Procesamiento digital de video en tiempo real y "video wall" con la PC

Tesis para obtener el grado de

Maestro en Ciencias de la Computación

Nombre del alumno: Ukranio Coronilla Contreras

Matricula: 98380044

Nombre del asesor: Dr. Carlos Avilés Cruz

Marzo del 2005

Síntesis

En el desarrollo de la presente tesis se abordaron tres tipos de problemas, el primero fue establecer el hardware mínimo necesario para llevar acabo el despliegue de una imagen de video en forma de videowall(arreglo cuadrado de monitores que presenta una sola imagen), haciendo uso de la PC y con cuatro monitores. El segundo problema consiste en hacer uso del sistema operativo para controlar las cuatro tarjetas de video, lo cual corresponde a un procesamiento de escalamiento en la imagen, y finalmente el problema fundamental de construir el software que permita el procesamiento de video en tiempo real, el cual se apoya en el sistema de desarrollo SDK de DirectX® y en particular la herramienta DirectShow. Se elaboraron procesamientos básicos de video que consisten en filtrado de componentes de color, adición de ruido a la imagen, superposición de imagen al video, eliminación de color, posterizado y realce.

INDICE

AGRADECIMIENTOS

Agradezco a la comunidad perteneciente al grupo de noticias de Microsoft DirectX MSDN newsgroup en gráficos y multimedia por su valiosa ayuda en las múltiples dudas surgidas durante el desarrollo del software.

Del mismo modo a nuestra institución educativa y en particular a la Maestría en Ciencias de la Computación por su labor formativa.

Finalmente reitero mi agradecimiento al Dr. Carlos Avilés Cruz por haber apoyado mi iniciativa en la elaboración del presente proyecto, y facilitarme los recursos necesarios del laboratorio multimedia.

Ukranio Coronilla

Introducción

1.1 Antecedentes

El procesamiento digital de señales y sus aplicaciones envuelven hoy día una gran cantidad de áreas como la medica, espacial, de telefonía, comercial, científica e industrial. Sus orígenes se ubican en las décadas de los 60´s y 70´s, momento en el cual las computadoras digitales comenzaron a estar disponibles. Los desarrollos iniciales se dieron solo en áreas estratégicas como radar y sonar, exploración petrolera y espacial, además de imágenes medicas. A partir de los 80´s y 90´s la revolución en las computadoras comenzó a expandir el campo del procesamiento digital de señales en nuevas aplicaciones, principalmente comerciales como telefonía móvil, reproductores de discos compactos, y videoconferencia.

Un área de desarrollo relacionada con el presente trabajo es el procesamiento de imágenes. La manipulación y almacenamiento de imágenes involucra una gran cantidad de recursos, por ejemplo para almacenar un segundo de video se requieren mas de 10Mb. Motivo por el cual hasta últimos años ha sido accesible la capacidad de procesamiento a costos moderados. Normalmente el procesamiento digital se lleva a cabo con dispositivos digitales de propósito especifico denominados DSP (Digital Signal Processors). Aunque en los últimos años el desarrollo de los DSP ha permitido su incorporación en diversos dispositivos electrónicos, actualmente el procesamiento de video a través de estos es caro debido a que el mercado no es muy extenso aun y su demanda reducida.

La ventaja del uso de microprocesadores de propósito general respecto a los DSP, radica en la celeridad con la que crece su mercado, provocando una disminución de costo y el consiguiente aumento en los rendimientos y capacidades de procesamiento. Las características de hardware paulatinamente van semejando mucho a las de un DSP. Baste como ejemplo citar la tecnología MMX que los procesadores Pentium de Intel® han añadido a sus procesadores:

> " Los diseñadores de Intel han añadido 57 nuevas y poderosas instrucciones específicamente diseñadas para manipular y procesar video, audio y datos gráficos de manera eficiente. Estas instrucciones están orientadas a conseguir secuencias altamente paralelas y repetitivas encontradas continuamente en operaciones multimedia. "
> http://www.intel.com/espanol/index.htm (Mayo 2001)

Estas novedosas capacidades son explotadas en el presente trabajo al construirse el software que permite llevar a cabo procesamiento de video en tiempo real en una computadora personal. Dicho software se apoya en el sistema de desarrollo SDK de DirectX® DirectShow el cual ofrece acceso directo a tarjetas de video y de audio conectadas en la PC.

El procesamiento de video puede incluir un amplio rango de efectos, que van desde la eliminación de componentes de color hasta los complejos que incluyen introducir parte de un video(por ejemplo la imagen en movimiento de una persona) dentro de otro video.

Asimismo se puede realizar procesamiento sobre video en tiempo real, y procesamiento de video en modo de edición cuadro por cuadro sobre cada imagen, para su posterior presentación. Este ultimo tipo de procesamiento se realiza en edición de películas para añadir efectos visuales especiales y requiere costosas estaciones de trabajo con mas de un procesador si se quiere ahorro de tiempo aunque una PC lo puede hacer con ciertas restricciones.

En la actualidad se dispone en el mercado de software para edición de video, pero no de software comercial para procesamiento de video en tiempo real para la PC, lo cual constituye un atractivo aliciente para este trabajo. Cabe resaltar que existen sistemas de procesamiento de video en la PC pero estos incluyen tarjetas con procesadores digitales de señales (DSP) externos.

En este proyecto se realiza solo procesamiento de video en tiempo real básico, y dado que la complejidad permisible del procesamiento viene determinada fuertemente por el hardware, se experimenta la capacidad de procesamiento de la computadora personal utilizada para establecer sus limitaciones.

Se elaboraron procesamientos básicos de video que consisten en filtrado de componentes de color, adición de ruido a la imagen, superposición de imagen al video, eliminación de color, posterizado y realce. Algunos efectos que requerían mayor capacidad de procesamiento no pudieron llevarse acabo, sin embargo el aumento en las capacidades del hardware seguramente solventara pronto dichas restricciones.

La interacción entre la televisión y las computadoras es uno de los campos con mayor potencial de desarrollo en la actualidad. Con las tecnologías recientes como el DVD (Digital Video Disc) y la HDTV (High-Definition Television), se amplían las perspectivas de éxito en la fusión de la televisión y la computadora para las próximas décadas.

La versatilidad de la computadora y el software actuales en la elaboración de productos multimedia interactivos, aunada a las imágenes en movimiento que pueden generarse con algún reproductor de video digital o analógico, permiten captar la atención del televidente, brindando así un excelente medio de aprendizaje.

Para la enseñanza en aulas escolares es importante además, la generación de imágenes amplias que faciliten a todos los alumnos la percepción clara de los detalles en texto y esquemas. Esto se logra mediante el uso de cañoneras o video proyectores, sin embargo tienen limitaciones en cuanto a la cantidad de lúmenes que producen, y por ende esta restringido su uso a lugares con poca iluminación.

Es en este contexto que surge el interés por elaborar un dispositivo de bajo costo que permita manipular señales de video en tiempo real provenientes de una video casetera,

una videocámara, un DVD o la computadora personal; y que proporcione una imagen ampliada en un arreglo cuadrado de monitores o televisores para su uso en aulas iluminadas o inclusive al aire libre.

Figura 1.1 Aspecto general de un videowall de 5x5 monitores

Al arreglo de monitores o dispositivos de proyección que están conectados a un procesador digital de señales vía software, se le denomina comercialmente videowall. Sus usos comprenden anuncios de comerciales, presentación de eventos al aire libre, presentación de videos, etc. . El costo de un videowall se incrementa con el numero de monitores que es capaz de manejar. Como ejemplo la empresa VIDEOWALLSUSA (www.vwusa.teklab.it) ofrece un procesador videowall 2x2 (4 monitores) en US$2,495. Por otra parte Minnesota Video Productions (www.tccom.com) ofrece la renta de un videowall 3x3 (9 monitores) por 4 días con algunos servicios adicionales en US$5,500.

Finalmente se ha de combinar el videowall con el procesamiento del video en tiempo real aunque esto añade una gran sobrecarga a la computadora y enfrenta no pocos problemas de diseño de software.

1.2 Definición del proyecto

La figura 1.3 muestra en modo general los subsistemas que comprenden al presente dispositivo.

A) Generación de la señal de video.
B) Conversión de la señal de video a un formato digital.
C) Sistema para el procesamiento de video, utilizando microprocesadores de propósito general.
D) Multiplexor digital encargado de repartir la señal de video procesada hacia cada monitor o televisor
E) Conversión de la señal de video digital a un formato analógico o compatible con el monitor

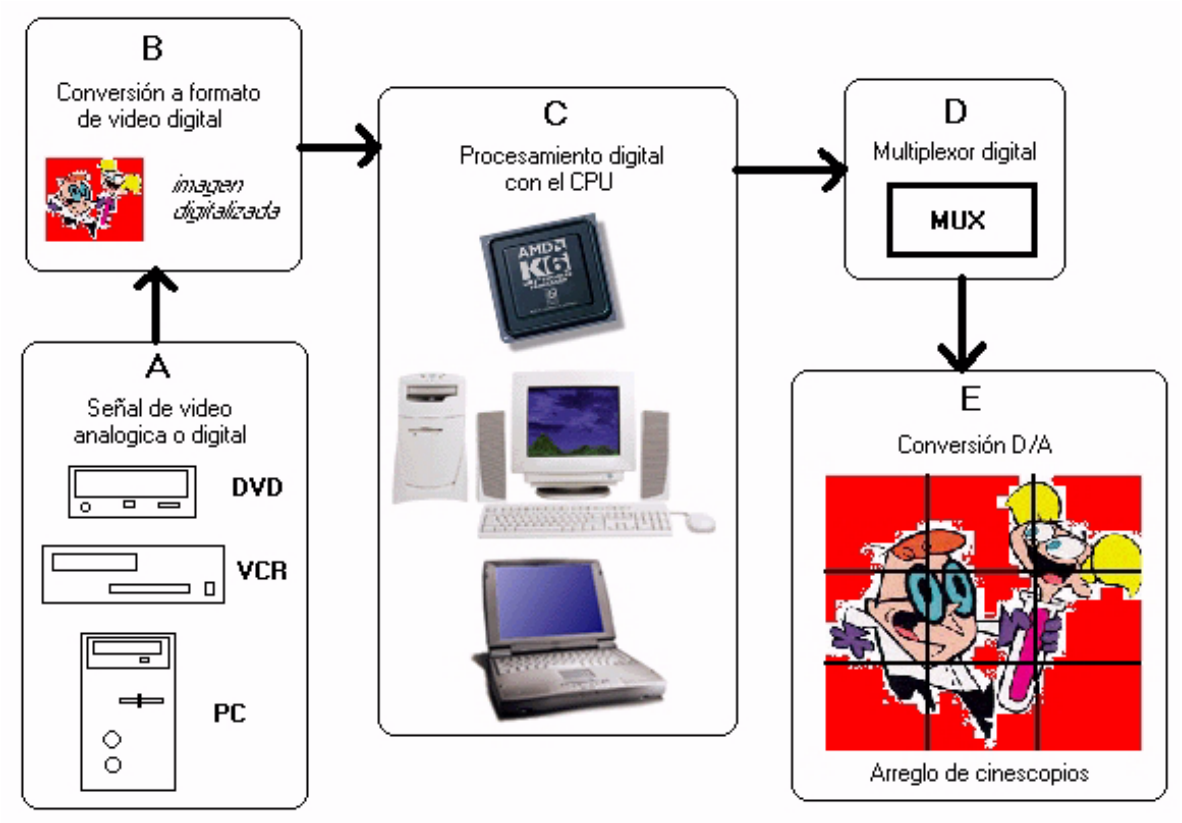


Figura 1.2 Diagrama a bloques del proyecto

Dentro de lo concerniente al bloque B se tiene contemplado el uso de una tarjeta de adquisición de video, la cual realiza la conversión analógica a digital y envía la información directamente a la memoria de video de la computadora, liberando de esta tarea al microprocesador.

En el bloque C de la figura anterior el procesador de propósito general embebido en una computadora personal realizará el procesamiento de video auxiliado por dispositivos para manejo de imágenes como las tarjetas aceleradoras de gráficos AGP.

La programación se llevará a cabo en Visual C++ y empleando librerías DirectX, las cuales permiten un acceso de bajo nivel al hardware multimedia de forma independiente del dispositivo y aprovechan los conjuntos de instrucciones ampliadas para el manejo de audio y video que ofrecen los microprocesadores centrales y gráficos.

La manipulación de las fuentes audiovisuales puede incluir diversos efectos como la mejora en la calidad de la imagen ampliada mediante técnicas de interpolación, la superposición de imágenes y/o subtítulos, el escalamiento de imágenes (aumento o disminución), y el filtrado de componentes de color.

Para el bloque D se tiene pensado elaborar una tarjeta que se inserte en un puerto de la computadora y realice la función de multiplexado para enviar la información digital de video, hacia cada una de las pantallas de televisión.

En lo que respecta al bloque E se tiene pensado elaborar un sistema de conversión de imagen por hardware para cada una de las pantallas o monitores. La presentación de la imagen se dará en un arreglo de 2x2 (4 televisores o monitores de alta resolución para computadora), la magnitud del arreglo responde a la necesidad de minimizar los recursos de la unidad central de proceso(CPU) que hagan realizable el procesamiento de video en tiempo real.

1.3 Objetivos

Objetivo general:

Diseñar e implementar un sistema digital para procesamiento de video en tiempo real, el cual proporcione una sola imagen en un arreglo cuadrado de televisores.

Objetivos particulares:

- Desarrollar el software y el hardware necesario para la conversión de la señal de video analógico o digital, a un formato estándar y manipulable por el microprocesador de propósito general.
- Desarrollar el software para el microprocesador que realice efectos diversos con las señales de video.
- Desarrollar el software y el hardware necesarios para dividir la imagen digital, además de proporcionar señales para cada monitor en el arreglo cuadrado.

1.4 Panorama general

En el capitulo dos se exponen los elementos que nos llevaron a seleccionar el presente proyecto, tomando en consideración el estado del arte y la posibilidad del mismo. El capitulo tres presenta los datos relevantes sobre la investigación realizada que permite el desarrollo práctico del sistema, se incluye terminología básica y conceptos clave para la comprensión del software. Para el capitulo cuatro se tocan los elementos de diseño básicos que posibilitarán la implementación del software. En el capitulo cinco se describen los pasos de desarrollo en software, este comprende la aplicación, el filtro y la interfase con el sistema operativo. Finalmente se expone una evaluación del proyecto con las conclusiones obtenidas.

2. Análisis de fundamentos

2.1 Estado del arte

A últimos días el procesamiento digital de video en tiempo real a tomado gran importancia y sus aplicaciones van desde la edición de películas, eliminación de ruido en la transmisión de video, compresión de video y adición de efectos especiales entre otras.

Diversas universidades en el mundo realizan investigación en esta área y se pueden encontrar una gran cantidad de trabajos que se incrementan día con día.

En la gran mayoría de centros de investigación los dispositivos utilizados para el procesamiento del video son los procesadores digitales de señales DSP(Digital Signal Processors). Estos procesadores de propósito especifico cumplen con altas normas de rendimiento, sin embargo los costos asociados son relativamente altos.

Uno de los centros con mayor producción de trabajos en este campo es el laboratorio de procesamiento de imágenes y video *VIPER* (Video and Image Processing Laboratory) de la Universidad de Purdue. Los trabajos más recientes se pueden encontrar en: http://dynamo.ecn.purdue.edu/~ace/delp-pub.html

Como ejemplo de lo mencionado encontramos investigaciones sobre compresión de video en tiempo real, filtrado y detección de contornos utilizando un sistema basado en el microprocesador TMS320C80 de la empresa Texas Instruments, el cual contiene 5 procesadores programables: Un procesador maestro y cuatro procesadores paralelos. Dicho trabajo se puede revisar en: http://dynamo.ecn.purdue.edu/~ace/c80/c80.html

Figura 2.1 Detección de contornos

En este caso el procesador asociado TMS320C80 tiene un costo de US$993, pero un sistema de desarrollo completo requiere de un kit de herramientas y una computadora adicional cuyo costo puede ascender a los US$4500.

Por otro lado el grupo de investigación de procesamiento de video en el departamento de electrónica e ingeniería eléctrica de la universidad de Dublin http://wwwdsp.ucd.ie realizan el filtrado para eliminar errores en la imagen producidas

durante el proceso de compresión, en todos los casos las pruebas e implementación se hacen en DSP para procesamiento de video.

En el ámbito nacional dentro del IIMAS en la UNAM el Dr. Fabián García Nocetti trabaja en la investigación y desarrollo de arquitecturas computacionales de alto desempeño y de algoritmos eficientes para implementar sistemas paralelos en aplicaciones de tiempo real. En su centro de investigación se desarrollan arquitecturas computacionales heterogéneas, integradas por procesadores paralelos y procesadores digitales de señales (DSPs), lo que permite explotar las características computacionales de los procesadores que la integran, optimizando el mecanismo de comunicación entre los mismos y permitiendo una distribución eficiente de las tareas ejecutadas por los módulos de procesamiento. El uso de arquitecturas heterogéneas ha resultado en un menor tiempo de ejecución de los algoritmos, además se ha contribuido en la generación de metodologías formales para automatizar la paralelización de las aplicaciones, permitiendo integrar sistemas de procesamiento de alto desempeño escalables y reconfigurables. En este caso se da énfasis al procesamiento paralelo para lograr las transformaciones en tiempo real. También se desarrollan métodos para la utilización del cómputo paralelo en la obtención, procesamiento y despliegue de imágenes ultrasónicas. Los resultados han permitido incrementar tanto la velocidad y como la resolución en el proceso de formación de la imagen y aplicar estos desarrollos al área de imagenología ultrasónica.

También del IIMAS se encuentra el **Dr. Julio Solano González**:.dedicado al Estudio y desarrollo de arquitecturas y algoritmos eficientes para el Cómputo de Alto Desempeño, utilizando arquitecturas paralelas homogéneas y heterogéneas, combinando procesadores paralelos con procesadores digitales de señales (DSP's). Los desarrollos descritos han sido aplicados directamente a problemas en los campos del Procesamiento de Señales e Imágenes, específicamente en las áreas de flujometría Doppler e Imagenología Ultrasónica.

Una investigación utilizando DirectShow se encuentra disponible en internet bajo la dirección: http://keeper.warhead.org.uk . El titulo es "DirectShow TV display with Real Time video processing" .

El proyecto combina técnicas de tiempo real para la reducción de ruido y desentrelazado derivado de publicaciones con un carácter puramente teórico. Ambos procesamientos se realizan bajo el marco de DirectShow. El filtro de procesamiento de video es un componente COM y puede ser reutilizado sin modificaciones en otros proyectos DirectShow. El filtro implementa una serie de algoritmos de reducción de ruido y de desentrelazado para ser utilizados en computadoras personales de varias velocidades.

Salvo este último caso no me ha sido posible encontrar algún otro proyecto de investigación que utilice a DirectShow como herramienta de procesamiento de video en tiempo real. Cabe destacarse que la herramienta DirectShow tuvo su primera aparición en la versión 8.0a de DirectX disponible en 1999.

2.2 Herramientas disponibles

En el presente proyecto hemos considerado el uso de una PC con un procesador de propósito general Intel para el procesamiento de video ejecutando el sistema operativo Windows 98. La razón principal es la economía, debido a que el costo de un microprocesador de propósito general es con mucho menor a un DSP, adicionalmente se tiene una gran cantidad de hardware de bajo costo con controladores para Windows 98 que permite adecuar el sistema a nuestras necesidades.

Por otro lado el desarrollo de los modernos procesadores de propósito general, incorporan poderosas funciones. Por ejemplo el "reciente" Pentium III de Intel, incluye la característica SIMD (Single Instruction Multiple Data) en punto flotante, lo cual definitivamente incrementa el rendimiento en aplicaciones multimedia.

Figura 2.2 DirectX

Adicionalmente y hace un par de meses se ha añadido DirectShow como parte de DirectX 8.1, software de desarrollo creado por Microsoft disponible gratuitamente y que permite la manipulación de video. DirectX proporciona una serie de librerías que permiten al desarrollador acceder a las características avanzadas y de alto rendimiento en hardware. Es con esta herramienta que se pretende explotar las capacidades novedosas del procesador de propósito general para lograr el procesamiento de video en tiempo real. Las limitaciones se desconocen y dada su dependencia directa con el hardware no dudo que al termino del proyecto sean muy distintas a las actuales. Mientras tanto se irán describiendo los limitantes de software asociados.

3. Investigación

3.1 Procesamiento digital de imágenes

El concepto de imagen está asociado a una función bidimensional *f(x,y)*, cuya amplitud o valor será el grado de iluminación (intensidad de la luz) en el espacio de coordenadas *(x,y)* de la imagen para cada punto. El valor de esta función depende de la cantidad de luz que incide sobre la escena vista, asi como de la parte que sea reflejada por los objetos que componen dicha escena. Estos componentes son llamados iluminación y reflexión, siendo descritos por *i(x,y)* y *r(x,y)* respectivamente. El producto de ambas funciones proporciona la función *f(x,y)*:

$$f(x,y) = i(x,y)r(x,y)$$

Siendo el intervalo de definición:

$$0 < i(x,y) < \infty$$
$$0 < r(x,y) < 1$$

Por lo que *f(x,y)* estará acotada en:

$$0 < f(x,y) < \infty$$

La naturaleza de la iluminación viene determinada por la fuente de luz, mientras que la reflexión depende de las caracteristicas del objeto en la escena.

Dependiendo de las dimensiones, la función puede ser escalar (como las imágenes en blanco y negro) o vectorial (como las imágenes en color que tienen tres componentes).

La función de imagen *f(x,y)* es digitalizada en la memoria de la computadora, tanto espacialmente como en amplitud. Por lo tanto *f(x,y)* está almacenada en una matriz de $N \times M$ elementos. Tomaremos el origen de coordenadas de la imagen en la esquina superior izquierda, el eje *x* el horizontal y el eje *y* el vertical.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(1,0) & \cdots & f(N-2,0) & f(N-1,0) \\ f(0,1) & f(1,1) & \cdots & f(N-2,1) & f(N-1,1) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f(0,M-2) & f(1,M-2) & \cdots & f(N-2,M-2) & f(N-1,M-2) \\ f(0,M-1) & f(1,M-1) & \cdots & f(N-2,M-1) & f(N-1,M-1) \end{bmatrix}$$

El muestreo es la conversión que sufren las dos dimensiones espaciales de la señal analogica, y que genera la noción de pixel. La cuantificación es la conversión que sufre la

amplitud de la señal analógica, así se genera el concepto de nivel de gris o intensidad. Para el caso de tener 256 niveles de gris (0–255). El 0 corresponde a un objeto no iluminado o que absorbe todos los rayos luminosos que inciden sobre él (negro), y el nivel 255 a un objeto muy iluminado o que refleja todos los rayos que inciden sobre el (blanco).

En el proceso de digitalización es preciso establecer el valor de N y M asi como el numero de niveles de gris asignados a cada pixel. Es una práctica común en el proceso de digitalización de imágenes que estas cantidades sean números enteros potencias de dos. El numero de bits b requeridos para almacenar la imagen digitalizada viene dado por el producto:

$$b = NMg$$

siendo 2^g el numero de niveles de gris.

El histograma contiene el numero de pixeles que tienen el mismo nivel de gris, y puede entenderse como la probabilidad de que un velor de gris determinado aparezca en la imagen.

Un ultimo elemento de la imagen son los planos de bits. Si una imagen tiene 256 niveles de gris, cada uno de los pixeles ocupa un byte (8 bits).

Cada **plano de bit** es una imagen formada por un bit de una determinada posición en el byte para cada pixel. Los planos de bits reflejan bien la influencia del ruido en la imagen. Así puede observarse como para planos de bits mas significativos se pueden distinguir los distintos objetos que forman la imagen, pero para bits menos significativos la impresión que dan es una distribución aleatoria de puntos.

3.1.1 Relaciones entre pixeles

Un pixel p de coordenadas *(x,y)* presenta un total de cuatro vecinos en el plano vertical y horizontal, siendo sus coordenadas:

	$x, y-1$	
$x-1, y$	x, y	$x+1, y$
	$x, y+1$	

Este conjunto de pixeles se denomina vecindad de tipo 4 del pixel p, y se representa por N4(p). Además se puede considerar la existencia de otros cuatro vecinos asociados a las diagonales, cuyas coordenadas son:

$x-1, y-1$		$x+1, y-1$
	x, y	
$x-1, y+1$		$x+1, y+1$

Los cuales se representan por ND(p). La suma de los anteriores define los ocho vecinos del pixel, N8(p).

Mediante el concepto de conectividad se quiere expresar que dos pixeles pertenecen al mismo objeto, por lo que está relacionado con el de vecindad. Dos pixeles están conectados si son adyacentes (vecinos) y si sus niveles de gris satisfacen algún criterio de especificación (por ejemplo ser iguales).

Existen tres tipos:

1. Conectividad-4. Dos pixeles p y q presentan una conectividad-4 si q pertenece al N4(p).
2. Conectividad-8. Dos pixeles p y q presentan una conectividad-8 si q pertenece al N8(p).
3. Conectividad-m (conectividad mixta). Dos pixeles p y q presentan una conectividad-m si:
    a) q pertenece a N4(p), o
    b) q pertenece a ND(p) y el conjunto $N4(p) \cap N4(q)$ es el conjunto vacio.

3.1.1.1 Distancia

Con la distancia se quiere obtener el minimo numero de pasos elementales que se necesitan para ir de un punto a otro. Dados tres pixeles *p*, *q* y *z*, con coordenadas *(x,y)*, *(s,t)* y *(u,v)* respectivamente, se puede definir una función de distancia D si cumple:

- $D(p,q) \geq 0, (D(p,q) = 0, si \quad p = q)$
- $D(p,q) = D(q,p)$
- $D(p,z) \leq D(p,q) + D(q,z)$

Las funciones de distancia usadas comunmente son:
La **distancia euclidea** entre p y q se define como:

$$D_E(p,q) = \sqrt{(x-s)^2 + (y-t)^2}$$

con esta definición, las distancias serán:

$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
2	1	0	1	2
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$

Distancia Manhattan. Se toman solamente en cuenta los vecinos de orden 4.

$$D = |x - s| + |y - t|$$

Las distancias serán por tanto:

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

Con esta definición los vecinos del tipo 4 están a la distancia unidad.

3.1.2 Color

Algunas definiciones básicas para comprender los espacios de colores son:

- Brillo: sensación que indica si un área está más o menos iluminada.
- Tono: sensación que indica si un área parece similar al rojo, amarillo, verde o azul o a una proporción de dos de ellos.
- Coloración: sensación por la que un área tiene un mayor o menor tono.
- Luminosidad: brillo de una zona respecto a otra blanca en la imagen.
- Croma: La coloridad de un área respecto al brillo de un blanco de referencia.
- Saturación: La relación entre la coloridad y el brillo.

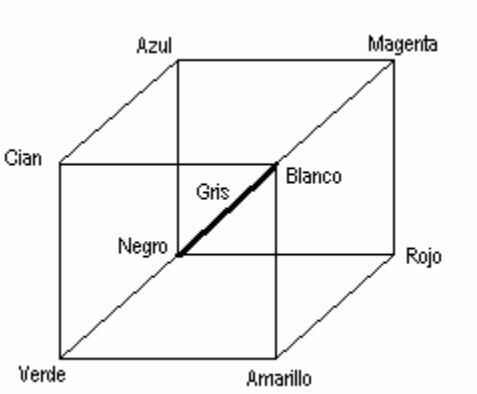
Un **espacio de color** es un método por el que se puede especificar, crear o visualizar cualquier color. Dependiendo del tipo de sensor y aplicación se han desarrollado diversos espacios de colores que se utilizan para la adquisición, la transmisión de las señales (por ejemplo para la televisión), y la impresión o los espacios que tratan de imitar la percepción humana.

3.1.2.1 Espacio RGB

El espacio RGB se basa en la combinación de tres señales de luminancia crómatica distinta: el rojo, el verde y el azul (Red, Green, Blue). La manera mas sencilla e intuitiva de conseguir un color concreto es determinar la cantidad de color rojo, verde y azul que se necesita combinar; para ello se realiza la suma aritmética de las componentes:

$$X = R + G + B$$

Graficamente se representa por un cubo. En la recta que une el origen con el valor máximo se hayan los grises, ya que las tres componentes son iguales.



Cuando una camara adquiere una imagen en color, para cada pixel en color se tienen en realidad tres, uno por cada componente; la ganancia máxima de cada uno de ellos corresponde a la longitud de onda de los tres colores básicos antes nombrados (rojo, verde y azul). Aunque el sistema RGB es el más intuitivo de todos y de hecho es en el que se basan las cámaras para adquirir imágenes en color, presenta un serio inconveniente: en sus tres valores mezcla la información del color (tono y saturación) y la intensidad.

3.1.2.2 Espacio HSI

El espacio de color HSI se basa en el modo de percibir los colores que tenemos los humanos. Dicho sistema caracteriza el color en términos de tono o tinte(Hue), saturación o cromatismo (Saturation) y brillo (Intensity); componentes que se muestran favorables de cara a realizar segmentaciones de la imagen en atención al tono o tinte del color.

Las transformaciones matemáticas que permiten el paso del espacio RGB al HSI son realizadas normalmente mediante hardware especifico. Las ecuaciones son:

$$I = \frac{R+G+B}{3}$$

$$H = \arctan\left(\frac{\sqrt{3}(G-B)}{(R-G)+(R-B)}\right)$$

$$S = 1 - \frac{\min(R,G,B)}{I}$$

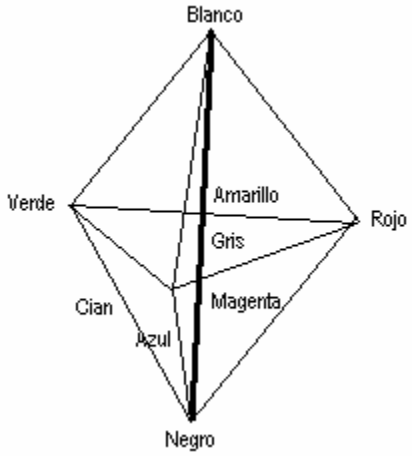
En la espresión del tono

$$H = \arctan\left(\frac{x}{y}\right)$$

se emplea el signo de x e y para establecer el cuadrante al que pertenece el ángulo resultante. Se puede considerar que el tono indica el ángulo formado entre el eje de referencia (el correspondiente al color rojo) y el punto que ocuparía el color a analizar.

Asi como el espacio RGB se representa para un cubo, el HSI lo forman dos piramides unidas por su base. Dependiendo del valor de la intensidad se tendrá un corte con las piramides obteniendose un triángulo. Dentro del triangulo obteniendo la componente H

viene definida por su orientación (el rojo es 0^o , el verde 120^o y el azul 240^o) y la saturación indica la distancia al centro del triangulo. Dentro de la circunferencia obtenida el color viene definido por su orientación (el rojo es 0^o , el verde 120^o y el azul 240^o) y la saturación indica la lejanía del centro del circulo.



3.1.3 Transformaciones matemáticas

Las transformaciones matematicas que se describen acontinuación , son aquellos algoritmos de los que se obtiene una imagen *g(x,y)* a partir de la original *f(x,y)*. El fin perseguido es transformar la imagen original. En este proyecto solo llevaremos a cabo transformaciones en el dominio del espacio.

El procesamiento espacial lo constituyen aquellas técnicas que operan directamete sobre los valores de los píxeles de la imagen. Serán transformaciones del tipo:

$$S(x, y) = F(I(x, y))$$

siendo *I(x,y)* la imagen original, *S(x,y)* la resultante y *F* la transformación. Estas transformaciones suelen aplicarse a un entorno cuadrado del píxel.

3.1.3.1 Convolucion

Para el caso unidimensional, se define como convolución de la función *f(x)* respecto a la función *h(x)* una nueva función *g(x)* tal que:

$$g(x) = h(x) * f(x) = \int_{i=-\infty}^{i=\infty} f(i)h(x-i)di$$

Si se trata del caso discreto con dos secuencias *f(x)* y *h(x)* se obtendría una secuencia de salida *g(x)*:

$$g(x) = h(x) * f(x) = \int_{i=-\infty}^{i=\infty} f(i)h(x-i)$$

Para el caso de imágenes digitales se usan convoluciones bidimensionales discretas, cuya formula es:

$$g(x,y) = h(x,y) * f(x,y) = \sum_{i=-\infty}^{i=\infty} \sum_{j=-\infty}^{j=\infty} f(i,j)h(x-i, y-j)$$

Lo mas normal es usar convoluciones de 3x3 elementos. Entonces la expresión anterior puede concretarse en:

$$g(x,y) = h(x,y) * f(x,y) = \sum_{i=0}^{i=2} \sum_{j=0}^{j=2} f(i,j)h(x-i, y-j)$$

que por ejemplo para g(2,2) será:

$$g(2,2) = \sum_{i=0}^{i=2} \sum_{j=0}^{j=2} f(i,j)h(2-i,2-j) =$$
$$f(0,0)h(2,2) + f(0,1)h(2,1) + f(0,2)h(2,0) +$$
$$f(1,0)h(1,2) + f(1,1)h(1,1) + f(1,2)h(1,0) +$$
$$f(2,0)h(0,2) + f(2,1)h(0,1) + f(2,2)h(0,0)$$

Generalmente en lugar de hablar de convolucionar la imagen *f(x,y)* con la transformación *h(x,y)*, se habla de convolucionar con una máscara que tiene la forma:

h(0,0)	*h(0,1)*	*h(0,2)*
h(1,0)	*h(1,1)*	*h(1,2)*
h(2,0)	*h(2,1)*	*h(2,2)*

h(-i, -j) es entonces:

h(2,2)	*h(2,1)*	*h(2,0)*
h(1,2)	*h(1,1)*	*h(1,0)*
h(0,2)	*h(0,1)*	*h(0,0)*

La porción inicial de la imagen correspondiente es:

f(0,0)	*f(0,1)*	*f(0,2)*
f(1,0)	*f(1,1)*	*f(1,2)*
f(2,0)	*f(2,1)*	*f(2,2)*

Gráficamente se trata de poner las dos matrices, una encima de la otra, multiplicar cada celda por la inferior y sumar los productos. Para obtener el siguiente valor de *g(x,y)* la máscara *h(x,y)* se desplaza un elemento a la derecha y se repite la operación. Al acabar las columnas se vuelve al principio a una fila inferior, y asi sucesivamente hasta llegar al ultimo píxel.

3.1.3.2 Operaciones matemáticas

Las operaciones aritméticas entre dos píxeles p y q son las siguientes:

Suma

Resta
Multiplicación
División

Al realizar dichas operaciones se puede tener algunos problemas como el efecto desbordamiento del píxel (pixel overflow). Las imágenes tienen una resolución que suele ser de 8 bits por lo que el valor mas alto es de 255. Si por una operación se sobrepasa este máximo, el píxel tendrá como nivel de gris, el valor menos 255, por eso después de una operación y antes de escribir los valores en la imagen, hay que normalizar los valores entre 0 y 255(si el tamaño de los píxeles es de 8 bits).

Las operaciones lógicas utilizadas son:
AND
OR
NOT

Es preciso tener en cuenta que las operaciones lógicas sólo se pueden aplicar sobre imágenes binarias, lo cual no ocurre con las operaciones aritméticas.

Las transformaciones geométricas modifican las relaciones espaciales entre los píxeles. Se necesitan dos tipos distintos de algoritmos. El primero es el que determina la relación entre las coordenadas de la imagen original y la imagen resultante. Como caso general, las transformaciones de numeros enteros no tienen por qué dar enteros, asi que será necesario un algoritmo de interpolación que determine el nivel de gris de la imagen final a partir de uno o varios píxeles de la imagen original.

Si se quiere trasladar el origen de la imagen se aplica la ecuación:

$$xf = xi + xo$$
$$yf = yi + yo$$

Que en coordenadas homogéneas es:

$$\begin{bmatrix} xf \\ yf \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & xo \\ 0 & 1 & yo \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} xi \\ yi \\ 1 \end{bmatrix}$$

Magnificación

$$xf = \frac{xi}{a}$$
$$yf = \frac{yi}{b}$$

o el equivalente

$$\begin{bmatrix} xf \\ yf \\ 1 \end{bmatrix} \begin{bmatrix} 1/a & 0 & 0 \\ 0 & 1/b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} xi \\ yi \\ 1 \end{bmatrix}$$

Rotación respecto al origen

$$\begin{bmatrix} xf \\ yf \\ 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -sen\theta & 0 \\ sen\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} xi \\ yi \\ 1 \end{bmatrix}$$

Todas estas transformaciones tomarían mucho tiempo si se tuviera que leer cada píxel, realizar la operación matemática, normalizar el valor a 255 y escribirlo en una nueva imagen. Para acelerar este proceso existen las tablas de consulta (look up tables LUT). Son tablas en las que su índice es el nivel de gris antiguo del píxel y su valor el nuevo nivel de gris que le corresponde. Así todos los calculos se realizan en la parte de inicialización del algoritmo para poder después trabajar en tiempo real.

3.1.4 Ruido

Todas las imágenes tienen una cierta cantidad de ruido, valores distorsionados, bien debidos al sensor CCD de la camara o al medio de transmisión de la señal. El ruido se manifiesta generalmente en píxeles aislados que toman un valor de gris diferente al de sus vecinos. El ruido puede clasificarse en cuatro tipos:

Gausiano. Produce pequeñas variaciones en la imagen y se debe a las diferentes ganancias en el sensor, ruido de los digitalizadores, perturbaciones en la transmisión etc. Se considera que el valor final del píxel sería el ideal más una cantidad correspondiente al error, que puede describirse como una variable gausiana.
Impulsional. El valor que toma el píxel no tiene relación con el valor ideal sino con el valor del ruido que toma valores muy altos o bajos. Se caraceteriza entonces porque el píxel toma un valor máximo, causado por una saturación del sensor, o minimo si se ha perdido su señal. También puede encontrarse si se trabaja con objetos a altas temperaturas, ya que las camaras tienen una ganancia en el infrarrojo de la que no dispone el ojo humano. Por ello las partes muy calientes de un objeto pueden llegar a saturar el píxel.
Frecuencial. La imagen obtenida es la suma entre imagen ideal y otra señal, la interferencia, caracterizada por ser una senoide de frecuencia determinada.
Multiplicativo. La imagen obtenida es fruto de la multiplicación de dos señales.

3.1.5 Deteccion de bordes

Una de las informaciones más utiles que se encuentran en una imagen la constituyen los bordes ya que al delimitar los objetos definen los limites entre ellos y el fondo y entre los objetos entre si.

Las tecnicas usadas en la detección de bordes tienen por objeto la localización de los puntos en los que se produce una variación de intensidad, empleandose para ello

métodos basados en los operadores derivada. Básicamente se tienen dos posibilidades: aplicar la primera (gradiente) o la segunda derivada (laplaciana). En el primer caso se buscarán grandes picos y en el segundo, pasos de respuesta positiva a negativa o viceversa. Se supone que el nivel de gris en el interior de los objetos es constante, esta es una situación que nunca se dará. Es por ello que después de la fase de detección de bordes viene otra que distingue lo que son los bordes del objeto de lo que es ruido. Ello llevará a que en el caso de los métodos basados en el gradiente, la respuesta sea bordes de anchura mayor de un píxel, lo que introduce una incertidumbre en la verdadera localización de los bordes. Este inconveniente no se encuentra en los métodos que se basan en la segunda derivada ya que los bordes vienen definidos por el paso por cero.

3.1.5.1 Tecnicas basadas en el gradiente

Para poder utilizar operadores de este tipo sobre una imagen muestreada es necesario obtener una aproximación del concepto de derivada para espacios discretos. La generaliación comúnmente usada se basa en el cálculo de diferencias entre píxeles vecinos. Estas diferencias, según la relación entre los píxeles considerados, pueden dar lugar a derivadas unidimensionales o bidimensionales, asi como aplicarse en alguna dirección determinada de la imagen o en todas las direcciopnes de forma global.
Se define el operador gradiente G aplicado sobre una imagen *f(x,y)* como:

$$\nabla f(x,y) = \left[G_x, G_y\right] = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

El vector gradiente representa la variación máxima de intensidad para el punto *(x,y)*. Por ello es interesante conocer su módulo y dirección que vendrán dados por:

$$\left|\nabla f\right| = \sqrt{G_x^2 + G_y^2}$$

$$\angle \nabla f = \arctan\left(\frac{G_y}{G_x}\right)$$

siendo la dirección del gradiente perpendicular al borde.

Debido al coste computacional el módulo a veces se simplifica por:

$$\left|\nabla f\right| = \left|G_x\right| + \left|G_y\right|$$

Como las imágenes digitales no son señales continuas, la nueva expresión del gradiente viene dad por:

$$\nabla f(x,y) = \left[G_x, G_y\right] = \left[\frac{\Delta f}{\Delta x}, \frac{\Delta f}{\Delta y}\right]$$

que se puede representar por las mascaras:

$$G_x = \frac{\Delta f}{\Delta x}$$

-1	1	*f(x,y)

$$G_x = \frac{\Delta f}{\Delta x}$$

-1	*f(x,y)
1	

Sin embargo estas máscaras no suelen utilizarse debido a que son muy sensibles al ruido al tener en cuanta solamente la información de dos píxeles. Con el tiempo han aparecido otros filtros que además de calcular el gradiente tienen cierto efecto de suavizado, o lo que es lo mismo, son sensibles al ruido.

Operadores de Roberts

0	-1
1	0

0	-1
1	0

Operador de Prewitt
Expande la definición de gradiente a un entorno de 3x3 para ser mas inmune al ruido

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Operador de Sobel
Sobel da mas importancia a los pixeles centrales que el operador de Prewitt.

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Operador isotrópico
Mientras que Prewitt detecta mejor los bordes verticales, Sobel lo hace en los diagonales. El operador isotropico intenta llegar a un equilibrio entre ellos.

-1	0	1
$-\sqrt{2}$	0	$-\sqrt{2}$
-1	0	1

-1	$-\sqrt{2}$	-1
0	0	0
1	$-\sqrt{2}$	1

Operadores de segundo orden

La laplaciana es la segunda derivada de una función y representa la derivada de esta respecto a todas las direcciones:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Las dos definiciones que de forma mas frecuente se utilizan para el operador laplaciana son:

-1	-1	-1
-1	8	-1
-1	-1	-1

0	-1	0
-1	4	-1
0	-1	0

Donde el pixel central toma el valor negativo de la suma de todos los que le rodean.

3.2 Hardware

Los elementos de hardware utilizados para la elaboración del videowall son básicamente además de la computadora, las tarjetas de video adicionales y sus respectivos monitores.

3.2.1 Monitores

Se realizó una comparación entre varios monitores de computadora cuyos precios oscilaban entre $1100 y $1600 obteniéndose la siguiente tabla:

					
Marca	BENQ	DAEWOO	DTS	LG	SAMSUNG
Modelo	V551	531X	1528	563N	551V
Precio	$1,366	$1,238	$1,298	$1,329	$1,206
Tipo	CRT color	CDT	Convencional	CRT color	CRT color
Área visible	14"	14"	13.9"	13.8"	13.9"
Tratamiento	Ninguno	Antiestático, antireflejante	ninguno	Anti Glare	Anti Glare
Resolución	1024x768	1024x768	1280x1024	1024x768	1024x768

El principal motivo de la selección reside en la resolución permisible, dado que esto expone mayor calidad en la imagen. Los tratamientos efectuados sobre la superficie del cinescopio influyen principalmente en los reflejos externos producidos, sin embargo no es

una característica mas importante que la anterior. Los monitores adquiridos son entonces de marca DTS y de modelo 1528.

3.1.2 Tarjetas de video

Las tarjetas de video se han escogido de diferentes tipos con la intención de obtener sus respectivos rendimientos y realizar comparaciones. Las mas comunes en el mercado fueron seleccionadas teniéndose solo en consideración que fuesen 3 tarjetas con puerto PCI y una tarjeta con puerto AGP.

				
Marca y modelo	ATI 3D Rage Pro	3Dblaster Savage 4	Voodoo 4 4500 PCI	Cirrus Logic 5446PCI
Memoria de video	4MB	8MB	32M	2M
Bus	PCI	AGP	PCI	PCI
Resolución máxima	1600x1200	1600x1200	1600x1200	1024x768
Características adicionales	Incluye acelerador gráfico de 64 bits con soporte de 2D, video y 3D	Soporta Directraw 2D y 3D	Acelerador gráfico de 128bits 2D/3D/Video	

Para la adquisición de video se ha contemplado el uso de la tarjeta WinTV-PCI de la empresa Hauppauge (www.hauppauge.com) con las siguientes características técnicas:

Figura 3.1 Tarjeta de adquisición de video

- Conectores de entrada auxiliares de audio/video para video cámaras, VCR u otras fuentes similares.
- Captura imágenes con calidad de 24-bits y digitalización YUV 4:2:2 en un formato de color a 60 frames por segundo. El máximo tamaño de imagen digitalizada es de 640x480 para fuentes de video NTSC.
- El video digitalizado se envía sobre el bus PCI hacia la memoria de la tarjeta de presentación VGA, sin necesidad de uso del CPU.

3.2.3 Procesador

Se ha seleccionado el microprocesador Pentium III de Intel, a 550MHz, debido a que en el momento presente expone las mejores prestaciones y capacidad de procesamiento para aplicaciones de video en tiempo real. En particular, el Pentium III incluye instrucciones Single Instruction Multiple Data (SIMD) en punto flotante, las cuales pueden incrementar significativamente la velocidad de los algoritmos de procesamiento. Por otro lado dispone de la tecnología MMX (Matrix Math extensión) la cual es muy similar a las instrucciones SPARC VIS que realizan operaciones con enteros en vectores de palabras de 8, 16, o 32bits. Esto facilita el procesamiento de video así como la estimación de movimiento y la interpolación.

3.3 Soporte de multimonitor

La característica **multimonitor** le permite al sistema operativo controlar mas de una tarjeta de video conectada en la misma tarjeta madre. Esta característica reduce considerablemente los costos y complejidad del videowall, y es soportada por Windows 98/Me/2000, y también por LINUX en sus versiones 7.0 de la distribución Mandrake y superiores. En todos los casos las tarjetas deben ser PCI o AGP, dado que las anteriores ISA/EISA/VESA no son soportadas. Se ha escogido Windows 98 debido a que soporta una mayor cantidad de hardware y considerando la herramienta de desarrollo disponible DirectX.

Para cada monitor se puede tener su propia resolución y cantidad de colores, y Windows permite especificar la posición de cada monitor respecto a cualquier otro. Una desventaja es que el modo de **pantalla-completa** (que se refiere al modo de video que se obtendría en una sesión de DOS en toda la pantalla o de una aplicación DirectX que ocupe toda la pantalla) solo se puede obtener en un solo monitor denominado monitor principal. Este es un problema que se espera resuelvan los desarrolladores de DirectX en el futuro.

También ocurre que el uso de múltiple monitor resulta en un menor desempeño del sistema, dependiendo de lo que se esté realizando y es uno de los elementos que queremos carácterizar.

3.4 DirectX

DirectX® de Microsoft® es el software que permitirá manipular el hardware del sistema a bajo nivel. DirectX ofrece un componente de desarrollo denominado DirectShow®, el cual implementa librerías para la manipulación de elementos multimedia. Desafortunadamente no se dispone actualmente de bibliografía para el uso de DirectShow, y toda la información al respecto ha sido obtenida del sitio oficial de DirectX:
http://www.microsoft.com/directx

DirectX proporciona acceso de bajo nivel al hardware multimedia de forma independiente del dispositivo, adicionalmente aprovecha los últimos desarrollos en

hardware a medida que se van originando como es el caso de la tecnología MMX de Intel®. Esto se logra simplemente instalando la ultima versión de DirectX.

DirectX incluye varios componentes (Figura 3.2) entre los que se encuentran:

- DirectDraw®. Este proporciona animación realista usando intercambio de paginas de video, acceso a coprocesadores gráficos especializados y administración de la memoria de video. También sirve de base para otros componentes como DirectShow® y Direct3D®.
- Direct3D®. Proporciona interfaces de alto y bajo nivel para generar polígonos con texturas en 3D por software y por hardware.
- DirectSound®. Proporciona sonido estéreo y 3D con mezcla de sonido por hardware, así como administración de la memoria de la tarjeta de sonido.
- DirectPlay®. Incluye servicios transparentes de mensajeria independientes del medio para crear juegos con varios jugadores, así como las funciones necesarias para organizar y ejecutar un juego multijugador.
- DirectShow®. Proporciona una interfaz para el manejo de flujos multimedia provenientes de archivos o dispositivos de adquisición de audio y video.

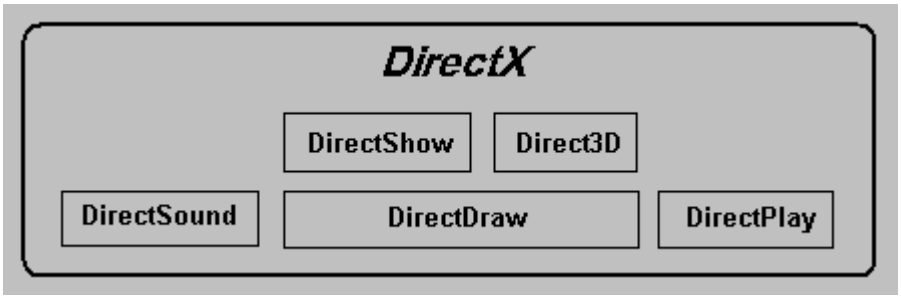


Figura 3.2 Componentes de DirectX

Directshow es la herramienta a utilizar en el presente trabajo, y para su comprensión es necesario revisar los conceptos sobre el modelo de objetos componentes COM desarrollado por Microsoft.

3.5 El modelo de componentes COM

COM (Component Object Model) es un estándar creado por Microsoft que define a nivel binario como los objetos se deben crear, destruir e interactuar entre ellos. El hecho de que sea un estándar a nivel binario implica que es independiente del lenguaje que se use para trabajar con los objetos, siempre y cuando el lenguaje en cuestión sea capaz de manejar apuntadores a funciones.

Los componentes de software COM son independientes de cualquier aplicación y residirán en el sistema en forma de DLL(Dynamic Link Library). Cada componente dispone de un identificador global único o GUID(globally unique identifier) que lo caracteriza(este identificador consiste de un entero de 128 bits y se puede generar mediante el programa guidgen.exe). Si el componente está registrado en el sistema, entonces

podremos crear un objeto concreto de esa clase. Estar registrado en el sistema significa que el sistema operativo conoce donde reside "físicamente" la implementación de la clase identificada por el GUID. En el sistema Windows esta información se almacena en una base de datos centralizada llamada **registro**. Este registro se utiliza para configurar el software y hardware que se ejecute en el ambiente Windows. De esta manera, podemos disponer de toda una librería de componentes que podemos usar libremente, siempre y cuando estén registrados en nuestro sistema.

Una **interfase** es un conjunto de métodos (funciones) y atributos (datos) que tienen una fuerte relación lógica entre ellos. Un **componente** es la implementación de **una o más** interfaces y queda definida por las interfaces que implementa. La herencia de un componente se concibe a nivel de interfaces. Así todas las interfaces derivan de la interfase **IUnknown**. Esta interfase se encarga de la gestión de memoria de los objetos. La gestión de memoria se basa en un contador de referencias, y la interfase IUnknown esta formada por tres métodos **AddRef**, **Release** y **QueryInterface** para controlar este contador. Las dos primeras permiten llevar la cuenta de si el componente está siendo utilizado y cuantos clientes lo están utilizando; de esta forma el sistema puede descargar un componente cuando no se esté utilizando. La tercera permite averiguar si el objeto en cuestión implementa una interfase en particular o no, además de obtener un acceso al objeto a través de esa interfase.

3.5.1 Creación y gestión de objetos

Los objetos se crean mediante la función **CoCreateInstance**, y se acceden a través de apuntadores a interfaces. Inicialmente, obtenemos un apuntador a una interfase a raíz de la creación del propio objeto. Posteriormente, podemos acceder a las otras interfaces que se implementen con el método QueryInterface de ésta. Por ejemplo, si tenemos un componente identificado por el GUID GUID_MYCOMPONENT, que implementa las interfaces InterfaceA e InterfaceB podemos escribir el siguiente código:

```
InterfaceA *pIA;
InterfaceB *pIB;

CoInitialize(NULL);
CoCreateInstance(GUID_MYCOMPONENT, NULL, CLSCTX_INPROC, GUID_INTERFACEA, (LPVOID *) &pIA);

// Ahora podemos acceder a los métodos y atributos de la InterfaceA a través de pIA
pIA->QueryInterface(GUID_INTERFACEB, (LPVOID *) &pIB);

// ... Ahora también podemos acceder a la InterfaceB a través de pIB
pIA->Release();

// ... Ahora el objeto aún no se ha destruido porque aún tenemos una interfase activa
pIB->Release();

// ... Ahora sí que se destruye el objeto
CoUninitialize();
```

Podemos entonces comentar que:

- Primero, tenemos que comunicar al sistema que nuestra aplicación va a ser cliente (o usuaria) de objetos COM mediante la función **CoInitialize**, que se deberá llamar

al principio del programa. De la misma manera, hay que comunicar que dejamos de ser clientes mediante **CoUninitialize**.

- Por otro lado, observamos que las interfaces también tienen un GUID asociado que las identifica de forma universalmente única, es decir, no debería existir ninguna otra interfase que tenga el mismo GUID.
- El parámetro **CLSCTX_INPROC** especifica el **contexto de ejecución** del objeto, es decir, si se ejecutará dentro del mismo espacio de proceso de la aplicación, fuera de éste, etc. En general siempre especificamos que se ejecute dentro de nuestro proceso.
- Las referencias a CoCreateInstance y QueryInterface ya incrementan el contador de referencias, por lo que no es necesario hacerlo explícitamente con AddRef. En cambio, sí es necesario liberar los apuntadores a interfase llamando a Release.

3.5.2 Gestión de errores

En el modelo COM se ha estandarizado la gestión de errores. Todas las funciones COM y métodos de objetos COM devuelven un valor de tipo **HRESULT** con un código de error. Si todo ha funcionado correctamente, este valor es **S_OK**. En cambio, si ha habido algún error, éste código dependerá de la función o método que sea invocado, ya que cada una puede generar tipos diferentes de errores. Se debe consultar la ayuda si se desea hacer un tratamiento exhaustivo de los diferentes tipos de errores.

En general, habrá que controlar como mínimo el resultado de la función CoCreateInstance y del método IUnknown::QueryInterface, ya que éstos devuelven apuntadores a interfaces COM a través de las que accederemos al objeto. Si hay algún error en estos casos, el apuntador a interfase que obtendremos será erróneo, y en el momento en que queramos llamar a alguno de sus métodos provocaremos un acceso a memoria no válido.

Explicado el funcionamiento de COM, comencemos a ver las características de DirectShow.

3.6 DirectShow

DirectShow es un paquete para desarrollar software que permite obtener *flujos* de datos de video y/o audio a partir de archivos o desde dispositivos de hardware como tarjetas de sonido y de video conectados a la computadora. Este soporta una amplia variedad de formatos, incluyendo ASF(Advanced Streaming Format), MPEG(Motion Picture Experts Group), AVI(Audio-Video Interleaved), MP3(MPEG Audio Layer-3), y archivos WAV. También soporta captura de video utilizando dispositivos WDM(Windows Driver Model) o antiguos dispositivos Video para Windows. DirectShow esta integrado junto con otras tecnologías DirectX. Detecta automáticamente y utiliza el hardware para la aceleración de audio y video cuando se encuentren disponibles en el sistema.

Adicionalmente permite la elaboración de componentes de software que realicen algún procesamiento (compresión, escalamiento, filtrado), sobre dicho flujo en tiempo real. Las aplicaciones se basan en un sistema modular de componentes llamados *filtros*. El filtro es un objeto COM y se ubica en un arreglo denominado *gráfico de filtros* (filter graph) el cual proporciona además una idea global del funcionamiento e interacción entre filtros.

3.6.1 Gráfico de Filtros

El bloque de construcción básico de DirectShow es un componente de software llamado *filtro*. Un filtro generalmente realiza una sola operación en un flujo multimedia. Por ejemplo, hay filtros que realizan las siguientes tareas:

- Obtiene video desde un dispositivo de captura de video.
- Lee archivos
- Decodifica un formato particular del flujo, como video MPEG-1.
- Pasa datos hacia la tarjeta de video o de sonido.

Un filtro recibe entrada(*input*) y producen salida(*output*). Por ejemplo, si un filtro decodifica video MPEG-1, la entrada es un flujo MPEG-codificado y el salida es un flujo de video RGB no comprimido. Para realizar una tarea dada, una aplicación conecta varios filtros donde el salida de un filtro es la entrada de otro. Un conjunto de filtros conectados se denomina gráfico de filtros(*filter graph*). Como una ilustración de este concepto, la figura 3.3 muestra un gráfico de filtros para ejecutar un archivo AVI.

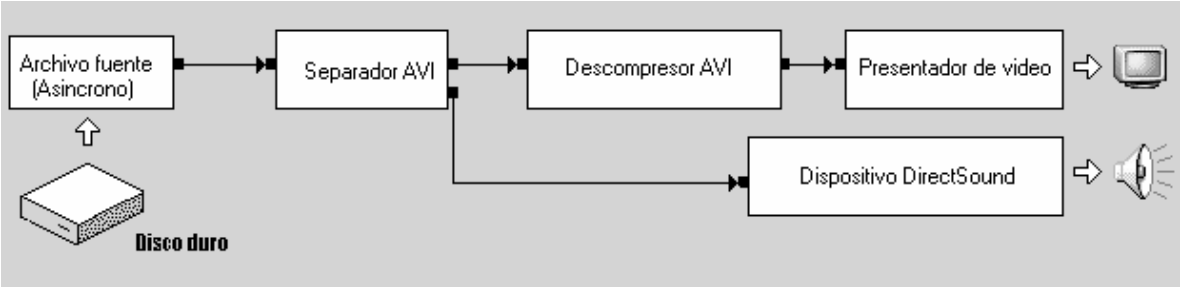


Figura 3.3 Gráfico de filtros

La aplicación no tiene que manejar los filtros individuales en el gráfico de filtros. En lugar de eso, DirectShow provee de un componente llamado *Manejador del Gráfico de filtros(Filter Graph Manager)*. El Manejador del Gráfico de filtros controla el flujo de datos en el grafico. Una aplicación de alto nivel solo realiza llamadas tales como "Run" (para mover los datos en el grafico) o "Stop" (para detener el flujo de datos). Si se requiere un control más directo de las operaciones sobre el flujo, se puede acceder directamente a los filtros a través de las interfaces COM. El Manejador del gráfico de filtros pasa notificaciones de eventos a la aplicación, de modo que la aplicación puede responder a eventos tales como el fin de flujo.

3.6.2 Aplicaciones DirectShow

Una aplicación típica DirectShow realiza tres pasos básicos, como ilustra la figura 3.4.

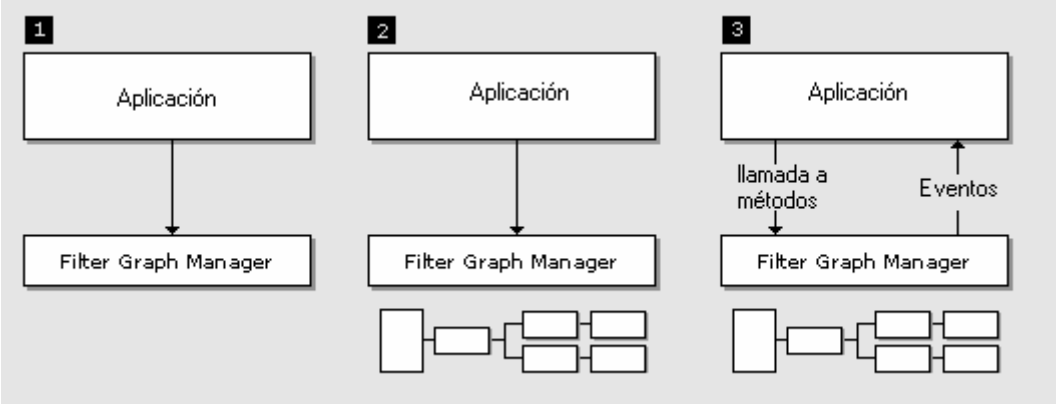


Figura 3.4 Desarrollo de una aplicación en DirectShow

1. Crea una instancia del Manejador del Gráfico de filtros, usando la función **CoCreateInstance**.
2. Utiliza el Manejador del gráfico de filtros para construir un gráfico de filtros.
3. Controla el gráfico de filtros y responde a eventos.

El Manejador del gráfico de filtros inspecciona la conexión de los filtros en el gráfico de filtros y controla el flujo de datos. Las aplicaciones del usuario controlan las actividades del grafico de filtros mediante la comunicación con el Manejador del Gráfico de filtros. Para elaborar un filtro se requiere el uso de programación COM y utilizar las clases base en las librerías de DirectShow implementadas en C++. En el presente trabajo será necesaria la construcción del filtro y la construcción de la aplicación, pero antes veamos mas detalles sobre el funcionamiento de DirectShow.

3.7 Componentes de DirectShow

Trabajar con multimedia presenta varios problemas principales.

- Los flujos multimedia contienen grandes cantidades de datos que deben ser procesados a altas velocidades.
- El audio, video y cualquier flujo adicional deben estar todos sincronizados para iniciar y detenerse todos al mismo tiempo y ejecutarse a la misma rapidez.
- Los flujos pueden venir desde muchas fuentes, incluyendo archivos multimedia locales, tarjetas de adquisición de video/audio, videocámaras y otros dispositivos.
- Los flujos vienen en una variedad de formatos, tales como AVI (Audio-Video Interleaved), ASF(Advanced Streaming Format), MPEG(Motion Picture Experts Group), (DV)Digital Video, y MJPEG(Motion JPEG).

Para llevar a cabo la tarea de proveer flujos de video y audio de manera eficiente hacia las tarjetas graficas y de sonido, DirectShow utiliza DirectDraw® y DirectSound®. La sincronización se lleva a cabo encapsulando los datos multimedia en muestras con información temporal. Para manejar la variedad de fuentes, formatos, y dispositivos de hardware, DirectShow utiliza una arquitectura modular en la cual componentes llamados filtros, pueden ser mezclados y acoplados para realizar tareas distintas.

DirectShow incluye filtros que soportan captura de multimedia y dispositivos sintonizadores basados en Windows Driver Model (WDM) así como también filtros que soportan Video for Windows (VfW). LA figura 3.5 muestra la relación entre una aplicación, los componentes DirectShow, y algunos de los componentes hardware y software que DirectShow soporta.

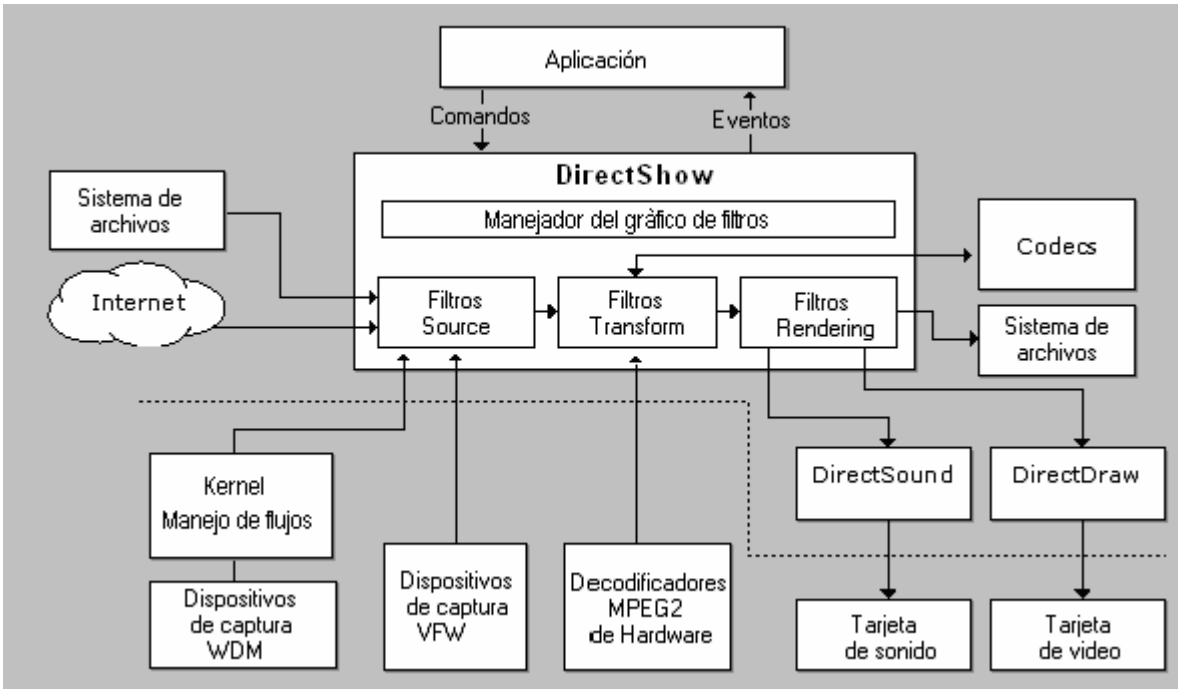


Figura 3.5 Esquema global de DirectShow

DirectShow proporciona filtros para ejecutar archivos y flujos provenientes de varias fuentes, incluyendo archivos locales, CD, Internet, dispositivos DVD, sintonizadores de TV y tarjetas de captura de video. DirectShow tiene compresores nativos, descompresores para algunos formatos de archivos, y muchos decodificadores de hardware y software compatibles con DirectShow. La reproducción hace completo uso de las capacidades de aceleración en hardware cuando este lo soporta.

3.7.1 El grafico de filtros y sus componentes

El Manejador del Gráfico de filtros provee un conjunto de interfaces COM para que las aplicaciones puedan acceder al gráfico de filtros. Así las aplicaciones pueden llamar a las interfaces del Manejador del Gráfico de filtros para controlar los flujos.

Un gráfico de filtros se compone de 3 tipos distintos de filtros:
- **Filtros Source**, toman datos provenientes de una fuente y los introducen a un gráfico de filtros.
- **Filtros Transform**, toma datos, los procesa y los envía.
- **Filtros Rendering**, proporciona los datos, a un dispositivo hardware para su presentación al usuario, pero puede enviarlos hacia cualquier localidad que acepte datos multimedia como memoria o disco.

Para que el gráfico de filtros trabaje, los filtros deben ser conectados en el orden apropiado y el flujo de datos comenzar y parar en dicho orden.

Controlar el flujo significa iniciar, pausar o detener el flujo multimedia. El Manejador del gráfico de filtros permite a la aplicación especificar estas actividades y llama a los métodos apropiados en los filtros. También la aplicación puede obtener el estado actual de los filtros, esta relación se puede apreciar en la figura 3.6 .

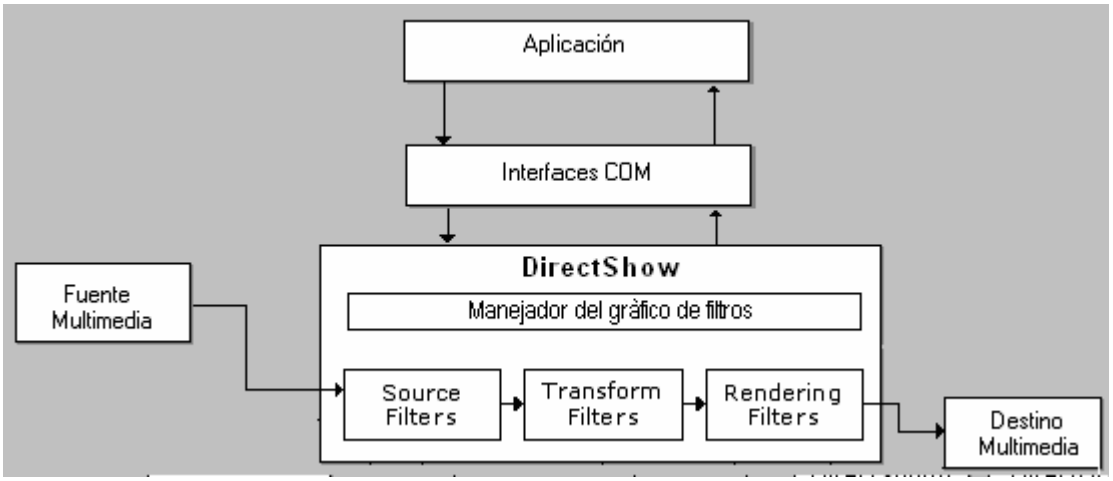


Figura 3.6 Interacción del gráfico de filtros con la aplicación

Un filtro es un objeto COM que realiza una tarea especifica. Para cada flujo se tiene al menos un pin. Un *pin* es un objeto COM creado por el filtro, que representa un punto de conexión para un flujo unidireccional de datos en el filtro.

3.7.2 Filtros

Los filtros son los bloques básicos de construcción en DirectShow. DirectShow separa el procesamiento de datos multimedia en una serie discreta de pasos, y un filtro representa un (o algunas veces mas de uno) paso de procesamiento. Esto permite a las aplicaciones "mezclar y acoplar" filtros para realizar distintos tipos de operaciones en diferentes formatos multimedia y utilizando diversos dispositivos de hardware y software. Por ejemplo el filtro Async File Source (Archivo fuente asíncrono) lee un archivo de un disco, el filtro TV Tuner (Sintonizador de TV) cambia el canal en una tarjeta de captura de

TV, y el filtro MPEG-2 Splitter (Bifurcador MPEG-2) *parsea*[1] los datos de audio y video en un flujo MPEG, así estos pueden ser decodificados. Aunque cada uno de estos filtros hace algo único internamente, desde el punto de vista de una aplicación, cada uno es solo un filtro DirectShow con ciertas características estándar, a saber: soporte para la interfaz IBaseFilter y uno o más pins de entrada y/o pins de salida que representan las conexiones a uno o más filtros DirectShow.

Todos los filtros caen en una de estas tres categorías: filtros source, filtros transform, y filtros renderer. Veamos las características de cada uno de ellos.

3.7.2.1 Filtros Source

Los filtros source presentan los datos multimedia crudos[2] para procesamiento. Ellos pueden obtenerlos de un archivo en un disco duro, o desde un CD o DVD, también pueden obtenerse desde una fuente "live" tal como una tarjeta receptora de televisión o una tarjeta de captura conectada a una cámara digital. Algunos filtros source simplemente pasan los datos crudos hacia un filtro parseador o un filtro splitter, mientras otros filtros también realizan el parseo.

3.7.2.2 Filtros Transform

Los filtros transform aceptan datos crudos o datos parcialmente procesados y realizan un procesamiento de dichos datos. Hay muchos tipos de filtros transform incluyendo parseadores que dividen los flujos de datos crudos en muestras o frames[3], filtros compresores, descompresores, y convertidores de formato.

3.7.2.3 Filtros Renderer

Los filtros renderer generalmente aceptan datos completamente procesados y ejecutan estos en el monitor o a través de las bocinas, o posiblemente a través de algún dispositivo externo. En esta categoría están incluidos los filtros "file-writer" que salvan datos a disco u otro elemento de almacenamiento. Los filtros renderer de video usan DirectDraw para desplegar video y el filtro renderer de audio usa DirectSound para ejecutar audio.

3.7.3 Pins

Los pins son responsables de proveer interfaces para conectarse con otros pins y para transportar los datos. Las interfaces de los pins soportan:

---

[1] Término que se entiende como la división de la entrada en partes pequeñas mas faciles de procesar y/o analizar.

[2] Traducción de raw. En este contexto se entiende como datos sin procesamiento alguno provenientes quizas de un convertidor A/D con un formato de bytes sin codofocar.

[3] Entendemos por frame una imagen digitalizada compuesta por un arreglo cuadrado n x m de pixeles

- La transferencia de datos entre filtros usando memoria compartida u otro método.
- La negociación del formato de datos para cada conexión pin a pin.
- La manipulación del buffer y negociación de su localización, con la intención de minimizar la copia de datos y maximizar su transferencia.

Las interfaces pin difieren poco dependiendo si son de entrada o salida.

Figura 3.7 Conexión de pins

3.7.4 Muestras multimedia

Después que los datos crudos han sido colocados en el grafico, provenientes de un archivo local o de una tarjeta de captura, los bytes deben ser codificados en unidades básicas llamadas muestras multimedia(*media samples*). Algunas veces el filtro source hace la codificación y a veces un filtro aparte realiza dicha tarea. Una muestra multimedia es contenida por un objeto COM que implementa IMediaSample2. Adicionalmente a los datos multimedia actuales, el objeto contiene información incluyendo el tipo de media especifico y los tiempos de sincronización. Un objeto muestra multimedia que contenga datos de video, mantendrá los datos para un frame de video. Para audio, contiene los datos de varias muestras de sonido. En cualquier caso, cuando los datos se mueven desde el filtro source hacia el render (*downstream*[4]) a través de un gráfico, se hace en la forma de objetos media sample.

3.7.5 Asignadores

Cuando dos filtros se conectan, sus pins deben ponerse de acuerdo en los detalles de cómo los objetos media sample serán transportados desde el filtro *upstream*[5] hacia el filtro downstream. En este contexto "conectar" significa determinar el tamaño, localización y numero de muestras que serán utilizadas. El tamaño de las muestras dependerá del tipo de media y el formato, la localización del buffer puede estar en memoria principal o en el dispositivo hardware tal como una tarjeta de captura de video. La creación y manipulación de las muestras se realiza por un *asignador,* este es un objeto COM usualmente creado por el pin de entrada en el filtro downstream. Para la mayoría de los casos, los detalles de la localización de los buffers serán completamente transparentes a las aplicaciones. Debe aclararse que el "movimiento" de datos en un gráfico de filtros no siempre involucra una operación de copia.

[4] Usaremos downstream para indicar un flujo de datos "normal" desde el filtro que hace la adquisición de datos hacia el filtro que hace la presentación o almacenamiento de los mismos. En algunas ocasiones también lo entenderemos como el filtro adyacente hacia el cual se dirige el flujo de datos

[5] El término upstream indica un flujo de datos opuesto al downstream.

3.7.6 Relojes

En cualquier operación relacionada con multimedia, es vital sincronizar las muestras, así los frames de video serán desplegadas a la razón apropiada, de modo que el flujo de audio no se adelantará al video ni lo opuesto. Un gráfico de filtros DirectShow contendrá exactamente un reloj que todos los filtros utilizarán como su base de tiempo. El Manejador del gráfico de filtros selecciona un reloj (o provee uno si es necesario) e informa a todos los filtros para que utilicen este reloj como fuente de sincronización.

3.8 Flujo de datos en el Gráfico de filtros

3.8.1 Samples y Buffers

Cuando dos pins se conectan, los datos pueden moverse desde el salida pin hacia el pin entrada. El pin salida entrega datos, mientras el pin entrada recibe datos. La dirección del flujo de datos, desde el pin salida hacia el pin entrada se denomina *downstream*, y la dirección opuesta es llamada *upstream*.

El tipo de los datos que se desplazan entre dos pines depende de la implementación de los pines. En la mayoría de los casos, los pines trabajan con datos multimedia que se encuentran en la memoria principal. Sin embargo son posibles otras configuraciones. Por ejemplo, si dos filtros controlan una parte del hardware de video, el hardware quizás se encargue de los datos de video, con los pins intercambiando información de control. El tipo de datos, y como se mueven estos entre los pines, se denomina *transporte*(transport). Esta sección se avoca al caso donde los datos multimedia son contenidos en la memoria principal, llamada *memoria local de transporte* (local memory transport).

En la memoria local de transporte, los datos son empacados en objetos discretos llamados muestras multimedia(media samples). Una muestra multimedia es un objeto COM que mantiene un apuntador a un buffer de memoria. Una muestra multimedia soporta la interfaz **ImediaSample**.

Otro objeto COM es el *asignador de memoria* (memory asignator), responsable de ubicar buffers y crear muestras multimedia. En el tiempo de conexión, el asignador reserva memoria para los buffers. El asignador también crea un conjunto de muestras multimedia, y da a cada muestra multimedia un apuntador a una dirección dentro del bloque de memoria. Mientras el buffer no sea liberado, el asignador mantiene una lista de cuales muestras están disponibles. Cualquier filtro que necesite una muestra nueva, la solicita al asignador. Después de que la muestra es procesada, la muestra regresa a la lista.

Este mecanismo reduce la cantidad de memoria, debido a que los filtros re-utilizan los mismos buffers. Esto también previene a los filtros de la escritura accidental sobre los

datos que no han sido procesados, debido a que el asignador mantiene la lista de las muestras disponibles. Finalmente este provee una manera eficiente de mover datos a través del grafico: Cuando un pin salida entrega una muestra, este pasa un apuntador a la interfaz **IMediaSample** de la muestra. Esto provoca que no tenga que copiar ningún dato.

La figura 3.8 muestra la relación entre el asignador, las muestras multimedia, y el filtro.

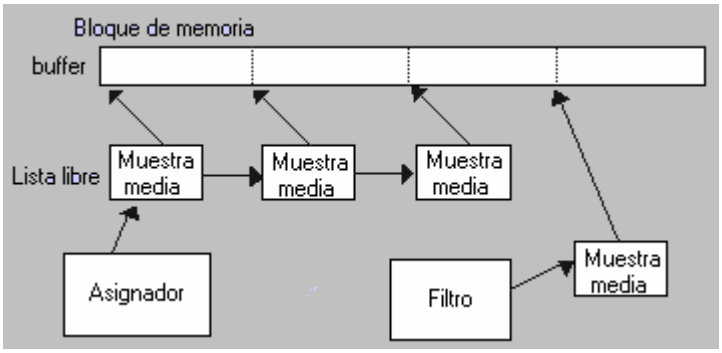


Figura 3.8 Función del asignador

Los pins conectados comparten un asignador de memoria. Un filtro puede usar diferentes asignadores para sus entradas y salidas. Esto es típico cuando el filtro expande los datos, como en el caso de un filtro descompresor. Si las salidas no son mayores que las entradas, el filtro puede procesar los datos en el mismo lugar, sin moverlos hacia un nuevo buffer. En este caso, dos o más conexiones de pines pueden compartir el mismo asignador.

3.8.2 Entrega de Muestras

Los pins entrada que soportan memoria local de transporte exponen la interfaz **ImemInputPin**. El pin salida entrega una muestra al llamar el método **ImemInputPin::Receive** en el pin entrada. El pin entrada realiza una de las siguientes funciones:

- Rechaza la muestra
- Se bloquea hasta que se finaliza el procesamiento de una muestra.
- Regresa inmediatamente y procesa la muestra en otro *hilo*(thread).

El método **ImemInputPin::ReceiveCanBlock** determina si el pin entrada puede bloquearse en la llamada a **Recibe**. El pin salida puede llamar a este método para determinar una estrategia apropiada de *hilado*(threading). Algunos filtros crean un hilo de ejecución, así que ellos pueden entregar muestras en segundo plano mientras están haciendo otro trabajo. Otros filtros simplemente se bloquean hasta que el filtro downstream esta listo para aceptar otra muestra.

Hay también un método para entregar mas de una muestra a un tiempo, **ImemInputPin::ReceiveMultiple**. Este trabaja como **Receive**, pero con un arreglo de muestras.

3.8.3 Detener, Pausa, y Ejecución

Los filtros tienen tres estados: detenido, pausado y ejecutándose. Para muchos filtros, pausado y ejecutándose son equivalentes. Cuando un filtro esta detenido, no se procesan muestras y se rechaza cualquier muestra de filtros upstream.

Cuando un filtro es pausado o esta ejecutándose, este acepta muestras y las procesa. Si este es un filtro source, genera nuevas muestras y las entrega downstream. Los filtros renderer son una excepción. Cuando un filtro renderer va de detenido a pausado, no se completa la transición sino hasta que reciba muestras media. En este punto, el filtro mantiene la muestra y se bloquea en la llamada a **Receive**. Los renderers de video dibujan la muestra como un frame fijo. Los renderers de audio no presentan la muestra hasta que ellos estén ejecutándose. En cualquier caso, el renderer no acepta mas muestras hasta que pase de pausado a ejecutándose.

El Manejador del gráfico de filtros controla el estado del gráfico de filtros como un conjunto. Las transiciones de estados validas son entre detenido y pausado, y entre pausado y ejecutándose. Transiciones entre detenido y ejecutándose deben ser a través de pausado (Si se llama a **ImediaControl::Run** en un grafico detenido, o **ImediaControl::Stop** en un filtro ejecutándose, el Manejador del gráfico de filtros pausa primero el grafico).

Cuando el grafico del filtro va de detenido a pausado, ocurre la siguiente secuencia:

El Manejador del gráfico de filtros llama a **ImediaFilter::Pause** en cada filtro, iniciando desde el filtro renderer y continuando upstream.

Como cada filtro conmuta a pausado, un filtro se encuentra listo para aceptar muestras desde su upstream más cercano. Este puede entregar muestras downstream con seguridad, debido a que los filtros downstream también están pausados.

El filtro source es el último en ser pausado. En este punto, el filtro inicia entregando muestras. Como las muestras se mueven downstream, los filtros intermedios las procesan.

Cuando las primeras muestras alcanzan al renderer, el renderer completa su estado de transición y se bloquea.

Estos eventos pueden tomar una cantidad arbitraria de tiempo para completarse (usualmente no mucho, pero el retardo podría ser significativo, especialmente si la fuente requiere descompresión). Al tiempo que el gráfico ha terminado de pausarse, estará apto para iniciar la presentación de datos inmediatamente. Una aplicación puede hacer esto adelantándose en tiempo, y entonces rápidamente conmutar el grafico a ejecutándose, en respuesta a un comando del usuario.

3.8.4 Notificación de eventos en DirectShow

Un filtro notifica al Manejador del gráfico de filtros acerca de un evento mediante el envío de una notificación de evento. El evento puede ser algunas veces previsto, tal como el fin de un flujo, o este podría representar un error, tal como una falla al presentar un flujo. El Manejador del gráfico de filtros maneja algunos eventos de los filtros y deja otros eventos para que los maneje la aplicación. Si el Manejador del gráfico de filtros no maneja el evento del filtro, este coloca la notificación del evento en una cola. El Gráfico de filtros puede también encolar sus propias notificaciones de eventos para la aplicación.

Una aplicación obtiene eventos de la cola y responde a ellos basado en el tipo de evento. La notificación de eventos en DirectShow es similar al esquema de encolado de mensajes de Windows. Una aplicación puede también cancelar el comportamiento por default del Manejador del gráfico de filtros para un tipo de evento dado. El Manejador del gráfico de filtros entonces mete estos eventos directamente en la cola para que la aplicación las maneje.

Este mecanismo permite:

- Al Manejador del gráfico de filtros comunicarse con la aplicación.
- A los filtros comunicarse con la aplicación y con el Manejador del Gráfico de filtros.
- A la aplicación determinar su grado de involucramiento en el manejo de eventos.

3.8.4.1 Capturando Eventos

El Manejador del gráfico de filtros expone tres interfaces que soportan la notificación de eventos.

IMediaEventSink contiene los métodos para que los filtros envíen eventos.
IMediaEvent contiene métodos para que las aplicaciones capturen eventos.
IMediaEventEx hereda desde y extiende la interfaz **IMediaEvent**.

Los filtros envían notificaciones de eventos mediante la llamada al método **IMediaEventSink::Notify** en el Manejador del Gráfico de filtros, y dos parámetros DWORD que dan información adicional. Dependiendo del código del evento, los parámetros pueden contener apuntadores, regresar códigos, tiempos de referencia, u otra información.

Para obtener un evento de una cola la aplicación llama al método **IMediaEvent::GetEvent** en el Manejador del Gráfico de filtros. Este método se bloquea hasta que regrese un evento o hasta que pase un lapso de tiempo especificado. Asumiendo que hay un evento encolado, el método regresa con el código del evento y los dos parámetros del evento. Después de llamar a **GetEvent**, una aplicación debe siempre llamar al método **IMediaEvent::FreeEventParams** para actualizar cualquier recurso asociado

con los parámetros del evento. Por ejemplo, un parámetro podría ser un valor BSTR que fuese localizado por el Gráfico de filtros.

El siguiente código provee una guía de cómo obtener eventos de una cola.

```
long evCode, param1, param2;
HRESULT hr;
while (hr = pEvent->GetEvent(&evCode, &param1, &param2, 0), SUCCEEDED(hr))
{
    switch(evCode)
    {
        // Llama a funciones definidas en la aplicación para cada
        // tipo de evento que se quiera manejar.
    }

    hr = pEvent->FreeEventParams(evCode, param1, param2);
}
```

Para sobrescribir el manejador por default del Manejador del Gráfico de filtros, se debe llamar al método **IMediaEvent::CancelDefaultHandling** con el código del evento como un parámetro. Se puede volver a crear una instancia del manejador por default llamando al método **IMediaEvent::RestoreDefaultHandling**.

3.8.4.2 Saber cuando un evento ocurre

Adicionalmente a lo descrito, una aplicación necesita una manera de saber cuando los eventos están esperando en la cola. El Manejador del gráfico de filtros provee dos formas de hacer esto.

- Usando notificaciones Windows, el Manejador del gráfico de filtros envía un mensaje tipo Windows a una ventana de aplicación en cualquier momento que exista un nuevo evento.
- Usando manejadores de eventos, la aplicación obtiene un manejador a un evento Windows manual-reset[6] . El Manejador del gráfico de filtros señala el evento manual-reset cuando haya notificaciones de eventos en la cola y reinicializa esta cuando la cola está vacía.
  Veamos cada una de las dos opciones.

3.8.4.3 Notificación Windows

Para utilizar una notificación windows, se debe llamar al método **IMediaEventEx::SetNotifyWindow** y especificar un mensaje privado. Las aplicaciones pueden usar números de mensajes en el rango de WM_APP hasta 0xBFFF como mensajes privados. En cualquier momento el Manejador del gráfico de filtros coloca una nueva notificación de evento en la cola, este envía este mensaje a la ventana destino. La aplicación responde al mensaje desde el lazo(loop) de mensajes de la ventana.

El siguiente código ejemplifica como ajustar la notificación window.

[6] El evento manual-reset es un tipo de evento creado por la función Windows **CreateEvent**; esta no tiene nada que ver con las notificaciones de eventos definidas por DirectShow.

```
#define WM_GRAPHNOTIFY WM_APP + 1   // Mensaje privado.
pEvent->SetNotifyWindow((OAHWND)g_hwnd, WM_GRAPHNOTIFY, 0);
```

Este mensaje es un mensaje ordinario Windows, y es enviado separadamente de la cola de notificación de eventos DirectShow. La ventaja de esta aproximación es que muchas aplicaciones ya tienen implementado un lazo de mensajes. Así mismo puede incorporar un manejador de eventos DirectShow sin mucho trabajo adicional.

El siguiente código muestra como responder al mensaje de notificación.

```
LRESULT CALLBACK WindowProc( HWND hwnd, UINT msg, UINT wParam, LONG lParam)
{
    switch (msg)
    {
        case WM_GRAPHNOTIFY:
            HandleEvent();  // Función definida en la aplicación.
            break;

        // Maneja otros mensajes de Windows aquí también.
    }
    return (DefWindowProc(hwnd, msg, wParam, lParam));
}
```

Debido a que la notificación de eventos, y el lazo de mensajes son ambos asíncronos, la cola puede contener mas de un evento para el tiempo que su aplicación responda al mensaje. También, los eventos pueden algunas veces ser quitados de la cola si ellos fuesen inválidos. Así mismo, en el código manejador del evento, hay que llamar a **GetEvent** hasta que este regrese un código de falla, indicando que la cola está vacía.

3.8.4.4 Manejadores de eventos

El gráfico de filtros guarda un evento manual-reset que refleja el estado de la cola de eventos. Si la cola contiene notificaciones de eventos pendientes, el gráfico de filtros señala los eventos manual-reset. Si la cola está vacía, una llamada al método **IMediaEvent::GetEvent** reinicializa el evento. Una aplicación puede usar este evento para determinar el estado de la cola.

El método **IMediaEvent::GetEventHandle** obtiene un manejador a el evento manual-reset. Espera para que el evento sea señalado mediante un llamado a una función tal como **WaitForMultipleObjects**. Al mismo tiempo que el evento es señalado, la llamada al evento **IMediaEvent::GetEvent** obtiene la notificación del evento.

El siguiente código de ejemplo ilustra este método. Este obtiene el manejador de evento, entonces espera en intervalos de 100 milisegundos para ser señalado. Si el evento es señalado, este llama a **GetEvent** e imprime el código de evento y los parámetros del evento a la ventana. El lazo termina cuando ocurre el evento EC_COMPLETE.

```
HANDLE  hEvent;
long    evCode, param1, param2;
BOOLEAN bDone = FALSE;
HRESULT hr = S_OK;

hr = pEvent->GetEventHandle((OAEVENT*)&hEv);

while(!bDone)
{
```

```
    if (WAIT_OBJECT_0 == WaitForSingleObject(hEvent, 100))
    {
          while (hr = pEvent->GetEvent(&evCode, &param1, &param2, 0), SUCCEEDED(hr))
        {
            printf("Event code: %#04x\n    Params: %d, %d\n", evCode, param1, param2);
            hr = pEvent->FreeEventParams(evCode, param1, param2);
            bDone = (EC_COMPLETE == evCode);
        }
    }
}
```

Debido a que el gráfico de filtros automáticamente ajusta o reinicializa el evento cuando sea apropiado, la aplicación no debería hacerlo. También, cuando se actualiza el gráfico de filtros, el gráfico de filtros cierra el manejador de eventos, así no hace uso del manejador de eventos después de este punto.

3.9 Hardware en el gráfico de filtros

3.9.1 Filtros Envoltura (Wrapper)

Todos los filtros DirectShow son componentes de software en el modo usuario. Para que un dispositivo hardware en el modo kernel como una tarjeta de captura de video/sonido se añada al gráfico de filtros de DirectShow, el dispositivo debe ser representado como un filtro en el modo usuario dentro del gráfico de filtros. Esta función se lleva a cabo para varios tipos de dispositivos por filtros especializados llamados *filtros envoltura*(wrapper filters), provistos con DirectShow.

Los filtros envoltura trabajan soportando las interfaces COM que representan las capacidades esperadas del dispositivo. La aplicación usa estas interfaces para pasar información hacia y desde el filtro, el filtro traduce las llamadas a la interfase en información que el controlador del dispositivo pueda comprender, y entonces el filtro pasa esta información hacia y desde el controlador en el modo kernel.

Para los desarrolladores de aplicaciones, el principio de envolver dispositivos de hardware como filtros en modo usuario significa que las aplicaciones controlan dispositivos de la misma forma que ellos controlan cualquier otro filtro DirectShow. No se requiere una programación especial en la parte de la aplicación; todos los detalles involucrados en la comunicación con el dispositivo en el modo kernel son encapsulados dentro del filtro mismo.

3.9.2 Video para dispositivos Windows

Para soportar las ultimas tarjetas de captura Video for Windows (VfW), DirectShow provee el *filtro de captura VFW*. Cuando una tarjeta VfW esta presente en el sistema, esta puede ser descubierta usando el System Device Enumerator[7]. El Enumerator se utiliza también para añadir el filtro de captura VfW al grafico y asociar este con la tarjeta que fue

[7] Aplicación contenida en DirectX 8.0

encontrada. El resto del gráfico de filtros puede entonces ser construido de la manera usual..

3.9.3 Captura de Audio y dispositivos Mezcladores

Las tarjetas de sonido modernas tienen conectores de entrada para micrófonos y otro tipo de dispositivos. Estas tarjetas también tienen capacidades de mezclado para controlar el volumen, graves y agudos de cada entrada individualmente. En DirectShow, las entradas de las tarjetas de sonido y mezcladoras son envueltas en el filtro de captura de audio (Audio Capture filter). Cada tarjeta de sonido en el sistema local puede ser descubierta con el system device enumerator. Para visualizar las tarjetas de sonido en su sistema basta abrir el programa GraphEdit que se instala con el SDK y seleccionar de la categoría Audio Capture Sources. Cada filtro representado aquí es una instancia separada del Audio Capture filter.

4 Diseño

4.1 Descripción general del sistema

El proyecto se va a dividir en dos subsistemas, uno es el que concierne a la construcción del videowall, el cual se llevará a cabo via hardware y utilizando las capacidades del sistema operativo. El otro subsistema consiste en la elaboración del software que permitirá hacer uso de todas las pantallas y realizar el procesamiento de video en tiempo real.

En la figura 4.1 se muestra un diagrama de flujo de datos. El filtro de gráficos DirectShow se muestra en gris. Los datos de video se desplazan downstream desde la fuente de video, ya sea la tarjeta de TV o un archivo de video ubicado en el disco duro. La aplicación controla el filtro de procesamiento múltiple de video, y entonces es posible que el usuario a través de la aplicación y mediante una GUI(interfaz Gráfica de usuario) seleccione el procesamiento de video de su interés. Finalmente un filtro renderer se ocupa de enviar el video de salida hacia el sistema operativo, quien se encarga de distribuir dicha imagen hacia las cuatro tarjetas de video conectadas en la PC. En el gráfico no se muestran los datos de control de calidad(quality control) utilizados por Directshow y que se desplazan upstream.

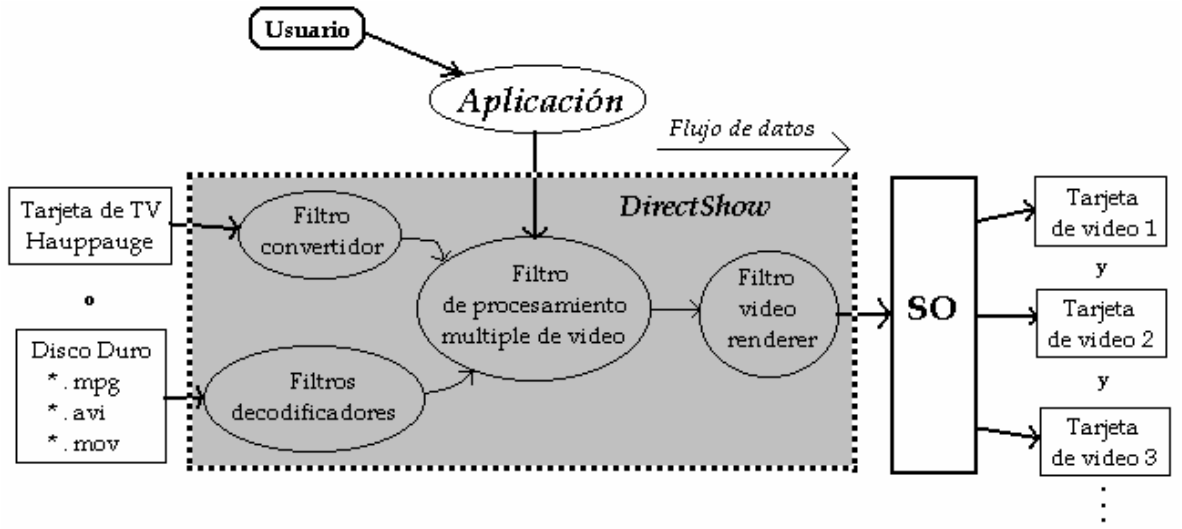


Figura 4.1 Diagrama general del proyecto

El proyecto esta elaborado con Microsoft Visual C++ 6.0, y haciendo uso de las librerías de DirectShow incluidas en DirectX 8.1.

4.2 Entrada de video

Se seleccionó la tarjeta de video Hauppauge WinTV, debido al soporte de controladores que brindan a través de su pagina web. Adicionalmente en la web se

encontraron multiples soportes de información para dicha tarjeta y finalmente Hauppauge representa cerca del 80% de ventas de tarjetas de TV en Europa , garantizando así su actualización constante.

El acceso a la tarjeta Hauppauge WinTV se logra a través de un filtro wrapper provisto por DirectShow. Se tiene después que conectar un filtro que convierta el formato de video provisto por la tarjeta WinTV a un formato que pueda ser procesado, en nuestro caso a RGB24. Para el caso de un archivo en Disco duro se han contemplado solo tres formatos de video de entrada *.mov, *.mpg y *.avi, por ser estos de mayor uso.

4.3 Filtro multiprocesamiento

Se tomó la decisión de elaborar un filtro en lugar de varios filtros, debido a que la conexión dinámica del grafico de filtros puede llevar a discontinuidades en la presentación del video. La intención es poder realizar el intercambio de filtros sin afectar la velocidad de reproducción del video. Adicionalmente por facilidad de procesamiento se pensó utilizar el formato de video RGB 24 donde cada byte representa una componente de color. El filtro de procesamiento múltiple de video incluirá los distintos procesamientos y permitirá su selección a través de la aplicación mediante una interfaz. El filtro será del tipo in-place, debido a que no se tiene contemplado hacer algún tipo de compresión sobre el video, de tal modo que los efectos de video solo manipularán las muestras sin afectar la longitud de las mismas. La figura 4.2 muestra un diagrama de la construcción del filtro.

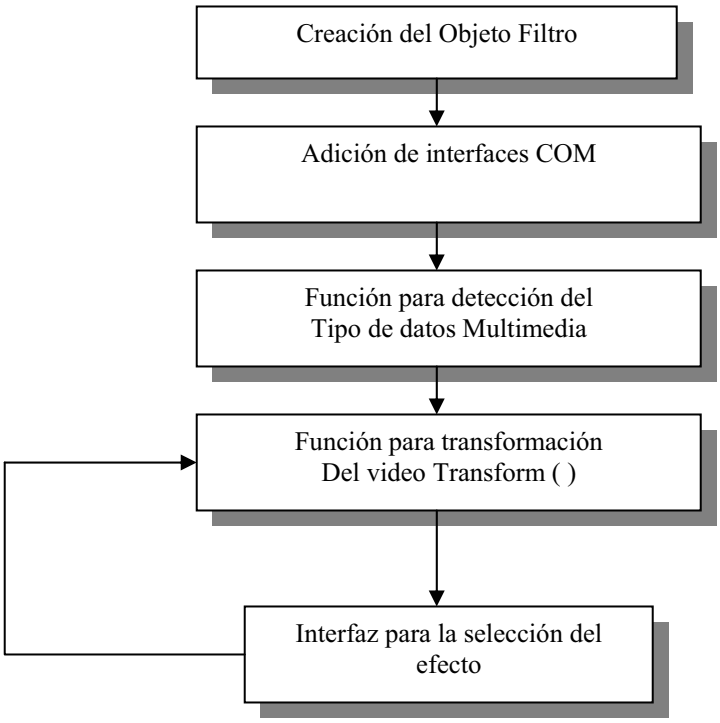


Figura 4.2 Diagrama del filtro

El filtro debe partir de la clase base CTransformFilter, la cual no realiza operaciones de copia sobre las muestras, esto permitirá ahorrar recursos de CPU.

Para las pruebas en el filtro se utilizará la herramienta provista por DirectX **GraphEdit**. Después de elaborar el filtro se iniciará el desarrollo de la aplicación. Cabe aclarar que por lo investigado, DirectShow no dispone de un control directo sobre las muestras multimedia. En el proyecto se irán experimentando las capacidades y limitaciones al respecto.

Sobre el diseño de los algoritmos a utilizar en cada procesamiento, se requiere conocer las características de DirectShow, para poder implementar las que tengan el mejor rendimiento.

4.4 Entrega de muestras al sistema operativo

La entrega de las muestras multimedia hacia el sistema operativo requiere elaborar un filtro renderer que se ocupe de mantener un buffer para almacenar las muestras y entregarlas hacia el sistema operativo. Teniendo en consideración que por limitantes propias de DirectX, la presentación del video no será a pantalla completa. Por tal motivo se implementara un manejo de ventanas típico de cuyo despliegue se encarga el sistema operativo.

4.5 Aplicación

La aplicación será responsable de la construcción del gráfico de filtros, esta se estructura en función de la fuente de video elegida por el usuario (tarjeta de TV o un archivo multimedia). Así la construcción del gráfico de filtros deberá hacerse automáticamente dependiendo de si el archivo de entrada se encuentra en formato AVI o MPEG. Posteriormente se requiere desconectar el grafico de filtros e insertar el filtro multiprocesamiento, lo cual debe considerar los tipos de datos que se manejan en dicho punto de flujo. Será importante realizar una función de chequeo sobre el tipo de datos multimedia para asegurar que se puede proporcionar a nuestro filtro un formato RGB24 sin problema alguno. Finalmente se reconstruye el gráfico de filtros incluyéndose el filtro renderer que proporcione la imagen de video final hacia el sistema operativo para su presentación en múltiples pantallas.

La interfaz grafica al usuario consistirá en una ventana con menú para seleccionar la fuente de video, del mismo modo se seleccionará el tamaño de la ventana, ya sea que ocupe un monitor o todos los monitores.

Se debe también incorporar una ventana que permita la selección de el efecto de video deseado. Esta ventana debe aparecer solo cuando sea solicitada por el usuario. La figura 4.3 muestra un diagrama de la construcción de la aplicaciòn.

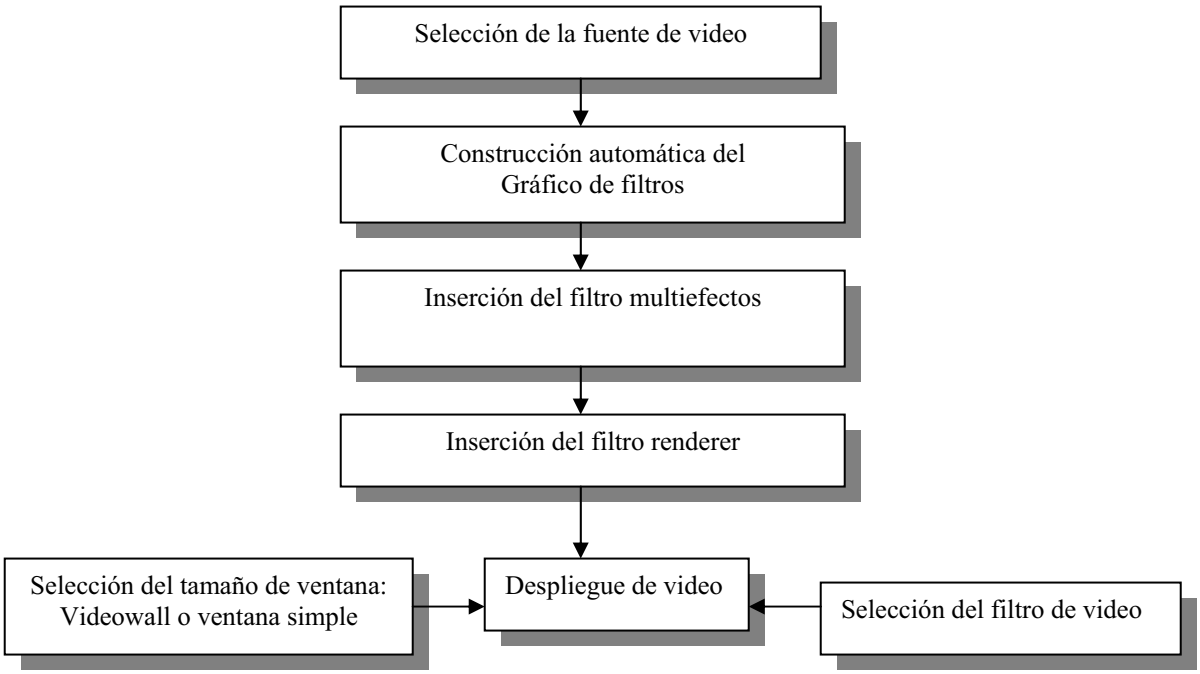


Figura 4.3 Diagrama de la aplicación

5. Implementación

5.1 Funcionamiento de la tarjeta WinTV

Para visualizar el video, la tarjeta utiliza una técnica denominada PCI Push. Con esta técnica, la tarjeta WinTV digitaliza el video y a continuación se lleva a través del bus PCI hacia la memoria de la tarjeta VGA sin necesidad de que el procesador realice trabajo alguno y por ende no se reduzca la velocidad de la PC. La conversión a video digital se logra con el chip Bt848, donde se realiza un muestreo 4:2:2 YUV con una resolución equivalente a 24 bits RGB por píxel de video.

5.1.1 Modo de superposición de video

La imagen de video se visualiza en la pantalla ya sea utilizando el modo de *superposición de video* o el modo *Superficie principal*. El modo que se utiliza depende del hardware y software de la PC.

Si la tarjeta VGA tiene compatibilidad con DirectDraw[2] y dispone de suficiente memoria de pantalla para mantener la imagen de video digitalizada, y adicionalmente tiene un puerto de video diseñado para aceptar video digital, entonces realizara un almacenamiento temporal hacia una parte de la memoria VGA fuera de la pantalla denominada **Superficie Secundaria**. Este método se denomina Superposición de video. A continuación el adaptador VGA convertirá la imagen de video de YUV 4:2:2 a video RGB y superpondrá continuamente la imagen de video sobre la pantalla VGA.

5.1.2 Modo de superficie principal

Si la tarjeta VGA tiene compatibilidad con DirectDraw pero no tiene un puerto de video o no dispone de suficiente memoria para mantener la imagen de video fuera de la pantalla, entonces la tarjeta de video convierte los pixels de video 4:2:2 YUV en un formato RGB que es compatible con el modo de funcionamiento de la tarjeta VGA (8 bits por píxel, 16 bits por píxel o 24 bits por píxel) y continuación lleva los pixels directamente hacia la memoria de pantalla o superficie principal de la VGA.

5.2 Multiplexado de video

La multiplexación de video se ha logrado haciendo uso de las capacidades del sistema operativo. Dicha característica se denomina soporte de múltiple monitor, y es ofrecida en la actualidad por Windows 98 y superiores además de LINUX.

[2] Componente de DirectX

La decisión de trabajar con Windows 98 es la facilidad con la que soporta el nuevo hardware económico y el bajo consumo de recursos, además de la viabilidad para trabajar con el software DirectX. A la fecha LINUX contaba con muy poca información sobre librerías para manipulación de video y la escasa se encontraba en etapa experimental.

La característica de múltiple monitor permite al sistema operativo manejar hasta 9 monitores conectados a la misma tarjeta madre. Las tarjetas deberán ser PCI o AGP, aunque en general solo se dispone de 6 slots PCI en una tarjeta de uso común puede conseguirse dispositivos para ampliar el numero de slots. Es importante recalcar que la adición de cada tarjeta provocará una sobrecarga en el procesador, lo que disminuirá el rendimiento del sistema en su conjunto.

5.2.1 Tarjeta primaria y tarjetas secundarias

Una de todas las tarjetas de video será la tarjeta primaria. Esta es la tarjeta que por omisión albergará las cajas de dialogo, dado que muchas aplicaciones iniciarán ahí, y básicamente todos los programas que hagan uso de pantalla completa se ejecutarán aquí.

Windows 98 no determina cual es la tarjeta primaria. Esta es determinada por el BIOS de la computadora. Esto significa que la primer tarjeta de video que la computadora detecte durante el proceso de arranque será la primaria. Se pudo observar que la computadora siempre detecta la misma tarjeta de video como la primaria, y lo que es mas, siempre arranca con la tarjeta que se ubica en el mismo puerto PCI. Se tiene que solo un determinado numero de tarjetas de video soportan múltiple monitor y se debe entonces revisar la lista de Microsoft para determinar la compatibilidad.

Por lo general la tarjeta madre dispone de un puerto AGP y de varios PCI, en estos casos la PC siempre determina la tarjeta PCI como la primaria, debido a que el BIOS inicializa el bus PCI antes de inicializar el bus AGP.

5.2.2 Configuración de multimonitor

Suponiendo que se tiene una tarjeta de video instalada, se añade la siguiente tarjeta con el equipo apagado y al reiniciar, el sistema deberá reconocer la nueva tarjeta. Si todo va bien solo se debe ir hacia Display Properties: Se da clic en Start, Settings, Control Panel, Display, y finalmente en, Settings.

Para habilitar el nuevo dispositivo, simplemente se da clic sobre el nuevo monitor, y entonces se selecciona "Extend my Windows desktop onto this monitor."

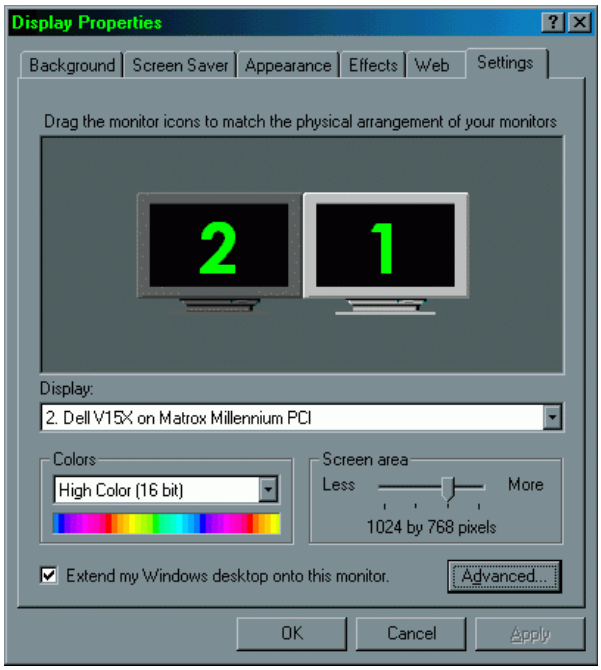


Figura 5.1 Extensión del escritorio

En el caso de que se tenga problemas en saber cual monitor es cual, sobre todo cuando se tienen 4 monitores o más, solo se da click con el botón derecho sobre el monitor y se selecciona Identify. Entonces aparecerá el numero correspondiente al monitor actual, el monitor principal tiene el numero 1 por default.

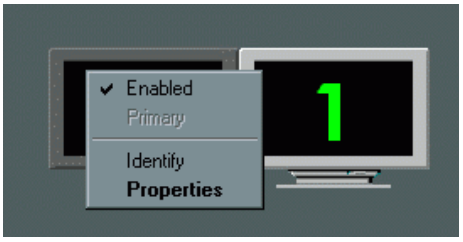


Figura 5.2 Activación del monitor secundario

No es necesario usar la misma resolución en cada monitor. Abajo se muestra un ejemplo con el monitor primario a 1024 x 768 y el secundario a 800 x 600.

Figura 5.3 Resoluciones distintas

Los monitores pueden ser colocados virtualmente uno al lado de otro, uno encima de otro, con las esquinas conectadas o un poco separados. Igualmente se puede configurar la resolución y numero de colores para cada uno de los monitores.

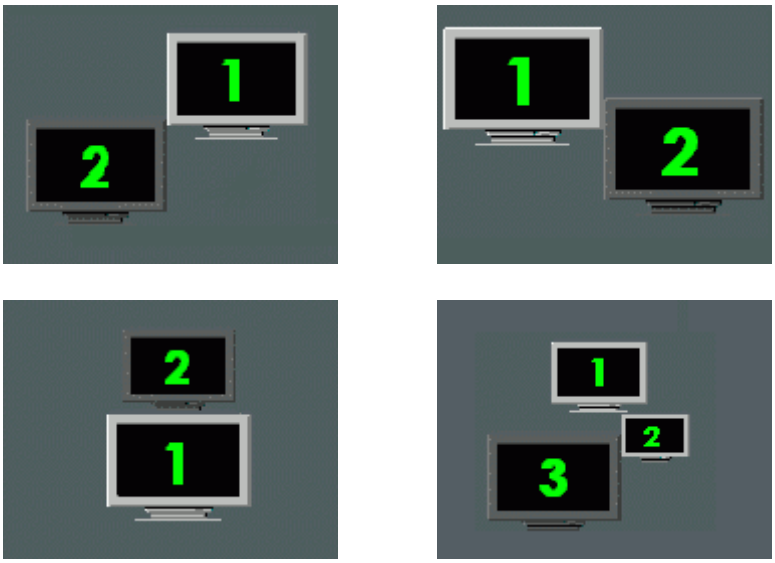


Figura 5.4 Configuraciones distintas

5.2.3 Pantalla virtual

Windows utiliza un sistema de coordenadas para referenciar cualquier elemento sobre la pantalla. La esquina superior izquierda en una pantalla con resolución 800x600 es la coordenada 0,0. La coordenada de la esquina inferior derecha es 800, 600.

En el caso de múltiple monitor, la esquina superior izquierda del monitor primario tiene la coordenada 0,0. Si tenemos un sistema con 2 monitores a 1024x768, con el secundario colocado al lado derecho del primario. Entonces la esquina inferior derecha de la pantalla virtual será 2048x768.

Para evitar inconsistencias con coordenadas de números negativos (algunas aplicaciones no pueden ser arrastradas hacia los monitores con coordenadas negativas) se ha realizado la siguiente configuración de monitores en el presente proyecto.

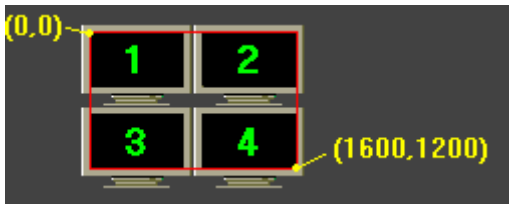


Figura 5.5 Pantalla virtual del videowall

Se han instalado las siguientes tarjetas de video VGA sin conflicto con la configuración multimonitor:

1.- ATI 3D Rage Pro

2.- 3Dblaster Savage 4

3.- ATI 3D Rage Pro

4.- Cirrus Logic 5446PCI

En alguna bibliografía e información de internet se especificaba no colocar tarjetas de video idénticas, sin embargo se utilizaron 2 tarjetas ATI sin problemas. En la figura 5.6 se observan las tarjetas ya instaladas en la tarjeta principlal de la PC.

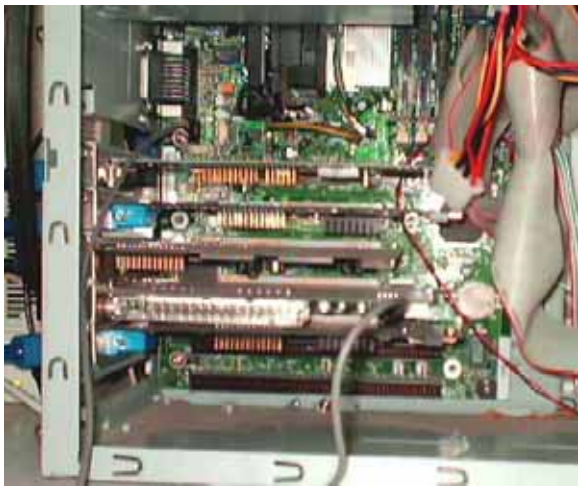


Figura 5.6 Tarjetas VGA y WinTV instaladas

5.3 Instalación del SDK DirectX

El software SDK(System Development Kit) de DirectX contiene las herramientas necesarias para construir aplicaciones graficas y de multimedia. Esta incluye el runtime, headers y libs, ejemplos de ejecutables, ejemplos en código fuente, documentación, utilidades, y soporte para desarrollo en C++ y Visual Basic.

El usuario que desea incorporar el soporte de DirectX solo requiere el runtime. El SDK puede obtenerse desde el sitio de Microsoft:
http://www.microsoft.com/windows/directx/default.asp
El tamaño del archivo es de 175MB, y solo requiere ejecutarse para comenzar la instalación de todo el sistema de desarrollo.

Para verificar la correcta instalación del SDK, se comienza por compilar alguno de los ejemplos, en este caso compilaremos el ejemplo PlayWnd (reproductor multimedia sencillo). En el presente trabajo se utilizó el compilador Microsoft Visual C++ 6.0 pero los procedimientos son validos también para la versión 5.0.
El código fuente del ejemplo se ubica en la siguiente ruta:

(SDK root)\Samples\Multimedia\DirectShow\Players\PlayWnd

Se dispone de dos modos de compilación, el modo **Debug** y el modo **Release**. En el modo Debug al compilarse se incluye código adicional que permite obtener texto sobre los errores en el proceso de compilación. En el modo Release esto no sucede pero el código ejecutable es significativamente menor.
La selección del modo se realiza en la barra de menús de Visual C++. Se da clic en Build después en Set Active Configuration y finalmente se selecciona el modo deseado.

Las aplicaciones de DirectShow requieren para su compilación de las librerías strmbasd.lib en el caso del modo Debug y de STRMBASE.lib en el caso de Release. Estas librerías deben crearse, por lo tanto se debe abrir el proyecto
(SDK root)\Samples\Multimedia\DirectShow\BaseClasses\baseclasses.dsw
posteriormente debe compilarse en los modos Debug y Release para que se creen las librerías correspondientes. La intención de esto es que el desarrollador pueda modificar dichas librerías.

A continuación se deben añadir las rutas de las librerías y headers de DirectX, ambas deben ubicarse en el primer lugar de búsqueda para que el compilador no escoja una librería antigua. Esto se realiza mediante la inclusión de las rutas
(SDK root)\include
(SDK root)\lib

En el menú Tools se da click en Options y se selecciona la pestaña Directories. Se ajusta la caja de dialogo para que se visualice como sigue en el caso de include:

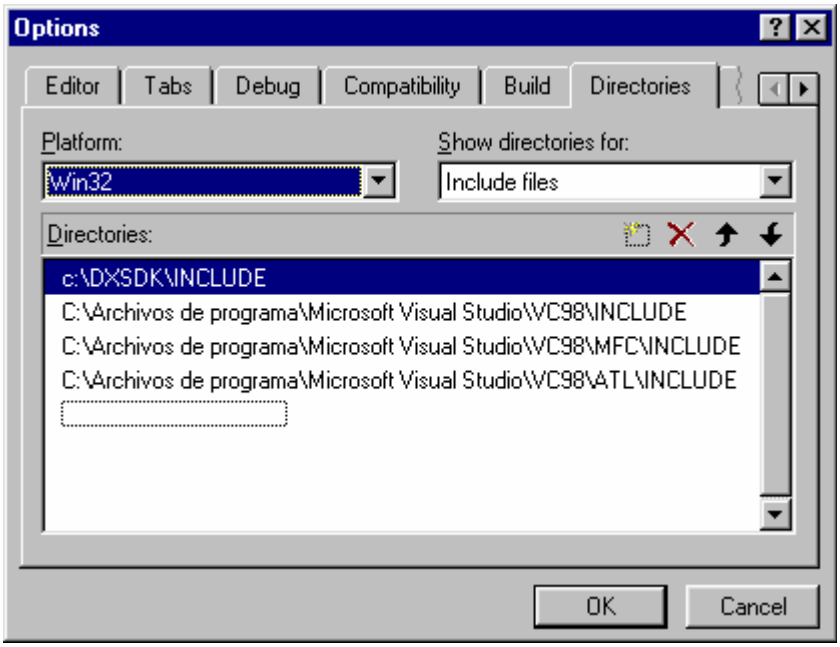


Figura 5.7 Configuración de directorios en Visual C++

Hecho lo anterior entonces podemos abrir el proyecto
(SDK root)\Samples\Multimedia\DirectShow\Players\PlayWnd\PlayWnd.dsw
y su compilación no debe tener problema adicional. Antes de la introducción con DirectShow es necesario revisar el modelo de objetos componentes COM con el que se basan todas las aplicaciones DirectX.

Iniciamos la construcción del filtro de procesamiento múltiple, pero como se comentaba en el capitulo anterior utilizaremos la herramienta GraphEdit para realizar las pruebas con el.

5.4 Simulación con GraphEdit

El SDK de Microsoft® DirectX® provee una utilidad de depuración llamada GraphEdit, la cual se puede usar para crear y probar un grafico de filtros.

GraphEdit es una herramienta visual para construir un grafico de filtros. Utilizando GraphEdit, se puede experimentar con un grafico de filtros antes de escribir el código de la aplicación. Se puede también cargar un grafico de filtros que la aplicación creó, para verificar que la aplicación está construyendo el grafico correcto. Si se desarrolla un filtro personalizado, GraphEdit permite hacer pruebas con él intentando correr un gráfico.

La figura 5.8 muestra como GraphEdit representa un grafico de filtros simple.

Figura 5.8 Grafico de filtros sencillo

Cada filtro es representado como un rectángulo. Los pequeños cuadros cerca de las esquinas de los filtros representan los pins. Los pins entrada están en el lado izquierdo del filtro, y los salida pins en el lado derecho. Las flechas representan la conexión entre pins.

5.5 Uso de GraphEdit

Cuando se instala el DirectX SDK 8.0, GraphEdit aparece en el menú **Start** bajo **Microsoft DirectX 8 SDK**, en el submenú **DX Utilities**. El archivo ejecutable es GraphEdit.exe. Por default, este es instalado en el fólder Mssdk\Bin\DXUtils.

A continuación hay una breve descripción de algunas cosas que se pueden hacer usando GraphEdit.

5.5.1 Construcción de un grafico de ejecución de archivos

GraphEdit puede construir un grafico de filtros para ejecutar un archivo. Esta característica es equivalente a hacer una llamada al método **IGraphBuilder::RenderFile** en una aplicación. Desde el menú **File**, hacer clic en **Render Media File**. GraphEdit despliega una caja de dialogo **Open File**. Se selecciona un archivo multimedia y se da clic en **Open**. GraphEdit construye un grafico de filtros para ejecutar el archivo que se ha seleccionado.

5.5.2 Construcción de un Grafico de filtros personalizado

GraphEdit puede construir un grafico de filtros personalizado, usando cualquiera de los filtros registrados en su sistema. Desde el menú **Graph**, hay que dar clic en **Insert Filters**. Una caja de dialogo aparece con una lista de los filtros en el sistema, organizados por categoría de filtros. GraphEdit construye esta lista a partir de información en el registro. La siguiente ilustración muestra la caja de dialogo.

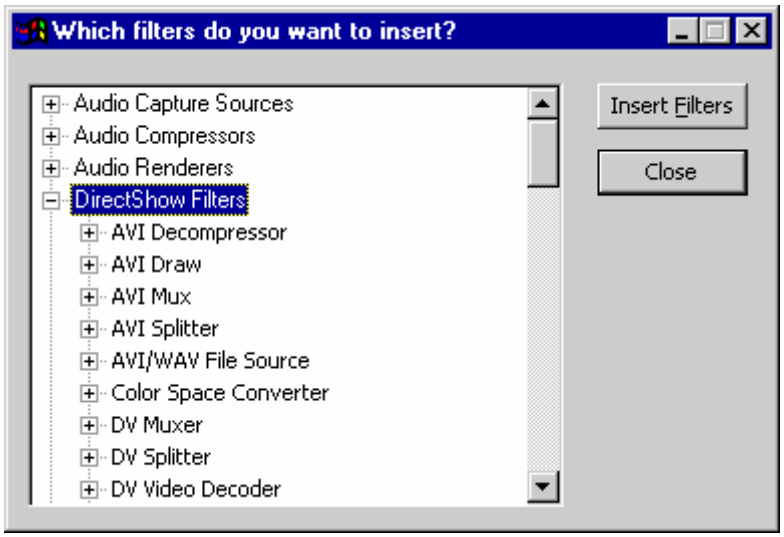


Figura 5.9 Selección del filtro a insertar

Para añadir un filtro al grafico, se selecciona el nombre del filtro y se da clic en el botón **Insert Filters**, o doble clic en el nombre del filtro. Después de que se han añadido los filtros, se pueden conectar dos filtros arrastrando el mouse desde el salida pin del filtro hacia el pin entrada de otro filtro. Si los pins aceptan la conexión, GraphEdit dibuja una flecha conectándolos.

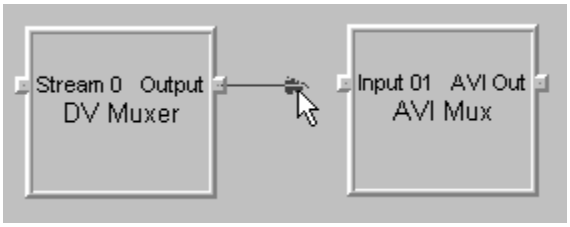


Figura 5.10 Conexión de dos filtros

5.5.3 Ejecución del grafico

En el momento que se ha construido un grafico de filtros en Graph Edit, se puede correr el grafico para ver si este trabaja como se esperaba. El menú **Graph** contiene los menús de comando **Play**, **Pause**, y **Stop**. Estos comandos invocan a los métodos **IMediaControl Run**, **Pause**, y **Stop**, respectivamente. La barra de herramientas de GraphEdit tiene botones para estos comandos como se muestra:

Figura 5.11 Controles de ejecución

Nota El comando **Stop** primero pausa el grafico y se recorre al tiempo cero. Para archivos de video se restaura la ventana de video con el primer cuadro. Entonces GraphEdit llama a **IMediaControl::Stop**.

Si el grafico es rebobinable, se puede bobinar arrastrando la barra deslizable que aparece debajo de la caja de herramientas. Arrastrar la barra deslizable invoca al método **IMediaSeeking::SetPositions**.

5.5.4 Ver paginas propietarias

Algunos filtros soportan paginas propietarias personalizadas, el cual provee una interfaz de usuario para ajustar las propiedades en el filtro. Para ver las paginas propietarias del filtro, hacer clic con el botón derecho en el filtro y seleccionar **Properties** de la ventana que aparezca. GraphEdit despliega una pagina propietaria que contiene las hojas definidas para el filtro (si hay alguna). Adicionalmente, GraphEdit incluye hojas para cada pin en el filtro. Las hojas del pin son definidas por GraphEdit, no por el filtro. Si el pin esta conectado, la hoja del pin despliega el tipo de media para la conexión, de lo contrario, esta lista los tipos de media preferidos del pin.

En la figura 5.12 se muestra un ejemplo típico de pagina propietaria, es la mezcladora de sonidos disponible en todo sistema Windows.

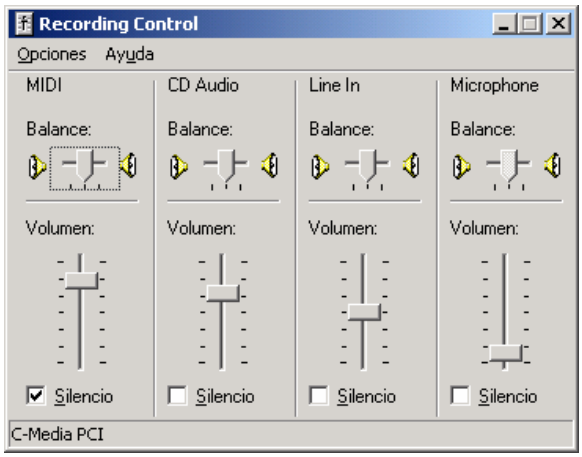


Figura 5.12 Pagina propietaria del dispositivo de grabación en Windows

5.6 Escritura del filtro

Como se mencionó arriba la aplicación será encargada de construir el gráfico de filtros que incorporará nuestro filtro multiprocesamiento. Es necesario entonces estudiar como funcionan internamente los filtros y como interaccionan entre ellos.

Para que un filtro interactúe con otros filtros se requiere que se puedan conectar lo cual requiere de un acuerdo para intercambiar un tipo de datos coherente.

5.6.1 Conexión de filtros

Los filtros se conectan a través de sus pines, haciendo uso de la interfaz **IPin**. Los pins salida se conectan a los pins entrada. Cada conexión de pin tiene un tipo de media, descrito por la estructura **AM_MEDIA_TYPE**.

Una aplicación conecta filtros mediante llamadas a métodos en el Manejador del Gráfico de filtros, nunca por llamadas a métodos en los filtros o en los pins mismos. La aplicación puede directamente especificar el filtro a conectar, llamando al método **IGraphBuilder::Connect**. También puede conectar filtros indirectamente con uno de los métodos de construcción de gráficos de filtros, tal como **IGraphBuilder::RenderFile**.

Para que la conexión sea exitosa, ambos filtros deben estar en el grafico de filtros. La aplicación puede añadir un filtro al grafico llamando al método **IFilterGraph::AddFilter**.

El bosquejo general del proceso de conexión es el siguiente:

1. El Manejador del gráfico de filtros llama a **IPin::Connect** en el pin salida, pasándole un apuntador al pin entrada.

2. Si el pin salida acepta la conexión, este llama a **IPin::ReceiveConnection** en el pin entrada.
3. Si el pin entrada también acepta la conexión, la conexión intenta llevarse a cabo y los pins serán conectados.

Algunos pins pueden ser desconectados y conectados mientras el filtro esta activo. Este tipo de reconexión se denomina reconexión dinámica. Sin embargo, la mayoría de los filtros no soportan la reconexión dinámica. De hecho no fue posible realizar una reconexión dinámica con filtros que manipulasen video y audio al mismo tiempo.

Los filtros son usualmente conectados en orden downstream (en otras palabras, los pins entrada de los filtros son conectados antes de sus pins salida). Un filtro siempre debe soportar este orden de conexión..

5.6.2 Negociando los tipos de Media

Cuando el Manejador del gráfico de filtros llama al método **IPin::Connect**, este tiene varias opciones para especificar un tipo de media:

- **Tipo completo:** Si el tipo de media esta completamente especificado, los pins intentan conectarse con este tipo. Si no son del mismo, el intento de conexión falla.
- **Tipo de media parcial:** Un tipo de media es parcial si el tipo principal, subtipo, o tipo de formato es GUID_NULL. El valor GUID_NULL actúa como un "comodín", indicando que cualquier valor es aceptable. El pin negocia un tipo que es consistente con el tipo parcial.
- **Sin tipo de media:** Si el Manejador del Grafico de filtros pasa un apuntador NULL, el pin puede aceptar cualquier tipo de media que sea compatible a ambos pins.

Si los pins se conectan, la conexión siempre tiene un tipo de media completo. El propósito del tipo de media determinado por el Manejador del gráfico de filtros es limitar los posibles tipos de conexión.

Durante el proceso de negociación, el salida pin propone un tipo de media, llamando al método **IPin::ReceiveConnection** del pin entrada. El pin entrada puede aceptar o rechazar el tipo propuesto. Este proceso se repite hasta que el pin entrada acepta el tipo, o el salida pin recorre todos los tipos y la conexión falla.

5.6.3 Negociación de Asignadores

Cuando dos pins se conectan, ellos necesitan un mecanismo para intercambiar datos multimedia. El mecanismo es denominado *transporte*. En general, la arquitectura DirectShow es neutral en relación con los transportes. Dos filtros pueden acordar conectarse utilizando cualquier transporte que ambos soporten.

El transporte más común es el de *memoria local*. Con este transporte, los datos media se mantienen en la memoria principal. Un objeto llamado un *asignador* es

responsable por asignar buffers de memoria. El asignador soporta la interfaz **ImemAsignador**. Los pins entrada soportan transporte de memoria local mediante la exposición de la interfaz **ImemInputPin**. El pin salida consulta por la interfaz durante el proceso de conexión.

Ambos pins comparten el mismo asignador. Cuando ellos se conectan, los pins negocian cual pin proveerá el asignador, como sigue:

1. Opcionalmente, el pin salida llama **IMemInputPin::GetAsignadorRequirements**. Este método obtiene los requerimientos de buffer del pin entrada, tales como la alineación de memoria. En general, el pin salida debe corresponder a la petición del pin entrada, hasta que no exista una buena razón para no hacerlo.
2. Opcionalmente, el pin salida llama a **IMemInputPin::GetAsignador**. Este método requiere un asignador del pin entrada. El pin entrada provee uno, o retorna un código de error.
3. El pin salida selecciona un asignador. Este puede usar uno provisto por el pin entrada, o proveer uno el mismo. El pin salida llama a **IMemInputPin::NotifyAsignador** para informar al pin entrada de la selección.

Si un filtro selecciona un asignador del pin entrada downstream, el filtro debe usar un asignador solo para entregar muestras al pin entrada. Este no debe usar el asignador para entregar muestras a otros pins.

5.6.4 Clase base a utilizar

Dado que en este proyecto el filtro realizará el procesamiento in-place, derivamos nuestro filtro de la clase **CTransformFilter** . Esto implica que no se realizarán copias de muestras. Debido a que las operaciones de copia utilizan una gran cantidad de recursos en la CPU, las tratamos de evitar.

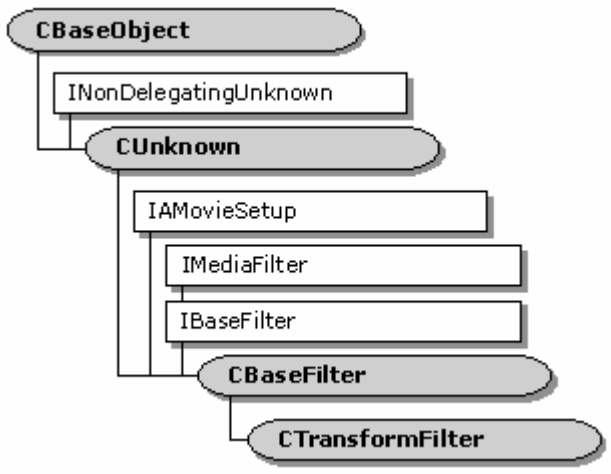


Figura 5.13 Derivación de la clase CTransformFilter

5.6.5 Instanciar el filtro

Todos los filtros deben añadir código para permitir a la clase base instanciar el filtro. Para instanciar un filtro, se deben incluir dos piezas de código en el filtro: una función miembro estática CreateInstance en la clase CtransformFilter y un medio para informar a la class factory de como acceder a esta función.

Típicamente, la función miembro **CreateInstance** llama al constructor para la clase filtro derivada. A continuación se muestra como implementar la función miembro CreateInstance:

```
CUnknown *CGargle::CreateInstance(LPUNKNOWN punk, HRESULT *phr) {

    CGargle *pNewObject = new CGargle(NAME("Gargle Filter"), punk, phr);
    if (pNewObject == NULL) {
        *phr = E_OUTOFMEMORY;
    }

    return pNewObject;
} // CreateInstance
```

Para comunicarse con la class factory, se declara un arreglo global CfactoryTemplate y se provee el nombre del filtro, el identificador de la clase (CLSID) del filtro, y un apuntador a la función miembro estática CocreateInstance que crea nuestro objeto filtro.

```
// Necesario para el mecanismo CreateInstance
CFactoryTemplate g_Templates[2]=
    { { L"Gargle filter"              , &CLSID_Gargle , CGargle::CreateInstance          }
    , { L"Gargle filter Property Page", &CLSID_GargProp, CGargleProperties::CreateInstance}
    };

int g_cTemplates = sizeof(g_Templates)/sizeof(g_Templates[0]);
```

Finalmente, se encadena el filtro a Strmbase.lib y exporta DllGetClassObject y DllCanUnloadNow utilizando un archivo .def .

5.6.6 Adición de interfaces

En nuestro proyecto se añadirá una interfaz para poder crear una pagina propietaria y así el usuario modifique las propiedades del filtro. DirectShow define una clase especial llamada **INonDelegatingUnknown** cuyos métodos hacen las mismas cosas que IUnknow. El método **NonDelegatingQueryInterface** será llamado por cualquier objeto o aplicación que quiera consultar un pin o un filtro por alguna interfaz que este implemente. El siguiente código ejemplo sobrescribe la función miembro que distribuye referencias a las interfaces **ISpecifyPropertyPages** y **IPersistFlujo**:

```
// Muestra las interfaces persistent flujo, property pages y Igargle.
STDMETHODIMP CGargle::NonDelegatingQueryInterface(REFIID riid, void **ppv) {
    if (riid == IID_IGargle) {
        return GetInterface((IGargle *) this, ppv);
    } else if (riid == IID_ISpecifyPropertyPages) {
        return GetInterface((ISpecifyPropertyPages *) this, ppv);
    } else if (riid == IID_IPersistFlujo) {
        AddRef();     // add a reference count to ourselves
```

```
        *ppv = (void *)(IPersistFlujo *)this;
        return NOERROR;

    } else {
        return CTransInPlaceFilter::NonDelegatingQueryInterface(riid, ppv);
    }
} // NonDelegatingQueryInterface
```

5.6.7 Funciones miembro

Se deben escribir algunas funciones miembro necesarias para el proceso de conexión, tales como ajustar el tamaño del asignador o proveer los tipos de media que se utilizarán.

5.6.7.1 Función miembro Transform

La función miembro **Transform** de nuestra clase derivada se llama cada vez que el método **IMemInputPin::Receive** en el pin entrada del filtro es llamado para transferir otra muestra. Aquí se coloca el código o desde aquí se llaman las funciones que realizan las transformaciones requeridas, en nuestro caso los distintos procesamientos sobre el video.

5.6.7.2 Función miembro CheckInputType

Durante la conexión del pin, la función miembro CheckMediaType del pin entrada es llamada para determinar cual de los tipos de media propuestos es aceptable. La función miembro **CTransformInputPin::CheckMediaType** es implementada para llamar a la función miembro CheckInputType de la clase filtro derivada con el tipo de media. Se debe implementar para acomodar los tipos de media que nuestro filtro puede manejar. El siguiente código muestra parte de la función miembro Cgargle::CheckInputType, la cual rechaza cualquier tipo de media que no sea MEDIATYPE_Audio:

```
HRESULT CGargle::CheckInputType(const CMediaType *pmt) {
    ...
    // Reject non-Audio type
    if (pmt->majortype != MEDIATYPE_Audio) {
        return E_INVALIDARG;
    }
```

5.6.7.3 Función miembro CheckTransform

Los filtros transform pueden modificar el tipo de media que va del pin entrada hacia el pin salida. Entonces es necesario revisar con la función miembro CheckTransform para verificar que la transformación desde el pin entrada hacia el pin salida es valida.

5.6.7.4 Función miembro DecideBufferSize

Los filtros de transformación pueden requerir el ajuste de las propiedades del asignador hacia el cual ellos realizarán la copia. En este caso, la función miembro **CBaseSalidaPin::DecideBufferSize** es llamada desde la función miembro **CBaseSalidaPin::DecideAllocator**, y la clase derivada ajusta los requerimientos para el

buffer mediante la llamada al método **IMemAllocator::SetProperties** en el objeto asignador para el cual este tiene una referencia.

5.6.7.5 Función miembro GetMediaType

Los pins proveen enumeradores para permitir a otros objetos determinar el tipo de media del pin. Un pin provee el enumerador del tipo de media(la interfaz IenumMediaTypes), cuya clase base implementa para llamar a la función miembro **GetMediaType**. Nuestra clase derivada debe implementar esta función miembro para proveer una lista de cada media soportada.

5.6.8 Filtro de efectos

En DirectShow los *filtros de efectos* se definen como aquellos que aplican algún efecto a los datos multimedia, pero que no cambian el tipo de los datos.

Dado que los formatos de entrada y salida son los mismos, y los efectos aplicados no pueden cambiar el formato, estos filtros contienen un código que checa continuamente el formato de los datos. Como utilizamos la clase CtransformFilter se deben utilizar los métodos **CheckMediaType**, **CTransformFilter::CheckInputType**, y **CheckTransform**.

El filtro de efectos debe implementar la interfaz **IPersistStream** si se quiere salvar el estado de los efectos en GraphEdit. Esto puede ser de mucha ayuda durante el diseño, y lo hemos implementado para regresar al estado inicial del filtro cuando se cierre el simulador GraphEdit.

Si se quiere poder manipular el efecto, se debe crear y desplegar su pagina propietaria y proveer un mecanismo para retornar las entradas del usuario hacia el filtro. Para lograrlo se implementa la clase **ISpecifyPropertyPages**, y se personaliza la interfaz que cambia los valores de la pagina propietaria. Se debe proveer también de los archivos fuente que despliegan los controles en la pagina propietaria.

Para implementar la clase de la pagina propietaria, se crea una clase que deriva de **CBasePropertyPage** y se implementa el método **OnReceiveMessage**, el método **CPersistStream::SetDirty**, y un dato miembro para cada parámetro de un efecto. Para acceder a las dos interfaces, se deriva la clase del filtro de efectos de **IspecifyPropertyPages** y la interfaz personalizada. Se puede consultar por las interfaces que se necesiten al sobrescribir el método **NonDelegatingQueryInterface** como se muestra en el siguiente ejemplo.

```
STDMETHODIMP CGargle::NonDelegatingQueryInterface(REFIID riid, void **ppv)
{
    CheckPointer(ppv,E_POINTER);
    if (riid == IID_IGargle) {
        return GetInterface((IGargle *) this, ppv);
    } else if (riid == IID_ISpecifyPropertyPages) {
        return GetInterface((ISpecifyPropertyPages *) this, ppv);
    } else if (riid == IID_IPersistFlujo) {
        return GetInterface((IPersistFlujo *) this, ppv);
```

```
    } else {
        return CTransInPlaceFilter::NonDelegatingQueryInterface(riid, ppv);
    }
}
```

La interfaz personalizada del filtro de efectos típicamente provee un método put y uno get para cada parámetro del efecto. Por ejemplo la interfaz IGargle provee los métodos put_GargleRate y get_GargleRate. Cuando el usuario accesa a uno de los controles en la pagina propietaria, la página genera un mensaje de la ventana. La función miembro OnReceiveMessage de la página propietaria maneja este mensaje. El siguiente código demuestra la generación del mensaje y su manejo. IDB_DEFAULT es el ID del botón Default. El usuario da clic en este botón para ajustar el contraste del video a un estado default. La clase CContrastPropierties implementa la pagina propietaria y el método IContrast::put_DefaultContrastLevel ajusta el nivel de contraste a su valor por default.

```
BOOL CContrastProperties::OnReceiveMessage(HWND hwnd, UINT uMsg,
    WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
       case WM_COMMAND:
        {
          if (LOWORD(wParam) == IDB_DEFAULT)
            {
            pIContrast()->put_DefaultContrastLevel();
            SendMessage(m_hwndSlider, TBM_SETPOS, TRUE, 0L);
              SetDirty();
        }
            return (LRESULT) 1;
        }
...
```

Los filtros de efectos usan secciones criticas internas para proteger el estado global del filtro. Los filtros de efectos pueden cerrar una sección critica para asegurar que el flujo de datos a través del gráfico de filtros es serializado y que el estado global del filtro no cambie mientras el efecto está ocurriendo. DirectShow cierra una sección critica al declarar un objeto de la clase CAutoLock . Típicamente los filtros de efectos cierran la región critica tan pronto como ellos entran a la función que aplica el efecto. En el siguiente fragmento, la función que aplica el efecto es MessItAbout:

```
// Declare the critical section data member in
// the effect filter class definition.
CCritSec   m_GargleLock;
void CGargle::MessItAbout(PBYTE pb, int cb)
{
    CAutoLock foo(&m_GargleLock);
```

Los métodos put y get de las propiedades de los efectos cierran las secciones criticas así los valores de los efectos no cambian a la mitad de una actualización.

5.6.9 Programación del filtro en C++

Como lo mencionamos arriba derivaremos nuestro filtro de la clase CtransformFilter.

```
class CEZrgb24 : public CtransformFilter
```

Implementamos la interfaz IUnknown, en la sección public de la definición de nuestra clase filtro creamos una instancia de CUnknown, y entonces llamamos la macro DECLER_IUNKNOWN.

```
public:
    DECLARE_IUNKNOWN;
    static CUnknown * WINAPI CreateInstance(LPUNKNOWN punk, HRESULT *phr);
```

Definimos el constructor en la sección private de la clase filtro.

```
private:
    // Constructor
    CEZrgb24(TCHAR *tszName, LPUNKNOWN punk, HRESULT *phr);
```

Añadimos el código para realizar las funciones transform y chequeo del InputType.

```
// Transform
// Copia la muestra de entrada hacia la muestra de salida - entonces transforma
// la muestra de salida 'in place'.
HRESULT CEZrgb24::Transform(IMediaSample *pIn, IMediaSample *pOut)
{
    // Copia las propiedades
    HRESULT hr = Copy(pIn, pOut);
    if (FAILED(hr)) {
        return hr;
    }

    // Checa para ver si es tiempo de hacer la muestra
    CRefTime tStart, tStop ;
    pIn->GetTime((REFERENCE_TIME *) &tStart, (REFERENCE_TIME *) &tStop);
    if (tStart >= m_effectStartTime) {
        if (tStop <= (m_effectStartTime + m_effectTime)) {
            return Transform(pOut);
        }
    }
    return NOERROR;
} // Transform

HRESULT CEZrgb24::CheckInputType(const CMediaType *mtIn)
{
    // checa que sea un tipo VIDEOINFOHEADER
    if (*mtIn->FormatType() != FORMAT_VideoInfo) {
        return E_INVALIDARG;
    }

    // Se puede transformar este tipo?
    if (CanPerformEZrgb24(mtIn)) {
          return NOERROR;
    }
    return E_FAIL;
}
```

La estructura **VIDEOINFOHEADER** describe el mapa de bits e información de la imagen de video.

Se implementa CreateInstance para nuestro objeto filtro.

```
// CreateInstance
//
// Usada por las clases base de DirectShow para crear instancias
//
CUnknown *CEZrgb24Properties::CreateInstance(LPUNKNOWN lpunk, HRESULT *phr)
{
    CUnknown *punk = new CEZrgb24Properties(lpunk, phr);
    if (punk == NULL) {
          *phr = E_OUTOFMEMORY;
    }
    return punk;
} // CreateInstance
```

Se declara un arreglo global g_Templates objeto CFactoryTemplate para informar a la clase factory como acceder a la función CreateInstance:

```
// La clase factory llamará a CreateInstance
CFactoryTemplate g_Templates[] = {
    { L"Image Effects"
    , &CLSID_EZrgb24
    , CEZrgb24::CreateInstance
    , NULL
    , &sudEZrgb24 }
  ,
    { L"Special Effects"
    , &CLSID_EZrgb24PropertyPage
    , CEZrgb24Properties::CreateInstance }
};
```

Observe que se ha añadido parámetros para la pagina propietaria que manejará el filtro.

Se genera un GUID par el objeto filtro, este identificador, permite que nuestro filtro tenga un numero único en todo el mundo, y así su posible uso en ambientes distribuidos. Para generar un GUID, se ejecuta el programa generador uuidgen.exe ubicado en el subdirectorio bin de directX. Finalmente se agrega el GUID para nuestro filtro quedando como sigue:

```
DEFINE_GUID(CLSID_EZrgb24,
0x8b498501, 0x1218, 0x11cf, 0xad, 0xc4, 0x0, 0xa0, 0xd1, 0x0, 0x4, 0x1b);
```

Dado que nuestro filtro implementa una interfaz que no se encuentra hecha en la clase base, nos referimos a la clase propietaria, entonces debemos escribir la función **NonDelegatingQueryInterface** y regresar apuntadores a la interfaz implementada. La función se coloca en la sección public de la definición de nuestra clase filtro.

```
// NonDelegatingQueryInterface
//
// Presenta las paginas propietarias IIPEffect y ISpecifyPropertyPages
//
STDMETHODIMP CEZrgb24::NonDelegatingQueryInterface(REFIID riid, void **ppv)
{
    CheckPointer(ppv,E_POINTER);

    if (riid == IID_IIPEffect) {
        return GetInterface((IIPEffect *) this, ppv);
    } else if (riid == IID_ISpecifyPropertyPages) {
        return GetInterface((ISpecifyPropertyPages *) this, ppv);
    } else {
        return CTransformFilter::NonDelegatingQueryInterface(riid, ppv);
    }
} // NonDelegatingQueryInterface
```

El Manejador del gráfico de filtros utiliza las entradas de registro de nuestro filtro para configurarlo asi como a sus conexiones. Nosotros proveemos la información de registro de nuestro filtro en las estructuras **AMOVIESETUP_MEDIATYPE**, **AMOVIESETUP_PIN**, y **AMOVIESETUP_FILTER.**

La estructura **AMOVIESETUP_MEDIATYPE** mantiene información de registro acerca de los tipos de media que nuestro filtro soporta.

```
// Información de Setup
const AMOVIESETUP_MEDIATYPE sudPinTypes =
```

```
{
    &MEDIATYPE_Video,       // tipo Major
    &MEDIASUBTYPE_NULL      // tipo Minor
};
```

Incorporamos la estructura **AMOVIESETUP_PIN** que mantiene un registro sobre los pines que nuestro filtro soporta.

```
const AMOVIESETUP_PIN sudpPins[] =
{
    { L"Input",             // Nombre de Pins
      FALSE,                // Será rendered
      FALSE,                // Este no es un salida
      FALSE,                // No estamos permitiendo ninguno
      FALSE,                // Permitimos muchos
      &CLSID_NULL,          // Conexiones al filtro
      NULL,                 // Conexion al pin
      1,                    // Numero de tipos
      &sudPinTypes          // Informacion del Pin
    },
    { L"Salida",            // Nombre del Pin
      FALSE,                // Sera rendered
      TRUE,                 // Este es un salida
      FALSE,                // No permitimos ninguno
      FALSE,                // Y soportamos muchos
      &CLSID_NULL,          // Conexiones al filtro
      NULL,                 // Conexion al pin
      1,                    // Numero de tipos
      &sudPinTypes          // Informacion del pin
    }
};
```

Proveemos la estructura **AMOVIESETUP_FILTER** que mantiene información de nuestro objeto filtro, a saber: su CLSID, descripción, numero de pins, el nombre de la estructura del pin, y el merito del filtro. El merito controla el orden en el cual el Manejador del gráfico de filtros accesa a nuestro filtro.

```
const AMOVIESETUP_FILTER sudEZrgb24 =
{
    &CLSID_EZrgb24,         // CLSID del filtro
    L"Image Effects",       // nombre
    MERIT_DO_NOT_USE,       // Merito del filtro
    2,                      // Numero de pins
    sudpPins                // Informacion del Pin
};
```

Solo queda entonces incluir un archivo de definición (.def) que exporta las funciones DLL, en nuestro caso:

```
LIBRARY             EZrgb24.ax
EXPORTS
                DllGetClassObject PRIVATE
                DllCanUnloadNow PRIVATE
                DllRegisterServer PRIVATE
                DllUnregisterServer PRIVATE
```

5.6.10 Algoritmos de procesamiento

Se han elaborado nueve procesamientos distintos sobre la imagen, todos incluidos dentro de la función Transform. Se explica cada uno de ellos junto con las limitaciones encontradas. Considerese en todos los casos a $\overline{F}(x, y)$ como la función vectorial de imagen

original con tres componentes R, G y B para cada color del espacio RGB y $\overline{G}(x,y)$ como la imagen resultado de la transformación.

5.6.10.1 Efecto rojo

Este es un filtro sencillo que elimina las componentes de color verde y azul del espacio de color RGB.

$$g_R(x,y) = \overline{F}(x,y) - \left(f_G(x,y) + f_B(x,y)\right)$$

Lo único que se hace es igualar las componentes no deseadas a cero y esto resulta bastante sencillo dado que se tienen un formato de video en píxeles RGB. La estructura RGBTRIPLE mantiene un miembro para cada componente de color. numPixels es el numero total de píxeles en la ventana de video. Prgb mantiene un apuntador al píxel actual.

```
prgb = (RGBTRIPLE*) pData;
for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
   prgb->rgbtGreen = 0;
   prgb->rgbtBlue = 0;
}
```

Figura 5.14 Filtro rojo

5.6.10.2 Efecto ruido

Se agrega ruido Gausiano a la imagen usando la función de generación de numeros aleatorios rand(), por lo tanto:

$\overline{G}(x,y) = \overline{F}(x,y)$ con una probabilidad $p = \frac{2}{3}$

y $\overline{G}(x,y) = 255$ con una probabilidad $p = \frac{1}{3}$

Dado que se tiene 1 byte por cada color, en decimal el máximo valor es 255 para cada componente. Igualando todas las componentes a 255 obtenemos el blanco y esto lo hacemos de manera aleatoria en cada píxel produciéndose un efecto de ruido en la proporción deseada. Dado que rand() % 3 produce números entre 0 y 2 entonces la tercera parte de los píxeles en la pantalla serán blancos.

```
prgb = (RGBTRIPLE*) pData;
```

```
for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
   if(!(rand() % 3)) {
      prgb->rgbtGreen = 255;
      prgb->rgbtBlue = 255;
      prgb->rgbtRed = 255;
   }
}
```

Figura 5.15 Adición de ruido

5.6.10.3 Efecto imagen BMP

Aquí se superpone una imagen previamente almacenada en un archivo del disco duro con formato BMP. La imagen se carga en un arreglo temporal, sin embargo esta imagen no puede contener un numero de píxeles mayor a 5000, porque el sistema se cae, considero que el motivo se debe a una cantidad de memoria preasignada al filtro y sobrepasarla la colapsa. Se pensó que el apuntador al píxel actual prgb siempre se encontraba en la esquina superior derecha de la ventana de video, sin embargo dependiendo de las características de la computadora y el tipo de procesamiento el origen se puede ubicar a en la izquierda y a la mitad del alto de la ventana de video. En nuestro caso se ubicó en la esquina inferior izquierda y el conteo de píxeles es hacia arriba y hacia la derecha. Es por ese motivo que la imagen BMP se debe almacenar invertida(de cabeza) para evitar un procesamiento adicional. La imagen se almacenó en el arreglo bidimensional bmiColors. El color verde puro se utilizó como un color transparente dado que nuestra imagen no lo contenía. Los for adicionales son para posicionar la imagen en la pantalla.

```
   prgb = (RGBTRIPLE*) pData;
   //offset de la imagen superpuesta
   for(i = 0; i < (int)((cyImage/25)*(1 + cxImage)); i++, prgb++);
   for(iPixel = 0; iPixel < 60; iPixel++) {
      for(jPixel = 0; jPixel < 98; jPixel++, prgb++){
         if(bmiColors[iPixel][jPixel].rgbGreen != 255) {
            prgb->rgbtRed = bmiColors[iPixel][jPixel].rgbRed;
            prgb->rgbtGreen = bmiColors[iPixel][jPixel].rgbGreen;
            prgb->rgbtBlue  = bmiColors[iPixel][jPixel].rgbBlue;
          }
       }
       for(jPixel = 98; jPixel < cxImage; jPixel++, prgb++);
`   }
```

Figura 5.16 Superposición de imagen BMP

5.6.10.4 Efecto imagen BMP diluida

En este efecto se diluye la imagen con el video, promediando los valores de la imagen y del video obteniéndose un 50% de cada una. Se puede obtener cualquier proporción de mezcla sin mayor problema. Asi entonces solo para la imagen BMP:

$$\overline{G}(x,y) = \frac{\overline{F}(x,y) + \overline{F}_{BMP}(x,y)}{2}$$

```
for(i = 0; i < (int)((cyImage/25)*(1 + cxImage)); i++, prgb++);

for(iPixel = 0; iPixel < 60; iPixel++) {
   for(jPixel = 0; jPixel < 98; jPixel++, prgb++){
      if(bmiColors[iPixel][jPixel].rgbGreen != 255) {
         prgb->rgbtRed   = (prgb->rgbtRed + bmiColors[iPixel][jPixel].rgbRed)/2;
         prgb->rgbtGreen = (prgb->rgbtGreen + bmiColors[iPixel][jPixel].rgbGreen)/2;
         prgb->rgbtBlue  = (prgb->rgbtBlue + bmiColors[iPixel][jPixel].rgbBlue)/2;
      }
    }
    for(jPixel = 98; jPixel < cxImage; jPixel++, prgb++);
}
```

Figura 5.17 Mezcla de imagen BMP con el video

5.6.10.5 Efecto Rayos X

Este es un efecto que produce el negativo de la imagen, aqui se invierten los bits, 1 por 0 y viceversa en la imagen digitalizada.

$$\overline{G}(x,y) = NOT\left(\overline{F}(x,y)\right)$$

```
prgb = (RGBTRIPLE*) pData;
for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
   prgb->rgbtRed   = (BYTE) (~prgb->rgbtRed);
   prgb->rgbtGreen = (BYTE) (~prgb->rgbtGreen);
   prgb->rgbtBlue  = (BYTE) (~prgb->rgbtBlue);
}
```

Figura 5.18 Efecto de inversión de bits

5.6.10.6 Efecto Poster

Se obtiene un **plano de bit** correspondiente a los tres bits mas significativos, esto es, se hacen cero los 5 bits menos significativos para cada componente de color, esto logra que las variaciones ligeras de color sean ignoradas y se tengan pocos colores en la imagen. Obsérvese que se logra haciendo una operación AND a nivel de bits del píxel con el byte 11100000 .

$$\overline{G}(x,y) = \overline{F}(x,y)AND(11100000)$$

```
prgb = (RGBTRIPLE*) pData;
for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
   prgb->rgbtRed   = (BYTE) (prgb->rgbtRed & 0xe0);
   prgb->rgbtGreen = (BYTE) (prgb->rgbtGreen & 0xe0);
   prgb->rgbtBlue  = (BYTE) (prgb->rgbtBlue & 0xe0);
}
```

Figura 5.19 Efecto de poster

5.6.10.7 Efecto Borroso

Para cada píxel se toma un píxel y el siguiente a dos píxeles a su derecha, los promedia y lo sobrescribe, lográndose así un efecto de imagen borrosa.

$$\overline{G}(x,y) = \frac{\overline{F}(x+1,y) + \overline{F}(x+2,y)}{2}$$

```
prgb = (RGBTRIPLE*) pData;
for (y = 0 ; y < pvi->bmiHeader.biHeight; y++) {
   for (x = 2 ; x < pvi->bmiHeader.biWidth; x++, prgb++) {
       prgb->rgbtRed   = (BYTE) ((prgb->rgbtRed + prgb[2].rgbtRed) >> 1);
       prgb->rgbtGreen = (BYTE) ((prgb->rgbtGreen + prgb[2].rgbtGreen) >> 1);
       prgb->rgbtBlue  = (BYTE) ((prgb->rgbtBlue + prgb[2].rgbtBlue) >> 1);
    }
    prgb +=2;
 }
```

Obsérvese la operación de bits que también podía ser aplicada al efecto BMP diluida, sin embargo dificultaría otras proporciones de la imágen.

Figura 5.20 Efecto de imagen borrosa

5.6.10.8 Efecto Gris

En este efecto se cambia la imagen a una escala de grises. Un calculo estándar en base a las componentes RGB es el siguiente:

$$g(x,y) = \frac{30 f_R(x,y) + 59 f_G(x,y) + 11 f_B(x,y)}{100}$$

Una mayor velocidad se obtiene con la siguiente simplificación:

$$g(x,y) = \frac{f_R(x,y) + f_G(x,y)}{2}$$

```
prgb = (RGBTRIPLE*) pData;
for (iPixel=0; iPixel < numPixels ; iPixel++, prgb++) {
   grey = (prgb->rgbtRed + prgb->rgbtGreen) >> 1;
   prgb->rgbtRed = prgb->rgbtGreen = prgb->rgbtBlue = (BYTE) grey;
}
```

Figura 5.20 Video en escala de grises

5.6.10.9 Efecto Realce

En este caso hacemos una detección de borde aplicando el gradiente a la función imagen.

$$\nabla f(x,y) = \left[G_x, G_y\right] = \left[\frac{\Delta f}{\Delta x}, \frac{\Delta f}{\Delta y}\right]$$

representada por las mascaras:

$$G_x = \frac{\Delta f}{\Delta x}$$

-1	1

*f(x,y)

$$G_x = \frac{\Delta f}{\Delta x}$$

-1
1

*f(x,y)

Si los valores de grises no son diferentes entonces se substituye un valor de gris intermedio, es decir (128, 128, 128). Si hay grandes diferencias (en contornos) entonces se alejarán mas de la escala de gris intermedio.

```
prgb = (RGBTRIPLE*) pData;
for (y = 0 ; y < pvi->bmiHeader.biHeight; y++) {
   grey2 = (prgb->rgbtRed + prgb->rgbtGreen) >> 1;
   prgb->rgbtRed = prgb->rgbtGreen = prgb->rgbtBlue = (BYTE) 128;
   prgb++;
   for (x = 1 ; x < pvi->bmiHeader.biWidth; x++) {
      grey = (prgb->rgbtRed + prgb->rgbtGreen) >> 1;
      temp = grey - grey2;
      if (temp > 127) temp = 127;
      if (temp < -127) temp = -127;
      temp += 128;
      prgb->rgbtRed = prgb->rgbtGreen = prgb->rgbtBlue = (BYTE) temp;
      grey2 = grey;
      prgb++;
   }
}
```

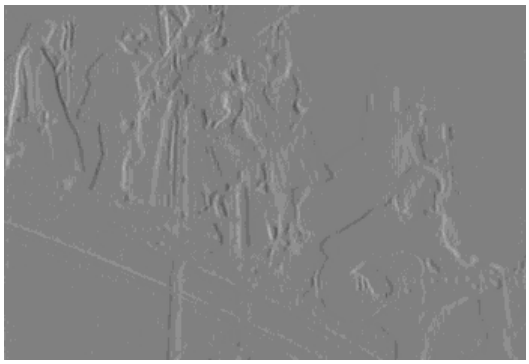


Figura 5.21 Efecto de realce

5.7 Aplicación

La aplicación construye en primera instancia el gráfico de filtros completo de manera automática. La configuración obtenida depende de la fuente seleccionada por el usuario (tarjeta de TV o un archivo multimedia), si la fuente es un archivo, se debe crear una instancia del admiministrador del grafico de filtros que cree el gráfico de filtros y lo ponga en ejecución.

Por otra parte debe proporcionar la ventana de video al sistema operativo para su despliegue en la ventana de la aplicación. También se requiere desplegar la pagina propietaria.

Si el usuario seleccionó la tarjeta de captura de video entonces se requiere la elaboración automática del gráfico de captura, para después insertar el filtro de efectos y finalmente la reproducción del video en la ventana de la aplicación. Veremos en los siguientes apartados casa uno de los pasos para la elaboración de nuestra aplicación.

5.7.1 Ejecución de un archivo

Una de las funciones de la aplicación es ejecutar un archivo multimedia. Esto incluye cuatro pasos básicos:

1. Crear una instancia del Manejador del Gráfico de filtros.
2. Usar el Manejador del gráfico de filtros para crear un grafico de filtros.
3. Usar el Manejador del gráfico de filtros para ejecutar el gráfico de filtros
4. Esperar a que la ejecución se complete

Para llevar a cabo estas operaciones se hace uso de las siguientes interfaces COM:

- IGraphBuilder: Construye el grafico de filtros.
- IMediaControl: Maneja el flujo de datos en el gráfico de filtros.
- IMediaEvent: Maneja los eventos en el gráfico de filtros.

Iniciamos llamando a la función **CoInitialize**, la cual inicializa la librería COM. Entonces llamamos a la función **CoCreateInstance** para crear el Manejador del gráfico de filtros.

```
IGraphBuilder *pGraph;
CoInitialize(NULL);
CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
                    IID_IGraphBuilder, (void **)&pGB);
```

La función **CoCreateInstance** regresa un apuntador a la interfaz **IgraphBuilder** del Manejador del gráfico de filtros. Utilizamos este apuntador para accesar a las otras dos interfaces **IMediaControl** y **IMediaEvent**:

```
IMediaControl *pMC = NULL;
IMediaEventEx *pME = NULL;
pGB->QueryInterface(IID_IMediaControl, (void **)&pMC);
pGB->QueryInterface(IID_IMediaEventEx, (void **)&pME);
```

Posteriormente se coloca el corazón del programa

```
pGB->RenderFile(L"C:\\Example.avi", NULL);
pMediaControl->Run();
```

El método **IGraphBuilder::RenderFile** construye un gráfico de filtros que ejecutará el archivo especificado. El prefijo "L" convierte una cadena ASCII a una cadena Unicode™ .

Después que el Manejador del gráfico de filtros ha construido el grafico, este se encuentra listo para ejecutarse. El método Run lo lleva a cabo, y los datos multimedia comienzan a circular en el grafico de filtros.

El gráfico de filtros queda configurado entonces como en la figura 5.23.

Figura 5.23 Configuración automática del gráfico de filtros

En el presente trabajo se requiere añadir el filtro de efectos que construimos por lo cual se procede a desconectar el filtro renderer y realizar la conexión de nuestro filtro como sigue:

```
// Desconectamos los últimos dos filtros del grafico
EnumFilters("MPEG Video Decoder", PINDIR_SALIDA);   //Busca el salida pin del penultimo
filtro...
hr = pGB->Disconnect(pOutPin);
EnumFilters("Video Renderer", PINDIR_INPUT);  //Busca el entradapin del ultimo filtro...
hr = pGB->Disconnect(pInputPin);
// Añadimos el filtro de efectos en la imagen
hr = CoCreateInstance(CLSID_EZrgb24, NULL, CLSCTX_INPROC_SERVER,
            IID_IBaseFilter, reinterpret_cast<void**>(&pFX));
hr = pGB->AddFilter(pFX, L"Image Effects");
//Conectamos filtro de efectos y MPEG video decoder
EnumFilters("Image Effects", PINDIR_INPUT); // Busca entradapin de filtro Efectos
hr = pGB->Connect(pOutPin, pInputPin);
//Raconexion automatica entre Image Effects y video render
EnumFilters("Image Effects", PINDIR_SALIDA);
EnumFilters("Video Renderer", PINDIR_INPUT);
hr = pGB->Connect(pOutPin, pInputPin);
```

El Gráfico de filtros resultante entonces se muestra en la figura 5.24.

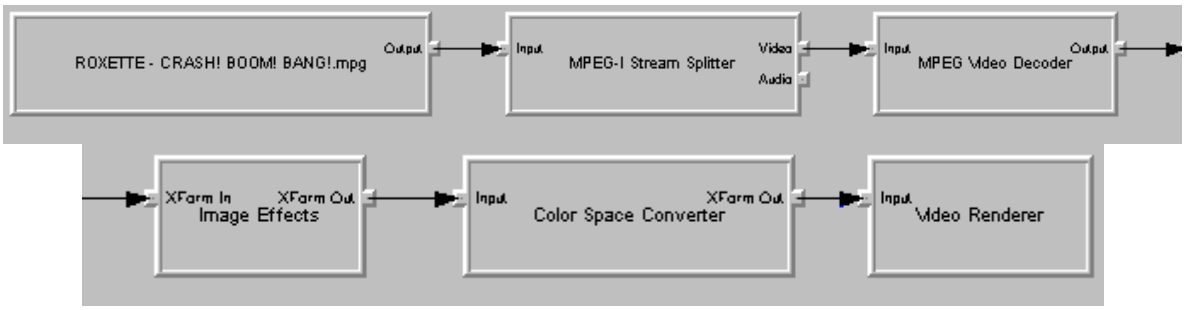


Figura 5.24 Gráfico con el filtro de efectos incorporado

5.7.2 Ajuste de la ventana de video

Cuando se presenta un archivo de video, el grafico de filtros contendrá un filtro video renderer. El video renderer toma datos de video sin comprimir como entrada y presenta estos hacia la pantalla dentro de una ventana. La ventana de video playback es una ventana totalmente independiente con sus propios bordes y barra de titulo. Pero si se requiere que aparezca en una ventana creada por nuestra aplicación, se debe hacer que la ventana de video sea una hija de la ventana de la aplicación. Esto se logra ajustando las propiedades de la ventana de video para especificar el propietario, el estilo, y posición de la ventana, mediante la interfaz **IVideoWindow**.

Para pegar la ventana de video en la de la aplicación entonces se llama al método **IVideoWindow::put_Owner** y se le pasa un handle a la ventana propietaria. Este método toma una variable del tipo OAHWND.

```
IVideoWindow  *pVW = NULL;
pGB->QueryInterface(IID_IVideoWindow, (void **)&pVW);
JIF(pVW->put_Owner((OAHWND)ghApp));
```

Se cambia el estilo de la ventana de video a una ventana hija. Esto mediante el método **IVideoWindow::put_WindowStyle** pasándole una combinación de banderas. La bandera WS_CHILD indica que la ventana es una ventana hija; la ventana WS_CLIPSIBLINGS previene a la ventana de dibujar dentro del área cliente de otra ventana hija.

```
pVW->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN);
```

Se ajusta la posición de la ventana de video mediante una llamada al método **IVideoWindow::SetWindowPosition**. Este método toma las coordenadas del dispositivo especificando la esquina superior izquierda, ancho y alto de la ventana.

```
RECT rect;
GetClientRect(ghApp, &rect);
pVW->SetWindowPosition(rect.left, rect.top, rect.right, rect.bottom);
```

Antes de que la aplicación salga, es importante que se ajuste la visibilidad de la ventana de video a false. De otro modo, un remanente de imagen de video quedaría en la pantalla y el usuario no podría quitarlo. Después se restaura el valor de owner a NULL; de otro modo los mensajes serán enviados a la ventana equivocada causando errores.

```
pVW->put_Visible(OAFALSE);
hr = pVW->put_Owner(NULL);
```

5.7.3 Despliegue de la pagina propietaria

El filtro de efectos múltiples que añadimos soporta una pagina propietaria mediante la cual podemos cambiar los efectos.

Los filtros con pagina propietaria exponen la interfaz **ISpecifyPropertyPages**. Para determinar si un filtro define una pagina propietaria, se consulta al filtro mediante **QueryInterface**.

Dado que nosotros creamos una instancia del filtro, ya tenemos un apuntador al filtro de efectos, entonces solo requerimos llamar al método **ISpecifyPropertyPages::GetPages**. Este método llena un arreglo de conteo de identificadores globales únicos (GUIDs) con los identificadores de clases (CLSID) de cada pagina propietaria. Un arreglo de conteo es definido por una estructura CAUUID, la cual se debe asignar pero no utilizar. El método **GetPages** localiza el arreglo, el cual está contenido en el miembro **pElems** de la estructura CAUUID. Cuando esto se haya realizado, se libera el arreglo mediante una llamada a la función **CoTaskMemFree**.

La función **OleCreatePropertyFrame** provee una forma simple de desplegar la pagina propietaria dentro de una caja de dialogo.

```
// Muestra la pagina propietaria
// Obtiene el nombre del filtro y un apuntador IUnknown.
FILTER_INFO FilterInfo;
pFX->QueryFilterInfo(&FilterInfo);
IUnknown *pFilterUnk;
pFX->QueryInterface(IID_IUnknown, (void **)&pFilterUnk);

CAUUID caGUID;
pProp->GetPages(&caGUID);
pProp->Release();
OleCreatePropertyFrame(
          hWnd,                  // Ventana padre
          0, 0,                  // (Reservado)
          FilterInfo.achName,    // Captura para la caja de dialogo
          1,                     // Numero de objetos (solo del filtro)
          &pFilterUnk,           // Arreglo de apuntadores a objetos
          caGUID.cElems,         // Numero de paginas propietarias
          caGUID.pElems,         // Arreglo de CLSIDs de paginas propietarias
          0,                     // Identificador local
          0, NULL                // Reservado
);
// Limpieza.
pFilterUnk->Release();
FilterInfo.pGraph->Release();
CoTaskMemFree(caGUID.pElems);
```

Esta es nuestra pagina propietaria con 9 efectos posibles:

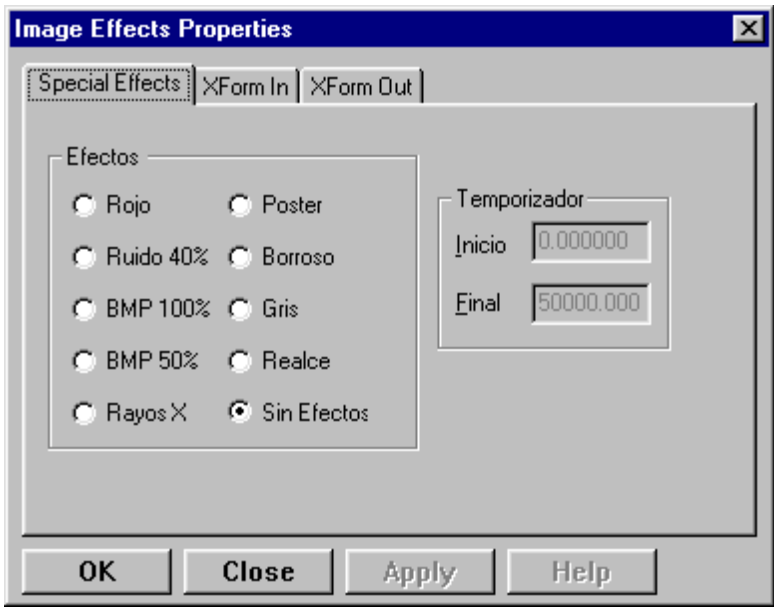


Figura 5.25 Pagina propietaria de nuestra aplicación

5.7.4 Captura de video WinTV

Entendemos por captura de video en nuestro caso no a la escritura de datos media sino a la previsualización de estos en una ventana de video. El dispositivo de captura es la tarjeta de video WinTV.

DirectShow maneja la captura de video también a través de filtros. Este tipo de gráfico se denomina *gráfico de captura*.

Si un dispositivo de captura utiliza un controlador Windows Driver Model(WDM), el grafico puede necesitar ciertos filtros upstream del filtro de captura. Estos filtros denominados filtros manejadores de clase flujo, soportan funcionalidades adicionales provistas por el hardware. Por ejemplo, una tarjeta de sintonía de TV tiene una funcionalidad para ajustar el canal.

El diagrama 5.26 muestra los tipos de filtros en un grafico de captura típico. Las líneas punteadas indican donde los filtros adicionales pueden requerirse, tales como filtros decodificadores o filtros splitter.

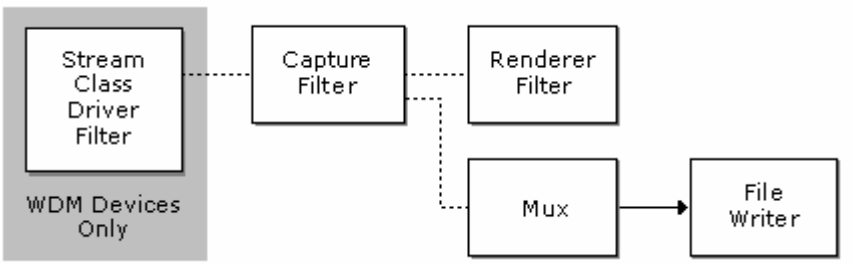


Figura 5.26 Gráfico de captura típico

Para simplificar el proceso de construcción de gráficos de captura, DirectShow provee una componente llamada el Capture Graph Builder. Este expone la interfaz **ICaptureGraphBuilder2**, la cual tiene métodos para construir y controlar gráficos de captura.

Un filtro de captura entrega datos a través de uno o más pins salida. Un pin salida puede ser clasificado por el tipo de datos media que este entrega, y por su *pin category*.
El tipo de media se representa por un major type GUID . Los tipos más comunes de datos media se muestran en la siguiente tabla.

Tipo de Media	Major type GUID
Audio	MEDIATYPE_Audio
Interleaved digital video (DV)	MEDIATYPE_Interleaved
MPEG Flujo	MEDIATYPE_Flujo
Video	MEDIATYPE_Video

Pin category describe el propósito o función del pin, y es representada por un GUID propietario. Hay varias pin category, pero las dos utilizadas son:

- PIN_CATEGORY_CAPTURE: El pin provee datos para capturar a un archivo.
- PIN_CATEGORY_PREVIEW: El pin provee datos para previsualizar.

Se puede consultar a un pin directamente por su tipo de media y su pin category. Para obtener el tipo de media, se llama al método **IPin::EnumMediaTypes**. Para obtener el pin category, se llama al método **IKsPropertySet::Get**. Utilizando los métodos, se puede limitar las operaciones a un tipo de media determinado o pin category, sin consultar a cada pin. A pesar de que por lógica debiéramos utilizar PIN_CATEGORY_PREVIEW en las pruebas con las cuatro pantallas no funciona, sin embargo con PIN_CATEGORY_CAPTURE si, una posible razòn es el tipo de tarjeta de adquisición de video utilizada.

5.7.4.1 Gráfico de captura

El primer paso para crear el gráfico de captura es crear el Manejador del gráfico de filtros y el Capture Graph Builder. Entonces se llama a para asociar el gráfico de filtros con el capture graph builder. como sigue:

```
IGraphBuilder *pGraph = NULL;
ICaptureGraphBuilder2 *pBuilder = NULL;

// Crea el FGM
CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC, IID_IGraphBuilder, (void **)&pGB);
// Crea el Capture Graph Builder.
CoCreateInstance(CLSID_CaptureGraphBuilder2, NULL, CLSCTX_INPROC, IID_ICaptureGraphBuilder2,
(void **)&pBuilder);
// Asocia el gráfico con el builder
pBuilder->SetFiltergraph(pGB);
```

El siguiente paso es seleccionar un dispositivo de captura, utilizando el System Device Enumerator. El siguiente código enumera los dispositivos de captura de video y selecciona el primer dispositivo:

```
// Crea el system device enumerator.
ICreateDevEnum *pDevEnum = NULL;
CoCreateInstance(CLSID_SystemDeviceEnum,  NULL,  CLSCTX_INPROC,  IID_ICreateDevEnum,  (void
**)&pDevEnum);

// Crea un enumerador para video capture devices.
IEnumMoniker *pClassEnum = NULL;
pDevEnum->CreateClassEnumerator(CLSID_VideoInputDeviceCategory, &pClassEnum, 0);

ULONG cFetched;
IMoniker *pMoniker = NULL;
IBaseFilter *pSrc = NULL;
if (pClassEnum->Next(1, &pMoniker, &cFetched) == S_OK)
{
   // Liga el primer moniker a un objeto filtro.
   pMoniker->BindToObject(0, 0, IID_IBaseFilter, (void**)&pSrc);
   pMoniker->Release();
}
pClassEnum->Release();
pDevEnum->Release();
```

Ahora se añade el filtro de captura al gráfico:

```
// Se añade el filtro de captura a nuestro gráfico
hr = pGB->AddFilter(pSrc, L"Video Capture");
```

El paso final en la construcción de un grafico de captura es presentar el flujo. El método **ICaptureGraphBuilder2::RenderFlujo** realiza dicha función.

```
// Muestra el pin de captura en el filtro de captura de video.
// Se usa esta instrucción en lugar de pBuilder->RenderFile
hr = pBuilder->RenderFlujo (&PIN_CATEGORY_CAPTURE , &MEDIATYPE_Video,
                                pSrc, NULL, NULL);
```

Los primeros dos parámetros especifican el pin category y el tipo de media. Los siguientes tres parámetros son apuntadores a los siguientes filtros:

- El filtro de captura
- Un filtro de compresión adicional (no usado)
- El multiplexor o filtro renderer. (no usado)

Finalmente y como en el apartado 5.7.1 se incorpora el filtro multiefectos.

5.8 Resultados

Se logró la elaboración del filtro aunque su inclusión en la aplicación resultó problemática, debido a que no se quería reconocer la existencia del filtro como librería de enlace dinámico, sin embargo se pudo finalmente resolver el problema con ayuda del grupo de usuarios especificado en el anexo.

En la figura 5.27 se muestra una imagen en pantalla completa del videowall funcionando. El video ofrece una presentación aceptable aunque se observa cierto retardo en la actualización de la imagen, sobre todo en las pantallas secundarias. Al ser reproducido

el video a partir de la tarjeta de video WinTV, se observan los mismos problemas anteriores pero de un modo mas significativo. También se aprecia un desfasamiento entre el audio y el video, presuponemos que otra tarjeta de video pudiera dar resultados distinto.

Figura5.27 Videowall de 4 monitores a pantalla completa

6. Evaluación

Las pruebas han sido parte interactiva del proyecto, y se han realizado en paralelo con el desarrollo. Cada efecto se probó hasta obtener un desempeño adecuado, y esto significa que la imagen se reproduzca sin pausas y sin omitir frames.

Estas pruebas se realizaron solo en el modo de una tarjeta de video conectada, en este caso nos referimos a la **ATI 3D Rage Pro** conectada al puerto PCI y en la configuración de 1024 x 768 a 24 bits.

Hubo un efecto adicional que no fue incluido debido a que no se pudo obtener el correcto funcionamiento del mismo, y se trató de obtener una animación que se superponga al video. Dicha animación se lograría almacenando en un buffer la siguiente imagen:

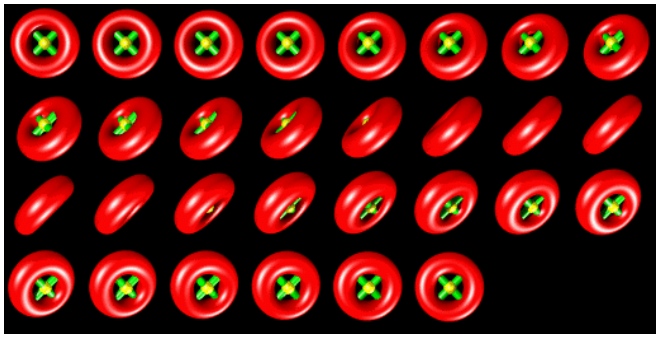


Figura 6.1 Imagen para animación

Cada una de las llantas se iría presentando consecutivamente superpuesta al video, lo cual daría el efecto de giro de la llanta. Sin embargo a pesar de los distintos métodos utilizados, no se logro el resultado esperado, debido a que solo se presentaba la primer imagen y no se hacia el cambio hacia la siguiente imagen. La posible razón es que la ejecución del código en la función Transform( ) no se respeta debido a que tiene prioridad la presentación de la muestra de video hacia la ventana de la aplicación, y entonces se omiten las instrucciones de código subsiguientes, reiniciándose el hilo de ejecución.

En todos los casos se ha tenido problemas en la presentación para las 4 pantallas al mismo tiempo, esto es en las pruebas del videowall, sobre todo cuando la fuente de video proviene de la tarjeta WinTV. El primer problema reside en la omisión de frames, provocando un video discontinuo, y el segundo problema consiste en la desincronización entre la señal de video y de audio, esto se pone de manifiesto cuando la imagen se va "retrasando" y los videos no se presentan con la rapidez adecuada. Se realizaron distintas pruebas y configuraciones para determinar las posibles causas.

Debido a que no hemos establecido un método para cuantificar el error producido en la presentación del video solo hacemos un comentario subjetivo sobre cual es la que exhibe una mejor imagen.

El primer problema en la configuración de videowall es que la imagen de video solo se puede mostrar en la pantalla principal, el motivo tiene que ver con el hardware de adquisición de video. Tanto en el modo de superposición de video como en el modo de superficie principal, la imagen digitalizada adquirida por WinTV se envía directamente a la memoria de la tarjeta VGA principal por el bus PCI.

Para lograr que se visualice en los demás monitores, se requiere hacer uso del pin Capture en la tarjeta WinTV, así en lugar de enviar la información a la pantalla principal se envía a un buffer que después es accesado por el sistema operativo para poder hacer la presentación en múltiples pantallas. Otra forma es la de obligar al sistema con el siguiente gráfico de filtros:

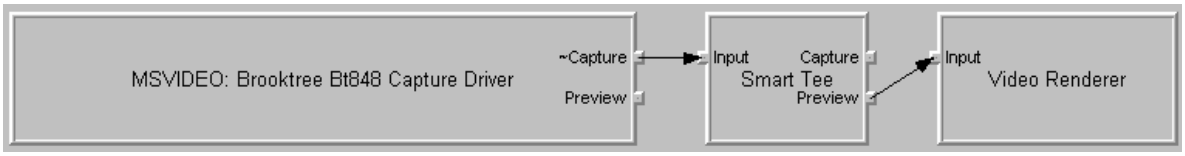


Figura 6.2 Incorporación del filtro Smart Tee

El filtro Smart Tee tiene las siguientes particularidades:

- Entrega los datos en preview únicamente cuando al hacerlo no se dañe el performance del pin Capture
- Remueve los time stamps del flujo preview, esto evita que el renderer elimine frames innecesariamente.

Desafortunadamente en ambos casos se siguen obteniendo errores en la presentación del video.

La afirmación anterior se deduce a partir de las pruebas realizadas con una cámara web, la cual no envía directamente las imágenes de video hacia la memoria de la tarjeta VGA, sino que pasan a un buffer intermedio, lo cual no provoca errores.

Los errores en la presentación de video tienen dos causas principales a saber:

1. Las dimensiones de la imagen adquirida por WinTV
2. El formato de la imagen obtenida por WinTV

En el primer caso la dimensión con la que puede adquirirse la imagen toma 9 posibles valores:
160x112, 160x120, 176x144, 192x142, 320x240, 352x288, 384x284, 400x320, 640x480.
La cantidad de pixeles obtenidos varia entonces desde 17,920 hasta 307,200.

Haciendo uso de la herramienta de Windows "Monitor del sistema" pudimos observar que para una dimensión de 320x240 el procesador se sobrecargaba a 100%, sin embargo para una dimensión de 640x480 el procesador solo trabajaba al 40%. En contraparte el mayor flujo de datos debido a una mayor resolución satura mas el bus y por consiguiente la imagen exhibe mas discontinuidades.

Dentro de la segunda causa se tienen 7 posibles formatos de adquisición:
32 bit RGBA, 24bit RGB, 15 bit RGB, YUY2, BTYUV, YUV9, YUV12.

Dependiendo del formato deseado se insertan filtros o no en el gráfico, esto requerirá un tiempo de procesamiento adicional dependiendo del tipo de filtro insertado. Cabe aquí aclarar que es importante mantener una resolución de la tarjeta VGA alta para evitar alguna conversión de color, en nuestro caso se mantuvieron todas a 24 bits como mínimo.

7. Conclusiones y perspectivas

Se ha obtenido el software para el procesamiento de video en tiempo real, cuyas capacidades se han explorado de manera experimental y han dejado ver las caracteristicas que exhiben los modernos procesadores de proposito general. Dado que el procesamiento de video cubre un amplio rango de posibilidades y complejidad, se comenzò realizandose procesamiento bàsico que involucrase pocas operaciones matematicas, y fuè incrementandose la complejidad de dichas operaciones hasta obtenerse defectos en el video. Considerese la dificultad de determinar las capacidades del presente software con antelación, considerando que se desconoce el porcentaje del procesador utilizado en las tareas bàsicas del sistema operativo. Entre las que se añadió el control simultaneo de varias tarjetas de video.

El procesamiento de video en tiempo real para los efectos descritos se ha logrado de manera satisfactoria solo en el caso de una pantalla, esto es no trabajando simultaneamente con el videowall debido a una considerable perdida en la calidad visual. El procesamiento que requiere la división de imágenes para formar un videowall se obtiene también de modo aceptable al trabajar de manera independiente a los efectos.

La determinaciòn de la variación de la calidad del video respecto a la cantidad de operaciones matematicas efectuadas, no se realizó, pero esta puede ser también una de las tareas siguientes sobre todo por la necesidad de evaluar el desempeño en este ambito con los nuevos procesadores y desarrollos de hardware.

Las herramientas de software DirectX sobre las cuales se ha construido nuestra aplicaciòn han sido fundamentales para el presente trabajo, algunas funciones que se han implementado en forma de objetos componentes han sido posibles tambièn solo gracias al apoyo del grupo de usuarios en internet. En otros casos se detectaron errores en el software de desarrollo DirectShow que produjeron salidas inesperadas y que se solventarón con algun truco no documentado. Estas modificaciones se exponen via codigo fuente con su respectivo comentario en el presente trabajo.

Es recomendable que se realicen pruebas con otros procesadores distintos al Intel® y con algunos dispositivos de adquisición de video diferentes, se espera también que otros sistemas operativos proporcionen características adicionales que puedan mejorar el desempeño.

Dentro de los tipos de procesamiento de video en tiempo real se tiene una amplia área de trabajo, ya sea desarrollando métodos de compresión o realizando procesamiento experimental sobre video en tiempo real.

Otra alternativa interesante es el procesamiento de video distribuido, considerando que se tienen velocidades altas en la transmisión de red local y que se dispone de mecanismos diversos de compresión se puede realizar algunos proyectos con mas de una computadora.

Se espera que el presente trabajo sea de utilidad para aquellos que se interesen en procesamiento de video a bajo costo, por lo pronto es difícil la inclusión de una mayor cantidad de monitores no solo por la limitante del numero de puertos PCI sino también porque se requiere mayor velocidad de procesamiento. Se puede incrementar el rendimiento si en posteriores versiones de DirectX se puede manejar pantalla completa en los monitores secundarios. Dentro de las pruebas hechas en LINUX el rendimiento fue muy bajo en el uso de multimonitor, aunque se tiene la ventaja de acceso a modificación del kernel.

Las aplicaciones de tiempo real en la actualidad también hacen uso de sistemas operativos dedicados y esta me parece otra gran alternativa para mantener el bajo costo y aumentar el desempeño del procesamiento.

Consultas realizadas

[1] Watkinson, John, ***An introduction to digital video,*** Focal Press, 1994

[2] Luther, Arch C., ***Principles of digital audio and video,*** Artech House, Inc., 1997

[3] Poynton, Charles A., ***A technical introduction to digital video***, John Wiley & Sons, 1996

[4] Bradley Bargen, ***Inside DirectX ,*** Mc Graw Hill, 1998

[5] Barry B. Brey, ***Los microprocesadores intel****, arquitectura, programación e interfases*, Prentice Hall, 1995

[6] Phillip E. Mattison, ***Practical digital video with programming examples in C***, John Wiley & Sons, 1994

[7] Rafael C. Gonzalez, ***Procesamiento digital de imágenes***, Addison Wesley, 1995

[8] Sean Wallace, ***Digital video effects processor***, Sr. Design Project, 1998

[9] Jorge Pascual, Francisco Charte, ***Programación Avanzada en Windows 2000***, Mc Graw Hill, 1994

[10] David Bennett, ***Visual C++ 5 Para Desarrolladores***, Prentice Hall, 1999

[11] Andrew Dowsey Meng, **DirectShow TV Display with Real –Time Video Processing**, http://keeper.warhead.org.uk

[12] M. Robin & M. Poulin, **Digital Television Fundamentals**

[13] G. Eddon, H. Eddon, **Inside Distributed COM,** Microsoft Press, (1998)

[14] Documentación Microsoft DirectShow, http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/directx/dx8_c/ds/default.htm

[15] Fco. Javier Cevallos, **Visual C++ 6 Aplicaciones para Win32**, Alfaomega

[16] Fco. Javier Cevallos, **Visual C++6 Programación Avanzada en Win32**, Alfaomega

Anexos

Fuentes de internet

[1] MSDN newsgroup
http://www.microsoft.com/isapi/redir.dll?prd=ie&pver=5.0&ar=IStart

[2] Corporación Intel
http://www.intel.com

[3] AV Science Forum, Sección Home Theatre Computers
http://www.avsforum.com/ubbcgi/Ultimate.cgi?action=intro&showall=1

[4] Hauppauge Computer Works
http://www.hauppauge.com

[5] AVerMedia
http://www.avermedia.com

[6] ATI Technologies Inc.
http://www.ati.com

[7] DirectX Low-Level Extensions for Windows, Microsoft Corporation,
http://www.microsoft.com/directx

[8] Video4Linux2
http://www.thedirks.org/v4l2/

[9] Advanced Micro Devices (AMD)
http://www.amd.com

[10] Procesamiento en tiempo real usando DSP
http://dynamio.enc.purdue.edu/~ace/delp-pub.html

[11] Sun Microsystems (JMF) Java Media Framework,
http://java.sun.com/products/java-media/jmf/index.html

Manual de usuario

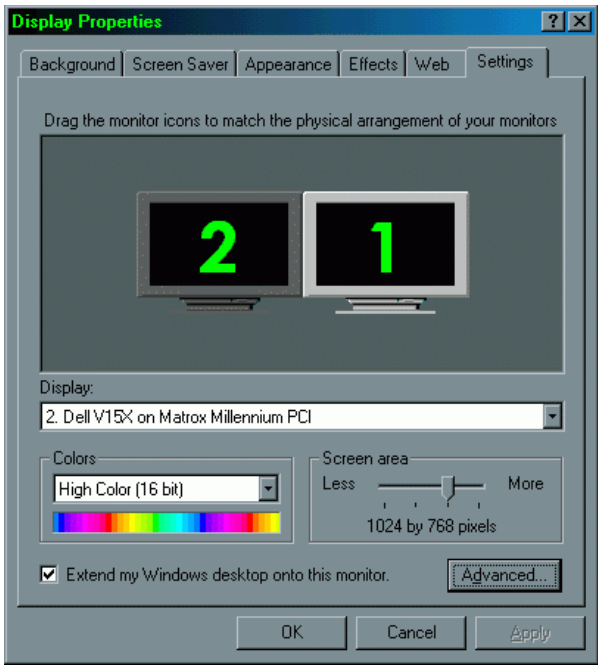
El programa DEXTERMedia se auto instala automáticamente a partir de la inserción del CD. Este proceso incluirá la librería de enlace dinámico ezrgb24.ax, la cual contiene los efectos de filtrado.

Para su funcionamiento en cuatro pantallas como videowall se requiere instalar en los puertos PCI libres las cuatro tarjetas de video y hacer la siguiente configuración.

Configuración de 4 monitores

Considerando que se tiene una tarjeta de video instalada, se añade la siguiente tarjeta con el equipo apagado y al reiniciar, el sistema deberá reconocer la nueva tarjeta. Si todo va bien solo se debe ir hacia Display Properties: Se da clic en Start, Settings, Control Panel, Display, y finalmente en, Settings.

Para habilitar el nuevo dispositivo, simplemente se da clic sobre el nuevo monitor, y entonces se selecciona "Extend my Windows desktop onto this monitor."



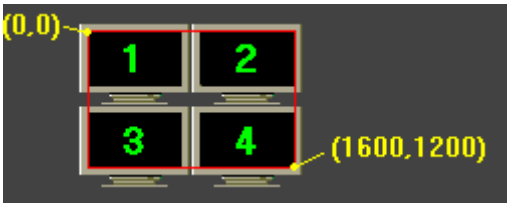
En el caso de que se tenga problemas en saber cual monitor es cual, sobre todo cuando se tienen 4 monitores o más, solo se da click con el botón derecho sobre el monitor y se selecciona Identify. Entonces aparecerá el numero correspondiente al monitor actual, el monitor principal tiene el numero 1 por default.

De la misma forma se añaden los subsiguientes monitores hasta obtener los 4 monitores. Posteriormente se procede a configurarlos todos con la misma resolución 800x600 y la cantidad de colores deberá ser de por lo menos 24 bits en todas las tarjetas.

Dado que Windows utiliza un sistema de coordenadas para referenciar cualquier elemento sobre la pantalla. La esquina superior izquierda en una pantalla con resolución 800x600 es la coordenada 0,0. La coordenada de la esquina inferior derecha es 800, 600.

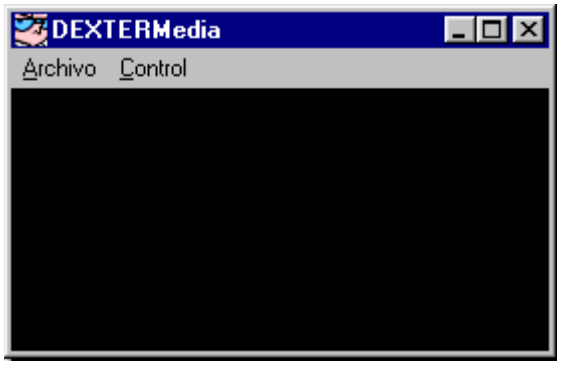
En el caso de múltiple monitor, la esquina superior izquierda del monitor primario tiene la coordenada 0,0. Si tenemos un sistema con 2 monitores a 1024x768, con el secundario colocado al lado derecho del primario. Entonces la esquina inferior derecha de la pantalla virtual será 2048x768.

La configuración de monitores debe realizarse de manera que se obtenga la siguiente pantalla virtual.



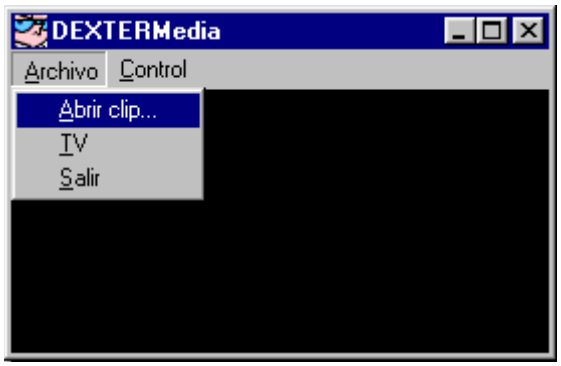
La aplicación

Al ejecutar la aplicación podemos observar la siguiente ventana principal.



Se tiene en la barra de Menús solo dos elementos, **Archivo** y **Control**, los cuales pueden accederse mediante la combinación de teclas ALT+A y ALT+C respectivamente.

El menú **Archivo**, muestra las siguientes opciones



La opción **Abrir clip...**, permite abrir archivos multimedia con extensiones avi, mov o mpg, ubicados en el disco duro o en CD-ROM. En el caso de que el archivo multimedia no sea compatible se indicará en el sistema un error de lectura.

La opción **TV** visualiza la imagen de televisión proporcionada por una tarjeta de televisión conectada al sistema.

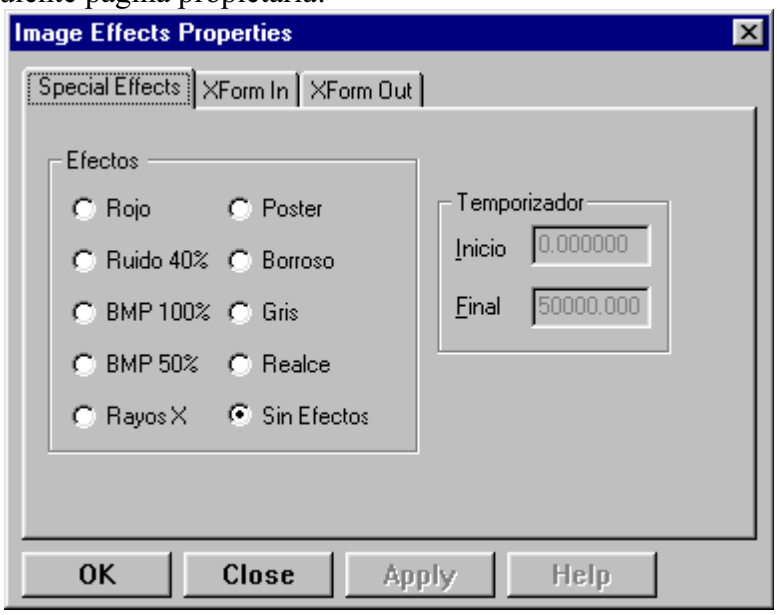
El menú Control muestra las siguientes opciones

La opción **Pausa/Reproducir** permite pasar del estado de reproducción al de pausa y viceversa. **Detener** para la reproducción del archivo multimedia, **Cerrar clip** ocasiona que se liberen los recursos asociados con la reproducción del video actual. **Silencio/Sonido** detiene el sonido pero continua la reproducción del video o la televisión.

La opción **Media pantalla** provoca que la ventana tome una dimensión del 50% de la original, del mismo modo **Tres cuartos** el 75% y **Normal** el 100%. **Videowall** permite observar la imagen de video en los cuatro monitores. Finalmente **Pantalla Completa** presentará la imagen en la totalidad de la pantalla.

Los efectos de video

Para acceder a los distintos efectos de video, solo se presiona la tecla F!, con lo cual aparecerá la siguiente pagina propietaria:



Para iniciar el efecto se selecciona el ovalo correspondiente y posteriormente el botón **Apply**, con lo cual este se ejecutará inmediatamente.

Código Fuente del filtro

```
//------------------------------------------------------------------------------
// Archivo: EZRGB24.cpp
//
// Desc: Filtro de efectos especiales en la imagen
//
//------------------------------------------------------------------------------

// Resumen
//
// Este es un filtro de efectos especiales y trabaja solo con formatos RGB24
// Este filtro puede ser insertado entre decodificadores de video y renderers
//
// Archivos
//
// ezprop.cpp           Pagina propietaria para controlar los efectos de video
// ezprop.h             Definición de clases para el objeto pagina propietaria
// ezprop.rc            Template para la caja de dialogo en la pagina propietaria
// ezrgb24.cpp          Codigo del filtro principal que hace los efectos especiales
// ezrgb24.def          Que APIs importamos y exportamos desde esta DLL
// ezrgb24.h            Definicion de clases para los filtros de efectos especiales
// ezuids.h             Archivos Header conteniendo los CLSID del filtro
// iez.h                Define la interfaz personalizada de los efectos especiales
// makefile
// resource.h           Archivos generados por Microsoft Visual C++
//
//
// clases Base utilizadas
//
// CTransformFilter     Un filtro transform con un input y un output
// CPersistStream       Maneja los soportes para IPersistStream
//
//


#include <windows.h>
#include <streams.h>
#include <initguid.h>
#if (1100 > _MSC_VER)
#include <olectlid.h>
#else
#include <olectl.h>
#endif
#include "EZuids.h"
#include "iEZ.h"
#include "EZprop.h"
#include "EZrgb24.h"
#include "resource.h"


// Información de Setup

const AMOVIESETUP_MEDIATYPE sudPinTypes =
{
    &MEDIATYPE_Video,       // tipo Major
    &MEDIASUBTYPE_NULL      // tipo Minor
};

const AMOVIESETUP_PIN sudpPins[] =
{
    { L"Input",             // Nombre de Pins
      FALSE,                // Será rendered
      FALSE,                // Este no es un output
      FALSE,                // No estamos permitiendo ninguno
      FALSE,                // Permitimos muchos
      &CLSID_NULL,          // Conexiones al filtro
      NULL,                 // Conexion al pin
      1,                    // Numero de tipos
      &sudPinTypes          // Informacion del Pin
```

```
    },
    { L"Output",            // Nombre del Pin
      FALSE,                // Sera rendered
      TRUE,                 // Este es un output
      FALSE,                // No permitimos ninguno
      FALSE,                // Y soportamos muchos
      &CLSID_NULL,          // Conexiones al filtro
      NULL,                 // Conexion al pin
      1,                    // Numero de tipos
      &sudPinTypes          // Informacion del pin
    }
};

const AMOVIESETUP_FILTER sudEZrgb24 =
{
    &CLSID_EZrgb24,         // CLSID del filtro
    L"Image Effects",       // nombre
    MERIT_DO_NOT_USE,       // Merito del filtro
    2,                      // Numero de pins
    sudpPins                // Informacion del Pin
};

// Lista de clases Ids y funciones para la clase factory. Esta provee el enlace
// entre el punto de entrada OLE en la DLL y un objeto que será creado.
// La clase factory llamará a CreateInstance

CFactoryTemplate g_Templates[] = {
    { L"Image Effects"
    , &CLSID_EZrgb24
    , CEZrgb24::CreateInstance
    , NULL
    , &sudEZrgb24 }
  ,
    { L"Special Effects"
    , &CLSID_EZrgb24PropertyPage
    , CEZrgb24Properties::CreateInstance }
};
int g_cTemplates = sizeof(g_Templates) / sizeof(g_Templates[0]);


//
// DllRegisterServer
//
// Maneja el registro y desregistro de la muestra
//
STDAPI DllRegisterServer()
{
    return AMovieDllRegisterServer2( TRUE );

} // DllRegisterServer


//
// DllUnregisterServer
//
STDAPI DllUnregisterServer()
{
    return AMovieDllRegisterServer2( FALSE );

} // DllUnregisterServer


//
// Constructor
//
CEZrgb24::CEZrgb24(TCHAR *tszName,
                   LPUNKNOWN punk,
                   HRESULT *phr) :
    CTransformFilter(tszName, punk, CLSID_EZrgb24),
    m_effect(IDC_NONE),
    m_lBufferRequest(1),
```

```
    CPersistStream(punk, phr)
{
    char sz[60];
    GetProfileStringA("Quartz", "EffectStart", "0.0", sz, 60);
    m_effectStartTime = COARefTime(atof(sz));
    GetProfileStringA("Quartz", "EffectLength", "50000.0", sz, 60);
    m_effectTime = COARefTime(atof(sz));

} // (Constructor)


//
// CreateInstance
//
// Provee la via para que COM cree un objeto EZrgb24
//
CUnknown *CEZrgb24::CreateInstance(LPUNKNOWN punk, HRESULT *phr)
{
    CEZrgb24 *pNewObject = new CEZrgb24(NAME("Image Effects"), punk, phr);
    if (pNewObject == NULL) {
        *phr = E_OUTOFMEMORY;
    }
    return pNewObject;

} // CreateInstance


//
// NonDelegatingQueryInterface
//
// Presenta las paginas propietarias IIPEffect y ISpecifyPropertyPages
//
STDMETHODIMP CEZrgb24::NonDelegatingQueryInterface(REFIID riid, void **ppv)
{
    CheckPointer(ppv,E_POINTER);

    if (riid == IID_IIPEffect) {
        return GetInterface((IIPEffect *) this, ppv);
    } else if (riid == IID_ISpecifyPropertyPages) {
        return GetInterface((ISpecifyPropertyPages *) this, ppv);
    } else {
        return CTransformFilter::NonDelegatingQueryInterface(riid, ppv);
    }

} // NonDelegatingQueryInterface


//
// Transform
//
// Copia la muestra de entrada hacia la muestra de salida - entonces transforma
// la muestra de salida 'in place'.
HRESULT CEZrgb24::Transform(IMediaSample *pIn, IMediaSample *pOut)
{
    // Copia las propiedades

    HRESULT hr = Copy(pIn, pOut);
    if (FAILED(hr)) {
        return hr;
    }

    // Checa para ver si es tiempo de hacer la muestra

    CRefTime tStart, tStop ;
    pIn->GetTime((REFERENCE_TIME *) &tStart, (REFERENCE_TIME *) &tStop);

    if (tStart >= m_effectStartTime) {
        if (tStop <= (m_effectStartTime + m_effectTime)) {
            return Transform(pOut);
        }
    }
```

```
    return NOERROR;

} // Transform


//
// Copy
//
// Hace en el destino una copia identica de la fuente Make
//
HRESULT CEZrgb24::Copy(IMediaSample *pSource, IMediaSample *pDest) const
{
    // Copia los datos en la muestra

    BYTE *pSourceBuffer, *pDestBuffer;
    long lSourceSize = pSource->GetActualDataLength();

#ifdef DEBUG
    long lDestSize  = pDest->GetSize();
    ASSERT(lDestSize >= lSourceSize);
#endif

    pSource->GetPointer(&pSourceBuffer);
    pDest->GetPointer(&pDestBuffer);

    CopyMemory( (PVOID) pDestBuffer,(PVOID) pSourceBuffer,lSourceSize);

    // Copia los tiempos de la muestra

    REFERENCE_TIME TimeStart, TimeEnd;
    if (NOERROR == pSource->GetTime(&TimeStart, &TimeEnd)) {
        pDest->SetTime(&TimeStart, &TimeEnd);
    }

    LONGLONG MediaStart, MediaEnd;
    if (pSource->GetMediaTime(&MediaStart,&MediaEnd) == NOERROR) {
        pDest->SetMediaTime(&MediaStart,&MediaEnd);
    }

    // Copia los puntos propietarios Sync

    HRESULT hr = pSource->IsSyncPoint();
    if (hr == S_OK) {
        pDest->SetSyncPoint(TRUE);
    }
    else if (hr == S_FALSE) {
        pDest->SetSyncPoint(FALSE);
    }
    else {  // un error inesperado ocurrió...
        return E_UNEXPECTED;
    }

    // Copia el tipo de media

    AM_MEDIA_TYPE *pMediaType;
    pSource->GetMediaType(&pMediaType);
    pDest->SetMediaType(pMediaType);
    DeleteMediaType(pMediaType);

    // Copia las propiedades preroll

    hr = pSource->IsPreroll();
    if (hr == S_OK) {
        pDest->SetPreroll(TRUE);
    }
    else if (hr == S_FALSE) {
        pDest->SetPreroll(FALSE);
    }
    else {  // un error inesperado ocurrió...
        return E_UNEXPECTED;
    }
```

```
    // Copia la propiedad Discontinuity

    hr = pSource->IsDiscontinuity();
    if (hr == S_OK) {
          pDest->SetDiscontinuity(TRUE);
    }
    else if (hr == S_FALSE) {
        pDest->SetDiscontinuity(FALSE);
    }
    else {  // un error inesperado ocurrió...
        return E_UNEXPECTED;
    }

    // Copia la longitud de datos actual

    long lDataLength = pSource->GetActualDataLength();
    pDest->SetActualDataLength(lDataLength);

    return NOERROR;

} // Copy


//
// Transformación (in place)
//
//  Se aplica el efecto a la imagen en el modo 'in place' dado que la muestra
//        de salida no es mayor que la de entrada
HRESULT CEZrgb24::Transform(IMediaSample *pMediaSample) //Objeto COM Media Sample
{
    AM_MEDIA_TYPE* pType = &m_pInput->CurrentMediaType(); // Esctructura que contiene el
bloque de formato
    VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) pType->pbFormat; //Estructura con informacion
sobre la imagen y por consiguiente razon de datos permisible

    BYTE *pData;                // Char sin signo, apunta al buffer de imagen actual
    long lDataLen;              // Contiene la longitud de una muestra dada
    unsigned int grey,grey2;    // Se usan para efectuar efectos de gris
    int iPixel, jPixel;         // Utilizado para circular por los pixeles de la imagen
    int temp,x,y;               // Contadores de ciclos para transformaciones
    RGBTRIPLE *prgb;            // Mantiene un apuntador al pixel actual la estructura
contiene los componentes RGB

    pMediaSample->GetPointer(&pData);
    lDataLen = pMediaSample->GetSize();

    // Obtiene las propiedades de la imagen usando BITMAPINFOHEADER
    int cxImage    = pvi->bmiHeader.biWidth;      // Ancho de la imagen
    int cyImage    = pvi->bmiHeader.biHeight;     // Altura de la imagen
    int numPixels  = cxImage * cyImage;
          int i = 0;

    // int iPixelSize = pvi->bmiHeader.biBitCount / 8;
    // int cbImage    = cyImage * cxImage * iPixelSize;

          // Incluyo la imagen en un arreglo de estructuras
          RGBQUAD bmiColors[61][99]; // Es importante no exceder eswte limite de matriz o se
congela

          // Para cargar la imagen .bmp
          HBITMAP hbitmap = (HBITMAP)LoadImage(NULL, "C:\\Mis documentos\\UAM6.bmp",
IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);

          // Crea un dc de memoria
          HDC hMemDC = ::CreateCompatibleDC(NULL);
          //Selecciona el mapa de bits en el dc de memoria
          SelectObject(hMemDC, hbitmap);

          for(iPixel = 0; iPixel < 61; iPixel++)
                    for(jPixel = 0; jPixel < 99; jPixel++){
```

```
                                COLORREF colores = GetPixel(hMemDC, jPixel, iPixel);
                                bmiColors[iPixel][jPixel].rgbBlue = GetBValue((DWORD)
colores);
                                bmiColors[iPixel][jPixel].rgbGreen = GetGValue((DWORD)
colores);
                                bmiColors[iPixel][jPixel].rgbRed = GetRValue((DWORD) colores);
                        }

                  //Borra el dc de memoria y el mapa de bits
                  ::DeleteDC(hMemDC);
                  ::DeleteObject(hbitmap);
                  ::DeleteDC(hMemDC); //Curioso, pero su repeticion evita que la matriz bmicolors se
vaya a blanco, ver pp251

            switch (m_effect)
            {
                case IDC_NONE: break;

                            // Se eliminan las otras dos componentes de color para dejar solo
                            // la roja, la verde o la azul

                case IDC_RED:
                    prgb = (RGBTRIPLE*) pData;
                    for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
                                                prgb->rgbtGreen = 0;
                                                prgb->rgbtBlue = 0;
                    }

                    break;

                case IDC_GREEN:
                    prgb = (RGBTRIPLE*) pData;
                    for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
                                                if(!(rand() % 3)) {
                                                          prgb->rgbtGreen = 255;
                                                          prgb->rgbtBlue = 255;
                                                          prgb->rgbtRed = 255;
                                                }
                    }

                    break;

                case IDC_BLUE:
                    prgb = (RGBTRIPLE*) pData;

                                //offset de la imagen superpuesta
                                for(i = 0; i < (int)((cyImage/25)*(1 + cxImage)); i++,
prgb++);

                                for(iPixel = 0; iPixel < 60; iPixel++) {
                                          for(jPixel = 0; jPixel < 98; jPixel++, prgb++){
                                                    if(bmiColors[iPixel][jPixel].rgbGreen !=
255) {
                                                              prgb->rgbtRed   =
bmiColors[iPixel][jPixel].rgbRed;
                                                              prgb->rgbtGreen =
bmiColors[iPixel][jPixel].rgbGreen;
                                                              prgb->rgbtBlue  =
bmiColors[iPixel][jPixel].rgbBlue;
                                                    }
                                          }
                                          for(jPixel = 98; jPixel < cxImage; jPixel++,
prgb++);
                                }

                    break;

                // Desplaza cada componente a la derecha 1 posicion
                // esto resulta en una imagen mas obscura

                  case IDC_DARKEN:
```

```
            prgb = (RGBTRIPLE*) pData;

                                //offset de la imagen superpuesta
                                for(i = 0; i < (int)((cyImage/25)*(1 + cxImage)); i++,
prgb++);

                                for(iPixel = 0; iPixel < 60; iPixel++) {
                                          for(jPixel = 0; jPixel < 98; jPixel++, prgb++){
                                                    if(bmiColors[iPixel][jPixel].rgbGreen !=
255) {
                                                              prgb->rgbtRed   = (prgb->rgbtRed
+ bmiColors[iPixel][jPixel].rgbRed)/2;
                                                              prgb->rgbtGreen = (prgb-
>rgbtGreen + bmiColors[iPixel][jPixel].rgbGreen)/2;
                                                              prgb->rgbtBlue  = (prgb-
>rgbtBlue + bmiColors[iPixel][jPixel].rgbBlue)/2;
                                                    }
                                          }
                                          for(jPixel = 98; jPixel < cxImage; jPixel++,
prgb++);
                                }

            break;
/*    PRUEBAS PARA QUITAR EL LOGO
                                //offset del Logotipo MTV
            prgb = (RGBTRIPLE*) pData;

                                for(i = 0; i < (cyImage-43) * cxImage + 303; i++, prgb++);

                                for(iPixel = 0; iPixel < 29; iPixel++) {
                                          for(jPixel = 0; jPixel < 34; jPixel++, prgb++){
                                                    //if(bmiColors[iPixel][jPixel].rgbGreen !=
255) {
                                                              prgb->rgbtRed   = prgb->rgbtRed
* 2 - 255;
                                                              prgb->rgbtGreen = prgb-
>rgbtGreen * 2 - 255;
                                                              prgb->rgbtBlue  = prgb->rgbtBlue
* 2 - 255;
                                                    //}
                                          }
                                          for(jPixel = 34; jPixel < cxImage; jPixel++,
prgb++);
                                }
*/
        // Cambia ceros por unos y viceversa produciendo un negativo

        case IDC_XOR:
            prgb = (RGBTRIPLE*) pData;
            for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
                prgb->rgbtRed   = (BYTE) (~prgb->rgbtRed);
                prgb->rgbtGreen = (BYTE) (~prgb->rgbtGreen);
                prgb->rgbtBlue  = (BYTE) (~prgb->rgbtBlue);
            }
            break;

        // Se hacen cero los 5 bits menos significativos para cada componente

        case IDC_POSTERIZE:
            prgb = (RGBTRIPLE*) pData;
            for (iPixel=0; iPixel < numPixels; iPixel++, prgb++) {
                prgb->rgbtRed   = (BYTE) (prgb->rgbtRed & 0xe0);
                prgb->rgbtGreen = (BYTE) (prgb->rgbtGreen & 0xe0);
                prgb->rgbtBlue  = (BYTE) (prgb->rgbtBlue & 0xe0);
            }
            break;

                    // Toma un pixel y el siguiente mas cercano a dos pixeles a su derecha
                    // Los promedia y lo sobreescribe.
        case IDC_BLUR:
            prgb = (RGBTRIPLE*) pData;
```

```
            for (y = 0 ; y < pvi->bmiHeader.biHeight; y++) {
                for (x = 2 ; x < pvi->bmiHeader.biWidth; x++,prgb++) {
                    prgb->rgbtRed   = (BYTE) ((prgb->rgbtRed + prgb[2].rgbtRed) >> 1);
                    prgb->rgbtGreen = (BYTE) ((prgb->rgbtGreen + prgb[2].rgbtGreen) >> 1);
                    prgb->rgbtBlue  = (BYTE) ((prgb->rgbtBlue + prgb[2].rgbtBlue) >> 1);
                                    }
                prgb +=2;
            }
            break;
                    // Un calculo de la escala de grises es:
        //      grey = (30 * red + 59 * green + 11 * blue) / 100
        // Un calculo mas rapido es el siguiente:
        //      grey = (red + green) / 2

        case IDC_GREY:
            prgb = (RGBTRIPLE*) pData;
            for (iPixel=0; iPixel < numPixels ; iPixel++, prgb++) {
                grey = (prgb->rgbtRed + prgb->rgbtGreen) >> 1;
                  prgb->rgbtRed = prgb->rgbtGreen = prgb->rgbtBlue = (BYTE) grey;
            }
            break;

                            // Comparamos los valores de escala de grises de los dos mas
cercanos.
                    // Si no son diferentes, entonces un gris intermedio (128, 128, 128)
                    // se sobrepone. Grandes diferencias se alejan mas de la escala de gris
medio.
        case IDC_EMBOSS:
            prgb = (RGBTRIPLE*) pData;
            for (y = 0 ; y < pvi->bmiHeader.biHeight; y++) {
                grey2 = (prgb->rgbtRed + prgb->rgbtGreen) >> 1;
                prgb->rgbtRed = prgb->rgbtGreen = prgb->rgbtBlue = (BYTE) 128;
                prgb++;
                                    for (x = 1 ; x < pvi->bmiHeader.biWidth; x++) {
                    grey = (prgb->rgbtRed + prgb->rgbtGreen) >> 1;
                    temp = grey - grey2;
                    if (temp > 127) temp = 127;
                    if (temp < -127) temp = -127;
                    temp += 128;
                    prgb->rgbtRed = prgb->rgbtGreen = prgb->rgbtBlue = (BYTE) temp;
                    grey2 = grey;
                    prgb++;
                                    }
            }
            break;
    }
    return NOERROR;

} // Transform (in place)


// Checa que el tipo input sea OK, retorna un error en caso contrario

HRESULT CEZrgb24::CheckInputType(const CMediaType *mtIn)
{
          // checa que sea un tipo VIDEOINFOHEADER

    if (*mtIn->FormatType() != FORMAT_VideoInfo) {
        return E_INVALIDARG;
    }

    // Se puede transformar este tipo?

    if (CanPerformEZrgb24(mtIn)) {
          return NOERROR;
    }
    return E_FAIL;
}


//
```

```
// Checktransform
//
// Checa que una transformacion pueda llevarse a cabo entre estos formatos
//
HRESULT CEZrgb24::CheckTransform(const CMediaType *mtIn, const CMediaType *mtOut)
{
    if (CanPerformEZrgb24(mtIn)) {
        if (*mtIn == *mtOut) {
            return NOERROR;
        }
    }
    return E_FAIL;

} // CheckTransform


//
// DecideBufferSize
//
// Le dice al localizador del pin output que tamaño de buffer requerimos.
// Podemos hacer esto solo cuando el input esta conectado
//
HRESULT CEZrgb24::DecideBufferSize(IMemAllocator *pAlloc,ALLOCATOR_PROPERTIES *pProperties)
{
    // Esta conectado el input pin?

    if (m_pInput->IsConnected() == FALSE) {
        return E_UNEXPECTED;
    }

    ASSERT(pAlloc);
    ASSERT(pProperties);
    HRESULT hr = NOERROR;

    pProperties->cBuffers = 1;
    pProperties->cbBuffer = m_pInput->CurrentMediaType().GetSampleSize();
    ASSERT(pProperties->cbBuffer);

          // Pregunta al locaclizador para reservarnos alguna memoria para la muestra.

    ALLOCATOR_PROPERTIES Actual;
    hr = pAlloc->SetProperties(pProperties,&Actual);
    if (FAILED(hr)) {
        return hr;
    }

    ASSERT( Actual.cBuffers == 1 );

    if (pProperties->cBuffers > Actual.cBuffers ||
            pProperties->cbBuffer > Actual.cbBuffer) {
                return E_FAIL;
    }
    return NOERROR;

} // DecideBufferSize


//
// GetMediaType
//
// Soporto un tipo, normalemente el tipo del input pin
// Este tipo solo esta disponible si mi input esta conectado
//
HRESULT CEZrgb24::GetMediaType(int iPosition, CMediaType *pMediaType)
{
    // Esta conectado el input pin?

    if (m_pInput->IsConnected() == FALSE) {
        return E_UNEXPECTED;
    }
```

```
    // Esto nunca debe suceder

    if (iPosition < 0) {
        return E_INVALIDARG;
    }

    // Tenemos mas items para ofertar?

    if (iPosition > 0) {
        return VFW_S_NO_MORE_ITEMS;
    }

    *pMediaType = m_pInput->CurrentMediaType();
    return NOERROR;

} // GetMediaType


//
// CanPerformEZrgb24
//
// Checa si este es un formato de color RGB24 valido
//
BOOL CEZrgb24::CanPerformEZrgb24(const CMediaType *pMediaType) const
{
    if (IsEqualGUID(*pMediaType->Type(), MEDIATYPE_Video)) {
        if (IsEqualGUID(*pMediaType->Subtype(), MEDIASUBTYPE_RGB24)) {
            VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) pMediaType->Format();
            return (pvi->bmiHeader.biBitCount == 24);
        }
    }
    return FALSE;

} // CanPerformEZrgb24


#define WRITEOUT(var)  hr = pStream->Write(&var, sizeof(var), NULL); \
                           if (FAILED(hr)) return hr;

#define READIN(var)    hr = pStream->Read(&var, sizeof(var), NULL); \
                           if (FAILED(hr)) return hr;


//
// GetClassID
//
// Este es el unico metodo de IPersist
//
STDMETHODIMP CEZrgb24::GetClassID(CLSID *pClsid)
{
    return CBaseFilter::GetClassID(pClsid);
} // GetClassID


//
// ScribbleToStream
//
// Imposibilitado para escribir nuestro estado hacia un stream
//
HRESULT CEZrgb24::ScribbleToStream(IStream *pStream)
{
    HRESULT hr;
    WRITEOUT(m_effect);
    WRITEOUT(m_effectStartTime);
    WRITEOUT(m_effectTime);
    return NOERROR;

} // ScribbleToStream


//
```

```
// ReadFromStream
//
// Imposibilitado para restaurar nuestro estado desde un stream
//
HRESULT CEZrgb24::ReadFromStream(IStream *pStream)
{
    HRESULT hr;
    READIN(m_effect);
    READIN(m_effectStartTime);
    READIN(m_effectTime);
    return NOERROR;

} // ReadFromStream


//
// GetPages
//
// Retorna el clsid de la pagina propietaria que nosotros soportamos
//
STDMETHODIMP CEZrgb24::GetPages(CAUUID *pPages)
{
    pPages->cElems = 1;
    pPages->pElems = (GUID *) CoTaskMemAlloc(sizeof(GUID));
    if (pPages->pElems == NULL) {
        return E_OUTOFMEMORY;
    }
    *(pPages->pElems) = CLSID_EZrgb24PropertyPage;
    return NOERROR;

} // GetPages


//
// get_IPEffect
//
// Retorna el efecto actual seleccionado
//
STDMETHODIMP CEZrgb24::get_IPEffect(int *IPEffect,REFTIME *start,REFTIME *length)
{
    CAutoLock cAutolock(&m_EZrgb24Lock);
    CheckPointer(IPEffect,E_POINTER);
    CheckPointer(start,E_POINTER);
    CheckPointer(length,E_POINTER);

    *IPEffect = m_effect;
    *start = COARefTime(m_effectStartTime);
    *length = COARefTime(m_effectTime);

    return NOERROR;

} // get_IPEffect


//
// put_IPEffect
//
// Selecciona el efecto de video requerido
//
STDMETHODIMP CEZrgb24::put_IPEffect(int IPEffect,REFTIME start,REFTIME length)
{
    CAutoLock cAutolock(&m_EZrgb24Lock);

    m_effect = IPEffect;
    m_effectStartTime = COARefTime(start);
    m_effectTime = COARefTime(length);

    SetDirty(TRUE);
    return NOERROR;

} // put_IPEffect
```

```
//------------------------------------------------------------------------------
// File: EZProp.cpp
//
// Desc: DirectShow - implementación de la clase CEZRgb24Properties
//
//------------------------------------------------------------------------------


#include <windows.h>
#include <windowsx.h>
#include <streams.h>
#include <commctrl.h>
#include <olectl.h>
#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
#include <tchar.h>
#include "resource.h"
#include "EZuids.h"
#include "iEZ.h"
#include "EZrgb24.h"
#include "EZprop.h"


//
// CreateInstance
//
// Usada por las clases base de DirectShow para crear instancias
//
CUnknown *CEZrgb24Properties::CreateInstance(LPUNKNOWN lpunk, HRESULT *phr)
{
    CUnknown *punk = new CEZrgb24Properties(lpunk, phr);
    if (punk == NULL) {
          *phr = E_OUTOFMEMORY;
    }
    return punk;

} // CreateInstance


//
// Constructor
//
CEZrgb24Properties::CEZrgb24Properties(LPUNKNOWN pUnk, HRESULT *phr) :
    CBasePropertyPage(NAME("Special Effects Property Page"),
                      pUnk,IDD_EZrgb24PROP,IDS_TITLE),
    m_pIPEffect(NULL),
    m_bIsInitialized(FALSE)
{
    ASSERT(phr);

} // (Constructor)


//
// OnReceiveMessage
//
// Maneja los mensajes para nuestra ventana propietaria
//
BOOL CEZrgb24Properties::OnReceiveMessage(HWND hwnd,
                                          UINT uMsg,
                                          WPARAM wParam,
                                          LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_COMMAND:
        {
            if (m_bIsInitialized)
            {
                m_bDirty = TRUE;
```

```
                if (m_pPageSite)
                {
                    m_pPageSite->OnStatusChange(PROPPAGESTATUS_DIRTY);
                }
            }
            return (LRESULT) 1;
        }

    }
    return CBasePropertyPage::OnReceiveMessage(hwnd,uMsg,wParam,lParam);

} // OnReceiveMessage


//
// OnConnect
//
// Se llama cuando nos conectamos  a un filtro transform
//
HRESULT CEZrgb24Properties::OnConnect(IUnknown *pUnknown)
{
    ASSERT(m_pIPEffect == NULL);

    HRESULT hr = pUnknown->QueryInterface(IID_IIPEffect, (void **) &m_pIPEffect);
    if (FAILED(hr)) {
        return E_NOINTERFACE;
    }

    ASSERT(m_pIPEffect);

          // Obtiene el efecto inicial
    m_pIPEffect->get_IPEffect(&m_effect, &m_start, &m_length);
    m_bIsInitialized = FALSE ;
    return NOERROR;

} // OnConnect


//
// OnDisconnect
//
// Se llama cuando nos desconectamos de un filtro
//
HRESULT CEZrgb24Properties::OnDisconnect()
{
          // Actualizacion de la interfaz despues de ajustar los valores de los antiguos
efectos

    if (m_pIPEffect == NULL) {
        return E_UNEXPECTED;
    }

    m_pIPEffect->Release();
    m_pIPEffect = NULL;
    return NOERROR;

} // OnDisconnect


//
// OnActivate
//
// Cuando seamos activados
//
HRESULT CEZrgb24Properties::OnActivate()
{
    TCHAR   sz[60];

    _stprintf(sz, TEXT("%f"), m_length);
    Edit_SetText(GetDlgItem(m_Dlg, IDC_LENGTH), sz);
    _stprintf(sz, TEXT("%f"), m_start);
```

```
    Edit_SetText(GetDlgItem(m_Dlg, IDC_START), sz);

    CheckRadioButton(m_Dlg, IDC_EMBOSS, IDC_NONE, m_effect);
    m_bIsInitialized = TRUE;
    return NOERROR;

} // OnActivate


//
// OnDeactivate
//
// Cuando seamos desactivados
//
HRESULT CEZrgb24Properties::OnDeactivate(void)
{
    ASSERT(m_pIPEffect);
    m_bIsInitialized = FALSE;
    GetControlValues();
    return NOERROR;

} // OnDeactivate


//
// OnApplyChanges
//
// Aplica cualquier cambio que sea realizado
//
HRESULT CEZrgb24Properties::OnApplyChanges()
{
    GetControlValues();
    m_pIPEffect->put_IPEffect(m_effect, m_start, m_length);

    return NOERROR;
} // OnApplyChanges


void CEZrgb24Properties::GetControlValues()
{
    ASSERT(m_pIPEffect);
    TCHAR sz[STR_MAX_LENGTH];
    REFTIME tmp1, tmp2 ;

    // Obtiene el inicio y final del efecto

    Edit_GetText(GetDlgItem(m_Dlg, IDC_LENGTH), sz, STR_MAX_LENGTH);

#ifdef UNICODE
    int rc;

          // Convierte cadenas multibyte a ANSI
    char szANSI[STR_MAX_LENGTH];
    rc = WideCharToMultiByte(CP_ACP, 0, sz, -1, szANSI, STR_MAX_LENGTH, NULL, NULL);
    tmp2 = COARefTime(atof(szANSI));
#else
    tmp2 = COARefTime(atof(sz));
#endif

    Edit_GetText(GetDlgItem(m_Dlg, IDC_START), sz, STR_MAX_LENGTH);

#ifdef UNICODE

          // Convierte cadenas multibyte a ANSI
    rc = WideCharToMultiByte(CP_ACP, 0, sz, -1, szANSI, STR_MAX_LENGTH, NULL, NULL);
    tmp1 = COARefTime(atof(szANSI));
#else
    tmp1 = COARefTime(atof(sz));
#endif

    // Validacion rapida de los campos
```

```
    if (tmp1 >= 0 && tmp2 >= 0) {
        m_start = tmp1;
        m_length = tmp2;
    }

    // Encuentra cual efecto especial hemos seleccionado

    for (int i = IDC_EMBOSS; i <= IDC_NONE; i++) {
        if (IsDlgButtonChecked(m_Dlg, i)) {
            m_effect = i;
            break;
        }
    }
}

//------------------------------------------------------------------------------
// File: EZProp.h
//
// Desc: DirectShow sample code - definition of CEZrgb24Properties class.
//
// Copyright (c) 1992-2001 Microsoft Corporation.  All rights reserved.
//------------------------------------------------------------------------------


class CEZrgb24Properties : public CBasePropertyPage
{

public:

    static CUnknown * WINAPI CreateInstance(LPUNKNOWN lpunk, HRESULT *phr);

private:

    BOOL OnReceiveMessage(HWND hwnd,UINT uMsg,WPARAM wParam,LPARAM lParam);
    HRESULT OnConnect(IUnknown *pUnknown);
    HRESULT OnDisconnect();
    HRESULT OnActivate();
    HRESULT OnDeactivate();
    HRESULT OnApplyChanges();

    void    GetControlValues();

    CEZrgb24Properties(LPUNKNOWN lpunk, HRESULT *phr);

    BOOL m_bIsInitialized;      // Used to ignore startup messages
    int m_effect;               // Which effect are we processing
    REFTIME m_start;            // When the effect will begin
    REFTIME m_length;           // And how long it will last for
    IIPEffect *m_pIPEffect;     // The custom interface on the filter

}; // EZrgb24Properties


//------------------------------------------------------------------------------
// File: EZUIDs.h
//
// Desc: DirectShow sample code - special effects filter CLSIDs.
//
// Copyright (c) 1992-2001 Microsoft Corporation.  All rights reserved.
//------------------------------------------------------------------------------


// Special effects filter CLSID

// { 8B498501-1218-11cf-ADC4-00A0D100041B }
DEFINE_GUID(CLSID_EZrgb24,
0x8b498501, 0x1218, 0x11cf, 0xad, 0xc4, 0x0, 0xa0, 0xd1, 0x0, 0x4, 0x1b);

// And the property page we support
```

```
// { 8B498502-1218-11cf-ADC4-00A0D100041B }
DEFINE_GUID(CLSID_EZrgb24PropertyPage,
0x8b498502, 0x1218, 0x11cf, 0xad, 0xc4, 0x0, 0xa0, 0xd1, 0x0, 0x4, 0x1b);



//------------------------------------------------------------------------------
// File: EZRGB24.h
//
// Desc: DirectShow sample code - special effects filter header file.
//
// Copyright (c) 1992-2001 Microsoft Corporation.  All rights reserved.
//------------------------------------------------------------------------------


class CEZrgb24 : public CTransformFilter,
                   public IIPEffect,
                   public ISpecifyPropertyPages,
                   public CPersistStream
{

public:

    DECLARE_IUNKNOWN;
    static CUnknown * WINAPI CreateInstance(LPUNKNOWN punk, HRESULT *phr);

    // Reveals IEZrgb24 and ISpecifyPropertyPages
    STDMETHODIMP NonDelegatingQueryInterface(REFIID riid, void ** ppv);

    // CPersistStream stuff
    HRESULT ScribbleToStream(IStream *pStream);
    HRESULT ReadFromStream(IStream *pStream);

    // Overrriden from CTransformFilter base class

    HRESULT Transform(IMediaSample *pIn, IMediaSample *pOut);
    HRESULT CheckInputType(const CMediaType *mtIn);
    HRESULT CheckTransform(const CMediaType *mtIn, const CMediaType *mtOut);
    HRESULT DecideBufferSize(IMemAllocator *pAlloc,
                                ALLOCATOR_PROPERTIES *pProperties);
    HRESULT GetMediaType(int iPosition, CMediaType *pMediaType);

    // These implement the custom IIPEffect interface

    STDMETHODIMP get_IPEffect(int *IPEffect, REFTIME *StartTime, REFTIME *Length);
    STDMETHODIMP put_IPEffect(int IPEffect, REFTIME StartTime, REFTIME Length);

    // ISpecifyPropertyPages interface
    STDMETHODIMP GetPages(CAUUID *pPages);

    // CPersistStream override
    STDMETHODIMP GetClassID(CLSID *pClsid);

private:

    // Constructor
    CEZrgb24(TCHAR *tszName, LPUNKNOWN punk, HRESULT *phr);

    // Look after doing the special effect
    BOOL CanPerformEZrgb24(const CMediaType *pMediaType) const;
    HRESULT Copy(IMediaSample *pSource, IMediaSample *pDest) const;
    HRESULT Transform(IMediaSample *pMediaSample);

    CCritSec        m_EZrgb24Lock;          // Private play critical section
    int         m_effect;               // Which effect are we processing
    CRefTime        m_effectStartTime;      // When the effect will begin
    CRefTime        m_effectTime;           // And how long it will last for
    const long m_lBufferRequest;        // The number of buffers to use

}; // EZrgb2
```

Código fuente de la aplicación

```
//------------------------------------------------------------------------------
// Archivo: PlayWnd.cpp
//
// Descripcion: Reproductor de audio y video que utiliza las librerias DirectShow
//       Pausa, detener, silencio, y cambio apantalla completa pueden llevarse
//       a cabo mediante la barra de menus o teclas.
//
//------------------------------------------------------------------------------


#include <dshow.h>
#include <commctrl.h>
#include <commdlg.h>
#include <stdio.h>
#include <tchar.h>

#include <initguid.h> //Es importante incluir antes de las definir los GUIDS en "ezuids.h"
                                                    // De lo contrario marca el error "error
LNK2001: unresolved external symbol"

#include "ezuids.h" // Lo incluí para que reconozca el filtro ezrgb


#include "playwnd.h"

//
// Para permitir el registro de este grafico de filtros se define REGISTER_FILTERGRAPH.
//
#define REGISTER_FILTERGRAPH

//
// Variables globales
//
HWND      ghApp=0;
HINSTANCE ghInst;
TCHAR     g_szFileName[MAX_PATH]={0};
BOOL      g_bAudioOnly=FALSE;
LONG      g_lVolume=VOLUME_FULL;
DWORD     g_dwGraphRegister=0;
PLAYSTATE g_psCurrent=Stopped;

// OJO CLSID CLSID_EZrgb24;

// Interfaces DirectShow
IGraphBuilder *pGB = NULL;
IMediaControl *pMC = NULL;
IMediaEventEx *pME = NULL;
IVideoWindow  *pVW = NULL;
IBasicAudio   *pBA = NULL;
IBasicVideo   *pBV = NULL;
IMediaSeeking *pMS = NULL;
IBaseFilter           *pFX = NULL; // Interfaz para el manejo del nuevo filtro F1
IPin                  *pOutPin = NULL;  //Interfaz del Pin de salida
IPin                  *pInputPin = NULL; //Interfaz del pin de entrada
ISpecifyPropertyPages *pProp; // Para exponer la pagina propietaria
ICaptureGraphBuilder2 *pBuilder = NULL; //Para la TV



// Función que inicia la reproducción del archivo multimedia
HRESULT PlayMovieInWindow(LPTSTR szFile)
{
    WCHAR wFile[MAX_PATH];
    HRESULT hr;
          IFilterGraph *pGraph = NULL;
          //CComPtr<ICreateDevEnum> pSysDevEnum; // Se añadió al ultimo

          pGraph = pGB;
```

```
    // Se limpian los remanentes de dialogo abiertos antes de llamar a RenderFile()
    UpdateWindow(ghApp);

#ifndef UNICODE
    MultiByteToWideChar(CP_ACP, 0, szFile, -1, wFile, MAX_PATH);
#else
    lstrcpy(wFile, szFile);
#endif

          // Se obtiene la interfaz para el constructor del grafico de filtros GraphBuilder
    JIF(CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER, IID_IGraphBuilder,
(void **)&pGB));

    if(SUCCEEDED(hr))
    {
        // Se construye automaticamente el grafico de filtros que ejecutara el archivo
especificado
        JIF(pGB->RenderFile(wFile, NULL));

                    // Desconectamos los ultimos dos filtros del grafico
                    EnumFilters("MPEG Video Decoder", PINDIR_OUTPUT);  //Busca el output pin
del penultimo filtro...
                    hr = pGB->Disconnect(pOutPin); //OK :-)

                    EnumFilters("Video Renderer", PINDIR_INPUT);  //Busca el input pin del
ultimo filtro...
                    hr = pGB->Disconnect(pInputPin); //OK :-)

                    // pSysDevEnum->CreateClassEnumerator(CLSID_EZrgb24, &pEnumCat, 0);

                    // Añadimos el filtro de efectos en la imagen
                    hr = CoCreateInstance(CLSID_EZrgb24, NULL, CLSCTX_INPROC_SERVER,
            IID_IBaseFilter, reinterpret_cast<void**>(&pFX));
                    hr = pGB->AddFilter(pFX, L"Image Effects"); //:-) OK

                    //Conectamos filtro de efectos y MPEG video decoder
                    EnumFilters("Image Effects", PINDIR_INPUT); // Busca input pin de filtro
Efectos
                    hr = pGB->Connect(pOutPin, pInputPin);

                    //Raconexion automatica entre Image Effects y video render
                    EnumFilters("Image Effects", PINDIR_OUTPUT);
                    EnumFilters("Video Renderer", PINDIR_INPUT);
                    hr = pGB->Connect(pOutPin, pInputPin);

        // QueryInterface para las interfaces DirectShow
        JIF(pGB->QueryInterface(IID_IMediaControl, (void **)&pMC));
        JIF(pGB->QueryInterface(IID_IMediaEventEx, (void **)&pME));
        JIF(pGB->QueryInterface(IID_IMediaSeeking, (void **)&pMS));

                    // Consulta las interfaces de video, las cuales pueden no ser relevantes
para archivos de audio
        JIF(pGB->QueryInterface(IID_IVideoWindow, (void **)&pVW));
        JIF(pGB->QueryInterface(IID_IBasicVideo, (void **)&pBV));

                    // Consulta por interfaces de audio, las cuales no pueden ser relevantes
para archivos solo de video
        JIF(pGB->QueryInterface(IID_IBasicAudio, (void **)&pBA));

                    // Es este un archivo solo de audio (no hay componente de video)?
        CheckVisibility();

        if (!g_bAudioOnly)
        {
            JIF(pVW->put_Owner((OAHWND)ghApp));
            JIF(pVW->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN));
        }

                    // Tiene grafico eventos via llamadas a la ventana por llevar a cabo
        JIF(pME->SetNotifyWindow((OAHWND)ghApp, WM_GRAPHNOTIFY, 0));
```

```
        if (g_bAudioOnly)
        {
            JIF(InitPlayerWindow());
        }
        else
        {
            JIF(InitVideoWindow(1, 1));
        }

        // Aqui viene lo bueno se vera?!
        ShowWindow(ghApp, SW_SHOWNORMAL);
        UpdateWindow(ghApp);
        SetForegroundWindow(ghApp);
        SetFocus(ghApp);

        UpdateMainTitle();

#ifdef REGISTER_FILTERGRAPH
        hr = AddGraphToRot(pGB, &g_dwGraphRegister);
        if (FAILED(hr))
        {
            Msg(TEXT("Falló al registrar el grafico de filtros con ROT!  hr=0x%x"), hr);
            g_dwGraphRegister = 0;
        }
#endif

                    // Corre el grafico de filtros que ejecutará el archivo multimedia
        JIF(pMC->Run());
        g_psCurrent=Running;

        SetFocus(ghApp);
    }

    return hr;
}

// Función que inicia la ventana de video con un factor determinado
HRESULT InitVideoWindow(int nMultiplier, int nDivider)
{
    LONG lHeight, lWidth;
    HRESULT hr = S_OK;
    RECT rect;

    if (!pBV)
        return S_OK;

          // Lee el tamaño de video actual
    JIF(pBV->GetVideoSize(&lWidth, &lHeight));

          // Actualiza al tamaño requerido, normal, mitad ...
    lWidth  = lWidth  * nMultiplier / nDivider;
    lHeight = lHeight * nMultiplier / nDivider;

    SetWindowPos(ghApp, NULL, 0, 0, lWidth, lHeight,
                 SWP_NOMOVE | SWP_NOOWNERZORDER);

    int nTitleHeight  = GetSystemMetrics(SM_CYCAPTION);
    int nBorderWidth  = GetSystemMetrics(SM_CXBORDER);
    int nBorderHeight = GetSystemMetrics(SM_CYBORDER);

          // Actualiza dependiendo del tamaño de la barra de titulo y bordes para un
          // acoplamiento exacto de la ventana de video a el area del cliente en la ventana
    SetWindowPos(ghApp, NULL, 0, 0, lWidth + 2*nBorderWidth,
            lHeight + nTitleHeight + 2*nBorderHeight,
            SWP_NOMOVE | SWP_NOOWNERZORDER);

    GetClientRect(ghApp, &rect);
    JIF(pVW->SetWindowPosition(rect.left, rect.top, rect.right, rect.bottom));

    return hr;
```

```
}

// Función que inicializa la ventana de despliegue en un tamaño por default
HRESULT InitPlayerWindow(void)
{
          // Restaura a un tamaño default para audio y despues de cerrar un video
    SetWindowPos(ghApp, NULL, 0, 0,
                 DEFAULT_AUDIO_WIDTH,
                 DEFAULT_AUDIO_HEIGHT,
                 SWP_NOMOVE | SWP_NOOWNERZORDER);

    return S_OK;
}

// Función  que hace un reajuste de la ventana en caso de mover o reajustar tamaño de la
ventana principal
void MoveVideoWindow(void)
{
    HRESULT hr;

    // Sigue el movimiento del contenedor de video y reajusta el tamaño si es necesario
    if(pVW)
    {
        RECT client;

        GetClientRect(ghApp, &client);
        hr = pVW->SetWindowPosition(client.left, client.top,
                                    client.right, client.bottom);
    }
}

// Revisa si el archivo tiene componente de video o no
void CheckVisibility(void)
{
    long lVisible;
    HRESULT hr;

    g_bAudioOnly = FALSE;

    if ((!pVW) || (!pBV))
    {
        g_bAudioOnly = TRUE;
        return;
    }

    hr = pVW->get_Visible(&lVisible);
    if (FAILED(hr))
    {
                    // Si este es un clip solo de audio, get_Visible no trabajará
        //
                    // Tambien si este video esta codificado con un codec no soportado,
                    // no veremos ningun video, en contraparte el audio si trabajara
                    //  si este se encuentra en un formato soportado
        //
        if (hr == E_NOINTERFACE)
        {
            g_bAudioOnly = TRUE;
        }
        else
        {
            Msg(TEXT("Falló(%08lx) in pVW->get_Visible()!\r\n"), hr);
        }
    }
}

// Función que implementa la pausa en el clip multimedia
void PauseClip(void)
{
    HRESULT hr;

    if (!pMC)
```

```
        return;

          // Hace el cambio entre los estados de Pausa/Reproducir
    if((g_psCurrent == Paused) || (g_psCurrent == Stopped))
    {
        hr = pMC->Run();
        g_psCurrent = Running;
    }
    else
    {
        hr = pMC->Pause();
        g_psCurrent = Paused;
    }

    UpdateMainTitle();
}

// Función que detiene el gráfico de filtros
void StopClip(void)
{
    HRESULT hr;

    if ((!pMC) || (!pMS))
        return;

          //  Detiene y restaura la posición al principio
    if((g_psCurrent == Paused) || (g_psCurrent == Running))
    {
        LONGLONG pos = 0;
        hr = pMC->Stop();
        g_psCurrent = Stopped;

        hr = pMS->SetPositions(&pos, AM_SEEKING_AbsolutePositioning ,
            NULL, AM_SEEKING_NoPositioning);

                  // Despliega el primer frame para indicar la condición de reset
        hr = pMC->Pause();
    }

    UpdateMainTitle();
}

void Tele()
{
          HRESULT hr;

    // Se limpian los remanentes de dialogo abiertos antes de la presentacion
    UpdateWindow(ghApp);


          // Crea el FGM
          CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC, IID_IGraphBuilder, (void
**)&pGB);

          // Crea el Capture Graph Builder.
          CoCreateInstance(CLSID_CaptureGraphBuilder2, NULL, CLSCTX_INPROC,
IID_ICaptureGraphBuilder2, (void **)&pBuilder);

          // Asocia el gráfico con el builder
          pBuilder->SetFiltergraph(pGB);

          // Obtiene interfaces para media control y Video Window
          hr = pGB->QueryInterface(IID_IMediaControl,(LPVOID *) &pMC);

          // QueryInterface para las interfaces DirectShow
    pGB->QueryInterface(IID_IMediaEventEx, (void **)&pME);
    pGB->QueryInterface(IID_IMediaSeeking, (void **)&pMS);

          // Consulta las interfaces de video, las cuales pueden no ser relevantes para
archivos de audio
    pGB->QueryInterface(IID_IVideoWindow, (void **)&pVW);
```

```
    pGB->QueryInterface(IID_IBasicVideo, (void **)&pBV);

          // Consulta por interfaces de audio, las cuales no pueden ser relevantes para
archivos solo de video
    pGB->QueryInterface(IID_IBasicAudio, (void **)&pBA);

          // Crea el system device enumerator.
          ICreateDevEnum *pDevEnum = NULL;
          CoCreateInstance(CLSID_SystemDeviceEnum, NULL, CLSCTX_INPROC, IID_ICreateDevEnum,
(void **)&pDevEnum);

          // Crea un enumerador para video capture devices.
          IEnumMoniker *pClassEnum = NULL;
          pDevEnum->CreateClassEnumerator(CLSID_VideoInputDeviceCategory, &pClassEnum, 0);

          ULONG cFetched;
          IMoniker *pMoniker = NULL;
          IBaseFilter *pSrc = NULL;
          if (pClassEnum->Next(1, &pMoniker, &cFetched) == S_OK)
          {
                    // Liga el primer moniker a un objeto filtro.
                    pMoniker->BindToObject(0, 0, IID_IBaseFilter, (void**)&pSrc);
                    pMoniker->Release();
          }
          pClassEnum->Release();
          pDevEnum->Release();

    // Se añade el filtro de captura a nuestro gráfico
    hr = pGB->AddFilter(pSrc, L"Video Capture");

          // Muestra el pin de captura en el filtro de captura de video.
    // Se usa esta instrucción en lugar de pBuilder->RenderFile
    hr = pBuilder->RenderStream (&PIN_CATEGORY_CAPTURE , &MEDIATYPE_Video,
                                 pSrc, NULL, NULL);
          // Ahora que el filtro se ha añadido al gráfico y hemos presentado
          // su stream, podemos actualizar esta referencia al filtro.
    pSrc->Release();

          // Desconectamos los ultimos dos filtros del grafico
          EnumFilters("Color Space Converter", PINDIR_OUTPUT);  //Busca el output pin del
penultimo filtro...
          hr = pGB->Disconnect(pOutPin);

          EnumFilters("Video Renderer", PINDIR_INPUT);  //Busca el input pin del ultimo
filtro...
          hr = pGB->Disconnect(pInputPin);

          // Añadimos el filtro de efectos en la imagen
          hr = CoCreateInstance(CLSID_EZrgb24, NULL, CLSCTX_INPROC_SERVER, IID_IBaseFilter,
reinterpret_cast<void**>(&pFX));
          hr = pGB->AddFilter(pFX, L"Image Effects");

          //Conectamos filtro de efectos y Color Space converter
          EnumFilters("Image Effects", PINDIR_INPUT); // Busca input pin de filtro Efectos
          hr = pGB->Connect(pOutPin, pInputPin);

          //Reconexion automatica entre Image Effects y video render
          EnumFilters("Image Effects", PINDIR_OUTPUT);
          EnumFilters("Video Renderer", PINDIR_INPUT);
          hr = pGB->Connect(pOutPin, pInputPin);

#ifdef REGISTER_FILTERGRAPH
    hr = AddGraphToRot(pGB, &g_dwGraphRegister);
#endif


        pVW->put_Owner((OAHWND)ghApp);
        pVW->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN);
                    InitVideoWindow(1, 1);
```

```
        // Aqui viene lo bueno se vera?!
        ShowWindow(ghApp, SW_SHOWNORMAL);
        UpdateWindow(ghApp);
        SetForegroundWindow(ghApp);
        SetFocus(ghApp);

        UpdateMainTitle();

    // Inicia la previsualización de los datos
    hr =pMC->Run();

    // Recuerda el estado actual
    g_psCurrent = Running;


}


// Función que abre un archivo multimedia
void OpenClip()
{
    HRESULT hr;

          // Si no hay un nombre de archivo especificado en la linea de comandos, muestra el
dialogo de abrir archivo
    if(g_szFileName[0] == L'\0')
    {
        TCHAR szFilename[MAX_PATH];

        UpdateMainTitle();

                    // Si no hay nombre de archivo especificado, entonces la ventana de
video
                    // no ha sido creado o se ha hecho visible. Hacemos muestra ventana
principal
                    // visible y la llevamos al frente para permitir la seleccion de
archivos
        InitPlayerWindow();
        ShowWindow(ghApp, SW_SHOWNORMAL);
        SetForegroundWindow(ghApp);

        if (! GetClipFileName(szFilename))
        {
            DWORD dwDlgErr = CommDlgExtendedError();

                                // No muestra una salida si el usuario canceló la selección
(no hay dialogo de error)
            if (dwDlgErr)
            {
                Msg(TEXT("¡Falló GetClipFileName! Error=0x%x\r\n"), GetLastError());
            }
            return;
        }

                    // Esta aplicacion no soporta reproduccion de arvhivos ASX
                    // Dado que esto podría confundir al usuario, desplegamos un mensaje
                    // de advertencia si un archivo ASX fué abierto.
        if (_tcsstr((_tcslwr(szFilename)), TEXT(".asx")))
        {
            Msg(TEXT("ASX Playlists are not supported by this application.  Please select a
media file.\0"));
            return;
        }

        lstrcpy(g_szFileName, szFilename);
    }

          // Reinicializa variables de estado
    g_psCurrent = Stopped;
    g_lVolume = VOLUME_FULL;
```

```
    hr = PlayMovieInWindow(g_szFileName);

          // Si no podemos ejecutar el clip, limpiamos.
    if (FAILED(hr))
        CloseClip();
}


BOOL GetClipFileName(LPTSTR szName)
{
    static OPENFILENAME ofn={0};
    static BOOL bSetInitialDir = FALSE;

    *szName = 0;

    ofn.lStructSize       = sizeof(OPENFILENAME);
    ofn.hwndOwner         = ghApp;
    ofn.lpstrFilter       = NULL;
    ofn.lpstrFilter       = FILE_FILTER_TEXT;
    ofn.lpstrCustomFilter = NULL;
    ofn.nFilterIndex      = 1;
    ofn.lpstrFile         = szName;
    ofn.nMaxFile          = MAX_PATH;
    ofn.lpstrTitle        = TEXT("Open Media File...\0");
    ofn.lpstrFileTitle    = NULL;
    ofn.lpstrDefExt       = TEXT("*\0");
    ofn.Flags             = OFN_FILEMUSTEXIST | OFN_READONLY | OFN_PATHMUSTEXIST;

          // Recordamos el path del primer archivo seleccionado
    if (bSetInitialDir == FALSE)
    {
        ofn.lpstrInitialDir = DEFAULT_MEDIA_PATH;
        bSetInitialDir = TRUE;
    }
    else
        ofn.lpstrInitialDir = NULL;

    return GetOpenFileName((LPOPENFILENAME)&ofn);
}

// Función que cierra el clip actual
void CloseClip()
{
    HRESULT hr;

    if(pMC)
        hr = pMC->Stop();

    g_psCurrent = Stopped;
    g_bAudioOnly = TRUE;

    CloseInterfaces();

          // Limpia del arreglo el nombre de archivo para permitir una nueva eleccion
    g_szFileName[0] = L'\0';

          // Borramos el estado actual de ejecucion multimedia
    g_psCurrent = Init;

    RECT rect;
    GetClientRect(ghApp, &rect);
    InvalidateRect(ghApp, &rect, TRUE);

    UpdateMainTitle();
    InitPlayerWindow();
}



//Funcion que busca un filtro en el grafico, y su pin input o output
```

```
void EnumFilters (char *nomFiltro, PIN_DIRECTION PinDir)
{
          IFilterGraph *pGraph = NULL;
    IEnumFilters *pEnum = NULL;
    IBaseFilter *pFilter;
    ULONG cFetched;

          pGraph = pGB;
    pGraph->EnumFilters(&pEnum);
    while(pEnum->Next(1, &pFilter, &cFetched) == S_OK)
    {
        FILTER_INFO FilterInfo;
        char szName[256];

        pFilter->QueryFilterInfo(&FilterInfo);
        WideCharToMultiByte(CP_ACP, 0, FilterInfo.achName, -1, szName, 256, 0, 0);

                    // Si el nombre del filtro se localiza busca el apuntador al pin output
                    if(!strcmp(szName, nomFiltro) && PinDir == PINDIR_OUTPUT)
                              pOutPin = GetPin(pFilter, PinDir);
                    if(!strcmp(szName, nomFiltro) && PinDir == PINDIR_INPUT)
                              pInputPin = GetPin(pFilter, PinDir);

        FilterInfo.pGraph->Release();
        pFilter->Release();
    }
    pEnum->Release();

}

// Funcion que obtiene la direccion de un Pin Output
IPin *GetPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir)
{
    BOOL       bFound = FALSE;
    IEnumPins  *pEnum;
    IPin       *pPin;

    pFilter->EnumPins(&pEnum);
    while(pEnum->Next(1, &pPin, 0) == S_OK)
    {
        PIN_DIRECTION PinDirThis;
        pPin->QueryDirection(&PinDirThis);
        if (bFound = (PinDir == PinDirThis))
            break;
        pPin->Release();
    }
    pEnum->Release();
    return (bFound ? pPin : 0);
}

void CloseInterfacesTV(void)
{
    // DEtiene los datos presentados
    if (pMC)
        pMC->StopWhenReady();

    g_psCurrent = Stopped;

    // Detiene la recepción de eventos
    if (pME)
        pME->SetNotifyWindow(NULL, WM_GRAPHNOTIFY, 0);

    if(pVW)
    {
        pVW->put_Visible(OAFALSE);
        pVW->put_Owner(NULL);
    }

#ifdef REGISTER_FILTERGRAPH
    // Remueve el grafico de filtros de la tabla
    if (g_dwGraphRegister)
```

```
        RemoveGraphFromRot(g_dwGraphRegister);
#endif

    // Actualiza las interfaces DirectShow
    SAFE_RELEASE(pMC);
    SAFE_RELEASE(pME);
    SAFE_RELEASE(pVW);
    SAFE_RELEASE(pGB);
    SAFE_RELEASE(pBuilder);
}


//  Función que cierra las interfaces de DirectShow
void CloseInterfaces(void)
{
    HRESULT hr;

          // Recuperacion de derechos de propietario despues de ocultar la ventana de video
    if(pVW)
    {
        hr = pVW->put_Visible(OAFALSE);
        hr = pVW->put_Owner(NULL);
    }

#ifdef REGISTER_FILTERGRAPH
    if (g_dwGraphRegister)
    {
        RemoveGraphFromRot(g_dwGraphRegister);
        g_dwGraphRegister = 0;
    }
#endif

    SAFE_RELEASE(pME);
    SAFE_RELEASE(pMS);
    SAFE_RELEASE(pMC);
    SAFE_RELEASE(pBA);
    SAFE_RELEASE(pBV);
    SAFE_RELEASE(pVW);
    SAFE_RELEASE(pGB);
}

#ifdef REGISTER_FILTERGRAPH
// Función que añade el grafico de filtros
HRESULT AddGraphToRot(IUnknown *pUnkGraph, DWORD *pdwRegister)
{
    IMoniker * pMoniker;
    IRunningObjectTable *pROT;
    if (FAILED(GetRunningObjectTable(0, &pROT))) {
        return E_FAIL;
    }
    WCHAR wsz[128];
    wsprintfW(wsz, L"FilterGraph %08x pid %08x", (DWORD_PTR)pUnkGraph,
GetCurrentProcessId());
    HRESULT hr = CreateItemMoniker(L"!", wsz, &pMoniker);
    if (SUCCEEDED(hr)) {
        hr = pROT->Register(0, pUnkGraph, pMoniker, pdwRegister);
        pMoniker->Release();
    }
    pROT->Release();
    return hr;
}

// Quita el grafico de filtros
void RemoveGraphFromRot(DWORD pdwRegister)
{
    IRunningObjectTable *pROT;
    if (SUCCEEDED(GetRunningObjectTable(0, &pROT))) {
        pROT->Revoke(pdwRegister);
        pROT->Release();
    }
}
```

```
#endif

// Despliega los nombres de archivo en la caja de mensajes
void Msg(char *szFormat, ...)
{
    TCHAR szBuffer[512];  // Buffer grande para nombres de archivos largos como los HTTTP

    va_list pArgs;
    va_start(pArgs, szFormat);
    vsprintf(szBuffer, szFormat, pArgs);
    va_end(pArgs);

    MessageBox(NULL, szBuffer, "UAM Multimedia", MB_OK);
}

// Función para activar el silencio shhh...
HRESULT ToggleMute(void)
{
    HRESULT hr=S_OK;

    if ((!pGB) || (!pBA))
        return S_OK;

    // Read current volume
    hr = pBA->get_Volume(&g_lVolume);
    if (hr == E_NOTIMPL)
    {
                    // Falla si este es un archivo de solo video
        return S_OK;
    }
    else if (FAILED(hr))
    {
        Msg(TEXT("Fallo al leer el volumen de audio !  hr=0x%x\r\n"), hr);
        return hr;
    }

          // Cambia los niveles de volumen
    if (g_lVolume == VOLUME_FULL)
        g_lVolume = VOLUME_SILENCE;
    else
        g_lVolume = VOLUME_FULL;

          // Ajusta el nuevo volumen
    JIF(pBA->put_Volume(g_lVolume));

    UpdateMainTitle();
    return hr;
}

// Función que actualiza la etiqueta que actualiza el estado actual del streaming
void UpdateMainTitle(void)
{
    TCHAR szTitle[MAX_PATH], szFile[MAX_PATH];

          // Si ningun archivo es cargado, solo muestra el titulo de la aplicacion
    if (g_szFileName[0] == L'\0')
    {
        wsprintf(szTitle, TEXT("%s"), APPLICATIONNAME);
    }

          // De otro modo, muestra información en uso, incluyendo nombre de archivo y estado
de ejecución
    else
    {
                    // Obtiene nombre de archivo sin path completo
        GetFilename(g_szFileName, szFile);

                    // Actualiza el titulo de la ventana para mostrar el estado
Silencio/Sonido
        wsprintf(szTitle, TEXT("%s [%s] %s%s"),
```

```
                szFile,
                g_bAudioOnly ? TEXT("Audio") : TEXT("Video"),
                (g_lVolume == VOLUME_SILENCE) ? TEXT("(En silencio)") : TEXT(""),
                (g_psCurrent == Paused) ? TEXT("(Pausado)") : TEXT(""));
    }

    SetWindowText(ghApp, szTitle);
}

//Funcion que obtiene el nombre del archivo a reproducir
void GetFilename(TCHAR *pszFull, TCHAR *pszFile)
{
    int nLength;
    TCHAR szPath[MAX_PATH]={0};
    BOOL bSetFilename=FALSE;

    _tcscpy(szPath, pszFull);
    nLength = _tcslen(szPath);

    for (int i=nLength-1; i>=0; i--)
    {
        if ((szPath[i] == '\\') || (szPath[i] == '/'))
        {
            szPath[i] = '\0';
            lstrcpy(pszFile, &szPath[i+1]);
            bSetFilename = TRUE;
            break;
        }
    }

          // Si no se proporcionó un Path (solo un nombre de archivo), entonces
          // solo copia el path completo a el path destino
    if (!bSetFilename)
        _tcscpy(pszFile, pszFull);
}

// Función que permite la pantalla completa
HRESULT ToggleFullScreen(void)
{
    HRESULT hr=S_OK;
    LONG lMode;
    static HWND hDrain=0;

          // No permitir pantalla completa en archivos de solo audio
    if ((g_bAudioOnly) || (!pVW))
        return S_OK;

    if (!pVW)
        return S_OK;

          // Lee el estado actual
    JIF(pVW->get_FullScreenMode(&lMode));

    if (lMode == OAFALSE)
    {
                    // Salvar el mensaje actual
        LIF(pVW->get_MessageDrain((OAHWND *) &hDrain));

                    // Ajustar el mensaje actual a la aplicación de la ventana principal
        LIF(pVW->put_MessageDrain((OAHWND) ghApp));

                    // Cambiar al modo de pantalla completa
        lMode = OATRUE;
        JIF(pVW->put_FullScreenMode(lMode));
    }
    else
    {
                    // Cambiar de regreso al modo de ventana
        lMode = OAFALSE;
        JIF(pVW->put_FullScreenMode(lMode));
```

```
        // Undo change of message drain
        LIF(pVW->put_MessageDrain((OAHWND) hDrain));

                    // Reinicializar ventana de video
        LIF(pVW->SetWindowForeground(-1));

                    // Reclamar la atención del teclado
        UpdateWindow(ghApp);
        SetForegroundWindow(ghApp);
        SetFocus(ghApp);
    }

    return hr;
}

// Manejador de eventos en el Gráfico de filtros
HRESULT HandleGraphEvent(void)
{
    LONG evCode, evParam1, evParam2;
    HRESULT hr=S_OK;

    while(SUCCEEDED(pME->GetEvent(&evCode, &evParam1, &evParam2, 0)))
    {
                    // Recorrido atraves de los eventos
        hr = pME->FreeEventParams(evCode, evParam1, evParam2);

        if(EC_COMPLETE == evCode)
        {
            LONGLONG pos=0;

                            // Reinicializa al primer frame del video clip
            hr = pMS->SetPositions(&pos, AM_SEEKING_AbsolutePositioning ,
                                   NULL, AM_SEEKING_NoPositioning);
            if (FAILED(hr))
            {
                                        // Algunos filtros personalizados pueden no
implementar
                                        // interfaces de adelantar/atrasar.Para permitir
rrecorrer
                                        // hacia el inicio. En este caso, solo detener
                                        // y reiniciar para el mismo efecto. Esto no deberia
                                        //ser necesario en muchos casos.
                if (FAILED(hr = pMC->Stop()))
                {
                    Msg(TEXT("!Fallo(0x%08lx) para detener el clip multimedia!\r\n"), hr);
                    break;
                }

                if (FAILED(hr = pMC->Run()))
                {
                    Msg(TEXT("Falló(0x%08lx) para reiniciar el clip multimedia!\r\n"), hr);
                    break;
                }
            }
        }
    }

    return hr;
}

// Gestor de mensajes de Windows
LRESULT CALLBACK WndMainProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
                    // Reajusta el tamaño de video cuando la ventana cambia
        case WM_MOVE:
        case WM_SIZE:
            if ((hWnd == ghApp) && (!g_bAudioOnly))
                MoveVideoWindow();
            break;
```

```
        case WM_KEYDOWN:

            switch(wParam)
            {
                case 'P':     // Pausar/ Reproducir
                    PauseClip();
                    break;

                case 'D':     // Detener clip
                    StopClip();
                    break;

                                        case 'C': // Cerrar Clip
                    CloseClip();
                    break;

                case 'S':     // Silencio/Sonido
                    ToggleMute();
                    break;

                case 'A':     // Pantalla completa
                    ToggleFullScreen();
                    break;

                case 'M':     // Media pantalla
                    InitVideoWindow(1,2);
                    break;

                case 'T':     // 3/4 pantalla
                    InitVideoWindow(3,4);
                    break;

                case 'N':     // Pantalla normal
                    InitVideoWindow(1,1);
                    break;

                case 'V':     // OJO Videowall
                    InitVideoWindow(5,1);
                    break;

                                        case VK_F1: // Veremos las paginas propietarias del
filtro
                                                    {
                                                    HRESULT hr = pFX-
>QueryInterface(IID_ISpecifyPropertyPages, (void **)&pProp);
                                                    if (SUCCEEDED(hr)) {
                                                              // Muestra la pagina propietaria
                                                              // Obtiene el nombre del filtro
y un apuntador IUnknown.
                                                              FILTER_INFO FilterInfo;
                                                              pFX-
>QueryFilterInfo(&FilterInfo);
                                                              IUnknown *pFilterUnk;
                                                              pFX-
>QueryInterface(IID_IUnknown, (void **)&pFilterUnk);

                                                              CAUUID caGUID;
                                                              pProp->GetPages(&caGUID);
                                                              pProp->Release();
                                                              OleCreatePropertyFrame(
                                                                        hWnd,
// Ventana padre
                                                                        0, 0,
// (Reservado)
                                                                        FilterInfo.achName,
// Captura para la caja de dialogo
                                                                        1,
// Numero de objetos (solo del filtro)
                                                                        &pFilterUnk,
// Arreglo de apuntadores a objetos
```

```
                                                                caGUID.cElems,
// Numero de paginas propietarias
                                                                caGUID.pElems,
// Arreglo de CLSIDs de paginas propietarias
                                                                0,
// Identificador local
                                                                0, NULL
// Reservado

                                                        );

                                                        // Limpieza.
                                                        pFilterUnk->Release();
                                                        FilterInfo.pGraph->Release();
                                                        CoTaskMemFree(caGUID.pElems);
                                                }
                                        }

                                                        break;

                case VK_ESCAPE:
                case VK_F12:
                case 'Q':
                case 'X':
                    CloseClip();
                    break;
            }
            break;

        case WM_COMMAND:

            switch(wParam)
            { // Menus

                case ID_FILE_OPENCLIP:
                                                        // Si tenemos algun archivo abierto, lo
cerramos y apagamos DShow
                    if (g_psCurrent != Init)
                        CloseClip();

                                                        // Abre el nuevo clip
                    OpenClip();
                    break;

                                        case ID_TV:
                                                        Tele();
                                                        break;

                case ID_FILE_EXIT:
                    CloseClip();
                    PostQuitMessage(0);
                    break;

                case ID_FILE_PAUSE:
                    PauseClip();
                    break;

                case ID_FILE_STOP:
                    StopClip();
                                                        CloseInterfaces();

                    break;

                case ID_FILE_CLOSE:
                    CloseClip();
                                                        CloseInterfacesTV();
                    break;

                case ID_FILE_MUTE:
                    ToggleMute();
                    break;
```

```
                case ID_FILE_FULLSCREEN:
                    ToggleFullScreen();
                    break;

                case ID_FILE_SIZE_HALF:
                    InitVideoWindow(1,2);
                    break;
                case ID_FILE_SIZE_NORMAL:
                    InitVideoWindow(1,1);
                    break;
                case ID_FILE_SIZE_DOUBLE:
                    InitVideoWindow(2,1);
                    break;
                case ID_FILE_SIZE_THREEQUARTER:
                    InitVideoWindow(3,4);
                    break;


            } // Menus
            break;


        case WM_GRAPHNOTIFY:
            HandleGraphEvent();
            break;

        case WM_CLOSE:
            SendMessage(ghApp, WM_COMMAND, ID_FILE_EXIT, 0);
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);

    } // Manejador de mensajes de la ventana

    return DefWindowProc(hWnd, message, wParam, lParam);
}

// Función main de Windows
int PASCAL WinMain(HINSTANCE hInstC, HINSTANCE hInstP, LPTSTR lpCmdLine, int nCmdShow)
{
    MSG msg;
    WNDCLASS wc;

    // Initialize COM
    if(FAILED(CoInitialize(NULL)))
    {
        Msg(TEXT("Falló CoInitialize!\r\n"));
        exit(1);
    }

          // Fué especificado el nombre del archivo en la linea de comandos?
    if(lpCmdLine[0] != L'\0')
        lstrcpy(g_szFileName, lpCmdLine);

          // Ajusta el estado de multimedia inicial
    g_psCurrent = Init;

          // Registra la clase window
    ZeroMemory(&wc, sizeof wc);
    wc.lpfnWndProc = WndMainProc;
    ghInst = wc.hInstance = hInstC;
    wc.lpszClassName = CLASSNAME;
    wc.lpszMenuName  = MAKEINTRESOURCE(IDR_MENU);
    wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
```

```
    wc.hIcon         = LoadIcon(hInstC, MAKEINTRESOURCE(IDI_INWINDOW));
    if(!RegisterClass(&wc))
    {
        Msg(TEXT("Falló RegisterClass! Error=0x%x\r\n"), GetLastError());
        CoUninitialize();
        exit(1);
    }

          // Crea la ventana principal. El estilo  WS_CLIPCHILDREN es requerido, se ubica en
0,0
    ghApp = CreateWindow(CLASSNAME, APPLICATIONNAME,
                    WS_OVERLAPPEDWINDOW | WS_CAPTION | WS_CLIPCHILDREN,
                    0, 0,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    0, 0, ghInst, 0);

    if(ghApp)
    {
                    ShowWindow(ghApp, nCmdShow);
                    UpdateWindow(ghApp);

                    // Lazo de mensajes principal
        while(GetMessage(&msg,NULL,0,0))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    else
    {
        Msg(TEXT("Falló para crear la ventana principal! Error=0x%x\r\n"), GetLastError());
    }

          //Terminamos con el modelo de objetos componentes COM
    CoUninitialize();

    return msg.wParam;
}



//--------------------------------------------------------------------------------
// Archivo: PlayWnd.h
//
// Descripción: Archivo header para el archivo reproductor de multimedia
//
//
//--------------------------------------------------------------------------------


//
// Prototipos de función
//
HRESULT InitPlayerWindow(void);
HRESULT InitVideoWindow(int nMultiplier, int nDivider);
HRESULT HandleGraphEvent(void);

BOOL GetClipFileName(LPTSTR szName);

void PaintAudioWindow(void);
void MoveVideoWindow(void);
void CheckVisibility(void);
void CloseInterfaces(void);
void CloseInterfacesTV(void);

void Tele(void);
void OpenClip(void);
void PauseClip(void);
void StopClip(void);
void CloseClip(void);
```

```
IPin *GetPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir); //Pin que localiza un pin en el
filtro
void EnumFilters (char *nomFiltro, PIN_DIRECTION PinDir); // Funcion que encuentra el filtro
y pin en el grafico
void PaginaProp(void); // Funcion que despliega la pagina propietaria


void UpdateMainTitle(void);
void GetFilename(TCHAR *pszFull, TCHAR *pszFile);
void Msg(char *szFormat, ...);

HRESULT AddGraphToRot(IUnknown *pUnkGraph, DWORD *pdwRegister);
void RemoveGraphFromRot(DWORD pdwRegister);

//
// Constantes
//
#define VOLUME_FULL     0L
#define VOLUME_SILENCE  -10000L

// Archivos filtrados en el dialogo OpenFile
#define FILE_FILTER_TEXT \
    TEXT("Archivos de Video (*.avi; *.qt; *.mov; *.mpg; *.mpeg; *.m1v)\0*.avi; *.qt; *.mov;
*.mpg; *.mpeg; *.m1v\0")\
    TEXT("Archivos de Audio (*.wav; *.mpa; *.mp2; *.mp3; *.au; *.aif; *.aiff; *.snd)\0*.wav;
*.mpa; *.mp2; *.mp3; *.au; *.aif; *.aiff; *.snd\0")\
    TEXT("Archivos WMT (*.asf; *.wma; *.wmv)\0*.asf; *.wma; *.wmv\0")\
    TEXT("Archivos MIDI (*.mid, *.midi, *.rmi)\0*.mid; *.midi; *.rmi\0") \
    TEXT("Archivos de Imagen (*.jpg, *.bmp, *.gif, *.tga)\0*.jpg; *.bmp; *.gif; *.tga\0") \
    TEXT("Todos los archivos (*.*)\0*.*;\0\0")

// Directorio raiz por default para la busqueda de media
#define DEFAULT_MEDIA_PATH  TEXT("\\\0")

// Valores por default utilizados con archivos de solo-audio
#define DEFAULT_AUDIO_WIDTH     240
#define DEFAULT_AUDIO_HEIGHT    120
#define DEFAULT_VIDEO_WIDTH     320
#define DEFAULT_VIDEO_HEIGHT    240

#define APPLICATIONNAME TEXT("DEXTERMedia")
#define CLASSNAME       TEXT("PlayWndMediaPlayer")

#define WM_GRAPHNOTIFY  WM_USER+13

enum PLAYSTATE {Stopped, Paused, Running, Init};

//
// Macros
//
#define SAFE_RELEASE(x) { if (x) x->Release(); x = NULL; }

#define JIF(x) if (FAILED(hr=(x))) \
    {Msg(TEXT("FAILED(hr=0x%x) in ") TEXT(#x) TEXT("\n"), hr); return hr;}

#define LIF(x) if (FAILED(hr=(x))) \
    {Msg(TEXT("FAILED(hr=0x%x) in ") TEXT(#x) TEXT("\n"), hr);}


//
// Constantes predefinidas
//
#define IDI_INWINDOW                    100
#define IDR_MENU                        101
#define ID_FILE_OPENCLIP                40001
#define ID_FILE_EXIT                    40002
#define ID_FILE_PAUSE                   40003
#define ID_FILE_STOP                    40004
#define ID_FILE_CLOSE                   40005
#define ID_FILE_MUTE                    40006
#define ID_FILE_FULLSCREEN              40007
```

```
#define ID_FILE_SIZE_NORMAL                 40008
#define ID_FILE_SIZE_HALF                   40009
#define ID_FILE_SIZE_DOUBLE                 40010
#define ID_FILE_SIZE_QUARTER              40011
#define ID_FILE_SIZE_THREEQUARTER         40012
#define ID_TV                                                       40013
```