Efecto Clean)

c. Como podemos observar en nuestros sliders, hay valores prefijados para que desde el primer momento que se seleccione el efecto oportuno se obtenga el efecto acústicamente correcto o deseado para el músico.

d. Si hacemos “click/pulsamos” sobre la pestaña en la parte superior izquierda de mi aplicación, en la que vemos escrito EFECTOS, accederemos a un menú dropdown tal y como se detalla en la figura.
Allí podremos acceder a cada uno de los efectos implementados simplemente haciendo “click/pulsamos” sobre el. Para volver a recoger el menú, basta con clickar/pulsar de nuevo sobre la pestaña EFECTOS, y el menú quedará recogido.

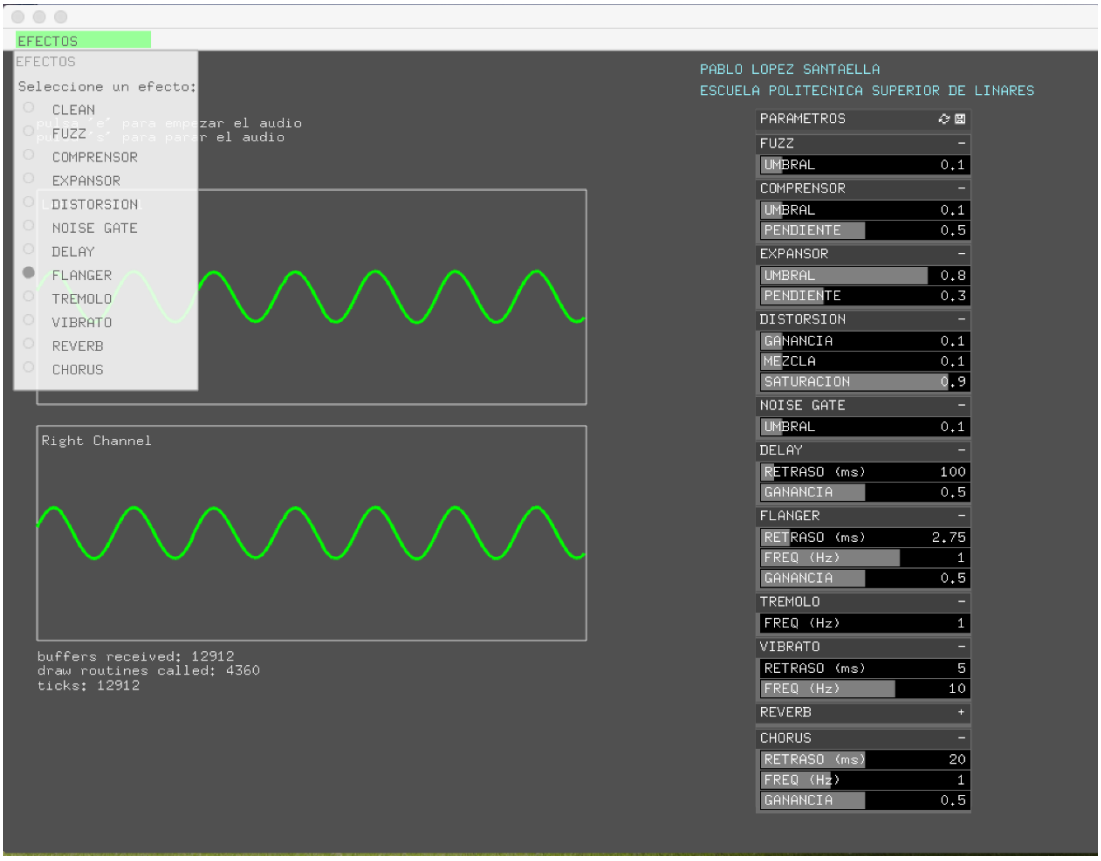


Figura 33. Interfaz gráfica con menu dropdown

e. En cuanto a los sliders simplemente tendremos que pinchar con nuestro ratón/dedo en las barras de modificación o arrastrar con nuestro ratón/dedo para observar como los valores cambian y poder ajustar el efecto acústico de la manera deseada. Podemos minimizar cada efecto con sus parámetros (en la figura está minimizado REVERB).

En todo momento deberemos tener en cuenta el nivel de volumen de nuestra guitarra, y sobre todo a la hora de acceder a algún efecto de distorsión como Fuzz, bajar el nivel del volumen para que no se acople.

Es interesante también explicar como debemos de compilar la aplicación una vez descargado openFrameworks.

Antes de nada debemos de presentar las dos librerías utilizadas en este TFG, y donde podemos descargarlas incluidos los manuales de usuario (este procedimiento es similar en los dos entornos):

- Librería openFrameworks para OSX:
  http://openframeworks.cc/download/

- Manual para OSX (entorno de Xcode):
  http://openframeworks.cc/setup/xcode/

- Librería openFrameworks para iOS:
  http://openframeworks.cc/download/

- Manual para iOS (entorno exclusivo de Xcode):
  http://openframeworks.cc/setup/iphone/

1) Una vez descargado openFrameworks, generaríamos un nuevo proyecto mediante `projectGenerator.app` incluyendo las tres librerías utilizadas en ambos entornos:

   - Interfaz gráfica (menú DROPDOWN):
     https://github.com/timknapen/ofxButtons

   - Sliders (librería propia de openFrameworks):
     /Users/usuario/Desktop/of_v0.9.3_ios_release/addons

   - XML Settings (opciones de restablecimiento de los sliders, librería propia de openFrameworks):
     /Users/usuario/Desktop/of_v0.9.3_ios_release/addons

   Una vez que hemos generado el proyecto con nuestras librerías incluidas, debemos de tomar el código adjunto a este trabajo enviado a la plataforma de ILIAS con nombre: *"Anexo II_Codigo.zip",* y ubicar este código en el directorio correcto de openFrameworks (/Users/usuario/Desktop/of_v0.9.3_ios_release/apps/myApps/ nombre_proyecto/src).

2) Sería necesario por tanto la instalación del entrono de programación gratuito Xcode.

3) Cuando hallamos hecho todos estos pasos, debemos de seleccionar (dentro de Xcode) el modo "debug" de nuestra app, esto es esencial para que la app pueda compilarse, sobre todo en el entorno de OSX, en el caso de iOS habría solo presente un

simulador y no haría falta seleccionar nada mas, solamente ejecutar.

4) Una vez hecho esto, solo tendríamos que presionar el botón de ejecutar y esperar que se compile.

Es importante tener en cuenta que para cada entorno deberíamos de hacer este procedimiento usando exclusivamente el generador de proyectos y librerías de openFrameworks para cada entorno, es decir, las librerías de entrada y salida de audio no son iguales para iOS que para OSX, al igual que todas las demás utilizadas en este trabajo.

10.2 Anexo II. Documentación técnica

Antes de detallar las funciones implementadas y su funcionamiento dentro del programa, representaremos un esquema con los conceptos básicos de openFrameworks (sobre Xcode) estudiados para realizar este proyecto.

Es importante destacar que tal y como hemos dicho en puntos anteriores, se ha realizado un desarrollo en dos plataformas, pero gracias a que openframeWorks muestra mucha similitud entre ambos entornos (a nivel de código) prácticamente es similar el desarrollo para ambas plataformas (funciones similares), solamente con pequeñas disyuntivas como la de la captación y muestra del sonido hacia el exterior la cual debe de realizarse en MONO a diferencia del desarrollo en ESTÉREO realizado sobre plataformas Mac OS. Por ello analizaremos el código y las funciones de uno de los dos entornos para no ser repetitivos.

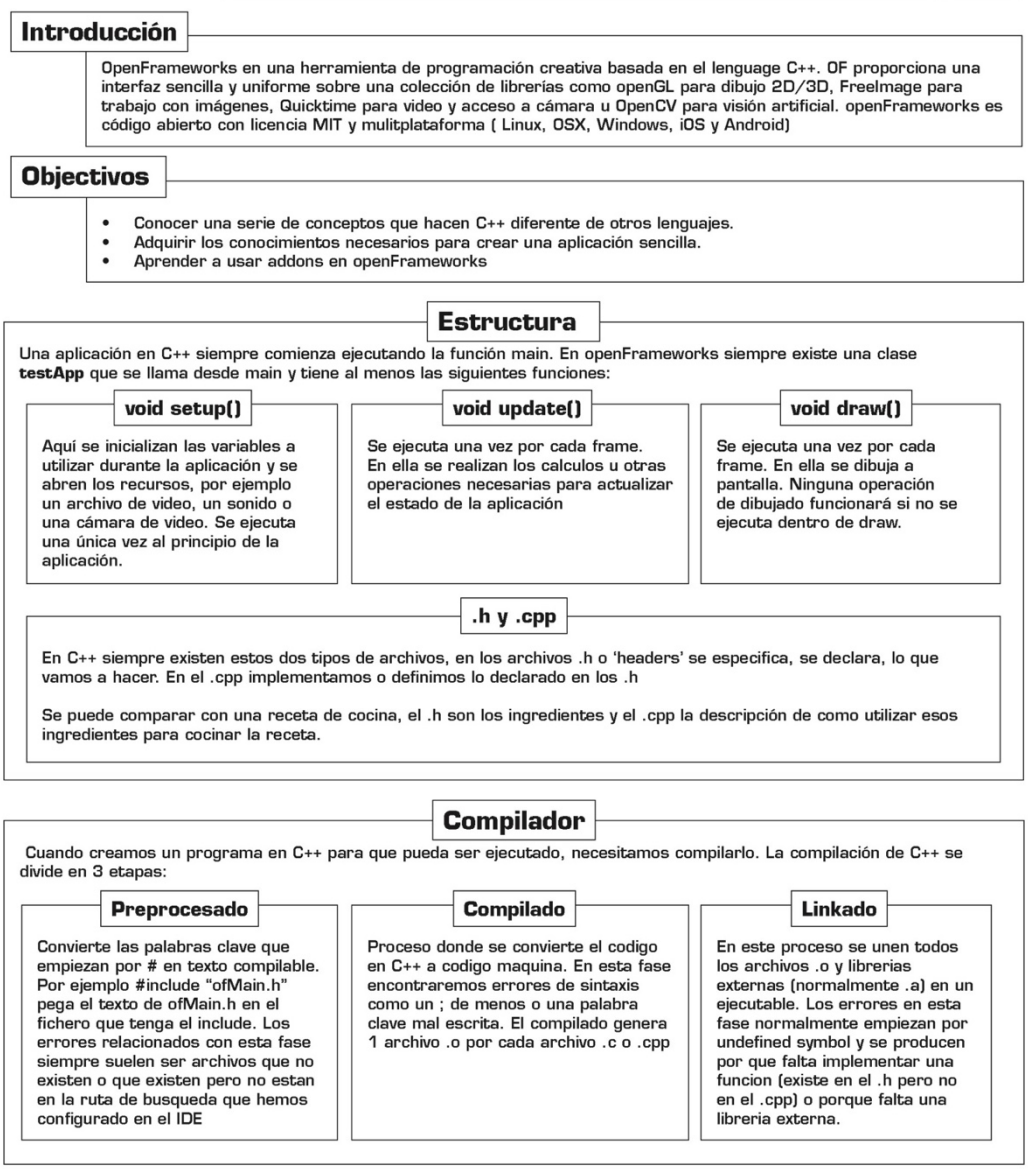


Figura 34.1 Esquema básico de openframeWorks (Desarrollo para Mac OS) [15]

Como hemos dicho anteriormente hemos ajuntado todo el código a este trabajo, enviado a la plataforma de ILIAS con nombre: *"Anexo II_Codigo.zip"*. En este comprimido hacemos presente el código para entorno iOS y para OSX, y dentro de este se ha comentado todo el código y funciones utilizadas.

La lista de archivos implementados para el entorno de OSX es:

src/main.cpp	Implementación del archivo principal de mi programa.
src/ofApp.cpp	Implementación de cada uno de los efectos e interfaz gráfica.
src/ofApp.h	Implementación de las librerías necesarias y la declaración de funciones y variable para el correcto funcionamiento de nuestro programa.

main.cpp

Librerías a utilizar:

```
#include "ofMain.h"
#include "ofApp.h"
```

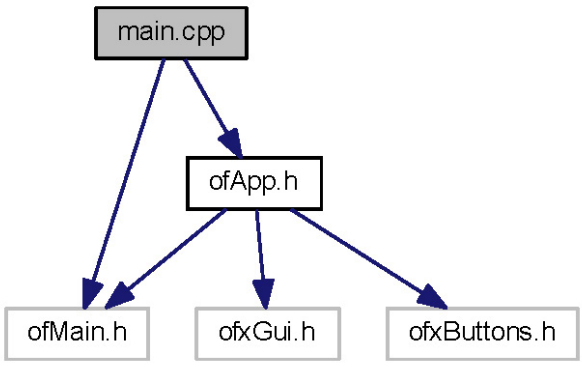


Figura 34.2 Dependencia gráfica adjunta para main.cpp [17]

Funciones a utilizar:

ofSetupOpenGL	Inicializa la ventana del programa con la resolución deseada.

ofRunApp	Inicia la aplicación y todas las funciones bajo el funcionamiento de ofApp()

ofApp.cpp

Librerías a utilizar:

#include “ofApp.h”

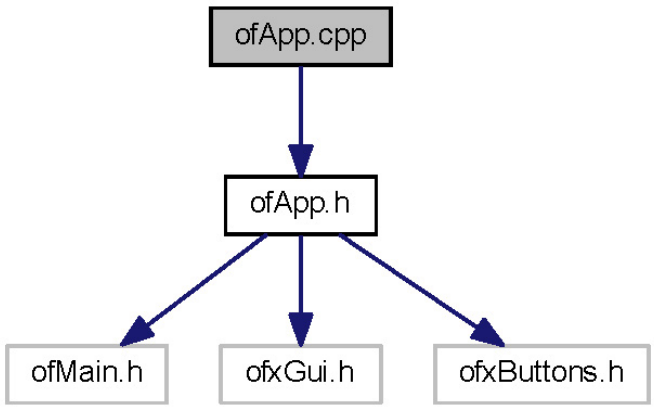


Figura 34.3 Dependencia gráfica adjunta para ofApp.cpp [17]

Funciones implementadas:

void ofApp::setup()	Inicializamos nuestro programa.
void ofApp::update()	Actualizamos variables y procedimientos.
void ofApp::draw()	Representamos los dos canales de nuestra señal.
void ofApp::auidoIn()	Obtenemos la señal introducida por el canal externo de nuestro Mac (entrada de Mic)
void ofApp::auidoOut()	Obtenemos la salida de la señal introducida para escucharla en nuestros auriculares.
void ofApp::keyPressed()	Pulsar el teclado.
void ofApp::keyReleased()	Soltar una tecla del teclado.
void ofApp::mouseMoved()	Mover el ratón.
void ofApp::mouseDragged()	Mover y pulsar el ratón.

void ofApp::mousePressed()	Pulsar el ratón.
void ofApp::mouseReleased()	Soltar el ratón.
void ofApp::windowResized()	Redimensiona la ventana.
void ofApp::gotMessage()	Mensaje de alerta.
void ofApp::dragEvent()	Eventos del programa.

Funciones a utilizar en SETUP:

ofSetVerticalSync	Se sincroniza la frecuencia de actualización.
ofSetCircleResolution	Se ajusta la resolución.
ofBackGround	Se establece el color de fondo.
soundStream.listDevices	Identifica el dispositivo de entrada de audio.
gui.setup	Inicializa la interfaz para agregar sliders.
globalGroup.setup	Hace grupos de sliders.
gui.add	Añade un slider.
buttons.setup	Inicializa el dropdown menú
vector.assign	Donde “vector” puede ser el nombre de un vector inicializado a un tamaño especifico.
soundStream.Setup	Configura el número de canales de salida y entrada.

Funciones a utilizar en UPDATE:

ofMap	Permite escalar el volumen.
volHistory.push_back	Graba el volumen en una matriz.
volHistory.size	Tamaño de la variable volumen.
volHistory.erase	Elimina el contenido de la matriz.

Funciones a utilizar en DRAW:

ofSetColor	Se utiliza para representar texto con un color.
ofDrawBitmapString	Se utiliza para escribir en la interfaz.
ofNoFill	Permite dibujar formas y contornos.

ofPushStyle	Guarda el estilo actual.
ofPushMatrix	Guarda el sistema de coordenadas actuales de la matriz.
ofTranslate	Modificación de los gráficos.
ofSetLineWidth	Establece el ancho de las líneas dibujadas.
ofRect	Dibujamos un rectángulo.
ofBeginShape	Representación de una nueva línea.
ofEndShape	Finaliza la representación.
ofPopMatrix	Restaura el sistema de coordenadas.
ofPopStyle	Restaura el estilo.

Funciones a utilizar en AUDIOIN:

En esta función no utilizamos ninguna otra llamada a funciones externas, puesto que en esta función simplemente captaremos el audio externo y aplicaremos el efecto dependiendo de la opción elegida por el usuario en el menú.

Los parámetros de entrada serán: la señal de entrada (audioin), el tamaño de buffer (256 para OSX y 512 para iOS) y el número de canales (en el caso de iOS es 2, ya que todo el tiempo es MONO (1 de entrada y 1 de salida), en el caso de OSX es estéreo por lo que serán 4 (2 de entrada y dos de salida)).

Esta función es propia de la librería de Streaming de audio de openFrameworks, y es aquí donde hemos realizado mas modificaciones a la hora de realizar el procesado de señal (los 11 efectos).

Funciones a utilizar en AUDIOOUT:

En esta función no utilizamos ninguna otra llamada a funciones externas, puesto que en esta función simplemente sacaremos el audio que hemos recibido y se ha procesado por los altavoces de nuestro ordenador Mac.

Los parámetros de entrada serán: la señal de salida (audioout), el tamaño de buffer (256 para OSX y 512 para iOS) y el número de canales (en el caso de iOS es 2, ya que todo el tiempo es MONO (1 de entrada y 1 de salida), en el caso de OSX es estéreo por lo que serán 4 (2 de entrada y dos de salida)).

Esta función es propia de la librería de Streaming de audio de openFrameworks, en esta función se han realizado muchas menos modificaciones que en AUDIOIN, aunque se ha modificado parte de su código para su adaptación a tiempo real.

Funciones a utilizar en KEYPRESSED:

soundStream.start	Inicializa el sonido.
soundStream.stop	Para el sonido.

En las funciones restantes no se llamarán a ninguna función externa.

ofApp.h

Librerías a utilizar:

```
#include “ofMain.h”
#include “ofxGui.h”
#include “ofxButtons.h”
```

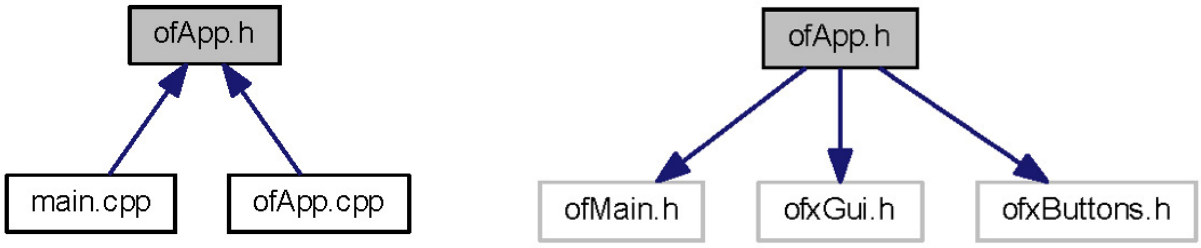


Figura 34.4 Dependencia gráfica adjunta para ofApp.h [17]

Funciones definidas que hemos utilizado:

void setup()	Inicializamos nuestro programa.
void update()	Actualizamos variables y procedimientos.
void draw()	Representamos los dos canales de nuestra señal.
void auidoIn()	Obtenemos la señal introducida por el canal externo de nuestro Mac (entrada de Mic)
void auidoOut()	Obtenemos la salida de la señal introducida para escucharla en nuestros auriculares.
void keyPressed()	Pulsar el teclado.
void keyReleased()	Soltar una tecla del teclado.
void mouseMoved()	Mover el ratón.

void mouseDragged()	Mover y pulsar el ratón.
void mousePressed()	Pulsar el ratón.
void mouseReleased()	Soltar el ratón.
void windowResized()	Redimensiona la ventana.
void gotMessage()	Mensaje de alerta.
void dragEvent()	Eventos del programa.
ofxGuiGroup	Realiza grupos de sliders.
ofxFloatSlider	Define un slider tipo float.
ofxPanel	Inicializa el panel de sliders.
Buttonmanager	Inicializa el menú.
ofSoundStream	Inicializa el sonido.

11. BIBLIOGRAFÍA

[1]. B. Goldstein, *Blackweel handbook of sensation and perception*. Oxford:
Blackweel Publishing, 2005.

[2]. ESPASA, *Diccionario de la lengua española*. Pozuelo de Alarcón: Espasa-Calpe, 2005.

[3]. W.A. Sethares, *Tuning, Timbre, Spectrum, Scale*, Segunda ed. Berlin: Springer–Verlag, 2004.

[4]. S. S Soliman, *Continuous and discrete signals and systems*, Segunda Ed.
Lebanon, Indiana: Prentice Hall, 1998.

[5]. ANSI, "American National Standard Acoustic Terminology," 1994.

[6]. PFC, ESTUDIO DE EFECTOS DE AUDIO PARA GUITARRA, E IMPLANTACIÓN MEDIANTE DSP. Ángel Pérez Rodríguez. Feb 2006.

[7]. PFC, Multi-Effect Processor for acoustic guitar. Eduardo Fonseca Montero. Dic 2007.

[8]. PFC, Development of Guitar Music FX to Aid DSP Teaching. Jody Monahan. Marzo 2010.

[9]. Smith, Julius O. "Chorus Effect." *Center for Computer Research in Music and Acoustics,* CCRMA. 2009. Web. 18 Apr. 2010.
https://ccrma-ww.stanford.edu/~jos/waveguide/Chorus_Effect.html

[10]. Bode, H. "History of Electronic Sound Modification." *Journal of the Audio Engineering Society*32 (1984): 730-39. Web. 2.0. Feb. 2010.

[11]. Wikipedia, the Free Encyclopedia 12 June 2008. Web. 18 Apr. 2010.
http://en.wikipedia.org/wiki/

[12]. The Scientist and engineer´s Guide to Digital Signal Processing. Ph.D. California Technical Publishing. ISBN 0-9660176-3-3, 1997.

[13]. Harmony- Central Effects Explained. Información sobre efectos digitales.

[14]. Página de IK Multimedia. Información sobre iRig.
http://www.ikmultimedia.com/products/irig1/

[15]. Hangar Interaction Lab. Alex Posada (electronics).

[16]. PFC de Iván Palomares Alguacil el cual nos ha servido de punto de inicio de este TFG. IMPLEMENTACION SOFTWARE DE UNA HERRAMIENTA PARA LA GRABACIÓN/EDICIÓN DE MAQUETAS MUSICALES AMATEUR:
http://avalos.ujaen.es/search~S2*spi/?searchtype=X&searcharg=Iván+Palomares+Alguacil&searchscope=2&sortdropdown=-&SORT=DZ&extended=0&SUBMIT=Buscar&searchlimits=&searchorigarg=X*

[17]. Doxygen - Generate documentation from source code:
http://www.stack.nl/~dimitri/doxygen/

[18]. TFG de El-Alga, Kaoutar - Programación de herramientas de afinación de instrumentos para dispositivos móviles:
http://tauja.ujaen.es/browse?type=author&value=El-Alga,+Kaoutar

[19]. Acústica Musical: Conceptos básicos sobre el Sonido:
https://www.google.es/ -q=https%3A%2F%2Fwww.lpi.tel.uva.es%2F~nacho%2Fdocencia&rct=j

[20]. Introducción a la Teoría del Procesamiento
Digital de Señales (DSP) de Audio - Luis Jure, Ernesto López, Martín Rocamora
http://www.eumus.edu.uy/eme/ensenanza/electivas/dsp.html