Organización del Computador II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico Final: PedalerIA64

Informe

Integrante	LU	Correo electrónico
Laporte Matías	686/09	matiaslaporte@gmail.com

1. Introducción

El presente Trabajo Práctico consiste en el desarrollo de diversos efectos de audio que se pueden encontrar en softwares de edición en el contexto musical (Cubase, Reaktor, Audacity, etc.), así como también en pedaleras (de allí el nombre del programa) para el caso específico de una guitarra.

1.1. Proceso de desarrollo del TP

El proceso de desarrollo del **TP** consistió en realizar en primera instancia un rápido prototipado en Matlab/RStudio/Scilab (dependiendo de la disponibilidad de la computadora que se usara en ese momento; de ahora en más, me referiré a esos programas como **MRS**) de algoritmos y ecuaciones que se pudieran encontrar en Internet de diversos efectos. El uso de esos programas como primer acercamiento a un efecto facilitaba enormemente el trabajo, pues al utilizar un lenguaje de muy alto nivel se simplificaba el manejo de la lectura y escritura de los archivos, el espacio en memoria para los mismos, la manipulación de los datos (poder trabajar trabajar sobre un vector entero con una sola operación, por ejemplo), etc.

Una vez certificado que con ese algoritmo se consiguiera el efecto auditivo deseado, se pasó a desarrollarlo en **C**, siempre verificando que el resultado final de la señal coincidiera con el obtenido en **MRS**. Como utilizar el comando *diff* de UNIX directamente sobre los archivos no siempre daba el resultado querido (aventuro a decir que por diferencias de aproximación entre **MRS** y **C**), se optó por otra metodología utilizando el programa **Audacity**, que se explicará en la sección 1.3.2.

Obtenido el resultado deseado utilizando **C**, se pasó finalmente a programar el mismo algoritmo en **Assembler**, haciendo uso de las instrucciones que ofrece el conjunto de instrucciones SSE para manejar múltiples datos con una única instrucción (SIMD). Se utilizaron instrucciones incluidas hasta la extensión SSE4 (principalmente por *PTEST*, y *ROUNDPS/ROUNDSS*). Luego de obtener en **Assembler** un código que parecía aceptable (es decir, que no terminara abruptamente con un segfault), se comenzó con un proceso iterativo de corrección, mediante la comparación del archivo de audio obtenido con el de **C**, utilizando **Audacity** como se mencionó previamente. Las diferencias entre ambos muchas veces provenían de casos bordes, que se los mencionará en la sección **??**.

El objeto de la programación en **C** además de Assembler, si bien implica el "doble" de trabajo, se debe a dos puntos en particular. En primer medida, era una buena manera de poder bajar el nivel del código original en **MRS**, despojándolo de las bondades que dichas herramientas ofrecen. Por otro lado, tener el código en **C** sirve también para comparar la mejora de rendimiento con el uso de las instrucciones SIMD.

Para calcular el rendimiento en ambos lenguajes se utilizó la librería *Tiempo.h* utilizada por la cátedra en el **TP N°2** del 1er. Cuatrimestre de 2011. Cuando los resultados con la misma no eran satisfactorios (p.ej., **Assembler** era más lento que **C**), se procedió a utilizar una herramienta de profiling (**callgrind** en conjunto con **KCacheGrind**, se hablará de ellas más adelante) para poder identificar dónde exactamente estaban las secciones lentas del código en **Assembler**, y poder tomar medidas para resolverlo. De los casos puntuales donde se utilizó esto se hablará en las secciones **??**, **??** y **??**.

Se desarrolló también una rudimentaria interfaz gráfica para tratar de evitar la utilización de la línea de comandos (en particular por la cantidad de argumentos que hay que manipular para los diversos efectos) y que la utilización del programa sea más amigable. Se hablará de ella en la sección **??**.

1.2. Audio

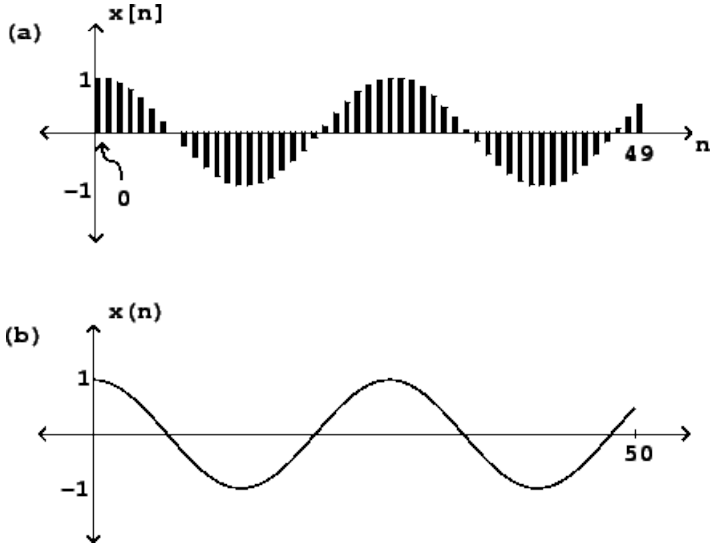

Figura 1: Señales de audio

Simplificándolo extremamente, una señal de audio (**(b)** en la imagen 1) es una representación del sonido, que puede ser visualizado como una curva continua (en el caso analógico, sus valores representan voltaje eléctrico) en función del tiempo. Al digitalizar una señal, se discretiza la curva (**(a)** en la imagen 1) tomando valores cada cierta cantidad de tiempo, lo que da lugar a la *frecuencia de muestreo* (**sampling rate**, expresada como cantidad de muestras por segundo, unidad **Hz**). A la vez, cada uno de esos valores no puede ser expresado con precisión infinita, sino que al pasar al dominio digital, debe poder ser representado con una cantidad específica de bits, lo que origina la *tasa de bits* (**bit rate**, *resolución* de una señal de audio).

El formato de audio elegido para el **TP** es WAV, por ser uno de los más simples para manipular. Los datos correspondientes al audio no están comprimidos, por lo que es posible realizar directamente sobre ellos las operaciones necesarias para aplicar los diversos efectos.

La librería utilizada para el manejo de este archivo se verá en la sección 1.3.1.

Por recomendación del profesor durante la presentación del proyecto de **TP**, en el archivo de audio final obtenido luego de la aplicación de alguno de los efectos, la *señal seca* (*dry sound/dry signal*, sin efecto) va por un canal, y la *señal húmeda* (*wet sound/wet signal*) por el otro. De este modo, se puede apreciar con mayor claridad el efecto en cuestión. Esto implica que todos los archivos de salida tendrán dos canales (**stereo**), a pesar de que el archivo de entrada pudiera haber tenido un único canal. En el caso de archivos de entrada con dos canales, se realiza un promedio de ambos (aunque esto sólo es válido en archivos stereo que mandan el mismo audio por los canales), y sobre ese nuevo "canal" se aplica el efecto correspondiente.

1.3. Herramientas externas utilizadas

Además de los ya mencionados **Matlab**, **RStudio**, y **Scilab**, se utilizaron las siguientas herramientas desarrolladas por terceros.

1.3.1. libsndfile

libsndfile es una librería de código abierto desarrollada en **C** para leer y escribir archivos de audio. Trabaja con el formato WAV, entre otros, por lo que se adaptaba a las necesidades del **TP**. La API puede consultarse aquí [1].

Una ventaja de la librería es que hace un pasaje de integer (tipo de datos utilizado en el formato WAV) a float, que es el tipo de datos utilizado para el **TP** [2]. Por otro lado, al leer un archivo utiliza una estructura propia llamada SF_INFO, que incluye datos importantes del mismo (cantidad de canales, de muestras, entre otros), y que son necesarios en algunas porciones del **TP** por diversos motivos.

1.3.2. Audacity

Audacity es un editor multiplataforma de audio digital de código abierto y que en el **TP** se utilizó para realizar las comparaciones entre los archivos finales obtenidos de los diferentes efectos para cada lenguaje. De este modo, al no encontrar diferencias entre las señales de dos archivos diferentes, se podía corroborar que el algoritmo coincide con el efecto a aplicar.

Chequeo diferencias

Para poder verificar si hay diferencias entre dos archivos de audio en Audacity, es necesario proceder del siguiente modo. Con el programa abierto, se arrastran los dos archivos hacia la ventana del mismo para que sean importados automáticamente (si es la primera vez, se va a preguntar si se quiere trabajar sobre los mismos archivos, o sobre una copia temporal de los mismos; por seguridad, se recomienda elegir esta última opción, y que el programa la recuerde).

Como ambos archivos son "de salida" (para nuestro programa)

[1] `http://www.mega-nerd.com/libsndfile/api.html`

[2] Originalmente se pensaba utilizar double, pero por recomendación del profesor se decidió pasar a float para poder procesar más datos

2. Bibliografía tentativa

2.1. Libros

Referencias

[1] Richard Boulanger, Victor Lazzarini *The Audio Programming Book*, 2011, The MIT Press, Massachussets (USA)

[2] F. Richard Moore, *Elements of Computer Music*, 1990, Prentice Hall, New Jersey (USA)

[3] Sophocles J. Orfanidis, *Introduction to Signal Processing*, `http://www.ece.rutgers.edu/~orfanidi/intro2sp/`

[4] Curtis Roads, *The Computer Music Tutorial*, 1996, The MIT Press, Massachussets (USA)

[5] Davide Rocchesso, *Introduction to Sound Processing*, `profs.sci.univr.it/~rocchess/SP/sp.pdf`

[6] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, Second Edition, 1999, California Technical Publishing, California (USA)

[7] Udo Zölzer, *DAFX: Digital Audio Effects* Second Edition, 2011, Wiley and Sons, Hamburg (Germany)

2.2. Internet

- `http://www.mega-nerd.com/libsndfile/api.html`
- `http://stackoverflow.com`
- `https://ccrma.stanford.edu/~jos/`
- `http://www.musicdsp.org/`
- `http://www.kvraudio.org/`