IMPLEMENTACION DE EFECTOS PARA SEÑALES DE AUDIO

REFERENCIA PACS: 43.60.Bf.

Enric Guaus, Ivana Rossell, Lucas-Ariel Vallejos
Departamento de Acústica
Ingenieria La Salle
Universidad Ramon Llull
Pg. Bonanova nº 8
08022 Barcelona
E-mail: eguaus@salleURL.edu

ABSTRACT

All the audio programs have some effects. These effects can be real time effects or off-line effects. We present a study of two of these effects (Trémolo & Vibrato) and a Noise Reduction real time implementation. These real-time algorythms are implemented to be a Plug-in in the Cubase VST audio and MIDI program.

1. INTRODUCCIÓN

Muchos sistemas de grabación, amplificación o generación de señales de audio tienen una parte dedicada a los efectos. Incluso hay módulos especializados a la generación de efectos. Qué son estos efectos? Como se construyen? Cuál es su funcionamiento? Un efecto para una señal de audio es, a grandes rasgos, la modificación de una señal de entrada para conseguir que el sonido alcance unas características en concreto a la salida. Así mismo, los parámetros de cada efecto pueden ser modificados durante la intepretación de una pieza o pasaje musical. Nuestro trabajo consiste en el estudio de efectos tanto desde el punto de vista analógico cómo digital, y cómo debe ser la implementación de estos algorismos a tiempo real. Se explicará con la ayuda de señales de audio para ver las diferéncias entre los distintos parámetros.

2. EFECTOS BÁSICOS: TRÉMOLO Y VIBRATO

Són los efectos más usuales en los programas de edición de audio y quizás los mas sencillos de implementar. El trémolo consiste en una modulación de la amplitud de la señal, mientras que el vibrato consiste en uma modulación en frecuencia de la señal.

2.1. Trémolo

El efecto de trémolo consiste en una modulación de amplitud de la señal de entrada. La señal portadora será nuestra señal de audio, mientras que la señal moduladora será típicamente una señal sinusoidal de baja frecuencia (normalmente entre 1Hz y 20Hz). Matemáticamente, el Trémolo se expresa mediante la fórmula de modulación AM.

$$s_{AM}(t) = A[1 + a * m_n(t)] * \cos(wt)$$

donde a es el índice de modulación (comprendido entre 0 y 1) y ω es la frecuencia del LFO. Debido a que el cálculo del cos(wt) es computacionalmente caro, utilizaremos una Tabla de Onda con el fin de tener precalculados en cada momento el valor del cos(wt) consiguiendo que el algorismo sea mucho más rápido. El funcionamiento de la tabla de onda se explica con más detalle en el próximo apartado.

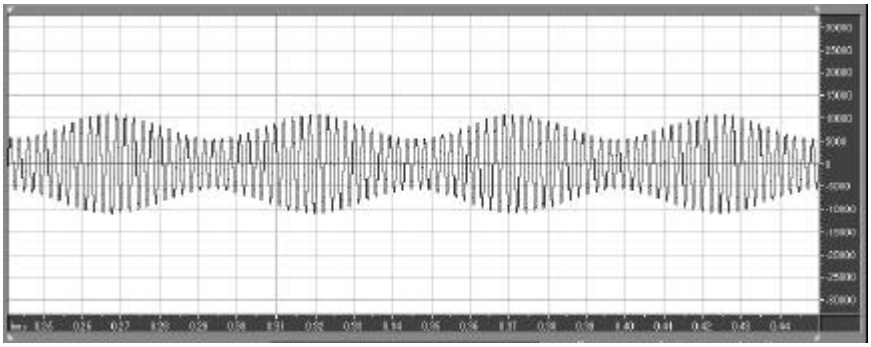


Fig. 1: Señal con Trémolo

2.2. Vibrato

Muchos son los cantantes que prefieren incorporar un poco de vibrato a sus interpretaciones principalmente por dos motivos: da mas profundidad a la voz y disimula pequeños errores de afinación. El efecto de Vibrato consiste en una modulación en frecuencia de una señal de Audio.

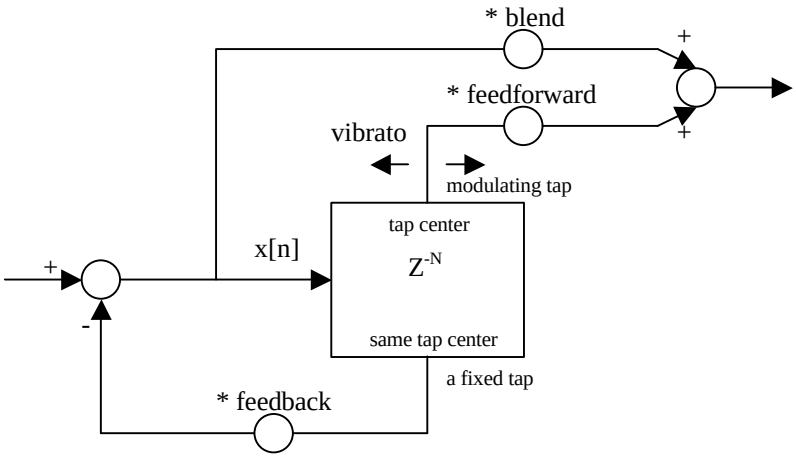
$$s_{FM}(t) = A_c * \cos\left(w_c t + c_f \int m(t)dt\right)$$

donde c_f es la frecuencia instantánea de la señal y m(t) es la señal de audio.

A la vista de la fórmula de una modulación FM vemos que no es implementable a la práctica, puesto que necesitamos calcular una integral cuyos límites llegan hasta el infinito. Para solucionar este problema, trabajaremos con líneas de retardo.

Las líneas de retardo son una técnica de procesado digital muy versátil, permitiendo una gran variedad de efectos. Con las líneas de retardo implementaremos un retardo temporal a la señal de audio, y con diferentes modificaciones de este retardo tendremos un efecto u otro. Podemos hacer ahora una primera división: efectos con retardo fijo y efectos con retardo variable. En los efectos con retardo fijo (típicamente llamados delay) el retardo se mantiene constante mientras la señal pasa a trevés de la línea. Si el retardo fijo es de muy corta duración (<10ms) conseguimos cambios en el dominio frecuencial puesto que estamos multiplicando el espectro de la señal de audio con la respuesta en frecuencia de un filtro FIR paso-bajo. Si el retardo es de duración media (entre 10 y 50 ms) conseguiremos efectos de ambiente, como pueden ser reverberaciones de salas, etc. Finalmente, si el retardo es de larga duración (>50ms) obtendremos eco. La combinación entre diferentes reverberaciones y ecos permiten infinidad de posibilidades, entrando ya en temas de localización de fuentes o distintos efectos de reverberación para distintas salas.

Hablemos ahora de los efectos con retardo variable. En este tipo de efectos, el retardo varia de forma controlada mientras la señal pasa a través de la línea. Los efectos más típicos con retardo variable són el Flanging, el Phasing, el Vibrato y el Chorus. El flanging es quizás el más antiguo de este tipo de efectos, y se decubrió casi por casualidad (Christian Huygens 1963). Consiste en reproducir por dos grabadores analógicos la misma señal de audio (de banda ancha) y en uno de los dos grabadores presionando ligeramente con el dedo la bobina (de ahi su nombre) modificando la velocidad de la cinta. Digitalmente, se inplementa con una línea de retardo variable cuyo retardo viene controlado por un LFO de fecuencias comprendidas entre 0.1 y 20 Hz. El efecto conseguido es un comb filter cuyos ceros se van moviendo en frecuencia. El Phasing es similar al flanging, pero los filtros utilizados tienen respuesta plana (filtros paso todo), y solo modificamos la fase de la señal original. Tendremos distintos retardos para distintas frecuencias. Con este efecto también conseguimos un comb filter aunque no tan pronunciado como el flanging e introducimos muchos harmónicos a nuestra señal de entrada. El Vibrato consiste en controlar el retardo de la línea mediante una señal senoidal, de la cual controlaremos su amplitud (depth) y la frecuencia (rate). Evidentemente, la amplitud (depth) del Vibrato nos determinará cuál es el retardo (delay) inicial sobre el cual oscilará la señal sinusoidal. Finalmente, el Chorus consiste en un Vibrato al que se le suma la señal original, consiguiendo un efecto que aplicado a un único instrumento parece que estén tocando varios de éllos a la vez.



Effect	Blend	Feedforward	Feedback
Vibrato	0	1	0
Flanger	0.7071	0.7071	-0.7071
Standard Chorus	1	0.7071	0
White Chorus	0.7071	1	0.7071
Doubling	0.7071	0.7071	0
Echo	1	<=1	<1

Fig. 2: Industry Standard Chorus Effect with feedback

Centrémonos ahora en el estudio del Vibrato. Para su implementación se recurre a una línea de retardo L con una longitud N=2048. El principal problema que nos encontramos es que según la elección de los parámetros CHORUS_WIDTH y RATE la posición del cursor nunca corresponderá a la posición exacta de una de las casillas de la línea. Será necesario recurrir a la interpolación de la línea de retardo. En la Fig. 3 puede verse mas detalladamente su funcionamiento.

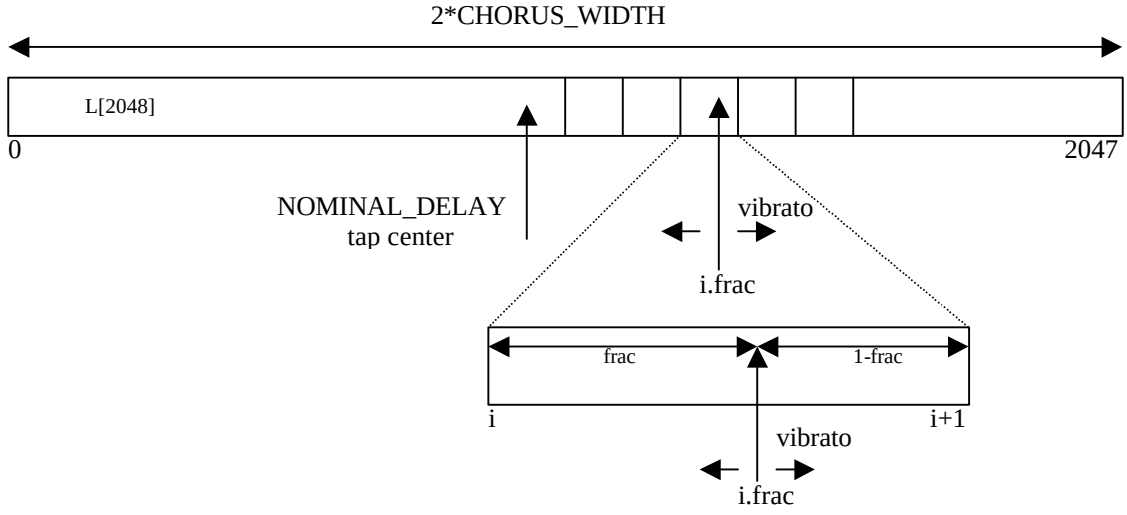


Fig. 3: Funcionamiento del Vibrato

La salida de nuestro sistema (i.frac) viene dada por el valor leido por el puntero que normalmente caerá entre dos muestras. El índice i (número entero) se define como la muestra actual calculada cómo índice entero relativo desde el principio de la línea de retardo, es decir

$$frac = i.frac - i$$

El índice i necesita calcularse para cada muestra, puesto que queremos que el retardo sea modulado por una señal de baja frecuencia (LFO) y[n], oscilando en función de la variable

n. El márgen dinámico de i estará comprendido en ±CHORUS_WIDTH y centrado en según el retardo original dado a nuestro vibrato NOMINAL_DELAY, número entero y siempre positivo.

$$i.frac = i + frac$$

Debido a que i.frac, real y positivo, variable en el tiempo, es el puntero relativo al orígen de la línea de retardo, el valor de frac será siempre real y positivo, comprendido entre los valores [0,1[. El algorismo también precisará calcular la fracción relativa hasta la muestra siguiente (1-frac), índice real y positivo.

Para la implementación a tiempo real es preciso disponer de una cola circular y de todas las funciones adjuntas (Push, Pull, etc.) Pero lo que es crítico en la implementación del Vibrato es la señal y[n]. A simple vista pensamos en calcular cada muestra de y[n] en función de CHORUS_WIDTH, y RATE, y de ahí calcular el valor de i.frac cómo

$$i.frac = NOMINAL\_DELAY + CHORUS\_WIDTH * y[n]$$

El problema de este método está en la pérdida de tiempo que supone el cálculo de y[n] para cada muestra (recordemos que se trata de un LFO). Para solucionar este problema hemos implementado una Tabla de Ondas. El funcionamiento de la tabla de ondas consiste en llenar de forma ordenada y progresiva mediante una cola circular una zona de memoria de nuestro programa con todos los valores que va a tomar y[n] a lo largo de la ejecución. De este modo, tan solo incrementando circularmente la posición del puntero n tendremos el valor de y[n] correspondiente a la muestra i.frac que deseemos calcular. Dentro de la Tabla de ondas se guarda un número entero de periodos de la señal y[n] y es de vital importancia que no haya saltos bruscos entre el principio y el fin de la tabla. Normalmente se hace coincidir los extremos de la tabla con pasos por cero de la señal y[n].

El cálculo de todas las muestras de y[n] se hará al principio de la ejecución, con lo que después de un pequeño retardo inicial, el tiempo de CPU gastado a partir de este momento será siempre constante y muy corto (tan solo un acceso a memoria!).

Llegados a este punto, se nos pantea un problema. En caso de modificar alguno de los parámetros del Vibrato (CHORUS_WIDTH, NOMINAL DELAY o RATE), cómo se midificará la tabla de ondas? Tenemos dos opciones: la primera consiste en modificar tan rápido como sea posible los valores de la tabla de ondas y seguir leyendo la tabla a la frecuencia de mostreo de la señal original (incrementos de n de uno en uno). La segunda opción consiste en mantener la tabla idéntica y modificar el incremento del índice n. Por ejemplo, si el RATE se multiplica por dos, en vez de incrementar el índice n de uno en uno, lo haremos de dos en dos. La elección de uno de estos dos métodos se deja en función de la poténcia de cálculo disponible. El primer método, aunque más sencillo, precisa de una poténcia de cálculo superior. Por ello, nosotros recomendamos el segundo método.

2. REDUCTOR DE RUIDO

Como se explica en la comunicación sobre "Reductor de ruido mediante la Transformada Wavelet", presentada en este mismo Tecniacústica'99 hemos implementado un reductor de ruido. Si bien los resultados han sido satisfactorios, la estructura de la implementación obliga a procesar toda una señal de audio por completo antes de sacar resultados. Evidentemente, esta implementación no es a tiempo real. Que hay que modificar para que la implementación sea a tiempo real?

2.1. Adaptación del algorismo a tiempo real

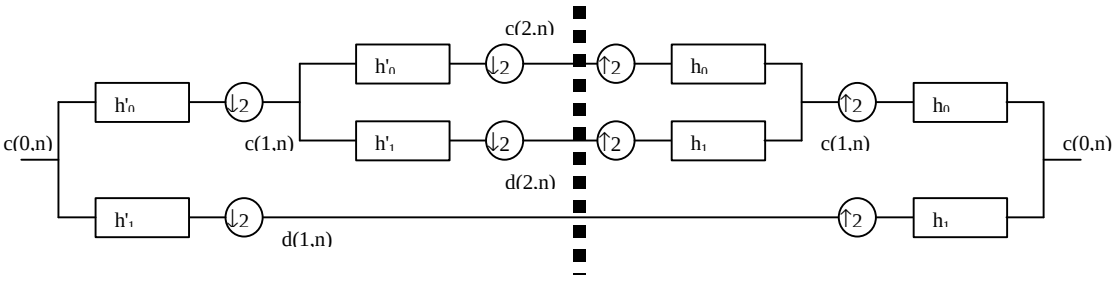


Fig. 4: Esquema Análisis multiresolución

Todos los buffers se implementarán mediante colas circulares. El filtrado se calcula en dominio temporal, es decir, mediante la convolución de la respuesta impulsional del filtro $h_i[n]$ y la señal de entrada en cada uno de los distintos niveles $x_i[n]$. Puesto que la longitud de los filtros no es muy elevada, resulta más rápida la convolución temporal que no calcular la FFT de la señal de entrada y la respuesta en frecuencia del filtro, multiplicar y finalmente calcular la FFT inversa a la salida. Es importante tener en cuenta que el aumento del número de niveles a descomponer no representa un aumento en la misma proporción del número de operaciones, puesto que a cada nivel la frecuencia de muestreo se divide por dos, con lo que tendremos que hacer muchas menos operaciones.

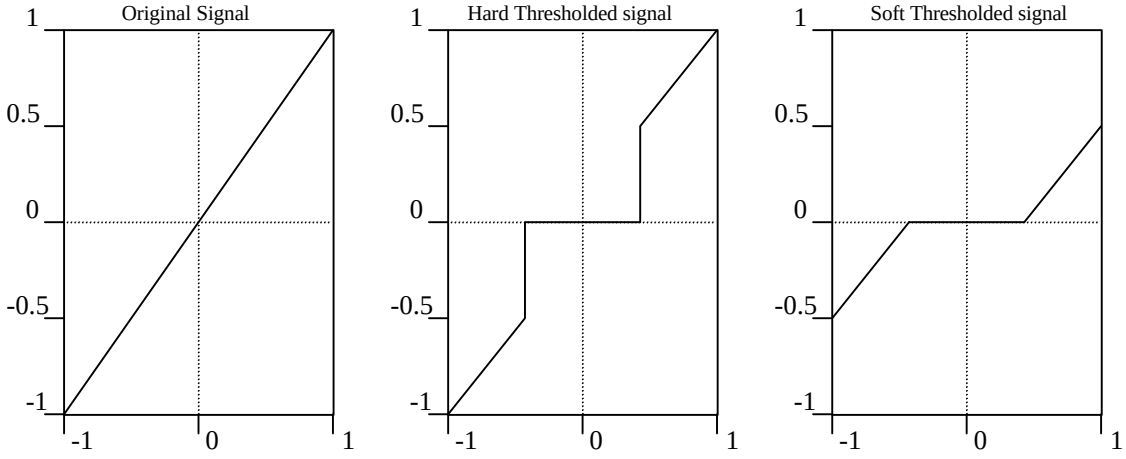


Fig: 5: Funcion de transferencia para los distintos métodos de Thresholding

Finalmente, el proceso de theresholding no presenta ningún problema puesto que se trata de una sencilla operación aplicada a cada muestra de cada nivel al final de la descomposición.

3. EL ENTORNO

Para implementar el algorismo en tiempo real nos hemos ayudado de un software libre en internet http://www.steinberg.net. Este software está pensado para la implementación de Plug-ins a tiempo real para el programa Cubase VST, de la firma Steinberg, que es un secuenciador MIDI con posibilidad de grabar audio y editarlo a través de la tarjeta de sonido. Dicho programa no es libre, puesto que es un programa muy completo y ampliamente usado en aplicaciones profesionales, tanto para PC como para Mac. Lo que nos encontramos en internet es un programa para testear nuestros plug-ins a tiempo real, es decir, que sobre un fichero *.wav, podemos escuchar y por tanto testear nuestro algorismo. Si poseemos el programa, solo hará falta copiar nuestro fichero *.dll dentro del directorio Vstplugins.

A la vez que nos bajamos este pequeño programa, en la página Web también encontramos documentación muy completa sobre cómo hay que implementar los plug-ins. Distintos ejemplos nos ayudan al aprendizage. Por otra parte, también se puede encontrar una serie de ficheros que son los encargados de gestionar las muestras de audio, hablar con el programa y, en definitiva, hacernoslo más fácil. Todo esto se programa en Visual C++ y toda la estructura de ficheros ya está completa. Sólo tendremos que implementar el algorismo propiamente dicho y, si es el caso, las ventanas gráficas de nuestro Plug-in.

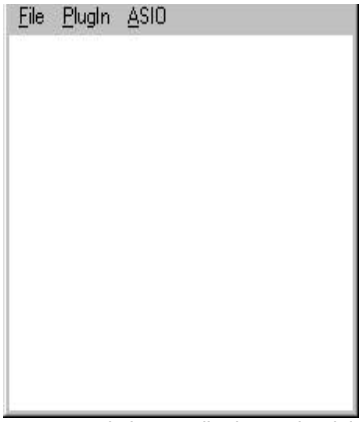


Fig. 6: Aspecto de la pantalla de prueba del Plug-in

Fig. 7: aspecto del Plug-in dentro el programa Cubase VST

La primera ventana pertenece al programa VplgTst, que es el que se facilita en internet, mientras que la segunda ventana es la del propio programa Cubase VST, en la que se pueden modificar los distintos parámetros (ganáncia y umbral para cada nivel, en el caso del DeNoiser)

4. RESULTADOS

Los resultados de la implementación del Trémolo y Vibrato han sido satisfactorios, puesto que se trata de efectos relativamente sencillos. En el caso del Reductor de ruido es muy difícil cuantificar la relación S/N conseguida puesto que depende de muchos parámetros (ganáncia y umbral para cada nivel, tipo de Wavelet utilizada, etc). Sin embargo, estamos hablando de unas reducciones de ruido de unos 10 dB, aunque también estamos modificando, involuntariamente, el espectro de la señal original. Evidentemente, para el futuro queda una implementación con más tipos de filtros distintos, diferentes algorismos para encontrar un umbral y una infinidad de detalles que permitan un plug-in digno de estar en cualquier estudio de grabación.

También queda pendiente la implementación de todos estos algorismos en un DSP, especialmente el "Industry standard chorus effect circuit with feedback" y el DeNoiser.

5. BIBLIOGRAFIA

- THE COMPUTER MUSIC TUTORIAL, Curtis Roads, Ed. The MIT Press 1998, 3rd. Printing
- DSP MUSIC TOOLBOX, PART 2: EFFECT DESIGN, DELAY-LINE MODULATION AND CHORUS. Jon Dattorro, J. Audio Eng. Soc., Vol. 45, No. 10, 1997 October
- PRINCIPLES OF DIGITAL AND ANALOG COMMUNICATIONS, Jerry D. Gibson. Ed. Prentice Hall 1993, 2nd edition
- FUNDAMENTOS DE ACUSTICA, Lawrence E. Kinsler, Ed. Limusa 1995
- THE SCIENCE OF SOUND, Thomas D. Rossing
- VISUAL C++ TUTORIALS, 1995 Microsoft Corporation