# 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 **??**.

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

Simplificándolo extremamente, una señal de audio 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 tomando valores cada cierta cantidad de tiempo, lo que da lugar a la *frecuencia de muestreo* (**sampling rate**). 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**).

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.

## 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].

# 2. Objetivos

La idea del TP es implementar una serie de algoritmos que produzcan efectos sobre audio, en los lenguajes *C* y *ASM* (mediante el uso de instrucciones *SSE*), para proceder a la comparación del

[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

rendimiento de ambos (mediante la metodología utilizada -contar ticks del procesador- en el TP2 de la materia, 1er. cuatrimestre de 2011). El TP consistirá en lo siguiente:

- Base teórica: conceptos de procesamiento de señales y de audio en particular, herramientas utilizadas (librerías, lenguajes).
- Utilización del programa: requerimientos a instalar para el programa, ejemplos de cómo correrlo
- Desarrollo: explicación de cada uno de los efectos implementados con pseudocódigo y prosa, cuál es el efecto que generan.
- Conclusiones: resultados obtenidos, comparación con lo esperado, dificultades encontradas a la hora de desarrollar el TP, etc.

Todavía no se encuentran definidas todos los efectos que se implementarán. Por lo pronto, ya se realizaron (visibles en el código adjunto) una función de copiado del archivo (sin efecto), y otra de delay simple sin feedback, ambas muy básicas. Se esperan desarrollar efectos que involucren FTT [3] (Fast Fourier Transform) y Convolución [4] (que utiliza la FTT); ambos, procesos fundacionales en procesamiento de audio.

Una prueba que me gustaría hacer para la entrega final es ver si lo desarrollado se puede aplicar en audio en tiempo real, algo que todavía no fue experimentado (principalmente porque el manejo de audio en tiempo real varía mucho según la instalación de Linux, la placa de audio, el soft que use el OS para Audio (ALSA, OSS, etc.), etc.).

# 3. Uso del código adjunto

Como se dijo previamente, como ejemplo se adjuntó el código base para empezar a desarrollar el TP entero, con dos funciones, una de copiado, y otra de delay simple sin feedback, ambas implementadas tanto en *C* como *ASM*. Para probarlo, se necesita instalar los paquetes **libsndfile1-dev** y **nasm** (sudo apt-get install libsndfile1-dev nasm), que se encuentra en los repositorios de Ubuntu. El paquete **libsndfile1-dev** es la librería que se utiliza para el manejo (lectura/escritura) de los archivos de audio. Se utilizará el formato WAV, pues no utiliza compresión y facilita el procesamiento de los datos [5].

Una vez instalado el paquete, compilar el programa con el makefile (comando: *make*). Ejemplos de uso del programa:

**Ver opciones**: *./main*
**Copiado de archivo en C**: *./main archivo_entrada.wav archivo_salida.wav -c*
**Copiado de archivo en ASM**: *./main archivo_entrada.wav archivo_salida.wav -C*
**Delay de 1.5 segundos con 0.6 de decay en C**: *./main archivo_entrada.wav archivo_salida.wav -d 1.5 0.6*
**Delay de 1.5 segundos con 0.6 de decay en ASM**: *./main archivo_entrada.wav archivo_salida.wav*

[3] `http://www.dspguide.com/ch12/2.htm`
[4] `http://en.wikipedia.org/wiki/Convolution$_$reverb`
[5] `http://en.wikipedia.org/wiki/WAV_file`

*-D 1.5 0.6*

Se incluyen archivos de Audio de ejemplo en la carpeta *inputExamples*.

# 4. Bibliografía tentativa

## 4.1. 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)

## 4.2. 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/`