Transmisión Progresiva de Vídeo en Internet

Sergio García Sanz

Ingeniero Informático

Febrero 2003

Directores de proyecto

Vicente González Ruiz
Leocadio González Casado

Índice general

Agradecimientos **11**

Preámbulo **13**

Abreviaturas **15**

1. Introducción a la compresión de vídeo **17**
- 1.1. Compresión de vídeo en el estándar MPEG-1 18
  - 1.1.1. Formato del vídeo de entrada 19
  - 1.1.2. Tipos de imagen MPEG 19
  - 1.1.3. Estructura del bit-stream de vídeo en MPEG-1 24
  - 1.1.4. El codificador y el descodificador en MPEG-1 26
- 1.2. Codificación escalable. Tipos 30
- 1.3. Motivación del proyecto 31

2. Compresión progresiva de vídeo **33**
- 2.1. La transformada wavelet discreta 33
- 2.2. La transformada S 35
- 2.3. La transformada 5/3 35
- 2.4. La transformada 13/7-T 37
- 2.5. La transformada wavelet discreta 3D 37
- 2.6. Compresión de los planos de bits 38
- 2.7. El algoritmo SPIHT-3D 40

3. El sistema desarrollado **49**
- 3.1. Formato de la información a transmitir 50
- 3.2. El servidor de vídeo progresivo 51
- 3.3. El cliente de vídeo progresivo 53
  - 3.3.1. Implementación del cliente de vídeo progresivo 53
  - 3.3.2. Mejora a la implementación 60

3.4. Descripción de la interfaz de la aplicación 62
3.4.1. Introducción a X Window 63
3.4.2. Diseño de la interfaz 63
3.4.3. Funcionamiento de la interfaz 65
3.4.4. Programación de la interfaz 66
3.4.5. Integración de la interfaz y el cliente 69

4. Evaluación 73
4.1. Evaluación objetiva 74
4.2. Evaluación subjetiva 76

5. Ampliaciones del proyecto 85

A. Funciones de la API de UNIX utilizadas 87
A.1. Señales y tiempos 87
A.1.1. signal 87
A.1.2. setitimer 88
A.2. Regiones críticas o mutex 90
A.2.1. pthread_mutex_init 90
A.2.2. pthread_mutex_lock 90
A.2.3. pthread_mutex_unlock 91
A.3. Procesos 91
A.3.1. fork 91
A.4. Sockets 91
A.4.1. socket 91
A.4.2. bind 92
A.4.3. listen 93
A.4.4. connect 93
A.4.5. accept 93
A.4.6. close 94
A.4.7. htonl 94
A.4.8. ntohl 94
A.4.9. select 94
A.5. Memoria compartida 96
A.5.1. shmget 96
A.5.2. shmat y shmdt 96
A.5.3. shmctl 97
A.6. X Window 98
A.6.1. XOpenDisplay 98
A.6.2. XCloseDisplay 98

A.6.3. DefaultScreen 98
A.6.4. XCreateSimpleWindow 98
A.6.5. XMapWindow 99
A.6.6. XSelectInput 99
A.6.7. XNextEvent 99
A.6.8. XCreateGC 100
A.6.9. XcopyArea 100

Bibliografía **101**

Índice de figuras

1.1. Ejemplo de estimación de movimiento 21
1.2. Uso periódico de la imagen I 21
1.3. Ejemplo de codificación bidireccional 22
1.4. Ejemplo de grupo de imágenes 24
1.5. Esquema de bloques de un codificador MPEG-1 27
1.6. Esquema de bloques de un descodificador MPEG-1 29
1.7. Ejemplo de escalabilidad espacial 31
1.8. Ejemplo de escalabilidad SNR 31

2.1. Cálculo de la DWT-3D 38
2.2. Árboles creados en la DWT-3D 40
2.3. Esquema de un PVC que utiliza el algoritmo SPIHT-3D . . . 41

3.1. Estructura de las etapas del cauce 50
3.2. Formato del fichero comprimido con SPIHT-3D 51
3.3. Ejemplo de interacción entre el servidor y el cliente 52
3.4. Representación gráfica del cliente de vídeo progresivo 55
3.5. Secuencia de operaciones con las funciones sin invertir 58
3.6. Secuencia de operaciones con las funciones invertidas 59
3.7. Representación del cauce de ejecución del cliente de vídeo progresivo con las funciones invertidas 60
3.8. Representación gráfica del cliente de vídeo progresivo con las funciones invertidas 61
3.9. Imagen principal de la interfaz 64

4.1. Reconstrucción de Claire en PSNR a 104 kbps 74
4.2. Reconstrucción de Claire en PSNR a 128 kbps 75
4.3. Reconstrucción de Claire en PSNR a 256 kbps 75
4.4. Reconstrucción de Charla en PSNR a 112 kbps 77
4.5. Reconstrucción de Charla en PSNR a 128 kbps 77

4.6. Reconstrucción de Charla en PSNR a 256 kbps 78
4.7. Fotogramas de "Claire" a 104 kbps 79
4.8. Fotogramas de "Claire" a 128 kbps 80
4.9. Fotogramas de "Claire" a 256 kbps 81
4.10. Fotogramas de "Charla" a 112 kbps 82
4.11. Fotogramas de "Charla" a 128 kbps 83
4.12. Fotogramas de "Charla" a 256 kbps 84

Índice de tablas

4.1. PSNR[dB] medio obtenido para la secuencia “Claire” usando PVC y MPEG-1 76
4.2. PSNR[dB] medio obtenido para la secuencia “Charla” usando PVC y MPEG-1 76

Agradecimientos

Ante todo agradecer a mis padres, que me han dado la posibilidad de estudiar siempre que he querido y me han apoyado en los buenos y malos momentos.

A mi compañero y amigo Manuel del Castillo por la facilidad para quedar a horas interpectivas para estudiar.

A José Ruano, gerente de mi empresa, que durante estos años no me ha puesto ningún impedimento y me ha adaptado el horario para poder compaginar estudios y trabajo.

Por último agradecer a mis tutores de proyecto, Vicente y Leocadio que se han portado de manera excelente durante todo este tiempo.

A todos mi gran sentido agradecimiento.

Preámbulo

Uno de los retos que existen en Internet es la transmisión en ”tiempo real”de secuencias de vídeo digital. Esto es debido a que en Internet no es posible garantizar una tasa de información que nos permita transmitir a una calidad aceptable, lo que provoca que el vídeo se entrecorte o incluso no pueda ser reproducido. Otro inconveniente que existe es que cuando almacenamos la secuencia a un bitrate fijo, obtener la secuencia original no es posible ya que normalmente se usan algoritmos de compresión con pérdida.

Sin duda alguna, uno de las principales objetivos es encontrar un algoritmo de compresión y transmisión adecuado que solucione estos dos problemas.

En la actualidad los algoritmos de compresión y transmisión de vídeo digital se basan en el estándar MPEG. Estos algoritmos están diseñados para transmitir a una tasa de bits fijos y además son algoritmos con pérdida, es decir, que una vez que la secuencia original es comprimida no es posible realizar el proceso inverso.

Una posible solución a los problemas anteriormente citados es el uso de algoritmos de compresión de vídeo digital que sean sin pérdida, es decir, que el proceso de compresión-descompresión sea reversible. Estos algoritmos tienen la característica de que no tienen ninguna restricción a la hora de transmitir la información por la red, es decir, no necesitan una tasa de bit fija para poder visualizarse.

Teniendo en cuenta estas premisas en este proyecto se pretende realizar un software que nos permita transmitir vídeo digital por Internet sin importar el ancho de banda disponible en cada momento, que sea escalable en calidad y que se pueda representar sin pérdida alguna si alguna situación especial lo requiera.

El documento se divide en los siguientes capítulos:

- El Capítulo 1, titulado “Introducción a la compresión de vídeo”, realiza una descripción de conceptos relacionados con la compresión de vídeo

pero centrándose en el estándar MPEG. Realiza una descripción del estándar explicando sus características, ventajas e inconvenientes. El capítulo termina con la motivación del proyecto.

- El Capítulo 2, titulado 'Compresión progresiva de vídeo", describe la teoría necesaria para comprender las transformadas wavelets. Realiza la descripción de los filtros usados en el proyecto. Se describen las características de las wavelet en 3D. Para finalizar el capítulo se describe el codec de vídeo progresivo de coeficientes wavelet SPIHT-3D.
- El Capítulo 3, llamado "El sistema desarrollado", describe como se ha desarrollado el PVC (Progressive Video Codec). Se describen todos los problemas encontrados a la hora del desarrollo y las soluciones que se le han dado.
- El Capítulo 4, titulado "Evaluación", muestra las pruebas realizadas al PVC comparado con MPEG-1 para comprobar la calidad del software desarrollado.
- El Capítulo 5, llamado "Ampliaciones del proyecto", describe las ampliaciones al proyecto para indicar las líneas de trabajo futuro.

El documento finaliza con un apéndice donde se describen las APIs del sistema operativo utilizadas para la realización del proyecto.

Abreviaturas

AC (Alternate Current)
AOE (Árbol de Orientación Espacial)
ATSC (Advance Television System Committee)
bitstream (flujo de bits)
bit-rate (ratio de bits)
BPG (bits per GOP)
bps (bits per second)
CCIR (International Radio Consultative Committee)
CD (Compact Disc)
Codec (Codificador-descodificador)
DC (Direct Current)
DCT (Discrete Cosine Transform)
DWT (Discrete Wavelet Transform)
fps (frames per second)
frame-rate (ratio de frames)
GOP (Group of Pictures)
GUI (Graphical User Interface)
HDTV (High Definition Television)
Hz (Hertz)
ISO (International Organization for Standardization)
JPEG (Joint Photographic Experts Group)
kbps (10^3 bits per second)
LIC (List of Insignificant Coefficients)
LIS (List of Insignificant Sets)
LSC (List of Significant Coefficients)
Mbps (Megabits per second –10^6 bits per second–)
MHz (10^6 Hz)
MPEG (Moving Pictures Experts Group)
MSE (Mean Square Error)
NTSC (National Television System Committee)

PAL (Phase Alternating Line)
PVC (Progressive Video Codec)
SIF (Source Input Format)
SNR (Signal Noise Ratio)
SPIHT (Set Partitioning In Hierarchical Trees)
VLC (Variable Lenght Coding)

Capítulo 1

Introducción a la compresión de vídeo

A medida que las comunicaciones digitales se imponen cada vez más en el mundo de hoy, el procesamiento digital de señales obtiene un interés especial, debido a que este es la base para muchas aplicaciones importantes, como la televisión digital, la multimedia, el sonido digital, etc..

Hace una década, la posibilidad de difundir vídeo digital destinada al público en general a través de una red de transmisión de datos como puede ser Internet parecía muy lejana, incluso se pensaba que su introducción no llegaría antes del final de este siglo XX.

La razón fundamental para afirmar esto era el importante flujo de datos que se necesitaba para transmitir una imagen de vídeo digitalizada, el cual era como mínimo de 100 Mbps.

A partir de finales de los años 80, el rápido desarrollo de eficaces algoritmos de compresión de vídeo, como el estándar JPEG (Joint Photographic Experts Group) para imágenes fijas y MPEG (Moving Pictures Experts Group) para imágenes en movimiento, reducirían de forma significativa el flujo necesario para la transmisión de imágenes, lo que cambió radicalmente el panorama, al llevar aquellas cantidades a valores mucho más razonables (de 1,5 a 30 Mbps, dependiendo de la resolución de las imágenes en movimiento). Además los progresos en integración permitían considerar la realización práctica de circuitos de descompresión, así como los circuitos de memoria asociados, a un precio asequible.

En los siguientes apartados describiremos el estándar MPEG-1, así como algunos conceptos de compresión de vídeo generales, particularizando para dicho estándar.

1.1. Compresión de vídeo en el estándar MPEG-1

En el año 1990, la ISO (International Organization for Standardization) y la IEC (International Electrotechnical Commission), preocupados por la necesidad de almacenar y reproducir imágenes de vídeo digitales y su sonido estéreo correspondiente, creo un grupo de expertos que llamó MPEG (Moving Pictures Expert Group), procedentes de aquellas áreas implicadas en el problema (telecomunicaciones, informática, electrónica, radio difusión, etc.).

El primer trabajo de este grupo se conoció en el año 1992 como la norma ISO/IEC 11172, mucho más conocida como MPEG-1. Este estándar está dividido en cinco partes:

- 11172-1 (Sistema): Describe la sincronización y multiplexación de señales de audio y vídeo.
- 11172-2 (Vídeo): Describe la compresión de señales de vídeo, centrándose en el escaneo progresivo (no entrelazado) y considerando especialmente las aplicaciones de vídeo en CD (Compact Disc).
- 11172-3 (Audio): Describe una familia genérica de codificadores de audio, con tres miembros jerárquicamente compatibles, denominados layer-1, layer-2 y layer-3.
- 11172-4 (Test de conformidad): Describe los procedimientos para determinar las características de los bitstreams (flujo de bits) codificados y el proceso de descodificación, así como los tests de conformidad con los requerimientos establecidos en las otras partes.
- 11172-5 (Simulación por software): Es un informe técnico sobre la implementación por software de las tres primeras partes de MPEG-1.

MPEG-1 fue creado para reproducir vídeo a pantalla completa con una resolución de 352×288 (NTSC a 30 fps –frames per second–), y a una tasa de transferencia de 1,5 Mbps[1], de los cuales 1,15 Mbps son para el vídeo y los 350 kbps[2] restantes para el sonido y datos auxiliares. Es el equivalente a un lector de CD-VideoCD, CDTV, CD-i, o CD-ROM de velocidad 1X. Este formato se acerca a la calidad ofrecida por los vídeos VHS.

Las aplicaciones de MPEG-1 son muchas. Es utilizado con frecuencia en enciclopedias, juegos y demostraciones publicitarias. El mayor éxito cosechado por este estándar es el sistema VideoCD. Se trata de un formato

[1] Mbps = 10^6 bits per second
[2] kbps = 10^3 bits per second

especial de vídeo, compatible MPEG-1 (352×288 pixeles a 30 fps en el caso de NTSC –National Television System Commitee–) capaz de almacenar 70 minutos de imágenes con calidad VHS y sonido con calidad CD, en un simple CD. Este vídeo es reproducible en cualquier unidad preparada para ello, como los reproductores de VideoCD y ordenadores equipados con un lector de CD-ROM y software o hardware reproductor de MPEG-1.

1.1.1. Formato del vídeo de entrada

El formato de entrada para MPEG-1 es el SIF (Source Input Format). Fue derivado del CCIR601 (International Radio Consultative Committee), un estándar de TV digital. CCIR601 especifica las coordenadas de color (Y, Cb, Cr) donde Y es la componente de luminancia (blanco y negro) y Cb y Cr son dos componentes de crominancia (diferencias de color). Se adoptó una frecuencia de muestreo de 13,5 MHz. Existen varios formatos de muestreo tales como 4:4:4, 4:2:2, 4:1:1 y 4:2:0. En 4:4:4, la frecuencia de muestreo para Y, Cb y Cr es la misma. En 4:2:2, la frecuencia de muestreo para Cb y Cr es la mitad que para Y.

Para convertir la señal de TV analógica a digital con la frecuencia de muestreo de 13,5 MHz del CCIR601 a 720 pixeles por 576 líneas (PAL –Phase Alternating Line–) y de 720 pixeles por 480 líneas (NTSC) a 30 y 25 fps respectivamente usando el formato 4:2:2 se necesitan aproximadamente 166 Mbps. Para poder conseguir vídeo con buena calidad a 1,5 Mbps, lo que se hizo fue reducir la resolución a un cuarto del CCIR601, el resultado es 360×240 para señal NTSC y 360×288 para señal PAL. Por motivos del algoritmo de estimación de movimiento el número de píxeles de ambas dimensiones deben ser múltiplos de 16. Para ello se descartan los 4 pixeles de los extremos derecho e izquierdo dando una resolución de 352×240 a 30 fps para NTSC y de 352×288 a 25 fps para PAL respectivamente.

1.1.2. Tipos de imagen MPEG

En MPEG cada secuencia de vídeo está dividida en uno o más GOPs (Group of Pictures). Estos están formados por tres tipos de imágenes: I, P y B.

Las imágenes I (Intra)

Son imágenes que no requieren información adicional para su descodificación. Son codificadas sin ninguna referencia a otras imágenes, es decir,

contienen todos los elementos necesarios para su reconstrucción por el descodificador.

Las imágenes I permiten realizar accesos aleatorios al vídeo comprimido, y además, movernos por la secuencia de una forma rápida. Una secuencia con imágenes de tipo I es más editable. Además, la inclusión de este tipo de imágenes hace que el error de compresión propagado de imágenes anteriores P y B termine, ya que no necesita de una imagen previa de cualquier tipo (I, P o B) para mostrarse.

Para comprimir las imágenes I se elimina la redundancia espacial, que es aquella provocada por el hecho de que los puntos adyacentes son en general bastante semejantes en intensidad.

Una técnica para eliminar dicha redundancia es la codificación basada en transformadas, donde se utiliza una transformada lineal y reversible que hace corresponder una imagen con un conjunto de coeficientes en el dominio de la frecuencia. La imagen en el dominio de la frecuencia se puede transformar inversamente al dominio espacial, reproduciendo la imagen tal y como estaba originalmente [3].

En el dominio de la frecuencia, los coeficientes mas grandes tienden a agruparse en regiones, especialmente alrededor de las zonas de baja frecuencia. Como resultado, hay áreas generalmente grandes de la imagen donde los coeficientes tienen un valor muy pequeño o cero. Esto ocurre porque existe una alta correlación entre las componentes de frecuencia presentes en la imagen y las funciones base que definen la transformada. En consecuencia, la versión en el dominio de la frecuencia de la imagen es generalmente una representación muy eficiente de la imagen original. Las técnicas de compresión de imágenes basadas en transformadas se aprovechan de esta característica de la imagen en el dominio de la frecuencia, simplemente eliminando los coeficientes de la imagen que tienen los valores más pequeños. Puesto que la energía acumulada por estos coeficientes es mínima, cuando la imagen se transforma nuevamente hacia el dominio espacial, la eliminación de estos coeficientes causa una pequeña distorsión.

Por último, indicar que las imágenes I tienen una tasa de compresión relativamente pequeña, comparada con la que se consigue para imágenes tipo P y B.

Las imágenes P (previstas)

Se basan en que en una secuencia de imágenes en movimiento, en general, existe una alta dosis de redundancia temporal, esto es, que dos imágenes adyacentes en el tiempo van a ser, en general, bastante parecidas. Puesto

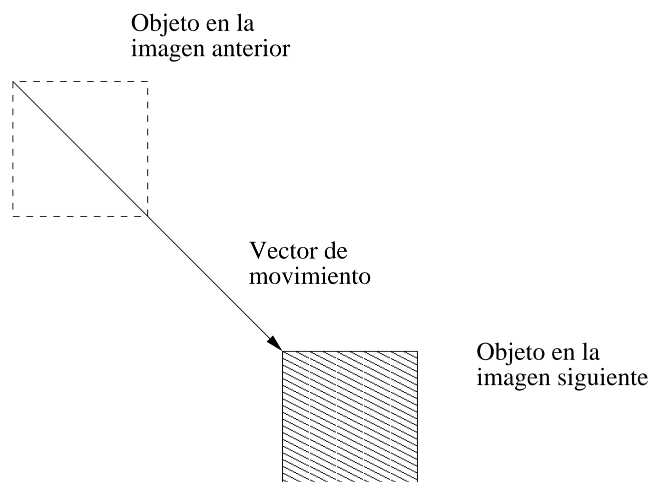

Figura 1.1: Un ejemplo de estimación de movimiento donde se obtienen los vectores de movimiento.

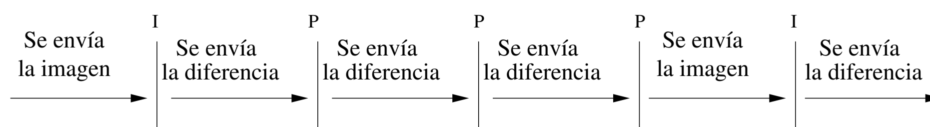

Figura 1.2: Uso periódico de la imagen I. Se muestran dos tipos de imágenes I (imagen intra) y P (imagen prevista).

que en la mayoría de las escenas lo que se producen son traslaciones de objetos, una técnica muy eficaz de estimar el contenido de una imagen en función de otra anterior es la estimación de movimiento.

La estimación de movimiento trata de calcular el movimiento de los objetos en una escena. La estimación de movimiento trabaja de la siguiente forma: cuando una imagen I es enviada, esta es almacenada de tal modo que pueda ser comparada con la siguiente imagen de entrada, para encontrar así los posibles vectores de movimiento (véase Figura 1.1), los cuales pueden ser localizados en diferentes áreas de la imagen. Luego la imagen I es combinada de acuerdo a estos vectores. La imagen prevista resultante es comparada con la imagen actual para producir una predicción de error, también llamada imagen residual. La predicción de error es transmitida con los vectores de movimiento. En el receptor la imagen I original es también retenida en la memoria, y es cambiada de acuerdo con los vectores de movimiento transmitidos, para crear la imagen prevista y luego la predicción de

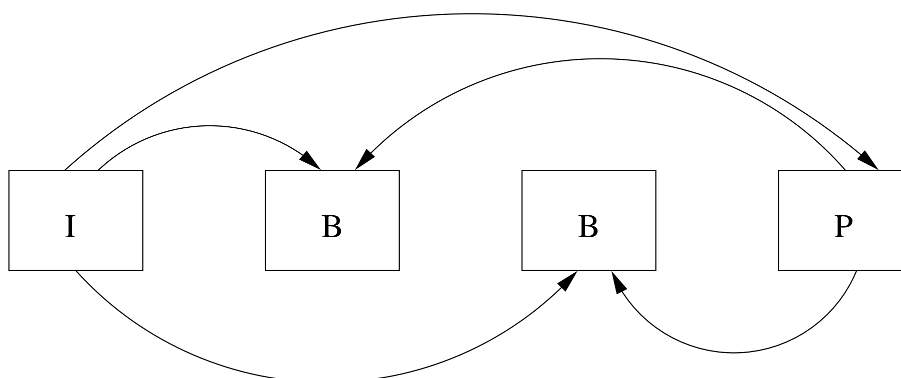

Figura 1.3: Un ejemplo de codificación bidireccional.

error es adicionada, recreando la imagen original.

La Figura 1.2 muestra la relación que existe entre las imágenes I y P. Mientras la imagen I requiere grandes cantidades de información, ya se que envía la información completa de la imagen, las imágenes P requieren una cantidad menor, ya que se codifica la diferencia con la imagen anterior. Por tanto, la tasa de compresión de imágenes P es claramente mayor que la de las imágenes I. En la práctica estas requieren aproximadamente la mitad de los datos de las imágenes I.

Las imágenes B (bidireccionales)

Las imágenes de tipo B se insertan entre las de tipo I y P, y son el resultado de una codificación bidireccional.

Cuando un objeto se mueve, este oculta lo que hay detrás de él, pero a medida que se va moviendo, permite observar el fondo. El revelado del fondo exige nuevos datos a ser transmitidos, ya que el área del fondo había sido ocultada anteriormente y la información no pudo ser obtenida desde una imagen previa.

La codificación bidireccional ayuda a minimizar este problema, ya que permite utilizar información de imágenes anteriores y posteriores a la imagen observada. Si el fondo ya ha sido revelado, y este es presentado en una imagen posterior, la información puede ser movida hacia atrás en el tiempo, creando parte de la imagen con anticipación.

La Figura 1.3 muestra un ejemplo de codificación bidireccional. Primero se toma una imagen I y con la ayuda de una imagen P se pueden obtener imágenes B, mediante interpolación, las cuales son llamadas también imáge-

nes bidireccionales. Las imágenes B utilizan estimación de movimiento para su compresión.

Como no se utilizan para describir otras imágenes, las imágenes B no propagan los posibles errores de compresión.

Este tipo de imágenes es el que ofrece el factor de compresión más alto, que generalmente es de una cuarta parte de los datos de las imágenes I.

Dependiendo de la complejidad del codificador utilizado, se podrán codificar sólo las imágenes I, las imágenes I y P o las imágenes I, P y B; sin duda, con resultados absolutamente diferentes a nivel del factor de compresión y en cuanto a las posibilidades de acceso aleatorio, así como del tiempo de codificación y de la calidad percibida.

Ejemplo de encadenado de imágenes. Factor de compresión

Como ya hemos indicado anteriormente, en el estándar MPEG-1, una secuencia de imágenes que está constituida por una imagen I y las siguientes imágenes P o B hasta el comienzo de otra imagen I se denomina grupo de imágenes GOP (Group of Pictures).

Existen dos parámetros que definen la manera en que las imágenes I, P y B se encadenan dentro del GOP, estos son M y N:

- M es la distancia (en número de imágenes) entre dos imágenes P (previstas) sucesivas.
- N es la distancia entre dos imágenes I (Intra) sucesivas.

Para alcanzar un flujo de vídeo de 1,15 Mbits con una calidad satisfactoria, al tiempo que se mantiene una velocidad de acceso aleatorio aceptable ($<$ 0,5 segundos)[3], los parámetros comúnmente utilizados son M=3 y N=12, como se muestra en la Figura 1.4.

En este caso, una secuencia de vídeo se compone de 1/12 (8,33 %) de imágenes I, 1/4 (25 %) de imágenes P y de 2/3 (66,66 %) de imágenes B. El factor de compresión global se ve favorecido por el hecho de que son las imágenes más frecuentes (imágenes B) las que tienen un factor de compresión mas alto.

En la visualización, tras la codificación y descodificación, es evidente que las imágenes de la secuencia de vídeo deben ser reproducidas en el mismo orden en que se captaron.

[3]El tiempo de acceso en una secuencia de vídeo varía en función del número de imágenes I que contenga. A mayor número de imágenes, mayor velocidad de acceso.

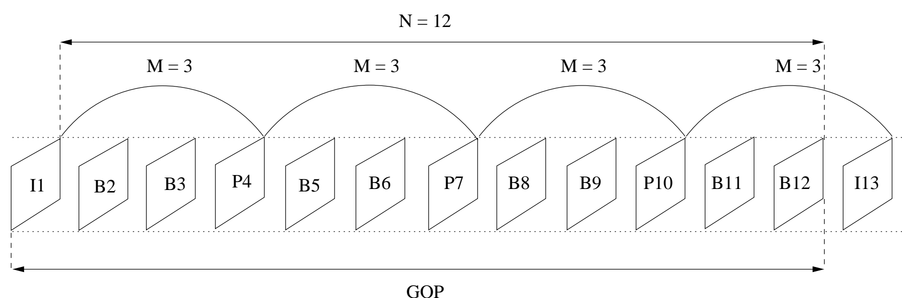

Figura 1.4: Un ejemplo de grupo de imágenes (GOP), para M=3, N=12.

Con los parámetros definidos anteriormente (M=3, N=12), la forma en que van llegando al compresor las imágenes es la siguiente:

> 1(I) 2(B) 3(B) 4(P) 5(B) 6(B) 7(P) 8(B) 9(B) 10(P) 11(B) 12(B) 13(I) 14(B) 15(B) 16(P)...

Sin embargo, para codificar o descodificar una imagen B (bidireccional), el codificador y el descodificador necesitarán la imagen I o P que la precede y la imagen P o I que la sigue. El orden de las imágenes será, por tanto, modificado antes de la codificación, de forma que el codificador y el descodificador dispongan, antes que de las imágenes B, de las imágenes I y/o P necesarias para su tratamiento, o sea que la llegada real de las imágenes al descompresor es:

> 1(I) 4(P) 2(B) 3(B) 7(P) 5(B) 6(B) 10(P) 8(B) 9(B) 13(I) 11(B) 12(B) 16(P) 14(B) 15(B)...

Para conseguir factores de compresión mayores, manteniendo la misma calidad de reproducción, se aumenta el número de imágenes P y B en el GOP. Esto sin embargo dificulta la recuperación eficaz de una secuencia de imágenes comprimidas que ha llegado con errores de transmisión.

1.1.3. Estructura del bit-stream de vídeo en MPEG-1

La estructura de una secuencia de MPEG-1 está formada por seis capas siendo estas: El bloque (block), el macrobloque (Macroblock), la rebanada (Slice), la imagen (Picture), el grupo de imágenes (Group of Pictures o GOP) y la secuencia de vídeo (video sequence).

Bloque (Block)

Es la unidad fundamental de la información de la imagen y está representada por un bloque de coeficientes DCT (Discrete Cosine Transform), que tiene un tamaño de 8×8, los cuales representan datos Y, Cr o Cb.

Aquí, el coeficiente DC es enviado primero ya que este representa con mayor precisión la información de este bloque (de hecho, a su media). Los demás coeficientes AC son enviados a continuación.

Macrobloque (Macroblock)

Cada macrobloque tiene un tamaño de 16×16 pixeles y es la unidad fundamental de la imagen que se emplea para compensar (estimar y eliminar) el movimiento. Cada macrobloque genera un vector de movimiento en las dos dimensiones del espacio. En una imagen B, el vector de movimiento puede ser hacia adelante o hacia atrás en el tiempo.

Usando los vectores, el descodificador obtiene información acerca de las imágenes anteriores y de las posteriores, produciendo así la predicción de imágenes que hizo el codificador (véase la Figura 1.5). Los macrobloques son transformados inversamente para producir una imagen de rectificación que es adicionada a la imagen prevista que ha sido producida a la salida del descodificador.

Rebanada (Slice)

Los macrobloques son agrupados en rebanadas, y siempre deben representar una fila horizontal que está ordenada de izquierda a derecha.

En MPEG, las rebanadas pueden comenzar en cualquier sentido y ser de tamaño arbitrario, pero el ATSC (Advance Television Systems Committee) estableció que deben comenzar en el borde izquierdo de la imagen. Las rebanadas son la unidad fundamental de sincronización para la codificación de la longitud variable que se encarga de encontrar una representación eficiente para los coeficientes DCT cuantificados y para los vectores de movimiento. Los vectores iniciales en una rebanada son enviados completamente, mientras que los demás vectores son transmitidos diferencialmente.

Por otra parte, en las imágenes I, los primeros coeficientes DC de las rebanadas son enviados completamente y los demás coeficientes AC son transmitidos en forma diferencial.

Imagen (Picture)

Cuando un número de rebanas se combinan, construyen una imagen. La imagen de soporte inicial define qué imágenes I, P o B codifica e incluye una referencia temporal (el número de secuencia) para que la imagen pueda ser representada en el momento adecuado. Un vector global puede ser enviado para toda la imagen, y luego se pueden enviar vectores individuales que lleguen a crear la diferencia en el vector global.

Grupo de imágenes (Group of Pictures o GOP)

Las imágenes son agrupadas para producir un GOP (grupo de imágenes) que siempre comienza con una imagen I. El GOP es la unidad fundamental de codificación temporal. Entre imágenes I, un número variable de imágenes P y/o B pueden ser colocadas como se ha descrito anteriormente.

Secuencia (video sequence)

La unión de dos o más GOPs produce una secuencia de vídeo. Al comienzo de la secuencia se especifica el tamaño horizontal y vertical de la imagen, norma de barrido, la tasa (temporal) de imágenes, si se usa un barrido progresivo o entrelazado, el perfil, nivel, velocidad de transferencia de bits, y qué matrices de cuantificación se usan. Sin estos datos, un descodificador no puede comprender el flujo de bits y por lo tanto no puede comenzar la operación de descodificación correcta. Esto ocurre generalmente cuando un televidente está cambiando canales de un lugar a otro en su televisor.

1.1.4. El codificador y el descodificador en MPEG-1

El estándar internacional no define explícitamente el proceso de codificación, sino únicamente la sintaxis y semántica del flujo de bits (bit-stream) a la salida del codificador. Esto deja una gran libertad a la hora de diseñar el codificador.

Un codificador de MPEG-1 cuyo esquema de bloques está representado por la Figura 1.5 incluye los siguientes módulos:

- Reordenación de las imágenes: A este módulo llegan las imágenes originales en el orden en que fueron tomadas. Se encarga de reordenarlas para poder usar la codificación diferencial y bidireccional utilizada al comprimir imágenes de tipo P y B.

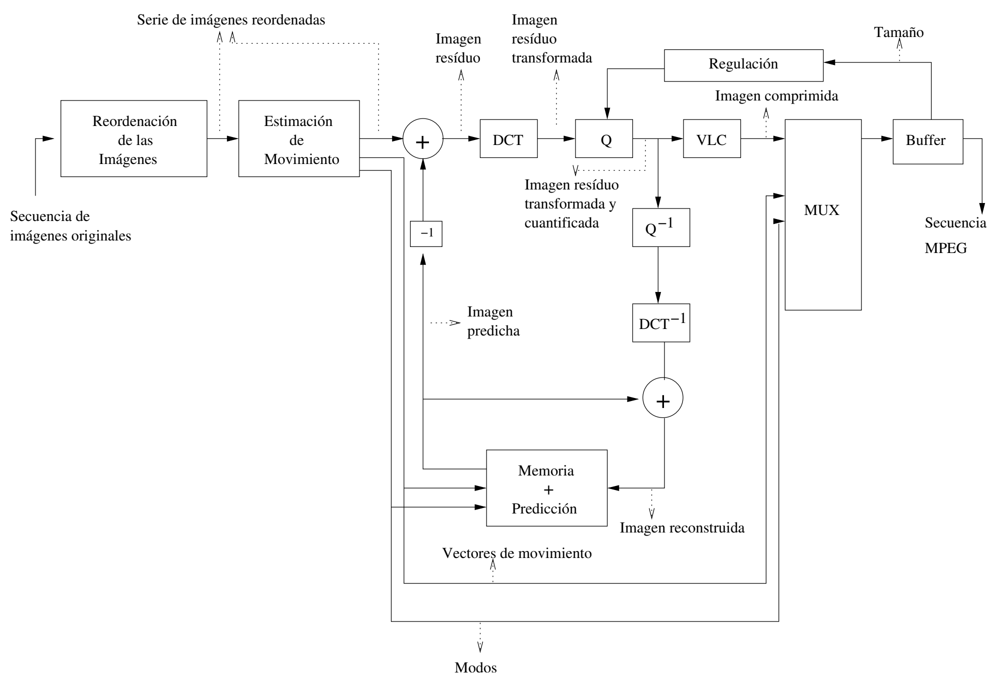

Figura 1.5: Esquema de bloques de un codificador MPEG-1.

- Estimación de movimiento: A la entrada de este módulo tenemos la serie de imágenes reordenadas. Se obtienen los vectores de movimiento y los modos de codificación para cada macrobloque de 16×16 pixeles. También se decide el modo de codificación de cada uno de ellos. A la salida además tendremos las imágenes reordenadas que recibimos a la entrada.

- Memoria + Predicción: Recibe la imagen reconstruida, los vectores de movimiento y los modos de codificación. En este módulo se calcula la imagen predicha, bien a partir de la imagen de referencia (I o P) pasada y en algunos casos futura (caso de imágenes B) que luego será eliminada de la imagen original, dando lugar a la imagen residuo.

- DCT: A este módulo van llegando las imágenes residuo las cuales se van separando en bloques de 8×8 aplicando la DCT a cada uno de ellos. A la salida tendremos la imagen residuo transformada.

- Q: A este módulo van llegando las imágenes residuo transformadas, se las cuantifica usando una matriz preestablecida y obteniendo de esta forma muchos ceros en los coeficientes transformados.

- VLC: A la entrada de este módulo tendremos la imagen residuo transformada y cuantificada. Se codifican los coeficientes de la imagen aplicando una tabla VLC (Variable Lenght Coding, en concreto una codificación Huffman[2]), haciendo que se pueda representar la imagen con una menor cantidad de datos. A la salida tendremos la imagen comprimida.

- Control del bit-rate: Tiene como entrada el tamaño ocupado del buffer. Se encarga de controlar el estado de ocupación. Para ello se cambian los valores de la matriz de cuantificación para obtener un mayor número de ceros, consiguiendo que después en la etapa VLC se realice una mayor compresión. A la salida tendremos una secuencia MPEG-1.

- Buffer: En este módulo se va almacenando el flujo de bits completo utilizable por un descodificador MPEG-1.

Un descodificador MPEG-1 invierte las operaciones del codificador. El esquema de bloques se muestra en la Figura 1.6. El descodificador incluye los siguientes módulos:

- Buffer: A este módulo va llegando la secuencia MPEG-1 del canal de transmisión.

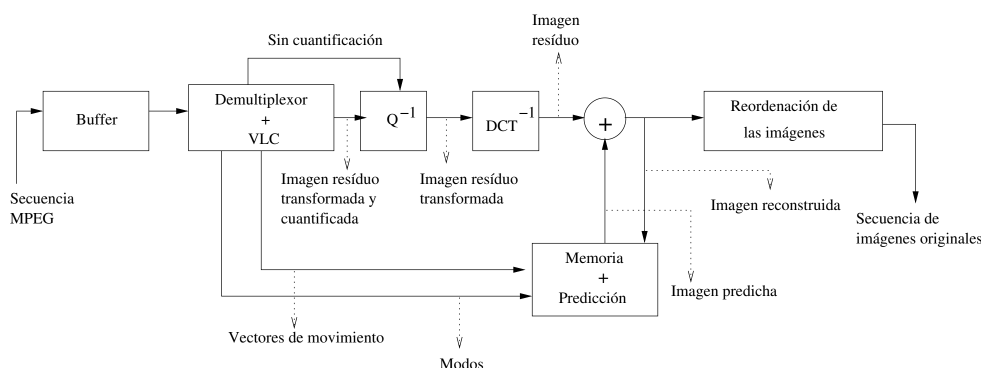

Figura 1.6: Esquema de bloques de un descodificador MPEG-1.

- Demultiplexor + VLC: A este módulo van llegando las imágenes provenientes del buffer, se realiza la separación de la información relativa a la imagen comprimida, los vectores de movimiento y los modos de codificación. Se aplica la descodificación VLC a la imagen comprimida dando lugar a la imagen residuo transformada y cuantificada.

- Q^{-1}: Este módulo realiza la descuantificación de cada uno de los bloques de 8×8 que forman la imagen residuo, obteniendo de esta forma la imagen residuo transformada.

- DCT^{-1}: Una vez descuantificada la imagen, se aplica la DCT^{-1} a los bloques de 8×8, dando lugar a la imagen residuo, que en el caso de imágenes I o P pasa a la memoria para ser usada como imagen de referencia para sucesivas imágenes.

- Memoria + Predicción: Este módulo se encarga de obtener la imagen predicha que después servirá para adicionada con la imagen residuo obtener la imagen reconstruida. Para ello necesita de los vectores de movimiento y de los modos de codificación de los bloques de 8×8 que forman la imagen además de la imagen de referencia previa.

- Reordenación de las imágenes: Una vez que tenemos las imágenes reconstruidas, se tienen que ir reordenando en función del orden de visualización.

1.2. Codificación escalable. Tipos

En este apartado describiremos los distintos tipos de escalabilidad que existen, particularizando después en el estándar MPEG.

En términos generales, la codificación escalable es aquella que nos permite visualizar el vídeo a distintas calidades (resolución, frame-rate o SNR), en función de la necesidad del momento. Para conseguir aumentar la calidad en alguna cualidad es necesario disponer de una información suplementaria que se añade a una información base. Este es el principio de la codificación escalable.

Esta técnica es útil para un gran número de aplicaciones donde el vídeo necesita ser descodificado a diferentes resoluciones y en distintos niveles de calidad, por ejemplo, en videoconferencia, en televisión de alta definición (HDTV), en televisión estándar, etc. Básicamente existen tres tipos de escalabilidad:

- Espacial (Spatial scalability). La escalabilidad espacial nos permite ver la secuencia de vídeo a distintas resoluciones, es decir, a distintos tamaños de imágenes (véase la Figura 1.7).

- Calidad (SNR scalability). La escalabilidad en calidad (SNR) nos permite ver la secuencia de vídeo a distinta calidad de imagen pero mantenimiento la misma resolución espacial y temporal (véase la Figura 1.8).

- Temporal (Temporal scalability). La escalabilidad temporal nos permite ver la secuencia de vídeo a distinta resolución temporal, es decir, a un número distinto de imágenes por segundo, mantenimiento la misma resolución espacial y calidad SNR.

Particularizando en el estándar MPEG, nos apoyaremos en el concepto de layer o capa. Esta técnica se basa en la idea de comprimir la información del vídeo a distintas calidades, formando la capa base (base layer) y la/s capas de refinamiento (enhancement layers). Para que las capas de refinamiento mejoren la calidad del vídeo es necesario complementarla con la información de la capa base.

Por ejemplo, en MPEG, para conseguir escalabilidad espacial, la capa base se codifica con la menor resolución mientras que las capas de refinamiento se codifican con resoluciones superiores. Para la escalabilidad en calidad, en la capa base se codifican toscamente los coeficientes de la transformada. En

las capas de refinamiento se almacenan los coeficientes que tienen información sobre los detalles, utilizados para mejorar la calidad de imagen. Y para la escalabilidad temporal, la capa base nos ofrece la resolución temporal más baja mientras que con la capa de refinamiento obtenemos la más alta.

Figura 1.7: Un ejemplo de escalabilidad espacial.

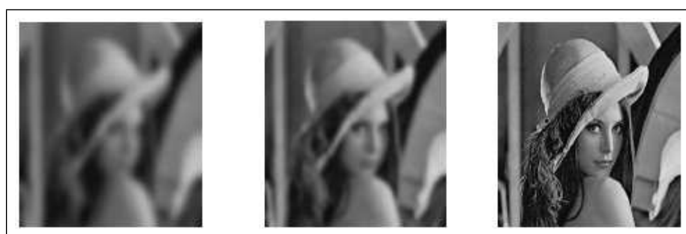

Figura 1.8: Un ejemplo de escalabilidad SNR.

1.3. Motivación del proyecto

En los últimos años, los avances en comunicaciones han permitido que la mayoría de usuarios tengamos acceso a Internet. La velocidad de transmisión/recepción cada vez es más rápida, pero todavía no es lo suficiente para poder disfrutar de una forma óptima de ciertos servicios como pueden ser videoconferencia, vídeo bajo demanda, etc.

Otro inconveniente añadido, es que en Internet el ancho de banda de que se dispone, en general, varía a lo largo del tiempo, por lo que no se puede garantizar la disposición de un valor determinado de ancho de banda en un instante dado. Por otra parte, para la transmisión de vídeo es necesario disponer del mayor ancho de banda disponible en cada momento para que su visualización sea a la mayor calidad posible. Si no somos capaces de

aprovechar al máximo el canal de comunicación, estaremos infrautilizando los recursos de que disponemos.

El estándar MPEG consigue escalabilidad mediante capas[4] según comentamos en el apartado anterior. La escalabilidad por capas tiene el problema de que bajo ciertas condiciones no aprovecha el canal de forma óptima.

Mostraremos un supuesto donde se ponga de manifiesto este problema: supongamos que disponemos de un canal de comunicación que nos permita transmitir a 128 kbps de forma constante. Disponemos también de un compresor MPEG escalable en calidad, que al comprimir genera tres capas, una capa base comprimida a 32 kbps y dos capas de mejora comprimidas cada una a 64 kbps. Este sistema podría transmitir la capa base y la primera capa de mejora, necesitando una velocidad de transmisión de 96 kbps. La segunda capa de mejora no se puede transmitir ya que no se dispone de ancho de banda suficiente, por lo tanto se está desaprovechando un 25 % del canal de transmisión.

Debido a los problemas propios de Internet y de los estándares existentes comentados anteriormente, el principal objetivo del proyecto es desarrollar un sistema escalable que nos permita aprovechar el canal de comunicación de una forma óptima.

El algoritmo desarrollado en este proyecto permite una escalabilidad a nivel de bits[5] con un bit-stream seccionable en cualquier momento de la transmisión, con lo que se aprovecha el canal de transmisión de forma completa y adaptativa.

[4]Escalabilidad de grano grueso.
[5]Escalabilidad de grano fino.

Capítulo 2

Compresión progresiva de vídeo

2.1. La transformada wavelet discreta

La atención que han recibido las transformadas wavelets en los últimos años es sencillamente abrumadora. Esto ha generado, tanto a nivel teórico como de aplicación (especialmente en el procesamiento y compresión de señales) una línea de investigación muy importante.

Como muchas de las transformadas usadas en compresión, su función es calcular un conjunto de coeficientes que indican la correlación que existe entre la señal a transformar y una serie de funciones base que forman un espacio ortogonal, tratando así de acumular en el mínimo número de coeficientes la máxima cantidad de información (energía), y si es posible, disminuir además la entropía de los datos [9].

Sin pérdida de generalidad, trataremos sólo por ahora el caso unidimensional. Sea $f[x], x = 0, 1, \cdots, 2^n - 1$ el vector de muestras a transformar. Para encontrar una representación que cumpla el objetivo propuesto, vamos a emplear un modelo espacial como predictor, pero en lugar de usarlo de forma secuencial, procesando el vector desde un extremo a otro vamos usar dicho modelo para calcular primero una versión reducida de tamaño mitad de $f[x]$ a la que llamaremos $l[x], x = 0, 1, \cdots, 2^{n-1} - 1$. La idea es usar $l[x]$ para generar un vector $\hat{f}[x]$ que sea tan parecido a $f[x]$ como sea posible. De esta forma, el vector residuo

$$h[x] = f[x] - \hat{f}[x]$$

contendrá la información que el modelo no es capaz de predecir.

Si el modelo de predicción es adecuado, las entropías de $h[x]$ y de $l[x]$ son inferiores a la de $f[x]$.[1] Por ejemplo, sea $f[] = \{1, 3, 2, 1\}$. Una posible forma de encontrar $l[]$ consiste en calcular la media de cada dos muestras: $l[] = \{2, 1{,}5\}$. A partir de este vector, una predicción que podemos hacer a cerca de $f[]$ es $\hat{f}[] = \{2, 2, 1{,}5, 1{,}5\}$, resultando un vector residuo $h[] = \{-1, 1, 0{,}5, -0{,}5\}$. Nótese que $h[]$ está formado por coeficientes que de dos en dos sólo se diferencian en el signo.

Como hemos visto, una forma de construir $l[x]$ es calcular la media de cada dos muestras de $f[x]$, es decir

$$l[x] = \frac{f[2x] + f[2x+1]}{2}, x = 0, 1, \cdots, 2^{n-1} - 1. \qquad (2.1)$$

Si para generar $\hat{f}[x]$ usamos cada muestra de $l[x]$ dos veces de forma consecutiva, entonces sólo se necesitan la mitad de las muestras (las pares o las impares) de $h[x]$ para reconstruir $f[x]$. De hecho, $h[x]$ puede calcularse también usando

$$h[x] = \frac{f[2x+1] - f[2x]}{2}, x = 0, 1, \cdots, 2^{n-1} - 1. \qquad (2.2)$$

En dicha transformada, se obtiene una representación alternativa para $f[x]$ formada por dos vectores de tamaño mitad: $l[x]$ y $h[x]$, que ocupan en conjunto el mismo espacio que $f[x]$. El vector $l[x]$ almacena la media de cada dos muestras de $f[x]$ y $h[x]$ almacena la diferencia entre cada par de muestras dividida entre dos.

Desde el punto de vista del procesamiento de señales [11], el proceso descrito coincide con la descomposición de una señal en dos bandas de frecuencia [10]. Una de ellas ($l[x]$) contiene la banda de baja frecuencia y la otra ($h[x]$) la banda de alta frecuencia. Esta descomposición puede ser aplicada de forma recursiva a la banda de baja frecuencia, generándose así lo que se conoce como transformada wavelet discreta o DWT (Discrete Wavelet Transform).

Para terminar la descripción de la DWT, sólo queda decir que es muy rápida. Como es conocido [1, 4], el filtrado de una señal $f[x]$ puede realizarse convolucionando dicha señal con la respuesta del filtro al impulso unitario y este cálculo es de complejidad lineal $O(N)$ (donde $N = 2^n$) y proporcional a la longitud del filtro.

[1] Los elementos de $h[x]$ tienden a ser cero y en $l[x]$ el número de símbolos usados tiende a disminuir.

2.2. La transformada S

La transformada S (ST: Sequential Transform) [7, 9] también llamada transformada de interpolación (1,1) evita la división en punto flotante (ver Ecuaciones (2.1) y (2.2)) lo cual es fundamental si se desea que la transformada sea totalmente reversible. Sólo de esta forma, la transformación inversa recupera siempre la señal original.

La transformada S directa para 2^n puntos se calcula aplicando los filtros de análisis

$$l[i] = \left\lfloor \frac{f[2i] + f[2i+1]}{2} \right\rfloor \tag{2.3}$$

y

$$h[i] = f[2i] - f[2i+1]. \tag{2.4}$$

donde $\lfloor\cdot\rfloor$ representa al operador "el mayor entero menor que".

Los filtros de síntesis usados en la transformada inversa se encuentran fácilmente a partir de las Ecuaciones (2.3) y (2.4), resultando

$$f[2i] = l[i] + \left\lfloor \frac{h[i]+1}{2} \right\rfloor \tag{2.5}$$

y

$$f[2i+1] = f[2i] - h[i]. \tag{2.6}$$

Nótese que la banda de baja frecuencia almacena la media de la señal y la banda de alta frecuencia representa los errores que resultan de predecir que las muestras impares son iguales a las pares, dos a dos. Por ejemplo, $h[0]$ sería igual a 0 si las dos primeras muestras de $f[i]$ fueran idénticas.

2.3. La transformada 5/3

La transformada 5/3 también llamada transformada de interpolación (2,2) hace la suposición de que la señal está compuesta por funciones lineales. Utiliza dos puntos para predecir una muestra. Para ello supone que las muestras impares $s[2i+1]$ están a mitad de camino entre la muestra anterior $s[2i]$ y la siguiente muestra $s[2i+2]$. Por esa razón, la transformada 5/3 se calcula aplicando los filtros de análisis

$$h[i] = s[2i+1] - \left\lfloor \frac{s[2i] + s[2i+2]}{2} \right\rfloor, \; i = 0, \cdots, \frac{N}{2} - 1 \tag{2.7}$$

y

$$l[i] = s[2i] + \left\lfloor \frac{h[i] + h[i-1]}{4} + \frac{1}{2} \right\rfloor, \; i = 0, \cdots, \frac{N}{2} - 1. \qquad (2.8)$$

Tenemos que contemplar los siguientes casos especiales:

1. Si el número N de muestras de s es impar, entonces la expresión (2.7) es definitiva. Nótese que la longitud de la secuencia de muestras será $\frac{N}{2} - 1$.

2. Si N es par, la muestra $s[2i+2]$ para $i = \frac{N}{2} - 1$ no existe y debemos usar
$$h[i] = s[N-1] - s[N-2]. \qquad (2.9)$$
Nótese que esto es el resultado de suponer que la muestra $s[N] = s[N-2]$. En este caso, la 5/3 es idéntica a la ST.

Los filtros de síntesis utilizados en la transformada inversa son los siguientes

$$s[2i] = l[i] + \left\lfloor \frac{h[i-1] + h[i]}{4} + \frac{1}{2} \right\rfloor, \; i = 1, \cdots, \frac{N}{2} - 1 \qquad (2.10)$$

y

$$s[2i+1] = h[i] + \left\lfloor \frac{s[2i] + s[2i+2]}{2} \right\rfloor, \; i = 0, \cdots, \frac{N}{2} - 1. \qquad (2.11)$$

Con los casos especiales:

1. Para $i = 0$
$$s[0] = l[0] + \left\lfloor \frac{h[0]}{2} + \frac{1}{2} \right\rfloor. \qquad (2.12)$$

2. Si N es impar entonces
$$s[N-1] = l[0] + \left\lfloor \frac{h[i-1]}{2} + \frac{1}{2} \right\rfloor. \qquad (2.13)$$

2.4. La transformada 13/7-T

La transformada 13/7-T también llamada transformada de interpolación (4,4) es la versión entera de la transformada 13/7 que es en punto flotante. La transformada 13/7-T incrementa a cuatro el número de puntos usados para predecir una muestra y esto produce unas aproximaciones más suaves. Los filtros de análisis de la transformada son los siguientes

$$\begin{aligned} h[i] &= s[2i+1] - \lfloor \tfrac{9}{16}(s[2i] + s[2i+2]) - \tfrac{1}{16}(s[2i-2] + s[2i+4]) + \tfrac{1}{2} \rfloor \\ l[i] &= s[2i] + \lfloor \tfrac{9}{32}(h[i-1] + h[i]) - \tfrac{1}{32}(h[i-2] + h[i+1]) + \tfrac{1}{2} \rfloor \\ & \text{con } i = 0, \ldots, N/2-1. \end{aligned} \tag{2.14}$$

De forma similar, la transformada 13/7-T inversa se obtiene aplicando

$$\begin{aligned} s[2i] &= l[i] - \lfloor \tfrac{9}{32}(h[i-1] + h[i]) - \tfrac{1}{32}(h[i-2] + h[i+1]) + \tfrac{1}{2} \rfloor \\ s[2i+1] &= h[i] + \lfloor \tfrac{9}{16}(s[2i] + s[2i+2]) - \tfrac{1}{16}(s[2i-2] + s[2i+4]) + \tfrac{1}{2} \rfloor \\ & \text{con } i = 0, \ldots, N/2-1. \end{aligned} \tag{2.15}$$

La transformada utilizada en el desarrollo del proyecto es la transformada 13/7-T, junto con la 5/3 y la ST cuando es necesario predecir algunas muestras cercanas a los límites de la señal.

2.5. La transformada wavelet discreta 3D

La transformada wavelet es separable. La transformada de más de una dimensión se obtiene aplicando la transformada unidimensional a cada una de las dimensiones del GOP. El filtro se aplica recursivamente sobre la banda de baja frecuencia tantas veces como se pueda en función del tamaño de cada una de las dimensiones del volumen. La Figura 2.1 muestra el cálculo de la DWT-3D.

Para transmitir los coeficientes de máxima potencia es necesario transmitir los coeficientes de bajas frecuencias antes que los de altas frecuencias a igualdad de potencia. De esta forma, se tiende a enviar primero los coeficientes situados en la esquina superior izquierda que son los que representan a las bandas de baja frecuencia. Para ello, las distintas bandas se ponderan por un factor, consiguiendo que la transformada sea unitaria (una transformación unitaria posee la propiedad de que la norma Euclídea es invariante,

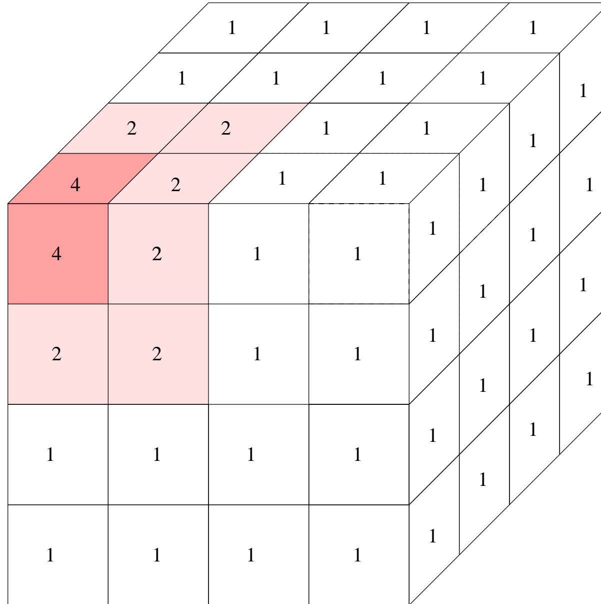

Figura 2.1: Cálculo de la DWT-3D. En este ejemplo se han aplicado dos niveles de descomposición.

esto es, una medida de distorsión efectuada sobre el dominio transformado es equivalente a realizarla sobre el dominio original[8]) y en consecuencia, ahora se cumple que el MSE (entre la imagen original y la imagen transmitida) se minimiza con la transmisión de los coeficientes atendiendo a su magnitud.

También indicar que la representación de los coeficientes se hace con aritmética entera, lo que permite que la compresión y descompresión de cualquier señal sea totalmente reversible. Los coeficientes más importantes (visualmente hablando) son los que tienen mayor potencia, pues contienen mayor cantidad de información. Como ya hemos indicado, dichos coeficientes tienden a agruparse sobre las bandas de baja frecuencia, que son los que primero se transmiten.

2.6. Compresión de los planos de bits

La transmisión incremental de imágenes puede ser mejorada si en lugar de enviar los coeficientes completos se transmiten por planos de bits. Si p es la posición que ocupa el bit más significativo de los coeficientes más

grandes, la idea es transmitir primero todos los bits que están en la posición p de todos los coeficientes. A continuación se envían todos los bits que están en la posición $p-1$ y así sucesivamente hasta enviar todos los planos. Éste es un razonamiento lógico si se tiene en cuenta que es más importante indicar al receptor los bit más significativos de los coeficientes menores que los bits menos significativos de los coeficientes mayores.

La clave para determinar un método efectivo de codificación de los planos de bits está en darse cuenta de que existe una cierta dependencia entre los coeficientes dentro del espacio wavelet. Esta correlación entre coeficientes puede expresarse diciendo que si un coeficiente es significativo en una octava de frecuencia[2] inferior, entonces los 8 coeficientes que ocupan la misma posición relativa dentro de la siguiente octava tienden a ser significativos, y viceversa.

La relación espacio-temporal exacta entre un coeficiente de coordenadas[3] (i, j, k) y sus ocho hijos (si es que existen), es que estos se encuentran en las coordenadas absolutas

$$\begin{aligned}\mathcal{O}(i,j,k) = \big\{&(2i,2j,2k),(2i,2j+1,2k),(2i+1,2j,2k),\\&(2i+1,2j+1,2k),(2i,2j,2k+1),(2i+1,2j,2k+1),\\&(2i,2j+1,2k+1),(2i+1,2j+1,2k+1)\big\},\end{aligned} \tag{2.16}$$

y esta relación se propaga de forma recursiva entre todos los coeficientes wavelet. Por lo tanto, un nodo (i, j, k) del árbol tiene 8 hijos agrupados en bloques de $2 \times 2 \times 2$ o no tiene ninguno, lo que ocurre en el nivel más bajo de la descomposición. La Figura 2.2 muestra un ejemplo de la dependencia espacio-temporal de los coeficientes en el nivel más alto. A cada árbol diseñado usando la Expresión 2.16 se le llama "árbol de orientación espacial" (AOE).

A todos los descendientes de (i, j, k) (hijos, nietos, etc.) los denotaremos por $\mathcal{D}(i, j, k)$. El número total de descendientes es 0 o una potencia de 8 cuyo exponente depende del nivel de la descomposición wavelet. Nótese que $\mathcal{D}(i, j, k) + (i, j, k)$ forman un AOE completo. Cuando un AOE está formado sólo por ceros decimos que se trata de un zerotree.

La relación estadística entre un elemento (i, j, k) y sus descendientes $\mathcal{D}(i, j, k)$ es que si (i, j, k) es 1 entonces al menos uno de sus descendientes es probablemente 1. Por el contrario, si (i, j, k) es 0, lo más probable es que

[2]Modelo de descomposición en bandas de frecuencia donde cada banda tiene el doble de coeficientes espectrales que la de frecuencia inmediatamente menor.

[3]Las coordenadas se expresan como (fila, columna, tiempo).

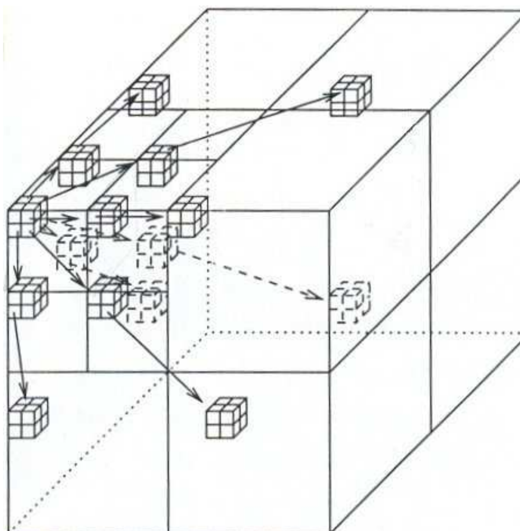

Figura 2.2: Algunos ejemplos de árboles creados en la DWT-3D.

todos sus descendientes sean 0. Esta es una de las principales razones por la que podemos diseñar compresores a partir del espacio wavelet.

2.7. El algoritmo SPIHT-3D

Una vez presentada el tipo de redundancia que es posible explotar en el dominio wavelet cuando los coeficientes son transmitidos por planos de bits, vamos a analizar uno de los compresores progresivos más eficientes que se conocen: SPIHT (Set Partitioning In Hierarchical Trees) [6, 8], en su implementación en tres dimensiones.

SPIHT fue ideado por Said y Pearlman en 1996 y extendido a su versión en tres dimensiones por Chen y Pearlman ese mismo año. Muestra un efectivo y computacionalmente simple codec de vídeo sin compensación de movimiento que obtiene unos excelentes resultados numéricos y visuales.

El algoritmo tiene las siguientes características:

- La información visualmente más importante es transmitida primero.
- Explota la redundancia entre las diferentes escalas espacio-temporales.
- El code-stream está comprimido y es completamente seccionable proporcionando calidad progresiva de forma que permite al algoritmo ser

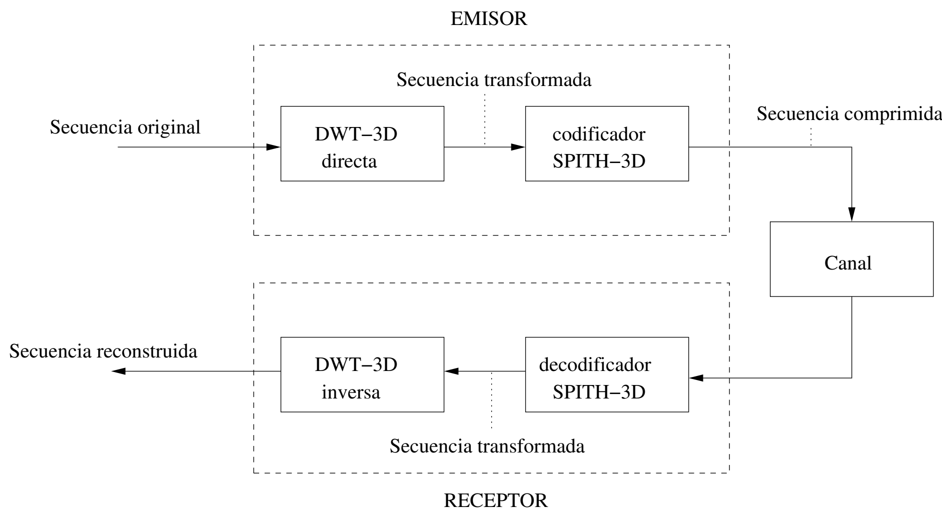

Figura 2.3: Muestra el esquema de un PVC que utiliza el algoritmo SPIHT-3D.

detenido en cualquier instante ó hasta que se obtenga una reconstrucción sin perdida.

- Permite la visualización a diferentes frames por segundo y con distintos tamaños de frame usando el mismo code-stream codificado gracias a la transformada wavelet.

La Figura 2.3 muestra el diagrama de un codec de vídeo que utiliza el algoritmo SPIHT-3D. Está formado por los siguientes componentes:

- DWT-3D directa: A este bloque llega la secuencia original a la que se le aplica la transformada directa obteniendo de esta forma la secuencia transformada.

- Codificador SPIHT-3D: Una vez que se genera la secuencia transformada, la codificamos usando el algoritmo SPIHT-3D. De esta forma obtenemos una secuencia transformada y comprimida de forma óptima para enviar al canal de comunicación. Esta secuencia cumple con las características del algoritmo.

- Descodificador SPIHT-3D: Desde el canal va llegando la secuencia transformada y comprimida, se le aplica la descodificación y obtenemos la secuencia transformada.

- DWT-3D inversa: A la secuencia transformada se le aplica la transformada inversa obteniendo la secuencia original o una aproximación en función de que haya procesado la secuencia completa o una parte de ella.

El codificador SPIHT-3D codifica eficientemente los volúmenes de significancia, aprovechando la similaridad o semejanza que existe entre las diferentes octavas.

Lo primero que hace es transmitir el plano de bits más significativo (los coeficientes se tratan en forma signo-magnitud). Después entra en un bucle que itera tantas veces como planos de bits tienen que transmitirse. En la emisión de cada plano de bits se diferencia entre los bits de significancia (indican que coeficientes comienzan a ser significativos en el volumen actualmente transmitido – que son comprimidos) y los bits de refinamiento (están formados por los bits de los coeficientes que ya son significativos en un plano superior, afinando el valor real del coeficiente wavelet – que no son comprimidos). Los bits de signo son también transmitidos sin comprimir.

SPIHT-3D trabaja particionando los AOEs de forma que tiende a mantener coeficientes no significativos en grandes conjuntos y comunica con un único bit si alguno de los elementos de un conjunto es significativo o no (es un zerotree).

Las decisiones de particionamiento son decisiones binarias, que son transmitidas al descodificador e indican los planos de significancia. Por lo tanto, SPIHT-3D en lugar de transmitir las coordenadas de los coeficientes significativos en el plano actual, transmite los resultados de las comparaciones que han provocado la determinación (por parte del codificador) de todos los zerotrees que forman dicho plano.

El descodificador ejecuta exactamente el mismo algoritmo que el codificador y como no dispone de los coeficientes wavelet para saber si son significativos o no, usa los resultados de las comparaciones que le llegan en el code-stream. De esta forma la traza de instrucciones es idéntica.

SPIHT-3D transmite los planos de significancia y de refinamiento en dos fases independientes llamadas de ordenación y de refinamiento, respectivamente. El nombre de la fase de refinamiento es obvio. Sin embargo la fase de ordenación se llama así porque lo que el codificador hace es ordenar los coeficientes atendiendo a su valor absoluto y luego enviarlos según ha resultado dicho orden. Pero nótese que la ordenación que se produce es muy suave ya que no es necesario ordenar todos los coeficientes que son significativos en el plano que se transmite, sólo debemos encontrar qué coeficientes van antes que otros porque son significativos en un plano superior.

La fase de ordenación

Como ya hemos indicado, SPIHT-3D es eficiente porque realiza un número mínimo de comparaciones casi equiprobables. La clave está en saber cómo formular dichas comparaciones, que se construyen usando un algoritmo de particionamiento de los AOEs.

El codificador y el descodificador manejan conceptualmente dos listas de coeficientes representados por sus coordenadas espaciales. En una se almacenan todos los coeficientes que son significativos en el plano actual de transmisión p. Todos estos coeficientes c verifican que

$$|c| \geq 2^p. \tag{2.17}$$

La otra lista almacena el resto de coeficientes que no son significativos. Inicialmente esta lista contiene todos los coeficientes (tantos como puntos existen en el GOP) y la lista de coeficientes significativos está vacía. Notaremos con LIC (List of Insignificant Coefficients) a la lista de coeficientes no significativos y con LSC (List of Significant Coefficients) a la de coeficientes significativos.

SPIHT-3D realiza una partición inicial

$$\begin{gathered}(0,0,0),(0,1,0),(1,0,0),(1,1,0),(0,0,1),\\(1,0,1),(0,1,1),(1,1,1),\mathcal{D}(0,1,0),\mathcal{D}(1,0,0),\\\mathcal{D}(1,1,0),\mathcal{D}(0,0,1),\mathcal{D}(1,0,1),\mathcal{D}(0,1,1),\mathcal{D}(1,1,1),\end{gathered} \tag{2.18}$$

donde como ya sabemos, $\mathcal{D}(i,j,k)$ representa a todos los coeficientes descendientes de (i,j,k) que se determinan aplicando la Ecuación (2.16) recursivamente.

SPIHT-3D averigua qué elementos de esta partición son significativos. Más formalmente, evalúa la función

$$S_p(\mathcal{T}) = \begin{cases} 1 & \text{si algún coeficiente } |c| \in \mathcal{T} \geq 2^p \\ 0 & \text{en caso contrario,} \end{cases} \tag{2.19}$$

donde $\mathcal{T}$ puede ser un único coeficiente o un conjunto de coeficientes.

En la codificación del primer plano de bits, las 8 raíces

$$\big\{(0,0,0),(0,1,0),(1,0,0),(1,1,0),(0,0,1),(1,0,1),(0,1,1),(1,1,1)\big\}$$

tienen un 50 % de posibilidades de ser significativas y a sus descendientes

$$\big\{\mathcal{D}(0,1,0),\mathcal{D}(1,0,0),\mathcal{D}(1,1,0),\mathcal{D}(0,0,1),\mathcal{D}(1,0,1),\mathcal{D}(0,1,1),\mathcal{D}(1,1,1)\big\}$$

($(0,0,0)$ no tiene descendientes) les ocurre lo mismo; tienen una probabilidad igual a 0,5 (aproximadamente) de ser un zerotree. SPIHT-3D realiza todas estas comparaciones y emite los bits de código correspondientes.

Para gestionar las particiones, SPIHT-3D usa realmente 3 listas: LIC, LSC y LIS (List of Insignificant Sets) o lista de conjuntos no significativos, porque es una forma sencilla de distinguir entre coeficientes y conjuntos de coeficientes. Por lo tanto, el contenido inicial de dichas listas es:

$$\begin{array}{rcl} \text{LSC} & \leftarrow & \varnothing \\ \text{LIC} & \leftarrow & \big\{(0,0,0),(0,1,0),(1,0,0),(1,1,0),(0,0,1),(1,0,1),(0,1,1),(1,1,1)\big\} \\ \text{LIS} & \leftarrow & \big\{(0,1,0),(1,0,0),(1,1,0),(0,0,1),(1,0,1),(0,1,1),(1,1,1)\big\} \end{array}$$

Cuando un coeficiente de LIC no es significativo, no ocurre nada en las listas, pero si lo es, se mueve desde LIC a LSC, para posteriormente ser refinado. De forma similar, si un coeficiente de LIS no es significativo (es un zerotree) no ocurre nada en las listas, pero si es significativo, debe ser particionado en subconjuntos que tengan tantas posibilidades de ser zerotrees como de no serlo.

SPIHT-3D particiona un $\mathcal{D}(i,j,k)$ en

$$\big\{(k,l,m) \in \mathcal{O}(i,j,k), \mathcal{L}(i,j,k)\big\},$$

donde

$$\mathcal{L}(i,j,k) = \mathcal{D}(i,j,k) - \mathcal{O}(i,j,k).$$

Cada $\mathcal{D}(i,j,k)$ se descompone en 9 partes: los 8 nodos hijo de (i,j,k) y el resto de descendientes.

Los $(k,l,m) \in \mathcal{O}(i,j,k)$ se insertan en LIC o en LSC dependiendo de si son significativos o no. En el caso de viajar a LIC la inserción debe realizarse al final de la lista para que los coeficientes sean evaluados en la pasada actual. Los $\mathcal{L}(i,j,k)$ se insertan en LIS para ser más tarde evaluados.

En LIS podemos encontrar, por tanto, 2 tipos de conjuntos: $\mathcal{D}(i,j,k)$ y $\mathcal{L}(i,j,k)$, que tienen un número diferente de elementos. SPIHT-3D los diferencia diciendo que los $\mathcal{D}(i,j,k)$ son de tipo A mientras que los $\mathcal{L}(i,j,k)$ son de tipo B[4].

El particionado de un $\mathcal{L}(i,j,k)$ es distinto, por tanto, al de un $\mathcal{D}(i,j,k)$. Si un $\mathcal{L}(i,j,k)$ es significativo entonces se particiona en 8 conjuntos

$$\big\{\mathcal{D}(k,l,m) \in \mathcal{O}(i,j,k)\big\},$$

[4]Recuérdese que tanto los AOEs como los coeficientes individuales se representan por una terna de coordenadas.

es decir, en los 8 árboles hijos de (i, j, k) que se vuelven a insertar al final de LIS. Finalmente $\mathcal{L}(i, j, k)$ desaparece de LIS porque este AOE ha sido particionado en sus 8 subárboles.

La fase de refinamiento

Entre cada fase de ordenación se realiza otra de refinamiento. Si p es el plano de bits a transmitir, en esta fase se emite el p-ésimo bit más significativo de cada coeficiente almacenado en LSC. Cuando un coeficiente ha sido totalmente enviado (recuérdese que 7/8 de los coeficientes están ponderados por una potencia de 2 y por tanto sus bits menos significativos son cero forzosamente) se elimina de LSC.

En este punto existen dos alternativas de implementación. Si O_p representa los bits emitidos durante la fase de ordenación de la capa p y R_p a los bits de refinamiento, una posibilidad de construcción del code-stream es

$$O_p R_p O_{p-1} R_{p-1} O_{p-2} \cdots . \tag{2.20}$$

Sin embargo, los autores del algoritmo recomiendan que los bits de ordenación del plano $p-1$ antecedan a los bits de refinamiento del plano p, es decir

$$O_p O_{p-1} R_p O_{p-2} R_{p-1} \cdots . \tag{2.21}$$

Este procedimiento tiene sentido porque de esta forma primero se envían los bits que definen nuevos coeficientes distintos de 0 y a continuación se refinan los coeficientes que ya eran significativos.

Las dos alternativas han sido evaluadas y sus rendimientos son muy similares [5]. En la exposición del pseudo-código que expone el codec, por sencillez, se usará la primera opción a la que llamaremos genéricamente "refinar antes" frente a la segunda que será referenciada por "refinar después".

La fase de cuantificación

La fase de cuantificación se usa para decrementar el umbral de significancia que en SPIHT-3D son siempre potencias de dos. De esta forma se seleccionan los planos de bits de los coeficientes wavelet en el orden correcto.

El codificador

Se presenta sólo el algoritmo del compresor ya que el descompresor es prácticamente idéntico.

1. **Fase de inicialización**.
   a) Emitir el índice del plano más significativo. Sea este valor p.
   b) LSC $\leftarrow \varnothing$.
   c) LIC $\leftarrow \big\{(0,0,0),(0,1,0),(1,0,0),(1,1,0),(0,0,1),(1,0,1),(0,1,1),(1,1,1)\big\}$.
   d) LIS $\leftarrow \big\{(0,1,0),(1,0,0),(1,1,0),(0,0,1),(1,0,1),(0,1,1),(1,1,1)\big\}$.
2. Mientras $p \geq 0$:
   a) **Fase de refinamiento**.
      1) Para cada $(i,j,k) \in$ LSC:
         a' Emitir el p-ésimo bit del coeficiente (i,j,k).
         b' Si (i,j,k) ha sido totalmente enviado borrarlo de LSC.
   b) **Fase de ordenación**.
      1) Para cada $(i,j,k) \in$ LIC:
         a' Emitir $S_p(i,j,k)$.
         b' Si $S_p(i,j,k) = 1$ entonces:
            - Mover (i,j,k) desde LIC a LSC.
            - Emitir el signo de (i,j,k).
      2) Para cada $(i,j,k) \in$ LIS:
         a' Si (i,j,k) es de tipo A entonces:
            - Emitir $S_p(\mathcal{D}(i,j,k))$.
            - Si $S_p(\mathcal{D}(i,j,k)) = 1$ entonces:
               - Para cada $(k,l,m) \in \mathcal{O}(i,j,k)$:
                  ⋆ Emitir $S_p(k,l,m)$.
                  ⋆ Si $S_p(k,l,m) = 1$ entonces:
                     · Añadir (k,l,m) a LSC.
                     · Emitir el signo de (k,l,m).
                  ⋆ Si no:
                     · Añadir (k,l,m) al final de LIC.
               - Si $\mathcal{L}(i,j,k) \neq \varnothing$ (tiene al menos nietos) entonces:
                  ⋆ Mover (i,j,k) al final de LIS como de tipo B.
               - Si no:
                  ⋆ Borrar (i,j,k) de LIS.
         b' Si no (es de tipo B):
            - Emitir $S_p(\mathcal{L}(i,j,k))$ (si alguno de los nietos de (i,j,k) es significativo).

- Si $S_p(\mathcal{L}(i,j,k)) = 1$ entonces:
  - Añadir cada hijo $(k,l,m) \in \mathcal{O}(i,j,k)$ al final de LIS como de tipo A.
  - Borrar (i,j,k) de LIS.

c) **Fase de cuantificación**.

1) $p \leftarrow p - 1$.

El descodificador

Para encontrar el algoritmo de descodificación debe tenerse presente que todas las instrucciones de salto condicional están controladas por el valor devuelto por $S_p(\cdot)$ (ver Ecuación (2.19)), los cuales forman el code-stream generado durante la fase de ordenación y que por lo tanto son conocidos por el descodificador. Éste puede realizar exactamente la misma traza de instrucciones si el algoritmo que ejecuta es idéntico al del codificador excepto porque donde aparece "emitir" ahora debe poner "recibir". Ya que se trata del mismo algoritmo, las tres listas LSC, LIC y LIS van a generarse exactamente de la misma forma a como se generaron en el codificador, gracias a lo cual, la reconstrucción del plano es trivial y además, las complejidades del codificador y del descodificador son idénticas.

Sin embargo, si SPIHT-3D va a ser usado como transmisor progresivo, el descodificador debe realizar una tarea extra para ajustar los coeficientes reconstruidos a partir del intervalo de incertidumbre durante la fase de ordenación. Cuando una coordenada se mueve a LSC, se sabe que el valor del coeficiente c cumple que

$$2^p \leq |c| < 2^{p+1}, \tag{2.22}$$

donde p es el plano de bit transmitido. El descodificador utiliza esta información (más el bit de signo que llega a continuación), para ajustar el valor reconstruido a la mitad del intervalo $[2^p, 2^{p+1} - 1]$ ya que de esta forma el error con respecto al valor real del coeficiente será la mitad en promedio. Por ejemplo, si $p = 15$, se sabe que el coeficiente es mayor que 32768 y menor que 65535. Si sólo hacemos que el bit 15 sea 1 estaremos reconstruyendo el coeficiente con el valor mínimo que puede realmente tener. La solución es hacer

$$|\hat{c}| = 1{,}5 \times 2^p = 3 \times 2^{p-1}, \tag{2.23}$$

que es el valor que divide al intervalo de incertidumbre en dos mitades iguales. En nuestro ejemplo, el valor de reconstrucción sería $3 \times 2^{15-1} = 49152$.

De forma similar, durante la fase de refinamiento, cuando el descodificador conoce el valor real del bit $(p-1)$-ésimo, resulta otro intervalo de incertidumbre. Si el bit recibido es un 1, entonces el bit $(p-1)$-ésimo ya es correcto, pero debemos hacer el bit $(p-2)$-ésimo igual a 1 para ajustar a la mitad del nuevo intervalo de incertidumbre. Si el bit recibido es un 0, entonces el bit $(p-1)$-ésimo está equivocado y debe ser puesto a 0. El bit $(p-2)$-ésimo debe hacerse 1 para ajustar el valor reconstruido a la mitad del intervalo de incertidumbre. Por lo tanto, en cualquier caso, cuando estamos refinando, colocamos el bit recibido a su valor adecuado y el siguiente bit menos significativo se hace 1.

Capítulo 3

El sistema desarrollado

Este capítulo explica los pasos a seguir para conseguir obtener un software cliente-servidor de vídeo progresivo. Los requerimientos exigidos a la hora de implementarlo han sido:

1. Que se adapte al ancho de banda de la red.
2. Que el sistema se adapte a la carga de trabajo de la CPU de la máquina cliente donde se ejecuta la aplicación.
3. Que proporcione escalabilidad en calidad.

Las características del algoritmo SPIHT hacen que los anteriores requisitos puedan llevarse a cabo. Esto es debido a que el sistema no es dependiente de la cantidad de información recibida en un instante dado (que depende del ancho de banda disponible). Además, el proceso de descodificación del algoritmo puede ser abandonado en cualquier instante lo que hace que el software se adapte a máquinas con distinta potencia de cálculo.

El sistema por tanto está formado por un software servidor que se ejecuta en una máquina remota, donde están almacenados los vídeos comprimidos y un software cliente, que se comunica con el software servidor, procesa la información recibida y la visualiza en la pantalla.

La idea general del software cliente se basa en un pipeline de tres etapas donde el flujo del programa avanza en el tiempo. Las tres etapas generales son:

- Etapa 1: Lectura (L) de los datos y descodificación (D).
- Etapa 2: Transformación inversa del volumen (T).

	Tiempo 1	Tiempo 2	Tiempo 3	Tiempo 4	
Etapa 1	L/D GOP1	L/D GOP2	L/D GOP3	L/D GOP4	...
Etapa 2		T GOP1	T GOP2	T GOP3	T GOP4
Etapa 3			V GOP1	V GOP2	V GOP3

Figura 3.1: Estructura de las etapas del cauce.

- Etapa 3: Visualización de las imágenes (V).

siendo su representación gráfica la que se muestra en la Figura 3.1.

Por lo tanto cuando todas las etapas del pipeline tienen datos se trabaja con información de tres slots de tiempo distintos, es decir, se tiene una latencia de dos slots. Al tercer slot de tiempo comienza la visualización del vídeo en la pantalla.

3.1. Formato de la información a transmitir

El formato del vídeo comprimido tiene una serie de características que vamos a describir en este apartado.

Para generar un vídeo progresivo, que el software desarrollado sea capaz de visualizar, es necesario tener un vídeo en formato RAW (vídeo que no tiene ningún tipo de compresión, es decir, que todos los fotogramas del vídeo están representados mediante matrices bidimensionales de puntos). Con una WebCam o videocámara digital se puede obtener una secuencia de vídeo con estas características.

Una vez que se tiene el vídeo en formato RAW, se comprime con un PVC[8] que nos genera un fichero con un formato especial. Este tiene una propiedad muy importante y es que si descomprimimos solo una parte del code-stream y lo volvemos a comprimir, generamos exactamente el mismo code-stream. Esto implica necesariamente que el algoritmo es sin pérdida y que es progresivo.

La representación gráfica del formato del fichero comprimido se muestra en la Figura 3.2.

Como se puede observar en la Figura 3.2 al comienzo del fichero tenemos la información relativa al número de fotogramas o frames, la altura y la anchura de que se compone cada GOP que se va a ir descomprimiendo en cada intervalo de tiempo.

La información que se envía de cada GOP es un conjunto de coeficientes

Niveles	Fotogramas	Filas	Columnas	Tamaño GOP 1	GOP 1	Tamaño GOP 2	GOP 2	Tamaño GOP 3	GOP 3	• • •	Tamaño GOP N	GOP N

Figura 3.2: Formato del fichero comprimido con SPIHT-3D.

wavelet que una vez descodificados se utilizarán para reconstruir el GOP original.

3.2. El servidor de vídeo progresivo

En este apartado nos vamos a centrar en la parte servidora, la que envía el vídeo por el canal a los clientes que lo soliciten.

La comunicación entre el cliente y el servidor se realiza utilizando sockets de tipo TCP. Este tipo de socket asegura la transmisión sin errores ya que realiza el control de flujo y errores mediante retransmisión.

El comportamiento es bastante simple. Una vez que el servidor se lanza, se queda a la escucha de peticiones. Cuando el cliente realiza la petición de conexión, el servidor da conformidad y recibe entonces el nombre del fichero de vídeo que tiene que comenzar a transmitir, lo busca en su árbol de directorios y envía conformidad en caso de existencia o fallo en caso negativo. Si existe el fichero solicitado y una vez enviada conformidad, se queda a la escucha de peticiones de envío de información.

El servidor es por tanto, un programa que hace lo que el cliente le ordena, es decir, es el cliente quien realmente decide qué es lo que el servidor le tiene que enviar y cuándo se lo tiene que enviar. Un ejemplo de interacción entre el servidor y el cliente sería el mostrado en la Figura 3.3.

Aunque el vídeo esté comprimido, este tiene un tamaño bastante grande (la compresión es sin pérdida), por lo que es muy difícil que sea necesario enviar completamente cada GOP. Ésto sólo ocurre si es preciso una representación exacta del vídeo original.

Es el cliente el que determina la cantidad de información del GOP que precisa, dependiendo de los requerimientos a los que esté sometido. Por lo tanto, el servidor tiene que posicionarse al comienzo de cada GOP en cada petición. La forma en que se posiciona es bastante rápida ya una de las partes definidas en el formato del vídeo es el tamaño de cada GOP. Así, si

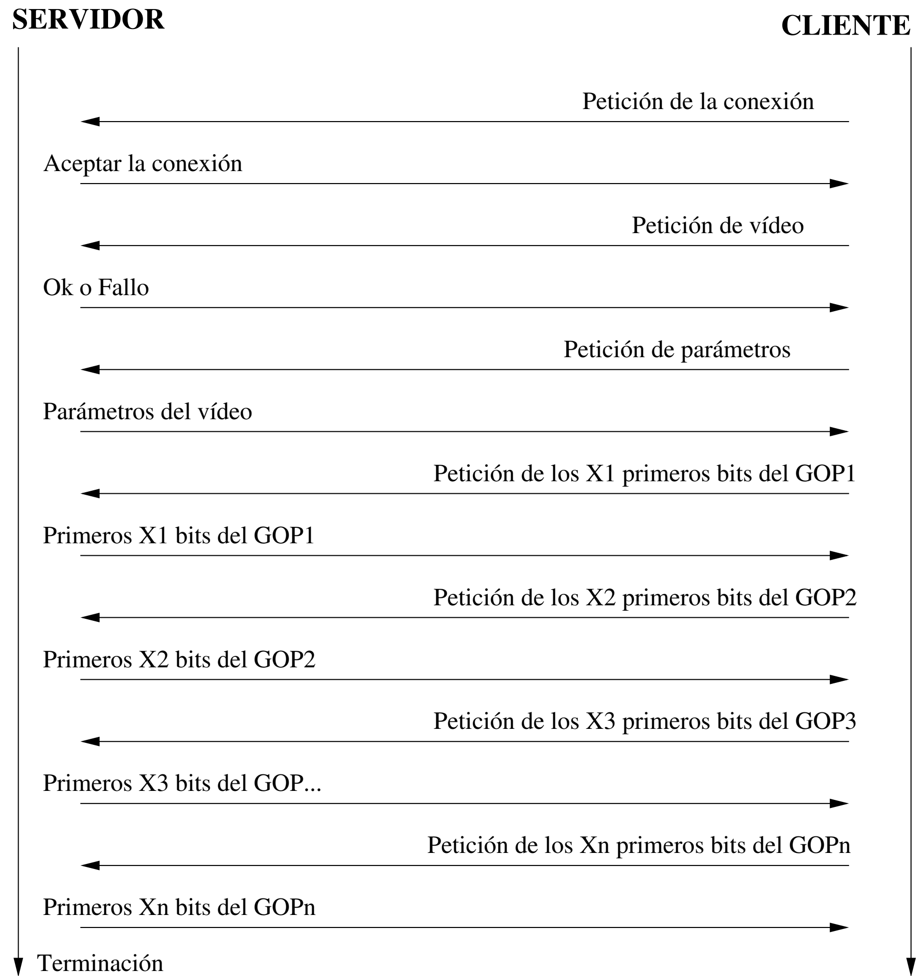

Figura 3.3: Un ejemplo de interacción entre el servidor y el cliente.

el cliente nos pide X bits de información y el tamaño completo es N bits, el servidor tendrá que avanzar $N - X$ bits.

3.3. El cliente de vídeo progresivo

Este apartado va a describir el software cliente que se encarga de recibir la información del canal, procesarla y visualizarla por la pantalla.

El cliente realiza una serie de operaciones que son necesarias para poder visualizar el contenido de cada GOP. Estas operaciones están divididas en cuatro tareas fundamentales, que a grandes rasgos son:

1. Lectura del socket: Este proceso lee del canal un code-stream de bytes (GOP codificado en la parte servidora) y lo irá almacenando en memoria para su posterior procesado.
2. Descodificación del code-stream: La descodificación tiene como entrada la información leída del socket que está comprimida y genera como salida un volumen de coeficientes wavelet.
3. Transformada inversa (DWT^{-1}): Este proceso tiene como entrada un volumen de coeficientes wavelet y genera como salida un volumen de puntos.
4. Visualización del GOP: Se encarga de representar la secuencia de imágenes.

3.3.1. Implementación del cliente de vídeo progresivo

Este apartado presenta la forma en que se han organizado las tareas, las estructuras de datos y técnicas de programación utilizadas en la implementación del cliente de vídeo progresivo. Una vez mostradas se describe el funcionamiento del software.

Las estructuras y técnicas más relevantes del cliente son:

- Organización de las tareas: Las tareas de descodificación y transformación inversa están englobadas en una misma función colocadas una detrás de otra de forma secuencial. El motivo es que son funciones intensivas en cálculo y se mejora el rendimiento si se mantienen juntas. Las funciones de lectura del socket y visualización de la secuencia de imágenes se realizan en procedimientos independientes. Los procesos de lectura del socket y visualización no necesitan potencia de cálculo,

por lo que la mayoría del tiempo la CPU estará ocupada realizando los procesos de descodificación y de transformación inversa.

- Señales del sistema operativo: Como debemos realizar las tareas anteriormente mencionadas para cada GOP que se recibe, se necesita un mecanismo que nos permita realizar procesos de forma simultánea para información relativa a GOPs distintos, según se vio en la Figura 3.1. Para ejecutar estos procesos de forma simultánea se utilizan las señales del sistema operativo. En nuestro sistema se utiliza una señal que ejecuta una interrupción un número determinado de veces. La cantidad de interrupciones es función del número de fotogramas del GOP. Es decir, que si nuestro GOP está formado por 16 fotogramas, la interrupción se ejecuta 16 veces durante el procesamiento completo del GOP. En cada ejecución de la interrupción se realiza la visualización de un fotograma del GOP y se lee del socket la información que hubiera.

- Tipo de socket: El sistema utiliza para las lecturas del canal socket de tipo "no bloqueantes". Su principal característica es que si al hacer una lectura sobre ellos no existen datos se retorna al punto de llamada sin ningún tipo de demora.

- Buffer de lectura: Como la tasa de bits disponible en la red no es constante a lo largo del tiempo sino que puede sufrir ciertos cambios, es posible que en un momento dado no se reciba información para procesar, o que la información llegue demasiado tarde para que sea útil en la ejecución. Para evitar estos problemas, la información leída del socket se almacena en un buffer de lectura de M posiciones. Cada posición del buffer alberga información de GOP de tiempos distintos. De esta forma se protege el sistema de los posibles altibajos de la red. Se aprovecha mejor el canal de comunicación debido a que no es necesario descartar información que pudiese llegar fuera de tiempo. Esta mejor utilización del canal, repercute positivamente en la calidad de visualización. Además de esta forma se consigue que la lectura del socket sea un proceso totalmente independiente del proceso de cálculo (descodificación y transformación inversa).

- Vectores de control de información: Para saber en todo momento la cantidad de información que vamos a obtener de la red y a qué posición del buffer corresponde se tienen unos vectores que se utilizan para esta función. De esta forma cuando se realiza una lectura del socket se co-

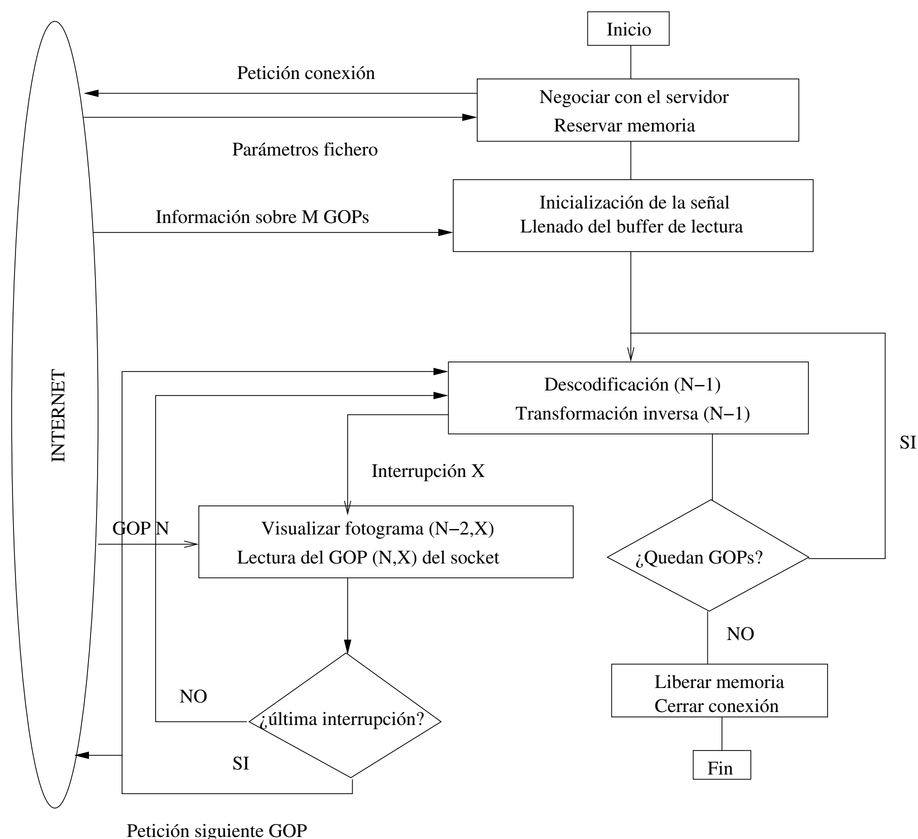

Figura 3.4: Representación gráfica del cliente de vídeo progresivo.

noce la posición del buffer de lectura a la que pertenece la información leída.

- Buffer de proceso: A la hora de procesar la información es necesario que la salida de un proceso sea la entrada del siguiente y así sucesivamente. Por lo tanto, para poder trabajar con slots de tiempo distinto en las funciones de descodificación, transformación inversa y visualización, evitando escrituras sobre información útil, se utilizan buffers con un determinado tamaño que nos permitan leer y escribir en los distintos procesos sin necesidad de hacer copia de la información procesada.

La Figura 3.4 muestra la representación gráfica de la implementación del cliente de vídeo progresivo. La implementación se comporta de la siguiente

forma: Una vez que el servidor está en ejecución, los clientes pueden conectarse al mismo. Para ello se negocia la aceptación de la conexión y la existencia del fichero de vídeo solicitado. Una vez comprobado que el fichero existe, el servidor envía al cliente los parámetros del fichero. Estos parámetros se utilizan para crear las estructuras de memoria necesarias para el buffer de lectura, los buffers de proceso, etc.

Una vez reservado espacio en memoria, se llena el buffer de lectura con M GOPs de información. Al disponer del buffer de lectura, el sistema puede estar M slots de tiempo sin recibir información del socket que no se presenta ningún problema en la visualización.

Se inicializa la interrupción y a partir de este momento cada cierto tiempo se ejecuta un procedimiento en el cual se visualiza un fotograma del GOP que se está representando, y se sondea el socket para comprobar si existen datos que almacenar en el buffer de lectura. Este proceso es inmediato y no consume tiempo de CPU. La mayor parte del tiempo de CPU se están realizando los procesos de descodificación y de transformación inversa.

Cuando se alcanza la última interrupción se realiza la petición de información de un nuevo GOP al servidor y se desplazan las estructuras de memoria para repetir el proceso completo.

Este proceso se repite hasta alcanzar el último GOP de la secuencia de vídeo pedida. Una vez terminada la visualización, se liberan todos los recursos reservados y se cierra la conexión con el servidor.

El sistema presenta un problema y es que, la función de descodificación y transformación inversa se realiza sobre el total de información recibida de la red, es decir, que si en un instante dado se tienen Y bytes de información sobre un GOP, se procesan Y bytes. Hasta que no se termina de procesar completamente dicha cantidad de datos no se pasa el resultado a la fase de visualización. En un caso normal, este tiempo de procesamiento es inferior a lo que se tarda en representar un volumen, pero existen casos donde esto no se cumple.

Por ejemplo, si nuestra CPU no es muy potente y se procesa mucha información en un cierto tiempo, es posible que no termine dicho proceso antes de que la fase de visualización represente el GOP completo. Otro caso ocurre cuando la CPU está procesando una cantidad de información de forma estable y un programa externo comienza a ejecutarse. La CPU tiene que atender a varios programas que se están ejecutando en paralelo y entonces su capacidad de procesamiento en nuestra aplicación deja de ser suficiente para procesar el volumen de información.

Estudiando el comportamiento de los algoritmos de descodificación y transformación inversa deducimos que:

- El algoritmo que realiza la transforma inversa wavelet es un proceso que no se puede abandonar[1] sin que se haya procesado completamente todo el volumen.

- El algoritmo que realiza la descodificación usando SPIHT-3D es un proceso que sí se puede abandonar en cualquier momento. Esto afecta a la cantidad de información que le haya dado tiempo a descodificar. En cualquier caso, el volumen obtenido es válido para que se le realice la transformada inversa.

Partiendo de estas premisas y para solucionar el problema anteriormente mencionado, modificamos la fase de procesamiento invirtiendo el orden de las funciones de descodificación y transformación inversa. Sabiendo que la función de transformación inversa tiene un tiempo de proceso constante y la función de descodificación tiene un tiempo de proceso que varía en función de la cantidad de información que se tenga que descodificar, haremos que primero se procese la transformación inversa de forma completa y posteriormente la descodificación con el tiempo restante. Cuando el tiempo disponible para realizar ambas funciones esté a punto de finalizar, abandonaremos la función de descodificación.

La Figura 3.5 muestra la secuencia de operaciones con las funciones de descodificación y transformada sin invertir, mientras que la Figura 3.6 muestra la secuencia de operaciones con las funciones de descodificación y transformada invertidas.

Podemos observar que, con esta implementación se tiene una latencia de tres slots de tiempo desde que se comienza a recibir información del primer GOP hasta que se visualiza. Este es el único efecto negativo de este diseño.

A la función de descodificación tenemos que indicarle que en un momento dado abandone lo que está procesando y termine. El tiempo necesario para abandonar la función es corto, pero no es inmediato. Esto es debido a la estructura del algoritmo y por lo tanto debemos tenerlo en cuenta.

El lugar idóneo para indicar a la función de descodificación que debe abandonar el proceso es dentro de la función de visualización ya que sabemos en qué fotograma del GOP nos encontramos y podemos informar en el momento adecuado. Para informar a la función de descodificación que termine, se utiliza un flag booleano. Este flag se activa a un determinado número de interrupciones antes del final del procesamiento del GOP. Dicho valor debe ser cuidadosamente seleccionado. Si es grande le indicamos a la

[1]Abandonar una proceso significa que termina y retorna de forma inmediata al punto de llamada.

Tiempo	Procesos realizados en ese instante
Slot de tiempo 1	Lectura GOP 1 Descodificación (sin datos) Transformación inversa (sin datos) Visualización (sin datos)
Slot de tiempo 2	Lectura GOP 2 Descodificación (GOP 1) Transformación inversa (GOP 1) Visualización (sin datos)
Slot de tiempo 3	Lectura GOP 3 Descodificación (GOP 2) Transformación inversa (GOP 2) Visualización (GOP 1)
Slot de tiempo 4	Lectura GOP 4 Descodificación (GOP 3) Transformación inversa (GOP 3) Visualización (GOP 2)
Slot de tiempo 5	Lectura GOP 5 Descodificación (GOP 4) Transformación inversa (GOP 4) Visualización (GOP 3)
Slot de tiempo 6	Lectura GOP 6 Descodificación (GOP 5) Transformación inversa (GOP 5) Visualización (GOP 4)
. . .	
Slot de tiempo n	Lectura GOP n Descodificación (GOP n−1) Transformación inversa (GOP n−1) Visualización (GOP n−2)

Figura 3.5: Secuencia de operaciones con las funciones sin invertir.

Tiempo	Procesos realizados en ese instante
Slot de tiempo 1	Lectura GOP 1 Tranformación inversa (sin datos) Descodificación (sin datos) Visualización (sin datos)
Slot de tiempo 2	Lectura GOP 2 Transformación inversa (sin datos) Descodificación (GOP 1) Visualización (sin datos)
Slot de tiempo 3	Lectura GOP 3 Transformación inversa (GOP 1) Descodificación (GOP 2) Visualización (sin datos)
Slot de tiempo 4	Lectura GOP 4 Transformación inversa (GOP 2) Descodificación (GOP 3) Visualización (GOP 1)
Slot de tiempo 5	Lectura GOP 5 Transformación inversa (GOP 3) Descodificación (GOP 4) Visualización (GOP 2)
Slot de tiempo 6	Lectura GOP 6 Transformación inversa (GOP 4) Descodificación (GOP 5) Visualización (GOP 3)
. . .	
Slot de tiempo n	Lectura GOP n Transformación inversa (GOP n−2) Descodificación (GOP n−1) Visualización (GOP n−3)

Figura 3.6: Secuencia de operaciones con las funciones invertidas.

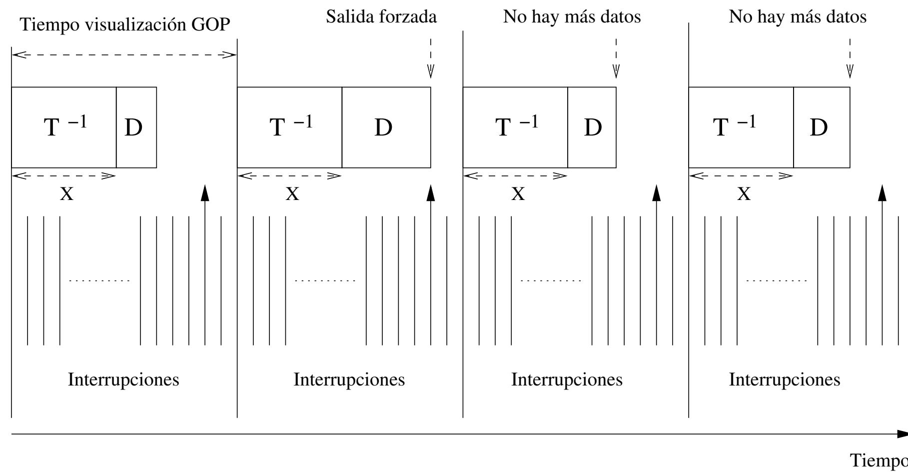

Figura 3.7: Representación del cauce de ejecución del cliente de vídeo progresivo con las funciones invertidas. La potencia de la CPU se considera constante.

función que abandone muy pronto, con lo que se descodifica poca información perdiendo calidad en la imagen. Si es pequeño entonces es posible que la función no abandone el proceso a tiempo. Debemos encontrar un punto de equilibrio. Un valor idóneo para la CPU donde se han realizado las simulaciones es 2. Un ejemplo de cómo se comporta el nuevo modelo se puede ver en la Figura 3.7. Se puede observar que en los GOPs número 1, 3 y 4 la fase de descodificación termina a tiempo mientras que en el GOP número 2 la fase de descodificación debe abandonar de forma forzada su proceso. Cada vez que se ejecuta una interrupción se visualiza un fotograma y se lee del socket.

Con este cambio en la ejecución, el cliente ha mejorado su comportamiento ante factores internos o externos que afecten a la capacidad de procesamiento de la CPU. La representación gráfica de la implementación del cliente de vídeo progresivo con las funciones invertidas es la mostrada en la Figura 3.8.

3.3.2. Mejora a la implementación

Para conseguir que el software se adapte a la tasa de transferencia de la red y a la capacidad de procesamiento de la CPU es necesario conocer cuáles son las cotas superiores de dichas medidas. Una vez obtenidas, se realiza una

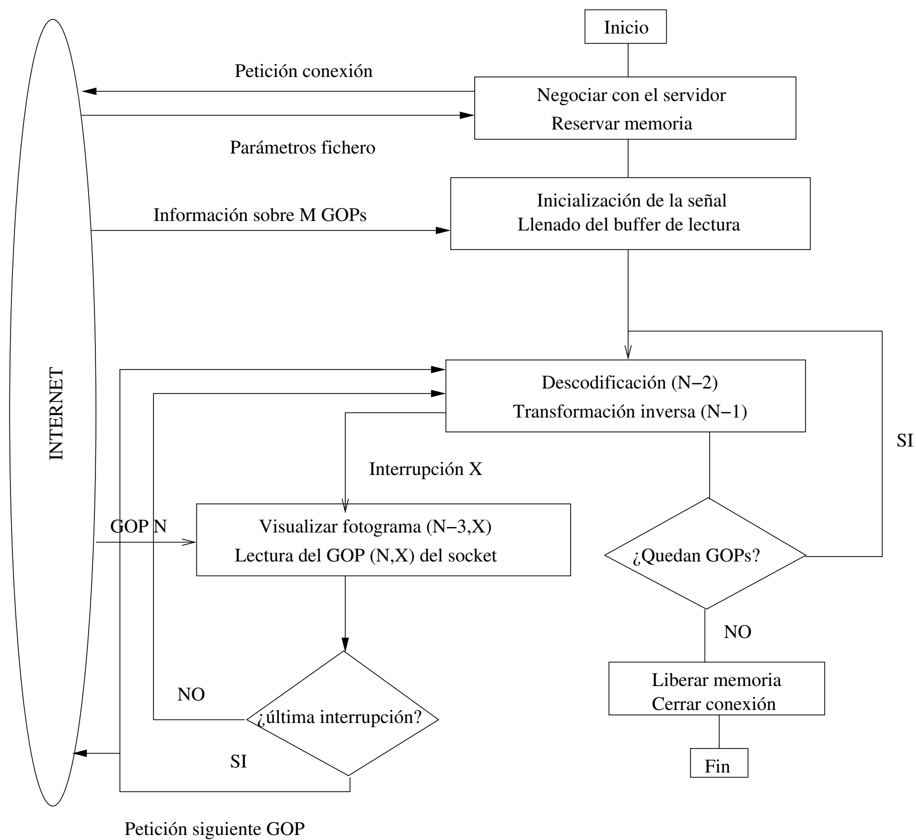

Figura 3.8: Representación gráfica del cliente de vídeo progresivo con las funciones invertidas.

petición al servidor tomando como referencia esa información.

Para saber en cada momento la tasa de transmisión existente se realiza un control sobre la cantidad de datos solicitada al servidor y el tiempo que se tarda en recibirse. Cuando el tiempo que se tarda en recibir la información solicitada es inferior al tiempo disponible para procesar y visualizar el GOP, nos indica que la tasa de transmisión ha aumentado. En caso contrario, es decir, si aun nos queda tiempo de procesamiento, significa que la tasa de transmisión ha disminuido.

Para calcular la tasa de procesamiento de la CPU nos basamos en la cantidad de información que la función de descodificación es capaz de procesar obteniendo así una buena aproximación. Se modifica dicha función para conocer en cada momento el número de bytes que se procesan. En base a esa información se hace una estimación de lo que realmente se puede procesar.

Al realizar la petición al servidor hay que tener en cuenta las nuevas tasas de transmisión y de CPU calculadas para cada GOP del vídeo. Ambas tasas están relacionadas de forma indirecta (el valor que se pide al servidor es el mínimo de estos dos valores). Esto es una deducción lógica ya que lo que indican las tasas de transmisión y de CPU es la cota superior (de lectura y procesamiento) aproximada para el sistema.

Como ejemplo demostrativo, supondremos que la transmisión de vídeo se realiza sobre una red local. En este escenario la tasa de transmisión que se puede obtener es muy superior a la tasa de CPU. Así, el límite superior lo impone la CPU por lo que se lee de la red sólo la cantidad de información que la CPU pueda procesar.

El escenario contrario, es la transmisión de vídeo usando un modem de 56 kbps, donde la tasa de transmisión es pequeña en comparación con la tasa de CPU. En este caso el valor máximo de información a pedir lo indica la tasa de transmisión.

Por último, incluso controlando la tasa de CPU y de transmisión, nunca estamos seguros de que toda la fase de procesamiento (transformada inversa y descodificación) acaben a tiempo para el siguiente GOP. En este caso se produce una espera en la reproducción lo más pequeña posible para que la sincronización permanezca. Esto es posible porque es el cliente quién controla la transmisión de los datos.

3.4. Descripción de la interfaz de la aplicación

Se ha diseñado la interfaz usando las APIs de X Window que reciben el nombre de Xlib. Estas bibliotecas ofrecen una libertad total para diseñar

cualquier tipo de interfaz pero tienen el inconveniente, de que hay que programar a muy bajo nivel. Los entornos X Window de UNIX están basados en estas bibliotecas. Otra ventaja es que aumenta la portabilidad de la aplicación.

3.4.1. Introducción a X Window

X Window es un sistema de ventanas desarrollado a mediados de los 80 y que se ha convertido en el estándar de los sistemas UNIX y, en particular, de Linux.

X Window es un sistema cliente/servidor, donde el servidor reside en cualquiera de los ordenadores de la red, y el cliente puede ser el mismo host que el servidor o cualquier otro host(u hosts) de la red.

El cliente se dedica a hacer peticiones al servidor, que es quien maneja el display o los dispositivos de entrada. Estos programas clientes son:

- Aplicaciones: Son los programas en sí mismos: procesadores de textos, hojas de cálculo, juegos, etc. Existen múltiples maneras de generar las aplicaciones (Gtk, Xlib, Imlib, etc), pero nosotros utilizaremos las llamadas a Xlib.

- Windows Managers: Son los gestores de ventanas, programas cliente con algunos privilegios extra que se desarrollan para que gestionar el interfaz o GUI comunicándose con el servidor X y que es el encargado de la forma en que se presentan las ventanas en pantalla, de dar facilidades con menús de configuración del sistema, barras de tareas, etc.

En cambio, el servidor X es el programa que proporciona las primitivas a las aplicaciones, con las que se comunica por medio de sockets. Los programas son ejecutados en la máquina servidora y se visualizan en la máquina cliente. El servidor controla el display ejecutando las peticiones de los clientes, generando los eventos y notificándoselos a las aplicaciones.

3.4.2. Diseño de la interfaz

Tomando como "referencia" una conocida aplicación de Windows realizamos una interfaz adaptada a nuestro fin. Para ello diseñamos una serie de imágenes JPEG con las partes de la interfaz. En conjunto es parecido a las aplicaciones que usan skins (pieles) para cambiar su aspecto gráfico.

Una vez obtenidas las imágenes que forman parte de nuestra interfaz, se integra de forma independiente en un programa aparte y le damos funcionalidad a las opciones.

La imagen de la Figura 3.9 es la base de nuestra interfaz.

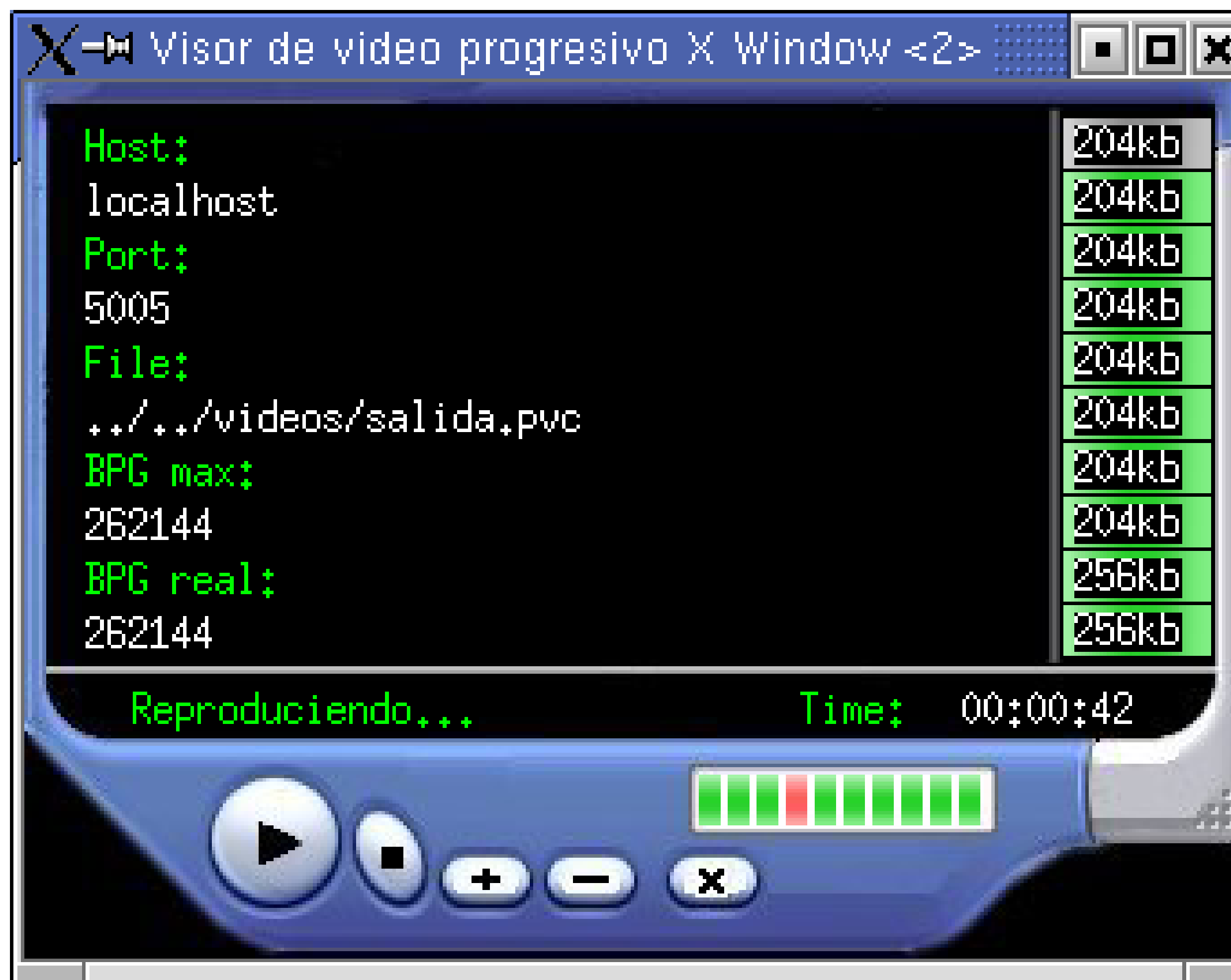

Figura 3.9: Imagen principal de la interfaz.

Para el diseño de las imágenes se ha usado un programa de diseño gráfico para la plataforma Windows. El software en cuestión es “Adobe Photoshop 5.0”. Este programa nos permite realizar cualquier tipo de tratamiento sobre una imagen. En nuestro caso nos hizo falta rediseñar parte de la interfaz, limpiarla, redondear los bordes, etc.

3.4.3. Funcionamiento de la interfaz

En este apartado vamos a explicar el funcionamiento de la interfaz y la información que nos ofrece como usuario de la misma.

La interfaz está dividida en 4 partes diferenciadas:

1. Zona de información: En esta zona se muestra información relativa a la conexión (máquina y puerto a donde estamos conectados), el nombre del vídeo que se está reproduciendo, la máxima cantidad de información a la que podemos reproducir y la cantidad de información del siguiente GOP que vamos a procesar.

2. Buffer lectura: Esta situado a la derecha de la interfaz y está formado por unas barras de colores, el color indica el estado de cada posición del buffer.

    a) Si el color es verde, indica que esa posición tiene información para ser procesada cuando sea su turno.

    b) Si el color es gris indica que esa posición del buffer está vacía o se está llenando, pero que aún no está completa.

    c) Existen dos colores más (azul y rojo), estos colores sólo se encienden en la posición más alta del buffer. Si la cantidad de información pedida al servidor en el slot de tiempo anterior no ha sido totalmente recibida o procesada, se indica con un color la causa (tasa de transmisión o de CPU). Si el color es rojo indica que el mínimo entre los dos valores es la tasa de transmisión. Si el color es azul indica que el valor mínimo es la tasa de CPU.

    En cada posición del buffer tiene un número que indica la cantidad de kbits (10^3 bits) que hay almacenado en dicha posición.

3. Marcador tiempo: Está situado debajo de la pantalla principal cada vez que el reloj avanza nos indica que un GOP ha sido procesado, así que realmente no es un temporizador de horas, minutos y segundos sino que es un "temporizador de GOPs".

4. Cuadro de mandos: El cuadro de mandos es la forma que tiene el usuario de comunicarse con la interfaz, existen diferentes opciones a realizar:

    a) Play: Este botón se utiliza para empezar a reproducir el vídeo, también se usará cuando el vídeo esté parado y queramos ponerlo en marcha.

b) Stop: Se utiliza para parar la imagen de la reproducción, el programa sigue recibiendo información del socket.

c) Más y menos: Estos botones se utilizan para indicarle al programa cúal es la cota máxima de bytes que podemos recibir, procesar, etc. de un GOP. Se utiliza para comprobar la progresividad de la reproducción. Con estos botones aumentamos o disminuimos esa cota.

d) Salida: Este botón sirve para abandonar la aplicación. Es necesario pulsar esta opción ya que así la aplicación se abandona de forma adecuada, liberando todos los recursos reservados.

e) Barra de desplazamiento: Esta barra se utiliza para forzar que se pida un porcentaje de la información calculada para el siguiente slot de tiempo. Se utiliza para ver la progresividad de la reproducción. Cada posición de la barra indica un porcentaje de dicha cantidad. Estos porcentajes son (20, 40, 60, 80, 100, 120, 140, 160, 180, 200) siempre que no sobrepasen la tasa calculada se pedirá el porcentaje indicado.

3.4.4. Programación de la interfaz

Después de estudiar la forma de programar usando las APIs de X Window y habiendo comprendido su funcionamiento, se realizó un programa para diseñar nuestra interfaz y le añadimos la funcionalidad.

La estructura básica de cualquier programa en X Window es la siguiente:

```
Programa:
{
        Inicialización();

        SeleccionarEventos();

        Repetir Espera_Evento
        {
                Si llega Pulsación_de_teclado:
                {
                        Código que gestiona teclado;
                }
                Si llega Pulsción_de_ratón:
```

```
                {
                        Código que gestiona ratón;
                }
                etc...

        }
        LiberarRecursos();
}
```

En la inicialización se suelen realizar las siguientes operaciones:

1. Apertura del display: Mediante XOpenDisplay() se abrirá el display deseado, normalmente el display 0.
2. Creación de una (o más) ventanas de trabajo: Utilizando funciones específicas para ello (XCreateSimpleWindow() o XCreateWindow()), creamos la ventana principal de nuestra aplicación.
3. Mapeado de la ventana: Con el fin de que la ventana sea visible, la mapeamos.

En la selección de eventos se suele realizar las siguientes operaciones:

- Indicar al servidor que tipo de eventos tiene que detectar, clasificar e identificar para que la aplicación defina su comportamiento ante cada evento. Para ello hay que estudiar bien los eventos y seleccionar sólo los necesarios, esto evita la sobrecarga de operaciones en el servidor.

Seleccionamos los eventos para los que nuestra aplicación debe responder. Los eventos configurados son:

- ExposureMask: Al seleccionar esta máscara se reciben los eventos de tipo Expose, que indican que una parte rectangular de la ventana se ha hecho visible. Un ejemplo ocurre en el caso de que una ventana de otra aplicación solapara en parte a nuestra ventana y de repente fuera cerrada o movida, con lo que el servidor se ve en la obligación de avisarnos de esta circunstancia para que el programa sepa que puede dibujar en dicha área. Si más de una zona se liberara simultáneamente, se generará un evento por cada zona rectangular que se haga visible, lo que nos permitirá redibujar cada una de estas zonas con los datos gráficos que deban contener y que antes no eran necesarios al estar tapados por otra ventana.

- ButtonPressMask: Nos avisa de pulsaciones del botón del mouse mediante un evento del tipo ButtonPress. Es decir, si seleccionamos el evento ButtonPressMask el servidor nos envía los mensajes ButtonPress pertinentes a la ventana especificada.

- PointerMotionMask: Esta máscara hace que se reciba un evento MotionNotify cuando se produzca cualquier movimiento del puntero del ratón (sin necesitar que el botón sea pulsado).

- Existen una serie de eventos que siempre están seleccionados, que son los siguientes: MappingNotify, ClientMessage, SelectionRequest, SelectionNotify y SelectionClear.

Dependiendo del tipo de evento que se desencadene realizamos unas operaciones u otras. Dichas operaciones son:

- Evento Expose: En este evento realizamos las operaciones de repintado de la pantalla con los datos almacenados durante la ejecución. Por ejemplo, imaginemos que estamos reproduciendo un vídeo y una ventana externa solapa nuestra ventana. Al moverla y volver a dejarla visible se genera este evento y por lo tanto debemos repintar la pantalla con los datos actuales.

- Evento ButtonPress: En este evento comprobamos si hemos pulsado con el ratón sobre ciertas partes de la interfaz, para controlar pulsaciones de botones, etc. Es el evento que genera las acciones en nuestra aplicación.

- Evento MotionNotify: En este evento controlamos si el puntero del ratón ha pasado sobre alguna parte de la ventana. Lo utilizamos para iluminar los botones de la aplicación cuando pasamos el puntero del ratón sobre ellos.

Otra de las tareas a realizar es la carga de las imágenes sobre una estructura adecuada para nuestro programa, dicha estructura esta formada por:

```
struct est_imagen
{
   int x,y,anchura,altura,xmin,xmax,ymin,ymax;
   Pixmap bitmap;
   Pixmap bitmap2;
```

```
   int origenx;
   int origeny;
   int desplazamiento;
};
```

donde:

- X,Y : Posición inicial de la imagen en la ventana.
- Anchura, Altura: Indican la altura y la anchura de la imagen.
- Xmin, Ymin, Xmax, Ymax: Indican el contorno de la imagen y que se utilizará para indicar si ha sido pulsado, seleccionado, etc.
- Bitmap, Bitmap2: En estas variables se cargarán los mapas de bits en si, es decir las imágenes JPEG, tenemos dos, porque normalmente cargamos la imagen activada y la desactivada.
- Origenx, Origeny: Se utiliza para cargar varias imágenes de forma paralela, por ejemplo el buffer, o la barra de desplazamiento.
- Desplazamiento: Indica el desplazamiento entre imagen e imagen, se usa para el buffer y la barra de desplazamiento.

Esta estructura es la misma para todas las imágenes aunque existan algunas que no utilizan todos los campos.

3.4.5. Integración de la interfaz y el cliente

Para poder integrar la interfaz y el cliente de vídeo progresivo se utilizaron funciones de programación de sistemas: creación de procesos mediante fork(), memoria compartida para la comunicación entre procesos, etc.

El grueso de la aplicación está dividido en dos procesos, uno que controla la interfaz, generación de eventos, etc., y otro proceso donde se hace la lectura de información sobre el socket, la descodificación, la transformación inversa y la visualización de GOP. Ambos procesos se comunicaban mediante una estructura almacenada en un área de memoria compartida.

La estructura compartida por ambos procesos y por tanto alojada en memoria compartida tiene la siguiente definición:

```
struct estructura {
   int bpgmemoria;
   int reloj;
   int bpgmax;
   int bpgreal;
   int stop;
   int salir;
   int cerrojo;
   float alfa;
   float porcentaje;
   int pics;
   int picscpu;
   int picscpuleft;
   int bpgred;
   int bpgdec;
   int causante;
   int swcpu;
};
```

donde :

- bpgmemoria: Esta variable se usa para almacenar los BPG[2] máximos permitidos durante la ejecución de la aplicación. Esto es debido a que la reserva de memoria está en función de esta variable.
- reloj: Variable usada para indicar en qué GOP de ejecución nos encontramos, además de alterar el tiempo en la interfaz, similar a un reloj digital.
- bpgmax: En esta variable almacenamos los BPG máximos actuales para un instante en concreto. Nunca supera a la variable bpgmemoria.
- bpgreal: Indican los bits que se pasan a la fase de procesamiento y que por tanto repercute en la calidad de la imagen.
- stop: Indica si se ha pulsado o no el botón de parada.
- salir: Indica si se ha pulsado el botón de salida. Esta variable activada hace que la aplicación termine.

[2]BPG (bits per GOP): Indica la cantidad de bits que forman un GOP. En nuestro caso la cantidad de bits que leemos de cada GOP.

- cerrojo: Variable de condición utilizada para los cerrojos.
- alfa: Variable utilizada para almacenar el valor de (necesaria para el cálculo de la tasa de transmisión.
- porcentaje: Indica el porcentaje seleccionado en la barra de desplazamiento.
- pics: indica en qué interrupción se ha terminado de leer la cantidad de información pedida al servidor en el slot de tiempo anterior.
- picscpu: Indica en qué interrupción se ha terminado el proceso de descodificación. Se utiliza para calcular la tasa de CPU.
- picscpuleft: Indica cuantas interrupciones quedan por ejecutarse una vez que se termina el proceso de descodificación.
- bpgred: Indica la cantidad de bytes que se han leído durante *pics* interrupciones. Se utiliza para calcular la tasa de transmisión.
- bpgdec: En esta variable es donde se almacenan los bytes reales que se procesan en la fase de descodificación. Se utiliza para calcular la tasa de CPU.
- causante: Almacena un número que indica quién es el responsable de la cantidad de información pedida al servidor ya que esta tasa se calcula como el valor mínimo de la tasa de transmisión y de CPU.
- swcpu: Se utiliza como variable de condición para saber en qué instante ha terminado de procesar la función de descodificación.

Capítulo 4

Evaluación

Durante el proyecto se ha desarrollado un PVC (codec de vídeo progresivo), que hemos utilizado para transmitir vídeo por Internet. Este algoritmo nos permite comprimir y reproducir vídeo con distintas formas de escalabilidad, aunque nuestra implementación sólo usa la SNR.

Se han realizado pruebas para comprobar la calidad de la imagen en la descodificación a diferentes niveles de compresión. Estas pruebas se compararon con el estándar en compresión de vídeo MPEG.

Para dichas pruebas se han comprimido dos secuencias de vídeo con el PVC y con MPEG-1. La primera secuencia, que la llamaremos "Claire", es bastante estática, donde apenas existe movimiento. En la segunda secuencia, que la llamaremos "Charla" contiene un mayor número de objetos que se mueven a una velocidad moderada.

Las secuencias se comprimieron a distintas tasas de bits, siendo estás:

- Claire : Las tasas de bits fueron 104, 128 y 256 kbps.
- Charla : Las tasas de bits utilizadas fueron 112, 128 y 256 kbps.

Pensamos realizar las pruebas a 56, 128 y 256 kbps, que es la equivalencia a un modem normal, una línea RDSI con dos canales y a una línea ADSL respectivamente, pero el algoritmo MPEG-1[1] utilizado tiene una tasa mínima de compresión y por debajo de esta tasa no es capaz de comprimir. La tasa no es un valor fijo sino que depende de las características del vídeo a comprimir.

Los valores de 104 kbps para el vídeo Claire y 112 kbps para el vídeo Charla son los valores aproximados a esa tasa mínima. Por debajo de estos valores el algoritmo MPEG-1 no realiza con éxito la compresión.

[1] Descargado de la página oficial www.mpeg.org.

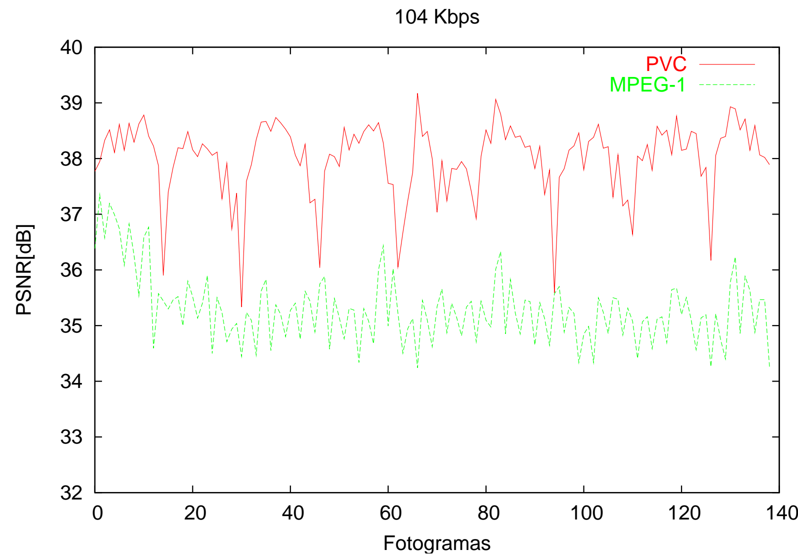

Figura 4.1: Calidad de las reconstrucciones a (en PSNR[dB]) de los fotogramas de la secuencia “Claire” usando PVC y MPEG-1 a 104 kbps.

4.1. Evaluación objetiva

Para la evaluación se ha procedido a medir la relación señal ruido (PSNR) de las imágenes descomprimidas usando PVC y MPEG de las secuencias de vídeo “Claire” y “Charla” con respecto a los fotogramas originales, previos a la compresión.

Para “Claire”, los PSNRs obtenidos a 104, 128 y 256 kbps han sido los mostrados en la Figura 4.1, 4.2 y 4.3 respectivamente.

El PSNR medio obtenido en la secuencia “Claire” usando PVC y MPEG-1 a las distintas tasas de compresión es el mostrado en la Tabla 4.1.

Se puede observar que el algoritmo PVC obtiene mejores resultados para tasas de bits bajas, a tasas de bits medias y altas la diferencia disminuye, siendo bastante similar en los dos algoritmos.

También observamos que los PSNRs obtenidos en media son bastante altos, esto es debido a que la secuencia de vídeo tiene poco movimiento, dos fotogramas contiguos son muy parecidos, por lo que tienen una alta correlación, los algoritmos se aprovechan de esta situación obteniendo buenos resultados.

Para “Charla”, los PSNRs obtenidos a 104, 128 y 256 kbps han sido los mostrados en las Figuras 4.4, 4.5 y 4.6 respectivamente.

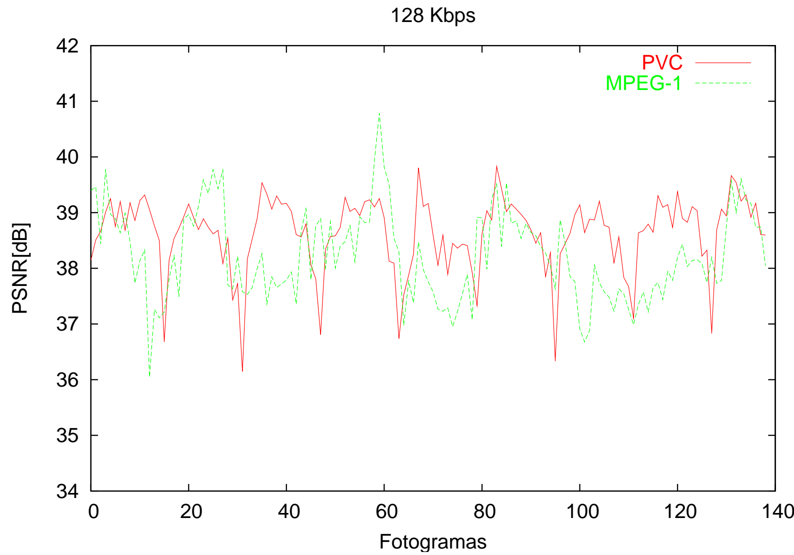

Figura 4.2: Calidad de las reconstrucciones a (en PSNR[dB]) de los fotogramas de la secuencia "Claire" usando PVC y MPEG-1 a 128 kbps.

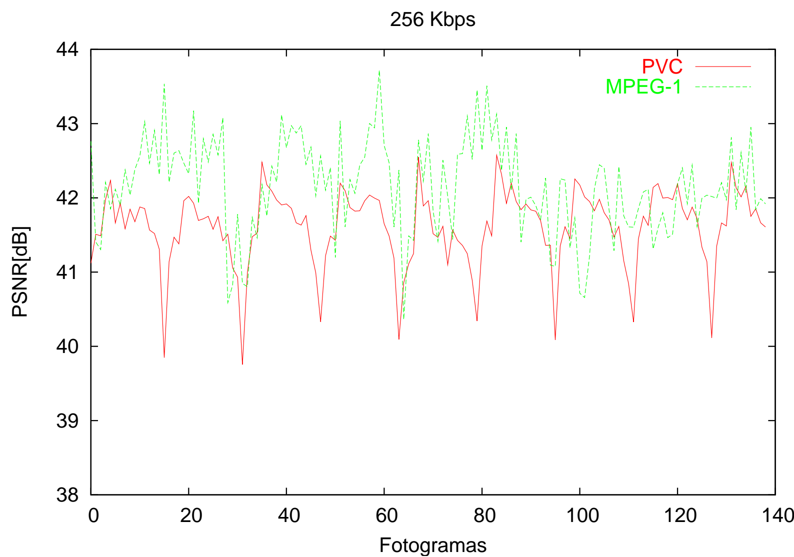

Figura 4.3: Calidad de las reconstrucciones a (en PSNR[dB]) de los fotogramas de la secuencia "Claire" usando PVC y MPEG-1 a 256 kbps.

Tabla 4.1: PSNR[dB] medio obtenido para la secuencia “Claire” usando PVC y MPEG-1.

	PVC	MPEG-1
104 kbps	**38,0 dB**	35,3 dB
128 kbps	**38,6 dB**	38,2 dB
256 kbps	41,6 dB	**42,1 dB**

Tabla 4.2: PSNR[dB] medio obtenido para la secuencia “Charla” usando PVC y MPEG-1.

	PVC	MPEG-1
112 kbps	**29,9 dB**	27,2 dB
128 kbps	**28,3 dB**	27,8 dB
256 kbps	31,3 dB	**32,4 dB**

El PSNR medio obtenido en la secuencia “Charla” usando PVC y MPEG-1 a las distintas tasas de compresión es el mostrado en la Tabla 4.2.

Como comentario a las gráficas observamos que los PSNRs obtenidos son inferiores a los de la otra secuencia, esto se debe a que en esta secuencia existe un mayor movimiento que en la anterior. La correlación entre fotogramas contiguos no es tan grande y por lo tanto no se obtienen unos resultados tan buenos.

Igual que en la anterior secuencia, a tasas de bits bajas y medias PVC obtiene un PSNR superior en casi toda la secuencia, es a altas tasas cuando la diferencia entre los dos algoritmos se estrecha y apenas existe diferencia en los resultados.

Para terminar comentar que PVC permite comprimir a cualquier tasa de bits y que obtiene sus mejores resultados a tasas de bits bajas y medias. Esta característica hace que sea un algoritmo idóneo para transmitir vídeo por Internet consiguiendo un resultado excelente.

4.2. Evaluación subjetiva

El criterio de evaluación será estrictamente visual. Para ello tomaremos distintos fotogramas de las dos secuencias de vídeo que han sido descompri-

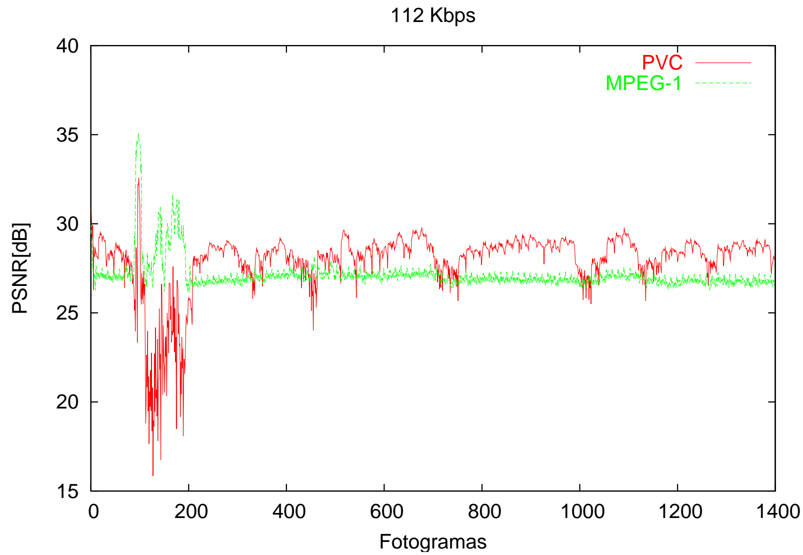

Figura 4.4: Calidad de las reconstrucciones a (en PSNR[dB]) de los fotogramas de la secuencia “Charla” usando PVC y MPEG-1 a 104 kbps.

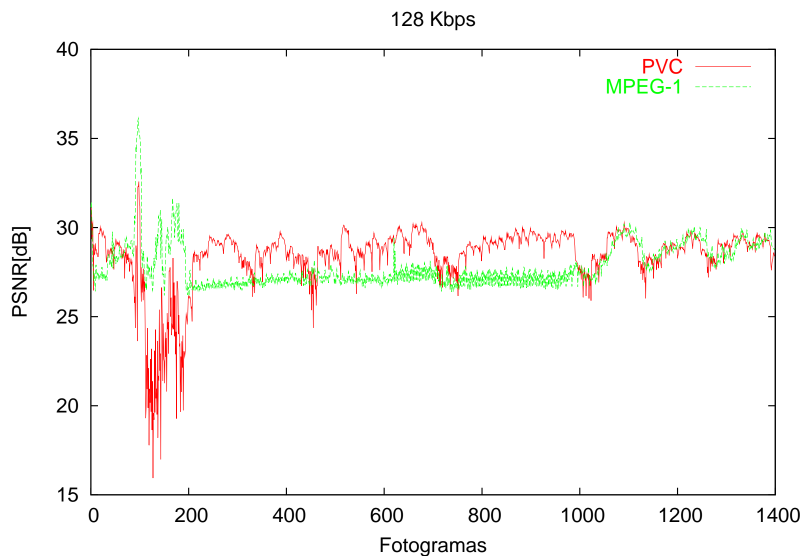

Figura 4.5: Calidad de las reconstrucciones a (en PSNR[dB]) de los fotogramas de la secuencia “Charla” usando PVC y MPEG-1 a 128 kbps.

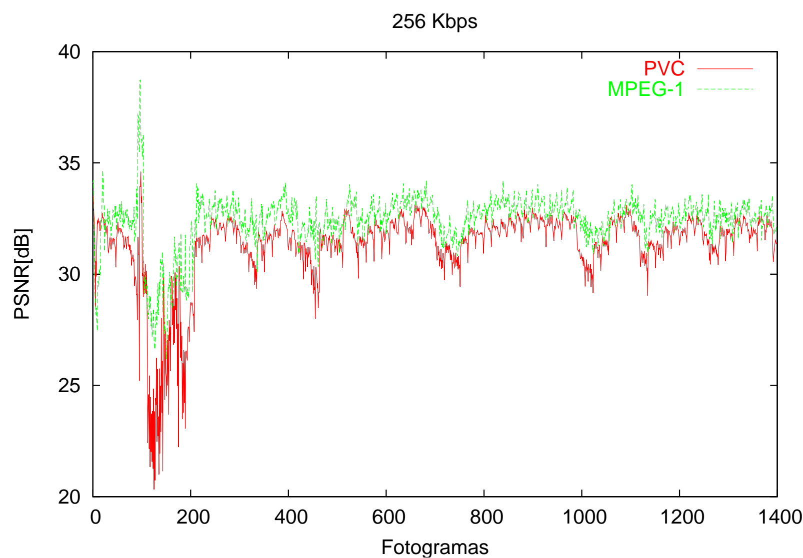

Figura 4.6: Calidad de las reconstrucciones a (en PSNR[dB]) de los fotogramas de la secuencia "Charla" usando PVC y MPEG-1 a 256 kbps.

midos usando PVC y MPEG-1.

Las Figuras 4.7, 4.8 y 4.9 muestran distintos fotogramas de la secuencia "Claire" a una tasa de bits de 104, 128 y 256 kbps.

Se puede observar que a 104 y 128 kbps los fotogramas en MPEG tienen un efecto de losetizado, en cambio PVC suaviza los contornos y las figuras son más redondeadas. A 256 kbps ya no es apreciable la diferencia entre ambos algoritmos. PVC se comporta de manera excelente en este tipo de secuencias.

Para la secuencia "Charla" observamos que MPEG-1 obtiene mejores resultados que PVC ya que las imágenes son más nítidas. Suponemos que la estimación de movimiento ha realizado un buen trabajo y es la diferencia que se observa en las imágenes. En este caso la PSNR obtenida para esta secuencia no concuerda con la evaluación subjetiva, ya que PVC obtenía mejores resultados para 112 y 128 kbps y un resultado muy similar a MPEG a 256 kbps.

Figura 4.7: Fotogramas de "Claire" a 104 kbps. La primera columna corresponde a MPEG-1, mientras que la segunda columna corresponde a PVC.

Figura 4.8: Fotogramas de “Claire” a 128 kbps. La primera columna corresponde a MPEG-1, mientras que la segunda columna corresponde a PVC.

Figura 4.9: Fotogramas de "Claire" a 256 kbps. La primera columna corresponde a MPEG-1, mientras que la segunda columna corresponde a PVC.

Figura 4.10: Fotogramas de "Charla" a 112 kbps. La primera columna corresponde a MPEG-1, mientras que la segunda columna corresponde a PVC.

Figura 4.11: Fotogramas de "Charla" a 128 kbps. La primera columna corresponde a MPEG-1, mientras que la segunda columna corresponde a PVC.

Figura 4.12: Fotogramas de “Charla” a 256 kbps. La primera columna corresponde a MPEG-1, mientras que la segunda columna corresponde a PVC.

Capítulo 5

Ampliaciones del proyecto

Como trabajo futuro para completar este proyecto y poder disponer de un software con multitud de aplicaciones comerciales es necesario añadirle nuevas funcionalidades. Entre ellas destacar:

1. Sonido: Es necesario incorporar sonido a la señal de vídeo que se envía al canal. El sonido debe estar comprimido de forma progresiva, para que se reproduzca a una mayor o menor tasa de bits en función de la capacidad del canal.

2. Color: Añadirle color al vídeo es una cualidad que haría que ganase en vistosidad, para cierto tipo de aplicaciones es necesario disponer del vídeo en color. No obstante requiere un incremento en tiempo de computación a la hora de descomprimir.

3. Escalabilidad espacial y temporal: Incorporar estas características al sistema haría que no se tuviese ningún tipo de limitación funcional y se adaptaría a cualquier tipo de aplicación.

Una vez que se consigan están mejoras, el sistema se podrá utilizar para multitud de aplicaciones, entre ellas están: vídeo bajo demanda, videoconferencia en un sólo sentido, formación a distancia, etc.

Apéndice A

Funciones de la API de UNIX utilizadas

Durante la programación del cliente y el servidor hemos utilizado APIs o estructuras del sistema UNIX que pueden resultar poco familiares o desconocidas, en este apartado vamos a explicar las más importantes. Las vamos a separar en diferentes grupos en función de su utilidad.

A.1. Señales y tiempos

A.1.1. signal

Para especificar qué tratamiento debe realiza un proceso al recibir una señal, se emplea la llamada signal. Su declaración es la siguiente:

```
#include <signal.h>

void  *signal (int sig, void (*action) ()) ();
```

- sig es el número de la señal sobre la que queremos especificar la forma de tratamiento.
- action es la acción que queremos que se inicie cuando se reciba la señal. action puede tomar tres tipos de valores:
  - SIF_DFL: Indica que la acción a realizar cuando se recibe la señal es la acción por defecto asociada a la señal.

- SIG_IGN: Indica que la señal se debe ignorar.
- Dirección: Es la dirección de la rutina de tratamiento de la señal (manejador suministrado por el usuario). La declaración de esta función debe ajustarse al siguiente modelo.

```
#include <signal.h>

void handler (sig [, code, scp])
int sig, code;
struct sigcontext *scp;
```

Cuando se recibe la señal sig, el núcleo es quien se encarga de llamar a la rutina handler pasándole los parámetros sig, code y scp. sig es el número de la señal, code es una palabra que contiene información sobre el estado del hardware en el momento de invocar a handler y scp contiene información de contexto definida en signal.h. Tanto code como scp son parámetros opcionales y dependientes de la arquitectura de nuestra máquina.

Un ejemplo de utilización de signal:

```
if (signal(SIGALRM, &ProcesoVisor) == SIG_ERR) {
        fprintf(stderr,"No puedo registrar el
        manejador de señales para SIGALRM\n");
}
```

A.1.2. setitimer

Esta llamada se utiliza para controlar los temporizadores que hay definidos por cada proceso. Su declaración es la siguiente:

```
#include <time.h>

setitimer (int which, struct itimerval *value,
           struct itimerval *ovalue)
```

Los valores que puede tomar which son:

- ITIMER_REAL: Temporizador en tiempo real.

- ITIMER_VIRTUAL: Temporizador en tiempo virtual. solamente contabiliza el tiempo mientras el proceso está ejecutando y no cuando duerme.
- ITIMER_PROF: Temporizador de perfilado. Se usa al crear perfiles de un proceso.

Setitimer se utiliza para definir el valor del temporizador especificado por which. Value es un puntero a una estructura con los nuevos valores del temporizador y ovalue es un puntero a una estructura donde se devuelven los antiguos valores del temporizador.

La estructura itimerval se define como sigue:

```
struct itimerval
{
        struct timeval it_interval;
        struct timeval it_value;
};
```

siendo it_interval el intervalo del temporizador y it_value el valor actual del temporizador.

La estructura timeval se define así:

```
struct timeval
{
        unsigned long tv_sec;
        long tv_usec;
};
```

siendo tv_sec los segundos transcurridos desde el día 1 de Enero de 1970 y tv_usec indica microsegundos. Su rango está comprendido entre el 0 y 999.999.

Un ejemplo del uso de setitimer:

```
itimer.it_interval.tv_usec =i_sec;
itimer.it_interval.tv_sec = 0;
```

```
itimer.it_value.tv_usec = i_sec;
itimer.it_value.tv_sec = 0;

setitimer(ITIMER_REAL, &itimer, NULL);
```

A.2. Regiones críticas o mutex

Entre las funciones utilizadas tenemos:

A.2.1. pthread_mutex_init

Inicializa el semáforo para poder ser utilizado. POSIX define distintos atributos que modifican el comportamiento de los semáforos. Por defecto el mutex no utilizará ningún tipo de herencia de prioridades.

Su definición es la siguiente:

```
int pthread_mutex_init(pthread_mutex_t *mutex,
        pthread_mutexattr_t *attr);
```

Siendo mutex un puntero a un parámetro del tipo pthread_mutex_t, que es el tipo de datos que usa la biblioteca Pthreads para controlar los mutex. Attr es un puntero a una estructura del tipo pthread_mutexattr_t, y sirve para definir qué tipo de mutex queremos: normal, recursivo o errorcheck. Si este valor es NULL (recomendado), la biblioteca le asignará un valor por defecto. La función devuelve 0 si se pudo crear el mutex o -1 si hubo algún error.

A.2.2. pthread_mutex_lock

Esta función pide el bloqueo para entrar en una RC (región crítica). Si queremos implementar una RC, todos los threads tendrán que pedir el bloqueo sobre el mismo semáforo. La definición es:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

siendo mutex un puntero al mutex sobre el cual queremos pedir el bloqueo o sobre el que nos bloquearemos en caso de que ya haya alguien dentro de la RC. Como resultado, devuelve 0 si no hubo error, o diferente de 0 si lo hubo.

A.2.3. pthread_mutex_unlock

Esta es la función contraria a la anterior. Libera el bloqueo que tuviéramos sobre un semáforo. La definición es:

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

siendo mutex el semáforo donde tenemos el bloqueo y queremos liberarlo. Retorna 0 como resultado si no hubo error o diferente de 0 si lo hubo.

A.3. Procesos

A.3.1. fork

La única forma de crear un proceso en el sistema UNIX es mediante la llamada fork. El proceso que invoca a fork se llama proceso padre y el proceso creado es el proceso hijo. La declaración es la siguiente:

```
#include <sys/types.h>

int fork();
```

La llamada a fork hace que el proceso actual se duplique. A la salida de fork, los dos procesos tienen una copia idéntica del contexto del nivel de usuario excepto el valor de pid, que para el proceso padre toma el valor del PID del proceso hijo y para el proceso hijo toma el valor 0. Si la llamada a fork falla, devolverá el valor -1 y en errno estará el código del error producido.

A.4. Sockets

A.4.1. socket

La llamada para abrir un canal bidireccional de comunicaciones es socket y se declara como sigue:

```
#include <sys/types.h>
#include <sys/socket.h>

int socket (int af, int type, int protocol);
```

Socket crea un punto de conexión a un canal y devuelve un descriptor. El descriptor del conector devuelto se usará en llamadas posteriores a funciones de la interfaz.

Af (address family) especifica la familia de conectores o familia de direcciones que se desea emplear. Las distintas familias están definidas en el fichero de cabecera ¡sys/socket¿. Las dos familias siguientes suelen estar presentes en todos los sistemas:

- AF_UNIX: Protocolos internos UNIX. Es la familia de conectores empleada para comunicar procesos que se ejecutan en una misma máquina.

- AF_INET: Protocolos Internet. Es la familia de conectores que se comunican mediante protocolos, tales como TCP o UDP.

El argumento type indica la semántica de la comunicación para el conector. Puede ser:

- SOCK_STREAM: Conector con un protocolo orientado a conexión.

- SOCK_DGRAM: Conector con un protocolo no orientado a conexión o datagrama.

Protocol especifica el protocolo particular que se va a usar con el conector. Si la llamada se ejecuta satisfactoriamente, devolverá un descriptor de fichero válido. En caso contrario, devolverá -1 y en errno estará codificado el error producido.

A.4.2. bind

La llamada bind se utiliza para unir un conector con una dirección de red y puerto determinados. Su declaración es la siguiente.

```
#include <sys/socket.h>
/* Para familia AF_UNIX */
#include <sys/un.h>
/* Para familia AF_INET */
#include <sys/netinet.h>

int bind (int sfd, const void *addr, int addrlen);
```

Cuando se crea un conector con la llamada socket, se le asigna una familia de direcciones, pero no una dirección particular. Bind hace que el conector cuyo descriptor es sfd se una a la dirección de conector especificada en la estructura apuntada por addr. Addrlen indica el tamaño de la dirección.

A.4.3. listen

Cuando se abre un conector orientado a conexión, el programa servidor indica que está disponible para recibir peticiones de conexión mediante la llamada a listen, que se declara como sigue:

```
int listen (int sfd, int backlog);
```

La llamada a listen la suele ejecutar el proceso servidor después de las llamadas a socket y bind. Listen habilita una cola asociada al conector descrito por sfd. La longitud de esta cola es la especificada en el argumento backlog. Para que la llamada a listen tenga sentido, el conector debe ser de tipo SOCK_STREAM (orientado a conexión).

A.4.4. connect

Para que un proceso cliente inicie una conexión con un servidor a través de un conector, es necesario que haga una llamada a connect. Esta función se declara de la siguiente forma:

```
#include <sys/socket.h>
/* familia AF_UNIX */
#include <sys/un.h>
/* familia AF_INET */
#include <sys/netinet.h>

int connect (int sfd, const void *addr, int addrlen);
```

sfd es el descriptor del conector que da acceso al canal y addr es un puntero a una estructura que contiene la dirección del conector remoto al que queremos conectarnos. Addrlen es el tamaño en bytes de la dirección.

A.4.5. accept

Los procesos servidores van a leer peticiones de servicio mediante la llamada accept. La declaración de esta llamada se muestra a continuación:

```
#include <sys/socket.h>

int accept (int sfd, void *addr, int *addrlen);
```

Esta llamada se usa con conectores orientados a conexión, como el tipo SOCK_STREAM. El argumento sfd es un descriptor del conector creado por una llamada previa a socket y unido a una dirección mediante bind. Accept extrae la primera petición de conexión que hay en la cola de peticiones pendientes creada con una llamada previa a listen. El argumento addr debe apuntar a una estructura local con la dirección del conector. La llamada accept rellenará esa estructura con la dirección del conector remoto que pide la conexión. El argumento addrlen debe ser un puntero a int.

A.4.6. close

Una vez que un proceso no necesita realizar más accesos a un conector, puede desconectarse del mismo. Para ello podemos usar la orden close. Esta llamada va a cerrar el conector en sus dos sentidos.

A.4.7. htonl

Función que convierte datos unsigned long del formato que maneja el ordenador al formato que maneja la red. La definición es:

```
unsigned long htonl (unsigned long hostlong);
```

A.4.8. ntohl

Función que convierte datos unsigned long del formato que maneja la red al formato que maneja el ordenador. La definición es:

```
unsigned long ntohl (unsigned long netlong);
```

A.4.9. select

Permite interrogar al sistema sobre el estado de varios dispositivos y devuelve cuáles están listos para leer datos de ellos y cuáles lo están para escribir en ellos.

La declaración de select es la siguiente:

```
#include <time.h>

int select (int nfds, int readfds, int writefds,
        int exceptfds, struct timeval *timeout);
```

select examina los descriptores de fichero especificados en las máscaras de bits readfds, writefds y exceptfds.

El significado de cada máscara es el siguiente:

- readfds representa los descriptores de los ficheros de lectura por los que preguntamos.
- writefds representa los descriptores de los ficheros de escritura que preguntamos.
- exceptfds representa los descriptores de los ficheros con alguna condición especial por los que preguntamos.

Si alguna de las condiciones anteriores no nos interesa, lo indicaremos pasándole a select el argumento 0.

Los argumentos se pasan por referencia, esto significa que select los va a modificar de acuerdo con el estado de los ficheros que interroga. timeout es un puntero no nulo que indica el tiempo máximo que se va a esperar desde que select entra en ejecución hasta que devuelve el control. Select puede devolver el control bien porque alguno de los ficheros interrogados cumpla los requisitos pedidos o bien porque el tiempo de espera especificado en timeout expire. Si timeout es un puntero a NULL, la llamada esperará a que se dé algún cambio de estado en alguno de los ficheros interrogados.

Las máscaras de bits que codifican los números de los descriptores agrupan 32 posibles descriptores por cada numero entero. Si queremos interrogar por ficheros con un descriptor más alto, hay que utilizar un array de enteros. Para encapsular estos detalles, junto con select se facilita un conjunto de 4 macros para manejar variables de ese tipo.

Estas macros se definen con la siguiente interfaz:

- FD_ZERO (fd_set *fdset): Pone a cero los bits de “fdset”.
- FD_SET (int fd, fd_set *fdset): Activa en fdset el bit correspondiente al descriptor fd.
- FD_CLEAR (int fd, fd_set *fdset): Desactiva en fdset el bit correspondiente al descriptor fd.

- FD_ISSET (int fd, fd_set *fdset): Comprueba en fdset el bit correspondiente al descriptor fd.

A.5. Memoria compartida

A.5.1. shmget

Esta función devuelve el identificativo del segmento de memoria compartida asociado al valor del argumento key. Su declaración es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget (key_t key, int size, int shmflg);
```

siendo:

- key es el identificativo del área de la memoria compartida.
- size es el tamaño en bytes de la zona de memoria que queremos crear.
- shmflg es una máscara de bits que está compuesta por los siguientes valores:
    - IPC_CREAT: para crear un nuevo segmento. Si este indicador no se usa, shmget() encontrará el segmento asociado con key, comprobará que el usuario tenga permiso para recibir el shmid asociado con el segmento, y se asegurará de que el segmento no esté marcado para destrucción.
    - IPC_EXCL: Usado con IPC_CREAT para asegurar el fallo si el segmento ya existe. Mode_flags (9 bits mas bajos): Especifican los permisos otorgados al dueño, grupo y resto del mundo.

A.5.2. shmat y shmdt

Antes de usar una zona de memoria compartida, tenemos que asignarle un espacio de direcciones virtuales de nuestro proceso. Esto es lo que se conoce como unirse o atarse al segmento de memoria compartida. Una vez

que dejamos de usar un segmento de memoria, tenemos que desatarnos de él. Al realizar esta operación, el segmento deja de estar accesible para el proceso. Sus definiciones son:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat (int shmid, char *shmaddr, int shmflg);
int shmdt (char *shmaddr);
```

siendo:

- shmid es el identificador de una zona de memoria creada mediante una llamada previa a shmget.
- shmaddr es la dirección virtual donde queremos que empiece la zona de memoria compartida.
- shmflg es una máscara de bits que indica la forma de acceso a la memoria.

A.5.3. shmctl

Con shmct realizamos operaciones de control sobre una zona de memoria previamente creada por una llamada a shmget. Su declaración es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (int shmid, int cmd, struct shmid_ds *buf)
```

shmid es un identificador válido devuelto por una llamada previa a shmget. cmd indica el tipo de operación a realizar. Entre las más importantes está:

- IPC_RMID: Borra del sistema la zona de memoria compartida identificada por shmid. Si el segmento está unido a varios procesos, el borrado no se hace efectivo hasta que todos los procesos liberen la memoria.

A.6. X Window

A.6.1. XOpenDisplay

Se usa para establecer una conexión entre un programa de aplicación y un servidor de display. Su definición es:

```
Display *XOpenDisplay (char *nombreDisplay);
```

Donde nombreDisplay es una cadena con el formato: nombreHost:numero.numeroPantalla.

nombreHost es el nombre del host en el que reside el servidor. El numero es el número del display, y el numeroPantalla es el numero de la pantalla.

El nombreDisplay puede ser NULL, en cuyo caso se usará por la cadena almacenada en la variable de entorno DISPLAY.

A.6.2. XCloseDisplay

Sirve para cerrar la conexión con el display. Su definición es la siguiente:

```
XCloseDisplay (Display *display);
```

Cierra la conexión con el servidor de display especificado y destruye todas las ventanas, todos los Ids de los recursos creados por el programa.

A.6.3. DefaultScreen

Devuelve el número asociado a la pantalla por defecto en referencia a un display abierto mediante XopenDisplay. Su definición es la siguiente:

```
DefaultScreen (display);
```

Este número de pantalla se utilizará en llamadas a otras funciones, por lo que es aconsejable que sea almacenado en una variable global.

A.6.4. XCreateSimpleWindow

Se utiliza para crear una ventana asociada a un display. Su definición es la siguiente:

```
Window XcreateSimpleWindow (Display *display,
        Window madre, int x, int y, unsigned int anchura,
        unsigned int altura, unsigned int anchuraborde,
        unsigned long colorborde, unsigned long colorfondo);
```

El display especifica la conexión con el servidor. La madre especifica la ventana madre donde se creará la ventana. Se pueden usar las macros RootWindow o DefaultRootWindow.

A.6.5. XMapWindow

Mapea la ventana especificada. Su definición es la siguiente:

```
XmapWindow (Display *display, Window window);
```

A.6.6. XSelectInput

Durante la inicialización de nuestra aplicación y creación de la ventana principal de la misma es necesario indicarle al servidor qué tipo de eventos se desean recibir, pues puede interesarnos no recibir información de algún tipo de evento ocurrido, mientras que otro pueda ser vital para nuestra aplicación. Para la indicación de los eventos deseados al servidor X se utiliza la función XselectInput y las llamadas máscaras de eventos.

La definición de la función es la siguiente:

```
XselectInput (display, window, event_mask);
```

El argumento display es la variable que conecta nuestro cliente con el servidor X, la variable window es el identificador de la ventana sobre la que deseamos seleccionar los eventos que se recibirán, y event_mask es una variable máscara. Cada bit de esta variable se refiere a un evento concreto. Para hacer más fácil la selección de eventos se usan unas cadenas o nombres simbólicos que identifican a los eventos.

Los principales son : ButtonPressMask, ButtonReleaseMask, KeyPressMask, KeyReleaseMask, ButtonMotionMask, EnterWindowMask, LeaveWindowMask, etc.

A.6.7. XNextEvent

Recoge el siguiente evento de la cola en modo bloqueante, es decir, recoge cualquier evento de cualquier ventana de nuestra aplicación y lo introduce en la estructura XEvent que se le pasa como parámetro. Su definición es la siguiente:

```
XNextEvent (Display *display, XEvent *evento);
```

La estructura XEvent es el paquete de información que el servidor envía al cliente cuando ocurre el evento.

A.6.8. XCreateGC

Un contexto gráfico o simplemente GC, es un recurso del servidor donde se almacenan los atributos de los gráficos. Es necesario al menos un GC para que el servidor acepte peticiones de primitivas gráficas por parte del cliente, ya que cada petición de primitiva gráfica debe especificar el identificador del GC que se debe usar.

Los GCs se crean y se almacenan en el servidor. Para pintar gráficos es necesario:

- Crear un GC y obtener su ID.
- Establecer los atributos del GC apropiadamente.
- Enviar las peticiones de las primitivas gráficas.

La definición para crear un contexto gráfico es la siguiente:

```
GC XcreateGC (Display *display, Drawable drw,
        unsigned long mascara, XGCValues *valores);
```

A.6.9. XcopyArea

La definición de esta función es la siguiente:

```
XcopyArea (Display *display, drawable src, drawable dest,
        GC gc, int src_x, int src_y, unsigned int width,
        unsigned int height, int dest_x, int dest_y);
```

Esta función copia un área cuadrada desde el drawable src al drawable dest. Ambos han de tener la misma profundidad de color. La principal ventaja de trabajar con estructuras XImage consiste en que éstas se almacenan localmente en el cliente haciendo que no tengan que pasar por la red.

Bibliografía

[1] P.M. Embree and B. Kimble. *C Language Algorithms for Digital Signal Processing*. Prentice-Hall, 1991.

[2] D.A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.

[3] W.K. Pratt. *Digital Image Processing*. John Wiley & Sons, Inc., 1991.

[4] L. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall, 1975.

[5] V.G. Ruiz. Compresión reversible y transmisión de imágenes. Master's thesis, Depto de Arquitectura de Computadores y Electrónica, Universidad de Almería, Julio 2000.

[6] A. Said and W.A. Pearlman. Image Compression Using the Spatial-Orientation Tree. In *IEEE Int. Symp. Circuits and Systems*, pages 279–282, Chicago, IL, 1993.

[7] A. Said and W.A. Pearlman. Reversible Image Compression via Multiresolution Representation and Predictive Coding. In *Proc. SPIE Conf. Visual Comm. and Image Proc.*, volume 2094, pages 664–674, 1993.

[8] A. Said and W.A. Pearlman. A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. *IEEE Trans. Circuits Syst. for Video Technol.*, 6:243–250, 1996.

[9] A. Said and W.A. Pearlman. An Image Multiresolution Representation for Lossless and Lossy Compression. *IEEE Trans. Image Process.*, 5(9):1303–1310, 1996.

[10] T. Duy Tran. *Linear Phase Perfect Reconstruction Filter Banks: Theory, Structure, Design, and Application in Image Compression.* PhD thesis, University of Wisconsin - Madison, 1999.

[11] J.W. Woods and S.D. O'Neil. Subband Coding of Images. *IEEE Trans. Acoust. Speech Signal Process.*, 34(5):1278–1288, 1991.