RODRIGO F. CÁDIZ

INTRODUCCIÓN A LA MÚSICA COMPUTACIONAL

Capítulo 0

Prólogo

La era computacional y la música estuvieron por mucho tiempo distanciadas debido a la gran cantidad de espacio de memoria que la música requiere para ser almacenada y procesada. Con el aumento de las capacidades de memoria de los computadores, la música comenzó rápidamente un desarrollo vertiginoso transformándose, hoy en día, junto al mundo audiovisual en las actividades predilectas en el uso del computador.

En efecto, en las últimas décadas son múltiples los softwares, las herramientas y las investigaciones que giran en torno a la actividad musical tanto en los terrenos del registro sonoro, como en el ámbito creativo y en el área de la reproducción. Tanto profesionales como aficionados recurren al computador para manipular información musical pero siempre con la sensación de que aprovechan sólo un mínimo porcentaje de las capacidades de los softwares o herramientas disponibles.

El libro que tiene en sus manos constituye el más importante aporte en lengua española para acercar a los interesados a la tecnología digital disponible. Desde la terminología básica, pasando por los principios acústicos del sonido hasta la psicoacústica, desde la noción de audio digital, su procesamiento hasta su aplicación en la música electroacústica constituyen el contenido de esta publicación dirigida a músicos, ingenieros, físicos y amantes de la tecnología aplicada a la música.

El sonido es la materia prima, dispuesta en el tiempo, de la música. Por lo tanto, su estudio y conocimiento por parte todo aquel que se interesa en ella, es fundamental. La dificultad de conseguir literatura en torno al sonido y sobre todo en el ámbito digital hacía difícil el acceso a la información necesaria, general y preferentemente en inglés, para todos los usuarios de tecnología aplicada al mundo musical.

Con este libro, el mundo técnico y conceptual del sonido digital queda al alcance de todos de una manera ordenada y sistemática cubriendo en detalle cada uno de los diversos aspectos tratados sin perder de vista, en ningún momento, las nociones básicas que permiten la comprensión profunda de los fenómenos y posibilidades de la era digital.

Su autor, Rodrigo Cádiz, en su doble condición de ingeniero y compositor ha puesto a nuestra disposición una información amplia, de enfoques múltiples que recorren lo técnico esencial hasta lo artístico creativo. De este modo, esta publicación se transforma en lectura obligada para músicos profesionales o no, interesados en la vinculación de la tecnología actual y la más antigua y misteriosa de las artes: la música. Inefable, inescrutable, inenarrable, mágica y misteriosa arte que, finalmente ha sido "atrapada", registrada y reproducida con la mayor de las fidelidades por las ciencias y tecnologías del sonido digital.

Aceptemos esta invitación a conocer este nuevo mundo de la mano de un autor atento y multidisciplinario que nos permite, en nuestra propia lengua, acercarnos y resolver nuestras inquietudes en torno a la música y la computación.

Alejandro Guarello
Santiago de Chile, Junio de 2008

Capítulo 0

Prefacio

Los computadores sin duda han cambiado nuestra forma de relacionarnos con el mundo externo. Específicamente en el ámbito de la música, los computadores permiten explorar nuevas posibilidades sonoras, crear nuevas formas de composición y nuevas aproximaciones a la interpretación musical. Las posibilidades musicales que nos ofrece la tecnología computacional son prácticamente infinitas.

El propósito de este libro es introducir a estudiantes de música, física o ingeniería, artistas, intérpretes y compositores al extenso mundo de la música computacional y electroacústica. Originalmente, este libro está pensado como un texto de apoyo para un curso básico de música computacional de uno o dos semestres de duración, pero perfectamente puede utilizarse fuera de un curso formal.

El libro está divido en siete capítulos, cada uno de los cuales se enfoca a un tema específico. El primer capítulo expone los conceptos básicos de la teoría de señales y sistemas, incluyendo conceptos claves como la representación en frecuencia y la transformada de Fourier, además de introducir algunos conceptos matemáticos imposibles de evitar al adentrarse en este intenso mundo. Dado que un computador permite diseñar sonidos con un máximo nivel de detalle y precisión, es muy relevante para esta disciplina el conocer en profundidad el fenómeno sonoro. No sólo es importante estudiar como se comporta el sonido en el mundo físico, sino también como es percibido en la cabeza de cada auditor. Esto se divide en dos capítulos: el segundo capítulo estudia el fenómeno del sonido en su aspecto físico y el tercero desde el punto de vista perceptual.

El computador es por naturaleza digital, lo que hace necesario conocer las propiedades de las señales de audio digitales que el computador es capaz de

generar y manipular. Esto se aborda en el capítulo cuarto, en conjunto con el estudio del computador y su uso en sistemas de audio. El quinto capítulo se enfoca en la síntesis digital de sonidos, exponiendo sus fundamentos, formas de clasificación y los principales métodos de síntesis utilizados. En el sexto capítulo se estudia el procesamiento digital de audio, con un especial énfasis en la teoría e implementación de los filtros digitales.

Para finalizar, se incluye un capítulo, el séptimo y último, en el cual se presentan las principales bases perceptuales, composicionales y analíticas de la llamada música electroacústica, la cual en la actualidad se realiza en su gran mayoría gracias a la tecnología computacional.

Se incluyen también varios anexos con ejemplos de código realizados en SuperCollider, una de las aplicaciones para síntesis y procesamiento digital de sonidos más utilizadas, poderosas y versátiles disponibles en la actualidad. Estos ejemplos complementan las materias presentadas en los distintos capítulos y permiten llevar a la práctica de forma inmediata mucho de los conceptos presentados en el libro.

Este libro tuvo sus orígenes en una serie de apuntes parcialmente financiados por la Fundación Andes, a través de su Beca de Creación e Investigación Artística, en el año 2000. La actual versión ha sido posible gracias al apoyo del Fondo de Desarrollo de la Docencia (FONDEDOC), de la Vicerrectoría Académica de la Pontificia Universidad Católica de Chile. Quisiera agraceder el aporte de mi ayudante Jorge Forero, quién realizó muchas de las tablas y figuras presentes en el libro y del Dr. Gary Kendall, quien fuera mi profesor guía y mentor en muchas de las materias tratadas en este libro, y en cuyos apuntes se basan muchos de las figuras y ejemplos de código SuperCollider que se pueden encontrar en el libro. También quisiera agradecer a Alejandro Guarello, por su apoyo constante en este proyecto.

Rodrigo F. Cádiz, Ph.D.
Santiago de Chile, Abril de 2008

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Señales y sistemas

El sonido es fundamentalmente una *señal*, que conlleva información de tipo acústica. Todos los tipos de señales, sean éstas naturales o artificiales, tienen características comunes que hacen posible su estudio en forma independiente de su naturaleza. Por lo general, las señales son procesadas o modificadas por *sistemas*. En el caso de la música computacional, el computador genera o modifica una señal acústica digitalizada, por lo tanto se comporta como un sistema. En este capítulo se presentan los conceptos básicos de la teoría de señales y sistemas, los cuales resultan fundamentales para entender y estudiar como una señal de audio puede ser modificada o creada en el computador de las más diversas maneras, con el objetivo final de generar música.

1.1. Señales

Una señal, en forma simplificada, se puede entender como cualquier mecanismo que es empleado para transmitir información [15]. Algunos ejemplos de señales son: un faro, ondas electromagnéticas enviadas por un radar, señales de humo, una onda de sonido viajando por el aire o las ondas de la actividad del cerebro captadas por un electrocardiograma.

Desde un punto de vista matemático, una señal es una variable de una o más dimensiones que toma valores de acuerdo a otra variable, como por ejemplo el tiempo en el caso del sonido o el espacio en el caso de imágenes. Matemáticamente, da exactamente lo mismo si la dependencia es temporal o espacial, ya que en ambos casos las señales de tratan de igual manera y sólo importa la función que modela su comportamiento.

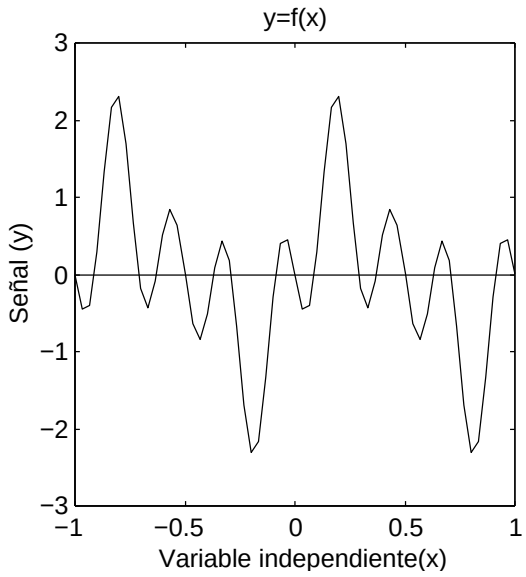


Figura 1.1: Ejemplo gráfico de una señal

En el caso de las señales acústicas, es común utilizar la letra t para designar a la *variable independiente*, que corresponde al tiempo, y la letra y para designar a la señal, o *variable dependiente*. La relación que define el comportamiento de la señal con respecto a la variable independiente corresponde a una función que usualmente se denomina por la letra f. Por lo tanto la ecuación:

$$y = f(t) \tag{1.1}$$

simplemente significa que y *depende* del tiempo de acuerdo a la función f. La figura 1.1 muestra una representación gráfica de una señal en función de dos variables, y y t. La forma o morfología de la función f es lo que determina la información contenida en la señal.

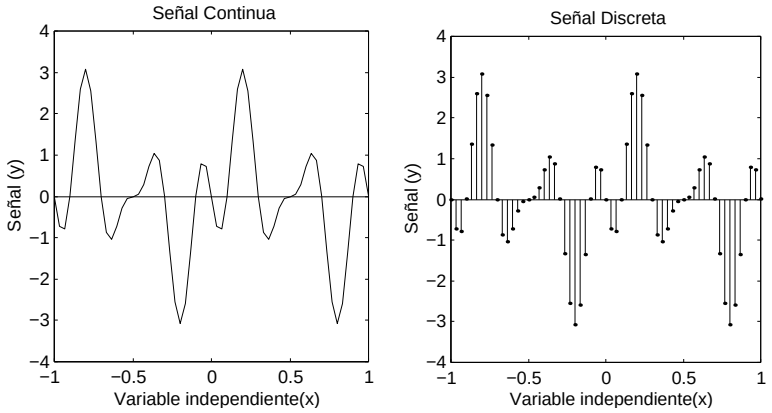
1.1.1. Naturaleza de las señales

Las señales son de distinta naturaleza. Una señal puede ser, por ejemplo, una onda de presión viajando por el aire producida por la voz de un cantante en algún escenario cercano. En este caso claramente la señal es de tipo físico, es decir involucra átomos, moléculas y otras partículas físicas. Sin embargo,

una señal puede no tener relación con algún fenómeno físico. Un ejemplo de esto es el valor del índice de acciones de la bolsa de comercio, que cambia todos los días, pero sólo corresponde a un valor numérico, sin relación con el mundo físico.

Si una señal toma valores en forma continua, es decir, si para todos los valores de la variable independiente existe un valor para la variable independiente, se habla de que la señal es *continua*. Un ejemplo de esto es la señal de temperatura que entrega un termómetro en el tiempo. El termómetro siempre está marcando alguna temperatura y cualquier persona puede leer la posición de la barra de mercurio ahora, o en una milésima de segundo después o en un año más y el termómetro siempre entregará un valor a menos que esté roto.

Por el contrario, si una señal toma valores sólo para algunos valores de la variable independiente, se habla de una señal *discreta*. Por ejemplo, la luz roja de un semáforo sólo se enciende durante algunos instantes de tiempo en forma cíclica.



(a) Señal continua (b) Señal discreta

Figura 1.2: Ejemplos de señales continuas y discretas

Un computador no funciona en forma continua sino en intervalos regulares de tiempo y por ende, sólo maneja en forma interna señales digitales. Para *digitalizar* una señal es necesario discretizarla. Es decir, tomar muestras a intervalos regulares de la señal analógica original y guardar estas muestras. Este proceso se conoce como, *digitalización*, *muestreo* o *sampleo* y será abordado en profundidad en el capítulo 4. Se puede adelantar aquí que

la calidad de la señal discreta depende de la frecuencia a la cual se muestrea la señal original.

La figura 1.2 muestra dos señales: una continua y otra discreta. Se puede observar allí que ambas tienen la misma forma, en el sentido que tienen la misma envolvente. En efecto, la señal de la derecha corresponde a la señal de la izquierda discretizada. La única información que esta señal contiene es el valor de la amplitud para cada una de las muestrasen esos instantes de tiempo. En una primera observación da la impresión que la señal muestreada pierde información respecto de la señal original ya que no almacena ningún tipo de información entre las muestras. Sin embargo, esto no ocurre si es que la frecuencia de muestreo es lo suficientemente alta para asegurar la calidad del proceso de digitalización. Esto se explica en detalle en la sección 4.1.1.

También las señales pueden variar de acuerdo a su periodicidad. En la figura 1.3 se aprecian dos formas de onda distintas. En la primera de ellas, se aprecia que la onda está compuesta de patrones repetitivos, mientras que la segunda posee una forma que parece ser aleatoria, sin un comportamiento definido.

En el caso de la primera forma de onda, se dice que corresponde a una *onda periódica*, es decir, que se repite cada cierto tiempo. En el caso de la segunda, se habla de una forma de onda *aperiódica* , pues no sigue un patrón determinado de repetición.

Una señal periódica $f(x)$ es una que cumple la siguiente relación:

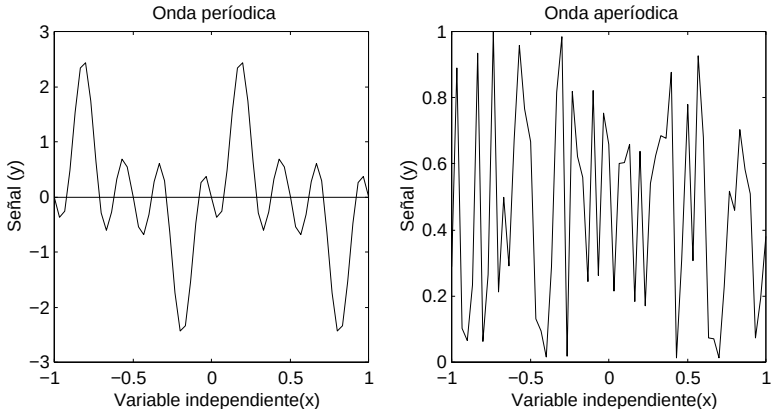
$$f(x) = f(x + T) \tag{1.2}$$

donde T se conoce como el *período* de la señal.

En cambio, para una señal aperiódica, no existe ninguna variable T que cumpla la relación anterior.

En el caso de un onda periódica, el patrón que se repite corresponde a un *ciclo* . La duración de cada uno de los ciclos de una onda se conoce como período y corresponde a la variable T de la ecuación 1.2. La tasa a la cual los ciclos de una onda periódica se repiten se conoce como *frecuencia* y por lo general se mide en ciclos por segundo o Hertz (Hz) si la variable independiente corresponde al tiempo. Matemáticamente, la frecuencia es el inverso del período, por lo tanto un período de 1 ms (milisegundos) tiene una frecuencia de 1000 Hz.

En las señales aperiódicas no se presentan patrones de repetición, dado que nada se repite en forma periódica. La ausencia de una o más frecuencias predominantes hace que este tipo de ondas sean muy complejas y difíciles de modelar. Generalmente, este tipo de señales corresponden a patrones



(a) Onda periódica (b) Onda aperiódica

Figura 1.3: Ejemplos de ondas periódica y aperiódica

aleatorios o simplemente a ruido.

1.1.2. Sinusoides

La sinusoide es una de las señales más simples e importantes que existen. Las dos funciones matemáticas sinusoidales básicas son las funciones *seno* y *coseno*, las cuales están derivadas de las funciones trigonométricas del mismo nombre.

Matemáticamente una función de este tipo se puede escribir como:

$$f(x) = A\sin(wx + \phi) \tag{1.3}$$

o como

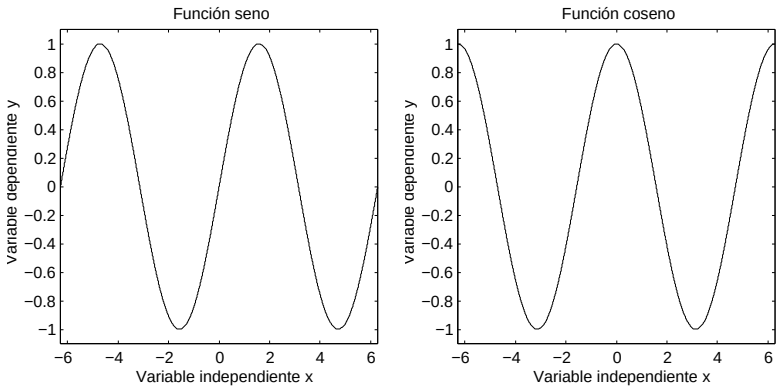
$$f(x) = A\cos(wx + \phi) \tag{1.4}$$

La amplitud A corresponde a la desviación máxima de la señal respecto del origen (posición de equilibrio). La frecuencia w corresponde a la cantidad de ciclos que existen en un determinado rango de la variable x. La fase, denotada por la letra ϕ, corresponde a la desviación o corrimiento de la señal respecto del eje Y.

Una señal simple, de tipo sinusoidal, está completamente determinada por estos tres parámetros: amplitud, frecuencia y fase.

Ambas funciones, seno y coseno, son cíclicas o periódicas, es decir, vuelven a tomar los mismos valores después de un cierto rango de valores de la variable independiente.

Por ejemplo, la función seno es una función que vale 0 cuando la variable independiente vale 0, 1 cuando ésta vale $\pi/2$, 0 cuando pasa por π, -1 cuando cruza en $3\pi/2$ y nuevamente cero cuando la variable independiente vale 2π.



(a) Función seno (b) Función coseno

Figura 1.4: Funciones seno y coseno

La función coseno es muy similar a la función seno, sólo que está "desfasada.en relación al eje Y. En relación al desfase, Las siguientes relaciones matemáticas se cumplen entre estas funciones:

$$\cos(x) = \sin(\frac{\pi}{2} - x) \tag{1.5}$$

$$\sin(x) = \cos(\frac{\pi}{2} - x) \tag{1.6}$$

La función seno es un función impar, es decir, no es simétrica respecto al eje Y. Matemáticamente esto es:

$$\sin(-x) = -\sin(x) \tag{1.7}$$

En cambio el coseno es una función par, porque es simétrica respecto al eje Y. Matemáticamente, esto equivale a:

$$\cos(-x) = \cos(x) \tag{1.8}$$

Las sinusoides son fundamentales para la física en general y la acústica en particular. Cualquier cosa que resuene u oscile produce un movimiento de tipo sinusoidal. Un ejemplo de esto es la oscilación de un péndulo, conocida como *movimiento armónico simple*.

Otra razón de la importancia de las sinusoides es que constituyen funciones básicas de los sistemas lineales. Esto significa que cualquier sistema lineal puede ser estudiado a través de su respuesta a funciones sinusoidales. En el campo del audio, las sinusoides son importantes para el análisis de filtros tales como reverberadores, equalizadores y otros tipos de efectos.

Desde un punto de vista matemático, las sinusoides constituyen bloques fundamentales que al ser combinados de cierta forma permiten la creación o síntesis de cualquier tipo de señal, por muy compleja que sea.

Pero quizás la razón más importante es que el sistema auditivo humano funciona como un *analizador de espectro*, tal como se detalla en el capítulo 3. Esto es, el oído humano físicamente separa un sonido en sus componentes de frecuencia sinusoidales. Por lo tanto, el oído humano funciona en forma muy similar a un analizador de Fourier (ver sección 1.2) y la representación de un sonido en el dominio de la frecuencia (ver sección 1.2.4) es mucho más cercana a lo que nuestro cerebro recibe que la representación temporal.

1.1.3. Exponenciales

Existe otra función matemática muy importante, conocida como exponencial. La forma canónica de una función exponencial es la siguiente:

$$f(t) = Ae^{-t/\tau}, t \geq 0 \tag{1.9}$$

A corresponde a la amplitud máxima de la exponencial y τ se conoce como la *constante de tiempo* de la exponencial. La constante de tiempo es el tiempo que demora la exponencial en decaer $1/e$, es decir:

$$\frac{f(\tau)}{f(0)} = \frac{1}{e} \tag{1.10}$$

La figura 1.5 muestra el gráfico de una función exponencial para $A = 1$ y $\tau = 1$. En la ecuación 1.9 e es el *número de Euler*, el cual tiene el valor irracional 2.718... y constituye la base de los logaritmos naturales. Este número es uno de los más importantes y fascinantes de la matemática y puede calcularse como:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = \lim_{n \to 0}(1 + \frac{1}{n})^n \approx 2{,}718... \tag{1.11}$$

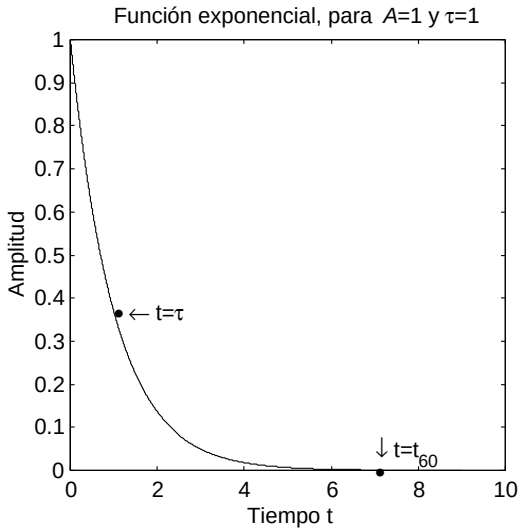


Figura 1.5: Función exponencial

El decaimiento exponencial ocurre naturalmente cuando una cantidad está decayendo a una tasa proporcional a lo que aún queda por caer. En la naturaleza, todos los resonadores lineales, tales como las cuerdas de los instrumentos musicales y los instrumentos de vientos exhiben un decaimiento exponencial en su respuesta a una excitación momentánea. La energía reverberante en una sala de conciertos decae exponencialmente una vez finalizada la emisión de sonido. Esencialmente todas las oscilaciones libres (sin una fuente mantenida en el tiempo) caen exponencialmente siempre que sean lineales e invariantes en el tiempo. Ejemplos de este tipo de oscilaciones incluyen la vibración de un diapasón, cuerdas pulsadas o pellizcadas y barras de marimba o xilófonos.

El crecimiento exponencial ocurre cuando una cantidad crece a una tasa proporcional al incremento actual. El crecimiento exponencial es inestable dado que nada puede crecer para siempre sin llegar a un cierto nivel límite.

Es necesario notar que en la ecuación 1.9, una constante de tiempo positiva implica a un decaimiento exponencial mientras que una constante de tiempo negativa corresponde a un crecimiento exponencial.

En sistemas de audio, una decaimiento de $1/e$ se considera muy pequeño para aplicaciones prácticas, como por ejemplo, para el diseño acústico de salas de concierto. Por lo general, se utiliza una cantidad tal que asegure que la señal ha caído 60 decibelios (`dB`). Esta cantidad, denotada por t_{60}, se

encuentra resolviendo la ecuación:

$$\frac{f(t_{60})}{f(0)} = 10^{-60/20} = 0{,}001 \tag{1.12}$$

Usando la definición de exponencial de la ecuación 1.9, se tiene entonces que:

$$t_{60} = \ln(1000)\tau \approx 6{,}91\tau \tag{1.13}$$

Esta ecuación nos dice que la constante t_{60} es aproximadamente 7 veces la constante de tiempo τ. Esto puede verificarse en la figura 1.5, donde se aprecia la ubicación de dicha cantidad en el eje del tiempo.

1.1.4. Relación entre sinusoides y exponenciales

Existen dos relaciones muy importantes para el análisis de señales y para el Teorema de Fourier, llamadas ecuaciones de Euler . Estas son:

$$e^{ix} = \cos(x) + i\sin(x) \tag{1.14}$$

$$e^{-ix} = \cos(x) - i\sin(x) \tag{1.15}$$

Esta ecuación nos dice que las exponenciales y las sinusoides están íntimamente relacionadas. En la ecuación 1.14, el número i representa al número *imaginario* y que está definido por la relación:

$$i = \sqrt{-1} \tag{1.16}$$

El número imaginario i tiene una importancia fundamental en el análisis de frecuencia de una señal, fundamentalmente porque, tal como se verá en forma siguiente, las sinusoides pueden representarse y manejarse en forma más compacta si se utilizan números complejos. Los números complejos están constituidos por un par ordenado de números, uno real y otro imaginario, y usualmente se grafican en lo que se denomina el plano complejo, mostrado en la figura 1.6, donde el eje de ordenadas representa los números reales y el eje de absisas los imaginarios. En este plano un número complejo es un vector que se puede representar de dos formas: cartesiana y polar. En la forma cartesiana, un número complejo Z se representa como la suma de su parte real con su parte imaginaria o bien $Z = x + iy$. Pero el mismo número se puede representar mediante el largo del vector y su ángulo, lo que se denomina forma polar. En este caso se tiene $Z = r\angle\Theta$, donde r es el

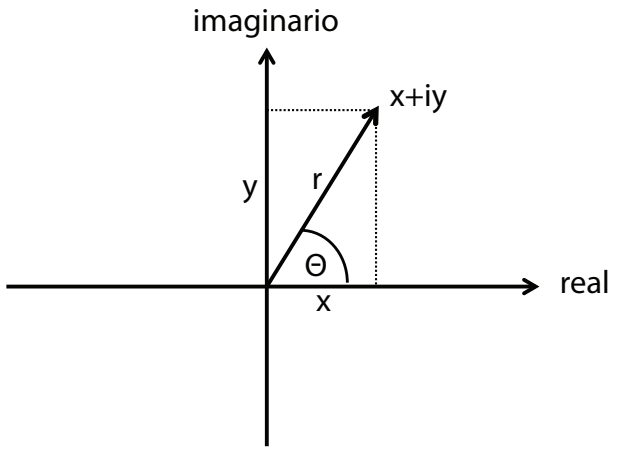


Figura 1.6: El plano complejo

módulo o magnitud del número complejo y que también suele representarse como $|Z|$.

Ambas representaciones están relacionadas por las siguientes ecuaciones:

$$r = \sqrt{x^2 + y^2} \tag{1.17}$$

y

$$\theta = \arctan\left(\frac{y}{x}\right) \tag{1.18}$$

La ecuaciones 1.14 y 1.15 pueden utilizarse para demostrar que:

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2} \tag{1.19}$$

y

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i} \tag{1.20}$$

Multiplicando la ecuación 1.14 por una amplitud $A \geq 0$ y utilizando $x = wt + \phi$ se tiene:

$$Ae^{i(wt+\phi)} = A\cos(wt + \phi) + iB\sin(wt + \phi) \tag{1.21}$$

Esta ecuación describe una *sinusoide compleja*. Por lo tanto, una sinusoide compleja contiene una parte real coseno y una parte imaginaria seno.

De acuerdo a las ecuaciones 1.19 y 1.20 y dado que e^{iwt} corresponde a una sinusoide compleja (ecuación 1.21), se tiene entonces que toda sinusoide real está compuesta por una contribución equitativa de frecuencias positivas y negativas. Dicho de otra forma, una sinusoide real consiste de una suma de dos sinusoides complejas, una de frecuencia positiva y la otra de frecuencia negativa.

Esto significa que el espectro de frecuencias de una sinusoide o de una función periódica compuesta por sinusoides es simétrico respecto del origen y contiene tanto frecuencias negativas como positivas.

Este hecho es de suma importancia para el análisis de señales y para lo que posteriormente se verá como *teorema del muestreo*, detallado en la sección 4.1.1.

1.2. Series y transformada de Fourier

1.2.1. Series de Fourier

En 1811, el matemático Jean Baptiste Fourier demostró que cualquier señal periódica *razonable* puede expresarse como la suma de una o más sinusoides de distinta frecuencia, fase y amplitud. Se entiende por una *señal razonable*, una que posee valores máximos menores que infinito y un número finito de saltos en un período.

Estas sinusoides están armónicamente relacionadas entre sí, es decir, sus frecuencias son múltiplos enteros de una frecuencia fundamental. Esta suma ponderada de señales sinusoidales se conoce como *Serie de Fourier*. La serie de Fourier de una señal periódica $f(x)$ de período T_0 puede escribirse como:

$$f(x) = a_0 + \sum_{k=1}^{\infty}(a_k \cos(kw_0x) + b_k \sin(kw_0x)) \tag{1.22}$$

donde w_0 es la frecuencia fundamental ($w_0 = 2\pi/T_0$) y los coeficientes a_0,a_k y b_k constituyen un conjunto de números asociados unívocamente con la función f(x). Esta forma de escribir la serie de Fourier se conoce como *forma trigonométrica*.

Utilizando las ecuaciones de Euler 1.14 y 1.15, se puede escribir la serie de Fourier en forma más compacta como:

$$f(x) = \sum_{k=-\infty}^{\infty} C_k e^{ikw_0x} \tag{1.23}$$

Esta forma de escribir la ecuación se conoce como *forma compleja*.

La ecuación 1.22 nos dice que para cualquier señal periódica $f(x)$ pueden encontrarse coeficientes $a_0, a_1, b_1, a_2, b_2, ...$ tales que multiplicados por sinusoides de frecuencias $w_0, 2w_0, 3w_0, ...$ den como resultado la función $f(x)$ cuando estas sinusoides se suman.

En palabras más simples, toda función periódica de frecuencia w, cualquiera sea su naturaleza, está compuesta por la suma de varias sinusoides de frecuencias mayores o iguales a w, cada una de las cuales tiene distinta amplitud, frecuencia y fase. Las frecuencias involucradas están armónicamente relacionadas entre sí.

Lo anterior implica dos cosas:

1. Si se toman varias sinusoides de distintas frecuencias, fases y amplitudes y se suman, se obtendrá una señal periódica.

2. Dada una señal periódica $f(x)$ cualquiera, ésta siempre podrá descomponerse en sus *componentes de frecuencia*, mediante la determinación de las sinusoides que la conforman. Si se grafican las frecuencias de estas sinusoides versus la amplitud de cada una de ellas, dicho gráfico se conoce como *espectro de frecuencias* (ver sección 1.2.4).

La figura 1.7 muestra a varias sinusoides de frecuencias relacionadas y su suma. Allí puede observarse que al sumar estas sinusoides se obtiene una nueva forma de onda de frecuencia igual a la frecuencia de la onda fundamental. Esto nos permite comprobar de manera empírica el descubrimiento de Fourier.

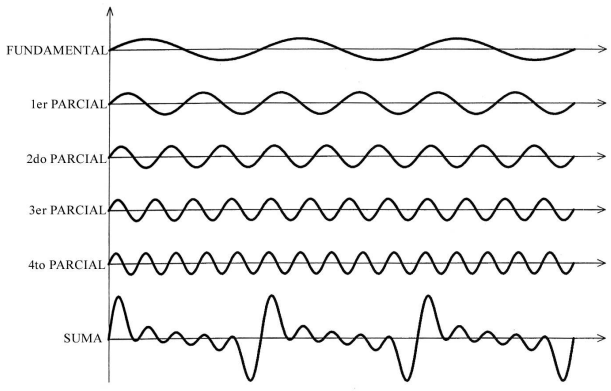


Figura 1.7: Suma de señales armónicas

La serie de Fourier se presenta en forma natural en la mayoría de los instrumentos musicales. Esto se conoce como la serie armónica y constituye el principio básico de ejecución de algunos instrumentos como la trompeta o tuba. En la figura 1.8 se muestra la serie armónica en notación musical. Cuando uno toca, por ejemplo, la nota Do en el piano

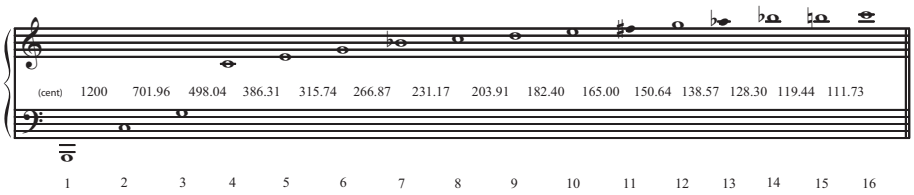


Figura 1.8: La serie armónica

1.2.2. Transformada de Fourier

Ahora cabe preguntarse ¿qué pasa con las funciones que no son periódicas?

Una forma de extender las series de Fourier a funciones no periódicas, es truncar las señales en un cierto punto y suponer que la zona truncada se repite hasta el infinito desde ahí hacia adelante. Otra forma es suponer que éstas poseen un período infinito y expandir un tanto las ecuaciones para poder trabajar con este tipo de señales. Esto da pie a la transformada de Fourier.

La *Transformada de Fourier* es una generalización de las Series de Fourier. En rigor, esta transformada se aplica a funciones continuas y aperiódicas, pero también es posible aplicarla a funciones discretas mediante la utilización de funciones impulso (ver sección 1.2.3).

Además, la transformada de Fourier es un subconjunto de la transformada de Laplace, de la cual provee una interpretación más simple. Estas relaciones hacen de la transformada de Fourier una herramienta clave para traducir señales desde los dominios del tiempo a la frecuencia y viceversa.

Matemáticamente, la transformada de Fourier se escribe:

$$F(w) = \int_{-\infty}^{\infty} f(x) e^{-iwx} dx \tag{1.24}$$

donde w denota la frecuencia y F a la transformada. Esta ecuación corresponde a *integrar* la función $f(x)$ multiplicada por una sinusoide compleja en todo el intervalo de la variable x (desde $-\infty$ hasta $+\infty$).

La transformada inversa, la que permite encontrar $f(x)$ cuando se conoce $F(w)$ está dada por la ecuación:

$$f(x) = \int_{-\infty}^{\infty} F(w)e^{iwx}dw \tag{1.25}$$

Utilizando estas relaciones, siempre es posible encontrar una correspondencia entre una función y su espectro o viceversa. Es decir:

$$f(x) \leftrightarrow F(w) \tag{1.26}$$

La transformada de Fourier nos permite conocer el espectro de frecuencias de cualquier señal $f(x)$, sea ésta periódica o aperiódica y cualquiera sea su naturaleza. Es decir, dada una función $f(x)$ cualquiera, mediante esta transformada podemos saber que frecuencias están presentes en ella.

1.2.3. DFT y FFT

La transformada de Fourier, tal como se presentó en las ecuaciones 1.24 y 1.25 permite encontrar transformadas para señales continuas. Sin embargo, esto no es aplicable directamente a señales discretas o digitales, que son las que nos interesan en este libro.

La Transformada de Fourier Discreta, o bien `DFT` (Discrete Fourier Transform), se emplea para encontrar el contenido de frecuencia de señales que son discretas. Esto implica que en el dominio de la frecuencia estas señales también serán periódicas y discretas. El desarrollo de la `DFT` históricamente se dio en forma paralela al de la transformada de Fourier continua. A pesar de su existencia, la `DFT` prácticamente no se utiliza dado que el cálculo de la transformada discreta es un proceso complejo y lento computacionalmente.

Lo que se utiliza en la mayoría de los casos para calcular espectros de señales discretas se llama *Transformada Rápida de Fourier*, o bien `FFT` (en inglés Fast Fourier Transform), la cual es un algoritmo desarrollado para obtener la `DFT` de una forma más rápida y eficiente computacionalmente. El tiempo de procesamiento de la `FFT` es considerablemente más rápido que calcular la `DFT` directamente.

1.2.4. Diagramas en la frecuencia

Dado que una señal comúnmente posee varias frecuencias aparte de la fundamental, también esta puede representarse completamente mediante un diagrama de frecuencias, comúnmente llamado espectro de frecuencias. En

la figura 1.9 puede apreciarse tanto la representación temporal como el espectro de las señales. Todos estos espectros fueron calculados utilizando la transformada rápida de Fourier o `FFT`.

Como es de esperarse, el espectro de la sinusoide simple, posee un sólo componente de 50 `Hz`. Es necesario notar que el espectro es simétrico respecto al origen y que contiene frecuencias negativas y positivas. El espectro de la suma de tres sinusoides, está caracterizado solamente por tres frecuencias distintas, que corresponden a las frecuencias de las tres sinusoides constituyentes de la señal.

En cambio, el espectro de la señal aleatoria es más bien plano y tiene componentes en prácticamente todas las frecuencias. Esto nos indica que mientras más componentes de frecuencia están presentes en una señal, su representación en el tiempo es más "compleja", no en el sentido de los números complejos sino en su contenido de información. Dicho de otro modo, para reproducir una señal de este tipo, de necesita sumar un número muy elevado de sinusoides.

1.3. Otras señales importantes

Existen otras señales de importancia en la teoría de sistemas, aparte de las sinusoides y exponenciales. Una función muy importante es el impulso, denotado por $\delta(x)$, la cual es una función que se define de la siguiente forma:

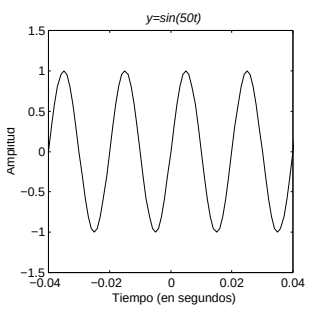
$$\delta(x) = \begin{cases} \infty & x = 0 \\ 0 & x \neq 0 \end{cases} \tag{1.27}$$

Es decir el impulso es una función que sólo adquiere un valor para $x = 0$ y vale para todos los otros valores de x. La importancia de esta función radica en que su transformada de Fourier es una función constante, es decir esta señal posee todas las frecuencias posibles. De manera intuitiva, esto puede entenderse si se piensa que el impulso representa el cambio más abrupto posible, y para generar una señal así se requieren de infinitas sinusoides de todas las frecuencias posibles.

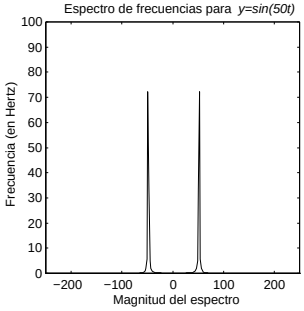
Otra función importante es la función rectangular o simplemente rect. Su importancia radica en que consituye un filtro pasa bajo ideal. Su transformada de Fourier es otra función importante y se denomina sinc.

La función rect se define como:

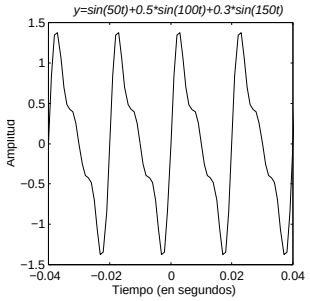
$$\text{rect}(x) = \begin{cases} 1 & \frac{-1}{2} \leq x \leq \frac{1}{2} \\ 0 & \text{para todo otro } x \end{cases} \tag{1.28}$$



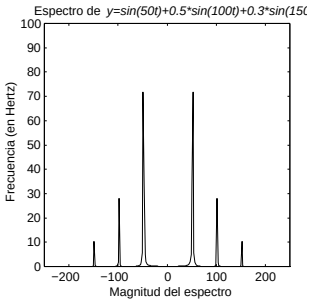
(a) Sinusoide simple en el tiempo



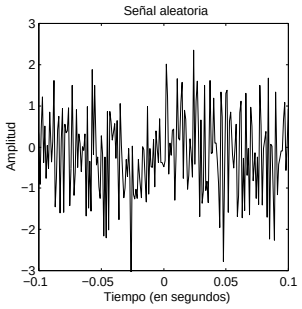
(b) Espectro de una sinusoide simple



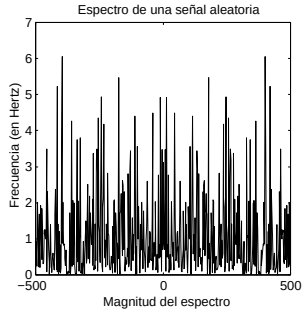
(c) Suma de tres sinusoides en el tiempo



(d) Espectro de una suma de tres sinusoides



(e) Señal aleatoria en el tiempo



(f) Espectro de una señal aleatoria

Figura 1.9: Representación en el tiempo y en la frecuencia de distintas señales

y la función sinc como:

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \tag{1.29}$$

Todas estas funciones se muestran en la figura 1.10.

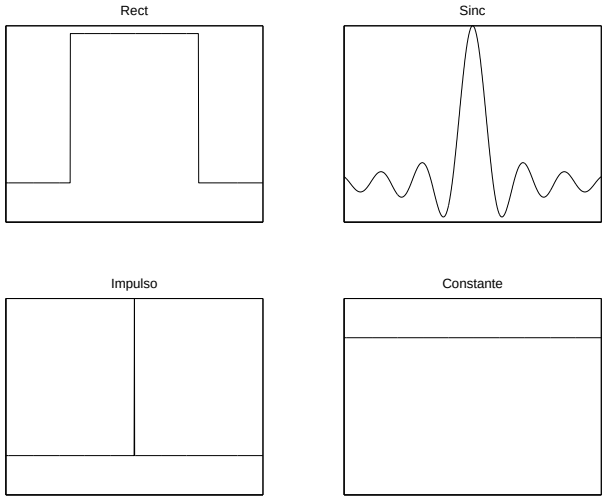


Figura 1.10: Otras señales importantes

La función Gaussiana también es de importancia, y es muy utilizada para construir ventanas. Su importancia radica en que es la función de distribución de probabilidades normal. Se define de la siguiente manera:

$$f(x) = Ae^{-x^2/(2\sigma^2)} \tag{1.30}$$

donde σ se conoce como la desviación estándar de la distribución.

1.4. Sistemas

Un *sistema* puede ser definido en forma general como una colección de objetos u operaciones para los cuales existen relaciones definidas de causa-efecto. Las causas o datos de entrada también son llamados *excitaciones* o simplemente, *entradas*. Los efectos o salidas del sistema también se conocen como *respuestas* o simplemente *salidas*.

Si se conocen los datos de entrada y de salida, es posible desarrollar un *modelo* que describa a cabalidad el sistema. Este modelo consiste en un conjunto de reglas tales que si son conocidas las entradas, permitan encontrar

las salidas. Para efectos de un computador, un sistema es un algoritmo que transforma una secuencia de números (entrada) en otra secuencia de números con propiedades distintas (salida).

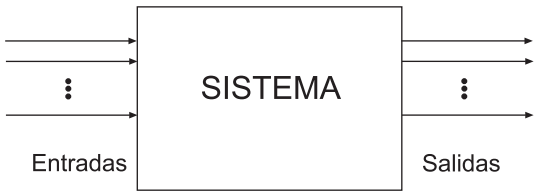


Figura 1.11: Diagrama de un sistema

Tal como lo muestra la figura 1.11, un sistema puede representarse por una "caja negra" a la cual ingresan datos y de la cual salen datos. En audio, los sistemas son muy importantes. Un filtro, como por ejemplo los que posee un equalizador de audio, es un sistema. Un efecto digital, como por ejemplo un *chorus*, *flanger* o *reverb* también son sistemas y se modelan como tales. La teoría e implementación de estos y otros tipos de efectos, se aborda en el capítulo 6.

1.4.1. Clasificación

Existen sistemas dinámicos y estáticos. Un *sistema estático* es uno en el cual los valores de las salidas dependen solamente de los valores actuales de las entradas. En cambio, en un *sistema dinámico* los valores de las salidas dependen de los valores presentes y pasados de las entradas. Un sistema de este tipo posee memoria. Un sistema estático, por ende, no posee memoria.

También los sistemas pueden clasificarse en lineales y no lineales. Cuando un sistema es *lineal*, se puede utilizar el *principio de superposición* , es decir, si un sistema posee varias entradas, pueden analizarse por separado las respuestas del sistema a cada una de las entradas y luego calcular la salida como la suma de las respuestas individuales. En cambio en un sistema *no lineal*, esto no ocurre y su análisis es bastante más complejo.

Los sistemas también pueden ser variantes o invariantes en el tiempo. Un *sistema variante en el tiempo* es uno en el cual las reglas del sistema dependen del tiempo. Es decir, éstas van cambiando a medida que el tiempo cambia. En cambio, los *sistemas invariantes en el tiempo* mantienen sus reglas indefinidamente.

También los sistemas pueden ser continuos o discretos, dependiendo si las variables de entrada y salida son continuas o discretas.

1.4.2. Respuesta al impulso y respuesta de frecuencia

Para estudiar el comportamiento de un sistema sistema lineal e invariante se utilizan normalmente dos tipos de entradas, dado que cualquier tipo de entrada puede ser descompuesta en señales más elementales y por lo tanto la respuesta total del sistema puede calcularse mediante el principio de superposición, si es que se conoce su respuesta a cada una de estas señales elementales.

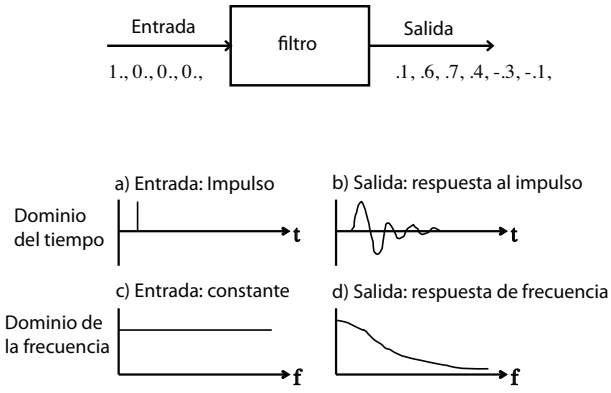


Figura 1.12: Respuesta al impulso

La *respuesta al impulso* corresponde a la respuesta de un sistema a un impulso cuando el sistema se encuentra en estado de reposo. Un impulso, definido en la ecuación 1.27 es una función matemática abstracta que tiene una amplitud infinita y una duración casi cero. Por una propiedad matemática llamada *propiedad del cedazo*, se puede demostrar que cualquier función puede descomponerse en una suma de impulsos. Dado que la transformada de Fourier de un impulso de una función constante en la frecuencia, esta señal es ideal para estudiar sistemas, ya que permite estimar la respuesta de un sistema cualquiera a señales con un contenido de frecuencias previamente determinado.

Otra respuesta muy utilizada para diseñar sistemas es la *respuesta de frecuencia*, que se define como la respuesta del sistema en el dominio de la frecuencia. Esta respuesta puede calcularse como la transformada de Fourier de la respuesta al impulso, o bien puede medirse o estimarse directamente, si se utilizan como entrada señales de tipo sinusoidal. Dado que cualquier señal puede descomponerse, de acuerdo a la serie o transformada de Fourier, en muchas sinusoides individuales, la respuesta total del sistema puede

calcularse mediante el principio de superposición.

En el ámbito de los sistemas digitales, dado que un impulso es una abstracción matemática no posible de representar en un computador, un impulso se implementa como una secuencia de números con valor cero salvo una sola muestra que toma el valor uno. Un tren de impulsos, en cambio, es una secuencia de muestras todas con valor unitario. La figura 1.12 muestra en forma gráfica la relación entre la respuesta al impulso y la respuesta de frecuencia de un sistema digital o filtro.

El estudio de la respuesta de frecuencia de un sistema, es clave en la teoría y diseño de los filtros digitales, la cual se detalla en el capítulo 6. Si se conoce la respuesta al impulso, basta realizar una operación matemática denominada convolución (ver ecuación 6.5) entre una señal de entrada cualquiera y la respuesta al impulso para obtener la respuesta del sistema a esa entrada.

Capítulo 2

El sonido

Si una piedra cae en la superficie de un desierto, donde no hay ser humano alguno ... ¿suena?. Si uno hace esta pregunta las respuestas generalmente son divididas. Hay gente que dice que sí suena y otra dice que no. Esto se debe mayormente a que el sonido existe en dos dimensiones paralelas: una física y otra perceptual. Por lo tanto existen dos definiciones posibles de sonido: una física, donde se considera al sonido como una *perturbación* en algún medio y otra psicoacústica, que se refiere al sonido como la *sensación* que produce una onda sonora en nuestro sistema auditivo.

El sonido en su dimensión perceptual se abordará en el capítulo 3. A continuación se describe el sonido en su dimensión física.

2.1. Ondas de sonido

El *sonido* es una señal producida por una fuente en vibración. Esta vibración perturba las moléculas de aire adyacentes a la fuente en sincronismo con la vibración, creando zonas donde la presión del aire es menor a la presión atmosférica (*rarefacción* o enrarecimiento) y zonas donde la presión del aire es mayor a la presión atmosférica (*compresión*).

Estas zonas de rarefacción y compresión, representadas en la figura 2.1 generan una *onda de sonido* la cual viaja a través del aire. Las ondas en general poseen ciertas propiedades comunes. Las ondas transportan información de un lugar a otro y también transportan energía. Las ondas son parametrizables, lo que quiere decir que pueden describirse de acuerdo a algunos pocos parámetros. Los cuatro parámetros más comunes son: amplitud, período, fase y forma de onda. Existen algunos otros parámetros tales como frecuencia, espectro o intensidad que pueden derivarse de los paráme-

tros mencionados anteriormente. En la figura 2.1 se muestran algunos de ellos.

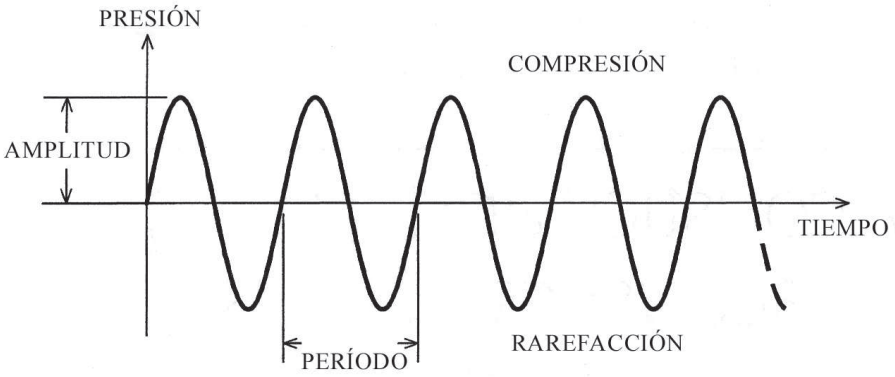


Figura 2.1: Rarefacción y compresión en una onda sonora

Existen distintos tipos de ondas que difieren en su naturaleza, por ejemplo ondas de radio, ondas de luz, ondas de agua, ondas electromagnéticas ondas de sonido o rayos X. También difieren en su forma de propagación, la que puede ser longitudinal o transversal. En una onda longitudinal, el desplazamiento de las partículas es paralelo a la dirección de propagación de la onda. En una onda transversal, el desplazamiento de las partículas es perpendicular a la dirección de propagación de la onda.

Las ondas de sonido son longitudinales. Cuando esta onda alcanza alguna superficie (como el tímpano del oído humano o la membrana de un micrófono), produce una vibración en dicha superficie por simpatía o resonancia. De esta forma, la energía acústica es transferida desde la fuente al receptor manteniendo las características de la vibración original.

El sonido necesita de un medio para propagarse. El sonido puede viajar a través de objetos sólidos, líquidos o gases. Su velocidad es proporcional a la densidad del medio por el cual viaja. A temperatura ambiente, la velocidad el sonido en el aire es de 343 [m/s] y en agua, es de 1500 [m/s]. La velocidad del sonido es independiente de su intensidad y su intensidad decae con la distancia en forma inversamente proporcional.

2.2. Fuentes sonoras

El sonido puede ser producido por distintos tipos de fuentes y procesos. Estos son:

1. Cuerpos en vibración. Un ejemplo de este tipo de fuentes es un diapasón, el cual al ponerse en vibración genera un cierto tipo de onda sonora. Al estar la fuente vibrando, causa un desplazamiento en el aire cercano, lo que produce cambios locales en la presión de aire. Estas fluctuaciones de presión viajan en forma de una onda. Los cuerpos en vibración son las fuentes sonoras más comunes.

2. Cambios en flujos de aire. Un ejemplo de este tipo de fuentes es lo que sucede cuando hablamos. Las cuerdas vocales se abren y cierran en forma alternada, produciendo cambios en la tasa del flujo de aire, lo que a su vez se traduce en una onda sonora. Este mismo principio se aplica a los instrumentos de viento como el clarinete u oboe. Otro ejemplo de este tipo de fuentes es una sirena, la cual produce sonido a través de una placa rotatoria bloquea en forma alternada el flujo proveniente de un compresor de aire.

3. Fuentes de calor. Una chispa eléctrica produce un sonido, tal como lo produce un trueno. En estos casos, el sonido se produce por un brusco cambio en la temperatura, el cual produce una veloz expansión del aire circundante.

4. Flujo supersónico. En el caso de un avión supersónico se producen ondas de choque que fuerzan al aire a viajar más rápido que la velocidad del sonido.

2.3. Medición del sonido

El sonido en su dimensión física es medible o cuantificable. Existen parámetros que pueden ser medidos en forma precisa como su intensidad y otros pueden ser estimados con mayor o menor precisión como la frecuencia o su forma de onda.

El sonido fundamentalmente es una onda de presión, y la presión P puede medirse de la siguiente forma:

$$P = F/a \tag{2.1}$$

lo que equivale a una fuerza F ejercida en una unidad de área a determinada. La presión corresponde a la fuerza ejercida en forma perpendicular a una superficie dividida por el área de ésta superficie.

Las ondas de sonido transportan energía. La energía es una medida abstracta de mucha utilidad en la física, dado que se define de tal forma que en un sistema cerrado la energía siempre es constante, principio que se conoce como la *conservación de la energía*. La energía se mide en unidades de masa por velocidad al cuadrado y usualmente se mide en [Joules]. Otras unidades de medida de la energía son [Btu], [Caloría] y el [kW/h]. Una importante medida del sonido es su potencia $\wp$, la cual corresponde a la energía total por unidad de tiempo y se mide en [Joules/sec].

A continuación se describen en detalle los parámetros físicos del sonido.

2.4. Amplitud e intensidad

La amplitud de un sonido corresponde a la magnitud del cambio, sea este positivo o negativo, de la presión atmosférica causado por la compresión y rarefacción de las ondas acústicas. Esta cantidad es un indicador de la magnitud de energía acústica de un sonido y es el factor que determina que tan fuerte se percibe un sonido. Por lo general, la amplitud se mide en $[N/m^2]$. El rango de amplitudes perceptible por el ser humano va desde los 0.00002 $[N/m^2]$ hasta los 200 $[N/m^2]$, donde el cuerpo entero siente las vibraciones.

La *intensidad* del sonido caracteriza la razón a la cual la energía es entregada en la sensación audible asociada con la amplitud. Suponiendo una fuente puntual que irradia energía uniforme en todas las direcciones, entonces la presión sonora varía en forma inversamente proporcional a la distancia medida desde la fuente y la intensidad cambia en forma inversamente proporcional al cuadrado de la distancia. Si esta distancia es r, entonces se tiene que:

$$I = \wp/4\pi r^2 \tag{2.2}$$

donde $\wp$ es la potencia sonora. La intensidad se mide en $[W/m^2]$. Esto se observa claramente en la figura 2.2. La pérdida de intensidad al incrementarse la distancia es de $dB = 20log_{10}(r_1/r_2)$. Al doblarse la distancia, se experimenta una pérdida en intensidad de 6 dB.

La presión e intensidad se relacionan a través de la siguiente ecuación:

$$P = \sqrt{I\beta c} \tag{2.3}$$

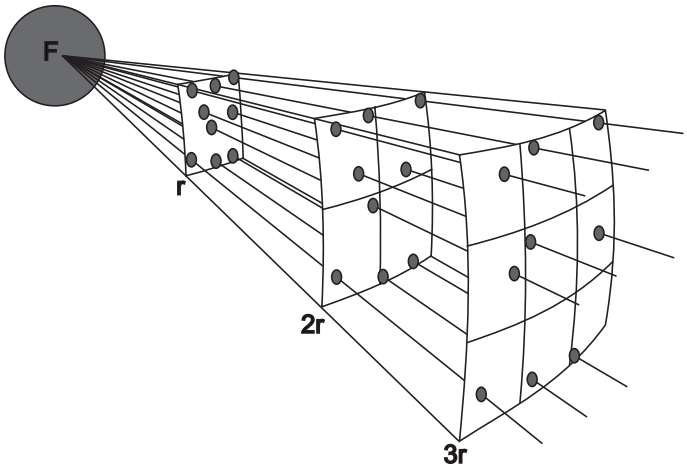


Figura 2.2: Intensidad vs distancia

donde β equivale a densidad del medio y c es la velocidad del sonido en el aire.

El sistema auditivo humano puede detectar un inmenso rango de intensidades desde 10^{-12} $[W/m^2]$ a 1 $[W/m^2]$. En términos de presión, el rango detectable va desde $2 * 10^{-5}$ [Pa] a 2 [Pa], lo que equivale a una razón de 10.000.000:1. En términos prácticos, medir la intensidad de sonido en $[W/m^2]$ resulta inmanejable debido a su enorme rango, por lo que una escala logarítmica de medición de intensidad resulta mucho más apropiada.

La intensidad (sound level) del sonido se mide en *decibeles*. Un *bel* indica una razón de 10:1, por lo tanto

$$1bel = log_{10}(I_1/I_0) \tag{2.4}$$

Pero los *beles* resultan muy grandes para efectos prácticos y por eso se utiliza el *decibel (dB)*, definido por:

$$1dB = 10log_{10}(I_1/I_0) \tag{2.5}$$

I_0 se escoge típicamente como 10^{-12} W/m^2. Un incremento de 10 dB equivale a un incremento de la intensidad del sonido de **un orden de magnitud**. Un incremento de 3dB equivale a doblar la intensidad y un incremento de 1dB representa un 25 % de incremento en la intensidad.

La intensidad es proporcional al cuadrado de la presión

$$1dB = 20log_{10}(P_1/P_0) \tag{2.6}$$

0 dB	Umbral de audición
10 dB	Respiración normal
20 dB	Susurro
50 dB	Conversación suave
70 dB	Tráfico
90 dB	Disparo
110 dB	Exposición prolongada (causa pérdida auditiva)
120 dB	Avión a propulsión despegando
140 dB	Jet despegando
160 dB	Perforación instantánea del tímpano
	(10^{16} veces 0 dB)

Cuadro 2.1: Tabla de intensidades sonoras

donde $P_0 = 2x10^{-5}$ $[Pa]$.

Esta medida se conoce como dB_{SPL} (sound pressure level). 0 dB se escoge para el umbral de audición, el sonido más tenue que puede ser detectado. En los equipos de audio suele usarse el dB_{VU} (volume unit), donde 0 dB corresponde al máximo nivel de audio posible sin tener distorsión (clipping). El área sobre los 0 dB en este caso se conoce como *headroom*.

2.5. Frecuencia

En el caso de un onda periódica, el patrón que se repite corresponde a un *ciclo*. La duración de cada uno de los ciclos de una onda se conoce como *perodo*. La tasa a la cual los ciclos de una onda periódica se repiten se conoce como *frecuencia* y por lo general se mide en ciclos por segundo o Hertz (Hz). Matemáticamente, la frecuencia es el inverso del período, por lo tanto un período de 1 ms (milisegundos) tiene una frecuencia de 1000 Hz. El oído humano percibe frecuencias que van entre los 20 y los 20.000 Hz, aunque esto puede variar para distintas personas.

En términos de su contenido de frecuencia, un sonido puede poseer lo que se denomina una *frecuencia fundamental*, comúnmente denotada por $f0$, que usualmente corresponde a la frecuencia más baja y de mayor amplitud presente en el espectro. Es la frecuencia fundamental de una onda la que determina en gran medida su *altura* musical, la cual es una medida perceptual, explicada en detalle en la sección 3.5. Las señales aperiódicas no poseen una frecuencia fundamental fácilmente determinable, dado que nada se repite en forma periódica. La estimación de $f0$ para señales complejas es

en sí un problema bastante complicado, que escapa a este texto. La ausencia de una fundamental hace que este tipo de ondas se perciba musicalmente como *ruido*.

2.6. Fase

La fase es simplemente el desfase o corrimiento de una señal respecto a un punto de referencia, el que se determina en forma arbitraria. La fase se mide en radianes, en el rango $[0, 2\pi]$ o en grados, en el rango $[0, 360]$. Una fase mayor a 2π o 360 carece de sentido físico, pues no es posible de distinguir de una que ocurre dentro del rango normal.

La fase de un sonido aislado no altera en nada su percepción. La fase adquiere importancia cuando dos o más sonidos se mezclan entre sí. Dos sonidos pueden ser idénticos, pero estar desfasados entre sí, lo que implica que un sonido comenzó antes que el otro. Al interactuar, el resultado percibido puede cambiar radicalmente dependiente del grado de desfase entre ellos. Si el desfase es 0, o bien 2π, los sonidos al mezclarse se suman, y como las zonas de rarefacción y compresión de ambos sonidos coinciden, como resultado se obtiene el mismo sonido pero amplificado. Si el desfase es de π o 180 grados, significa que las zonas de rarefacción de un sonido coinciden con las zonas de compresión del otro, y al mezclarse, los sonidos se anulan completamente. El resultado es que no se percibe nada. Esto no es un problema perceptual, es un fenómeno puramente físico.

La fase es una variable comúnmente ignorada y poco tomada en cuenta, pero es sumamente importante en el análisis de Fourier, abordado en el capítulo 4 y algunas técnicas de síntesis, como las descritas en el capítulo 5, entre otras cosas.

2.7. Forma de onda

El patrón de variaciones de presión producido por una fuente de acuerdo al tiempo se conoce como la *forma de onda*. La forma de onda determina en gran medida la cualidad del sonido. Un factor importante de considerar en un sonido es su periodicidad. En el capítulo 1 se estudiaron distintos tipos de ondas, entre ellas las ondas periódicas y aperiódicas. Ellas se muestran en la figura 1.3. En la primera de ellas, se aprecia que está compuesta de patrones repetitivos, mientras que la segunda posee una forma aleatoria, sin un comportamiento definido.

La forma de onda de un sonido determina y está determinada por su contenido de frecuencias o *espectro*. La forma de onda se asocia comúnmene con lo que se denomina *timbre*, detallado en la sección 3.6, cualidad perceptual que le otorga identidad al sonido. Es la forma de onda la que permite diferenciar, por ejemplo, el sonido de una trompeta del de un violín.

El espectro contiene en el eje de las ordenadas el rango de frecuencias presente en el sonido y en las abscisas, la amplitud de cada componente. Tal como se detalla en el capítulo 4, el espectro de una señal real, como el sonido, no es real, si no complejo o imaginario. Esto significa que las amplitudes no son números reales, sino números complejos y en realidad consisten en un par de números y no en uno sólo. En la sección 1.1.4 se describen las dos formas básicas de representar números complejos: como un par ordenado, lo que se denomina forma cartesiana, o bien mediante una magnitud y un ángulo o fase, lo que se llama forma polar. Usualmente, en el mundo del audio profesional se utiliza el espectro como sinónimo de la magnitud solamente. Si bien la información que entrega la magnitud de un espectro es útil, no lo es todo. Por lo tanto, una representación que sólo contenga la magnitud es en realidad incompleta, ya que no muestra otro componente igual de importante que es la fase.

El espectro puede estimarse en base a la transformada de Fourier, la cual se explica en mayor detalle en la sección 1.2. Sin embargo, existen limitaciones en cuanto a la resolución de ésta estimación tanto de tipo teóricas como prácticas.

2.8. Representación gráfica

Existen diversas formas de representar el sonido en forma gráfica. La forma de representación más utilizada se basa en un diagrama de amplitud versus tiempo, tal como se observa en la figura 2.1, donde se pueden apreciar las zonas de compresión y rarefacción, además de indicaciones algunos parámetros como amplitud y período, que tienen que ver con la *forma de onda*.

Un sonido también puede ser representado por su espectro, mediante un gráfico amplitud versus frecuencia. Este gráfico muestra las amplitudes de cada componente de frecuencia contenido en el sonido.

Por lo general, siempre es posible pasar de una representación en el tiempo a una representación en la frecuencia y viceversa, mediante la transformada de Fourier. No obstante, es importante destacar que el diagrama de tiempo no contiene información alguna sobre el contenido de frecuencias del

sonido y el espectro no contiene información de tipo temporal. Por lo tanto, si bien cada una de estas representaciones basta por sí sola para determinar unívocamente un sonido, son en cierto sentido complementarias.

Una representación intermedia es lo que se llama el sonograma. Un sonograma consiste básicamente en un eje tridimensional donde se grafica la magnitud del espectro de un sonido versus el tiempo. Esto se logra mediante la subdivisión de la señal de audio en varias pequeñas ventanas de tiempo, usualmente traslapadas entre sí. En cada una de éstas ventanas temporales, se estima el espectro mediante lo que se denomina la transformada de Fourier de tiempo corto (o short time Fourier transform en inglés). De esta forma es posible determinar como va cambiando el contenido de frecuencia del sonido en el tiempo. Si bien el sonograma es muy útil, la información que entrega es altamente dependiente de los parámetros que se utilicen para su cálculo, como el tipo de ventana, el tamaño de cada ventana y el porcentaje de traslape, entre otros.

Capítulo 3

Psicoacústica

La psicoacústica es la ciencia que estudia la percepción de los sonidos. Tal como veremos en este capítulo, la percepción sonora está basada en fenómenos bastante complejos. Una vez que una onda sonora proveniente del mundo físico ingresa al sistema auditivo humano, se suceden una serie de reacciones en forma casi instantánea que producen como resultado una representación mental de lo escuchado, que no corresponde exactamente a lo que sucede en el mundo físico. Las variables físicas del sonido estudiadas en el capítulo anterior no constituyen una representación fidedigna de lo que ocurre en el mundo perceptual. Por ejemplo, en ciertas situaciones hay sonidos que bloquean a otros sonidos, incluso si éstos ocurren en forma asincrónica. Este fenómeno se conoce como enmascaramiento. En estos casos, si bien todas las ondas sonoras en juego existen en el mundo físico, al presentarse todas juntas algunas de ellas simplemente no se perciben, a pesar de que si cada uno de estos sonidos se presentaran por separado, si se percibirían.

3.1. Conceptos básicos

3.1.1. Mínima diferencia notoria

La psicoacústica intenta medir o cuantificar la percepción de los sonidos. Una forma de estudiar la percepción es medir el mínimo cambio de alguna variable que produzca efectivamente un cambio notorio en la percepción de algún estímulo. Esto se conoce como la *mínima diferencia notoria* y usualmente se abrevia como JND (del inglés *just noticeable difference*). También se conoce como umbral diferencial o limen diferencial.

Usualmente, la mínima diferencia notoria varía de persona a persona y de acuerdo al método de medición. Incluso, puede cambiar para la misma persona, si las condiciones externas son modificadas.

3.1.2. Ley de Weber

La Ley de Weber es una ley obtenida experimentalmente a través de experimentos perceptuales. Cuando se está midiendo la percepción de algún estímulo, esta ley afirma que la mínima diferencia notoria es proporcional a la magnitud del estímulo original. Matemáticamente, si tenemos un estímulo I y nos interesa entonces medir la mínima diferencia ΔI que producirá un cambio efectivo en la percepción de I, entonces se cumple que:

$$\frac{\Delta I}{I} = k \tag{3.1}$$

donde k es una constante. Esto significa que a mayor intensidad de estímulo, mayor es la diferencia que se necesita para producir un cambio en la percepción de ese estímulo.

3.1.3. Modos de percepción

Existen dos modos principales de percepción: lineal y logarítmico. En el primer caso, el cambio entre dos valores de una variable se percibe sobre la base de la diferencia entre ambos valores. En el segundo, el cambio entre dos valores de una variable se percibe de acuerdo a la razón entre ellos.

Tal como se verá a continuación, la percepción de sonidos por parte del sistema auditivo humano es altamente logarítmica.

3.2. El sistema auditivo humano

En su famoso libro “Auditory Scene Analysis”, Albert Bregman [5], describe la siguiente analogía: supongamos que estamos frente a un lago, del cual salen dos surcos largos y angostos hacia la orilla. En cada surco se coloca una hoja de árbol, la cual se moverá a medida que el agua del lago fluya hacia los surcos. Sólo analizando el movimiento de ambas hojas, sin mirar nada más que las hojas, la idea es poder responder preguntas como:

1. ¿Cuántos barcos hay en el lago y donde están?

2. ¿Qué barco esta más cerca?

3. ¿Cual es más potente?

4. ¿Hay viento?

5. ¿Ha caido algún objeto grande sobre el lago?

6. ¿Hay algún nadador y qué tan rapido nada?

Esto puede parecer una misión imposible. Pero la verdad es que el sistema auditivo humano hace este tipo de análisis todo el tiempo y a una velocidad extremadamente alta. Nuestros tímpanos equivalen a las hojas depositadas en los surcos. Nuestro sistema auditivo, basándose únicamente en el análisis del patrón de variaciones en ambos tímpanos, es capaz de generar una idea muy precisa de nuestro entorno auditivo. Suponiendo que nos encontremos en una fiesta muy ruidosa, lo que típicamente de conoce en inglés como el cocktail party problem, donde hay muchas fuentes sonoras de todo tipo interactuando al mismo tiempo, no necesitamos realizar ningún esfuerzo para responder acertadamente preguntas cómo ¿Cuánta gente está hablando? o ¿Quién suena más fuerte o está más cerca? o ¿Hay algún máquina haciendo ruido de fondo?. El sistema auditivo humano se extiende desde el oído externo hasta la corteza cerebral y es capaz de realizar una cantidad impresionante de operaciones complejas en muy poco tiempo. A continuación, se detallan los principales componentes de nuestro sistema auditivo.

3.2.1. El oído

Sin duda el uno de los componentes más importantes del sistema auditivo humano es el oído. El oído se encarga de convertir las ondas de presión acústica en impulsos nerviosos que le permiten al cerebro crear una representación mental de la sensación auditiva. La figura 3.1 muestra las partes y componentes más importantes del oído humano. Tal como se puede observar en la figura, el oído humano está dividido en tres partes principales: oído externo, oído medio e oído interno. Cada una de éstas partes cumplen roles específicos en el proceso general de la audición.

El funcionamiento general del oído es el siguiente: cuando una onda sonora llega al oído viaja desde el pabellón auricular o pinna hasta el tímpano, a través del canal auditivo. El sonido es modificado en términos de su contenido de frecuencias por el oído externo. El tímpano se encarga de traspasar el patrón de vibraciones de presión hacia el oído medio al hacer contacto con tres diminutos huesecillos, que a su vez traspasan la vibración hacia la ventana oval, lo que causa una onda de propagación del líquido contenido

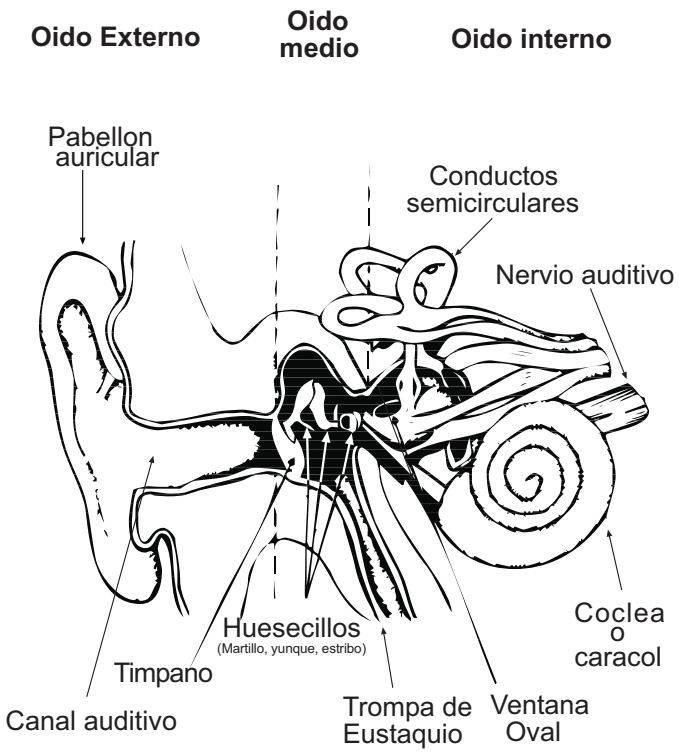


Figura 3.1: El oído humano

al interior de la cóclea, estimulando las células ciliares de la membrana basilar, la que está conectada a un gran número de terminales nerviosos que envían a su vez señales eléctricas al cerebro. De esta manera, el cerebro puede recibir la información proveniente de la onda sonora para su posterior procesamiento.

A continuación se detalla el funcionamiento de cada una de las partes del oído.

Oído externo

El oído externo está constituido por el pabellón auricular o pinna , el canal auditivo y el tímpano. El efecto de la pinna es atenuar y enfatizar cierto contenido de frecuencias y juega un rol fundamental en la localización de sonidos. El canal auditivo mide alrededor de 2,5 cm y actúa como

un resonador para el rango de frecuencias entre 1000 y 4000 Hz, siendo el máximo alrededor de 3000 Hz. El tímpano vibra en respuesta al sonido y transmite esta vibración de presión en forma de vibración mecánica hacia el oído medio.

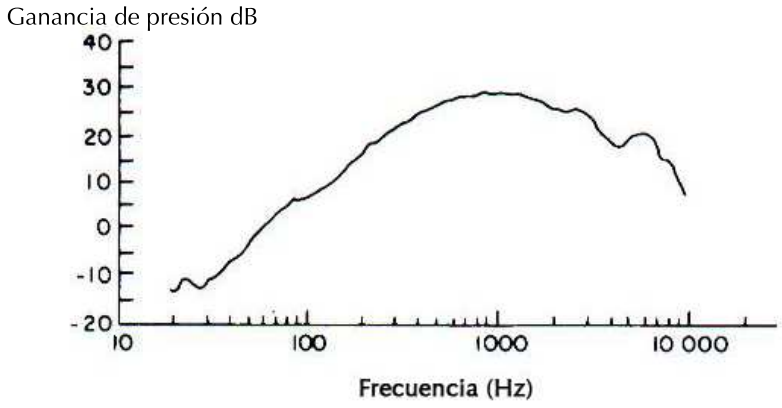


Figura 3.2: Respuesta de frecuencia del canal auditivo

La figura 3.2 detalla la respuesta de frecuencia del canal auditivo. Claramente se puede observar que el canal auditivo enfatiza el rango desde 1 a 4 kHz. Esto corresponde al rango de la voz humana hablada. El rol que cumple el canal auditivo, por lo tanto, es el de optimizar la señal acústica del tal forma de resaltar la voz humana.

Oído medio

El oído medio actúa como un transductor de vibración. Su rol es amplificar la vibración de presión mediante un sistema mecánico. Esto se hace mediante tres huesitos llamados martillo, yunque y estribo. La vibración de estos huesitos puede ser amortiguada por un músculo anexado a ellos llamado stapedius, lo cual otorga protección contra sonidos muy intensos, los cuales hacen contraer este músculo, pero no protege contra sonidos muy intensos y súbitos, como un disparo.

Oído interno

El oído interno consiste básicamente de la cóclea, ya que los canales semicirculares, si bien se encuentran allí, no tienen rol alguno en la audición. Pero

si tienen un rol fundamental en el equilibrio, es por esto que una infección en el oído interno usualmente afecta la capacidad de mantener el equilibrio. La figura 3.3 muestra un esquema que representa un corte del oído interno, con sus componentes principales. La cóclea se muestra desenrollada en éste corte.

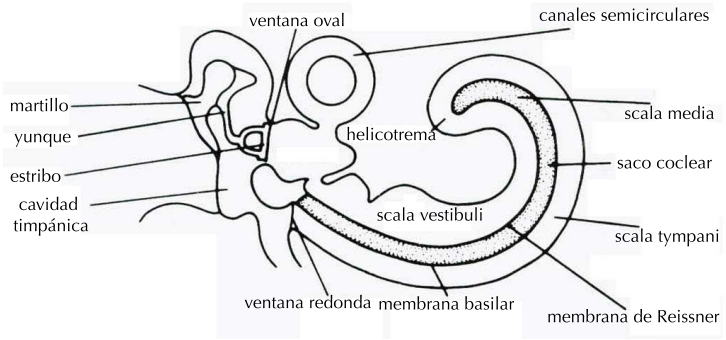


Figura 3.3: Sección del oído interno

La cóclea es un pasaje angosto, con forma de caracol, lleno de líquido incompresible, largo y enrollado 3,5 veces sobre sí mismo. El diámetro de este pasaje es de 2 mm y su largo es 35 mm. La figura 3.4 muestra un corte perpendicular de la cóclea, la cual está dividida en tres secciones: scala tympani, scala vestibula y scala media. Estas zonas están unidas por el helicotrema y contienen fluídos de distintas densidades que vibran de acuerdo a la onda transmitida por el sistema mecánico del oído medio. Estos fluídos transmiten una onda de propagación que estimula la membrana basilar, lugar donde se codifica la información sonora.

3.2.2. Membrana basilar

El órgano de corti es el lugar donde ocurre la producción de impulsos nerviosos. En su fondo se encuentra la membrana basilar. La membrana basilar separa la scala media de la scala tympani. La membrana basilar responde a estímulos sonoros y causa vibración en algunas de las 3500 células ciliares, conectadas a la membrana tectorial encima de ellas. Esta vibración es transmitida por el nervio auditivo al cerebro para su procesamiento.

La membrana basilar comienza en la base, conectada a la ventana oval y termina en el apex, en el helicotrema. Al presionarse lentamente la ventana

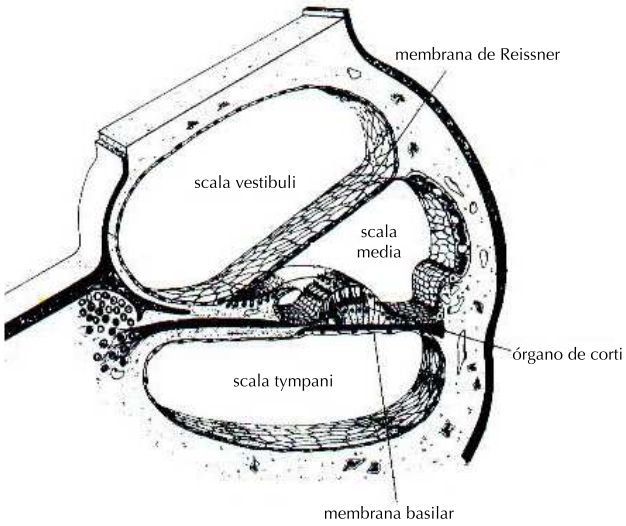


Figura 3.4: Sección perpendicular de la cóclea

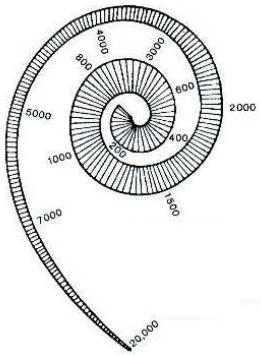


Figura 3.5: La membrana basilar desenrrollada

oval, la ventana redonda también se ve presionada porque el volumen de líquido al interior de la cóclea es constante. Si la ventana oval se presiona en forma rápida, se produce una onda en la membrana basilar que viaja desde la ventana oval hacia el helicotrema. Esta onda no puede pasar más allá de un punto determinado, dependiendo de la frecuencia a la cual vibra la membrana basilar. La membrana basilar entonces responde en forma diferente

dependiendo de la frecuencia presente en su base, que está conectada a la ventana oval. Esto significa que sonidos graves pueden enmascarar a sonidos agudos, pero no al revés.

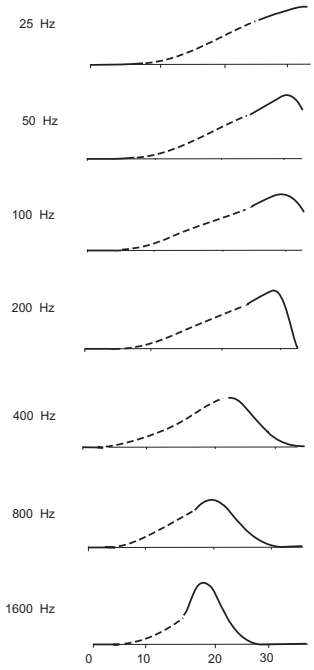


Figura 3.6: Patrones de vibración en la membrana basilar

El punto de máximo desplazamiento de la membrana basilar está cerca del apex para frecuencias bajas y cerca de la base para las altas frecuencias. Las frecuencias bajas estimulan tanto el apex como la base. No así las altas frecuencias, que están confinadas a la zona basal. Este hecho explica fenómenos como el enmascaramiento y la razón por la cual la teoría de la posición (ver sección 3.5) para la percepción de alturas no funciona para las bajas frecuencias. En la figura 3.5 se muestra un dibujo de la membrana basilar desenrrollada y se indican los puntos que resuenan de acuerdo a la frecuencia del estímulo recibido.

Como cada punto de la membrana basilar alcanza un desplazamiento máximo a distintas frecuencias, es posible estudiar la respuesta de la membrana en forma global de acuerdo a la frecuencia. La figura 3.6 muestra los patrones de vibración de la membrana basilar para tonos de distintas frecuencias fundamentales. Es fácil de observar que el punto de máxima

excitación de la membrana varía con la frecuencia de entrada.

3.2.3. Células auditivas

En la figura 3.7 se muestra un esquema del órgano de corti, el cual se encuentra entre la membrana basilar y la membrana tectorial. Contiene en su interior las células ciliares o auditivas, que son las encargadas de transmitir la información proveniente de la membrana basilar en forma de impulsos eléctricos hacia el nervio auditivo y la corteza auditiva. Hay dos tipos de células auditivas: internas y externas. Las células internas están compuestas por fibras que envían pulsos hacia el cerebro. Las células externas están compuestas por fibras que reciben pulsos nerviosos provenientes de las áreas superiores del sistema auditivo.

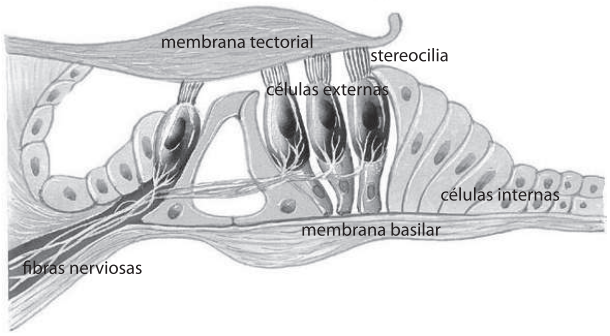


Figura 3.7: Esquema del órgano de corti, que contiene las células auditivas o ciliares

3.2.4. El cerebro auditivo

El cerebro funciona como una superficie 2D (materia gris) altamente convolucionada. La superficie del cerebro contiene el final neuronas cuyos axones están debajo de esta superficie y las distintas regiones del cerebro se interconectan por esta causa. La corteza auditiva esta organizada tonotópicamente, es decir existen áreas del cerebro organizadas espacialmente en torno a distintos rangos de frecuencia. Es como si ciertas áreas del cerebro fueran sensibles a sólo un grupo reducido de frecuencias. Esto indica que hay algo fundamental en torno a la frecuencia en la respuesta del cerebro al sonido, incluso al más alto nivel, como es la corteza auditiva.

3.3. Bandas críticas y enmascaramiento

Una característica fundamental del sistema auditivo humano es su capacidad resolución de frecuencia e intensidad. Al momento de estudiar éste aspecto de nuestra audición es fundamental el concepto de banda crítica. Una forma de entender el funcionamiento del sistema auditivo es suponer que contienen una serie o banco de filtros pasa banda sobrelapados conocidos como filtros auditivos (Fletcher (1940) citado en (Moore, 1998)). Estos filtros se producen a lo largo de la membrana basilar y tienen como función aumentar la resolución de frecuencia de la cóclea y así incrementar la habilidad de discriminar entre distintos sonidos. Este banco de filtros no sigue una configuración lineal, y el ancho de banda y morfología de cada filtro depende de su frecuencia central. El ancho de banda de cada filtro auditivo se denomina banda crítica (Fletcher (1940) citado en (Gelfand 2004).

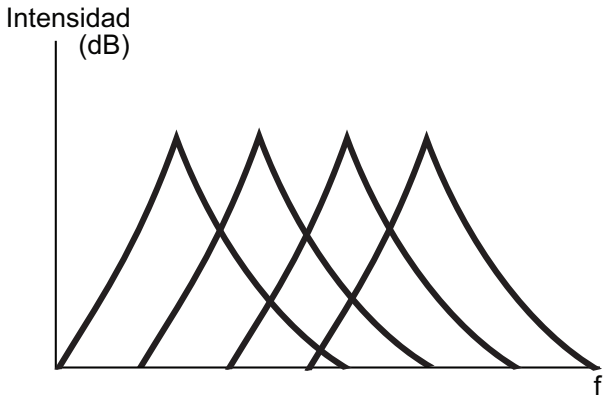


Figura 3.8: Esquema de las bandas críticas del sistema auditivo humano

Las bandas críticas, esquematizadas en la figura 3.8, son rangos de frecuencia dentro de los cuales un sonido bloquea o enmascara la percepción de otro sonido. Las bandas críticas conceptualmente están ligadas a lo que sucede en la membrana basilar, ya que una onda que estimula la membrana basilar perturba la membrana dentro de una pequeña área más allá del punto de primer contacto, excitando a los nervios de toda el área vecina. Por lo tanto, las frecuencias cercanas a la frecuencia original tienen mucho efecto sobre la sensación de intensidad del sonido. La intensidad percibida no es afectada, en cambio, en la presencia de sonidos fuera de la banda crítica. Es importante destacar aquí que el concepto de banda crítica es una

construcción teórica y no algo físicamente comprobado.

3.3.1. Escala de Barks y ERB

Una inquietud que surge de inmediato es preguntarse cuantas bandas críticas existen en el sistema auditivo y cual es la frecuencia central de cada una. Existe una escala de medición de las bandas críticas llamada la escala de Barksla cual se detalla en la tabla 3.3.1. La escala tiene un rango del 1 al 24 y corresponde a las primeras veinticuatro bandas críticas del sistema auditivo. Esta escala tiene relación con la escala mel, abordada en la sección 3.5.1 pero no es tan utilizada.

La relación entre un Bark y un Hz está dada por:

$$\text{Bark} = 13\arctan(0{,}00076f) + 3{,}5\arctan((\frac{f}{7500})^2) \tag{3.2}$$

Otro concepto asociado al concepto de banda crítica es el equivalente rectangular de ancho de banda, o ERB, por sus siglas en inglés.Esta medida muestra la relación entre el conducto o canal auditivo y el ancho de banda de un filtro auditivo. La idea de esta medida es reemplazar una banda crítica por un rectángulo equivalente cuya área sea la misma, de manera de permitir en su interior la misma cantidad de energía.

La ERB se calcula mediante la siguiente ecuación:

$$\text{ERB} = 24{,}7(4{,}37f + 1) \tag{3.3}$$

La figura 3.22 muestra la escala de Barks, la ERB y la escala mel en función de la frecuencia. Ambas tienen curvas parecidas, pero difieren en algunos detalles.

La figura 3.9 muestra una aproximación computacional a los filtros auditivos, basado en la escala ERB.

3.3.2. Enmascaramiento

El enmascaramiento ocurre cuando la presencia de un sonido, llamado máscara, hace inaudible otro sonido que sería perfectamente audible en la ausencia de la máscara. Esto ocurre si es que ambos sonidos se encuentran dentro de la misma banda crítica, tal como lo muestra la figura 3.10.

El sistema auditivo no es capaz de diferenciar dos sonidos al interior de una banda crítica. Basta con que exista algo de energía al interior de una banda crítica para que ésta se active y el sistema auditivo perciba actividad en esa banda. Si existe más de un sonido o se incrementa la energía

Banda crítica (Bark)	Frec. central (Hertz)	Ancho de banda (Hertz)	Frec. min. (Hertz)	Frec. max. (Hertz)
1	50	-	-	100
2	150	100	100	200
3	250	100	200	300
4	350	100	300	400
5	450	110	400	510
6	570	120	510	630
7	700	140	630	770
8	840	150	770	920
9	1000	160	920	1080
10	1170	190	1080	1270
11	1370	210	1270	1480
12	1600	240	1480	1720
13	1850	280	1720	2000
14	2150	320	2000	2320
15	2500	380	2320	2700
16	2900	450	2700	3150
17	3400	550	3150	3700
18	4000	700	3700	4400
19	4800	900	4400	5300
20	5800	1100	5300	6400
21	7000	1300	6400	7700
22	8500	1800	7700	9500
23	10500	2500	9500	12000
24	13500	3500	12000	15500
25	18775	6550	15500	22050

Cuadro 3.1: Escala de Barks, para estimación de las bandas críticas del sistema auditivo

al interior del filtro, esto no cambia la información desde el punto de vista del sistema auditivo. Entonces, si un sonido se encuentra al interior de una banda crítica de otro sonido y si su amplitud no es lo suficientemente grande como para traspasar el umbral de dicha banda y activar otra banda crítica cercana, se produce el fenómeno denominado enmascaramiento. La codificación perceptual de audio, detallada en la sección 3.8, se basa fuertemente en éste fenómeno para reducir la cantidad de información necesaria para almacenar y reproducir una señal sonora.

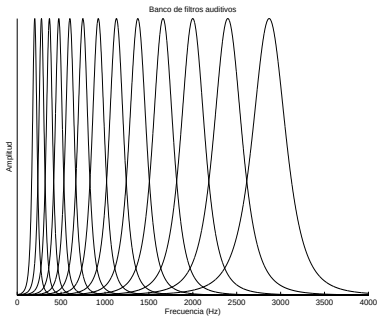


Figura 3.9: Banco de filtros auditivos

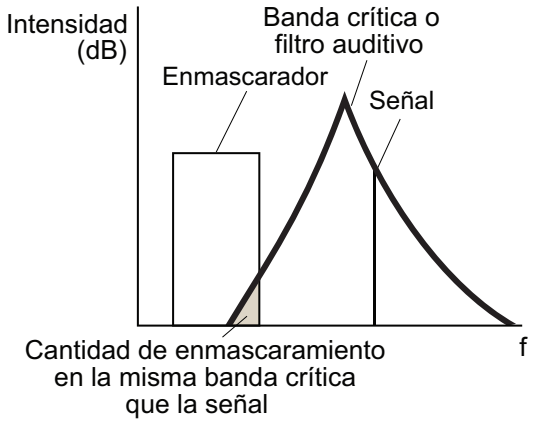


Figura 3.10: Enmascaramiento

Las bandas críticas pueden medirse en función del enmascaramiento que producen de acuerdo al procedimiento descrito en la figura 3.11. El ancho de banda de una máscara constituída por ruido blanco se ensancha contínuamente y se mide el nivel de enmascaramiento que produce respecto a una señal de prueba. Una vez que se alcanza el punto sobre el cual el enmascaramiento no cambia significativamente al agrandarse el ancho de banda, se está en la presencia de los límites de la banda crítica en medición.

3.3.3. Midiendo la banda crítica

El tipo de enmascaramiento más común es el que se denomina simultáneo, es decir cuando la máscara y el sonido enmascarado coinciden en el tiempo,

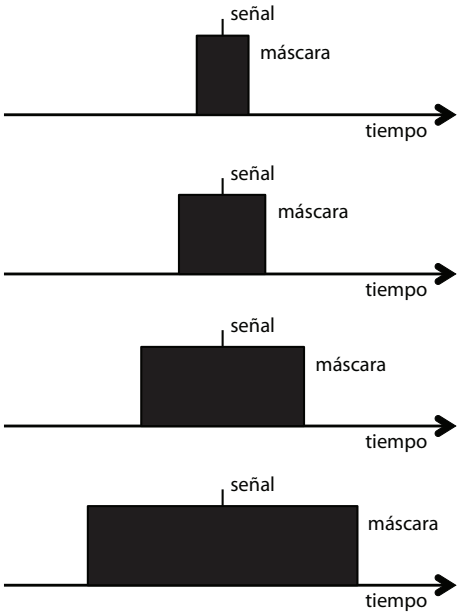


Figura 3.11: Medición de la banda crítica

tal como lo muestra la figura 3.12.

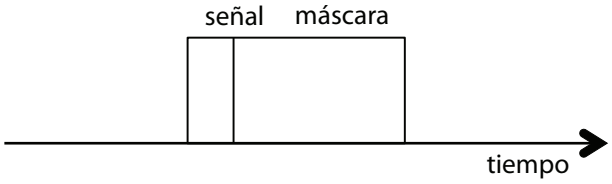


Figura 3.12: Enmascaramiento simultáneo

Pero existe también enmascaramiento de tipo temporal, que ocurre cuando la máscara se encuentra antes o después del sonido enmascarado en el tiempo. La figura 3.13 muestra lo que se denomina enmascaramiento hacia atrás (backward masking en inglés) y la figura 3.14 muestra lo que se denomina enmascaramiento hacia adelante (forward masking en inglés)

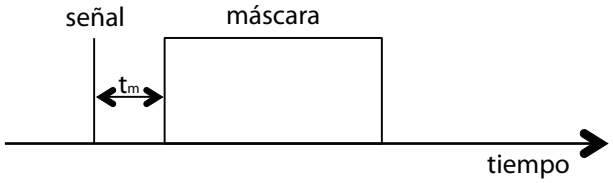


Figura 3.13: Enmascaramiento hacia atrás

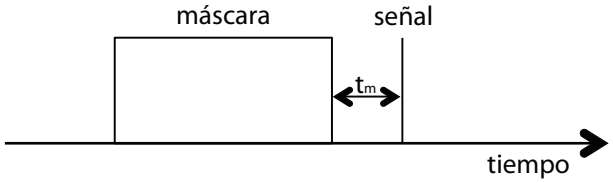


Figura 3.14: Enmascaramiento hacia adelante

3.4. Intensidad perceptual (loudness)

La figura 3.15 muestra los umbrales de audibilidad del sistema auditivo humano, en función de la frecuencia. En la línea mas gruesa se muestran los límites de la audición humana, en la parte baja el umbral de audibilidad, que corresponde a la mínima intensidad necesaria para percibir un sonido y en la parte superior el umbral del dolor, correspondiente a la máxima intensidad que nuestro sisema auditivo es capaz de tolerar. Tal como se puede apreciar, la percepción de intensidad no es uniforme para todas las frecuencias. Para comenzar a percibir un tono de 100 Hz se necesita una intensidad de unos 40 dB SPL mientras que para un tono de 2000 Hz sólo basta con una intensidad de unos 5 dB SPL. Claramente, el rango entre 1 a 5 kHz es el más fácil de percibir.

La percepción de la intensidad de un sonido no corresponde exactamente a su intensidad física. En el idioma inglés existe el término loudness para referirse a la intensidad perceptual de un sonido y diferenciar ésta medida de la intensidad física. En el idioma castellano no existe un término equivalente y en el presente texto se utilizará la expresión *intensidad perceptual* para referirse a la medida perceptual de la intensidad de un sonido. Usando una expresión coloquial, la idea es determinar *que tan fuerte* suena algo.

La sensación de intensidad perceptual depende principalmente de la cantidad de energía acústica recibida por el oído. Tal como sucede con la percep-

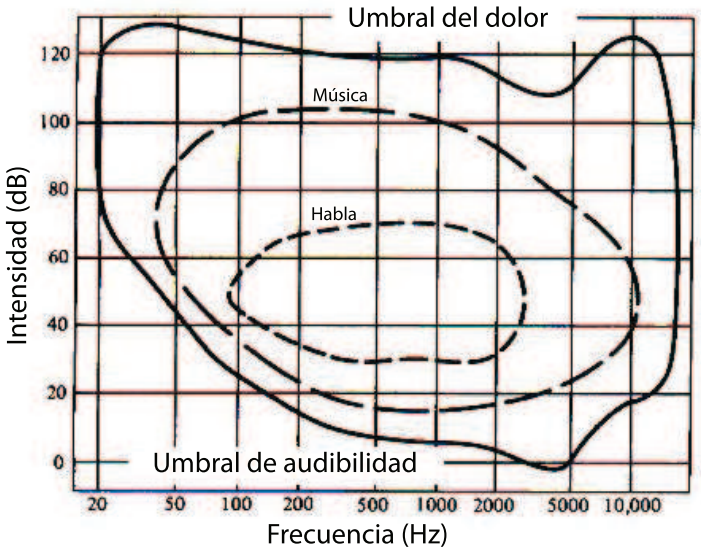


Figura 3.15: Umbrales de audibilidad

ción de altura, la intensidad de un sonido es percibida en forma logarítmica, siguiendo en forma casi exacta la ley de Weber.

En relación con las bandas críticas, si las frecuencias de dos sonidos caen dentro de una cierta banda crítica, su intensidad perceptual será la suma de sus intensidades individuales. Sin embargo, si las frecuencias de dos sonidos están fuera de esta banda crítica, su intensidad perceptual será mayor a la suma de las intensidades individuales.

3.4.1. Escala de fonos y escala de sonos

Existe una escala establecida para medir la intensidad perceptual, denominada *escala de fonos*. Un fono (phone en inglés) se define como el nivel de un tono de 1000 Hz medido en dB SPL que iguala la intensidad percibida de un sonido de referencia. Por ejemplo, si un motor eléctrico es igual en intensidad perceptual a un tono de 1000 Hz a 65 dB SPL, el motor tiene una intensidad perceptual de 65 fonos. La figura 3.16 muestra una serie de curvas para las cuales la intensidad perceptual medida en fonos es constante. Estas curvas fueron inicialmente medidas por Fletcher y Munson y son denominadas *contornos de intensidad equivalentes* (en inglés equal loudness contours). Por ejemplo, de acuerdo a dichos contornos, un tono de 9 kHz a

57dB tiene una intensidad perceptual de 50 fonos.

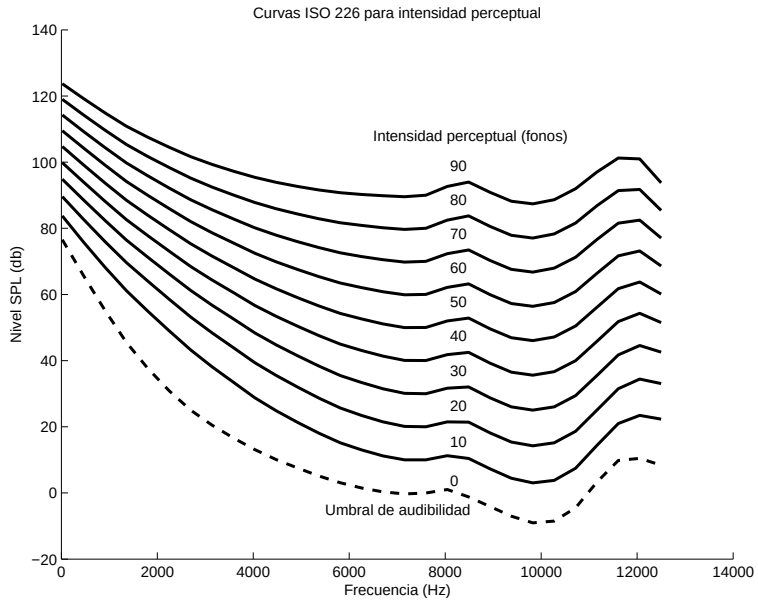


Figura 3.16: Contornos de intensidad perceptual, norma ISO 226

Existe otra escala que permite medir la intensidad perceptual denominada *escala de sonos*. Un son se define arbitrariamente como la intensidad perceptual de un tono de 1000 Hz a 40 dB SPL. Por lo tanto, un sonido tiene una intensidad perceptual de 2 sones si se juzga como el doble de intenso que un tono de 1000 Hz a 40 dB SPL.

Basándose en información proveniente de experimentos psicoacústicos, es posible determinar una relación matemática entre la intensidad perceptual (L) y la intensidad física (I), dada por la siguiente ecuación:

$$L = kI^{0,3} \tag{3.4}$$

donde k es una constante. Esta ecuación nos indica que para sonidos sobre 40 dB, la intensidad perceptual se duplica cuando el nivel de sonido es incrementado en 10 dB.

Para cada una de las curvas, el nivel de sonido total es constante. El ruido tiene mayor intensidad perceptual que un tono de referencia después de un cierto punto crítico.

3.5. Altura (pitch)

Tal como sucede con la intensidad perceptual, en el idioma inglés existe un término especial para referirse a la percepción de la periodicidad de un sonido, llamado pitch. En el idioma castellano podría traducirse como tono o altura. Ambas palabras tienen inconvenientes. Usualmente, la palabra tono se refiere indistintamente a un tipo especial de sonido, de carácter sinusoidal o bien a la percepción de la frecuencia o periodicidad. Por otra parte, la palabra altura proviene del espacio, es una medida de distancia y no aparece como muy apropiada para referirse a un fenómeno exclusivamente temporal. Además, tal como se verá más adelante, la percepción de periodicidad sonora es un fenómeno bi-dimensional, y se divide en lo que se llama en inglés pitch height (que podría traducirse como altura) y pitch chroma (color). Por lo tanto, en realidad la palabra altura se refiere a sólo una de las dimensiones de esta medida.

A pesar de estas limitaciones, en el presente texto se utilizará la palabra altura para referirse a la medida perceptual de la periodicidad de un sonido, siempre teniendo en cuenta la otra dimensión de color, y la palabra tono se referirá a un sonido sinusoidal, comúnmente referido como tono puro.

La altura, que es un fenómeno puramente perceptual, está muy correlacionada con la frecuencia física, pero no existe una relación exacta uno a uno entre ellas. La altura no debe confundirse con lo que se conoce como "brillo", el cual depende de la distribución de energía entre las altas y bajas frecuencias y no exclusivamente de la periodicidad de un sonido.

El rango de frecuencias que percibe el sistema auditivo humano se extiende usualmente desde los 20 Hz hasta los 20.000 Hz o 20 kHz. Si bien el límite inferior es comúnmente aceptado, existe controversia sobre el límite superior. Existe evidencia que indica que el rango normal de una persona adulta no supera los 17 kHz como límite superior, dada la típica pérdida auditiva asociada a la edad, pero hay investigadores que afirman que el rango de audición humana va más allá de los 20 kHz.

La mínima diferencia notoria (JND) de alturas es de alrededor de 1/30 de la banda crítica para una frecuencia determinada. Ésta depende de la frecuencia, intensidad y la duración, como también de cambios súbitos en la frecuencia. La resolución de frecuencia auditiva es muy superior a la resolución de frecuencia visual o la percepción de color. La visión humana abarca en total unas 120 JND, lo que equivale a una octava. En cambio, el sistema auditivo humano abarca unas 5000 JND, lo que equivale a unas diez octavas.

Los mecanismos de percepción de altura funcionan en forma distinta para las altas y bajas frecuencias. No existe una teoría que explique com-

pletamente el funcionamiento de la percepción de alturas por sí sola y es necesario, entonces, recurrir a dos teorías distintas pero complementarias para explicar nuestra percepción de alturas.

La teoría de la posición (o frecuencia) argumenta que la percepción de alturas está determinada por el lugar de máxima vibración en la membrana basilar, tal como se muestra en la figura 3.6. Esta teoría funciona muy bien para altas frecuencias, y no así para las bajas frecuencias, dado que en este caso la membrana entera vibra con los sonidos graves y no sería posible discriminar alturas usando solamente esta información.

La teoría temporal (o de periodicidad) considera que la altura percibida está determinada por la distribución de los impulsos eléctricos en las neuronas auditivas. Sucede que éstos impulsos eléctricos producidos por las neuronas se sincronizan con la frecuencia fundamental del estímulo que está siendo percibido. Este fenómeno se conoce como sincronización de fase (en inglés phase locking). Esta teoría funciona para bajas frecuencias, hasta unos 5 kHz, porque hay un límite físico que impide a las neuronas disparar impulsos eléctricos a tasas mayores.

La altura es en realidad una medida bi-dimensional que puede ser mejor descrita en forma circular, o por una espiral, tal como lo muestra la figura 3.17. Esta representación fue propuesta por Roger Shepard en 1982.

De acuerdo a esta idea, la altura posee realmente dos atributos, denominados en inglés pitch height y pitch chroma. El primero efectivamente mide la altura de un sonido en una escala ascendente y el segundo mide algo que se podría denominar como color o croma, que se refiere a la nota musical más próxima a una determinada frecuencia física y que se representa en forma circular. Esta es la razón por la cual los seres humanos son capaces de reconocer las notas musicales independiente de la octava en que se encuentren. En algunas situaciones es más fácil determinar el croma de un sonido que su altura efectiva.

La relación entre frecuencia, medida en Hz, y la altura percibida no es lineal. Esto se debe a que la estimulación en la membrana basilar ocurre en puntos espaciados de acuerdo a una función logarítmica de la frecuencia. Cuando la frecuencia de un estímulo sonoro se duplica, la distancia entre los puntos de estimulación de la membrana basilar se encuentran aproximadamente a una distancia constante, de unos 3.4mm para el caso de las altas frecuencias. Esto implica que los seres humanos comparamos sonidos de distintas frecuencias no en base a la diferencia entre estas frecuencias, si no en base a la razón entre ellas, lo que musicalmente se conoce como un intervalo musical. Un ejemplo claro de aquello es justamente la octava, intervalo que implica una razón 2:1, cuando la frecuencia se duplica. Percep-

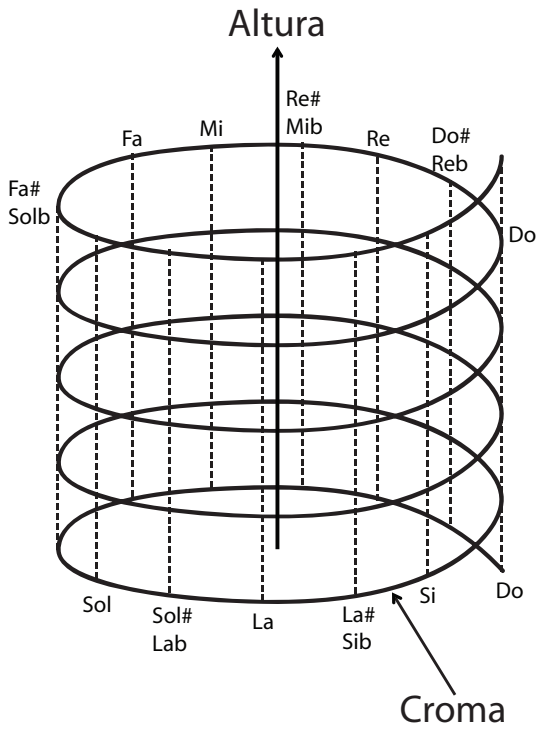


Figura 3.17: Pitch

tualmente, cuando hay una razón 2:1 esta distancia se juzga como la misma, una octava, independiente del rango de frecuencias donde ocurra.

Por lo general, la altura está altamente correlacionada con la frecuencia fundamental de un sonido. Sin embargo, existe evidencia que muestra que la percepción de altura no está relacionada únicamente con la fundamental. Ejemplo de esto son algunas ilusiones auditivas tales como los tonos de Shepard o de Risset, diseñadas especialmente para engañar al sistema auditivo.

Los tonos de Shepard se construyen con componentes sinusoidales separados por una octava, a los cuales se les aplica un envolvente de tipo Gaussiano, tal como se muestra en la figura 3.18. Si se elevan las frecuencias de los componentes de frecuencia en un semitono, se percibe un incremento en la altura. Pero cuando esto se repite 12 veces, se vuelve al punto de partida y el sistema auditivo ya no es capaz de determinar si la altura ha subido o bajado. Si el incremento es de 6 semitonos, alguna gente escucha

un movimiento ascendente y otras uno descendente.

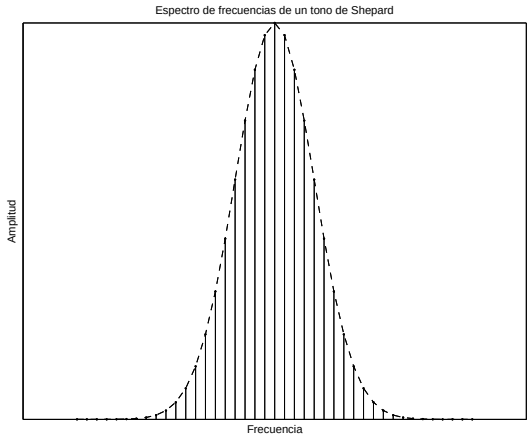


Figura 3.18: Espectro de un tono de Shepard

Otra variante de los tonos de Shepard, son los de Risset, en los cuales los parciales no se encuentran en una relación armónica y los parciales están separados por un poco más de una octava. Si se doblan las frecuencias de los parciales, se percibe un descenso en la altura. Esto se debe a que el sistema auditivo explica este fenómeno mediante el reemplazo del tono por parciales de frecuencias un poco menores que las existentes, de otra manera, la situación no le hace ningún sentido.

Estas ilusiones demuestran que la percepción de altura depende del patrón de armónicos del sonido y no sólo de la frecuencia fundamental. Contrariamente a lo que se pudiera pensar, ni siquiera depende de la presencia física de un tono de la frecuencia correspondiente a la altura percibida. Este fenómeno se conoce como altura virtual y se muestra en la figura 3.19. En la figura se observa un experimento dividido en tres partes. En la parte a) se tiene una sinusoide pura con frecuencia fundamental f_0 y un tono virtual compuesto por los tres primeros armónicos $f0$, $f1$ y $f2$. En ambos casos se percibe la misma altura, correspondiente a la frecuencia fundamental. Esto sucede por que el sistema auditivo al encontrar los primeros armónicos automáticamente trata de encontrar algún patrón que tenga sentido y concluye que lo más evidente es que se trate de un tono con frecuencia fundamental $f0$, aunque no esté presente físicamente. Para probar ésta hipótesis, en la parte b) se aplica un filtro pasa bajo con frecuencia de corte entre $f0$ y $f1$ y en la parte c) un filtro pasa altos con la misma frecuencia de corte. En

el caso b) la sinusoide no se percibe porque es eliminada por el filtro pasa bajos pero el tono virtual se percibe con la misma altura de antes, ya que el filtro en realidad no elimina nada. En el caso c) la sinusoide se percibe ya que no es afectada por el filtro pasa altos, pero el tono virtual deja de percibirse, ya que desaparecen los armónicos que permitían su percepción.

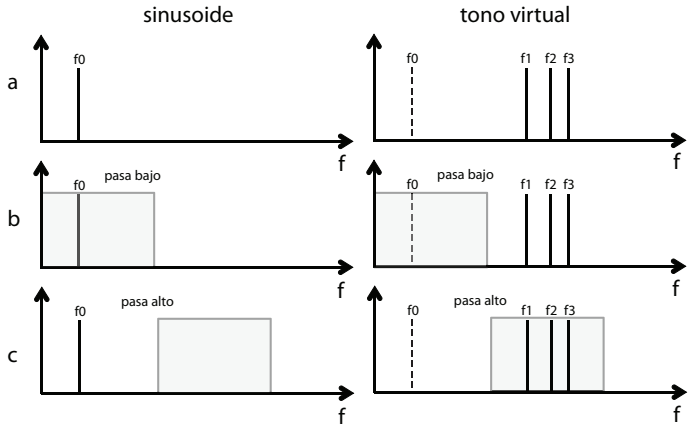


Figura 3.19: Demostración de la existencia del tono virtual

El fenómeno de la altura virtual por lo general sólo se presenta para frecuencias bajo 1000 Hz. Para frecuencias sobre éste valor, la altura sólo se percibe cuando la fundamental correspondiente a esa altura está efectivamente presente.

La sensación de altura también depende de otros factores, tales como la intensidad y la duración. En términos de la intensidad, la percepción de una determinada depende de la intensidad del estímulo, tal como se muestra en la figura 3.20. Cambios en la intensidad de un sonido pueden generar pequeños cambios en la altura percibida.

La altura también está condicionada por la duración de los estímulos. Para sonidos de bajas frecuencias la duración mínima necesaria para percibir esa altura es considerablemente mayor que para tonos de frecuencias más altas. Dependiendo de la frecuencia fundamental, si la duración del estímulo es muy corta el sistema auditivo no es capaz de generar la percepción de una altura determinada. Esto se ejemplifica en la figura 3.21, donde se muestra la percepción de la altura en función de la duración.

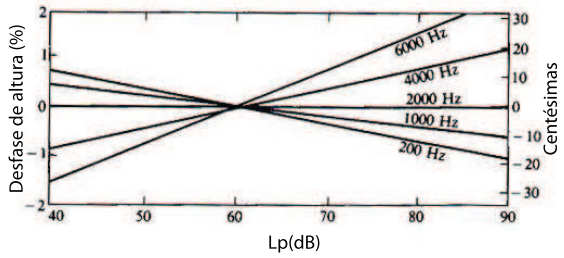


Figura 3.20: Altura e intensidad

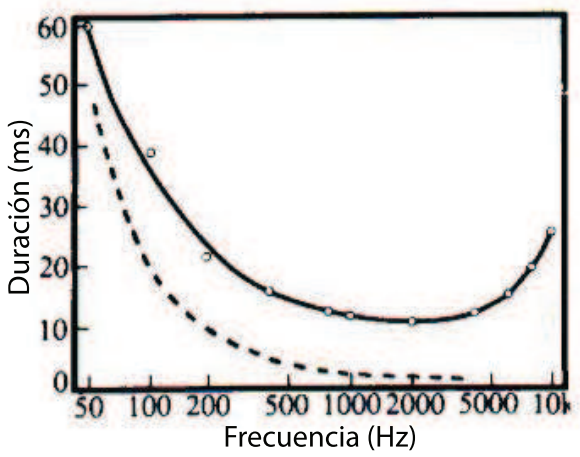


Figura 3.21: Altura y duración

3.5.1. Escala de mels

Al igual como sucede con la intensidad perceptual y la escala de fonos y sonos, es útil el contar con una escala perceptual de alturas, que pueda representar de manera más fidedigna nuestra percepción de la periodicidad de un sonido. Esta escala se conoce como la escala mel, y fue propuesta por Stevens, Volkman y Newmann en 1937. El nombre mel deriva de melodía, como una forma de explicitar que se trata de una escala basada en comparaciones entre alturas.

Esta escala se construye equiparando un tono de 1000 Hz y a 40 dBs por encima del umbral de audición del oyente, con un tono de 1000 mels.

Sobre los 500 Hz, los intervalos de frecuencia espaciados exponencialmente son percibidos como si estuvieran espaciados linealmente. En consecuencia, sobre este punto, cuatro octavas en la escala lineal de frecuencias medida en Hz se comprimen a alrededor de dos octavas en la escala mel.

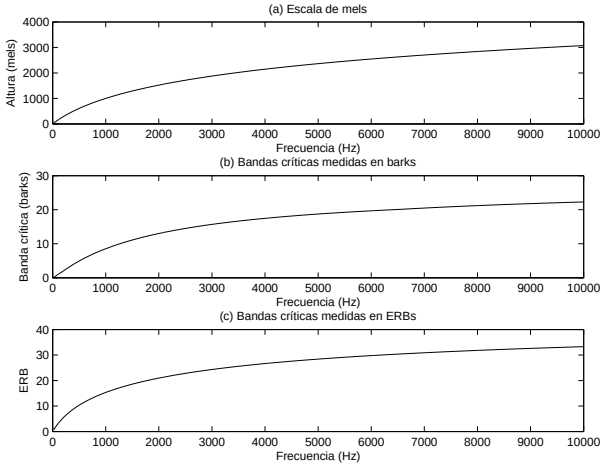


Figura 3.22: Altura y duración

La figura 3.22 muestra la escala mel en comparación con la escala de Barks y la escala ERB, discutidas en la sección 3.3.1. Claramente se aprecia su no linearidad respecto a la frecuencia.

La relación entre mels (m) y Hz (f) es la siguiente:

$$m = 1127{,}01048 \ln(1 + \frac{f}{700}) \tag{3.5}$$

y por lo tanto:

$$f = 700(e^{\frac{m}{1127{,}01048}} - 1) \tag{3.6}$$

3.6. Timbre

¿Qué es el timbre? ¿Cómo podemos definirlo? ¿Cómo se caracteriza? ¿Cómo se mide? ¿Cómo se representa o anota musicalmente?. El timbre, en claro contraste con la altura y la intensidad perceptual, sigue siendo un atributo auditivo mal entendido. Personas que tratan de entender se puede confundir tanto por su naturaleza como su definición. De hecho, el timbre

es un atributo raro y multidimensional del sonido, que se define justamente por lo que no es: no es ni la altura, ni la intensidad, ni la duración.

En 1885, Helmholtz propone una definición de timbre limitada al estado de reposo de un sonido. De acuerdo a su interpretación, el timbre está basado principalmente en la estructura armónica de sus componentes de frecuencia. En 1934, Fletcher lo define como esa característica de la sensación que permite al oyente escuchar el tipo de instrumento que emite el sonido. En 1938, Seashore lo liga al término "Sonoridad", que es la calidad del tono cuando el la altura y la intensidad percibida varían. Más aún, el comité de estándares ANSI define el timbre como "el atributo de la sensación auditiva en términos de la cual un auditor puede juzgar dos sonidos que presentados en forma similar, teniendo la misma altura e intensidad percpetual, como no similares".

Según Donnadieu, la ausencia de una definición satisfactoria del timbre se debe principalmente a dos grandes problemas [12]. El primero se refiere a la naturaleza multidimensional del timbre. De hecho, no es posible medir el timbre en forma unidimensional, en contraste con al altura (baja o alta), la duración (corta o larga), o la intensidad (suave o fuerte). El vocabulario empleado para describir los distintos timbres de los sonidos producidos por instrumentos musicales indica su aspecto multidimensional. Por ejemplo, "brillante", "opaco", "apagado" o "claro" son términos frecuentemente utilizados para describir los sonidos musicales. El segundo problema está relacionado con el hecho de que el timbre es un concepto que hace referencia a diferentes niveles de análisis. Schaeffer señaló que se puede hablar de .[el] timbre de un sonido sin atribuirlo directamente a un determinado instrumento, sino más bien como una característica intrínseca del sonido, percibida per se. Habrían, por lo tanto, dos conceptos de timbre: uno relacionado con el instrumento, la indicación de la fuente que nos es dada por la simple escucha, y la otra en relación con cada uno de los objetos proporcionados por el instrumento y el reconocimiento de los efectos musicales en ellos. El concepto de timbre, es entonces, mucho más general, que la simple habilidad de reconocer instrumentos.

Donnadieu enfatiza que el gran problema es que un sólo término se refiere a muchas nociones diferentes. El timbre puede describirse entonces en distintos términos:

1. Un conjunto de sonidos de un instrumento y también de la especificidad de cada sonido del timbre de un instrumento en particular

2. Un sonido aislado

3. Una combinación de diferentes instrumentos

4. La composición de una compleja estructura sonora

5. En el caso de timbres producidos por el análisis/resíntesis, sonidos híbridos o quimeras, sonidos nunca antes escuchados, que se no se pueden asociar con una fuente natural conocida

El concepto de timbre está muy ligado al problema de la fuente, tan común en la música electroacústica y tratado en detalle en la sección 7.1.1, ya que el timbre transmite la identidad de una fuente de sonido. En otras palabras, el timbre de un sonido complejo se compone de la información pertinente para la identificación de fuentes de ruido o eventos, incluso en un contexto musical.

Hay dos enfoques o teorías prinicipales sobre el timbre. El primer modelo es el procesamiento de la información, que describe las dimensiones de percepción de timbre resumen, en términos de los atributos de los sonidos. En otras palabras, los parámetros acústicos (espectrales, temporales, y espectrotemporal) de la señal sonora son procesadas por el sistema sensorial, perceptivo y el resultado es el timbre de los sonidos complejos. El escalamiento multidimensional o MDS ha sido fructífero en la determinación de estas diferentes dimensiones de percepción de timbre.

El segundo enfoque, basado en la teoría ecológica propuesta por Gibson, sólo recientemente ha dado lugar a la experimentación sistemática en la percepción auditiva. Según este punto de vista, el timbre de la percepción es una función directa de las propiedades físicas del objeto sonoro. El objetivo de estos estudios es describir los parámetros físicos que son perceptualmente de interés para el objeto que vibra.

Es importante destacar que el timbre no es un fenómeno estático. Por el contrario, el espectro de un sonido cambia contínuamente en el tiempo. Al momento de crear sonidos a través de la síntesis aditiva, detallada en la sección 5.4, es muy importante generar trayectorias dinámicas en el tiempo para cada parcial. Al momento de analizar sonidos en términos de su timbre es importante estudiar qué tan rápido decae la energía en los armónicos superiores, el número de armónicos presentes en la señal y que frecuencias son enfatizadas a medida que pasa el tiempo, entre otras cosas.

La figura 3.23 muestra parte del espectro de un sonido de trompeta. Es posible observar en la figura como cada parcial sigue una trayectoria distinta en el tiempo. Esto demuestra que el timbre no es algo estático y dependiende únicamente de la configuración o patrón de armónicos, sino que está muy ligado a la evolución temporal de la información que conlleva.

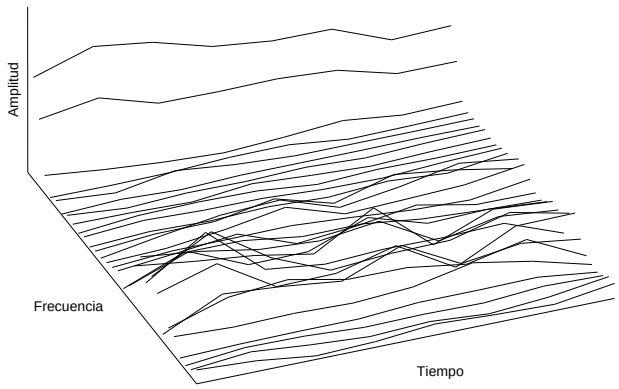


Figura 3.23: Trayectoria de armónicos en la trompeta

3.7. Consonancia y disonancia

Los fenómenos de la consonancia y disonancia han despertado el interés de científicos y estudiosos hace mucho tiempo. Galileo Galilei formulo una teoría que afirma que "las consonancias son pares de tonos que llegan al oído con una cierta regularidad; esta regularidad consiste en el hecho de que pulsos entregados por los dos tonos, en el mismo intervalo de tiempo, deben ser conmensurables en número, de manera de no mantener el oído en un tormento perceptual". En 1877 Helmholtz explicó la consonancia mediante la ley acústica de Ohm al afirmar que la altura que corresponde a cierta frecuencia sólo se puede escuchar si la onda acustica contiene energía en esa frecuencia, algo que sabemos hoy que no es cierto (ver sección 3.5). Helmholtz propuso que la disonancia aparece dado los abatimientos (beatings en inglés) que se producen entre tonos adyacentes en tonos complejos y calculó que la disonancia máxima aparecería entre dos tonos puros cuando la razón de abatimiento es de unos 35 Hz.

Existen varios enfoques distintos sobre las causas de la consonancia. El primer enfoque se basa en las razones entre frecuencias de los distintos intervalos, los que se muestran en la tabla 3.7. Según esta idea, la consonancia se forma simplemente de acuerdo a razones entre frecuencias. Las

2:1	Octava
3:2	Quinta justa
4:3	Cuarta justa
5:3	Sexta mayor
5:4	Tercera mayor
8:5	Sexta menor
6:5	Tercera menor

Cuadro 3.2: Razones entre intervalos musicales

teorías acústicas explican el fenómeno basadas en las propiedades físicas de la señal acústica, una de las cuales es la razón de frecuencias. Otras teorías de carácter psicoacústico se basan en principios perceptuales del sistema auditivo humano, tal como la influencia de la membrana basilar. Las teorías cognitivas se fundamentan en fenómenos cognitivos de nivel superior, tal como la categorización de intervalos musicales mediante entrenamiento. Por último, las teorías culturales dan cuenta de normas sociales, culturales o estilísticas que son internalizadas por los auditores.

Estudios bastante más recientes han permitido una mayor comprensión del fenómeno y relacionarlo con el concepto de banda crítica. En 1965, Plomp y Levelt llegaron a la conclusión que la consonancia o disonancia depende más de la diferencia entre frecuencias que de sus razones. Si la diferencia de frecuencia entre dos tonos es mayor que una banda crítica, suenan consonantes, si es menor, suenan disonantes. Esto se conoce como disonancia tonotópica. Kameoka y Kuriyagowa también en 1965 encontraron que la máxima disonancia ocurre cuando la diferencia de frecuencia es aproximadamente un cuarto de la banda crítica.

Según Boomsliter y Creel (1961) la consonancia aparece cuando los disparos neuronales del sistema auditivo están sincronizados, lo que se denomina teoría de sincronía de las descargas neuronales. Según Irvine (1946), la consonancia y disonancia están relacionadas con el largo del período de un ciclo. Por ejemplo, cuando dos tonos están relacionados por razones de frecuencia simples, el ciclo de repetición de la señal combinada es relativamente corto. Cuando la razón de frecuencia entre los tonos no están relacionados por razones sencillas, el ciclo de repetición es largo.

Stumpf (1890) argumenta que la fusión tonal es la base para la consonancia, entendiéndose por fusión a la propensión de dos o más tonos de fusionarse perceptualmente y sonar como un sólo tono complejo. Las mayores consoancias se producen cuando los tonos tienden a sonar como uno

sólo.

Se han determinado distintos tipos de disonancia de acuerdo al contexto donde ésta se inserta. Por ejemplo, la disonancia de resolución de altura aparece cuando el tiempo que toma resolver alturas de una señal completa es relativamente largo. Resnick (1981). En la disonancia de categoría de intervalo, la disonancia aparece cuando dos pitches forman un intervalo que es categóricamente ambiguo para un auditor. Esto es, cuando el intervalo está cerca de una categoría interválica aprendida. La disonancia de pitch absoluto es el componente de la disonancia que aparece cuando un pitch es ambiguo para un auditor que posee oído absoluto. En el caso de la disonancia de flujo incoherente, la disonancia aparece debido a la confusión que pueden causar distintos flujos de eventos musicales. (Wright and Bregman, 1987). La disonancia temporal es el componente de la disonancia que aparece por la rápida fluctuación de amplitud o abatimientos. La disonancia de altura virtual aparece por la compentencia entre alturas virtuales no resueltas (Terhardt, 1974).

Hay un último tipo de disonancia que es importante destactar denominada disonancia de expectación. Consiste en el componente de la disonancia que aparece debido al retraso de expectativas aprendidas. Existen tres niveles para este tipo de disonancia. (Cazden, 1980) El primer nivel se denomina tono disonante, y ocurre cuando un tono no acordal o no armónico tiene una tendencia de resolver dentro del marco de un acorde o armonía subyacente. En un segundo nivel, un momento disonante de acorde es cuando un acorde puede ser disonante hasta el extremo de generar una expectativa de resolución hacia otro acorde o progresión armónica. Por último, la disonancia de centro tonal se manifiesta cuando un pasaje musical retiene un cierto centro tonal dominante, y la disonancia aparece cuando al área dominante finalmente se mueve hacia el área original.

3.8. Codificación perceptual de audio

La codificación perceptual de audio consiste, de modo general, en un método para reducir la cantidad requerida de datos para representar una señal de audio digital. Este método inevitablemente genera pérdidas en términos de calidad, introduciendo una cierta cantidad de ruido que podría perfectamente caer dentro del rango de la audición humana si se analiza en forma aislada. Sin embargo, la codificación perceptual está diseñada de tal manera que el ruido generado por el codificador cae fuera de los límites de audición humana en presencia de la señal original. Esta distinción es muy

importante, ya que los algoritmos de compresión basados en percepción, a diferencia de los esquemas puramente numéricos o algorítmicos, tales como μ-law o ADPCM, se aprovechan de las limitaciones del sistema auditivo humano.

La idea fundamental de la codificación perceptual de audio es que la presencia de ciertos estímulos auditivos pueden influenciar la habilidad del cerebro humano para percibir otros estímulos. En palabras más simples, este tipo de algoritmos se basa fuertementente en le fenómeno de enmascaramiento, descrito en detalle en la sección 3.3.2. Un codificador perceptual por lo tanto, no codifica aquellos componentes de la señal de audio que se verán enmascarados por otros, ahorrando de esta manera una considerable cantidad de datos perceptualmente redundantes e innecesarios.

Hoy día existen numerosos esquemas de compresión basados en esta premisa, siendo el más conocido el MPEG-1, capa 3, comúnmene conocido como mp3, pero existen muchos otros tales como MPEG layers 1-3, AAC, OGG, Microsoft's Windows Media Audio, Lucent's PAC, ATRAC (utilizado en los minidiscs) y Real Audio.

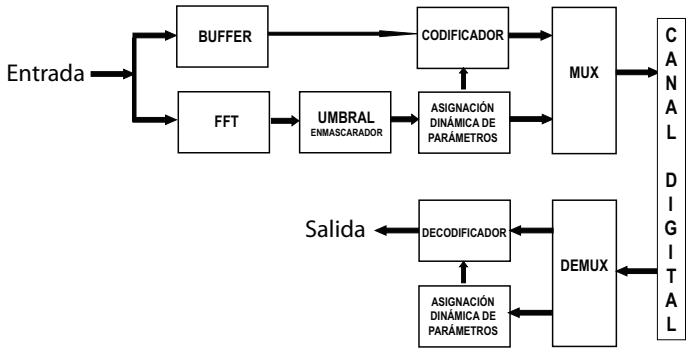


Figura 3.24: Esquema de un codificador perceptual de audio

La figura 3.24 muestra los componentes principales de una cadena de codificación perceptual. En el codificador, la señal de entrada se descompone en múltiples bandas de frecuencia. De esta forma, los datos de cada banda se pueden procesar de manera independiente y cada banda puede ser representada con un grado variable de resolución.

La idea es asignar una menor resolución en aquellas bandas de frecuencia que pueden ser representadas con una menor cantidad de información,

debido principalmente al enmascaramiento. Cuando la resolución se reduce en alguna banda en particular, crece el ruido de cuantización, detallado en la sección 4.1.2, en esa zona de frecuencias. La idea es cambiar el nivel de cuantización de esa banda de manera de satisfacer la tasa de bits objetivo manteniendo la mayor cantidad de detalles posible. El codificador está constantemente analizando la señal de entrada y toma decisiones acerca de que zonas del espectro se ven emascaradas y por lo tanto, al ser inaudibles, pueden descartarse de la señal y así disminuir la resolución.

Para decodificar se aplica una transformada inversa de manera de combinar las bandas y restaurar la señal original. En el caso en que la resolución de una banda no se vea reducida, el proceso es ideal y sin pérdida.

La efectividad de un codificador perceptual depende de que tan bien puede modelar las limitaciones perceptuales del sistema auditivo humano, pero también depende de si dispone de ancho de banda necesario para contener todo el detalle sonoro que los seres humanos somos capaces de percibir.

Capítulo 4

Audio digital

La música computacional, tal como su nombre lo indica, usualmente se hace mediante un computador, ya sea mediante la síntesis o el procesamiento de una *señal de audio digital*, que consiste básicamente en una secuencia discreta de números. Una señal digital tiene características especiales que la hacen muy diferente de una señal analógica, como por ejemplo una onda de presión acústica en el mundo físico. En consecuencia es importante estudiar los principios básicos de operación de un computador y la naturaleza de las señales digitales.

4.1. Análogo versus digital

Una señal continua o analógica es una que posee valores en cada instante de tiempo. En contraste, una señal discreta o digital sólo posee valores en ciertos instantes de tiempo, distanciados a intervalos regulares. Una señal discreta no está definida para valores intermedios de tiempo. Esto se visualiza claramente en la figura 4.1.

Una señal continua es convertida en una señal digital mediante un proceso denominado digitalización , el cual a su vez es un proceso que depende de dos operaciones: muestreo (sampling en inglés), que corresponde a tomar muestras de la señal en el tiempo y cuantización, que consiste en aproximar valores continuos de la amplitud a un grupo de valores manejables en el computador. Este proceso se visualiza en la figura 4.2. Una señal continua (a) es primero muestreada en el tiempo para producir una versión de tiempo discreto (b). Luego, su amplitud es cuantizada, lo que implica aproximar su amplitud a un conjunto discreto de valores, proceso que se muestra en (c). Bajo ciertas condiciones, es posible reconstruir desde (c) la señal original, la

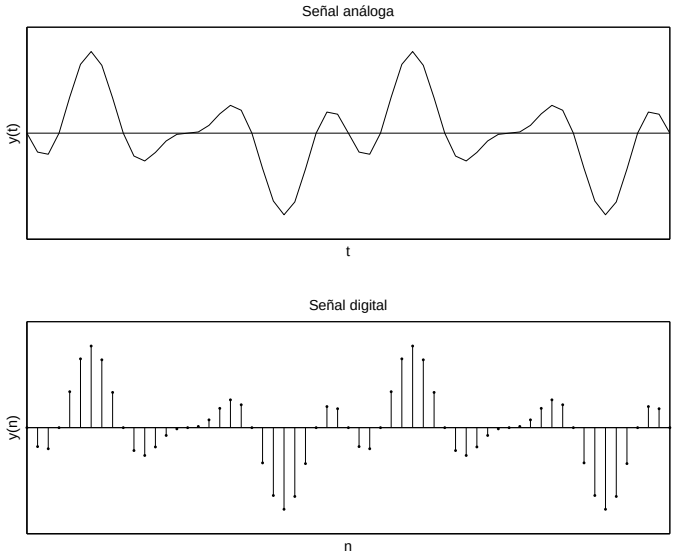


Figura 4.1: Señal análoga versus digital

que se muestra en (d).

Una señal digital consiste en una secuencia de numeros,

$$...x_{n-2}, x_{n-1}, x_n, x_{n+1}, x_{n+2}... \tag{4.1}$$

donde n corresponde al número de muestra y sólo toma valores enteros. Una señal sinusoidal digital se escribe como:

$$x_n = A\cos(\omega n + \phi) \tag{4.2}$$

donde A es la amplitud, ω es la frecuencia angular y ϕ la fase inicial.

Al ser n meramente un índice que indica el número de muestra, las señales digitales de audio no tienen una dependencia intrínseca con el tiempo. Para poder escuchar un sonido digital es necesario escoger una tasa de muestreo, usualmente denotada f_s, la cual índica cuantas muestras hay por unidad de tiempo, típicamente en un segundo. Esto es, $f_s t = n$. Por lo tanto, una sinusoide con velocidad angular ω tiene una frecuencia temporal dada por

$$f = \frac{\omega f_s}{2\pi} \tag{4.3}$$

Al tratarse una señal digital de una secuencia de números, por lo general para analizar cualquier característica de ella es necesario escoger una ventana

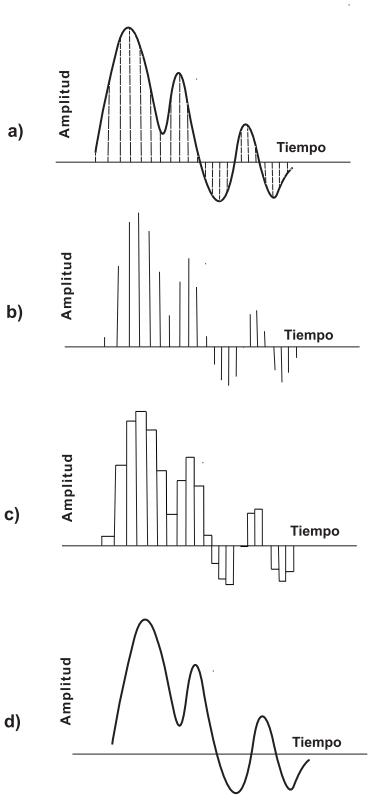


Figura 4.2: Digitalización de una señal análoga

de análisis. Esto es una ventana que comienza en una muestra M arbitraria y con un largo de N muestras:

$$x_M, x_{M+1}, ..., x_{M+N-1} \tag{4.4}$$

Hay muchas formas de elegir una ventana apropiada. El tipo de ventana indice directamente en lo que se denomina spectral leaking, que podría traducirse como derramado espectral, como se observa en el espectro de frecuencias de cada ventana. Una ventana ideal debiera tener un derramado los más angosto y atenuado posible. La ventana más simple es una de tipo rectangular, en la cual la secuencia digital se multiplica por una función rectangular pero presenta problemas sobre todo en los bordes, dado que introduce un cambio muy brusco en la señal, lo que provoca que la señal se desparrame en la frecuencia. Hay muchas otras formas de ventanas que

pueden resultar más apropiadas para ciertas operaciones, y tratan de suavizar el efecto en los bordes, en la figura 4.3 se muestran tres de las más utilizadas: rectangular (a), hamming (b) y Hann (c). Claramente, la ventana Hann presenta un mejor derramado, respecto a las otras dos.

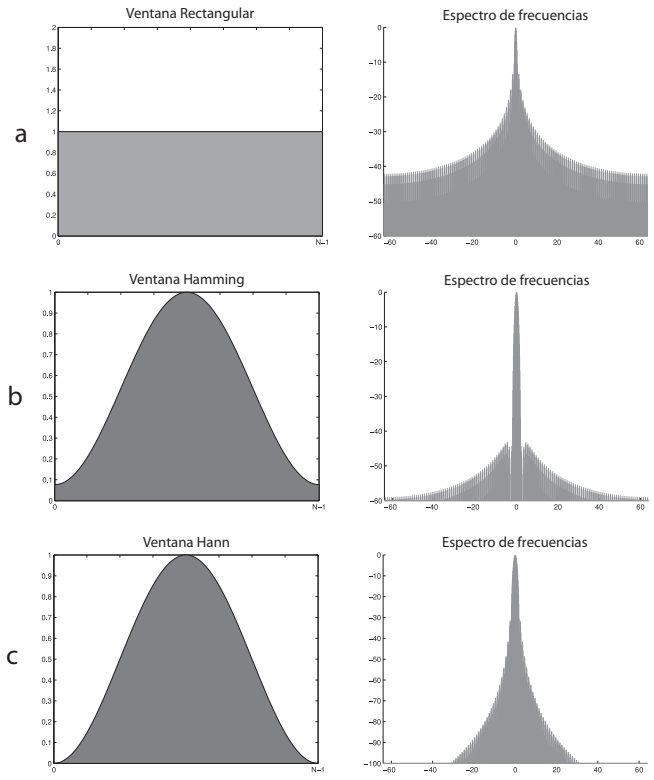


Figura 4.3: Algunas funciones de ventanas

4.1.1. Muestreo

Este proceso consiste en tomar muestras de la señal continua cada cierto tiempo a intervalos constantes. La frecuencia a la cual se toman dichas muestra se conoce como tasa de muestreo. En una forma más precisa, el muestreo consiste en multiplicar una señal análoga por un tren de impulsos unitarios discretos, tal como se muestra en la figura 4.4. La separación del los impulsos en el dominio del tiempo se conoce como período de muestreo y la separación en el dominio de la frecuencia la tasa de muestreo.

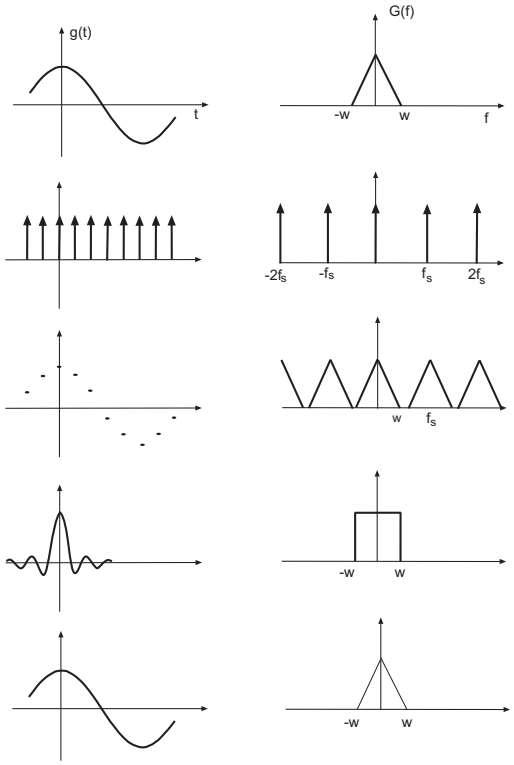


Figura 4.4: Muestreo de una señal digital

Tal como se aprecia en la figura, la señal muestreada consiste sólo en muestras de la señal original. A primera impresión, en el dominio del tiempo, da la sensación de que se ha perdido información, pero en realidad no es así. Esto es bastante más fácil de entender en el dominio de la frecuencia.

Teorema del muestreo

La figura 4.4 muestra el proceso de muestro en los dominios del tiempo y de la frecuencia. En el tiempo, al muestrear se multiplica la señal por un tren de impulsos. En la frecuencia lo que sucede es que se convoluciona el espectro de la señal original con otro tren de impulsos, cuya separación entre impulsos sucesivos es inversamente proporcional al período de muestreo. Esta operación genera infinitas réplicas del espectro original, separadas entre sí de acuerdo a la tasa de muestreo.

Para reconstruir la señal original desde la señal muestreada, en el dominio

de la frecuencia basta con filtrar la señal por un filtro pasa bajos con una frecuencia de corte tal que permita sólo aislar el espectro original. Dado que al filtrar el espectro original se obtiene en forma intacta, no hay pérdida de información alguna. En el dominio del tiempo, este filtraje equivale a convolucionar con la transformada de Fourier de una función rectangular, es decir una función sinc, ambas funciones definidas en la sección 1.3. Es por esto que para reconstruir una señal muestreada en su forma analógica, es necesario realizar una interpolación sinc.

Es por lo tanto, clave tener una separación suficiente entre las réplicas de manera que no interfieran unas con otras y pueda recuperarse el espectro original de la señal muestreada. Por simple inspección es claro que la tasa mínima de muestreo debe ser al menos el doble de la máxima frecuencia presente en el espectro original. Esta condición se conoce como el teorema del muestreo. Al muestrear una señal, la máxima frecuencia permitida en el espectro de la señal original se conoce como frecuencia de Nyquist.

Aliasión

Si la tasa de muestreo no es suficiente, las réplicas del espectro se traslaparán entre sí y no será posible reconstruir el espectro original al filtrar con el pasa bajos. Este fenómeno se conoce como aliasión, tal como se puede observar en la figura 4.5.

En términos sonoros, el fenómeno de la aliasión es claramente audible ya que se introducen frecuencias en el espectro correspondientes a la diferencia entre la frecuencia de muestreo y la máxima frecuencia de la señal original. Esto es claro de observar en el dominio del tiempo, como lo muestra la figura 4.6. Al reconstruir la señal mediante la interpolación sinc, se obtiene otra con una frecuencia bastante menor. La figura 4.6 muestra el caso de una señal submuestreada (con aliasión) versus una señal sobremuestreada, donde la tasa de muestreo es mayor a la máxima frecuencia de la señal original. Claramente la señal reconstruida en el caso submuestrado se encuentra distorsionada respecto a la original.

4.1.2. Cuantización

Una señal digital no solamente es discretizada en el tiempo, sino también en su amplitud. Esto se debe a que el computador, por su naturaleza discreta, no es capaz de representar ni manipular una función continua de amplitud. El proceso por el cual se discretiza la amplitud de una sonido en el computador se denomina cuantización.

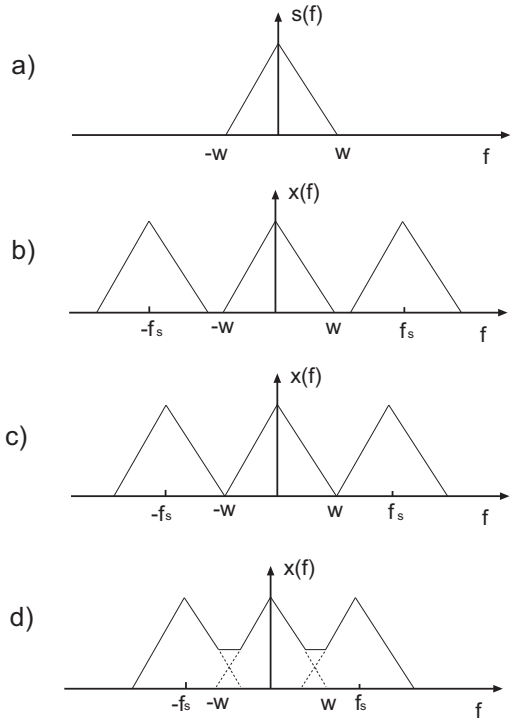


Figura 4.5: Aliasión en el dominio de la frecuencia

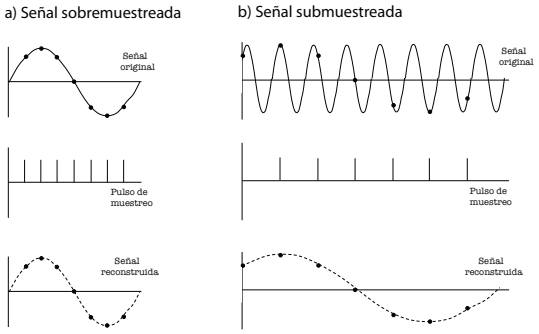


Figura 4.6: Aliasión en el dominio del tiempo

La cuantización es un proceso claramente no lineal, como se muestra en la figura 4.7. Esto implica que genera distorsiones o errores no lineales. La figura 4.8 muestra el proceso de la cuantización de una señal analógica.

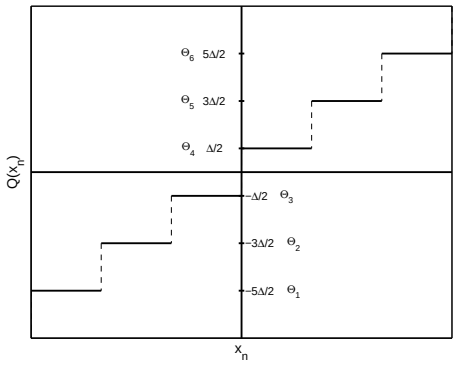


Figura 4.7: Proceso de cuantización

La cuantización se encarga de otorgarle a un rango de la señal una única salida. La diferencia que resulta de restar la señal de entrada a la de salida es el error de cuantización, esto es, la medida en la que ha sido necesario cambiar el valor de una muestra para igualarlo a su nivel de cuantización más próximo.

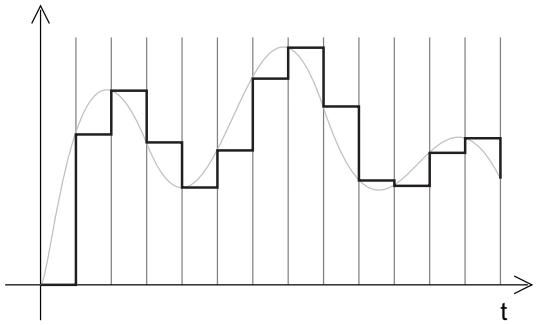


Figura 4.8: Cuantización de una señal analógica

Los valores continuos de la señal son aproximados a 2^n niveles de amplitud cuantizados donde n corresponde al número de bits disponible, tal como se explica en la sección 4.2. Esto depende de cada sistema. La resolución de la señal por ende tendrá relación con el numero de niveles que se tenga para codificar. En el caso del compact disc o CD, se utilizan 16 bits para representar la amplitud. Esto significa que hay $2^{16} = 65536$ niveles distintos para representar la amplitud. Esto claramente induce un error en la señal

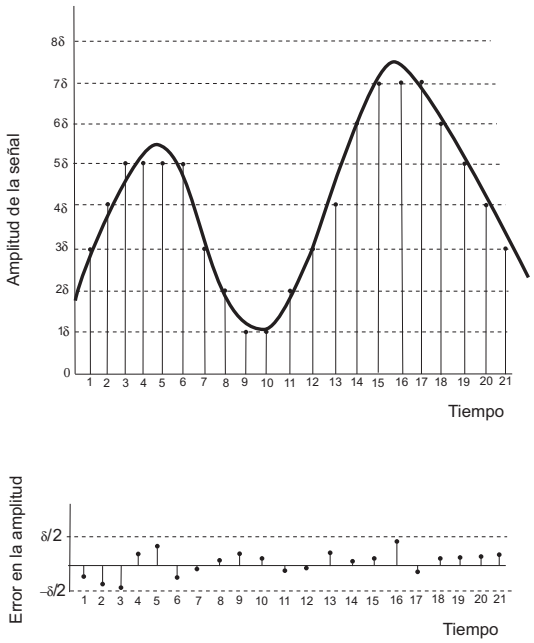


Figura 4.9: Error de cuantización

cuantizada, a diferencia de lo que sucede con el muestreo, donde es posible reconstruir la señal original si se muestra a una tasa adecuada.

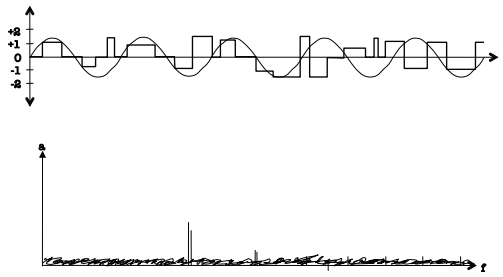


Figura 4.10: Dithering

El error de cuantización puede ser audible dependiendo del contexto, por lo cual no es un tema menor. Una forma de minimizar el efecto de este ruido es mediante la adición de una señal aleatoria llamada dither,, tal

como se muestra en la figura 4.10. Al aplicar esta señal aleatoria, se logra desestructurar el ruido de cuantización y convertirlo en ruido uniforme, el cual es bastante menos audible que el ruido estructurado.

4.2. Sistema binario

La unidad básica de información que un computador puede reconocer es un *bit*. Un bit representa el estado o posición de un switch dentro del hardware del computador y solamente puede asumir dos estados: encendido o apagado (electrónicamente esto es equivalente a decir que por un cierto punto de un circuito pasa o no pasa corriente eléctrica). Normalmente el estado de encendido se representa por un 1 y el apagado por un 0. Un sistema de información de este tipo, en el cual sólo hay dos estados posibles, se conoce como *sistema binario*. .

La cantidad de información que puede ser representada por un sistema binario depende del número de bits disponible. Por ejemplo, si sólo se cuenta con 2 bits, las únicas posibilidades distintas de información son: 00, 01, 10 y 11. En general, n bits pueden representar 2^n estados distintos.

Hexadecimal (base 16)	Decimal (base 10)	Octal (base 8)	Binario (base 2)
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

Cuadro 4.1: Representación numérica en distintos sistemas

Por lo general, los bits son agrupados en *bytes*, que corresponden normalmente a agrupaciones de ocho bits cada una. En consecuencia, un byte puede asumir $2^8 = 256$ estados distintos. La tabla 4.1 muestra los números del cero al quince representados en distintos sistemas: hexadecimal, decimal, octal y binario. Estos sistemas difieren en la base que utilizan.

4.3. El computador

Un computador es básicamente una máquina que realiza dos funciones básicas: ejecuta en forma muy rápida una secuencia de instrucciones (*un programa*) y almacena y recupera grandes cantidades de información (*datos*). De acuerdo a esto, un computador puede caracterizarse por su velocidad de operación (medida en *Hertz* o ciclos por segundo), el tipo de instrucciones que puede ejecutar (esto determina el *tipo* de computador) y la capacidad de su memoria (medida en *bytes*).

El término *hardware* se refiere a la circuitería electrónica que posee un determinado computador y *software* a los programas que dicho computador puede ejecutar. Dentro del hardware están, entre otras cosas, el procesador, la memoria RAM, el disco duro, la placa madre, la tarjeta de sonido, el monitor y el lector de CD. Incluídos en el software están el *sistema operativo* y programas tales como procesadores de texto, planillas de cálculo, agendas y juegos. El sistema operativo es un conjunto de programas que permiten que el hardware funcione correctamente y que el usuario cree otros programas para usos más específicos.

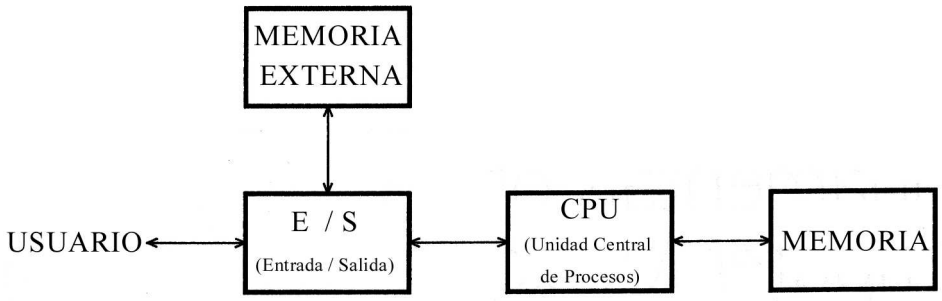


Figura 4.11: Estructura básica de un computador

En la figura numero 4.11 se puede apreciar la estructura básica de un computador. En ella se aprecian cuatro unidades fundamentales: memoria

externa, entrada/salida, unidad central de procesos (CPU) y memoria interna. La *Unidad Central de Procesos* (*CPU* en inglés), es el sistema central del computador. Controla la operación del sistema leyendo e interpretando instrucciones para el flujo de la información. Estas instrucciones pueden también tomar decisiones basadas en los datos que son almacenados en el computador o en dispositivos externos. Es esta propiedad de tomar decisiones la que diferencia a un computador de, por ejemplo, una simple calculadora.

La *memoria* guarda tanto programas (instrucciones) como datos. Está dividida en distintas particiones en las cuales la información es almacenada. Cada partición de memoria está unívocamente asociada a un número de dirección. La CPU utiliza este número tanto para almacenar como para leer la información contenida en una partición determinada.

Cuando el computador es encendido, el harware interno le da a la CPU la dirección de memoria que contiene la primera instrucción de programa. La ejecución del programa comienza en este momento. Las instrucciones siguentes son obtenidas en forma secuencial a partir de las sucesivas direcciones de memoria a menos que el programa indique a la CPU que hay que hacer un salto a alguna dirección de memoria distinta.

Mientras la CPU está ejecutando algo, el programa puede hacer referencias a direcciones de memoria distintas a las que contienen las instrucciones del programa ya sea para almacenar datos o para ejecutar otras instrucciones.

El programa almacenado en memoria le otorga al computador la flexibilidad de ejecutar una gran cantidad de tareas diversas, porque el programa puede ser cambiado fácilmente cambiando sólo las direcciones de memoria. Esto no siempre fue así. Antiguamente los programas y los datos no compartían la misma memoria, por lo tanto los cambios en los programas eran dificultosos porque muchas veces implicaban también cambio en el hardware. Pero a fines de 1940, John von Neumann llevó a cabo la idea de utilizar la misma memoria tanto para los datos como para los programas. Este nueva concepción fue todo un hito en la historia de la computación, pues permitió que los programas fueran tratados como datos. De esta manera, los programas pueden utilizarse para escribir otros programas, lo que facilita la labor del programador. Actualmente, el noventa por ciento de los computadores modernos utiliza la arquitectura de von Neumann.

Existen dos tipos básicos de memoria: ROM y RAM. La memoria ROM (Read Only Memory) es una memoria que no puede alterada durante el proceso normal de ejecución de la CPU. Por lo general, las instrucciones de inicialización de un computador están contenidas en memoria de este tipo,

de manera de garantizar que el computador siempre se reinicie de la misma manera. La memoria RAM (Random Access Memory) es una memoria volátil, en el sentido de que puede ser leída y escrita por la CPU y una vez que el computador se apaga, su contenido es borrado. La cantidad de memoria RAM que un posee un computador determina el tamaño de los programas que éste puede ejecutar.

Una instrucción de CPU puede utilizar uno o más bytes. La mayoría de los computadores utiliza instrucciones de largo variable: de esta manera se optimiza el uso de la memoria, dado que instrucciones más simples necesitan menos memoria que las que son más complejas.

Para que el trabajo de la CPU sea útil, es necesario que ésta tenga acceso a información externa y tenga los mecanismos necesarios para reportar el resultado de sus cálculos. El sistema en Entrada/Salida (Input/Output en inglés, o simplemente I/O), permite a la CPU interactuar con una gran cantidad de dispositivos externos. Ejemplos de dispositivos de entrada son el teclado y el mouse y de salida el monitor o una impresora que permiten al usuario interactuar adecuadamente con el computador.

Un ejemplo de este tipo de dispositivo es el *disco duro*, que constituye la memoria externa del computador. Este disco almacena datos y programas que no están siendo utilizados por la CPU en un determinado momento y tiene la capacidad de guardar su contenido en el tiempo, es decir, su contenido no desaparece una vez que el computador es apagado. Otro ejemplo es el CD-ROM, dispositivo que permite guardar información en un disco removible. Por lo general, este tipo de dispositivos tiene una capacidad de almacenamiento bastante mayor que la memoria RAM y pueden llegan al orden de 20 o más Gigabytes (mil millones de bytes).

Todas las instrucciones que ejecuta la CPU están sincronizadas por un reloj interno que oscila con una frecuencia del orden de los 1000 MHz en el caso de los computadores más modernos. Este sincronismo es necesario para que, por ejemplo, la CPU no trate de leer y escribir en una misma dirección de memoria al mismo tiempo. La velocidad de oscilación de este reloj se conoce como la velocidad de operación del computador y determina el número de instrucciones por segundo que la CPU puede ejecutar aunque no necesariamente en una relación uno a uno.

4.4. Programación de computadores

En general un computador puede ser programado para realizar cualquier tarea siempre y cuando el hardware y el sistema operativo lo permitan. El

programador debe determinar un procedimiento o *algoritmo* que puede ser escrito como una secuencia de instrucciones que permiten realizar una tarea determinada. Cada paso del algoritmo debe ser especificado sin ambigüedades y debe haber un camino (trazo) claro para que el algoritmo pueda ser completado.

Un algoritmo puede describir, por ejemplo, el proceso de ejecución de una obra musical por parte de un pianista. El algoritmo debe describir claramente cada paso que el músico debe seguir al ejecutar cada una de las notas de la pieza. En este caso, el algoritmo debiera ser el siguiente:

1. Encontrar el inicio de la pieza.
2. Leer la primera nota.
3. Ir al paso 5.
4. Leer la siguiente nota.
5. Apretar la tecla apropiada en el piano.
6. Contar la duración de la nota y soltar la tecla.
7. Era esta la última nota? Si la respuesta es si, entonces ir al paso 8. Si la respuesta es no, entonces ir al paso 4.
8. Fin del algoritmo.

4.5. Sistema operativo y sistema de archivos

Dado que un computador por lo general tiene acceso a una gran cantidad de programas de distinta naturaleza, es necesario contar con algún grupo de programas que se encarguen de la operación básica del computador y permitan la correcta operación de otros programas. Esto es lo que se conoce como *sistema operativo*. Un sistema operativo no es más que una colección de programas que prestan servicios útiles para la operación del computador. Por ejemplo, un sistema operativo contiene el software necesario para la comunicación de la CPU con los dispositivos de Entrada/Salida. También provee el orden necesario para que la CPU pueda cambiar de un programa a otro en forma correcta.

Por lo general, los sistemas operativos más simples (como DOS o Windows 3.1) solamente pueden ejecutar un sólo programa a la vez. Esto es porque la CPU está exclusivamente dedicada a ejecutar un programa y no

Sistema Operativo	Fabricante	Multitasking	Multiusuario
DOS	Microsoft	NO	NO
Windows 3.0	Microsoft	NO	NO
Windows 95/98	Microsoft	SI	NO
Windows NT	Microsoft	SI	SI
MacOS	Apple	SI	NO
Linux	No tiene (gratis)	SI	SI
SunOS / Solaris	Sun Microsystems	SI	SI
AIX	IBM	SI	SI
IRIX	Silicon Graphics	SI	SI
HP-UX	Hewlett Packard	SI	SI

Figura 4.12: Sistemas Operativos más utilizados

abandona la ejecución hasta que el programa termine o hasta que reciba una interrupción de la ejecución. Pero los sistemas operativos más avanzados (como UNIX) son multitarea (multitasking, en inglés) y permiten que varios programas se ejecuten en forma simultánea y además que hayan varios usuarios conectados al sistema al mismo tiempo.

En la tabla 4.5 se pueden apreciar los sistemas operativos más utilizados y alguna de sus características.

El dramático aumento en la capacidad de almacenamiento de dispositivos tales como el disco duro o el CDROM obliga al sistema operativo a ser eficiente en el proceso de almacenar y recuperar grandes cantidades de bytes en un medio físico cada vez más pequeño. Por lo tanto, el sistema operativo debe disponer de un *sistema de archivos* adecuado.

El elemento básico de almacenamiento de información reconocido por los sistemas operativos modernos es el *archivo*. Un archivo (file en inglés), puede contener programas o datos indistintamente. El largo de un archivo está determinado solamente por su contenido. Para ejecutar un programa que está contenido en una memoria externa, la CPU carga en la memoria principal el archivo apropiado y ejecuta la primera instrucción que éste contenga. Los archivos de datos pueden contener cualquier tipo de información.

Para manejar las grandes cantidades de información de un sistema computacional, los archivos se agrupan en *directorios* o carpetas (folder en inglés). Estos directorios pueden contener archivos y también otros directorios, que se llaman *subdirectorios*. Para referenciar un directorio determinado y no confundirlo con otros directorios, los sistemas operativos utilizan ciertos caracteres delimitadores. Por ejemplo, si se trata de un sistema operativo Windows, la expresión `\Windows\Escritorio\Carpeta1`, da cuenta de

tres directorios distintos, uno llamado `Windows`, que contiene a su vez un subdirectorio llamado `Escritorio` y que a su vez contiene otro subdirectorio llamado `Carpeta1`. En el caso de un sistema Macintosh esta expresión sería `Windows:Escritorio:Carpeta1` y en el caso de un sistema Unix `Windows/Escritorio/Carpeta1`. Lo único que cambia es el caracter delimitador.

4.6. Uso del computador en sistemas de audio

Un computador puede generar y manipular sonidos de varias formas diferentes. Sin embargo, el sonido debe estar representado en un formato que el computador pueda procesar. Dado que los computadores son binarios, es necesario *digitalizar* el audio para que el computador pueda leer la información sonora. En otras palabras, un computador sólo puede procesar *audio digital*. Esto no significa que el computador no pueda interactuar con audio análogo. La diferencia entre audio análogo y digital es abordada en forma detallada en la sección 4.1.

La figura 4.13 muestra las tres formas más usuales del uso del computador en sistemas de audio.

El caso a) corresponde al proceso de *grabación* o *procesamiento digital*. El audio análogo es captado por un transductor, por ejemplo un micrófono, que convierte las la variaciones de presión del aire (energía acústica) en una señal eléctrica, que consiste básicamente en variaciones de voltaje. Esta señal es luego filtrada por un filtro pasabajos con el fin de eliminar componentes de frecuencia no deseados y eliminar la posibilidad de aliasión, explicada en la sección 4.1.1.

Una vez filtrada la señal, ésta es digitalizada (sampleada o muestreada) por un conversor análogo/digital. La salida de este conversor contiene audio digital, que puede ser procesado o guardado en memoria externa por el computador. Un ejemplo de procesamiento de la señal podría ser aplicar reverberación al sonido original o mezclarlo con otro previamente almacenado en memoria externa.

Una vez realizado el proceso, el audio es introducido esta vez en un conversor digital/análogo y pasado por un filtro pasabajos. La salida de este filtro produce audio análogo que podría ser amplificado y lanzado nuevamente al aire mediante parlantes.

La grabación digital tiene varias ventajas por sobre su similar análoga. Dado que la información digitalizada contiene sólo números y no una señal análoga, tiene una calidad superior ya que no se deteriora con el tiempo

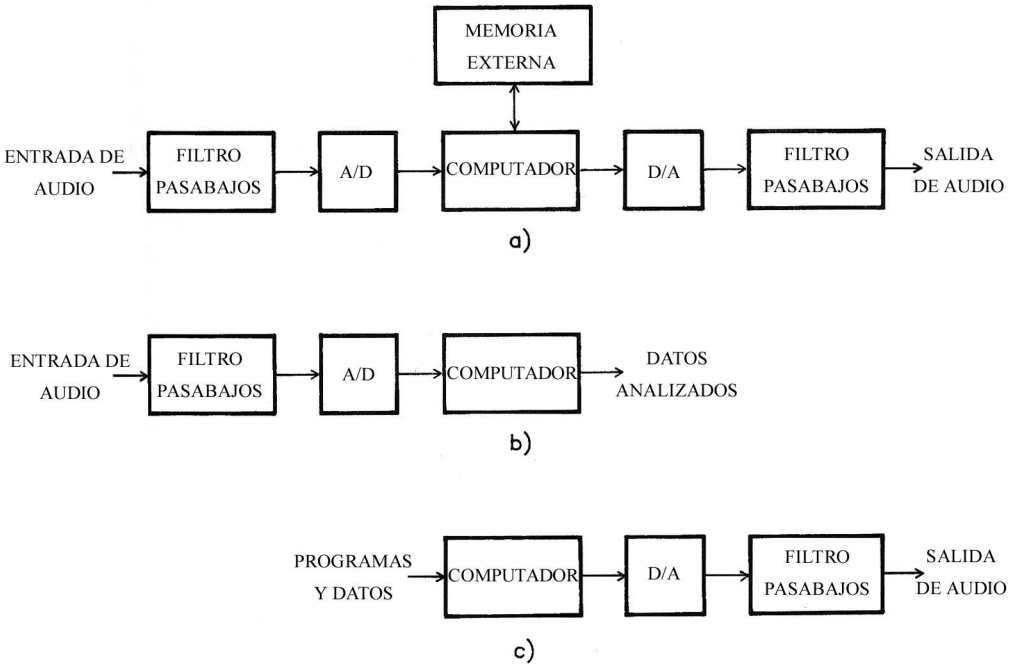


Figura 4.13: Sistemas computacionales de audio

ni depende de variables como la temperatura ambiente, presión atmosférica, viscosidad del aire o el ruido ambiental. Además, una copia digital del original digital es exactamente idéntica y fiel.

El caso b) corresponde al análisis de señales. Cuando el computador opera como un analizador de señales, el computador toma una señal digital y matemáticamente determina sus características. Por ejemplo, un análisis computacional puede revelar información importante sobre las propiedades acústicas de un determinado sonido.

El computador también puede *sintetizar* sonido mediante el cálculo de secuencias de números que corresponden a la forma de onda del sonido deseado como salida. El proceso de utilizar hardware digital para generar sonidos mediante el cálculo de su forma de onda se conoce como *síntesis digital directa*. Este proceso tiene varias ventajas respecto a la síntesis análoga de sonidos, dado que las señales digitales dentro de un computador son extremadamente precisas, por lo tanto se pueden representar numéricamente en forma exacta y reproducir en el tiempo con gran exactitud.

Otra ventaja es la repetibilidad. Un sistema digital siempre va a producir

exactamente la misma salida para los mismos datos de entrada, cosa que no sucede en los sistemas análogos.

Pero la ventaja más importante es la generalidad. Dado que cualquier sonido puede ser convertido en una señal digital, consecuentemente cualquier sonido puede ser sintetizado utilizando algoritmos apropiados para calcular su representación digital. En teoría, los sonidos sintetizados digitalmente pueden ser arbitrariamente complejos y tener cualquier carácter. Uno de los desafíos más importantes de la síntesis digital es encontrar los algoritmos adecuados para producir determinados sonidos. Muchos modelos de síntesis se basan en las características naturales de los sonidos, pero otros buscan modificar sus características para ampliar los sonidos originales y darle propiedades no naturales.

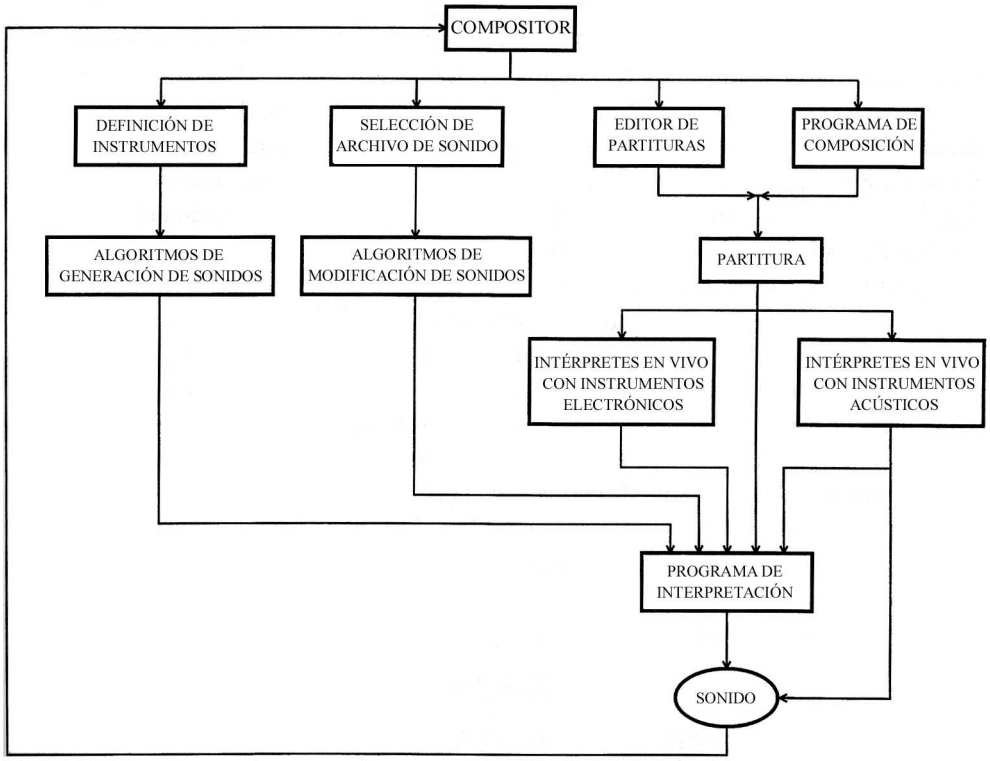


Figura 4.14: Uso del computador por parte de los compositores

La figura 4.14 muestra alguna de las formas en que los compositores

hacen uso de los sistemas computacionales.

4.7. Software para música computacional

La mayoría de los software de música computacional caen en una de las siguientes categorías: algoritmos para síntesis de sonidos, algoritmos para modificación de sonidos sintetizados o sampleados, programas que asisten al músico en la composición ya sea con sonidos sintetizados o sonidos acústicos de instrumentos tradicionales y programas que permiten una interpretación composicional de una obra. Esta división es sólo un modelo que responde a actividad musical tradicional, pero no es una verdad absoluta y pueden haber enfoques diferentes.

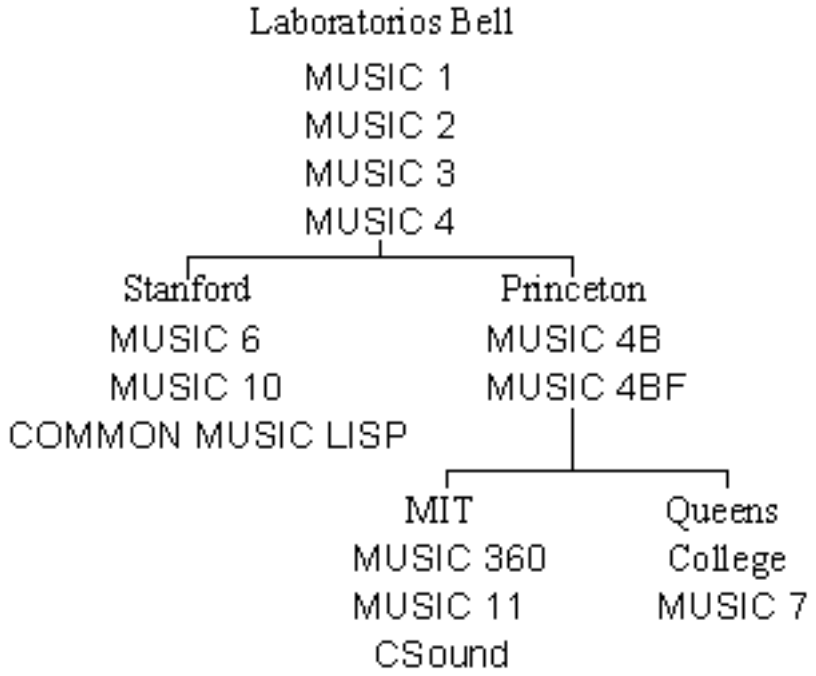


Figura 4.15: Familia de los programas de síntesis de sonidos

El primer software para síntesis de sonidos fue el `Music 3`, creado por Max Matthews en los Laboratorios Bell en la primera mitad de los años sesenta. Su sucesor, `Music 4`, fue distribuido entre distintas universidades, donde la actividad musical era más intensa.

Síntesis directa
Csound
Cmix
Cmusic
Chant
Common Music Lisp
UPIC
Granular Synthesis

Procesadores de partituras y programas de composición
Score 11
Common Music
HMSL
Finale

Modificación digital de sonidos
Sound Hack
Protools (Digidesign)
Dyaxis
Sound Forge (Sonic Solutions)
RT

Control en tiempo real
MAX
jMax
Vision
Studio Vision
Cypher
HMSL

Figura 4.16: Softwares de música más utilizados en los años noventa

La figura 4.15 muestra una serie de programas que han evolucionado en distintos lugares a partir de los originales `Music 1 2 3` y `4`. Allí se aprecia que en Princeton Godfrey Winham y Hubert Howe hicieron variaciones de la versión de `Music 4` y más tarde en el Massachusetts Institute of Technology (MIT) se llevaron a cabo las versiones `Music 360` y `Music 11` por Barry Vercoe, quien años más tarde daría luz a la primera versión de `CSound`. Por otra parte, en Stanford University se programaron las variaciones `Music 6` y `Music 10` y mucho tiempo después se creó `Common Music Lisp`, un lenguaje de síntesis basado en inteligencia artificial con una raíz absolutamente distinta.

Si bien la familia de lenguajes `Music-N` se encuentra actualmente obsoleta debido al avance de la tecnología, la mayoría de los programas de síntesis utilizados actualmente (`CSound`, `Cmusic`, `Cmix`) están basados en ella.

La figura 4.7 muestra algunos de los softwares de música más utilizados en los últimos años. Tal como se mencionó anteriormente, éstos pueden clasificarse de acuerdo a cuatro categorías.

4.8. MIDI

`MIDI` significa Musical Instruments Digital Interface y consiste en un protocolo digital de comunicaciones entre sintetizadores. Es decir, consiste solamente en instrucciones que los sintetizadores entienden y que pueden compartir entre si. **MIDI no es audio digital**.

En la misma forma en que dos computadores se comunican, dos sintetizadores pueden intercambiar información vía MIDI. Esta información es musical por naturaleza. La información MIDI le dice a un sintetizador, por ejemplo, cuando empezar a tocar una nota y cuando terminar. También puede indicar con que volumen se toca esa nota, o cuando cambiar sonidos o incluso puede contener información de hardware específica.

Cada comando MIDI tiene una secuencia de bytes específica. El primer byte es el byte de status, que le indica al sintetizador que función ejecutar. Incluído en este byte de status va el canal MIDI. MIDI opera con 16 canales de audio diferentes en forma simultánea, denominador por los números del 0 al 15.

Algunas de las funciones indicadas por el byte de status son Note On, Note Off, System Exclusive (SysEx) o Patch Change. Dependiendo del contenido del byte de status, un número determinado de bytes seguirán al byte de status hasta que se reciba un nuevo byte de status.

Los fabricantes de computadores pronto se dieron cuenta que el computador podría ser una fantástica herramienta para MIDI, dado que los dispositivos MIDI y los computadores hablan en el mismo lenguaje. Dado que la tasa de transmisión de los datos MIDI (31.5 kbps) no es la misma que utilizan los computadores, los fabricantes tuvieron que diseñar una interfaz especial que permitiera a los computadores comunicarse con cualquier sintetizador.

Virtualmente cualquier cosa que puede ser hecha vía MIDI posee software especializado en el mercado. En primer lugar, están los secuenciadores (`Cakewalk, Encore`) que simplemente ordenan eventos MIDI en listas y los envían en los tiempos apropiados a las distintos dispositivos externos o internos. Estos secuenciadores permiten grabar, almacenar, tocar y editar datos MIDI.

También existe una gran variedad de editores de sonido y librerías de sonidos que permiten flexibilizar los sonidos de un determinado sintetizador. Las librerías más avanzadas permiten editar y grabar bancos enteros de sonidos en las memorias de los sintetizadores.

Capítulo 5

Síntesis digital de sonidos

Una de las grandes ventajas que ofrece la tecnología digital es la posibilidad de diseñar sonidos con un alto nivel de precisión y flexibilidad. El computador permite crear sonidos directamente mediante algoritmos o manipulación numérica, sin la mediación del mundo acústico y sin la necesidad de partir sobre la base sonidos previamente grabados. Este proceso de diseño o creación númerica se denomina síntesis.

Desde los primeros experimentos de síntesis sonora a través de computadores, numerosas técnicas y algoritmos han sido propuestos en la literatura los cuales se basan en principios muy diversos e incluso en algunos casos bastante opuestos. Por lo tanto, es importante contar con criterios de clasificación que nos permitan agrupar las diferentes técnicas de acuerdo a sus caracerísticas comunes.

5.1. Clasificación

Un primer esquema básico de clasificación divide las distintas técnicas entre las que intentan reproducir un sonido existente y las que no. El primer grupo puede ser llamado "concreto" (no el sentido de la música concreta, sino en el de reproducir algún sonido ya existente) y el segundo "abstracto". Esta clasificación puede parecer confusa ya que hay técnicas de síntesis que no están basadas en ninguno de estos criterios y al mismo tiempo satisfacen ambos, como es el caso de la síntesis FM, la cual puede ser utilizada para simular bronces o campanas como también para producir sonidos electrónicos "abstractos" que responden a criterios matemáticos no necesariamente musicales. No obstante, esta primera clasificación resulta adecuada para discutir el problema concerniente a la fuente sonora, o el objeto sonoro según

Schaeffer, y nos provee de un claro acercamiento a estas técnicas (ver sección 7.2.4).

Otro criterio de clasificación muy utilizado divide las técnicas en tres areas principales: abstracta, modelos físicos y modelos espectrales. En el primer grupo se pueden incluir las técnicas de síntesis más clasicas que pretenden simplemente explorar algoritmos matemáticos para generar sonidos. Es así como en 1977, Moorer [38] ya propone que la forma natural de clasificación debiera considerar técnicas de síntesis aditiva, subtractiva y modulación (también llamada síntesis no lineal o de distorsión), y que en general son usadas en combinación. Esta clasificación es ratificada por Moore en su libro "Elements of Computer Music" [19]. Es importante aclarar que hoy en día la síntesis no lineal no se refiere exclusivamene a la modulación, sino que agrupa un gran numero de técnicas basadas en ecuaciones matemáticas con comportamiento no lineal [32].

En el segundo grupo se incluyen todas las técnicas que se basan en el estudio de las propiedades físicas de los instrumentos musicales u otras fuentes sonoras y su posterior implementación y simulación en el computador. Existen autores que sólo toman en cuenta estos dos primeros grupos para clasificación. Por ejemplo, Borin et al. sugiere que la síntesis algorítmica de sonidos se puede dividir en dos clases: Síntesis algorítmica directa clásica, la que incluye transformaciones de síntesis aditiva, granular, subtractiva y no lineal y las técnicas de modelos físicos, en la cual se encuentran la familia de métodos que modelan la acústica de los instrumentos [26].

El tercer grupo consiste en técnicas que pretenden modelar el comportamiento del sonido directamente en su espectro de frecuencias. Este tipo de técnicas de síntesis se basa fuertemente en el mundo perceptual de los sonidos y no en el mundo físico. Tal como se describe en detalle en el capítulo 3, el sistema auditivo humano puede modelarse como un analizador de frecuencias, de forma muy similar a un analizador de Fourier (ver sección 1.2). Es por esto que este tipo de técnicas son más apropiadas para sintetizar sonidos naturales que instrumentales y usualmente requieren de una etapa previa de análisis, antes de la síntesis.

Existen otros esquemas de clasificación bastante divergentes de este modelo tripartito. En 1983, De Poli [22] sugirió la siguiente clasificación para las síntesis algorítmicas: técnicas de generación, las cuales producen una señal de datos entregados, y técnicas de transformación que pueden ser divididas en dos sub-clasificaciones, la generación de una o mas señales simples y sus modificaciones. De Poli también sugiere que habitualmente se utilizan combinaciones de estas técnicas.

En el último tiempo, se han propuesto clasificaciones más detalladas

y sin grupos tan divisorios o fuertemente delimitados. Es así como en su famoso libro "The Computer Music Tutorial" [27], Curtis Roads propone la siguiente clasificación para la síntesis de sonidos:

- Sampling y síntesis aditiva.
- Síntesis mediante multiples tablas de ondas, wave terrain, síntesis granular, y subtractiva.
- Síntesis por modulación : Ring, AM, FM, distorsión de fase y forma de onda.
- Síntesis mediante modelos físicos.
- Síntesis por segmento de ondas, síntesis gráfica y estocástica.

En cambio, Dodge and Jerse [11] usan el siguiente esquema de clasificación:

- Síntesis utilizando técnicas no perturbativas: aditiva, AM, modulación de anillo.
- Síntesis utilizando técnicas pertubativas: FM, waveshaping, síntesis usando formulas aditivas discretas.
- Síntesis subtractiva.
- Síntesis para el análisis de datos : voz, STFT, phase vocoder, wavelets.
- Síntesis granular.
- Modelos físicos.

En 1991, Smith propone una clasificación mas detallada dividiendo los métodos de síntesis en cuatro grupos: algoritmos abstractos, grabación procesada, modelos espectrales y modelos físicos. Propone además la siguiente "taxonomía" para las técnicas de síntesis digital [36]:

De acuerdo con Serra, una forma de entender la clasificación de Smith es hablar de los métodos de síntesis como técnicas digitales que nos permiten obtener una sonoridad continua que va desde la reproducción de sonidos pre-existentes (grabados) hasta la generación de sonidos mediante una abstracción (sonidos imaginados), pasando por todos los pasos intermedios. En este contexto, técnicas basadas en el procesamiento de grabaciones tienen como fundamento la pre-existencia de sonidos e intentan crear sonidos nuevos o imaginarios. En el otro extremo de la clasificación están los algoritmos

Procesado	**Modelos espectrales**	**Modelos físicos**	**Algoritmos abstractos**
Concreta	Tabla de Onda F	Cuerdas de Ruiz	VCO, VCA, VCF
Tabla de Onda T	Aditiva	Karplus-Strong Ext.	Music V
Muestreo Vector	Phase Vocoder PARSHL	Ondas guías Modal	FM FM con retroalimentación
Granular	Sinosoides + Ruido (Serra)	Cordis-Anima	Formateo de onda
Componentes principales T	Componentes principales F	Mosaico	Distorsión de fase
Wavelet T	Chant		Karplus-Strong
	VOSIM		
	FM de Risset (bronces)		
	FM de Chowning (voz)		
	Subtractiva		
	LPC		
	FFT inversa		
	Clusters de líneas de Xenakis		

Cuadro 5.1: Taxonomía de las técnicas de síntesis digital de sonidos

abstractos, que de ecuaciones matemáticas generan sonidos que distan de los sonidos naturales, pero que manipulándolos se obtienen sonidos que nos permiten una comunicación musical especifica, como por ejemplo, los sintetizadores basados en técnicas de FM. Los modelos espectrales y físicos están en la zona intermedia y tienen su fundamento en modelos o abstracciones que describen sonidos pre-existentes y los objetos que lo generan. Esto nos permite la exploración de la conexión entre el sonido real y el sonido virtual [32].

En 2004, Bank et al. [2] dividen las técnicas de síntesis de audio en tres grupos, unificando dos grupos de la clasificación propuesta por Smith. El

primer grupo es la familia de métodos abstractos. Estos son diferentes algoritmos que generan fácilmente sonidos sintéticos. Métodos tales como FM y waveshaping pertenecen a esta categoría. El modelar instrumentos reales con este método es en extremo complejo si la relación entre los parámetros de la técnica y aquellos de los instrumentos reales no pueden ser determinados con facilidad.

El segundo grupo, denominado modelo de señales, es el que modela los sonidos de los instrumentos musicales. En este caso, la entrada al modelo es solo la forma de onda o un conjunto de ellas generadas por el instrumento y la física no es tratada en detalle. Los métodos de síntesis tales como sampling y SMS (spectral modeling synthesis) pertenecen a esta categoría. Este grupo corresponde al procesamiento de sonidos pre-grabados en la taxonomía de Smith.

El tercer grupo, denominado como modelamiento físico, es el intenta reproducir el comportamiento físico de un instrumento o fuente sonora. Usualmente, los sistemas físicos (tales como una cuerda de un instrumento o la membrana de un tambor) pueden ser descritos resolviendo una ecuación de onda. Dada una excitación inicial (tal como tocar la cuerda o golpear el tambor) y estableciendo las condiciones de contorno para el problema en particular, las ecuaciones pueden ser resueltas y utilizadas como entradas al sistema por lo que la salida se espera sea cercana al sonido original. Un método conocido en esta categoría es la síntesis digital con ondas guías o waveguides, que modela eficientemente ondas unidimensionales.

5.2. Evaluación

Dado el gran número de técnicas de síntesis que han sido propuestas en la literatura, surge la legitima inquietud de como evaluar o compara cada una de estas aproximaciones. Como Jaffe dice “Un gran numero de técnicas de síntesis y procesamiento se han inventado. La pregunta es cual es la mejor” [16]. Lamentablemente no hay una respuesta simple a eso y la mejor técnica depende de las prioridades de quien la utiliza y del problema a tratar.

Serra [32] menciona cuatro consideraciones que se debiesen tener en cuenta al escoger una técnica de síntesis especifica. Estas son:

1. Calidad del sonido. Con esto queremos expresar la riqueza del sonido. Un sonido de gran calidad será uno que se asemeje a un sonido natural, en cambio uno de baja calidad será un sonido simple, sintetizado electrónicamente sin micro-variaciones durante su duración.

2. Flexibilidad. Este término describe la habilidad que tiene una técnica de síntesis especifica para modificar el sonido desde el control de un conjunto reducido de parámetros. Bajo este criterio, un sampler no es considerado como flexible, en cambio que la síntesis FM si lo es.

3. Generalidad. Por generalidad se entiende la posibilidad de una técnica de síntesis de crear una gran variedad de timbres. Así, la síntesis aditiva sera una técnica general, en tanto que el sampler sera una técnica mucho más específica.

4. Tiempo de cómputo. Esto se refiere al número de instrucciones computacionales que se necesitan para sintetizar un sonido. En este contexto, la síntesis FM es muy económica en comparación con la síntesis aditiva, la cual resulta muy costosa.

Jaffe postuló 10 criterios específicos para evaluar métodos de síntesis de sonidos[16]. Formuló sus criterios en forma de las siguientes preguntas:

1. ¿Qué tan intuitivos son los parámetros?

2. ¿Qué tan perceptibles son los cambios en los parámetros?

3. ¿Qué tan físicos son los parámetros?

4. ¿Qué tan bien se comportan los parámetros?

5. ¿Qué tan robusta es la identidad sonora?

6. ¿Qué tan eficiente es el algoritmo?

7. ¿Qué tan escaso es el control sobre el stream?

8. ¿Qué clase de sonidos puede ser representado?

9. ¿Cual es la mas pequeña latencia?

10. ¿Existen herramientas de análisis?

Si bien es importante contestar estas preguntas para cada método nuevo de síntesis, la respuesta sobre cual método es el más adecuado depende en gran medida de su aplicación y de los recursos disponibles.

5.3. Fundamentos

5.3.1. Osciladores

Una oscilación de la variación periódica en el tiempo de alguna medida en torno a un valor central o de equilibrio o entre dos o más estados. Ejemplos de oscilaciones en el mundo físico son un péndulo o un resorte. En un computador, un oscilador puede implementarse en forma algorítmica, el requisito indispensable es que la señal se repita después de un cierto tiempo.

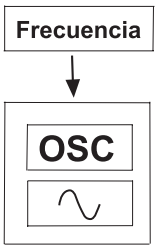


Figura 5.1: Oscilador

Para sintetizar sonidos en un computador es necesario disponer de señales básicas mediante las cuales se puedan constriur sonidos más complejos. Esto se hace a través de un oscilador. Un oscilador consiste básicamente en un algoritmo o dispositivo que es capaz generar señales periódicas o aleatorias. El oscilador más común es un oscilador sinusoidal, el cual genera señales sinusoidales con una amplitud, frecuencia y fase determinada. Un oscilador puede también generar distintos tipos de señales estocásticas o ruidos, tales como ruido blanco, ruido cafe o ruido rosa.

Por lo general, los osciladores generan funciones básicas, como las descritas en la figura 5.2. Esto es debido a un límite en el ancho de banda disponible dada la tasa de muestreo del sistema. Si la señal generada por el oscilador contiene componentes de frecuencia muy elevados, es probable que al generar el audio se produzca aliasión. Para evitar esto, es común implementar osciladores de ancho de banda limitada, los cuales evitan generar componentes de frecuencia más allá de un cierto límite.

En el apéndice A.3 se describen los principales osciladores existentes en SuperCollider, denominados UGens, abreviación de unidades generadoras.

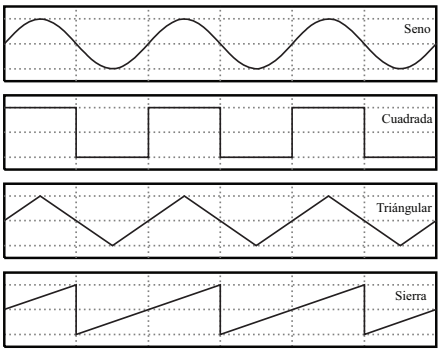


Figura 5.2: Señales básicas comúnmente utilizadas en osciladores

5.3.2. Tabla de ondas

Un oscilador por lo general lee los datos necesarios desde una tabla de ondas. Una tabla de ondas (wavetable en inglés) consiste básicamente en un pedazo de memoria donde se puede almacenar una señal de audio. La tabla puede contener audio generado en forma sintética o una señal proveniente del mundo real muestreada. Es común almacenar en una tabla de onda un ciclo de una onda de sonido cualquiera.

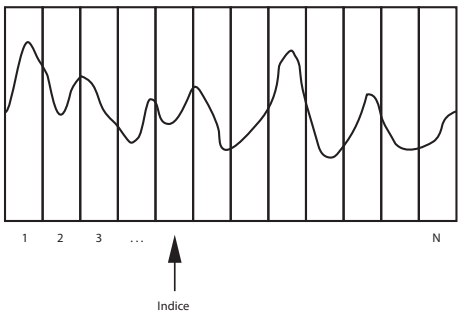


Figura 5.3: Tabla de ondas

Un oscilador puede entonces leer datos desde la tabla de onda a distintas frecuencias y con eso puede generar una onda periódica de frecuencia arbitraria cuya forma de onda se encuentra determinada por los datos almacenados en la tabla.

La figura 5.4 detalla el principio básico de funcionamiento de un oscila-

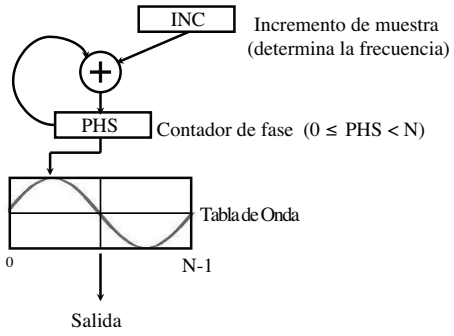


Figura 5.4: Algoritmo de un oscilador digital de sonidos

dor. Un oscilador tiene dos contadores: uno que lleva la posición en la tabla de ondas desde la cual se está leyendo, denominado contador de fase, y otro llamado incremento, que contiene la cantidad de muestras que deben saltarse para leer desde la siguiente posición de fase. La cantidad de incremento determina la frecuencia del sonido generado por el oscilador. Mientras mayor sea el incremento, la tabla de onda se completa más rápido y mayor es la frecuencia de la señal generada.

Dado que usualmente se especifica la frecuencia deseada y no directamente el valor del incremento, es necesario realizar algún tipo de interpolación

5.3.3. Envolventes

El sonido no es un fenómeno estático. Muy por el contrario, usualmente cada componente de frecuencia de un sonido tiene vida propia, en el sentido de que adquiere un comportamiento independiente (o casi independiente) en el tiempo. Esto significa que cada parcial recorre una trayectoria o envolvente particular en su evolución temporal. En el dominio del tiempo, el sonido como un todo también tiene un comportamiento definido por una envolvente de amplitud, tal como el mostrado en la figura 5.5.

La figura 5.6 muestra la forma de la envolvente de sonidos reales para distintas dinámicas. Claramente, la evolución temporal del sonido es distinta dependiendo del nivel de energía, definido por su contenido de frecuencias, presente en la señal acústica.

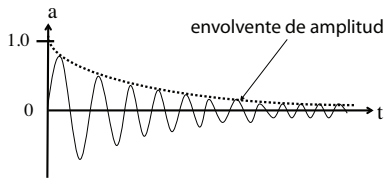


Figura 5.5: Envolvente de amplitud

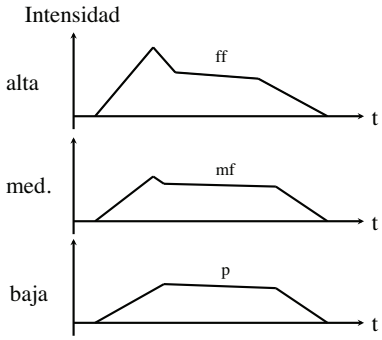


Figura 5.6: Envolventes para distintas dinámicas

Existen muchas formas de envolvente, pero por lo general siguen un patrón o evolución común. La forma más genérica de envolvente se denomina ADSR (ataque, decaimiento, sostenimiento, relajo) por su abreviatura y mostrada en la figura 5.7. A continuación se describe cada sección de esta envolvente.

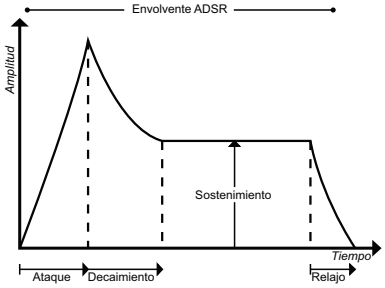


Figura 5.7: Envolvente ADSR

- Ataque. El ataque es el tiempo que toma la señal en alcanzar su máximo
- Decaimiento. El decamiento es el tiempo que toma la señal para estabilizarse
- Sostenimiento. Tiempo que dura la señal estable.
- Relajo. El relajo es el tiempo que toma la señal en desvanecerse.

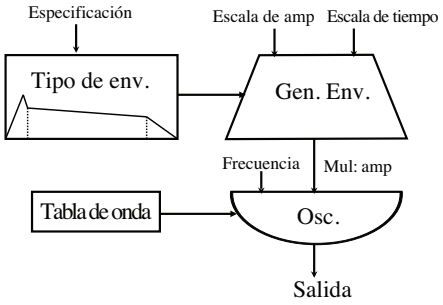


Figura 5.8: Proceso de generación de envolventes

Las envolventes usualmente se utilizan para controlar la salida de un oscilador, tal como se detalla en la figura 5.8. Una vez elegida la forma de la envolvente, esta se escala tanto en tiempo como en amplitud y se multiplica por la salida de un oscilador. De esta forma, es posible manipular el comportamiento temporal de la señal sintetizada. En el apéndice A.4 se describen las principales funciones de envolventes existentes en SuperCollider y su forma de utilización.

5.4. Síntesis aditiva

La sínteis aditiva consiste básicamente en la superposición o mezcla de ondas sinusoidales con el objetivo de generar ondas más complejas. En el mundo acústico, este es el principio de funcionamiento de un órgano de tubos. El concepto tras la síntesis aditiva es simplemente la aplicación directa de la serie de Fourier con un número finito de componentes. Si bien esta

técnica se basa en Fourier, no todo es necesario seguirlo al pie de la letra, ya que perfectamente en un computador se pueden sumar sinusoides no armónicas, es decir, señales cuyas frecuencias no estén relacionadas por múltiplos enteros. En este caso, los armónicos se denominan parciales. La figura 5.9 muestra la suma de 5 sinusoides de distinta amplitud, frecuencia y fase, con relaciones no armónicas entre si.

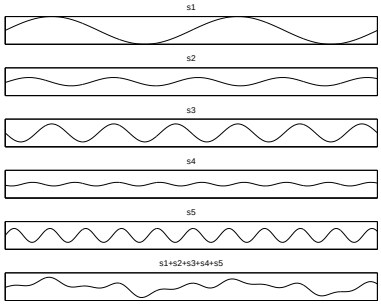


Figura 5.9: Suma de señales simples para generar una compleja

Un sonido generado mediante síntesis aditiva está formado entonces por una cantidad variable de armónicos o parciales que evolucionan a lo largo del tiempo con respecto a un tono o frecuencia fundamental. El número de parciales y su amplitud, frecuencia y fase relativa es lo que permite crear un sonido con un timbre determinado. Sin embargo, por lo general el timbre no es estático y cada parcial sigue una curva particular de evolución temporal, tal como se describe en la sección 3.6. Es por esto que en la síntesis aditiva es muy importante la utilización de diferentes envolventes que se encarguen del manejo la amplitud sobre cada parcial y es lo que estructura el comportamiento global del sonido en el tiempo.

Para realizar el proceso se hace necesario disponer de un banco de osciladores, cada uno con su amplitud, fase y frecuencia respectiva, además de su propia envolvente de amplitud, creándose un sonido dinámico y realista, tal como lo muestra la figura 5.10

5.5. Síntesis substractiva

En la síntesis substractiva se sintetiza el sonido mediante la filtración o simplificación de una onda compleja. La señal pasa a través de uno o más filtros que modifican su contenido armónico, atenuando o reforzando determinadas áreas del espectro de la señal, tal como se describe en la figura

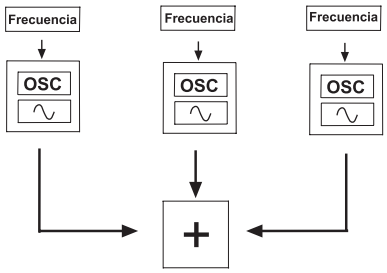


Figura 5.10: Síntesis aditiva

5.11. La síntesis sustractiva está basada de cierta manera en la idea inversa de la síntesis aditiva, ya que a partir de señales ricas en contenido de frecuencia la idea es eliminar ciertas bandas o componentes de manera de generar una señal simplificada. Es por esta razón que usualmente se utilizan señales aleatorias o de ruido como punto de partida ya que poseen un espectro de frecuencias muy rico. Detalles completos de la implementación y teoría de los filtros digitales se encuentra en la sección 6.1.

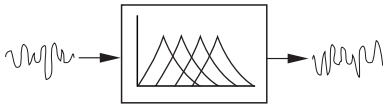


Figura 5.11: Síntesis substractiva

Los resultados de la síntesis substractiva dependen en gran medida de la calidad y diseño de los filtros utilizados, de manera de poder producir en forma efectiva los cambios requeridos en la señal de entrada. Las características de los filtros se determinan por su función de transferencia y su orden. La primera determina la forma en que la señal aplicada cambia en términos de su amplitud y fase al pasar por el filtro y la segunda describe el grado de precisión en la aceptación o rechazo de frecuencias por encima o por debajo de la respectiva frecuencia de corte.

Si se desea obtener más precisión en las frecuencias de corte o bien procesar la señal de distintas maneras, se pueden combinar filtros de primer o segundo orden en dos formas: serie y paralelo, tal como se muestra en la figura 5.12. En el primer caso, la salida de un filtro alimenta a otro filtro. Un ejemplo de esta configuración podría ser un sonido que primero es pasado por un filtro pasa bajos y luego reverberado mediante otro proceso. En el

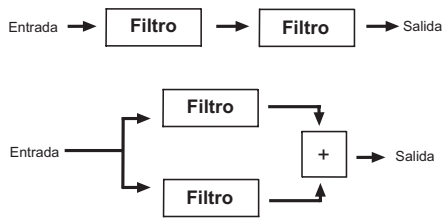


Figura 5.12: Configuración de filtros en la síntesis substractiva

caso de filtros en paralelo, la señal se ve modificada por uno o más filtros al mismo tiempo y las salidas de estos se suman para generar la señal modificada. Un ejemplo típico de esta configuración es un ecualizador multibanda, descrito en la sección 6.6.

5.6. Modulación

El concepto modulación se basa en la idea de alterar algún parámetro de una onda sonora en razón de otra onda. Las formas más comunes de modulación son modulación de amplitud o AM, que consiste en alterar la amplitud de una señal portadora en función de la amplitud de otra señal llamada moduladora y la modulación de frecuencia o FM, que consiste básicamente en variar la frecuencia de una portadora en función de otra señal moduladora.

La síntesis aditiva y las substractiva descritas anteriormente pueden ser consideradas como lineales en cuanto no afectan el contenido armónico de la señal. Simplemente se basan en operaciones lineales sobre la señal en entrada. En contraste, las técnicas de modulación suelen también llamarse de distorsión o no lineales, dado que generan componentes armónicos no posibles de obtener mediante operaciones lineales.

Modular una señal consiste en modificar alguna de las características de esa señal, llamada portadora, de acuerdo con las características de otra señal llamada modulador. Sea,

$$x_p(n) = A_p sin(\omega_p n) \tag{5.1}$$

la señal portadora o modulada y

$$x_m(n) = A_m sin(\omega_m n) \tag{5.2}$$

la señal modulante o moduladora.

5.6.1. Modulación Ring

En la modulación ring o anillo, ambas señales simplemente de multiplican, como lo muestran la figura 5.13 y la ecuación 5.3.

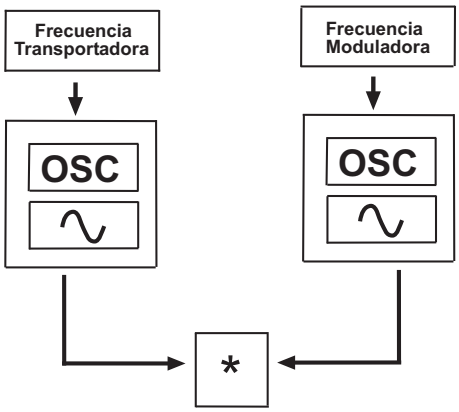


Figura 5.13: Modulación ring o anillo

$$x(n) = x_m(n)x_p(n) \tag{5.3}$$

El resultado que es en el espectro los componentes de frecuencia se reagrupan, generando dos componentes en $f_p + f_m$ y $f_p - f_m$, tal como se muestra en la figura 5.14. Este caso de modulación es un caso especial de la más general síntesis AM, que se detalla a continuación.

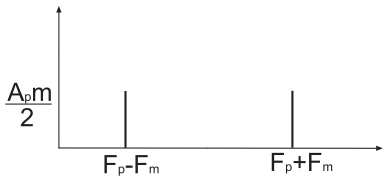


Figura 5.14: Espectro de la modulación ring

5.6.2. Síntesis AM

En la síntesis AM la señal modulada tendrá una amplitud igual al valor máximo de la señal portadora más el valor instantáneo de la señal modulada:

$$x(n) = (A_p + A_m sin(\omega_m n))sin(\omega_p n) \tag{5.4}$$

Factorizando, la ecuación anterior puede escribirse como:

$$x(n) = A_p(1 + \frac{A_m}{A_p} sin(\omega_m n))sin(\omega_p n) \tag{5.5}$$

donde $m = \frac{A_m}{A_p}$ es el índice de modulación.

$$x(n) = A_p(1 + msin(\omega_m n))sin(\omega_p n) \tag{5.6}$$

De acuerdo a los principios de la trigonometría la señal puede ser reescrita como

$$x(n) = A_p sin(\omega_p n) + A_p msin(\omega_m n)sin(\omega_p n) \tag{5.7}$$

y expandiendo se tiene que,

$$x(n) = A_p sin(\omega_p n) + \frac{A_p m}{2} cos((\omega_p + \omega_m)n) - \frac{A_p m}{2} cos((\omega_p - \omega_m)n) \tag{5.8}$$

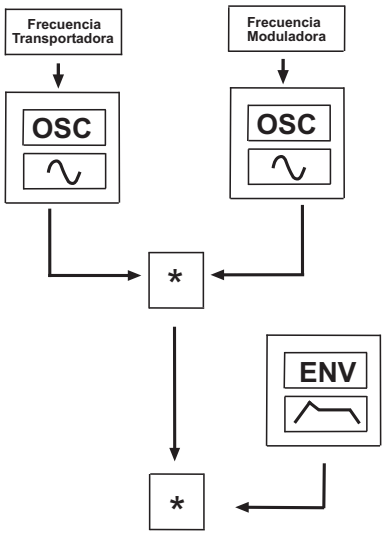


Figura 5.15: Síntesis AM

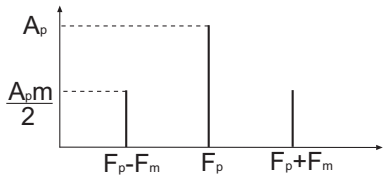


Figura 5.16: Espectro de la modulación AM

La ecuación 5.8 nos permite deducir que al modular la amplitud se generan dos armónicos laterales, tal como se muestra en la figura 5.16.

Cuando la señal moduladora sea una onda compleja y no sinusoidal como resultado de la modulación AM se obtendra un conjunto de bandas laterales dados por los armónicos que ésta contenga.

5.6.3. Síntesis FM

Otra forma de modular la señal es variar periódicamente la frecuencia mediante otra señal. Esto se denomina síntesis FM o de frecuencia modulada.

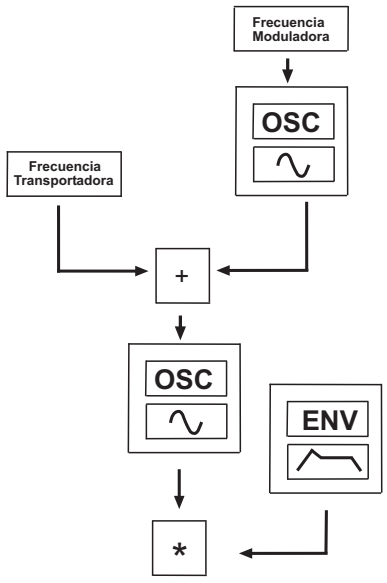


Figura 5.17: Síntesis FM

Matemáticamente, esto es:

$$x(n) = sin(\omega_p n + Asin(\omega_m n)n) \tag{5.9}$$

Esta expresión puede ser reescrita como una serie de potencia que expresa la serie espectral resultante:

$$x(n) = \sum J_i(A)[sin(\omega(f_t + nf_m)n + sin(\omega(f_t - nf_m)n)] \tag{5.10}$$

Donde los J_i son funciones de Bessel. Al analizar el espectro de frecuencias de una señal FM, observamos infinitas frecuencias laterales, espaciadas en f_m, alrededor de la frecuencia de la señal portadora f_p, tal como se observa en la figura 5.6.3. Sin embargo la mayor parte de las frecuencias laterales tienen poca amplitud, lo cual depende del índice de modulación.

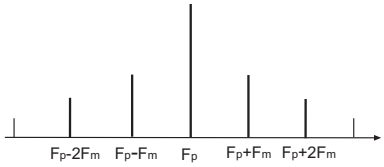


Figura 5.18: Espectro de la modulación FM

En el apéndice A.6 se aborda la modulación SuperCollider, a través de ejemplos tanto de modulación de amplitud como de frecuencia.

5.7. Síntesis granular

La síntesis de sonido basada en granos o síntesis granular es una técnica de producción de sonidos que se basa en una concepción del sonido en términos de partículas o cuantos, pequeñas unidades de energía encapsuladas en una envolvente y agrupados en conjuntos mayores, cuya organización será determinada por dos métodos principales de distribución temporal: sincrónico y asincrónico. En el método sincrónico, los granos son disparados a frecuencias más o menos regulares para producir sonidos con una altura definida. En contraste, el método asincrónico genera secuencias aleatorias de separación entre los granos con el objetivo de producir una nube sonora (Dodge & Jerse, 262).

La unidad mínima de la síntesis granular es el cuanto sonoro o grano. Estos son fragmentos de sonido muy cortos, cuya longitud oscila entre 5 y

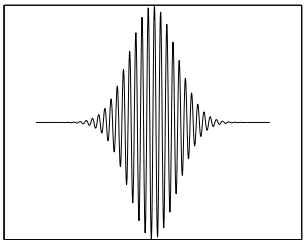


Figura 5.19: Grano sonoro sinusoidal

100 milisegundos para evitar que un grano individual pueda producir una respuesta perceptiva de altura en el oyente. La envolvente de los granos es determinada por una ventana usualmente de tipo gaussiana. La figura 5.19 muestra un grano basado en una onda sinusoidal con una envolvente de tipo gaussiana y la figura 5.20 un grano con la misma envolvente pero basado en ruido blanco. Las diferencias en la forma de la ventana y el contenido interno definen la cantidad de información espectral en el grano.

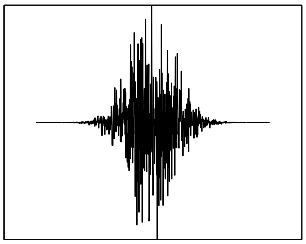


Figura 5.20: Grano sonoro de ruido blanco

El apéndice A.7 muestra ejemplos de síntesis granular realizados en SuperCollider.

5.8. Modelos físicos

La síntesis basada en modelos físicos se hace a partir de la simulación en una computadora de un objeto físico y sus características. Estos métodos de síntesis generan sonidos describiendo el comportamiento de los elementos, tal como lo son las cuerdas, cañas, labios, tubos, membranas y cavidades que conforman el instrumento. Todos estos elementos, estimulados mecáni-

camente, vibran y producen perturbaciones, por lo general periódicas en el medio que los rodea.

5.9. Modelos espectrales

Los modelos espectrales están basados en la descripción de las características del sonido que el auditor percibe. Por lo tanto, se basa n la percepción y no en el mundo físico. El sonido es generado en base a medidas perceptuales del timbre que se desea generar. La idea general de la síntesis espectral es analizar un sonido de manera de obtener representaciones espectrales alternativas, las cuales pueden ser a su vez alteradas de manera de producir nuevos sonidos. La mayoría de esos enfoques involucra, por lo tanto, una etapa de análisis previa a la síntesis.

5.9.1. Phase vocoder

Un vocoder (nombre derivado del inglés voice coder), o codificador de voz, es un analizador y sintetizador de voz. Fue desarrollado en la década de 1930 como un codificador de voz para telecomunicaciones. Su primer uso fue la seguridad en radiocomunicaciones, donde la voz tiene que ser digitalizada, cifrada y transmitida por un canal de ancho de banda estrecho.

Muchos vocoders usan un gran número de canales, cada uno en una frecuencia. Los diversos valores de esos filtros no son almacenados como números, que están basados en la frecuencia original, sino por una serie de modificaciones que el fundamental necesita para ser modificado en la señal vista en el filtro. Durante la reproducción esos números son enviados de vuelta a los filtros y entonces se modifican con el conocimiento de que el habla varía típicamente entre esas frecuencias. El resultado es habla inteligible, aunque algo mecánica. Los vocoders a menudo incluyen también un sistema para generar sonidos sordos, usando un segundo sistema para generar sonidos sordos consistente en un generador de ruido en lugar de la frecuencia fundamental.

El vocoder examina el habla encontrando su onda básica, que es la frecuencia fundamental, y midiendo cómo cambian las características espectrales con el tiempo grabando el habla. Esto da como resultado una serie de números representando esas frecuencias modificadas en un tiempo particular a medida que el usuario habla. Al hacer esto, el vocoder reduce en gran medida la cantidad de información necesaria para almacenar el habla. Para recrear el habla, el vocoder simplemente revierte el proceso, creando la

frecuencia fundamental en un oscilador electrónico y pasando su resultado por una serie de filtros basado en la secuencia original de símbolos.

El vocoder está relacionado con el algoritmo denominado phase vocoder, o vocoder de fase, aunque esencialmente es diferente de éste. Una phase vocoder es un tipo de vocoder que permite escalar una señal de audio tanto en el dominio de la frecuencia como en el dominio del tiempo usando información de fase. El algoritmo permite la modificación del espectro de un señal de audio, mediante lo cual es posible realizar efectos tales como compresión o expansión temporal y desfase de alturas (pitch shifting).

El phase vocoder se basa en la Transformada de Fourier de tiempo corto, o STFT. La STFT genera un representación combinada de tiempo y frecuencia del sonido, o lo que se denomina un sonograma, a través de sucesivas FFT en intervalos de tiempo relativamente cortos. En cada frame es posible modificar la información de amplitud o fase de la FFT, para luego resintetizar el sonido, generando de esta manera variaciones espectrales.

5.9.2. Síntesis de formantes

La voz humana consiste en sonidos generados por la apertura y cierre de la glotis o cuerdas vocales, lo que produce una onda periódica por lo general rica en contenido de frecuencia. Este sonido básico es modificado por el tracto vocal, generando zonas de concentración de le energía espectral, denominados formantes. Un formante consiste en una alta concentración de energía que se da en una determinada banda de frecuencias.

En el caso de la voz humana, para cada sonido usualmente se producen entre tres y seis formantes principales, denotados como F1, F2, F3, etc. Normalmente sólo los dos primeros son necesarios para caracterizar una vocal, si bien la pueden caracterizar hasta seis formantes. Los formantes posteriores determinan propiedades acústicas como el timbre. Los dos primeros formantes se determinan principalmente por la posición de la lengua. F1 tiene una frecuencia más alta cuanto más baja está la lengua, es decir, cuanta mayor abertura tenga una vocal, mayor es la frecuencia en que aparece el F1. F2 tiene mayor frecuencia cuanto más hacia delante está posicionada la lengua, es decir, cuanto más anterior es una vocal, mayor es el F2.

La síntesis basada en formantes se basa en este fenómeno de la voz humana. La idea es crear zonas de concentración de energía en el espectro del sonido sintetizado, lo que imita el sonido de la voz.

5.10. Modelos basados en partículas

Una gran variedad de síntesis basada en modelos de partículas, sin tratarse de la síntesis granular propiamente tal, puede ser encontrada en la literatura [28]. Algunas de ellas derivan de la síntesis granular, otras de técnicas gráficas, formantes y de la física.

Castagne y Cadoz destacan que los esquemas de masa-interacción son usualmente citados como técnicas de modelo físico, pero que muchas de las posibilidades musicales no han sido exploradas [8]. Ellos propusieron un entorno gráfico llamado GENESIS, el cual esta basado en el paradigma CORDIS-ANIMA mass-interaction y diseñado para músicos. El principal objetivo de este acercamiento es el de proveer de un 'pensamiento físico". En otras palabras, descubrir y experimentar nuevas formas de crear música.

Los modelos basados en partículas están basados comúnmente en modelos físicos [37], los cuales usan una descripción matemática del instrumento físico que se desea sintetizar. Un interesante enfoque presenta lo que Cook llamo modelamiento estocástico físicamene informado (PhISEM) [9]. Este algoritmo se basa en el traslape pseudo-aleatorio y a la suma de pequeños granos de sonido. Esta técnica considera modelos puntuales caracterizados por las ecuaciones de Newton. Este algoritmo ha sido exitoso en el modelaje de pequeños instrumentos de percusión tal como las maracas y otros instrumentos afines.

Sturm [39] [40] [41] ha trabajado en la sonificación de partículas basado en la hipotesis de de Broglie. Como el sonido es una superposición dinámica de frecuencias, existe la posibilidad de establecer una metáfora que relacione la física de partículas y la síntesis de sonidos. De acuerdo con Sturm, pensar en el sonido en términos de la evolución de un sistema de partículas nos otorga una nueva interpretación y una contra parte sonora a los eventos físicos. Bajo este criterio Sturm, deriva sus ecuaciones de transformación sónica.

5.11. Modelos basados en dinámica no lineal y caos

Se han propuesto variadas técnicas de síntesis basadas en dinámica no lineal y sistemas caóticos. Uno de los mas importantes trabajos ha sido el de Röbel [30], quien propuso una nuevo modelo para la síntesis de sonido basada en atractores y un método llamado modelo dinámico.

Originalmente, este método fue desarrollado para modelar sistemas caóticos dado solo un series de muestras obtenidos de la salida del sistema. El

modelo para la dinámica de los atractores no solo se limita al caso caótico sino que también para modelar sistemas con parámetros fijos. Además, se propuso el método como un modelo de redes neuronales para el habla y la música mediante la aplicación de series de tiempo [29].

Polotti et al. propusieron un método llamado síntesis aditiva fractal, el cual permite separar las componentes estocásticas de la voz para luego poder re-sintetizarlas separadamente [23] [24].

5.12. Síntesis basada en complejidad

El acercamiento tradicional para la síntesis de sonido ha sido intentar generar los sonidos mediante la combinación de elementos simples. En los últimos años se no han hecho evidente las dificultades para producir sonidos complejos de interés musical y que estos puedan competir con los instrumentos reales en términos de complejidad y expresión. Ya en 1997, Serra evidenció la necesidad de un nuevo foco para la síntesis digital [32].

Una forma natural de abarcar esta problemática es sintetizar sonidos directamente a través de sistemas complejos, esto es sistemas compuestos por muchos agentes individuales, con un comporamiento complejo e incluso en muchos casos caótico, y no recurrir al modelo de la simplificación. Un esquema de este tipo debiera ser muy útil para sintetizar sonidos provenientes de la naturaleza o que presenten una alta complejidad tanto espectral como temporal.

Capítulo 6

Procesamiento digital de audio

6.1. Filtros digitales

Un filtro es básicamente una caja negra con una entrada y una salida. Si la salida es diferente a la entrada, significa que la señal original ha sido filtrada. Cualquier medio por el cual una señal de audio pasa, cualquiera sea su forma, puede describirse como un filtro. Sin embargo, algo no nos parece un filtro a menos que pueda modificar el sonido de alguna manera.

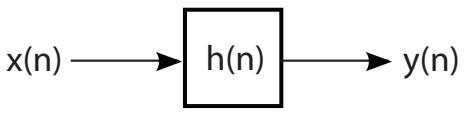


Figura 6.1: Un filtro como una caja negra

Los filtros pueden ser análogos, como el caso de un filtro solar de un telescopio, o digitales, como un eliminador de ruido implementado en el computador. Un filtro digital es un sistema de tiempo discreto que deja pasar ciertos componentes de frecuencia de una secuencia de entrada sin distorsión y bloquea o atenúa otros. Se trata simplemente de un filtro que opera sobre señales digitales, tales como las que operan dentro de un computador. Un filtro digital es una computación que recibe una secuencia de números (la señal de entrada) y produce una nueva (la señal de salida).

La figura 6.2 muestra el cambio que produce un filtro en el dominio de la frecuencia. En el primer caso el espectro de frecuencias se mantiene

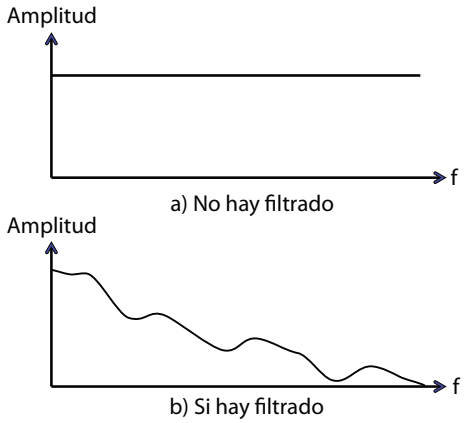


Figura 6.2: Filtrado versus no filtrado

simplemente igual, lo que indica que ese filtro no realiza cambio alguno en la señal de entrada. El segundo caso claramente muestra una modificación en el contenido de frecuencias de la señal original.

Los filtros análogos pueden ser usados para cualquier tarea, sin embargo, los filtros digitales pueden alcanzar resultados muy superiores. Un filtro digital puede hacer todo lo que un filtro análogo es capaz de realizar.

6.1.1. Ecuación de diferencias

En forma general, cualquier filtro digital puede ser descrito por la siguiente ecuación de diferencias:

$$y(n) = \sum_{i=0}^{M} b_i x(n-i) - \sum_{k=1}^{N} a_k y(n-k) \tag{6.1}$$

De esta ecuación se concluye que lo único que se necesita para implementar un filtro digital son sumas (o restas), multiplicaciones y retrasos, ya que nos dice que la salida del filtro depende de versiones presentes y pasadas de la entrada menos versiones pasadas de la salida. El índice n es un número entero que representa unidades discretas y los números a_k y b_i se denominan los coeficientes del filtro. Los coeficientes a_k multiplican a valores pasados de la salida y los coeficientes b_i a valores pasados de la entrada. Si todos los coeficientes a_k son cero, entonces el filtro sólo depende de valores pasados y presentes de la entrada y se trata de un filtro no recursivo o FIR. Si los valores a_k no son todos nulos, entonces se trata de un filtro recursivo o IIR.

6.1.2. Función de transferencia

En el dominio de la frecuencia compleja discreta (Z), cualquier fitro digital puede ser representado por una función de transferencia de la siguiente forma:

$$H(z) = \frac{Y(z)}{X(z)} = A\frac{1 + b_1 z^{-1} + ... + b_M z^{-M}}{1 + a_1 z^{-1} + ... + a_N z^{-N}} \tag{6.2}$$

donde z^{-1} es un retraso unitario y $H(z)$ es la función de transferencia y corresponde a la respuesta de frecuencia del filtro. El teorema fundamental del algebra nos permite reescribir la ecuación 6.2 de la siguiente forma:

$$H(z) = \frac{Y(z)}{X(z)} \tag{6.3}$$

$$= A\frac{(1 - q_1 z^{-1})...(1 - q_M z^{-M})}{(1 - p_1 z^{-1})...(1 - p_N z^{-N})} \tag{6.4}$$

donde q_i son llamados ceros y p_i polos. Los ceros corresponden a valores donde la función de transferencia es cero y los polos a valores donde la función de transferencia tiende a infinito (ver sección 6.1.5).

Dado que la multiplicación en el dominio del tiempo corresponde a una convolución en el dominio de la frecuencia, tomando la transformada Z inversa de la ecuación 6.2 se obtiene:

$$y(n) = h(n) * x(n) = \sum_{i=0}^{n} h(i)x(n - i) \tag{6.5}$$

donde $h(n)$ se conoce como la respuesta al impulso del filtro, discutida previamente en la sección 1.4.2. Por simple inspección, es fácil comprobar que la ecuación 6.5 se corresponde con la ecuación 6.1. En el caso de filtros no recursivos, o filtros de respuesta al impulso finita, los coeficientes del filtro corresponden directamente a los valores de la respuesta al impulso.

Todo sistema lineal e invariante en el tiempo (LTI) se caracteriza unívocamente por una respuesta al impulso y una respuesa de frecuencia. Cada una de estas respuestas contiene información completa sobre el filtro, pero codificada de una manera diferente. Si una de éstas respuestas es conocida, la otra puede ser obtenida en forma directa a través de la transformada de Fourier. La manera más directa de implementar un filtro digital es mediante la convolución de la señal de entrada con la respuesta al impulso del filtro. Todas los filtros lineales pueden ser implementados de esta manera.

6.1.3. Respuesta de frecuencia

Si bien en el dominio del tiempo, un sistema LTI de tiempo discreto se caracteriza completamente por su respuesta al impulso $h(n)$, interesa conocer el comportamiento del filtro en términos de su contenido de frecuencias. Utilizando el principio de superposición, es posible estudiar la respuesta global de un filtro a señales de distinta frecuencia mediante una combinación lineal de señales sinusoidales aisladas. Es así que en el dominio de la frecuencia, la respuesta de un sistema LTI a una combinación lineal de señales sinusoidales se denomina respuesta de frecuencia, usualmente denotada por $H(Z)$. La idea de esta respuesta es determinar el comportamiento del filtro en regimen permanente de reposo en la presencia de distintos componentes de frecuencia en la entrada. La respuesta de frecuencia $H(Z)$ puede ser calculada directamente como la transformada de fourier discreta de la respuesta al impulso $h(n)$.

Los filtros usualmente se clasifican dependiendo de la morfología de sus respuestas de frecuencia, típicamente en una de las siguientes formas: pasa-bajo, pasa-alto, pasa-banda o rechaza-banda, detallados en la figura 6.3.

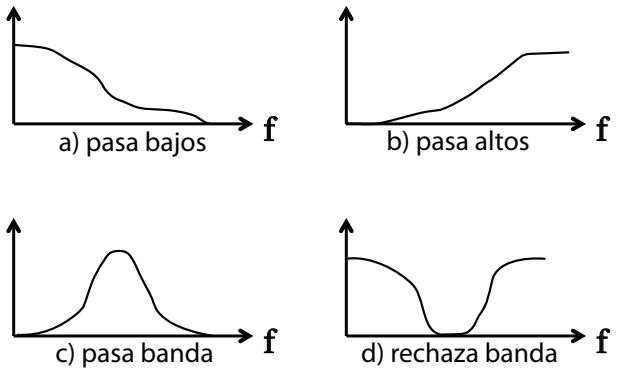


Figura 6.3: Respuestas de frecuencia típicas

Dado que la variable de frecuencia Z es una variable compleja, es importante no olvidar que la respuesta de frecuencia se compone de una magnitud y una fase. La magnitud de esta respuesta es lo que comúnmente se conoce como respuesta de frecuencia o respuesta de ampliud, mientras que su ángulo se conoce como respuesta de fase.

6.1.4. Respuesta de fase

La respuesta de fase $\Phi(\omega)$ de un filtro LTI se define como la fase (o ángulo complejo) de la respuesta de frecuencia $H(e^{j\omega t})$. La figura 6.4 muestra una respuesta de fase típica.

La respuesta de fase usualmente es referida simplemente como la *fase* del filtro. Para filtros con coeficientes reales, la fase puede ser definida sin ambiguedades como la la fase de la respuesta de frecuencia. La respuesta de fase determina el desfase medido en radianes al que será cometido cada componente de frecuencia de la señal de entrada.

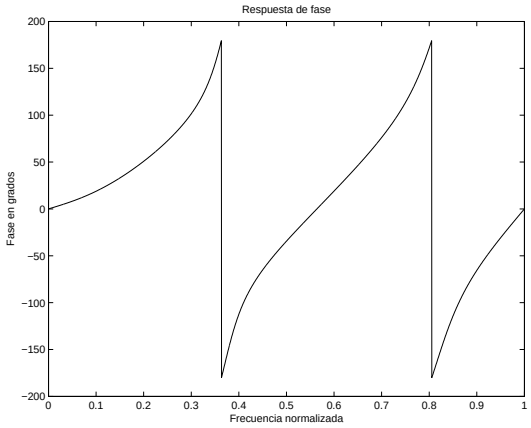


Figura 6.4: Respuesta de fase

La figura 6.5 muestra respuestas de fase lineales y no lineales tanto para filtros FIR, descritos en la sección 6.2, como para filtros IIR, abordados en la sección 6.3.

Retraso de fase (phase delay)

Dado que la respuesta de fase $\Phi(\omega)$ otorga información sobre el desfase en radianes o grados, no es muy util en términos prácticos, por lo que resulta más intuitivo disponer de alguna forma de conocer el retraso de cada componente medido en segundos. Esto se conoce como retraso de fase (en inglés phase delay), el cual se define como:

$$-\frac{\Phi(\omega)}{\omega} \tag{6.6}$$

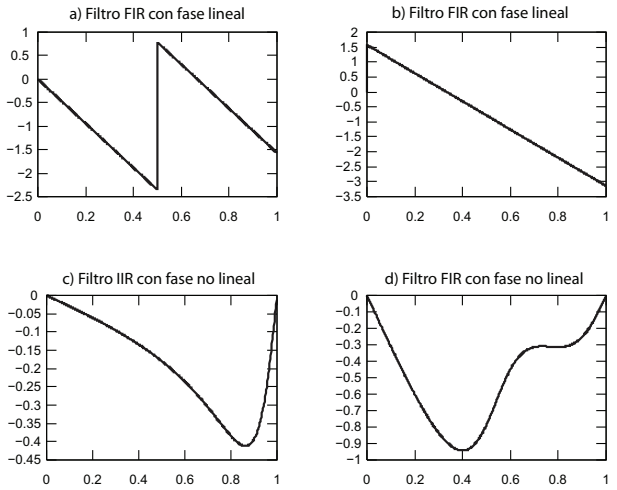


Figura 6.5: Respuestas de fase lineales versus no lineales

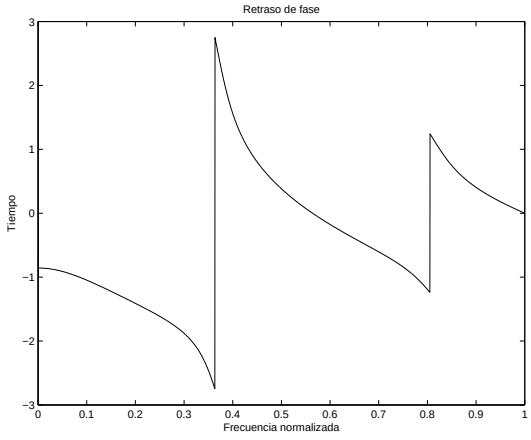


Figura 6.6: Retraso de fase

La figura 6.6 muestra el retraso de fase de la respuesta de fase mostrada anteriormente en la figura 6.4.

Retraso de grupo (Group delay)

Otra forma común de presentar la información de fase de un filtro se denomina retraso de grupo (en inglés group delay), definido como:

$$-\frac{d\Phi(\omega)}{d\omega} \tag{6.7}$$

Para respuestas de fase lineales, es decir cuando $\Phi(\omega) = -\alpha\omega$ para alguna constante α, el retraso de grupo y el retraso de fase son idénticos, y cada uno de ellos puede interpretarse como retraso de tiempo. Pero si la respuesta de fase es no lineal, por lo general las fases relativas de los componentes sinusoidales de la entrada se ven alterados. En este caso, el retraso de grupo puede ser interpretato como el retraso temporal de la envolvente de la amplitud de una sinusoide de frecuencia ω. El ancho de banda de esta envolvente debe estar restringido a un intervalo de frecuencias sobre el cual la respuesta de fase es aproximadamente lineal. Por lo tanto, el nombre retraso de grupo se refiere al hecho que éste especifica el retraso que experimenta una banda de frecuencias con un intervalo de frecuencias pequeño en torno a ω.

Desenrollado de fase (phase unwrapping)

Muchas veces es necesario desenrollar la respuesta de fase, $\Phi(\omega)$, dado que, comúnmente, ésta contiene saltos o discontinuidades, como el caso de la figura 6.4, donde se aprecian saltos de magnitud 2π.

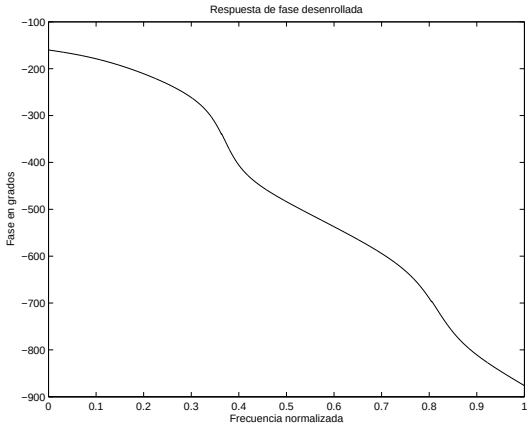


Figura 6.7: Fase desenrrollada

Al desenrrollar la fase, se asegura que todos lo múltiplos de 2π hayan sido incluidos en $\Phi(\omega)$. Por si misma, $\Phi(\omega)$ no es suficiente para obterner una respuesta de fase que pueda ser convertida a un desfase de tiempo verdadero. Si se descartan los múltiplos de 2π, tal como se hace en la definición de un ángulo complejo, el retraso de fase se modifica de acuerdo a múltiplos enteros del período sinusoidal. Dado que el análisis de los filtros LTI se basa en sinusoides periódicas, sin principio ni fin, no es posible en principio distinguir entre retraso de fase verdadero y un retraso de fase con periódos descartados si se observa a una salida sinusoidal de una frecuencia determinada. Sin embargo, frecuentemente es útil definir la respuesta de fase de un filtro como una función continua de la frecuencia con la propiedad $\Phi(0) = 0$ (para filtros reales). Esto especifica como desenrrollar la respuesta de fase en el dominio de la frecuencia. La figura 6.7 muestra la respuesta de fase de la figura 6.4 desenrollada.

6.1.5. Diagramas de polos y ceros

La ecuación 6.4 muestra la función de transferencia $H(z)$ de un filtro escrita en términos de ceros y polos. El término cero se refiere al número mientras la palabra polo es una traducción del inglés pole, que podría traducirse como palo. Estos términos adquieren sentido cuando se grafica la magnitud de $H(z)$ como una función de la frecuencia compleja z. Esto se realiza en lo que se denomina el plano-z, que es simplemente un plano donde el eje de ordenadas representa los números reales y el eje de abscisas números puramente imaginarios. Dado que z es una variable compleja, ésta puede graficarse como un vector que parte desde el origen y cuya distancia al origen representa su magnitud mientras su ángulo respecto al eje de los números reales representa la fase.

La magnitud de $H(Z)$ es real y puede interpretarse como la distancia sobre el plano-z, mostrado en la figura 6.8. El gráfico de $H(Z)$ aparece entonces como una superficie infinitamente delgada que se extiende en todas las direcciones del plano. Los ceros son puntos donde la superficie casi toca el plano-z. Por el contrario, los polos representan puntos donde ésta superficie alcanza una gran altitud y se hacen cada vez más angostos a medida que se alejan del plano.

Este tipo de diagrama resulta útil porque muestra una representación gráfica tanto de la respuesta de amplitud como de fase de un filtro digital. Todo lo que se necesita conocer son los valores de los polos y ceros para poder estimar en forma gráfica la respuesta de fase y magnitud.

La magnitud de la respuesa de frecuencia, o respuesta de amplitud para

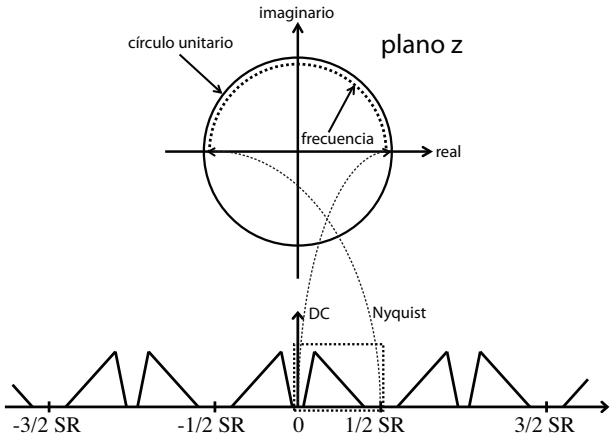


Figura 6.8: El plano z

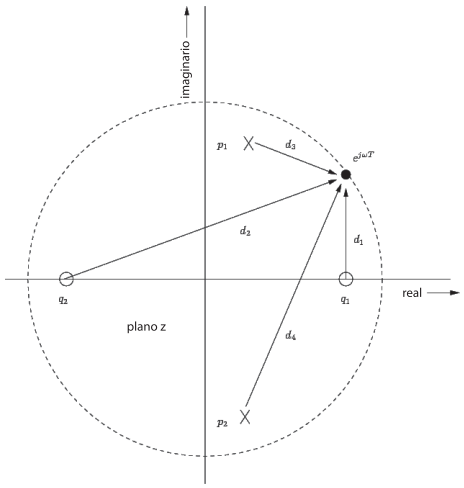


Figura 6.9: Estimación de la respuesta de amplitud mediante un diagrama de polo y ceros

cada frecuencia está dada por el producto de las magnitudes de los vectores dibujados desde los ceros hacia el círculo unitario divivido por el producto de las magnitudes de los vectores dibujados desde los polos hacia el círculo unitario.

La respuesta de fase de obtiene dibujando líneas desde todos los polos

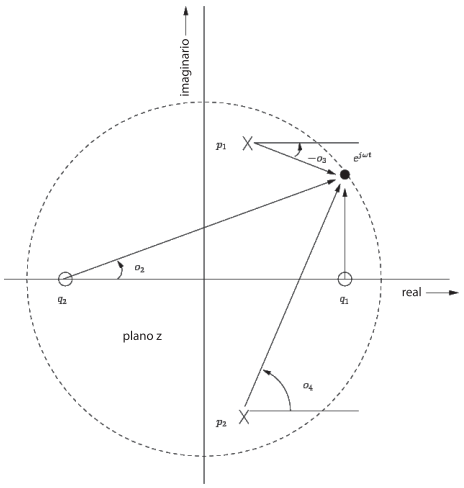


Figura 6.10: Estimación de la respuesta de fase mediante un diagrama de polo y ceros

y ceros hacia el círculo unitario, tal como se muestra en la figura 6.10. Los ángulos de las líneas de los ceros se suman y los ángulos de las líneas de los polos se restan.

6.1.6. Filtros de primer orden

Los filtros digitales se clasifican segun el número de retrasos involucrados en su implementación y esto determina su orden. Un filtro de primer orden sólo involucra un retraso y por ende sólo puede estar constituido por un cero o un polo.

La figura 6.11 muestra un filtro de un cero y la figura 6.12 uno de un polo. Dependiendo del coeficiente se pueden construir filtros pasabajos o pasa altos, tal como muestra la figura.

La forma de la respuesta de frecuencia en ambos casos cambia dependiendo si el filtro es de un polo o de un cero.

6.1.7. Filtros de segundo orden

Si el filtro involucra dos retrasos, se habla de un filtro de segundo orden. Al disponer de dos polos y dos ceros, es posible construir filtros pasa banda o rechaza bandas, tal como lo muestra la figura 6.13.

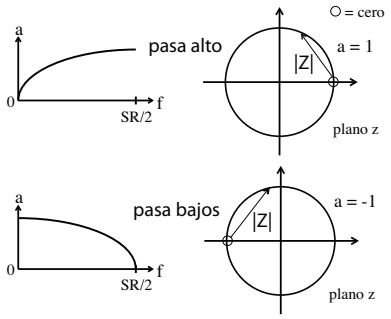


Figura 6.11: Filtro de primer orden de un cero

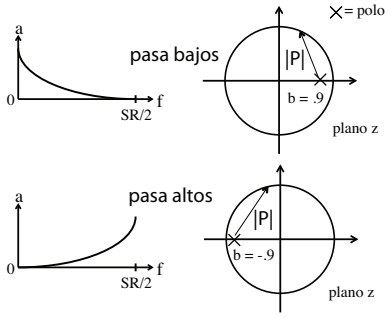


Figura 6.12: Filtro de primer orden de un polo

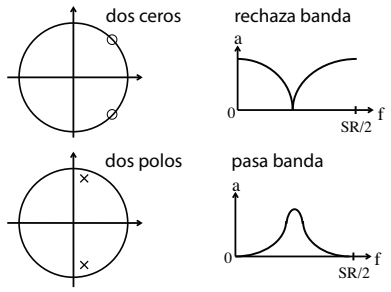


Figura 6.13: Filtro de segundo orden de dos polos y dos ceros

El caso general de un filtro de segundo orden, de dos polos y dos ceros, denominado, filtro bi-cuadrático, se detalla en la sección 6.4.

6.2. Filtros FIR

En un filtro con respuesa al impulso finita o FIR, los coeficientes a_k son cero, lo que significa que la respuesta del filtro depende sólamente de la entrada y no de valores pasados de la salida. Este tipo de filtros tiene una respuesta finita ya que no exhiben recursión.

La figura 6.14 muestra la implementación de un filtro FIR. Una de las propiedades más interesantes de este tipo de filtros es la simetría de sus coeficientes y el hecho que pueden ser diseñados de tal forma de exhibir una respuesta de fase lineal.

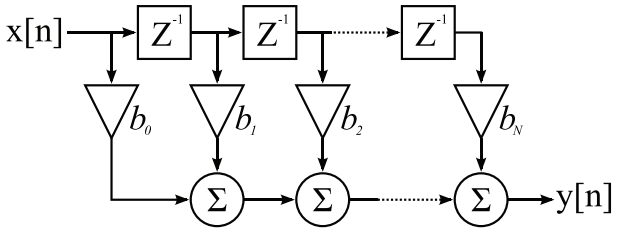


Figura 6.14: Esquema de implementación de un filtro FIR

La función de transferencia de un filtro FIR está dada por:

$$H(z) = \sum_{i=-\infty}^{\infty} h_n z^{-n} = \sum_{n=0}^{M} b_n z^{-n} \tag{6.8}$$

Los filtros FIR tienen las siguientes propiedades:

- Respuesta de fase lineal (si se diseñan adecuadamente)
- Estabilidad con coeficientes cuantizados
- En general, requieren de un mayor orden que los filtros IIR

El diseño de filtros FIR se basa usualmente en una aproximación directa de la magnitud de la respuesta deseada a la cual se le agrega la condición de una respuesa de fase lineal. Este requerimiento es:

$$h(N-1-n) = \pm h(n) \tag{6.9}$$

6.2.1. Diseño de filtros FIR

Un filtro FIR puede ser diseñado utilizando distintas técnicas tanto en el dominio del tiempo como de la frecuencia. Existe un método óptimo llamado algoritmo de Parks McClellan que produce coeficientes óptimos y fase lineal. Este método es muy popular dado su facilidad de usar, flexibilidad y excelentes resultados.

Las técnicas de ventaneo (en inglés windowing), básicamente toman la respuesta al impulso de un filtro ideal con duración infinita y aplican una ventana para truncar y limitar los coeficientes a un número específico, para luego desplazar la secuencia de manera de garantizar causalidad. Esta técnica es simple y produce resultados razonables, pero con distorsión respecto a la respuesta ideal.

Otras técnicas se basan en muestreo en la frecuencia. La idea es muestrear la respuesta de amplitud deseada a intervalos regulares y luego utilizar la DFT inversa para general una respuesa al impulso de duración finita. El número de muestras determina el nivel de precisión del filtro respuesto a la respuesa ideal. Esta técnica es bastante sofisticada y requiere una tasa de muestreo alta.

Por último, hay una técnica ad-hoc denominada posicionamiento de ceros. La idea es colocar ceros en el plano-z de forma estratégica con el fin de aproximarse a la respuesta deseada y realizar el cálculo de los coeficientes en forma gráfica. Mientras más cercano esté el cero al círculo unitario, mayor efecto tendrá en la respuesta de frecuencia del filtro para ese rango de frecuencias. Esta técnica sin dudas es rápida y no muy precisa, pero puede funcionar para aplicaciones simples.

6.3. Filtros IIR

Los filtros de respuesta al impulso infinita o IIR también son llamados filtros recursivos, porque la salida del filtro depende de valores pasados de si misma. La función de transferencia de un filtro IIR es:

$$H(z) = \frac{1}{\sum_{n=0}^{N} a_n z^{-N}} \tag{6.10}$$

Los filtros IIR tienen las siguientes características:

- Mejor atenuación que los filtros FIR
- Fórmulas cerradas de aproximación

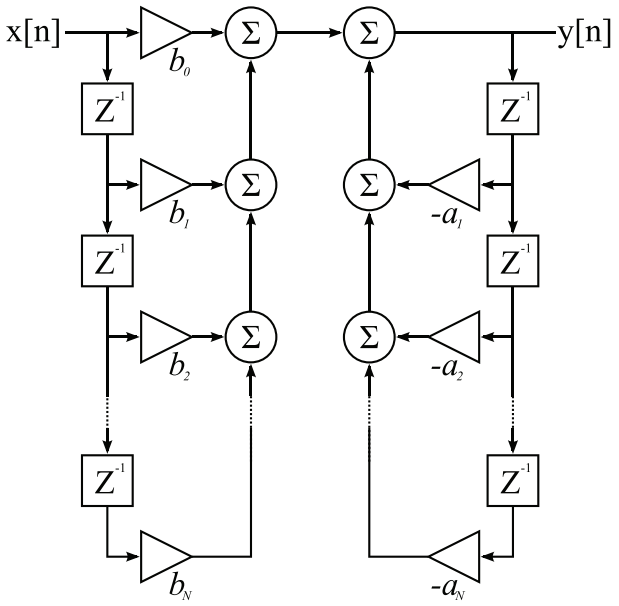


Figura 6.15: Esquema de implementación de un filtro IIR

- Respuesta de fase no lineal
- Inestabilidad (oscilaciones de ciclo límite) en computaciones de largo de palabra finito

6.3.1. Diseño de filtros IIR

Los filtros IIR pueden tener polos (debido a la retroalimentación) y ceros, y las técnicas de diseño difieren enteramente de las técnicas para diseño de filtros FIR. Los métodos directos de diseño se llevan a cabo en el plano complejo discreto Z, y los métodos indirectos empiezan en el plano análogo o continuo S, para ser transformados después al plano-Z. Las técnicas son indirectas son las más utilizadas y por lo general entregan mejores resultados.

Entre las técnicas directas se encuentran el posicionamiento ad-hoc de polos y zeros con el subsiguiente análisis de frecuencia, análisis en el dominio del tiempo y en el dominio de la frecuencia. En el dominio del tiempo, se transforma la respuesta de frecuencia de un filtro ideal $H(Z)$ en su respuesta al impulso y se resuelve un sistema de ecuaciones de N coeficientes de manera de minimizar el error cuadrático medio. En el dominio de la frecuencia, le

objetivo es mapear una respuesta de frecuencia ideal hacia una estructura IIR.

Las técnicas indirectas implican el diseño de un filtro analógico apropiado en el plano S y luego mapear desde éste al plano discreto Z, usando las siguientes transformaciones:

$$Z = e^{sT_s} \tag{6.11}$$

$$s = \frac{1}{T_s} \ln(Z) \tag{6.12}$$

El logaritmo natural presenta un problema cuando se trata de mapear desde el plano S al Z. Una solución muy utilizada es expandir $\ln(Z)$ como una serie y truncar el primer elemento. Una vez hecho esto, hay dos técnicas que permiten generar la transformada Z: la transformada bilinear y la diferencia hacia atrás (backward difference), de la cual la más utilizada es la transformada bilinear.

6.3.2. Transformada Z bilinear

Este es usualmente el método que se utiliza para el diseño de filtros IIR. El enfoque más utilizado es empezar de las especificaciones del filtro y luego escoger la función apropiada para obtener un filtro análogo equivalente y su función de transferencia $H(s)$. Luego, se transforma o mapea este filtro al plano Z. Esto se realiza mediante la siguiente ecuación:

$$s = \frac{2}{T_s} \frac{1 - Z^{-1}}{1 + Z^{-1}} \tag{6.13}$$

Típicamente, los filtros escogidos son:

- Filtros Butterworth. Poseen una respuesta de fase razonable y una respuesta de amplitud monotónica.
- Filtros Chebyschev. Poseen una zona de corte más pronunciada que un filtro Butterworth.
- Filtos elípticos. Poseen la mejor respuesta de amplitud y el orden óptimo para diseños IIR, pero la peor respuesta de fase en términos de linearidad.

6.4. Filtros bi-cuadráticos

Los filtros bi-cuadráticos o simplemente biquad en inglés , son filtros que constan de dos polos y dos ceros y cuya función de transferencia está dada por:

$$H(z) = g\frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{6.14}$$

donde g es la ganancia. Dado que tanto el numerador como el denominador son polinomios de segundo orden en z, esta funci;on de transferencia se conoce como bi-cuadrática en z. Existen diferentes maneras de implementar un filtro bi-cuadrático. Una de las más conocidas es a través de las fórmulas obtenidas por Robert Bristow-Johnson. Estas formulas pueden utilizarse para calcular los coeficientes de los tipos más comunes de filtros digitales: pasa-bajos, pasa-altos, pasa-bandas, notch, peakingEQ, lowShelf y highShelf.

El apéndice A.9 contiene código SuperCollider que implementa filtros bi-cuadráticos utilizando las ecuaciones de Bristow-Johnson.

6.5. Filtros comb

Un tipo de filtro muy común en aplicaciones de audio es el llamado filtro comb (peineta en inglés), el cual es implementado mediante una línea de retraso. La idea de este filtro es combinar la señal de entrada con una versión retrasada de la misma. Hay dos subtipos de filtros comb: no recursivos (o feedforward) y recursivos (o feedback). Este filtro debe su nombre a que su respuesta de amplitud se asemeja mucho a la forma de una peineta o comb en inglés.

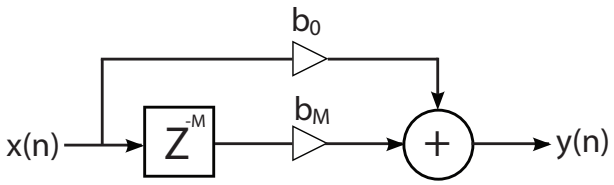


Figura 6.16: Filtro comb no recursivo

El filtro comb no recursivo usualmente se implementa tal como se muestra en la figura 6.16 y el filtro recursivo se grafica en la figura 6.17.

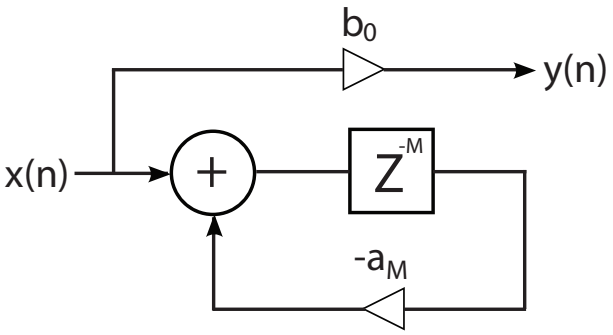


Figura 6.17: Filtro comb recursivo

La ecuación de diferencias de un filtro comb no recursivo es:

$$y(n) = b_0 * x(n) + b_M * x(n - M) \tag{6.15}$$

y para un filtro comb recursivo:

$$y(n) = b_0 * x(n) - a_M * y(n - M) \tag{6.16}$$

La respuesta de amplitud típica de un filtro comb se muestra en la figura6.18.

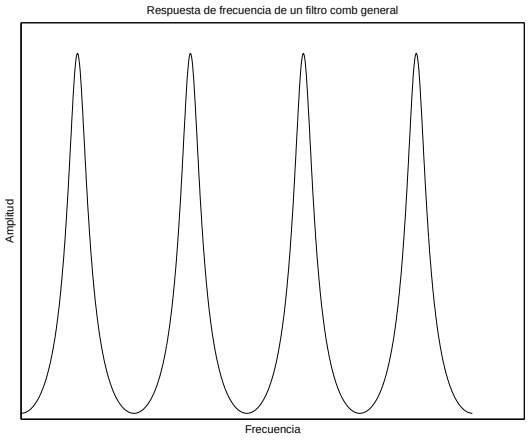


Figura 6.18: Respuesta de amplitud de un filtro comb

6.5.1. Phasing y flanging

Un phaser, que podría traducirse como faseador, genera zonas de peaks y depresiones en el espectro. La posición de éstos varía en el tiempo, creando un efecto de glissando. Para estos efectos, se utiliza usualmente un oscilador de baja frecuencia.

Este efecto se logra al mezclar la señal de entrada con una versión modificada de sí misma, a través uno o varios filtros pasa-todo, que mantienen la magnitud del espectro de frecuencias, pero alteran su fase en forma no lineal. Para cada componente de frecuencia, el cambio en la fase es distinto. Al mezclarse ambas versiones, las frecuencias que se encuentran fuera de fase, se cancelan, creando las zonas de depresión características de este efecto, las cuales usualmente no ocurren en forma armónica. Al cambiar la mezcla se logra modificar la profundidad de las depresiones, alcanzandose los máximos valores cuando la mezcla es de 50 %.

El flanging es un tipo específico de phaser, donde la sección pasa-todo se implenta como una secuencia de líneas de retraso. La señal de entrada se agrega a una copia retrasada de la misma, lo que resulta en una señal de salida con peaks y depresiones en relación armónica.

Ambos efectos son variaciones de filtros comb. El flanger utiliza un filtro comb con dientes regularmente espaciados, mientras que el phasing utiliza una filtro con dientes separados irregularmente.

Los controles habituales en este tipo de efectos son:

- Retraso: Es el umbral máximo de desfase de la onda duplicada respecto a la original, se suele expresar en milisegundos.
- Frecuencia: Es la frecuencia de oscilación del desfase de la onda duplicada.
- Profundidad: Es la cantidad de onda original que se mezcla con la duplicada.

6.6. Ecualizador

Un ecualizador modifica el contenido en frecuencias de la señal de entrada. Consiste básicamente en un banco de filtros pasa-banda, cada uno centrado en una banda de frecuencias específicas. Comúnmente se utilizan para reforzar ciertas bandas de frecuencias, ya sea para compensar la respuesta del equipo de audio (amplificador + altavoces) o equilibrar en términos espectrales el sonido. Existen ecualizadores analógicos y digitales, activos

o pasivos, paramétricos, gráficos y paragráficos. Los ecualizadores suelen poseer control sobre diez bandas de frecuencia o más.

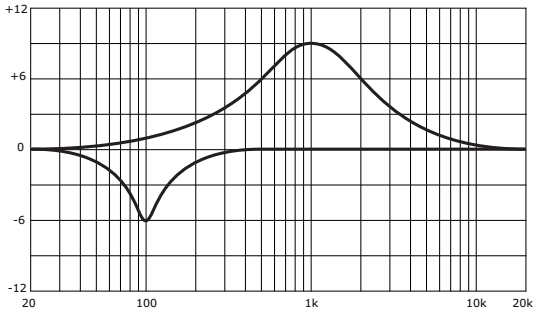


Figura 6.19: Dos curvas de ecualización

La figura 6.19 muestra dos curvas de ecualización posibles, una que enfatiza la zona alrededor de los 1000 Hz y otra que atenúa frecuencias cercanas a 100 Hz.

6.7. Compresión

Tal como se detalla en el capítulo 3, una característica notable del sistema auditivo humano es su capacidad de detectar un amplio rango de amplitud o intensidad sonora, desde el más débil murmullo hasta el sonido ensordecedor de un avión a propulsión. Al tratar de grabar o reproducir este vasto espectro de amplitudes sonoras, es imposible evitar limitaciones físicas, tanto electrónicas como acústicas, de la tecnología de almacenamiento y reproducción de sonido. Cuando el rango dinámico de la señal acústica excede las capacidades de un sistema de reproducción o grabación de sonido, inevitablemente los puntos máximos de la señal serán distorsionados por truncamiento (en inglés clipping), los altavoces pueden sufrir daño al verse sometidos a niveles muy altos de intensidad, o pasajes muy suaves en términos de amplitud pueden pasar desapercibidos en la presencia de ruido eléctrico o de fondo (en inglés noise floor). Por lo tanto, se hace necesario mantener el nivel operativo de la señal lo más alto posible de manera de minimizar la distorsión que se produce por el ruido de fondo, y al mismo tiempo limitar los picos de la señal para no causar truncamiento o sobrecarga del sistema. Esto se realiza a través de un *compresor*. Es importante no confundir a este tipo de compresores de amplitud de audio con la compresión

de información digital o perceptual.

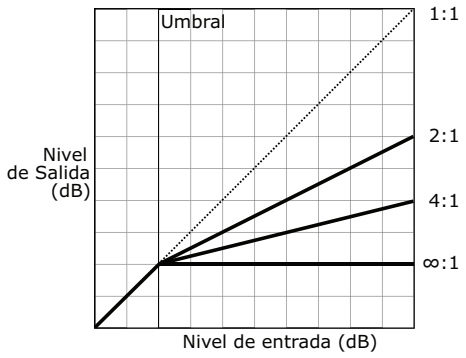


Figura 6.20: Funciones de compresión a distintas tasas

Los compresores usualmente tienen un sistema automático de control de ganancia que constante monitorea la señal y ajusta la ganancia para maximizar la razón señal a ruido sin producir distorsión. Este tipo se dispositivos se conocen como procesadores dinámicos, compresores y limitadores. La utilizar procesadores dinámicos, los niveles peaks de la señal se pueden reducir, lo que a su vez permite incrementar el nivel global o promedio de la señal de audio.

Los principales parámetros de un compresor son:

- Umbral (usualmente desde -40 a +20 dBu). Este parámetro fija el nivel sobre el cual las señales serán comprimidas o limitadas. Al incrementarse el umbral, la señal se ve truncada a niveles mayores y por lo tanto se reduce la cantidad de compresión o limitación.

- Razón (1:1 a ∞:1). Este parámetro determina la pendiente de la compresión, lo que incide en como la señal de salida cambia en relación a la señal de entrada una vez que excede el umbral. El primer dígito indica cuando cambio de la señal de entrada medido en dB causará un cambio de 1 dB en la salida. Mientras más alta la razón, mayor es la compresión y más achatado se vuelve el sonido.

La figura 6.21 muestra como normalmente se implementa la correción de ganancia automática. Primero, la amplitud estimada se compara con el umbral, se toma el valor máximo entre los dos y se sustrae esa cantidad al umbral. Esto determina el rango de amplitud que es necesario comprimir.

Luego, este rango es multiplcado por la cantidad $1 - (1/R)$ donde R es la tasa de compresión. Finalmente se resta el bias a esta cantidad y el resultado se convierte a ganancia lineal.

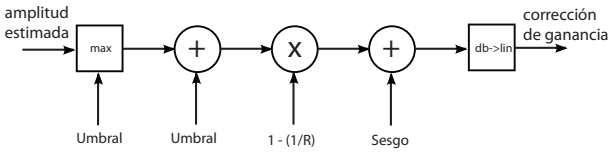


Figura 6.21: Corrección automática de ganancia

En el apéndice A.10 se puede encontrar código SuperCollider que implementa un compresor de audio completo, con corrección de ganancia como la recién descrita.

6.8. Reverberación

Si un sonido no es absorbido o transmitido cuando impacta una superficie, simplemente se refleja. La ley de reflexión en este caso funciona exactamente como para el caso de la luz, y el ángulo de incidencia de una onda de sonido es equivalente al ángulo de reflexión, tal como si ésta hubiera sido producida por una imagen espejo del estímulo en el lado opuesto de la superficie. Sin embargo, esta ley de reflexión sólo es aplicable cuando la longitud de onda del sonido es pequeña en comparación con las dimensiones de la superficie reflectante.

La reflexión del sonido da paso a la difusión, reverberación y el eco. Para superficies distintas, el grado de reflexión, medido a través de su coeficiente de absorción o reflexión, es distinto. Por lo general, las superficies cóncavas concentran las ondas de sonido en áreas específicas, mientras las formas convexas diseminan las ondas sonoras, promoviendo una buena difusión del sonido.

Retrasos muy cortos provocan un desplazamiento en la imagen sonora hacia ambos lados y una coloración del sonido debido al efecto phasing. Retrasos más prolongados contribuyen a la impresión espacial de la reverberación, mientras que retrasos mayores a 50 ms perturban la imagen sonora y son percibidos como ecos.

La *reverberación* es el resultado de un gran número de reflexiones sonoras que ocurren en una sala. Desde cualquier fuente sonora usualmente hay un camino directo entre esa fuente y los oídos del auditor. Pero esa no es la

única forma en que un sonido llega a quienes lo escuchan. Las ondas de sonido pueden tomar caminos más largos mediante rebotes o reflejos en las paredes, cielo o piso, antes de llegar a los oídos. Un sonido reflejado toma un tiempo mayor en llegar que el sonido directo, dado que recorre una mayor distancia y generalmente se encuentra debilitado, ya que las paredes y otras superficies absorben parte de su energía. Onda onda sonora que ya ha rebotado en alguna superficie puede seguir experimentando rebotes sucesivos antes de llegar a los oyentes. Esta sucesión de sonidos atenuados y retrasados es lo que se conoce como reverberación y es lo que causa nuestra sensación de espacio dentro de una sala de conciertos. La reverberación es, en efecto, una multiplicidad de ecos cuya frecuencia de repetición es demasiado rápida como para ser percibidos como sonidos separados entre uno y otro.

Otra característica importante de la reverberación es la *correlación* de las señales que llegan a nuestros oídos. Para obtener una sensación realista de la espacialidad de una sala es necesario que los sonidos en cada oído estén en alguna medida *decorrelacionados*. Esto es parte de la causa por la cual las salas de concierto tiene cielos tan elevados, ya que con un cielo de baja altura, las primeras reflexiones serían las provenientes del cielo y alcanzarían ambos oídos al mismo tiempo. Si se dispone de un cielo más elevado, las reflexiones tempranas provendrían en su mayoría de las paredes, y dado que éstas se encuentran comúnmente a diferentes distancias del auditor, el sonido que llega a cada oído es diferente.

Una unidad de medida que típicamente se utiliza en la reverberación de salas es lo que se denomina el *tiempo de reverberación*, el cual es la cantidad de tiempo que toma la intensidad de la señal en decar 60 dB respecto a su valor original. Tiempos de reverberación largos significan que la energía del sonido permanece en la sala una cantidad de tiempo considerable antes de ser absorbidos. El tiempo de reverberación se asocia con el tamaño de una sala. Las salas se conciertos típicamente tienen tiempos de reverberación del orden de 1,5 a 2 segundos.

El tiempo de reverberación es controlado primordialmente mediante dos factores: las superficies de la sala y su tamaño. El tipo de superficie determina cuanta energía se pierde en cada reflexión. Los materiales altamente reflectivos, como el concreto, cerámicas, ladrillo o vidrio, incrementan el tiempo de reverberación debido a su rigidez. Los materiales absorbentes, como cortinas, alfombras y la gente, reduce el tiempo de reverberación. Además es necesario considerar que los coeficientes de absorción de cada material usualmente varían con la frecuencia.

La gente tiende a absorber una gran cantidad de energía acústica, reduciendo el tiempo de reverberación. Las salas grandes tienden a tener tiempo

de reverberación más largos, dado que, en promedio, las ondas de sonido viajan una distancia más larga entre reflexiones. El aire en la sala también atenúa las ondas sonoras, reduciendo el tiempo de reverberación. Esta atenuación varía con la humedad y la temperatura, y las altas frecuencias son las más afectadas en estos casos. Debido a esto, muchos reverberadores artificiales incorporan una etapa de filtros pasa bajo para lograr un mayor realismo en el efecto.

6.8.1. Reverberación artificial

La reverberación artificial es necesaria para asegurar un grado convincente de realismo de una escena auditiva virtual y para proveer control sobre las distancias percibidas de las fuentes sonoras. Las aplicaciones del procesamiento espacial de audio y técnicas de reverberación artificial incluyen la producción de música grabada o en vivo, multimedia o realidad virtual o evaluación de acústica de arquitecturas.

La reverberación de una sala se puede simular mediante la convolución de una señal de entrada con la respuesta al impulso de la sala, sea esta medida en forma física o simulada en el computador. Mediante los algoritmos disponible hoy en día, es posible implementar convolución de respuestas al impulso de larga duración en tiempo real, sin experimentar retrasos significativos y sin exceder la capacidad computacional de los procesadores digitales de señales.

El enfoque de la convolución resulta adecuado para la auralización predictiva o comparativa de salas de conciertos, auditorios o sistemas de sonidos, cuando un ambiente de escucha controlado, o a través de audífonos se pueda experimentar todas las ventajas de la técnica.

La aproximación tradicional a la reverberación sintética en tiempo real se basa en redes de retraso basadas en algoritmos de feedforward para generar las reflexiones tempranas y algoritmos de feedback para la reverberación tardía. Este enfoque no puede garantizar el mismo grado de precisión que la convolución con la respuesta al impulso de la sala, pero permite una parametrización más eficiente del control dinámico del efecto de reverberación sintético.

El apéndice A.11 contiene código SuperCollider que implementa un reverberador artificial completo.

6.9. Otros efectos

6.9.1. Chorus

El efecto chorus o de coro se obtiene al mezclar una señal de entrada con vibrato junto con la señal sin procesar. Para producir el efecto, ya sea natural o artificialmente, deben mezclarse varios sonidos individuales con aproximadamente el mismo timbre y alturas levemente disímiles, de manera de ser percibidos como un sólo sonido proveniente de una misma fuente. El efecto de coro se enfatiz cuando los sonidos se originan en tiempos y lugares del espacio levemente distintos. Esto es lo que típicamente sucede en un coro de voces, y es lo que original el nombre del efecto.

6.9.2. Wah wah

El wah-wah es un tipo de efecto usualmente utilizado en instrumentos como la guitarra o bajo eléctrico. Básicamente, se trata de un filtro pasa bajos con una frecuencia de corte variable. Al variar esta frecuencia de corte se produce el sonido característico que da origen el nombre de este efecto. El efecto wah-wah también es posible de producir mediante el modelamiento de un sonido instrumental utilizando un vocoder y el sonido wah-wah hablado como señal moduladora. De esta forma, se transfieren los formantes del sonido hablado al sonido musical.

6.9.3. Delay

Un efecto de delay o retraso consiste en la multiplicación y retraso modulado de una señal de entrada, que se mezcla con la señal original, obteniendose un efecto de eco sonoro. Los parámetros más comunes de un delay son el tiempo de retraso, que corresponde al tiempo que tarda en producirse un eco, la cantidad de feedback o retroalimentación, definida como la cantidad de veces que se repite la señal sonora y la proporción de mezcla, que es la cantidad de sonido retrasado que se mezcla con el original.

6.10. Procesador de efectos genérico

Un procesador general de efectos se puede implementar insertando en primer lugar la señal de entrada en una línea de retraso recursiva. La señal desfasada es luego filtrada utilizando un filtro de primer orden, usualmente un filtro de un polo. La cantidad de retroalimentación y su signo pueden ser especificados.

Usualmente, el retraso es modulado de manera de obtener efectos más interesantes tales como el flanging. Esta arquitectura es básicamente un filtro comb con la variación de poseer un filtro pasa bajos en vez de un multiplicador simple para la linea de retroalimentación.

Los parámetros usuales de un procesador genérico son:

- Ganancia de entrada en dB (no mostrada en la figura)
- Tiempo de retreaso en ms
- Fase de retroalimentación
- Cantidad de retroalimentación
- Mezcla de salida
- Oscilador de frecuencia de retraso variable
- Desviación de retraso

En el apéndice A.12 se detalla la implementación de un procesador de efecto genérico en SuperCollider, basado en ésta descripción.

Capítulo 7

La música electroacústica

La música electroacústica constituye un género bastante particular en la historia de la música y en mi opinión, uno que no es muy bien entendido incluso por algunos músicos profesionales, teóricos y musicólogos. Son los compositores quienes parecen ser más conscientes de las características y particularidades de este tipo de música. Una muestra de esto es que la mayoría de los escasos análisis publicados sobre música electroacústica están hechos por compositores, ya sea el propio autor u otro compositor.

Al hablar de música electroacústica es importante determinar claramente qué se entiende por el término. Hay dos posibles definiciones. La primera considera formas de música que pueden utilizar cualquier tipo de fuente sonora que pase a través de un circuito eléctrico y que resuene a través de un altoparlante. Cualquier tipo de música grabada electrónicamente y reproducida mediante altoparlantes cae en esta categoría. La segunda definición contempla la música que es generada mediante aparatos electrónicos o mediante una combinación de éstos con instrumentos acústicos [4]. En el presente texto, se considerará la segunda definición.

Esta música manifiesta grandes diferencias con respecto a la música instrumental [25]. Tanto es as que algunos compositores de música electroacústica han llegado a cuestionar si lo que hacen puede ser considerado música en el sentido en que normalmente entendemos el término. Otros prefieren la definición arte auditivo (audio art) o arte sonoro (sonic art) y muchos rechazan la idea de ser compositores en el sentido tradicional. Por lo general, comprenden a la música como un subconjunto del más amplio arte auditivo [20].

Estas diferencias afectan también a los roles de compositor e intérprete. Claramente separados y definidos en la tradición musical de occidente, ahora

se han vuelto difusos y muchas veces no es posible separarlos. La posibilidad de ser compositor/intérprete abre nuevas interrogantes y nuevos espacios en la práctica de este arte [25].

A pesar de que el campo de la música electroacústica, y en especial de la música computacional, es muy rico en publicaciones e investigación, su análisis está todavía en pañales. De hecho, las publicaciones sobre análisis de obras electroacústicas son muy escasas en comparación con la cantidad de análisis publicados sobre música instrumental o incluso contemporánea.

Al comparar la música electroacústica con la música instrumental o vocal tradicional, naturalmente surgen algunas interrogantes. ¿Por qué es tan diferente la música electroacústica respecto a formas más tradicionales de música? ¿Cómo escuchamos este tipo de música? ¿Cómo la percibimos? ¿Cómo podemos aproximarnos su análisis?

7.1. Características de la música electroacústica

7.1.1. Material sonoro

Smalley identifica tres tipos diferentes de sonidos [34]:

1. Sonidos naturales o ambientales capturados mediante micrófonos, sonidos que antes de su captura no posean propósito musical alguno.

2. Sonidos especialmente creados con un propósito musical, tales como sonidos vocales o instrumentales.

3. Sonidos electroacústicos, creados mediante síntesis o transformaciones de otros sonidos y totalmente separados de las otras dos categorías.

Una inmediata observación sobre esta categorización es que estos distintos tipos de sonidos difieren en sus fuentes sonoras. En el último caso es difícil identificar la fuente, mientras que en los dos primeros usualmente no necesitamos de ninguna pista visual para reconocerlas. Esto sugiere que la percepción de estos tipos de sonidos difiere en cada caso.

Desde otro punto de vista, también es importante clasificar los sonidos de acuerdo a su capacidad de referencia y significación.

Sonidos abstractos

El concepto de sonido abstracto está directamente relacionado con el concepto de audición reducida (ver sección 3.4) y con el término acusmático. Un sonido abstracto puede privilegiar perceptualmente alguna cualidad o

parámetro en el cual el compositor ha puesto su atención en dentro del estudio. Esto puede relacionarse con las técnicas empleadas en su creación o manipulación y/o en la contribución de ese sonido en particular al discurso musical de una obra. La percepción de un sonido abstracto, por lo tanto, está frecuentemente determinada por su contexto.

Un sonido puede ser denominado abstracto o acusmático simplemente por la incapacidad del auditor de asignarle una fuente sonora real o imaginaria y por la negación de una fuente visual [46]. Esto nos permite organizar y clasificar el material sonoro en base a sus características morfológicas y no en base a una referencia a la posible fuente de emisión sonora.

Muchos compositores de música electroacústica aprovechan este fenómeno y construyen un continuo entre lo abstracto y lo referente que funciona como un principio micro o macroestructural, o que determina la narrativa general de la música [6].

Sonido referencial

Un sonido referencial es un sonido que expone, sugiere, o al menos no oculta su fuente sonora. Un sonido de este tipo apunta en la dirección opuesta a la interioridad; se refiere a algo concreto, a un contexto determinado y no a criterios perceptuales intrínsecos. No es su origen real lo que importa, sino más bien su poder de evocar situaciones extrínsecas. Un sonido referencial incluso puede tener un origen sintético [6].

Es interesante notar que autores como Windsor creen que todos los sonidos poseen ambas cualidades y que no existen los sonidos puramente abstractos o referenciales [45]. Incluso el mismo Smalley, quien propuso esta clasificación en un comienzo, algunos años más tarde optó por los términos intrínseco/extrínseco para diferenciar ambas clasificaciones [35].

Por lo general los sonidos instrumentales son referenciales, por el simple hecho de que la fuente es conocida. Un sonido de violín, por ejemplo, puede ser muy abstracto en términos musicales pero el auditor siempre tiene la opción de hacer referencia al instrumento que emite el sonido. Un sonido creado artificialmente tiende a ser abstracto, dada la imposibilidad de asignarle una fuente real. No obstante, Smalley afirma que la mayoría de los sonidos tienen fuentes reconocibles e incluso, cuando dichas fuentes están ocultas, pueden ser percibidas fuentes sustitutas (ver sección 4.4) [33].

Usualmente, cuando uno considera el material sonoro de la música electroacústica, se trata de material que tradicionalmente no es considerado como musical [44]. Las técnicas electroacústicas han llevado a la inclusión de todos los tipos de sonidos anteriormente excluidos del discurso musical:

ruidos, sonidos no instrumentales, no vocales, obtenidos del medio ambiente o a través de la manipulación de otros sonidos previamente grabados, o puramente mediante síntesis. La música electroacústica usualmente presenta sonidos en una forma directa, inmediata, en contraposición con el mundo estructurado y altamente mediado de la música tradicional. A este respecto, Barry Truax nota, por ejemplo, que las líneas divisorias entre lenguaje, música y paisaje sonoro son cada vez más difusas cuando éstos son usados como material sonoro en el estudio [43].

Desde un punto de vista de la composición, los sonidos dejan de ser sólo soportes para las alturas o estructuras rítmicas. Su compleja multidimensionalidad se torna en el factor formal preponderante y gran parte de la composición electroacústica se basa en el valor del sonido por sí mismo y no solamente en relaciones entre sonidos [18].

7.1.2. Ausencia de notación y representación abstracta

El análisis musical, tal como lo conocemos, es un producto del siglo XIX y la representación abstracta de la música (partituras o equivalentes) es un pre-requisito analítico incluso más antiguo. Bennett y Camilleri sugieren que no existen herramientas analíticas o un vocabulario analítico para la música electroacústica, dado que ésta en general ni siquiera posee una representación de este tipo [3] [7].

La dificultad para pensar en términos abstractos acerca de la música electroacústica y la falta de un vocabulario adecuado hacen muy difícil la concepción de modelos analíticos que tengan una validez más allá de las experiencias personales del compositor o auditor. La música electroacústica existe sólo como sonido, existe sólo en el presente y al no tener una representación gráfica, no tiene un pasado. Desde un punto de vista analítico, una partitura o una representación gráfica de otro tipo nos permite detener el tiempo y formarnos una idea del discurso temporal de la música sin tener que lidiar con la urgencia del tiempo mismo. Una partitura no es la música en sí, sino sólo una intermediaria, una representación, cuyas características nos permiten pensar acerca de la música que representa.

La notación tiene dos grandes propósitos en la música de arte occidental [44]. Puede prescribir las acciones de los ejecutantes, actuando como un conjunto de restricciones sobre las variables interpretativas. En este sentido, la notación provee un medio mediante el cual un compositor puede tratar de determinar qué es lo que debe ser tocado, cundo y cómo. Una partitura puede ser considerada como un mensaje, con una fuente (el compositor) y un receptor (el o los intérpretes).

Sin embargo, una partitura también funciona como un ente descriptor en un sentido bastante más estricto. Es posible analizar una pieza musical tal como si fuera una representación de la pieza en sí misma. La notación musical provee al analista de una representación fidedigna y lista sobre lo que puede ser escuchado en una pieza musical cuando ésta es interpretada correctamente.

Frente a esta lectura estricta de la notación, es importante tener en cuenta que una partitura es sólo una de las fuentes de información sobre una pieza de música. Otros factores, tales como análisis acústicos, datos psicológicos o perceptuales, documentos históricos, impresiones de los intérpretes, compositores o auditores también proveen al analista de distintas e importantes descripciones de una pieza.

En consecuencia, dado que la música electroacústica no posee ningún tipo de partitura, que no hay en ella ninguna correlación entre una representación gráfica y sonido, el enfoque analítico no puede ser otro que de orden estético/perceptivo/cognitivo. Lo único que puede analizarse, entonces, es el discurso musical; lo que Camilleri llama el texto sonoro [7].

A pesar de esta aparente dificultad del enfoque analítico, al concentrarse el analista solamente en el material sonoro sin ninguna referencia escrita, se evita el riesgo de realizar un análisis basado en un solo aspecto (como, por ejemplo, la relación entre alturas), un proceder que comúnmente no tiene en cuenta lo que los auditores realmente escuchan.

7.1.3. Composición y análisis

De acuerdo con Camilleri, un aspecto interesante de la música electroacústica es que su análisis por lo general no coincide con un examen del proceso compositivo, como usualmente sucede con la música instrumental [7]. El proceso de composición en este caso sólo sirve para enfatizar y correlacionar algunas características físicas de las fuentes sonoras que dan origen a la pieza. Este es un hecho importante que no debe ser subestimado. Tradicionalmente uno de los objetivos del análisis es descubrir o explicitar las intenciones del compositor y su metodología. Generalmente esto no es posible en el caso de la música electroacústica. Lo que se infiere al escuchar una pieza de música puede tener muy poco que ver con la metodología del compositor, dadas las múltiples transformaciones y orígenes que el material sonoro puede tener.

En consecuencia, el análisis de la música electroacústica debe centrarse en otros aspectos y el divorcio entre compositor y la obra de arte una vez terminada es más pronunciado en este caso que en formas de música más

tradicionales. Tal como ya se mencionó, es habitual que los compositores de música electroacústica no consideren que el resultado de su arte sea música. Morthenson es bastante drástico en este sentido y prefiere directamente el término arte electrosónico [20].

7.1.4. Composición e improvisación

Stephen Pope sostiene que muchos compositores de música electroacústica que utilizan computadores realizan improvisaciones estructuradas en tiempo real, o en otras palabras, composiciones interactivas [25]. El debate acerca de donde termina la improvisación y donde empieza la composición es algo que toma extraordinaria fuerza en el caso de la música electroacústica. Sin duda, el concepto tradicional de composición aquí se ve amenazado.

7.1.5. Música electroacústica y su significado

Jan Morthenson sugiere que la música electroacústica no puede ser psicológicamente catalogada como música bajo las condiciones culturales actuales, porque no conlleva el significado esperado que usualmente le asignamos a la música [20]. Bajo su punto de vista, hay dos tipos de significado. Uno es el representado por un signo y el otro es el significado de colocar ese signo en particular en algún contexto o lugar específico. El primer significado es mecánico y automático y está basado en signos aprendidos, mientras el segundo depende de juicios, experiencias, situaciones y personalidades. El arte, en su forma de ver, no es un requisito necesario para la vida y por lo tanto no necesitamos aprender el significado de los signos o símbolos artísticos y estéticos.

De acuerdo a estas ideas, los eventos musicales representan fundamentalmente un significado interno, porque se refieren a otros eventos musicales que están por suceder o que han ocurrido recientemente. Para reafirmar esta idea, Morthenson cita a Leonard Meyer:

Si, en base a la experiencia pasada, un estímulo actual produce la expectativa de un evento musical posterior, entonces ese estímulo posee un significado. De esto se deriva que un estímulo o gesto que no apunta o no genera expectativas de un evento posterior carece de significado. Dado que la expectativa es un producto derivado de la experiencia estilística, cualquier música en un estilo con el cual no estemos familiarizados no nos significa nada

Este punto constituye la esencia del argumento de Morthenson: incluso la música contemporánea más extraña involucra a intérpretes haciendo cosas

con sus instrumentos, pero ese simple hecho le permite al auditor asignarle un significado hipotético porque aquello se relaciona con ocurrencias pasadas en la historia de la música. Se le puede asignar el segundo tipo de significado, como una variante de la música con todas sus implicaciones históricas.

En cambio, la música electroacústica por lo general no precisa de artistas en el escenario para influenciar las evaluaciones de los auditores. Más aún, la mayor parte del tiempo están ausentes las acciones otrora reconocibles de los compositores. En este tipo de música, usualmente no existe una discriminación evidente entre creatividad artística y sólo operaciones matemáticas o técnicas. Algunas veces el compositor se convierte en un supervisor, dado que son operaciones estadísticas las que deciden el curso de los eventos.

Los sonidos instrumentales tradicionales permiten distinguir la experiencia musical de cualquier otra experiencia auditiva. En el caso de la música electroacústica esto no sucede, porque los sonidos electroacústicos se relacionan más con el medio ambiente que con la música. Por lo tanto, los auditores no poseen referencias suficientes con las cuales comparar lo que están escuchando.

En resumen, cada pieza de música electroacústica es nueva y, de acuerdo con la definición de Meyer, musicalmente no posee significado. En consecuencia, la música electroacústica no puede ser considerada como música.

Este enfoque propuesto por Morthenson puede parecer hoy un poco radical, pero seguramente tuvo mucho sentido en el tiempo en que dichas ideas fueron escritas (1985). Actualmente existe una cultura sobre música electroacústica con referentes históricos suficientes como para poder comparar cada nueva pieza que se crea. Al respecto, Adkins introduce el concepto de cadena acústica, que consiste básicamente en vínculos que se establecen cuando un compositor se refiere al trabajo de otro en la propia música, no por una mención directa, sino porque utiliza los mismos objetos sonoros [1]. Una cadena acusmática de este tipo claramente establece una nueva categoría de significación para la música electroacústica.

7.2. Estrategias auditivas

Investigaciones recientes en música electroacústica sugieren que no escuchamos en una forma fija, uniforme u objetiva, sino que percibimos la música de maneras diversas y adoptamos distintas aproximaciones en nuestra audición. Más aún, es bastante frecuente que adoptemos estrategias auditivas diferentes durante el transcurso de una misma pieza.

Los aspectos más sobresalientes de las estructuras y superficies musicales

(espectromorfología) llevan a los auditores a adoptar estrategias diversas en la audición. Por ejemplo, una rápida secuencia monofónica de eventos sonoros autónomos no se escucha de la misma manera que una densa textura granular que evoluciona en el tiempo. Similarmente, es poco probable que escuchemos de la misma manera una pieza con una clara narrativa musical y otra con una superficie musical altamente abstracta.

7.2.1. Audición musical

Smalley menciona que la audición musical es muy distinta a la audición ordinaria. Escuchar no es lo mismo que oír [34]. Oír implica un acto involuntario, mientras que la audición conlleva un acto intencional, una voluntad de aprehender el sonido. De la misma manera, escuchar música es muy distinto a oírla. Cuando oímos una música en particular, solamente podemos escucharla si decidimos prestar atención. Tal como sugieren Jones y Yee, no es posible percibir, entender o recordar música si no prestamos atención a ella [17].

7.2.2. Modos Schaefferianos de audición

Pierre Schaffer identifica cuatro modos de audición [34]. Los primeros dos modos se relacionan con la audición ordinaria y los modos tres y cuatro guardan relación con la audición musical. Estos modos son:

1. Ècouter. El auditor se ocupa de la proveniencia de los sonidos y el mensaje que conllevan. La atención se enfoca en la ocurrencia o en los eventos conectados al sonido. Por ejemplo, si escuchamos el sonido de un automóvil atravesando una calle, no nos interesa el sonido en s mismo, sino lo que significa; probablemente el vehículo va muy rápido y nos podrá atropellar. Este modo trata al sonido como un signo de su fuente.

2. Ouìr. No hay intención de escuchar, pero un sonido nos llega y no podemos evitarlo. Un ejemplo de esto puede ser una explosión o un llanto súbito. Este es el nivel más crudo y elemental de percepción auditiva.

3. Entendre. Corresponde a un proceso selectivo donde algunos sonidos son preferidos respecto a otros. De acuerdo a su etimología, significa mostrar una intención de escuchar.

4. Comprendre. Un paso más allá del modo tercero. Implica la intención de aprehender un significado o unos valores, tratando al sonido como un signo que se refiere a este significado como una función de un lenguaje o de algún código (audición semántica).

7.2.3. Audición de fondo

Barry Truax propone el término audición de fondo (background listening) para diferenciar un nivel distinto de audición que ocurre cuando existen sonidos que permanecen en el fondo de nuestra atención [43]. Esto sucede cuando un auditor no escucha un sonido en particular que sin embargo está presente, debido a que ese sonido no tiene alguna significación inmediata o relevante. Sin embargo, el auditor es consciente de él y si se le pregunta si ha escuchado el sonido en cuestión probablemente responder de forma afirmativa. Los sonidos que son escuchados en el fondo ocurren frecuentemente y por lo tanto son esperados y predecibles. Se les presta atención solamente si es necesario; de lo contrario, no son siquiera notados.

7.2.4. Audición reducida

En la teoría de Schaeffer, la audición reducida corresponde a la actitud con la que el auditor escucha sonidos por su propio valor, como un objeto sonoro puro, carente de una fuente sonora y sin un significado atribuible a ésta. En la audición reducida, el foco es el objeto sonoro en sí mismo (no a lo que se refiere) y con los valores que conlleva (y no los que sugiere).

El nombre audición reducida se refiere a la noción de una reducción fenomenológica, porque consiste en descartar de la percepción auditiva todo lo que no le es propio, de manera que se llegue a escuchar solamente el sonido puro, en toda su materialidad, sustancia y dimensión. La audición reducida y el objeto sonoro se correlacionan directamente y se definen mutuamente como la actividad y el objeto de la percepción.

Dado que en la audición ordinaria el sonido siempre es tratado como un vehículo, entonces la audición reducida se convierte en un proceso antinatural, que va en contra de todo condicionamiento. El acto de remover las referencias auditivas habituales es un acto voluntario y artificial.

7.2.5. Modos auditivos de Smalley

Denis Smalley extiende los modos Schaefferianos al combinarlos con los conceptos de autocentricidad y alocentricidad de Schachtel (ver sección 4.3). Como resultado, propone los siguientes modos [34]:

1. Indicativo. Corresponde al primer modo de Schaeffer y considera al sonido como un mensaje y centrado en el objeto. En este caso, el sonido acta como una señal o un signo de algo que no es sonoro por naturaleza, algo necesario, por ejemplo, para la supervivencia.

2. Reflexivo. Este modo está centrado en el sujeto, es claramente autocéntrico y se basa en respuestas emocionales al objeto percibido. El objeto y la emoción no son separables en este caso.

3. Interactivo. Corresponde a los modos tres y cuatro de Schaeffer y es claramente alocéntrico. Involucra una relación activa de parte del sujeto para explorar continuamente las cualidades y estructuras del objeto. Esta modo abarca la audición estructural, actitudes estéticas hacia los sonidos y la música y el concepto de audición reducida de Schaeffer (ver sección 3.4).

Los modos indicativos y reflexivos son los dominantes. En cambio, el embarcarse en una relación interactiva requiere de esfuerzo y voluntad. A pesar de esto, en mi opinión, una relación de este tipo es la necesaria para apreciar realmente la música y constituye un requisito básico para el análisis.

7.2.6. Conductas auditivas de Delalande

La teoría analítica de Francois Delalande, basada en el análisis de comportamientos auditivos, obtiene conclusiones estructurales del estudio de diversos testimonios auditivos [10]. Delalande argumenta que la mejor forma de representar los distintos aspectos de una obra musical se consigue al basar el análisis en una amplia muestra de experiencias auditivas, con el objetivo de extraer los elementos estructurales y narrativos comunes a ellas. Este tipo de análisis es llamado por Delalande análisis de conductas auditivas (ver sección 5.3).

Delalande propone tres diferentes conductas auditivas: taxonómica, enfática y figurativa.

1. La audición taxonómica se manifiesta a través de la tendencia a distinguir grandes unidades morfológicas tales como secciones y hacer una lista mental de ellas. También se manifiesta en la habilidad para notar cómo estas unidades se relacionan e interconectan entre ellas.

2. La conducta enfática presta atención a las sensaciones, las cuales son comúnmente descritas como el producto psicológico del sonido. Esto está claramente relacionado con el modo reflexivo de Smalley.

3. En la audición figurativa el auditor tiende a pensar que los sonidos evocan algo que se mueve y que está vivo y presta su atención a la superficie musical, a los detalles sobresalientes.

7.3. Percepción de la música electroacústica

A pesar de que el campo de la percepción musical es bastante amplio, la percepción de la música electroacústica es un tema que an no ha sido estudiado en profundidad. En los últimos años ha habido un gran avance en la investigación sobre percepción de música instrumental o vocal, pero los estudios publicados sobre percepción de música electroacústica son todavía escasos.

A continuación se presentan los enfoques perceptuales más importantes que han sido propuestos hasta ahora

7.3.1. Enfoque ecológico

Luke Windsor propone que los sonidos sean entendidos con relación a los ambientes dinámicos en los cuales éstos son producidos, ya sean sonidos creados naturalmente o mediante alguna mediación cultural [45]. En consecuencia, las dificultades teóricas y analíticas que la música electroacústica posee requerirían soluciones basadas en una conciencia dual del auditor sobre los aspectos naturales y culturales del sonido.

En otras palabras, esto sugiere que la música electroacústica posee una naturaleza dual. Por una parte, los sonidos son percibidos y por la otra significan. Son percibidos porque proveen información que permite a un organismo reaccionar adecuadamente a su ambiente de una manera directa. Y también significan porque son interpretados como artefactos culturales con un contenido explícito.

Esta idea puede ser directamente relacionada con los modos tercero y cuarto de los modos de audición de Schaeffer (ver sección 3.2). El tercer modo (Entendre) implica seleccionar algunos sonidos sobre otros y el modo cuarto (Comprendre) implica asignar un significado a esa experiencia.

En el contexto de la acústica ecológica, los sonidos no son concebidos como entidades abstractas relacionadas una con la otra, como por ejemplo tonos de colores o timbres, ni tampoco como signos referentes a conceptos o cosas. Antes bien, se considera que los sonidos proveen un contacto directo entre los auditores y ocurrencias ambientales significativas [46].

Estas ideas están basadas en el trabajo de Gisbon, creador del enfoque ecológico para la percepción visual, quien afirma que la percepción no

requiere de la mediación de representaciones mentales del mundo externo. Este argumento es contrario al punto de vista dominante de la percepción y cognición musical, en el cual el procesamiento, el almacenamiento y la manipulación de la información en la mente son lo principal.

De acuerdo con Windsor, el enfoque ecológico asume que el ambiente está altamente estructurado y que los organismos son sensibles a dicha estructura. Sin embargo, esto no significa que la memoria, el lenguaje y otros sistemas simbólicos no tengan un rol importante en nuestra experiencia del mundo, sino que la percepción, por un lado; y los sistemas simbólicos que facilitan la mediación, el almacenamiento y la comunicación de las percepciones, por el otro, son cosas distintas.

Esta perspectiva ecológica se basa en la percepción de eventos. Este tipo de percepción trata de identificar las propiedades invariantes de eventos que especifican las características permanentes y cambiantes del ambiente que son importantes para un organismo. Este enfoque es distinto de la psicoacústica tradicional en el sentido de que ésta última se ocupa de identificar cambios en alturas, estructuras espectrales o duraciones, mientras que la primera trata de identificar las transformaciones en las estructuras acústicas que informan al auditor de cambios importantes en el medio ambiente.

Cabe destacar que este punto de vista no implica que los sonidos no puedan ser percibidos con ciertas cualidades, tales como los objetos sonoros de Schaeffer (ver sección 7.2.4) o que ellos no puedan significar. Lo que sí implica es que los sonidos no necesitan de tal mediación para informarnos acerca de nuestro ambiente y que las fuentes sonoras pueden difíciles de ignorar, debido principalmente a la naturaleza no musical de los sonidos electroacústicos.

En conclusión, esta visión es fundamentalmente opuesta al concepto acusmático de Schaeffer en el sentido de que la música electroacústica, más que proveer un método de lidiar con fuentes sonoras, es más bien una fuente informativa de estímulos acústicos estructurados. La percepción de música electroacústica puede ser radicalmente distinta a la percepción de eventos reales, pero esto no es sólo audición acusmática o reducida. El auditor que habita en un ambiente rico en estimulación y estructura percibir eventos no sólo a través de la información presente en una pieza de música sino del ambiente como un todo, sea éste acústico o no.

7.3.2. Paisajes sonoros

Un paisaje sonoro se define como la fuente imaginaria de los sonidos percibidos [47]. Por ejemplo, el paisaje sonoro de los sonidos escuchados en

una sala de conciertos está constituido por músicos tocando instrumentos. El paisaje sonoro de la misma música tocada a través de altoparlantes también corresponde a músicos tocando instrumentos.

¿Cuál es el paisaje sonoro de una pieza de música electroacústica? La respuesta a esta pregunta no es fácil y depende de cada pieza en particular. La mayoría de la gente acostumbrada a escuchar conciertos de música tradicional en vivo se desorienta al escuchar un concierto de música electroacústica presentado mediante altoparlantes. Esta desorientación se debe a la incapacidad de definir una fuente sonora imaginable, un paisaje sonoro, para los sonidos percibidos.

Smalley menciona que el extenso rango de fuentes sonoras que han sido incorporado a la música electroacústica ha provocado una revolución en el contenido sonoro, ante lo cual se observa una variada gama de respuestas auditivas [34]. Wishart propone que nuestra percepción de un paisaje sonoro puede ser descompuesta en tres partes: naturaleza del espacio acústico percibido, la disposición de los objetos sonoros en tal espacio y la identificación de objetos sonoros individuales [47].

7.3.3. Autocentricidad y alocentricidad

Ernest Schachtel establece una diferencia entre actividades perceptuales centradas en el sujeto y centradas en el objeto. Para hacer la diferenciación, propone dos modos perceptuales [34].

El modo autocéntrico es el basado en el sujeto y en respuestas básicas y sentimientos de satisfacción o insatisfacción. Este enfoque está relacionado con las reacciones subjetivas a los estímulos. Esta idea de percepción es primitiva, basada en necesidades y se asocia con las primeras relaciones del niño con el mundo. Cabe destacar aquí la similitud con la percepción ecológica propuesta por Windsor.

Por el contrario, el modo alocéntrico es el centrado en el objeto en el sentido de que no involucra las necesidades del individuo. La satisfacción o insatisfacción no son relevantes en este caso. Este modo implica un proceso activo y selectivo de focalización en un objeto para posteriormente discernir sus propiedades.

Las actitudes musicales autocéntricas se relacionan con respuestas emocionales al sonido. En cambio, las actitudes alocéntricas involucran la aprehensión de las estructuras musicales y la apreciación de sonidos fuera de un contexto musical. Este modo constituye un encuentro directo con el sonido y está directamente relacionado con el modo cuarto de Schaeffer (ver sección 3.2) y con el modo interactivo de Smalley (ver sección 3.5).

La autocentricidad y alocentricidad pueden existir en forma paralela y también varían de acuerdo con cada auditor. Hay algunos que en la presencia de un sonido inmediatamente utilizarán un enfoque autocéntrico mientras otros seguirán una estrategia alocéntrica. Esta idea complementa la naturaleza sonora dual que propone Windsor, en el sentido de que los sonidos son percibidos (en el sentido autocéntrico) pero también significan (de una manera alocéntrica). También esta idea puede vincularse directamente con las conductas auditivas de Delalande (ver sección 3.6).

7.3.4. Sustitución

Denis Smalley introduce el término sustitución (surrogacy) como una forma de describir el nivel o el grado en los cuales los auditores relacionan perceptualmente los sonidos con fuentes físicas y gestuales reales o imaginarias en un contexto acusmático. A medida que el nivel de sustitución se hace más fuerte, los auditores pueden asignar causas imaginarias a distintos sonidos y a su evolución espectromorfológica, o etiquetar el sonido como abstracto. Smalley denomina a estos varios niveles como órdenes de sustitución [33].

7.4. Estrategias analíticas

El análisis de la música, como teoría y praxis, ha pasado últimamente por un período de autocuestionamiento, sin duda como resultado de la incapacidad de las técnicas tradicionales de abordar música de la avant-garde de la post-guerra [21]. Incluso hoy, mucha música contemporánea parece inescrutable. Este es el contexto en el cual se encuentra la música electroacústica.

Tal como Delalande señala, el objeto del análisis es el sonido. Y los objetos sonoros (ver sección 5.1) existen sólo en las mentes de los auditores [10]. Camilleri afirma que la introducción de la tecnología en sus varias fases de desarrollo desde 1940 hasta hoy ha provocado no solo un enriquecimiento de la gama sonora, sino que también ha generado una gran cantidad de reflexiones teóricas acerca de cómo clasificar los sonidos de las nuevas obras y como analizarlas [7]. Para algunos autores, entre ellos Nicola Sani, el rol del análisis debe ser no sólo el proveer análisis individuales sobre obras específicas, sino más bien identificar la especificidad de este género en el contexto de la producción artística de nuestro tiempo [31].

La música electroacústica se basa fuertemente en el uso de la tecnología, la cual avanza a una velocidad vertiginosa. En el momento de analizar,

esto se convierte en un problema, dado que existe una fuerte tendencia a basar el análisis en los aspectos técnicos [31]. La música computacional y electroacústica se presenta y legitima como una parte del conocimiento (know-how) tecnológico, lo cual no puede estar más lejos de la verdad. El conocimiento tecnológico hace posible la creación de ciertas composiciones, pero no puede ser la justificación estética de las mismas. En este sentido, el análisis de una pieza de música electroacústica no debe convertirse en un manual de instrucciones o una gua de uso de la tecnología.

Hay dos cosas que hay que considerar al momento de analizar música electroacústica. En primer lugar, el punto de partida del análisis es la audición de la obra. Segundo, la actitud receptiva del auditor influye en el análisis porque sus propiedades no son meramente asociativas, sino una parte estructural de la potencialidad del medio electroacústico [7].

Camilleri sugiere que las estrategias analíticas para la música electroacústica deben considerar tres aspectos:

1. Reflexión del lenguaje musical, sus potencialidades y sus ligazones al mundo sonoro natural

2. Relación entre las propiedades psicoacústicas del fenómeno sonoro y su descripción

3. Creación de un léxico analítico

Estos tres aspectos son esenciales e igualmente importantes, porque permiten crear un marco apropiado al proceso de análisis. Una de las características de la música electroacústica es que puede fusionar fácilmente sonidos naturales y artificiales de forma casi imperceptible. Este solo hecho obliga al analista a reflexionar sobre el tipo de lenguaje musical utilizado. El establecer relaciones estructurales entre los elementos presentados en la música es crucial. El análisis musical no es una mera descripción, y esto también es válido para la música electroacústica. Además, dado que aún no existen métodos y estrategias generales para este tipo de análisis, tampoco existe un léxico analítico, por lo que un objetivo importante de cualquier nueva estrategia debe ser el proveer un marco de referencia (léxico) consistente y exhaustivo.

A continuación se presentan las principales metodologías analíticas propuestas hasta ahora.

7.4.1. Objetos sonoros

La teoría tipomorfológica de Pierre Schaeffer, a pesar de que no constituye realmente una metodología analítica sino más bien compositiva y teórica, ha sido utilizada por muchos como un punto de partida para el análisis de música electroacústica. En su propuesta, Schaeffer identifica tres tipos de objetos sonoros, presentes en el discurso musical, de acuerdo a criterios topológicos y morfológicos; los cuales indican los tipos generales de estos objetos y sus características respectivamente. En esencia, los tres tipo de objetos sonoros propuestos por Schaeffer son los continuos, los iterativos y los impulsivos. Esta clasificación claramente se basa en las características morfológicas de los sonidos, y ser extendida más adelante por Smalley, tal como se describe en la sección 7.4.2.

Schaeffer también considera tres planes de referencia a través de los cuales los objetos sonoros son descritos y clasificados:

- Plan melódico o de textura: la evolución de las alturas en el tiempo
- Plan dinámico o formal: los parámetros de la intensidad en el tiempo
- Plan armónico o tímbrico: las relaciones entre los parámetros anteriores y sus componentes espectrales

Cada plan de referencia presenta diversos sistemas de clasificación de acuerdo a los tipos de movimientos melódicos, dinámicos y tímbricos. Además, Schaeffer establece cuatro clases de criterios de clasificación: material, mantenimiento, forma y variación, los cuales se correlacionan con las características morfológicas de los objetos sonoros.

7.4.2. Espectromorfología

Denis Smalley desarrolló el concepto de espectromorfología como una herramienta para describir y analizar experiencias auditivas [33] [35]. Las dos partes del término se refieren al sonido (espectro) y la evolución de su molde o forma de onda (morfología) a través del tiempo. El espectro no puede existir sin la morfología: algo tiene que ser moldeado y debe poseer también contenido sonoro.

Desde un punto de vista analítico, la propuesta de Smalley es extremadamente importante, dado que en los primeros treinta y cinco años de actividad electroacústica no fue propuesta ninguna metodología comprehensiva para el análisis. La teoría de Pierre Schaeffer sobre tipos morfológicos

o tipomorfología (objetos sonoros, ver sección 5.1) fue el intento más serio. La espectromorfología es más exhaustiva, puede ser aplicada a distintos contextos musicales y extiende las ideas de Schaeffer al incluir estructuras jerárquicas y otras propiedades como movimiento y espacio [7].

Es importante destacar, no obstante, que la espectromorfología es una herramienta descriptiva basada en la percepción auditiva. Un enfoque espectromorfológico considera modelos y procesos espectrales y morfológicos y provee un marco de referencia para entender relaciones estructurales y comportamientos sonoros experimentados en el flujo temporal de la música [35]. El marco teórico de la espectromorfología se articula en cuatro partes principales: los arquetipos espectrales, morfología, movimiento y procesos estructurales. A continuación se detallan las primeras dos partes, dado que son las más relevantes para efectos del presente texto.

Arquetipos espectrales

Smalley identifica tres tipos básicos de espectros: nota, nodo y ruido. La nota corresponde a un tono musical normal, con una frecuencia fundamental y armónicos. Un nodo corresponde a una densidad de sonido más compleja, donde la identificación de un croma o altura determinada es difícil. Un ruido es una densidad sonora extremadamente comprimida donde no hay identificación de una altura o croma posible.

Cada uno de estos arquetipos espectrales puede tener un comportamiento temporal distinto, que Smalley clasifica en arquetipos morfológicos.

7.4.3. Arquetipos morfológicos, modelos y cadenas

Smalley va mucho más allá de las definiciones de Schaeffer al designar diversos arquetipos morfológicos y crear una serie de modelos obtenidos a través de diversas transformaciones tales como la retrogradación. Pueden discernirse tres arquetipos básicos en las fuentes de sonidos: ataque impulsivo, ataque resonante y continuo graduado.

El ataque impulsivo corresponde a un rápido impulso que es inmediatamente terminado. En este caso el inicio del sonido es también su final. La atención se enfoca por completo en el ataque súbito. El ataque resonante es modelado en sonidos cuyo ataque impulsivo es extendido a través de una resonancia (como por ejemplo una cuerda pulsada o una campana) para luego decaer ya sea en forma rápida o graduada dependiendo del caso. La atención se enfoca en la sorpresiva aparición y en la forma en que ésta se convierte en un sonido sostenido. El tercer arquetipo es un continuo graduado el cual

es modelado en sonidos sostenidos. El ataque es graduado, dando paso a un comportamiento continuo que luego decae en forma también graduada. En este caso el ataque es percibido como una influencia mucho más débil que en el caso de los otros dos arquetipos. La atención se enfoca principalmente en la forma en que el sonido es mantenido. Desde estos tres arquetipos principales puede generarse una sutil variedad de articulaciones temporales.

Smalley extiende estos arquetipos en una mayor cantidad de modelos morfológicos. Estos modelos permiten lo que Smalley denomina modulación morfológica, dado que proveen un marco de referencia más completo y versátil para la clasificación de unidades morfológicas estructurales. No obstante, estas morfologías no son solamente objetos aislados sino que pueden enlazarse y mezclarse formando cadenas, creando as objetos híbridos.

7.4.4. Análisis basado en conductas auditivas

Delalande [10] hace una crítica explícita de la espectromorfología de Smalley. De acuerdo con sus propuestas, hay un tipo de análisis simplemente inaceptable: uno que apunte a demostrar lo que la música es, o cual sea la forma o la estructura. En consecuencia, uno debe descartar la ilusión de un análisis único y definitivo. Delalande enfatiza que al analizar música electroacústica (o incluso un sonido medianamente complejo) es siempre posible revelar una infinidad de características morfológicas y distintas configuraciones de morfologías. Esto no sólo se aplica a todas las piezas de música sino también a todos los objetos sonoros. Un análisis siempre implica una reducción. Sólo algunas características y configuraciones son seleccionadas de entre todas las posibles. La pregunta del objetivo del análisis y la pertinencia de las características en relación con este objetivo es, por lo tanto, evadida.

Como solución a esto, Delalande propone el curso de acción opuesto. El objetivo del análisis, según sus ideas, es revelar las elecciones (implícitas o explícitas) y acciones del compositor, o explicar los comportamientos auditivos de los receptores (ver sección 3.6), o ambos al mismo tiempo. Solamente después que un objetivo analítico ha sido elegido es posible definir un criterio de pertenencia. En un nivel general y teórico no hay razón alguna para que las unidades pertinentes coincidan con unidades espectromorfológicas.

En sus análisis, Delalande realiza en forma separada análisis taxonómicos, enfáticos y figurativos de la misma pieza. Los resultados son totalmente distintos, pero complementarios. Cada comportamiento da su propia forma a los objetos sonoros.

7.4.5. El sonograma

La música electroacústica frecuentemente contiene cambios espectrales y otros aspectos tímbricos que hacen inadecuada la notación musical. En estos casos, representaciones visuales de la música pueden ayudar a entender y apreciar las características que pueden haberse ignorado sin este soporte [14]. Al respecto, Mara Helmut propone el sonograma como una representación útil del espectro de frecuencias de una pieza y de su evolución en el tiempo. En adición a esto, propone la utilización de capas adicionales de información relevante tales como relaciones de alturas, fraseo, dinámicas locales y globales y técnicas empleadas. El sonograma se propone como una interesante herramienta de análisis y que ha sido utilizada para entender el diseño sonoro de obras electroacústicas.

Un sonograma consiste básicamente en un gráfico bi-dimensional en donde el eje X corresponde al tiempo y el eje Y al eje de frecuencias. Distintos colores son utilizados para representar las intensidades individuales de los componentes de frecuencia presentes en la música. A pesar de que la información visual que un sonograma proporciona no corresponde directamente al sonido, puede proveer información útil de la música y servir como un complemento a otras formas de análisis.

7.4.6. Análisis narrativo

Cada proceso creativo desarrolla su propia narrativa, incluso en forma independiente de los propósitos del creador [13]. En su trabajo, Giomi y Ligabue proponen un análisis narrativo de música electroacústica utilizando un proceso analítico basado en aspectos perceptuales y una metodología estética-cognitiva, desarrollada por los propios autores. Este análisis pretende operar en forma análoga al análisis de una obra literaria, donde existen protagonistas y antagonistas, y una trayectoria narrativa claramente establecida.

Este tipo de análisis permite el examen de los objetos analíticos desde distintos puntos de vista, empezando por unidades pequeñas como los objetos o eventos sonoros para seguir con cadenas sintagmáticas, estrategias compositivas y la segmentación de la estructura formal de la pieza. Esta división estructural se basa en criterios tales como material, comportamiento rítmico- dinámico, coherencia, morfología tímbrica, densidad, movimiento y tensión. Este análisis también considera estrategias de comienzo y final, el uso de la analepsia (repetición narrativa) y la prolepsia (anticipación narrativa), asociaciones semánticas tanto musicales como metafóricas y asociaciones

psicológicas extra musicales.

7.5. Conclusiones

La música electroacústica es, sin dudas, un género particular y distinto a la música instrumental. El sólo hecho de utilizar cualquier fuente sonora en un contexto musical establece una gran diferencia. La utilización de sonidos no musicales en un contexto musical genera todo tipo de interrogantes acerca de la forma en que percibimos dichos sonidos y su significado. Propuestas como la audición reducida de Schaeffer intentan enfocar la atención auditiva sólo en aspectos morfológicos, obviando toda referencia externa. Por el contrario, autores como Windsor y Wishart argumentan que los sonidos no pueden dejar de significar y que una audición reducida no es posible. Tal como Smalley propone, aún cuando no seamos capaces de asignar una fuente real o virtual a un sonido electroacústico, podemos asignarles una fuente sustituta.

Autores como Morthenson cuestionan la música electroacústica como música y ofrecen diversos argumentos por los cuales ésta no debiera ser considerada música. Algunos compositores también cuestionan el rol de la composición en este contexto. Desde mi punto de vista, creo que la música electroacústica puede perfectamente ser considerada como un género más dentro de la música, a pesar de sus diferencias, y aún sigo pensando que es posible componerla, toda vez que se trata de posicionar sonidos en el tiempo. Sin lugar a dudas, la música electroacústica genera serios cuestionamientos perceptuales y teóricos, debido en gran parte a su naturaleza, pero también debido a su poco tiempo de vida. Este género está naciendo, y posee, según mi parecer, un futuro muy rico e interesante.

Cada una de las propuestas analíticas mencionadas en este texto son importantes de considerar ya que se enfocan en distintos aspectos de la música. La espectromorfología de Smalley se basa en las propiedades físicas (acústicas) del sonido, mientras Delalande considera las características psicoacústicas. El sonograma permite observar la estructura espectral y en enfoque narrativo busca estructuras y estrategias narrativas identificables en la música. Cada uno de estas ideas tiene fortalezas y debilidades.

Personalmente, creo que la mejor forma de analizar música electroacústica es usar la mayor cantidad de estrategias como sea posible y tratar de abordar todos los aspectos de la obra.

Estoy convencido que la música electroacústica ha cambiado las bases teóricas del análisis musical. El análisis ya no se basa exclusivamente en una

partitura escrita y las intenciones del compositor ya no son el principal foco del análisis. Lo que ahora importa es el resultado sonoro.

Camilleri sugiere que la falta de una partitura en este caso no es una debilidad sino una fortaleza. Un modelo teórico y analítico basado solo en el texto sonoro representa un punto de contacto real entre la teoría musical y el modelamiento de estrategias musicales perceptuales y cognitivas.

A mi modo de ver, la música electroacústica no sólo ha cambiado sino ampliado el campo del análisis. Bajo este punto de vista, es posible afirmar que el análisis tradicional es un subconjunto de un tipo más general de análisis como el descrito en éste texto. El lector habrá notado que cada una de las estrategias descritas aquí podrán perfectamente ser aplicadas a música tradicional e incluso música popular.

Para finalizar, me gustara enfatizar que desde 1940 se ha generado una intensa actividad en este campo y no hay señales de que esto se haya acabado en ningún sentido. La investigación en música electroacústica sigue su curso y nuevas ideas y enfoques son propuestos constantemente. Tal como Camilleri desea, quizás en un futuro cercano podamos disponer de un léxico analítico común y sólidamente establecido.

Apéndice A

Ejemplos en SuperCollider

El siguiente apéndice contiene una introducción al lenguaje de síntesis SuperCollider y ejemplos tanto de síntesis de sonido como procesamiento digital desarrollados en este lenguaje.

A.1. SuperCollider: introducción

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 1
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  http://www.audiosynth.com/
6
7  SuperCollider es:
8
9  - Editor de texto
10 - Lenguaje de programacion (interprete)
11 - Compilador
12 - Sintetizador digital
13
14 todo en uno.
15
16 SuperCollider es:
17 - Open Source y gratis
18 - Originalmente implementado en computadores Apple Macintosh
19 - La version 3 esta disponible fundamentalmente para plataformas
       POSIX (Unix, Linux, OSX).
20 - Tambien existe un port actualmente en desarrollo para Windows.
21
22 Historia:
23 - SuperCollider versiones 2 and 3d5.1 para MacOS 8 and 9
```

```
24 - SC Server originalmente para MacOSX
25 - SuperColider3 (Open Source)
26
27 Arquitectura
28
29 SuperCollider tiene una estructura cliente/servidor.
30
31 Las operaciones en SuperCollider estan divididas en un
       interprete del lenguaje de programacion  de SuperCollider (
       sclang) y un servidor de sintesis (scserver). Estas dos
       aplicaciones se comunican mediante Open Sound Control a
       traves de UDP o TCP.
32
33 Esto significa entre otras cosas:
34 - El servidor puede ser controlado por otros programas aparte de
        sclang
35 - El interprete puede caerse pero el servidor sigue corriendo
36 - El servidor puede caerse y el interprete sigue corriendo
37 - El interprete y el servidor pueden correr en maquinas
       distintas, incluso en distintas partes del mundo, permitiendo
        una mejor distribucion de recursos en la red
38
39 Desventajas: esta arquitectura produce latencia. No latencia de
       audio, la cual es muy baja sino latencia de comunicacion.
       Esto se minimiza usando el servidor interno (internal).
40
41 - Cada lado cumple una funcion distinta.
42 - scserver es un programa simple y eficiente dedicado a tareas
       de audio.
43 - No tiene idea de codigo, objetos, OOP, o cualquier cosa
       relacionada con el interprete.
44
45 Servidor interno y local
46 - Servidor local: corre un proceso aparte del interprete, sin
       compartir memoria
47 - Servidor interno: corre un proceso que comparte memoria con el
        interprete, y por lo tanto permite cosas como osciloscopios
       y ademas minimiza latencia.
48
49 Servidor por defecto.
50 - Siempre hay un servidor por defecto, que esta almacenado en la
        variable de clase default.
51 - Cualquier sintetizador (Synths) o grupo (Groups) creado sin un
        destino son creados en el servidor por defecto.
52 - Al momento de inicio, el servidor por defecto es el servidor
       local, pero esto puede ser cambiado.
53
54
55 Ejemplo de codigo SuperCollider
```

```
56
57 // definir el servidor por defecto como interno
58 Server.default = Server.internal;
59
60 // asignarlo a la variable 's'
61 s = Server.default;
62
63 // partir el servidor
64 s.boot;
65
66 // definir un sintetizador y "enviarlo" al servidor
67 SynthDef("sine", { Out.ar(0, SinOsc.ar(440, 0, 0.2)) }).send(s);
68
69 // correr la sintesis
70 s.sendMsg("s_new", "sine", n = s.nextNodeID, 0, 1);
71
72 // mostrar el osciloscopio interno (OSX)
73 Server.internal.scope(1);
74
75 // parar el sintetizador (delete)
76 s.sendMsg("/n_free", n);
77
78 // parar el servidor (opcional)
79 s.quit;
80
81
82 Ejemplos:
83
84     (
85     Server.default = Server.internal;
86     s = Server.default;
87     s.boot;
88     )
89
90     "Hello World".speak;
91
92         {SinOsc.ar(LFNoise0.ar([10, 15], 400, 800), 0, 0.3)}.
               play;
93
94         {
95                 RLPF.ar(
96                         LFSaw.ar([8, 12], 0, 0.2),
97                         LFNoise1.ar([2, 3].choose, 1500, 1600),
98                         0.05, mul: 0.4
99                 )
100        }.play;
```

A.2. SuperCollider: el lenguaje

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 2
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  El lenguaje SuperCollider (SC)
6
7  sclang:
8  - Orientado al objeto
9  - Bastante similar al lenguaje SmallTalk
10 - Sintaxis es similar a C++.
11
12 Sentencias
13 Las sentencias SC son terminadas por un punto y coma ";"
14
15 Literales
16 Los literales son valores que tienen una representacion
       sintactica directa
17
18 Numeros
19 Un entero es una serie de digitos precedidos opcionalmente por
       un signo menos.
20 Ejemplos :
21         -13
22         666
23         2112
24         96
25
26 Un numero real (float, de punto flotante) esta compuesto por
       digitos seguidos por un punto decimal despues del cual hay
       mas digitos.
27 Deben haber digitos a ambos lados del punto decimal. Por ejemplo
       :
28
29     0.0
30     3.1415
31     -0.5
32
33 Esto distingue numeros reales de expresiones como:
34
35         8.rand
36
37
38 Tambien se puede utilizar notacion exponencial.
39
40         1.2e4
41         1e-4
```

```
42
43 La constante pi se puede agregar a un numero para crear una
       constante de punto flotante:
44
45         2pi
46         0.5pi
47         -0.25pi
48
49
50 Valores especiales
51
52 Las instancias singulares de las clases True, False, Nil y
       Infinitum se escriben: true, false, nil y inf.
53 True y False son valores booleanos (Boolean).
54
55 x = true;
56 y = false;
57 z = nil;
58 n = inf;
59
60
61 Variables
62
63 Identificadores
64
65 Los nombres de variables (y muchas otras cosas) empiezan con un
       caracter alfabetico en minusculas seguido de cero o mas
       caracteres alfanumericos.
66
67 Las variables son "case sensitive":
68
69         estaesmivariable no es lo mismo que estaEsMiVariable
70
71 Por convencion, los nombres de variables usan la siguiente
       convencion:
72
73         unaCosa, otraCosa
74
75 Las variables se declaran con la clave 'var':
76
77 var abc, z123, func;
78
79 Las variables pueden ser inicializadas de la siguiente manera:
80
81 var abc=3, z123=6;
82
83 SuperCollider define las letras del alfabeto automaticamente
       como variables y, por lo tanto, pueden ser utilizadas sin ser
        declaradas.
```

```
84  Esto es conveniente para demostraciones rapidas de codigo, pero
        no es aconsejable para codigo real.
85
86  Alcance (scope) de variables
87  Los programas SC se organizan ente pares de parentesis.
88  Las variables se definen y son entendidas entre un par de
        parentesis, pero no afuera de ellos.
89
90  algoAqui (
91          var miNumero, myOtroNumero;
92          miNumero = miOtroNumero;
93          /* Estas variables son reconocidas aqui */
94  );
95  miOtroNumero = miNumero;
96  /* Pero no aqui!!! */
97
98  Nombres de clases
99
100 Algunas palabras usadas en programas son asociadas con una parte
        del lenguaje llamada "clases".
101 Los nombres de clases siempre empiezan con una letra mayuscula.
102 Las "unidades de generacion" (Unit Generators o UGen) que
        modifican y crean audio son todas clases.
103
104 Object
105 Point
106 Synth
107 Osc
108 Reson
109
110
111 Comentarios
112
113 Los comentarios son lineas de codigo que no son tomadas en
        cuenta por el interprete (o compilador)
114 Empiezan con // y terminan en el final de la linea.
115
116 Tambien pueden ser delimitados con /* y */.
117 Ejemplos:
118
119         // comentarios de una linea
120
121         /*
122                 comentario
123                 multi
124                 linea
125         */
126
```

```
127         /* A diferencia de C, se pueden tener comentarios /*
                anidados */ */
128
129
130 Expresiones
131
132 Las expresiones son combinaciones de literales, variables y
        operadores que producen un resultado.
133 Los operadores pueden ser operaciones aritmeticas comunes.
134
135 Ejemplos de operadores
136
137         1 + 2           // suma de uno y dos
138         a - b           // diferencia de a y b
139
140 Precedencia de operadores
141
142 No hay. Todos los operadores tienen el mismo nivel de
        precedencia y se asocian de izquierda a derecha. Por ejemplo,
         la expresion:
143
144                 x = a * b + c * d;
145
146         es equivalente a:
147
148                 x = ((a * b) + c) * d;
149
150         y no:
151
152                 x = (a * b) + (c * d);
153
154 Por lo tanto, bueno utilizar parentesis en las expresiones.
155
156
157 Asignaciones
158
159 Asignacion simple. Una asignacion simple asigna el valor de una
         expresion en el lado derecho a una variable en el lado
        izquierdo.
160 Una asignacion simple tiene la forma:
161
162 <variable> = <expresion>;
163
164 Ejemplo:
165
166 c = a + b;
167
168 cup = java;
169
```

```
170 resultado = esto * aquello + mas;
171
172 signal = SinOsc.ar(800, 0, 0.1);
173
174 Una asignacion no es lo mismo que el signo igual.
175 Una asignacion transfiere algo del lado derecho al izquierdo.
176
177
178 Objetos, Mensajes
179
180 - SuperCollider es un lenguaje orientado al objeto.
181 - Todas las entidades del lenguaje son objetos.
182 - Un objecto es algo que tiene datos que representan el estado
      del objeto y un set de operaciones que se pueden aplicar a el
      .
183 - Todos los objetos son instancias de alguna clase que describe
      la estructura del objecto y sus operaciones.
184 - Los objectos en SuperCollider incluyen numeros, cadenas de
      caracteres, colecciones, unidades generadoras, ondas de audio
      , puntos, rectangulos, etc.
185
186 El estado interno de un objecto es capturada por sus variables
      instanciadas.
187
188 - Las operaciones sobre los objectos son invocadas mediante
      mensajes.
189 - Un mensaje es un requerimiento por un objecto, llamado el
      receptor (receiver) que ejecutan alguno de sus metodos
      instanciados.
190 - El metodo a ejecutar depende de la clase que define al objeto.
191 - Objetos de clases distintas pueden implementar el mismo
      mensaje de formas distintas.
192 - Por ejemplo, las clases de unidades generadoras responden al
      mensaje "ar" y "kr".
193 - "ar" provoca al objeto ejecutar procesamiento de audio a la
      tasa de audio y "kr" a la tasa de control.
194
195
196 Notacion de objetos
197
198 Para acceder o cambiar el valor de una variable de objecto se
      usa la notacion de punto.
199
200         miInstrumento.freq = 440.0;      // setear la variable
                freq del objecto miInstrumento
201         s = Server.default;                    // accesar una
                instancia de variable, en este caso de una Clase
202
203 Notacion de mensajes
```

```
204 - Un mensaje consiste de un selector de mensaje, que nombra el
      tipo de operacion, una receptor al cual el mensaje es enviado
       y en algunos casos una lista de argumentos que otorgan
      informacion adicional relacionada a la operacion en cuestion.
205 - Un mensaje siempre produce un resultado.
206 - El tipo de resultado depende del tipo de mensaje. Por ejemplo,
       las unidades generadoras retornan senales de auio.
207 - Los mensajes pueden ser escritos usando operadores (como los
      matematicos), notacion funcional o de receptor.
208
209 Notacion funcional
210 - El selector de mensaje precede la lista de parentesis.
211 - El primer argumento en la lista es el receptor.
212 - Las operaciones pueden ser unarias o binarias.
213
214 Ejemplos:
215         sin(x)                    // sino of x----unaria
216         max(a, b)                 // maximo entre a y b----binaria
217
218
219 Notacion de receptor
220 - Una llamada a un metodo en notacion funcional se puede
      convertir a notacion de receptor al poner el receptor antes
      del nombre del metodo seguido por un punto (notacion de punto
      ).
221
222
223         1.25.sin    es equivalente a    sin(1.25)
224         x.cos    es equivalente a  cos(x)
225
226         -1.max(3)  equivale a  max(-1,3)
227         f(a, b)    equivale a    a.f(b)
228
229 Los mensajes de las UGen se escriben siempre en notacion de
      receptor:
230
231         Osc.ar(800.0, 0.0, 0.4)
232
233 SuperCollider tiene una gran coleccion de operadores unarios y
      binarios, muchos de los cuales son especificos para audio.
234
235 Operadores unarios  (ver [SimpleNumber])
236         ampdb   dbamp                     reciprocal
237         cpsmidi midicps cpsoct  octcps  midiratio
                ratiomidi
238         abs             ceil            floor           frac
                    sign            neg
239         sqrt            squared cubed   distort softclip
```

```
240         sin             cos             tan             asin
                    acos            atan
241                                                         sinh
                                                                cosh
                                                                tanh
242         exp             log             log2            log10
243         isNegative              isPositive
                isStrictlyPositive
244         x.rand  x.rand2
245
246 Operadores binarios (ver [SimpleNumber])
247         +       -       *       /       **              %
248         div             pow             max             min
249         thresh  trunc           amclip  clip2
250         round   scaleneg        excess  fold2           wrap2
251         difsqr  sumsqr  sqrsum  sqydif  absdif
252         atan2   hypot   hypotApx
253         ring1   ring2   ring3   ring4
254         x.rand(y)       x.exprand(y)
255
256
257 Argumentos claves
258 - Los argumentos (parametros) para metodos pueden ser
      especificados por el nombre mediante el cual son declarados
      en una clase.
259 - Tales argumentos son llamados argumentos claves (keyword
      arguments).
260 - Si un argumento clave y un argumento posicional especifican el
       mismo argumento, el argumento clave tiene preferencia,
261 - Por ejemplo, el metodo 'ar' de la clase SinOsc toma los
      argumentos freq, phase, mul, and add, en ese orden.
262
263 Todos los ejemplos siguientes son llamadas validas para ese
      metodo
264
265         SinOsc.ar(800, pi, 0.2, 0);
266
267         {SinOsc.ar(800, pi, 0.2, 0)}.play;
268
269         // freq = 800, mul = 0.2, others get default values.
270         SinOsc.ar(800, mul: 0.2);
271
272         // freq = 800, phase = pi, mul = 0..
273         SinOsc.ar(phase: pi, mul: 0.2, freq: 800);
274
275         // 1200 predomina por sobe 800
```

```
276         SinOsc.ar(800, freq: 1200);
277
278         SinOsc.ar(zorg: 999); // argumento invalido
279
280
281 Argumentos por defecto
282 - Se pueden llamar a metodos con menos argumentos definidos en
        la clase correspondiente.
283 - Si esto sucede, dichos argumentos se inicializan envalores por
        defecto, si es que estos estan definidos en la clase, o si
        no se les asigna el valor nulo (nil).
284 - Si se pasan argumentos en exceso, generalmente se coleccionan
        en un arreglo (Array) o son ignorados.
285 - Si se llama a un metodo sin argumentos, los parentesis pueden
        ser omitidos.
286
287
288         // x es declarada para tomar dos argumentos, a y b son
                valores por defecto de 1 y 2.
289         // Retorna la suma de los argumentos
290         x = { arg a=1, b=2; a + b };
291
292         z = x.value;                    // z vale 3. (a = 1, b =
                2)
293
294         z = x.value(10);                // z vale  12. (a es 10,
                b = 2)
295
296     z = x.value(10, 5);      // z vale 15. (a es 10, b es 5)
297
298         z = x.value(10, 5, 9);  // z vale 15. (a es 10, b es 5,
                9 es ignorado)
```

A.3. SuperCollider: Unidades generadoras

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 3
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  (
6          Server.default = Server.internal;
7          s = Server.default;
8          s.boot;
9  )
10
11 Unidades generadoras (UGen)
```

```
12 - Objetos que procesan o generan sonidos
13 - Pueden tener muchas entradas, pero una sola salida
14 - Canales multiples se pueden hacer mediante arreglos de
       unidades generadoras
15
16 Creacion, tasa de audio, tasa de control
17 - Se crean a traves de los mensajes ar o kr
18
19 FSinOsc.ar(800, 0.0, 0.2); // crear un oscilador sinusoidal a
       800 Hz, fase 0.0, amplitud 0.2
20
21 Argumentos
22 - Pueden ser otras UGen, escalares (numeros) o arreglos de UGen
       o escalares
23 - Por convencion, los ultimos dos argumentos de la mayoria de
       las UGen son mul (multiplicador) y add (suma)
24 - Valores por defecto de mul=1.0 y add=0.0
25 - Usar mul y add es mas eficiente que escribir expresiones con
       multiplicaciones y adiciones
26
27 Notacion funcional
28
29         { SinOsc.ar(440, 0, 0.2) }.play;
30
31 - Objeto se espefica entre { y } y se llama al metodo play
32 - Como no se especifica servidor, se usa el por defecto (
       variable "s").
33 - Tampoco se almaceno nada en una variable, por lo que no se
       puede reusar.
34
35 (
36         {                                   // Abrir la funcion
37                 SinOsc.ar(          // Crear un objeto SinOsc a tasa
                       de audio
38                         440,      // frecuencia de 440 Hz, LA
39                         0,                // fase inicial 0
40                         0.2)      // mul (amplitud) of 0.2
41         }.play;                     // cerrar la funcion y llamar '
               play' sobre ella
42 )
43
44 Otro ejemplo,
45
46 (
47         { var ampOsc;               // Abrir la funcion y declarar
               una variable
48                 ampOsc = SinOsc.kr(0.5, 1.5pi, 0.5, 0.5);
                       // Crear un objeto SinOsc a tasa de control
```

```
49                                          // asignar su resultado
                                                a una variable
50                  SinOsc.ar(440, 0, ampOsc);      // crear un
                        objecto SinOsc a tasa de audio
51                                          // y usar la variable
                                                para controlar
                                                amplitud
52          }.play;
53  )
54
55
56  Conceptos basicos de osciladores
57
58  Forma mas facil de generar sonido
59
60          { SinOsc.ar(800, 0, 0.1) }.play;
61
62  Que pasa en caso de errores
63
64          { SinOsc.ar(800, 0, 0.1) },.play;
65
66  Ejemplo de ruido browniano:
67
68          { BrownNoise.ar(0.1) }.play;
69
70  Server, BrownNoise y SinOsc son objetos
71
72  play, y ar son mensajes
73
74  800, 0, y 0.1 son literales
75
76
77  Ejemplo con aliasion (alasing o foldover)
78
79  { SinOsc.ar(Line.kr(1,43000,6), 0, 0.1) }.play;
80
81
82  Frecuencia lineal versus tono (pitch)
83
84  Line crea una interpolacion lineal entre dos puntos
85  XLine crea una interpolacion exponenial, usualmente usada para
        frecuencia
86
87  ar es tasa de audio
88  kr es tasa de control
89
90  { SinOsc.ar(Line.kr(400,4000,5), 0, 0.3) }.play;  // Suena como
        una curva
91
```

```
92  { SinOsc.ar(XLine.kr(400,4000,5), 0, 0.3) }.play; // Suena
        lineal
93
94
95  freq, phase, mul, and add are the names of arguments
96
97  Otros ejemplos:
98
99          { SinOsc.ar(freq: 440, phase: 0, mul: 0.4, add: 0) }.
                play;
100
101         { SinOsc.ar(phase: 0, freq: 440, add: 0, mul: 0.4) }.
                play;
102
103         { SinOsc.ar(freq: 440, mul: 0.4) }.play;
104
105 freq:440 es un ejemplo de argumento clave
106
107
108 SC tiene muchos otros objetos que son osciladores:
109
110 Por ejemplo:
111         SinOsc          FSinOsc         Osc
                Buffer  OscN
112
113 Tambien son utiles:
114         Impulse         Blip                    Pulse
                LFPulse
115         Saw                     LFSaw
116         Klang           Formant
117         VOsc            VOsc3
118
119 Algunos un poco mas complicados:
120         SyncSaw VarSaw
121         LFPar           LFCub
122         Gendy1          Gendy2          Gendy3
```

```
1   TUTORIAL DE SUPERCOLLIDER 3 - PARTE 4
2   Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3   Northwestern University
4
5   (
6   Server.internal.boot;
7   Server.default = Server.internal;
8   s = Server.default;
9   )
10
11
12
```

```
13 Mas sobre generadores de senal
14
15 Sumar sinusoides es ineficiente:
16
17 ({
18         var f=200;
19         a = SinOsc.ar(f,0,0.1);
20         b = SinOsc.ar(2*f,0,0.1);
21         c = SinOsc.ar(3*f,0,0.1);
22         d = SinOsc.ar(4*f,0,0.1);
23         e = SinOsc.ar(5*f,0,0.1);
24         a+b+c+d+e;
25 }.play;
26 )
27
28
29 Look at the Help pages for the following:
30         SinOsc          FSinOsc          Osc
               Buffer                  OscN
31
32 These are also useful:
33         Impulse         Blip                    Pulse
               LFPulse
34         Saw                     LFSaw
35         Klang                   Formant
36         VOsc                    VOsc3
37
38
39
40 Generadores de senal de banda limitada
41
42 SinOsc, FSinOsc, Blip, Saw, Pulse, Formant no produciran
       aliasion.
43
44
45 SinOsc, FSinOsc
46 arguments:  frequency, phase, mul, add
47
48 { SinOsc.ar(800,0,0.5) }.play;
49
50 { FSinOsc.ar(800,0,0.5) }.play;
51
52
53 Saw
54 argumentos:  frequency, mul, add
55
56 { Saw.ar(800, 0.5) }.play;
57 { Saw.ar(XLine.kr(200,20000,6),0.5) }.play;  // no aliasing
58
```

```
59
60 Blip
61 argumentos:  frequency, numHarmonics, mul, add
62
63 { Blip.ar(800, 10, 0.5) }.play;
64 { Blip.ar(XLine.kr(200,20000,6),10,0.5) }.play; // no aliasing
65 // modular el numero de armonicos
66 { Blip.ar(400,Line.kr(1,30,20),0.2) }.play;
67 //  a traves de un fitro pasa bajos resonante
68 { RLPF.ar(Blip.ar(400,30,0.5), XLine.kr(400,8000,5), 0.1) }.play
       ;
69
70
71 Pulse
72 argumentos:  frequency, width, mul, add
73
74 { Pulse.ar(800,0.3, 0.5) }.play;
75 { Pulse.ar(XLine.kr(200,20000,6),0.3,0.5) }.play;
76 // modulate pulse width
77 { Pulse.ar(400, Line.kr(0.01,0.99,8), 0.5) }.play;
78
79
80 Klang - banco sinusoidal de osciladores
81 argumentos:  `[ frequencies, amplitudes, phases ], mul, add
82
83 { Klang.ar(`[ [800, 1000, 1200],[0.3, 0.3, 0.3],[pi,pi,pi] ], 1,
       0) * 0.4}.play;
84 // 8 frecuencias aleatorias
85 { Klang.ar(`[ {exprand(400, 2000)}.dup(8), nil, nil ], 1, 0) *
       0.04 }.play;
86
87 Formant                    oscilador de formante
88 argumentos: kfundfreq, kformfreq, kwidthfreq, mul, add
89
90 Generates un set de armonicos sobre una frecuencia de formante,
       dada una frecuencia fundamental
91 kfundfreq - fundamental frequency in Hertz.
92 kformfreq - formant frequency in Hertz.
93 kwidthfreq - pulse width frequency in Hertz. Controla el ancho
       de banda del formante.
94
95 // modula frec fund, frec.formantes constante
96 { Formant.ar(XLine.kr(400,1000, 8), 2000, 800, 0.125) }.play;
97
98 // modula frec formante, frec fundamental constante
99 { Formant.ar(200, XLine.kr(400, 4000, 8), 200, 0.125) }.play;
100
101 // modula mediante frecuencia, otras quedan constantes
102 { Formant.ar(400, 2000, XLine.kr(800, 8000, 8), 0.125) }.play;
```

```
103
104
105
106 Generadores de senal basados en tablas
107
108
109 Osc, COsc, VOsc, VOsc3
110
111 Es necesario usar un buffer alocado en el servidor
112 (
113         // allocate buffer with server, size, numChans,
                bufferNumber
114         b = Buffer.alloc(s, 2048, 1, bufnum: 80);
115         // fill buffer with array of amplitudes, 3 * true (
                default)
116         b.sine1([1,0.5,0.33,0.25,0.2], true, true, true);
117 )
118
119 // se puede escribir tambien cmo
120         b.sine1(1.0/[1,2,3,4,5], true, true, true);
121 // o:
122         b.sine1(1.0/(1..12));
123         b.sine1(1.0/(1..24));
124         b.sine1(1.0/(1..32));
125
126
127 Osc
128 argumentos: buffer number, frequency, phase, mul, add.
129
130 { Osc.ar(b.bufnum, 400, 0, 0.5) }.play;
131
132 COsc - dos osciladores, detuned
133 argumentos: buffer number, frequency, beat frequency, mul, add.
134
135 { COsc.ar(b.bufnum, 200, 0.4, 0.3) }.play;
136 // change buffer as above.
137
138 { COsc.ar(b.bufnum, 200, MouseX.kr(0.0,20.0,'linear'), 0.3) }.
        play;
139
140
141 VOsc - oscilador crossface de multiples wavetables
142 argumentos: buffer number, frequency, phase, mul, add.
143
144 (
145         // allocar buffers 80 a 87
146         b = Array.new(8);
147         8.do {arg i; b = b.add(Buffer.alloc(s, 2048, 1, bufnum:
                80+i))};
```

```
148 )
149
150 (
151         // llenar buffers 80 a 87
152         8.do({arg i;
153                 var n, a;
154                 // generar arreglo de amplitudes armonicas
155                 n = (i+1)**2; // armonicos:
                        [1,4,9,16,25,36,49,64]
156                 a = {arg j; ((n-j)/n).squared }.dup(n);
157                 // llenar tabla
158                 b[i].sine1(a);
159         });
160 )
161
162 // go!
163 { VOsc.ar(MouseX.kr(80,87), 120, 0, 0.5) }.play;
164
165
166
167 VOsc3 - tres VOscs sumados.
168 argumentos: buffer number, freq1, freq2, freq3, beat frequency,
        mul, add.
169
170 // chorusing
171 { VOsc3.ar(MouseX.kr(80,87), 120, 121.04, 119.37, 0.2) }.play;
172
173 // acordes
174 { VOsc3.ar(MouseX.kr(80,87), 120, 151.13, 179.42, 0.2) }.play;
175
176
177 Generadores de baja frecuencia (LF)
178
179 LFTri, Impulse, LFSaw, LFPulse, VarSaw, SyncSaw
180
181 Formas de onda geometricas, no limitados en banda
182
183 LFTri, LFSaw, Impulse
184 argumentos:  frequency, phase, mul, add
185
186 { LFTri.ar(200,0,0.5) }.play;
187 { LFSaw.ar(200,0,0.5) }.play;
188 { Impulse.ar(200,0,0.5) }.play;
189
190  // aliasion?
191 { LFTri.ar(XLine.kr(200,20000,6), 0, 0.5) }.play;
192
193
194 LFPulse, VarSaw
```

```
195 argumentos:  frequency, phase, width, mul, add
196
197 { LFPulse.ar(200,0,0.3,0.5) }.play;
198 { VarSaw.ar(200,0,0.3,0.5) }.play;
199 // pulso con modulacion
200 { LFPulse.ar(200,0,MouseX.kr(0,1),0.5) }.play;
201 { VarSaw.ar(200,0,MouseX.kr(0,1),0.5) }.play;
202
203
204 SyncSaw
205 argumentos:  syncFreq, sawFreq, mul, add
206 Onda diente de sierra vinculada con una frecuencia fundamental.
207 Efecto similar a formantes que se mueven o puls con modulacion.
208
209 { SyncSaw.ar(100, MouseX.kr(100, 1000), 0.1) }.play;
```

A.4. SuperCollider: Envolventes

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 5
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  (
6  Server.internal.boot;
7  Server.default = Server.internal;
8  s = Server.default;
9  )
10
11 Envolventes (envelopes)
12
13 Envelope: Env
14 Puede tener cualquier numero de segmentos y puede parar en algun
       lugar en particular o loopear varios segmentos al hacer
      sustain.
15
16
17 Creacion
18
19 *new(levels, times, curves, releaseNode, loopNode)
20
21         levels - arreglo de niveles
22         times - arreglo de duraciones
23         curve - tipo de curva para los segmentos
24         Valores posibles son:
25                 'step' - segmento plano
26                 'linear' - segmento lineal
```

```
27                  'exponential' - exponencial natural de caida y
                        crecimiento
28                  'sine' - segmento sinusoidal con forma de S
29                  'welch' - segmento sinusoidal basado en una
                        ventana Welch
30                  un Float - valor de curvatura para todos los
                        segmentos
31                  arreglo de Floats - valores de curvaturas para
                        cada segmento
32          releaseNode - un entero o nulo
33          loopNode - un entero o nulo
34
35
36                  Env.new([0.001,1,0.3,0.8,0.001],[2,3,1,4],'
                        linear').plot.test;
37                  Env.new([0.001,1,0.3,0.8,0.001],[2,3,1,4],'
                        exponential').plot.test;
38                  Env.new([0.001,1,0.3,0.8,0.001],[2,3,1,4],'sine'
                        ).plot.test;
39                  Env.new([0.001,1,0.3,0.8,0.001],[2,3,1,4],'step'
                        ).plot.test;
40                  Env.new([0.001,1,0.3,0.8,0.001],[2,3,1,4], -2).
                        plot.test;
41                  Env.new([0.001,1,0.3,0.8,0.001],[2,3,1,4], 2).
                        plot.test;
42                  Env.new
                        ([0.001,1,0.3,0.8,0.001],[2,3,1,4],[0,3,-3,-1])
                        .plot.test;
43
44
45  Metodos de envolventes de duracion fija
46
47
48          *linen(attackTime, sustainTime, releaseTime, level,
                curve)
49          *triangle(duration, level)
50          *sine(duration, level)
51          *perc(attackTime, releaseTime, peakLevel, curve)
52
53  Metodos de envolvente con un segmento sostenido
54
55          *adsr(attackTime, decayTime, sustainLevel, releaseTime,
                 peakLevel, curve)
56          *asr(attackTime, sustainLevel, releaseTime, peakLevel,
                curve)
57
58
59
60
```

```
61 EnvGen: generador de envolvente
62
63     "Ejecuta" envolventes (Env)
64         *ar(envelope, gate, levelScale, levelBias, timeScale,
               doneAction)
65
66         envelope                an instance of Env (or an array
               of Controls)
67         gate                    a control signal that holds the
               EnvGen open (except Env.linen)
68         levelScale              scales the levels of the
               breakpoints.
69         levelBias               offsets the levels of the
               breakpoints.
70         timeScale               scales the breakpoint durations.
71         doneAction      the doneAction arg causes the EnvGen to
               stop or end the
72                                 synth without having to use a
                                       PauseSelfWhenDone or
                                       FreeSelfWhenDone ugen.
73                                 It is more efficient to use a
                                       doneAction (see below)
74
75 Usando Env y EnvGen juntos
76
77 Enfoque simple:
78
79 {
80         var env, amp;
81         env = Env.perc(attackTime:0.2);
82         amp = EnvGen.kr(envelope: env, levelScale: 0.5,
               doneAction: 2);
83         SinOsc.ar(freq: 440,phase: 0,mul: amp)
84 }.play;
85
86 Se pueden incluir envolventes multiples y generadores
87
88 {
89         var env1, env2, amp1, amp2;
90         env1 = Env.perc(attackTime:0.2, releaseTime: 2);
91         env2 = Env.linen(attackTime:0.2, sustainTime: 0.2,
               releaseTime: 14, level:0.6);
92         amp1 = EnvGen.kr(envelope: env1, levelScale: 0.1);
93         amp2 = EnvGen.kr(envelope: env2, levelScale: 0.5,
               doneAction: 2);
94
95         Blip.ar(freq: 440, numharm: 20, mul: amp1) +
96                                 SinOsc.ar(freq: 2*440, phase: 0,
                                       mul: amp2)
```

```
97  }.play;
98
99
100 Triggers (gatilladores) y Gates (compuertas)
101
102 Ejemplo: implementacion directa de una Gate
103
104         (
105         SynthDef("env-help", { arg out, gate;
106                 var z;
107                 z = EnvGen.kr(Env.adsr,gate) * SinOsc.ar
                        (440,0,0.1);
108                 Out.ar(out, z)
109         }).send(s);
110         )
111
112         s.sendMsg("/s_new", "env-help", 1980); // empezar un
                synth en forma silenciosa
113
114         // prender
115         s.sendMsg("/n_set", 1980, \gate, 1);
116
117         // apagar
118         s.sendMsg("/n_set", 1980, \gate, 0);
119
120         // da lo mismo el valor mientras sea > 0
121         s.sendMsg("/n_set", 1980, \gate, 2);
122
123         s.sendMsg("/n_free", 1980);
124
125
126 Gates como Triggers
127
128 ( {  // Impulso va desde 0 a valores positivos a una tasa
        regular
129         e = Env.perc(0.001, 1.0);
130         t = Impulse.kr(2); // frequencia de los impulsos
131         a = EnvGen.kr(e,t);
132          SinOsc.ar( 440, mul:a )
133 }.play )
134
135  ( {  // Impulsos a valores aleatorios
136         e = Env.perc(0.001, 1.0);
137         t = Dust.kr(2); // numero promedio de impulsos
138         a = EnvGen.kr(e,t);
139          SinOsc.ar( 440, mul:a )
140 }.play )
141
142
```

```
143 ( {  // MouseX crea el trigger
144         e = Env.perc(0.001, 1.0);
145         t = MouseX.kr(-0.1, 0.1);
146         a = EnvGen.kr(e,t);
147          SinOsc.ar( 440, mul:a )
148 }.play )
149
150 ( {
151         var triggerSpeed;
152
153         triggerSpeed = 2;
154         e = Env.perc(0.001, 1/triggerSpeed);
155
156         // Trigger es usado para TRand y EnvGen
157         t = Impulse.kr(triggerSpeed);
158         f = TRand.kr(100, 2000, t);
159         a = EnvGen.kr( e,  t );
160
161         SinOsc.ar( f, mul:a);
162 }.play
163 )
164
165
166 Gates
167
168
169 ( {  // MouseX create la gate
170         e = Env.adsr(0.001, 0.3, 0.2, 0.1);
171         t = MouseX.kr(-0.1, 0.1);
172         a = EnvGen.kr(e,t);
173          SinOsc.ar( 440, mul:a )
174 }.play )
175
176 ( {  // LF Pulse create la gate
177         e = Env.adsr(0.001, 0.3, 0.2, 0.1);
178         t = LFPulse.kr(0.8, 0, MouseX.kr(0.0, 1.0)); // Mouse
              controla el ancho
179         a = EnvGen.kr(e,t);
180          SinOsc.ar( 440, mul:a )
181 }.play )
182
183
184 ( {  // LFNoise create la gate
185         e = Env.adsr(0.001, 0.3, 0.2, 0.1);
186         t = LFNoise0.kr(3);
187         a = EnvGen.kr(e,t);
188          SinOsc.ar( 440, mul:a )
189 }.play )
```

A.5. SuperCollider: SynthDefs

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 6
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  (
6  Server.internal.boot;
7  Server.default = Server.internal;
8  s = Server.default;
9  )
10
11 SynthDefs (definicion de sintetizador)
12 - Equivalente a un "patch" or "preset"
13 - Se puede escribir codigo para un SynthDef
14 - SynthDefs con "compilados" por sclang en una especie de
       bytecode
15 - Se pueden precompilar y almacenar en disco
16 - Tienen dos partes: un nombre y una funcion (ugenGraphFunc)
17
18 SynthDef(
19         "aSynthDef",
               // primer argumento: nombre
20         { .... yo soy una ugenGraphFunc ... }   // segundo
               argumento: ugenGraphFunc
21 )
22
23 Ejemplo:
24 To make the template functional
25
26 (
27 a = SynthDef(
28         "aSynthDef",                                    //
               nombre
29         {
                       // funcion
30                 arg freq = 440;                         //
                       argumentos a la funcion
31                 x = SinOsc.ar(freq, 0, 0.5);
32                 Out.ar(0, x)
33         }
34 ).load(s);
35 )
36
37 load escribe la definicion y la envia al servidor
38
39 a.play(s);
40 or
```

```
41 a.play;
42
43 play ejecuta la synthdef
44
45
46 Notes:
47
48 Arguments are important because these can change from note to
       note.
49
50 The audio output of Synths is generally handled with the Out
       unit generator.
51
52         *ar(bus, channelsArray)  - write a signal to an audio
               bus.
53
54 The output bus numbers start at 0 and go up by integer for each
       additional channels.  The stereo output channels are 0 and 1.
55
56
57
58 Synth
59 - Representacion en el cliente de un nodo de sintetizador en el
       servidor
60 - Representa una unica unidad productora de sonido
61 - Lo que hace depende de su SynthDef
62
63
64 Ejemplos:
65
66
67 y = Synth.new("aSynthDef");
68 o
69 y = Synth("aSynthDef", nil, s);
70 o
71 y = Synth.new("aSynthDef", [\argument1, value,\argument2, value,
       etc.], s);
72
73 y.free;
74
75 // Con argumentos
76 y = Synth("aSynthDef", [\freq, 262]);
77 z = Synth("aSynthDef", [\freq, 392]);
78 y.free;
79 z.free;
80
81
82
83 Ejemplos
```

```
84
85 (  // DEFINIR UN SYNTHDEF
86         SynthDef( "Ringing", { arg freq;        // frequencia es
               el unico argumento
87                 var out;
88                 out = RLPF.ar(
89                         LFSaw.ar( freq, mul: EnvGen.kr( Env.perc
                               , levelScale: 0.3, doneAction: 2 )),
90                         LFNoise1.kr(1, 36, 110).midicps,
91                         0.1
92                 );
93                 4.do({ out = AllpassN.ar(out, 0.05, [0.05.rand,
                       0.05.rand], 4) });
94                 Out.ar( 0, out );
95         }).send(s);
96 )
97
98 ( {  // Tocar una nota
99         Synth( "Ringing", [ \freq, 440 ] );
100 }).play(s);
101
102
103 (       // Tocar dos notas al mismo tiempo
104         ( {
105                 Synth( "Ringing", [ \freq, 440 ] );
106                 Synth( "Ringing", [ \freq, 660 ] );
107         }).play;
108 )
109
110 (       // Para tener pausas, Synth estar dentro de una Routine
111         Routine({
112                 Synth( "Ringing", [ \freq, 440 ] );
113                 1.0.wait;
114                 Synth( "Ringing", [ \freq, 660 ] );
115         }).play;
116 )
117
118 (       // Loop dentro de la Routine permite repetir un patron
119         Routine({
120                 loop ({
121                         Synth( "Ringing", [ \freq, 440 ] );
122                         1.0.wait;
123                         Synth( "Ringing", [ \freq, 660 ] );
124                         1.0.wait;
125                 });
126         }).play;
127 )
128
129
```

```
130 Convirtiendo de forma antigua a nueva
131
132 Ejemplo anterior
133
134 {
135         var env, amp;
136         env = Env.perc(attackTime:0.2);
137         amp = EnvGen.kr(envelope: env, levelScale: 0.5,
                doneAction: 2);
138         SinOsc.ar(freq: 440,phase: 0,mul: amp)
139 }.play;
140
141 Usando SynthDef y frecuencia como argumento
142
143 (   // DEFINIR UN SYNTHDEF
144         SynthDef( "SinEnv", { arg frequency;
145                 var out, env, amp;
146                 env = Env.perc(attackTime:0.2);
147                 amp = EnvGen.kr(envelope: env, levelScale: 0.5,
                        doneAction: 2);
148                 // argument is now used as the frequency value
149                 out = SinOsc.ar(freq: frequency,phase: 0,mul:
                        amp);
150                 Out.ar( 0, out );
151         }).send(s);
152 )
153
154 (       // Tocar dos notas
155         ( {
156                 Synth( "SinEnv", [ \frequency, 440 ] );
157                 Synth( "SinEnv", [ \frequency, 660 ] );
158         }).play;
159 )
160
161 (       // Loop
162         Routine({
163                 loop ({
164                         Synth( "SinEnv", [ \frequency, 440 ] );
165                         1.0.wait;
166                         Synth( "SinEnv", [ \frequency, 660 ] );
167                         1.0.wait;
168                 });
169         }).play;
170 )
171
172 Se puede agregar un argumento para el canal de salida
173
174 (   // DEFINICION
175         SynthDef( "SinEnv", { arg frequency, chan;
```

```
176                 var out, env, amp;
177                 env = Env.perc(attackTime:0.2);
178                 amp = EnvGen.kr(envelope: env, levelScale: 0.5,
                        doneAction: 2);
179                 // argument is now used as the frequency value
180                 out = SinOsc.ar(freq: frequency,phase: 0,mul:
                        amp);
181                 Out.ar( chan, out );
182         }).send(s);
183 )
184
185 (       // Tocar dos notas
186         ( {
187                 Synth( "SinEnv", [ \frequency, 440,\chan, 0 ] );
188                 Synth( "SinEnv", [ \frequency, 660, \chan, 1] );
189         }).play;
190 )
191
192 (       // Loop
193         Routine({
194                 loop ({
195                         Synth( "SinEnv", [ \frequency, 440,\chan
                                , 0 ] );
196                         1.0.wait;
197                         Synth( "SinEnv", [ \frequency, 660, \
                                chan, 1 ] );
198                         1.0.wait;
199                 });
200         }).play;
201 )
202
203 Ejemplo mas complejo
204 ( // Flujo de tonos basado en una escala de Do Mayor
205         var stream, dur, cScale;
206
207         dur = 1/4;
208         cScale = [60, 62, 64, 65, 67, 69, 71, 72];
209         // Routines pueden generar secuencias de tonos
210         stream = Routine({
211                 loop({cScale.choose.yield});
212         });
213         // Rountines pueden ser tocadas
214         Routine({
215                 loop({
216                         Synth( "Ringing", [ \freq, stream.next.
                                midicps ] );
217                         dur.wait;       // used by .play to
                                schedule next occurence
218                 })
```

```
219         }).play
220 )
221
222 ( // Ejemplo anterior, esta vez los tonos son aleatorios
223         var stream, dur;
224         dur = 1/4;
225         // esta rutina produce una secuencia randon de tonos
226         stream = Routine.new({
227                 loop({
228                         if (0.5.coin, {
229                                 // run of fifths:
230                                 24.yield;       // C1
231                                 31.yield;       // G1
232                                 36.yield;       // C2
233                                 43.yield;       // G2
234                                 48.yield;       // C3
235                                 55.yield;       // G3
236                         });
237                         rrand(2,5).do({
238                                 // arpegio variado
239                                 60.yield;
                                                        // C4
240                                 #[63,65].choose.yield;
                                        // Eb, F
241                                 67.yield;
                                                        // G
242                                 #[70,72,74].choose.yield;
                                        // Bb, C5, D
243                         });
244                         // melodia aleatoria
245                         rrand(3,9).do({  #[74,75,77,79,81].
                                choose.yield });  // D5, Eb, F, G, A
246                 });
247         });
248         // tocar la rutina
249         Routine({
250                 loop({
251                         Synth("Ringing", [\freq, stream.next.
                                midicps ]);
252                         dur.wait;
253                 })
254         }).play
255 )
```

A.6. SuperCollider: Modulación

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 7
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  (
6  Server.internal.boot;
7  Server.default = Server.internal;
8  s = Server.default;
9  )
10
11
12 Amplitude Modulation
13
14 Type #1 Ring/Balanced/Double-Sideband-Suppressed-Carrier
       Modulation
15
16 The amplitude of one signal varies according to another signal.
17
18 Slow amplitude modulation, that is, when the modulating
       frequency is less than 20 Hz., can
19 sound like a steady rhythm.  But when the modulating frequency
       is above 20 Hz, it sounds
20 like a continuous tone with sidebands.
21
22 {SinOsc.ar(1000,0) * SinOsc.ar(Line.kr(1,50,30),0) }.play;
23 // Nothing more than two sinusoids multiplied
24
25
26 Slow modulation of the modulator frequency reveals the sound of
       sidebands moving up and down simultaneously.
27
28 {
29         var modFreq;
30         modFreq = Line.kr(0, 600, 30); // Modulation frequency
               goes from 0 to 600 Hz
31         SinOsc.ar(1000,0) * SinOsc.ar(modFreq,0)
32 }.play;
33
34 The multiplication of one signal by another is better done with
       the mul: argument.
35
36 {
37         var modFreq, amp;
38         modFreq = Line.kr(0, 600, 30);
39         amp = SinOsc.ar(modFreq,0);
40         SinOsc.ar(1000,0,amp); // Rewritten for faster multiply
41 }.play;
42
43
```

```
44
45 Type #2 AM/Double-Sideband Modulation
46
47 Here the modulation index provides a way of controlling the
       amount of modulation
48 and simultaneously the amount of energy in the sidebands.
49
50 This example is easy to see in the time domain.
51
52 {
53         var modIndex, modSignal;
54         modIndex = Line.kr(0,1,12);              // Modulation
               Index goes from 0 to 1
55         // Normalization is needed if you want to keep the
               signal level constant
56         modSignal = (1 + (modIndex * SinOsc.ar(378,0))) / (1 +
               modIndex);
57         modSignal * SinOsc.ar(1000,0,0.5);
58 }.play;
59
60 Below we use a conditional to mimic the behavior of an analog
       system. Overmodulation occures
61 when the modIndex is greater than 1.  Here like in an analog
       system, the modution signal
62 can not become negative and therefore the process in non-linear
       and produces many sidebands.
63
64 {
65         var modIndex, modSignal;
66         modIndex = Line.kr(0,5,10);
67         modSignal = (1 + (modIndex * SinOsc.ar(79,0))) / (1 +
               modIndex);
68         modSignal = modSignal.max(0); // passes just the
               positive part of the control signal
69         modSignal * SinOsc.ar(1000,0,0.5);
70 }.play;
71
72
73
74 Frequency Modulation
75
76 In the following example the frequency of the audio rate
       oscillator is modulated by the control rate oscillator.
77
78 Carrier frequency is 700 Hz and the frequency deviation is 300
       Hz.  The control frequency is only 0.5 Hz.
79
80 {
81         SinOsc.ar(               // carrier oscillator
```

```
82                  SinOsc.ar(      // modulator oscillator
83                          0.5,            // modulating frequency
84                          0,              // zero phase
85                          300,            // mul field is the
                                freqency deviation
86                          700             // add field is the
                                carrier frequency
87                                          // These settings of mul
                                                and add will cause
                                                the
88                                          // frequency to vary
                                                from -300+700 to
                                                +300+700 Hz
89                                          // which equals from 400
                                                to 1000 Hz
90                  ),
91                  0,                      // zero phase
92                  0.5                     // amplitude 0.5
93          )
94  }.play;
95
96
97  The control frequency will now increate linearly from 0.5 to 100
        Hz. Note the changes in the resulting sound from a sine
        wave with changing pitch to a complex tone with multiple
        sidebands.
98
99  // sweep freq of LFO
100 {
101         SinOsc.ar(                      // carrier oscillator
102                 SinOsc.ar(              // modulator oscillator
103                         XLine.kr(               // make an
                                exponential line generator for
                                modulating freuqency
104                                 1,              // begin at 1 Hz
105                                 1000,   // end at 1000 Hz
106                                 20              // in 20 seconds
107                         ),
108                         0,                      // zero phase
109                         300,            // mul field is the
                                freqency deviation
110                         1000            // add field is the
                                carrier frequency
111                 ),
112                 0,                              // zero phase
113                 0.5                             // amplitude 0.5
114         )
115 }.play;
116
```

```
117 By changing the frequency deviation (and thus, the modulation
        index) the distribution of energy in the sidebands changes.
118
119 (The nested style used above can become hard to read, so often
        it is preferable to use variables to make it  more readable.)
120
121 This one makes it easy to see the frequency modulation in the
        time domain.
122
123 {
124         var freqDev, freq;              // declare the variables
                we will use in this function.
125
126         freqDev = Line.kr(              // make a linear line
                generator
127                                 ({
128         var fundFreq, peakFMI, carrierFreq, modulatorFreq, env1,
                env2, amp, instFreqDv, instFreq;
129         fundFreq = 440.0;       // fundamental frequency
130         c = 3; m = 1;    // All harmonics
131         peakFMI = 2;
132         carrierFreq = c * fundFreq;             // C:M ratio
                determines the carrier and
133         modulatorFreq = m * fundFreq;    //  the modulator
                frequencies
134
135         env1 = Env.new
                ([0,5.0,3.75,3.0,0]/5,[0.15,0.15,0.15,0.15],'linear')
                ;
136         amp = EnvGen.kr(env1);
137         //env2 = Env.new([3*peakFMI, peakFMI, 0.7 * peakFMI
                ],[0.3,0.45],'linear');
138         env2 = Env.new([0.3*peakFMI, peakFMI, 2.0 * peakFMI
                ],[0.3,0.45],'linear');
139         instFreqDv = modulatorFreq * EnvGen.kr(env2);
140
141         instFreq = SinOsc.ar(modulatorFreq,0,instFreqDv,
                carrierFreq);
142         SinOsc.ar(instFreq,0,amp);
143 }.play;)
144 0,              // begin at 0 Hz
145                                 600,            // end at 600 Hz
146                                 20              // in 20 seconds
147                         );
148
149         freq = SinOsc.ar(       // modulating signal
150                         100,            // modulating frequency
                            fixed at 100 Hz
151                         0,              // zero phase
```

```
152                         freqDev,        // mul field is the
                                frequency deviation
153                         1000            // add field is the
                                carrier frequency
154                 );
155
156         SinOsc.ar(              // create a sine oscillator
157                 freq,           // frequency modulator
158                 0,                      // zero phase
159                 0.5                     // amplitude 0.5
160         )
161 }.play;
162
163
164 The example below implements a very simple Chowning FM
        instrument in which
165         the carrier and the modulating frequency are identical
            and one envelope controls the amplitude and the
166         modulation index.
167
168 {
169                 var env, amp, instFreqDv, instFreq;
170                 //env = Env.linen(3.0, 3.0, 3.0, 0.5);
171                 env = Env.linen(0.1, 0.2, 0.3, 0.5);
172                 amp = EnvGen.kr(env);   // amplitude envelope
173
174                 // instantaneous frequency = carrierFreq +
                        FreqDv * Sin(modulatorFreq)
175                 // carrierFreq = modulatorFreq = 500 Hz
176                 // FreqDv = 2000 Hz
177                 instFreqDv = amp * 2000.0;  // amp envelop is
                        used to change
178                                 // instantaneous frequency
                                        deviation
179                 instFreq = SinOsc.ar(500,0,instFreqDv,500);  //
                        modulator = 500 Hz
180                 SinOsc.ar(instFreq,0,amp);
                                        // carrier = 500 Hz
181 }.play;
182
183 Here is an example in which the C:M ratios and peakFMI can be
        altered.
184
185 ({
186         var fundFreq, peakFMI, carrierFreq, modulatorFreq, env,
                amp, instFreqDv, instFreq;
187         fundFreq = 400.0;       // fundamental frequency
188         c = 1; m = 1;   // All harmonics
189         //c = 1; m = 2; // Missing every 2nd harmonic
```

```
190         //c = 1; m = 3; // Missing every 3rd harmonic
191         //c = 5; m = 1; // All harmonics and starts on 5th
                harmonic
192         //c = 1; m = 1.414;  //  Inharmonic
193         peakFMI = 10;
194         carrierFreq = c * fundFreq;              // C:M ratio
                determines the carrier and
195         modulatorFreq = m * fundFreq;    //  the modulator
                frequencies
196
197         env = Env.new
                ([0,5.0,3.75,3.0,0]/5,[0.15,0.15,0.15,0.15],'linear')
                ;
198         amp = EnvGen.kr(env);
199
200         instFreqDv = peakFMI * amp * modulatorFreq;
201         instFreq = SinOsc.ar(modulatorFreq,0,instFreqDv,
                carrierFreq);
202         SinOsc.ar(instFreq,0,amp);
203 }.play;)
204
205 Here is an example with separate envelopes for the amplitude and
        for the frequency modulation index.
206
207 ({
208         var fundFreq, peakFMI, carrierFreq, modulatorFreq, env1,
                env2, amp, instFreqDv, instFreq;
209         fundFreq = 440.0;        // fundamental frequency
210         c = 3; m = 1;    // All harmonics
211         peakFMI = 2;
212         carrierFreq = c * fundFreq;              // C:M ratio
                determines the carrier and
213         modulatorFreq = m * fundFreq;    //  the modulator
                frequencies
214
215         env1 = Env.new
                ([0,5.0,3.75,3.0,0]/5,[0.15,0.15,0.15,0.15],'linear')
                ;
216         amp = EnvGen.kr(env1);
217         //env2 = Env.new([3*peakFMI, peakFMI, 0.7 * peakFMI
                ],[0.3,0.45],'linear');
218         env2 = Env.new([0.3*peakFMI, peakFMI, 2.0 * peakFMI
                ],[0.3,0.45],'linear');
219         instFreqDv = modulatorFreq * EnvGen.kr(env2);
220
221         instFreq = SinOsc.ar(modulatorFreq,0,instFreqDv,
                carrierFreq);
222         SinOsc.ar(instFreq,0,amp);
223 }.play;)
```

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 8
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  Mas FM
6
7  Detuning by a small factor.
8
9  ({
10         var fundFreq, detune, peakFMI, carrierFreq,
               modulatorFreq, env1, env2, amp, instFreqDv, instFreq;
11         fundFreq = 440.0;        // fundamental frequency
12         detune = 0.5;
13         c = 1;
14         m = 1;
15         peakFMI = 3;
16         carrierFreq = c * fundFreq;
               // C:M ratio determines the carrier and
17         modulatorFreq = m * fundFreq + detune;      //  the
               modulator frequency + detuning
18
19         env1 = Env.new([0,5.0,3.75,3.0,0]/5,[0.15,0.15,2,0.15],'
               linear'); // amp Env
20         amp = EnvGen.kr(env1);
21         env2 = Env.new([2 * peakFMI, peakFMI, peakFMI
               ],[0.3,0.45],'linear');   // FMI Env
22         instFreqDv = modulatorFreq * EnvGen.kr(env2);
23
24         instFreq = SinOsc.ar(modulatorFreq,0,instFreqDv,
               carrierFreq);
25         SinOsc.ar(instFreq,0,amp);
26 }.play)
27
28
29 Multiple-Carrier FM can often produce tones with a vowel-like
       quality.
30
31 ({
32         var fundFreq, c1, c2, peakFMI, carrierFreq1,
               carrierFreq2, modulatorFreq, env1, env2;
33         var amp, freqDv, instFreqDv;
34         fundFreq = 440.0;
35         c1 = 1;
36         c2 = 6;
37         m = 1;
38         peakFMI = 0.5;
39         carrierFreq1 = c1 * fundFreq;  // C1 : C2 : M ratio
```

```
40          carrierFreq2 = c2 * fundFreq;
41          modulatorFreq = m * fundFreq;
42
43          env1 = Env.new
                ([0,5.0,3.75,3.0,0]/5,[0.15,0.15,2.0,0.15],'linear');
44          amp = 0.3 * EnvGen.kr(env1);
45          env2 = Env.new([0.5 * peakFMI, peakFMI, 0.7 * peakFMI
                ],[0.3,0.45],'linear');
46          freqDv = modulatorFreq * EnvGen.kr(env2);
47
48          instFreqDv = SinOsc.ar(modulatorFreq,0,freqDv);
49          SinOsc.ar(instFreqDv+carrierFreq1,0,amp) +
                // Two carriers are being modulated
50          SinOsc.ar(instFreqDv+carrierFreq2,0,amp);
51  }.play)
52
53
54  Multiple-Modulator FM can often produce tones with a smoother
        quality than raw FM.
55
56  ({
57          var fundFreq, m1, m2, peakFMI1, peakFMI2, carrierFreq,
                modulatorFreq1, modulatorFreq2;
58          var env1, env2, amp, instFMI, instFreqDv1, instFreqDv2,
                instFreq;
59          fundFreq = 440.0;
60          c = 1;
61          m1 = 1;
62          m2 = 4;
63          peakFMI1 = 1;
64          peakFMI2 = 0.4;
65          carrierFreq = c * fundFreq + 0.5;  // C : M1 : M2 ratio
66          modulatorFreq1 = m1 * fundFreq;
67          modulatorFreq2 = m2 * fundFreq;
68
69          env1 = Env.new
                ([0,5.0,3.75,3.0,0]/5,[0.15,0.15,2.0,0.15],'linear');
70          amp = 0.5 * EnvGen.kr(env1);
71          env2 = Env.new([2.0, 1.0, 0.6],[0.3,0.45],'linear');
72          instFMI = EnvGen.kr(env2);
                                        //  One FMI envelope
73
74          instFreqDv1 = peakFMI1 * modulatorFreq1 * instFMI;  //
                    with individual
75          instFreqDv2 = peakFMI2 * modulatorFreq2 * instFMI; //
                instantaneous frequency deviations
76
77          instFreq = SinOsc.ar(modulatorFreq1,0,instFreqDv1) +  //
                 Two modulating oscillators
```

```
78                                      SinOsc.ar(modulatorFreq2,0,
                                            instFreqDv2) +
79                                      carrierFreq;
80
81          SinOsc.ar(instFreq,0,amp);    // One carrier
82  }.play)
83
84  ({
85          // Try different values for fundFreq
86          var fundFreq = 1760;
87          var duration, detune, peakFMI, vibRate, vibDepth;
88          var carrierFreq, modulatorFreq, octave,attackTm,
                sustainTm, releaseTm;
89          var env1, env2, env3, amp, instFreqDv, instVib, instFreq
                ;
90
91          // parameters
92          duration = 2.0;
93          detune = 0.5;
94          c = 2; m = 1;
95          peakFMI = 3.0;
96          vibRate = 4.0;
97          vibDepth = 0.006;
98
99          //  calculate actual carrier and modulator frequencies
100         carrierFreq = c * fundFreq;
101         modulatorFreq = m * fundFreq + detune.rand;  // random
                variations
102
103         // modifications based on fundamental frequency
104         octave = fundFreq.cpsoct;       // determine the octave
105         post('octave = ');octave.postln;
106         peakFMI = peakFMI * 20.0 / (octave*octave);  // peakFM
                decreases in higher octaves
107         vibRate = vibRate * octave / 5.0;  // vibRate increases
                in higher octaves
108         vibDepth = vibDepth * 5.0 / octave;
109         postln('peakFMI/vibRate/vibDepth = '); peakFMI.postln;
                vibRate.postln;vibDepth.postln;
110         // attack and release times based on fundamental
                frequency
111         attackTm = 0.4 * 4.0 / fundFreq.cpsoct;    // attackTM
                decreases in higher octaves
112         releaseTm = 1.5* attackTm;
113         sustainTm = duration - attackTm - releaseTm;
114         if (releaseTm > sustainTm, {releaseTm = sustainTm = (
                duration - attackTm)/2});
115
116         //  amplitude envelope
```

```
117         env1 = Env.new([0,1.0,0.75,0],[attackTm,sustainTm,
                releaseTm],'linear');
118         amp = EnvGen.kr(env1);
119         // FMI envelope
120         env2 = Env.new([2*peakFMI, peakFMI, 0.8*peakFMI,
121                                 0.5*peakFMI],[attackTm,sustainTm
                                    ,releaseTm],'linear');
122         instFreqDv = modulatorFreq * EnvGen.kr(env2);
123         // vibrato envelope
124         env3 = Env.new([0,vibDepth,0.8*vibDepth,0],[attackTm,
                sustainTm,releaseTm],'linear');
125         instVib = 1 + SinOsc.kr(vibRate + 3.0.rand,0,EnvGen.kr(
                env3));
126
127         instFreq = SinOsc.ar(instVib*modulatorFreq,0,instFreqDv,
                instVib*carrierFreq);
128         SinOsc.ar(instFreq,0,amp);
129 }.play;)
130 More complete instrument design Version #1:
131
132 Vibrato added with its own envelope.
133
134 ({
135                 var fundFreq, duration, detune, peakFMI, vibRate
                        , vibDepth;
136                 var carrierFreq, modulatorFreq, attackTm,
                        sustainTm, releaseTm;
137                 var env1, env2, env3, amp, instFreqDv, instVib,
                        instFreq;
138
139                 // parameters
140                 fundFreq = 880.0;
141                 duration = 2.0;
142                 detune = 0.5;
143                 c = 2; m = 1;
144                 peakFMI = 3;
145                 vibRate = 5.0;  // vibrator rate
146                 vibDepth = 0.008;  // vibrato depth as a
                        fraction of fundament frequency
147
148                 //  calculate actual carrier and modulator
                        frequencies
149                 carrierFreq = c * fundFreq;
150                 modulatorFreq = m * fundFreq + detune.rand;  //
                        random detuning
151                 post('carrier frequency = '); carrierFreq.postln
                        ;
152                 post('modulator frequency = '); modulatorFreq.
                        postln;
```

```
153
154                 // attack and release times
155                 attackTm = 0.3;
156                 releaseTm = 2.0* attackTm;
157                 sustainTm = duration - attackTm - releaseTm;
158                 if (releaseTm > sustainTm, {releaseTm =
                        sustainTm = (duration - attackTm)/2});
159                 post('attack time = '); attackTm.postln;
160                 post('sustain time = '); sustainTm.postln;
161                 post('release time = '); releaseTm.postln;
162
163                 //  amplitude envelope
164                 env1 = Env.new([0,1.0,0.75,0],[attackTm,
                        sustainTm,releaseTm],'linear');
165                 amp = EnvGen.kr(env1);
166                 //  FMI envelope
167                 env2 = Env.new([2*peakFMI, peakFMI, 0.8*peakFMI,
168                                 0.5*peakFMI],[attackTm,
                                    sustainTm,releaseTm],
                                    'linear');
169                 instFreqDv = modulatorFreq * EnvGen.kr(env2);
170                 //  vibrato envelope
171                 env3 = Env.new([0,vibDepth,0.8*vibDepth,0],[
                        attackTm,sustainTm,releaseTm],'linear');
172                 instVib = 1 + SinOsc.kr(vibRate + 3.0.rand,0,
                        EnvGen.kr(env3)); // random variations too
173
174                 instFreq = SinOsc.ar(instVib*modulatorFreq, 0,
                        instFreqDv, instVib*carrierFreq);
175                                 // vibrator affects both
                                        carrier and
                                        modulators
176                 SinOsc.ar(instFreq,0,amp);
177 }.play;)
178
179
180 More complete instrument design Version #2:
181
182 Parameters are made sensitive to the octave of the fundamental.
183
184 ({
185         // Try different values for fundFreq
186         var fundFreq = 1760;
187         var duration, detune, peakFMI, vibRate, vibDepth;
188         var carrierFreq, modulatorFreq, octave,attackTm,
                sustainTm, releaseTm;
189         var env1, env2, env3, amp, instFreqDv, instVib, instFreq
                ;
190
```

```
191         // parameters
192         duration = 2.0;
193         detune = 0.5;
194         c = 2; m = 1;
195         peakFMI = 3.0;
196         vibRate = 4.0;
197         vibDepth = 0.006;
198
199         //  calculate actual carrier and modulator frequencies
200         carrierFreq = c * fundFreq;
201         modulatorFreq = m * fundFreq + detune.rand;  // random
                variations
202
203         // modifications based on fundamental frequency
204         octave = fundFreq.cpsoct;       // determine the octave
205         post('octave = ');octave.postln;
206         peakFMI = peakFMI * 20.0 / (octave*octave);  // peakFM
                decreases in higher octaves
207         vibRate = vibRate * octave / 5.0;  // vibRate increases
                in higher octaves
208         vibDepth = vibDepth * 5.0 / octave;
209         postln('peakFMI/vibRate/vibDepth = '); peakFMI.postln;
                vibRate.postln;vibDepth.postln;
210         // attack and release times based on fundamental
                frequency
211         attackTm = 0.4 * 4.0 / fundFreq.cpsoct;   // attackTM
                decreases in higher octaves
212         releaseTm = 1.5* attackTm;
213         sustainTm = duration - attackTm - releaseTm;
214         if (releaseTm > sustainTm, {releaseTm = sustainTm = (
                duration - attackTm)/2});
215
216         //  amplitud({
217         // Try different values for fundFreq
218         var fundFreq = 1760;
219         var duration, detune, peakFMI, vibRate, vibDepth;
220         var carrierFreq, modulatorFreq, octave,attackTm,
                sustainTm, releaseTm;
221         var env1, env2, env3, amp, instFreqDv, instVib, instFreq
                ;
222
223         // parameters
224         duration = 2.0;
225         detune = 0.5;
226         c = 2; m = 1;
227         peakFMI = 3.0;
228         vibRate = 4.0;
229         vibDepth = 0.006;
230
```

```
231         //  calculate actual carrier and modulator frequencies
232         carrierFreq = c * fundFreq;
233         modulatorFreq = m * fundFreq + detune.rand;  // random
                variations
234
235         // modifications based on fundamental frequency
236         octave = fundFreq.cpsoct;       // determine the octave
237         post('octave = ');octave.postln;
238         peakFMI = pe({
239         // Try different values for fundFreq
240         var fundFreq = 1760;
241         var duration, detune, peakFMI, vibRate, vibDepth;
242         var carrierFreq, modulatorFreq, octave,attackTm,
                sustainTm, releaseTm;
243         var env1, env2, env3, amp, instFreqDv, instVib, instFreq
                ;
244
245         // parameters
246         duration = 2.0;
247         detune = 0.5;
248         c = 2; m = 1;
249         peakFMI = 3.0;
250         vibRate = 4.0;
251         vibDepth = 0.006;
252
253         //  calculate actual carrier and modulator frequencies
254         carrierFreq = c * fundFreq;
255         modulatorFreq = m * fundFreq + detune.rand;  // random
                variations
256
257         // modifications based on fundamental frequency
258         octave = fundFreq.cpsoct;       // determine the octave
259         post('octave = ');octave.postln;
260         peakFMI = peakFMI * 20.0 / (octave*octave);  // peakFM
                decreases in higher octaves
261         vibRate = vibRate * octave / 5.0;  // vibRate increases
                in higher octaves
262         vibDepth = vibDepth * 5.0 / octave;
263         postln('peakFMI/vibRate/vibDepth = '); peakFMI.postln;
                vibRate.postln;vibDepth.postln;
264         // attack and release times based on fundamental
                frequency
265         attackTm = 0.4 * 4.0 / fundFreq.cpsoct;   // attackTM
                decreases in higher octaves
266         releaseTm = 1.5* attackTm;
267         sustainTm = duration - attackTm - releaseTm;
268         if (releaseTm > sustainTm, {releaseTm = sustainTm = (
                duration - attackTm)/2});
269
```

```
270         // amplitude envelope
271         env1 = Env.new([0,1.0,0.75,0],[attackTm,sustainTm,
                releaseTm],'linear');
272         amp = EnvGen.kr(env1);
273         // FMI envelope
274         env2 = Env.new([2*peakFMI, peakFMI, 0.8*peakFMI,
275                                 0.5*peakFMI],[attackTm,sustainTm
                                    ,releaseTm],'linear');
276         instFreqDv = modulatorFreq * EnvGen.kr(env2);
277         // vibrato envelope
278         env3 = Env.new([0,vibDepth,0.8*vibDepth,0],[attackTm,
                sustainTm,releaseTm],'linear');
279         instVib = 1 + SinOsc.kr(vibRate + 3.0.rand,0,EnvGen.kr(
                env3));
280
281         instFreq = SinOsc.ar(instVib*modulatorFreq,0,instFreqDv,
                instVib*carrierFreq);
282         SinOsc.ar(instFreq,0,amp);
283 }.play;)akFMI * 20.0 / (octave*octave);  // peakFM decreases in
    higher octaves
284         vibRate = vibRate * octave / 5.0;  // vibRate increases
                in higher octaves
285         vibDepth = vibDepth * 5.0 / octave;
286         postln('peakFMI/vibRate/vibDepth = '); peakFMI.postln;
                vibRate.postln;vibDepth.postln;
287         // attack and release times based on fundamental
                frequency
288         attackTm = 0.4 * 4.0 / fundFreq.cpsoct;    // attackTM
                decreases in higher octaves
289         releaseTm = 1.5* attackTm;
290         sustainTm = duration - attackTm - releaseTm;
291         if (releaseTm > sustainTm, {releaseTm = sustainTm = (
                duration - attackTm)/2});
292
293         // amplitude envelope
294         env1 = Env.new([0,1.0,0.75,0],[attackTm,sustainTm,
                releaseTm],'linear');
295         amp = EnvGen.kr(env1);
296         // FMI envelope
297         env2 = Env.new([2*peakFMI, peakFMI, 0.8*peakFMI,
298                                 0.5*peakFMI],[attackTm,sustainTm
                                    ,releaseTm],'linear');
299         instFreqDv = modulatorFreq * EnvGen.kr(env2);
300         // vibrato envelope
301         env3 = Env.new([0,vibDepth,0.8*vibDepth,0],[attackTm,
                sustainTm,releaseTm],'linear');
302         instVib = 1 + SinOsc.kr(vibRate + 3.0.rand,0,EnvGen.kr(
                env3));
303
```

```
304         instFreq = SinOsc.ar(instVib*modulatorFreq,0,instFreqDv,
                instVib*carrierFreq);
305         SinOsc.ar(instFreq,0,amp);
306 }.play;)e envelope
307         env1 = Env.new([0,1.0,0.75,0],[attackTm,sustainTm,
                releaseTm],'linear');
308         amp = EnvGen.kr(env1);
309         //  FMI envelope
310         env2 = Env.new([2*peakFMI, peakFMI, 0.8*peakFMI,
311                                 0.5*peakFMI],[attackTm,sustainTm
                                    ,releaseTm],'linear');
312         instFreqDv = modulatorFreq * EnvGen.kr(env2);
313         //  vibrato envelope
314         env3 = Env.new([0,vibDepth,0.8*vibDepth,0],[attackTm,
                sustainTm,releaseTm],'linear');
315         instVib = 1 + SinOsc.kr(vibRate + 3.0.rand,0,EnvGen.kr(
                env3));
316
317         instFreq = SinOsc.ar(instVib*modulatorFreq,0,instFreqDv,
                instVib*carrierFreq);
318         SinOsc.ar(instFreq,0,amp);
319 }.play;)
```

A.7. SuperCollider: Síntesis granular

```
1  TUTORIAL DE SUPERCOLLIDER 3 - PARTE 9
2  Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3  Northwestern University
4
5  (
6          Server.default = Server.internal;
7          s = Server.default;
8          s.boot;
9  )
10
11 Sintesis granular
12
13 Gendy1  (tambien Gendy2 y Gendy3)
14 - Implementacion del generador dinamico estocastico concebido
       por Xenakis
15
16
17 Metodos de clase
18 *ar(ampdist=1, durdist=1, adparam=1.0, ddparam=1.0, minfreq=20,
       maxfreq=1000, ampscale= 0.5, durscale=0.5, initCPs=12, knum
       =12, mul=1.0, add=0.0)
```

```
19
20 Check the help page for an explanation of the parameters!
21
22 //defaults
23
24 {Pan2.ar(RLPF.ar(Gendy1.ar(2,3,minfreq:20, maxfreq:MouseX.kr
       (100,1000), durscale:0.0, initCPs:40), 500,0.3, 0.2), 0.0)}.
       play
25
26 ({
27 var mx, my;
28 mx= MouseX.kr(220,440);
29 my= MouseY.kr(0.0,1.0);
30 Pan2.ar(Gendy1.ar(2,3,1,1,minfreq:mx, maxfreq:8*mx, ampscale:my,
        durscale:my, initCPs:7, mul:0.3), 0.0)
31 }.play)
32
33
34 Sintesis granular ala Gabor
35
36 // Granular Synthesis
37 (SynthDef(\grainMaker, {arg dur = 1, amp = -3, freq = 440;
38         var env, halfdur, car;
39         halfdur = dur * 0.5;
40         amp = amp.dbamp;
41         env = EnvGen.ar(
42                 Env([0, amp, 0], [halfdur, halfdur], \lin),
43                 doneAction: 2);
44         car = SinOsc.ar(freq, 0, 1);
45         OffsetOut.ar(0, car * env);
46 }).load(s);)
47
48 (z = Routine({
49         var freq, amp, time=0;
50         freq = Env([1000, 500], [15], \exp);
51         amp = Env([-12, -3, -12], [7.5, 7.5], \lin);
52         1500.do{
53                 s.sendBundle(0.1, [\s_new, \grainMaker, -1, 0,
                       1, \dur, 0.009, \amp, amp[time], \freq, freq[
                       time]]);
54                 0.01.wait;
55                 time = time + 0.01;
56         }
57 }).play)
```

A.8. SuperCollider: Patrones

```
TUTORIAL DE SUPERCOLLIDER 3 - PARTE 10
Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
Northwestern University

(
    Server.internal.boot;
    Server.default = Server.internal;
    s = Server.default;
)

Patterns


Making Music with Patterns

Here is an example that uses a Pattern to create two instances of
the random melody stream.

(
        SynthDef( "SPE2", { arg i_out=0, i_dur=1, freq;
                var out;
                out = RLPF.ar(
                        LFSaw.ar( freq ),
                        LFNoise1.kr(1, 36, 110).midicps,
                        0.1
                ) * EnvGen.kr( Env.perc, levelScale: 0.3,
                                                        timeScale:
                                                        i_dur,
                                                        doneAction: 2 );
                4.do({ out = AllpassN.ar(out, 0.05, [0.05.rand,
                    0.05.rand], 4) });
                Out.ar( i_out, out );
        }).send(s);
)
(
// streams as a sequence of pitches
        var pattern, streams, dur, durDiff;
        dur = 1/7;
        durDiff = 3;
        pattern = Prout.new({
                loop({
                        if (0.5.coin, {
```

```
40                                      #[ 24,31,36,43,48,55 ].do({ arg
                                            fifth; fifth.yield });
41                              });
42                              rrand(2,5).do({
43                                      // varying arpeggio
44                                      60.yield;
45                                      #[63,65].choose.yield;
46                                      67.yield;
47                                      #[70,72,74].choose.yield;
48                              });
49                              // random high melody
50                              rrand(3,9).do({  #[74,75,77,79,81].
                                    choose.yield });
51                      });
52              });
53              streams = [  // Stream.at(0)  combines Prout's output
                    with Pfunc transposing notes randomly
54                      (pattern - Pfunc.new({ #[12, 7, 7, 0].choose }))
                            .midicps.asStream, // 0 in array
55                      pattern.midicps.asStream // 1 in array
56              ];
57              Routine({
58                      loop({
59                              // Do this
60                              Synth( "SPE2", [ \freq, streams.at(0).
                                    next, \i_dur, dur * durDiff ] );
61                              durDiff.do({  // then do this durDiff
                                    times
62                                      Synth( "SPE2", [ \freq, streams.
                                            at(1).next, \i_dur, dur ] );
63                                      dur.wait;
64                              });
65                      })
66              }).play
67      )
```

```
1       TUTORIAL DE SUPERCOLLIDER 3 - PARTE 11
2       Rodrigo F. Cadiz, basado en apuntes de Gary S. Kendall
3       Northwestern University
4
5       ListPatterns
6
7       ListPatterns are Patterns that iterate over arrays of objects in
            some fashion.
8       All ListPatterns have in common the instance variables list and
            repeats.
9       The list variable is some Array to be iterated over. The repeats
            variable is
```

```
10 some measure of the number of times to do something, whose
       meaning
11 varies from subclass to subclass. The default value for repeats
       is 1.
12
13
14
15 Making Music with ListPatterns
16
17 Here is an example we have heard before rewritten to use
       ListPatterns.
18 It uses nested patterns and results in much more concise code.
19
20 (
21 SynthDef( "Allpass6", { arg freq, dur= 1.0;  // Frequency and
       duration parameters
22         var out, env;
23         out = RLPF.ar(
24                 LFSaw.ar( freq, mul: EnvGen.kr( Env.perc,
                       levelScale: 0.15, timeScale: dur, doneAction:
                        2 ) ),
25                 LFNoise1.kr(1, 36, 110).midicps,
26                 0.1
27         );
28         6.do({ out = AllpassN.ar(out, 0.05, [0.05.rand, 0.05.
               rand], 4) });
29         Out.ar( 0, out  );
30 }).send(s)
31 )
32
33 (   // This is a good more complex model
34         var freqStream;
35
36         freqStream = Pseq( // list, repeats, offset
37                 [Prand([ nil, Pseq(#[24, 31, 36, 43, 48, 55])]),
38                                                 // no repeat,
                                                       nil causes
                                                       pattern to
                                                       end
39                         Pseq([ 60, Prand(#[63, 65]), 67, Prand
                               (#[70, 72, 74]) ], rrand(2, 5)),
40                         Prand(#[74, 75, 77, 79, 81], { rrand(3,
                               9) })
41                 ],
42                 inf).asStream.midicps;
43
44         Task({
45                 loop({
```

```
46                          Synth( "Allpass6", [\freq, freqStream.
                                next]);
47                          0.13.wait;
48                  });
49          }).play;
50  )
51
52  Here is an example that uses a Pattern to create a rhythmic solo
        . The values in the pattern
53  specify the amplitudes of impulses fed to the Decay2 generator.
54
55  (
56  SynthDef( "Mridangam", { arg t_amp; // Amplitude is argument (
        dur isn't meaningful)
57          var out;
58
59          out = Resonz.ar(
60                  WhiteNoise.ar(70) * Decay2.kr( t_amp, 0.002, 0.1
                        ),
61                  60.midicps, 0.02,4
62          ).distort * 0.4;
63          Out.ar( 0, out );
64          DetectSilence.ar( out, doneAction: 2 );
65  }).send(s);
66
67  SynthDef( "Drone", {
68          var out;
69          out = LPF.ar(
70                  Saw.ar([60, 60.04].midicps) + Saw.ar([67,
                        67.04].midicps),
71                  108.midicps,
72                  0.007
73          );
74          Out.ar( 0, out );
75  }).send(s);
76  )
77
78  (// percussion solo in 10/8
79  var stream, pat, amp;
80
81  pat = Pseq([
82          Pseq(#[0.0], 10),
83
84          // intro
85          Pseq(#[0.9, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0], 2),
86          Pseq(#[0.9, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.2, 0.0,
                0.0], 2),
```

```
87          Pseq(#[0.9, 0.0, 0.0, 0.2, 0.0, 0.2, 0.0, 0.2, 0.0,
                0.0], 2),
88          Pseq(#[0.9, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.2, 0.0,
                0.2], 2),
89
90          // solo
91          Prand([
92                  Pseq(#[0.9, 0.0, 0.0, 0.7, 0.0, 0.2, 0.0, 0.7,
                        0.0, 0.0]),
93                  Pseq(#[0.9, 0.2, 0.0, 0.7, 0.0, 0.2, 0.0, 0.7,
                        0.0, 0.0]),
94                  Pseq(#[0.9, 0.0, 0.0, 0.7, 0.0, 0.2, 0.0, 0.7,
                        0.0, 0.2]),
95                  Pseq(#[0.9, 0.0, 0.0, 0.7, 0.2, 0.2, 0.0, 0.7,
                        0.0, 0.0]),
96                  Pseq(#[0.9, 0.0, 0.0, 0.7, 0.0, 0.2, 0.2, 0.7,
                        0.2, 0.0]),
97                  Pseq(#[0.9, 0.2, 0.2, 0.7, 0.2, 0.2, 0.2, 0.7,
                        0.2, 0.2]),
98                  Pseq(#[0.9, 0.2, 0.2, 0.7, 0.2, 0.2, 0.2, 0.7,
                        0.0, 0.0]),
99                  Pseq(#[0.9, 0.0, 0.0, 0.7, 0.2, 0.2, 0.2, 0.7,
                        0.0, 0.0]),
100                 Pseq(#[0.9, 0.0, 0.4, 0.0, 0.4, 0.0, 0.4, 0.0,
                        0.4, 0.0]),
101                 Pseq(#[0.9, 0.0, 0.0, 0.4, 0.0, 0.0, 0.4, 0.2,
                        0.4, 0.2]),
102                 Pseq(#[0.9, 0.0, 0.2, 0.7, 0.0, 0.2, 0.0, 0.7,
                        0.0, 0.0]),
103                 Pseq(#[0.9, 0.0, 0.0, 0.7, 0.0, 0.0, 0.0, 0.7,
                        0.0, 0.0]),
104                 Pseq(#[0.9, 0.7, 0.7, 0.0, 0.0, 0.2, 0.2, 0.2,
                        0.0, 0.0]),
105                 Pseq(#[0.9, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                        0.0, 0.0])
106         ], 30),
107         // conclusion
108         Pseq(#[2.0, 0.0, 0.2, 0.5, 0.0, 0.2, 0.9,
109                 1.5, 0.0, 0.2, 0.5, 0.0, 0.2, 0.9,
110                 1.5, 0.0, 0.2, 0.5, 0.0, 0.2], 3),
111         Pseq(#[5], 1),
112         Pseq(#[0.0], inf)
113 ]);
114 stream = pat.asStream;
115
116 Task({
117         Synth("Drone");
118         loop({
119                 if( ( amp = stream.next ) > 0,
```

```
120                         { Synth("Mridangam", [ \t_amp, amp ]) }
121                 );
122                 (1/8).wait;
123         })
124 }).play
125
126 )
```

A.9. SuperCollider: Filtros bi-cuadráticos

```
1  (
2          // Filtros biquad
3
4          var w;
5          var caption1,caption2,caption3,captionfbSlider,
               captionddSlider;
6          var lpfBox,hpfBox,bpfBox,notchBox,peakingEQBox,
               lowShelfBox,highShelfBox;
7          var in1Box,in2Box,in3Box,captionSource;
8          var captiongainSlider,gainSlider,gainNum;
9          var captionfilterSlider,filterSlider,filterNum;
10         var captionshelfSlider,shelfSlider,shelfNum;
11         var captionQSlider,qSlider,qNum;
12         var captionOmegaSlider,omegaSlider,omegaNum;
13
14         var filename, sound, signal;
15         var fore_color;
16
17         var xoff1,xoff2;
18         var ftoW;
19
20         // convert dB to amplitude
21         var dBtoA;
22         dBtoA = { arg decibels;
23                 r = 0.5011872336;
24                 r*10**(decibels/20.0);};
25
26
27         ftoW = { arg hertz;
28                 2pi*(hertz/Synth.sampleRate);};
29
30
31         // GUI Window
32         w = GUIWindow.new("First order biquad filters >>>
               Rodrigo F. Cadiz",
33 Rect.newBy(500, 70, 650, 400));
```

```
34          w.backColor = Color.new(99,130,160);
35
36
37          // Offsets
38          xoff1 = 10;
39          xoff2 = 350;
40
41          // Colors
42          fore_color = Color.new(255,255,0);
43
44          // Various labels
45          caption1 = StringView.new( w, Rect.newBy(xoff1, 10, 228,
                20), "Advanced
46  Audio Signal Processing");
47          caption1.labelColor = fore_color;
48          caption2 = StringView.new( w, Rect.newBy(xoff1, 30, 328,
                20),
49  "Assignment 2: Second order biquad filters");
50          caption2.labelColor = fore_color;
51          caption3 = StringView.new( w, Rect.newBy(xoff1, 50, 128,
                20), "Rodrigo
52  F. Cadiz");
53          caption3.labelColor = fore_color;
54
55          // Input Gain
56          captiongainSlider = StringView.new( w, Rect.newBy(xoff2,
                10, 235, 20),
57  "Input gain [dB] ");
58      gainSlider = SliderView.new( w, Rect.newBy( xoff2, 30, 230,
            20 ),
59  "Mix", 0, -30, 6, 1, 'linear');
60      gainNum = NumericalView.new( w, Rect.newBy( xoff2+235, 30,
            50, 20 ),
61  "NumericalView", 0, -30, 6, 1, 'linear');
62      gainNum.backColor = fore_color;
63      gainSlider.action = { gainNum.value = gainSlider.value};
64
65
66      // Filter Gain
67          captionfilterSlider = StringView.new( w, Rect.newBy(
                xoff2, 60, 235, 20),
68  "Filter gain [dB] ");
69      filterSlider = SliderView.new( w, Rect.newBy( xoff2, 80,
            230, 20 ),
70  "Mix", 0, -0.02, 0.02, 0.0001, 'linear');
71      filterNum = NumericalView.new( w, Rect.newBy( xoff2+235,
            80, 50, 20
72  ), "NumericalView", 0, -0.02, 0.02, 0.0001, 'linear');
73      filterNum.backColor = fore_color;
```

```
74      filterSlider.action = { filterNum.value = filterSlider.
            value};
75
76
77      // Shelf slope
78          captionshelfSlider = StringView.new( w, Rect.newBy(xoff2
                , 110, 235, 20),
79  "Shelf slope ");
80      shelfSlider = SliderView.new( w, Rect.newBy( xoff2, 130,
            230, 20 ),
81  "Mix", 1, 0.001 , 2 , 0.001, 'exponential');
82      shelfNum = NumericalView.new( w, Rect.newBy( xoff2+235,
            130, 50, 20
83  ), "NumericalView", 1, 0.001, 2, 0.001, 'exponential');
84      shelfNum.backColor = fore_color;
85      shelfSlider.action = { shelfNum.value = shelfSlider.value};
86
87          // Omega
88          captionOmegaSlider = StringView.new( w, Rect.newBy(xoff2
                , 160, 235, 20),
89  "Cutoff frequency [Hz] ");
90      omegaSlider = SliderView.new( w, Rect.newBy( xoff2, 180,
            230, 20 ),
91  "Mix", 400, 100 , 15000 , 10, 'exponential');
92      omegaNum = NumericalView.new( w, Rect.newBy( xoff2+235,
            180, 50, 20
93  ), "NumericalView", 400, 100, 15000, 10, 'exponential');
94      omegaNum.backColor = fore_color;
95      omegaSlider.action = { omegaNum.value = omegaSlider.value};
96
97          // Q
98          captionQSlider = StringView.new( w, Rect.newBy(xoff2,
                210, 235, 20), "Q ");
99      qSlider = SliderView.new( w, Rect.newBy( xoff2, 230, 230,
            20 ),
100 "Mix", 0.5, 0.01 , 10.0 , 0.01, 'exponential');
101     qNum = NumericalView.new( w, Rect.newBy( xoff2+235, 230,
            50, 20 ),
102 "NumericalView", 0.5, 0.01 ,10.0, 0.01, 'exponential');
103     qNum.backColor = fore_color;
104     qSlider.action = { qNum.value = qSlider.value};
105
106         lpfBox = CheckBoxView.new( w, Rect.newBy( xoff1, 90,
                230, 20 ), "Low
107 pass ", 1, 0, 1, 0, 'linear');
108         hpfBox = CheckBoxView.new( w, Rect.newBy( xoff1, 110,
                230, 20 ), "High
109 pass ", 0, 0, 1, 0, 'linear');
```

```
110         bpfBox = CheckBoxView.new( w, Rect.newBy( xoff1, 130,
                230, 20 ), "Band
111 pass ", 0, 0, 1, 0, 'linear');
112         notchBox = CheckBoxView.new( w, Rect.newBy( xoff1, 150,
                230, 20 ),
113 "Notch  ", 0, 0, 1, 0, 'linear');
114         peakingEQBox = CheckBoxView.new( w, Rect.newBy( xoff1,
                170, 230, 20 ),
115 "PeakingEQ ", 0, 0, 1, 0, 'linear');
116         lowShelfBox = CheckBoxView.new( w, Rect.newBy( xoff1,
                190, 230, 20 ),
117 "Low Shelf ", 0, 0, 1, 0, 'linear');
118         highShelfBox = CheckBoxView.new( w, Rect.newBy( xoff1,
                210, 230, 20 ),
119 "High Shelf ", 0, 0, 1, 0, 'linear');
120
121         lpfBox.action = {
122              lpfBox.value = 1.0;
123                 hpfBox.value = 0.0;
124                 bpfBox.value = 0.0;
125                 notchBox.value = 0.0;
126                 peakingEQBox.value = 0.0;
127                 lowShelfBox.value = 0.0;
128                 highShelfBox.value = 0.0;};
129
130         hpfBox.action = {
131              lpfBox.value = 0.0;
132                 hpfBox.value = 1.0;
133                 bpfBox.value = 0.0;
134                 notchBox.value = 0.0;
135                 peakingEQBox.value = 0.0;
136                 lowShelfBox.value = 0.0;
137                 highShelfBox.value = 0.0;};
138
139         bpfBox.action = {
140              lpfBox.value = 0.0;
141                 hpfBox.value = 0.0;
142                 bpfBox.value = 1.0;
143                 notchBox.value = 0.0;
144                 peakingEQBox.value = 0.0;
145                 lowShelfBox.value = 0.0;
146                 highShelfBox.value = 0.0;};
147
148         notchBox.action = {
149              lpfBox.value = 0.0;
150                 hpfBox.value = 0.0;
151                 bpfBox.value = 0.0;
152                 notchBox.value = 1.0;
153                 peakingEQBox.value = 0.0;
```

```
154                 lowShelfBox.value = 0.0;
155                 highShelfBox.value = 0.0;};
156
157         peakingEQBox.action = {
158             lpfBox.value = 0.0;
159                 hpfBox.value = 0.0;
160                 bpfBox.value = 0.0;
161                 notchBox.value = 0.0;
162                 peakingEQBox.value = 1.0;
163                 lowShelfBox.value = 0.0;
164                 highShelfBox.value = 0.0;};
165
166         lowShelfBox.action = {
167             lpfBox.value = 0.0;
168                 hpfBox.value = 0.0;
169                 bpfBox.value = 0.0;
170                 notchBox.value = 0.0;
171                 peakingEQBox.value = 0.0;
172                 lowShelfBox.value = 1.0;
173                 highShelfBox.value = 0.0;};
174
175         highShelfBox.action = {
176             lpfBox.value = 0.0;
177                 hpfBox.value = 0.0;
178                 bpfBox.value = 0.0;
179                 notchBox.value = 0.0;
180                 peakingEQBox.value = 0.0;
181                 lowShelfBox.value = 0.0;
182                 highShelfBox.value = 1.0;};
183
184       // Select source
185        captionSource = StringView.new( w, Rect.newBy(xoff2, 250,
             235, 20),
186 "Select source");
187        captionSource.labelColor = fore_color;
188          in1Box = CheckBoxView.new( w, Rect.newBy( xoff2, 270,
               230, 20 ), "Sound
189 file ", 1, 0, 1, 0, 'linear');
190          in2Box = CheckBoxView.new( w, Rect.newBy( xoff2, 290,
               230, 20 ),
191 "LFSaw", 0, 0, 1, 0, 'linear');
192          in3Box = CheckBoxView.new( w, Rect.newBy( xoff2, 310,
               230, 20 ),
193 "BrownNoise ", 0, 0, 1, 0, 'linear');
194
195
196
197
198          filename = ":Sounds:redobles.aiff";
```

```
199         sound = SoundFile.new;
200         if (sound.read(filename), {
201
202                 Synth.dualScope({
203                         var signal;   // for audiofile stream
204                         var input,input1,input2,input3;     //
                                for effects processor
205                         var
206 lpf_sig,hpf_sig,bpf_sig,notch_sig,peakingEQ_sig,lowShelf_sig,
    highShelf_sig;
207
208
209                         signal = sound.data.at(0);
210
211                         // Different sources
212                         input1 = PlayBuf.ar(signal, sound.
                                sampleRate, 1, 0, 0,
213 signal.size-2)*dBtoA.value(gainSlider.kr);
214                         input2 = LFSaw.ar(200, 0.1)*dBtoA.value(
                                gainSlider.kr);
215                         input3 = BrownNoise.ar(0.1)*dBtoA.value(
                                gainSlider.kr);
216
217                         // Source mix
218                         input = Mix.ar([input1 * in1Box.kr,
219                                         input2 * in2Box.kr,
220                                         input3 * in3Box.kr]);
221
222
223
224                         // LPF
225
226                         lpf_sig = Pause.ar({
227                                 var alpha,sn,cs;
228                                 var a0,a1,a2,b0,b1,b2;
229                                 sn = sin(ftoW.value(omegaSlider.
                                        kr));
230                                 cs = cos(ftoW.value(omegaSlider.
                                        kr));
231                                 alpha = sn/(2*qSlider.kr);
232
233
234                                 b0 = 1 + alpha;
235                                 b1 = (-2*cs)/b0;
236                                 b2 = (1 - alpha)/b0;
237                                 a0 = (1 - cs)/(2*b0);
238                                 a1 = (1 - cs)/b0;
239                                 a2 = (1 - cs)/(2*b0);
240
```

```
241                             SOS.ar(input, a0, a1, a2, b1.neg
                                    , b2.neg);} , lpfBox.kr(0));
242
243
244
245
246                         // HPF
247
248                         hpf_sig = Pause.ar({
249                                 var alpha,sn,cs;
250                                 var a0,a1,a2,b0,b1,b2;
251                                 sn = sin(ftoW.value(omegaSlider.
                                        kr));
252                                 cs = cos(ftoW.value(omegaSlider.
                                        kr));
253                                 alpha = sn/(2*qSlider.kr);
254
255
256                                 b0 = 1 + alpha;
257                                 b1 = (-2*cs)/b0;
258                                 b2 = (1 - alpha)/b0;
259                                 a0 = (1 + cs)/(2*b0);
260                                 a1 = (1 + cs).neg/b0;
261                                 a2 = (1 + cs)/(2*b0);
262
263                                 SOS.ar(input, a0, a1, a2, b1.neg
                                        , b2.neg);} , hpfBox.kr(0));
264
265
266                         // BPF
267
268                         bpf_sig = Pause.ar({
269                                 var alpha,sn,cs;
270                                 var a0,a1,a2,b0,b1,b2;
271                                 sn = sin(ftoW.value(omegaSlider.
                                        kr));
272                                 cs = cos(ftoW.value(omegaSlider.
                                        kr));
273                                 alpha = sn/(2*qSlider.kr);
274
275
276                                 b0 = 1 + alpha;
277                                 b1 = (-2*cs)/b0;
278                                 b2 = (1 - alpha)/b0;
279                                 a0 = alpha/b0;
280                                 a1 = 0;
281                                 a2 = alpha.neg/b0;
282
```

```
283                         SOS.ar(input, a0, a1, a2, b1.neg
                                , b2.neg);} , bpfBox.kr(0));
284
285
286                 // NOTCH
287
288                 notch_sig = Pause.ar({
289                         var alpha,sn,cs;
290                         var a0,a1,a2,b0,b1,b2;
291                         sn = sin(ftoW.value(omegaSlider.
                                kr));
292                         cs = cos(ftoW.value(omegaSlider.
                                kr));
293                         alpha = sn/(2*qSlider.kr);
294
295
296                         b0 = 1 + alpha;
297                         b1 = (-2*cs)/b0;
298                         b2 = (1 - alpha)/b0;
299                         a0 = 1/b0;
300                         a1 = (-2*cs)/b0;
301                         a2 = 1/b0;
302
303                         SOS.ar(input, a0, a1, a2, b1.neg
                                , b2.neg);} , notchBox.kr(0))
                                ;
304
305                 // PEAKINGEQ
306
307                 peakingEQ_sig = Pause.ar({
308                         var alpha,sn,cs,a;
309                         var a0,a1,a2,b0,b1,b2;
310                         sn = sin(ftoW.value(omegaSlider.
                                kr));
311                         cs = cos(ftoW.value(omegaSlider.
                                kr));
312                         alpha = sn/(2*qSlider.kr);
313                         a = 10**(filterSlider.kr/40.0);
314
315                         b0 = 1 + alpha/a;
316                         b1 = (-2*cs)/b0;
317                         b2 = (1 - alpha/a)/b0;
318                         a0 = (1 + alpha*a)/b0;
319                         a1 = (-2*cs)/b0;
320                         a2 = (1 - alpha*a)/b0;
321
322                         SOS.ar(input, a0, a1, a2, b1.neg
                                , b2.neg);} , peakingEQBox.kr
                                (0));
```

```
323
324
325                     // LOWSHELF
326
327                     lowShelf_sig = Pause.ar({
328                             var alpha,beta,sn,cs,a;
329                             var a0,a1,a2,b0,b1,b2;
330                             sn = sin(ftoW.value(omegaSlider.
                                    kr));
331                             cs = cos(ftoW.value(omegaSlider.
                                    kr));
332                             alpha = sn/(2*qSlider.kr);
333                             a = 10**(filterSlider.kr/40.0);
334                             beta = sqrt((a**2 + 1)/
                                    shelfSlider.kr - (a - 1)**2);
335
336                             b0 =        (a + 1) + (a - 1)*cs
                                     + beta*sn;
337                             b1 =   (-2*((a - 1) + (a + 1)*cs
                                    ))/b0;
338                             b2 =       ((a + 1) + (a - 1)*cs
                                    - beta*sn)/b0;
339                             a0 =     a*((a + 1) - (a - 1)*cs
                                    + beta*sn)/b0;
340                             a1 =  (2*a*((a - 1) - (a + 1)*cs
                                    ))/b0;
341                             a2 =     a*((a + 1) - (a - 1)*cs
                                    - beta*sn)/b0;
342
343                             SOS.ar(input, a0, a1, a2, b1.neg
                                    , b2.neg);} , lowShelfBox.kr
                                    (0));
344
345
346                     // HIGHSHELF
347
348                     highShelf_sig = Pause.ar({
349                             var alpha,beta,sn,cs,a;
350                             var a0,a1,a2,b0,b1,b2;
351                             sn = sin(ftoW.value(omegaSlider.
                                    kr));
352                             cs = cos(ftoW.value(omegaSlider.
                                    kr));
353                             alpha = sn/(2*qSlider.kr);
354                             a = 10**(filterSlider.kr/40.0);
355                             beta = sqrt((a**2 + 1)/
                                    shelfSlider.kr - (a - 1)**2);
356
```

```
357                                     b0 =        (a + 1) - (a - 1)*cs
                                            + beta*sn;
358                                     b1 =    (2*((a - 1) - (a + 1)*cs
                                            ))/b0;
359                                     b2 =        ((a + 1) - (a - 1)*cs
                                            - beta*sn)/b0;
360                                     a0 =     a*((a + 1) + (a - 1)*cs
                                            + beta*sn)/b0;
361                                     a1 = (-2*a*((a - 1) + (a + 1)*cs
                                            ))/b0;
362                                     a2 =     a*((a + 1) + (a - 1)*cs
                                            - beta*sn)/b0;
363
364                                     SOS.ar(input, a0, a1, a2, b1.neg
                                            , b2.neg);} , highShelfBox.kr
                                            (0));
365
366
367                             Mix.ar([lpf_sig,hpf_sig,bpf_sig,
                                    notch_sig,peakingEQ_sig,lowShelf_sig,
                                    highShelf_sig]);
368
369
370                             },0.1);
371
372
373         },{ (filename ++ " not found.\n").post });
374
375         w.close;
376
377
378
379 )
```

A.10. SuperCollider: Compresión

```
1  (
2          // Compresión de amplitud
3
4          var w;
5          var caption1,caption2,caption3,captionfbSlider,
               captionddSlider;
6          var in1Box,in2Box,in3Box,captionSource;
7          var applyBox;
8          var captiongainSlider,gainSlider,gainNum;
9          var captionratioSlider,ratioSlider,ratioNum;
```

```
10          var captionthrSlider, thrSlider, thrNum;
11          var captionattackSlider, attackSlider, attackNum;
12          var captionreleaseSlider, releaseSlider, releaseNum;
13          var captiondelaySlider, delaySlider, delayNum;
14          var ampview, ampdbview, captionampview, captionampdbview;
15          var originalview, compressedview, captionoriginalview,
                captioncompressedview;
16          var est_amp_box, est_amp_label;
17          var attack_coef_label, release_coef_label, attack_coef_box
                , release_coef_box;
18          var rect_db_label, rect_db_box, rect_label, rect_box;
19          var factor_label, factor_box, output_label, output_box;
20          var bias_label, bias_box, input_db_label, input_db_box;
21
22          var filename, sound, signal;
23          var fore_color;
24
25          var attack_coef;
26          var release_coef;
27
28          var xoff1, xoff2;
29          var ftoW;
30
31          // convert dB to amplitude
32
33          var a_to_db;
34          var dBtoA, dB_to_A;
35
36
37          var show_value_in_box;
38
39          dBtoA = { arg decibels;
40                  r = 0.5011872336;
41                  r*10**(decibels/20.0);};
42
43      // 96 decibels scale
44          dB_to_A = { arg decibels;
45                r = 0.008229747050;
46                  r*10**(decibels/20.0);};
47
48          // 96 decibels scale
49          a_to_db = { arg amp;
50              r = 0.008229747050;
51              20*log(amp/r);};
52
53          // Hertz to radian frequency
54          ftoW = { arg hertz;
55                  2pi*(hertz/Synth.sampleRate);};
56
```

```
57
58          // Show any signal value on a number box
59          show_value_in_box = {arg input, box, period;
60                  Sequencer.kr({ var val;
61                          val = input.poll;
62                          box.value = val;
63                          val
64                  }, Impulse.kr(period));};
65
66
67
68          // GUI Window
69          w = GUIWindow.new("Compressor >>> Rodrigo F. Cadiz",
                Rect.newBy(500, 70,
70  550, 650));
71          w.backColor = Color.new(99,130,160);
72
73
74          // Offsets
75          xoff1 = 10;
76          xoff2 = 250;
77
78          // Colors
79          fore_color = Color.new(255,255,0);
80
81          // Various labels
82          caption1 = StringView.new( w, Rect.newBy(xoff1, 10, 228,
                20), "Advanced
83  Audio Signal Processing");
84          caption1.labelColor = fore_color;
85          caption2 = StringView.new( w, Rect.newBy(xoff1, 30, 328,
                20),
86  "Assignment 3: Compressor");
87          caption2.labelColor = fore_color;
88          caption3 = StringView.new( w, Rect.newBy(xoff1, 50, 128,
                20), "Rodrigo
89  F. Cadiz");
90          caption3.labelColor = fore_color;
91
92          // Input Gain
93          captiongainSlider = StringView.new( w, Rect.newBy(xoff2,
                10, 235, 20),
94  "Input gain [dB] ");
95        gainSlider = SliderView.new( w, Rect.newBy( xoff2, 30, 230,
              20 ),
96  "Mix", 0, -30, 6, 1, 'linear');
97        gainNum = NumericalView.new( w, Rect.newBy( xoff2+235, 30,
              50, 20 ),
98  "NumericalView", 0, -30, 6, 1, 'linear');
```

```
99      gainNum.backColor = fore_color;
100     gainSlider.action = { gainNum.value = gainSlider.value};
101
102
103     // Compression ratio
104         captionratioSlider = StringView.new( w, Rect.newBy(xoff2
                , 60, 235, 20),
105 "Compression ratio");
106     ratioSlider = SliderView.new( w, Rect.newBy( xoff2, 80,
            230, 20 ),
107 "Mix", 1.0, 1.0, 10.0 , 0.1, 'linear');
108     ratioNum = NumericalView.new( w, Rect.newBy( xoff2+235, 80,
             50, 20
109 ), "NumericalView", 1.0, 1.0, 10.0, 0.1, 'linear');
110     ratioNum.backColor = fore_color;
111     ratioSlider.action = { ratioNum.value = ratioSlider.value};
112
113
114     // Threshold
115         captionthrSlider = StringView.new( w, Rect.newBy(xoff2,
                110, 235, 20),
116 "Threshold [dB]");
117     thrSlider = SliderView.new( w, Rect.newBy( xoff2, 130, 230,
             20 ),
118 "Mix", 36.0, 0.0, 96.0 , 1, 'linear');
119     thrNum = NumericalView.new( w, Rect.newBy( xoff2+235, 130,
            50, 20 ),
120 "NumericalView", 36.0, 0.0, 96.0, 1, 'linear');
121     thrNum.backColor = fore_color;
122     thrSlider.action = { thrNum.value = thrSlider.value};
123
124
125     // Attack time
126         captionattackSlider = StringView.new( w, Rect.newBy(
                xoff2, 160, 235,
127 20), "Attack time [msec]");
128     attackSlider = SliderView.new( w, Rect.newBy( xoff2, 180,
            230, 20 ),
129 "Mix", 0.1, 0.001 , 40.0 , 0.0001, 'exponential');
130     attackNum = NumericalView.new( w, Rect.newBy( xoff2+235,
            180, 50, 20
131 ), "NumericalView", 10.0, 0.001, 40.0, 0.0001, 'exponential');
132     attackNum.backColor = fore_color;
133     attackSlider.action = { attackNum.value = attackSlider.
            value};
134
135         // Release time
136         captionreleaseSlider = StringView.new( w, Rect.newBy(
                xoff2, 210, 235,
```

```
137 20), "Release time [msec] ");
138       releaseSlider = SliderView.new( w, Rect.newBy( xoff2, 230,
              230, 20
139 ), "Mix", 0.1, 0.001 , 500.0 , 0.0001, 'exponential');
140       releaseNum = NumericalView.new( w, Rect.newBy( xoff2+235,
              230, 50,
141 20 ), "NumericalView", 50.0, 0.001, 500.0, 0.0001, 'exponential'
      );
142       releaseNum.backColor = fore_color;
143       releaseSlider.action = { releaseNum.value = releaseSlider.
              value};
144
145           // Delay time
146           captiondelaySlider = StringView.new( w, Rect.newBy(xoff2
                  , 260, 235, 20),
147 "Delay time [msec] ");
148       delaySlider = SliderView.new( w, Rect.newBy( xoff2, 280,
              230, 20 ),
149 "Mix", 50.0, 0.0 , 500.0 , 0.01, 'linear');
150       delayNum = NumericalView.new( w, Rect.newBy( xoff2+235,
              280, 50, 20
151 ), "NumericalView", 50.0, 0.0 ,500.0, 0.01, 'linear');
152       delayNum.backColor = fore_color;
153       delaySlider.action = { delayNum.value = delaySlider.value};
154
155
156           // Number boxes for variables
157
158           est_amp_label = StringView.new( w, Rect.newBy(xoff1,
                  207, 150, 20),
159 "Estimated amplitude [dB] ");
160           est_amp_box = NumericalView.new( w, Rect.newBy( xoff1
                  +160,207,50,20 ),
161 "amp", 0.0, 0.0, 100.0, 0.0001, 'linear');
162           attack_coef_label = StringView.new( w, Rect.newBy(xoff1,
                   230, 150, 20),
163 "Attack time coefficient");
164           attack_coef_box = NumericalView.new( w, Rect.newBy(
                  xoff1+160,230,50,20
165 ), "amp", 0.0, 0.0, 1.5, 0.00001, 'linear');
166           release_coef_label = StringView.new( w, Rect.newBy(xoff1
                  , 253, 150, 20),
167 "Release time coefficient [dB] ");
168           release_coef_box = NumericalView.new( w, Rect.newBy(
                  xoff1+160,253,50,20
169 ), "amp", 0.0, 0.0, 1.5, 0.00001, 'linear');
170           rect_label = StringView.new( w, Rect.newBy(xoff1, 276,
                  150, 20),
171 "Rectified signal [lin]");
```

```
172         rect_box = NumericalView.new( w, Rect.newBy( xoff1
                +160,276,50,20 ),
173 "amp", 0.0, 0.0, 1.5, 0.0001, 'linear');
174         rect_db_label = StringView.new( w, Rect.newBy(xoff1,
                300, 150, 20),
175 "Rectified signal [dB] ");
176         rect_db_box = NumericalView.new( w, Rect.newBy( xoff1
                +160,300,50,20 ),
177 "amp", 0.0, 0.0, 96.0, 0.0001, 'linear');
178         factor_label = StringView.new( w, Rect.newBy(xoff1, 324,
                 150, 20),
179 "Amplitude correction");
180         factor_box = NumericalView.new( w, Rect.newBy( xoff1
                +160,324,50,20 ),
181 "amp", 0.0, 0.0, 500, 0.0001, 'linear');
182         output_label = StringView.new( w, Rect.newBy(xoff1, 347,
                 150, 20),
183 "Output signal [lin] ");
184         output_box = NumericalView.new( w, Rect.newBy( xoff1
                +160,347,50,20 ),
185 "amp", 0.0, 0.0, 30.0, 0.0001, 'linear');
186
187         input_db_label = StringView.new( w, Rect.newBy(xoff2,
                324, 125, 20),
188 "Input signal [dB] ");
189         input_db_box = NumericalView.new( w, Rect.newBy( xoff2
                +160,324,50,20 ),
190 "bias", 0.0, 0.0, 100.0, 0.001, 'linear');
191         bias_label = StringView.new( w, Rect.newBy(xoff2, 347,
                125, 20), "Bias ");
192         bias_box = NumericalView.new( w, Rect.newBy( xoff2
                +160,347,50,20 ),
193 "bias", 0.0, 0.0, 100.0, 0.001, 'linear');
194
195
196         // Apply compressor
197     applyBox = CheckBoxView.new( w, Rect.newBy( xoff1, 77, 230,
             20 ),
198 "Apply compressor ", 1, 0, 1, 0, 'linear');
199
200     // Select source
201     captionSource = StringView.new( w, Rect.newBy(xoff1, 100,
            235, 20),
202 "Select source");
203     captionSource.labelColor = fore_color;
204     in1Box = CheckBoxView.new( w, Rect.newBy( xoff1, 120, 230,
            20 ),
205 "Sound file ", 1, 0, 1, 0, 'linear');
```

```
206         in2Box = CheckBoxView.new( w, Rect.newBy( xoff1, 140, 230,
                20 ),
207 "Sinusoid", 0, 0, 1, 0, 'linear');
208         in3Box = CheckBoxView.new( w, Rect.newBy( xoff1, 160, 230,
                20 ),
209 "BrownNoise ", 0, 0, 1, 0, 'linear');
210
211
212         // Scopes
213
214         captionampview = StringView.new( w, Rect.newBy(xoff1,
                370, 235, 20),
215 "RMS estimator");
216         ampview = ScopeView.new( w, Rect.newBy(xoff1, 395, 200,
                100), 4410, 0, 1);
217
218         captionampdbview = StringView.new( w, Rect.newBy(xoff1,
                500, 235, 20),
219 "RMS estimator [dB]");
220         ampdbview = ScopeView.new( w, Rect.newBy(xoff1, 525,
                200, 100), 4410, 0, 96);
221
222         captionoriginalview = StringView.new( w, Rect.newBy(
                xoff2, 370, 235,
223 20), "Original signal");
224         originalview = ScopeView.new( w, Rect.newBy(xoff2, 395,
                200, 100), 4410,
225 -1, 1);
226
227         captioncompressedview = StringView.new( w, Rect.newBy(
                xoff2, 500, 235,
228 20), "Compressed signal");
229         compressedview = ScopeView.new( w, Rect.newBy(xoff2,
                525, 200, 100),
230 4410, -1, 1);
231
232         filename = ":Sounds:redobles.aiff";
233         sound = SoundFile.new;
234         if (sound.read(filename), {
235
236                 Synth.play({
237                         var signal;   // for audiofile stream
238                         var input,input1,input2,input3;    //
                                for effects processor
239                         var rect,rect_db,derivative,est_amp,
                                est_amp_db;
240                         var bias,delayed_input;
241
242                         signal = sound.data.at(0);
```

```
243
244                             // Different sources
245                             input1 = PlayBuf.ar(signal, sound.
                                    sampleRate, 1, 0, 0,
246 signal.size-2)*dBtoA.value(gainSlider.kr);
247                             input2 = SinOsc.ar(200, 0.1)*dBtoA.value
                                    (gainSlider.kr);
248                             input3 = BrownNoise.ar(0.1)*dBtoA.value(
                                    gainSlider.kr);
249
250                             // Source mix
251                             input = Mix.ar([input1 * in1Box.kr,
252                                             input2 * in2Box.kr,
253                                             input3 * in3Box.kr]);
254
255
256                             // Filter coefficients
257                             attack_coef = 10**(-2.0/((attackSlider.
                                    kr*0.001)*Synth.sampleRate));
258                             release_coef = 10**(-2.0/((releaseSlider
                                    .kr*0.001)*Synth.sampleRate));
259
260
261                             // Rectify the signal
262                             rect = abs(input);
263                             rect_db = a_to_db.value(rect);
264
265                             // Take derivative
266                             //derivative = HPZ1.ar(rect_db);
267                             derivative = HPZ1.ar(rect);
268
269
270                             // Filtered signal
271                             //est_amp = OnePole.ar(rect,if (
                                    derivative > 0, attack_coef,
                                    release_coef),
272                             //                                  if
                                    (derivative > 0, (1.0 - attack_coef),
                                     (1.0 - release_coef)));
273
274
275                              //est_amp = OnePole.ar(rect_db,if (
                                     derivative > 0, attack_coef,
276 release_coef));
277
278                              est_amp = OnePole.ar(rect,if (
                                     derivative > 0, attack_coef,
                                     release_coef));
279
```

```
280                     //est_amp = OnePole.ar(rect_db,
                            attack_coef);
281
282                     // est_amp_db = ampdb(est_amp);
283                     est_amp_db = a_to_db.value(est_amp);
284                     // est_amp_db = 20*log(est_amp);
285
286
287                     bias = (96 - (thrSlider.kr+((96.0 -
                            thrSlider.kr)/ratioSlider.kr)));
288
289                     //Peep.kr(bias,"bias");
290                     //Peep.kr(derivative,"derivative",10);
291                     //Peep.kr(rect_dB),"rect in dB",10);
292                     //Peep.kr(est_amp,"estimated amplitude
                            in dB",10);
293
294                     //Peep.kr(attack_coef,"attack",1);
295                     //Peep.kr(release_coef,"release",1);
296
297                     //bias = 0;
298
299
300
301                     //ampestNum.value = est_amp;
302
303                     a = max(est_amp_db,thrSlider.kr);
304                     b = a - thrSlider.kr;
305                     c = b * (1 - (1/ratioSlider.kr));
306                     d = bias - c;
307                     //d = c;
308                     //e = dB_to_A.value(d);
309                     //e = dbamp(d);
310                     e = 10**(d/20.0);
311
312
313                     delayed_input = DelayN.ar(input
                            ,500*0.001,delaySlider.kr);
314
315                     show_value_in_box.value(gainSlider.kr
                            +90,input_db_box,7);
316                     show_value_in_box.value(bias,bias_box,7)
                            ;
317                     show_value_in_box.value(est_amp_db,
                            est_amp_box,7);
318                     show_value_in_box.value(attack_coef,
                            attack_coef_box,5);
319                     show_value_in_box.value(release_coef,
                            release_coef_box,5);
```

```
320                             show_value_in_box.value(rect,rect_box,7)
                                    ;
321                             show_value_in_box.value(rect_db,
                                    rect_db_box,7);
322                             show_value_in_box.value(e,factor_box,7);
323                             show_value_in_box.value(delayed_input*e*
                                    applyBox.kr,output_box,7);
324
325
326                             Scope.ar(ampview,est_amp);
327                             Scope.ar(ampdbview,est_amp_db);
328                             Scope.ar(originalview,delayed_input);
329                             Scope.ar(compressedview,delayed_input*e*
                                    applyBox.kr);
330
331                             Mix.ar([delayed_input*e*applyBox.kr,
                                    delayed_input*(1.0 - applyBox.kr)]);
332
333
334                             });
335
336
337         },{ (filename ++ " not found.\n").post });
338
339         w.close;
340
341
342
343 )
```

A.11. SuperCollider: Reverberación

```
1  (
2          // Reverberación
3
4          var w;
5          var caption1,caption2,caption3,captionfbSlider,
               captionddSlider;
6          var in1Box,in2Box,in3Box,in4Box,captionSource;
7          var applyBox;
8          var captiongainSlider,gainSlider,gainNum;
9          var captionreverbSlider,reverbSlider,reverbNum;
10         var captionpredelaySlider,predelaySlider,predelayNum;
11         var captionbwSlider,bwSlider,bwNum;
12         var captiondecaySlider,decaySlider,decayNum;
13         var captiondampingSlider,dampingSlider,dampingNum;
```

```
14          var captionindiff1Slider,indiff1Slider,indiff1Num;
15          var captionindiff2Slider,indiff2Slider,indiff2Num;
16          var captiondecdiff1Slider,decdiff1Slider,decdiff1Num;
17          var captiondecdiff2Slider,decdiff2Slider,decdiff2Num;
18          var captionrate1Slider,rate1Slider,rate1Num;
19          var captionrate2Slider,rate2Slider,rate2Num;
20          var captionexcursionSlider,excursionSlider,excursionNum;
21          var captiondecaytimeSlider,decaytimeSlider,decaytimeNum;
22
23
24          var
25  view1,view2,view3,view4,captionview1,captionview2,captionview3,
        captionview4;
26          var view5,view6,captionview5,captionview6;
27
28
29          var diff1_coef_label,diff1_coef_box;
30          var diff2_coef_label,diff2_coef_box;
31          var diff3_coef_label,diff3_coef_box;
32          var diff4_coef_label,diff4_coef_box;
33          var decdiff1_coef_label,decdiff1_coef_box;
34          var decdiff2_coef_label,decdiff2_coef_box;
35          var decdiff3_coef_label,decdiff3_coef_box;
36          var decdiff4_coef_label,decdiff4_coef_box;
37
38
39          var filename, sound, signal;
40          var fore_color;
41
42          var attack_coef;
43          var release_coef;
44
45          var xoff1,xoff2,xoff3;
46          var ftoW;
47
48          // convert dB to amplitude
49
50          var a_to_db;
51          var dBtoA,dB_to_A;
52
53          var convert_delay,convert_sample,revtime;
54          var show_value_in_box;
55
56          dBtoA = { arg decibels;
57                  r = 0.5011872336;
58                  r*10**(decibels/20.0);};
59
60        // 96 decibels scale
61          dB_to_A = { arg decibels;
```

```
62              r = 0.008229747050;
63                  r*10**(decibels/20.0);};
64
65          // 96 decibels scale
66          a_to_db = { arg amp;
67              r = 0.008229747050;
68              20*log(amp/r);};
69
70          // Hertz to radian frequency
71          ftoW = { arg hertz;
72                  2pi*(hertz/Synth.sampleRate);};
73
74
75
76          // Convert samples to delay time
77          revtime = {arg coef, delay;
78                  (-6.907755279*delay)/log(coef);};
79
80          // Convert samples to 44.1 sampling rate
81          convert_sample = {arg samples;
82                  (samples*(44100/29761)).ceil;};
83
84          // Convert delays to 44.1 sampling rate
85          convert_delay = {arg delay;
86                  delay/29761;};
87
88
89          // Show any signal value on a number box
90          show_value_in_box = {arg input, box, period;
91                  Sequencer.kr({ var val;
92                          val = input.poll;
93                          box.value = val;
94                          val
95                  }, Impulse.kr(period));};
96
97
98
99          // GUI Window
100         w = GUIWindow.new("Reverberator >>> Rodrigo F. Cadiz",
                Rect.newBy(200,
101 70, 850, 650));
102         w.backColor = Color.new(99,130,160);
103
104
105         // Offsets
106         xoff1 = 10;
107         xoff2 = 250;
108         xoff3 = 550;
109
```

```
110         // Colors
111         fore_color = Color.new(255,255,0);
112
113         // Various labels
114         caption1 = StringView.new( w, Rect.newBy(xoff1, 10, 228,
                20), "Advanced
115 Audio Signal Processing");
116         caption1.labelColor = fore_color;
117         caption2 = StringView.new( w, Rect.newBy(xoff1, 30, 328,
                20),
118 "Assignment 4: Reverberator");
119         caption2.labelColor = fore_color;
120         caption3 = StringView.new( w, Rect.newBy(xoff1, 50, 128,
                20), "Rodrigo
121 F. Cadiz");
122         caption3.labelColor = fore_color;
123
124
125         // Input Gain
126         captiongainSlider = StringView.new( w, Rect.newBy(xoff2,
                10, 235, 20),
127 "Dry gain [dB] ");
128       gainSlider = SliderView.new( w, Rect.newBy( xoff2, 30, 230,
              20 ),
129 "Mix", -11, -60 , 6 , 1, 'linear');
130       gainNum = NumericalView.new( w, Rect.newBy( xoff2+235, 30,
              50, 20 ),
131 "NumericalView", -11, -60, 6, 1, 'linear');
132       gainNum.backColor = fore_color;
133       gainSlider.action = { gainNum.value = gainSlider.value};
134
135
136       // Reverb gain
137          captionreverbSlider = StringView.new( w, Rect.newBy(
                 xoff2, 60, 235, 20),
138 "Reverb gain [dB]");
139       reverbSlider = SliderView.new( w, Rect.newBy( xoff2, 80,
              230, 20 ),
140 "Mix", 0, -60, 6 , 1, 'linear');
141       reverbNum = NumericalView.new( w, Rect.newBy( xoff2+235,
              80, 50, 20
142 ), "NumericalView", 0, -60, 6, 1, 'linear');
143       reverbNum.backColor = fore_color;
144       reverbSlider.action = { reverbNum.value = reverbSlider.
              value};
145
146
147       // Predelay
```

```
148         captionpredelaySlider = StringView.new( w, Rect.newBy(
                xoff2, 110, 235,
149 20), "Predelay [s]");
150         predelaySlider = SliderView.new( w, Rect.newBy( xoff2, 130,
                230, 20
151 ), "Mix", 0.0231, 0.0, 0.2 , 0.0001, 'linear');
152         predelayNum = NumericalView.new( w, Rect.newBy( xoff2+235,
                130, 50,
153 20 ), "NumericalView", 0.0231, 0.0, 0.2, 0.0001, 'linear');
154         predelayNum.backColor = fore_color;
155         predelaySlider.action = { predelayNum.value =
                predelaySlider.value};
156
157
158         // Bandwidth
159         captionbwSlider = StringView.new( w, Rect.newBy(xoff2,
                160, 235, 20),
160 "Bandwidth ");
161         bwSlider = SliderView.new( w, Rect.newBy( xoff2, 180, 230,
                20 ),
162 "Mix", 0.7995, 0.00001 , 0.99999 , 0.00001, 'linear');
163         bwNum = NumericalView.new( w, Rect.newBy( xoff2+235, 180,
                50, 20 ),
164 "NumericalView", 0.7995, 0.000001, 0.99999, 0.00001, 'linear');
165         bwNum.backColor = fore_color;
166         bwSlider.action = { bwNum.value = bwSlider.value};
167
168         // Decay
169         captiondecaySlider = StringView.new( w, Rect.newBy(xoff2
                , 210, 235, 20),
170 "Decay factor");
171         decaySlider = SliderView.new( w, Rect.newBy( xoff2, 230,
                230, 20 ),
172 "Mix", 0.75, 0.0 , 1.0 , 0.0001, 'linear');
173         decayNum = NumericalView.new( w, Rect.newBy( xoff2+235,
                230, 50, 20
174 ), "NumericalView", 0.75, 0.0, 1.0, 0.0001, 'linear');
175         decayNum.backColor = fore_color;
176         decaySlider.action = { decayNum.value = decaySlider.value};
177
178         // Damping
179         captiondampingSlider = StringView.new( w, Rect.newBy(
                xoff2, 260, 235,
180 20), "Damping ");
181         dampingSlider = SliderView.new( w, Rect.newBy( xoff2, 280,
                230, 20
182 ), "Mix", 0.3, 0.0 , 1.0 , 0.000001, 'linear');
183         dampingNum = NumericalView.new( w, Rect.newBy( xoff2+235,
                280, 50,
```

```
184 20 ), "NumericalView", 0.3, 0.0 ,1.0, 0.000001, 'linear');
185         dampingNum.backColor = fore_color;
186         dampingSlider.action = { dampingNum.value = dampingSlider.
                value};
187
188             // Input diff 1
189             captionindiff1Slider = StringView.new( w, Rect.newBy(
                    xoff3, 10, 235,
190 20), "Input diff 1 ");
191         indiff1Slider = SliderView.new( w, Rect.newBy( xoff3, 30,
                230, 20 ),
192 "Mix", 0.945, 0.0 , 1.0 , 0.0001, 'linear');
193         indiff1Num = NumericalView.new( w, Rect.newBy( xoff3+235,
                30, 50, 20
194 ), "NumericalView", 0.945, 0.0, 1.0, 0.0001, 'linear');
195         indiff1Num.backColor = fore_color;
196         indiff1Slider.action = { indiff1Num.value = indiff1Slider.
                value};
197
198             // Input diff 2
199             captionindiff2Slider = StringView.new( w, Rect.newBy(
                    xoff3, 60, 235,
200 20), "Input diff 2 ");
201         indiff2Slider = SliderView.new( w, Rect.newBy( xoff3, 80,
                230, 20 ),
202 "Mix", 0.915, 0.0 , 1.0 , 0.0001, 'linear');
203         indiff2Num = NumericalView.new( w, Rect.newBy( xoff3+235,
                80, 50, 20
204 ), "NumericalView", 0.915, 0.0, 1.0, 0.0001, 'linear');
205         indiff2Num.backColor = fore_color;
206         indiff2Slider.action = { indiff2Num.value = indiff2Slider.
                value};
207
208             // Decay diff 1
209             captiondecdiff1Slider = StringView.new( w, Rect.newBy(
                    xoff3, 110, 235,
210 20), "Decay diff 1 ");
211         decdiff1Slider = SliderView.new( w, Rect.newBy( xoff3, 130,
                 230, 20
212 ), "Mix", 0.965, 0.0 , 1.0 , 0.0001, 'linear');
213         decdiff1Num = NumericalView.new( w, Rect.newBy( xoff3+235,
                130, 50,
214 20 ), "NumericalView", 0.965, 0.0 , 1.0 , 0.0001, 'linear');
215         decdiff1Num.backColor = fore_color;
216         decdiff1Slider.action = { decdiff1Num.value =
                decdiff1Slider.value};
217
218             // Decay diff 2
```

```
219         captiondecdiff2Slider = StringView.new( w, Rect.newBy(
                xoff3, 160, 235,
220 20), "Decay diff 2 ");
221         decdiff2Slider = SliderView.new( w, Rect.newBy( xoff3, 180,
                230, 20
222 ), "Mix", 0.815, 0.25 , 0.85 , 0.0001, 'linear');
223         decdiff2Num = NumericalView.new( w, Rect.newBy( xoff3+235,
                180, 50,
224 20 ), "NumericalView", 0.815, 0.25 , 0.85 , 0.0001, 'linear');
225         decdiff2Num.backColor = fore_color;
226         decdiff2Slider.action = { decdiff2Num.value =
                decdiff2Slider.value};
227
228         // Rate 1
229             captionrate1Slider = StringView.new( w, Rect.newBy(xoff3
                    , 210, 235, 20),
230 "Osc. Rate 1 [Hz] ");
231         rate1Slider = SliderView.new( w, Rect.newBy( xoff3, 230,
                230, 20 ),
232 "Mix", 0.032, 0.0 , 0.3 , 0.001, 'linear');
233         rate1Num = NumericalView.new( w, Rect.newBy( xoff3+235,
                230, 50, 20
234 ), "NumericalView", 0.032, 0.0 ,0.3, 0.001, 'linear');
235         rate1Num.backColor = fore_color;
236         rate1Slider.action = { rate1Num.value = rate1Slider.value};
237
238             // Rate 2
239             captionrate1Slider = StringView.new( w, Rect.newBy(xoff3
                    , 260, 235, 20),
240 "Osc. Rate 2 [Hz]");
241         rate2Slider = SliderView.new( w, Rect.newBy( xoff3, 280,
                230, 20 ),
242 "Mix", 0.106, 0.0 , 0.3 , 0.001, 'linear');
243         rate2Num = NumericalView.new( w, Rect.newBy( xoff3+235,
                280, 50, 20
244 ), "NumericalView", 0.106, 0.0 ,0.3, 0.001, 'linear');
245         rate2Num.backColor = fore_color;
246         rate2Slider.action = { rate2Num.value = rate2Slider.value};
247
248             // Excursion
249             captionexcursionSlider = StringView.new( w, Rect.newBy(
                    xoff3, 310, 235,
250 20), "Excursion ");
251         excursionSlider = SliderView.new( w, Rect.newBy( xoff3,
                330, 230, 20
252 ), "Mix", 11, 0, 16 , 1, 'linear');
253         excursionNum = NumericalView.new( w, Rect.newBy( xoff3+235,
                330, 50,
254 20 ), "NumericalView", 11, 0 ,16, 1, 'linear');
```

```
255         excursionNum.backColor = fore_color;
256         excursionSlider.action = { excursionNum.value =
                excursionSlider.value};
257
258
259             // Decay time
260             captiondecaytimeSlider = StringView.new( w, Rect.newBy(
                    xoff2, 310, 235,
261     20), "Decay time [s] ");
262         decaytimeSlider = SliderView.new( w, Rect.newBy( xoff2,
                330, 230, 20
263     ), "Mix", 2.0, 0.01, 4.0 , 0.001, 'linear');
264         decaytimeNum = NumericalView.new( w, Rect.newBy( xoff2+235,
                330, 50,
265     20 ), "NumericalView", 2.0, 0.01 ,4.0, 0.001, 'linear');
266         decaytimeNum.backColor = fore_color;
267         decaytimeSlider.action = { decaytimeNum.value =
                decaytimeSlider.value};
268
269             // Number boxes for variables
270
271             diff1_coef_label = StringView.new( w, Rect.newBy(xoff1,
                    200, 150, 20),
272     "Input diff 1 [s]");
273             diff1_coef_box = NumericalView.new( w, Rect.newBy( xoff1
                    +160,200,50,20
274     ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
275
276             diff2_coef_label = StringView.new( w, Rect.newBy(xoff1,
                    220, 150, 20),
277     "Input diff 2 [s]");
278             diff2_coef_box = NumericalView.new( w, Rect.newBy( xoff1
                    +160,220,50,20
279     ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
280
281             diff3_coef_label = StringView.new( w, Rect.newBy(xoff1,
                    240, 150, 20),
282     "Input diff 3 [s]");
283             diff3_coef_box = NumericalView.new( w, Rect.newBy( xoff1
                    +160,240,50,20
284     ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
285
286             diff4_coef_label = StringView.new( w, Rect.newBy(xoff1,
                    260, 150, 20),
287     "Input diff 4 [s]");
288             diff4_coef_box = NumericalView.new( w, Rect.newBy( xoff1
                    +160,260,50,20
289     ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
290
```

```
291         decdiff1_coef_label = StringView.new( w, Rect.newBy(
                xoff1, 280, 150,
292 20), "Decay diff 1 [s]");
293         decdiff1_coef_box = NumericalView.new( w, Rect.newBy(
294 xoff1+160,280,50,20 ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
295
296         decdiff2_coef_label = StringView.new( w, Rect.newBy(
                xoff1, 300, 150,
297 20), "Decay diff 2 [s]");
298         decdiff2_coef_box = NumericalView.new( w, Rect.newBy(
299 xoff1+160,300,50,20 ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
300
301         decdiff3_coef_label = StringView.new( w, Rect.newBy(
                xoff1, 320, 150,
302 20), "Decay diff 3 [s]");
303         decdiff3_coef_box = NumericalView.new( w, Rect.newBy(
304 xoff1+160,320,50,20 ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
305
306         decdiff4_coef_label = StringView.new( w, Rect.newBy(
                xoff1, 340, 150,
307 20), "Decay diff 4 [s]");
308         decdiff4_coef_box = NumericalView.new( w, Rect.newBy(
309 xoff1+160,340,50,20 ), "amp", 0.0, 0.0, 10.0, 0.0001, 'linear');
310
311
312         // Apply compressor
313     applyBox = CheckBoxView.new( w, Rect.newBy( xoff1, 77, 230,
              20 ),
314 "Apply reverberation ", 1, 0, 1, 0, 'linear');
315
316     // Select source
317     captionSource = StringView.new( w, Rect.newBy(xoff1, 100,
            235, 20),
318 "Select source");
319     captionSource.labelColor = fore_color;
320     in1Box = CheckBoxView.new( w, Rect.newBy( xoff1, 120, 230,
            20 ),
321 "Sound file ", 1, 0, 1, 0, 'linear');
322     in2Box = CheckBoxView.new( w, Rect.newBy( xoff1, 140, 230,
            20 ),
323 "Sinusoid", 0, 0, 1, 0, 'linear');
324     in3Box = CheckBoxView.new( w, Rect.newBy( xoff1, 160, 230,
            20 ),
325 "BrownNoise ", 0, 0, 1, 0, 'linear');
326     in4Box = CheckBoxView.new( w, Rect.newBy( xoff1, 180, 230,
            20 ),
327 "Impulses ", 0, 0, 1, 0, 'linear');
328
329
```

```
330         // Scopes
331
332         captionview1 = StringView.new( w, Rect.newBy(xoff1, 370,
                235, 20),
333 "Original signal");
334         view1 = ScopeView.new( w, Rect.newBy(xoff1, 395, 200,
                100), 4410, -1, 1);
335
336         captionview2 = StringView.new( w, Rect.newBy(xoff1, 500,
                235, 20),
337 "Signal after BW filter");
338         view2 = ScopeView.new( w, Rect.newBy(xoff1, 525, 200,
                100), 4410, -1, 1);
339
340         captionview3 = StringView.new( w, Rect.newBy(xoff2, 370,
                235, 20),
341 "Signal after diffusors");
342         view3 = ScopeView.new( w, Rect.newBy(xoff2, 395, 200,
                100), 4410, -1, 1);
343
344         //captionview4 = StringView.new( w, Rect.newBy(xoff2,
                500, 235, 20),
345 "Compressed signal");
346         //view4 = ScopeView.new( w, Rect.newBy(xoff2, 525, 200,
                100), 4410, -1, 1);
347
348         captionview5 = StringView.new( w, Rect.newBy(xoff3, 370,
                235, 20), "Left
349 channel output");
350         view5 = ScopeView.new( w, Rect.newBy(xoff3, 395, 200,
                100), 4410, -1, 1);
351
352         captionview6 = StringView.new( w, Rect.newBy(xoff3, 500,
                235, 20),
353 "Right channel output");
354         view6 = ScopeView.new( w, Rect.newBy(xoff3, 525, 200,
                100), 4410, -1, 1);
355
356         filename = ":Sounds:redobles.aiff";
357         sound = SoundFile.new;
358         if (sound.read(filename), {
359
360                 Synth.play({
361                         var signal;   // for audiofile stream
362                         var input,input1,input2,input3,input4;
                                // for effects processor
363
364                         var diff1,diff2,diff3,diff4;
365                 var decdiff1,decdiff2,decdiff3,decdiff4;
```

```
366                     var in2,in3,in4,in5,in6,in7,in8;
367                     var outL,outL2,outL3,outL4,outL5,outL6,
                            outL7,outL1;
368                     var outR,outR2,outR3,outR4,outR5,outR6,
                            outR7,outR1;
369                     var outL8=0;
370                     var outR8=0;
371                     var osc1,osc2,osc3,osc4;
372                     var buffer_left,buffer_right,delL,delR;
373
374
375                     signal = sound.data.at(0);
376
377                     // Different sources
378                     input1 = PlayBuf.ar(signal, sound.
                            sampleRate, 1, 0, 0, signal.size-2);
379                     input2 = SinOsc.ar(200, 0.1);
380                     input3 = BrownNoise.ar(0.1);
381                     input4 = Impulse.ar(100,0.2);
382
383                     // Source mix
384                     input = Mix.ar([input1 * in1Box.kr,
385                                     input2 * in2Box.kr,
386                                     input3 * in3Box.kr,
387                                     input4 * in4Box.kr]);
388
389
390
391                     // Amplification factor
392                     in2 = 1.0*input;
393
394                     // Predelay
395                     in3 = DelayN.ar(in2,1.0,predelaySlider.
                            kr);
396
397                     // First filter for bandwidth
398                     in4 = OnePole.ar(in3,(1.0-bwSlider.kr));
399
400
401                     diff1 = revtime.value(indiff1Slider.kr,
                            convert_delay.value(142));
402                     diff2 = revtime.value(indiff1Slider.kr,
                            convert_delay.value(107));
403                     diff3 = revtime.value(indiff2Slider.kr,
                            convert_delay.value(379));
404                     diff4 = revtime.value(indiff2Slider.kr,
                            convert_delay.value(277));
405
406
```

```
407                             // First diffusion filter
408                             in5 = AllpassN.ar(in4,1.0,convert_delay.
                                    value(142),diff1);
409
410                             // Second diffusion filter
411                             in6 = AllpassN.ar(in5,1.0,convert_delay.
                                    value(107),diff2);
412
413                             // Third diffusion filter
414                             in7 = AllpassN.ar(in6,1.0,convert_delay.
                                    value(379),diff3);
415
416                             // Fourth diffusion filter
417                             in8 = AllpassN.ar(in7,1.0,convert_delay.
                                    value(277),diff4);
418
419
420                             buffer_left = Signal.new(Synth.
                                    sampleRate*0.3);
421                             buffer_right = Signal.new(Synth.
                                    sampleRate*0.3);
422
423                             delL = TapN.ar(buffer_left,convert_delay
                                    .value(3720));
424                             delR = TapN.ar(buffer_right,
                                    convert_delay.value(3163));
425
426                             decdiff1 =
427 revtime.value(decdiff1Slider.kr,convert_delay.value(672+
        excursionSlider.kr));
428                             decdiff2 =
429 revtime.value(decdiff1Slider.kr,convert_delay.value(908+
        excursionSlider.kr));
430                             decdiff3 = revtime.value(decdiff2Slider.
                                    kr,convert_delay.value(1800));
431                             decdiff4 = revtime.value(decdiff2Slider.
                                    kr,convert_delay.value(2656));
432
433                             outL2 =
434 AllpassN.ar(delL,1.0,convert_delay.value(672+excursionSlider.kr)
        ,decdiff1.neg);
435                             outR2 =
436 AllpassN.ar(delR,1.0,convert_delay.value(908+excursionSlider.kr)
        ,decdiff2.neg);
437
438                             osc1 = SinOsc.kr(rate1Slider.kr,0.0,
                                    convert_delay.value(4453)*0.5);
439                             osc2 = SinOsc.kr(rate2Slider.kr,0.0,
                                    convert_delay.value(4217)*0.5);
```

```
440
441                             outL3 = DelayN.ar(outL2,1.0,(
                                    convert_delay.value(4453)+osc1));
442                             outR3 = DelayN.ar(outR2,1.0,(
                                    convert_delay.value(4217)+osc2));
443
444                             outL4 = OnePole.ar(outL3,dampingSlider.
                                    kr);
445                             outR4 = OnePole.ar(outR3,dampingSlider.
                                    kr);
446
447                             outL5 = decaySlider.kr*outL4;
448                             outR5 = decaySlider.kr*outR4;
449
450                             outL6 =
451 decaySlider.kr*AllpassN.ar(outL5,1.0,convert_delay.value(1800),
    decdiff3);
452                             outR6 =
453 decaySlider.kr*AllpassN.ar(outR5,1.0,convert_delay.value(2656),
    decdiff4);
454
455                             outL1 = dBtoA.value(reverbSlider.kr)*
                                    applyBox.kr*(in8 + outR6);
456                             outR1 = dBtoA.value(reverbSlider.kr)*
                                    applyBox.kr*(in8 + outL6);
457
458                             DelayWr.ar(buffer_left,outL1);
459                             DelayWr.ar(buffer_right,outR1);
460
461
462                             show_value_in_box.value(diff1,
                                    diff1_coef_box,7);
463                             show_value_in_box.value(diff2,
                                    diff2_coef_box,7);
464                             show_value_in_box.value(diff3,
                                    diff3_coef_box,7);
465                             show_value_in_box.value(diff4,
                                    diff4_coef_box,7);
466
467                             show_value_in_box.value(decdiff1,
                                    decdiff1_coef_box,7);
468                             show_value_in_box.value(decdiff2,
                                    decdiff2_coef_box,7);
469                             show_value_in_box.value(decdiff3,
                                    decdiff3_coef_box,7);
470                             show_value_in_box.value(decdiff4,
                                    decdiff4_coef_box,7);
471
472
```

```
473
474                             Scope.ar(view1,input);
475                             Scope.ar(view2,in4);
476                             Scope.ar(view3,in8);
477
478                             Scope.ar(view5,outL1);
479                             Scope.ar(view6,outR1);
480
481                             [Mix.ar([dBtoA.value(gainSlider.kr)*
                                    input,outL1]),
482                              Mix.ar([dBtoA.value(gainSlider.kr)*
                                    input,outR1])];
483
484
485                             });
486
487
488         },{ (filename ++ " not found.\n").post });
489
490         w.close;
491
492
493
494 )
```

A.12. SuperCollider: Procesador de efectos genérico

```
1  // Procesador de efectos generico
2  // Rodrigo F. Cadiz
3  // Codigo para SuperCollider 2
4
5  var w;
6  var caption1,caption2,caption3,captionfbSlider,captionddSlider;
7  var captionfreqSlider,captiondelaySlider,captionmixSlider,
       captiongainSlider;
8  var vdBox, radio1, range1;
9  var freqSlider,delaySlider,mixSlider,gainSlider,fbBox,fbSlider,
       ddSlider;
10 var delayNum,freqNum,mixNum,gainNum,fbNum,ddNum;
11 var in1Box,in2Box,in3Box,captionSource;
12 var filename, sound, signal;
13 var fore_color;
14 var xoff1,xoff2;
15
16 // convert dB to amplitude
```

```
17 var dBtoA;
18 dBtoA = { arg decibels; r = 0.5011872336; r*10**(decibels/20.0)
       ;};
19
20 // GUI Window
21 w = GUIWindow.new("Generic Effects Processor >>> Rodrigo F.
       Cadiz", Rect.newBy(500, 70, 650, 400));
22 w.backColor = Color.new(99,130,160);
23
24 // Offsets
25 xoff1 = 10;
26 xoff2 = 350;
27
28 // Colors
29 fore_color = Color.new(255,255,0);
30
31 // Various labels
32 caption1 = StringView.new( w, Rect.newBy(xoff1, 10, 228, 20), "
       Advanced Audio Signal Processing");
33 caption1.labelColor = fore_color;
34 caption2 = StringView.new( w, Rect.newBy(xoff1, 30, 328, 20), "
       Generic Effects Processor");
35 caption2.labelColor = fore_color;
36 caption3 = StringView.new( w, Rect.newBy(xoff1, 50, 128, 20), "
       Rodrigo F. Cadiz");
37 caption3.labelColor = fore_color;
38
39 // Mix
40 captionmixSlider = StringView.new( w, Rect.newBy(xoff1, 90, 235,
        20), "Output Mix : 0.0 (dry) - 1.0 (wet) ");
41 mixSlider = SliderView.new( w, Rect.newBy( xoff1, 110, 230, 20 )
       , "Mix", 0.5, 0.0, 1.0, 0.1, 'linear');
42 mixNum = NumericalView.new( w, Rect.newBy( xoff1+235, 110, 50,
       20 ), "NumericalView", 0.5, 0.0, 1.0, 0.1, 'linear');
43 mixNum.backColor = fore_color;
44 mixSlider.action = { mixNum.value = mixSlider.value};
45
46 // Input Gain
47 captiongainSlider = StringView.new( w, Rect.newBy(xoff2, 10,
       235, 20),"Input gain [dB] ");
48 gainSlider = SliderView.new( w, Rect.newBy( xoff2, 30, 230, 20 )
       ,"Mix", 0, -30 , 6 , 1, 'linear');
49 gainNum = NumericalView.new( w, Rect.newBy( xoff2+235, 30, 50,
       20 ), "NumericalView", 0, -30, 6, 1, 'linear');
50 gainNum.backColor = fore_color;
51 gainSlider.action = { gainNum.value = gainSlider.value};
52
53 // Feedback phase
```

```
54 fbBox = CheckBoxView.new( w, Rect.newBy( xoff2, 65, 230, 20 ), "
       Feedback phase : -1 (off) +1 (on) ", 1, 0, 1, 0, 'linear');
55
56 // Feedback
57 captionfbSlider = StringView.new( w, Rect.newBy( xoff2, 90, 230,
       20), "Feedback [%]");
58 fbSlider = SliderView.new( w, Rect.newBy( xoff2, 110, 230, 20 ),
       "Feedback", 50, 0, 99, 1, 'linear');
59 fbNum = NumericalView.new( w, Rect.newBy( xoff2+235, 110, 50, 20
       ), "NumericalView", 50, 0, 99, 1, 'linear');
60 fbNum.backColor = fore_color;
61 fbSlider.action = { fbNum.value = fbSlider.value};
62
63 // Delay time
64 captiondelaySlider = StringView.new( w, Rect.newBy(xoff1, 170,
       235, 20), "Delay time : (seconds)");
65 delaySlider = SliderView.new( w, Rect.newBy( xoff1, 190, 230, 20
       ),"Delay time", 0.001, 0, 0.05, 0.0005, 'linear');
66 delayNum = NumericalView.new( w, Rect.newBy( xoff1+235, 190, 50,
       20), "NumericalView", 0.001, 0, 1, 0.0005, 'linear');
67 delayNum.backColor = fore_color;
68 delaySlider.action = { delayNum.value = delaySlider.value};
69
70 // Delay switch
71 vdBox = CheckBoxView.new( w, Rect.newBy(xoff2, 145, 428, 20), "
       Variable delay", 0, 0, 1, 0, 'linear');
72
73 // Oscillator frequency
74 captionfreqSlider = StringView.new( w, Rect.newBy(xoff2, 170,
       235, 20),"Oscillator Frequency (delay) [Hz]");
75 freqSlider = SliderView.new( w, Rect.newBy( xoff2, 190, 230, 20
       ),"SliderView", 10, 0, 20, 1, 'linear');
76 freqNum = NumericalView.new( w, Rect.newBy( xoff2+235, 190, 50,
       20), "NumericalView", 10, 0, 20, 1, 'linear');
77 freqNum.backColor = fore_color;
78 freqSlider.action = { freqNum.value = freqSlider.value};
79
80 // Delay deviation
81 captionddSlider = StringView.new( w, Rect.newBy( xoff1, 240,
       230, 20),"Delay deviation : 0% - 99%");
82 ddSlider = SliderView.new( w, Rect.newBy( xoff1, 260, 230, 20 ),
       "Deviation", 50, 0, 99, 1, 'linear');
83 ddNum = NumericalView.new( w, Rect.newBy( xoff1+235, 260, 50, 20
       ),"NumericalView", 50, 0, 99, 1, 'linear');
84 ddNum.backColor = fore_color;
85 ddSlider.action = { ddNum.value = ddSlider.value};
86
87 // Select source
```

```
88  captionSource = StringView.new( w, Rect.newBy(xoff2, 220, 235,
        20),"Select source");
89  captionSource.labelColor = fore_color;
90  in1Box = CheckBoxView.new( w, Rect.newBy( xoff2, 240, 230, 20 ),
        "Soundfile ", 1, 0, 1, 0, 'linear');
91  in2Box = CheckBoxView.new( w, Rect.newBy( xoff2, 260, 230, 20 ),
        "LFSaw", 0, 0, 1, 0, 'linear');
92  in3Box = CheckBoxView.new( w, Rect.newBy( xoff2, 280, 230, 20 ),
        "BrownNoise ", 0, 0, 1, 0, 'linear');
93
94  filename = ":Sounds:redobles.aiff";
95  sound = SoundFile.new;
96
97  if (sound.read(filename), {
98          Synth.dualScope({
99
100         var signal;// for audiofile stream
101         var input,input1,input2,input3;// for effects processor
102         var buffer;// for delay line
103         var msignal;// wet signal
104         var delsignal=0;// delayed signal
105
106         buffer = Signal.newClear(Synth.sampleRate * 3 * 0.5);
107         signal = sound.data.at(0);
108
109         // Different sources
110         input1 = PlayBuf.ar(signal, sound.sampleRate, 1, 0, 0,
                signal.size-2)*dBtoA.value(gainSlider.kr);
111         input2 = LFSaw.ar(200, 0.1)*dBtoA.value(gainSlider.kr);
112         input3 = BrownNoise.ar(0.1)*dBtoA.value(gainSlider.kr);
113
114         // Source mix
115         input = Mix.ar([input1 * in1Box.kr,input2 * in2Box.kr,
                input3 * in3Box.kr]);
116
117         // Oscillator for variable delay
118         v = SinOsc.kr(freqSlider.kr, 0.0, vdBox.kr * delaySlider
                .kr * ddSlider.kr * 0.01);
119
120         delsignal = TapN.ar(buffer,delaySlider.kr + v);
121         msignal = Mix.ar([input ,2*(fbBox.kr - 0.5) * delsignal
                * fbSlider.kr * 0.01]);
122         DelayWr.ar(buffer, msignal);
123
124         Mix.ar([mixSlider.kr*msignal,(1.0 - mixSlider.kr)*input
                ]);
125
126         },0.1);
127
```

```
128 },{ (filename ++ " not found.\n").post });
129
130 w.close;
```

Bibliografía

[1] Mathew Adkins. Acoustic chains in acousmatic music. *http://www.hud.ac.uk/schools/music/humanities/music/newmusic/acoustic_chains.html*, 1999.

[2] Balázs Bank, János Márkus, Attila Nagy, and László Sujbert. Signal and physics-based sound synthesis of musical instruments. *Periodica Polytechnica Ser. El. Eng.*, 47(3-4):269–295, 2004.

[3] Gerald Bennett. Thoughts on the oral culture of electro-acoustic music. *http://www.computermusic.ch/files/articles/ThoughtsontheOral.html*, 1995.

[4] Pierre Boeswillwald. Analysis in the electroacoustic world. In *Analysis in Electroacoustic Music. Proceedings Volume II of the works of the International Academy of Electroacoustic Music*, Bourges, 1996.

[5] Albert S. Bregman. *Auditory scene analysis*. MIT Press, 1990.

[6] Rodolfo Caesar. *The Composition of Electroacoustic Music*. PhD thesis, University of East Anglia, 1992.

[7] Leo Camilleri. *Electro-acoustic music: analysis and listening processes*. Retrieved March 31, 2004 from http://www.sonus-online.org/camilleri.htm. 1993.

[8] Nicolas Castagne and Claude Cadoz. Creating music by means of "physical thinking" : The musician oriented genesis enviroment. In *5th international Conference on Digital Audio Effects (Dafx-02)*, pages 169–174, Hamburg, Germany, 2002.

[9] Perry R. Cook. *Real sound synthesis for interactive applications*. A K Peters, Natick Mass, 2002.

[10] François Delalande. Music analysis and reception behaviors: Sommeil by pierre henry. *Journal of New Music Research*, 27:13–66, 1998.

[11] Charles Dodge and Thomas A. Jerse. *Computer music : synthesis, composition, and performance*. Schirmer Books ; Prentice Hall International, New York London, 2nd edition, 1997.

[12] Sophie Donnadieu. *Mental Representation of the Timbre of Complex Sounds*, chapter 8. Springer, New York, 2007.

[13] Francesco Giomi and Marco Ligabue. Understanding electroacoustic music: analysis of narrative strategies in six early compositions. *Organised Sound*, 3(1):45–49, 1998.

[14] Mara Helmut. Multidimensional representation of electroacoustic music. *Journal of New Music Research*, 25:77–103, 1996.

[15] Pablo Irarrázaval. *Análisis de señales*. McGraw-Hill Interamericana, Santiago (Chile), 1999. Pablo Irarrazaval M. il. ; 23 cm. 20010913.

[16] David A. Jaffe. Ten criteria for evaluating synthesis techniques. *Computer Music Journal*, 19(1):76–87, 1995.

[17] Mari Riess Jones and William Yee. *Attending to auditory events: the role of temporal organization*, chapter 4, pages 69–112. Clarendon Press, New York, 1993.

[18] Francisco Kröpfl. An approach to the analysis of electroacoustic music. In *Analysis in Electroacoustic Music. Proceedings Volume II of the works of the International Academy of Electroacoustic Music*, Bourges, 1996.

[19] F. Richard Moore. *Elements of computer music*. Prentice Hall, Englewood Cliffs, N.J., 1990.

[20] Jay W. Morthenson. The concept of meaning in electronic music. In *Proceedings of the ICEM Conference on Electro-acoustic Music*, Stockholm, 1985.

[21] Michael Norris. The status of analysis in an electroacoustic context. *http://farben.latrobe.edu.au/mikropol/volume5/norris_m/norris_m.html*, 1999.

[22] Giovanni De Poli. A tutorial on digital sound synthesis techniques. *Computer Music Journal*, 7(4), 1983.

[23] Pietro Polotti and Gianpaolo Evangelista. Fractal additive synthesis via harmonic-band wavelets. *Computer Music Journal*, 25(3):22–37, 2001.

[24] Pietro Polotti, Fritz Menzer, and Gianpaolo Evangelista. Inharmonic sound spectral modeling by means of fractal additive synthesis. In *Proceedings of the 5th Conference on Digital Audio Effects*, Hamburg, Germany, 2002.

[25] Stephen T. Pope. Why is good electroacoustics music so good? why is bad electroacoustic music so bad? *Computer Music Journal*, 18(3), 1994.

[26] Curtis Roads. *Musical signal processing*. Studies on new music research 2. Swets & Zeitlinger, Lisse Netherlands ; Exton PA, 1997.

[27] Curtis Roads. *The computer music tutorial*. MIT Press, Cambridge, Mass., 4th edition, 1999.

[28] Curtis Roads. *Microsound*. MIT Press, Cambridge, Mass., 2001.

[29] Axel Röbel. Neural network modeling of speech and music signals. In *Neural Network Information Processing Systems 9, NIPS 96*, Denver, 1996.

[30] Axel Röbel. Synthesizing natural sounds using dynamic models of sound attractors. *Computer Music Journal*, 25(2):46–61, 2001.

[31] Nicola Sani. Thoughts on analysis of electroacoustic music. In *Analysis in Electroacoustic Music. Proceedings Volume II of the works of the International Academy of Electroacoustic Music*, Bourges, 1996.

[32] Xavier Serra. Current perspectives in the digital synthesis of musical sound. *Formats*, 1, 1997.

[33] Denis Smalley. *Spectro-morphology and structural processes*, pages 61–93. Harwood Academic, 1986.

[34] Denis Smalley. *The Listening Imagination: Listening in the Electroacoustic Era*, chapter 26, pages 514–554. Routeledge, London, 1992.

[35] Denis Smalley. Spectromorphology: explaining sound-shapes. *Organised Sound*, 2(2):107–126, 1997.

[36] Julius O. Smith. Viewpoints on the history of digital synthesis. In *Proceedings of the 1991 International Computer Music Conference*, San Francisco, CA, 1991.

[37] Julius O. Smith. Physical modeling synthesis update. *The Computer Music Journal*, 20(2):44–56, 1996.

[38] John Strawn and James F. McGill. *Digital audio engineering : an anthology*. Computer music and digital audio series; [v. 3]. W. Kaufmann, Los Altos, Calif., 1985.

[39] Bob L. Sturm. Sonification of particle systems via de broglie's hypothesis. In *Proc. of the Int. Community for Auditory Display Conf*, Atlanta, GA, USA, April 2000.

[40] Bob L. Sturm. Composing for an ensemble of atoms: the methamorphosis of scientific experiment into music. *Organised Sound*, 6(2):131–145, 2001.

[41] Bob L. Sturm. Synthesis and algorithmic composition techniques derived from particle physics. In *Proc. of the Eighth Biennial Arts and Tech. Symposium*, New London, CT, USA, March 2001.

[42] Tero Tolonen, Vesa Välimäki, and Matti Karjalainen. Evaluation of modern sound synthesis methods. *Report No. 48, Helsinki University of Technology, Department of Electrical and Communications Engineering*, 1998.

[43] Barry Truax. *Acoustic communication*. Ablex Pub., 2001.

[44] W. Luke Windsor. *A Perceptual Approach to the Description and Analysis of Acousmatic Music*. PhD thesis, City University, London, 1995.

[45] W. Luke Windsor. *Perception and Signification in Electroacoustic Music*, chapter 7. Edinburgh University Faculty of Music, 1996.

[46] W. Luke Windsor. *Through and around the acousmatic: the interpretation of electroacoustic sounds*, chapter 1. Ashgate, London, 2000.

[47] Trevor Wishart. *Sound symbols and landscapes.*, chapter 3. Harwood Academic, New York, 1986.