INFORME FINAL
PRACTICA EMPRESARIAL EN TYT LTDA

GERSON JOHAN SAMANIEGO RODRÍGUEZ
2001219

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
BUCARAMANGA
2005

INFORME FINAL
PRACTICA EMPRESARIAL EN TYT LTDA

TECNOLOGIAS DE INFORMACION Y TELECOMUNICACIONES

GERSON JOHAN SAMANIEGO RODRIGUEZ
2001219

Investigación Descriptiva en:
Digitalización, codificación y control de archivos de sonido, especialmente sonidos de voz procedentes de conversaciones telefónicas

Director:
OSCAR GONZALEZ PIMENTEL
Profesional Senior de Sistemas

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
BUCARAMANGA
2005

AGRADECIMIENTOS

Mucha gente ayudo a que mi trabajo de grado se pudiera cumplir. Resulta inevitable un agradecimiento especial para las siguientes personas:

A Oscar González, mi director de proyecto, quien me ayudó durante los 6 meses de trabajo en todo lo relacionado con el proyecto. Fue el Gestor del tema de investigación y el guía continuo que me hacía volver al camino correcto cada vez que mis ideas equivocas me desviaban del objetivo real.

A Sergio Cajias, Socio de TYT Ltda. y amigo. Quien me abrió las puertas en su empresa para dejarme poner en práctica los conocimientos adquiridos durante mi carrera en un ambiente laboral que me ha dejado muchos frutos. Por su interés en el bienestar de quienes lo rodean y por contagiarme de su espíritu de emprendedor y trabajador.

A mis compañeros de práctica Julio Cesar Ruiz y Marino Pérez. A Julio, por que gracias a su contacto se pudo llevar a cabo mi proyecto de grado en la modalidad de práctica empresarial. Y a ambos, por el apoyo, compañía y amistad durante todo el trayecto de la práctica.

A TYT Ltda, sus socios y empleados, por el espacio, el tiempo y los recursos prestados durante la práctica.

A Mi Familia. Padres, Abuelos y demás, que han sido un apoyo incondicional en todo momento de mi vida.

RESUMEN

Titulo: [*] Práctica empresarial en TYT Ltda. (Tecnologías de Información y Comunicaciones)

Autor: GERSON JOHAN SAMANIEGO RODRÍGUEZ[**]

Palabras claves: Sonido**,** Señales de voz, Fonética, Digitalización, Formatos de archivos de sonido, Formato Wav, Formato MP3, MPEG Layer III, Codificación de voz, sonidos en el PC con Visual Basic 6.0.

Descripción: Éste Informe Final de Práctica Empresarial reúne las bases teóricas y prácticas de las actividades desarrolladas durante el periodo. Con el objetivo general en mente: tener conocimiento y control total sobre el sonido en un PC con sistema operativo Microsoft Windows, y profundizando un poco más sobre sonidos de voz, se documentan los temas desde sus bases hasta lo relacionado con el objetivo. El sonido en su forma física es una onda mecánica de presión que describe un movimiento ondulatorio y es percibida por el ser humano mediante el oído. Por describir un movimiento ondulatorio posee características como Amplitud (Intensidad), Frecuencia (Tono), y Superposición (Timbre), y se puede convertir su señal en información digital para almacenarla en el computador. Para poder convertir la señal analógica a digital se llevan a cabo los procesos de muestreo, cuantización y codificación en binario. El resultado es un archivo con datos en formato PCM (Modulación por Impulsos Codificados) que Windows almacena como un Archivo con Formato de Intercambio de Recursos (RIFF) y extensión Wav. Dicho archivo, inicialmente ocupa demasiado espacio en el disco, por lo que es necesario recurrir a procedimientos de codificación para reducir su tamaño (Compresión). Aprovechando las limitaciones de escucha del ser humano y las propiedades cíclicas de la onda de sonido es posible eliminar algunos de los datos muestreados y codificar muchos otros a expresiones más reducidas. Los procedimientos utilizados requieren de un alto conocimiento en matemáticas en cuanto a Transformadas de Fourier (Espectros de Frecuencia y Filtros) y Álgebra Lineal (Vectores), pero por suerte, grupos de científicos se han encargado de implementar algoritmos que hoy en día se han convertido en estándares y están disponibles para cualquier persona. Ejemplo de ellos, el MP3 (estándar ISO) del grupo MPEG, el GSM, de la Unión Europea de Telecomunicaciones, el CS-ACELP, de la Unión Internacional de Telecomunicaciones, entre otros. Los archivos de sonido que contienen solo voz, se pueden comprimir aún más que los normales. Esto debido a características propias del Aparato Fonador Humano que permiten modelarlo matemáticamente (Filtros Lineales). Dichas características son explotadas por algoritmos de codificación usados en la transmisión de voz por los diferentes canales de telecomunicaciones, ya sean inalámbricos, por redes de telefonía o redes IP (Internet).

[*] Proyecto de grado en la modalidad de práctica empresarial. Subtema: Digitalización, codificación y control de archivos de sonido, especialmente sonidos de voz procedentes de conversaciones telefónicas

[**] Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas e Informática. Director: Ing. Oscar González Pimentel

ABSTRACT

Title: * Enterprise practice in TYT Ltda. (Technologies of Information and Communications)

Author: GERSON JOHAN SAMANIEGO RODRÍGUEZ**

Key words: Sound, voice signals, Phonetics, Digitalization, sound file formats, Wav Format, MP3 Format, MPEG Layer III, voice codification, sounds in the PC with Visual BASIC 6.0.

Description: This Closing Report of Enterprise Practice compact the theoretical and practical bases of activities developed during the period. With the general objective in mind: to have knowledge and total control on the sound in a PC with operating system Microsoft Windows, and deepening more on voice sounds, the subjects are documented from their bases to the related thing to the objective. The sound in its physical form is a pressure mechanical wave that describes an undulatory movement and is perceived by the human through ear. To describe an undulatory movement it has characteristics like Amplitude (Intensity), Frequency (Tone), and Superposition (Timbre), and it can turn his signal digital information to store it in the computer. In order to be able convert a analogical signal to digital the sampling processes, cuantización and codification, are carried out. The result is a file with data in format PCM (Pulse Code Modulation) that Windows stores like a Resources Interchange File Format (RIFF) and .Wav extension. This file, initially occupies too much space in the disc, reason why it is necessary resort to codification procedures to reduce his size (Compression). Taking advantage of the limitations listening of the human and the cyclical properties of the sound wave it is possible to eliminate some samples data and to codify many others to reduced more the expressions. The used procedures require of a high knowledge in mathematics as Transformed of Fourier (Espectrums of Frequency and Filters) and Linear Algebra (Vectors), but luckily, groups of scientists have been in charge to implement algorithms that nowadays have become standards and are available for any person. Example of them, the MP3 (standard ISO) of MPEG group, the GSM, of Telecommunications European Union, the CS-ACELP, of Telecommunications International Union, among others. The sound archives that contain single voice, can be compressed still more that any other. This due to own characteristics of Human Apparatus Fonador that allow to model it mathematically (Linear Filters). These characteristics are operated by algorithms of codification used in the transmission of voice by the different channels from telecommunications, wireless, by networks of telephony or networks IP (Internet).

* Project of degree in the modality of enterprise practice. Digitalization, codification and control of sound archives, specially voice sounds coming from telephone conversations

** Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas e Informática. Director: Ing. Oscar González Pimentel

CONTENIDO

LISTA DE TABLAS

TABLA DE FIGURAS

INTRODUCCION

La empresa TYT Ltda. trabaja en un proyecto software para Centros de llamadas (Call Center) patrocinado por conciencias. Un Call Center (CC) es un centro especializado que une la telefonía tradicional y los diferentes medios de comunicación con la tecnología de los sistemas de información, permitiendo manejar grandes volúmenes de llamadas tanto de entrada como de salida, con personal especializado, software e información oportuna, administrando eficientemente la relación de las empresas con sus clientes.

En el proceso de operación de un CC se requiere la grabación de las conversaciones con los clientes, ya sea con los agentes (personas que contestan las llamadas personalizadas) o con un IVR (Contestador Interactivo de Voz). Dada la cantidad y el tamaño de los archivos de sonido grabados, se necesitan procesos de *operación*: Grabación, Procesamiento digital, codificación, compresión y reproducción; y *administración*: Asociación de etiquetas de control; que permitan su adecuado *manejo*: reducción de tamaños en disco, mejoras en la calidad, flexibilidad y rapidez en las búsquedas.

El proyecto software de TYT Ltda. plantea en sus fases iniciales un periodo de investigación previa sobre las bases tecnológicas que necesita el proyecto. Dicha investigación es de tipo descriptiva, es decir, comprende la descripción, registro, análisis e interpretación de los elementos actualmente existentes y su composición y procesos. No es investigación como tal, es el análisis de la información recolectada para su aplicación en al campo de interés. Basados en ésta definición de Mario Tamayo en su libro "*El proceso de la investigación científica"* se puede ver el objetivo final de la práctica empresarial como la entrega a TYT de una Monografía que reúna la base teórica en cuanto a lo mencionado en el párrafo anterior. En esto consiste la **primera parte** de éste informe final.

La investigación descriptiva también supone pruebas de validez, por lo cual dentro de los objetivos del proyecto se plantea la elaboración de pequeños programas a medida que se avanza en el tema. Al final de la práctica se propone integrar esos programas en una solución tipo componente que muestre el alcance de la investigación. La **segunda parte** de éste informe final describe los programas resultado más importantes junto con el componente.

PARTE I

MARCO TEORICO

1. NATURALEZA DEL SONIDO

"El término «sonido» tiene un doble sentido: por un lado se emplea en sentido subjetivo para designar la sensación que experimenta un observador cuando las terminaciones de su nervio auditivo reciben un estímulo, pero también se emplea en sentido objetivo para describir las ondas producidas por compresión del aire que pueden estimular el nervio auditivo de un observador" [3].

"El sonido es la vibración de un medio elástico, bien sea gaseoso, líquido o sólido. Cuando nos referimos al sonido audible por el oído humano, estamos hablando de la sensación detectada por nuestro oído, que producen las rápidas variaciones de presión en el aire por encima y por debajo de un valor estático. Este valor estático nos lo da la presión atmosférica el cual tiene unas variaciones pequeñas y de forma muy lenta, tal y como se puede comprobar en un barómetro" [1].

"La **acústica,** etimológicamente procede del griego ***akustike*** (Ciencia del sonido), es un término que define a la vez una ciencia y una tecnología. Como ***ciencia*** estudia las propiedades de las vibraciones de las partículas de un medio susceptible de engendrar sonidos, infrasonidos o ultrasonidos. Como ***tecnología*** se orienta a la producción, transmisión y el análisis de los efectos del sonido" [1].

1.1. MOVIMIENTO ONDULATORIO

Las ondas son el producto de las oscilaciones de un cuerpo que traza en su recorrido a través del tiempo una línea sinusoidal. Cuando un cuerpo vibra produce en el medio que le rodea una diferencia de presión que se propaga a las partículas contiguas y produce así una onda, también conocida como onda de presión. Las ondas electromagnéticas, son producidas por las altas velocidades y frecuencias de oscilación de electrones que forman campos eléctricos y magnéticos. La onda electromagnetica se acompaña de transmisión de energía sin transmisión de materia.

Elementos encontrados en la representación de las Ondas:

Velocidad de propagación: velocidad a la que se transmite la onda.

Intensidad o amplitud (A): distancia máxima que separa un punto de su posición de equilibrio.

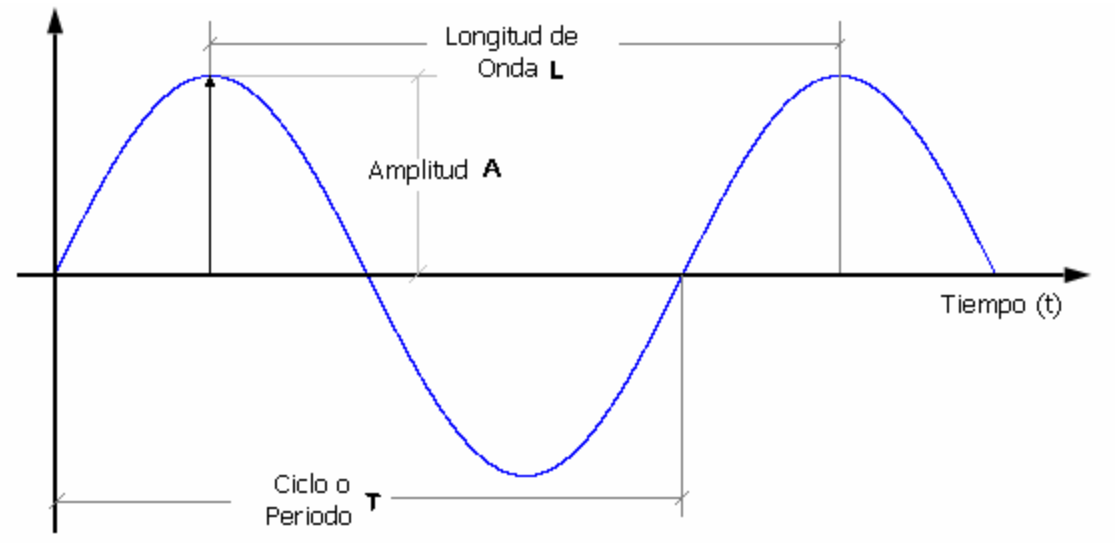
Duración de la oscilación: viene definida por el tiempo (T Periodo).

Longitud de onda (L): Es la distancia recorrida durante un periodo. También se denomina así la distancia entre dos regiones adyacentes.

Frecuencia (f): Es el número de ondas que pasan por un punto determinado en un tiempo. Es la inversa del periodo y se expresa en Hz.

Ciclo: Una oscilación completa y única se denomina ciclo.

Figura 1: representación gráfica de una onda.



1.2. LA ONDA SONORA

"Es la sucesión de compresiones y expansiones del medio en que se propaga" [1]. Es una onda de tipo longitudinal (Vibra en la misma dirección de propagación) y mecánica, y por lo tanto para propagarse necesita un medio elástico. Una perturbación producida en un punto del medio se va transmitiendo (se transmite la fuerza pero no la materia) a las partículas siguientes y de éstas alas más alejadas. La velocidad de propagación depende de diversos factores, entre otros de la densidad (a mayor densidad mayor velocidad de propagación) y la elasticidad del medio.

A continuación se describen las características físicas de Amplitud, Frecuencia y velocidad, y sus relaciones con las cualidades del sonido como Intensidad, tono y timbre.

1.2.1. Amplitud del sonido

"La primera propiedad que una onda de sonido ha de tener es la amplitud. Subjetivamente, la *intensidad* de un sonido corresponde a nuestra percepción del mismo como más o menos fuerte. Cuando elevamos el volumen de la cadena de música o del televisor, lo que hacemos es aumentar la intensidad del sonido. La amplitud es la distancia por encima y por debajo de la línea central de la onda de sonido. La línea central es la línea horizontal, llamada cero grados. La mayor distancia arriba y debajo de la línea central nos da el volumen del sonido. (Volumen es la palabra que se utiliza en los amplificadores de sonido) en el caso de estaciones o editores de audio digital, se llama amplitud" [3].

1.2.2. La Intensidad y su medida

Ya se mencionó que la intensidad del sonido esta relacionada con la amplitud de la onda, pero aparte de esta relación, "acústicamente es una magnitud que da idea de la cantidad de energía que está fluyendo por el medio como consecuencia de la propagación de la onda" [2].

"Se define como la energía que atraviesa por segundo una superficie unidad dispuesta perpendicularmente a la dirección de propagación. Equivale a una potencia por unidad de superficie y se expresa en W/m². La intensidad de una onda sonora es proporcional al cuadrado de su frecuencia y al cuadrado de su amplitud y disminuye con la distancia al foco. La magnitud de la sensación sonora depende de la intensidad acústica, pero también depende de la sensibilidad del oído" [2].

El nivel de intensidad (β):

"El intervalo de intensidades acústicas que va desde el *umbral de audibilidad,* o valor mínimo perceptible, hasta el *umbral del dolor,* es muy amplio, estando ambos valores límite en una relación del orden de 10 a la 14. Debido a la extensión de este intervalo de audibilidad, para expresar intensidades sonoras se emplea una escala cuyas divisiones son potencias de diez y cuya unidad de medida es el decibelio (dB)" [2].

"Ello significa que una intensidad acústica de 10 decibelios corresponde a una energía diez veces mayor que una intensidad de cero decibelios; una intensidad de 20 dB representa una energía 100 veces mayor que la que corresponde a 0 decibelios y así sucesivamente. Otro de los factores de los que depende la intensidad del sonido percibido es la frecuencia. Ello significa que para una frecuencia dada un aumento de intensidad acústica da lugar a un aumento del

nivel de sensación sonora, pero intensidades acústicas iguales a diferentes frecuencias pueden dar lugar a sensaciones distintas" [2].

Tabla 1: Algunas escalas predefinidas de decibeles [5]:

Sonido	dBs
Jet despegando	140
Concierto de rock	120
Martillo neumático a 50 pies	85
Ruido en la calle	75
Conversación en lugar público	60
Ambiente de oficina	45
Murmullo a 15 pies	30
Estudio de "TV" en silencio	20

"Matemáticamente el nivel de intensidad esta definido logarítmicamente:

$$\beta = 10 \cdot \log \frac{I}{I_0}$$

siendo I_0 una intensidad arbitraria de referencia que se toma igual a 10^{-16} W/cm^2, valor que corresponde al umbral de la sensación sonora" [3].

"La energía de las ondas sonoras (y por tanto la intensidad del sonido) caerán con el cuadrado de la distancia a la fuente sonora. En otras palabras, si el oyente se aleja 200 metros de la fuente de sonido, el nivel de sonido será un cuarto del que tenía a 100 metros. Y así, al multiplicar por dos su distancia hará que el nivel de dB se divida por 6" [4].

1.2.3. Frecuencia

"La segunda propiedad es la frecuencia. Se mide en Hercios (Hertz, Hz) y permite saber a cuantos ciclos por segundo va una onda. Un ciclo es cuando la onda sube hasta un punto máximo de amplitud, baja hasta atravesar la línea central y llega hasta el punto de amplitud máximo negativo y vuelve a subir hasta alcanzar la

línea central. El *tono* o *altura* (pitch en ingles) de un sonido depende de su frecuencia, es decir, del número de oscilaciones por segundo. El ciclo, que puede tener cualquier longitud, se conoce como longitud de onda y el número de veces que pasa en un segundo, se conoce como frecuencia de la onda" [3].

"Cuanto mayor sea la frecuencia, más agudo será el sonido. Cuantos más ciclos por segundo, más elevado será el tono. Así, la frecuencia hace el tono. La altura de un sonido corresponde a nuestra percepción del mismo como más grave o más agudo. Esto puede comprobarse, por ejemplo, comparando el sonido obtenido al acercar un trozo de cartulina a una sierra de disco: cuanto mayor sea la velocidad de rotación del disco más alto será el sonido producido" [3].

1.2.4. El Tono (*pitch*)

Se dijo en los párrafos anteriores que se puede reconocer si un sonido es agudo o grave dependiendo de su tono y que a la vez, éste estaba relacionado con la frecuencia. Los sonidos percibidos como graves corresponden a frecuencias bajas, mientras que los agudos son debidos a frecuencias altas. "Así el sonido más grave de una guitarra corresponde a una frecuencia de 82,4 Hz y el más agudo a 698,5 hertzs. Junto con la frecuencia, en la percepción sonora del tono intervienen otros factores de carácter psicológico. Así sucede por lo general que al elevar la intensidad se eleva el tono percibido para frecuencias altas y se baja para las frecuencias bajas. Entre frecuencias comprendidas entre 1000 y 3000 Hz el tono es relativamente independiente de la intensidad" [2].

1.2.5. La velocidad

"Esta es la propiedad más simple y precisa del sonido. La velocidad del sonido en un medio puede medirse con gran precisión. Se comprueba que dicha velocidad es independiente de la frecuencia y la intensidad del sonido, dependiendo únicamente de la densidad y la elasticidad del medio. Así, es mayor en los sólidos que en los líquidos y en éstos mayor que en los gases. En el aire, y en condiciones normales, es de 330,7 m/s" [3].

1.2.6. El timbre

En ésta característica se puede captar que la onda sonora no es tan sencilla como parece. Sucede que dos sonidos con la misma intensidad y la misma frecuencia pueden ser captados de diferente forma por el oído humano. Esto se debe a que algunos sonidos no son puros, es decir, están formados por superposiciones de varios sonidos con intensidades y frecuencias diferentes. O sea, que todo sonido musical es un sonido complejo que puede ser considerado como una

superposición de sonidos simples. De esos sonidos simples, el *sonido fundamental* de frecuencia *n* es el de mayor intensidad y va acompañado de otros sonidos de intensidad menor y de frecuencia 2*n*, 3*n*, 4*n*, etc.

“Los sonidos que acompañan al fundamental constituyen sus *armónicos* y de sus intensidades relativas depende el timbre. Sin embargo, muchos instrumentos, tales como el piano, el arpa, etc., no emiten un único sonido musical que quepa considerar como una superposición de sonidos simples armónicos, sino que emiten un sonido constituido por superposición de sonidos parciales” [3].

El timbre es entonces la propiedad del sonido que depende de la forma de la onda y nos permite diferenciar dos sonidos con la misma intensidad y tono pero con diferente fuente.

2. LA VOZ HUMANA

Cápitulo tomado de [6].

2.1. CONCEPTOS PRELIMINARES

2.1.1. Comunicación y lenguaje

Los sistemas de comunicación transportan información. El siguiente objetivo es estudiar un sistema de comunicación específico, el de la comunicación a través de señales de voz, es decir señales acústicas tradicionalmente emitidas y recibidas por seres humanos en forma oral. Algunos de los objetivos que se tienen en mente cuando se estudia la voz humana son: la representación, análisis, modificación, mejora de la relación señal/ruido, generación artificial de mensajes vocales inteligibles para el ser humano y el reconocimiento automático de mensajes vocales pronunciados por seres humanos.

En todo sistema de comunicación hay varios componentes: e*misor*, r*eceptor*, m*ensaje*, c*ódigo*, c*anal* y *contexto.* Es necesario conocer algunos aspectos de cada uno de ellos para poder integrar sistemas que funcionen de manera eficaz y eficiente. En nuestro caso el emisor es el conjunto integrado por el cerebro que "piensa" el mensaje y el aparato fonatorio que lo "traduce" a una emisión acústica. El receptor es el aparato auditivo que recibe la onda sonora y la transforma en impulsos nerviosos que luego son interpretados por el cerebro.

El mensaje es la idea a comunicar. El código es el lenguaje hablado. La combinación del mensaje y el código constituyen la *señal*. El canal puede ser el medio en el cual se propaga la onda sonora (en general el aire) o un medio de transmisión electrónico que constituye en sí mismo otro subsistema de comunicación cuyas propiedades son bien conocidas y que se aproxima en muchos casos (aunque no siempre) a la idealidad. El contexto puede tener un sinnúmero de componentes, que van desde factores puramente subjetivos o psicológicos, como el interés, la atención, la motivación hasta factores físicos tales como respuesta en frecuencia, interferencias, distorsiones, ruido, etc.

2.1.2. Algunos conceptos sobre lenguaje

La *lengua* es un sistema de signos lingüísticos que permiten la comunicación en una comunidad. Es un sistema pues cada uno de sus elementos tiene *entidad propia y entidad relativa* a su posición o relación con los otros elementos. Es un código de signos. Tiene carácter social, ya que es común a una sociedad.

El *habla* es el acto de seleccionar los signos de entre los disponibles y organizarlos a través de ciertas reglas. Materializa el código. Es individual, vale decir que cambia de un individuo a otro. Los signos pueden corresponder al lenguaje escrito o al oral.

El *lenguaje* es un *sistema* articulado ya que los sonidos y otros componentes se integran entre sí. Está formado por *signos lingüísticos*, nombre que recibe la señal en el lenguaje. El lenguaje tiene modalidades regionales llamadas *dialectos*.

Un *signo* es algo que reemplaza a otra cosa para comunicarla en un *mensaje*. Los signos lingüísticos se clasifican en dos tipos: *significado* y *significante*. El *significado* es el concepto mental, idea o contenido a comunicar. El *significante* es la *imagen*, ya sea *gráfica* o *acústica* que se le asigna.

En el lenguaje escrito, el significante es la *grafía escrita*, formada por combinaciones de *letras*, en tanto que en el lenguaje hablado es su *realización acústica* mediante la *palabra hablada*. Las *palabras* son los elementos *libres* mínimos del lenguaje. La *sintaxis* es el conjunto de reglas para la coordinación de las palabras en frases u oraciones. En su versión escrita las palabras están formadas por letras o *grafemas*, es decir unidades gráficas mínimas, y, en el caso oral, por *fonemas*. Los *fonemas* son la unidad fónica ideal mínima del lenguaje. Se materializan a través de los sonidos, pero de una manera no unívoca.

Los *monemas* son unidades mínimas con significado, que puede ser *gramatical*, dando origen a los *morfemas*, o *léxico*, representado por los *lexemas*. Los morfemas tienen relación con la gramática, o la forma de organizar o dar estructura a las categorías básicas del lenguaje (género, número, tiempo o persona de los verbos, etc.), mientras que los lexemas se refieren a significados externos al lenguaje mismo. Las palabras constan de al menos un monema, siendo las más comunes *bimonemáticas*, que incluyen un lexema y un morfema. En la tabla siguiente se dan dos ejemplos en los que se identifican los componentes de la palabra:

Tabla 2: Ejemplos de las partes que componen las palabras.

Palabra	Monemas		Grafema	Fonemas
	Lexema	Morfema		
Gato	Gat	o	G, a, t, o	/g/, /a/, /t/, /o/
Amaban	Ama	ban	A, m, a, b, a, n	/a/, /m/, /a/, /b/, /a/, /n/

2.1.3. Fonología y fonética

La *Fonología* estudia los *fonemas*, es decir el modelo fónico convencional e ideal del lenguaje. La *Fonética*, en tanto, se refiere a los *sonidos* en el habla, incluyendo su producción acústica y los procesos físicos y fisiológicos de emisión y articulación involucrados. Así, la Fonología es el estudio de los sonidos de la lengua en cuanto a su carácter simbólico o de representación mental. Procede detectando regularidades o recurrencias en los sonidos del lenguaje hablado y sus combinaciones, y haciendo abstracción de las pequeñas diferencias debidas a la individualidad de cada hablante y de características *suprasegmentales* como la entonación, el acento (*tónico*, es decir por aumento de la intensidad y *agógico*, por aumento de la duración), etc. Cada uno de los sonidos abstractos así identificados es un *fonema*. Uno de los objetivos de la fonología es acotar al máximo la cantidad de fonemas requeridos para representar cada idioma de una manera suficientemente precisa.

La Fonética estudia experimentalmente los mecanismos de producción y percepción de los sonidos utilizados en el habla a través del análisis acústico, articulatorio y perceptivo. Se ocupa, por consiguiente, de las realizaciones de los fonemas.

2.2. FONETICA

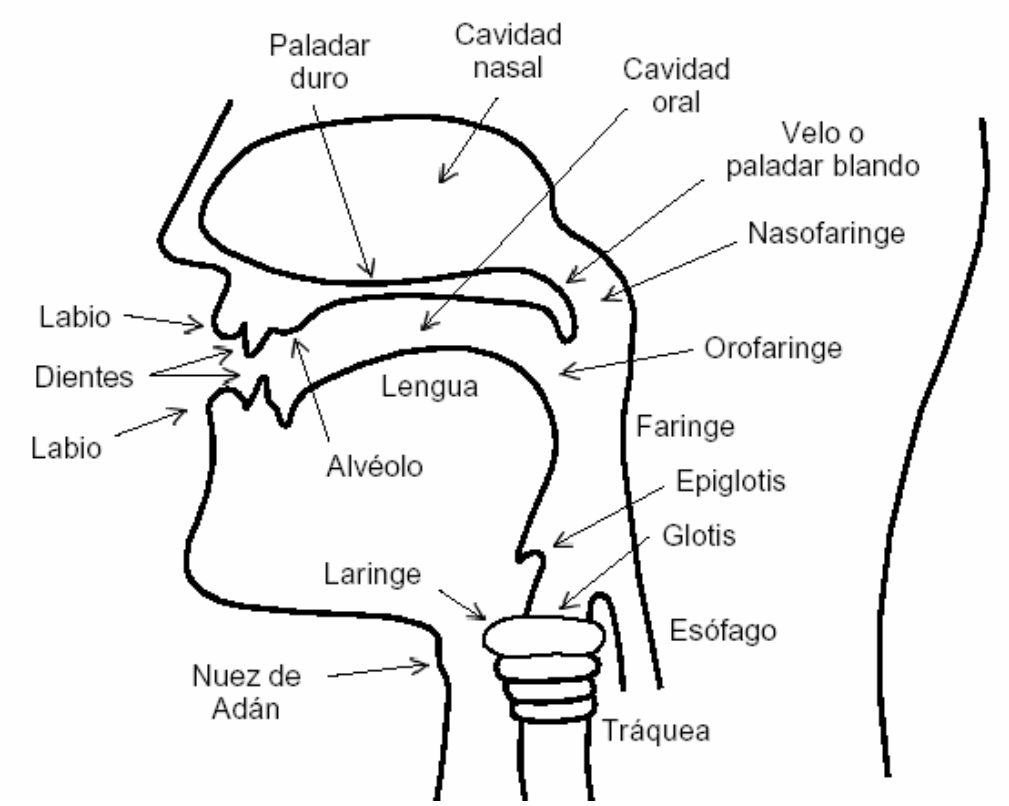
2.2.1. Breve anatomía del aparato fonatorio

La voz humana se produce voluntariamente por medio del aparato fonatorio. Éste está formado por los pulmones como fuente de energía en la forma de un flujo de aire, la *laringe*, que contiene las *cuerdas vocales*, la *faringe*, las *cavidades oral* (o bucal) y *nasal* y una serie de elementos articulatorios: los *labios*, los *dientes*, el *alvéolo*, el *paladar*, el *velo del paladar* y la *lengua.*

Las *cuerdas vocales* son, en realidad, dos membranas dentro de la laringe orientadas de adelante hacia atrás. Por adelante se unen en el *cartílago tiroides* (que puede palparse sobre el cuello, inmediatamente por debajo de la unión con la cabeza; en los varones suele apreciarse como una protuberancia conocida como *nuez de Adán*). Por detrás, cada una está sujeta a uno de los dos *cartílagos aritenoides*, los cuales pueden separarse voluntariamente por medio de músculos. La abertura entre ambas cuerdas se denomina *glotis*.

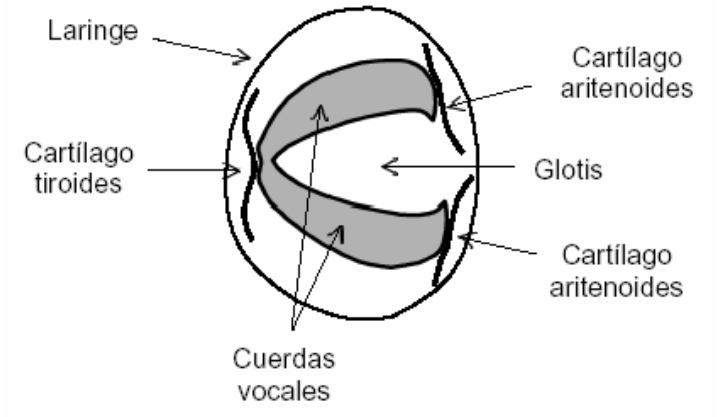
Cuando las cuerdas vocales se encuentran separadas, la glotis adopta una forma triangular. El aire pasa libremente y prácticamente no se produce sonido. Es el caso de la respiración.

Figura 2: Aparato Fonatorio



Cuando la glotis comienza a cerrarse, el aire que la atraviesa proveniente de los pulmones experimenta una turbulencia, emitiéndose un ruido de origen aerodinámico conocido como *aspiración* (aunque en realidad acompaña a una espiración o exhalación).

Figura 3: Cuerdas Vocales



Esto sucede en los sonidos denominados “aspirados” (como la h inglesa). Al cerrarse más, las cuerdas vocales comienzan a vibrar a modo de lengüetas, produciéndose un sonido tonal, es decir periódico. La frecuencia de este sonido depende de varios factores, entre otros del tamaño y la masa de las cuerdas vocales, de la tensión que se les aplique y de la velocidad del flujo del aire proveniente de los pulmones. A mayor tamaño, menor frecuencia de vibración, lo cual explica por qué en los varones, cuya glotis es en promedio mayor que la de las mujeres, la voz es en general más grave. A mayor tensión la frecuencia aumenta, siendo los sonidos más agudos. Así, para lograr emitir sonidos en el registro extremo de la voz es necesario un mayor esfuerzo vocal.

También aumenta la frecuencia (a igualdad de las otras condiciones) al crecer la velocidad del flujo de aire, razón por la cual al aumentar la intensidad de emisión se tiende a elevar espontáneamente el tono de voz. Finalmente, es posible obturar la glotis completamente. En ese caso no se produce sonido. Sobre la glotis se encuentra la *epiglotis*, un cartílago en la faringe que permite tapar la glotis durante la deglución para evitar que el alimento ingerido se introduzca en el tracto respiratorio. Durante la respiración y la fonación (emisión de sonido) la epiglotis está separada de la glotis permitiendo la circulación del flujo de aire. Durante la deglución, en cambio, la laringe ejecuta un movimiento ascendente de modo que la glotis apoya sobre la epiglotis.

La porción que incluye las cavidades faríngea, oral y nasal junto con los elementos articulatorios se denomina genéricamente *cavidad supraglótica*, en tanto que los espacios por debajo de la laringe, es decir la tráquea, los bronquios y los pulmones, se denominan *cavidades infraglóticas*.

Varios de los elementos de la cavidad supraglótica se controlan a voluntad, permitiendo modificar dentro de márgenes muy amplios los sonidos producidos por las cuerdas vocales o agregar partes distintivas a los mismos, e inclusive producir sonidos propios. Todo esto se efectúa por dos mecanismos principales: el *filtrado* y la *articulación*.

El *filtrado* actúa modificando el espectro del sonido. Tiene lugar en las cuatro cavidades supraglóticas principales: la faringe, la cavidad nasal, la cavidad oral y la cavidad labial. Las mismas constituyen resonadores acústicos que enfatizan determinadas bandas frecuenciales del espectro generado por las cuerdas vocales, conduciendo al concepto de *formantes*, es decir una serie de picos de resonancia ubicados en frecuencias o bandas de frecuencia que, son bastante específicas para cada tipo de sonido.

La *articulación* es una modificación principalmente a nivel temporal de los sonidos, y está directamente relacionada con la emisión de los mismos y con los fenómenos transitorios que los acompañan. Está caracterizada por el lugar del

tracto vocal en que tiene lugar, por los elementos que intervienen y por el modo en que se produce, factores que dan origen a una clasificación fonética de los sonidos que se verá más adelante.

2.2.2. El alfabeto fonético internacional

El castellano es un idioma cuya escritura es eminentemente fonética, ya que salvo pocos casos, hay correspondencia entre grafema y fonema. No todos los idiomas tienen esta característica. El inglés es un caso quizás extremo, a tal punto que se han creado posibles ortografías alternativas para algunas palabras basándose en la forma en que sus fonemas aparecen escritos en otras palabras.

Se ha compilado un extenso conjunto de símbolos fonéticos conocido como el *Alfabeto Fonético Internacional* (International Phonetic Alphabet, IPA) que contiene una gran cantidad de fonemas de los diversos idiomas, y que permite representar de una manera inequívoca los fonemas independientemente del idioma. El subconjunto correspondiente al idioma castellano se indica en la siguiente tabla:

Tabla 3: Subconjunto IPA para el idioma castellano

Fonemas castellanos					
Sonido	Ejemplo	Sonido	Ejemplo	Sonido	Ejemplo
[p]	**p**aso	[θ]	**z**orz**z**al, lápi**z**	[ɲ]	ma**ñ**ana, **ñ**o**ñ**o
[b]	**b**ase, **v**ena	[s]	**s**olo, co**s**a	[d3]	**y**o, **Y**apey**ú**
[ß]	la**b**or, la**v**ar	[x]	**g**iro, **j**arabe	[j]	**i**en, **i**ólogo
[t]	**t**res, can**t**o	[tʃ]	he**ch**o, **Ch**ubut	[w]	h**u**eso, b**u**itre
[d]	**d**ama, an**d**ar	[r]	ar**r**e**r**, ja**r**abe		
[ð]	ce**d**ro, verda**d**	[rr]	pe**rr**o, **r**ojo	[a]	**a**m**a**
[k]	**c**aso, dis**c**o	[l]	**l**oa**bl**e, fie**l**	[e]	**e**spe**r**a, v**e**r
[g]	**g**ula, **g**oma	[λ]	**ll**anto, ca**ll**e	[i]	v**i**ne, **i**ris
[γ]	a**g**ua, ne**g**ro	[m]	**m**a**m**á, á**m**bar	[o]	l**o**r**o**, p**o**s
[f]	**f**ino, ti**f**ón	[n]	**n**e**n**e, jove**n**	[u]	b**u**rla, h**u**racán

3. DIGITALIZACIÓN DEL SONIDO

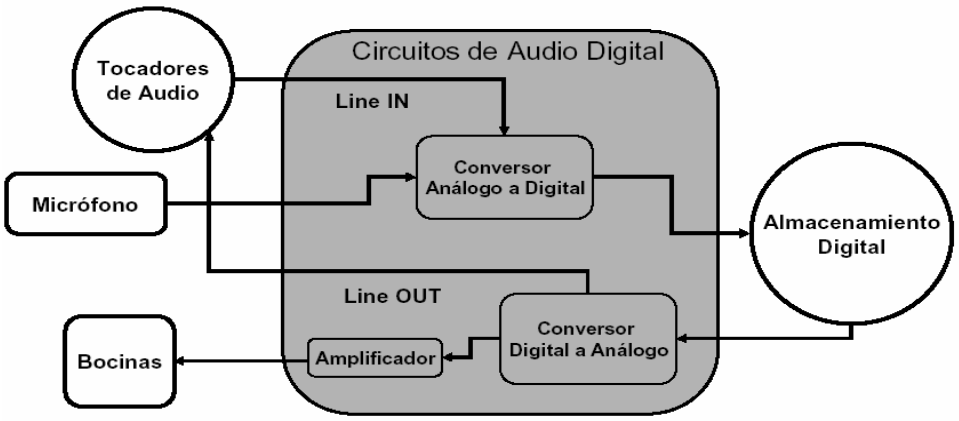
Anterior al sonido digital, el sonido se almacenaba en forma analógica grabando mediante medios magnéticos las perturbaciones del medio sonoro en cintas. El método consistía en convertir la onda sonora con el uso de micrófonos en ondas eléctricas cuyas variaciones de voltaje a lo largo del tiempo son equivalentes a las variaciones de amplitud o intensidad de la onda sonora. Estas ondas se almacenaban en cintas magnéticas. Uno de los más grandes problemas de la grabación analógica era o es el ruido incorporado a los datos almacenados.

"La grabación analógica esta sujeta a un montón de anomalías de sonido tales como, interferencias de cruce de señal entre las pistas, fluctuación, distorsión, degeneración de la calidad de sonido con el repicado y efecto de copia por cercanía" [8].

La prueba de lo fastidioso que es el problema del ruido en cintas magnéticas la encontramos al oír aquellos viejos casetes, y encontrar en ellos el típico zumbido antes de empezar las canciones y aun menos notable cuando éstas ya están sonando. Aún así, si se emplean técnicas y equipos de alta reducción de ruido, la calidad del sonido analógico puede ser muy buena. Pero sin necesidad de recurrir a esas técnicas y equipos que incrementan los costos existen ya desde finales de los años 60 métodos de grabación digitales que mejoran la calidad del sonido y reducen los costos.

"Por **AUDIO DIGITAL** entendemos la grabación (digitalización) de sonido real, ya sea procedente de voces, instrumentos musicales acústicos o electrónicos, grabaciones, etc, en el computador, en forma de ficheros informáticos" [7].

Figura 4: proceso de la digitalización

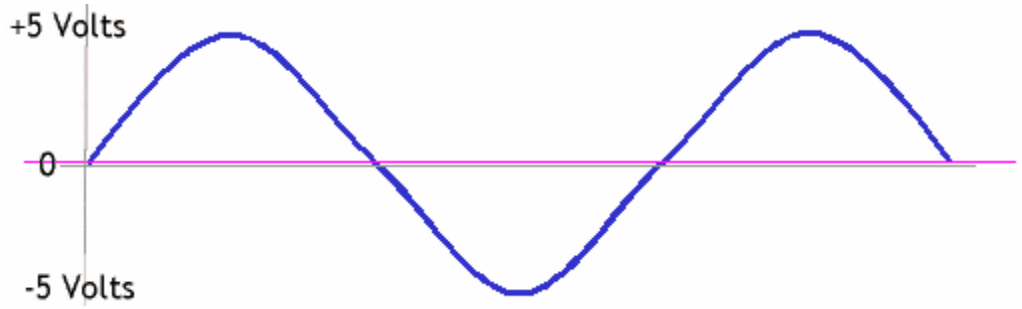


Una abreviatura a tener en cuenta es PCM (Pulse Code Modulation), que hace referencia a la toma de muestras (pulsos) de la señal analógica para almacenarlos en forma digital y que además, se ha convertido en un estándar en la digitalización del sonido.

3.1. SEÑALES ANALOGICAS

"El término **ANALÓGICO** en la industria de las telecomunicaciones y el cómputo significa todo aquel proceso entrada/salida cuyos valores son *continuos*. Algo continuo es todo aquello de puede tomar una infinidad de valores dentro de un cierto limite, superior e inferior" [9].

Figura 5: Ejemplo de señal analógica [9].



3.2. SEÑALES DIGITALES

"El término DIGITAL de la misma manera involucra valores de entrada/salida *discretos*. Algo discreto es algo que puede tomar valores fijos. En el caso de las comunicaciones digitales y el cómputo, esos valores son el CERO (0) o el UNO (1) o Bits (BInary DigiTs)" [9].

Figura 6: Ejemplo de una señal digital [9]

En el ejemplo anterior, la señal solo puede tomar valores de 0 y 1, pero las señales digitales pueden tomar valores enteros infinitos que a su vez se guardan en forma de ceros y unos.

3.3. COMVERSIÓN ANALOGO / DIGITAL (A/D)

El proceso de conversión análogo digital se lleva a cabo en cuatro etapas: filtro antialiasing, muestreo, cuantificación y codificación.

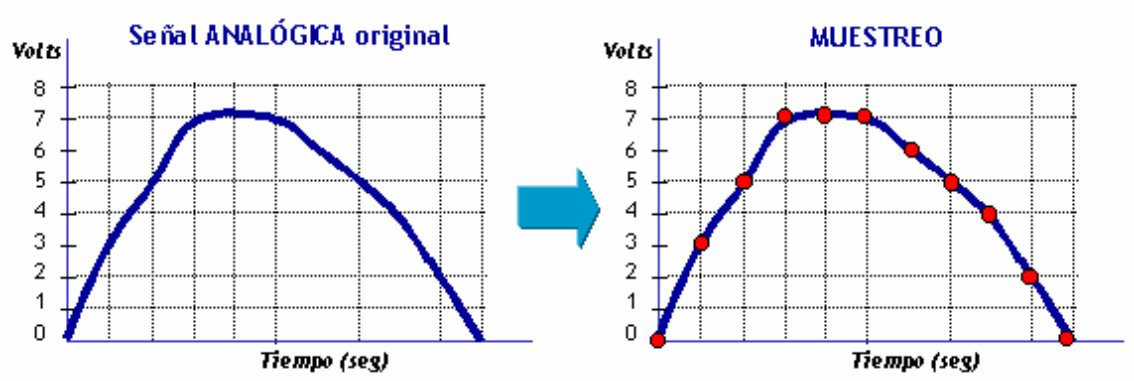
3.3.1. Filtro Antialiasing

"Antes de realizarse el muestreo de la señal, se realiza un proceso que se conoce como antialiasing. Este proceso consiste en hacer pasar la señal de audio analógica por un filtro pasabajos (que deja pasar las frecuencias situadas por debajo de una frecuencia previamente determinada y elimina las que están por encima de la misma) para eliminar de la señal aquellas frecuencias que por muy elevadas se encuentran más allá de las necesidades y de los medios del proceso digital. Aunque estas frecuencias por lo general están por encima del intervalo audible humano (de 20 Hz a 20.000 Hz), si no se eliminan por filtrado antes de la etapa de muestreo, serán consideradas como pertenecientes a dicho intervalo durante la grabación y la reproducción del sonido, con los errores que ello conlleva" [8].

3.3.2. Muestreo

Toda la tecnología digital (eje, audio, video) está basada en la técnica de muestreo (sampling en inglés). En música, cuando una grabadora digital toma una muestra, básicamente toma una *fotografía fija* de la *forma de onda* y la convierte en bits, los cuales pueden ser almacenados y procesados. Comparado con la grabación analógica, la cual está basada en registros de voltaje como patrones de magnetización en las partículas de óxido de la cinta magnética. El muestreo digital convierte el voltaje en números (0s y 1s) los cuales pueden ser fácilmente representados y vueltos nuevamente a su forma original. "Es tomar muestras (voltajes) de la señal analógica un número determinado de veces por unidad de tiempo" [8].

Figura 7: Muestreo de una señal analógica [9].



"La **Frecuencia de muestreo** se refiere al número de mediciones que se realizan por segundo. Cuanto mayor sea esta frecuencia, más parecido será el resultado obtenido al sonido original. Según el teorema de Nyquist, la frecuencia mínima de muestreo debe ser el doble del ancho de banda de la señal original. En términos más sencillos: Si el sonido original llega, en la zona de los agudos, supongamos, 10.000 hertzios, debemos muestrear a 20.000 hertzios" [7].

"Dado que el oído humano es capaz de escuchar sonidos en el rango de 20 a 20.000 Hertzios, aproximadamente, se ha elegido como frecuencia de muestreo más adecuada, la de 44,1 KHz, es decir, aproximadamente el doble de la frecuencia más aguda que se puede escuchar" [7].

"Si deseamos digitalizar sonidos acústicos, no es necesario alcanzar esas frecuencias de muestreo, ya que ni las voces ni los instrumentos acústicos producen frecuencias relevantes por encima de los 10 Khz, aproximadamente (de hecho, un sistema de reducción de ruido de Phillips, y la codificación usada en los DCC - Digital Compact Cassette, se basa en esta realidad, por lo que corta todas aquellas frecuencias por encima de dicho límite). Así pues, en estos casos se puede utilizar una frecuencia de 32 Khz, o incluso de 22 Khz" [7].

Si reducimos la frecuencia de muestreo, se puede apreciar que el sonido es menos nítido, más apagado, porque perdemos frecuencias agudas. Por conveniencia, hoy en día, las frecuencias de muestreo más usadas según [9] son:

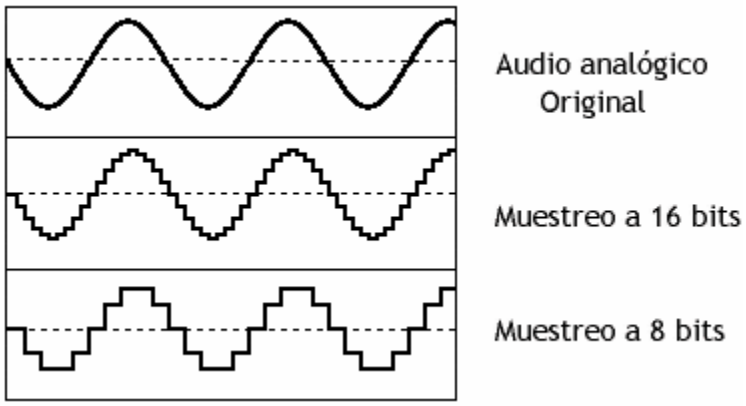
24,000 = 24 kHz - 24,000 mps. Una muestra cada 1/24,000 de segundo.
30,000 = 30 kHz - 30,000 mps. Una muestra cada 1/30,000 de segundo.
44,100 = 44.1 kHz - 44,100 mps. Una muestra cada 1/44,000 de segundo. (CD)
48,000 = 48 kHz - 48,000 mps. Una muestra cada 1/48,000 de segundo.

3.3.3. Cuantificación o Cuantización

“Mientras que el muestreo representa el tiempo de captura de una señal, la cuantificación o cuantización es el componente amplitud del muestreo. En otras palabras, mientras que el muestreo mide el tiempo (por instancia 44,100 muestras por segundo), la cuantización es la técnica donde un evento analógico es medido asignándosele un valor numérico” [9].

”Para hacer esto, la amplitud de la señal de audio es representada en una serie de pasos discretos. Cada paso está dado entonces por un número en código binario que digitalmente codifica el nivel de la señal. La longitud de la palabra determina la calidad de la representación. Una vez más, una palabra más larga, mejora la calidad de un sistema de audio (comparando una palabra de 8 bits con una de 16 bits o 32 bits) (ver figura)” [9].

Figura 8: Comparación en el número de bits de cuantización [9]



“Ya sabemos que en el audio digital la base de numeración empleada es la binaria. Cada cifra binaria esta compuesta por un número de bits. Cuantos más bits tenga una cifra binaria más valores diferentes será capaz de representar” [8].

1 bit => 2^1 = 2 valores distintos
2 bits => 2^2 = 4 valores distintos
8 bits => 2^8 = 256 valores distintos
16 bits => 2^16= 65536 valores distintos

“El audio digital grabado en un CD (formato CDa) se corresponde con una cuantificación a 16 bits que logran una cuantización bastante precisa de la señal. El número de bits a los que se realiza la cuantización también influye en la relación señal ruido. Al cuantificar las señales analógicas en números binarios discretos se produce un ruido llamado ruido de cuantización. La relación de señal-ruido en un

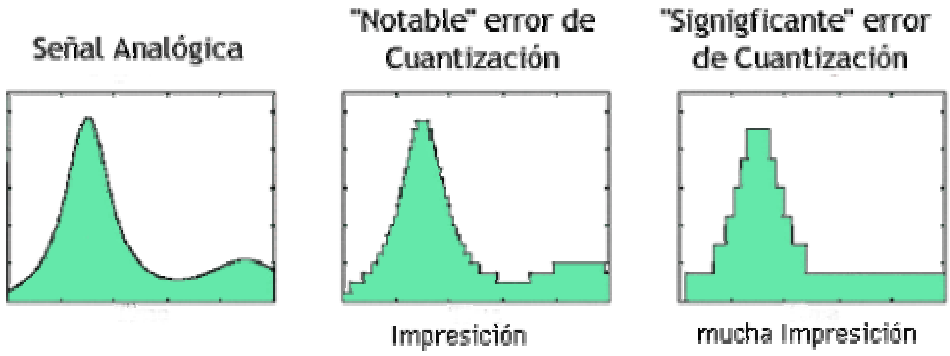
sistema ADC es de 6 dB por bit. Se ha convenido, que un sistema de 16 bits, es suficiente para tratar con el ruido de cuantización que da una relación señal-ruido de 96 dB lo que indica que es muy bueno" [8].

Otros ejemplos:

8 bits equivale a **256 estados** = 48 dB (decibeles)
16 bits equivalen a **65,536 estados** = 96 dB.

Entonces, se deben tomar muestras a tiempos menores y se debe de cuantificar a mayores niveles (bits), si sucede lo contrario suceden **errores de cuantización**:

Figura 9: Errores de Cuantización [9]

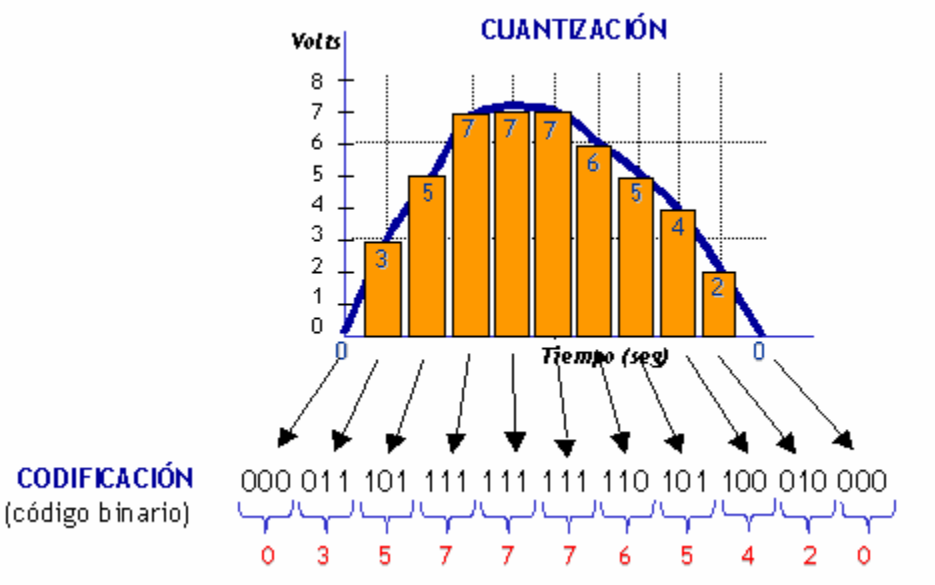


3.3.4. Codificación

"La codificación es la representación numérica de la cuantización utilizando códigos ya establecidos y estándares. El código más utilizado es el código binario, pero también existen otros tipos de códigos que son empleados" [9].

"Para adaptar los datos a las características propias del sistema de grabación se emplea un ***código de canal*** que permite un óptimo aprovechamiento del espacio de grabación y posibilita que los datos puedan recuperarse después fácilmente" [8.]

Figura 10: Codificación binaria [9]



"La codificación del canal implica además estructurar los datos que van a ser grabados de acuerdo a una señal del reloj que se registra al mismo tiempo, de manara que exista la suficiente información como para que durante la reproducción puedan recuperarse tanto los datos como el sincronismo" [8].

3.4. TARJETAS DE SONIDO

Tomado de [11]:

> Las dos funciones principales de estas tarjetas son la generación o reproducción de sonido y la entrada o grabación del mismo. Para reproducir sonidos, las tarjetas incluyen un chip sintetizador que genera ondas musicales.
>
> Este sintetizador solía emplear la tecnología FM, que emula el sonido de instrumentos reales mediante pura programación; sin embargo, una técnica relativamente reciente ha eclipsado a la síntesis FM, y es la síntesis por tabla de ondas (WaveTable).
>
> Una buena tarjeta de sonido, además de incluir la tecnología WaveTable, debe permitir que se añada la mayor cantidad posible de memoria. Algunos modelos admiten hasta 128 Megas de RAM (cuanta más, mejor).

Una tarjeta de sonido también es capaz de manipular las formas de onda definidas; para ello emplea un chip DSP (Digital Signal Processor, Procesador Digital de Señales), que le permite obtener efectos de eco, reverberación, coros, etc. Las más avanzadas incluyen funciones ASP (Advanced Signal Processor, Procesador de Señal Avanzado), que amplía considerablemente la complejidad de los efectos. Por lo que a mayor variedad de efectos, más posibilidades ofrecerá la tarjeta.

Al hablar de tarjetas de sonido para PC, no se puede dejar de mencionar la popular ***Sound Blaster*** que se puede decir, es el estándar de hoy en día. "Las tarjetas de Creative Labs de Singapur conquistaron rápidamente el mercado y establecieron un estándar gracias a su relación precio beneficio. Y eso, a pesar de que otro productor se había introducido primero al mercado: Adlib, con su tarjeta del mismo nombre" [10]. De todas formas, hoy en día ningún productor puede prescindir de que sus tarjetas anuncien un cierto grado de compatibilidad con las Sound Blaster.

Funcionalidad de la Sound Blaster y compatibles

Las siguientes son las tres áreas que caracterizan la esencia de la funcionalidad de la Sound Blaster según [10]:

- La creación sintética de sonidos, una tarea que realizan no solo las tarjetas Sound Blaster, sino también muchas otras que cuentan con el llamado Chip OPL de Yamaha. El mismo crea hasta 11 o más tonos diferentes, que pueden ser ejecutados simultáneamente y que arrancan del PC una verdadera diversidad sonora, mediante una adecuada programación.
- El muestreo a través del llamado DSP que se encuentra en cada Sound Blaster. Cuando se quieren grabar señales desde un CD, desde un equipo estéreo o a través de un micrófono, con la mayor calidad posible, grabarlas en un soporte y volver a ejecutarlas posteriormente, entra a trabajar el DSP.
- El mezclador, que controla los volúmenes de las distintas fuentes de señales y los combina entre si. En la medida que se han ido desarrollando las capacidades del DSP, se ha ido haciendo también más eficiente el mezclador.

4. DIGITALIZACIÓN DE VOZ

Cápitulo tomado de [12]:

La digitalización de la voz se ha estudiado profundamente para las redes de telefonía mundial. Sabemos que uno de los grandes problemas de la información analógica es el ruido introducido a través de sucesivas copias, pero a pesar del ruido y atenuación, las señales pueden regenerarse. Se establece que el canal de voz es de 300 a 3.400 Hz. Sobre esta gama de frecuencias se establecen 128 niveles de cuantificación de tonos positivos y 128 negativos, dando un total de 256 niveles, que son suficientes para obtener una reproducción concreta de la voz (8 bits son suficientes para representar lo 256 niveles).

Todo ello puede parecer muy simple, pero en la práctica es bastante más complejo, pues los niveles de cuantificación de tono no son divisiones uniformes, sino que se aumenta su densidad en las zonas de más información o principal contenido, y que se encuentran entre 800 y 1.500 Hz.

Los 256 niveles se codifican con trenes de impulsos de 8 bits, de los que siete son bits de cuantificación y uno es de polaridad o signo. Para el canal de voz se necesitan 64 kbit/s (kilobits por segundo) para obtener una buena nitidez.

Actualmente los enlaces telefónicos por microondas se realizan perfectamente en forma digitalizada, con las grandes ventajas de la regeneración de señal. El ancho de banda no es una preocupación al trabajar en bandas de UHF y SHF, teniendo en cuenta además de que el ancho de banda ocupado es doble por ser las conversaciones telefónicas con emisión y recepción simultánea, lo que se conoce como *full duplex*, y conllevando un mismo emisor de microondas hasta 1.296 conversaciones simultáneas.

Existen dos tendencias para realizar la digitalización de la voz con velocidades más lentas, la llamada modulación diferencial de impulsos codificados (DPCM), que consiste en trocear la banda vocal en cinco intervalos llegando a obtener 16 kbit/s, que aún mantiene los parámetros de la persona que habla, es decir del timbre. Esto permitiría utilizar canales de FM en VHF (144 MHz) de 25 kHz, si bien la tendencia es pasar a 12,5 kHz de ancho de banda por canal, con lo que debería reducirse la velocidad de emisión de la voz digitalizada bajando por debajo de los 16 kbit/s.

Otra técnica es la llamada del sintetizador o LPC (Predicción Lineal) Que se verá más en detalle en el capitulo de compresión de voz. Se basa en el estudio de la voz del que se deduce que sólo se utilizan unos 40 fonemas y de estos sólo 10 por

segundo. Teóricamente con un ancho de banda de 400 a 600 Hz debería poderse enviar la voz. Actualmente con esta técnica se hacen síntesis de mensajes pregrabados. Se extraen los formantes de la voz y de ahí todos los fonemas. Codificando los fonemas o parámetros LPC se pueden introducir en una ROM. Con un generador de ruido y los parámetros de la ROM se reproduce la voz de forma aceptable. Los formantes se estudian por ordenador, que sacan redundancias hasta obtener tasas de error despreciables. El denominado *vocoder* es un LPC para emisión.

Van llegando al mercado de consumo aplicaciones de la digitalización de la voz, facilitados por los chips realizados con LSI (*Large Scale Integration*; integración a gran escala); diversos juegos como el de ajedrez que habla; diccionarios electrónicos que indican vocalmente la palabra en el idioma buscado, dando por lo tanto la pronunciación correcta; tocadiscos compactos en los que se ha sustituido el fonocaptor por un rayo láser, ya no hay desgaste, y la música y voz han sido codificados digitalmente. NEC de Japón comercializa el terminal SR-100 / AR-100 capaz de reconocer 120 palabras con una precisión del 99%. NEC suministra tres chips LSI, el MC-4760 o procesador analógico, el microprocesador PD7761 D, procesador de reconocimiento, y el PD7762G, controlador. Se necesitan memorias de 16 kbyte para grabar 128 palabras en forma digitalizada.

Se encuentran ya en el mercado español tarjetas de circuito impreso con mensajes pregrabados. De forma estática se obtienen avisos de alarma en caso de incendio, alarmas por exceso de temperatura, fugas de agua, etc., y en los casos en que se necesita un aviso de voz real. Se podría hacer por casete, pero cuando un dispositivo electromecánico se pone a punto como alarma, si ésta se activa al cabo de un año por ejemplo, es casi seguro de que el casete falle.

La digitalización de la voz permite:

- Mejorar las comunicaciones frente al ruido e interferencias.
- Almacenar la voz, procesarla y tratarla exactamente como cualquier otro tipo de información digital, por ejemplo la introducida manualmente por el teclado de un Terminal.
- Permite la robotización o control sin manos, de equipos, vehículos, ascensores, etc., con lo que los mandos, botones pulsadores, etc. irán desapareciendo y quedarán sustituidos por un micrófono.

Todo ello permite entrever o intuir que la aplicación de la digitalización de la voz abre una nueva época para el radioaficionado, que simplemente se está iniciando.

5. FORMATOS DE ARCHIVOS DE SONIDO

5.1. INTRODUCCION

Introducción tomada de [13].

Los datos multimedia digitalizados suelen almacenarse en ficheros de gran capacidad y pueden contener imágenes, audio y/o vídeo, aunque se puede generalizar su contenido, diciendo que son todos los datos capturados, digitalizados y codificados por sistemas electrónicos. Una de las principales características de la formación de formatos con contenidos multimedia, es el gran tamaño que requiere el almacenamiento de los mismos. En pocos años el contenido de los datos se ha multiplicado en gran medida debido a la creciente calidad de los sistemas de captura y la necesidad de almacenamiento de muestras de calidad.

La calidad de contenidos multimedia se determina (dependiendo de la naturaleza de los datos) por el rango de los valores que representan esos datos en la cuantificación, y por la frecuencia de muestreo y resolución espacial en el caso de las imágenes. A mayor frecuencia, resolución, rango mayor calidad, eso implica una mayor cantidad de muestras por segundo en secuencias temporales, o mayor número de bits por muestra, en definitiva tamaños de ficheros en media muy superiores a otro tipo de formatos con otros contenidos.

Para tratar de disminuir la cantidad de datos a almacenar, la industria ha estudiado diversos sistemas de compresión/descompresión (códecs) con y/o sin perdidas. Es necesario tener en cuenta que el primer y más simple sistema de compresión es la codificación binaria, en frente de la codificación en texto.

Otra de las principales características de estos tipos de ficheros es la necesidad de representar y encapsular por un lado los datos que representan los sucesos físicos audio y/o vídeo, y por otro lado las variables del entorno de captura y digitalización, que hacen reproducible esa secuencia de datos, en caso de audio la frecuencia de muestro, canales y la codificación; en el caso de imagen/vídeo además de las anteriores, el número de colores representados o la paleta de colores usada en la cuantificación, o la resolución espacial.

La creciente demanda por clasificar, ordenar y buscar de forma sencilla cierta información sobre el contenido de un determinado fichero, ha llevado a la industria a la necesidad de incluir otro tipo de información además de los datos capturados y de las variables de captura, como pueden ser Autor del contenido, palabras clave que clasifican el contenido, fecha de creación, puntos de control etiquetados,

etc. Hoy día es común encontrar lo que se llama habitualmente meta-ficheros, o ficheros que contienen datos de diferente naturaleza.
Es habitual en la jerga informática y especialmente en el ámbito de desarrollo multimedia, confundir el modo de encapsular y organizar datos, el formato, con el sistema de representación de estos datos digitalizados, es decir, la codificación.

Se debe entender como formato el sistema de encapsulación que el fichero va usar para organizar los datos dentro del conjunto de bits que lo forman, sean de la naturaleza que sean (datos de información extendida o los datos multimedia), y se debe entender como codificación la representación de los datos como el sistema de almacenar de forma digital los datos multimedia capturados, también denominados códecs (codificador/decodificador) habitualmente de compresión, debido a la necesidad de encapsular más información en menos espacio, pero podría ser un sistema de representación que al contrario integrase redundancia (aumenta el tamaño) para tener la posibilidad de detectar y corregir (si el sistema de codificación lo permite) errores debidos a almacenamiento o transmisión en medios poco fiables.

Se considera importante el estudio de los sistemas de encapsulación de datos o formatos, por dos motivos. Por un lado es un tema poco y pobremente tratado y por otro lado, puesto que el software de edición normalmente permite la creación de nuevos contenidos es importante conocer los medios de organización y clasificación de los datos creados para su posterior almacenamiento, transmisión o reproducción.

5.2. ESTÁNDAR EA-IFF

Tomado de [13]:

El estándar EA-IFF-85 (Electronic Arts Standard for Interchange Format Files) fue redactado en 1985 por desarrolladores de Electronic Arts y Commodore-Amiga basándose en iniciativas previas de Adobe. La redacción de este estándar supuso el primer intento con éxito de estandarizar un sistema abierto de almacenamiento de datos básicamente dirigido a la formación de archivos con contenidos multimedia.

El estándar IFF define un formato de archivo que se caracteriza por ser **binario** y **estructurado** con una unidad básica de estructuración denominada **“chunk”** O Pedazo. El chunk no es más que una etiqueta asociada a unos datos, el sistema de encapsulación recurrente de chunks le permite agrupar diferentes tipos de datos, lo que lo hace tremendamente útil para la formación de meta-archivos, además es **extensible**, la extensibilidad permite añadir nuevos chunks sin alterar el funcionamiento de las aplicaciones que no requieren de esta nueva información.

A pesar de ser un estándar redactado hace 17 años aun hoy día sigue siendo vigente. A la redacción de este estándar se siguió una aceptación de Apple Computer, y basándose en IFF publicó su estándar para archivos de audio AIFF y audio compreso AIFC. IBM y Microsoft publicaron su propio estándar RIFF (Resource Interchange File Format) basándose en el IFF. Los hoy populares archivos WAV y AVI se basan en RIFF o lo que es lo mismo en la estructura de estándar IFF. Este estándar es hoy tan valido con el primer día, y continua siendo utilizado por miles de programas, de manera directa o a través de formatos RIFF, pero en el caso de audio toma una especial relevancia si se tiene en cuenta que en 1997 la MMA (MIDI Manufacturers Asociation) creo el estándar DLS 1 y 2 (Downloable Sounds) que permite el almacenamiento de audio así como la articulación asociada para crear instrumentos y bancos de instrumentos. El formato de los ficheros se basan en el estándar RIFF, y a su vez el Formato SASBF (Structured Audio Sample Bank Format) del estándar MPEG4 es una derivación del estándar DLS de la MMA.

5.3. El FORMATO WAVE

“La técnica de recuperar datos de un archivo de sonido digital para reproducirlos recibe el nombre de audio WaveForm, audio digital o sonido basado en muestras y corresponde a los sonidos grabados con extensión WAV” [14].

El formato Wave (Onda), es el estándar de Microsoft para el almacenamiento de sonidos generalmente sin ninguna compresión y que soporta una variedad de resoluciones de bits, rangos de muestreo y canales de audio. Éste se apoya en el estándar RIFF publicado por IMB y Microsoft que aparte de soportar el formato wave, también soporta otros formatos multimedia de imagen estática y de video. RIFF, de sus siglas en ingles “Resource Interchange File Format” o Formato de Archivos de Intercambio de Recursos, no constituye en sí un formato, es decir, no se almacena nada en formato RIFF, sino que es un marco de referencia o una guía de especificaciones a seguir para definir ficheros de datos.

Como resultado de un trabajo en común realizado por Microsoft e IBM, en 1991 aparece el "Multimedia Programming Interface and Data Specifications 1.0", considerado por muchos como la Biblia de los formatos de multimedia. En este documento se explica la filosofía de diseño y las características específicas de cada tipo de formato multimedia para Windows y OS/2. Algunos de los ficheros de datos que tienen su origen en la especificación RIFF son:

AVI: Vídeo y Audio Entrelazado (Audio/Vídeo Interleaved)
BND: Paquete (Bundle)
DIB: Mapa de Bits Independiente de Dispositivo (Device-independent bitmap)
MIDI: Interfaz Digital de Instrumentos Musicales (Musical Instrument Digital Interface)

RTF: Formato de Texto Enriquecido (Rich Text Format)
WAVE: Audio Ondulatorio (WaveForm Audio)

5.4. EL ESTANDAR RIFF

El elemento básico de construcción de un fichero RIFF es el "chunk" (pedazo). Un "chunk" es una estructura que contiene:

1. **El nombre del chunk**, de cuatro caracteres como máximo, no tiene por qué acabar en nulo (aunque puede ser una cadena de caracteres, hay que tener en cuenta que no se sigue el estándar de C de terminar con un nulo); puede considerarse que es un array de 4 caracteres, o un dato de tipo ulong (Unsigned Long, que ocupa cuatro bytes).
2. **El tamaño**, que es un entero largo sin signo (ulong).
3. **Datos**, que son la información que la aplicación sabe interpretar.

Hay dos tipos de chunks que pueden contener a otros chunks anidados (subchunks). Son los "RIFF" y "LIST", pero, en general, los chunks solo almacenan un tipo de elementos en los datos binarios. Un formulario o fichero RIFF se podría definir, y de hecho se hace, como un chunk con nombre "RIFF", de forma que es un chunk que comprende todo el fichero de datos que sigue la convención RIFF.

Todos los ficheros RIFF empiezan, por lo tanto, por las cuatro letras "R", "I", "F", "F", a continuación el tamaño, y luego ya los datos binarios. Dentro de estos datos los primeros cuatro caracteres identifican el tipo de **formulario**, y a continuación le seguiría uno o varios tipos de subchunks. El código definido para un formulario RIFF debe ser único: hay que registrarlo en Microsoft especificando cómo se llama y cuál es la estructura del chunk que se quiere definir dentro del estándar, qué tipo de compresión llevan los datos digitalizados, cuáles son los nuevos nombres de chunk que se quieren definir, etc., etc. El nombre del **formulario** de un archivo .wav es “WAVE”. De todas formas, hay tantos chunks definidos ya para todo tipo de aplicaciones multimedia que, en general, no hace falta definirse uno nuevo, sino que puede usarse uno ya existente.

A continuación presentamos la estructura de un RIFF wave según [17]:

Desplazamiento (Hex)	**Contenido**
0000	'R', 'I', 'F', 'F' (0x46464952)
0004	Tamaño del archivo completo sin contar Los primero 4 caracteres (32 bits)
0008	Tipo de Formulario (4 Caracteres)
000C	Tipo del Primer chunk (4 character)
0010	Tamaño del primer chunk (32 bits)
0014	Datos del primer chunk

5.4.1. Tipos de Datos y su organización

Las cadenas de caracteres en un fichero de estructura RIFF se almacenan como se almacenan las variables string en Pascal: guardan en el primer byte de la cadena, el tamaño de la misma:

7	'e'	'j'	'e'	'm'	'p'	'l'	'o'

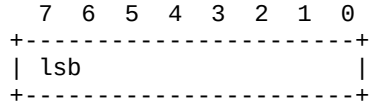
Ejemplo del formato string

El ejemplo anterior no es del todo correcto. El primer byte tiene el carácter 7 cuyo código ASCII = 55 lo que indicaría que la cadena tiene un tamaño de 55 caracteres. En su lugar debería mostrar el carácter correspondiente al ASCII 7.

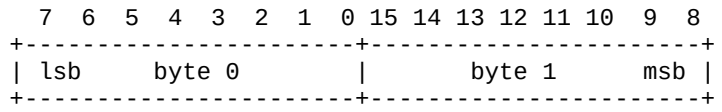
Acerca de cómo se organizan los datos en el área de datos de un chunk hay que mencionar que se guardan como se guarda un dato en memoria para ser leído por un procesador Intel (Little-Endian). Esto es, invirtiendo los bytes que contienen los bits más y menos significativos:

Figura 11: Almacenamiento de las muestras según su tamaño [18].

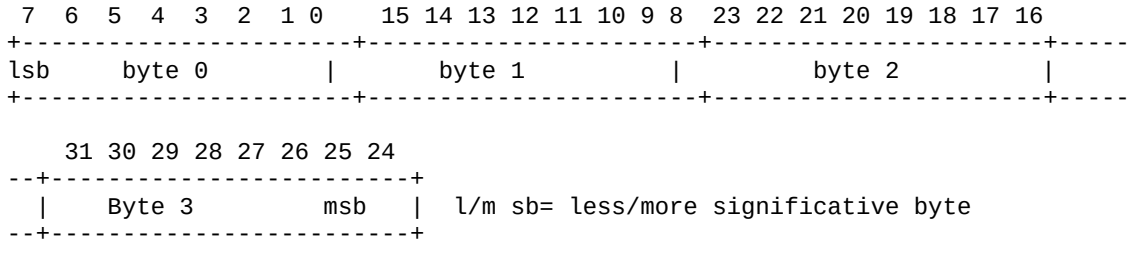
Para un byte (byte)



Para 2 bytes (Integer)



Para 4 bytes (Long)



5.4.2. Puntos de Muestra y Frames de Muestra

Una gran parte de la interpretación de los ficheros WAVE se resuelve alrededor de dos conceptos como son puntos de muestra y frames de muestra.

Un punto de muestra es un valor que representa una muestra de un sonido en un instante de tiempo. Para las formas de onda con resoluciones mayores de 8 bits, cada punto de muestra es almacenado como un "linear", un valor en complemento a dos el cual puede tener de 9 a 32 bits de ancho (como se determina en el campo BitsPerSample en el chunk Formato, asumiendo el formato PCM [1] --un formato descompreso). Por ejemplo, cada punto de muestra de una forma de onda de 16 bits se puede poner en una palabra de 16 bits (es decir, 2 bytes de 8 bits) donde el 32767 (0x7FFF) es el valor más alto y -32768 (0x8000) es el valor más bajo. Para formas de onda de 8 bits y menores, cada punto de muestra es un byte sin signo lineal donde el 255 es el valor más alto y el 0 es el valor más bajo.

Debido a que la mayoría de las CPU leen y escriben en operaciones con 8 bits (byte), se decidió que un punto de muestra debería ser redondeado a un tamaño que sea múltiplo de 8 cuando se almacena en un WAVE. Esto hace a los WAVE más fáciles de leer en la memoria.

Los bits de datos deberían estar justificados a la izquierda, con algunos bits de relleno puestos a cero cuando sea necesario. Por ejemplo, consideremos el caso de un punto de muestra de 12 bits, con lo que el punto de muestra debería ser almacenado como una palabra de 16 bits. Estos 12 bits deberían estar justificados a la izquierda por lo que estos deberían ser los bits 4 a 15 inclusive y los bits 0 a 3 deberían de ponerse a cero.

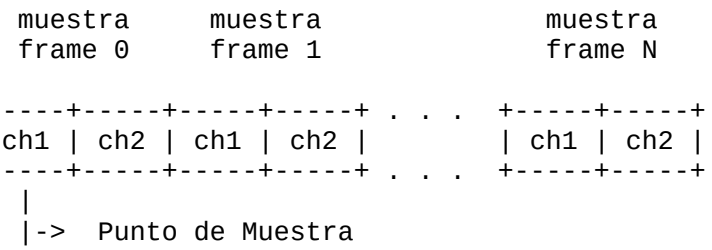
Sin embargo, hay que notar que, debido a que el formato WAVE usa el orden de bytes Little-endian, el LSB se almacena primero en el fichero WAVE.

Para sonidos **multicanal** por ejemplo, una forma de onda estéreo, los puntos de muestra de cada canal se intercalan. Por ejemplo, asumiendo una forma de onda estéreo (de 2 canales), en lugar de almacenar todos los puntos de muestra para el canal izquierdo primero, y entonces almacenar todos los puntos de muestra para el canal derecho después, se mezclan los puntos de muestra de los dos canales juntos empezando por el canal izquierdo.

[1] PCM (Pulse Code Modulation): Las muestras son almacenadas en forma lineal no comprimida. Consiste en tomar muestras del sonido a intervalos regulares dados por la frecuencia de muestreo y asignarle un número entero a cada muestra. La cantidad de valores distintos que puede tomar cada muestra depende de la resolución que se use (8 ó 16 bits)

Los puntos de muestra que se quieren reproducir (enviándolos a un DSP) simultáneamente son colectivamente llamados “Frames” de muestra. En el ejemplo de nuestra forma de onda estéreo, cada dos puntos de muestra crea un frame de muestra. Este ejemplo queda ilustrado en la siguiente figura:

Figura 12: Almacenamiento de las muestras según el número de canales.



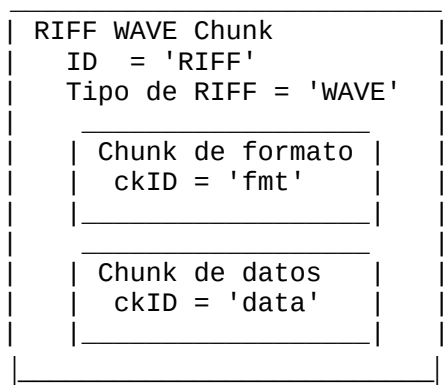
Para una forma de onda mono canal, un frame de muestra es meramente un punto de muestra simple (es decir, no hay nada que intercalar). Para formas de ondas multicanal, se deberían seguir las convenciones mostradas, las cuales ordenan el almacenamiento de canales dentro del frame de muestra.

Los puntos de muestra dentro de un frame de muestra se empaquetan juntos, no hay bits sin uso dentro de ellos. De tal forma, los frames de muestra son empaquetados juntos sin bits de relleno.

5.5. CHUNKS DE UN WAVE

Un archivo wave esta conformado como mínimo por dos chunks: el **fmt** y el **data.** El chunk fmt (formato) contiene los parámetros más importantes de los datos de la forma de onda tales como la frecuencia de muestreo y la resolución de bits, los canales de audio (mono o stereo). Y el chunk data contiene los datos de muestreo digital en sí. El siguiente diagrama nos ilustra:

Figura 13: Esquema de un archivo Wave.



Aparte del fmt y el data existen otros chunks opcionales que han sido incluidos como: el chunk de hechos, pistas, lista y lista datos asociados. Entre los parámetros opcionales hay algunos que pueden definir información específica del almacenamiento por parte de las aplicaciones. Todas las aplicaciones que usen WAVE deberán ser capaces de leer los dos campos requeridos, Format y Data, y pueden elegir selectivamente ignorar los chunks opcionales. Un programa que copie un WAVE debería ser capaz de copiar todos los "chunks" del fichero WAVE, incluso aquellos que se eligieron no interpretar.

No hay restricciones sobre el orden de los chunks dentro de un fichero WAVE, con la excepción de que el chunk **fmt** debe preceder al chunk **data**. Algunos programas escritos de forma inflexible esperan el chunk **fmt** como el primero (después de la cabecera RIFF) a pesar de que no debería ser así puesto que las especificaciones no requieren este punto.

5.5.1. El chunk fmt (formato)

El chunk Formato (fmt) describe los parámetros fundamentales de los datos de la forma de onda tales como el rango de muestreo, la resolución de bits, y cuantos canales de audio digital son almacenados en el WAVE.

Tabla 4: Chunk Formato [16].

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"fmt " (0x666D7420)
0x04	4	Tamaño de datos del chunk	16 + bytes de formato extra
0x08	2	Código de compresión	1 - 65,535
0x0a	2	Numero de Canales	1 - 65,535
0x0c	4	Rata de muestreo	1 - 0xFFFFFFFF
0x10	4	Promedio de bytes por segundo	1 - 0xFFFFFFFF
0x14	2	Alineación de los bloques	1 - 65,535
0x16	2	Bits significativos por muestra	2 - 65,535
0x18	2	Bytes de formato extra	0 - 65,535
0x1a	Contenido de formato extra		

***El chunk ID*:** es siempre "**fmt** " (incluyendo el espacio al final para rellenar los 4 caracteres). El valor en hexa que se muestra en la tabla corresponde a:

Carácter	ASCII en Decimal	ASCII en Hex
f	102	0x66
m	109	0x6D
t	116	0x74
Espacio	32	0x20

El tamaño de datos del chunk (Chunk Data Size): es el número de bytes en el chunk. Este no incluye los 8 bits usados en el ID y en el campo de tamaño. Para el chunk Formato, el Chunksize puede variar de acuerdo a que formato de fichero WAVE se especifique. Es decir, depende del valor del código de compresión o FormatTag.

Código de compresión (FormatTag): Los datos del WAVE pueden ser almacenados sin compresión, en cuyo caso los puntos de muestra son los recibidos directamente del DSP en el proceso de grabación. Alternativamente, se pueden usar diferentes formas de compresión cuando se almacenan los datos de sonidos en el chunk Data. Con compresión, cada punto de muestra debería tocar un número diferente de bytes a almacenar. El FormatTag indica si se ha utilizado compresión en el almacenamiento de los datos.

Si se usa compresión (es decir, FormatTag tiene un valor distinto de 1), habrán campos adicionales unidos al chunk Format los cuales darán la información necesaria para un programa que desee retirar y descomprimir los datos almacenados. El primer campo adicional será un unsigned short que indica cuantos bytes más han sido añadidos (después de este unsigned short). Además, los formatos compresos deben tener un chunk Fact el cual contiene un unsigned long indicando el tamaño (en puntos de muestra) de una forma de onda después de que esta haya sido descomprimida. Hay demasiados formatos de compresión que serán descritos brevemente más adelante en el numeral 10.3 (codecs soportados por Windows). Por ahora, la siguiente es una lista de los formatos más conocidos:

Tabla 5: Algunos formatos de compresión del formato wave [16].

Código	Descripción
0 (0x0000)	Unknown
1 (0x0001)	PCM / uncompressed
2 (0x0002)	Microsoft ADPCM
6 (0x0006)	ITU G.711 a-law
7 (0x0007)	ITU G.711 μ-law

17 (0x0011)	IMA ADPCM
20 (0x0016)	ITU G.723 ADPCM (Yamaha)
49 (0x0031)	GSM 6.10
64 (0x0040)	ITU G.721 ADPCM
80 (0x0050)	MPEG
85 (0x0055)	MPEG Layer III
65,536 (0xFFFF)	Experimental

Si no se usa compresión (es decir, FormatTag = 1), entonces no habrá más campos adicionales.

Numero de Canales (Channels): contiene el número de canales de audio para el sonido. Un valor de 1 significa sonido monofónico, 2 significa stereo, 4 significa 4 canales de sonido, etc. Se puede representar cualquier número de canales de audio. Para sonidos multicanales, los puntos de muestra simple de cada canal se deben intercalar. El establecimiento de puntos de muestra intercalados se llama frame de muestra. (Ver numeral 5.4.2)

Rata de muestreo (SamplesPerSec): es el rango de muestreo al cual el sonido será reproducido en frames de muestra por segundo (es decir, Hz). Los tres estándares de rangos PCM son 11025, 22050 y 44100 KHz, aunque se pueden usar otros rangos.

Promedio de bytes por segundo (AvgBytesPerSec): Indica cuantos bytes se reproducen por cada segundo. Este campo puede ser usado por las aplicaciones para estimar que tamaño del buffer de RAM se necesita para reproducir el WAVE sin problemas de latencia o retardos. Este valor debería coincidir con la siguiente fórmula redondeado al siguiente valor:

AvgBytesPerSec =SamplesPerSec * BlockAlign

Alineación de Bloques (BlockAlign): debería ser igual a la siguiente fórmula, también redondeando:

Channels * (BitsPerSample / 8)

Esencialmente, BlockAlign es el tamaño de un frame de muestra, en términos de bytes. (Ej. un frame de muestra para una onda de 16 bit-mono es de 2 bytes. Un frame de muestra para una onda de 16-bits stereo es de 4 bytes, etc).

Bits significativos por muestra (bitsPerSample): es la resolución de bits de un punto de muestra (Ej. una forma de onda de 16 bits debería tener BitsPerSample=16).

Bytes de formato extra: especifica el número de bytes extra usados para la información de compresión. Es cero si el campo FormatTag es 0 o 1, y toma un valor si el campo FormatTag es mayor que 1.

5.5.2. El chunk data

El chunk Data contiene los frames de muestra a tratar, es decir, todos los canales de los datos de la forma de onda. El contenido es el siguiente:

Tabla 6: Chunk de datos [16].

Desplazamiento	Tamaño	Tipo	Descripción	Valor
0x00	4	char[4]	chunk ID	"data" (0x64617461)
0x04	4	Palabra doble	Tamaño de los datos	Depende del tamaño de la muestra y de la compresión.
0x08	Datos de la muestra			

El campo ID es siempre **data**. El tamaño de los datos (Chunksize) es el número de bytes en el chunk, sin contar los 8 bytes usados por el campo ID y el propio campo de tamaño. Tampoco se cuenta los bytes de relleno. Para más información acerca de la colocación de los datos ver **puntos de muestra y frames de muestra,** Numeral 5.4.2**.**

5.5.3. El Chunk Fact

Un chunk que almacena la información dependiente del código de compresión aplicado a lo datos de onda. Es requerido por todos los formatos compresos de onda, pero no se requiere para los archivos sin comprimir, o sea, con formato PCM (código de compresión = 1).

Tabla 7: Chunk Adicional para formatos compresos [16].

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"fact" (0x66616374)
0x04	4	Chunk Data Size	Depende del formato
0x08	Formato del que depende el chunk DATA		

5.5.4. El chunk Silent (Silencio)

Se utiliza para especificar un segmento de silencio que dure un cierto número de muestras. Representa tiempos de silencio pero, estos silencios no forman parte de los datos en el chunk data.

Tabla 8: Chunk Silencio [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"slnt" (0x736C6E74)
0x04	4	Chunk Data Size	4
0x08	4	Número de muestras de silencio	0 - 0xFFFFFFFF

Este chunk no especifica que tipo de silencio se va a mezclar con la onda de sonido, está a cargo de eso la aplicación que interprete el silencio. Hay que tener cuidado si se quiere insertar silencios con las diferencias entre la cuantización a 8 y a 16 bits, pues en ambos casos un silencio es diferente (ver: Puntos de muestra y frames de muestra).

5.5.5. El chunk Cue (pista o señal)

Contiene uno o más "puntos de aviso" o "marcas". Cada punto de aviso referencia un *offset* (desplazamiento) específico dentro del array waveformData, y tiene su propia estructura de puntos de aviso dentro de él mismo. En conjunto con el chunk Playlist el chunk Cue y otros, puede ser usado para almacenar información de repeticiones o búsquedas rápidas de marcas predefinidas.

Los chunks en adelante dependen mucho del chunk cue. El chunk Cue, junto con los chunks PlayList, list, notes, labels, Labeled text, conforman una estructura completa que permite asociar información a los datos de la muestra en el chunk data. Una pista, además de apuntar a datos en el chunk data, también puede apuntar a un chunk silencio.

Cabe aquí hacer un pequeño paréntesis relacionado con el proyecto. Uno de los objetivos del proyecto es: Determinar la forma de realizar grabación y reproducción parcial o total, según la **Incorporación o asociación de etiquetas de control con archivos de sonido digitalizados**. El problema como tal, se pensó resolver en un principio, creando archivos adjuntos al archivo de sonido en formato texto, con una estructura de datos que sirviera para identificar puntos de información en el bloque de datos de muestra de sonido. Pero, según lo mencionado en el párrafo anterior, no es necesario crear más archivos, sino usar la estructura que nos ofrece el estándar wave para la tarea propuesta.

La siguiente es la estructura del chunk Cue:

Tabla 9: Chunk Pista [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"cue " (0x63756520)
0x04	4	Tamaño de datos en el chunk	Depende del número de puntos pista
0x08	4	Número de Puntos Pista	
0x0c	Lista de Puntos Pista		

Para calcular el tamaño del chunk se usa:

```
ChunkDataSize = 4 + (NumCuePoints * 24)
```

Una pista o señal contenida en el campo (Lista de Puntos Pista) contiene los siguientes datos:

Tabla 10: Lista de puntos de pista en un chunk Cue [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	ID	Identificador único
0x04	4	Posición	Posición relacionada con el chunk lista de ejecución (PlayList)
0x08	4	Data Chunk ID	RIFF ID del correspondiente chunk data
0x0c	4	Chunk Start	Desplazamiento en el Chunk Data
0x10	4	Block Start	Relacionado con el desplazamiento del primer canal
0x14	4	Sample Offset	Relacionado con el desplazamiento del primer canal

ID

Cada punto de la señal tiene un valor único de identificación usado para asociar puntos de la pista con información en otros chunks. Por ejemplo, un chunk de etiqueta (Label) contiene el texto que describe un punto en el archivo refiriéndose al punto asociado de la señal.

Posición

Este campo es usado principalmente para asociar la pista con el chunk PlayList. Dicho de otra manera, si se especifica en un chunk playlist un segmento (ver chunk playlist) que apunte a una pista (cue), el valor de la posición es igual al número de la muestra en el cual entran a jugar tanto el cue como el segmento del playlist. Si no se especifica ningún chunk playlist, éste valor debe ser 0.

Data Chunk ID

Especifica el chunk donde están los datos a los cuales hace referencia la pista, casi siempre tiene el valor “data”, pero puede también tener el valor “slnt” para referirse a un chunk con silencios.

Chunk Start

Es un valor opcional, generalmente es 0. Es usado para apuntar a un chunk de tipo wavelist que no es muy usado por los programadores debido a que genera inconcordancias.

Block Start

Especifica el byte de desplazamiento o la dirección en el chunk data, el bloque de inicio debe empezar con una muestra.

Sample Offset

Cuando los datos en el chunk data se encuentran sin compresión, la posición a la que apunta la pista queda completamente descrita por el byte de desplazamiento (Block Start), pero cuando los datos usan algún tipo de compresión, una muestra no necesariamente inicia en un múltiplo de byte (8), por lo cual, combinando con el formato de compresión y el número de muestra, se puede localizar la dirección a la que apunta la pista.

5.5.6. El chunk PlayList

El chunk PlayList especifica un orden de reproducción para una serie de puntos de aviso. El chunk Cue contiene todos los puntos de aviso, pero el chunk Playlist, cómo los puntos de aviso se usan cuando se reproduce la forma de onda. El chunk Playlist contiene una o más estructuras de segmento, cada una de las cuales identifica una sección de bucle (un segmento puede indicar, además del orden de ejecución, cuantas veces se repite dicho segmento) de la forma de onda.

Tabla 11: Chunk Lista de ejecución [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"plst" (0x736C6E74)
0x04	4	Chunk Data Size	Número segmentos * 12
0x08	4	Numero de segmentos	1 - 0xFFFFFFFF
0x0a	Lista de Segmentos		

Lista de Segmentos:

Una lista de segmentos es simplemente un sistema de descripciones consecutivas de segmento. Los segmentos no tienen que estar en ningún orden particular, pues contienen un ID de cue, y éste a la vez, contiene la posición asociada a los datos.

La siguiente es la estructura de un segmento de playlist:

Tabla 12: Segmento de lista de ejecución [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Cue Point ID	0 - 0xFFFFFFFF
0x04	4	Tamaño (en muestras)	1 - 0xFFFFFFFF
0x08	4	Numero de repeticiones	1 - 0xFFFFFFFF

Tamaño (length): indica el tamaño en número de muestras a partir de la posición inicial de los datos a los que apunta la pista identificada con “Cue point ID”.

Número de Repeticiones: indica si un segmento se repite más de una vez. El valor para una sola reproducción es 1. Con 0, el segmento del playlist no será tenido en cuenta.

5.5.7 Chunk de lista de datos asociados

La lista de datos asociados contiene etiquetas de texto o nombres que están asociados con las estructuras de los puntos de aviso en el chunk Cue. En otras palabras, esta lista contiene las etiquetas de texto para estos puntos de aviso.

¿Qué es una lista? Una lista es simplemente un chunk maestro que contiene varios subchunks. Como cualquier otro pedazo, el chunk maestro tiene un ID y un Chucksize, pero dentro de este chunk hay subchunks, cada uno con su propio ID y

su ChuckSize. Por supuesto, el ChuckSize del chunk maestro contiene el tamaño total de todos estos subchunks incluyendo sus campos ID y sus ChuckSize). Dentro de estos subchunks existen tres estándares definidos y son el Label Chunk, el Note Chunk y el Labeled Text Chunk.

La estructura del Chunk Lista de datos asociados es:

Tabla 13: Chunk Lista de datos asociados [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"list" (0x6C696E74)
0x04	4	Chunk Data Size	Depende de los subchunks contenidos
0x08	4	Type ID	"adtl" (0x6164746C)
0x0c	Lista de texto, etiquetas y nombres		

El *type ID* es un identificador de lista para el wave dentro de todo el estándar RIFF. El tamaño del chunk **list** se calcula conociendo todos los subchunks contenidos en la lista de texto, etiquetas y nombres.

5.5.8. Los chunks Label y Note (Etiqueta y Nota)

Son dos subchunks del chunk lista de datos asociados. Ambos cumplen la función de asociar un texto a un Cue ID. Realmente no hay diferencia funcional entre los dos subchunks, el uso del uno o el otro depende de las preferencias de los programadores. Su formato es el siguiente:

Tabla 14: Chunks Etiqueta y Nota [16]:

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"labl" (0x6C61626C) "note" (0x6E6F7465)
0x04	4	Chunk Data Size	Depende del texto contenido
0x08	4	Cue Point ID	0 - 0xFFFFFFFF
0x0c	Texto		

5.5.9. El chunk Texto Etiquetado

Es al igual que los chunks Label y Note, un subchunk del chunk Lista de datos asociados. Este chunk, a diferencia de Label o Note, se asocia a una región del área de datos del chunk data. El label o el note, solo almacenan el Cue ID (con el que accedemos al Cue y por tanto al inicio de un bloque de datos, más no a una región). Para completar los datos de la región, además de la posición de inicio, necesitamos saber el ancho de la región (algo parecido a lo que se hace en el chunk playlist), que es un campo incluido en el chunk Texto etiquetado como se ve en el formato, en el cual además de definir regiones, se pueden usar otros campos que describen información de interés:

Tabla 15: Chunk Texto Etiquetado [16].

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"ltxt" (0x6C747874)
0x04	4	Chunk Data Size	Depende del texto contenido
0x08	4	Cue Point ID	0 – 0xFFFFFFFF
0x0c	4	Sample Length	0 - 0xFFFFFFFF
0x10	4	Purpose ID	0 - 0xFFFFFFFF
0x12	2	Country	0 - 0xFFFF
0x14	2	Language	0 - 0xFFFF
0x16	2	Dialect	0 - 0xFFFF
0x18	2	Code Page	0 - 0xFFFF
0x1A	Text		

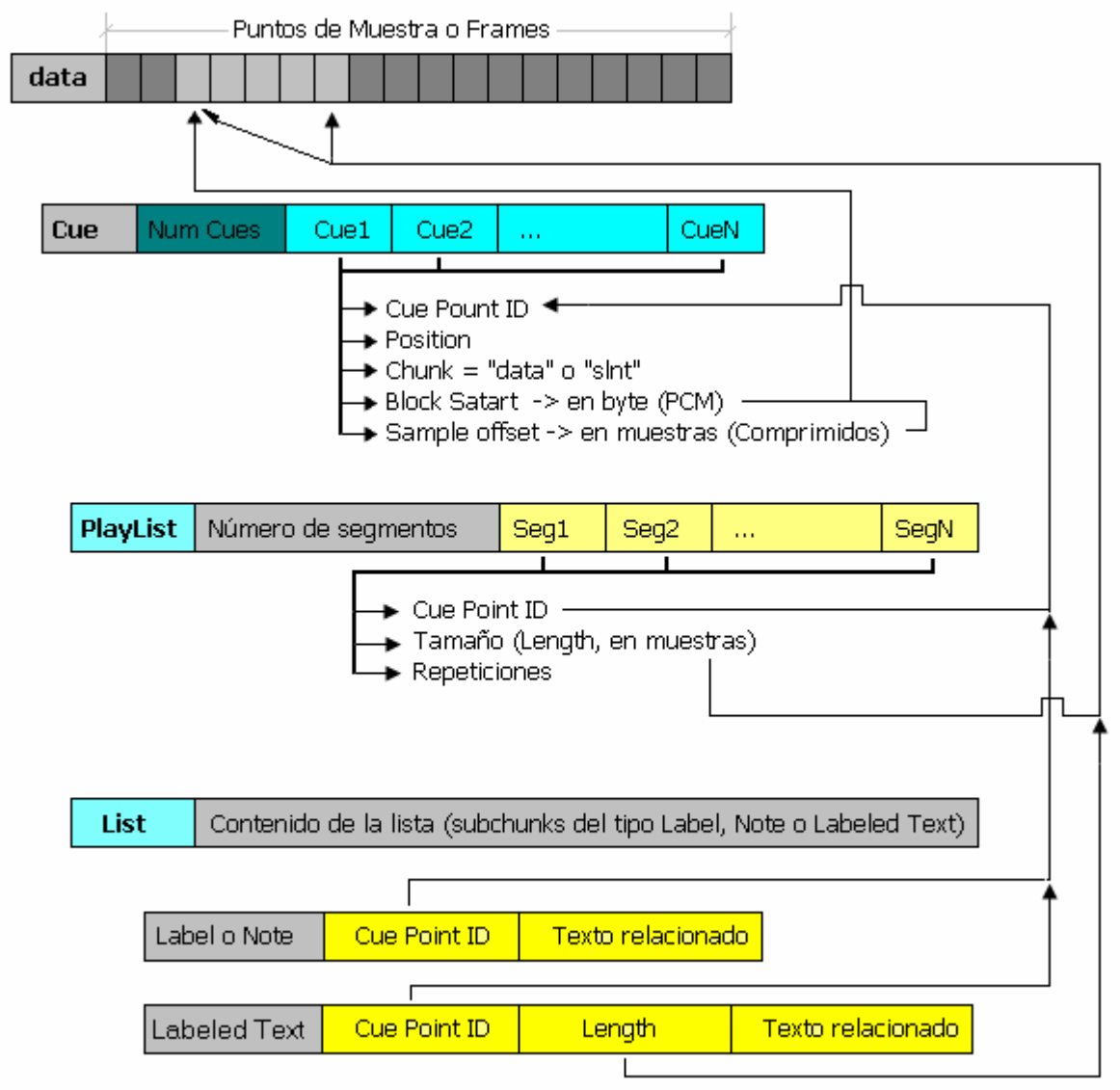
Sample Length (Tamaño de la Muestra): indica el ancho de la región a la que hace referencia la etiqueta por medio del chunk Cue.

Porpuse ID (ID del Propósito): son cuatro caracteres usados para el identificador del propósito para el que se usa el texto etiquetado. Realmente puede personalizarse por los programadores para el caso de archivos de sonido wave. Pero para otros tipos de RIFF como el AVI existen una gran cantidad de ID estandarizados. Se usa “scrp” (script) para propósitos generales en archivos de sonido wave.

Country, Language, Dialect and Code Page (País, Lenguaje, Dialecto y página de códigos): éstos campos contienen información acerca del lenguaje usado en el texto y algo de información del sistema operativo en el que fue creado.

El siguiente diagrama nos ilustra los chunks más importantes del RIFF wave:

Figura 14: chunks de un Wave.



5.6. OTROS FORMATOS

Existen muchos otros formatos de archivos de sonido y multimedia como el VOC (Creative Voice), el RA (Real Audio) pero ninguno presta mucha ayuda en cuanto a establecer etiquetas y puntos de aviso como el RIFF. Otros formatos ya definidos de forma compresa como el MP3, el VQF o el WMA, se estudiarán en los siguientes capítulos. Se menciona por ahora, que los archivos RIFF wave en su estructura admiten datos codificados o compresos, es decir, se puede tener un wave (con cabecera y chunks RIFF) con datos compresos en MP3. Se describirán a continuación algunos otros formatos existentes.

5.6.1. VOC (Creative Voice)

Es un archivo de sonido propuesto inicialmente por Creative Labs como un archivo de voces perfecto para incluir con videojuegos, pero con el tiempo se convirtió en el archivo más compatible con las Sound Blaster por lo que creció en formato para adaptarse a la reproducción de video, sincronización con imágenes u otros sonidos. Aún así, en cuanto a sonidos de voz, lo mejor que puede hacer el archivo VOC es guardar las muestras a menos frecuencia y bits de muestreo teniendo en cuanta la intensidad, el tono de la voz y las capacidades auditivas del ser humano, cosa que también se puede lograr con el wav y el debido codificador de sonido.

5.6.2. Formato RA (Real Audio)

“Este es el formato mas usado en Internet por su capacidad de reproducción en tiempo real, esto significa que mientras el archivo es bajado se escucha el sonido y cuando se termina de bajar el ya fue reproducido. Este formato fue desarrollado por RealNetworks. Esta empresa a puesto a disposición de los usuarios software para recibir y enviar en tiempo real (Tanto video como Sonido), La empresa es reconocida como una de las mas importantes en el mundo informático pues a puesto a su formato RA a la altura del Wav o del MIDI en popularidad. El problema mas grabe que tiene es que puede cortarse la reproducción del audio cuando hay interrupción en la señal de datos, esto ocurre cuando el usuario usa un módem muy lento o hay mucho tráfico en la red. Real Audio desarrolla una mejora en su formato (RealSystemG2), que incrementa la frecuencia de audio en un 80% logrando en módem de 28,8 Kbps una mejora en la calidad del audio. El problema surge en el almacenaje pues producirá archivos demasiado grandes sobretodo para el envió por correo electrónico” [19].

5.6.3. AU

Es un formato de Sun Microsystems muy común encontrado en Internet y en plataformas Unix/Linux. Usa codificación u-Law (ver siguiente capitulo) especial para voz. Por lo general son de 8 bit y poseen menor calidad que otros formatos. “El programa mas usado para este tipo de archivo es WaveForm Hold and Modify, que soporta muchos formatos, conversiones entre ellos y funciones de edición y corrección de archivo” [19].

5.6.4. AIFF (Audio Interchange File Format)

“Formato de sonido muy simple y popular en Internet, es un formato originario para Macintoch parecido al wav por su tamaño, también puede ser usado en otras plataformas, requiere los mismos programas que el formato AU, anteriormente mencionado” [19].

6. COMPRESIÓN DE SONIDO

Capitulos tomado del sitio Web HispaMp3 [21].

6.1. INTRODUCCION

Las ondas sonoras poseen las características propias y estudiables de las ondas en general, tales como reflexión, refracción y difracción. Al tratarse de una onda continua, se requiere un proceso de digitalización para representarla como una serie de números. Actualmente, la mayoría de las operaciones realizadas sobre señales de sonido son digitales, pues tanto el almacenamiento como el procesado y transmisión de la señal en forma digital ofrece ventajas muy significativas sobre los métodos analógicos. La tecnología digital es más avanzada y ofrece mayores posibilidades, menor sensibilidad al ruido en la transmisión y capacidad de incluir códigos de protección frente a errores, así como encriptación. Con los mecanismos de decodificación adecuados, además, se pueden tratar simultáneamente señales de diferentes tipos transmitidas por un mismo canal. La desventaja principal de la señal digital es que requiere un ancho de banda mucho mayor que el de la señal analógica, de ahí que se realice un exhaustivo estudio en lo referente a la compresión de datos, algunas de cuyas técnicas serán el centro de estudio de éste capitulo.

6.1.1. Codificación y Compresión

Antes de describir los sistemas de codificación y compresión, se hará un breve análisis de la percepción auditiva del ser humano, para comprender por qué una cantidad significativa de la información que proporciona el PCM puede desecharse. El centro de la cuestión, desde un punto de vista no tan profundo, se basa en un fenómeno conocido como enmascaramiento.

El oído humano percibe un rango de frecuencias entre 20 Hz. y 20 Khz. En primer lugar, la sensibilidad es mayor en la zona alrededor de los 2-4 Khz., de forma que el sonido resulta más difícilmente audible en cuanto más cercano esté a los extremos de la escala. En segundo lugar está el enmascaramiento, cuyas propiedades utilizan exhaustivamente los algoritmos más interesantes: cuando la componente a cierta frecuencia de una señal tiene una energía elevada, el oído no puede percibir componentes de menor energía en frecuencias cercanas, tanto inferiores como superiores. A una cierta distancia de la frecuencia enmascaradora, el efecto se reduce tanto que resulta despreciable; el rango de frecuencias en las que se produce el fenómeno se denomina banda crítica (critical band). Las componentes que pertenecen a la misma banda crítica se influyen mutuamente y no afectan ni se ven afectadas por las que aparecen fuera de ella. La amplitud de la banda crítica es diferente según la frecuencia en que esté situada y viene dada

por unos determinados datos que demuestran que es mayor con la frecuencia. Hay que señalar que estos datos se obtienen por experimentos psicoacústicos que se realizan con expertos entrenados en percepción sonora, dando origen con sus impresiones a los modelos psicoacústicos.

El descrito, es el llamado enmascaramiento simultáneo o en frecuencia. Existe, asimismo, el denominado enmascaramiento asimultáneo o en el tiempo, así como otros fenómenos de la audición que no resultan relevantes en este punto. Por ahora, es importante la idea de que ciertas componentes en frecuencia de la señal admiten un mayor ruido del que generalmente se considera tolerable y, por tanto, requieren menos bits para ser codificadas si se dota al codificador de los algoritmos adecuados para resolver máscaras.

La digitalización de la señal mediante PCM es la forma más simple de codificación de la señal, y es la que utilizan tanto los CD como los sistemas DAT. Como toda digitalización, añade ruido a la señal, generalmente indeseable. Como se ha visto, cuantos menos bits se utilicen en el muestreo y la cuantización, mayor será el error al aceptar valores discretos para la señal continua, esto es, mayor será el ruido. Para evitar que el ruido alcance un nivel excesivo hay que emplear un gran número de bits, de forma que a 44'1 Khz. y utilizando 16 bits para cuantizar la señal, uno de los dos canales de un CD produce más de 700 kilobits por segundo (kbps). Como se verá, gran parte de esta información es innecesaria y ocupa un ancho de banda que podría liberarse, a costa de aumentar la complejidad del sistema decodificador e incurrir en cierta pérdida de calidad. El compromiso entre ancho de banda, complejidad y calidad es el que produce los diferentes estándares del mercado.

La siguiente tabla muestra una comparación de formatos de calidad de audio:

Tabla 16: Comparación de formatos de calidad de audio:

Calidad	Muestreo	Bits/muestra	Modo	Tasa de bits	Frecuencia
Teléfono	8 Khz	8	Mono	64 kbps	200-3400 Hz
Radio AM	11'025 Khz	8	Mono	88 kbps	
Radio FM	22'050 Khz	16	Estéreo	705'6 kbps	
CD	44'1 Khz	16	Estéreo	1411'2 kbps	20-20000 Hz
DAT	48 Khz	16	Estéreo	1536 kbps	20-20000 Hz

Un modo mejor de codificar la señal es mediante PCM no-lineal o cuantización logarítmica, que como se mencionará posteriormente, consiste en dividir el eje de la amplitud de tal forma que los escalones sean mayores cuanta más energía tiene la señal, con lo que se consigue una relación señal/ruido igual o mejor con menos bits. Con este método se puede reducir el canal de CD audio a 350 kbps, lo cual

evidentemente es una mejora sustancial, aunque puede reducirse mucho más. Otros sistemas similares llevan a la cuantización adaptativa (APCM), diferencial (DPCM) y la mezcla de ambas, ADPCM. Así prosigue la reducción del ancho de banda, pero sin llegar a los niveles que proporciona el tener en cuenta los efectos del enmascaramiento.

6.1.2. Codificación sub-banda (SBC)

La codificación sub-banda o SBC (sub-band coding) es un método potente y flexible para codificar señales de audio eficientemente. A diferencia de los métodos específicos para ciertas fuentes, el SBC puede codificar cualquier señal de audio sin importar su origen, ya sea voz, música o sonido de tipos variados. El estándar MPEG Audio es el ejemplo más popular de SBC, y lo analizaremos posteriormente en detalle.

El principio básico del SBC es la limitación del ancho de banda por descarte de información en frecuencias enmascaradas. El resultado simplemente no es el mismo que el original, pero si el proceso se realiza correctamente, el oído humano no percibe la diferencia. Existe por tanto un codificador y un decodificador que participan en el tratamiento de la señal.

La mayoría de los codificadores SBC utilizan el mismo esquema. Primero, un filtro o un banco de ellos, o algún otro mecanismo descomponen la señal de entrada en varias subbandas. A continuación se aplica un modelo psicoacústico que analiza tanto las bandas como la señal y determina los niveles de enmascaramiento utilizando los datos psicoacústicos de que dispone. Considerando estos niveles de enmascaramiento se cuantizan y codifican las muestras de cada banda: si en una frecuencia dentro de la banda hay una componente por debajo de dicho nivel, se desecha. Si lo supera, se calculan los bits necesarios para cuantizarla y se codifica. Por último se agrupan los datos según el estándar correspondiente que estén utilizando codificador y decodificador, de manera que éste pueda descifrar los bits que le llegan de aquél y recomponer la señal.

La decodificación es mucho más sencilla, ya que no hay que aplicar ningún modelo psicoacústico. Simplemente se analizan los datos y se recomponen las bandas y sus muestras correspondientes.

En los últimos diez años la mayoría de las principales compañías de la industria del audio han desarrollado sistemas SBC. A finales de los años ochenta, un grupo de estandarización del ISO llamado Motion Picture Experts Group (MPEG) comenzó a desarrollar los estándares para la codificación tanto de audio como de vídeo.

6.2. EL ESTÁNDAR MPEG AUDIO

A continuación se tratará de explicar brevemente que se esconde tras un MP3 y en que se basan sus capacidades. Para saber como funciona **no** es necesario (ni es el objeto de este marco teórico) llegar a las matemáticas profundas del modelo psicoacústico solo nos basta con entender algunos conceptos relativamente sencillos. Un MP3 es un sistema de compresión de audio con el cual se puede almacenar música con calidad CD en 1/12 del espacio original. Las siglas MP3 corresponden a la abreviación MPEG 1 layer 3. Es un algoritmo de codificación perceptual desarrollado por el consorcio MPEG (Motion Picture Expert Group) junto con el Instituto Tecnológico Fraunhofer que finalmente se ha estandarizado como norma ISO-MPEG Audio Layer 3 (IS 11172-3 e IS 13818-3) y que viene a ser un avance importante sobre los anteriores desarrollos (Layer 1 y Layer 2).

El estándar MPEG Audio contempla tres niveles diferentes de codificación-decodificación de la señal de audio, de los cuales sólo el primero está totalmente terminado. Los otros dos son aplicables, y de hecho se utilizan habitualmente, pero siguen abiertos a ampliaciones.

Estos tres niveles son:

- MPEG-1: "Codificación de imágenes en movimiento y audio asociado para medios de almacenamiento digital hasta 1’5 Mbit/s"
- MPEG-2: "Codificación genérica de imágenes en movimiento e información de audio asociada"
- MPEG-3: la planificación original contemplaba su aplicación a sistemas HDTV; finalmente fue incluido dentro de MPEG-2.
- MPEG-4: "Codificación de objetos audiovisuales"

A su vez, MPEG-1 describe tres esquemas de codificación de audio denominados esquema (layer)-1, esquema-2 y esquema-3. Del primero al tercero aumentan tanto la complejidad del codificador como la calidad del sonido. Los tres son compatibles jerárquicamente, esto es, el decodificador esquema-i es capaz de interpretar información producida por un codificador esquema-i y todos los niveles por debajo de i. Así, un decodificador esquema-3 acepta los tres niveles de codificación, mientras el esquema-2 sólo acepta el 1 y el 2.

6.2.1. MPEG 1 Layer 3 de forma general

El hecho de que haya sido adoptado como una norma ISO es más importante de lo que cabría suponer. Las normas ISO definen muchos estándares del mercado y tienen peso frente a la industria. Además eso habilita a las personas que quieran desarrollar aplicaciones o cualquier otra cosa dado que tiene a su alcance el funcionamiento del sistema. Esta tecnología no es nueva, realmente ya lleva

desarrollándose más de 10 años, lo que ocurre es que ahora es el momento en el que la velocidad de proceso de los ordenadores la han hecho asequible para el usuario medio.

6.2.1.1. Codificación Perceptual y Oído Humano

El sistema de codificación perceptual es un sistema de compresión con pérdida, esto quiere decir que el sonido original y el comprimido no son exactamente iguales. Estas pérdidas responden al funcionamiento del oído humano, así aunque los sonidos no son iguales si los percibimos como si lo fuesen.

Como se dijo anteriormente, el rango de frecuencias que percibe el oído humano esta aproximadamente entre los 20Hz y los 20kHz siendo más sensible entre los 2Hz y 4Hz. Además cuando tenemos una señal de un volumen alto en una frecuencia y otra de un volumen más bajo en una frecuencia cercana esta queda "tapada" por la anterior. Esto es lo que se llama efecto enmascaramiento.

Así pues de lo que se trata es de aprovechar los "defectos" del oído humano para desechar todo aquello que realmente no vamos a oír. Por supuesto cada uno tiene su oído y por eso para probar el éxito de estos sistemas se utilizan métodos estadísticos.

6.2.1.2. Codificación de Sub-Bandas

Para aprovechar estas características se utiliza el sistema de Sub-Bandas. Recordando lo dicho anteriormente, en este proceso la señal original se descompone en subbandas mediante un banco de filtros o algún método parecido. Estas subbandas son comparadas con el original mediante el modelo psicoacústico que determina que bandas son importantes cuales no y cuales pueden ser eliminadas. Dependiendo del bitrate (Rata de Bits) a codificar, éste proceso eliminara más o menos datos siguiendo el modelo psicoacústico hasta lograr la compresión necesaria. Luego se cuantifican y codifican las subbandas restantes y el resultado es finalmente comprimido mediante un algoritmo Standard Huffman o LZW.

Dentro del formato MP3 se puede comprimir con distinto ancho de banda, modo y bitrate, obteniendo distintas calidades, según el objetivo para el cual se desee utilizar ese sonido.

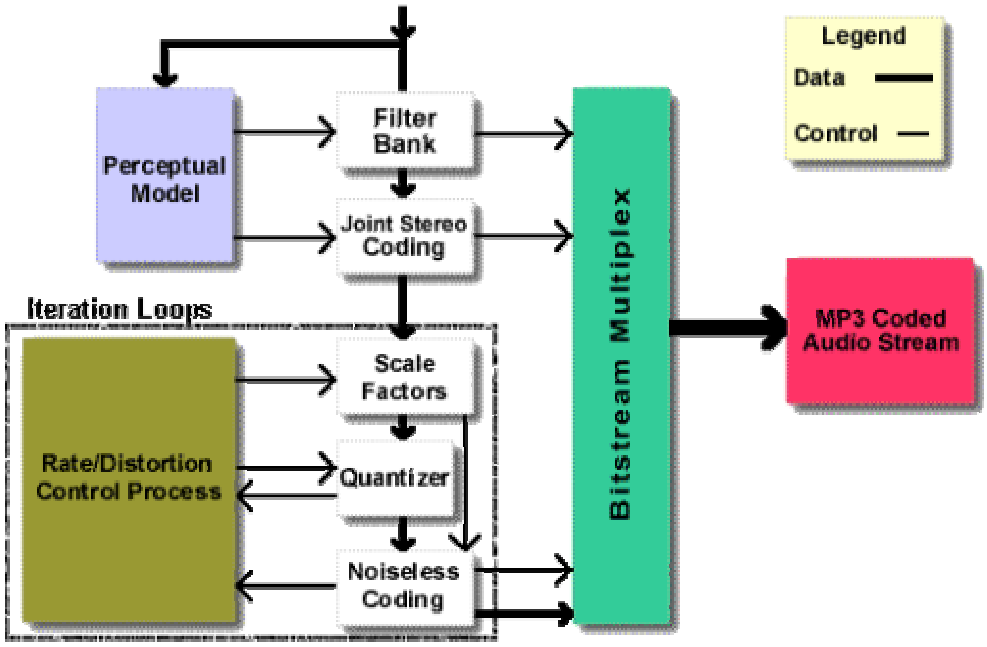
Tabla 17: Relaciones obtenidas variando los parámetros de codificación Sub-Bandas [28].

calidad del sonido	ancho de banda	modo	Bitrate	ratio de compresión
Sonido telefónico	2.5 kHz	mono	8 kbps	96:1
Mejor que onda corta	4.5 kHz	mono	16 kbps	48:1
mejor que radio AM	7.5 kHz	mono	32 kbps	24:1
similar a radio FM	11 kHz	estéreo	56...64 kbps	26...24:1
cercano al CD	15 kHz	estéreo	96 kbps	16:1
CD	>15 kHz	estéreo	112..128kbps	14..12:1

Tabla tomada del Instituto Tecnológico Fraunhofer [27]

El siguiente esquema nos permite ver de forma general el proceso:

Figura 15: Esquema general de codificación MP3.



En un disco compacto se tiene una onda de 44.1kHz 16bit estéreo eso significa aproximadamente 1400Kbps (44100 x 16 x 2 bits por segundo). Codificándolo por ejemplo a un MP3 de 128kbps se logra una reducción en torno al 1/12 del espacio inicial.

También se puede optar por compresiones a mayor bitrate llegando a 192 o incluso 256kbps. Pero el más popular es el de 128kbps con el que se consigue una calidad excelente con una compresión sobresaliente.

6.2.1.3. Compresión a MP3 con herramientas existentes

"Hay que elegir un programa y un bitrate. El bitrate (los kbits por segundo) a utilizar estará entre los 112kbps y los 256kbps y normalmente se utiliza 128kbps debido a que es el que ofrece mejor relación calidad/compresión. Con esa cifra llegaremos a reducir en 1/12 el tamaño de la pista" [22]. Las anteriores recomendaciones toman validez para comprimir música, en el caso de voz, el bitrate mínimo puede ser de 16 kbps.

"Hoy en día algunas personas están empezando a trabajar a 256kbps, con este bitrate solo se consigue una reducción aproximada de 1/6. La ventaja es que así es IMPOSIBLE diferenciar el original del comprimido. Tal vez la llegada del DVD quite importancia al tamaño de los archivos. Últimamente los compresores permiten lo que se denomina VBR (Variable BitRate) donde se pierde el concepto de flujo constante en el cual lo importante es alcanzar un nivel de compresión independientemente de la complejidad de ese fragmento de audio. Con VBR lo importante pasa a ser la calidad del resultado y el compresor asignará automáticamente un bitrate más elevado a las partes donde exista mayor complejidad" [22].

"El formato en el que se obtiene el MP3: Hay compresores que permiten como formato de salida el WAV (WAV layer 3). Evidentemente se trata realmente de un MP3 con la extensión WAV y con la cabecera de éste" [22]. Lo único útil es que ese WAV lo podremos reproducir con cualquier aplicación (reproductor multimedia, etc.) y que podemos hacer uso de la estructura RIFF del wave para personalizar su contenido. Para poder escuchar dicho Wav, es necesario tener instalado el CODEC que se encarga de que el sistema operativo (Windows 95 en adelante) sepa "traducirlo". Para transformar un WAV-Layer3 a MP3 no bastara con renombrarlo, debemos quitar la cabecera de este.

La lista de posibles programas para comprimir es muy extensa, se nombran a continuación los mejores o más representativos. Hay muchos otros que, siendo programas diferentes utilizan los mismos códigos o librerías que alguno de los que se nombran en [22]:

- Producer 2.01 (W95/Comercial): El mejor sin duda por su relación velocidad/calidad. Esta desarrollado por el Instituto Tecnológico Franhofer y comercializado por Opticon. Y eso supone que trae los últimos avances. Esta versión además permite proceso por lotes y llegar hasta los 256kbps.

- MP3Compressor (W95/ilegal): Es simplemente un shell (ilegal) de la anterior versión del Producer, pero sigue siendo muy usado debido a su comodidad.

- BladeEnc (W95/freeware): Sin duda una opción a tener en cuenta, no solo por su calidad sino también porque es totalmente gratis para usos no comerciales. Además se puede encontrar como rutina dll lo que ha hecho que sea la combinación perfecta con los extractores más populares.

- MP3Enc (DOS/Shareware): Heredero del LameEnc es por ahora el que mayor calidad puede ofrecer. Pero a costa de un elevadísimo tiempo de compresión.

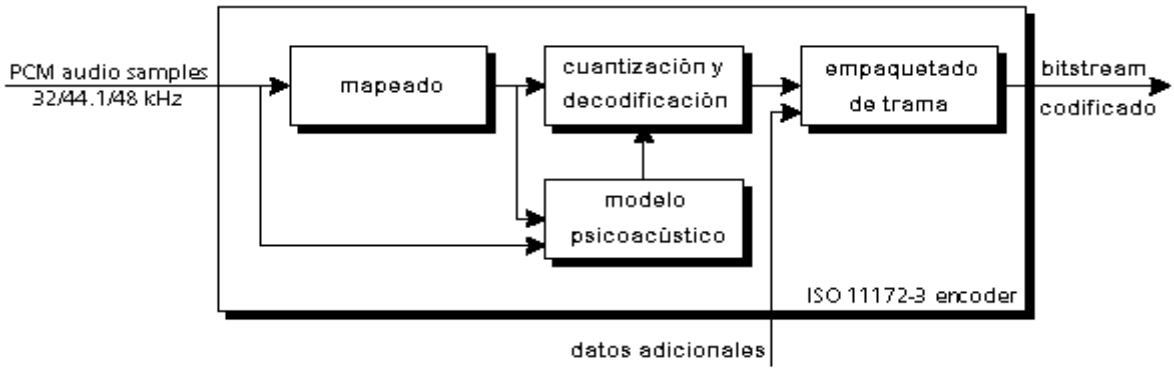
- Xing (W95/Shareware): Sin duda el más rápido. Aunque probablemente el que peor calidad ofrece.

6.2.2. MPEG-1 Según la norma ISO

Este es el sistema que describe la norma ISO en lo referente al sistema MPEG-1:

Codificación: el codificador procesa la señal de audio digital y produce el bitstream empaquetado para su almacenamiento y/o transmisión. El algoritmo de codificación no está determinado, y puede utilizar enmascaramiento, cuantización variable y escalado. Sin embargo, debe ajustarse a las especificaciones del decodificador.

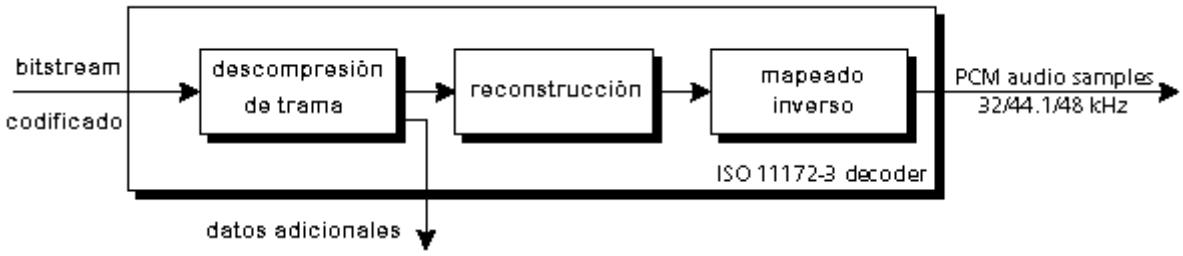
Figura 16: Esquema general ISO MPEG Encoder.



Las muestras se introducen en el codificador y a continuación el mapeador crea una representación filtrada y submuestreada de la señal de entrada. Las muestras mapeadas se denominan tanto muestras de subbanda (esquemas 1 y 2) como muestras de subbanda transformadas (esquema 3). El modelo psicoacústico crea una serie de datos (dependiendo de la implementación del codificador) que sirven para controlar la cuantización y codificación. Este último bloque crea a su vez su propia serie de datos, de nuevo dependiendo de la implementación. Por último, el bloque de empaquetamiento de trama se encarga de agrupar como corresponde todos los datos, pudiendo añadir algunos más, llamados datos adicionales, como por ejemplo CRC (Chequeo) o información del usuario.

Decodificación: el decodificador debe procesar el bitstream para reconstruir la señal de audio digital. La especificación de este elemento sí esta totalmente definida y debe seguirse en todos sus puntos. La figura ilustra el esquema del decodificador.

Figura 17: Esquema general ISO MPEG Decoder



Los datos del bitstream son desempaquetados para recuperar las diversas partes de la información. El bloque de reconstrucción recompone la versión cuantizada de la serie de muestras mapeadas. El mapeador inverso transforma estas muestras de nuevo a **PCM**.

Esquemas (Layers):

1. Incluye la división del mapeado básico de la señal de audio digital en 32 subbandas, segmentación para el formateo de los datos, modelo psicoacústico y cuantización fija. El retraso mínimo teórico es de 19 ms.

2. Incluye codificación adicional, factores de escala y diferente composición de trama. El retraso mínimo teórico es de 35 ms.

3. Incluye incremento de la resolución en frecuencia, basado en el uso de un banco de filtros híbrido. Cuantización no uniforme, segmentación adaptativa y codificación entrópica de los valores cuantizados. El retraso mínimo teórico es de 59 ms.

Tabla 18: Esquemas o Layers de MPEG-1.

Esquema	Objetivo	Compresión	Calidad 64 kbps	Calidad 128 kbps	Retardo
Esquema-1	192 kbps	4 a 1			19 ms
Esquema-2	128 kbps	6 a 1	2'1 a 2'6	Más de 4	35 ms
Esquema-3	64 kbps	12 a 1	3'6 a 3'8	Más de 4	59 ms

La calidad viene dada del 1 al 5, siendo el 5 la superior. Hay que señalar que pese a los números de la norma ISO, el retraso típico acostumbra a ser tres veces mayor en la práctica.

***Modos**:*

Hay cuatro modos de funcionamiento para cualquiera de estos tres esquemas:

a. Single channel o canal único: una señal en un bitstream.

b. Dual channel o canal doble: dos señales independientes en un mismo bitstream.

c. Stereo: como el anterior, perteneciendo las señales al canal izquierdo y derecho de una señal estéreo original.

d. Joint Stereo: como el anterior, aprovechando ciertas características del estéreo como irrelevancia y redundancia de datos para reducir la tasa de bits.

6.2.3. MPEG-1 en detalle

Tras haber visto la introducción que figura en los documentos ISO, podemos pasar a analizar en detalle el funcionamiento del sistema. A continuación se enfatizara en las características y diferencias entre los tres esquemas de MPEG-1.

Especificaciones:

1. El banco de filtros: realiza el mapeado del dominio del tiempo al de la frecuencia. Existen dos tipos: el polifase y el híbrido polifase/MDCT. Estos bancos proporcionan tanto la separación en frecuencia para el codificador como los filtros de reconstrucción del decodificador. Las muestras de salida del banco están cuantizadas.

2. El modelo psicoacústico: calcula el nivel a partir del cual el ruido comienza a ser perceptible, para cada banda. Este nivel se utiliza en el bloque de asignación de bit/ruido para determinar la cuantización y sus niveles. De nuevo, se utilizan dos diferentes. En ambos, los datos de salida forman el SMR (signal-to-mask ratio) para cada banda o grupo de bandas.

3. Asignación de bit/ruido: examina tanto las muestras de salida del banco de filtros como el SMR proporcionado por el modelo psicoacústico, y ajusta la asignación de bit o ruido, según el esquema utilizado, para satisfacer simultáneamente los requisitos de tasa de bits y de enmascaramiento.

4. El formateador de bitstream: toma las muestras cuantizadas del banco de filtros, junto a los datos de asignación de bit/ruido y otra información lateral para formar la trama.

Los tres esquemas utilizan diferentes algoritmos para cumplir estas especificaciones:

Esquema I:

- El mapeado tiempo-frecuencia se realiza con un banco de filtros polifase con 32 subbandas. Los filtros polifase consisten en un conjunto de filtros con el mismo ancho de banda con interrelaciones de fase especiales que ofrecen una implementación eficiente del filtro subbanda. Se denomina filtro subbanda al que cubre todo el rango de frecuencias deseado. En general, los filtros polifase combinan una baja complejidad de computación con un diseño flexible y múltiples opciones de implementación.

- El modelo psicoacústico utiliza una FFT (Fast Fourier Transform) de 512 puntos para obtener información espectral detallada de la señal. El resultado de la aplicación de la FFT se utiliza para determinar los enmascaramientos en la señal, cada uno de los cuales produce un nivel de enmascaramiento, según la frecuencia, intensidad y tono. Para cada subbanda, los niveles individuales se combinan y forman uno global, que se compara con el máximo nivel de señal en la banda, produciendo el SMR que se introduce en el cuantizador.

- El bloque de cuantización y codificación examina las muestras de cada subbanda, encuentra el máximo valor absoluto y lo cuantiza con 6 bits. Este valor es el factor de escala de la subbanda. A continuación se determina la asignación de bits para cada subbanda minimizando el NMR (noise-to-mask ratio) total. Es posible que algunas subbandas con un gran enmascaramiento terminen con cero bits, es decir, no se codificará ninguna

muestra. Por último las muestras de subbanda se cuantizan linealmente según el número de bits asignados a dicha subbanda concreta.

- El trabajo del empaquetador de trama es sencillo. La trama, según la definición ISO, es la menor parte del bitstream decodificable por sí misma. Cada trama empieza con una cabecera para sincronización y diferenciación, así como 16 bits opcionales de CRC para detección y corrección de errores. Se emplean, para cada subbanda, 4 bits para describir la asignación de bits y otros 6 para el factor de escala. El resto de bits en la trama se utilizan para la información de samples, 384 en total, y con la opción de añadir cierta información adicional. A 48 Khz, cada trama lleva 8 ms de sonido.

Esquema II:

- El mapeado de tiempo-frecuencia es idéntico al del esquema I.

- El modelo psicoacústico es similar, salvo que utiliza una FFT de 1024 puntos para obtener mayor resolución espectral. En los demás aspectos, es idéntico.

- El bloque de cuantización y codificación también es similar, generando factores de escala de 6 bits para cada subbanda. Sin embargo, las tramas del esquema II son tres veces más largas que las del esquema I, de forma que se concede a cada subbanda tres factores de escala, y el codificador utiliza uno, dos o los tres, según la diferencia que haya entre ellos. La asignación de bits es similar a la del esquema I.

- El formateador de trama: la definición ISO de trama es la misma que en el punto anterior. Utiliza la misma cabecera y estructura de CRC que el esquema I. El número de bits que utilizan para describir la asignación de bits varía con las subbandas: 4 bits para las inferiores, 3 para las medias y dos para las superiores, adecuándose a las bandas críticas. Los factores de escala se codifican junto a un número de dos bits que indica si se utilizan uno, dos o los tres. Las muestras de subbanda se cuantizan y a continuación se asocian en grupos de tres, llamados gránulos. Cada uno se codifica con una palabra clave, lo que permite interceptar mucha más información redundante que en el esquema I. Cada trama contiene, pues, 1152 muestras PCM. A 48 Khz. cada trama lleva 24 ms de sonido.

Esquema III:

- El esquema III es sustancialmente más complicado que los dos anteriores e incluye una serie de mejoras cuyo análisis resultaría desbordante, de manera que no se entrará en tantos detalles. Su diagrama de flujos es conceptualmente semejante al visto para los otros dos esquemas, salvo que se realizan múltiples iteraciones para procesar los datos con el mayor nivel de calidad en un cierto tiempo, lo cual complica su diseño hasta el punto de que los diagramas ISO ocupan decenas de páginas.

- El mapeado de tiempo-frecuencia añade un nuevo banco de filtros, el DCT (Discrete Cosine Transform), que con el polifase forman el denominado filtro híbrido. Proporciona una resolución en frecuencia variable, 6x32 o 18x32 subbandas, ajustándose mucho mejor a las bandas críticas de las diferentes frecuencias.

- El modelo psicoacústico es una modificación del empleado en el esquema II, y utiliza un método denominado predicción polinómica. Incluye los efectos del enmascaramiento temporal.

- El bloque de cuantización y codificación también emplea algoritmos muy sofisticados que permiten tramas de longitud variable. La gran diferencia con los otros dos esquemas es que la variable controlada es el ruido, a través de bucles iterativos que lo reducen al mínimo posible en cada paso.

- El formateador de trama: la definición de trama para este esquema según ISO varía respecto de la de los niveles anteriores: "mínima parte del bitstream decodificable mediante el uso de información principal adquirida previamente". Las tramas contienen información de 1152 muestras y empiezan con la misma cabecera de sincronización y diferenciación, pero la información perteneciente a una misma trama no se encuentra generalmente entre dos cabeceras. La longitud de la trama puede variarse en caso de necesidad. Además de tratar con esta información, el esquema III incluye codificación Huffman de longitud variable, un método de codificación entrópica que sin pérdida de información elimina redundancia. Los métodos de longitud variable se caracterizan, en general, por asignar palabras cortas a los eventos más frecuentes, dejando las largas para los más infrecuentes.

6.2.4. Formato del Archivo MP3

Un archivo de audio MPEG esta formado por pequeñas partes llamadas frames. Generalmente estos frames son independientes unos de otros. Cada frame contiene su propia cabecera e información de sonido. El archivo mp3 como tal, no tiene una cabecera, por tanto se puede cortar cualquier frame y escucharlo correctamente. Sin embargo, en el Layer III debido a la capacidad VBR (variable bitrate) lo anterior no es 100% correcto.

La información acerca del mp3 como tal (bitrate, frecuencia de muestreo, número de canales, etc.) se lee del primer frame y se asume igual para el resto. Pero como se dijo anteriormente si el archivo fue comprimido con la tecnología VBR éste método no funciona.

Aparte de los frames, un archivo .MP3 puede contener Tags de información que describen al archivo de sonido. MPEG 1 Layer 3 soporta ID3v1 e ID3v2. Estos tags contienen información como: Titulo, Artista, álbum, año, género, compositor, etc. Relativos a discos comerciales.

6.2.4.1. Cabecera de un Frame MPEG Audio

La cabecera de un frame consta de 32 bits (4 bytes) los cuales se encuentran al principio del frame. Los primeros 11 bits del FH (Frame Header, Cabecera de Frame) contienen la sincronización del frame. Es decir, todos los 11 bits deben estar establecidos a 1, entonces, para encontrar los frames de un MP3 hay que buscar 11 bits en 1. La siguiente tabla describe los demás bits:

Tabla 19: Estructura de un Frame MP3 [29 - Traducido].

AAAAAAAA AABBCCD EEEEFFGH IIJJKLMM

Letra	Tamaño (bits)	Posición (bits)	Descripción
A	11	(31-21)	Sincronización del Frame (todos los bits en 1)
B	2	(20,19)	ID de la Versión MPEG Audio 00 - MPEG Versión 2.5 01 – Reservado 10 - MPEG Versión 2 (ISO/IEC 13818-3) 11 - MPEG Versión 1 (ISO/IEC 11172-3)

<table>
<tr><td>C</td><td>2</td><td>(18,17)</td><td>Layer (esquema):
00 – reservado
01 - Layer III
10 - Layer II
11 - Layer I</td></tr>
<tr><td>D</td><td>1</td><td>(16)</td><td>Bit de Protección:
0 – Protegido por CRC (16bit CRC seguidos del FH)
1 – No Protegido

Nota: La protección (CRC, Chequeo de Redundancia Ciclica) se usa para ver que los datos del FH no tienen errores.</td></tr>
<tr><td>E</td><td>4</td><td>(15,12)</td><td>Indicador de Bitrate:

<table>
<tr><td>bits</td><td>V1,L1</td><td>V1,L2</td><td>V1,L3</td><td>V2,L1</td><td>V2, L2 & L3</td></tr>
<tr><td>0000</td><td>libre</td><td>libre</td><td>libre</td><td>libre</td><td>Libre</td></tr>
<tr><td>0001</td><td>32</td><td>32</td><td>32</td><td>32</td><td>8</td></tr>
<tr><td>0010</td><td>64</td><td>48</td><td>40</td><td>48</td><td>16</td></tr>
<tr><td>0011</td><td>96</td><td>56</td><td>48</td><td>56</td><td>24</td></tr>
<tr><td>0100</td><td>128</td><td>64</td><td>56</td><td>64</td><td>32</td></tr>
<tr><td>0101</td><td>160</td><td>80</td><td>64</td><td>80</td><td>40</td></tr>
<tr><td>0110</td><td>192</td><td>96</td><td>80</td><td>96</td><td>48</td></tr>
<tr><td>0111</td><td>224</td><td>112</td><td>96</td><td>112</td><td>56</td></tr>
<tr><td>1000</td><td>256</td><td>128</td><td>112</td><td>128</td><td>64</td></tr>
<tr><td>1001</td><td>288</td><td>160</td><td>128</td><td>144</td><td>80</td></tr>
<tr><td>1010</td><td>320</td><td>192</td><td>160</td><td>160</td><td>96</td></tr>
<tr><td>1011</td><td>352</td><td>224</td><td>192</td><td>176</td><td>112</td></tr>
<tr><td>1100</td><td>384</td><td>256</td><td>224</td><td>192</td><td>128</td></tr>
<tr><td>1101</td><td>416</td><td>320</td><td>256</td><td>224</td><td>144</td></tr>
<tr><td>1110</td><td>448</td><td>384</td><td>320</td><td>256</td><td>160</td></tr>
<tr><td>1111</td><td>malo</td><td>malo</td><td>malo</td><td>malo</td><td>malo</td></tr>
</table>

Notas: Todos los valores están en Kbps

V1 - MPEG Versión 1
V2 - MPEG Versión 2 and Versión 2.5
L1 - Layer I
L2 - Layer II</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td>L3 - Layer III

Modos aceptados dependiendo del número de canales:

<table>
<tr><td>bitrate</td><td>Modos Permitidos</td></tr>
<tr><td>free</td><td>Todos</td></tr>
<tr><td>32</td><td>Mono</td></tr>
<tr><td>48</td><td>Mono</td></tr>
<tr><td>56</td><td>Mono</td></tr>
<tr><td>64</td><td>Todos</td></tr>
<tr><td>80</td><td>Mono</td></tr>
<tr><td>96</td><td>Todos</td></tr>
<tr><td>112</td><td>Todos</td></tr>
<tr><td>128</td><td>Todos</td></tr>
<tr><td>160</td><td>Todos</td></tr>
<tr><td>192</td><td>Todos</td></tr>
<tr><td>224</td><td>Join Stereo o dos canales</td></tr>
<tr><td>256</td><td>Join Stereo o dos canales</td></tr>
<tr><td>320</td><td>Join Stereo o dos canales</td></tr>
<tr><td>384</td><td>Join Stereo o dos canales</td></tr>
</table>
</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>(11,10)</td>
<td>Frecuencia de Muestreo (Valores en Hz):

<table>
<tr><td>bits</td><td>MPEG1</td><td>MPEG2</td><td>MPEG2.5</td></tr>
<tr><td>00</td><td>44100</td><td>22050</td><td>11025</td></tr>
<tr><td>01</td><td>48000</td><td>24000</td><td>12000</td></tr>
<tr><td>10</td><td>32000</td><td>16000</td><td>8000</td></tr>
<tr><td>11</td><td>reserv.</td><td>reserv.</td><td>reserv.</td></tr>
</table>
</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>(9)</td>
<td>Bit de relleno (Padding):

0 – Frame no rellenado
1 – Frame Rellenado

El Padding indica si fue necesario completar el frame con bits de relleno para ajustarlo al bitrate establecido.</td>
</tr>
</table>

			Como calcular el tamaño del Frame Primero hay que diferenciar los siguientes dos términos: Tamaño del Frame y Ancho del frame. El tamaño del frame es el número de muestras de sonido (Ver Formatos de archivos de sonido, en el Marco teórico 2) contenidas en un frame y es siempre 384 para Layer I y 1152 para Layer II y Layer III. El ancho del frame, es el espacio ocupado por el frame en forma comprimida. Este es calculado de forma diferente para layer I y, layer II y III. Además, recuerde que el tamaño del frame depende también del padding, y del bitrate. *Para Layer I use la siguiente formula:* FrameLengthInBytes = (12 * BitRate / SampleRate + Padding) * 4 *Para Layer II y III use la siguiente formula:* FrameLengthInBytes = 144 * BitRate / SampleRate + Padding Ejemplo: Layer III, BitRate=128000, SampleRate=441000, Padding=0 ⇨ FrameSize=417 bytes
H	1	(8)	Bit privado para aplicaciones. Puede ser usado de la forma más adecuada que una aplicación convenga.
I	2	(7,6)	Modo de Canales: 00 – Stereo 01 - Joint stereo (Stereo) 10 - Dual channel (Stereo) 11 - Single channel (Mono)
J	2	(5,4)	Extensión de modo (solo si el modo es Joint stereo)

			No aplica en nuestro caso, ya que vamos a trabajar los archivos de voz en modo mono (1 canal).
K	1	(3)	Derechos de copia (Copyright): 0 – El Audio no tiene copyright 1 - Audio con copyright
L	1	(2)	Original o copia: 0 – Copia de un archivo original 1 – Original
M	2	(1,0)	Énfasis: 00 – ninguno 01 - 50/15 ms 10 – reservado 11 - CCIT J.17

7. CODIFICACIÓN DE VOZ

"En condiciones normales de transmisión de voz, se utiliza una tasa de muestreo de 8.000 datos por segundo, codificándola en 8 bits, por lo que se necesita 64 Kbps de capacidad de transmisión. Con los métodos de compresión utilizados actualmente, se necesitan sólo 8-13 Kbps de capacidad y la señal está catalogada como buena" [30]. Con MPEG, se pueden comprimir las señales a 16 kbps, pero la calidad realmente es muy mala ya que MPEG Layer 3 no esta pensado para comprimir voz.

7.1. REPRESENTACIÓN DE LA VOZ

7.1.1. Muestras en el Tiempo

"Puesto que la voz consiste en una secuencia de sonidos elementales diferenciados, el análisis de su señal se lleva a cabo sobre una secuencia temporal de observaciones, conteniendo cada una de ellas las características de la señal de voz correspondientes a un segmento temporal determinado que son relevantes para los objetivos finales. Idealmente, estas características deberían representar un segmento de voz uniforme, es decir, un evento acústico bien definido. Sin embargo, una simple mirada a la forma de onda de la señal de voz entregada por el micrófono muestra que ésta no puede describirse simplemente como la concatenación de segmentos uniformes, y que resulta imposible determinar una frontera nítida entre sonidos, como consecuencia de la coarticulación entreexistente" [31].

"Esta es la razón que hace preferible que la voz se segmente en tramos de longitud y desplazamiento fijos, y que los intentos de representarla como una sucesión de segmentos variables no hayan dado lugar a una mejora de resultados de codificación o reconocimiento" [31].

"Así pues, se trata de representar cada tramo de la señal de voz (de longitud entre 20 y 30 milisegundos) mediante un conjunto de características que se corresponden con un modelo paramétrico de dicha señal. Dichas características, a menudo denominadas parámetros, no pretenden captar toda la complejidad de la señal acústica, sino sólo su espectro, es decir, la distribución de la energía de la señal a lo largo de la frecuencia. Y, simplificando más todavía, los parámetros representan únicamente la forma que envuelve el espectro (envolvente espectral o formantes), dejando de lado la estructura armónica que caracteriza los segmentos sonoros, es decir, los que se generan a partir del efecto de vibración de las cuerdas vocales" [31].

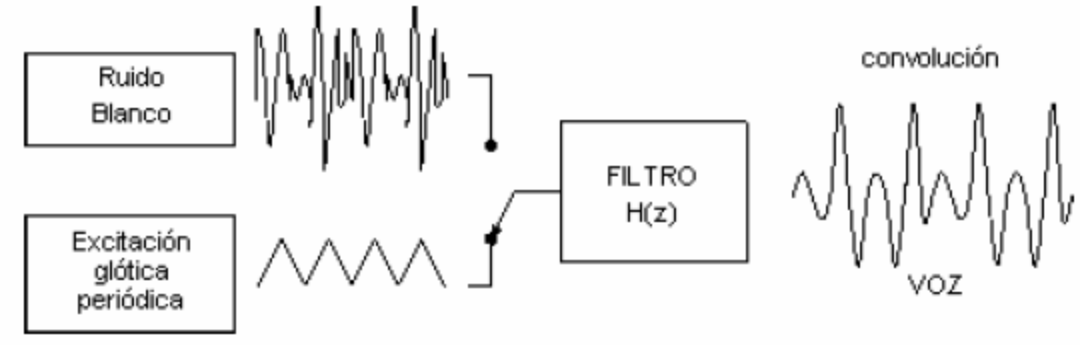
7.1.2. Modelado mediante Filtros Lineales

"Contemplando el funcionamiento del aparato fonador humano se observa ante todo el movimiento de los órganos articulatorios que dan forma a una cavidad acústica, el tracto vocal. Pero, para que se produzca el sonido es necesaria una fuente de ondas de presión del aire, conformada por la vibración de las cuerdas vocales (caso sonoro), o por una fricación o aspiración (caso sordo). Ante un observador habituado a la teoría de sistemas, este mecanismo acústico sugiere enseguida un modelo de entrada-salida, en el que la señal de la voz es la salida de un sistema lineal o filtro en cuya entrada se encuentra la fuente acústica antes mencionada" [31].

"Dicho modelo de producción de la voz basa su sencillez en la separación que realiza entre el filtro, que simula el funcionamiento del tracto vocal, el cual a su vez confiere a cada sonido su timbre característico, y la excitación o entrada, que da cuenta del tipo de fuente acústica (sorda o sonora) y, en el caso sonoro, de la frecuencia de vibración de las cuerdas vocales, denominada *frecuencia fundamental* o *tono de la voz"* [31].

"La producción de voz es modelable por sistemas matemáticos conocidos como filtros lineales, los cuales se utilizan para representar la función de transferencia del sistema H(z): la voz puede ser considerada como una señal que se genera a partir del filtrado de la excitación glótica más una señal de amplitud aleatoria de espectro uniforme, conocida como ruido blanco, ver figura" [30]:

Figura 18: Representación de la voz.



"En este modelo simplificado se considera que el filtro es puramente recurrente, es decir, que es un sistema lineal sin ceros, sólo con unos 10 polos. Puesto que la envolvente espectral de la señal de voz generada con este modelo la determina el filtro, ello equivale a suponer que sólo interesan los picos de la envolvente espectral, no sus valles" [31].

“Gracias a dicho modelo, basado en el análisis de la transmisión de las ondas planas del sonido a través de un tubo sin pérdidas de sección no uniforme, es posible el uso de una técnica eficiente de determinación de los parámetros del filtro que, fundamentada en la idea de predicción lineal óptima, constituye la base de la mayoría de sistemas de compresión de la señal de voz” [31].

7.1.3. Percepción de la voz

Tomado de [31].

Hasta ahora se ha tratado el modo cómo se produce la voz. Pero al ser la voz un medio de comunicación, interesa conocer también el mecanismo de percepción ya que, lógicamente, las características de la señal estarán en función no sólo del aparato productor sino también del receptor, el oído.

Hay dos propiedades del aparato auditivo humano que son relevantes en el desarrollo de sistemas tanto de reconocimiento del habla como de compresión y codificación de la voz. La primera es el efecto de enmascaramiento, por el que un sonido puede dejar de oírse cuando está situado frecuencialmente (o temporalmente) cerca de otro sonido de intensidad suficientemente alta. Esta propiedad ha resultado de gran valor en el desarrollo de sistemas de compresión de audio y, en particular, de voz. En efecto, a la distorsión introducida por efecto de la reducción del número de bits que representan la voz se le hace jugar el papel del sonido enmascarado. El efecto de enmascaramiento se mencionó también en el capitulo anterior.

En segundo lugar, la cóclea del oído funciona como un analizador espectral, trabajando en bandas frecuenciales no uniformes que se hacen sucesivamente más anchas a medida que crece la frecuencia (se tiene una muestra de ello en el piano, donde las teclas de sonidos agudos están más distanciadas en frecuencia que las de sonidos más graves). Pues bien, la técnica basada en subbandas y que, según se ha afirmado, es la más utilizada para representar la envolvente espectral de la voz en la representación del habla, imita de algún modo el análisis frecuencial realizado por la cóclea. Los parámetros representativos del tramo de voz que ésta técnica determina son las fracciones de energía de la señal correspondientes a unas 20 bandas frecuenciales distribuidas según una escala no uniforme denominada *mel* que, determinada con experimentos perceptivos, refleja la resolución frecuencial del oído humano.

7.1.4. Estimación de parámetros de un filtro lineal - Predicción lineal (LPC)

"Para el análisis (y síntesis) de la voz se utiliza el modelo fuente/filtro, basado en una batería de filtros lineales y causales que modelan los distintos componentes del aparato fonador humano. Para la estimación de los parámetros de este filtro se utiliza la predicción lineal denominada LPC" [30].

"El análisis de LPC se utiliza para encontrar los coeficientes que representarán la función de transferencia del filtro que modela el sistema. Si el modelo es capaz de predecir la señal con un error muy bajo, se tiene que el LPC ha sido capaz de almacenar la información necesaria de un trozo de señal como para reproducirla mediante alguna excitación. En analogía con un instrumento musical, el LPC sería un instrumento de viento que al ser soplado emite el sonido con el timbre particular del trozo de voz que representa" [30].

"El principio de un LPC es que el valor actual de una muestra de señal de voz, s(n), puede predecirse a partir de un número finito de muestras anteriores: s(n-1), ... , s(n-p), con un error asociado e(n) utilizando un filtro lineal sólo polos. Los modelos matemáticos resultantes no son del interés de éste documento, lo importante es entender al menos como funcionan" [30].

7.2. CODIFICADORES DE VOZ EXISTENTES

"Los sistemas de compresión más utilizados en la actualidad son: CELP (creado para la comunicación radial federal de Estados Unidos y utilizado actualmente como estándar de compresión de voz de la telefonía celular digital); VSELP (creado a partir de CELP por la empresa MOTOROLA y estandarizado para su uso en telefonía celular análoga) y GSM 06.10 (estándar de telefonía celular utilizado en Europa)" [30]. Según la recomendación H. 323 de la ITU (Unión Internacional de Telecomunicaciones) que se encarga de reglamentar los sistemas de comunicaciones multimedia basados en Paquetes (como TCP/IP), todo lo relacionado con audio esta en las siguientes subrecomendaciones, según [32]:

- **G. 711** Modulación por impulsos codificados (PCM) de frecuencias vocales. Ratificada en 1988.
- **G. 722.** Aspectos generales de los sistemas de transmisión digital, Codificación de Audio de 7 KHz en 64 Kbps.
- **G. 723.1** Códec de voz de doble velocidad para la transmisión en comunicaciones multimedios a 5,3 y 6,3 kbit/s
- **G. 729** CS-ACELP. Excelente codec de voz. Es el más usado para tecnologías de voz sobre IP en la actualidad.

Nota: La teoria sobre los codificadores siguientes (7.2.1 a 7.2.4) es tomada de [30]

7.2.1. CELP

Todas las técnicas de compresión de voz están basadas en dos operaciones intrínsecas:

- Eliminar la redundancia.
- Eliminar la irrelevancia.

La primera operación utiliza predicciones o transformaciones para eliminar los datos redundantes, lo cual reduce el ancho de banda necesario para la señal. La segunda operación reduce el ancho de banda realizando una cuantización de, ya sea los componentes de la predicción (o su error) o de los coeficientes de la transformación. Obteniendo una señal parecida a la original pero siempre con un grado de distorsión o error de reconstrucción.

Al aumentar la compresión, es necesario que el codificador minimice la percepción del error utilizando propiedades inherentes al habla humano. Esto quiere decir que el mismo nivel de error de la distorsión es percibido de distinta manera si es aplicado a señales de voz con distinta energía y bandas de frecuencia.

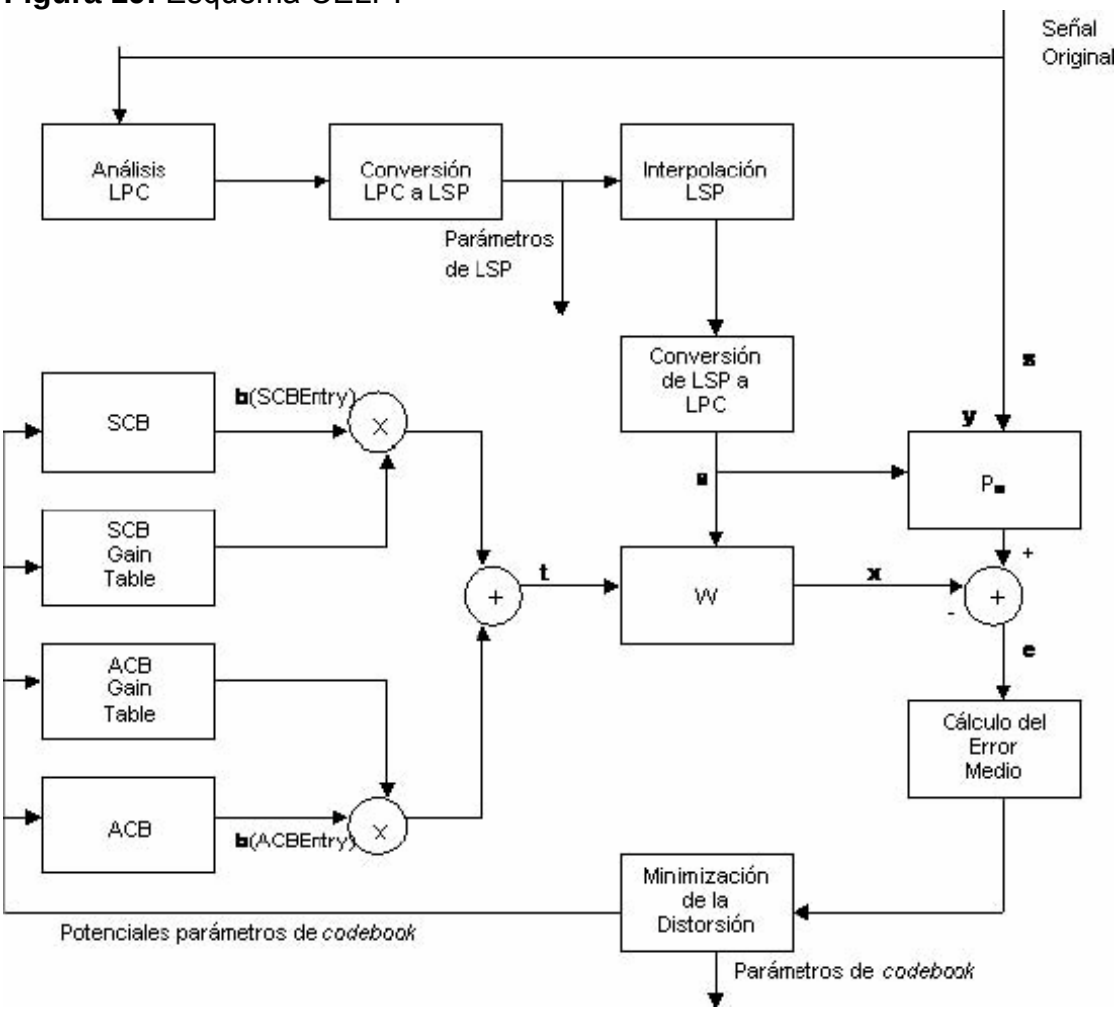
La solución de CELP (*code excited linear prediction,* codificación mediante la predicción lineal con excitación por código) a ese problema es utilizar la aproximación análisis por síntesis, donde se mide la percepción de la distorsión.

Un codebook consiste en una tabla de muestras de señal residual, conocidas como codewords, los cuales se utilizarán como excitación de los filtros. Además, un filtro llamado “de peso de percepción”, es utilizado para asegurar que la medida del error cuadrático medio refleje el error de percepción.

Al aplicar un filtro de percepción sobre la señal se mejora el rendimiento del codificador. Los formantes de alta energía disimulan mejor el ruido que las porciones de baja energía del espectro. La señal de error generada por cada paso del sintetizador es ponderada apropiadamente para mejorar este efecto de percepción. El filtro amplifica la señal de error en las regiones en que no hay formantes y lo atenúa en las que sí. De este modo, una señal de error cuya energía es concentrada en los formantes es considerada mejor que una que no.

En la práctica, los sistemas CELP emplean algoritmos rápidos de búsqueda explotando la estructura computacional de éste. Es por eso que el esquema original derivó en un nuevo esquema:

Figura 19: Esquema CELP:



El decodificador toma los parámetros codificados y utilizando el mismo esquema, pero en sentido inverso, reconstruye la señal ŝ. Además, se encarga de sincronizar la señal construida del ACB, para ello, utiliza las dos últimas muestras del subframe anterior.

7.2.2. VSELP

VSELP fue patentado por Motorola quien es el responsable del diseño y desarrollo del algoritmo. VSELP es un tipo de algoritmo CELP que codifica a 7.950 [bps], utilizando un adicional de 5.050 [bps] para control de errores y sincronización de tramas.

La diferencia entre VSELP y los codificadores comunes (CELP) radica principalmente en la estructura de sus codebooks. Mientas CELP utiliza un

stochastic codebook para realizar sus búsquedas, VSELP utiliza dos conjuntos de vectores base para generar un espacio de “vectores candidatos”. De este modo, la búsqueda en el codebook de CELP corresponde a dos búsquedas en VSELP.

Hay siete vectores base ortogonales en todo el espacio para cada búsqueda. Cada uno de ellos contiene 40 elementos. La selección de los vectores base es fundamental para una rápida búsqueda en los codebook.

Se realiza un análisis LPC para cada ventana de señal de voz y se obtienen una serie de coeficientes del filtro LPC. Estos coeficientes son expandidos en bandas de frecuencia para utilizarlos en un filtro de error perceptual.

El análisis por síntesis procede con 3 codebook. Primero, se busca en el adaptive codebook, obteniéndose “la mejor” entrada y ganancia. Esta entrada multiplicada por su factor de ganancia es ortogonalizada para el espacio de los primeros 7 vectores base. De este modo, la búsqueda en el segundo codebook puede realizarse independiente del primero.

Un nuevo espacio de 7 vectores es utilizado para la segunda búsqueda; y una nueva “mejor” entrada y ganancia se obtienen de este segundo codebook, ortogonalizándola como en la etapa anterior.

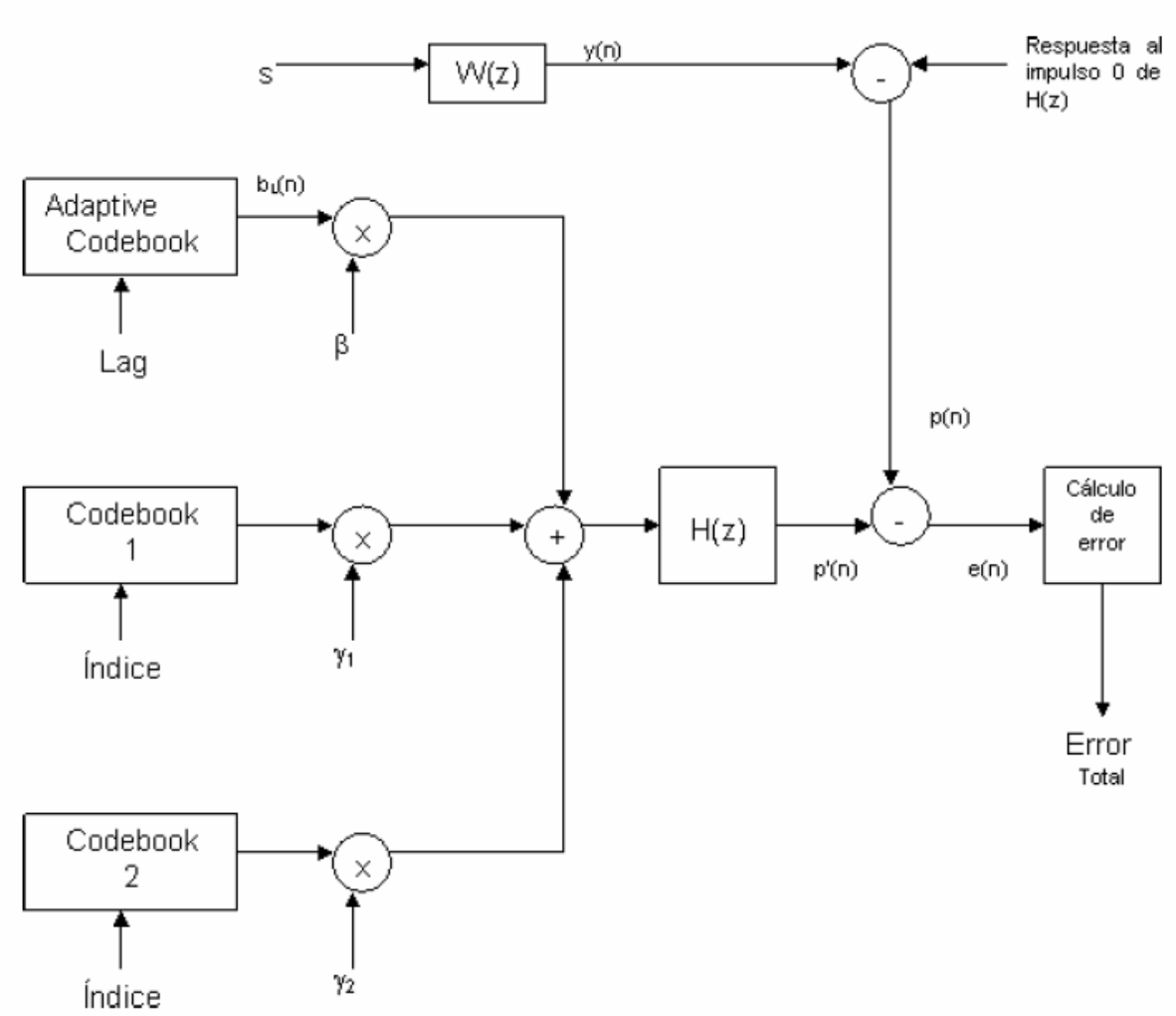
Finalmente se realiza una búsqueda en un tercer codebook. La ganancia obtenida de cada uno de los 3 codebooks se cuantiza y transmite con los 3 índices del codebook al receptor.

Los principales módulos de VSELP son:

- Análisis de LPC de orden 10.
- Predicción de largo plazo.
- Búsqueda en el adaptive codebook (pitch).
- Búsqueda de la primera base de vectores en el codebook.
- Búsqueda de la segunda base de vectores en el codebook.
- Cuantización vectorial del codebook de la ganancia.

Este algoritmo utiliza una frecuencia de muestreo de 8 [Khz], 160 muestras por frame, dividiéndola en 4 subframes de 40 muestras. El factor de expansión de banda es 0.8. La siguiente figura representa el esquema del analizador VSELP:

Figura 20: Esquema VSELP:



El decodificador toma los datos enviados desde el codificador y, al igual que todos los demás de la familia CELP, rehace la secuencia con las siguientes excepciones:

- Los coeficientes para la síntesis LPC son los “originales” (no los expandidos).
- No hay ciclo de búsqueda en base al error (close-loop).
- Hay un filtro adaptivo para la señal de salida.

7.2.3. Codificación basada en Redes Neuronales Artificiales

El sistema se compone esencialmente de cuatro partes:

- Compresor: encargado de identificar toda la información redundante en la señal para luego generar los parámetros que se van a transmitir.
- Transmisor: contiene un codificador diseñado para hacer un uso eficiente de las características estacionales de los parámetros generados en la etapa anterior.
- Receptor: recibe los parámetros codificados desde el canal de transmisión e incorpora un decodificador.
- Descompresor: sintetiza la señal de voz, a partir de los parámetros recibidos.

Los parámetros esenciales

La idea de la compresión de voz supone que existen formas básicas cuya combinación es capaz de crear un trozo de señal más complejo. Estas formas básicas se caracterizan a través de lo que se denominó parámetros esenciales.

Tanto en el transmisor como en el compresor existen arreglos gigantes (codebooks) que contienen las formas básicas (codewords) del habla. La compresión sólo consiste en encontrar la posición de los codewords que definen la señal y transmitir su ubicación en los arreglos.

Para lograr esto fueron desarrollados cinco módulos encargados de transmitir los parámetros esenciales. Estos son:

- Detector de pitch (tono). Predicción de éste utilizando redes neuronales retropropagativas.

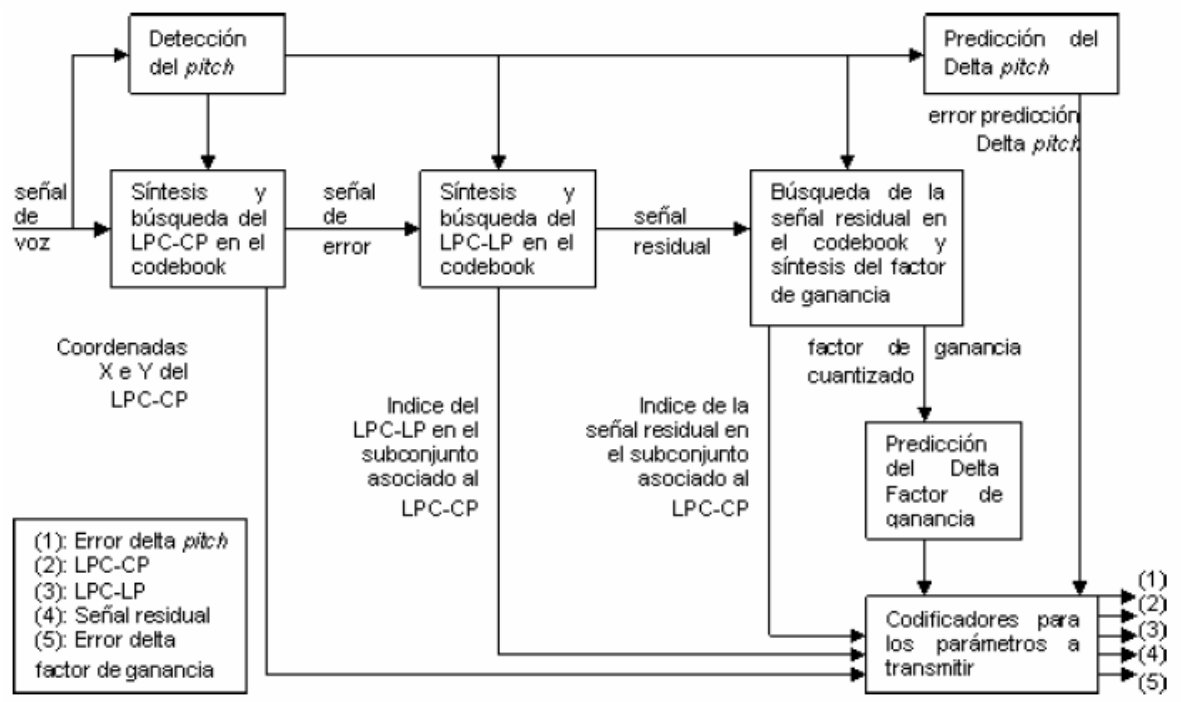
- Síntesis de LPC de corto plazo y búsqueda en el codebook. El codebook está formado por los centroides de una red neuronal autoorganizativa de Kohonnen.

- Síntesis de LPC de largo plazo y búsqueda en el codebook.

- Síntesis de señal residual y búsqueda en el codebook.

- Transmisión del factor de ganancia.

Figura 21: Esquema de Codificación basada en Redes Neuronales:



Se utiliza una cuantización de 32 niveles, normalmente codificadas en 5 bits, pero aprovechando que el factor ganancia es un parámetro que cuantizado tiene variaciones suaves en el tiempo se utiliza la transmisión de la diferencia de ganancia, codificada en 3 bits.

7.2.4. GSM 6.10 RPE-LTP

(Regular Excitation Long-Term Predictor)

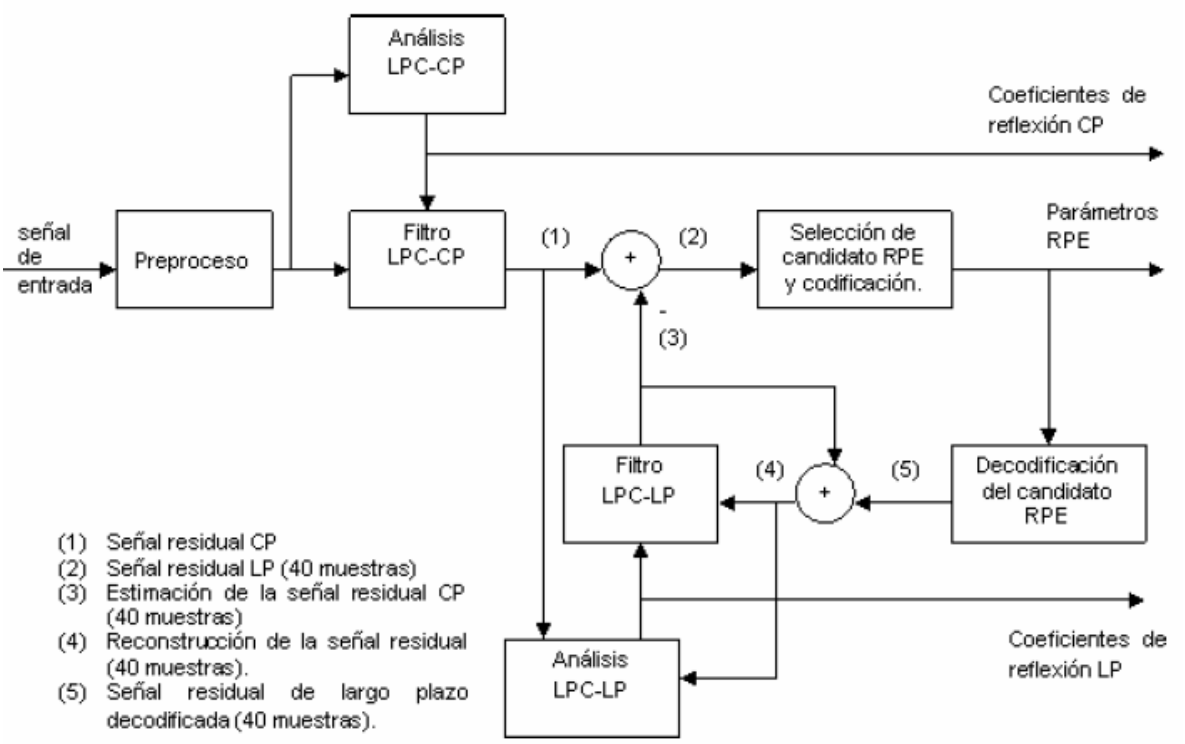
Este es el sistema de compresión utilizado por la telefonía celular europea, utiliza la tecnología LPC para realizar la predicción de largo plazo.

La entrada de GSM 6.10 consiste en ventanas de 160 valores PCM [KHz], codificadas a 13 bits con signo. Cada ventana es de 20 ms, lo cual es alrededor de un período glotal para una voz grave o 10 para voces agudas. Este tiempo es bastante corto y durante él la señal de voz no cambia demasiado.

El compresor GSM 6.10 modela el habla con dos filtros y una excitación inicial. Un filtro de predicción lineal de corto plazo; el primer paso en la compresión (y la

última en la descompresión) y asume el rol del tracto vocálico y nasal. Éste es excitado por la salida de un segundo filtro de predicción lineal de largo plazo que transforma su entrada (la señal residual) en una mezcla de onda glotal y ruido.

Figura 22: Esquema GSM 6.10:



El bloque de 40 muestras de la señal residual de largo plazo es representado como una de 4 subsecuencias candidatas de 13 pulsos. La subsecuencia elegida es identificada por su posición (M) en la matriz RPE (matriz que contiene a los 4 candidatos).

El algoritmo elige la secuencia de mayor energía, esta es, la de mayor suma cuadrática de sus valores. Un índice de 2 bits transmite la elección al decodificador. Eso deja 13 muestras de 3 bits y un factor de escala de 6 bits que transforma la codificación PCM en APCM (Adaptive PCM, el algoritmo se adapta a la amplitud total aumentando o disminuyendo el factor de escala).

Finalmente, el codificador prepara la siguiente predicción a largo plazo actualizando su "salida pasada", es decir, la señal residual de corto plazo reconstruída. Para asegurarse de que el codificador y el decodificador trabajan sobre la misma señal residual, el codificador simula los pasos de progresión del decodificador hasta momentos antes de la etapa a corto plazo.

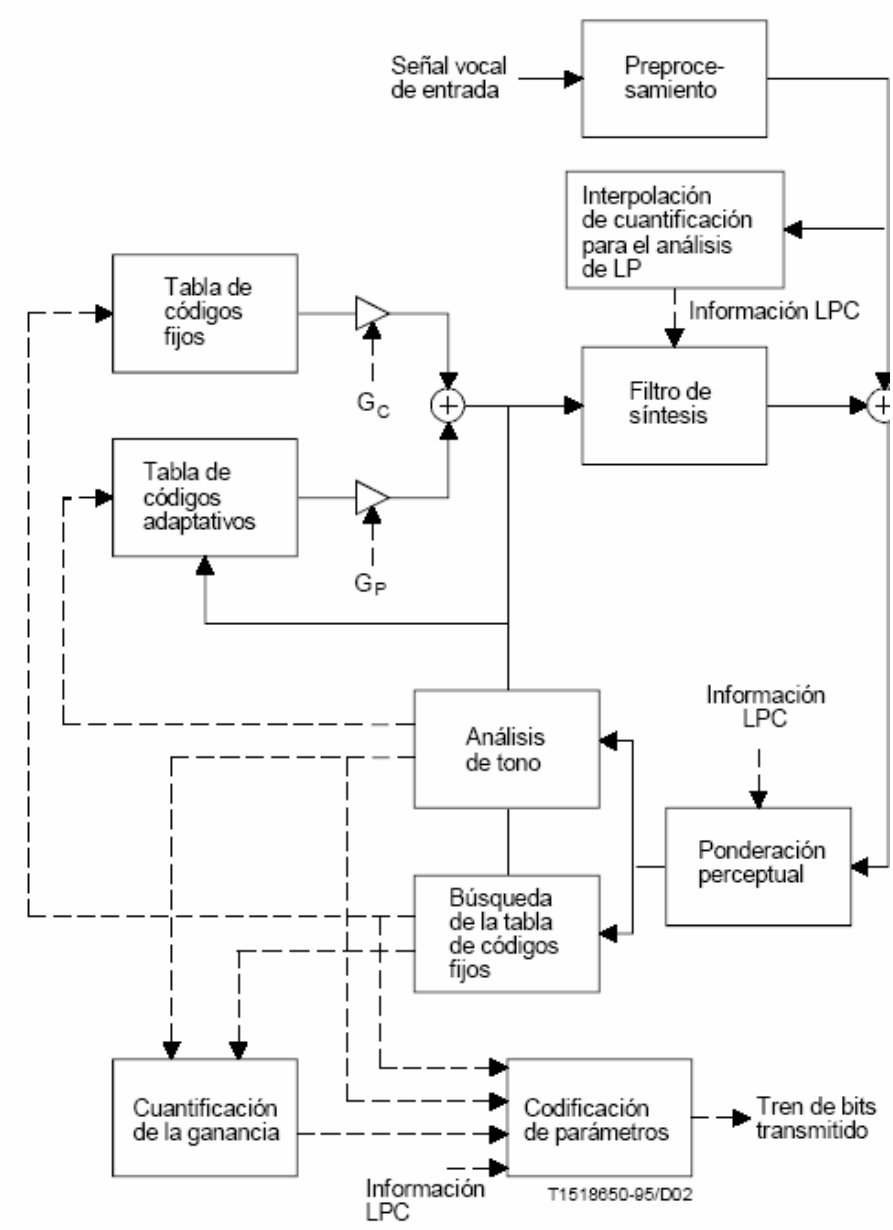
7.2.5. CS-ACELP

G. 729 es la recomendación de la UIT que describe el algoritmo para la codificación de la voz a 8 kbit/s mediante predicción lineal con excitación por código algebraico con estructura conjugada (CS-ACELP, *conjugate-structure algebraic-codeexcited linear-prediction*).

"El códec en cuestión está diseñado para operar con una señal digital obtenida tras efectuar, primero un filtrado con la anchura de banda telefónica (Recomendación G.712) de la señal analógica de entrada, seguido de su muestreo a 8000 Hz y su conversión a una modulación por impulsos codificados (MIC o, PCM-Pulse Code Modulation) lineal de 16 bits, para entrar en el codificador. La salida del decodificador deberá reconvertirse a una señal analógica siguiendo un método similar. Otras características de entrada/salida, como las que se especifican en la Recomendación G.711 para datos PCM de 64 kbit/s, deberán convertirse a PCM lineal de 16 bits antes de codificar, o de PCM lineal de 16 bits al formato apropiado después de decodificar. El tren de bits del codificador al decodificador se define dentro de esta norma" [34].

"El códec CS-ACELP se basa en el modelo de codificación mediante la predicción lineal con excitación por código (CELP). Opera con tramas vocales de 10 ms correspondientes a 80 muestras a una velocidad de muestreo de 8000 muestras por segundo. En cada trama de 10 ms se analiza la señal vocal para extraer los parámetros del modelo CELP (coeficientes de filtros de predicción lineal, ganancias e índices de las tablas de códigos adaptativos y fijos). Los parámetros en cuestión se codifican y se transmiten. En el decodificador, dichos parámetros se utilizan para recuperar los parámetros de excitación y del filtro de síntesis. La voz se reconstruye filtrando la excitación a través del filtro de síntesis de corto plazo. El filtro de síntesis de corto plazo se basa en un filtro de predicción lineal (PL) de décimo orden. El filtro de síntesis de largo plazo o de tono se aplica mediante el método de la llamada tabla de códigos adaptativos. Tras calcular la señal vocal reconstruida, ésta se mejora con un postfiltrado" [34].

Figura 23: Esquema CS-ACELP [34]

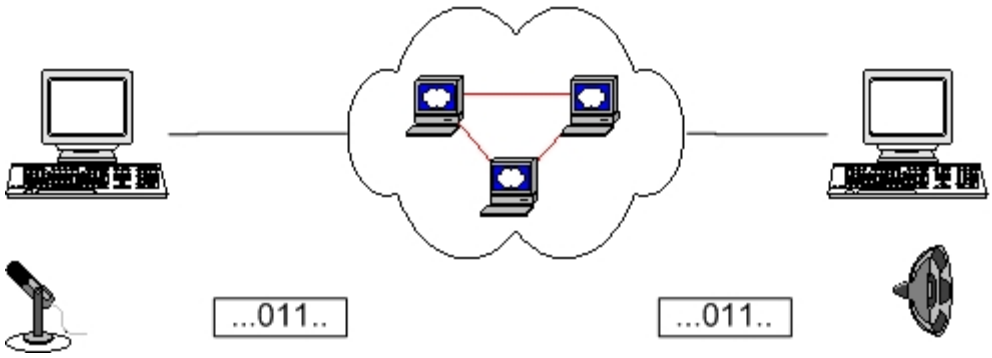


Influencia de pérdida de paquete en calidad perceptiva de CS-ACELP

Tomado del documento [35]:

El desempeño de codificadores de voz puede disminuir considerablemente si el canal que se esta utilizando para su transmisión presenta pérdidas de paquetes. Debido a que las tecnologías de codificación difieren unas de otras, no es posible establecer conclusiones de comportamiento general.

Figura 24: Transmisión de datos en ambientes de pérdida de paquetes [35]:



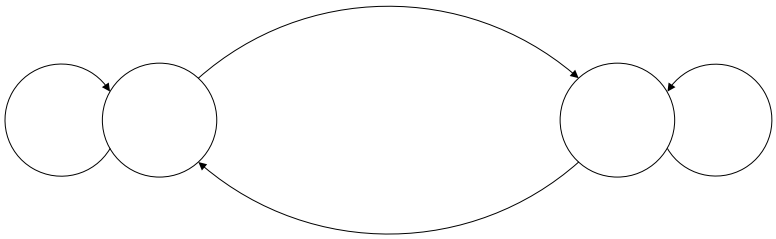
La Figura muestra el diagrama en bloque de un sistema de transmisión de voz entre dos *hosts* remotos, utilizando Internet como medio de comunicación. En uno de los hosts, el cliente, se genera una señal acústica la cual es codificada para posteriormente ser enviada fragmentada en paquetes hacia el otro *host*, el servidor. El agrupará los paquetes, decodificará la información y reproducirá la señal codificada. La red Internet presenta pérdidas de paquetes principalmente por que cada uno de los *router* que componen la red posee buffer de información finita por lo que si la carga presente en el nodo supera su capacidad, irremediablemente esa información se perderá.

En general, paquetes consecutivos tienden a seguir una misma ruta por lo que las pérdidas de paquetes suelen aparecer en ráfagas (*burst*): o sea la probabilidad de que un paquete se pierda aumenta si el paquete anterior se perdió. Debido a este comportamiento del medio de transmisión, la señal decodificada bajará su calidad perceptiva con respecto a la señal original.

Con el objetivo de evaluar el desempeño del codificador CS-CELP a 6.4 kbps, 8 kbps y 11.8 kbps frente a medios de transmisión semejantes al descrito en la Figura 24, se implementó un sistema de simulación que utiliza el modelo de Gilbert para representar el comportamiento de pérdida de paquete de la red.

Este modelo corresponde a una cadena de Markov de dos estados que modela la tasa de pérdida de paquetes, *packet loss rate* (PLR), como puede ser visto en la siguiente Figura:

Figura 25: Modelado de perdida de paquetes.



Un estado es caracterizado por tener baja probabilidad de pérdida de paquetes (P_1) a diferencia de la otra, que tiene una alta probabilidad de pérdida de paquetes (P_2). P_s y P_t son las probabilidades de cambiar de estado. Si las restricciones (1) y (2) son consideradas, el sistema permanecerá en el estado 1 la mayoría del tiempo, por lo que pocos paquetes se perderán; pero cuando el sistema cambia al estado 2, los paquetes se perderán con mayor frecuencia y en ráfagas.

$P_s << 1 - P_t$ (1)
$P_1 << P_2$ (2)

Una consideración incorporada al modelo establece que aproximadamente el 90% de las ráfagas consisten en tres paquetes o menos. Dado que el modelo cuenta con más variables que restricciones, el problema presenta múltiples soluciones. Algunas de ellas se presentan en la siguiente Tabla:

Tabla 20: Soluciones del modelo de pérdida de paquetes:

PLR	Ps	Pt	P1	P2	%<3
0.2	0.001000	0.4200	0.0001	0.8000	91.0
0.4	0.001241	0.3500	0.0010	0.8500	90.1
0.6	0.002370	0.4000	0.0010	0.8500	90.3
0.8	0.003270	0.3700	0.0010	0.8000	90.2
1.0	0.001481	0.2000	0.0040	0.8200	90.3
1.5	0.003473	0.2900	0.0050	0.8500	90.1
2.0	0.006325	0.3500	0.0050	0.8500	90.1
2.5	0.004545	0.2500	0.0100	0.8500	90.5
3.0	0.007317	0.3000	0.0100	0.8500	90.2
3.5	0.004908	0.2000	0.0150	0.8500	90.0
4.0	0.008333	0.2700	0.0150	0.8500	90.2
4.5	0.011180	0.3000	0.0150	0.8500	90.0
5.0	0.012375	0.3000	0.0170	0.8500	90.1
5.5	0.009172	0.2500	0.0240	0.9000	90.1

$1 - P_s$ S

6.0	0.012353	0.3000	0.0250	0.9100	90.4
6.5	0.007811	0.2000	0.0320	0.9100	90.5
7.0	0.008333	0.2000	0.0350	0.9100	90.6
7.5	0.009102	0.2000	0.0370	0.9100	90.5
8.0	0.009524	0.2000	0.0400	0.9200	90.4

En la tabla anterior se entrega un conjunto de parámetros obtenidos para distintos niveles PLR, sujeto a las restricciones mencionadas.

Las señales que fueron utilizadas para evaluar la influencia de pérdida de paquete en la calidad perceptiva de los codificadores de voz fueron extraídas de la base de datos LATINO (LDC, 1995), compuesta por 40 locutores nativos de Latino América en el cual cada uno leyó 125 frases de diarios en español. En total fueron utilizadas 400 frases correspondientes a 10 locutores distintos. Las señales fueron codificadas con el codificador CS-CELP a las tasas de 6.4 kbps, 8 kbps, y 11.8 kbps. Una vez codificada, la señal fue sometida al simulador de pérdida para posteriormente decodificarla. Es importante mencionar que este codificador incorpora un procedimiento de reducción del error producido por perdidas de paquetes o por error en la transmisión, que está definido en el estándar (ITU, 1996) y que fue incorporado en la simulación.

En conclusión, según resultados entregados los Test *mean opinion score* (MOS) el CS-ACELP ha demostrado una aceptable precisión frente a pérdidas de paquete en codificadores basados en tecnología CELP. La mayoría de los sistemas de voz sobre IP desarrollados en la actualidad confían en el algoritmo CS-ACELP. El Test MOS consiste en una evaluación subjetiva de la calidad de síntesis de voz de un sistema. Fue normalizado por el CCITT a principio de los años 80 y se le ha utilizado principalmente para medir la calidad en sistemas de comunicación celular digital. El test consiste en realizar una encuesta de opinión a un conjunto de individuos de prueba los cuales deben evaluar una grabación de voz según la siguiente tabla:

Tabla 21: Calificación de un test MOS.

NOTA	CALIDAD	ESFUERZO DE ESCUCHA	DEGRADACIÓN
5	Excelente	Posible relajación completa, no requiere ningún esfuerzo.	Inaudible
4	Buena	Atención necesaria, no se requiere esfuerzo apreciable.	Audible pero no molesta.
3	Aceptable	Se necesita esfuerzo moderado.	Ligeramente molesta.
2	Mediocre	Se necesita esfuerzo considerable	Molesta.
1	Mala	Cualquier esfuerzo no permite comprender	Muy Molesta.

PARTE II

SONIDO DIGITAL EN LA PRÁCTICA

8. PROGRAMACION DE SONIDOS EN EL PC

8.1. A NIVEL DE MS-DOS

A nivel del MS-DOS (Microsoft - Sistema Operativo de Disco) en programación toman mucha importancia las llamadas ***interrupciones*** ya sean hardware (generadas por algún dispositivo físico) o software (generadas por programas que invocan funciones). Las interrupciones en el DOS son como la API de Windows. Un conjunto completo de funciones disponibles al programador para interactuar con el hardware presente en el PC. El código de las interrupciones permanece en algún lugar de la memoria direccionable y su respectiva dirección se almacena en una tabla (vector de interrupciones) al principio de la memoria. La memoria direccionable incluye la ROM del BIOS (Basic Input/Output System), por lo que existen interrupciones del BIOS como estándares incluidas en el PC y también interrupciones DOS que varían de acuerdo a cada versión de éste.

Las interrupciones hardware, son producidas por los chips de apoyo al procesador y controladas por un IRQ (Interrupt Request). Por ejemplo, el teclado genera una interrupción al procesador (esta tiene asignado un número, para el teclado es el 9) y éste le entrega el control al procedimiento encargado del tratamiento de la interrupción cuya dirección se almacena en la posición 9 del vector de interrupciones.

Al hablar de programación (algo a bajo nivel) en el DOS no se pueden dejar de mencionar los puertos. Los puertos son la forma de interacción del procesador con los demás elementos hardware a través de los Buses del sistema. Cada elemento del sistema tiene asignado uno o varios puertos de control, por ejemplo, el teclado tiene asignado el rango de direcciones 60H a 6FH.

El programador puede a nivel de Assembler interactuar directamente con el hardware a través de las interrupciones y los puertos, pero esto demanda un gran conocimiento. Afortunadamente, en los lenguajes de alto nivel para el DOS como el C, C++, Basic o Pascal, ya existen funciones que realizan acceso al hardware sin que se tenga que saber ni las direcciones de los puertos ni las interrupciones usadas. Pero, las ultimas versiones de los lenguajes de alto nivel como Turbo Pascal 7, o C++ 3.0 no incluyen funciones de soporte para les nuevas tecnologías multimedia como las tarjetas de sonido, el CD ROM, o las mejoras en la calidad del video; por lo que los programadores haciendo uso de las funciones que estos lenguajes proveen, para controlar interrupciones y direccionar datos a los puertos, manejan los nuevos dispositivos y tarjetas adicionales, como por ejemplo, tarjetas de sonido, de red, modems, etc.

8.1.1. Sonidos en el mini Speaker del PC

La producción de sonido más nativa en el PC se hacía a través del pequeño speaker adjunto que solo producía sonidos de variada frecuencia y duración y de forma de onda única (sinusoidal) para dar origen a unas pobres notas musicales del compás.

En Pascal por ejemplo, producir un sonido en el speaker de cierta frecuencia y con determinada duración se logra con el siguiente código:

```
Sound(300);    (* 300 es la frecuencia *)
Delay(1000);   (* Duración *)
Nosound;
```

Es claro que no es el mejor sonido que se puede lograr con el PC pero era lo que existía entonces.

8.1.2. Sonidos con la Sound Blaster (SB)

Ya se mencionó anteriormente (Numeral 3.4) que la SB se convirtió en un estándar para PC en cuanto a la grabación y reproducción del sonido. Lograr producir un sonido a nivel de DOS con una tarjeta de sonido SB no es una tarea fácil y requiere de amplios conocimientos de Programación a bajo nivel en cuanto a interrupciones, puertos, direccionamiento, etc. En éste documento se explicará con breves palabras, más o menos como es el funcionamiento, pero no se incurrirá en código pues estos códigos son bastante largos y para entenderlos requieren bastantes conocimientos previos que se salen del alcance de éste documento.

8.1.2.1. El DSP

“En el centro del muestreo se encuentra la programación del procesador de sonido digital (DSP), que esta presente de diferentes formas en las tarjetas SB” [10]. Existen varias versiones del DSP, casi cada una, con cada tarjeta SB que fue saliendo. “Los principales criterios para diferenciar entre las distintas versiones DSP están asociados a la velocidad y a la profundidad de los bits en el muestreo. Por velocidad se entiende el número de muestras que se pueden grabar o reproducir por segundo. Con la profundidad de los bits se define la resolución de la imagen de la señal analógica original y su representación digital” [10].

En resumidas palabras el DSP es un chip programable que nos permite a través de una interfaz de comandos y parámetros realizar el muestreo y la reproducción

del sonido. “El DSP soporta cinco diferentes modos de muestreo que pueden ser utilizados según la tarea que se quiere acometer. Cuatro de esos modos utilizan el controlador DMA (Acceso directo a memoria) para llevar a cabo la transferencia de datos entre la memoria principal y el DSP y el otro atiende el DSP directamente” [10].

8.1.2.2. El controlador DMA

DMA significa Acceso Directo a Memoria (Direct Memory Access) y es la abreviatura de una técnica mediante la cual se pueden transferir datos de un dispositivo directamente a la memoria y viceversa. Anteriormente, cuando las velocidades de los procesadores eran bastante bajas, el uso del DMA era una buena manera de lograr eficacia. Pero con las velocidades de los procesadores actuales, el controlador DMA se ha vuelto anticuado. Aun así, el hecho de lograr que se ejecuten procesos sin que el procesador trabaje no ha dejado de ser importante, por lo que también en la actualidad han salido controladores DMA de gran velocidad y capacidad.

En cuanto a la tarea del tema, para la producción y grabación de sonido, las SB hasta la SB 16 incluyen modos de operación del DSP donde se hace muy indispensable el uso del controlador DMA. “En los diferentes modos de transferencia DMA, trabajan mano a mano el controlador DMA y el DSP. En la reproducción de muestras, por ejemplo, el controlador DMA transfiere los datos de muestras a la CPU, pasando por un buffer de la memoria principal al DSP. El controlador DSP la convierte en señales analógicas y las traslada al amplificador de la tarjeta Sound Blaster. Desde allí puede llegar al usuario a través de auriculares, de amplificadores o de un equipo stereo” [10].

“En el sentido contrario, es el DSP el que toma las señales analógicas de una fuente de sonido (Micrófono, Line-In, CD, etc), las convierte en una muestra digital y finalmente las transfiere a un buffer en la memoria principal a través del DMA” [10]. El DOS en modo real tiene limitaciones de memoria debido a la compatibilidad con el procesador 8086, el buffer debe estar por debajo del límite de 1MB y no debe sobrepasar ningún límite de página de 64k. Windows por el contrario, trabaja los procesadores 80386 y superiores en modo protegido y por tanto, explota al máximo la capacidad de direccionamiento de 4 GB, eliminando el problema de buffers reducidos.

8.1.2.3. Grabando y Reproduciendo

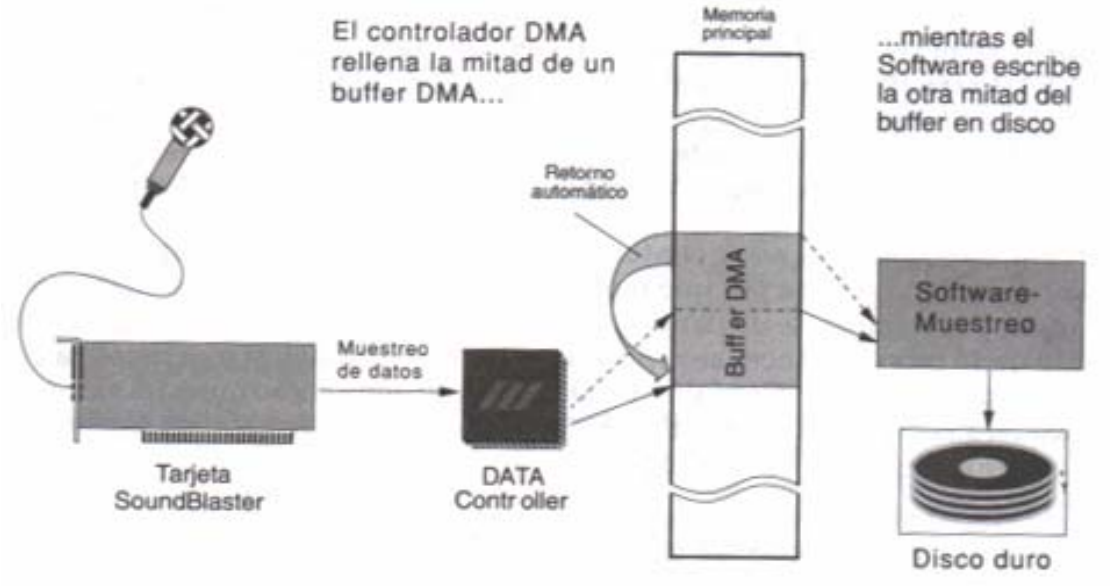
Para grabar o reproducir sonido en el PC con una tarjeta Sound Blaster o Compatible a nivel de programación en lenguajes de alto nivel para DOS (aunque programando en lenguajes como Visual C++, o Delphi en Windows que permiten la inclusión de código Assembler se pueden realizar estas tareas), los elementos

básicos a tener en cuenta son el DSP, el DMA y conocimientos de almacenamiento en disco.

El método más usado por los programadores, es conocido como la técnica de Double-Buffering, y tiene el objetivo de ir copiando periódicamente los datos del buffer de sonido al disco duro a medida que se realiza el muestreo. Y para el caso contrario de reproducción, es lo mismo, se van leyendo periódicamente datos del disco duro y se llevan al buffer de sonido en memoria principal de donde son tomados por el DSP a través del DMA. Este proceso puede no ser muy bueno en equipos lentos o especialmente con discos duros de bajas velocidades, pues el sonido muestreado puede quedar con ciertas falencias o saltos. Pero funciona perfectamente en los equipos actuales que tienen además de discos más rápidos, mecanismos de arreglos DMA y memoria RAM incluidos en la misma tarjeta de sonido mejorando completamente las calidades de los sonidos.

En pocas palabras la técnica de double-buffering, se trata de que ni el DSP ni el controlador DMA estén desocupados. Que todo el tiempo, ninguno de los dos tenga que esperar (o esperar el menor tiempo) a que el otro termine para poder empezar excepto al principio mientras el DSP llena la primera mitad del buffer. Realmente la técnica no es tan sencilla como parece, para implementarla es necesario conocer más detalles y trucos a la hora de interactuar con los chips. También cabe mencionar que ésta técnica no es usada solo en las tarjetas de sonido, sino, que también la usan otros dispositivos como los discos duros, las disqueteras e incluso algoritmos que manejen buffers como por ejemplo, algoritmos de imágenes. .

Figura 26: Técnica de Double-Buffering [10].



8.2. A NIVEL DE WINDOWS CON VISUAL BASIC

En Windows la base primordial para la programación se encuentra en la API (Interfaz de Programación de Aplicaciones) y es un conjunto de funciones almacenadas generalmente en librerías de enlace dinámico DLL que sirven a los programadores para comunicarse, a través del sistema operativo, con todos los dispositivos presentes en el PC. Entre las dll del sistema más importantes están: Kernel.dll (o kernell32 según la versión de Windows) que es el núcleo del sistema y almacena las funciones del inicio y fin de las aplicaciones; la User.dll (o user32.dll) con funciones generales; y la gdi.dll con funciones especialmente para el entorno gráfico. Una librería que no se puede dejar de mencionar para éste caso, es la winmm.dll (Windows Multimedia) que es la encargada de controlar todos los dispositivos multimedia instalados en el PC.

Los primeros compiladores para Windows, como pascal, C++, de Borland, hacían un completo uso de las funciones del sistema operativo, desde las funciones de creación de ventanas hasta las más complicadas como el manejo de la memoria. El lenguaje de programación más acogido por muchos programadores fue entonces el Visual Basic de Microsoft, que presentaba un entrono de desarrollo amigable permitiendo a los usuarios elaborar programas de ventanas con solo arrastrar y soltar elementos de una barra de herramientas (controles de Windows). Pero luego, Borland actualizó sus lenguajes, dotándolos también de entornos de desarrollo similares al de Visual Basic y entregó a los programadores el Delphi (con código Pascal) y el C++ Builder.

En los nuevos lenguajes visuales, mucho código queda escondido al programador. Aún así, algunos de ellos permiten hacer llamadas a la API de Windows e incluso incluir código Assembler en sus aplicaciones para dar más libertad y control sobre el sistema operativo.

A medida que fue creciendo la popularidad del PC fueron incrementándose los fabricantes de partes para éste. La tarea para los programadores de comprobar el tipo o clase de hardware instalado en el PC fue volviéndose más dura. Este problema se presentaba principalmente en el DOS debido a la falta de estandarización. El DOS en algunas versiones ya incluía los llamados drivers o controladores de dispositivo, pero no eran muy completos ni amigables. En Windows, el propio sistema operativo, reconoce el tipo y marca de hardware instalado gracias a estándares y acuerdos con los fabricantes quienes distribuyen con sus productos los debidos controladores. Los drivers generalmente se distribuyen en archivos .drv o .vxd (Dispositivo Virtual) en las primeras versiones de Windows (95, 98), y en las últimas (2000, Xp) en archivos WDM (Windows Driver Model).

Al poder usar funciones de la API de Windows obtenemos la ventaja de no tener que interactuar directamente con los dispositivos hardware y ni siquiera con sus

drivers (controladores). Por ejemplo, en la producción de sonidos, existen funciones a las que solo se les indica el correspondiente archivo de sonido que se quiere escuchar y ellas se encargan de todo; al programador no le interesa ni que marca de tarjeta de sonido esta instalada ni como se controla, solo se llama a la función y ya.

Existe incluso, en Visual Basic, un control ActiveX (librería especial con interfaz gráfica para cargar en diseñadores) llamado MMC, o control multimedia de Microsoft, que incluye todas las funciones necesarias para grabar y reproducir sonidos y controla además al CD ROM para escuchar música.

Existen muchas formas de lograr escuchar o grabar sonidos fácilmente desde Visual Basic:

8.2.1. Usando Llamadas a la API de Windows

Ya se mencionó que la programación en Windows, de cierta forma a bajo nivel, se realiza a través de llamadas a las funciones de la API y que además existe una librería que agrupa las funciones para el control de la Multimedia, lo que incluye al sonido, que es la **winmm.dll**. Si se desea reproducir un sonido guardado en un archivo .wav utilizamos la función sndPlaySound como se muestra en el siguiente código de Visual Basic 6.0.

```
Public Declare Function sndPlaySound Lib "winmm.dll" Alias sndPlaySoundA"
            (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long
Sub Main()
    sndPlaySound("C:\WINDOWS\MEDIA\Start.wav", &H1)
End Sub
```

El código anterior es el de un proyecto en Visual Basic con un solo modulo y que al ejecutarse reproduce el archivo de sonido que le pasemos a la función sndPlaySound.

La función sndPlaySound es muy buena y fácil de usar, sin embargo si se quiere tener más información del archivo a reproducir, como por ejemplo, información de las características de muestreo, posición de reproducción, y hasta poder detenerlo una vez iniciada la reproducción, habría que incluir muchas otras funciones de la API. Una desventaja grande de la función, es que tiene que cargar por completo los datos de sonido a la memoria para poder reproducir el sonido, un problema cuando el archivo a cargar es muy grande.

Existen un grupo de funciones de la API que permiten manejar cualquier dispositivo multimedia abierto (también con funciones de la API) mediante una interfaz de comandos. La función *mciSendCommand* Permite enviar comandos a

la MCI (Multimedia Control Interface) que se ahorra bastante trabajo a la hora de programar. La siguiente es una lista reducida de los comandos disponibles:

Tabla 22: Algunos Comandos de la MCI.

Comando	Descripción
Close	Termina un dispositivo en uso.
Info	Devuelve información del estado del dispositivo
Open	Abre o inicializa un dispositivo o archivo
Pause	Suspende la ejecución o grabación
Play	Ejecuta el archivo asociado al dispositivo
Record	Graba
Stop	Detiene por completo la ejecución

8.2.2. Usando el Control MMC

El control MMC (Multimedia Control) es un control ActiveX disponible para Visual Basic. Este control es como la interfaz gráfica de las funciones de la MCI de la API de Windows. El siguiente sencillo programa en VB6 nos permite ver la aplicación del control:

Figura 27: Vista gráfica del Control MMC en Visual Basic.

El control MMC es el que contiene los botones relacionados con la ejecución de archivos multimedia. El código del Programa es el siguiente:

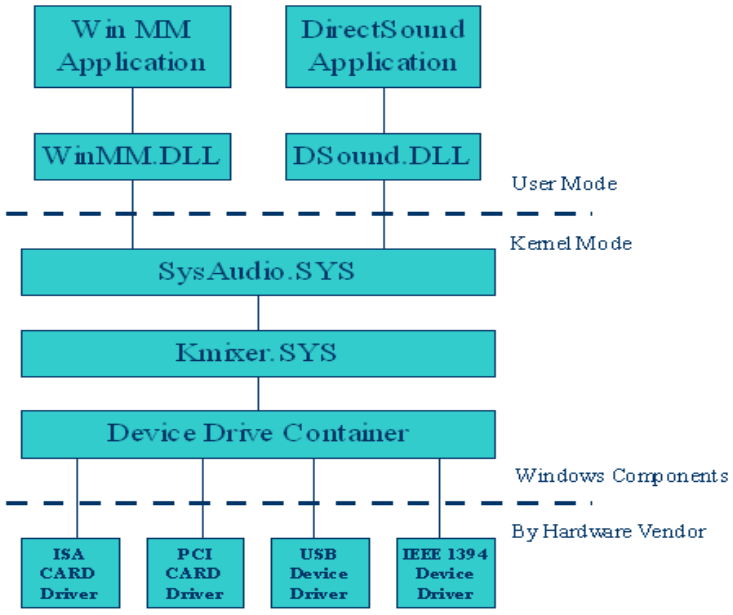
```
Private Sub Command1_Click()
    On Error Resume Next
    Dialogo.CancelError = True
    Dialogo.ShowOpen()
    If Err = 0 Then
        MM.FileName = Dialogo.FileName
        MM.Command = "Open"
    End If
End Sub
```

El código anterior es muy sencillo, y la aplicación es probablemente la más sencilla que se puede lograr con el control MMC. En el ejemplo, basta con especificar el archivo a enlazar y enviar el comando "Open" a la MCI. Luego, el control completo queda a cargo de los botones del MM Control, que en este caso, no tienen absolutamente nada de código y aun así se pueden escuchar y grabar sonidos y ver videos. Los formatos y tipos de archivos reconocidos por el control MM dependen de los codecs y drivers instalados en el sistema.

8.2.3. Usando Direct Audio de DirectX

Antes de empezar a describir de qué se trata direct audio vamos a analizar la siguiente imagen:

Figura 28: Esquema de Audio en Windows.



Como se observa, las aplicaciones de sonido tienen principalmente dos opciones para la administración de éste en Windows. La primera es usando directamente o a través de un componente (Control Ocx, o Dll sin interfaz visual) las llamadas a la API y la segunda usando DirectSound. Pero, según la gráfica, ambos caminos se encuentran en los controladores estándares de sonido de Windows. Kmixer.sys se encarga de administrar el mezclado en las tarjetas de sonido.

Directrsound es un elemento del conjunto DirectX actualmente integrado con directMusic en DirectAudio. DirectX es un conjunto de herramientas tipo componentes de Windows que nació principalmente para brindarle a los desarrolladores de juegos mejoras en cuanto al acceso a los dispositivos de video, sonido, red, entre otros. Una de las características de DirectX es que acelera los dispositivos optimizando su uso. La última versión disponible de DirectX es la 9c, que incluye: DirectX Graphics, que se ocupa de los gráficos tanto en 2 como en 3 dimensiones; DirectX Audio, que se encargan de la ejecución del sonido; DirectInput, que controla los dispositivos de entrada como el teclado y el Mouse; DirectSetup, que presta servicios para la instalación de DirectX y DirectPlay que facilita el desarrollo de juegos multijugador (en red).

Microsoft distribuye un SDK (Software Developer Kit) de DirectX que contiene las librerías necesarias para usarlo Desde Visual Basic, Visual C++, Delphi, o cualquier otro lenguaje de programación que soporte la Automatización, COM (Modelo de Objetos COmponentes) y ActiveX. La versión 9 de DirectX y su SDK incluyen los ensamblados listos para ser usados con la tecnología .NET desde cualquier lenguaje de Visual Studio .Net.

Por ahora, para probar, se usará DirectSound de DirectX 7 en un proyecto de Visual Basic 6.0. Para poder usarlo, se incluye su referencia en el proyecto. El código que usa solo un formulario y un botón es el siguiente:

```
Dim DX As New DirectX7
Dim DS As DirectSound
Dim DsBuffer As DirectSoundBuffer
Dim DsDesc As DSBUFFERDESC
Dim DsWave As WAVEFORMATEX

Private Sub Form_Load()
    DS = DX.DirectSoundCreate("")
    DS.SetCooperativeLevel(Me.hWnd, DSSCL_PRIORITY)
    DsBuffer = DS.CreateSoundBufferFromFile("c:\prueba.wav", DsDesc,
                                            DsWave)
End Sub

Private Sub Command1_Click()
    DsBuffer.Play(DSBPLAY_DEFAULT)
End Sub
```

El código anterior es todo lo que se necesita para escuchar un sonido en formato wav usando DirectSound. Además de mejoras en la aceleración del sonido, directSound incluye los llamados buffers secundarios:

“Al inicializar DirectSound se crea automáticamente el **buffer de sonido primario**, usado para mezclar los sonidos y enviarlos a la tarjeta de sonido. El buffer primario normalmente no lo tiene en cuenta el usuario, a menos que desee obtener los valores de la muestra de sonido en un tiempo determinado; aunque también se puede usar para mezclar sonidos. Los **buffers secundarios**, generalmente contienen los efectos de sonido a usar. Para cada efecto de fondo existente (un disparo, una puerta que se abre) se crea un buffer secundario” [36].

“Los buffers secundarios pueden ser de dos tipos, "*static*" o "*streaming*". Los buffers static contienen un sonido completo, a diferencia de los bufferes streaming, que sólo contienen una parte del sonido, y los programas deben de encargarse de ir cargando en la memoria reservada para el buffer las distintas partes del sonido, antes de que DirectSound vaya a tocarlo. Por ejemplo, si se tiene una canción de 3 minutos y un buffer secundario de 10 segundos, el programa debe encargarse de que DirectSound encuentre en esos 10 segundos el trozo de memoria de la canción que debe tocar. Evidentemente, los buffers "streaming" ahorran memoria, pero para efectos de sonido cortos (disparos, pro ejemplo) no son necesarios” [36].

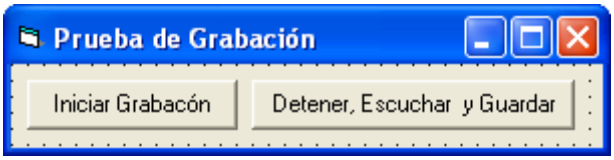
“Cuando se crea un buffer secundario, DirectSound intentará primero reservar memoria de la tarjeta de sonido para guardar el sonido, y si no la encuentra lo guardará en la memoria del ordenador. Tocar los sonidos que estén en la memoria de la tarjeta de sonido es más rápido, por lo que se debe procurar que aquellos sonidos que se vayan a utilizar más habitualmente durante el programa se carguen antes que los menos usados, para que así los sonidos más importantes estén almacenados en la memoria de la tarjeta de sonido” [36].

8.2.4. Grabando sonidos con DirectSound 7

Para grabar un sonido se puede usar tanto la MCI (directamente o a través de un componente ActiveX) como DirectSound. Por ahora se vera un pequeño ejemplo donde solo se graba con DirectSound un sonido de la entrada de sonido seleccionada por el sistema y lo se llevará a un archivo wav.

Primero se debe crear un proyecto y agregar a él una referencia a la librería DirectX para Visual Basic. El formulario principal tiene la siguiente interfaz:

Figura 29: Grabación con Direct Sound 7.



Y el código es el siguiente:

```
Dim DX As New DirectX7
Dim DS As DirectSound
Dim DSB As DirectSoundBuffer
Dim DSC As DirectSoundCapture
Dim DSCB As DirectSoundCaptureBuffer

Dim DsCDesc As DSCBUFFERDESC
Dim DsCWaveDesc As WAVEFORMATEX
Dim DSDesc As DSBUFFERDESC

Dim Cursor As DSCURSORS

Dim Buffer() As Integer

Private Sub Command1_Click()
    DSCB.start(DSCBSTART_DEFAULT)
    Command1.Enabled = False
End Sub

Private Sub Command2_Click()
    DSCB.Stop()
    DS = DX.DirectSoundCreate("")
    DS.SetCooperativeLevel(Me.hWnd, DSSCL_NORMAL)

    DSCB.GetCurrentPosition(Cursor) 'Obtiene la ultima posición del buff
    DSDesc.lBufferBytes = Cursor.lWrite * DsCDesc.fxFormat.nBlockAlign
    DSDesc.lFlags = DSBCAPS_CTRLVOLUME Or DSBCAPS_STATIC

    If Cursor.lWrite = 0 Then
        Exit Sub
    End If


    DSB = DS.CreateSoundBuffer(DSDesc, DsCDesc.fxFormat)
    ReDim Buffer(Cursor.lWrite * DsCDesc.fxFormat.nBlockAlign + 1)
    DSCB.ReadBuffer(0, (Cursor.lWrite * DsCDesc.fxFormat.nBlockAlign),
                    Buffer(0), DSCBLOCK_DEFAULT)
    DSB.WriteBuffer(0, Cursor.lWrite * DsCDesc.fxFormat.nBlockAlign,
                    Buffer(0), DSBLOCK_DEFAULT)
    DSB.SaveToFile(App.Path & "\Prueba.wav")
    DSB.Play(DSBPLAY_DEFAULT)
```

```
    Command2.Enabled = False
End Sub

Private Sub Form_Load()
    'Graba en formato 16 bit stereo a 44100 hz  -> Calidad de CD

    Canales = 2
    BITS = 16
    DsCWaveDesc.nFormatTag = WAVE_FORMAT_PCM
    DsCWaveDesc.nChannels = Canales
    DsCWaveDesc.lSamplesPerSec = 44100
    DsCWaveDesc.nBitsPerSample = BITS
    DsCWaveDesc.nBlockAlign = Canales * BITS / 8
    DsCWaveDesc.lAvgBytesPerSec = DsCWaveDesc.lSamplesPerSec *
                                  DsCWaveDesc.nBlockAlign
    DsCWaveDesc.nSize = 0

    DsCDesc.fxFormat = DsCWaveDesc
    DsCDesc.lBufferBytes = DsCWaveDesc.lAvgBytesPerSec * 20
    DsCDesc.lFlags = DSCBCAPS_WAVEMAPPED

    DSC = DX.DirectSoundCaptureCreate("")
    DSCB = DSC.CreateCaptureBuffer(DsCDesc)
End Sub
```

La metodología usada en el código anterior es: Grabar el sonido en un buffer de DirectSound Capture, copiar dicho buffer a un buffer primario de sonido y enviarlo al disco duro.

9. APLICACIÓN MGWAVE

A partir de la segunda fase de la práctica empresarial, se empezó a desarrollar un programa en Visual Basic que reuniera lo fundamental de la investigación. Con la culminación de la segunda fase y los conocimientos adquiridos sobre digitalización y el formato Wave se creo un programa que permitía:

- Abrir Archivos de Sonido con extensión .wav y con formato PCM
- Grafica de forma parametrizada la onda de sonido de archivos con un solo canal de muestras (mono) y con cuantificación a 8 bits. Esta limitación, pensando en las necesidades de los archivos de voz.
- Escuchar el sonido, conocer la posición actual en bytes de muestra y en segundos, posicionarse (saltando) en cualquier parte del sonido, conocer el tamaño en bytes de muestra y en segundos del archivo, Seleccionar una sección del sonido para escucharla, etc.
- Insertar etiquetas Simples (Solo guardan la posición inicial) y dobles (Tienen inicio y fin). Agregar, eliminar y cambiar los nombres de las etiquetas. Permite también insertar una etiqueta doble asociada a una selección de sección de sonido.
- Modificar las posiciones de las etiquetas ya creadas arrastrando sus textos visuales.
- Insertar etiquetas prenombradas.
- Grabar archivos de voz con el uso del micrófono o el dispositivo de entrada seleccionado por en el sistema operativo (Control de volumen-Propiedades-Grabación).
- Grabar los cambios en los archivos abiertos (solo cambios en cuanto a etiquetas) y guardar los archivos grabados junto con las etiquetas creadas.

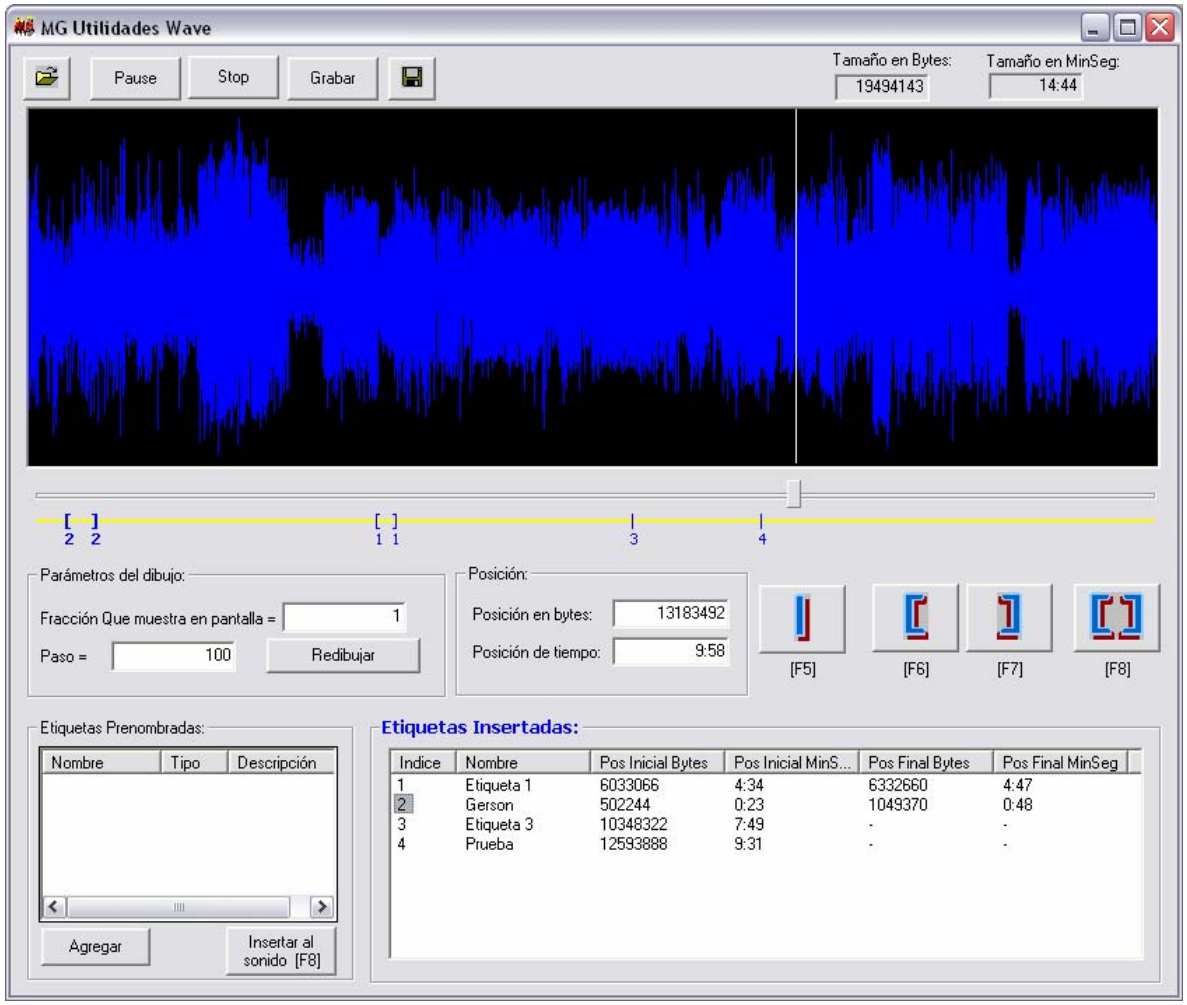
En la página siguiente encontramos una imagen de la ventana principal del programa.

A medida que se avanzó en la investigación, se fueron agregando nuevas características al programa. Con la culminación de la tercera fase (Compresión de Sonido) y con los conocimientos necesarios sobre el formato MP3 el programa permitía adicionalmente:

- Abrir, decodificar y dibujar archivos con formato Wave pero compresos con MPEG Layer 3.
- Agregar, modificar y eliminar las etiquetas adjuntas a dichos archivos.
- Grabar los archivos abiertos con los nuevos cambios en cuanto a etiquetas.

- Grabar archivos de voz con el uso del micrófono o el dispositivo de entrada seleccionado por en el sistema operativo (Control de volumen-Propiedades-Grabación) en el formato seleccionado por el usuario PCM o MP3.
- El nuevo MGWave permite dibujar ondas de 16 bits por muestra y mezclar en un solo dibujo las ondas de archivos multicanal.

Figura 30: Ventana principal del programa MGWave.



10. PROGRAMANDO MP3

Todo el contenido siguiente hace énfasis en la programación sobre Visual Basic 6.0. La documentación técnica existente acerca del MP3 es muy escasa cuando se trata de codificación en Visual Basic, ya que codificar y decodificar MP3 son procesos que requieren mucha máquina y velocidad de código, por lo que todos los programas verdaderamente útiles y comerciales están desarrollados en C\C++. Aun así, con un fin más académico que comercial y con el objetivo de hacer más entendible la cuestión y poder continuar con el Programa MGWave descrito en el capitulo anterior (en su segunda fase), en todo lo relacionado con programación que sigue a continuación, se busca una adecuación en Visual Basic, así la velocidad no sea la mejor.

Para poder escuchar un archivo MP3 en Visual Basic, basta con usar los componentes a modo de librerías, distribuidos con aplicaciones de sonido como el Windows Media Player, Active Movie u otros. Pero para éste caso no es suficiente con escucharlos, ya que se necesitan las muestras de sonido descompresas para graficarlas y asociarles etiquetas, por lo que hay que profundizar más. Implícitamente toda aplicación que reproduzca archivos MP3 tiene que primero descomprimir los datos (a un formato PCM) para poder tocarlos.

10.1. EL PROCESO DE CODIFICACIÓN

Como se mencionó en el numeral 6.2.1.3, existen muchas herramientas comerciales para comprimir a MP3. Pero también existe BladeEnc, un hijo del encoder LAME de licencia GNU (Publica y gratuita). LAME (Acrónimo recursivo de Lame Encoder) por ser GNU provee el código fuente, pero aún así, recompilar el código fuente de LAME no es tarea fácil ya que son más de 300 archivos de código fuente en lenguaje C estándar que se deben compilar con un una herramienta también GNU multiplataforma. Para Windows, y desarrollos en Visual C++ existe la librería compilada “lame_enc.dll” que provee las funciones necesarias para codificar a MP3.

La librería ya compilada es muy útil, pero no la se puede usar en Visual Basic ya que sus funciones exportables fueron definidas bajo el estándar de C, y no de Pascal (StdCall) que es el que usa Visual Basic. Entonces existen dos opciones, modificar el código fuente de LAME (DLL de Windows), redefiniendo las funciones a exportar con la convención _stdcall, y volviendo a compilar el proyecto, o bien crear otra Dll con la convención adecuada, que simplemente llame a las funciones de lame_enc.dll. Optando por la segunda opción se creó la librería “LameEncStdCall.dll“, lista para usar en Visual Basic.

Cómo usar la librería lame_enc.dll?

El proceso de forma general se lleva a cabo con las siguientes funciones:

1. *beInitStream*
   Crea un nuevo stream de codificación. Stream es un término muy usado para referirse al canal de comunicación entre una aplicación y un dispositivo o entre las mismas aplicaciones. Esta función, recibe los parámetros de configuración del futuro MP3 y retorna el número de muestras que se pueden codificar con cada llamada de la función beEncodeChunk junto con el handle (manejador) del stream.

2. *beEncodeChunk*
   Codifica un pedazo de datos puestos en un buffer y devuelve el buffer codificado con su respectivo tamaño.

3. beDeinitStream
   Cierra el stream limpiando la memoria.

4. beCloseStream
   Cierra todos los recursos empleados por el stream. Implica que la codificación fue terminada.

El código de la función ComprimirAMP3 agregada a la aplicación MGWave que se mencionó en el capitulo anterior es:

```
Function ComprimirAMP3(ByRef DataFmt As InfoFormato, ByVal BufferPCM() As Byte, _
                       ByRef BufferMP3() As Byte) As Long

    Dim MiBeConfig As BE_CONFIG
    Dim Muestras As Long
    Dim BytesEscritos As Long
    Dim BuffMP3Size As Long
    Dim BuffPCMSize As Long
    Dim hMP3Stream As Long   'puntero al stream abierto
    Dim rc As Long
    Dim TamPCM As Long
    Dim HBuffPCM As Long
    Dim PBuffPCM As Long
    Dim HBuffMP3 As Long
    Dim PBuffMP3 As Long
    Dim i As Long, t As Long

    ComprimirAMP3 = 0

    With MiBeConfig
        .dwConfig = 256
        With .format.MG
            .dwStructVersion = 1
            .dwStructSize = 331
            .dwSampleRate = 11025
```

```
        .dwReSampleRate = 0
        .nMode = 3 'mono
        .dwBitRate = 32
        .nPreset = -1 'No_Preset
        .dwMpegVersion = 0
        .dwPsyModel = 0
        .dwEmphasis = 0
        .bOriginal = 1
        .bWriteVBRHeader = 1
        .bNoBitRes = 1
    End With
End With

rc = beInitStream(MiBeConfig, Muestras, BuffMP3Size, hMP3Stream)

TamPCM = UBound(BufferPCM) - 1
BuffPCMSize = Muestras * 2

HBuffPCM = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or GMEM_ZEROINIT,_
                       BuffPCMSize)
PBuffPCM = GlobalLock(HBuffPCM)

ReDim BufferMP3(0)

HBuffMP3 = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or GMEM_ZEROINIT, _
                      BuffMP3Size)
PBuffMP3 = GlobalLock(HBuffMP3)

i = 0
Do
    If i + BuffPCMSize < TamPCM Then
        Call CopyBYTEStoPTR(PBuffPCM, BufferPCM(i), BuffPCMSize)
        i = i + BuffPCMSize
    Else
        Call CopyBYTEStoPTR(PBuffPCM, BufferPCM(i), TamPCM - i)
        i = 0
    End If

    rc = beEncodeChunk(hMP3Stream, Muestras, ByVal PBuffPCM, ByVal PBuffMP3,_
                  BytesEscritos)

    t = UBound(BufferMP3)
    ReDim Preserve BufferMP3(t + BytesEscritos)
    Call CopyPTRtoBYTES(BufferMP3(t), PBuffMP3, BytesEscritos)
Loop Until i = 0

rc = beDeinitStream(hMP3Stream, ByVal PBuffMP3, BytesEscritos)
rc = beCloseStream(hMP3Stream)

rc = GlobalUnlock(PBuffPCM)
rc = GlobalFree(HBuffPCM)
rc = GlobalUnlock(PBuffMP3)
rc = GlobalFree(HBuffMP3)

DataFmt.BitsPorMuestra = 0
DataFmt.BlockAlign = 1
DataFmt.Canales = 1
DataFmt.CodigoCompresion = 85
DataFmt.MuetrasPorSegundo = 11025
DataFmt.PromedioBitsPorSegundo = 4000
```

```
    'hay que quitar los primeros bytes basura

    Dim j As Long
    i = 0
    Do
        j = BufferMP3(i)
        i = i + 1
    Loop Until j = 255

    Dim HBuffTmp As Long
    Dim PBuffTmp As Long
    Dim TSB As Long

    i = i - 1
    TSB = UBound(BufferMP3) - i


    HBuffTmp = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or GMEM_ZEROINIT, TSB)
    PBuffTmp = GlobalLock(HBuffTmp)

    Call CopyBYTEStoPTR(PBuffTmp, BufferMP3(i), TSB)
    ReDim Preserve BufferMP3(TSB)
    Call CopyPTRtoBYTES(BufferMP3(0), PBuffTmp, TSB)

    rc = GlobalUnlock(PBuffTmp)
    rc = GlobalFree(HBuffTmp)
End Function
```

La función anterior, recibe como parámetros el formato de datos (Una adaptación de la estructura WAVEFORMATEX) el cual contiene las características de onda de los datos pasados en BufferPCM (Array de bytes a comprimir). Y devuelve en BufferMP3 el Array de bytes comprimidos.

10.2. EL PROCESO DE DECODIFICACIÓN

Para comprimir los datos a MP3 se uso el encoder LAME, el cual desafortunadamente (en su forma de DLL) no provee herramientas para descomprimir. En la aplicación MGWave se usan buffers de DirectSound para reproducir sonido, pero DirectSound solo reproduce datos PCM (Libres de codificación). Y entonces como descomprimen los datos MP3 las aplicaciones existentes? Hay dos opciones: la primera es usar librerías de tipo comercial para poder reproducir MP3 (Los datos se van decodificando a medida que se va reproduciendo la música). La segunda opción es usar ACM de Microsoft (Audio Compression Manager) en español: Administrador de Compresión de Sonido, que generalmente, consiste en usar los codecs de sonido instalados en el PC a través de funciones API de Windows. La segunda opción es la que se va a explicar en este documento, ya que fue la implementada en la aplicación MGWave.

10.2.1. Audio Compression Manager (ACM) de Microsoft

ACM es un conjunto de funciones de la API de Windows disponibles en la librería **msacm32.dll** que prestan servicios de compresión y descompresión de sonido para los formatos soportados por Wave de Microsoft. En el numeral 5.3 (Formato de archivos Wave) se menciono, que el archivo .wav de Microsoft contiene un chunk (pedazo) de información del formato de sonido: “fmt ”, el contenido de ése chunk es el siguiente:

Tabla 23: Contenido del Chunk Formato (Repaso);

Desplazamiento	Tamaño	Descripción	Valor
0x00	4	Chunk ID	"fmt " (0x666D7420)
0x04	4	Tamaño de datos del chunk	16 + bytes de formato extra
0x08	2	Código de compresión	1 - 65,535
0x0a	2	Numero de Canales	1 - 65,535
0x0c	4	Rata de muestreo	1 - 0xFFFFFFFF
0x10	4	Promedio de bytes por segundo	1 - 0xFFFFFFFF
0x14	2	Alineación de los bloques	1 - 65,535
0x16	2	Bits significativos por muestra	2 - 65,535
0x18	2	Bytes de formato extra	0 - 65,535
0x1a	Contenido de formato extra		

La estructura anterior es la misma que se carga en el registro WAVEFORMATEX del entorno de programación multimedia de Windows. Cuando el archivo guarda sonido en formato descompreso (PCM) la variable FormatTag (o Código de compresión) es igual a 1 (uno) y la variable BytesExtras (Bytes de formato extra) es igual a 0 (cero). Pero cuando el archivo contiene sonido compreso o codificado con algún codec especial, el valor de FormatTag es el código de dicho codec, y el valor de BytesExtras es al número de bytes extras que se necesitan para guardar información especial de cada codec. La lista de Codecs soportados por WAVE y AVI se publica semestralmente en el documento **RFC (**Request for Comments**)** del Network Working Group de Microsoft. En la tabla 5 del capitulo 5, encontramos la lista de algunos formatos existentes.

Entre esos formatos soportados o registrados, esta el MPEG Layer 3. La especificación es la siguiente:

```
MPEG Layer 3

WAVE form Registration Number (hex):    0x0055
Codec ID in the IANA Namespace:         audio/vnd.wave;codec=55
WAVE form wFormatTag ID:                WAVE_FORMAT_MPEGLAYER3
Additional Information:                 ISO/MPEG Layer3 Format Tag
Contact:
Tomislav Grcanac
(408) 576-1361
AT&T Labs, Inc.
2665 North First Street
San Jose, California 95134 USA
```

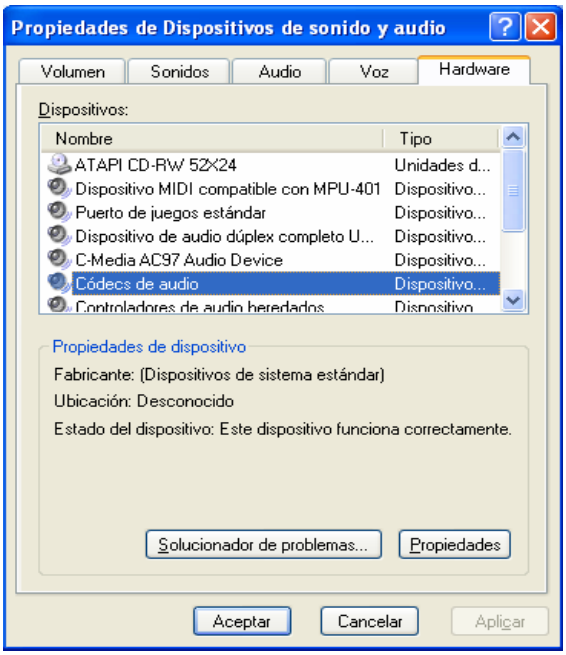
Lo anterior se resume, en que se puede tener un archivo de sonido con formato y estructura WAVE pero compreso con MPEG Layer 3. Para poder decodificar dicho archivo el sistema operativo debe tener un codec ACM instalado que se encargue de la descompresión. Las funciones ACM lo que hacen es crear una interfaz (stream) entre el codec y las aplicaciones.

10.2.2. El codec Fraunhofer IIS MPEG Layer-3

Este codec es el más popular, ya lo distribuye el Instituto Tecnológico Fraunhofer (De los colaboradores de la estandarización MPEG). El codec sólo, no es gratuito, siempre hay que adquirirlo a través de otra aplicación que ya haya pagado por él. Pero igual, muchas aplicaciones de sonido lo incluyen (entre ellas el Reproductor de Windows en sus últimas versiones) y otras aplicaciones lo pueden usar. También se consigue una distribución (no se hasta que punto legal) del codec en un ejecutable instalable.

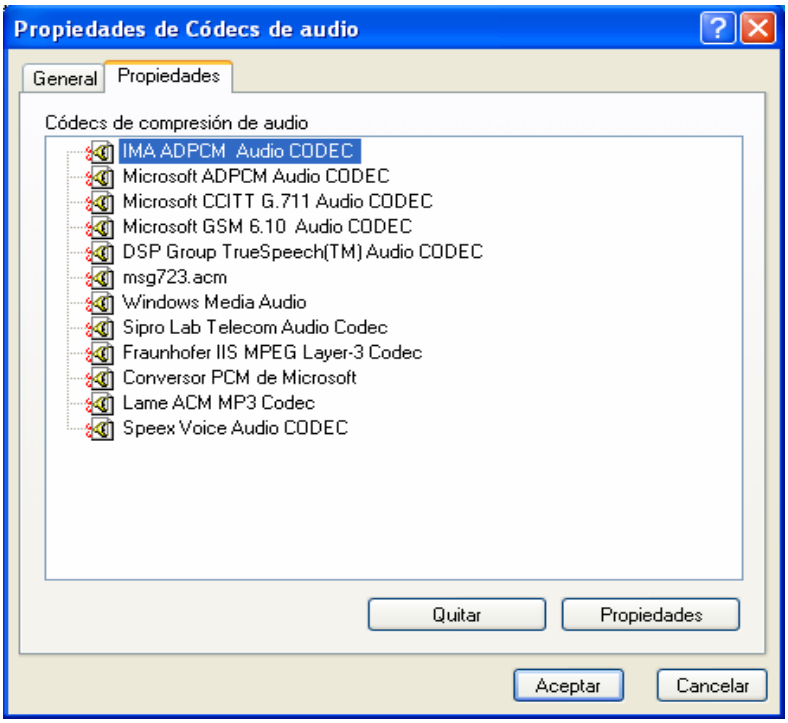
Para saber si el codec está instalado en el PC En Windows Xp, vamos al panel de control, Dispositivos de Sonido y Audio, seleccionamos la ficha Hardware, hacemos clic en “Codecs de Audio” y luego clic en propiedades.

Figura 31: Encontrando los Codecs de Audio en Windows.



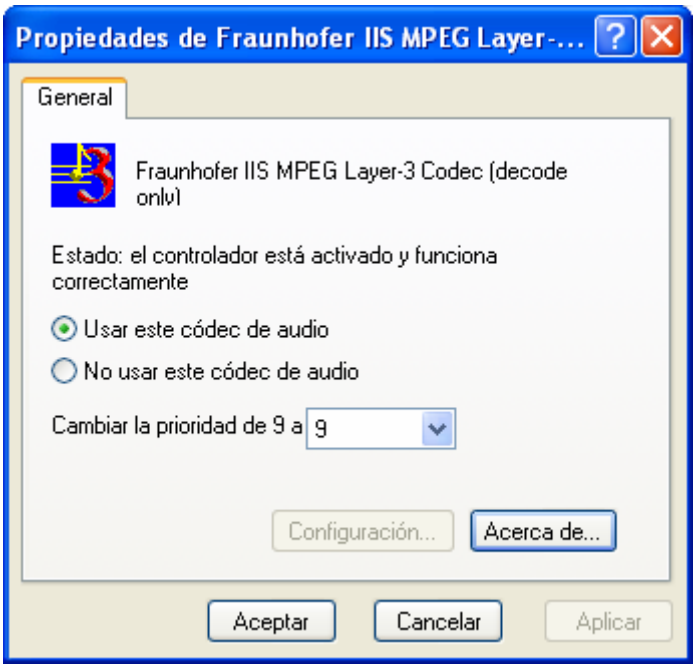
En la ventana que aparece se selecciona la ficha “Propiedades” y se puede ver la lista de codecs de audio instalados en el sistema:

Figura 32: Codecs de Audio Instalados.



Al ver las propiedades del codec, nos aparece la siguiente ventana de información:

Figura 33: Codec Fraunhofer IIS MPEG Layer-3 en Windows.

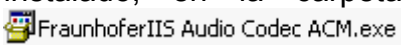


Y al hacer clic en Acerca de… obtenemos:

Figura 34: Codec Fraunhofer IIS MPEG Layer-3 en Windows (Acerca de…).

Como se puede ver, éste codec solo decodifica, es por eso que se tuvo que recurrir a LAME (Encoder Gratuito) para codificar. No se usó el codec ACM de

LAME para codificar porque es más lento y poco confiable (Según comentarios en la Red).

Es muy probable que si en un PC se ha escuchado música desde un archivo MP3, éste tenga instalado el Codec Fraunhofer IIS MPEG Layer-3. Aun así, si no esta instalado, en la carpeta del proyecto MGWave se incluye el archivo 

que instala el codec en el sistema. O, El Reproductor de Windows Media lo descarga de Internet y lo instala automáticamente a lo que se intenta reproducir un wave con el FormatTag=85 (MPEG Layer 3).

10.2.3. Descomprimiendo con ACM

El Administrador de Compresión de Audio de Microsoft puede trabajar con codecs de Compresión y Descompresión, codecs de conversión de formato y codecs de filtros. En la parte de Compresión y Descompresión, lo que hace es cambiar entre tipos de archivo (lo que se necesita, cambiar de PCM a MP3). En la Parte de conversión de formatos, cambia los formatos, pero no los tipos de archivo, por ejemplo, convierte un sonido de 44 kHz, 16 bits, Stereo a uno de 11KHz, 16 bits, Mono. Y en la parte de Filtros, lo que hace es aplicar un filtro (especificado en un driver) a un sonido para producir un sonido diferente (mejorado, sin ruidos, etc.).

Por ahora solo interesa la parte de conversión de tipos de archivos, y como objetivo principal, convertir un archivo mp3 (con extensión .wav pero compreso con MPEG Layer 3) a PCM para poderlo escuchar, graficar su onda y controlarlo adjuntando etiquetas. Una vez implementado el algoritmo que hace uso de las funciones de la ACM para MP3 es muy sencillo hacer adaptaciones para otros codecs (Como GSM, DSP Group TrueSpeech, G.723, G.723 o G.729 “CS-ACELP” usados para transmitir voz por canales de comunicación). El interés sobre el MP3 es más por conocimiento general, pero es sabido, que para comprimir archivos de voz, MP3 no es el mejor codificador. Cabe mencionar que MPEG 4 incluye mejoras en la codificación de voz, pero aun así, son copias de los algoritmos de predicción lineal como el CELP, ACELP y CS-ACELP junto con GSM.

La forma o metodología para descomprimir archivos MP3 es muy parecida a la compresión con LAME, solo que ahora las funciones no las provee una librería especializada (lame_enc.dll) sino la API de Windows sirviendo como traductor entre la aplicación y el driver ACM.

En el proceso de decodificación participan las siguientes funciones, llamándolas en al orden en que aparecen:

1. *acmStreamOpen:* Esta función abre el stream de conversión.
2. acmStreamSize: Calcula los tamaños apropiados tanto del buffer de origen como del buffer de destino.
3. *acmStreamPrepareHeader:* Prepara los buffers de origen y destino que se van a usar en la conversión.
4. *acmStreamConvert:* Convierte grupos de datos tomándolos del buffer de origen al formato de destino y los almacena en el buffer de destino.
5. *acmStreamUnPrepareHeader:* Libera los recursos usados para preparar los buffers.
6. *acmStreamClose:* Cierra el stream abierto.

Detalles sobre las funciones ACM Converter

Estructuras necesarias:

```
Type WAVEFORMATEX
    wFormatTag As Integer       ' Formato
    nChannels As Integer        ' numero de canales
    nSamplesPerSec As Long      ' rata de muestreo
    nAvgBytesPerSec As Long     ' promedio de bytes por segundo
    nBlockAlign As Integer      ' tamaño del bloque de datos
    wBitsPerSample As Integer   ' bits por muestra
    cbSize As Integer           ' número de bytes extras
    xBytes(11) As Byte ' bytes extras, 12 para MPEG Layer 3
End Type
```

La estructura WAVEFORMATEX almacena información relacionada con el formato de los datos de sonido. Como se dijo anteriormente, es la estructura que representa el chunk fmt (formato) en memoria.

```
Type ACMSTREAMHEADER            '
    cbStruct As Long            ' tamaño de la estructura
    dwStatus As Long            ' estado del buffer de conversón
    dwUser As Long              ' datos de usuario
    pbSrc As Long               ' puntero al buffer de origen
    cbSrcLength As Long         ' tamaño del buffer de origen
    cbSrcLengthUsed As Long     ' bytes usados del buffer de origen
    dwSrcUser As Long           ' datos de usuario
    pbDst As Long               ' puntero al buffer de destino
    cbDstLength As Long         ' tamaño del buffer de destino
    cbDstLengthUsed As Long     ' bytes usados del buffer de destino
    dwDstUser As Long           ' datos de usuario
    dwReservedDriver(9) As Long ' reservado, no debe usarse
End Type
```

La estructura ACMSTREAMHEADER es usada para administrar los buffers de datos tanto destino como origen mientras se realizan las conversiones.

***Constantes Necesarias**:*

```
Public Const WAVE_FORMAT_PCM = &H1    ' Microsoft Windows PCM Wave Format
Public Const WAVE_FORMAT_MPEGLAYER3 = &H55  'ISO/MPEG Layer3 Format Tag
```

acmStreamOpen

Declaración:

```
Public Declare Function acmStreamOpen Lib "MSACM32"
(
hAS As Long,
ByVal hADrv As Long,
wfxSrc As WAVEFORMATEX,
wfxDst As WAVEFORMATEX,
ByVal wFltr As Long,
ByVal dwCallback As Long,
ByVal dwInstance As Long,
ByVal fdwOpen As Long
) As Long
```

phas

Devuelve el puntero al manejador del stream abierto.

hADrv

Manejador de un driver ACM. Existen otras funciones no mencionadas aquí, que sirven para buscar los drivers y que devuelven manejadores de esos drivers. La aplicación MGWave supone que el driver adecuado esta instalado en el sistema. Para eso, éste parámetro debe ser NULL.

wfxSrc

Puntero a la estructura WAVEFORMATEX del formato de origen.

wfxDst

Puntero a la estructura WAVEFORMATEX del formato de destino.

wFltr

Puntero a la estructura WAVEFILTER. Usada para trabajar filtros. No usada en éste objetivo, por lo tanto es igual a NULL.

dwCallback

Puntero a la función que controla el callback (para conocer el avance en el tiempo). La función controladora solo es llamada si el stream es abierto con la bandera ACM_STREAMOPENF_ASYN (ver más adelante). Si no se va a usar función callback, éste parámetro debe ser cero.

dwInstance

Instancia de usuario para la función callback.

fdwOpen

Banderas o flags para abrir el stream. Ver la especificación de Microsoft en el MSDN Libray para más detalle. En la aplicación MGWave este parámetro se establece a cero. (Sin flags).

Valores Devueltos

Retorna cero si el stream se abrió con éxito, de lo contrario retorna uno de los siguientes valores:

Valor	**Descripción**
ACMERR_NOTPOSSIBLE	The requested operation cannot be performed.
MMSYSERR_INVALFLAG	At least one flag is invalid.
MMSYSERR_INVALHANDLE	The specified handle is invalid.
MMSYSERR_INVALPARAM	At least one parameter is invalid.
MMSYSERR_NOMEM	The system is unable to allocate resources.

El valor de base de error es 512. Es decir, ACMERR_NOTPOSSIBLE=512+0 y MMSYSERR_INVALFLAG=512+1, y así sucesivamente.

acmStreamSize

Declaración:

```
Public Declare Function acmStreamSize Lib "MSACM32"
(
ByVal hAS As Long,
ByVal cbInput As Long,
dwOutBytes As Long,
ByVal dwSize As Long
) As Long
```

has

Manejador del stream abierto. Es la variable que devuelve acmStreamOpen.

cbInput

Tamaño en bytes, del buffer de origen o de destino. Cuál de los dos es el pasado? Se indica en el parámetro Flags.

pdwOutputBytes

Parámetro que devuelve el tamaño optimo del buffer de origen o destino según el parámetro flgas.

dwSize

Indica cual de los dos tamaños de buffers es pasado, si el de origen (ACM_STREAMSIZEF_SOURCE) o el de destino (ACM_STREAMSIZEF _DESTINATION).

Valores Devueltos

Retorna cero si no se presentaron errores, de lo contrario retorna uno de los siguientes valores:

Value	**Description**
ACMERR_NOTPOSSIBLE	The requested operation cannot be performed.
MMSYSERR_INVALFLAG	At least one flag is invalid.
MMSYSERR_INVALHANDLE	The specified handle is invalid.
MSYSERR_INVALPARAM	At least one parameter is invalid.

acmStreamPrepareHeader

Declaración:

```
Public Declare Function acmStreamPrepareHeader Lib "MSACM32"
(
ByVal hAS As Long,
hASHdr As ACMSTREAMHEADER,
ByVal dwPrepare As Long
) As Long
```

has

Manejador del stream abierto.

hASHdr

Puntero a la estructura ACMSTREAMHEADER que se va a preparar.

fdwPrepare

Campo reservado, debe ser cero.

Valores Devueltos

Retorna cero si no se presentaron errores, de lo contrario retorna uno de los siguientes valores:

Value	**Description**
MMSYSERR_INVALFLAG	At least one flag is invalid.
MMSYSERR_INVALHANDLE	The specified handle is invalid.
MMSYSERR_INVALPARAM	At least one parameter is invalid.
MMSYSERR_NOMEM	The system is unable to allocate resources.

acmStreamConvert

Declaración:

```
Public Declare Function acmStreamConvert Lib "MSACM32"
(
ByVal hAS As Long,
hASHdr As ACMSTREAMHEADER,
ByVal dwConvert As Long
) As Long
```

has

Manejador del stream abierto.

hASHdr

Puntero al ACMSTREAMHEADER preparado devuelto por la función acmStreamPrepareHeader.

dwConvert

Banderas de conversión. Ver la especificación de Microsoft en el MSDN Libray para más detalle. En la aplicación MGWave este parámetro se establece a cero. (Sin flags).

Valores Devueltos

Retorna cero si no se presentaron errores, de lo contrario retorna uno de los siguientes valores:

Value	**Description**
ACMERR_BUSY	The stream header specified in *pash* is currently in use and cannot be reused.
ACMERR_UNPREPARED	The stream header specified in *pash* is currently not prepared by the acmStreamPrepareHeader function.
MMSYSERR_INVALFLAG	At least one flag is invalid.
MMSYSERR_INVALHANDLE	The specified handle is invalid.
MMSYSERR_INVALPARAM	At least one parameter is invalid.

10.2.4. El código de descompresión de MGWave

El siguiente es el código de la función Descomprimir MP3 del programa MGWave:

```
Sub DescomprimirMP3(ByVal BytesAdicionales() As Byte, ByRef DataFmt As
InfoFormato, ByVal BufferData() As Byte, ByVal BufferInt() As Integer)

    Dim i As Long
    Dim Max, t As Long

    Dim rc As Long, hAS As Long

    Dim WFPCM As WAVEFORMATEX 'Wave Format PCM
    Dim WFMP3 As WAVEFORMATEX 'Wave Format MP3
```

```
Dim WMP3StreamH As ACMSTREAMHEADER  'Wave MP3 Setream Header

Dim PCMBuffSize As Long
Dim MP3BuffSize As Long

Dim Mp3Buff As Long  'buffers en Memoria
Dim PCMBuff As Long

Dim hBMP3 As Long  'Cabecera de Buffer MP3 en memoria global (GlobalAlloc)
Dim hBPCM As Long  'Cabecera de Buffer PCM en memoria global (GlobalAlloc)

Dim PBMP3 'Puntero a Buffer MP3 en memoria global (GlobalLock)
Dim PBPCM 'Puntero a Buffer PCM en memoria global (GlobalLock)

Dim BufferPCM() As Byte

If DataFmt.CodigoCompresion = WAVE_FORMAT_MPEGLAYER3 Then

    WFMP3.nAvgBytesPerSec = DataFmt.PromedioBitsPorSegundo
    WFMP3.nSamplesPerSec = DataFmt.MuetrasPorSegundo
    WFMP3.wBitsPerSample = 0
    WFMP3.nBlockAlign = 1
    WFMP3.nChannels = DataFmt.Canales
    WFMP3.wFormatTag = DataFmt.CodigoCompresion
    WFMP3.cbSize = 12

    For i = 0 To 11
        WFMP3.xBytes(i) = BytesAdicionales(i)
    Next

    MP3BuffSize = 2048

    WFPCM.wFormatTag = WAVE_FORMAT_PCM
    WFPCM.nChannels = 1
    WFPCM.nSamplesPerSec = 11025
    WFPCM.wBitsPerSample = 16
    WFPCM.nBlockAlign = 2
    WFPCM.nAvgBytesPerSec = WFPCM.nSamplesPerSec * WFPCM.nBlockAlign
    WFPCM.cbSize = 0

    FormatOriginal = DataFmt

    DataFmt.PromedioBitsPorSegundo = WFPCM.nAvgBytesPerSec
    DataFmt.CodigoCompresion = WAVE_FORMAT_PCM
    DataFmt.BitsPorMuestra = WFPCM.wBitsPerSample
    DataFmt.BlockAlign = WFPCM.nBlockAlign
    DataFmt.MuetrasPorSegundo = WFPCM.nSamplesPerSec
    DataFmt.Canales = WFPCM.nChannels

    rc = acmStreamOpen(hAS, 0&, WFMP3, WFPCM, 0&, 0&, 0&, 0)

    rc = acmStreamSize(hAS, MP3BuffSize, PCMBuffSize, ACM_STREAMSIZEF_SOURCE)

    hBMP3 = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or GMEM_ZEROINIT,
                        MP3BuffSize)
    PBMP3 = GlobalLock(hBMP3)

    hBPCM = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or GMEM_ZEROINIT,
                        PCMBuffSize)
```

```
PBPCM = GlobalLock(hBPCM)


WMP3StreamH.cbStruct = Len(WMP3StreamH)
WMP3StreamH.pbSrc = PBMP3
WMP3StreamH.cbSrcLength = MP3BuffSize
WMP3StreamH.pbDst = PBPCM
WMP3StreamH.cbDstLength = PCMBuffSize

WMP3StreamH.dwStatus = 0
WMP3StreamH.dwUser = 0
WMP3StreamH.cbSrcLengthUsed = 0
WMP3StreamH.cbDstLengthUsed = 0

rc = acmStreamPrepareHeader(hAS, WMP3StreamH, 0&)

i = 0
Max = UBound(BufferData) - 1
ReDim BufferPCM(0)

Do
    If i + MP3BuffSize < Max Then
        Call CopyBYTEStoPTR(PBMP3, BufferData(i), MP3BuffSize)
        i = i + MP3BuffSize
    Else
        Call CopyBYTEStoPTR(PBMP3, BufferData(i), Max - i)
        i = 0
    End If

    rc = acmStreamConvert(hAS, WMP3StreamH, ACM_STREAMCONVERTF_BLOCKALIGN)

    t = UBound(BufferPCM)
    ReDim Preserve BufferPCM(t + WMP3StreamH.cbDstLengthUsed)
    Call CopyPTRtoBYTES(BufferPCM(t), PBPCM, WMP3StreamH.cbDstLengthUsed)
Loop Until i = 0

rc = acmStreamUnprepareHeader(hAS, WMP3StreamH, 0&)

rc = GlobalUnlock(PBMP3)
rc = GlobalFree(hBMP3)
rc = GlobalUnlock(PBPCM)
rc = GlobalFree(hBPCM)

rc = acmStreamClose(hAS, 0&)

Dim HBuffTmp As Long
Dim PBuffTmp As Long
Dim Tam As Long

'para pasar de array de bytes a array de enteros
Tam = UBound(BufferPCM) - 1
HBuffTmp = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or GMEM_ZEROINIT, Tam)
PBuffTmp = GlobalLock(HBuffTmp)
Call CopyBYTEStoPTR(PBuffTmp, BufferPCM(0), Tam)
ReDim BufferInt(Tam / 2)
Call CopyPTRtoWORD(BufferInt(0), PBuffTmp, Tam)
rc = GlobalUnlock(PBuffTmp)
rc = GlobalFree(HBuffTmp)
BufferOriginal = BufferData
BufferData = BufferPCM
```

```
        AbrioMP3 = True
    Else
        ReDim BufferPCM(UBound(BufferData))
        BufferData = BufferPCM
    End If

End Sub
```

La función recibe los bytes (array de bytes) de formato extras leídos del archivo .wav junto con la estructura de información de Formato (WAVEFORMATEX personalizada). Recibe los datos a decodificar en el parámetro BufferData, y devuelve los datos ya decodificados en la misma variable. BufferInt es un buffer auxiliar para facilitar el dibujado de la onda con muestras de 16 bits. El código fue una adaptación de [42].

10.3. ALGUNOS CODECS SOPORTADOS POR WINDOWS

Tabla 24: Codecs Soportados por windows según [43].

Codec de audio	Descripción
CCITT A-Law	El códec Microsoft International Telecommunications Union (ITU) CCITT A-Law proporciona compatibilidad con los estándares TAPI (Interfaz de programación de aplicaciones de telefonía) para Europa. También denominado G.711, este códec es compatible con muchas configuraciones de hardware y ofrece una razón de compresión de 2:1 (de 16 bits a 8 bits por muestra).
CCITT u-Law	Igual que el códec CCITT A-Law, pero proporciona compatibilidad con los estándares TAPI que se utilizan en América del Norte.
DSP Group TrueSpeech	El códec DSP Group TrueSpeech proporciona buena compresión del sonido orientado a voz y está diseñado para sonido de voz con una velocidad de bits de baja a media. TrueSpeech ofrece una velocidad de datos mejor que GSM 6.10, el otro códec de audio orientado a voz suministrado con Windows 2000. TrueSpeech es una buena elección para grabar notas en documentos u hojas de cálculo o para almacenar correo de voz en el equipo, pero no está diseñado para sonido que no sea voz. TrueSpeech no ofrece relaciones de compresión en tiempo real sino relaciones de descompresión en tiempo real, lo que convierte a este códec en una buena alternativa para utilizarlo con módems y redes.
GSM 6.10	El códec Microsoft Groupe Spécial Mobile (GSM) está diseñado para la compresión eficaz de voz y resulta adecuado para el sonido orientado a voz con velocidades de bits de media a alta.

	GSM proporciona compresión 2:1 en tiempo real (siempre que el hardware sea la suficientemente rápido para admitirlo), lo que convierte a este códec en una buena opción para grabar voz con la Grabadora de sonidos. Si utiliza GSM para comprimir música puede obtener una calidad de sonido deficiente. GSM es conforme a la recomendación 6.10 del European Telecommunications Standards Institute.
IMA ADPCM	El códec Microsoft Interactive Multimedia Association (IMA) ADPCM está diseñado para múltiples plataformas de hardware y ofrece compresión de alta calidad en tiempo real con alto contenido de velocidad de bits. IMA ADPCM es parecido a Microsoft ADPCM, pero proporciona compresión 4:1 más rápida.
Lernout & Hauspie CELP 4,8 kbps	El códec Lernout & Hauspie CELP está diseñado para utilizarlo con voz y es adecuado para el sonido orientado a voz con velocidad de bits baja. Si utiliza Lernout & Hauspie CELP para comprimir música puede obtener una calidad de sonido deficiente. Este códec sólo tiene una configuración, que lo comprime todo a 4,8 kbps. El audio comprimido de esta forma no tiene la riqueza asociada al audio de calidad de CD; sin embargo, el audio de voz, como cuando habla alguien, sonará bien porque Lernout & Hauspie CELP está diseñado para trabajar mejor con fuentes de audio menos dinámicas. Lernout & Hauspie CELP resulta útil cuando se desea tener ancho de banda suficiente para enviar otro tipo de información por la red como vídeo, imágenes o secuencias de comandos.
Lernout & Hauspie SBC 8 kbps	Parecido al códec Lernout & Hauspie CELP 4,8 kbps.
Lernout & Hauspie SBC 12 kbps	Parecido al códec Lernout & Hauspie CELP 4,8 kbps.
Lernout & Hauspie SBC 16 kbps	Parecido al códec Lernout & Hauspie CELP 4,8 kbps.
Microsoft ADPCM	El códec Microsoft Adaptive Delta Pulse Code Modulation (ADPCM) proporciona compresión 4:1 de alta calidad, en tiempo real o no, y es adecuado para secuencias de audio asociadas a vídeo de alta velocidad de bits.
Microsoft G.723.1	El códec Microsoft G.723.1 está diseñado para crear archivos de transmisión activa (.asf) o transmisiones de audio ASF para utilizarlas en Internet o con el Reproductor de Windows Media. El estándar G.723.1 especifica el formato y el algoritmo para enviar o recibir voz en conexiones de red con ancho de banda bajo. Los

	archivos comprimidos con Microsoft G.723.1 pueden contener varios tipos de datos además de sólo audio y admite muchas combinaciones de compresión. Microsoft G.723.1 funciona muy bien en conexiones de red a baja velocidad de datos, como 14,4 o 28,8 kbps.
MPEG Layer-3	El códec Fraunhofer Institut Integrierte Schaltungen IIS (FhG) MPEG Layer-3 está diseñado para crear archivos de música con calidad de CD a una velocidad de bits de baja a media para utilizarlos en una intranet o en Internet. El códec MPEG Layer-3 ofrece una excelente compresión de alta fidelidad para muchos tipos distintos de audio y es una buena elección cuando se necesita una velocidad de bits alta y un uso de CPU bajo.
PCM	El convertidor Microsoft Pulse-Code Modulation (PCM) proporciona audio sin comprimir para contenidos de velocidad de bits mayor. Con PCM se puede reproducir una muestra de audio a una velocidad de un kilohercio en una tarjeta de sonido que admita otra velocidad. Por ejemplo, las tarjetas de 8 bits pueden reproducir muestras de 16 bits mediante la reducción de la calidad del audio para que se corresponda con las capacidades de la tarjeta de sonido.
VivoActive G.723.1	El códec VivoActive G.723.1 es adecuado para crear archivos de audio con una velocidad de bits baja de propósito general, archivos de formato de transmisión activa (.asf) o transmisiones de audio ASF para utilizarlos en una intranet. El estándar G.723.1 especifica el formato y el algoritmo para enviar o recibir voz en conexiones de red con ancho de banda bajo.
VivoActive Siren	El códec VivoActive Siren es adecuado para crear archivos de audio con una velocidad de bits baja de propósito general.
Voxware MetaSound	El códec Voxware MetaSound está diseñado para crear archivos de música con una velocidad de bits de baja a media, que van de mono a 6 kbps hasta estéreo de alta fidelidad a 96 kbps. Este códec ofrece una excelente compresión de alta fidelidad para muchos tipos distintos de audio. Sin embargo, MPEG Layer-3 es una buena alternativa si precisa un velocidad de bits alta y un uso de CPU bajo.
Voxware MetaVoice	El códec Voxware MetaVoice proporciona buen rendimiento para pistas de audio de sólo voz a una velocidad de bits muy baja. Con la configuración RT24 a 2,4 kbps, el códec MetaVoice comprime el audio más que ninguno de los otros códecs instalados con el Reproductor de Windows Media.

11. DESCRIPCION DE OTROS PROGRAMAS DE LA PRÁCTICA EMPRESARIAL

11.1. PROGRAMA DE REPRODUCCIÓN EN TIEMPO REAL

MGWave reproduce el sonido basado en Direct Sound, lo que trae como inconveniente que toda la onda de sonido tenga que ser decodificada y guardada en memoria antes de poderla escuchar. Desde el punto de vista del proyecto de Contact Center de TYT Ltda. éste método no es funcional. Se trata de reproducir sonidos que vienen codificados (p. ejem. GSM) y empaquetados por pedazos a través de una red IP. Por tanto, se necesita ir decodificando y reproduciendo en tiempo real. Queda entonces desechado Direct Sound. Hay que ir más abajo: Multimedia de Bajo nivel.

Se elaboró durante la quinta y última fase de la práctica empresarial un programa en Visual Basic 6.0 que logra reproducir sonidos codificados o no, en tiempo real. Mediante éste método se logra un control total sobre los dispositivos de sonido. El programa en su núcleo, envía los datos ya decodificados (con ACM – Audio Compression Manager) a la tarjeta de sonido mediante una función de Win32 (waveOutWrite). Dicha función, en su mejor empleo (de varios posibles) necesita una función de retorno (CallBack) para indicarle al programa que la llama, cuando termina de reproducirse el bloque de sonido enviado. Lo anterior es un gran problema para Visual Basic, en especial a la hora de la depuración, pero mientras no se depure el programa funciona. Cuando una función controladora de llamadas (callback) se encuentra en una clase de Visual Basic (Método SubClassing), VB genera errores de “dead lock” entre los procesos. Por tanto no es posible lograr que un componente ActiveX DLL (basado en una clase) de Visual Basic implemente correctamente las funciones de sonido de bajo nivel de la API. La solución a esto, es crear el componente en Visual C++ que si soporta el SubClassing.

Los dos objetivos principales de éste programa son:

- Mostrar las funciones de la API de Win32 para el sonido a bajo nivel (API MLL Multimedia Low Level) y la potencia de éstas.
- Probar lo investigado en cuanto al formato del archivo MP3. (Frame Header)

11.1.1. Alcances del Programa

- Permite reproducir archivos wave (.wav) guardados en cualquier formato, es decir, con cualquier código de compresión (Format tag), Por ejemplo, GSM 6.10, ADPCM, G. 723, ACELP, MPEG Layer III, etc, siempre y cuando, el códec se encuentre instalado en el sistema.
- Permite reproducir archivos MPEG Layer III (.mp3) siempre y cuando esté instalado el codec apropiado en el sistema (Por ejemplo Fraunhofer IIS MPEG Layer III codec (decode only)).

Figura 35: Ventana principal del programa de reproducción en tiempo real.

11.1.2. Funcionamiento del Programa

11.1.2.1. Abriendo el Archivo

Para poder reproducir archivos de sonido en tiempo real y de la manera más óptima en cuanto a memoria, primero se debe abrir el archivo. Ya que todas las funciones de la API en cuanto a MLL y ACM funcionan con buffers mapeados en memoria, es necesario abrir el archivo con funciones que devuelvan punteros a éste en memoria. Algunas funciones de la API que permiten esto están en la librería Kernel32.dll y son:

- *CreateFile:* Crea o abre un archivo o dispositivo del sistema y devuelve un controlador de archivo que se usa en funciones como ReadFile, WriteFile, CloseHandle, etc para referenciar al archivo abierto.
- *ReadFile*: Lee los datos del archivo abierto y los copia en memoria según el puntero al buffer pasado como parámetro.

- *SetFilePointerEx:* Se mueve en el archivo abierto.
- *CloseHandle*: Cierra un archivo abierto con CreateFile.
- *GetFileSize*: Devuelve el tamaño de un archivo abierto.

Las funciones anteriores permiten abrir cualquier archivo, y en el programa son usadas especialmente para abrir los archivos .MP3. Para el caso de los archivos wave es más conveniente usar las funciones de la API para entrada y salida de datos multimedia (mmio – Multimedia Input/Output), ésta es una forma alternativa a como se hacía en el programa MGWave donde se usaba un algoritmo repetitivo para leer todos los chunks del archivo. Las Funciones Usadas son las siguientes:

- *mmioOpen*: abre un archivo multimedia RIFF y devuelve un controlador.
- *mmioRead*: lee y pasa a la memoria los datos de un archivo abierto con mmioOpen.
- *mmioSeek*: se posiciona en cualquier parte del archivo.
- *mmioStringToFOURCC:* devuelve el código que corresponde al nombre del chunk pasado como parámetro.
- *mmioClose:* cierra un archivo multimedia abierto con mmioOpen.
- *mmioDescend: o*btiene el chunk cuyo código coincida con el parámetro requerido.
- *mmioDescendParent:* obtiene el chunk parent RIFF.

Si el archivo tiene extensión MP3, es necesario implementar el algoritmo que lea los datos de éste basándose en su formato.

El código de la función abrir archivo es el siguiente:

```
Private Function AbrirArchivo() As Boolean

        Dim mmioInfo As mmioInfo
        Dim ChunkWAVE As MMCKINFO  'Chunk Info
        Dim AnyChunk As MMCKINFO
        Dim sFormat As String
        Dim ext As String

        AbrirArchivo = True

        ext = UCase(Right(Archivo, 3))

        If ext = "WAV" Then

            hFile = mmioOpen(Archivo, mmioInfo, MMIO_READ)

            ChunkWAVE.fccType = mmioStringToFOURCC("WAVE", 0)
            rc = mmioDescendParent(hFile, ChunkWAVE, 0, MMIO_FINDRIFF)
```

```
            AnyChunk.ckid = mmioStringToFOURCC("fmt", 0)
            rc = mmioDescend(hFile, AnyChunk, ChunkWAVE, MMIO_FINDCHUNK)

            sFormat = String(50, 0)
            rc = mmioReadString(hFile, sFormat, AnyChunk.ckSize)
            rc = mmioAscend(hFile, AnyChunk, 0)
            CopyMemoryFromString(waveFmt, sFormat, AnyChunk.ckSize)

            AnyChunk.ckid = mmioStringToFOURCC("data", 0)
            rc = mmioDescend(hFile, AnyChunk, ChunkWAVE, MMIO_FINDCHUNK)

            InicioData = mmioSeek(hFile, 0, SEEK_CUR)
            TamañoData = AnyChunk.ckSize

            LengthBytes = TamañoData
            PosBytes = 0
            FrecuenciaMuestreo = waveFmt.nSamplesPerSec
            NumCanales = waveFmt.nChannels
            BitsPorMuestra = waveFmt.wBitsPerSample

        ElseIf ext = "MP3" Then

            GetMP3FileInfo()
            MP3 = True

            hFile = CreateFile(Archivo, GENERIC_READ, FILE_SHARE_READ,_
            ByVal 0&, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
        End If
    End Function
```

La función GetMP3FileInfo carga la estructura WAVEFORMATEX con los datos apropiados a partir de la información que provee el primer frame encontrado en el archivo MP3. Esto implica que el archivo MP3 tiene un Bitrate constante CBR. La función GetMP3FileInfo se desglosa en varias funciones que obtienen de los datos en forma binaria información como: Versión del MPEG, Layer, Bitrate, Frecuencia de Muestreo y Número de canales.

El código de la función GetMP3FileInfo es el siguiente:

```
    Sub GetMP3FileInfo()
        Dim FrameHdrMP3 As String
        Dim Sinc As Integer  'sincronización
        Dim lee(1) As Byte
        Dim lee2(3) As Byte
        Dim lee3 As String * 3
        Dim f As Integer
        Dim Encontro As Boolean
        Dim P As Long
        Dim LenFile As Long
        Dim Version As Single
        Dim Layer As Integer
```

```
f = FreeFile
Open Archivo For Binary Access Read As #f
Encontro = False

Do While Not Encontro And Not EOF(f)
    Get #f, , lee
    P = Seek(f)
    If lee(0) = 255 Then
        If Left(DecimalBinar(lee(1)), 3) = "111" Then
            Encontro = True
        End If
    End If
    Seek #f, P - 1
Loop

'Encontro se refiere a si encontró los bits de
'sincronización de inicio de un frame mpeg, éstos
'no se encuentran el principio del archivo, si éste
'contiene Tags ID3 v2

If Encontro Then
    InicioData = P - 2
    Seek #f, InicioData
    Get #f, , lee2
    FrameHdrMP3 = DecimalBinar(lee2(0))
    FrameHdrMP3 = FrameHdrMP3 & DecimalBinar(lee2(1))
    FrameHdrMP3 = FrameHdrMP3 & DecimalBinar(lee2(2))
    FrameHdrMP3 = FrameHdrMP3 & DecimalBinar(lee2(3))
    LenFile = LOF(f)
    Seek #f, LenFile - 127
    Get #f, , lee3
    If lee3 = "TAG" Then
        TamañoData = LenFile - InicioData - 128
        'quita los tags ID3 v1
    Else
        TamañoData = LenFile - InicioData
    End If

    LengthBytes = TamañoData

    Version = GetVersionMP3(Mid(FrameHdrMP3, 12, 2))
    Layer = GetLayerMP3(Mid(FrameHdrMP3, 14, 2))
    BitrateMP3 = GetBitRateMP3(Mid(FrameHdrMP3, 17, 4), Version, Layer)
    FrecuenciaMuestreo = GetFMMP3(Mid(FrameHdrMP3, 21, 2), CSng(Version))
    NumCanales = GetCanalesMP3(Mid(FrameHdrMP3, 25, 2))
    BitsPorMuestra = 0

    waveFmt.cbSize = 12
    waveFmt.nAvgBytesPerSec = BitrateMP3 * (1000 / 8)
    waveFmt.nBlockAlign = 1
    waveFmt.nChannels = NumCanales
    waveFmt.nSamplesPerSec = FrecuenciaMuestreo
    waveFmt.wBitsPerSample = 0
    waveFmt.wFormatTag = 85
```

```
            waveFmt.xBytes(0) = 1
            waveFmt.xBytes(1) = 0
            waveFmt.xBytes(2) = 2
            waveFmt.xBytes(3) = 0
            waveFmt.xBytes(4) = 0
            waveFmt.xBytes(5) = 0
            waveFmt.xBytes(6) = 161
            waveFmt.xBytes(7) = 1
            waveFmt.xBytes(8) = 1
            waveFmt.xBytes(9) = 0
            waveFmt.xBytes(10) = 113
            waveFmt.xBytes(11) = 5
        End If
    Close #f
    End Sub
```

11.1.2.2. Inicializando Datos y Estructuras

Una vez abierto el archivo, se tiene una referencia a él en un puntero *hFile.* Como el archivo puede estar codificado es necesario hacer uso de las funciones de la ACM mencionadas en el numeral 10.2.3:

- acmStreamOpen
- acmStreamClose
- acmStreamPrepareHeader
- acmStreamUnprepareHeader
- acmStreamConvert
- acmStreamReset
- acmStreamSize

Cuando tenemos los datos de sonido en la memoria listos y decodificados (en formato PCM) los enviamos a la tarjeta de sonido haciendo uso de las siguientes funciones:

- *waveOutOpen:* Abre para salida el dispositivo de sonido seleccionado. Si el dispositivo seleccionado es WAVE_MAPPER, la función selecciona el dispositivo más apropiado para la salida según el WAVEFORMATEX pasado como parámetro. En ésta función también se indica la forma en que se va a interactuar con las funciones de la API MLL:

    - Esperando a que se termine de reproducir el sonido: En esta forma, los programas deben implementar ciclos donde, una vez llamada la función de salida a la tarjeta de sonido waveOutWrite, se pregunte constantemente por el estado de las banderas que se activan una vez el

sonido termine de reproducirse. Es un método poco usado debido al alto uso de la CPU mientras se reproduce el sonido.

- ❖ Obteniendo la posición actual del buffer de salida: se trata de conocer a cada instante en que posición se encuentra el sonido enviado. Se logra con la función WaveOutGetPosition. Al igual que el método anterior ocupa demasiado la CPU.
- ❖ Usando mensajes devueltos por la API MLL a la función CallBack: Es la forma más óptima. Se puede realizar de dos maneras: Procesando los mensajes con un Windows Form o Procesando los mensajes con una función callback. Para la segunda, (la utilizada en el programa), es necesario pasarle como parámetro la dirección de la función callback a la función waveOutOpen, esto lo se logra mediante el operador *addressof* de Visual Basic.

- *waveOutPrepareHeader:* prepara los dispositivos para salida de acuerdo a la estructura WAVEHDR pasada como parámetro. La estructura WAVEHDR contiene información sobre los buffers de salida como: dirección en memoria, manejador o *handle*, tamaño y banderas que indican su estado.
- *waveOutUnprepareHeader:* elimina los recursos utilizados por los buffers de sonido.
- *waveOutWrite:* Envía al dispositivo de sonido seleccionado los datos contenidos en el buffer de sonido apuntados por la estructura WAVEHDR pasada como parámetro. Activa la bandera WHDR_DONE de WAVEHDR y envía el mensaje MM_WOM_DONE a la función callback, cuando el buffer enviado termina de reproducirse.
- *waveOutClose:* Cierra el dispositivo multimedia abierto por waveOutOpen.
- *waveOutReset, waveOutPause, waveOutRestart:* Controlan la reproducción de un buffer de sonido enviado recientemente con waveOutWrite.

Técnica de Double-Buffering:

Ésta técnica se mencionó en el numeral 10.1.2.3. y es la implementada en éste caso para lograr escuchar el sonido sin pausas ni alteraciones. En el programa se crean dos buffers de salida, mientras uno es escuchado, el otro se está cargando de disco a memoria, para que una vez el primero termine, el segundo esté listo para escucharse y no se noten saltos. Se puede jugar con el tamaño de los buffers, todo depende de la velocidad del procesador. Dos buffers con tamaño grande (0.5 segundos) son suficientes hasta para los procesadores más lentos, pero la latencia es muy alta para el control de los programas, es decir, si no se desea avanzar en el sonido cada 500 ms sino cada 100 ms, para medir tiempos más exactos. En la realidad, no se usan dos buffers, sino arrays de buffers de tamaño pequeño, así, la latencia se disminuye lo suficiente y lo que varía según la velocidad del procesador es el tamaño del array.

El código de la función que inicializa todo lo mencionado anteriormente es el siguiente:

```
Sub IniciarTodo()


        cWavefmt.wFormatTag = WAVE_FORMAT_PCM
        cWavefmt.nChannels = waveFmt.nChannels
        cWavefmt.nSamplesPerSec = waveFmt.nSamplesPerSec
        If waveFmt.wBitsPerSample = 0 Then
            cWavefmt.wBitsPerSample = 16
        Else
            cWavefmt.wBitsPerSample = waveFmt.wBitsPerSample
        End If
        cWavefmt.nBlockAlign = cWavefmt.nChannels * cWavefmt.wBitsPerSample / 8
        cWavefmt.nAvgBytesPerSec = cWavefmt.nSamplesPerSec * cWavefmt.nBlockAlign
        cWavefmt.cbSize = 0


        rc = waveOutOpen(hWaveOut, WAVE_MAPPER, waveFmt, AddressOf waveOutProc,_
                         0&, CALLBACK_FUNCTION)

        BuffSize = waveFmt.nAvgBytesPerSec * 0.2 ' 200 ms

        'WaveOutHdr y WaveOutHDR_x controlan los dos buffers (double buffering)

        WaveOutHdr.hData = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or _
                                       GMEM_ZEROINIT, BuffSize)
        WaveOutHdr.lpData = GlobalLock(WaveOutHdr.hData)
        WaveOutHdr.dwBufferLength = BuffSize
        WaveOutHdr.dwFlags = 0

        WaveOutHdr_x.hData = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or_
                                         GMEM_ZEROINIT, BuffSize)

        WaveOutHdr_x.lpData = GlobalLock(WaveOutHdr_x.hData)
        WaveOutHdr_x.dwBufferLength = BuffSize
        WaveOutHdr_x.dwFlags = 0

        rc = waveOutPrepareHeader(hWaveOut, WaveOutHdr_x, Len(WaveOutHdr_x))
        rc = waveOutPrepareHeader(hWaveOut, WaveOutHdr, Len(WaveOutHdr))

        PlayPosDataFile = InicioData

        'Los dos ciclos esperan a que las estructuras WAVEHDR sean preparadas

        Do Until (((WaveOutHdr_x.dwFlags And WHDR_PREPARED) = WHDR_PREPARED) Or _
                  (WaveOutHdr_x.dwFlags = WHDR_PREPARED) Or _
                  (WaveOutHdr_x.dwFlags = 0))
            DoEvents()
        Loop

        Do Until (((WaveOutHdr.dwFlags And WHDR_PREPARED) = WHDR_PREPARED) Or _
                  (WaveOutHdr.dwFlags = WHDR_PREPARED) Or _
                  (WaveOutHdr.dwFlags = 0))
            DoEvents()
        Loop
```

```
    rc = acmStreamOpen(hAS, 0&, waveFmt, cWavefmt, 0&, 0&, 0&,_
                       ACM_STREAMOPENF_NONREALTIME)

    '== Init ACM Header ================================================

    acmHdr.cbStruct = Len(acmHdr)
    acmHdr.dwStatus = 0
    acmHdr.dwUser = 0
    acmHdr.cbSrcLengthUsed = 0
    acmHdr.cbDstLengthUsed = 0
    acmHdr.pbSrc = WaveOutHdr.lpData
    acmHdr.cbSrcLength = WaveOutHdr.dwBufferLength

    rc = acmStreamSize(hAS, acmHdr.cbSrcLength, acmHdr.cbDstLength, _
                       ACM_STREAMSIZEF_SOURCE)

    acmHdr.dwDstUser = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or _
                                   GMEM_ZEROINIT, acmHdr.cbDstLength)
    acmHdr.cbDst = GlobalLock(acmHdr.dwDstUser)

    '==Init ACM Header X ================================================

    acmHdr_x.cbStruct = Len(acmHdr_x)
    acmHdr_x.dwStatus = 0
    acmHdr_x.dwUser = 0
    acmHdr_x.cbSrcLengthUsed = 0
    acmHdr_x.cbDstLengthUsed = 0
    acmHdr_x.pbSrc = WaveOutHdr_x.lpData
    acmHdr_x.cbSrcLength = WaveOutHdr_x.dwBufferLength

    rc = acmStreamSize(hAS, acmHdr_x.cbSrcLength, acmHdr_x.cbDstLength, _
                       ACM_STREAMSIZEF_SOURCE)

    acmHdr_x.dwDstUser = GlobalAlloc(GMEM_MOVEABLE Or GMEM_SHARE Or _
                                     GMEM_ZEROINIT, acmHdr_x.cbDstLength)
    acmHdr_x.cbDst = GlobalLock(acmHdr_x.dwDstUser)

    '====================================================================

    rc = acmStreamPrepareHeader(hAS, acmHdr, 0&)
    rc = acmStreamPrepareHeader(hAS, acmHdr_x, 0&)


    Do Until (((acmHdr_x.dwStatus And ACMSTREAMHEADER_STATUSF_PREPARED) = _
                ACMSTREAMHEADER_STATUSF_PREPARED) Or _
               (acmHdr_x.dwStatus = 0))
        DoEvents()
    Loop

    Do Until (((acmHdr.dwStatus And ACMSTREAMHEADER_STATUSF_PREPARED) = _
                ACMSTREAMHEADER_STATUSF_PREPARED) Or _
               (acmHdr.dwStatus = 0))
        DoEvents()
    Loop

End Sub
```

11.1.2.3. Reproduciendo el sonido

Una vez preparados los buffers e inicializados los datos necesarios se procede a empezar a reproducir el sonido. La función del programa encargada es *Play* que hace uso de la función waveOutWrite mencionada anteriormente. La función Play sola, no logra nada, necesita de la función controladora de la llamada de retorno (callback) *waveOutProc* definida en Visual Basic. Los códigos son los siguientes:

```
Public Sub Play()

    If Not MP3 Then
        rc = mmioRead(hFile, WaveOutHdr_x.lpData,
                      WaveOutHdr_x.dwBufferLength)
    Else
        rc = ReadFile(hFile, ByVal WaveOutHdr_x.lpData, _
                       WaveOutHdr_x.dwBufferLength, numread, ByVal 0&)
    End If

    PlayPosDataFile = PlayPosDataFile + WaveOutHdr_x.dwBufferLength
    rc = acmStreamConvert(hAS, acmHdr_x, ACM_STREAMCONVERTF_BLOCKALIGN)
    rc = waveOutWrite(hWaveOut, WaveOutHdr_x, Len(WaveOutHdr_x))

    BufferAct = 0
    BufferOut = 0

    If Not MP3 Then
        rc = mmioRead(hFile, WaveOutHdr.lpData, WaveOutHdr.dwBufferLength)
    Else
        rc = ReadFile(hFile, ByVal WaveOutHdr.lpData, _
                       WaveOutHdr.dwBufferLength, numread, ByVal 0&)
    End If

    PlayPosDataFile = PlayPosDataFile + WaveOutHdr.dwBufferLength
    rc = acmStreamConvert(hAS, acmHdr, ACM_STREAMCONVERTF_BLOCKALIGN)
    rc = waveOutWrite(hWaveOut, WaveOutHdr, Len(WaveOutHdr))
    PosBytes = PlayPosDataFile - InicioData

End Sub
```

En éste momento se cargaron los dos buffers en el DSP de la tarjeta de sonido y se esta reproduciendo el primero. Una vez el primero termine de reproducirse se corre la interrupción que llama a waveOutProc quien carga de disco a memoria nuevamente el primer buffer con los datos que siguen. El proceso continua iterando entre los dos buffers hasta que el archivo es reproducido completamente.

```
Public Sub waveOutProc(ByVal hWaveOut As Long, ByVal Msg As Long, ByVal
    dwInstance As Long, ByVal dwParam1 As Long, ByVal dwParam2 As Long)

      Select Case Msg
            Case MM_WOM_DONE
                If PlayPosDataFile + WaveOutHdr.dwBufferLength < _
                   (TamañoData + InicioData) Then
                    If BufferAct = 0 Then
                        PosBytes = PlayPosDataFile - InicioData
                        If Not MP3 Then
                           rc = mmioRead(hFile, WaveOutHdr_x.lpData, _
                                 WaveOutHdr_x.dwBufferLength)
                        Else
                           rc = ReadFile(hFile, ByVal
                                        WaveOutHdr_x.lpData,_
                                        WaveOutHdr_x.dwBufferLength,
                                        numread, ByVal 0&)
                        End If

                        PlayPosDataFile = PlayPosDataFile + _
                                          WaveOutHdr_x.dwBufferLength
                        rc = acmStreamConvert(hAS, acmHdr_x,
                                           ACM_STREAMCONVERTF_BLOCKALIGN)

                        rc = waveOutWrite(hWaveOut, WaveOutHdr_x, _
                                          Len(WaveOutHdr_x))
                        BufferAct = 1
                    Else
                        PosBytes = PlayPosDataFile - InicioData
                        If Not MP3 Then
                            rc = mmioRead(hFile, WaveOutHdr.lpData, _
                                          WaveOutHdr.dwBufferLength)
                        Else
                            rc = ReadFile(hFile,
                                          ByVal WaveOutHdr.lpData,
                                          WaveOutHdr.dwBufferLength,
                                          numread, ByVal 0&)
                        End If

                        PlayPosDataFile = PlayPosDataFile + _
                                          WaveOutHdr.dwBufferLength
                        rc = acmStreamConvert(hAS, acmHdr, _
                                           ACM_STREAMCONVERTF_BLOCKALIGN)

                        rc = waveOutWrite(hWaveOut, WaveOutHdr, _
                                          Len(WaveOutHdr))
                        BufferAct = 0
                    End If
                End If
        End Select
    End Sub
```

Del los códigos anteriores se eliminaron algunos controles de la reproducción de la música, como los estados de Playing y Stoping, dejando solo la funcionalidad

básica para hacer más entendible la lógica. El código real se encuentra en la carpeta de programas adjuntos a éste informe final. En éste programa se reproducen ondas de sonido que se encuentran en archivos, pero la arquitectura se presta para reproducir cualquier sonido que este almacenado en buffers, por ejemplo, los datos que llegan por la tarjeta de red entregados por WinSock si éstos son de sonido (voz).

11.2. COMPONENTE ACTIVEX DLL

Buscando el objetivo final de la práctica empresarial se diseño y elaboro un componente COM de Microsoft (Components Objets Model – Modelo de Objetos Componentes) del tipo "*In Process*" es decir, ActiveX DLL. La siguiente es la estructura de la clase:

Tabla 25: Estructura de la Clase principal CMGPlayer

Propiedades	Métodos	Eventos	Tipados
Archivo	CmdPlay	Converting	TipoEstado
Estado	CmdPause		Codecs
Etiquetas	CmdStop		WAVEFORMATEX
FormatoOriginal	CmdRecord		WAVEFORMAT
FormatoPCM	Guardar		
Posición	GetBufferOriginal		
Recorded	GetBufferRecorded		
Tamaño	GetWaveFormat		
Etiquetas	DecodeToBuffer		
	BufferByteToBufferInt		

11.2.1. Propiedades

- *Archivo:* Devuelve o establece el nombre del archivo abierto. Al establecer el nombre del archivo, la clase internamente abre el archivo y carga las etiquetas (si tiene) dejándolo listo para la reproducción.
- *Estado:* Devuelve el estado actual de la reproducción. El estado se publica en el tipo enumerado TipoEstado:

Playing = 0
Paused = 1
Stoped = 2
Recording = 3
Converting = 4

- *Posición:* Devuelve o establece la posición en muestras PCM de la reproducción actual.
- *Tamaño:* Devuelve el tamaño en muestras PCM del archivo actual.
- *FormatoOriginal:* Propiedad de tipo WAVEFORMATEX que devuelve el formato del archivo sin decodificación.
- *FormatoPCM:* Propiedad de tipo WAVEFORMATEX que devuelve el formato de la reproducción actual en PCM. Si el archivo en su forma original es un PCM *FormatoOriginal* y *FormatoPCM* son Iguales.
- *Etiquetas:* Referencia a la colección de clases Etiquetas:

  Propiedades de la Clase **Etiqueta**:
  - **Nombre**
  - **Descripción**
  - **PosIni**
  - **PosFin**

  Propiedades de la colección **Etiquetas**:

  - **Count:** Devuelve el numero actual de etiquetas.
  - **Item:** Elemento de conversión entre el nombre (string) y el índice (integer).

  Métodos de la clase **Etiquetas**:

  - **Add:** Agrega una nueva etiqueta. Sirve de constructor de la clase Etiqueta.
  - **Remove:** Elimina la etiqueta cuyo índice coincida con el parámetro pasado.
  - **Clear:** Borra todas la etiquetas de la colección.

- *Recorded:* Indica si el contenido de reproducción actual fue el resultado de la grabación a través de un dispositivo de entrada.

11.2.2. Métodos

Los métodos CmdPlay, CmdPause, CmdStop y CmdRecord se encargan de la reproducción y grabación del sonido. Los métodos *GetBufferOriginal y*

GetBufferRecorded Devuelven un array de bytes con los datos de la onda de sonido. Buffer devuelto por *GetBufferOriginal* Puede no estar en formato PCM, para lo cual se usa el método *DecodeToBuffer* que decodificar los datos. Mientras se está ejecutando cualquier operación de codificación (Cuando se guarda) o decodificación se desencadena el evento ***Converting*** que tiene como argumentos la posición actual de conversión y el tamaño total de los datos que se están convirtiendo. Por facilidad para dibujar la onda cuando las muestras de sonido son de 16 bits, es mejor tener los datos en buffers de enteros. Para esto, MGPlayer dispone del método BufferByteToBufferInt.

El método *Guardar* recibe como parámetros: El formato de origen, el buffer de datos PCM a guardar, el formato de destino y un valor opcional con el nombre del archivo a guardar. Tanto el formato de origen como el formato de destino son estructuras del tipo WAVEFORMATEX. Cuando el usuario no esta familiarizado con el llenado de la estructura WAVEFORMATEX para el caso de codecs especiales, puede usar el método *GetWaveFormat*.

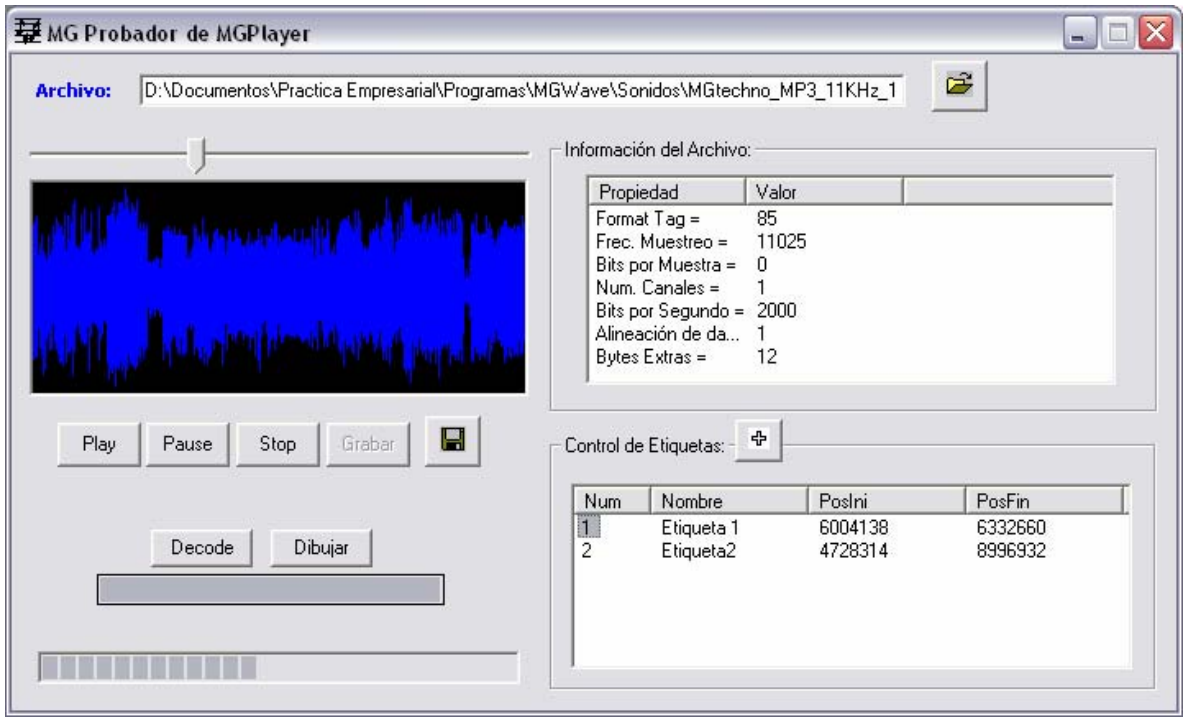
El método *GetWaveFormat* recibe como parámetro un nombre de codec escogido de la lista del tipo enumerado *Codecs* y devuelve en su segundo parámetro la estructura WAVEFORMATEX llena según valores predefinidos por la clase. El tipo enumerado Codecs se define así:

```
Public Enum Codecs
   WAVE_FORMAT_PCM = 1
   WAVE_FORMAT_ADPCM = 17
   WAVE_FORMAT_GSM610 = 49
   WAVE_FORMAT_MSN_AUDIO = 50
End Enum
```

11.2.3. Programa de prueba del componente

Para probar el componente se elaboró un programa con funcionalidades muy parecidas a MGWave, pero en éste caso, demostrando la potencia, flexibilidad y facilidad a la hora de trabajar con componentes COM.

Figura 36: Ventana principal del programa test de MGPlayer.



12. CONCLUSIONES

La brecha entre las ciencias de las Telecomunicaciones y la Informática cada día se cierra más. El punto en común más importante es la información digital, que en la actualidad tiende a crecer en volúmenes agigantados. Grupos de científicos e ingenieros en todo el mundo han trabajado en la implementación de algoritmos computacionales que permitan reducir el tamaño de los archivos digitales. Las técnicas son bastante variadas y requieren de bases matemáticas fuertes. Al reducir el tamaño de un archivo en disco también se reduce la latencia a la hora de transmitir dicho archivo por una red de telecomunicaciones ya sea telefónica o IP (Internet).

En este proyecto se orientó la investigación hacia archivos digitales de sonido en especial archivos de voz. Los objetivos principales para le empresa (como práctica empresarial) eran: Asociar etiquetas de control y comprimir, archivos de sonido procedentes de conversaciones telefónicas. Antes de empezar la práctica y sin el suficiente conocimiento, se sugirió investigar principalmente el formato de archivo de sonido compreso MP3. Pensando que era el mejor a usar por ser un estándar internacional. A medida que se fue investigando se encontró que el MP3 funciona muy bien para archivos de sonido en General (Música) pero no explota al máximo las características especiales de la voz humana que permiten que un archivo de sonido que contiene solo voz se pueda reducir aun más que un MP3.

Antes de llegar a estudiar los formatos de compresión de sonido fue necesario conocer sus fundamentos tanto físicos como digitales, y todo el proceso que se lleva a cabo en la digitalización. Desde el punto de vista de un programador, se documentaron las bases del sonido en el PC bajo los sistemas operativos de Microsoft (DOS y Windows). Se explicó como intervienen los componentes de las tarjetas de sonido en el proceso de grabación y reproducción, y como los sistemas operativos han prestado a los programadores los recursos para interactuar con el hardware de sonido de cualquier fabricante.

El formato de archivo más popular en la hegemonía de los PC ha sido el WAV de Microsoft. Es un archivo que en su forma más simple almacena los pulsos muestreados y codificados de la onda de sonido (PCM – Pulse Code Modulation) pero su estructura de archivo RIFF (Estructura estándar – Formato de Archivo de Intercambio de Recursos) le permite expandirse para soportar muchos otros formatos de almacenamiento (Generalmente Codificados para comprimir) existentes en la actualidad y poder agregar información personalizada sin que se pierda su esencia de WAV. Los formatos de almacenamiento codificado requieren de un Codec especial para que el sistema operativo pueda reproducir los sonidos que representan los datos de un WAV.

Entre los Codecs más populares podemos encontrar el mismo MP3 (MPEG Layer III) del consorcio “Motion Picture Expert Group” que es quizás el más usado en el mundo para comprimir Música. En cuanto a Codecs especiales para voz encontramos el GSM (Global System Movile) de la Unión Europea de Telecomunicaciones; el ADPCM (Adaptive Diferencial PCM) de Microsoft; el CS-ACELP (Conjugate Structure Algebraic Codeexcited Linear Prediction) de la Unión Internacional de Telecomunicaciones (UIT). Este último es quizás el más recomendado para el propósito del proyecto y aplica tanto para Almacenamiento como para transmisión por redes IP. Pruebas de alto redimiendo han demostrado la superioridad del CS-ACELP sobre otros codecs.

El CS-ACELP es precedido por el CELP, ACELP, VSEPL todos basados en la explotación de las características propias del Aparato Fonador Humano que permiten modelarlo matemáticamente (Filtros Lineales). La UIT permite descargar los algoritmos en código C ANSI estándar del CS-ACELP y queda como un pendiente para éste proyecto adaptar dichos códigos el componente final entregado.

La investigación debe tener un soporte experimental, y teniendo en cuneta esto, a medida que se avanzaba en la práctica se elaboraron los debidos programas de prueba más que todo en Visual Basic 6.0 (Con fines académicos). Aunque la programación en Visual Basic tiene fama de ser fácil, cuando se trata de hacer herramientas de alta complejidad éste lenguaje no provee los componentes necesarios. Algunas empresas venden dichos componentes, pero la meta es realmente conocer lo suficiente para lograr crearlos. En el caso del proyecto, la mayoría de las funciones había que invocarlas directamente al Sistema Operativo (API), y Visual Basic se convirtió en un cascarón que soporta las llamadas de un código que podría ser fácilmente adaptado a funcionar en otros lenguajes más complejos como el C++.

El resultado más relevante de la práctica es un programa desarrollado en Visual Basic al que llamé MGWave. Éste programa se fue perfeccionando con el paso del tiempo y reúne casi todos los aspectos de la práctica. Después se implementó el componente software que sí hace parte de los objetivos planteados. El componente no fue más que una adaptación de MGWave a COM (Modelo de Objetos Componentes) con ciertas mejoras. El componente por ser desarrollado en Visual Basic no es de alto rendimiento, es decir, no es tan rápido como lo sería una versión suya en Visual C++ (El padre de COM). Por tanto una recomendación para el componente a agregar al proyecto de Contact Center es desarrollarlo mejor en Visual C++, lo que no es una tarea tan complicada, ya que la lógica y metodología sería la misma del desarrollado en Visual Basic.

Nota adicional: En la fase de investigación previa del proyecto de Contact Center de TYT Ltda. se designaron a dos personas en el tema de Reconocimiento de Voz. En las fases finales de mi práctica se planeó tomar la información recolectada por esas dos personas para, de cierta forma, lograr insertar etiquetas a los WAV de acuerdo a palabras reconocidas en las conversaciones. La investigación de los dos compañeros no arrojo los resultados necesarios para lograr esto. Al tener entre los objetivos de mi práctica algo relacionado con el Reconocimiento de Voz, se dedicó una fase a documentar la investigación de mis compañeros y otras adiciones personales. Dicha documentación se entregó en un CD con el cuarto informe de avance y no se agregó a éste documento para no hacerlo tan voluminoso y no sacarlo del contexto general.

13. BIBLIOGRAFIA

1. Acústica: Ondas, movimiento ondulatorio y sonido. Disponible en Internet: <http://html.rincondelvago.com/acustica_ondas-movimiento-ondulatorio-y-sonido.html>

2. El Sonido y las Ondas. Sitio Web: <http://www.fisicanet.com.ar/fisica/f3ap04/apf3_25b_Ondas_Sonoras.html>

3. Propiedades del Sonido. Disponible en Internet: < http://www.video-computer.com/Propiedades del sonido.htm>

4. Medición y cálculo de los niveles de sonido. Disponible en Internet: <http://www.windpower.org/es/tour/env/db/dbdef.htm>

5. Sonido en TV: Principios Básicos. Disponible en Internet: <http://www.cybercollege.com/span/tvp037.htm>

6. MIYARA, Federico. La Voz Humana. PDF disponible en Internet: <http://www.eie.fceia.unr.edu.ar/~acustica/biblio/fonatori.pdf>

7. Descripción General sobre Audio Digital. Disponible en Internet: <http://www.euskalnet.net/manzano/apuntes/cuerpos/cuerpo-audio-digital.htm>

8. IGLESIAS, Pablo Simón. Postproducción digital del sonido por computadora. Ed. Alfaomega.

9. MARTINEZ, Evelio. Conversión Análogo – Digital.
   E-mail: eveliom@eveliux.com
   URL: <http://www.eveliux.com/fundatel/analogdigital.html>

10. TISCHER / JENNRICH. PC Interno 5. ed. Alfaomega. Pags. 605-729

11. Tarjetas de Sonido. Disponible en Internet: <http://www.duiops.net/hardware/tarjson/tarjson.htm>

12. LLAURADO, Ricardo. Digitalización de la Voz. Nociones Básicas. Disponible en Internet: <http://www.cq-radio.com/articles/007_abr_84.htm>

13. MORENO, Antonio. Diseño e implementación de una interfaz para el desarrollo de software de audio. Disponible en Internet: <http://nexus.hispalinux.es/docs/gsound/design/html/memoria.html>

14. VICTORIA, Virgilio. Multimedia y Website: Formatos utilizados en la tecnología multimedia. Disponible en Internet: <http://www.utp.ac.pa/seccion/topicos/multimedia/formatos.html>

15. Conozca Wave. Disponible en Internet: <http://seritel.teleco.ulpgc.es/trabajos/repetidores/wave.htm>

16. Wave Files. Página Sonic Spot. Todo sobre archivos de sonido. En internet: <http://www.sonicspot.com/guide/wavefiles.html>

17. Wave File Format. Disponible en Internet: <http://www.lightlink.com/tjweber/StripWav/WAVE.html>

18. Wave File Format. Descripción completa y detallada. Disponible en Internet: <http://www.borg.com/~jglatt/tech/wave.htm>

19. VEJAR, Iván. Fundamentos del Sonido Digital. E-mail: rexvej2@hotmail.com URL:<http://www.monografias.com/trabajos7/sodi/sodi.shtml>

20. K. Brandenburg, H. Popp. An introduction to MPEG Layer-3 PDF. *Fraunhofer Institut für Integrierte Schaltungen (IIS).*

21. MIKEL. MP3 en Profundidad. Página principal del sitio: <http://www.hispamp3.com/tallermp3/tutoriales/mp3profundidad/mp3profundidad.shtml>

22. Cómo Hacer un MP3. Internet: <http://www.hispamp3.com/tallermp3/como/comohacerunmp3.shtml>

23. MIKEL. Compresión de Audio. Internet: <http://www.hispamp3.com/tallermp3/tutoriales/mp3profundidad/2.shtml>

24. MIKEL. MPEG Audio. Internet: <http://www.hispamp3.com/tallermp3/tutoriales/mp3profundidad/3.shtml>

25. MIKEL. Breve FAQ sobre MPEG. Disponible en Internet: <http://www.hispamp3.com/tallermp3/tutoriales/mp3profundidad/5.shtml>

26. MIKEL. Especificaciones ISO MPEG. Internet: <http://www.hispamp3.com/tallermp3/tutoriales/mp3profundidad/6.shtml>

27. Página Oficial IIS sobre MP3. URL:
<http://www.iis.fraunhofer.de/amm/techinf/layer3/index.html>

28. ¿Qué es un MP3? Disponible en Internet:
<http://www.hispamp3.com/tallermp3/como/queesunmp3.shtml>

29. Formato MP3. Disponible en Internet:
<http://mpgedit.org/mpgedit/mpeg_format/mpeghdr.htm>

30. BUSTOS, Javier Alejandro. Estudio de sistemas de compresión de voz digital orientado a telefonía celular. Disponible en Internet:
<http://www.inf.udec.cl/revista/ediciones/edicion7/jbustos.pdf>

31. NADEU, Climent. Representación de la Voz en el Reconocimiento del habla. Disponible en Internet:
<http://www.imim.es/quark/21/021063.htm>

32. ROZADA, José M. Nuevos estándares para comunicaciones multimedia. Disponible en Internet:
<http://neutron.ing.ucv.ve/revista-e/No5/JRozada.html>

33. Recomendaciones UIT para sonido y video por IP. URL:
<http://www.itu.int/rec/recommendation.asp?type=products&lang=s&parent=T-REC-G>

34. ITU-T G.729 (1996), Recomendación G.729-Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear-prediction (CS-ACELP).

35. CS-ACELP y Pérdida de Paquetes. Disponible en Internet:
<http://redesopticas.reuna.cl/publicaciones/nestor_best_practices_29_10_02.doc>

36. ECENARRO, Iñaki. Introducción a Direct Sound. Disponible en Internet:
<http://usuarios.lycos.es/macedoniamagazine/directx2.htm>

37. Arquitectura de sonido en Windows. Disponible en Internet:
<http://www.cs.columbia.edu/~coms6181/ slides/6/**windows_audio.ppt**>

38. "Request for Comments". Microsoft. Formatos RIFF Wave registrados. Disponible en Internet: <http://rfc-2361.rfc-search.net/rfc-2361.htm>

39. Como usar el encoder LAME (Incluye ejemplo). Disponible en Internet:
<http://www.vbaccelerator.com/home/VB/Code/vbMedia/Audio/MP3_Encoding_with_LAME/article.asp>

40. Sitio de descargas LAME. URL: <http://mitiok.free.fr/>

41. Documentación del Administrador de Compresión de Sonido (ACM) en el MSDN Library de Microsoft. URL: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_audio_compression_manager.asp >

42. Código Fuente de la implementación de ACM para pasar de MP3 a PCM en C++. Url: <http://david.weekly.org/code/mp3acm.html>

43. Lista y descripción de los codecs incluidos con Windows 2000. URL: <http://www.microsoft.com/windows2000/es/professional/help/compressing_sound_files.htm>

CONTENIDO DEL CD-ROM ADJUNTO

El CD que se incluye con éste informe final reúne todo el trabajo desarrollado durante la práctica. En él se encuentran 3 carpetas principales:

Información recolectada: contiene todas las páginas, documentos, PDFs, código fuente, etc, organizados en carpetas según la fase de la práctica y el tema.

Programas: contiene el código fuente en Visual Basic 6.0 y en algunos casos los instaladores de los programas desarrollados en la práctica: MGWave, MGPlayer DLL y su Test, Ejecución en tiempo real, y otras pruebas que se menciona en éste informe. En ésta carpeta también encontramos algunos programas encontrados en Internet y que fueron útiles en la creación de los primeros.

Documentos: contiene todos los documentos relacionados con la práctica empresarial, desde el tema y el plan, hasta éste informe final. Por tanto estan allí todos los informes de avance junto con sus respectivos marcos teóricos.