Universitat Autònoma de Barcelona

Facultad de Ciencias

Sección de Ingeniería Informática

Visualización 3D de la modelización sinusoidal más residual del sonido

Memoria del proyecto Fin de Carrera
de Ingeniería en Informática
presentada por Bernardo Macías Bécares
y dirigida por Jaume Pujol Capdevila y
Xavier Serra Casals.
Bellaterra, a 8 de Septiembre de 1998

El abajo firmante, Jaume Pujol Capdevila
Profesor del Departamento de Informática
De la Universitat Autónoma de Barcelona

Certifica

Que el trabajo al que corresponde la presenta memoria
ha sido realizada bajo su dirección
por Bernardo Macías Bécares

Y para ello que conste firma la presente
en Bellaterra, a 8 de Septiembre de 1998.

Jaume Pujol Capdevila

El abajo firmante, Xavier Serra Casals
En nombre y representación de
Phonos Fundació Privada

Certifica

Que el trabajo al que corresponde la presenta memoria
ha sido realizada bajo su dirección
por Bernardo Macías Bécares

Y para ello que conste firma la presente
en Barcelona, a 8 de Septiembre de 1998.

Xavier Serra Casals

A mis padres y hermano

Agradecimientos

Agradezco a Xavier Serra sus explicaciones.

A Jaume Pujol por la confianza.

A Jordi por ayudarme siempre que ha podido.

A Esther por esos minutos.

A Patxi por su ayuda

A Máximo por su amistad

A Gloria por su apoyo

Se acabó la luz. Se acabó el color

El Paciente Inglés

Michael Ondaatje

1 Introducción *10*

1.1 Motivaciones **10**

1.2 Objetivos **10**

1.3 Contenido del proyecto **11**

2 Antecedentes *12*

2.1 La técnica SMS **12**

2.1.1 Introducción 12

2.1.2 La transformada corta de Fourier 12

2.1.3 El proceso de análisis 15

2.1.4 Formato del fichero para datos SMS 18

2.1.4.1 Introducción 18

2.1.4.2 Características generales 18

2.1.4.3 Estructura del fichero 19

2.2 Representación del sonido **22**

2.2.1 Introducción 22

2.2.2 Representación en el dominio temporal 23

2.2.3 Representación en el dominio frecuencial 24

2.2.3.1 Estático 27

2.2.3.2 Variando en el tiempo 30

2.3 Entornos de programación de 3 dimensiones **38**

2.3.1 Introducción 38

2.3.2 Aplicaciones más usuales 39

2.3.3 OpenGL 45

2.3.3.1 Operación básica en OpenGL 47

2.3.3.2 El pipeline de OpenGL 50

2.3.3.3 La integración en un sistema de ventanas 62

2.3.3.4 OpenGL y Windows de Microsoft 65

3 Desarrollo de la aplicación *66*

3.1 Opciones adoptadas **66**

3.2 Representación gráfica **66**

3.2.1 Introducción 67

3.2.2 El lenguaje Visual C++ 68

3.2.3 Representación gráfica de diferentes tipos de datos 69

3.2.3.1 Parciales (Sinusoides) 69

3.2.3.2 Residuo 71
3.2.3.3 Parciales y residuo 72
3.2.4 La ventana principal 72
3.2.5 Espacio de trabajo 74
3.2.6 Conclusiones 75

3.3 Modularización de la representación gráfica 76
3.3.1 Introducción 76
3.3.2 Clases 76
3.3.2.1 La clase CRepresentation3D 78
3.3.2.2 La clase CTrans3D 79
3.3.2.3 La clase CMath3D 80
3.3.3 Descripción del funcionamiento interno 80
3.3.4 Uso de la aplicación 82
3.3.5 Pruebas 83

4 Conclusiones 85

4.1 Posibles ampliaciones 86

5 Bibliografía 87

Figura 1 Ejemplo de transformada corta de Fourier (STFT) 14
Figura 2 Diagrama de bloques del proceso de análisis 17
Figura 3 Relaciones entre chunks en el formato del fichero 20
Figura 4 Representación en el domino temporal de una señal 24
Figura 5 Representaciones en el dominio temporal y frecuencial 26
Figura 6 Representaciones estáticas del espectro 28
Figura 7 Representación del espectro variante en el tiempo 31
Figura 8 Imágenes de representaciones de tiempo real en cascada 33
Figura 9 Dibujo de un sonograma de un golpe a un tam-tam 34
Figura 10 Dibujo en perspectiva de una síntesis aditiva 37
Figura 11 Dibujos separados para envolventes de la frecuencia y la amplitud 37
Figura 12 Diagrama de bloques de OpenGL 48
Figura 13 Asociación de los valores actuales con un vértice 53
Figura 14 Cliente GLX, servidor X y renderizador OpenGL 65
Figura 15 Pantalla de SMS Tools para el análisis de un sonido de voz 68
Figura 16 Parciales de un sonido de voz. 70
Figura 17 Representación del residuo obtenido en el análisis de una melodía 71
Figura 18 Representación conjunta de parciales y residuo 72
Figura 19 Ventana principal de la aplicación 74
Figura 20 Distribución dentro de la zona de trabajo 75
Figura 21 Zona de control del área de trabajo 75
Figura 22 Operaciones para la representación 77
Figura 23 Clases de la aplicación 77

1 Introducción

1.1 Motivaciones

Este proyecto desde el primer momento despertó mi curiosidad. Lo que más me interesó fue su relación con la música, el tratamiento de señales y la representación gráfica. Me permitía conocer más sobre un tema de máxima actualidad: la representación gráfica en tres dimensiones. Por otro lado. Tenía la oportunidad de aprender un nuevo lenguaje: Visual C++, que me será de gran utilidad en el futuro.

1.2 Objetivos

Se pretende realizar una aplicación que permita visualizar en un entorno de tres dimensiones el análisis realizado por medio de la técnica SMS y poder desplazarse dentro de este entorno.

Esta aplicación llegaría a formar parte de un software de análisis y síntesis de sonidos denominado *Spectral Modeling Synthesis* (SMS) desarrollado en el Institut Universitari de l'Audiovisual dependiente de la Universitat Pompeu Fabra por Xavier Serra.

Existe actualmente una aplicación llamada SMS que permite el análisis y la síntesis de sonidos por medio de la técnica del mismo nombre. Tiene una interfase diseñada e implementada en Visual C++ basada en ventanas para controlar los diferentes procesos de análisis, transformación y síntesis, y mostrar las representaciones de los diferentes tipos de datos implicados en ellos.

A estas representaciones le faltaba una visión tridimensional de los datos de análisis para una interacción con el usuario más cómoda y fácil. No pretende ser una aplicación comercial sino un entorno de trabajo y experimentación.

1.3 Contenido del proyecto

La memoria está dividida en capítulos que sintetizan las diferentes etapas llevadas a cabo durante el desarrollo del proyecto:

2. ANTECEDENTES

Detalla todo el estudio realizado, la búsqueda de información sobre la situación actual y los conceptos adquiridos.

3. DESARROLLO DE LA APLICACIÓN

Se centra en el diseño de la aplicación llevado a cabo, describiendo las opciones adoptadas, la representación gráfica, la modularización de esta representación, como ha sido creada la aplicación y las pruebas realizadas.

Como parte final de todo el proceso se exponen una serie de conclusiones donde se incluye las posibles ampliaciones de este proyecto.

2 Antecedentes

2.1 La técnica SMS

2.1.1 Introducción

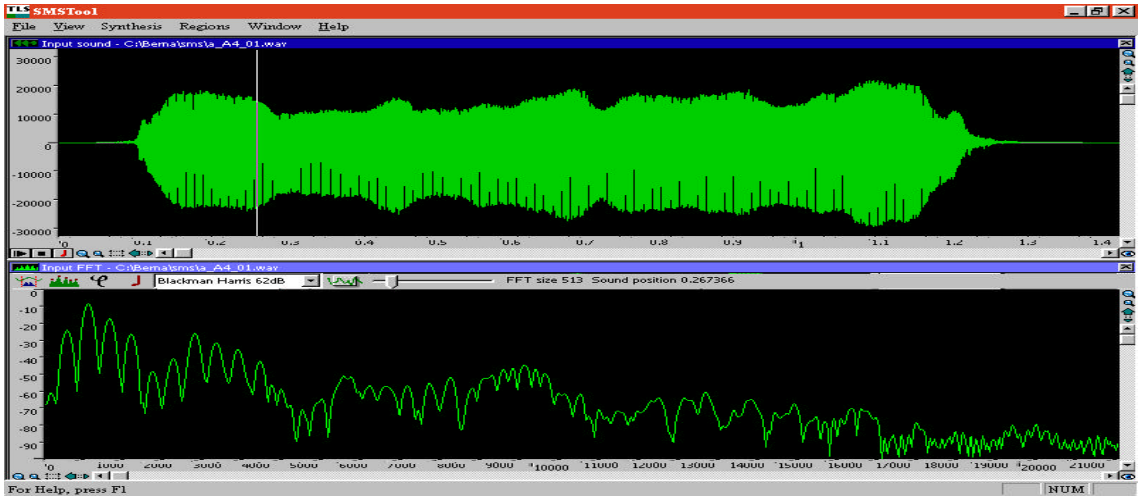
El objetivo de la técnica SMS (*Spectral Modeling Synthesis*) es el de modelar los sonidos como sinusoides estables (parciales) más ruido (la componente residual). En el procedimiento de análisis se detectan los parciales estudiando como varían en el tiempo los picos del espectro frecuencial y se representan como sinusoides. Estos parciales se restan del sonido original y se obtiene la componente residual, que se representa como ruido blanco pasado por un filtro que varia a lo largo del tiempo. Por tanto, el procedimiento de síntesis es una combinación de síntesis aditiva para los parciales y síntesis sustractiva para la parte residual [9].

Esta técnica se puede usar tanto para **generar sonidos** como para **transformar sonidos** existentes. En el primer caso, en general se pretende modelar un instrumento o una familia tímbrica. Para conseguirlo se han de analizar diferentes notas y transiciones del instrumento y construir una base de datos que lo caractericen, a partir de la cual se pueden sintetizar nuevos sonidos. En cambio, en el segundo caso, lo que se pretende es manipular un sonido dado y entraría dentro del ámbito del procesamiento del sonido.

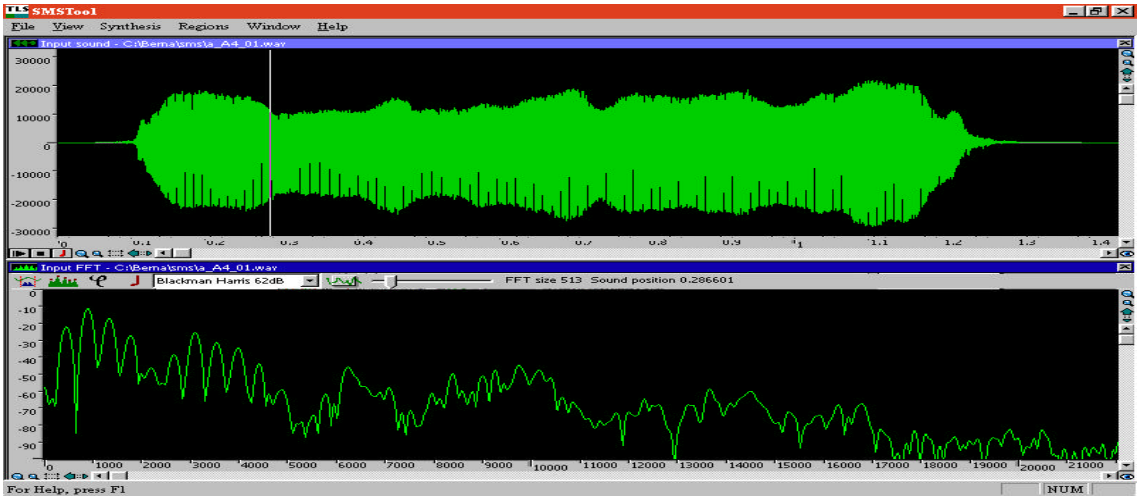
2.1.2 La transformada corta de Fourier

Como veremos a continuación, en el proceso de análisis se cogen fragmentos del sonido y se pasan por una ventana antes de calcular la información espectral de cada fragmento. Así se construye una serie de *fotogramas* del sonido a partir de los cuales en el proceso de síntesis se podrá reconstruir el sonido original y transformarlo. La técnica a través de la cual se realiza todo este proceso es la llamada transformada corta de Fourier (STFT, *Short Time Fourier Transform*) [12]. No es de interés para este trabajo entrar en todos los detalles sobre la transformada, así que se darán por supuestas una serie de propiedades.

(a)



(b)



(c)

Figura 1 Ejemplo de transformada corta de Fourier (STFT). Se ven las magnitudes del espectro de tres fragmentos enventanados (a, b y c) de un sonido de voz. La ventana utilizada es una Blackmann-Harris de 62 dB y 513 muestras.

En la práctica, los sonidos musicales son no periódicos y su forma de onda cambia con el tiempo. Para poder evaluar sus características, se calculan sucesivas transformadas cortas de Fourier haciendo avanzar una ventana corta a lo largo del sonido. Esta operación se puede expresar de la siguiente manera:

$$X_l(k) = \sum_{n=0}^{N-1} w(n)x(n+lH)e^{-jw_k n}$$

Donde $X_l(k)$ es la transformada discreta de Fourier, es decir, el espectro, de la secuencia $\omega(n)x(n+lH)$, con $0 \leq n \leq N-1$. $\omega(n)$ es la ventana que determina el fragmento del sonido $x(n)$ que se analiza en el *frame l*. *H* representa el número de muestras que avanza la ventana $\omega(n)$ en cada *frame*. Esta descomposición del sonido recibe el nombre en inglés de *overlap-add descomposition* [12]. La ventana, además de determinar la parte del sonido que se ha de procesar, también convierte el espectro de frecuencia en una función suavizada, y permite encontrar cualquier pico aunque no este centrado en una de las muestras discretas del espectro.

Se puede recuperar la onda original a partir de los espectros sucesivos que se han obtenido haciendo uso de la inversa de la STFT. Esta reconstrucción se expresa con la fórmula siguiente:

$$s(n) = \sum_{l=0}^{L-1} \mathrm{Desp}_{lH,n}\left[\frac{1}{k}\sum_{k=0}^{K-1} X_l(k) e^{jw_k m}\right]$$

Donde $X_l(k)$ representa el espectro en el *frame l*, y m = 0,1,...,K-1 es el índice de tiempo dentro del *frame*. La función Desp realiza un desplazamiento del fragmento de sonido obtenido con la inversa corta de Fourier en el *frame l*.

$$\mathrm{Desp}_{S,n}[x(m)] = \begin{cases} 0 & si \quad n < S \\ x(n-S) & si \quad S \le n \le S+K \\ 0 & si \quad n \ge S+K \end{cases}$$

El resultado es el sonido original multiplicado por la suma de las ventanas usadas en el proceso de análisis. Si suponemos que el sonido ocupa *L frames*

$$s(n) = x(n)\sum_{l=0}^{L-1} w(n - lH)$$

Por tanto, si las ventanas sumadas con un desplazamiento H dan una secuencia aproximadamente constante con el valor 1, el resultado obtenido será el sonido original.

2.1.3 El proceso de análisis

El proceso de análisis está basado en *frames*, es decir, que se computa un *frame* detrás de otro. En la figura 2 se puede ver un diagrama de bloques

de este proceso. En primer lugar se multiplica por una ventana el sonido que se quiere analizar y se obtiene su espectro mediante el algoritmo de la transformada rápida de Fourier (FFT). Del espectro se detectan todos los picos[1], que servirán para detectar el *pitch*[2] si se trata de un sonido armónico. De estos picos se cogen los que se consideren parciales y se dejan los que se consideren ruido, a través del algoritmo de *continuación de picos*. Este algoritmo, además, incorpora los picos escogidos en las trayectorias correspondientes[3]. En el caso de un sonido armónico, la ventana de análisis se va adaptando al *pitch* detectado, de manera que si el *pitch* aumenta la ventana será más grande y, al contrario, si el *pitch* disminuye, la ventana será más pequeña. Esto permite obtener la resolución en frecuencia necesaria en cada *frame* para poder detectar los picos, aunque el sonido analizado sea una melodía y el *pitch* cambie mucho. Cuando se han incorporado los picos en las trayectorias, se hace una síntesis aditiva del *frame*, teniendo en cuenta la fase[4], y se resta del sonido original en el dominio temporal. Idealmente, si los parciales corresponden a la parte determinista del sonido, después de hacer la resta, se debería obtener la parte ruidosa[5] de este. Del residuo se calcula, a través de la FFT, la envolvente de magnitud espectral que se desglosa en una constante (la energía) y una serie de coeficientes.

---

[1] La detección de picos se hace con una interpolación parabólica cogiendo las magnitudes en dB de un máximo y de los dos *bins* laterales.

[2] Se utiliza un algoritmo basado en las diferencias ponderadas entre los picos medidos y la serie armónica ideal.

[3] El proceso es simple: para cada trayectoria se escoge el pico de frecuencia y magnitud más próximas. Pero se ha de tener en cuenta que dos trayectorias pueden escoger el mismo pico (*peak conflict*). Entonces se ha de escoger a que trayectoria se ajusta mejor el pico, y se puede buscar otro pico para la otra trayectoria. En el caso de un sonido armónico, las trayectorias se ajustan a unas guías que en cada *frame* se recalculan según el *pitch* que se ha encontrado y la frecuencia del pico que se ha cogido en el f*rame* anterior.

[4] Es muy importante conservar la fase porque la parte determinista se ha de restar del sonido original. Si no se conservase la fase, por ejemplo podría pasar que la sinusoide que se restase y la sinusoide presente en el sonido original tenga una fase opuesta, que resultaría en un desfase de la sinusoide original, todo lo contrario de lo que se pretende.

[5] La *parte ruidosa* seria, por ejemplo, el sonido producido por el martillo del piano cuando se ataca una nota o el rozamiento del arco de un violín con la cuerda. Si el sonido no se haya grabado en una situación óptima, la *parte ruidosa* también abarcaría la reverberación de la sala o el ruido de fondo.

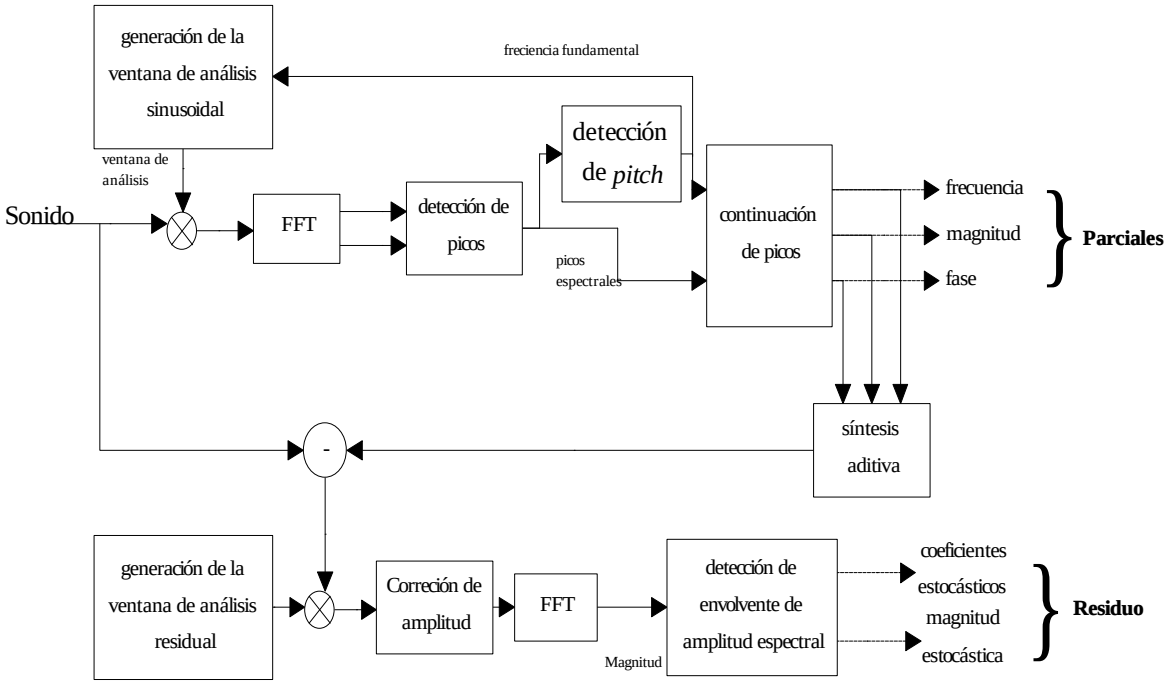


Figura 2 Diagrama de bloques del proceso de análisis

Es importante tener en cuenta el compromiso entre resolución en frecuencia y en tiempo. Para calcular los parciales es importante tener una buena resolución en frecuencia, ya que se ha de detectar los picos con mucha exactitud. Por esto, hace falta una ventana grande. En cambio, para obtener el residuo es necesaria una buena resolución temporal (ya que el espectro cambia rápidamente) y, por tanto, una ventana pequeña. Esto se puede solucionar de la siguiente manera: para detectar los parciales se usa una ventana bastante grande para detectar los picos. Entonces por síntesis aditiva se obtiene la parte determinista que se resta del sonido original. En teoría, cuando se restan los parciales, el residuo mantendrá todas las variaciones temporales rápidas del sonido original. Aquí es donde hará falta una gran resolución temporal y, por tanto, una ventana pequeña para calcular el espectro. Además, el *frame rate* deberá de ser elevado. Pero no hace falta que coincidan el *frame rate* para encontrar los parciales y el *frame rate* para calcular el residuo. De esta manera se pueden combinar una ventana grande y una ventana pequeña, para obtener una gran resolución frecuencial y temporal en cada situación [11].

2.1.4 Formato del fichero para datos SMS

2.1.4.1 Introducción

En este capítulo se describe el formato de fichero usado en el software SMS para representar la información espectral que resulta del programa de análisis y que es utilizado como entrada en la síntesis [8].

Este formato es compatible con la propuesta de *Spectral Description Interface Format* (SDFI), manteniendo su sintaxis y extendiendo sus especificaciones para incluir un conjunto de *chunk* es específico de SMS. Aunque solo se usa como formato de fichero se intenta que funcione también en aplicaciones corrientes.

Este formato va más allá de las capacidades del software SMS actual, soportando posibilidades que están aún bajo desarrollo. Puede incluir no solo la información espectral estándar de un único sonido sino también deferentes niveles de abstracción espectral que es útil para varias aplicaciones que van desde la codificación de alta calidad de audio hasta las aplicaciones de síntesis que usan bancos de datos espectrales. Integra descripciones complementarias del espectro, está basado en *frame*, cada *frame* puede tener una representación diferente y acepta la superposición de *frames*, esto es, "canciones" paralelas.

2.1.4.2 Características generales

- Integración de representaciones espectrales complementarias.
- Soporta representaciones espectrales optimizadas para flexibilidad y orientada hacia las transformaciones del sonido.

- Integración de varios niveles de abstracción de datos espectrales.
- Puede representar un único sonido o una familia tímbrica entera (una colección organizada de sonidos).
- Representación basada en *frame* con *frame rate* variable y soporte para "canciones" paralelas independientes.
- Puede ser usado para codificación de audio de alta calidad.
- Compatible con el formato SDFI.

2.1.4.3 Estructura del fichero

La estructura del fichero sigue el estándar EA IFF 1985 basado en la modularidad. Del estándar viene la idea de *chunks* de datos, como una forma modular y portátil de organizar los datos dentro de un fichero. Siguiendo este estándar, los *chunks* son los bloques básicos de construcción, cada uno incluye una parte de cabecera fija con un *Identificador* y un *Tamaño*, seguido de los datos específicos para cada tipo de *chunk*. La figura 3 muestra los diferentes *chunks* para las necesidades de SMS y sus relaciones.

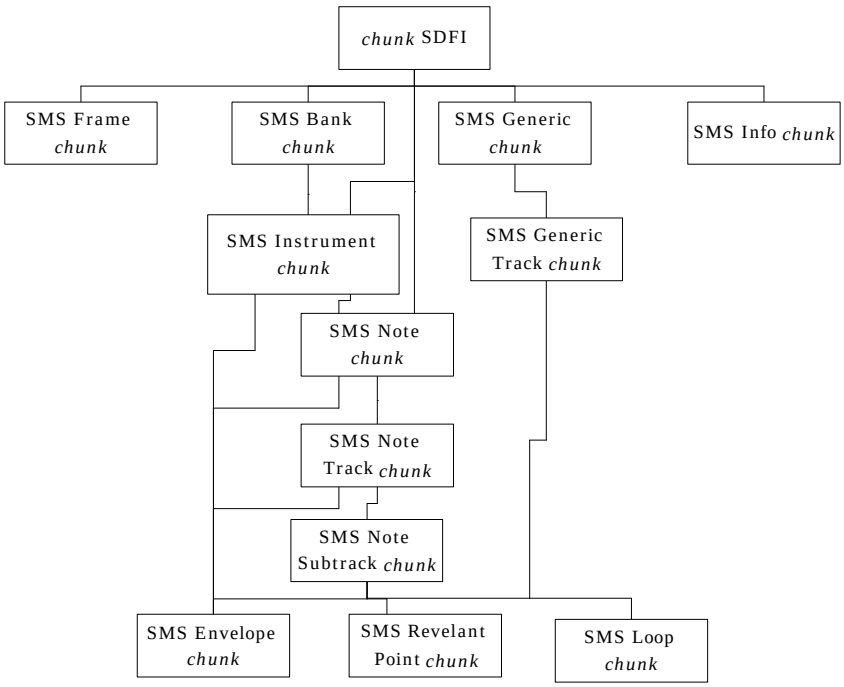


Figura 3 Relaciones entre *chunks* en el formato del fichero

Un fichero simple, que representa un solo sonido sin abstracciones de alto nivel, incluirá un *SDFI chunk*, un *Generic chunk*, un *Generic Track chunk* (como un *sub-chunk*) y un número de *Frame chunks*. Para representar un fichero de los datos de una nota la estructura mínima será un *SDFI chunk*, un *Note chunk*, un *Note Track chunk* (como un *sub-chunk*), un *Note Subtrack chunk* (como un *sub-chunk*) y varios *Frame chunks*. Las notas pueden ser agrupadas poniendo todos los *Note chunk* bajo un *Instrument chunk*. A su vez, los *Instrument chunk* puede ser agrupado en un *Bank chunk* y puede haber resultados más complejos al combinar los *chunks* de diferentes formas.

Los *Frame chunks* incluyen los datos espectrales de bajo nivel y cada uno apunta a un *track* o *subtrack*. Cada *frame* puede tener su propio tamaño y representación y soporta *frame rate* variable. El *Info chunk* incluye información textual que describe el fichero. El *Loop* y *Revelant Point chunks* están asociados a un *track* particular para complementar la información espectral. El *Envelope chunk* puede ser una parte de varios tipos de *chunks*

para representar atributos de alto nivel de los datos espectrales asociado a ellos.

- SDFI chunk: Este es el primer *chunk* en el fichero, donde está especificado el formato SDFI, el tamaño del fichero entero y los *chunks* de alto nivel de los *chunks* incluidos.
- SMS Information chunk: *Chunk* con información general del fichero y de los *chunks* que incluye.
- SMS Generic chunk: *Chunk* para información espectral genérica sin información global y con uno o varios *tracks*. Es útil para las aplicaciones de codificación y decodificación estándar de cualquier tipo de sonidos.
- SMS Generic Track chunk: *Chunk* para información espectral genérica con alguna información global y apunta a *Frame chunks*. Este tipo de *chunk* siempre irá dentro de un *Generic chunk*.
- SMS Bank chunk: [en desarrollo].
- SMS Instrument chunk: [en desarrollo].
- SMS Note chunk: *Chunk* para la información de una nota donde se da información global y se incluye *Note Tracks*. Es útil para construir colecciones de notas de una familia tímbrica dada. Las características que unifican todas las canciones incluidas en un *Note chunk* es un *pitch* común, para sonidos armónicos, o un tipo de nota.
- SMS Note Track chunk: Es un *track chunk* para información de nota donde se da alguna información global y contiene varios *subtracks*. Este tipo de *chunk* siempre se encontrará dentro de un *Note chunk*. Todos los *Note Track Chunk* de un *Note* tienen el mismo *pitch* o tipo de nota pero pueden tener diferentes amplitudes.
- SMS Note Subtrack chunk: Es un *track chunk* para información de una nota donde se da alguna información global y contiene varios tipos de *sub-chunks* y apunta a *Frame chunks*. Este tipo de *chunk* siempre estará dentro de un *Note chunk*. Todo los *Note Track chunks* de un *Note* dado tienen el mismo *pitch* o tipo de nota pero pueden tener diferentes

amplitudes. Dentro de un *Note Track* puede haber más de un *sub-track* para expresar diferentes componentes de un mismo *track*.

- SMS Envelope chunk: Un *Envelope chunk* se usa para almacenar los atributos de alto nivel de un sonido, y puede ser parte de un *Generic Track, Instrument, Note, Note Track* o *Note Subtrack chunks*. Incluye información que ha sido extraída de los datos espectrales y que puede ser añadida durante la síntesis para recuperar el sonido original. Esta envolvente añade mucho control y flexibilidad durante la síntesis siendo capaz de manipularlos antes de ser añadidos a la información espectral en los *frames*.
- SMS Loop chunk: Los lazos son usados para la compresión de datos. Cuando varios frames son similares se puede asignar un punto de lazo especificando que un *frame* dado puede ser repetido varias veces.
- SMS Revelant Point chunk: *Revelant points chunks* se usan para identificar áreas o puntos en un *track* particular que puede ser relevante para la manipulación de ese *track*.
- SMS Frame chunk: *Chunk* que incluye un *frame* de datos espectrales de un *track* dado. Dependiendo de los datos globales especificados (Amplitud y *pitch*) y las envolventes asociados al *track*, los datos del *frame* serán normalizados usando los datos extraídos. Con esta estructura un *Frame chunk* puede almacenar varias representaciones de espectro (STFT, Sinusoidal o Sinusoidal más Residual).

2.2 Representación del sonido

2.2.1 Introducción

Dos visiones separadas han dominado la representación de señales musicales: representaciones en el dominio temporal y en el dominio frecuencial. La visualización de un osciloscopio simple presenta una

representación en el dominio temporal (mostrando la amplitud de la señal contra el tiempo). Los histogramas de un analizador espectral muestran el domino frecuencial, que visualiza la magnitud de cada componente frecuencial (o banda) del espectro.

Avances recientes en la teoría han convergido con fuerza hacia técnicas de programación y procesado de señales que hacen posible una nueva generación de representaciones musicales. Una característica de muchos de estos métodos es la combinación de las vistas en domino temporal y frecuencial, como *grains* o *wavelets* [1]. El incremento de la potencia computacional permite otro aproximamiento, que representa la señal en términos de un modelo matemático (físico) del mecanismo de producción de sonido. Los avances en software tales como la programación visual, lenguajes orientados a objetos y técnicas de inteligencia artificial como redes neuronales, nos dan nuevos medios para trabajar con sonidos musicales. En realidad, para los compositores, quizás el aspecto más importante de estos desarrollos es que sugieren nuevos e interesantes formas de organizar la estructura musical.

2.2.2 Representación en el dominio temporal

Un método simple para describir un sonido es dibujarlo en la forma de un gráfico de la presión del aire contra el tiempo (figura 4). Esto se denomina representación en el dominio temporal. Cuando la línea curva está cerca de la parte inferior del gráfico, entonces la presión del aire es baja, y cuando la curva está cerca de la parte superior del gráfico, la presión del aire se ha incrementado. La amplitud de la onda es la cantidad de cambio de la presión del aire; se puede medir la amplitud como la distancia vertical desde el punto de presión cero al mayor (o menor) al punto de un segmento de onda dado.

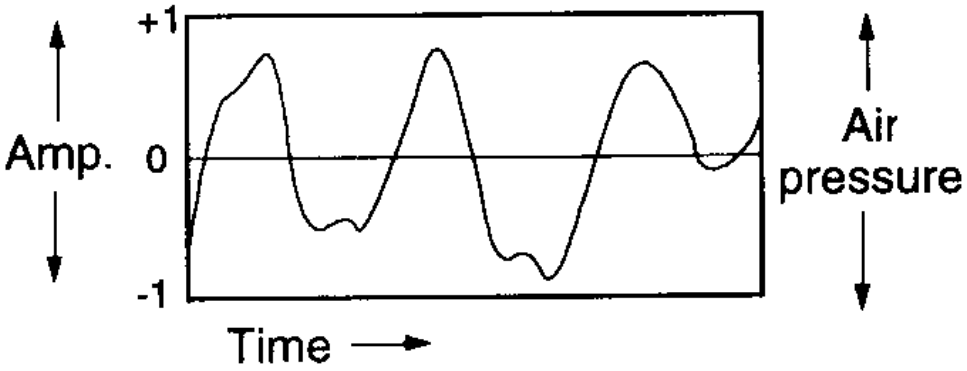


Figura 4 Representación en el domino temporal de una señal. La dimensión vertical muestra la presión del aire. Cuando la línea curva está cerca de la parte superior del gráfico, la presión del aire es grande. Por debajo de la línea horizontal sólida, la presión el aire se reduce. Las variaciones de la presión atmosférica oídas como un sonido pueden ocurrir rápidamente; para sonidos musicales, el gráfico entero quizás no tenga más que un milésima de segundo (1 ms)

Un instrumento acústico crea sonido emitiendo vibraciones que cambian la presión del aire alrededor del instrumento. Un altavoz crea sonido moviéndose atrás y delante de acuerdo con los cambios de voltaje en la señal eléctrica Cuando el altavoz se mueve "dentro" desde su posición de reposo, entonces la presión del aire disminuye. Cuando el altavoz se mueve "fuera", la presión del aire cerca del altavoz se incrementa. Para crear un sonido audible estas vibraciones dentro/fuera deben de producirse en un rango frecuencial entre 20 y 20.000 Hz.

2.2.3 Representación en el dominio frecuencial

Los componentes individuales frecuenciales del espectro se pueden denominar armónicos o parciales. Las frecuencias armónicas son simples múltiplos enteros de la frecuencia fundamental. Asumiendo una fundamental o primer armónico en 440 Hz, su segundo armónico está en 880

Hz, el tercero en 1.760 Hz y así. De forma más general, cualquier componente frecuencial puede ser llamado parcial, sea o no múltiplo entero de la fundamental. En realidad, muchos sonidos no tienen una frecuencia fundamental particular [2].

El contenido frecuencial de una onda puede ser mostrado de varias formas. Una forma estándar es dibujar cada parcial como una línea a lo largo del eje X. La altura de cada línea indica el peso (o amplitud) de cada componente frecuencial. La señal más pura es una onda sinusoidal, llamada así porque puede ser calculada usando la fórmula trigonométrica del seno de un ángulo. Una pura onda sinusoidal representa solo un componente frecuencial, o una línea en el espectro. La figura 5 describe el dominio temporal y frecuencial de varias ondas. Adviértase que los gráficos espectrales están marcados en su eje horizontal con "armónico" ya que se asume que en el algoritmo de análisis la entrada es un periodo de la fundamental de una onda periódica. En el caso de una señal de ruido como la de la figura 5, esto no es válido y redenominamos los parciales como “componentes frecuenciales”.

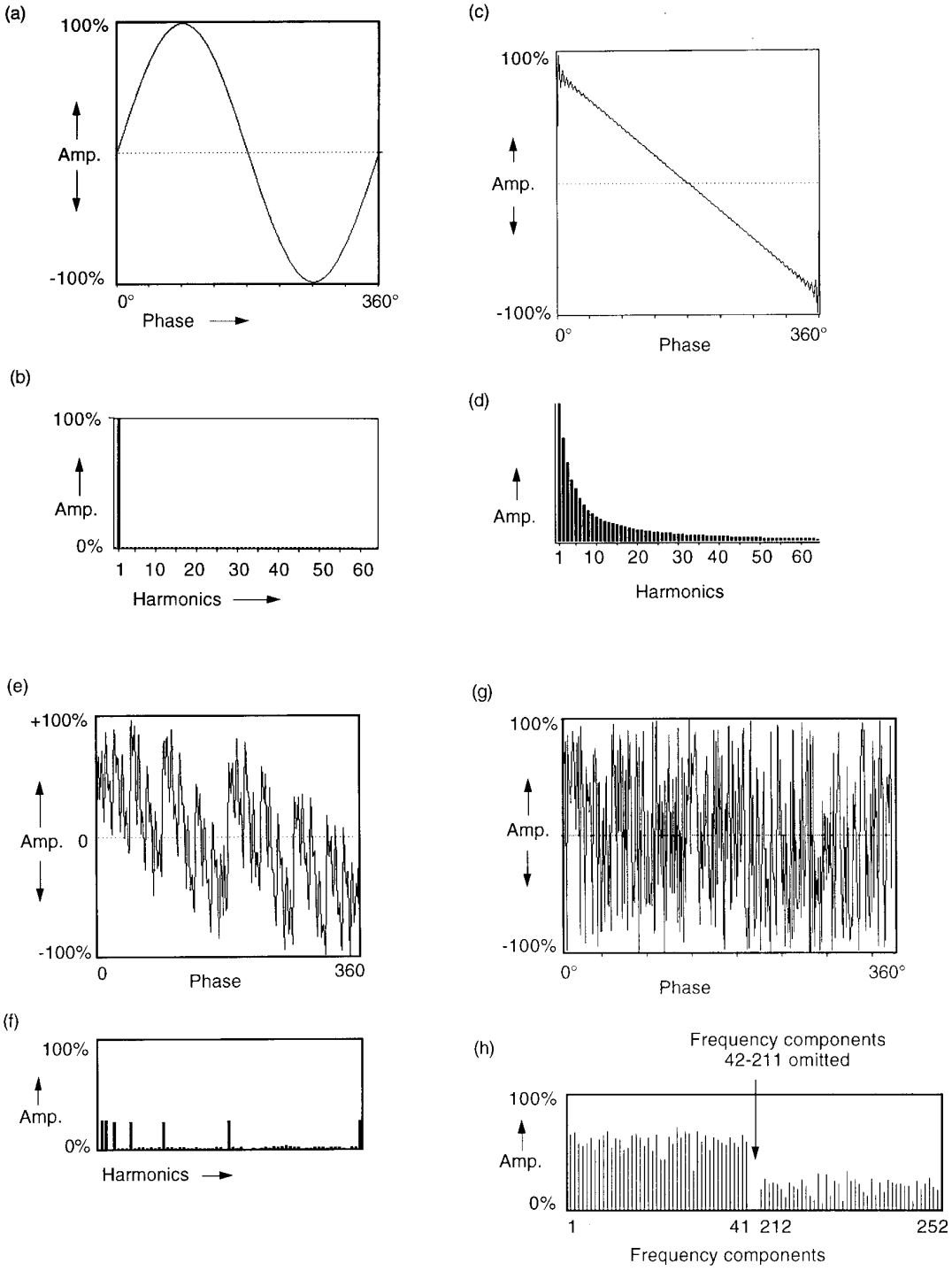


Figura 5 Representaciones en el dominio temporal y frecuencial de cuatro señales. (a) Vista del dominio temporal de un ciclo de una onda seno. (b) Espectro de un componente frecuencial de una onda seno. (c) Vista en el dominio temporal de un ciclo de una onda de diente de sierra. (d) Representación del espectro del decremento exponencial del contenido frecuencial de una onda de diente de sierra. (e) Vista del dominio temporal de un ciclo de una onda compleja. A lo mejor la onda parece compleja, pero cuando se repite una y otra vez su sonido es simple, como el de una estrecha lengüeta del sonido de un órgano. (f) El espectro de la onda (e) muestra que está dominado por unas pocas frecuencias. (g) Una onda de ruido

aleatorio. (h) Si la onda está constantemente cambiando (cada ciclo es diferente del último ciclo) entonces oímos un ruido. El contenido frecuencial de un ruido es muy complejo. En este caso el análisis extrajo 252 frecuencias. Esta fotografía no revela como sus amplitudes están constantemente cambiando en el tiempo.

Existen muchas estrategias para medir y representar el espectro. Podemos dividirlo en dos categorías básicas: estático (como la fotografía del espectro) y variante en el tiempo (como una película del espectro en el tiempo).

2.2.3.1 Estático

Las representaciones estáticas capturan la imagen de un sonido. Esta fotografía del sonido proyecta una imagen bidimensional de la amplitud contra la frecuencia. El análisis mide la energía media en cada región frecuencial sobre un periodo de tiempo del segmento analizado. Este periodo de tiempo o ventana puede variar desde un instante breve a varios segundos o más.

Un tipo de representación estática es un espectro discreto o lineal, donde una línea vertical representa cada componente frecuencial. Para la mayoría de los tonos armónicos, el análisis más claro es *pitch-síncrono* [2]. Este tipo de análisis mide la amplitud de los armónicos de un tono que su *pitch* puede ser determinado a mano. La figura 6 muestra el espectro de línea para una parte estable de un tono de trompeta que se ha medido usando la técnica de *pitch-síncrono.* Adviértase que el instante en el que el espectro ha sido medido, el tercer armónico es mayor en amplitud que el fundamental.

La figura 6 muestra otro espectro de trompeta que representa una escala de amplitud logarítmica (dB). La escala comprime el dibujo dentro de una banda vertical estrecha. Trazando una línea en los picos se puede ver la forma del formante.

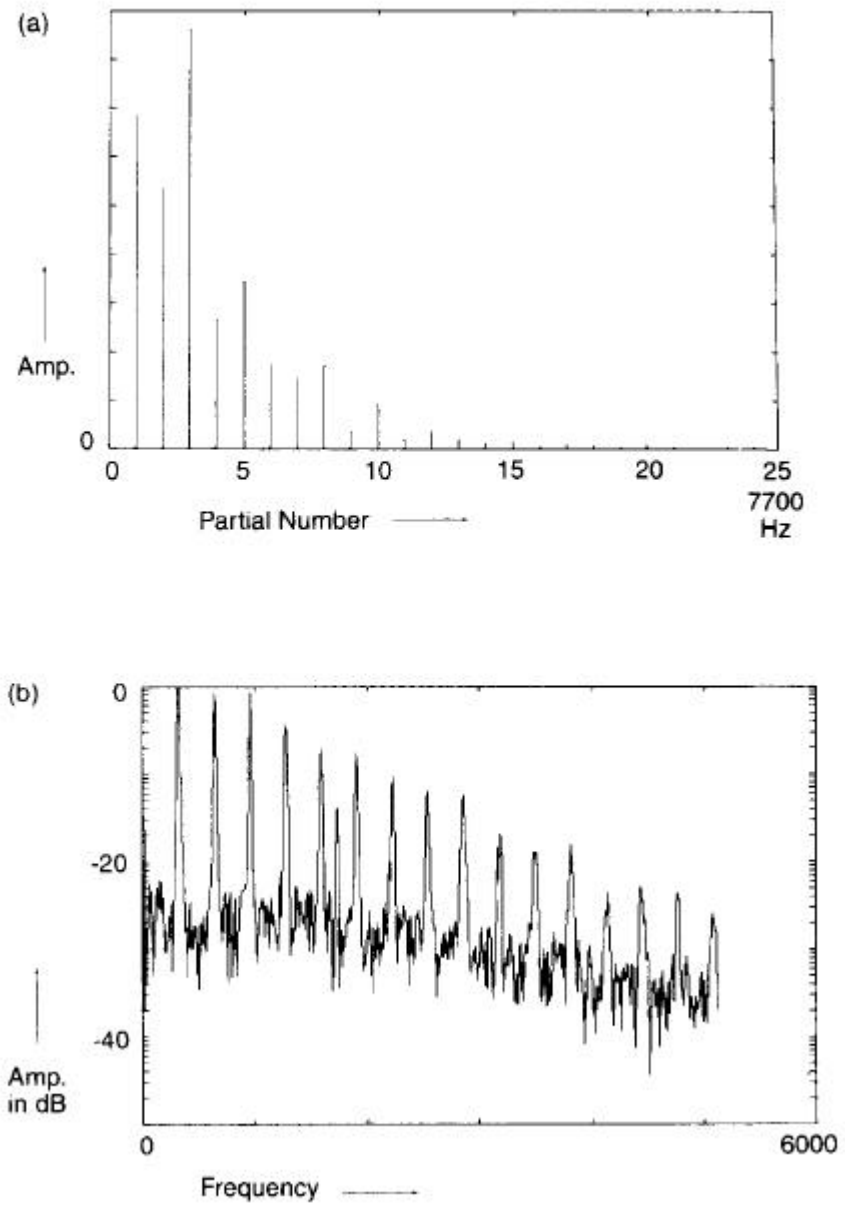


Figura 6 Representaciones estáticas del espectro. (a) Representación en línea del espectro de la amplitud contra la frecuencia de una porción de tono sostenido de trompeta. Cada línea representa el peso de un armónico de la frecuencia fundamental de 309 Hz. La escala de amplitud es lineal. (b) Espectro del tono de trompeta de (a) dibujado en una escala logarítmica (dB), que comprime la representación dentro de una banda vertical estrecha. (c) Representación del espectro de forma continua, muestra el contorno de los picos de los formantes para el sonido de la vocal "ah". La escala de la amplitud es lineal.

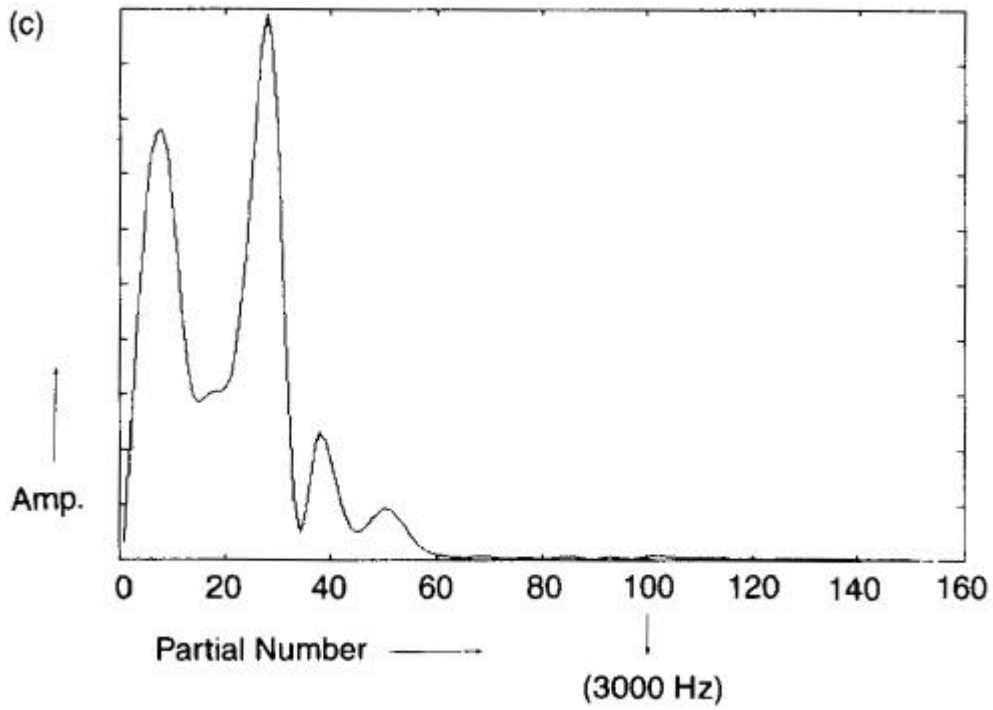


Figura 6 Cont.

La figura 6 representa el espectro del sonido vocal de "ah" de forma continua, donde los puntos discretos medidos por el analizador han sido rellenados con interpolación gráfica. Los componentes sinusoidales independientes están escondidos, pero la forma del espectro está clara.

Cada tipo de representación estática del espectro tiene sus ventajas [2], dependiendo de la señal analizada y del objetivo del análisis.

Espectro de potencia

Del espectro de amplitud se puede derivar el espectro de potencia. Los físicos definen potencia como el cuadrado de la amplitud de una señal. Así, la potencia del espectro es el cuadrado de la amplitud del espectro. Algunas veces se muestra la potencia en vez de la amplitud, porque esta corresponde mejor a la percepción humana. Otra medida es la densidad de potencia

espectral o PSD, que se aplica a los espectros continuos como el ruido. Una definición simple del PSD es que es la potencia del espectro en un ancho de banda específico [2].

2.2.3.2 Variando en el tiempo

Los detalles en el espectro del tono de un único instrumento están constantemente cambiando, así que las representaciones estáticas solo representan una porción de la forma de la envolvente de un sonido. Un espectro variante en el tiempo describe los cambios de las frecuencias sobre la duración de un evento. Puede ser representado como un gráfico tridimensional del espectro contra el tiempo (figura 7). Estas representaciones son esencialmente una serie de representaciones estáticas, una después de otra.

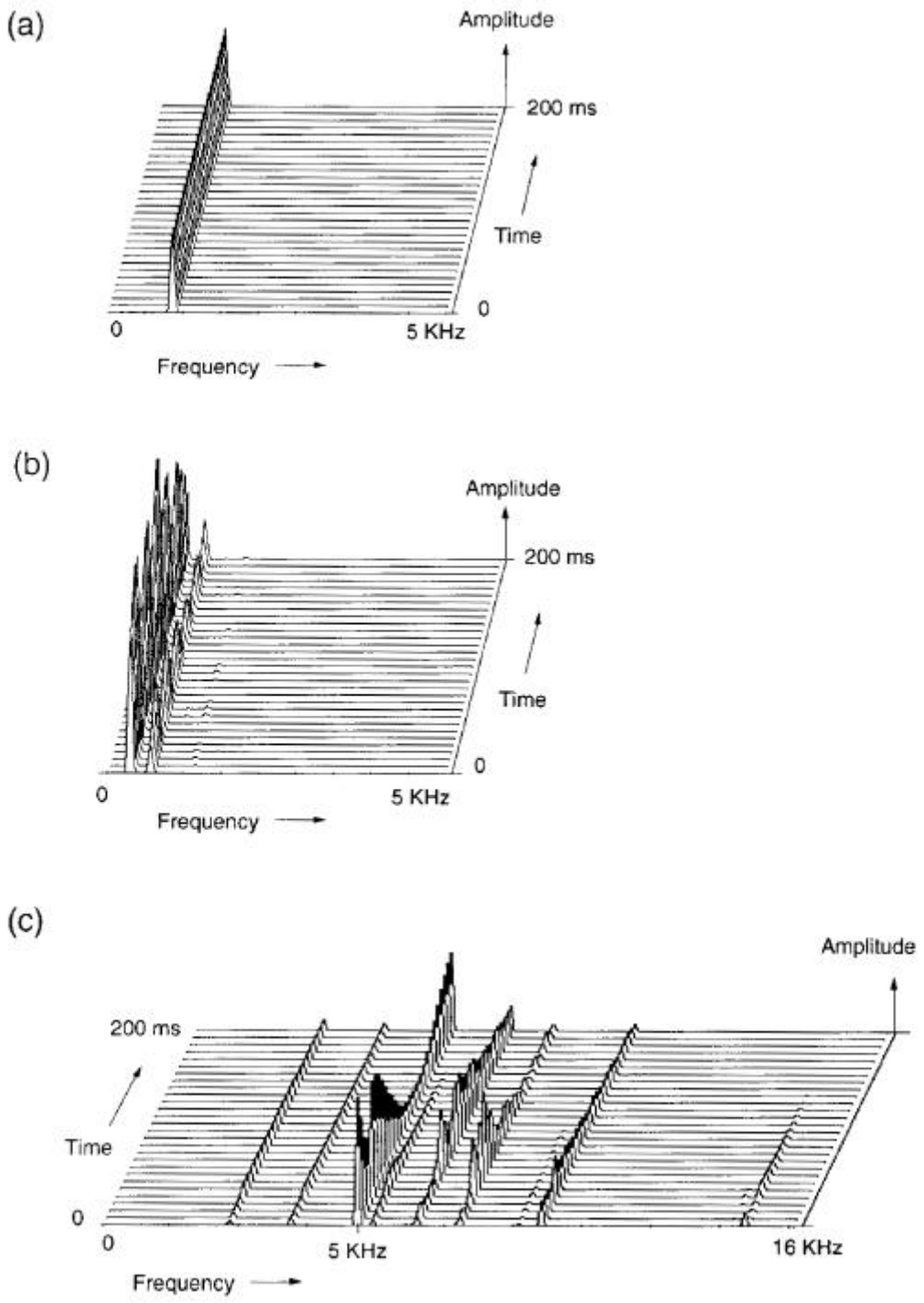


Figura 7 Representación del espectro variante en el tiempo con la escala de amplitud lineal. El tiempo va del frente al fondo. (a) Onda seno a 1 KHz. (b) Flauta interpretando con pitch E4. (c) Triángulo, golpeado una vez.

La figura 8 muestra dos formatos de representación para el análisis del espectro variando en el tiempo. La figura 8 es una fotografía de una representación en cascada (un dibujo del espectro en el que el eje del tiempo se mueve en tiempo real). El término representación en cascada viene del hecho de que este tipo de representación muestra ondas de creciente y decreciente energía de frecuencia en una descripción como un fluido. La figura 8 muestra una relación vocal.

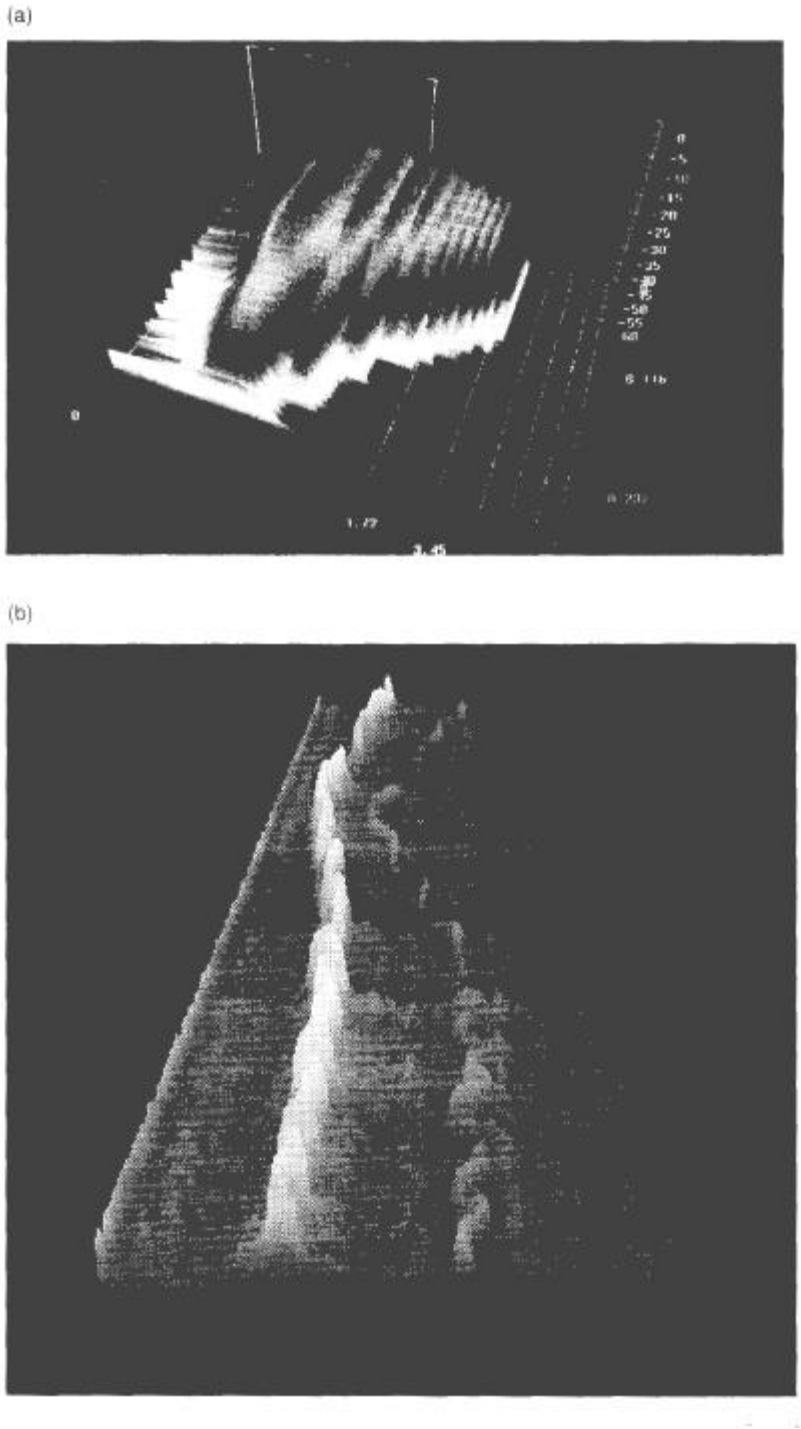


Figura 8 Imágenes de representaciones de tiempo real en cascada. (a) Tono de una trompeta sintética. El tiempo va de atrás al frente, con el tiempo más reciente en el frente. La escala de frecuencia es logarítmica, que va de la izquierda a la derecha. La frecuencia fundamental esta aproximadamente en 1KHz. La amplitud es dibujada verticalmente en una escala logarítmica de dB. (b) Melodía vocal. El tiempo va hacia el lector, con el tiempo más reciente en el frente. Las bajas frecuencias están en la izquierda.

Otra forma de mostrar el espectro con variación en el tiempo es dibujar un sonograma o espectrograma [2] (una dibujo común en el análisis de voz), que se llama originalmente habla visible. Un sonograma muestra la frecuencia contra el contenido temporal de una señal, donde la frecuencia está dibujada verticalmente, el tiempo está dibujado horizontalmente y las amplitudes de la frecuencia en el espectro son dibujadas en términos de la oscuridad del trazado. Esto es, los componentes intensos de la frecuencia están dibujados oscuros, mientras que los componentes frecuenciales bajos son dibujados claros (figura 9).

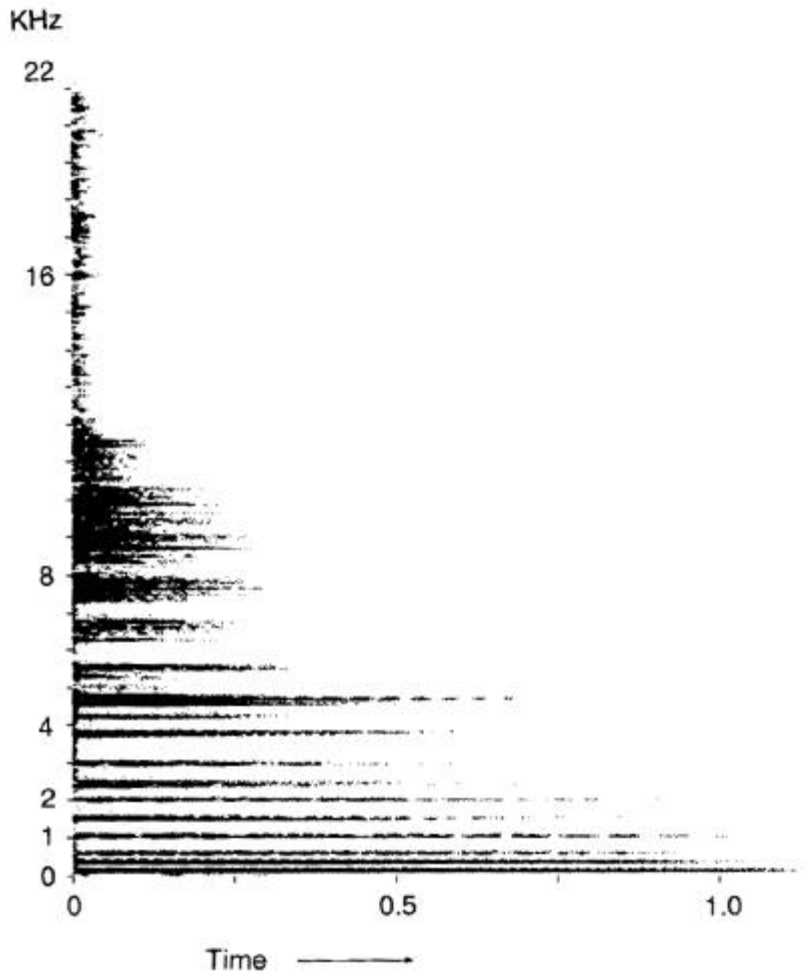


Figura 9 Dibujo de un sonograma de un golpe a un tam-tam. El eje vertical es la frecuencia, y el eje horizontal es el tiempo. Este sonograma usa 1024 puntos de datos de entrada, y una ventana Hamming. El dibujo tiene una resolución frecuencial de 43 Hz y una resolución temporal de 1 ms. El ancho de banda

del análisis va de 0 a 22 KHz, y el rango dinámico medido es -10 a 44.5 dB, dibujado en una escala de amplitud lineal.

Representación por sonograma

Un sonograma, sonografo o espectrograma es una técnica de representación del espectro muy bien conocida en la investigación del habla, que ha sido usada durante una década en el análisis del habla. Un sonograma muestra una descripción general del espectro durante varios segundos del sonido. Esto permite al observador ver las características generales como el comienzo de las notas o fonemas, picos del formante y transiciones grandes. Un observador entrenado puede leer un sonograma de voz. La representación por sonograma ha sido empleada como interfase de editores de espectro.

El sonograma original fue un sistema de Backhaus's (1932). En 1950 el sonografo de Kay era el mecanismo estándar para hacer sonogramas. Consistía en un número de filtros analógicos estrechos pasa-bandas y un sistema de grabación que imprimía barras oscuras en un rollo de papel. Las barras crecían en grosor en proporción a la energía de salida de cada filtro. Hoy los sonogramas se implementan generalmente con la STFT [2].

La figura 9 muestra un sonograma, representando una señal de un sonido en un dibujo bidimensional de tiempo contra "frecuencia + amplitud". La dimensión vertical describe la frecuencia (las altas frecuencias están en la parte superior del diagrama) y las sombras de gris indican la amplitud, donde las sombras más oscuras muestran mayor intensidad.

Un sonograma de voz puede ser modificado para soportar las demandas más severas de la música. Los sonogramas musicales tienden a ser más largos que los sonogramas de voz, incluyendo secciones o piezas enteras. El rango dinámico de la música es mucho más amplio que el vocal.

También, los sonogramas de voz están orientados para una precisa representación física del espectro, mientras que los músicos están más interesados en una visión perceptual que esté en concordancia con lo que podemos oír.

Representaciones tridimensionales del espectro

Un espectro 3D proyecta cada componente frecuencial sobre el tiempo. Esto puede tomar 2 formas:

- Representaciones en perspectiva (representación en cascada o envolvente del espectro).
- Representaciones en funciones de control.

Las representaciones en perspectiva (figura 10) muestran la topología variante en el tiempo de la energía en un terreno de la frecuencia contra la amplitud, donde la amplitud es usualmente la dimensión vertical. Cuando estas representaciones operan en tiempo real se llama cascadas, debido a sus formas ondulantes.

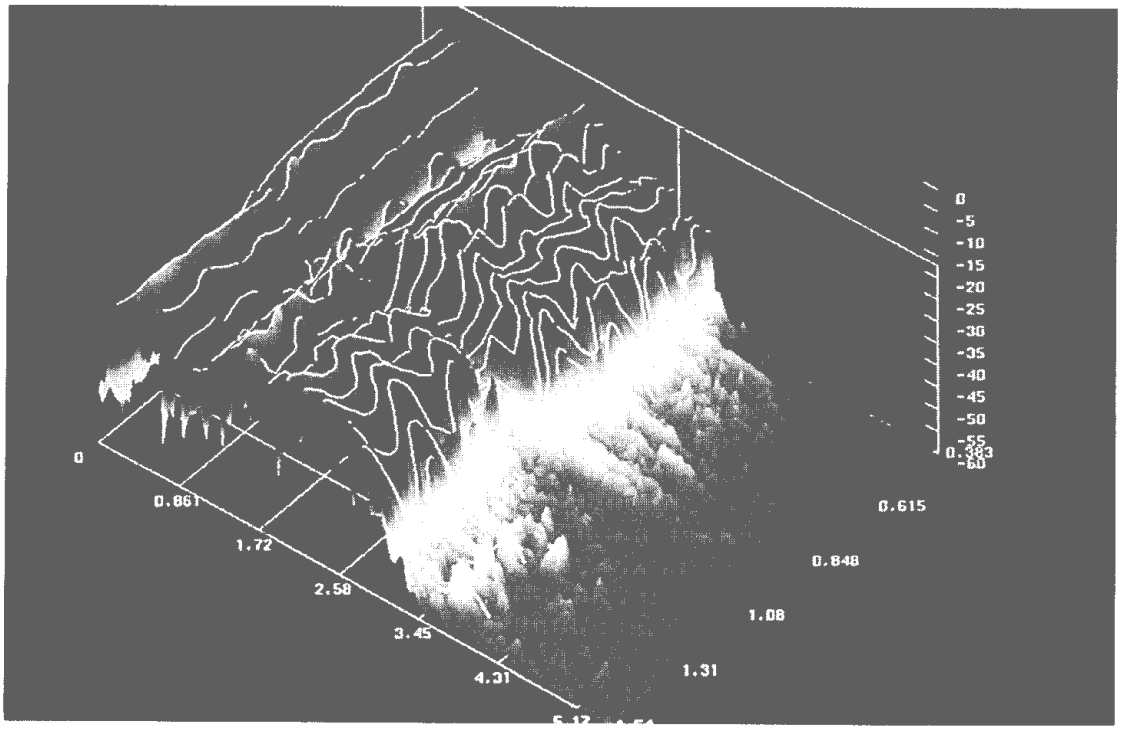


Figura 10 Dibujo en perspectiva de una síntesis aditiva de la voz de un tenor cantando dos notas. El intervalo entre las notas es visible. La frecuencia está dibujada de la izquierda a la derecha; el tiempo se proyecta de la parte de atrás al frente. La amplitud (dibujada en escala logarítmica) es el eje vertical- Uno de las ventajas de las representaciones 3D es que revela la modulación en frecuencia y amplitud en un tono. Las líneas blancas a lo largo de la parte superior de los parciales son la estimación de la envolvente de la amplitud, después de una algoritmo por X. Rodet.

Las representaciones de funciones de control [2] separan las envolventes de amplitud y de frecuencia para cada parcial (figura 11).

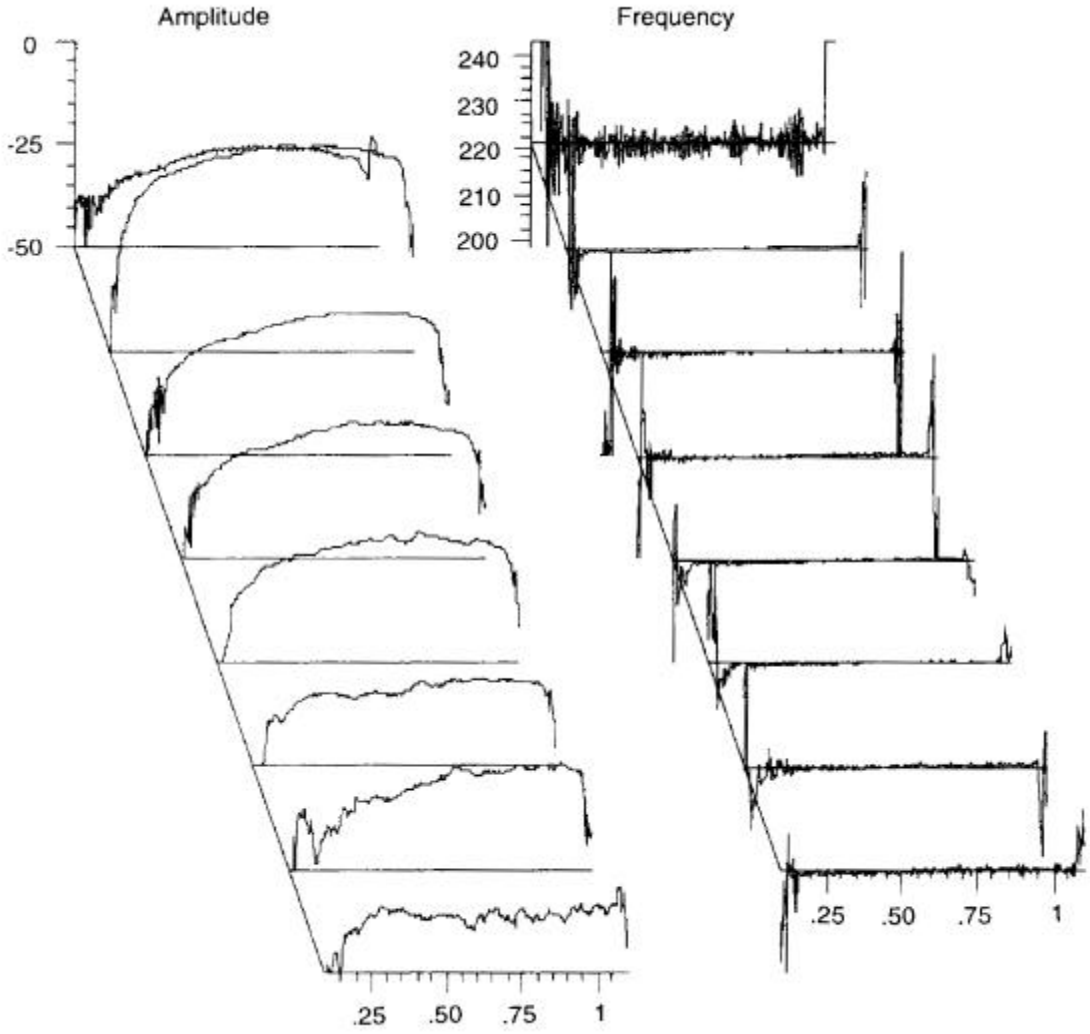


Figura 11 Dibujos separados para envolventes de la frecuencia y la amplitud de los primeros ocho armónicos de un tono de violín. Los armónicos bajos son los de atrás.

2.3 Entornos de programación de 3 dimensiones

2.3.1 Introducción

A la hora de crear aplicaciones en 3D se pueden encontrar más de 50 APIs (Application Programing Interface) [4]. ¿Cuál podemos elegir?

Si se busca información sobre muchas de estas aplicaciones 3D, las 3 primeras a las que se debe acudir son: OpenGL de Silicon Graphics, zQuick Draw 3D de Apple y Direct3D de Microsoft.

Para un trabajo multiplataforma, se ha de considerar OpenGL [6] (que está disponible en Unix, Windows NT y 95 y en Mac), especialmente si se están desarrollando aplicaciones técnicas. Es una arquitectura cliente/servidor que permite a los sistemas pequeños descargar en el servidor el procesamiento de gráficos. QuickDraw 3D (QD3D) tiene la ventaja de ejecutarse en Power Mac y Windows (NT y 95).

OpenGL, sin embargo, no maneja algunas funciones de alto nivel (p. e. formato de ficheros). QD3D y Direct3D lo hacen. QD3D también tiene abstracción del hardware, objetos ya preparados y soporte en la construcción de la edición.

¿Cuál hay que elegir? Si el programa es tanto para Mac como para Windows, se debe desarrollar en QD3D. Si se quieren tener todas las ventajas del hardware del PC (quizás para un juego), tanto Direct3D como Apple's RAVE (Rendering Acceleration Virtual Engine, la interfase independiente del dispositivo que QD3D usa) ofrecen soluciones.

2.3.2 Aplicaciones más usuales

OpenGL

OpenGL es el único motor de gráficos que se extiende más allá del PC y Mac a muchas características de Unix. En un sistema Unix ejecutando el X-Windows, una biblioteca de ampliación (llamada GLX) que consiste, aproximadamente, en una docena de llamadas, inicializa el entorno de ventanas. La biblioteca de ampliación entonces prepara un contexto de renderizado y hace una conexión con el *frame buffer* de la ventana. Las implementaciones de OpenGL en otras plataformas usan una biblioteca de interfaces similar para manejar estos detalles dependientes del sistema.

Comparado con QD3D y Direct3D, OpenGL no es una API de alto nivel. Su propósito es dibujar objetos. Tareas de alto nivel como la edición de objetos o la entrada/salida de ficheros son dejadas a la aplicación (p. e. Open Inventor de Silicon Graphics). En vez de esto, OpenGL proporciona una interfase independiente de la plataforma a un motor de renderizado de bajo nivel. Esta interfase consiste en unos 250 comandos de dibujo que permite al programador describir objetos y ejecutar un conjunto de operaciones para producir la imagen final. Se puede llamar a OpenGL desde una gran variedad de lenguajes, incluyendo C/C++, FORTRAN, Ada y Java.

La característica más interesante en OpenGL es que usa el mecanismo cliente/servidor para el procesado de gráficos. Un cliente de gráficos (p. e. una aplicación 3-D) usa la interfase de OpenGL y el sistema operativo para transmitir tanto primitivas gráficas como coordenadas de vértices, colores y coordenadas de textura a un servidor de procesamiento de gráficos que ejecuta el rasterizador de OpenGL.

El rasterizador ejecuta una o más operaciones gráficas basadas en el estado actual de OpenGL y genera pixeles. Los valores del pixel

experimentan un procesado final que incluye el mapeado de textura, *antialiasing*, test de profundidad y combinación. Esta información va al cliente para que se muestre en la ventana de la aplicación.

Normalmente, el cliente y el servidor se ejecutan en el mismo sistema. Sin embargo, la ventaja de la separación es que un sistema de bajo coste puede transmitir comandos OpenGL a través de una red a un sistema de altas prestaciones ejecutando el proceso servidor, que devuelve imágenes a la máquina de sobremesa.

OpenGL soporta un modo inmediato y un modo retenido para las operaciones gráficas. En el modo inmediato, una aplicación manda comandos gráficos a OpenGL, que inmediatamente los ejecuta. Esto proporciona una respuesta rápida a cualquier cambio en la imagen, una característica crítica para las aplicaciones de gráficos interactivos.

En el modo retenido, secuencias de comandos gráficos son almacenadas en estructuras de datos conocidos como *display lists*. Los *display lists* tienen dos ventajas. Primero, si se tiene que mostrar frecuentemente un objeto complejo, solo se tiene que referir a su *display list*. Segundo, los *display lists* permiten que la información del objeto sea rápidamente encaminada a través de una red. Una desventaja es que las modificaciones frecuentes en un objeto generan nuevas descripciones de objetos.

Open Inventor es un conjunto de herramientas 3-D que proporciona un gran número de funciones para la construcción de gráficos como editores de colores y texturas. Está disponible para estaciones de trabajo y para Windows. Graba los gráficos 3-D en un fichero de formato Inventor que es un superconjunto del *Virtual Modeling Language* (VRML), que es el estándar de la industria del formato de los ficheros que describen los objetos 3-D en la Web.

A OpenGL le falta el diseño oriento a objetos como tiene QD3D o Direct3D, pero está disponible en Unix, Windows (NT y 95) y Mac. Es también un estándar, administrado por un grupo llamado el Architecture Review Board, entre cuyos miembros se incluyen Digital Equipment, Microsoft y Evans & Sutherland.

OpenGL soporta un amplio rango de entornos gráficos que cubren todo el espectro en términos de corte y rendimiento. En la parte baja, proporciona software de renderizado para PCs de sobremesa. Pero se puede comunicar directamente con estaciones de trabajo de gama alta equipadas con hardware de visualización que pueden dibujar 10 millones de polígonos por segundo. Con sus primitivas de gráficos es ideal para CAD y programas de diseño de arquitectura.

QuickDraw 3D

QD3D es un entorno completo de gráficos 3-D. El nivel más alto es una API que permite a los programadores crear y manipular objetos 3-D y leer o escribir datos 3-D de ficheros. Esta API se comunica con un motor de gráficos extensible que procesa las operaciones de dibujo. El motor a su vez se comunica con una fina capa de abstracción del hardware (HAL) que aporta una API independiente del dispositivo para los diseñadores. (Esta HAL es conocida como QD3D RAVE y esta disponible como un equipo de desarrollo separado tanto para Mac como para Windows).

QD3D soporta tanto modo inmediato como modo retenido. El modo inmediato es idéntico al de OpenGL en la forma en que la aplicación suministra comandos de dibujo al renderizador. En el modo retenido las estructuras de programación orientada a objetos mantienen la geometría de la escena para representación visual y su manipulación.

Mientras el modo inmediato ofrece un control muy afinado para las aplicaciones que lo necesiten (p. e. animación, donde todos los objetos cambian constantemente), el modo retenido deja almacenar la geometría de la escena en una base de datos de objetos. En el modo retenido es fácil leer y escribir objetos, y permite que las estructuras de datos sean almacenadas para una rápida exhibición o aceleración hardware. Los programadores pueden usar un modo de imagen exclusivamente o ambos, si la situación lo pide.

QD3D es un sistema gráfico orientado a objetos que lo diferencia radicalmente de OpenGL. Un nuevo ejemplo de un objeto puede heredar las características de su clase, incluyendo geometría, tamaño, orientación, color, texturas e iluminación, que hace que la construcción de los objetos de una escena sea rápida. Esto simplifica el mantenimiento de la información de cada objeto para la manipulación y la representación.

Los comandos de alto nivel de la API permite crear, girar, editar, iluminar y aplicar transformaciones a objetos. Un mecanismo de *widgets* proporciona manipuladores visuales que permite editar o escalar un objeto interactivamente. Gracias al diseño orientado a objetos de QD3D, no es necesario conocer la estructura interna de los objetos 3-D para realizar estas acciones. Actualmente solo se puede usar C/C++ para hacer llamadas a QD3D.

También difiere de OpenGL en que permite leer o escribir imágenes 3-D en el formato *3-D metafile* (3DMF). Este formato no solo graba la geometría del objeto, sino su iluminación y textura. También, 3DMF permite a las aplicaciones copiar y pegar gráficos 3-D o insertarlo en otras aplicaciones.

Apple proporciona el código fuente ANSI de las funciones que leen ficheros 3DMF. Se pueden encontrar versiones UNIX, Windows y Mac.

Netscape, Silicon Graphics y Apple han anunciado el desarrollo de un formato de fichero binario para Moving Worlds, una especificación entre plataformas para entornos dinámicos 3-D en el Web, basado en el formato 3DMF.

QD3D es para quien quiera que el sistema operativo se encargue de los detalles técnicos 3-D. La API proporciona una amplia variedad de objetos básicos (p. e. líneas, esferas y conos) que permite realizar un prototipo de la escena rápidamente. Proporciona una interfase que permite editar visualmente estos objetos. Su arquitectura es ampliable, ya que se puede obtener acceso a aceleración hardware. Se puede copiar y pegar gráficos entre aplicaciones o grabar información dentro de un formato 3DMF para varias plataformas. Como ha sido aceptado como una parte de Moving Worlds, el formato 3DMF de QD3D se convierte de hecho en un estándar. Finalmente, QD3D está disponible en Mac y PC.

Direct3D

Direct3D es la API 3-D de Microsoft. Es una nueva adición a la familia DirectX de tecnología multimedia de la compañía. Se puede encontrar solamente en Windows 95. Direct3D provee de un entorno excelente para los programadores 3-D para Windows 95, especialmente para los de juegos. Sin embargo, no existen capacidades multiplataforma.

Aunque no sea multiplataforma, Direct3D esconde algunas de las diferencias entre los fabricantes de aceleradoras 3-D hardware. Esto lo realiza usando una HAL. La interfase HAL deja a los programadores obtener información sobre las características hardware que hay debajo, incluyendo especificaciones del dispositivo e información del funcionamiento. Como algunos sistemas no tienen aceleración hardware, una Hardware Emulator Layer (HEL) emula estas funciones en software. La HEL asegura que todos los servicios de Direct3D están disponibles para

la aplicación en cualquier sistema, posiblemente con un rendimiento reducido.

Como OpenGL y QD3D, Direct3D soporta modos de operación inmediato y retenido. El modo retenido de Direct3D se parece a QD3D en que ofrece a los programadores una interfase para aplicaciones de alto nivel orientada a objetos para manipular objetos 3-D. Después se puede cargar un objeto usando una llamada API. Se puede girar y escalar con otra llamada API. El modo retenido proporciona llamadas API que leen y escriben un formato de fichero que almacena datos 3-D. Estos datos consisten en objetos 3-D predefinidos, superficies, texturas y conjuntos de animación. Este formato de fichero deja a las aplicaciones intercambiar información 3-D y ejecutar secuencias de animación en tiempo real.

El modo inmediato es una delgada capa API que trata con polígonos y vértices. La API de modo inmediato pasa *display lists* compuestas por datos de vértices y comandos gráficos (conocidos como *execute buffers* en terminología Microsoft) al motor de renderizado para el procesamiento. Este sistema proporciona un mecanismo de alto rendimiento independiente del dispositivo que deja a los programadores acceder al sistema hardware de gráficos o introducirnos dentro las aceleradoras hardware. Este modo no ofrece ningún motor de manejo de objetos o de escenas; las funciones son responsabilidad de la aplicación.

El modo inmediato es más apropiado para programadores que tienen una aplicación con su propio motor de renderizado, ya que es deseable acceder al sistema hardware de aceleración. Actualmente, se usa C/C++ para acceder a las llamadas Direct3D. Esta planeado el soporte para Visual Basic.

Direct3D ofrece una gran variedad de objetos gráficos. En el modo inmediato, estos objetos incluyen *execute buffer*, matrices (para

transformaciones), iluminación, textura, material (una propiedad de la superficie del objeto), *viewport* (la región de la pantalla ocupada por el objeto), dispositivo (el hardware manejador de la pantalla) y interfase. En el modo retenido proporciona objetos adicionales basados en objetos del modo inmediato, incluyendo caras de polígonos, superficies, *frame* (maneja la posición de todos los objetos en una escena y su orientación), sombreado, animación (usado para aplicar transformaciones a un objeto, tanto en escalado como en posición).

Direct3D es más parecido a OpenGL en que provee primitivas de dibujo, pero no objetos básicos como esferas, cilindros o conos, tal como hace QD3D. Ahora mismo no está claro si tales objetos estarán disponibles en una librería separada o, en cambio, se tendrán que construir. También está ausente la información para la edición interactiva de estos objetos.

Direct3D hace tiempo que ha llegado, pero elimina todos los problemas hardware que hay tanto para programadores 3-D como diseñadores de juegos. Esta y otras tecnologías DirectX prometen limpiar al PC de acciones de bajo nivel, posiblemente proporcionando independencia real del dispositivo para todo el software, no solo para aplicaciones 3-D. Como QD3D, tiene una arquitectura ampliable con aceleración hardware. También proporciona funciones de animación que pueden ser usadas por programadores de juegos y multimedia. Pero, Direct3D está disponible solo en sistemas con Windows 95, que limita su audiencia a corto plazo [4].

2.3.3 OpenGL

OpenGL ("GL" de *Graphics Library*) proporciona características avanzadas en renderizado tanto en modo inmediato como en modo de lista de visualización. OpenGL es una interfase software al hardware gráfico [6]. La interfase consiste en un conjunto de alrededor de un centenar de

procedimientos y funciones que permiten al programador especificar los objetos y las funciones involucradas en producir imágenes de alta calidad gráfica, específicamente imágenes en color de objetos tridimensionales. OpenGL integra dibujo 3-D dentro de sistemas de ventanas o puede ser usado sin este.

OpenGL dibuja primitivas dentro de un *framebuffer* sujeto a varios modos que se pueden seleccionar. Cada primitiva es un punto, segmento de línea, polígono, pixel rectangular o mapas de bits. Cada modo puede ser cambiado independientemente; el establecimiento de uno no afecta a los otros (aunque muchos modos pueden interaccionar para determinar como acabará el *framebuffer*). Los modos son establecidos, las primitivas especificadas, y otras operaciones OpenGL se describen mandando comandos en forma de función o llamada a procedimientos.

Primitivas geométricas (puntos, segmentos de líneas y polígonos) están definidos por un grupo de uno o más vértices. Un vértice define un punto, un punto final de un borde o la esquina de un polígono donde dos bordes se cruzan. Los datos (consistentes en coordenadas de posición, colores, normales y coordenadas de textura) están asociados con un vértice y cada vértice es procesado independientemente, en orden y de la misma forma. La única excepción a esta regla es si el grupo de vértices ha de ser recortados si la primitiva indicada está dentro de una región específica; en este caso los datos de los vértices pueden ser modificados y nuevos vértices pueden ser creados. El tipo de recorte depende de que primitiva representa el grupo de vértices.

OpenGL proporciona control directo sobre las operaciones fundamentales en gráficos 2-D y 3-D. Esto incluye especificaciones de los parámetros de las matrices de transformación, coeficientes de la ecuación de iluminación, métodos de *antialiasing* y operaciones de actualización de pixel. No proporciona un medio para describir o modelar objetos geométricos

complejos. Otra forma de describir esta situación es decir que OpenGL proporciona mecanismos para describir como serán renderizados objetos geométricos complejos en vez de mecanismos para describir los objetos complejos por ellos mismos.

El modelo de interpretación de los comandos OpenGL es el cliente - servidor. Esto es, un programa (el cliente) proporciona comandos, y estos comandos son interpretados y procesados por OpenGL (el servidor). El servidor puede o no operar en la misma computadora que el cliente.

Los efectos de los comandos OpenGL en el *framebuffer* son controlados finalmente por el sistema de ventanas que asigna recursos al *framebuffer*. Es el sistema de ventanas el que determina a que porciones del *framebuffer* pueden acceder OpenGL en un momento dado y comunica a OpenGL como estas porciones están estructuradas. De forma parecida, visualizar los contenidos del *framebuffer* en un monitor CRT (incluyendo la transformación de valores individuales del *framebuffer* con técnicas como la corrección gamma) no está direccionado por OpenGL. La configuración del *framebuffer* ocurre fuera de OpenGL en conjunción del sistema de ventanas; la inicialización de un contexto OpenGL ocurre cuando el sistema de ventanas asigna una ventana para el renderizado de OpenGL. Adicionalmente, OpenGL no tiene facilidades para obtener entrada del usuario, ya que se espera que cualquier sistema de ventanas en el que se esté ejecutando OpenGL proporcione estas facultades. Estas consideraciones hacen OpenGL independiente de cualquier sistema de ventanas particular.

2.3.3.1 Operación básica en OpenGL

La figura 12 muestra un diagrama esquemático de OpenGL. Los comandos de OpenGL entran por la izquierda. Muchos comandos pueden ser

acumulados en una *display list* para su procesado posterior. Si no, los comandos son efectivamente mandados a través del *pipeline* de procesado.

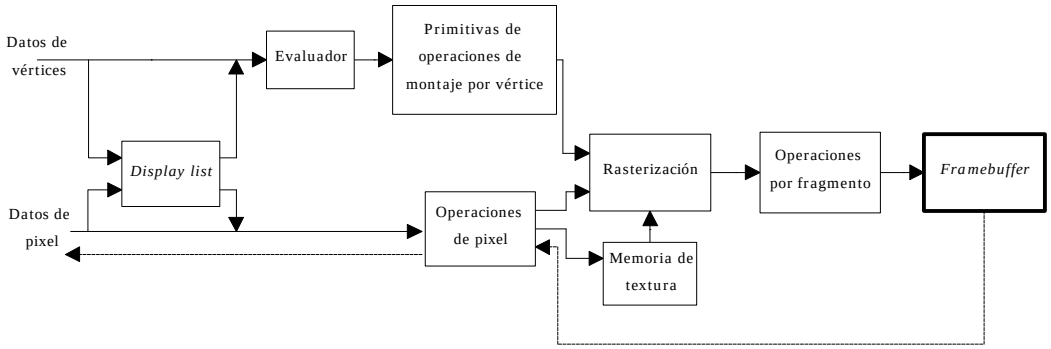


Figura 12 Diagrama de bloques de OpenGL

La primera etapa proporciona un medio eficiente para aproximar la geometría de curvas y superficies evaluando las funciones polinomiales de los valores de entrada. La próxima etapa opera en las primitivas geométricas descritas por vértices: puntos, segmentos de línea y polígonos. En esta etapa los vértices son transformados e iluminados, y las primitivas son recortadas con el volumen de visualización para preparar el próximo estado, la rasterización. El rasterizador produce una serie de direcciones y valores de *framebuffer* usando los valores de la descripción bidimensional de un punto, segmento de línea o polígono. Cada *fragmento* producido es dado al siguiente estado que realiza las operaciones en los fragmentos individuales antes de que modifiquen finalmente el *framebuffer*. Estas operaciones incluyen actualizaciones condicionales dentro del *framebuffer* basadas en el valor de profundidad entrante y el previamente almacenado (para que tenga efecto el *depth buffering*[6]), combinación de colores de los fragmentos entrantes con los colores almacenados, así como el enmascarado y otras operaciones lógicas en los valores de los fragmentos.

[6] Algoritmo de eliminación de líneas y caras ocultas.

Finalmente, los rectángulos de pixeles y mapas de bits evitan la porción del *pipeline* del procesado de vértices para mandar un bloque de fragmentos directamente a través de la rasterización a las operaciones de los fragmentos individuales, finalmente haciendo que un bloque de pixeles sea escrito en el *framebuffer*. Los valores pueden ser leídos otra vez desde el *framebuffer* o copiados de una porción del *framebuffer* a otra. Estas transferencias pueden incluir algún tipo de decodificación o codificación.

La Librería de Utilidades de OpenGL

Una de las principales guías en el diseño de OpenGL fue el proveer la portabilidad al programa sin decir como se han de describir los objetos gráficos de alto nivel. Como resultado, la interfase básica de OpenGL no soporta algunos objetos geométricos que están tradicionalmente asociados a los estándares gráficos. Por ejemplo, una implementación de OpenGL no necesita renderizar polígonos cóncavos. Una razón para esta omisión es que los algoritmos de renderizado de polígonos cóncavos son más complejos que los de renderizado de polígonos complejos, y algoritmos diferentes para polígonos cóncavos pueden ser apropiados en diferentes dominios. En particular, si un polígono cóncavo va ha ser dibujado más de una vez, es más eficiente descomponerlo en polígonos convexos (o triángulos) y entonces dibujar los polígonos convexos.

Un algoritmo general de descomposición de polígonos cóncavos es proporcionado como una parte de la Librería de Utilidades de OpenGL, que es proporcionada con cada implementación de OpenGL. La Librería de Utilidades también proporciona una interfase, construidos en evaluadores polinomiales de OpenGL, para describir y visualizar NURBS curvas y superficies (con dominio del recorte del espacio), así como recursos para renderizar esferas, conos y cilindros. La Librería de Utilidades sirve tanto para renderizar objetos geométricos útiles y como modelo para construir otras librerías que usan OpenGL para renderizar.

2.3.3.2 El pipeline de OpenGL

Vértices y primitivas

En OpenGL, muchos objetos geométricos son dibujados rodeando una serie de coordenadas que especifican vértices y opcionalmente normales, coordenadas de textura y colores entre la pareja de comandos **glBegin/glEnd**. Por ejemplo, para especificar un triángulo con vértices en **(0,0,0)**, **(0,1,0)** y **(1,0,0)**, se debe escribir:

```
glBegin(GL_POLYGON)
     glVertex3i(0,0,0);
     glVertex3i(0,1,0);
     glVertex3i(1,0,1);
glEnd();
```

Objeto	Interpretación de los vértices
Point	Cada vértice describe la localización de un punto.
Line strip	Serie de segmentos de línea conectados; cada vértice después del primero describe el final de cada segmento.
Line loop	Lo mismo que line strip pero el segmento final es añadido desde el vértice final al primer vértice.
Separate line	Cada par de vértices describe un segmento de línea.
Polygon	Lazo de líneas formado por vértices describe la frontera de un polígono convexo.
Triangle strip	Cada vértice después de los dos primeros describe un triángulo dado por este vértice, y los dos previos.
Triangle fan	Cada vértice después de los dos primeros describe un triángulo dado por este vértice, el vértice previo y el

Objeto	Interpretación de los vértices
	primer vértice.
Separate triangle	Cada triada consecutiva de vértices describe un triángulo.
Quadrilateral strip	Cada par de vértices después de los dos primeros describe un cuadrilátero dado por este par de vértices y los dos anteriores
Independent quad	Cada grupo consecutivo de cuatro vértices describe un cuadrilátero.

Tabla 1 glBegin/glEnd objetos

Los diez objetos geométricos que son dibujados de esta forma están resumidos en la tabla 1. Este grupo particular de objetos fue seleccionado porque la geometría de cada objeto esta especificada con una lista simple de vértices, porque cada una admite un algoritmo eficiente de renderizado y porque se determinó que tomar juntos estos objetos satisface las necesidades de todas las aplicaciones gráficas.

Cada vértice puede ser especificado con dos, tres o cuatro coordenadas (cuatro coordenadas indica una localización homogénea tridimensional). Además, una *normal actual*, *coordenadas de textura actual* y *color actual* pueden ser usados en el procesado de cada vértice. OpenGL usa normales en los cálculos de la iluminación; la normal actual es un vector tridimensional que puede ser establecido con tres coordenadas que lo especifiquen. El color puede consistir en cada valor de rojo, verde, azul y alfa (cuando OpenGL ha sido inicializado en modo RGBA) o un único índice de valor del color (cuando la inicialización especifica el modo de color indizado). Una, dos, tres o cuatro coordenadas de textura determina como una imagen de textura se mapea dentro de una primitiva.

Cada uno de los comandos que especifica coordenadas de vértices, normales, colores o coordenadas de textura viene de diferentes formas para

acomodar los diferentes formatos de datos de las aplicaciones y número de coordenadas. Los datos pueden ser pasados a cualquiera de estos comandos como una lista de argumentos o como un puntero a un bloque de almacenamiento conteniendo los datos. Las variantes son distinguidas (en el lenguaje c) con sufijos nemotécnicos.

Muchos comandos OpenGL que no especifican vértices e información asociada no pueden aparecer entre **glBegin** y **glEnd**. Esta restricción permite a las implementaciones ejecutar en un modo optimizado el procesado de las especificaciones de primitivas y así las primitivas pueden ser procesadas lo más eficientemente posible.

Cuando un vértice es especificado, el color, la normal y las coordenadas de textura actuales son usados para obtener valores que son asociados con el vértice (figura 13). El vértice mismo es transformado por la matriz de visualización, una matriz 4x4 que puede representar tanto transformaciones lineales como de translación. El color es obtenido calculando el color desde la iluminación o, si la iluminación está invalidada, del color actual. Las coordenadas de textura son pasadas similarmente a través de una función generadora de coordenadas de textura (que puede ser la identidad). Las coordenadas de textura resultantes son transformadas por la matriz de textura (esta matriz puede ser usada para escalar o girar eficazmente una textura que es aplicada a una primitiva).

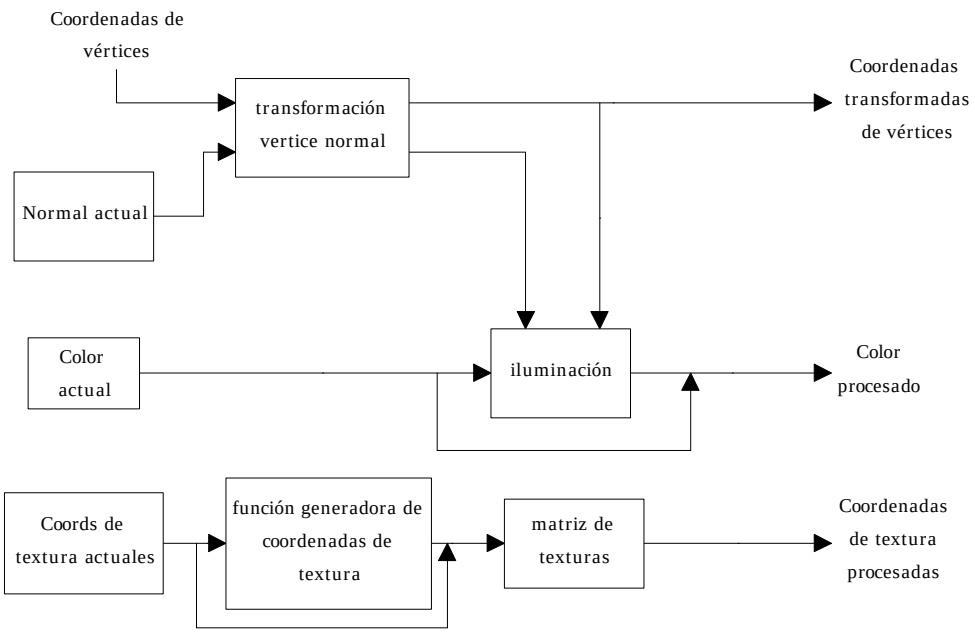


Figura 13 Asociación de los valores actuales con un vértice

Algunos comandos controlan los valores de los parámetros usados en el procesado de vértices. Un grupo de comandos manipula las matrices de transformación; estos comandos están diseñados para formar un método eficiente para generar y manipular las transformaciones que ocurren de modo jerárquico en las escenas de gráficos 3-D. Una matriz puede ser cargada o multiplicada por un escalado, rotación, translación o una matriz general. Otros comandos controla que matriz está afectada por una manipulación: la matriz de visualización, la matriz de textura o la matriz de proyección. Cada una de estos tres tipos de matrices también tiene asociada una pila donde las matrices pueden ser metidas o sacadas.

Los parámetros de iluminación son agrupados dentro de tres categorías: parámetros de material, que describe las características de reflexión de la superficie a iluminar, parámetros de la fuente de iluminación, que describe las propiedades de emisión de cada fuente de luz, y los parámetros del modelo de iluminación, que describe las propiedades globales del modelo de iluminación. La iluminación se lleva a cabo basándose en cada vértice; los resultados de la iluminación son finalmente

interpolados a lo largo de un segmento de línea o polígono. La forma general de la ecuación de iluminación incluye términos para iluminación constante, difusa y especular, que puede ser atenuada por la distancia del vértice a la fuente de luz. Un programador puede sacrificar realismo en favor de un cálculo más rápido en la iluminación indicando que el espectador, las fuentes de luz o ambos están infinitamente lejos de la escena.

Recorte y proyección

Una vez que la primitiva ha sido ensamblada desde un grupo de vértices, es sometido a recorte por los planos de corte. Las posiciones de estos planos (cada implementación de OpenGL debe proveer como mínimo seis) son especificables usando el comando **glClipPlane**. Cada plano puede ser activado o desactivado individualmente.

En el caso de un punto, los planos de recorte pueden no tener efecto en el punto o eliminarlo dependiendo de sí el punto se encuentra dentro o fuera del espacio determinado por los planos de recorte. En el caso de un segmento de línea o de un polígono, los planos de recorte pueden no tener efecto, eliminarlo o alterar la primitiva original. En este último caso, nuevos vértices pueden ser creados entre los bordes descritos para los vértices originales; los valores de color y coordenadas de textura par estos nuevos vértices son encontrados interpolando los valores asignados a los vértices originales.

Después de que los planos de corte (si hay) han sido aplicados, las coordenadas de los vértices de la primitiva resultante son transformados por la matriz de proyección. Entonces se produce el recorte contra el volumen de visualización. El recorte contra el volumen de visualización es como el recorte contra los planos de recorte, pero con planos fijos: si las coordenadas después de la transformación son dadas por **(x, y, z, ω)**, entonces los seis

espacios medios definidos por estos planos son **$-\omega \leq x$, $x \leq \omega$, $-\omega \leq y$, $y \leq \omega$, $-\omega \leq z$, $z \leq \omega$**.

Cuando el recorte contra el volumen de visualización esta acabado, cada grupo de coordenadas de vértices es proyectado calculando **x/ω, y/ω y z/ω**. Los valores resultantes (que deben encontrarse entre [-1, 1]) son multiplicados y compensados por los parámetros que controlan el tamaño del *viewport* donde las primitivas van a ser dibujadas. Los comandos **glViewport** (para **x/ω** y **y/ω**) y **glDepthRange** (para **z/ω**) controlan estos parámetros.

Rasterización

La rasterización convierte una primitiva proyectada y escalada en el *viewport* en una serie de fragmentos. Cada fragmento consta de la localización de un pixel en el *framebuffer* junto con el color, coordenadas de textura y profundidad (**z**). Cuando un segmento de línea o polígono es rasterizado, sus datos asociados son interpolados a lo largo de la primitiva para obtener el valor de cada fragmento.

La rasterización de cada tipo de primitiva es controlado por su correspondiente grupo de parámetros. Uno puede afectar a la rasterización de un punto y otro puede afectar a la rasterización de un segmento de línea. Adicionalmente, una secuencia de puntos puede especificar un segmento de línea y unos puntos de estructura pueden especificar un polígono.

El *antialiasing* puede ser activado o desactivado individualmente para cada tipo de primitiva. Cuando está activado, se calcula un valor de recubrimiento para cada fragmento describiendo la porción de este fragmento que es recubierta por la primitiva de proyección. Este valor de recubrimiento se usa después de que se ha puesto textura para modificar el

valor alfa de los fragmentos (en modo RGBA) o el valor del índice de color (en modo índice de color).

Rectángulos de pixeles y mapas de bits

Rectángulos de pixeles y mapas de bits son las dos primitivas que no están afectadas por las operaciones geométricas que se producen en el *pipeline* anterior a la rasterización. Un pixel rectangular es un conjunto de valores destinados al *framebuffer* (normalmente los valores representan colores, aunque están disponibles para otro tipo de datos, tales como valores de profundidad). Los valores, guardados como un bloque de datos en memoria principal, son mandados usando **glDrawPixels**. Los argumentos de **glDrawPixels** indican la dirección de memoria de los datos, el tipo de datos, y el ancho y alto del rectángulo que los valores de los datos forman. Además, dos grupos de parámetros para controlar la decodificación de los valores almacenados. El primer grupo describe como los valores son empaquetados en memoria y proporciona un medio para seleccionar un subrectángulo dentro del rectángulo mayor. El segundo grupo controla las conversiones que pueden ser aplicadas a los valores después de que se han obtenido: los valores pueden ser escalados, compensados y mapeados por medio de las *lut*. Estos parámetros forman un medio flexible para especificar imágenes rectangulares guardadas en una variedad de formatos.

Una vez obtenidos, los valores resultantes producen un rectángulo de fragmentos. La localización de este rectángulo esta controlada por la posición actual del rasterizador, que es tratado de forma parecida a un punto (incluyendo la asociación de un color y coordenadas de textura a él), excepto de que es asignado por un comando separado (**glRasterPos**) que no se encuentra entre un **glBegin** y un **glEnd**. El tamaño del rectángulo esta determinado tanto por su propia altura y anchura como por los parámetros de zoom del rectángulo de pixeles (asignados por **glPixelZoom**).

Un mapa de bits es similar a un rectángulo de pixeles, excepto en que especifica un rectángulo de ceros y unos, y está diseñado para describir caracteres que son colocados en un lugar de la proyección 3-D (a través de la actual posición de rasterización). Cada uno de los mapas de bits produce un fragmento que tiene valores asociados que son los de la actual posición del rasterizador, mientras que cada cero no produce ningún fragmento. El comando **glBitmap** también especifica compensaciones que controlan como el mapa de bits está colocado con respecto a la posición actual del rasterizador y como avanzará la actual posición del rasterizador después de que el mapa de bits ha sido dibujado (esto determina las posiciones relativas de mapas de bits secuenciales).

Texturas y niebla

OpenGL proporciona un medio general para generar primitivas de mapeado de texturas. Cuando el mapeado de texturas esta activado, cada coordenada de textura del fragmento indica una textura de la imagen, generando un *textel*. Este textel puede tener entre uno y cuatro componentes, así la textura de la imagen pude representar, por ejemplo, solo intensidad (una componente), color RGB (tres componentes) o color RGBA (cuatro componentes). Una vez el textel ha sido obtenido, modifica el color del fragmento de acuerdo con el entorno específico de textura.

La textura de una imagen es especificada usando **glTexImage**, que toma argumentos similares a los de **glDrawPixels**, así el mismo formato de imagen puede se usado tanto si la imagen está destinado para el *framebuffer* como para la memoria de textura. Además, **glTexImage** puede ser usado para especificar mipmaps y así la textura puede ser filtrada y aplicada a la primitiva. La función de filtrado (tanto si implica mipmaps como si no) está controlada por un número específico de parámetros usando **glTexParameter**. El entorno de la textura es seleccionado por **glTexEnv**.

Finalmente, después de mapear la textura, una función de niebla (si está activada) es aplicada a cada fragmento. La función de niebla mezcla el color entrante con una constante (especificable) de color de niebla de acuerdo con factor calculado. Este factor es una función de la distancia (o una aproximación de la distancia) desde el observador al punto 3-D que corresponde al fragmento. Niebla exponencial simula niebla atmosférica y neblina, mientras que niebla lineal puede ser usada para producir profundidad.

El Framebuffer

El destino de los fragmentos rasterizados es el *framebuffer*, donde los resultados del renderizado de OpenGL pueden ser visualizados. En OpenGL, el framebuffer consiste en una matriz rectangular de pixeles que corresponde a la ventana destinada por OpenGL para el renderizado. Cada pixel es simplemente un conjunto de números de bits. Los bits correspondientes a cada pixel en el *framebuffer* están agrupados juntos en un plano de bits; cada plano de bits contiene un único bit de cada pixel.

Los planos de bits están agrupados dentro de buffers lógicos: los buffers de color, profundidad, plantilla y acumulación. El buffer de color es donde la información del color de un fragmento está guardado. El buffer de profundidad es donde la información de profundidad está puesta y es normalmente usada para la eliminación de caras ocultas por medio del *z-buffering*. El buffer de plantilla contiene valores cada uno del cual puede ser actualizado cada vez que un fragmento alcanza el framebuffer. Los valores de plantilla son útiles en algoritmos de multi-paso, en donde una escena es renderizada varias veces, para lograr efectos como operaciones CSG (unión, intersección y diferencia) en un número de objetos y tapar los objetos cortados por los planos de recorte.

Operaciones Por-Fragmento

Antes de ser colocado dentro de su posición en el *frame buffer*, un fragmento esta sujeto a una serie de tests i modificaciones, cada uno de los cuales puede ser individualmente activado, desactivado o controlado. Los tests y modificaciones incluyen test de plantilla, test de buffer de profundidad (típicamente usado para realizar la eliminación de superficies ocultas) y combinación. Ahora se describirá brevemente solo un subconjunto de tests.

El test de plantilla, cuando está activado, compara el valor en el buffer de plantilla correspondiente al fragmento con el valor de referencia. Si la comparación tiene éxito, entonces el valor almacenado de plantilla puede ser modificado por una función como incremento, decremento o limpieza y el fragmento prosigue con el siguiente test. Si el test falla, el valor guardado puede ser cambiado por una función diferente, y el fragmento es desechado. De forma parecida, el test de buffer de profundidad compara el valor de profundidad del fragmento con el valor correspondiente almacenado en el buffer de profundidad. Si la comparación tiene éxito, el fragmento es pasado a la siguiente etapa y el valor de profundidad del fragmento reemplaza el valor almacenado en el buffer de profundidad (si el buffer ha sido activado para escritura). Si la comparación falla, el fragmento es desechado y no se produce ninguna modificación en el buffer de profundidad.

La combinación mezcla el color de un fragmento con el color que ha sido guardado en el *framebuffer* (la combinación tiene lugar para cada uno de los buffers de colores que han sido activados para escritura). La función exacta de combinación puede ser especificada con **glBendFunction**.

Combinación es la operación que realiza el *antialiasing* para colores RGBA. Recalcar que el cálculo de la cobertura solo modifica el valor alfa del fragmento; este valor alfa debe ser usado para combinar el color del

fragmento con el color guardado de fondo para obtener el efecto *antialiasing*. El combinado también es usado para realizar transparencias.

Además para modificar los valores individuales del *framebuffer* con una serie de argumentos, un buffer entero o los buffers pueden ser limpiados con una valor constante. Limpiar los valores puede ser realizado para los buffers de color (todos los buffers de color comparten un único valor), de plantilla, de profundidad y el de acumulación

Funciones diversas

Evaluadores

Los evaluadores permiten la especificación de funciones polinomiales de una o dos variables cuyos valores determinan las coordenadas de los vértices de las primitivas, coordenadas de las normales, color o coordenadas de textura. Un mapa polinomial, especificado en términos de Bezier puede ser dado por uno de estos grupos de valores. Una vez definido y activado, los mapas son invocados de dos formas. La primera forma es causar una única evaluación de cada mapa activado especificando un punto en el dominio de los mapas usando **glEvalCoord**. Este comando debería estar entre **glBegin** y **glEnd** así las primitivas individuales pueden ser construidos con aproximaciones de una porción de curvas o superficie. El segundo método es especificar una trama en el dominio espacial usando **glEvalMesh**. Cada vértice de la trama es una función de los polinomios definidos. **glEvalMesh** genera sus propias primitivas y no puede ser colocadas entre **glBegin** y **glEnd**.

El evaluador proporciona un paquete con una base para construir curvas y superficies de forma general sobre OpenGL. Una ventaja de proporcionar los evaluadores en OpenGL en vez de una interfase más compleja de NURBS es que las aplicaciones que representan curvas y

superficies o otras que usan NURBS o hacen uso de propiedades especiales de las superficies aun tiene acceso a evaluadores polinomiales eficientes (quizás implementados en hardware gráfico) sin los costes de convertir en una representación NURBS.

Display List

Una *display list* encapsula un grupo de comandos OpenGL que pueden ser mostrados más tarde (en el orden en que han sido especificados originalmente) simplemente llamando a la *display list*. Esto se consigue circundando los comandos que van a ser encapsulados con **glBeginList** y **glEndList**. **glBeginList** coge un entero como argumento que es el nombre numérico de la *display list*.

Las *display lists* pueden ser redefinidas, pero no editadas. La falta de edición simplifica el manejo de memoria de las *display lists* en el servidor de OpenGL, eliminando las consecuencias que el manejo pueda crear. Las *display lists* pueden ser anidadas (una *display list* puede invocar a otra). Un efecto similar a la edición de puede ser obtenido: (1) construir una lista conteniendo listas subordinadas; (2) redefinir las listas subordinadas.

Una única *display list* es invocada con **glCallList**. **glCallLists** llama a una serie de *display lists* en sucesión. El argumento de **glCallLists** especifica un array de enteros que son añadidos a una lista base para formar la serie de números de *display list*. **glCallLists** es útil para mostrar una serie de caracteres cuando los comandos que generan cada carácter han sido encapsulados en su propia *display list*.

Feedback y selección

Como se ha descrito antes, OpenGL renderiza las primitivas en el *framebuffer*. OpenGL tiene dos modos adicionales. El modo *feedback*

devuelve información sobre primitivas (coordenadas de vértices, color y coordenadas de textura) después de que han sido procesadas pero antes de que sean rasterizadas. Este modo es útil, por ejemplo, si la salida de OpenGL es hacia un plotter en vez de un *framebuffer*.

En el modo de selección, OpenGL devuelve un *hit* cuando una primitiva (recortada) se encuentra dentro del volumen de visualización. Este modo es usado, por ejemplo, para determinar que porciones de una escena se encuentra dentro de una región centrada alrededor de la posición del ratón (esto se llama *picking*). La Librería de Utilidades proporciona rutinas para manipular las transformaciones para que cuando la escena es redibujada, solo las porciones que se encuentren en una región específica dentro de una posición determinada devolverán *hits*. Cada *hit* devuelve los contenidos de la pila de selección, que pueden ser manipulados cuando la escena es dibujada. Con una manipulación apropiada de la pila, la aplicación puede identificar las características de la escena que intersecciona la región seleccionada.

Estado OpenGL

Finalmente, el valor de algún parámetro de OpenGL puede ser obtenido con el comando *get* apropiado. También los valores de la pila que pueden ser introducidos o sacados. Para propósitos de pila, todos los parámetros están divididos en 21 grupos funcionales; cualquier combinación de estos grupos puede ser introducida dentro de la pila de atributos en una operación (una operación automática de salida restaura solo los últimos valores introducidos). Los comandos de obtención y la pila de parámetros hacen posible implementar varias librerías, cada una sin interferir con otros usos de OpenGL.

2.3.3.3 La integración en un sistema de ventanas

OpenGL dibuja escenas 2-D y 3-D dentro de un *framebuffer*, pero para ser útil en entonos heterogéneos, OpenGL debe subordinarse a un sistema de ventanas que asigna y controla los recursos del *framebuffer*. Ahora se describirá como OpenGL está integrado dentro de *X Window System*, pero la integración en otros sistemas de ventanas es similar [10].

X proporciona una interfase de procedimientos y un protocolo de red para crear y manipular el *framebuffer* de las ventanas y dibujar ciertos objetos 2-D en esas ventanas. OpenGL está integrado dentro de X haciendo una extensión formal de X llamada GLX. GLX consiste en una docena de llamadas (con la correspondiente codificación de red) que proporciona una forma compacta y general de integrar OpenGL en X. Como con otras extensiones X (dos ejemplos son *Display PostScript* y *PEX*), hay un protocolo de red específico para encapsular los comandos de renderizado en la corriente de bits de X.

OpenGL requiere una región de un *framebuffer* dentro del cual las primitivas pueden ser renderizadas. En X, esta región es llamada *drawable*. Una ventana, un tipo de *drawable*, es asociada con una *visual* que describe la configuración del *framebuffer* de la ventana. En GLX, el *visual* es ampliado para incluir información sobre los *buffers* de OpenGL que no están presentes en X (profundidad, plantilla, acumulación, frente, trasero, etc.).

X también proporciona un segundo tipo de *drawable*, el *pixmap*, que es un *framebuffer* fuera de pantalla. GLX proporciona un *GLX pixmap* que corresponde a un *pixmap*, pero con *buffers* adicionales como los indicados en el *visual*. EL *pixmap* de GLX proporciona un medio a las aplicaciones de OpenGL para renderizar fuera de pantalla dentro de un *buffer* software.

Para poder usar un *drawable* de OpenGL, el programador crea un contexto de OpenGL apuntando al *drawable*. Cuando se crea el contexto, una copia del renderizador de OpenGL es inicializado con la información

visual del *drawable*. Este renderizador de OpenGL es conceptualmente una parte del servidor X, pero, una vez creado, un cliente X puede conectar con el contexto OpenGL y emitir comandos OpenGL (figura 14). Pueden ser creados varios contextos de OpenGL que pueden apuntar a *drawables* distintos o compartir uno. Un *drawable* de OpenGL puede ser usado por la herramienta estándar de dibujo de X (los *buffers* de un *drawable* que no son usados por X son ignorados por él). Las llamadas proporcionan el dibujo sincronizado entre OpenGL y X; y es responsabilidad del cliente cumplir con esta sincronización si es requerida.

Un cliente GLX que se está ejecutando en una computadora que tiene un subsistema gráfico puede evitar pasar símbolos de OpenGL al servidor. Este renderizado directo puede dar como resultado un incremento en el rendimiento gráfico ya que se elimina el sobregasto de codificar, decodificar y repartir los símbolos. El renderizado directo está soportado pero no es requerido por GLX (un cliente determina cuando un servidor proporciona renderizado directo). Renderizado directo es factible porque no es necesario mantener la secuencialidad entre los comandos X y los comandos OpenGL excepto cuando los comandos han sido explícitamente sincronizados.

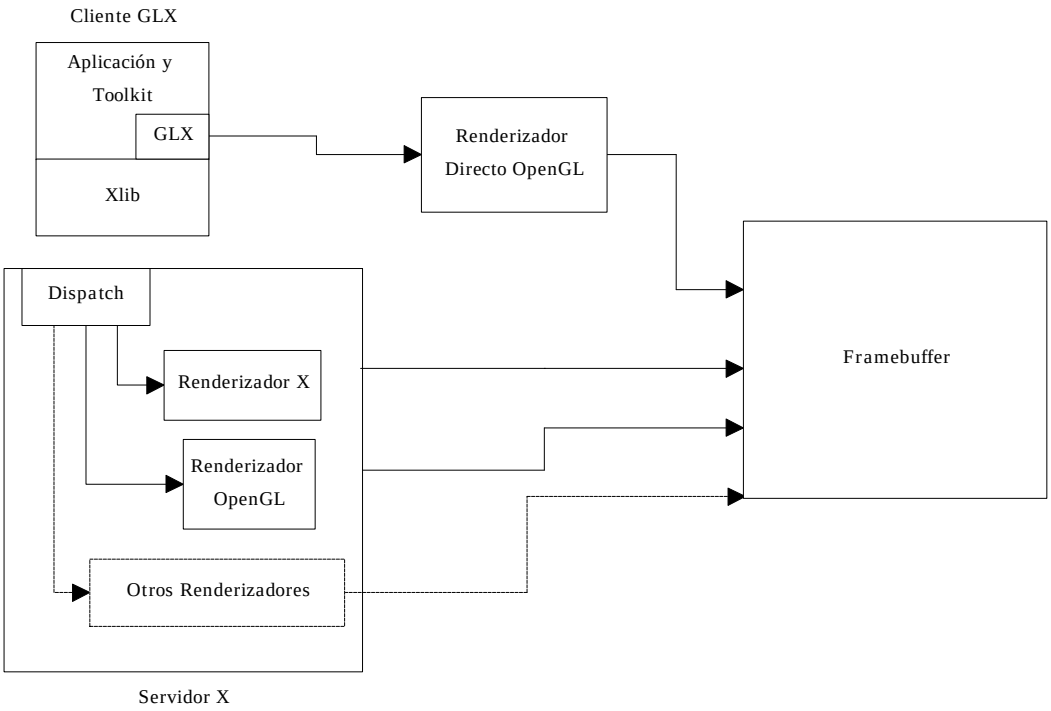


Figura 14 Cliente GLX, servidor X y renderizador OpenGL

2.3.3.4 OpenGL y Windows de Microsoft

Con la versión 3.5 de NT, OpenGL formó parte del sistema operativo Windows. Ahora como Windows 95 soporta OpenGL y las aceleradoras gráficas de bajo precio van a ser una realidad en máquinas domésticas, la perspectiva de usar OpenGL en cualquier máquina Windows se hará más atractiva cada día.

El mínimo requerimiento para montar un programa Windows que muestre gráficos OpenGL será el siguiente [10]:

Como un Graphical Device Interface (GDI) necesita un Device Context (DC) para dibujar imágenes, OpenGL necesita un Rendering Context (RC). A diferencia de GDI, que en cada comando GDI necesita que un DC sea pasado, OpenGL usa el concepto de RC actual. Una vez que el contexto de renderizado se ha hecho en un *thread*, todas las llamadas OpenGL en ese *thread* usará el mismo contexto de renderizado. Mientras que múltiples contextos de renderizado pueden ser usados para dibujar en una única ventana, solo un contexto de renderizado puede ser el actual a un tiempo en un único *thread*.

3 Desarrollo de la aplicación

3.1 Opciones adoptadas

Cuando se comenzó a desarrollar la aplicación, se adoptaron las siguientes decisiones:

- Se usaría OpenGL a la hora de representar los datos del análisis realizado por SMS debido a su facilidad a la hora de realizar dibujos técnicos, ser soportado por Windows y ofrecer un gran aislamiento del hardware.
- La representación del sonido se haría por medio de espectrogramas tridimensionales para continuar con la representación que se realiza en el programa SMS.
- La representación gráfica seria en perspectiva con un ángulo de 60 grados.
- En principio es una aplicación separada del programa original SMS y que leería los datos de un fichero.
- Para aumentar la velocidad se podrá disminuir la resolución del residuo y calcular el volumen de visualización para representar solo lo que se encuentre dentro.

A continuación de explica con detalle cada una de estas opciones adoptadas y el funcionamiento de la aplicación.

3.2 Representación gráfica

3.2.1 Introducción

SMS tiene un entorno de control gráfico en el cual se puede experimentar fácilmente y hacer música (figura 15) [11]. Representa gráficamente todos los tipos de datos implicados en los diversos procesos y puede interaccionar entre ellos. Tiene la posibilidad de realizar *zooms* de los diferentes gráficos para poder observar los pequeños detalles. Tiene una estructura de ventanas flexible que puede mostrar uno o más gráficos simultáneamente y que automáticamente se reorganiza cuando se elimina o se añade un nuevo gráfico. Tiene diferentes *workspaces* para el análisis o para la síntesis y se puede pasar de uno a otro mediante un menú.

A la hora de representar el residuo se usa un sonograma mientras que para los parciales se usa una representación estática del espectro. Faltaba una representación tridimensional tanto del residuo como del espectro para que se completase la información.

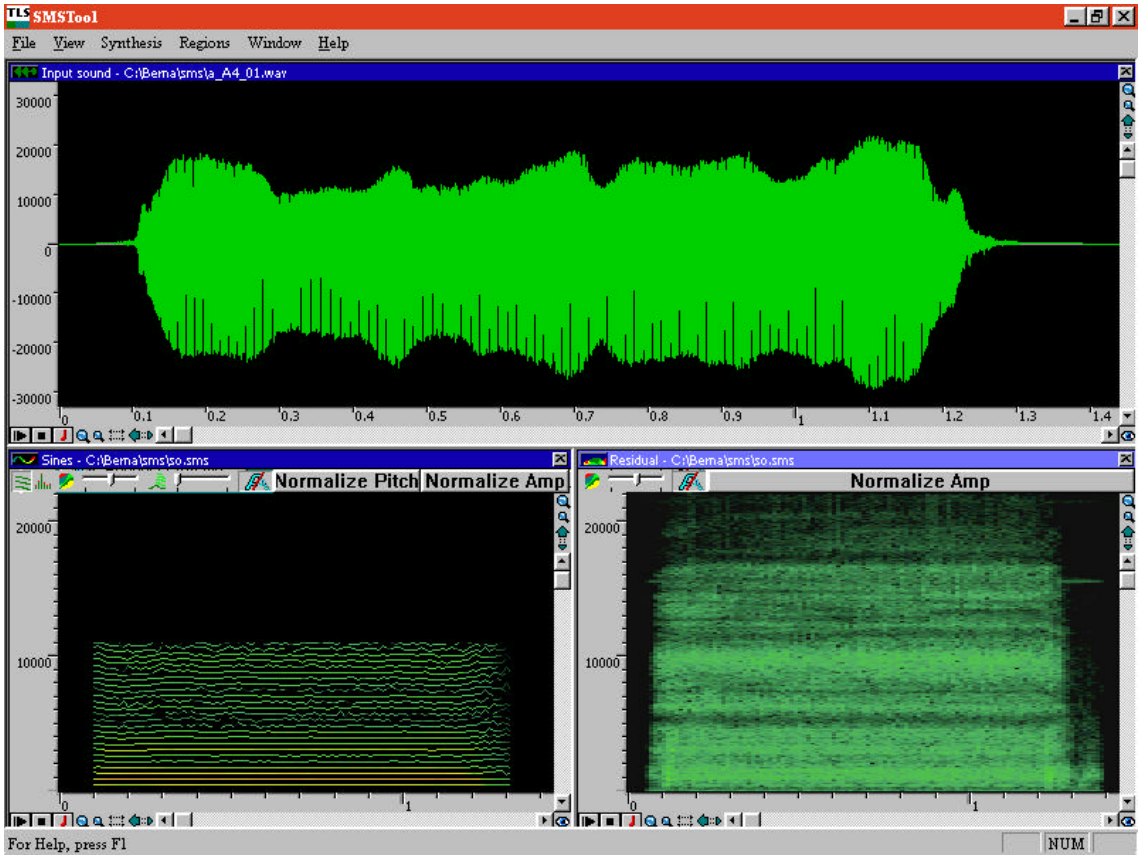


Figura 15 Pantalla de SMS Tools para el análisis de un sonido de voz

3.2.2 El lenguaje Visual C++

Es un lenguaje de programación en C++ con un entorno de diseño integrado basado en ventanas. Tiene muchas herramientas visuales, es decir, destinadas a aplicaciones basadas en ventanas y gráficos. Diseñar y modificar menús, cursores, iconos y mapas de bits es extremadamente sencillo. Y incorporarlos en un programa también es muy fácil. Además, todo esto se realiza en un solo entorno denominado *Developer Studio*, creado por *Microsoft* [3].

Dispone de una completa librería C++ destinada a crear fácilmente y de una manera intuitiva aplicaciones basadas en ventanas. Se denomina *Microsoft Foundation Class* (MFC). Entre otras cosas, se encarga de gestionar todo el entorno de ventanas y menús, de controlar todos los

mensajes, tanto de sistema como de control (ratón, teclado,...) i se implementa fácilmente una respuesta del programa a los mensajes. Si no fuese por las MFC, programar directamente las API sería una tarea mucho más larga y monótona, ya que hay mucho código repetitivo que se debería de escribir para controlar las ventanas y los mensajes.

Si se escribe un programa con la librería MFC, entonces es fácil llevarlo a otras plataformas. Las plataformas que pueden compilar un código nativo a partir de MFC son PC, PowerPC, Macintosh, Alpha AXP y UNIX.

3.2.3 Representación gráfica de diferentes tipos de datos

Antes de comenzar a hablar del entorno basado en ventanas que permite trabajar con la representación 3D del proceso de análisis SMS, primero veremos que tipos de datos hacen falta representar gráficamente y como se ha hecho.

La representación de los parciales encontrados y del residuo que resulta de sustraer los parciales al sonido analizado se hace según tres ejes. Estas son el tiempo, la frecuencia y la magnitud.

3.2.3.1 Parciales (Sinusoides)

Lo que interesa representar son las trayectorias de los parciales, es decir, como evolucionan en el tiempo. Cada parcial equivale a una sinusoide con un valor de magnitud, frecuencia y fase. La magnitud, a parte de ser un eje, también esta representada según los colores, variando del violeta al azul en orden creciente. La fase se omite, ya que no se considera importante dibujarla. Hace falta recordar que los procesos SMS están basados en *frames* y, por tanto, solo se tienen datos en ciertos momentos. Si solo

representásemos los parciales en el centro de las ventanas de análisis, obtendríamos series de puntos. Pero cada parcial representa una sinusoide continua que varia lentamente en frecuencia. Por tanto, es lógico unir los puntos de cada *frame* y dibujar las trayectorias. Estas se dibujan linealmente.

Podemos ver en la figura 16 una representación de los parciales encontrados en el análisis de un sonido de voz. En el eje que se aleja se encuentran las frecuencias que varían entre 0 Hz y 20 KHz aproximadamente. En el eje vertical se ve la magnitud que varia entre 0 y -50 y el tiempo varia de 0 a 1 s. Gracias a los colores que indican la magnitud se puede apreciar donde están las frecuencias más altas. Nos hemos movido desde la posición original para que se pudiera ver la gran mayoría de los parciales.

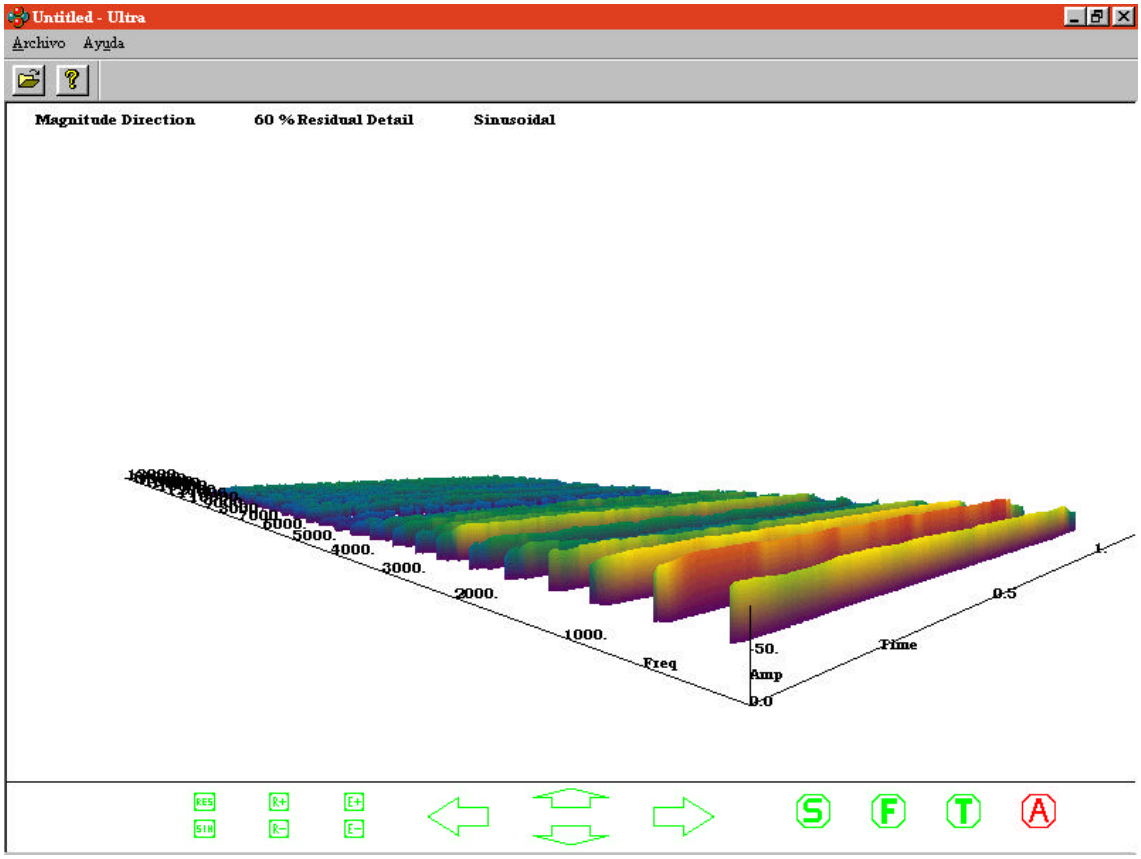


Figura 16 Parciales de un sonido de voz.

3.2.3.2 Residuo

Lo importante del residuo es la envolvente de magnitud y como evoluciona en el tiempo. Por tanto intervienen tres variables: tiempo, frecuencia y magnitud. La magnitud, además de ser un eje, también está se muestra mediante el color que va variando del violeta al rojo en orden creciente. Como de la envolvente de amplitud solo se tienen muestras discretas es apropiado representarlas en bloques. Así, la ventana del residuo se rellenará con rectángulos que variaran de color sin dejar ningún espacio vacío.

Podemos ver en la figura 17 la representación gráfica del residuo obtenido en el análisis de un sonido de voz. Como se puede apreciar gracias a los colores, la magnitud del residuo es muy baja.

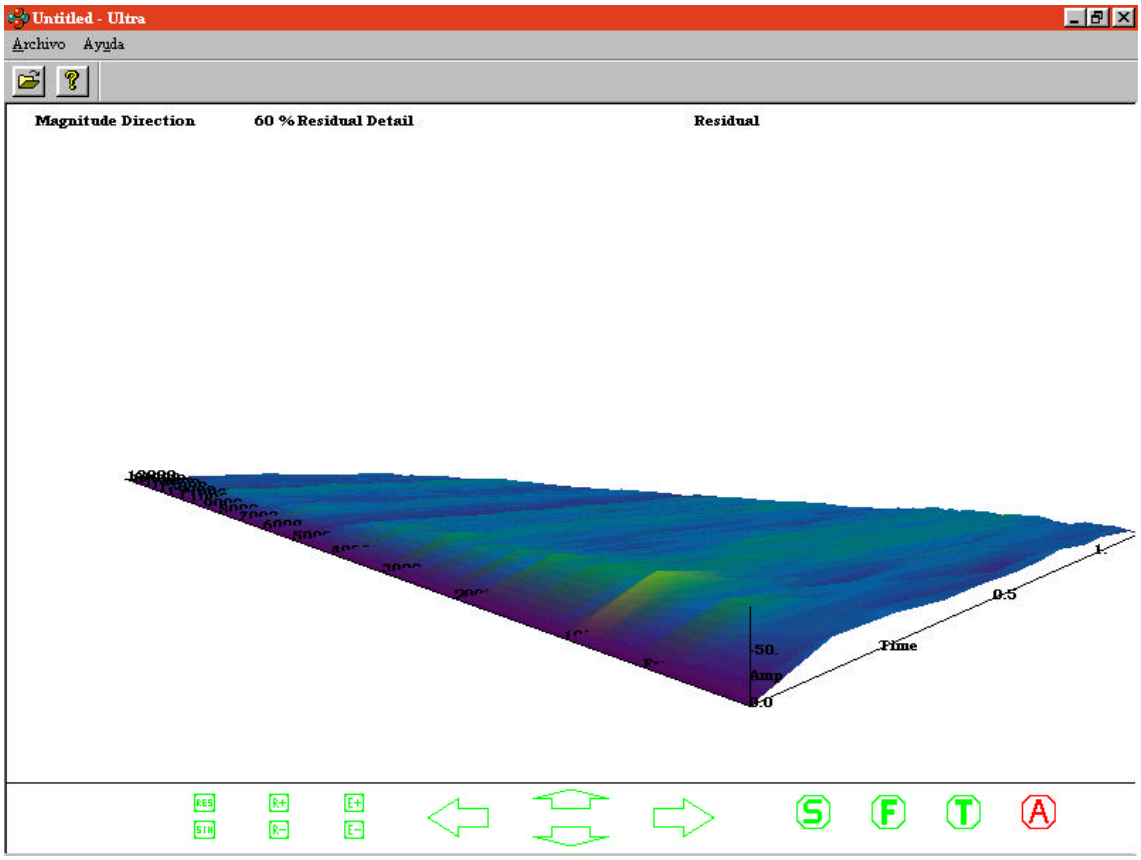


Figura 17 Representación del residuo obtenido en el análisis de una melodía

3.2.3.3 Parciales y residuo

Para representar conjuntamente los parciales y el residuo únicamente se juntan ambas.

Como se aprecia en la figura 18 al tener ambas representaciones juntas se ven los cambios en el residuo y en los parciales a lo largo del tiempo y su magnitud de una forma sencilla y práctica.

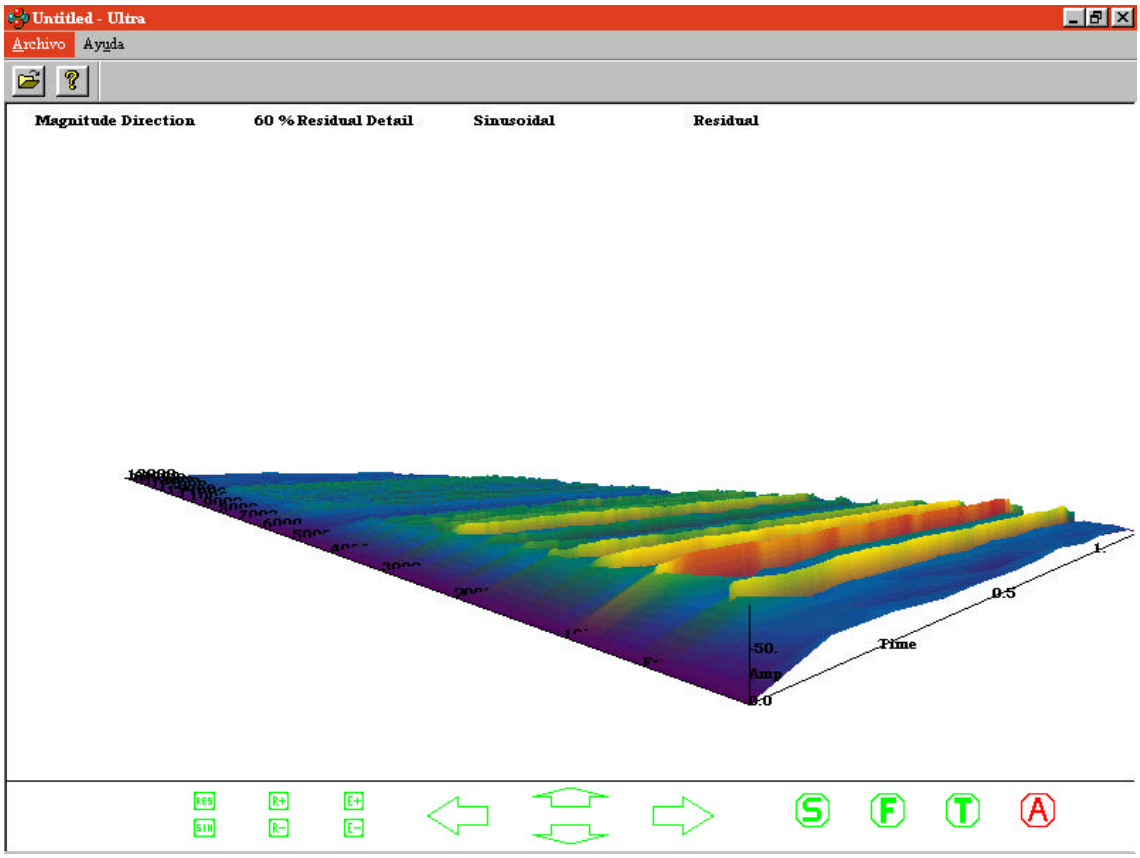


Figura 18 Representación conjunta de parciales y residuo

3.2.4 La ventana principal

La interfase gráfica del programa está estructurada de la misma manera que la mayoría de aplicaciones de Windows. Encontramos en la parte superior la barra del título con los botones de minimizar, maximizar y acabar y el icono del programa.

Justo debajo está la barra de menús.

Y la barra de iconos que activa las diferentes operaciones.

Podemos ver como que la ventana entera en la figura 19.

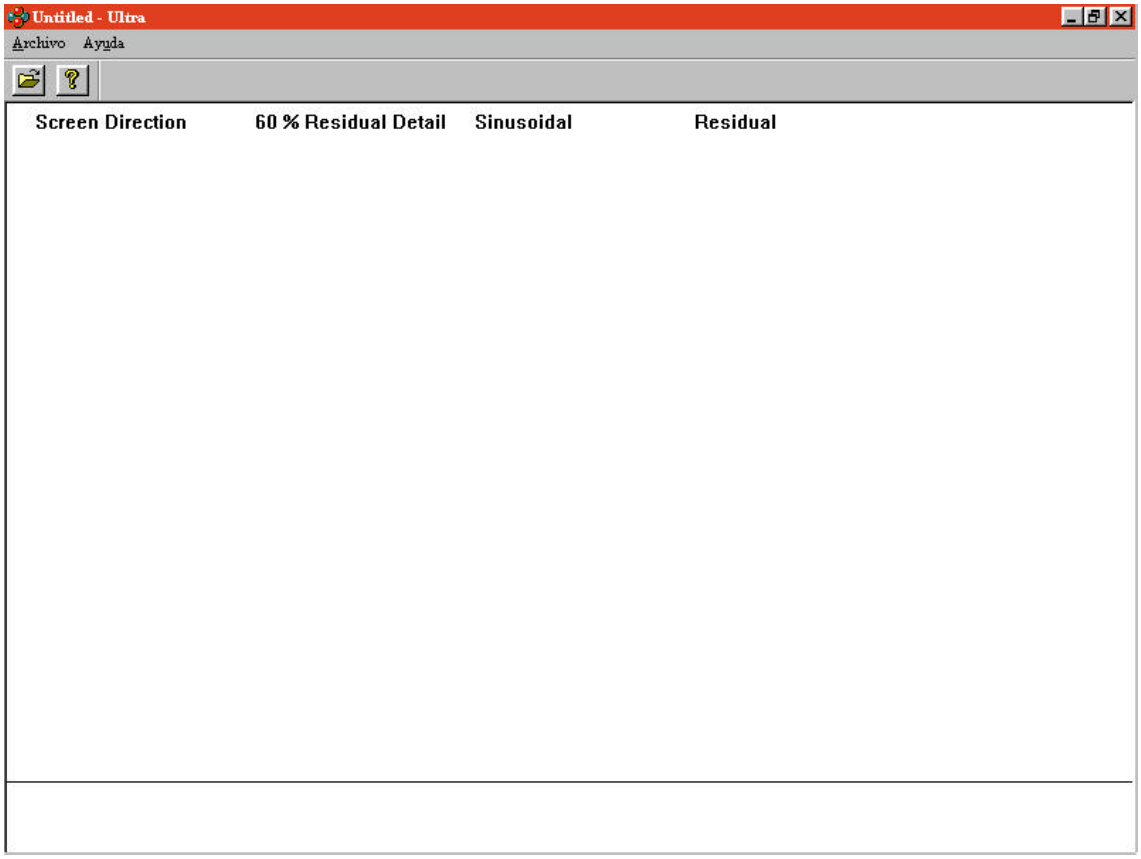


Figura 19 Ventana principal de la aplicación

3.2.5 Espacio de trabajo

Hay un único espacio de trabajo donde se muestran la representación gráfica tridimensional del sonido analizado. Este espacio de trabajo está dividido en dos partes, en una está la representación y en la parte inferior el control de los movimientos y de la visualización de la representación.

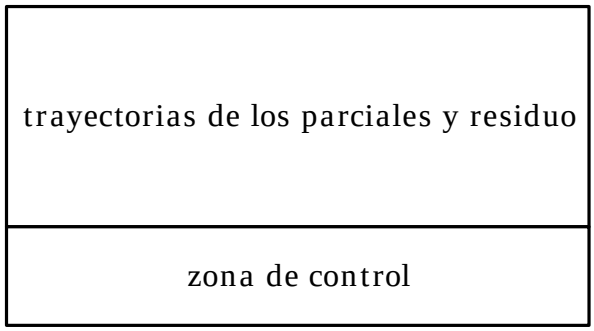


Figura 20 Distribución dentro de la zona de trabajo

En la izquierda el control de la visualización, donde se puede mostrar solo los parciales (SIN) o el residuo (RES), aumentar o disminuir la resolución del residuo y aumentar o disminuir la escala del eje seleccionado

En el centro se encuentran las flechas de movimiento para moverse o para rotar. La flecha que indican arriba y abajo nos permiten desplazarnos en el sentido positivo y negativo de los ejes respectivamente. En el caso de que se haya elegido el eje S (dirección de pantalla), nos acercaremos o alejaremos de la representación. Las flechas izquierda y derecha rotan en la dirección de las agujas del reloj o al contrario en la dirección de los ejes. Si se ha elegido el eje S, no rotará la figura sino que rotaremos nosotros.

A la derecha está el eje sobre el que se van ha realizar los movimientos o el escalado.

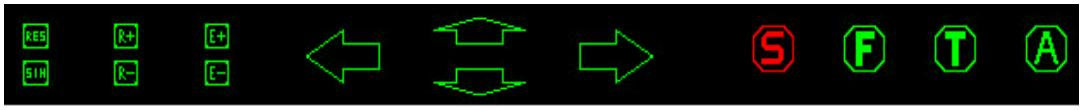


Figura 21 Zona de control del área de trabajo

3.2.6 Conclusiones

En este capítulo se han visto los criterios que se han seguido a la hora de diseñar la interfase gráfica que trabaje con la representación tridimensional del análisis de un sonido mediante SMS.

Primero se ha explicado la manera en que se pueden representar los tipos de datos implicados en los procesos de análisis. Hemos visto como es diferente la representación de los parciales y del residuo. Se ha visto como nos podemos trasladar por toda la representación para una mejor percepción

de un punto en concreto o un acercamiento a una zona que nos interese. Como elegir lo que se quiere ver (parciales o residuo) y como adaptar la representación para una mejor visualización de los componentes del análisis.

3.3 Modularización de la representación gráfica

3.3.1 Introducción

Como desde un comienzo el proyecto se iba ha realizar en C++, para aprovechar todas sus características se debería de usar clases a la hora de programas la aplicación. En primer lugar se tuvieron que distinguir los diferentes procesos que hacía falta implementar y diseñar para cada uno los módulos (clases) necesarios.

3.3.2 Clases

Las operaciones necesarias para la representación del análisis de un sonido están en la figura 22.

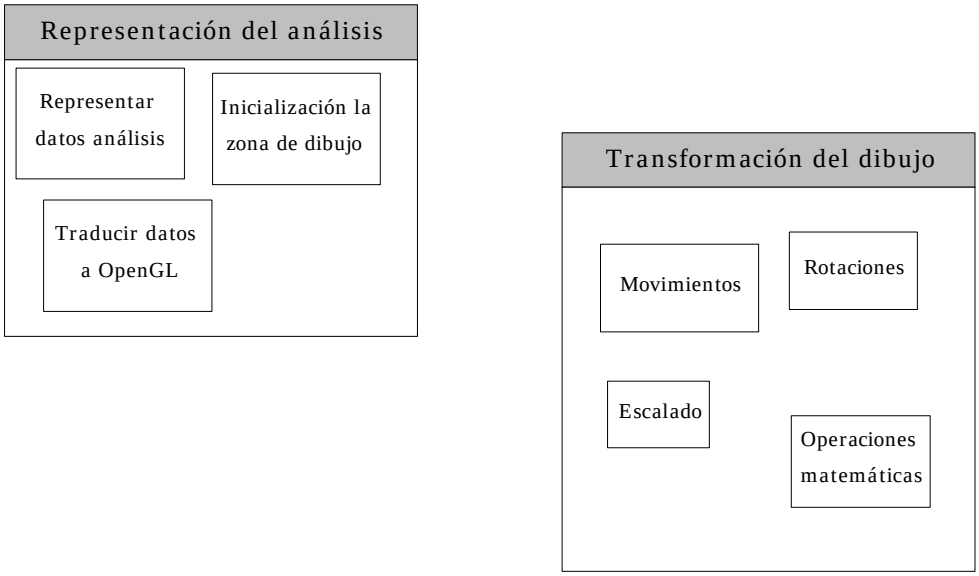


Figura 22 Operaciones para la representación

En la figura 23 se encuentra la propuesta de distribución de clases que implementan las operaciones necesarias.

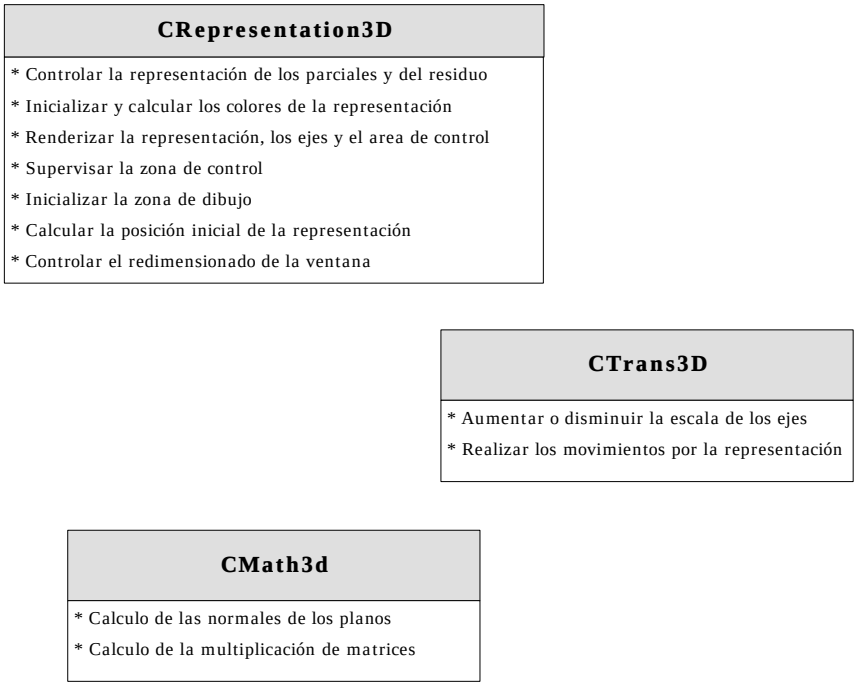


Figura 23 Clases de la aplicación

En los siguientes apartados se estudiaran detalladamente todas estas clases explicando las operaciones que implementan, una por una.

Comenzaremos por la clase base de representación y a continuación las clases que realizan operaciones con la representación.

En el siguiente capítulo se verá como se interrelacionan estas clases y como se conectan con la ventana.

3.3.2.1 La clase CRepresentation3D

En esta clase se desarrolla toda la representación de los datos desde un fichero SMS. En el cuadro 1 encontramos la definición de esta clase. Primero encontramos al constructor (CRepresentation3D) y el destructor (~CRepresentation3D). A continuación un apuntador a la clase *CTrans3D*. Siguen los diferentes procedimientos de la clase, que permiten inicializar la ventana para el dibujo (GLSetupRC), obtener los datos del eje (DataAxis), los de la representación (DataParRes) y los de los mandos de control (DataControl), representar todos los datos (GLRenderScene), controlar el redimensionado de la ventana (GLResize), actualizar si se cambia entre parciales (ChangeSinu) o residuo (ChangeResi), inicializar la tabla de colores (InitializeColors), calcular los colores de la representación (calcColor), inicializar la posición de la representación (InitialPosition), reinicializar la posición, controlar la selección dentro del área de control (Select), borrar las selecciones anteriores (ReleaseBuffer), aumentar o disminuir la resolución del residuo (MoreResolution y LessResolution) y mostrar texto en pantalla (DisplayText). A continuación viene una serie de variables internas necesarias para la representación.

```
class Crepresentation3D
{
public:
      Crepresentation3D();              // Constructor
      ~Crepresentation3D();            // Destructor
      Ctrans3D *pTrans3D;
      void GLSetupRC(void *pData,CSMSGenericSubtrack*pSMSGenericSubtrack, BOOL
Nuevo);
      void DataAxis(void);
      void DataParRes (CSMSGenericSubtrack *pSMSGenericSubtrack);
      void DataControl(void);
      void GLRenderScene();
      void GLResize(Glsizei h, Glsizei w);
```

```
        void ChangeSinu(void);
        void ChangeResi(void);
        void InitializeColors(void);
        void Restore(void);
        void calcColor(float corner, Glubyte out[3]);
        void InitialPosition(void);
        void Select(Cpoint point);
        void ReleaseBuffer();
        void MoreResolution(void);
        void LessResolution(void);
        void DisplayText(char cBuffer1[30], char cBuffer2[40],char cBuffer3[20],
char cBuffer4[20]);

private:
        Cmath3D *pMath3D;
        int Resolution;
        Gluint selectBuf[5];
        T_Size nFrames;
        float FrameTime;
        float Xmax;
        float Ymax;
        float Zmax;
        int listIndex;
        COLORREF pColor[201];
        double modelMatrix[16];
        double projMatrix[16];
        int viewport[4];
        float hView, vView;
        BOOL Sinu;
        BOOL Resi;
};
```

Cuadro 1

3.3.2.2 La clase CTrans3D

En esta clase se realizan todas las funciones que tengan relación con la transformación de la representación tridimensional. Primero está el constructor y el destructor. A continuación variables necesarias para los procedimientos y las funciones que realizan el control de las teclas que se han pulsado (Up, Down, Left, Right, Add, Subtract), aumentar la escala (ScalePlus) o disminuirla (ScaleMinus), avanzar o retroceder respecto a un eje (AxisForw y AxisBack), girar respecto a un eje (AxisRotationPlus y AxisRotationMinus) y realizar los mismos movimientos respecto de la pantalla (ScreenForw, ScreenBack, ScreenRotationPlus y ScreenRotationMinus).

```
class CTrans3D
{
public:
        CTrans3D();                // Constructor
        ~CTrans3D(){};             // Destructor
        CMath3D *pMath3D;
        Mov Move;
```

```
        float szAxis;
        float sxAxis;
        float rangy;
        float movematrix[16];
        float identmatrix[16];
        float ScaleX;
        float ScaleY;
        float ScaleZ;
        void Up(void);
        void Down(void);
        void Left(void);
        void Right(void);
        void Add(void);
        void Subtract(void);
        void ScalePlus(float& axis);
        void ScaleMinus(float& axis);
        void AxisForw (int direction);
        void AxisBack (int direction);
        void AxisRotationPlus(int dire1, int dire2, int dire3, int dire4);
        void AxisRotationMinus(int dire1, int dire2, int dire3, int dire4);
        void ScreenForw(void);
        void ScreenBack(void);
        void ScreenRotationPlus(void);
        void ScreenRotationMinus(void);
};
```

Cuadro 2

3.3.2.3 La clase CMath3D

Esta clase ha sido creada para poner en ella todas las funciones matemáticas. Esta clase hace un vector unitario (ReduceToUnit), calcula la normal de un plano determinado por tres puntos (calcNormal). Multiplica dos matrices 4x4 (MultMatrix4x4) (cuadro 3).

```
class CMath3D
{
public:
        CMath3D(){};
        ~CMath3D(){};
        void ReduceToUnit(float vector[3]);
        void calcNormal(float p0[3], float p1[3], float p2[3], float out[3]);
        void MultMatrix4x4(float operationmatrix[16], float movematrix[16]);
};
```

Cuadro 3

3.3.3 Descripción del funcionamiento interno

En este capítulo se describirá como se desarrolla el funcionamiento interno de la aplicación y la interrelación entre las clases y las funciones de las clases.

Cuando se ejecuta la aplicación, antes de crear la ventana, se describe el formato de pixel de la ventana. El formato de pixel describe como los gráficos que muestra la ventana están representados en memoria.

Una vez que se ha definido el formato del pixel, se crea el contexto de renderizado y se hace que sea el actual.

Se crea una ventana vacía y se espera hasta que se abre un fichero o se cierre la aplicación.

Cuando se abre un fichero, se obtienen los datos del análisis (SMSGenericSubtrack) y se pasan a la función que prepara el área de representación (GLSetupRC). Dentro de esta función se inicializa la tabla de colores (InitializaColors) y se determina la posición inicial de la representación (InitialPosition). El siguiente paso es pasar los datos del formato SMS a OpenGL (DataParRes). Dentro de esta función creamos las *display lists* necesarias para la representación y se calcula el color de cada dato (calcColor). Una vez obtenidos los datos de los parciales y del residuo, se obtiene las representaciones de los ejes (DataAxis) y de la zona de control (DataControl). Lo siguiente es representar todos estos datos (GLRenderScene).

Cada vez que se cambian los datos se ha volver a recalcular la posición de la representación y redibujarla (GLRenderScene). En esta función se divide el contexto de renderizado en dos partes (control y dibujo), se realizan todas las transformaciones sobre la representación, dibujamos los ejes y el área de control.

Cuando se redimensiona la ventana (GLResize), se vuelve a redibujar toda la representación (GLRenderScene).

Si se produce una entrada de teclado, la ventana llama a las funciones que calculan las transformaciones (CTrans3D) dependiendo de la tecla accionada y se redibuja la escena (GLRenderScene).

En cambio, si se produce una entrada de ratón, se halla la transformación seleccionada (Select) y se realiza la transformación correspondiente (CTrans3D).

Cuando se cierra la aplicación, se destruyen todos los objetos y el contexto de renderizado.

3.3.4 Uso de la aplicación

En este capítulo se explica como se utiliza la aplicación y su funcionamiento. Con esto se intenta que cualquier persona que lo lea sepa usarla con ayuda de las figuras en el apartado 3.2.3

Cuando se inicia la aplicación aparecerá una ventana en blanco. Únicamente se verán las opciones activadas de representación. En este momento se ha de cargar un fichero de datos sms. Esto se hace por medio del icono o por medio del menú Archivo. Aparecerá un dialogo donde se deberá elegir el archivo con extensión sms que se desee representar.

Una vez elegido, aparecerá en pantalla la representación del sonido y la zona de control. Las posibles acciones que se pueden realizar son las siguientes:

- Se puede elegir el eje por el que se va a desplazar o seguir la dirección de “pantalla”. La S, F, T y A corresponden a pantalla, eje de

frecuencia, eje del tiempo y eje de amplitud respectivamente. También se pueden realizar estos movimientos con las correspondientes teclas del teclado

- Nos podemos desplazar gracias a las flechas. Si se ha elegido un eje, nos desplazaremos en sentido positivo o negativo si damos arriba o abajo, respectivamente. Se puede girar a la izquierda o a la derecha si damos a las flechas izquierda o derecha respectivamente. Si se ha elegido la dirección "pantalla", entonces nos desplazaremos hacia delante o hacia detrás si son las flechas arriba o abajo. El giro se realiza del mismo modo. Con las flechas del teclado también se realizan estos movimientos.
- Se puede aumentar o disminuir el escalado del eje seleccionado con E+ o E- respectivamente. En el teclado corresponden a F1 y F2.
- Se puede aumentar o disminuir la resolución del residuo con R+ o R-
- Se puede mostrar u ocultar los parciales o el residuo con SIN y RES respectivamente. En el teclado se puede usar F9 o F10 respectivamente.
- Si se quiere volver a la posición inicial se deberá presionar la tecla F5.

Para salir de la aplicación se puede hacer con Alt+F4, por menú o por la barra de estado.

3.3.5 Pruebas

Durante el desarrollo de la aplicación se realizaron diferentes pruebas.

En una de las primeras pruebas se vio que cuando se realizaban los movimientos estos se realizaban de forma lenta y se intentaron diferentes estrategias para acelerar estos movimientos.

La primera prueba fue calcular el volumen de visualización y los puntos de corte de la representación con este volumen, para eliminar lo que no esté dentro antes de procesarlo. Una vez realizada esta prueba se vio que no aumentaba la velocidad por lo que se descartó.

Como el problema es a la hora de representar el residuo, se probó a disminuir la resolución del mismo. En vez de representar un polígono por cada *frame*, se variaría la resolución del residuo y se representaría un polígono cada 5, 10 o más *frames*. Cuando se realizó esta opción, se vio que la velocidad aumentaba considerablemente, por lo que se implementó y es lo que se denomina la resolución del residuo y que el usuario puede determinar.

Una vez realizada estas pruebas, se vio que el programa tenia un funcionamiento satisfactorio.

4 Conclusiones

En este trabajo se ha partido de la técnica de análisis, transformación y síntesis de sonido (SMS). Ha consistido en el desarrollo de una aplicación que permitiese ver el análisis en 3 dimensiones para una mejor visualización del mismo.

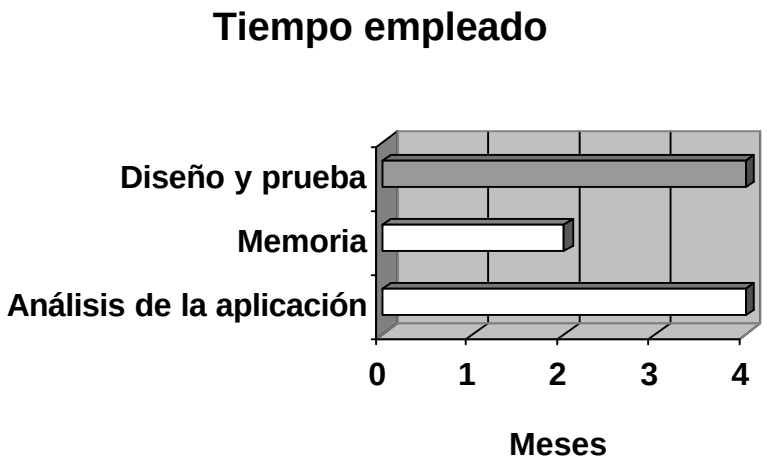
Esta aplicación se desarrolla en Visual C++ usando la API OpenGL gracias a su versatilidad y su facilidad a la hora de hacer diseños técnicos. Ya que nos encontrábamos en un entorno orientado a objetos, se ha modularizado esta representación con la incorporación de tres clases para aprovechar las características del Visual C++. Permite moverse por toda la representación y realizar transformaciones para una mejor visualización de los detalles del análisis.

El desarrollo de aplicaciones para Windows por medio de Visual C++ se realiza de una forma fácil. Solo hay que preocuparse de la aplicación y no de la interfase de ventanas, que está totalmente oculta al programador.

OpenGL permite realizar cualquier dibujo tridimensional en un sistema de ventanas. Tiene mucha flexibilidad a la hora de representar cualquier figura, ya que solo con especificar los vértices de esta, el dibujo saldrá en pantalla. Tiene multitud de efectos y se pueden realizar fácilmente múltiples transformaciones. Un inconveniente se presenta a la hora de representar objetos sencillos (p. e. las flechas del sistema de control de esta aplicación), ya que se han de definir totalmente. Tampoco es sencillo introducir texto dentro de la aplicación (p. e. el texto de la zona de control se ha hecho por medio de mapas de bits para poder distinguirlo del texto de los ejes).

El objetivo principal de este proyecto ha sido cumplido.

La estimación del coste total es de 10 meses repartidos de la siguiente forma:



4.1 Posibles ampliaciones

El trabajo realizado no está cerrado. Hay varias posibles ampliaciones. Por ejemplo, su incorporación dentro del programa original SMS. También se podrían incorporar posibilidades de transformación de sonidos por medio de la edición de la representación tridimensional. Se podría añadir la posibilidad de visualizar más de un sonido. Investigar la posibilidad de aumentar la velocidad de representación por medio de tarjetas aceleradoras 3D, ya que en el mercado existen tarjetas diseñadas para acelerar OpenGL. También se pueden realizar los movimientos en la representación por medio de un joystick.

5 Bibliografía

[1] DE POLI, G.; PICCIALLI, A.; ROADS, C., [eds.]. “Representations of Musical Signals”. Cambridge, Massachussetts: The MIT Press, 1991.

[2] ROADS, C. “The Computer Music Tutorial”. Cambridge, Massachussets: The MIT Press, 1996

[3] KRUGLINSKI, DAVID J. “Programación avanzada con Visual C++”. McGraw-Hill, 1996

[4] THOMPSON, TOM. “Must-See 3-D Engines” [en línea]. Byte Magazine. Junio 1996. [Consulta: 16 junio 1998].

[5] SEGAL, MARK; AKELEY, KURT. “The OpenGL Graphics Interface” [en línea]. California: Silicon Graphics Systems, 1994 [Consulta: 21 mayo 1998]

[6] NEIDER, JACKIE; DAVIS, TOM; WOO, MASON. “OpenGL Programming Guide. The Official Guide to Learning OpenGL, Release 1 (Red book)”. Addison-Wesley, 1993

[7] KENNETH E. HOFF III. “Faster 3D Game Graphics by Not Drawing What Is Not Seen” [en línea].

[8] SERRA, XAVIER. “File Format for SMS data” [en línea]. Music Technology Group, Audiovisual Institute, Pompeu Fabra University, Mayo 1997. [Consulta: 9 junio 1998]

[9] SERRA, XAVIER. “A system for sound analysis/transformation/synthesis based on a deterministic plus

stochastic decomposition". Stanford. CCRMA Departament of music, 1989. [Tesis doctoral]

[10] OURSLAND, N. ALAN. "Using OpenGL in Visual C++ Version 4.x" [en línea]. Interface Technologies, Inc., 1997.

[11] BONADA, JORDI. "Desenvolupament d'un entorn gràfic per a l'anàlisi, transformació i síntesi de sons mitjançant models espectrals". Proyecto de fin de carrera de la Escuela Técnica Superior de Ingeniería de Telecomunicaciones de Barcelona, 1997

[12] ALLEN, JONT B.; RABINER, LAWRENCE, R. "A unifed approach to short-time fourier analysis and synthesis". *Proc. IEEE*. Vol. 65 (1977).

[13] *comp.graphic.api.opengl* [en línea]. Usenet newsgroup.

Proyecto realizado por

Bernardo Macías Bécares

RESUMEN

Este proyecto desarrolla una aplicación que permite la visualización en tres dimensiones del análisis realizado por medio de la técnica SMS. Permite el desplazamiento y transformación de la representación. Se ha desarrollado en Visual C++ usando la API para la representación 3D OpenGL.

RESUM

Aquest projecte desenvolupa una aplicació que permet la visualització en tres dimensions de l'anàlisi realitzat mitjançan la tècnica SMS. Permet el desplaçament i la transformació de la representació. S'ha desenvolupat en Visual C++ fen servir la API per la representació 3D OpenGL.

SUMMARY

This project develops an application that allows the three dimensions visualisation of the analysis made by means of SMS technique. Allows the displacement and the transformation of the representation. Has been developed in Visual C++ using API for 3D representation OpenGL.