Agradecimientos

Primeramente agradecer este trabajo a mi familia, especialmente a mi madre, ya que siempre ha luchado por mi carrera académica, tanto técnica como musical. Sin ella no habría llegado hasta donde lo he hecho.

En segundo lugar, es evidente agradecer este trabajo a todos mis profesores de la Universidad de Alicante implicados en mayor o menor grado en el estudio de señales de audio. En especial a mi tutor Antonio Pertusa, ya que él me ha ido marcando el camino para poder llevar a cabo el presente trabajo. Sin él este proyecto no habría visto la luz.

Finalmente agradezco a todas aquellas personas que, sin estar involucradas para nada en el proyecto, han aportado su granito de arena ya sea revisando código fuente o animándome a seguir, y han hecho posible que escriba estas líneas. Sin ellos el camino habría sido más costoso.

Gracias.

Prólogo

Este trabajo surge como la concurrencia entre la búsqueda por mi parte de un proyecto relacionado con la creación de música digitalmente y la propuesta de mi tutor de hacer un efecto de audio, modificable en reproducción, íntimamente relacionado con su tesis doctoral.

El cometido de este ensayo es extraer y reproducir la melodía principal de un fichero de audio que contenga varios instrumentos musicales.

En la extensión del estudio se procurará introducir al lector de una forma más precisa en la problemática, mostrando y justificando cada una de las decisiones tomadas además de definir y aclarar todos los conocimientos previos requeridos para la plena comprensión del documento.

La memoria, la aplicación y todo el material relacionado con el proyecto se pueden encontrar en:

http://grfia.dlsi.ua.es/cm/worklines/anoguera-pfc

ÍNDICE GENERAL

ÍNDICE DE FÓRMULAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

Introducción

1

1.1. - MOTIVACIÓN

Dadas las herramientas informáticas de las que disponemos hoy en día, el procesado de audio digital está en pleno auge.

Queda mucho trabajo que hacer e investigación pendiente en este campo, ya que en él están envueltos diversas disciplinas como son el procesado de señal, la informática, la psicoacústica, la percepción musical o la teoría musical.

La detección de las diferentes voces en una composición musical se puede abordar de diferentes maneras, debido a la implicación de las disciplinas mencionadas. Este proceso se realiza detectando las frecuencias fundamentales que suenan simultáneamente en un determinado instante de tiempo, ya que estas nos dan la altura de cada una de las notas musicales presentes.

Una vez familiarizados con la temática pertinente y acercándonos más al presente fin, diremos que el poder variar los parámetros de un efecto que estemos aplicando a una composición musical mientras la escuchamos ya no es algo que quede fuera de nuestro alcance.

La meta es diseñar un programa informático que sea capaz de extraer la melodía principal de un fichero de audio musical y se pueda escuchar el resultado en tiempo real, esto es, mientras se reproduce. Entendemos por melodía principal como el instrumento armónico que tenga un volumen mayor sobre el resto de los presentes en la composición musical. Aunque esta definición no sea la más precisa en todos los casos, en términos de percepción es probablemente la más práctica.

Debido a que el propósito del presente trabajo es hacer un efecto en tiempo real, el objeto que se persigue no es obtener soluciones altamente precisas, sino resultados modificables mientras son escuchados.

Figura 1: Plug-in en programa de edición profesional de audio

Para el control del mismo se usará la plataforma Audio Unit desarrollada por Apple Inc. y que funciona como aplicación huésped en programas de edición profesional de audio.

El software implementado será de libre distribución.

1.2.- SOFTWARE RELACIONADO

1.2.1.- SMS TOOLS

Aplicación gratuita desarrollada en la universidad Pompeu Fabra de Barcelona. Analiza , transforma y sintetiza el sonido que le proporcionemos. Para hacerlo se sirve del modelo Spectral Modeling Synthesis llamado SMS o HILN en el contexto MPEG4.

Se puede encontrar información detallada de la aplicación en http://clam-project.org/wiki/SMSToolsDetails

1.2.2.- MELODYNE

Software de corrección de tempo y altura musical desarrollado por Celemony que nos permite extraer las diferentes voces de una melodía polifónica. Existen diferentes versiones que nos permiten trabajar con una sola voz o varias y que funcionan por sí solos o como huésped en programas de producción musical.

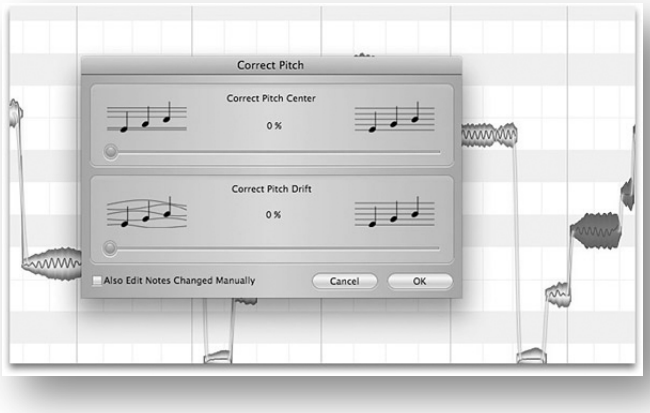

Figura 2: Interfaz Melodyne

Introducción

1

2 Conocimientos previos

2 Conocimientos previos

2.1. – ANÁLISIS DE SEÑALES DE AUDIO

El sonido, con el que tratamos en el presente proyecto, es una señal en función de la presión en el aire. Esto es, cuando un cuerpo vibra, transmite esa vibración o movimiento a través del aire y llega a nuestro oído que se encarga mediante una serie de órganos receptores de hacerlo llegar a nuestro cerebro para interpretarlo. Esta representación será posible siempre y cuando esa vibración ocurra entre 20 y 20000 veces por segundo, o lo que es lo mismo, el oído humano tiene una respuesta en frecuencia de 20 Hz a 20 kHz aproximadamente. Cabe decir que no sólo interpretamos vibraciones sencillas de una sola frecuencia, si no que somos capaces de percibir sonidos que son combinación de múltiples frecuencias.

Si representamos estas variaciones de la presión en el aire en un eje de tiempos, obtenemos lo que se llama forma de onda. Gracias a este hecho, desde la segunda mitad del siglo IXX ha sido posible registrar en diferentes medios, y cada vez más, material fonográfico.

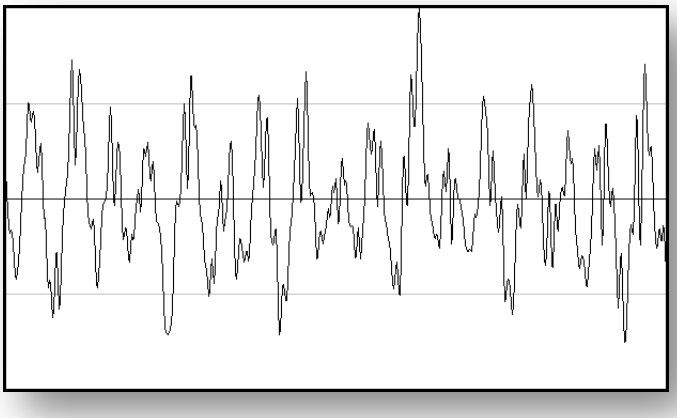

Figura 3: Forma de onda

Si dibujamos esta forma de onda de manera continua, ya sea sobre vinilo o sobre una cinta magnética decimos que estamos haciendo una grabación analógica.

Por otra parte, si almacenamos el valor de las muestras a determinados intervalos de tiempo y ajustándolas a determinados valores de amplitud ya sea en CD, DVD, memorias Flash o incluso cintas magnéticas, entonces estamos haciendo una grabación digital. La calidad de la grabación digital viene determinada por la cantidad de muestras por segundo que sean registradas (frecuencia de muestreo) y por la cantidad de valores a los que se pueda ajustar la amplitud de la señal a digitalizar (resolución).

Llegado a este punto es importante destacar que para poder registrar una señal de 100 ciclos por segundo o hercios es necesario tomar 200 muestras por segundo o lo que es lo mismo es necesario tener una frecuencia de muestreo de 200 hercios. Esto se conoce como teorema de Nyquist y nos dice que se necesita el doble de frecuencia de muestreo que la máxima frecuencia que queramos registrar. De manera intuitiva podemos pensar que esto es así ya que para detectar un ciclo debemos tener al menos dos valores de amplitud para detectar el cambio de presión.

2.1.1. – TRANSFORMADA DE FOURIER

2.1.1.1. – DESCRIPCIÓN BÁSICA

La forma de onda nos permite extraer entre otras cosas la envolvente temporal que nos ayuda a detectar por ejemplo el tempo musical.

Pero para el pertinente trabajo es necesario realizar una transformación matemática de la señal que nos proporcione información más relevante, esto es, las frecuencias presentes en un intervalo de tiempo determinado con sus magnitudes correspondientes.

Para ello se viene usando la conocida transformada de Fourier, que lo que consigue es pasar del dominio del tiempo al dominio de la frecuencia la señal que estamos tratando. Por tanto, en lugar de tener los diferentes niveles de presión a lo largo del tiempo tenemos las diferentes amplitudes a lo largo de la frecuencia. Esto es, en lugar de tener 20 ciclos en un segundo, tendremos un pico en la frecuencia de 20 Hz.

La definición matemática de la transformada de Fourier es:

$$\mathrm{FT}_x(f) = X(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi ft}\, dt \qquad [1]$$

Y pasada al terreno digital tenemos:

$$\mathrm{DFT}_x(k) = X(k) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j2\pi kn} \qquad [2]$$

Que para señales reales de duración finita se queda definitivamente en:

$$\mathrm{DFT}_x[k] = X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \quad , \qquad k = 0, \ldots, N-1 \qquad [3]$$

El teorema de Shannon nos dice que la máxima frecuencia que podremos volver a reproducir será la de Nyquist, esto es, la mitad de la frecuencia de muestreo. Se conoce por resolución frecuencial como el cociente entre la frecuencia de muestreo entre el tamaño de la ventana. Si multiplicamos cualquier posición de la ventana por esta resolución, obtendremos pues su frecuencia correspondiente.

Para poder hacer una transformada de Fourier computacionalmente eficiente, el tamaño de la ventana a analizar debe ser potencia de 2. El algoritmo desarrollado en base a esto se conoce como FFT (*Fast Fourier Transform*). En él, para calcular el valor de la posición radial y la posición angular necesarios para su representación, se usa la fórmula z = a + jb. En ocasiones esta representación se hace en escala logarítmica, ya que de

este modo podemos visualizar de manera más óptima el resultado. En el caso de la frecuencia, nos facilita la visualización de las diferentes octavas.

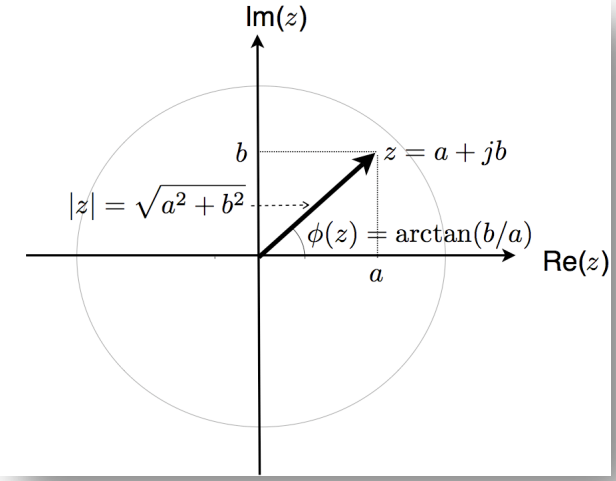

Figura 4: Diagrama de plano complejo. Magnitud y fase del numero complejo z. (Pertusa, 2010)

2.1.1.2. – LIMITACIONES

Trabajar con señales discretas y finitas presenta ciertas restricciones. En la conversión analógico-digital se puede producir aliasing o dicho de otra manera, solapamiento espectral. Para evitar este efecto se realiza un filtro paso-bajo con una frecuencia de corte igual a la máxima en la señal de entrada. Este filtrado recibe el nombre de filtro anti-aliasing y cabe decir que puede ser contraproducente en caso de que en el proceso de digitalización se usen varios filtros de este tipo en cascada, ya que volver a filtrar algo que ya está filtrado provoca que la onda resultante presente una marcada pendiente. Para evitar este posible efecto se recurre al aumento de la frecuencia de muestreo, con el consiguiente aumento de consumo de recursos.

Además, debido a la naturaleza discreta de las señales con que trabajamos, podemos observar derramamiento espectral en la

representación de la transformada de Fourier. Esto consiste en una serie de lóbulos que se presentan en lugares en los que no existían en la señal original. Para reducir esto se hace uso de ventanas de coeficientes de amplitud no rectangulares como Hanning, Hamming, Blackman o Blackman-Harris. Estas ventanas se caracterizan por tener formas simétricas, positivas y acampanadas.

Adicionalmente también se dan valores en el espectro generado que no se encontraban en el original debido a que trabajamos con una señal formada por diferentes partes separadas, esto es, nos topamos con picos con un valor inferior y valles con un valor superior a los correspondientes. Este inconveniente no tiene solución, aunque sí se puede mitigar mediante la interpolación de los huecos. La técnica zero-padding es la más usada, ya que es eficaz y no supone un gran coste computacional; consiste en añadir ceros al final de la señal de entrada antes de hacer la transformada de Fourier.

2.2. – ANÁLISIS DE SEÑALES MUSICALES

2.2.1.- DINÁMICA

Hace referencia a las variaciones de amplitud de la señal en el tiempo. Aquí usamos términos como envolvente, onset, offset o RMS.

La envolvente acústica se divide en ataque, decaimiento, sostenimiento y relajación; debido a su acrónimo en inglés, esto se conoce como ADSR.

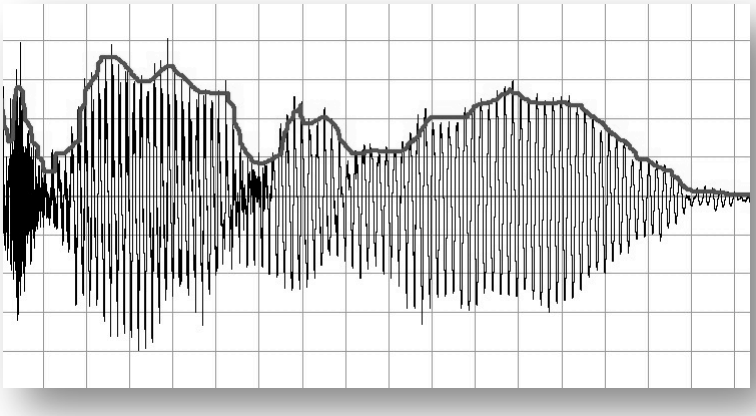

Figura 5: Envolvente de una señal (http://en.wikipedia.org/wiki/File:Envelope_follower.jpg)

El "Root Mean Square" o valor cuadrático medio nos dice el nivel de continua equivalente para proporcionar la misma energía que la señal alterna en el tiempo que estemos analizando. Sigue la siguiente fórmula:

$$E[n] = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x^2[n+i]} \quad [4]$$

2.2.2.- TIMBRE

Mediante él se distinguen los diferentes instrumentos musicales. Es la cualidad que nos permite discernir dos sonidos del mismo altura y sonoridad. Desde el punto de vista analítico es interesante entender las características que nos permitan diferenciar unos instrumentos de otros. Esto se usa como base en la clasificación automática de instrumentos y es algo que se queda fuera de los propósitos de este trabajo.

2.2.3.- SONIDOS MUSICALES AFINADOS

Los sonidos con frecuencia fundamental, que es la que determina su altura, tienen una forma de onda casi periódica en el dominio del tiempo. El periodo fundamental T_0 de una señal es la duración de un ciclo en un evento repetitivo por lo que es recíproco de la frecuencia fundamental f_0 ($T_0=1/f_0$).

La altura es el atributo perceptual referido a la frecuencia fundamental de una señal que permite ordenar los sonidos en una escala en función de la frecuencia de abajo a arriba o, musicalmente hablando, de grave a agudo.

Por tanto los sonidos musicales afinados son en los que se aprecia claramente una sensación de altura, como pueden ser una flauta o un piano.

Dentro de los sonidos afinados encontramos dos tipos: armónicos e inarmónicos.

2.2.3.1- SONIDOS ARMÓNICOS

La mayoría de los instrumentos clásicos (más concretamente los de cuerda y los de aire) producen sonidos armónicos, ya que están basados en un oscilador armónico como puede ser una cuerda o una columna de

aire. El espectro de estos sonidos muestra una serie de parciales equiespaciados llamados armónicos.

Idealmente los armónicos son múltiplos de la frecuencia fundamental aunque en la práctica esto no es así exactamente. Este efecto se conoce como inarmonicidad. Aunque esto resulta problemático a la hora de analizar una señal, para la escucha puede resultar incluso agradable ya que le da un carácter propio al sonido.

Hay que tener en cuenta que tal como Schouten (1940) dice el tono de una señal compleja se puede percibir aunque el componente frecuencial no esté presente (fundamental perdida). Para comprobar esto se puede usar como ejemplo la señal telefónica que esta filtrada por debajo de los 300 Hz, por lo que la voz humana pierde su primer parcial.

En la mayoría de instrumentos de cuerda las frecuencias agudas se desvían hacía arriba dependiendo de las características físicas de las cuerdas, como por ejemplo la tensión.

Por otra parte, también se da el caso de que en la parte de ataque de las diferentes notas se dé desafinación debido al ruido de el propio soplido y de las llaves en viento, el arco en cuerda o el sonido del martillo para el piano.

La síntesis aditiva descrita por Moorer (1977) fue la base del modelado espectral armónico original, que aproxima una señal armónica a una suma de sinusoides. Por tanto expresamos un sonido armónico como suma de sinusoides más un error:

$$x[n] = \sum_{h=1}^{H} A_h[n] \cos(2\pi f_h n + \phi_h(0)) + \epsilon[n] \quad \textbf{[5]}$$

2.2.3.2- SONIDOS INARMÓNICOS

Estos son los que tienen un tono pero sus armónicos no son aproximadamente múltiplos de la frecuencia fundamental. Usualmente suelen ser los basados en barras vibratorias como el xilófono o el vibráfono. Por tanto esta clase de instrumentos se escapan de los propósitos de este proyecto.

2.2.4.- SONIDOS NO AFINADOS

Son generalmente sonidos percusivos como los de un bombo, un tambor o un platillo. Se caracterizan por tener una ataque brusco con una ancha dispersión en frecuencia. Esta clase de instrumentos también queda fuera del alcance de este trabajo.

2.2.5.- SONIDOS CANTADOS

Tampoco se contemplan estos dentro del rango abarcado por el efecto ya que la voz humana produce una señal muy compleja muy por encima de las posibilidades del presente escrito.

2.3.- ANTECEDENTES MUSICALES

La sucesión de notas en una composición musical, así como su aparición múltiple simultánea no es algo aleatorio visto desde un punto de vista musical. Las notas siguen patrones melódicos, armónicos y rítmicos para que, en principio, resulte agradable al oyente.

En esta sección se hace una breve descripción de estas normas se hace a continuación.

2.3.1.- ESTRUCTURA TONAL

Se puede definir melodía como la sucesión de sonidos afinados y con una determinada estructura rítmica. La formación y relación de la combinación de las notas simultáneas en el tiempo se conoce como armonía.

Se define intervalo melódico como la proporción entre dos tonos sucesivos en el tiempo. Si estos tonos son simultáneos entonces hablamos de intervalos armónicos.

En música clásica lo común es que se den intervalos consonantes que en términos espectrales diremos que tienen en común componentes armónicos.

2.3.1.1.- TEMPERAMENTO MUSICAL

En términos frecuenciales describimos un intervalo musical como la proporción entre las respectivas frecuencias de dos notas. Después del unísono, esto es, dos notas del mismo tono, el intervalo más simple es la octava que cumple una proporción de frecuencia 2:1.

El oído humano tiende a interpretar de la misma manera dos notas iguales pero de octavas diferentes. Esta es la razón de porque en distintas

culturas musicales de todo el mundo las notas se dividen en octavas en una escala frecuencial logarítmica.

Aún así hay diferentes maneras de dividir las octavas dependiendo de la cultura que estemos tratando. De todos modos la más común es la usada en la música clásica, que divide la octava en doce partes logarítmicamente iguales que reciben el nombre de semitonos. Por tanto cada par de notas adyacentes en la escala de semitonos tiene una proporción en frecuencia igual a $1:2^{1/12}$ o 100 centésimas. Dos semitonos forman un tono.

Además, desde la primera mitad del siglo XX se usa la afinación de USA estándar que parte de la nota La_4 a 440 Hz. Hasta ese momento se usaban diferentes afinaciones.

2.3.1.2.- NOTAS DE LA MÚSICA CLÁSICA

A la hora de dar nombre a las diferentes notas que componen la escala encontramos principalmente dos maneras: la usada en el sur y el este de Europa, y la usada en América y el norte de Europa.

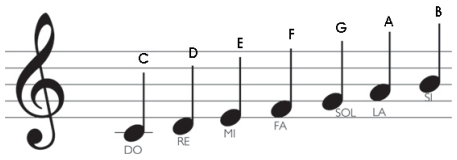

Figura 6: Correspondencia de notas musicales

En la primera, los doce semitonos se nombran mediante el uso de siete notas sujetas a alteraciones: Do, Re, Mi, Fa, Sol, La, Si. En la segunda manera se sigue el mismo patrón, pero con nombres diferentes: A, B, C, D, E, F, G. Hay que tener en cuenta la equivalencia entre una y otra escala es: partiendo de La = A se sigue el mismo orden para el resto de las notas de la octava, esto es, Do = C. La escala esta formada por

tonos y semitonos y en ella se hace uso de alteraciones ascendentes o descendentes, esto es, sostenidos (#) o bemoles (b).

La tonalidad de una composición musical es su centro gravitatorio armónico. Los intervalos dentro de la composición son consonantes a la tónica que es la que le da nombre a la tonalidad.

2.3.2.- NOTACIÓN MUSICAL MODERNA

Las composiciones musicales se interpretan mediante la lectura de partituras. A lo largo de la historia ha habido diferentes maneras de representar la música gráficamente, aunque hoy en día la más usada es la creada en Europa para la interpretación de música clásica.

Esta representación se hace en dos dimensiones y se crea a partir de pentagramas que son agrupaciones de cinco líneas paralelas en las que el tono y el instante de las notas viene determinado por la posición con respecto a estas agrupaciones.

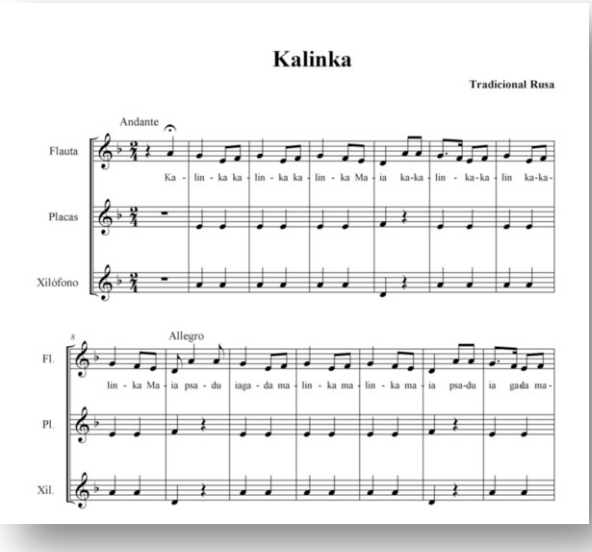

Figura 7: Partitura (http://mariajesuscamino.com/partituras/actividades/kalinka/6b616c696e6b61315f50c3a167696e615f31.jpg)

Las notas más graves se posicionan en la parte baja del pentagrama y las agudas en la parte alta. Se pueden usar sostenidos y bemoles para variar el altura de una nota en una misma línea o hueco.

El instante de lectura de cada nota esta relacionado con su posición horizontal, por tanto se lee antes una nota que esté situada a la izquierda que una que esté más a la derecha. Cuando tenemos varios pentagramas se leen de arriba hacia abajo.

La duración musical de cada nota la determina su dibujo, pudiendo encontrar redondas, blancas, negras, corcheas... Esta duración puede ser extendida mediante el uso de puntillos.

Lo primero que encontramos en el pentagrama es la clave, que nos indica la tonalidad de las notas escritas. Después vemos la armadura que determina las alteraciones a las que están sujetas las notas. Por último y antes de que aparezcan las notas tenemos la métrica que nos indicará el tipo y la cantidad que habrán en cada compás. Por ejemplo, podemos tener clave de sol, con tres sostenidos y un compás de tres por cuatro. Esto indica que la segunda línea del pentagrama corresponde con sol, estamos en La mayor o su relativo menor y podemos poner tres negras en cada compás. Los parámetros que proporcionan la velocidad y la dinámica (partes más suaves o más fuertes) se escriben encima del pentagrama.

Además también existen notaciones para percusión y notaciones en tablatura que muestran las progresiones de acordes.

2.3.3.- NOTACIÓN MUSICAL POR COMPUTADORA

En un ordenador, la música se puede almacenar y representar de diferentes maneras. Sin duda la más conocida es el estándar MIDI que almacena una serie de ordenes con la información de la composición musical para diferentes instrumentos electrónicos digitales puedan

reproducirla así como grabarla. Cabe decir que el fichero generado sólo contiene las instrucciones, esto es, siempre necesitaremos un instrumento virtual para poder reproducir la composición.

Los ficheros MIDI pueden ser representados de diferentes maneras. Una de ella es la ya descrita notación musical moderna. Otra muy extendida en software musical es el diagrama de pianola, que consiste en una serie de barras horizontales colocadas en un plano con ejes tiempo-frecuencia cuyo tono se determina por la posición vertical, la duración viene determinada por la longitud de la barra y la intensidad por el color.

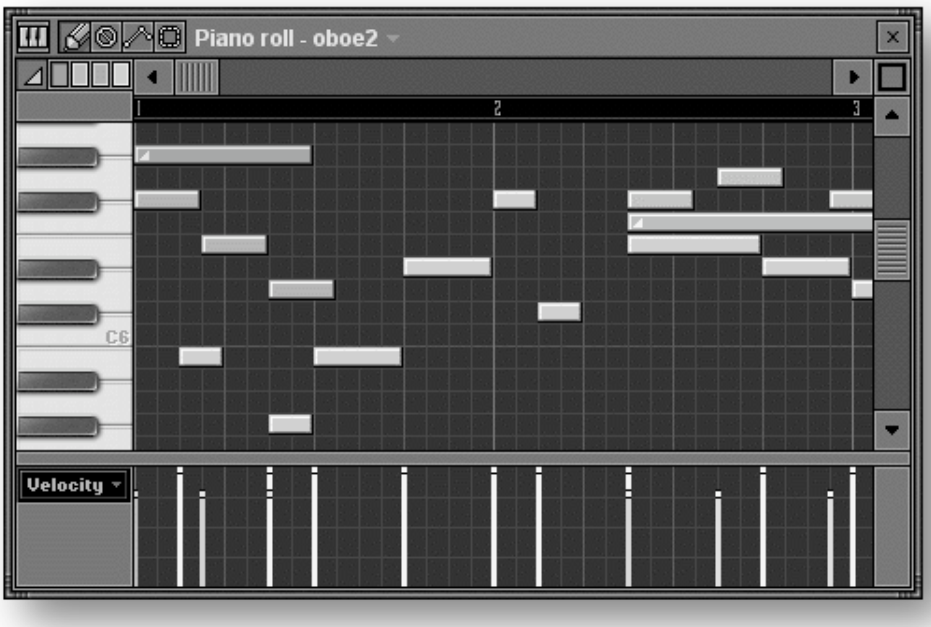

Figura 8: Diagrama de pianola
(http://lib.store.yahoo.net/lib/worshipmusic/FL-PianoRoll-big.gif)

2.4.-SOFTWARE MUSICAL

2.4.1.- SECUENCIADORES

Entendemos por "Digital Audio Workstation" como la combinación de hardware/software para manipular el proceso de crear música digitalmente. Aunque en sus inicios el hardware adicional a parte de la computadora era muy necesario, cada vez se tiende a virtualizar la mayor cantidad de aparatos posible, dejando todas las tareas de procesado a la CPU. De este modo solo son necesarias opcionalmente únicamente interfaces de control y por tanto se abaratan costes.

Figura 9: Ejemplo de una estación de trabajo de audio digital (http://2.bp.blogspot.com/_glMviwVtgzE/SctGinli_nI/AAAAAAAAAPY/mpWv5bnYlKg/S1600-R/MBD-DAW-1-web.jpg)

2.4.1.1.- CUBASE

Es un paquete de software de edición de audio digital, MIDI y secuenciador de música. Este secuenciador se creó originalmente en

Alemania por la firma Steinberg.

Nació como editor MIDI para Atari ST. Después se hizo la versión para Apple Macintosh y finalmente a mediados de los 90 para Microsoft Windows.

En 1991 el lanzamiento de Cubase Audio para Atari Falcon supuso un hito en la historia del audio digital ya que fue pionero en la edición del audio en tiempo real sin necesidad de costosas tarjetas adicionales como ocurría con otros sistemas como Pro Tools.

Actualmente se encuentra en la versión 5 y nos permite hacer mezclas y ediciones de pistas MIDI y audio pudiendo exportarlas en diferentes formatos.

2.4.1.2.- LOGIC PRO

Es un secuenciador con bastante historia. Todo empezó cuando en 1993 C-Lab pasó llamarse Emagic y sacó una nueva versión de Notator, su conocido secuenciador MIDI para Atari. Esta versión se llamo Notator Logic, que en posteriores lanzamientos adquirió el nombre de Logic, cuando se hizo compatible con Mac OS y Windows.

En 2002 se encontraban en la versión 6 del software y Apple Inc. compró Emagic. A partir de ahí, el programa sólo está disponible para Mac OS X.

Este programa se encuentra en su versión 9 y nos permite mezclar y editar pistas MIDI y audio pudiendo sacar la mezcla en diferentes formatos de audio digital. Además viene provisto de miles de archivos musicales clasificados y libres de derechos que por tanto se pueden usar en las creaciones siempre y cuando se tenga la licencia del software.

2.4.1.3.- AUDACITY

Es una aplicación multiplataforma libre que nos permite grabar y editar diferentes pistas de audio. Se distribuye bajo la licencia GPL.

Fue creada por un universitario estadounidense, que actualmente es empleado de Google aunque sigue siendo el principal desarrollador de la aplicación.

La versión de Mac soporta Audio Units, pero no VST (ver. 2.4.2). Para ello se debe de descargar un plug-in adicional. Los efectos no trabajan en tiempo real. Los ficheros MIDI sólo pueden ser visualizados.

2.4.2.- PLUG-INS

En esencia, un Plug-in es un componente de procesamiento de audio, y no una aplicación de audio: se utiliza dentro de una aplicación anfitriona. Esta aplicación "host" proporciona las secuencias de audio que son procesadas por el plug-in.

2.4.2.1.- VST

Son de Steinberg y fueron pioneros en este tipo de software. En términos generales, un plug-in VST puede tomar una secuencia de datos de audio, aplicar un proceso para el audio, y devolver el resultado a la aplicación host. Un plug-in VST realiza su proceso normal utilizando el procesador del equipo, no tiene por qué usar procesadores de señal digital dedicados. El flujo de audio se divide en una serie de bloques. El anfitrión suministra los bloques de forma secuencial. El anfitrión y su control el entorno actual del tamaño de bloque. El plug-in VST mantiene el estado de todos sus propios parámetros relacionados con el proceso en ejecución: El anfitrión no mantiene ninguna información sobre lo que el plug-in hizo con el último bloque de datos que procesa.

Desde el punto de vista de la aplicación host, un plug-in VST es una caja negra con un número arbitrario de entradas, parámetros de salida (MIDI o de audio) y parámetros asociados. El anfitrión no necesita el conocimiento implícito del plug-in de proceso para poder usarlo. El proceso del plug-in puede usar los parámetros que desee, internamente en el proceso, pero dependiendo de las capacidades de la máquina, se puede permitir que los cambios de los parámetros de usuario puedan ser automatizados por el programa anfitrión.

Estos pueden ser usados en programas de Audio como Cubase, Nuendo, Virtual Dj, Audacity (usando un componente opcional), Ableton Live o Cakewalk Sonar.

2.4.2.2.- AUDIO UNIT

Arquitectura plug-in provista por Core Audio en Mac OS X desarrollado por Apple Computer. Son un conjunto de interfaces que generan, procesan, reciben o manipulan tramas de audio en casi tiempo real con latencia mínima. Se podría decir que es la arquitectura de Apple equivalente al otro popular formato de plug-in, VST de Steinberg.

Por tanto encontramos Audio Units ecualizadores, procesadores de dinámica, retardos, reverberaciones o instrumentos de sintetizadores de bancos de sonido.

Los AU son soportados en programas de Mac OS X tales como Garage Band, Soundtrack Pro, Logic Express, Logic Pro, Final Cut Pro, Ardour, Ableton Live, Digital Performer o Audacity.

2.5.-PROGRAMACIÓN

2.5.1.- LENGUAJE C++

Diseñado a mediados de los 80 por Bjarne Stroustrup. Su propósito fue extender el leguaje de programación C con mecanismos que permitan manejar objetos, por tanto es un leguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y programación orientada a objetos). Por ello también se dice que es un lenguaje de programación multiparadigma.

Una particularidad del C++ es la posibilidad de redefinir los operadores(sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.

Su nombre inicial era "C con clases" pero se cambió al actual debido a que la expresión "C++" significa incremento de C.

2.5.2.- XCODE

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con Mac OS X. Xcode. Trabaja conjuntamente con Interface Builder, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo, pero no limitado a Cocoa, Carbón y Java. Otras compañías han añadido soporte para GNU Pascal, Free Pascal, Ada y Perl.

Figura 10: Xcode

(http://www.applenext.com/wp-content/uploads/2010/02/xcode.jpg)

Entre las características más apreciadas de Xcode está la tecnología para distribuir el proceso de construcción a partir de código fuente entre varios ordenadores, utilizando Bonjour.

2.5.3.- SDK

Los "*Software Development Kit*" (SDK) son un conjunto de herramientas de desarrollo que permite a un programador crear aplicaciones para un software concreto. Es algo tan sencillo como una interfaz de programación de aplicaciones (API) creada para permitir el uso de cierto lenguaje de programación. Las herramientas más comunes incluyen soporte para la detección de errores de programación como un entorno desarrollado integrado o IDE y otras utilidades. Suelen incluir también códigos de ejemplo y notas técnicas de soporte para ayudar a entender ciertos puntos del material de referencia primarios.

2.5.3.1.- AU SDK

Forma parte de la instalación de Xcode Tools en el DVD de Mac OS X. También se puede descargar de internet. Este paquete instala

superclases C++, proyectos Xcode de ejemplos y documentación, así como plantillas de proyecto para efectos e instrumentos Audio Unit.

2.5.3.2.- VST SDK

Se descarga gratuitamente (previo registro) del sitio web de Steinberg y contiene las clases C++ necesarias, así como ejemplos y documentación. Actualmente el SDK está en la versión 3.

2 Conocimientos previos

3 Metodología

Metodología 3

3.1. – DESCRIPCIÓN DEL ALGORITMO

Antes de la descripción explícita del código se hará una mención de los pasos del proceso a realizar a la señal de entrada.

En primer lugar realizaremos la transformada de Fourier de la entrada. Una vez obtenida, buscaremos picos espectrales para después encontrar un máximo determinado por un estimador de frecuencia fundamental. Entonces iremos buscando los armónicos correspondientes de dicha frecuencia fundamental.

Cuando lo hayamos conseguido, generaremos una señal en el tiempo resultado de sintetizar la frecuencia fundamental predominante con sus respectivos armónicos. Con el fin de obtener mas resolución temporal se hará un solapado de la señal de salida.

Como se puede observar tenemos claramente diferenciadas dos partes: por una parte análisis de la señal y síntesis de la misma. A continuación se describen estas partes con más detalle. Se ha de tener en cuenta que aquí sólo se describe la parte de procesado de la señal, dejando para puntos posteriores otros aspectos de la aplicación como la interfaz gráfica o la comunicación entre el programa anfitrión y el pertinente efecto.

3.1.1. ANÁLISIS

3.1.1.1.- ALGORITMO DE TRANSFORMADA DE FOURIER

El algoritmo FFTW3 Fráge & Johnson (2005) se decidió adaptar al proyecto, ya que tiene un funcionamiento eficaz. A este se le pasa un vector de entrada de la señal en el tiempo, los coeficientes de la ventana a

usar, los vectores en los que se almacenará la transformada y los tamaños de ventana usados.

3.1.1.2.- ESTIMADOR DE FRECUENCIA FUNDAMENTAL

En el presente proyecto se han puesto a prueba varios estimadores de frecuencia fundamental, para decidir cuál se adapta mejor a las condiciones de trabajo del efecto a desarrollar. A continuación se hace una descripción de cada uno.

Joint Estimation - Antonio Pertusa

Algoritmo creado en C++ por Pertusa (2010) basado en las propiedades de los sonidos armónicos. Se parte de que las frecuencias fundamentales son picos presentes en el espectro, por tanto los timbres con fundamental ausente no se consideran.

La estimación de frecuencia se fundamenta en la suma de armónicos y la suavidad en la envolvente espectral de la señal de entrada. El algoritmo fue evaluado en el MIREX (2008) obteniendo muy buenos resultados.

YIN - Alain de Cheveigné

Algoritmo creado en lenguaje M para la estimación de la frecuencia fundamental de voz o música. Hay una descripción del mismo en "YIN, a fundamental frequency estimator for speech and music", Alain de Cheveigné et al. (2002).

Se basa en el método de autocorrelación combinado con más métodos para prevenir errores. El algoritmo está preparado para trabajar con notas agudas de voz y música. Da la posibilidad de variar parámetros

para obtener mejores resultados. Esta basado en un modelo de señal periódica que podría ser extendido para manejar aperiodicidades particulares. Además, finalmente, debido a que está hecho en Matlab puede mostrar los resultados en gráficas.

Harmonic Product Spectrum - Gareth Middleton

Este sencillo algoritmo producido por The Connexion Project y con licencia de atribución Creative Commons está basado en el hecho de que los instrumentos musicales afinados armónicos generan notas formadas por una frecuencia fundamental y una serie de parciales que son múltiplos de esta frecuencia.

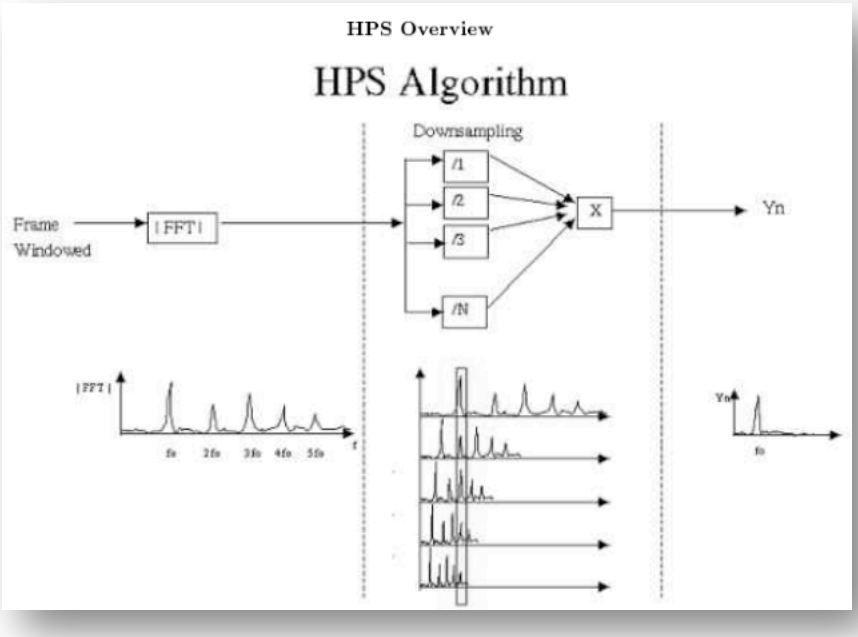

Figura 11: Algoritmo HPS

Gareth Middleton (2003)

Una vez dicho esto, el proceso es sencillo de explicar: partiendo del espectro de la señal, almacenamos en primer lugar una de cada dos muestras del mismo, luego una de cada tres, después una de cada cuatro y así hasta cuantos parciales queramos tener en cuenta. Una vez tenemos

los vectores anteriores hacemos la multiplicación componente a componente de todos los vectores. Esto es, si tenemos cinco vectores hacemos los productos múltiples del elemento i de los diferentes vectores, obteniendo como resultado un único vector con los valores de los productos de cada elemento i.

Hay que tener en cuenta que el número de elementos a medida que vamos tomando una de cada x muestras se ve reducido al factor x. Por tanto el resto de muestras hasta llegar al numero de muestras original se rellena con ceros. Esto provocará que sólo los armónicos realcen la frecuencia fundamental, por que los demás componentes se verán anulados por los ceros.

3.1.1.2.- ESTIMADOR DE PARCIALES

Para la estimación de parciales se ha escogido un algoritmo que se fundamenta en seleccionar el candidato que mejor cumpla la condición de máxima amplitud dentro de un rango de frecuencia determinado por la fundamental y su posición en el espectro. Los candidatos están sometidos a un coeficiente de ponderación triangular que disminuye a medida que se alejan de los múltiplos de la frecuencia fu8ndamental, Pertusa (2010).

3.1.1. SÍNTESIS

Debido a que sintetizamos una señal a partir de información extraída de la señal de entrada debemos aplicar una interpolación entre las distintas frecuencias de un frame a otro para obtener transiciones suavizadas y evitar sonidos indeseados. En su día se decidió usar las fórmulas descritas por Serra (1997), que están clara y concisamente explicadas en su artículo.

Por tanto, tal como se dice en el apartado 10 del citado artículo partimos de que la señal a sintetizar se forma a partir del sumatorio de tonos puros, esto es, la frecuencia fundamental y sus respectivos parciales. Matemáticamente

tenemos:

$$d(m) = \sum_{r=1}^{R} \hat{A}_r \cos\left[m\hat{\omega}_r + \hat{\varphi}_r\right], \qquad m = 0,1,2,\ldots,S-1 \quad [6]$$

donde R es el numero de parciales más uno (ya que se tiene en cuenta la fundamental) presentes en nuestra señal a sintetizar, S es el tamaño de la ventana, *m* la posición del vector, *A* la amplitud, ω la frecuencia en radianes y $\hat{\varphi}_r$ la fase.

Si definimos:

$$\hat{\theta}(m) = m\hat{\omega} + \hat{\varphi} \quad [7]$$

Finalmente tenemos:

$$d^l(m) = \sum_{r=1}^{R^l} \hat{A}_r^l(m)\cos\left[\hat{\theta}_r^l(m)\right] \quad [8]$$

donde l hace referencia al numero de frame o ventana en la que nos encontremos.

Para evitar transiciones bruscas de frame a frame estas son interpoladas calculando la fase instantánea según las fórmulas que a continuación se muestran. Estas se tienen que calcular para cada uno de los parciales que forman la señal resultado.

Para la amplitud se tiene en cuenta el módulo del parcial a sintetizar en el frame actual y el anterior, así como la posición de la ventana.

$$\hat{A}(m) = \hat{A}^{l-1} + \frac{(\hat{A}^l - \hat{A}^{l-1})}{S} m \quad [9]$$

Para la fase instantánea es necesario usar un polinomio cúbico, ya que en su cálculo se ven envueltas cuatro variables. Estas son los valores de frecuencia y fase del frame actual y anterior.

$$\hat{\theta}(m) = \varphi^{(l-1)} + \hat{\omega}^{(l-1)}m + \eta m^2 + \iota m^3 \quad [10]$$

$$\eta = \frac{3}{S^2}\left(\hat{\varphi}^{l} - \hat{\varphi}^{l-1} - \hat{\omega}^{l-1}S + 2\pi M\right) - \frac{1}{S}\left(\hat{\omega}^{l} - \hat{\omega}^{l-1}\right) \quad [11]$$

$$\iota = -\frac{2}{S^3}\left(\hat{\varphi}^{l} - \hat{\varphi}^{l-1} - \hat{\omega}^{l-1}S + 2\pi M\right) - \frac{1}{S^2}\left(\hat{\omega}^{l} - \hat{\omega}^{l-1}\right) \quad [12]$$

donde *M* es el entero más cercano a x, que se define como:

$$x = \frac{1}{2\pi}\left[\left(\hat{\varphi}^{l-1} + \hat{\omega}^{l-1}S - \hat{\varphi}^{l}\right) + \frac{S}{2}\left(\hat{\omega}^{l} - \hat{\omega}^{l-1}\right)\right] \quad [13]$$

3.2. – PASOS PREVIOS A LA IMPLEMENTACIÓN DEL ALGORITMO

Este proyecto se ha realizado en el lenguaje de programación C++, ya que es el de más extendido uso en aplicaciones de procesado de audio. El compilador utilizado ha sido Xcode, suministrado por Apple computer en su sistema operativo Mac OS X.

A la hora de llevar a cabo este trabajo se optó, en primer lugar, partir del SDK de la interfaz estándar VST (Virtual Studio Technology), desarrollada por Steinberg. En la fecha de inicio del presente proyecto (Marzo de 2008) los paquetes de software proporcionados por su empresa creadora para el desarrollo de la misma se encontraban en transición de la versión de aquel momento (2.4) a una más nueva (3.0). No había suficiente documentación clara para su realización, además de que en los foros de desarrolladores se presentaban muchos problemas para la elaboración de efectos mediante este estándar. Por tanto, tras dos meses de indagación e intentos de implementación sin éxito, se decidió crear el efecto bajo otra interfaz.

En la búsqueda de otra interfaz se adoptó la arquitectura plug-in Audio Unit proporcionada por Core Audio en Mac OS X. Es, a grosso modo, el equivalente de VST pero desarrollado por Apple Computer. La única limitación que presenta este estándar es que es sólo compatible con Mac OS X, mientras que VST es compatible tanto para Windows como para Mac OS X. Aún así, existe software de conversión de VST a AU y viceversa.

La elección final de esta interfaz fue sin duda por la clara documentación suministrada por su empresa creadora para su uso y desarrollo. De este modo, para poder implementar el algoritmo concerniente, es necesario revisar "Audio Unit Programming Guide" (rev. 31 de octubre de 2007).

A partir de ella se ha implementado el algoritmo de proceso siguiendo las pautas proporcionadas a partir de un ejemplo sencillo sugerido en la misma.

Este modelo también contiene cómo realizar una sencilla interfaz gráfica para el software, que nos permite cambiar el valor de diferentes parámetros en tiempo de reproducción mediante deslizadores. También podemos guardar configuraciones de parámetros para usos posteriores. Incluso nos permite usar ajustes "de fábrica" o preestablecidos por el desarrollador, dando así al usuario ciertas orientaciones sobre el uso de los parámetros para diferentes instrumentos.

3.3. – IMPLEMENTACIÓN DEL ALGORITMO

3.3.1.- PUESTA EN MARCHA

Siguiendo las pautas de la guía de programación de Audio Unit, creamos un proyecto de Audio Unit Effect en Xcode. Este nos proporciona los ficheros necesarios para un correcto proceso y comunicación entre el plug-in y el secuenciador. Nos da un fichero cpp, con su *header* asociado, que recibe el nombre asignado al proyecto. Estos son los únicos que debe modificar el desarrollador para crear el efecto propuesto. No obstante, se pueden añadir o enlazar al proyecto tantos ficheros como necesitemos.

3.3.2.- CÓDIGO DE PROCESAMIENTO

Antes de describir el procesamiento hay que dejar constancia de que al final de la presente memoria se encuentra un anexo con los códigos fuente y la explicación de las sub-funciones usadas por la función de proceso principal. Es por ello que en este apartado no se describen estos sub-procesos, para un mejor entendimiento del proceso general.

Además también hay que mencionar que dado que la plantilla por defecto traía la variables de tipo flotante como Float32 todos las variables añadidas de tipo flotante también son Float32. Para una mayor comodidad y legibilidad en el trabajo con el código se han creado registros para almacenar las transformadas de Fourier (compuestos por el tamaño de ventana y vectores de módulo, fase y coeficientes de ventana). También se han creado registros para almacenar las características de los parciales presentes en nuestra señal a sintetizar, esto es, posición, módulo, fase y omega correspondientes al vector de la transformada.

El tamaño de los vectores de la transformada se ha fijado en 8192, ya que aunque el valor máximo de ventana permitido por el programa de pruebas es 4096, el algoritmo desarrollado está preparado para aplicar zero-padding, que requiere un aumento de tamaño de ventana.

Una vez recibimos la señal de entrada comprobamos que no sea silencio (en cuyo caso no hay procesamiento) y que tenga un tamaño de ventana igual a 1024 o 2048 o 4096. En caso contrario haremos bypass de la señal, ya que para valores menores habría demasiada desafinación y para valores superiores, demasiada latencia. Si hubiera sido posible implementar la técnica zero-padding con éxito en este algoritmo, el tamaño de ventana usado quizá podría ser menor.

Si se cumplen las condiciones anteriores comprobamos que el rango de la señal es correcto, esto es, se encuentra entre [-1,1]. Después sacamos su valor cuadrático medio. Se hace baipás y no habrá procesamiento si este valor no supera el umbral de silencio. Este es introducido por el usuario en tanto por ciento y se pasa a tanto por uno para que se ajuste al rango de la señal.

Es entonces cuando realizamos la FFT de la entrada con ventana Hanning mediante fftw3, que se encuentra en el fichero añadido al proyecto myfft.cpp. De aquí obtenemos los vectores de módulo y fase. Entonces ejecutamos el algoritmo HPS para quince parciales descrito en el apartado 3.1.1.2. e incluido en el anexo final del presente documento. En el vector resultado buscamos picos espectrales y dentro del rango de frecuencias que el usuario puede variar en tiempo de reproducción. Una vez obtenidos los picos, entonces buscamos el de máximo valor mediante MaximumSearch y este nos dará la frecuencia fundamental que pretendemos obtener según nuestro algoritmo.

Una vez obtenidos los datos necesarios, pasamos a la parte de síntesis de nuestra nueva señal. Primero comprobamos el numero de parciales posibles de la señal a generar, limitándolo a 15 para reducir el coste computacional.

Lo primero que hacemos es sintetizar nuestra frecuencia fundamental usando las fórmulas de interpolación descritas por Serra (1997) y almacenándola en un vector que se usará como sumatorio. Acto seguido, buscamos las posiciones y características de los parciales de nuestra frecuencia fundamental y se van sumando al vector mencionado.

Cuando ya tenemos completo el vector de nuestra señal sintetizada

hacemos un solapamiento ajustable por el usuario a la señal de salida con los valores del frame anterior.

Finalmente, reproducimos nuestra señal de salida.

3.4. – INTERFAZ GRÁFICA

Para la realización de la interfaz gráfica estándar de Audio Unit se han seguido los pasos de la guía de programación. Los parámetros variables en este trabajo son:

-Frecuencias fundamentales máxima y mínima a buscar en nuestra señal de entrada. Se dan en hercios y son muy útiles ya que evita que cometamos errores de confundir frecuencias fundamentales con sus armónicos si sabemos en que octava se encuentra la melodía que pretendemos extraer.

-Umbral de silencio en la señal de entrada μ. Se introduce en porcentaje con respecto al valor RMS de la señal de entrada. Nos evita tener que reproducir instantes en los que no hay notas musicales sino ruido de fondo.

-Solapamiento. Se introduce en porcentaje. Nos permite obtener mayor resolución temporal y por tanto un posible mejor resultado.

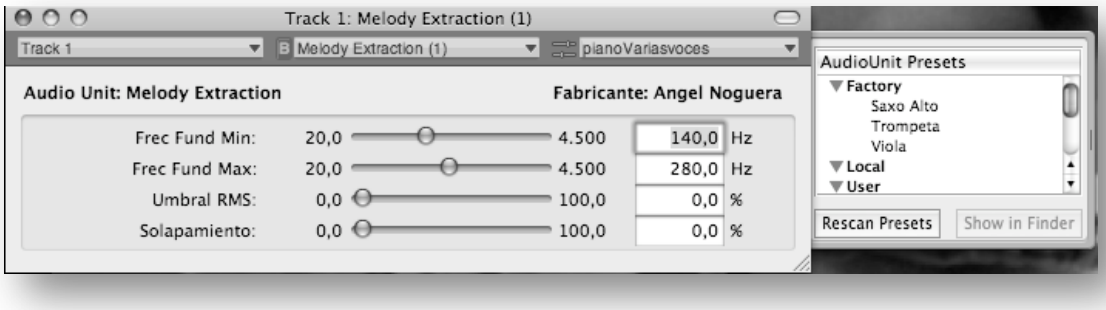

Figura 12: Interfaz gráfica de un Audio Unit

Para implementar estos parámetros en nuestro código se deben establecer valores mínimos y máximos, así como por defecto. En adición también creamos ajustes de fábrica para el posterior uso por parte de usuario.

4 Experimentación

Experimentación 4

4.1. - EXPERIMENTACIÓN DURANTE EL DESARROLLO

Una vez descrito por completo el presente proyecto se procederá a ponerlo a prueba. Antes de nada cabe decir que hasta que se ha conseguido que el programa funcione aceptablemente, esto es, que no se cuelgue, ni suene de manera incorrecta ya sea por un sonido discontinuo o con clics, se han usado diversos métodos para comprobar los valores de las diferentes variables involucradas en el procesamiento.

Por una parte, para hacer pruebas de ejecución del programa se ha usado el programa oficial de testeo de Audio Unit: AU Lab. Este programa nos permite crear una mesa de mezclas virtual a la que podemos asignar Audio Unit, ya sean instrumentos o, como es el presente caso, efectos. A cada canal de la mesa se le puede asignar como generador de sonido un reproductor en el que cargaremos los ficheros de audio para aplicarles el pertinente efecto.

Figura 13: AU Lab

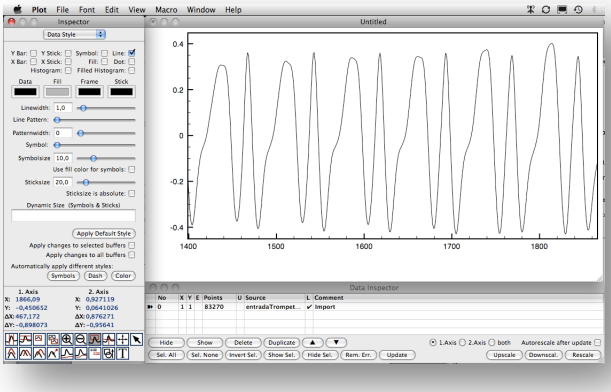

Figura 14: Plot

Complementariamente se han ido añadiendo líneas de código temporales, que se han eliminado en la versión distribuida, para generar ficheros de texto con los valores que se han necesitado en su momento y de este modo, además de ver los resultados en texto también se han representado en gráficas mediante el programa gratuito Plot, disponible para Mac OSX. También se ha usado el editor de audio Audacity ya que permite realizar la transformada de Fourier de la señal que se indique siendo representada en un gráfico además de pudiendo ser exportada a texto. Dado que estas han sido pruebas de desarrollo y no de funcionalidad, se ha desestimado dejarlas reflejadas en la presente memoria.

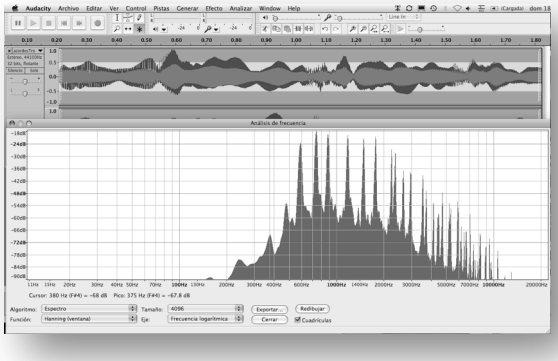

Figura 15: Audacity

4.2. - PRUEBAS DE EVALUACIÓN

Para evaluar el proyecto se han tomado dos medidas: una objetiva con el registro de ciertos datos de entrada y salida, y otra perceptual que nos da resultados audibles de los que se puede opinar en cuanto a lo que se refiere a la exactitud del resultado perseguido.

En ambos casos se ha dispuesto una serie de ficheros de audio con características concretas para poder determinar así si el programa es aceptable dentro del rango de casos para el que se ha desarrollado. Esto es, deben de ser composiciones en las que la melodía tenga un volumen claramente mayor que el resto de las voces, además de ser un instrumento con espectro teóricamente armónico.

Primero se hará una descripción de la forma de trabajar para realizar las pruebas y después se dejará constancia de las pruebas realizadas con los diferentes candidatos.

Por último cabe reseñar que, dado que el programa funciona en tiempo real y dispone de parámetros ajustables y guardables durante la escucha, las pruebas se han realizado con el mejor ajuste de parámetros posible. Además se realizarán pruebas con diferentes tamaños de ventana, para comprobar la influencia de este parámetro que depende del programa anfitrión que se use para aplicar el efecto.

4.2.1. PRUEBAS OBJETIVAS

Para obtener datos concretos de la evaluación del software, se ha comprobado la fidelidad de la estimación de frecuencia fundamental del algoritmo. Se tomó este camino dado que esta estimación es el motor de los resultados obtenidos, esto es, sin una frecuencia fundamental correcta nunca podremos obtener un espectro armónico semejante al que

buscamos. Para poder llevar a cabo esta tarea se han utilizado varios programas.

4.2.1.1.-CREACIÓN DE PIEZAS MUSICALES A USAR

En primer lugar para generar las composiciones musicales a usar se ha usado el secuenciador Garage Band, que es el hermano pequeño de Logic Pro, descrito en puntos anteriores. Se ha creado un proyecto en dicho programa y se ha generado un archivo MIDI del instrumento de la melodía principal que es leído por un sampler[1] virtual que contiene muestras de los diferentes instrumentos que se usan. Ese archivo MIDI se exporta para la posterior lectura de las notas ejecutadas. Entonces añadimos al proyecto las pistas para generar el sonido restante que estará en segundo plano y se compone tanto de instrumentos afinados como no afinados, o incluso ruido. Finalmente exportamos la composición en formato mp3.

4.2.1.2.-EXTRACCIÓN DE DATOS DE ENTRADA Y SALIDA

Para poder comparar datos de entrada y salida fácilmente se ha hecho uso de el software "Standard MIDI file to text" desarrollado por el Departamento de Lenguaje y Sistemas Informáticos de la Universidad de Alicante[2]. Este programa, que se maneja desde el terminal de Mac OSX, nos permite crear un fichero de texto a partir de uno MIDI en el que podemos reflejar entre otras cosas la altura, el instante de aparición y la duración de las notas presentes en el fichero MIDI. De este modo obtenemos en texto los valores de la melodía original.

[1] Un sampler es un aparato que permite muestrear (grabar) digitalmente secuencias sonoras o samples para ser reproducidas posteriormente tal cual fueron grabadas, o transformadas mediante efectos.

[2] Descarga gratuita en: http://grfia.dlsi.ua.es/

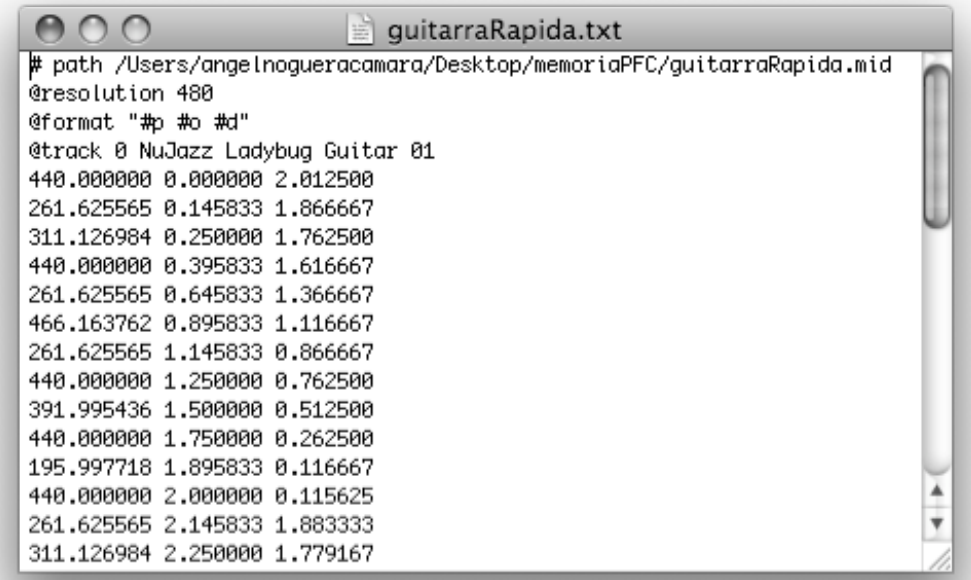

Figura 16: Fichero de salida de smf2txt con la altura, instante de aparición y duración de las diferentes notas presentes en un fichero MIDI

Para obtener los valores de la frecuencia generada por el efecto lo hacemos mediante líneas de código temporales que generen ficheros de texto con los valores de la frecuencias fundamentales predominantes estimadas. Estas líneas su usan para la evaluación y por tanto no estarán presentes en el código final.

4.2.1.2.-COMPARACIÓN DE DATOS DE ENTRADA Y SALIDA

Finalmente comparamos los resultados con un programa de hojas de cálculo, ya que los valores de onset de la entrada los tenemos en segundos y los de la salida son frame a frame y habrá que calcular por tanto la correspondencia en segundos dependiendo de la frecuencia de muestreo y el tamaño de ventana escogidos. Se considerarán valores correctos los que no difieran en un valor igual a un semitono o superior, esto es, el producto o la división del valor a evaluar por la raíz duodécima de dos.

4.2.1. PRUEBAS PERCEPTUALES

Una vez hechas las pruebas objetivas, se procederá a eliminar las líneas de código temporales para generar textos con los valores de las variables, y se ejecutará el efecto con los mismos archivos de las pruebas objetivas para que sean escuchados y dar una opinión en cuanto a fidelidad del sonido sintetizado con respecto al instrumento original. Esto es, si la similitud es aceptable, o si al menos nos permite diferenciar de qué instrumento se trata.

4.3. – BATERÍA DE PRUEBAS

4.3.1. UNA SOLA VOZ

Aquí usaremos cada instrumento tocando una escala de su octava más característica, esto es, para un trombón usaremos la octava segunda y para una flauta, que es más aguda, la sexta. En todos los casos se empezará y acabará con la misma nota, pero en una octava superior. En teoría, con los parámetros bien ajustados las pruebas objetivas deberían dar un 0% de fallo, ya que no hay más sonido que el del instrumento. Aún así este grupo de pruebas es útil para obtener una idea de cómo afecta al timbre del instrumento, ya que aunque la fundamental sea correctamente detectada el espectro generado no tiene por qué ser exactamente igual. El error se calcula en base al número de ventanas acertadas con respecto al número de ventanas reproducidas.

INSTRUMENTO	FRECUENCIAS	ERROR
Trombón	98Hz-196Hz	4,3%
Guitarra	330Hz-660Hz	0 %
Saxofón Soprano	587Hz-1175Hz	26%
Trompeta	587Hz-1175Hz	0%
Viola	587Hz-1175Hz	22%
Flauta	1045Hz-2093Hz	0%
Piccolo	1045Hz-2093Hz	57%

Tabla 1: Pruebas de detección de frecuencia fundamental con una sola voz

VALORACIÓN OBJETIVA

Las pruebas de la tabla 1 se han realizado con un tamaño de ventana igual a 4096 muestras. Para tamaños inferiores las notas graves se ven gravemente afectadas.

La gran mayoría de errores son causados porque la frecuencia fundamental no siempre es el armónico de mayor amplitud, en muchos casos se ha detectado el primer parcial como fundamental ya que la escala estaba acotada exactamente a una octava. Por tanto si la melodía que pretendemos extraer no llega al rango de una octava sería posible la estimación sin error, siempre y cuando acotáramos los parámetros de frecuencias mínima y máxima dentro de esta escala.

También se ha observado que para casos con espectros menos comunes, como es el del Piccolo el error es mayor, ya que nuestro caso teórico no se cumple siempre debido a que los armónicos nos siempre son múltiplos de la fundamental. Aunque sus notas sean agudas y no tengamos problemas de distancias cortas de nota a nota.

Hay que recordar que estas pruebas se han realizado con los parámetros bien ajustados. Aún así, para mostrar lo importante que es el ajuste de los parámetros diremos que por ejemplo para el caso específico de la guitarra que tiene un 0% de error, este se ve incrementado a un 39%, que es una cifra demasiado grande teniendo en cuenta que sólo está sonando este instrumento.

VALORACIÓN PERCEPTUAL

En cuanto al timbre de los sonidos generados, todos comparten más o menos las mismas características con respecto a los sonidos originales. Son sonidos ligeramente menos brillantes y con algo menos de detalle, aunque todos son reconocibles y diferenciables. El efecto produce sonidos indeseados en los cambios de nota, debido a que las interpolaciones de amplitud y fase no son perfectas. Esto no es un problema si se pretende obtener las notas separadas. Por supuesto, tampoco se perciben los sonidos producidos por la propia interpretación, ya sea por percutir o por rasgar la cuerda, aunque esto es algo que no se busca en el presente trabajo.

4.3.2. UNA SOLA VOZ CON RUIDO

A continuación se procederá a coger el archivo de escala de trompeta del apartado anterior y se introducirá ruido. Serán dos tipos de ruido: por una parte el crujido de un vinilo viejo y por otra parte ruido blanco, que simula el causado por la grabación en cintas magnéticas. Se introducirán por separado y juntos. Como el archivo sin ruido no tenía error, el error que pueda salir será causado por tanto por la presencia del ruido.

INSTRUMENTO	FRECUENCIAS	ERROR
Trompeta + ruido blanco	587Hz-1175Hz	0%
Trompeta + ruido de vinilo	587Hz-1175Hz	8,7%
Trompeta + ambos ruidos	587Hz-1175Hz	0 %

Tabla 2: Pruebas de detección de frecuencia fundamental con una voz y ruido

VALORACIÓN OBJETIVA

Como observamos el error es nulo o bajo, por tanto parece que el efecto no es muy sensible al ruido de fondo. Si el ruido es demasiado fuerte, por supuesto el efecto deja de ser funcional.

4.3.3. VARIAS VOCES DE UN INSTRUMENTO

Seguidamente se mostrará el resultado obtenido al procesar una pieza musical de piano con varias voces y la misma pieza interpretada por una sección de cuerdas, esto es violines, violas y cellos. Se compone de cuatro compases con una voz sobresaliente que se mueve en dos octavas diferentes. Las voces acompañantes son más graves y por supuesto con un volumen menor.

INSTRUMENTO	FRECUENCIAS	ERROR
Piano (varias voces)	523Hz-1661Hz	21,83%
Sección cuerdas (staccato)	523Hz-1661Hz	11,49%

Tabla 3: Pruebas de detección de frecuencia fundamental con varias voces de la misma familia

VALORACIÓN OBJETIVA

Se puede observar como el error aumenta en ambos casos con respecto a los apartados anteriores, debido a que es posible que los primeros armónicos de las voces de acompañamiento tengan mayor amplitud que las frecuencias fundamentales de la melodía sobre todo en sus tiempos de relajación.

En cuanto a la diferencia entre los dos casos de este apartado es probablemente debido a que las notas de el piano tienen un tiempo de relajación mayor que las de la sección de cuerda que son más cortas ya que están interpretadas en "staccato". De hecho es posible que las frecuencias detectadas en varios momentos determinados del segundo caso sean parciales del acompañamiento ya que debido a la interpretación hay instantes en los que no hay melodía.

VALORACIÓN PERCEPTUAL

Del parecido del sonido original con el sintetizado en este caso se puede decir que es menos fiel que en los casos anteriores ya que ahora los armónicos de la melodía se ven solapados por armónicos de las notas más graves del acompañamiento. El sonido molesto entre cambios de notas sigue percibiéndose.

4.3.4. VARIAS VOCES DE INSTRUMENTOS DIFERENTES

En este último apartado de pruebas se procesa el tipo de composición para la cual el efecto se ha desarrollado, esto es, una melodía de un instrumento armónico sobre un acompañamiento en segundo plano, compuesto en este caso concreto por percusión, bajo y acordes de órgano

electrónico. La melodía es diferente para cada instrumento, aunque el acompañamiento es el mismo, para así poder observar las diferencias entre instrumentos.

INSTRUMENTO	FRECUENCIAS	ERROR
Viola con acompañamiento	750 Hz - 1100 Hz	2,30%
Piccolo con acompañamiento	587 Hz - 784 Hz	26,74%
Guitarra con acompañamiento	155 Hz - 261 Hz	42,52%
Clarinete con acompañamiento	587 Hz - 1174 Hz	1,14%

Tabla 4: Pruebas de detección de frecuencia fundamental con voz principal y acompañamiento

VALORACIÓN OBJETIVA

A la vista de la tabla hay que comentar que la detección de frecuencia no es correcta, al igual que en apartados anteriores, en notas graves y en espectros fuera de los casos teóricos armónicos. Los fallos en la detección del pasaje de guitarra se ven acentuado porque las notas del bajo se pueden cruzar en la detección con las de la melodía. Respecto al caso del Piccolo ya se ha comentado que no tiene el espectro en el que se basa el algoritmo de detección usado.

En los casos donde la melodía no es grave ni excesivamente rápida, tiene mayor volumen que el acompañamiento y un espectro con la frecuencia fundamental presente, las tasas de error son muy bajas.

VALORACIÓN PERCEPTUAL

De la percepción del timbre hay que decir que en notas sostenidas la dinámica espectral sintetizada es bastante aceptable, siempre y cuando la frecuencia fundamental sea correcta. Para las notas cortas se da el problema de que la interpolación en la transición de nota a nota genera sonidos indeseados.

No hay que olvidar que para tamaños de ventana inferiores los resultados son peores debido a la pérdida de precisión en la detección de la frecuencia fundamental.

Conclusiones 5

5 Conclusiones

5.1. – CONCLUSIONES Y TRABAJO FUTURO

Sin lugar a duda, trabajando con un tamaño de ventana grande, se han conseguido resultados bastante aceptables en los casos más ideales. Esto ya es un logro sobre todo teniendo en cuenta que el efecto desarrollado funciona en tiempo real.

Un posible uso que se puede dar actualmente al efecto es el de generar los archivos necesarios para un sampler y poder así rescatar instrumentos individuales grabados con más voces. Se ha visto que para notas sostenidas los resultados son buenos, por tanto se puede hacer una grabación de la salida del efecto de estas notas separadas, posteriormente cargarlas en un sampler y finalmente tocarlas mediante un teclado controlador.

Otro posible uso, como ya se ha mencionado en el apartado de experimentación es el de limpieza de archivos con ruido debido a los soportes de grabación o el ruido ambiente, como por ejemplo las canciones antiguas o una grabación exterior.

Como trabajo futuro lo primordial es conseguir añadir exitosamente la técnica 'zero padding' u otra técnica que permita obtener más precisión en la detección de frecuencia fundamental, sin sobrecargar demasiado el proceso. Con esto se reducirían notablemente los errores en la detección de frecuencia fundamental, obteniendo sonidos más afinados y espectros más parecidos a los originales ya que los armónicos se calculan en base a esta.

El algoritmo de detección de frecuencia también puede ser optimizado o modificado, ya que aunque el actual es sencillo y tiene en cuenta el espectro de la señal de entrada, se ha podido comprobar que en determinados casos no funciona correctamente. Sería interesante encontrar o crear un algoritmo que tenga en cuenta más parámetros y

simplificarlo para poder añadirlo al efecto.

También se pueden añadir nuevos parámetros ajustables en reproducción y de configuraciones guardables, ya que se ha podido ver la importancia de tener bien ajustados los mismos si queremos obtener los mejores resultados.

Además, la interpolación de frecuencia y fase debería corregirse para obtener un sonido más agradable en las transiciones de nota. De este modo se podrían encontrar nuevos y diferentes usos para el efecto desarrollado, como por ejemplo usar directamente la salida del efecto sobre una composición creada o una interpretación en vivo.

El código es abierto y está disponible en Internet en la página del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante para poder ser descargado libremente[1]. La memoria y la batería de archivos de prueba también están accesibles al público en la red.

Como conclusión final se puede decir que se ha dado un primer paso hacia la extracción de melodías en tiempo real en archivos de audio musicales mediante procesamiento digital de señal.

[1] http://grfia.dlsi.ua.es/cm/worklines/anoguera-pfc

A.- CÓDIGOS FUENTE

A.A.- SUBFUNCIONES DE PROCESO

A.A.A.- POSICIÓN A FRECUENCIA

Esta función recibe una frecuencia en hercios y una resolución frecuencial y devuelve el correspondiente índice del vector.

```
int      MelodyExtraction::MelodyExtractionKernel::FreqToPos(Float32 freq, Float32
freqRes)
{
 int pos;
 pos = freq / freqRes;
 if (pos < 1)
 {
        pos = 1;
 }
 return pos;
}
```

A.A.B.- FRECUENCIA A POSICIÓN

Esta función recibe índice del vector y una resolución frecuencial y devuelve la correspondiente frecuencia en hercios.

```
Float32  MelodyExtraction::MelodyExtractionKernel::PosToFreq(int pos, Float32
freqRes)
{
 Float32 freq;
 freq = (Float32)pos * freqRes;
 if (freq < 20)
 {
        freq = 20;
 }
 return freq;
}
```

A.A.C.- RMS

Esta función recibe un vector y su tamaño y devuelve el valor RMS de sus elementos.

```
Float32 MelodyExtraction::MelodyExtractionKernel::RMS(Float32 array[], int size)
{
       int i;
       Float32 sum = 0;
       for(i = 0 ; i < size ; i++)
       {
              sum = sum + array[i]*array[i];
       }
       sum = sqrt(sum/size);
       return sum;
}
```

A.A.D.- HPS

Esta función implementa el 'Harmonic Product Spectrum'.Recibe un vector lo reduce al factor dado (el resto se rellena con ceros) y lo multiplica por el vector resultado. Se debe llamar a esta función tantas veces como armónicos se quiera tener en cuenta.

```
void MelodyExtraction::MelodyExtractionKernel::HPS(int winSize, int parNum)
{
  int i;
  Float32 reducedSpectrum[winSize];

  for(i = 0; i < winSize/(parNum) ; i++)
  {
         reducedSpectrum[i] = inSpectrum.mod[i*parNum];
  }

  for(i = winSize/(parNum); i < winSize ; i++)
  {
         reducedSpectrum[i] = 0;
  }

  for(i = 0 ; i < winSize ; i++)
  {
         averageSpectrum[i] = averageSpectrum[i]*reducedSpectrum[i];
  }
}
```

A.A.E.- BUSCADOR DE MÁXIMO

Función que busca primero picos dentro de un rango de un vector y después el máximo de esos picos.

```
int      MelodyExtraction::MelodyExtractionKernel::MaximumSearch(int begOfRange, int
endOfRange){
  int maxPos = - 1, i, j = 0;
  Float32 maxVal = 0;
  if(begOfRange > endOfRange){
         begOfRange = endOfRange -1;
  }
  for(i = begOfRange ; i < endOfRange - 1; i++){
         if(averageSpectrum[i-1] < averageSpectrum[i] && averageSpectrum[i+1] <
averageSpectrum[i]){
                peaks[j] = i;
                j++;
         }
  }
  for(i = 0 ; i < j ; i++){
         if(averageSpectrum[peaks[i]] > maxVal){
                maxVal = averageSpectrum[peaks[i]];
                maxPos = peaks[i];
         }
  }
  return maxPos;
}
```

A.A.F.- SÍNTESIS

Función que sintetiza una señal suma de sinusoides a partir de modulo, fase y frecuencia de los diferentes parciales. Se hace uso de fórmulas de interpolación de amplitud y fase.

```
void     MelodyExtraction::MelodyExtractionKernel::Synthesis(UInt32 nSample, int
partInd)
{
Float32 sinusoidal[nSample], amp, ampmed, x = 0, nu = 0, tau = 0, faseinstantanea;
int n;

ampmed = (part[partInd].mod - prePart[partInd].mod)/((double)nSample);

x = (1.0/(2.0 * M_PI)) * ((prePart[partInd].phas + prePart[partInd].omega *
(double)nSample - part[partInd].phas) + ((double)nSample/2.0) * (part[partInd].omega-
prePart[partInd].omega));

nu = (3.0/((double)nSample*(double)nSample)) * (part[partInd].phas -
prePart[partInd].phas - prePart[partInd].omega * (double)nSample + 2.0 * M_PI *
round(x)) - (1.0/(double)nSample) * (part[partInd].omega - prePart[partInd].omega);

tau = -(2.0/((double)nSample*(double)nSample*(double)nSample)) * (part[partInd].phas
- prePart[partInd].phas - prePart[partInd].omega * (double)nSample + 2.0 * M_PI *
round(x)) - (1.0/(double)nSample*(double)nSample) * (part[partInd].omega -
prePart[partInd].omega);

for(n = 0 ; n < (int)nSample ; n++)
{
 amp = prePart[partInd].mod + (ampmed * (Float32)n);
 faseinstantanea = prePart[partInd].phas + prePart[partInd].omega * (Float32)n + nu *
((Float32)n * (Float32)n) + tau * ((Float32)n * (Float32)n * (Float32)n);

 sinusoidal[n] = amp * cos(faseinstantanea/GetSampleRate());

 synthSignal[n] = synthSignal[n] + sinusoidal[n];
}
 return;
}
```

A.A.G.- SOLAPADO

Función que solapa un tanto por ciento un vector, con su correspondiente en el frame anterior.

```
void MelodyExtraction::MelodyExtractionKernel::Overlap(int size, Float32 coef)
{
 int i, cue = size*coef/100;

 for(i = 0 ; i < cue ; i++)
 {
        overlapSignal[i] = preSynthSignal[size - cue + i];
 }

 for(i = cue ; i < size ; i++)
 {
        overlapSignal[i] = synthSignal[i - cue];
 }
}
```

A.A.H.- SELECTOR DE ARMÓNICOS

Función que selecciona el armónico correspondiente alrededor de una frecuencia central dentro de un rango dado. Aplica un coeficiente triangular reductor a medida que las frecuencias se alejan de la central.

```
int MelodyExtraction::MelodyExtractionKernel::SelectPeakCloseTo(Float32 spectrum[],
int tamwindow, int sr, Float32 centerfreq, int tolerance)
{
 int nearest=-1;
 double maxval;
 double lowlimit, highlimit;

 if (centerfreq < sr/2)
 {
        lowlimit = centerfreq - tolerance;
        highlimit = centerfreq + tolerance;

        maxval=0;

        int i;
        for (i = 0; i < tamwindow && (Float32)i * ((Float32)sr/(Float32)tamwindow)
<= lowlimit; i++);
        while (i < tamwindow && (Float32)i * ((Float32)sr/(Float32)tamwindow) <=
centerfreq)
        {
                double bandfactor = ( (Float32)i * ( (Float32)sr / (Float32)tamwindow
) - lowlimit) / ( centerfreq - lowlimit);
                double val = spectrum[i] * bandfactor;
                if (val > maxval)
                {
                        maxval = val;
                        nearest = i;

                }
                i++;
        }

        while (i < tamwindow &&
((Float32)i*((Float32)sr/(Float32)tamwindow)<=highlimit))
        {
                double bandfactor = ((highlimit - centerfreq) - ((Float32)i * (
(Float32)sr / (Float32)tamwindow ) - centerfreq)) / (highlimit - centerfreq);

                double val = spectrum[i] * bandfactor;
                if (val > maxval)
                {
                        maxval = val;
                        nearest = i;

                }
                 i++;
        }
 }
 return nearest;
}
```

A.A.I.- TRANSFORMADA DE FOURIER RÁPIDA

Función que realiza la transformada de Fourier rápida mediante el algoritmo FFTW.

```
      void mus_fftw(Float32 *rl, int n, int dir)
      {
        int i;
        if (n != last_fft_size)
          {
            if (rdata) {fftw_free(rdata); fftw_free(idata); fftw_destroy_plan(rplan);
fftw_destroy_plan(iplan);}
            rdata = (double *)fftw_malloc(n * sizeof(double));
            idata = (double *)fftw_malloc(n * sizeof(double));
            rplan = fftw_plan_r2r_1d(n, rdata, idata, FFTW_R2HC, FFTW_ESTIMATE);
            iplan = fftw_plan_r2r_1d(n, rdata, idata, FFTW_HC2R, FFTW_ESTIMATE);
            last_fft_size = n;
          }
        memset((void *)idata, 0, n * sizeof(double));
        for (i = 0; i < n; i++) {rdata[i] = rl[i];}
        if (dir != -1)
          fftw_execute(rplan);
        else fftw_execute(iplan);
        for (i = 0; i < n; i++) rl[i] = idata[i];
      }

      void fourier_spectrum(Float32 sf[], Float32 fft_data[], Float32 fft_phase[], int
fft_size, int data_len, Float32 window[], int win_len)
      {
       int i;
       for (i = 0; i < win_len; i++)

       fft_data[i] = window[i] * sf[i];

       int j;
       mus_fftw(fft_data, fft_size, 1);

       fft_data[0] = fabs(fft_data[0]);
       fft_data[fft_size / 2] = fabs(fft_data[fft_size / 2]);

       for (i = 1, j = fft_size - 1; i < fft_size / 2; i++, j--)
       {
              fft_data[i] = hypot(fft_data[i], fft_data[j]);
              fft_phase[i]= atan(fft_data[j]/fft_data[i]);
       }

       for (int i=0; i<fft_size/2; i++)
       if (isNaN(fft_data[i]))
              fft_data[i]=0.0;
      }
```

A.A.J.- VENTANA HANNING

Función que devuelve los coeficientes de una ventana Hanning.

```
      void Hanning(Float32 *window, int size) {
       for (int i=1; i<=size; i++)
              window[i-1]=0.5*(1-cos(2*M_PI*((double)i/(double)(size+1))));
      }
```

BIBLIOGRAFÍA

Apple Inc. (2007). "Audio Unit Programming Guide"

Berenzweig, A. L. & Ellis, D.P.V. (2001). "Locating singing voice segments within music signals"

Cheveigné, A. (2002). "YIN, a fundamental frequency estimator for speech and music"

Eggink, J. & Brown, G.J. (2004). "Extracting melody lines from complex audio"

Middleton, G. (2003). "Pitch detection algorithms"

MIREX (2008). "Music Information Retrieval Evaluation Exchange. Multiple fundamental frequency estimation and Trucking contest" http://www.musicir.org/mirex/2007/index.php/Multiple_Fundamental_Frequency_Estimation_%26_Tracking_Results.

Moorer, J. A. (1977) . "Signal processing aspects aspects of computer music: A Survey. Computer Music Journal"

Pertusa, A. (2010). PhD "Computationally efficient methods for polyphonic music transcription"

Pertusa, A., Iñesta, J.M. (2010). "Multiple fundamental frequency estimation using gaussian smoothness"

Schouten, J.F. (1940). "The residue and the mechanism of hearing. En Proccedings Kononklijke Nederlandse Akademie van Wetenschappen, volumen 43"

Serra, X. (1997). "Musical Sounds Modelling with sinusoids plus Noise"

Serra, X. (1989). PhD "A system for sound analisis/transformation/synthesis based on a deterministic plus stochastic decomposition"

Universidad de Alicante (2007). Apuntes de Síntesis Digital de Sonido

Universidad de Alicante (2008). Apuntes de Tratamiento Digital de Audio