# EL TRATAMIENTO DEL SONIDO A TRAVÉS DE UN LENGUAJE DE PROGRAMACIÓN

[1] Jesús Bernal, [2] Pedro Gómez y [1] Jesús Bobadilla

[1]
Departamento de Informática Aplicada
Universidad Politécnica de Madrid
Ctra. De Valencia Km. 7, 28031 Madrid
Tfn: +34.913367860, Fax: +34.913367527
e-mail: jbernal@eui.upm.es, jbobi@eui.upm.es

[2]
Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n, Boadilla del Monte, 28660 Madrid
Tfn: +34.913367384, Fax: +34.913367412
e-mail: pedro@pino.datsi.fi.upm.es

## RESUMEN

En el presente artículo se pretende dar una visión práctica de cómo implementar programas que trabajen con ficheros de sonidos en el formato WAVE. Se realiza el estudio de aspectos tanto en el dominio del tiempo como en el dominio de la frecuencia.

En primer lugar se describe el formato de ficheros WAVE para su manejo directo desde programación. Se realiza una especial atención a la problemática de las variables a utilizar mediante el lenguaje de programación Turbo Pascal 7.0 de Borland.

Posteriormente se indica como abordar la realización de programas que sepan manejar formatos WAVE y calculen los parámetros típicos en el tiempo, como la magnitud, energía y cruces por cero.

Por último se indica como implementar el cálculo del espectro a través de la Transformada de Fourier, indicando como aplicar los distintos parámetros que intervienen en el mismo.



## ABSTRACT

This article presents a practical overview showing programs working with WAVE files. This files are computed in the time domain as well as in the frequency domain.

First, we describe the format of the WAVE files to use computer programs. Working on the WAVE formats we show the way to calculate typical time domain parameters.

Finally we show the way to get spectra representations using the Fourier Transform.

## 1. INTRODUCCIÓN

En este artículo pretendemos mostrar todos aquellos conocimientos que se consideran necesarios para el manejo y tratamiento de ficheros de sonido utilizando un lenguaje de programación (Turbo Pascal 7.0).

Para el procesamiento informático del sonido es necesario realizar su captura mediante ordenador y almacenarlo en memoria o disco. Existe una infinidad de tarjetas en el mercado que lo pueden realizar, aunque la más implantadas son las Sound Blaster de Creative. Si queremos información actual nos podemos conectar a su página web: *www.creaf.com*. Principalmente ofrece dos modelos: SB16 VALUE y SB AWE64.

El proceso básico para la captura de sonidos es el siguiente:

1. El sonido se trasmite por el aire hasta el micrófono, el cual transforma la señal acústica a niveles eléctricos, la calidad del micrófono es crucial en esta etapa.
2. La señal analógica pasa a través de un convertidor analógico/digital. Este resulta ser el elemento más importante, sus características establecen la frecuencia máxima a que pueden trabajar y el número de bits que se utilizan para cuantificar cada muestra.
3. Mediante un canal de DMA se copia cada muestra a la memoria principal del ordenador. Posteriormente se podrá copiar el total de las muestras a disco en algún formato estándar de sonido.

**Descriptores**: Sonido, WAV, Transformada de Fourier, Espectro, Espectrograma, Fonética Acústica.

## 2. FORMATO WAV

Vamos a describir el formato de un fichero WAV de Microsoft. El fichero está formado por dos bloques: la cabecera y las muestras digitalizadas del sonido. La cabecera tiene un tamaño de 44 bytes y

contiene el tipo y organización de las muestras. Su contenido es el siguiente:

- Campo 1: bytes 0..3. Contiene la palabra "RIFF" en código ASCII.
- Campo 2: bytes 4..7. Tamaño total del fichero en bytes menos 8 (no incluye los dos primeros campos).
- Campo 3: bytes 8..14. Contiene la palabra "WAVEfmt " en código ASCII (fijarse que hay un blanco detrás de la *t*).
- Campo 4: bytes 16..19. Formato, para PCM vale 16.
- Campo 5: bytes 20..21. Formato: para PCM vale 1.
- Campo 6: bytes 22..23. Se indica si es mono (1) o estéreo (2).
- Campo 7: bytes 24..27. Frecuencia de muestreo, puede valer: 11.025, 22.050 o 44.100.
- Campo 8: bytes 28..31. Indica el número de bytes por segundo que se debe intercambiar con la tarjeta de sonido para una grabación o reproducción.
- Campo 9: bytes 32..33. Número de bytes por captura, pueden ser 1, 2 o 4.
- Campo 10: bytes 34..35. Número de bits por muestra, pueden ser 8 0 16.
- Campo 11: bytes 36..39. Contienen la palabra "data" en código ASCII.
- Campo 12: bytes 40..43: Número total de bytes que ocupan las muestras.

Algunos campos contienen información redundante. El *campo 8* se puede calcular mediante: (*campo 7*)*(*campo 9*) o (*campo 6*)*(*campo 7*)*(*campo 10*)/8,... Si la captura es mono y de 16 bits por muestra el *campo 9* debe valer 2...

A continuación presentamos el código en Turbo Pascal 7.0 que define un *tipo* capaz de leer la cabecera de forma completa:

```
type
  tCab_WAV=record      {Campos que forman una cabecera WAV}
    Riff:array[0..3] of char; {'RIFF'}
    Tamanyo:longint;          {Tamaño del fichero menos 8}
    Wave:array[0..7] of char; {'WAVEfmt '}
    Format1:longint;  {Formato: para PCM vale 16}
    Format2:integer;  {Formato: para PCM vale 1}
    Mono:integer;     {Canales: mono:1, estereo:2}
```

```
    Frec:longint;      {Frecuencia de muestreo (Hz)}
    Bytes_s:longint;   {Nº de bytes por segundo}
    Bytes_c:integer;   {Nº de bytes por captura}
    Bits_m:integer;    {Nº de bits por muestra}
    Data:array[0..3] of char; {"data"}
    Bytes_cuestras_fich:longint;{nº de bytes de las muestras}
  end;
```

Cuando la grabación es mono todas las muestras se almacenan de forma consecutiva. Si es de 8 bits por muestra cada muestra ocupa un byte, si es de 16 bits por muestra cada muestra ocupa dos bytes (en Intel se almacena en primer lugar el byte menos significativo).

Cuando la grabación es en estéreo se almacena de forma alternativa una muestra de cada canal.

En la figura siguiente tenemos un cuadro resumen.


*Figura 1. Organización de datos en un fichero WAV.*

Cuando capturamos en 8 bits por muestra el tipo de dato leído es *byte*, es un tipo sin signo con un rango de 0 a 255; en este caso el silencio acústico se encuentra en el valor 128. Éste no es un tipo adecuado para operar matemáticamente, por ello debemos realizar un desplazamiento

para que el silencio se sitúe en el valor 0. Pasaremos al tipo *shortint* cuyo rango va desde -128 a 127. La operación será la siguiente:

```
muestra_8_aux {shortint} := muestra_8 {byte} - 128;
```

Al capturar en 16 bits por muestra el tipo leído es *integer*, se define con signo y su rango va desde -32.768 a 32.767, el silencio se sitúa en el valor 0, con lo se puede operar directamente.

A continuación presentamos los fuentes de un programa realizado en Turbo Pascal 7.0 que a partir del nombre de un fichero tipo WAV escribe en pantalla las características del mismo.

```
program Cabecera_WAV;
{Lee la cabecera de un fichero *.wav y la presenta por pantalla}

uses CRT;

type
  tCab_WAV=record     {Campos que forman una cabecera WAV}
    Riff:array[0..3] of char;   {'RIFF'}
    Tamanyo:longint;            {Tamaño del fichero - 8}
    Wave:array[0..7] of char;   {'WAVEfmt '}
    Format1:longint;            {PCM=16}
    Format2:integer;            {PCM=1}
    Mono:integer;               {Canales:mono(1), estereo (2)}
    Frec:longint;               {Frecuencia de muestreo en Hz}
    Bytes_s:longint;            {N° de bytes por seg.}
    Bytes_c:integer;            {N° de bytes por captura}
    Bits_m:integer;             {N° de bits por muestra}
    Data:array[0..3] of char;   {'data'}
    Bytes_muestras_fich:longint;{N° de bytes de las muestras}
  end;
var
    Fichero_WAV:file;   {Fichero sin formato}
    Cab_WAV:tCab_WAV;   {Almacena la cabecera del fichero WAV }
    Cadena:string[8];
{**************  PROGRAMA  PRINCIPAL  **********************}
begin
  clrscr;
  write('>> Nombre de fichero sin extension: ');
  readln(cadena);
  assign(Fichero_WAV,cadena+'.WAV');
  reset(Fichero_WAV,1);{Prepara el fichero para ser leido,
                  si no existe da un error de ejecucion}
  blockread(Fichero_WAV,Cab_WAV,sizeof(Cab_WAV)); {Lee la
          cabecera}

  with Cab_WAV do begin
    writeln('>>>>>>>> Cabecera del fichero: '+Cadena+'.WAV');
    writeln;
    Cadena:=Riff;
    writeln('Riff                    :',Cadena);
```

```
    writeln('Tamaño                    :',Tamanyo);
    Cadena:=Wave;
    writeln('Wave                      :',Cadena);
    writeln('Formato                   :',Format1);
    writeln('Formato                   :',Format2);
    writeln('Canales                   :',Mono);
    writeln('Frecuencia de muestreo :',Frec);
    writeln('N° de bytes por seg.    :',Bytes_s);
    writeln('N° de bytes por captura:',Bytes_c);
    writeln('N° bits por muestra     :',Bits_m);
    writeln('N° de bytes de muestras:',Bytes_muestras_fich);
    writeln;
    writeln('>>> Pulsa <intro> para salir');
    readln;
  end;
  close(Fichero_WAV)
end.
```

## 3. CÁLCULOS BÁSICOS SOBRE FICHEROS DE VOZ

El primer paso que debemos dar es capturar el sonido y almacenarlo en soporte informático. Esta tarea debe realizarse con cuidado para obtener una buena calidad de la grabación. Debemos procurar que la grabación aproveche al máximo los bits utilizados para la codificación de las muestras, pero sin llegar a saturarlo en ningún momento; si lo saturamos introducimos una distorsión al sonido original disminuyendo su calidad; si no aprovechamos el rango y sólo nos movemos con una pequeña parte de él, perdemos capacidad de cuantificación y por lo tanto disminuiremos la calidad de la grabación final. En la siguiente figura tenemos tres ejemplos de grabaciones, en la primera está saturado, en la segunda tiene un volumen muy bajo y en la tercera la grabación es adecuada.

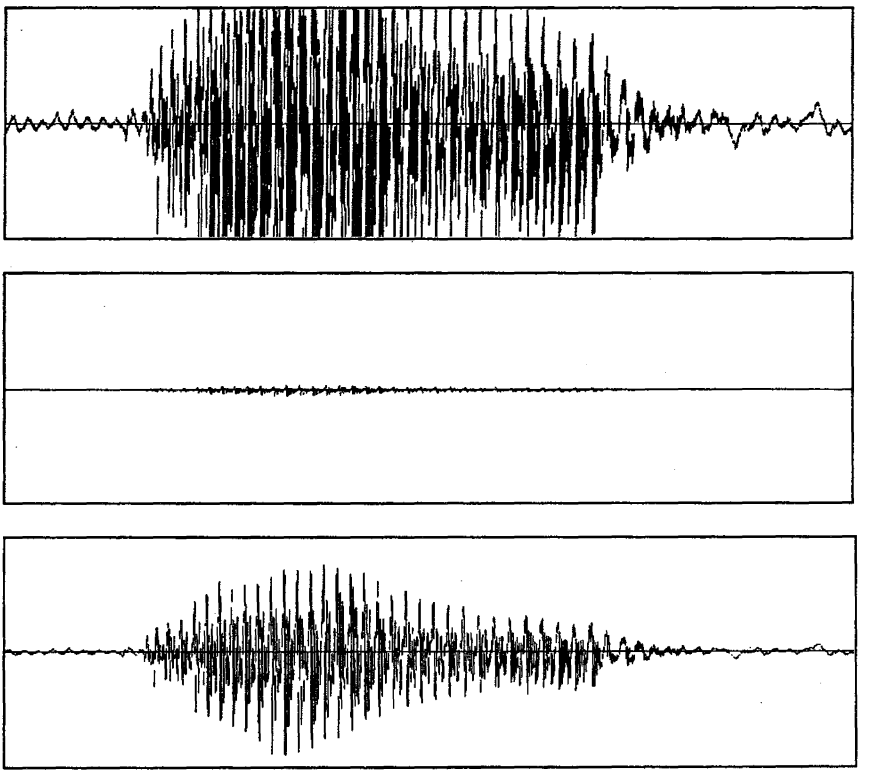


*Figura 2. Grabaciones de sonido*

Una vez que tenemos el sonido en soporte informático ya podemos operar con él. Vamos a explicar varios operaciones básicas como modificar el volumen, sumar varios sonidos, realizar un fade, cambiar la frecuencia de muestreo,...

Cuando el compilador de Turbo Pascal opera con dos variables el resultado temporal lo almacena en el tipo de mayor rango de las dos variables que intervienen. Si multiplicamos dos variables de tipo *integer* el resultado temporal lo almacena en un tipo *integer*, con lo que la operación será errónea al poderse producir desbordamiento. Debemos tener muy en cuenta los rangos de la variables para evitar resultados incorrectos o con poca exactitud.

- **Modificación del volumen**

Para modificar el volumen debemos multiplicar todas las muestras por un factor. Si el factor es mayor que uno aumentamos el volumen, si es menor que uno disminuimos el volumen y si es uno el volumen se queda sin modificar. Hay que recordar que si las muestran son de 8 bits debemos variar su rango para que el silencio se sitúe en el valor cero antes de cualquier operación, posteriormente, habrá que restaurar el silencio en la posición 128.

Un caso particular es cuando al multiplicar una muestra por un factor mayor que uno se produce desbordamiento. El compilador trunca los dígitos superiores, pero ésta solución no nos conviene; sería más recomendable dejar la muestra con el mayor valor absoluto posible. Esto lo debemos implementar mediante código específico.

Para un caso particular en que pretendamos aumentar el volumen al máximo, sin que se produzca saturación, debemos buscar la muestra de mayor valor absoluto y calcular el factor que debemos multiplicar para convertir esta muestra en el mayor valor que permita el rango de la muestra (127 en 8 bits y 32.767 en 16 bits). Posteriormente multiplicar todas las muestras por ese factor.

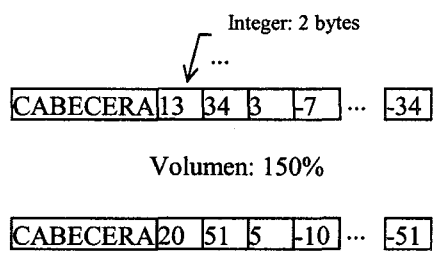



*Figura 3. Modificar el volumen.*

- **Suma de sonidos**

Para sumar varios ficheros de sonido debemos sumar sus muestras de forma ordenada, si queremos evitar que se desborde, antes de realizar la suma, se multiplica cada sonido por un factor menor que la unidad,

siendo la suma de los factores igual a la unidad. Con esta regla evitamos la posibilidad de que se produzca un desbordamiento, pero en general, el volumen del sonido final es muy bajo; deberemos reajustarlo para que ronde el 90% del rango permitido. Según aumentemos el factor de cada fichero haremos que influya más en el sonido final.

Al sumar una muestra de cada fichero hacemos que suenen en el mismo instante, es nuestra elección el decidir qué primera muestra de cada sonido vamos a sumar, pero el resto de muestras deben ir en el mismo orden.

En la gráfica siguiente presentamos la suma de tres ficheros de sonido: *a*, *b* y *c*; el fichero *a* y *b* queremos que suenen con la misma intensidad, el *c* con la mitad de intensidad. Primero se oirá el fichero a, pasadas 100 muestras se empezarán a oír el *b* y *c*. El fichero *a* tiene un total de 250 muestras, el *b* 500 muestras y el *c* 150 muestras. Todos ellos son mono, de 16 bits por muestra y poseen la misma frecuencia de muestreo.

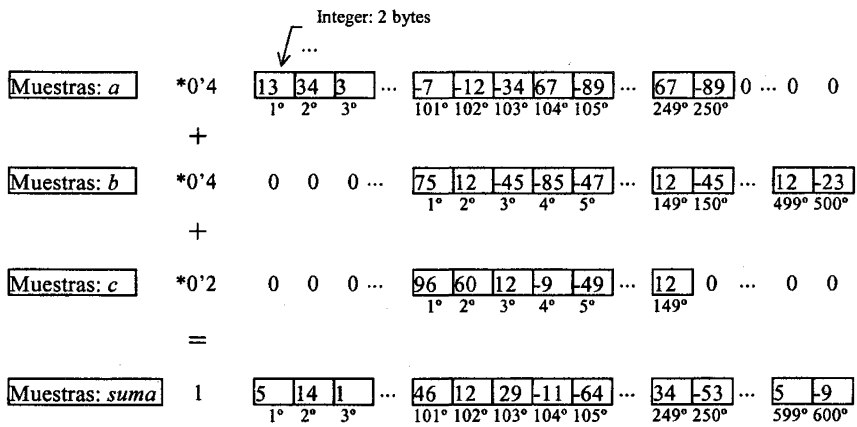



*Figura 4. Suma de sonidos.*

En el proceso de la suma se deben utilizar variables de tipo flotante para no perder los decimales, pero el resultado final se debe dejar en el tipo establecido por las muestras.

En la gráfica anterior podemos comprobar que cuando no existen muestras de un fichero en un instante de tiempo debemos tomar el silencio, en este caso ceros. Se supone que el formato de los tres ficheros es el mismo, sino fuera así, deberíamos igualarlos antes de realizar ninguna operación con ellos.

- **Modificar la frecuencia de muestreo**

El objetivo de esta operación es modificar la frecuencia de muestreo, distorsionado en la menor manera posible el sonido original del fichero.

Para disminuir la frecuencia de muestreo (de 44.100Hz a 22.050Hz o de 22.050Hz a 11.025Hz) se eliminará una muestra de cada dos, de forma ordenada.

Para aumentar la frecuencia de muestreo ( de 11.025Hz a 22.050Hz o de 22.050Hz a 44.100Hz) se añadirá una nueva muestra por cada una que exista. Para calcular su valor debemos realizar una interpolación. La forma más sencilla sería calculando la media aritmética entre la muestra anterior y posterior originales, aunque si se quiere más calidad se pueden utilizar interpolaciones de segundo orden, o tercer orden,... En la gráfica siguiente tenemos un ejemplo:

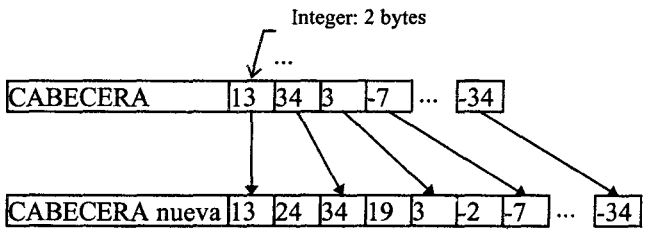



*Figura 5. Aumento de la frecuencia de muestreo.*

A continuación presentamos los fuentes de un programa que aumenta en un grado (11.025Hz → 22.050Hz o 22.050Hz → 44.100Hz) la frecuencia de muestreo de un fichero de sonido mono y de 16 bits por muestra.

```
program mas_frec;
```

```
{ Aumenta en un grado la frecuencia de muestreo para ficheros
mono y 16 bits}

uses CRT;

type
  tCab_WAV=record          {Campos que forman una cabecera WAV}
    Riff:array[0..3] of char;    {'RIFF'}
    Tamanyo:longint;             {Tamaño del fichero - 8}
    Wave:array[0..7] of char;    {'WAVEfmt '}
    Format1:longint;             {PCM=16}
    Format2:integer;             {PCM=1}
    Mono:integer;                {Canales:mono(1), estereo (2)}
    Frec:longint;                {Frecuencia de muestreo en Hz}
    Bytes_s:longint;             {N° de bytes por seg.}
    Bytes_c:integer;             {N° de bytes por captura}
    Bits_m:integer;              {N° de bits por muestra}
    Data:array[0..3] of char;    {'data'}
    Bytes_muestras_fich:longint;{N° de bytes de las muestras}
  end;
var
  Fich1,Fich2:file;        {Fichero sin formato}
  Cab1,Cab2:tCab_WAV;      {Almacena la cabecera del fichero WAV }
  Muestra_16, Anterior,Intermedia:integer;
  i:longint;
  Cadena:string[8];

{******************PROGRAMA PRINCIPAL************************}
begin
  clrscr;
  write('>> Nombre de fichero mono y 16 bits sin extension: ');
  readln(cadena);
  assign(Fich1,cadena+'.WAV');
  reset(Fich1,1);     {Prepara el fichero para ser leido,
                       si no existe da un error de ejecucion}
  blockread(Fich1,Cab1,sizeof(Cab1));  {Lee la cabecera}

  if((Cab1.Mono<>1)or(Cab1.Bits_m<>16)) then begin
    writeln('El fichero debe ser mono y de 16 bits');
    close(Fich1);
    exit;
  end;
  if(Cab1.Frec>22050)  then begin
    writeln('El fichero debe tener una Frec. menor a 44100Hz');
    close(Fich1);
    exit;
  end;

  assign(Fich2,'mas_frec.wav');
  rewrite(Fich2,1);
  Cab2:=Cab1;
  Cab2.Tamanyo:=Cab1.Tamanyo+Cab1.Bytes_muestras_fich;
  Cab2.Frec:=Cab1.Frec * 2;
  Cab2.Bytes_s:=Cab1.Bytes_s * 2;
  Cab2.Bytes_muestras_fich:=Cab1.Bytes_muestras_fich * 2;
  blockwrite(Fich2,Cab2,sizeof(Cab2));
```

```
  blockread(Fich1,Muestra_16,sizeof(Muestra_16));
  anterior:=Muestra_16;
  blockwrite(Fich2,Muestra_16,sizeof(Muestra_16));
  for i:=1 to (Cab1.Bytes_muestras_fich div 2)-1 do begin
    blockread(Fich1,Muestra_16,sizeof(Muestra_16));
    Intermedia:=round(Anterior*0.5+Muestra_16*0.5);
    blockwrite(Fich2,Intermedia,sizeof(Muestra_16));
    blockwrite(Fich2,Muestra_16,sizeof(Muestra_16));
    Anterior:=Muestra_16;
  end;
  blockwrite(Fich2,Muestra_16,sizeof(Muestra_16));

  close(Fich1);
  close(Fich2);
end.
```

## 4. CÁLCULOS EN EL DOMINIO DEL TIEMPO

Los cálculos básicos que se realizan en el dominio del tiempo son: magnitud, energía, media, cruces por cero y número de máximos. Para que estos parámetros nos den una información optima debemos conocer su evolución en el tiempo, y por lo tanto, calcularlos sobre ventanas temporales. Si las ventanas son muy pequeñas nos proporcionan excesivos detalles de la evolución, y si son muy grandes podemos perder detalles significativos. En la Figura 5 tenemos un esquema que representa la utilización de ventanas. Se manejan tres parámetros: tamaño de ventana (*Tv*), incremento de ventana (*Iv*) y solapamiento de ventana (*Sv*).

Existe una relación entre ellos: $Tv = Iv + Sv$. El tamaño y el incremento pueden ser cualquiera, no existe a priori ningún tipo de limitación. Puede ocurrir que el incremento sea mayor que el tamaño de la ventana, y por lo tanto, no existir solapamiento entre ventanas.

Hay que fijarse que al final del fichero pueden quedar algunas muestras de resto ya que no rellenan por completo una ventana, se deben rechazar ya que no darían un resultado comparable al resto de ventanas.

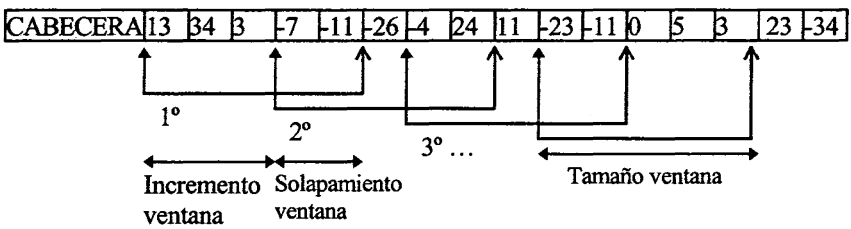


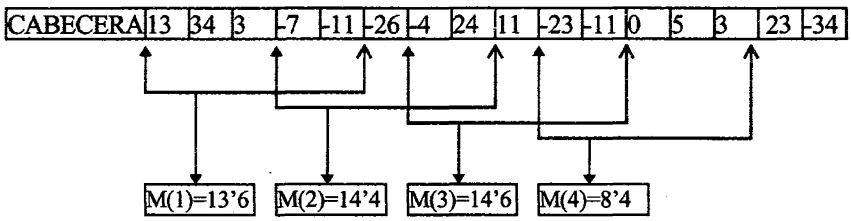

*Figura 6. Aplicación de ventanas*

En la figura se describe un ejemplo de la aplicación de ventanas. En este caso se ha utilizado un tamaño de 5 muestras con un incremento de 3 muestras. Al final de fichero ha quedado 2 muestras de resto que se desechan.

También se debe decidir el tipo de ventana a utilizar, podría ser de tipo rectangular, Hamming, Hanning...; en general, aplicaremos la ventana rectangular a los ejercicios resueltos.

Al realizar la operación o algoritmo sobre cada ventana nos da como resultado un valor, es decir, tendremos una función con *Nv* valores, siendo *Nv* el número de ventanas analizada. El rango de los valores calculados dependen principalmente de dos factores: tamaño de la ventana y tipo de operación. Para normalizar respecto a la ventana, dividiremos el resultado por el tamaño de la ventana.

Como ejemplo vamos a realizar el cálculo de la magnitud del fichero presentado (en el apartado siguiente se describe con detalle este cálculo).




Normalmente serán muchos resultados y con rangos variados. Si lo presentamos en pantalla de forma numérica, el usuario tendría dificultad para su interpretación, por ello es más aconsejable visualizarlo en modo gráfico. Para dibujarlo en pantalla debemos realizar primero una normalización de valores, ya que tenemos reservado un espacio limitado de puntos.

Supongamos que tengamos una pantalla de *MaxX* puntos de ancho por *MaxY* puntos de alto, numerados desde el punto (0,0) hasta el punto (*MaxX*-1,*MaxY*-1), el punto (0,0) está en la posición superior e izquierda. Queremos representar la gráfica resultante en una ventana con un alto de *H* puntos y un ancho de *B* puntos, situando el punto (0,0) de nuestro resultado en la coordenada (*OrigenX*, *OrigenY*) de pantalla.

Para normalizar los resultados debemos convertir el valor máximo de la función calculada en el máximo permitido (*H*), manteniendo la proporción con el resto de valores. Si $M_{max}$ fuera el valor máximo entre todos los valores de *M(i)* y *H* la altura de la ventana se normalizaría mediante: $M(i) := parte\_entera\left(H\frac{M(i)}{M_{max}}\right), \forall i$.

Normalizando el ejemplo anterior para una altura de 10 puntos nos quedaría:

$$M(1) := parte\_entera\left(10\frac{136}{146}\right)=9,\quad M(2) := parte\_entera\left(10\frac{144}{146}\right)=10,\quad M(3):=10,\quad M(4):=6$$

En la siguiente figura tenemos un gráfico del resultado.

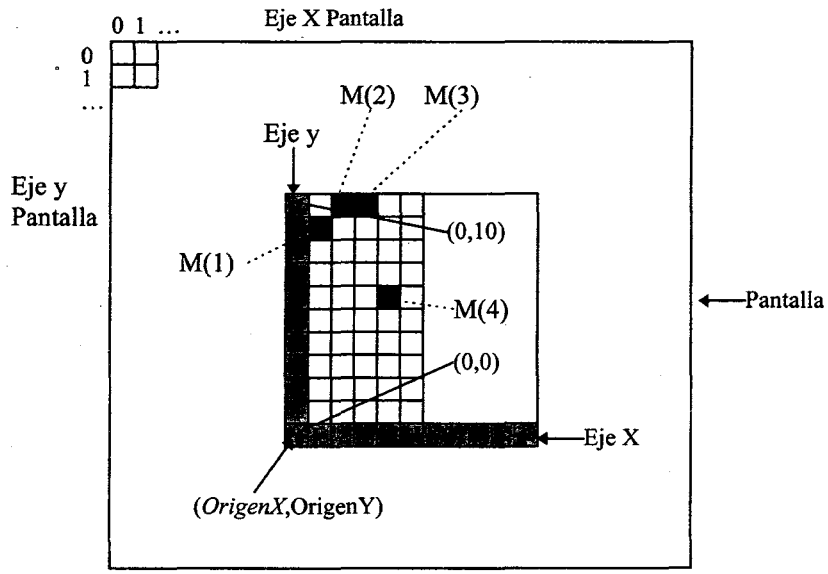



*Figura 7. Representación gráfica de la función*

La secuencia de pixeles que se deben iluminar son: $(EjeX+i, EjeY-M(i))\forall i$.

En el ajuste del eje Y tenemos dos opciones: o normalizar al ancho permitido realizando una media de los resultados o presentar una parte de los cálculos realizados.

- **Cálculo de la magnitud**

La magnitud se define como el sumatorio del valor absoluto de las muestras que se encuentran en una ventana; se divide por el número de muestras para normalizar los resultados: $M(i)=\frac{1}{Tv}\sum_{k=0}^{Tv-1}|m(k)|$, siendo $M(i)$ la magnitud de la ventana $i$, $Tv$ el tamaño de la ventana y $m(k)$ la muestra de sonido capturada.

A continuación presentamos un programa que calcula la magnitud de un fichero mono de 8 bits por muestra y presenta una gráfica normalizada de los resultados.

```
program magnitud_WAV;
 {Presenta de forma grafica la magnitud}

uses crt, graph;

const
  Ox=20;       {Posicion en pantalla del origen en el eje X}
  Oy=230;      {Posicion en pantalla del origen en el eje Y}
  H=200;       {Se tomara solo parte positiva}

type
  tCab_WAV=record          {Campos que forman una cabecera WAV}
    Riff:array[0..3] of char;     {'RIFF'}
    Tamanyo:longint;              {Tamaño del fichero - 8}
    Wave:array[0..7] of char;     {'WAVEfmt '}
    Formato:longint;              {16}
    Pcm:integer;                  {PCM (1)}
    Mono:integer;                 {Canales:mono(1), estereo (2)}
    Frec:longint;                 {Frecuencia de muestreo en Hz}
    Bytes_s:longint;              {N° de bytes por seg.}
    Bytes_c:integer;              {N° de bytes por captura}
    Bits_m:integer;               {N° de bits por muestra}
    Data:array[0..3] of char;     {'data'}
    Bytes_muestras_fich:longint;{N° de bytes de las muestras}
  end;

var
  Fich:file;
  Leido:integer;
  Nombre:string[8];

  Gd, Gm: Integer;       { Variables para uso del modo grafico}

  Cab:tCab_WAV;
  Muestra:byte;       {Fichero WAV de 8 bits por muestra}
  Ventana:array [0..1000] of byte;
  Incremento_ventana,Tamanyo_ventana:integer;

  i,m:integer;           {Indices}
  Magnitud, MagnitudMax:real;
  Magnitudes:array [1..600] of real;

begin

  clrscr;
  write('>> Nombre del fichero sin extension: ');
  readln(Nombre);
  assign(Fich,Nombre+'.WAV');
  reset(Fich,1); {Prepara el fichero para ser leido,
                  si no existe da un error de ejecucion}
  blockread(Fich,Cab,sizeof(Cab)); {Lee la cabecera}
```

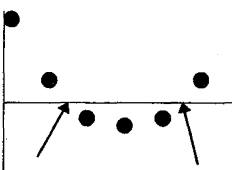
```
  if (Cab.Mono<>1) or (Cab.bits_m<>8) then begin
     writeln('El fichero debe se mono y de 8 bits/muestra');
     exit;
  end;

  write('>> Indica el incremento y tamaño de las ventanas
        (1..999): ');
  readln(Incremento_ventana, Tamanyo_ventana);

    {Se activa el modo grafico}
  Gd := Detect;
  InitGraph(Gd, Gm, '');
  if GraphResult <> grOk then begin
    writeln('Error grafico');
    exit;
  end;

  {Se establece el eje de coordenadas de la media}
  setcolor(4);
  line(Ox,Oy,Ox,Oy-H);
  line (Ox,Oy,Ox+600,Oy);{Se toman 600 pixel de ancho de
       pantalla}
  for i:=1 to 600 do begin {Se dibujan las divisiones del eje}
    if i mod 5 = 0 then line(i+Ox,Oy-1,i+Ox,Oy+1);
    if i mod 25 = 0 then line(i+Ox,Oy-5,i+Ox,Oy+5);
  end;
  for i:=0 to H do begin  {Se dibujan las divisiones del eje}
    if i mod 5 = 0 then line(Ox-1,i+Oy,Ox+1,i+Oy);
    if i mod 25 = 0 then line(Ox-5,i+Oy,Ox+5,i+Oy);
  end;

  {Se dibuja las medias de las ventanas}
  i:=0;
  seek(fich,44);
  for m:=1 to 600 do begin {se limita a 600 puntos de pantalla}
    blockread(Fich,Ventana,tamanyo_ventana,Leido);
    if Leido < tamanyo_ventana then break; {Fin de fichero}
    seek(Fich,filepos(Fich)-
         tamanyo_ventana+Incremento_ventana*2);
    Magnitud:=0;
    for i:=0 to Tamanyo_ventana-1 do
      Magnitud := Magnitud + abs(Ventana[i]-
                  128)/Tamanyo_ventana;
      Magnitudes[m]:=Magnitud;
      if Magnitud>MagnitudMax then MagnitudMax:=Magnitud;
  end;
  for m:=1 to 600 do
    putpixel(m+Ox,Oy-round(H*Magnitudes[m]/MagnitudMax),15);

  repeat until keypressed;

  closegraph;
end.
```

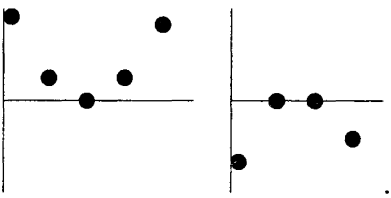
- **Cálculo de la energía**

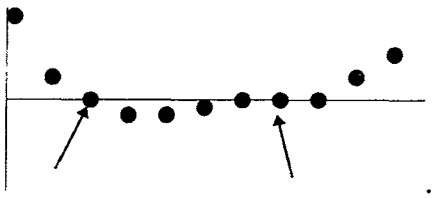
La energía se define como el sumatorio de las muestras al cuadrado; se divide por el número de muestras para normalizar los resultados: $E(i) = \frac{1}{Tv}\sum_{k=0}^{Tv-1} m^2(k)$, siendo $E(i)$ la energía de la ventana $i$, $Tv$ el tamaño de la ventana y $m(k)$ la muestra de sonido capturada.

- **Cálculo de los cruces por cero**

Los cruces por cero nos indican el número de veces que la señal atraviesa el nivel cero, en cualquiera de los dos sentidos. Debido a que trabajamos con valores discretos nos podemos encontrar con los siguientes casos:

1.  En esta ocasión existen dos cruces por cero sin que ninguna muestra valga cero. Se Debe mirar si se produce cambio de signo en dos muestras consecutivas

2.  . Aquí nos encontramos con dos casos en que no se puede considerar un cruce por cero, ya que la señal no atraviesa el nivel cero.

3.  . Aquí se debe considerar solamente dos cruces por cero, aunque existan varias muestras consecutivas con el valor cero.

## 5. CÁLCULOS EN EL DOMINIO DE LA FRECUENCIA: LA TRANSFORMADA DE FOURIER

En todos los cálculos realizados en el dominio del tiempo, cada vez que se realizaban en una ventana nos daban como resultado un único valor. En el cálculo de la Transformada de Fourier (TF) no ocurre lo mismo; cuando se aplica sobre una ventana nos devuelve un conjunto de resultados. Por ello, para poder representar una evolución en el tiempo necesitamos utilizar tres dimensiones; en los programas desarrollados en el presente artículo se utiliza el color como tercera dimensión debido a que el espectro resulta más claro.

A continuación presentamos los fuentes de un programa que calcula y visualiza la TF de la primera ventana de un fichero WAV.

En el procedimiento "*Leer_ventana*" se incluye los cálculos del tipo de ventana. Hay que fijarse que sólo se debe aplicar al conjunto de muestras, sin incluir los ceros que finalmente se añaden a la ventana.

En el programa principal se realiza una normalización de los resultados para que ocupen la totalidad de los puntos de pantalla reservados en el eje Y (representa la amplitud ). Para presentar el conjunto de frecuencias se utiliza un punto por cada una, con un máximo de 600.

```
program Transformada_Fourier;

  {Obtiene la Transformada de Fourier de la primera ventana de un
fichero *.WAV. Trabaja sobre fichero de 11025Hz, 16 bits/muestra
y mono. Estudia los aspectos de tamaño de ventana, añadir ceros y
tipo ventana}
```

```
uses crt, graph;

const
  Tamanyo_v_t=512; {Tamaño de la ventana}
  Tamanyo_v_m=512; {Tamaño de las muestras, sin incluir los 0}
  Tamanyo_v_f=Tamanyo_v_t div 2;
     {La segunda mitad del espectro es una copia }
type
  tCab_WAV=record          {Campos que forman una cabecera WAV}
    Riff:array[0..3] of char;     {'RIFF'}
    Tamanyo:longint;              {Tamaño del fichero - 8}
    Wave:array[0..7] of char;     {'WAVEfmt '}
    Formato:longint;              {16}
    Pcm:integer;                  {PCM (1)}
    Mono:integer;                 {Canales:mono(1), estereo (2)}
    Frec:longint;                 {Frecuencia de muestreo en Hz}
    Bytes_s:longint;              {N° de bytes por seg.}
    Bytes_m:integer;              {N° de bytes por muestra}
    Bits_m:integer;               {N° de bits por muestra}
    Data:array[0..3] of char;     {'data'}
    Bytes_fich:longint;           {N° de bytes de las muestras }
  end;
  tVentana=array [0..Tamanyo_v_t-1] of real;

var
  Fich:file;
  Nombre:string[8];
  Cab:tCab_WAV;
  i:integer;
  Gd, Gm: Integer;
  max:real;
  Ventana,TF:tVentana;

{************************************************************}
procedure Leer_ventana(var Ventana:tVentana);
  {Lee una ventana de muestras del fichero *.wav especificado,
    posteriormente le aplica el tipo de ventana seleccionado}
var
  V_rect,V_hamming,V_hanning:real;
  Muestra:integer;
begin
  for i:=0 to Tamanyo_v_m-1 do begin
     blockread(Fich,Muestra,sizeof(Muestra));
     V_rect:= 1;
     V_hamming:=0.54-0.46*cos(2*pi*i/(Tamanyo_v_m-1));
     V_hanning:=(1-cos(2*pi*i/(Tamanyo_v_m-1)))/2;
     Ventana[i]:=V_rect*Muestra;
  end;
  for i:=Tamanyo_v_m to Tamanyo_v_t do
    Ventana[i]:=0;
end;
{************************************************************}
procedure T_fourier(var Ventana,TF:tVentana);
```

```
    {Obtiene el modulo de la Transformada de Fourier de una
ventana temporal}
var
  n,k:word;
  p_real,p_imag,modulo:real;
begin
  for n:=0 to Tamanyo_v_f-1 do begin
    p_real:=0;
    p_imag:=0;
    for k:=0 to Tamanyo_v_m-1 do begin
      p_real := p_real + Ventana[k]*cos(2*pi*n*k/Tamanyo_v_t);
      p_imag := p_imag - Ventana[k]*sin(2*pi*n*k/Tamanyo_v_t);
    end;
    modulo := sqrt(sqr(p_real)+sqr(p_imag));
    TF[n]:=modulo;
  end;
end;

{************************ PRINCIPAL ************************}
begin

  clrscr;
  write('>> Nombre del fichero sin extension: ');
  readln(Nombre);
  assign(Fich,Nombre+'.WAV');
  reset(Fich,1);          {Prepara el fichero para ser leido}

  blockread(Fich,Cab,sizeof(Cab));
  if (Cab.Mono<>1) or (Cab.bits_m<>16) or (Cab.Frec<>11025)then
  begin
    writeln('El fichero debe se mono, 16 bits/m. y 11025 Hz');
    exit;
  end;

 {Se activa el modo grafico}
  Gd := Detect;
  InitGraph(Gd, Gm, '');
  if GraphResult <> grOk then begin
    writeln('Error grafico');
    exit;
  end;

 {Se establece el eje de coordenadas de las muestras}
  setcolor(10);
  line(20,10,20,210);
  line (20,110,620,110);
  for i:=0 to 600 do begin
    if i mod 5 = 0 then line(i+20,109,i+20,111);
    if i mod 25 = 0 then line(i+20,105,i+20,115);
  end;
  for i:=0 to 200 do begin
    if i mod 5 = 0 then line(19,i+10,21,i+10);
    if i mod 25 = 0 then line(15,i+10,25,i+10);
  end;
 {Se establece el eje de coordenadas de la T. de Fourier}
  setcolor(12);
```

```
  line(20,230,20,430);
  line (20,430,620,430);
  for i:=0 to 600 do begin
    if i mod 10 = 0 then line(i+20,429,i+20,431);
    if i mod 50 = 0 then line(i+20,425,i+20,435);
  end;
  for i:=0 to 200 do begin
    if i mod 5 = 0 then line(19,i+230,21,i+230);
    if i mod 25 = 0 then line(15,i+230,25,i+230);
  end;

  Leer_ventana(Ventana);

 {Se dibuja la ventana de muestras, maximo 600}
  for i:=0 to Tamanyo_v_t do begin
    putpixel(i+20,round(Ventana[i]/320)+110,15);
    if i>600 then break;
  end;

  T_fourier(Ventana,TF);

 {Se normaliza la TF para que ocupe todo el rango }
  max:=1;
  for i:=0 to Tamanyo_v_f-1 do
    if TF[i]>max then max:=TF[i];
 {Se dibuja la TF con una maximo de 600 puntos}
  setcolor(15);
  for i:=0 to Tamanyo_v_f-1 do begin
    bar(2*i+20,430-round(TF[i]/max*200),2*i+20+1,430);
    if i>600 then break;
  end;

  repeat until keypressed;

  closegraph;

end.
```

En las dos figuras posteriores tenemos los resultados de la ejecución del programa con dos ficheros diferentes: uno con un armónico de 3000Hz y otro con un armónico de 200Hz. En el eje X, que corresponden a las frecuencias, se ha establecido una división pequeña por cada cinco puntos y una grande por cada 25. Como la frecuencia de muestreo es de 11025Hz y el tamaño de la ventana es de 512, cada punto equivale a un incremento de $\frac{F}{N} = \frac{11025}{512} \approx 21'5Hz$.

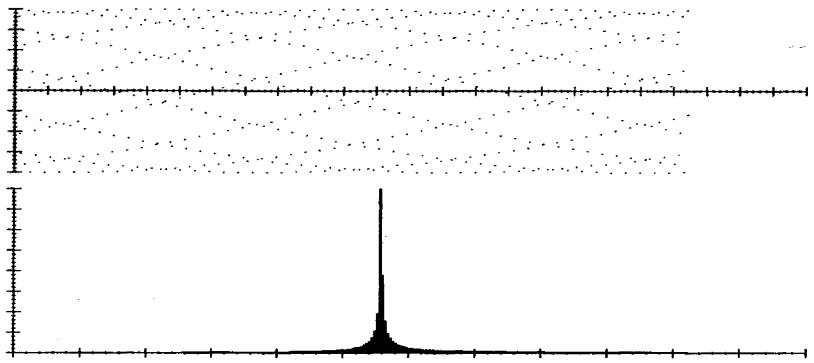


*Figura 8. Un armónico de 3000Hz.*

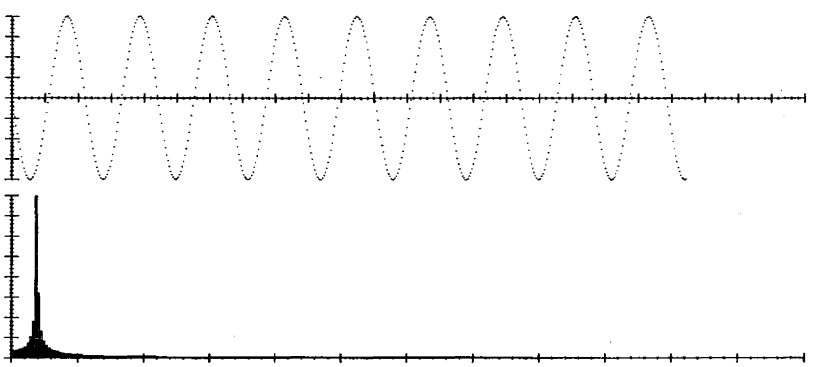


*Figura 9. Un armónico de 200Hz.*

Para poder representar la evolución de un conjunto de ventanas codificamos la amplitud de las frecuencias en colores, representando una ventana como una línea de puntos en el eje Y. La evolución temporal queda reflejada en el eje X.

Al estudiar las amplitudes alcanzadas de las distintas frecuencias, encontramos diferencias muy grandes. Mediante una codificación lineal, a los valores pequeños sólo se les asigna el último color de la escala. Por ello, se aplica una escala logarítmica en la codificación. Al aplicar una

escala logarítmica se aumenta de forma indeseada el ruido de fondo. Para evitarlo se establece un umbral mínimo y se elimina toda señal que no lo supere. Por último, para una distribución completa de la escala de colores se establece un umbral máximo y se asignan los colores en el rango establecido.

La figura siguiente es un esquema de los pasos seguidos.

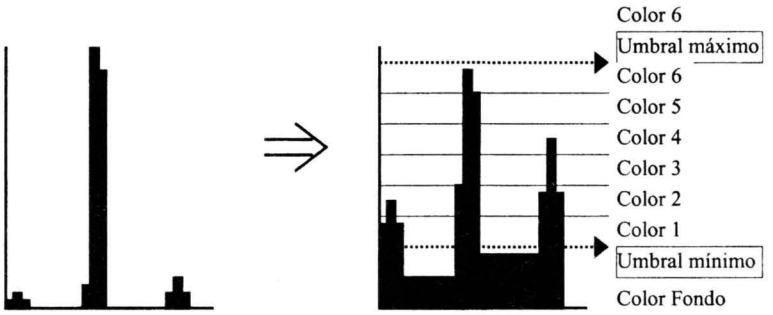



*Figura 10. A la izquierda tenemos una ventana después de calcular la TF. A la derecha se ha aplicado una escala logarítmica y se han establecidos los umbrales.*

A continuación presentamos el programa que calcula el espectro de un fichero WAV utilizando la Transformada Rápida de Fourier.

```
program Transformada_Fourier;
 {Obtiene la Transformada de Fourier de un fichero *.WAV.
  Trabaja sobre fichero de 11025Hz, 16 bits/muestra y mono.
  Se dibuja la TF en gama de colores}

uses crt, graph;

const
  Tamanyo_v_t=256; {Tamaño de la ventana}
  Tamanyo_v_m=100; {Tamaño de las muestras, sin incluir los 0}
  Tamanyo_v_f=Tamanyo_v_t div 2;
     {La segunda mitad del espectro es una copia }
  Incremento_v=40;
  Max_db=6.3; {Limite superior de representacion grafica}
  Min_db=3.5; {Limite inferior}

type
  tCab_WAV=record          {Campos que forman una cabecera WAV}
    Riff:array[0..3] of char;     {'RIFF'}
    Tamanyo:longint;              {Tamaño del fichero - 8}
```

```
    Wave:array[0..7] of char;      {'WAVEfmt '}
    Formato:longint;               {16}
    Pcm:integer;                   {PCM (1)}
    Mono:integer;                  {Canales:mono(1), estereo (2)}
    Frec:longint;                  {Frecuencia de muestreo en Hz}
    Bytes_s:longint;               {N° de bytes por seg.}
    Bytes_m:integer;               {N° de bytes por muestra}
    Bits_m:integer;                {N° de bits por muestra}
    Data:array[0..3] of char;      {'data'}
    Bytes_fich:longint;            {N° de bytes de las muestras }
  end;
  tVentana=array [0..Tamanyo_v_t-1] of real;

var
  Fich:file;
  Nombre:string[8];
  Cab:tCab_WAV;
  i,j:integer;          {Indices generales}
  Gd, Gm: Integer;      {Variable para activar el modo grafico}
  Ventana,TF:tVentana;
  Fin_m:boolean;        {Indica el final del fichero}
  Gamma:integer;        {Constante que se utiliza en la FFT}
  Coseno,Seno,FFT_imag,FFT_real:tVentana;

{************************************************************}
procedure Leer_ventana(var Ventana:tVentana; var Fin_m:boolean);
 {Lee una ventana de muestras del fichero *.wav especificado,
  posteriormente le aplica el tipo de ventana seleccionado por
  programa}
var
  V_rect,V_hamming,V_hanning:real;
  Muestra:integer;
  i,Leido:word;
begin
  for i:=0 to Tamanyo_v_m-1 do begin
    if eof(Fich) then begin
      Fin_m:=true;
      exit;
    end;
    blockread(Fich,Muestra,sizeof(Muestra));

    V_rect:= 1;
    V_hamming:=0.54-0.46*cos(2*pi*i/(Tamanyo_v_m-1));
    V_hanning:=(1-cos(2*pi*i/(Tamanyo_v_m-1)))/2;

    Ventana[i]:=V_hamming*Muestra;
  end;
  for i:=Tamanyo_v_m to Tamanyo_v_t do
    Ventana[i]:=0;
end;

{************************************************************}
procedure T_fourier(var Ventana,TF:tVentana);
  {Obtiene el modulo de la Transformada de Fourier de una ventana
temporal}
var
```

```
  n,k:word;
  P_real,P_imag,Modulo:real;
begin
  for n:=0 to Tamanyo_v_f-1 do begin
    P_real:=0;
    P_imag:=0;
    for k:=0 to Tamanyo_v_m-1 do begin
      P_real := P_real + Ventana[k]*cos(2*pi*n*k/Tamanyo_v_t);
      P_imag := P_imag - Ventana[k]*sin(2*pi*n*k/Tamanyo_v_t);
    end;
    Modulo := sqrt(sqr(p_real)+sqr(p_imag));
    TF[n]:=Modulo;
  end;
end;

{************************************************************}
                {transformada rapida de fourier}

procedure Ini_fft(var Coseno,Seno:tVentana);
 {Inicializa la tabla de senos y de cosenos para obtener mayor
  velocidad en los accesos dentro de la transformada rapida de
  fourier}
var
  i:word;
begin
 Gamma:=round( (ln(Tamanyo_v_t)/ln(2)) );{obtiene gamma}

 for i:= 0 to Tamanyo_v_t-1 do
  begin
    Coseno[i]:=cos(2*pi*i/Tamanyo_v_t);{Obtiene los cosenos}
    Seno[i]:=sin(2*pi*i/Tamanyo_v_t)   {Obtiene los senos}
  end
end;

procedure Inverso(var Result:integer;Numero:integer);
    {Obtiene el numero invertido}
var
  i:word;
  Num_aux,Num_aux2:integer;
begin
  Result:=0;
  Num_aux:=Numero;
  for i:=1 to Gamma do begin
    Num_aux2:=Num_aux div 2;
    Result:=Result*2+(Num_aux - Num_aux2*2);
    Num_aux:=Num_aux2
  end
end;

procedure T_rapida_fourier(Ventana:tVentana;var FFT:tVentana);
 {Procedimiento para calcular la T. rapida de fourier}
var
 Rshift,        {Desplazamiento a la derecha}
 C,             {Nivel de la fft}
 K,             {k de la transformada}
```

```
  I,              {Cuenta de nodos duales}
  M,              {k desplazado Rshift veces}
  P,              {m invertido}
  Separacion,     {Separacion entre nodos duales}
  Ele:integer;    {almacena la potencia de un numero}
  Modulo,         {almacena el modulo de un numero}
  Aux,            {Auxiliar para intercambio de valores de trans}
  P_real,          {almacena el valor de la parte real}
  P_imag:real;     {almacena el valor de la parte imaginaria}

   function Elevado(a,b:integer):integer;
    {Obtiene el resultado de elevar el numero a a su b potencia}
   var
     Aux,i:integer;
   begin
     Elevado:=0;
     Aux:=1;
     for i:=1 to b do Aux:=Aux*a;
     Elevado:=round(Aux)
   end;

begin
   for k:= 0 to Tamanyo_v_t-1 do
     begin
       FFT_real[k]:=ventana[k];
       FFT_imag[k]:=0.0
     end;

   Separacion:=Tamanyo_v_t div 2;
   Rshift:=Gamma-1;

   for C:=1 to Gamma do
     begin
       K:=0;
       I:=1;
       while K<Tamanyo_v_t do
        begin
          Ele:=Elevado(2,Rshift);
          M:=K div Ele;
          Inverso(P,M);
          P_real:=Coseno[P]*FFT_real[K+Separacion]+
                  Seno[P]*FFT_imag[K+Separacion];
          P_imag:=Coseno[P]*FFT_imag[K+Separacion]-
                  Seno[P]*FFT_real[K+Separacion];


          FFT_real[K+Separacion]:=FFT_real[K]-P_real;
          FFT_imag[K+Separacion]:=FFT_imag[K]-P_imag;
          FFT_real[K]:=FFT_real[K]+P_real;
          FFT_imag[K]:=FFT_imag[K]+P_imag;

          if I=Separacion then
           begin
             K:=K+Separacion;
             I:=0;
           end;
```

```
        K:=K+1;
        I:=I+1;
      end;

    Separacion:=Separacion div 2;
    Rshift:=Rshift-1
  end;

  for K:=0 to Tamanyo_v_t-1 do
   begin
    Modulo:=sqrt(sqr(FFT_real[k])+sqr(FFT_imag[k]));
    FFT[k]:=Modulo;
   end;

  for K:=0 to Tamanyo_v_t-1 do
  begin
    Inverso(I,K);

    if K>I then
     begin
       Aux:=FFT[K];
       FFT[K]:=FFT[I];
       FFT[I]:=Aux
     end

  end

end;

{************************************************************}
procedure Normalizar(var TF:tVentana);
  {Pasa los valores de la TF a escala logaritmica (para mejora
   su visualizacion, ademas elimina hasta un nivel de ruido y
   limita la señala un maximo de señal}
var
  Modulo:real;
  i:word;
begin
  for i:=0 to Tamanyo_v_f-1 do begin
    Modulo:=TF[i];
    if Modulo<0.00001 then Modulo:=0.00001;
    Modulo := ln(Modulo)/ln(10);
    Modulo := Modulo-Min_db;
    if Modulo<0 then Modulo:=0;
    if Modulo>(Max_db-Min_db) then Modulo:=Max_db-Min_db;
    TF[i]:=Modulo;
  end;
end;

{************************************************************}
function Color(Valor:real):word;
  {Cambia un rango de valores reales a una escala de colores}
begin
  Color:=trunc(7*Valor/(Max_db-Min_db)+0.999)
```

```
end;

{************************ PRINCIPAL ************************}
begin
  {Se abre el fichero *.wav}
  clrscr;
  write('>> Nombre del fichero sin extension: ');
  readln(Nombre);
  assign(Fich,Nombre+'.WAV');
  reset(Fich,1);

  blockread(Fich,Cab,sizeof(Cab));
  if (Cab.Mono<>1) or (Cab.bits_m<>16) or (Cab.Frec<>11025)then
  begin
    writeln('El fichero debe ser mono, 16 bits/m.y 11025 Hz');
    exit;
  end;

 {Se activa el modo grafico}
  Gd := Detect;
  InitGraph(Gd, Gm, '');
  if GraphResult <> grOk then begin
    writeln('Error grafico');
    exit;
  end;

 {Se establece la escala de colores}
  setpalette(0,0);
  setpalette(1,1);
  setpalette(2,11);
  setpalette(3,4);
  setpalette(4,36);
  setpalette(5,34);
  setpalette(6,22);
  setpalette(7,63);
 {Se establece el eje de coordenadas de la T.de Fourier}
  setcolor(12);
  line(20,230,20,430);
  line (20,430,620,430);
  for i:=0 to 600 do begin
    if i mod 10 = 0 then line(i+20,429,i+20,431);
    if i mod 50 = 0 then line(i+20,425,i+20,435);
  end;
  for i:=0 to 200 do begin
    if i mod 5 = 0 then line(19,i+230,21,i+230);
    if i mod 25 = 0 then line(15,i+230,25,i+230);
  end;
 {Se dibuja la escala de colores}
  rectangle(19,449,176,471);
  for i:=0 to 7 do begin
    setfillstyle(solidfill,i);
    bar(i*20+20,450,i*20+20+15,470);
  end;

{************************************************************}
  j:=0;
```

```
  Ini_fft(Coseno,Seno);
  repeat
    Leer_ventana(Ventana,Fin_m);
    if Fin_m then break; {Si es el final del fichero sale}
   {T_fourier(Ventana,TF);}
    T_rapida_fourier(Ventana,TF);
    Normalizar(TF);

   {Se dibuja una linea espectral}
    for i:=0 to Tamanyo_v_f-1 do begin
      putpixel(j+20,430-i,color(TF[i]));
      putpixel(j+20+1,430-i,color(TF[i]));
    end;

    j :=j+2;
  until j>600; {Se dibujan 301 lineas espectrales}
  repeat until keypressed;
  closegraph;

end.
```

En la siguiente figura tenemos el resultado de la ejecución sobre un fichero que contiene un armónico de 3000Hz con una duración de 1 segundo. En el eje Y se representan las frecuencias (cada punto representa 43Hz), y en el eje X el tiempo (cada punto representa 45'4ms, se calcula multiplicando el periodo de muestreo por el incremento de ventana).

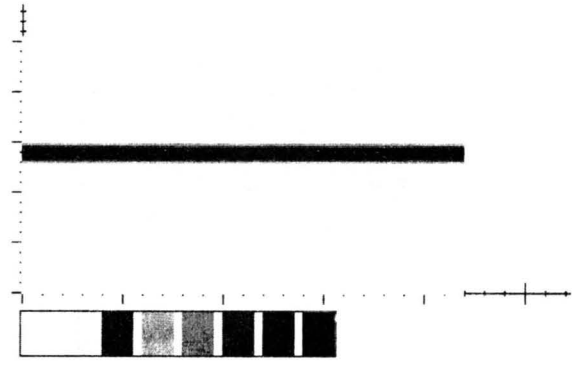


*Figura 11. Espectro de un armónico de 3000Hz.*

## 6. CONCLUSIONES

Hemos descrito de una forma detallada y práctica los fundamentos para el manejo de fichero de sonido a través de programación.

Se ha descrito la problemática que existe en los diferente formatos cuando manejamos ficheros de sonido. A modo de ejemplo se realizan dos operaciones básicas: suma de sonidos y cambio de frecuencia de muestreo. Con ello se quiere presentar la forma en que se deben operar.

Para el cálculo de parámetros en el dominio del tiempo se han descrito la utilización de ventanas, nos vemos obligados a trabajar así por la no estacionaridad de la voz. Se describe como presenta la información calculada de modo gráfica para su mejor interpretación, aplicando factores de normalización.

Para obtener el espectrograma de la señal de voz se utiliza la Transformada de Fourier. Se presenta una explicación de los aspectos necesarios para una presentación gráfica en dos dimensiones (frecuencias y tiempo), utilizando el color como tercera dimensión (energía).

## 7. REFERENCIAS

[Bri88] E. O. Brigham, *The Fast Fourier Transform and its Applications*, Prentice-Hall, Gran Bretaña, 1988.

[Coo93] M. Cooke, S. Beet & M. Crawford, *Visual Representations of Speech Signals*, Wiley, Inglaterra, 1993.

[Gon87] R. C. González & P. Wintz, *Digital Image Processing*, Addison-Wesley, EE.UU., 1987.

[Mar87] J. Martí Roca, "FFT como herramienta de análisis en fonética", *Estudios de fonética experimental*, mayo 1987.

[Rab78] L. R. Rabiner & R. W. Schafer, *Digital Procesing of Speech Signals*, Prentice-Hall, New Jersey, 1978.

[Rab93] L. R. Rabiner & B. H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, New Jersey, 1993.

[Cre93] Creative Labs, *Developer kit for Sound Blaster series*, Creative Labs, 1993.

[Hei93] R. Heimlich, D. M. Golden, I. Luk & P. M. Ridge, *Sound Blaster: the official book*, McGraw-Hill, California, 1993.

[Opp89] A. V. Oppenheim & R. W. Shafer, *Discrete-Time Signal Procesing*, Prentice-Hall, New Jersey, 1989.

[Par86] T. W. Parsons, *Voice and Speech Processing*, McGraw-Hill, EE.UU., 1986.

[Row92] C. Rowden, *Speech Procesing*, McGraw Hill, Cambridge, 1992.