30

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"IMPLEMENTACION DE UN SISTEMA PARA ANALISIS Y CODIFICACION EN TIEMPO REAL DE IMAGENES DE VIDEO UTILIZANDO LA TRANSFORMADA POLINOMIAL"

T E S I S

QUE PARA OBTENER EL GRADO DE

INGENIERO EN COMPUTACION

P R E S E N T A N:

JOSE RAFAEL DIAZ RAMIREZ

RAUL EDUARDO SANCHEZ AQUINO

DIRECTOR: DR. BORIS ESCALANTE RAMIREZ

MEXICO, D. F. 2000

El presente agradecimiento no sólo es por la realización de nuestra tesis, sino también es la oportunidad para dejar escrito, lo que sentimos por las personas que más apreciamos.

Agradezco a Dios, por todo lo que he tenido en esta vida.

A mis padres, hermanos y a mi abuela: *Por haberme dado oportunidad, apoyo y creer en mí.*

A Jacqueline: *Por todo lo que significa en mi vida.*

A mis amistades Yoselin, Ale y Carmen: *Por estar siempre cuando les he necesitado.*

A mis mejores amigos de la Universidad: *Yazmin, Raúl, Fabiola y Javier:*
Porque uno se encuentra con las personas más especiales en el día menos pensado.

A mis amigas: *Margarita, Sandra, Gabriela, Nidia, Carolina, Elizabeth, Rosario, Luz, Italia, Benny, Zayd, Blanca, Erika, Liza, Miriam, Janete, Dulci, Verónica, Lulú, Alma y Patricia:*
Por haber compartido su amistad conmigo todos estos años, gracias.

A Mauricio, Eduardo, Yoes y Julian: *Gracias por su confianza.*

A los Doctores Boris y Savage: *Por su apoyo, tiempo y especialmente por haber permitido que se concluyera nuestra tesis en el Laboratorio de Interfaces Inteligentes. Gracias.*

A nuestros sinodales y maestros: *Les agradezco por sus enseñanzas y tiempo dedicado.*

Hay muchas personas que he conocido y que viven en mis recuerdos.
Por ello, sus nombres están escritos donde nunca han de ser borrados.

José Rafael Díaz Ramírez

A Dios, por permitirme alcanzar uno de mis mayores sueños.

A mi madre, por ser el eje impulsor en mi vida
y haberme demostrado en todo momento su cariño, amor y compromiso.

A Vivi, por su apoyo y paciencia que siempre me ha expresado.

A Coco, por haber sido mi ejemplo en el estudio, dedicación y trabajo.

A Gerardo, por hacerme comprender el valor de los sentimientos y del esfuerzo.

A Letty y Fabiola, por su apoyo incondicional, compromiso y consejos.

A Rafael, por ser mi amigo incondicional.

Raúl Edwardo Sánchez Aguirre

ÍNDICE

INTRODUCCIÓN

En los últimos años, producto del avance tecnológico en el campo de la electrónica, ha aumentado el número aplicaciones que hacen uso de imágenes para visualizar los resultados de algún proceso. Pero a pesar de este importante desarrollo tecnológico, la mayoría de los dispositivos de adquisición de datos son analógicos, lo que provocan que las imágenes sean sensibles al ruido. Para minimizar el efecto de este ruido, se han desarrollado varios algoritmos de procesamiento de imágenes, que se utilizan para mejorar la calidad de la imagen.

Muchos de estos algoritmos hacen uso de las derivadas de múltiple orden, debido a que éstas nos permiten detectar las características locales (bordes, líneas, esquinas y texturas) de una imagen. (Por lo que para poder hacer un análisis adecuado de la imagen se requiere utilizar un algoritmo que nos dé el valor de las derivadas de múltiple orden)

Las representaciones multiresolución han mostrado gran utilidad como modelos matemáticos y computacionales útiles en un gran número de aplicaciones de procesamiento de imágenes, tales como codificación y compresión, restauración, segmentación y visión computacional, etc.

Uno de los proyectos de investigación que se llevan a cabo en la División de Estudios de Posgrado de la Facultad de Ingeniería, desarrolla un modelo de representación de imágenes de vídeo por medio de transformaciones en tres dimensiones basadas en polinomios hermitianos y wavelets. El objetivo de este proyecto es estudiar la factibilidad de llevar a cabo tareas importantes de procesamiento de vídeo, como lo son la predicción de movimiento, la segmentación tridimensional y, finalmente, la compresión, por métodos multirresolución que incluyen la descomposición de video digital en bases ortogonales en tres dimensiones, es decir, dos espaciales y una temporal.

Entre los problemas principales que se han enfrentado esta el de disminuir el tiempo de ejecución del software que lleva a cabo la expansión polinomial o expansión en wavelets de una secuencia de vídeo. Para atacar este problema, es necesario paralelizar tanto el algoritmo como su implantación.

El presente desarrollo trata de mostrar las estrategias de paralelización para modelos de representación de imágenes, específicamente la paralelización de la transformada Hermitiana y, una evaluación del beneficio obtenido al paralelizar dicha transformada.

CAPITULO I
CONCEPTOS GENERALES

1.1 PROBLEMAS Y MODELOS DE PROCESAMIENTO DE IMÁGENES

Todas las aplicaciones de procesamiento de imágenes utilizan alguna forma de modelo matemático. Los continuos avances en los procesadores digitales de alta velocidad, memoria digitales e integración de muy alta escala(VLSI) permiten la implantación de algoritmos complejos. La siguiente tabla da una descripción de alguno de los problemas típicos en procesamiento de imágenes así como los requerimientos de su modelo.

	Problema	**Descripción**	**Modelos**
1.	SUAVIZADO	Dada una imagen ruidosa, filtrarla para suavizar las variaciones por ruido.	Ruido e imagen, espectro de potencia
2.	REALCE	Resaltar ciertas características de la imagen, por ejemplo bordes	Características (Se determina que característica es la que se desea resaltar)
3.	RESTAURACIÓN Y FILTRADO	Restaurar una imagen con una degradación conocida (o desconocida) tan cercana como sea posible a su forma original	Degradaciones, Criterio de "cercanía"
4.	COMPRESIÓN DE DATOS	Minimizar el numero de bits requeridos para guardar/transmitir una imagen a un nivel de distorsión establecido	Criterio de distorsión, Imagen como una fuente de información
5.	EXTRACCIÓN DE CARACTERÍSTICAS	Extraer ciertas características de una imagen	Características, criterio de detección
6.	DETECCIÓN E IDENTIFICACIÓN	Detectar e identificar la presencia de un objeto en una escena, por ejemplo análisis de texturas, etc.	Criterio de detección, objeto y escena
7.	INTERPOLACIÓN Y EXTRAPOLACIÓN	Dada una imagen con ciertos puntos en una región, estimar los valores de la imagen para todos los otros puntos dentro de esa región(interpolación) y también los puntos fuera de esa región (extrapolación)	Criterio de estimación, y grado de suavización de los datos.

8.	ESTIMACIÓN ESPECTRAL	Dada una imagen en una región, estimar su espectro de potencia	Criterio de estimación, Modelos a priori para datos
9.	FACTORIZACION ESPECTRAL	Dada la magnitud de la respuesta en frecuencia de un filtro, diseñar un filtro realizable, por ejemplo un filtro estable causal	Criterio de realizable (Se habla de un filtro realizable, cuando se considera que es posible hacerlo)
10.	SÍNTESIS	Dada una descripción de algunas características de una imagen, diseñar un sistema el cual reproduzca una replica de esa imagen, por ejemplo síntesis de textura	Características, Criterio de reproducción

Un algoritmo típico requiere la cuantificación del criterio de procesamiento y un modelo para el ancho de banda, espectro de potencia, etc. de los datos (entrada del algoritmo). A pesar de que la mayoría de los problemas mostrados en la tabla anterior ocurre en el procesamiento de señales de una dimensión, un cuidado especial es requerido en el desarrollo de algoritmos de dos dimensiones (o más). La mayor diferencia entre dimensiones de mayor orden es la causalidad. Una gran cantidad de métodos de procesamiento de señales de una dimensión están basados en el hecho de que los datos observados es la salida de un sistema causal. Para imágenes de dos dimensiones las coordenadas de los datos son espaciales y alguna causalidad asociada a la imagen sólo podrá ser por la técnica de adquisición. Por lo tanto no es extraño que un gran número de algoritmos de procesamiento de imágenes para extracción de bordes, realce, restauración, compresión de datos, etc. sean no causales.

1.1.1 Eficiencia Computacional

La eficiencia computacional de los algoritmos es medida por su requerimiento de la cantidad de memoria y operaciones requeridas. Los algoritmos más eficientes deben ser tales que el numero requerido de operaciones por pixel sean independientes del tamaño de la imagen. Desgraciadamente, un gran numero de algoritmos requieren una cantidad de operaciones que son proporcionales a logaritmo de N, N o mayor para imágenes de N x N **(el número de operaciones dependen del tamaño de la imagen)**. La siguiente tabla muestra algunas de las propiedades deseables para los modelos y algoritmos de dos dimensiones, las cuales tienden a minimizar su complejidad computacional.

Propiedad	2.1.1 Descripción
Linealidad	Operaciones lineales sobre los datos
Separabilidad	Operaciones independientes de renglones y columnas
Invarianza al desplazamiento	Operaciones tendientes a manipulación de matrices Toeplitzy circulares

Memoria Finita o Markoviana	Solo operaciones locales son requeridas en cada pixel, por ejemplo los filtros FIR

Lo que deseamos obtener de un algoritmo es que nos resuelva el problema de interés a la vez que cumpla las caracteristicas anteriores, lo que aumenta su eficiencia computacional y le permite hacer un calculos rápidos. Por lo general, utilizamos los algoritmos que nos resuelven el problema deseado o nos den una resultado aceptable, en un tiempo también aceptable.

1.2 ALGORITMOS EXISTENTES PARA CALCULO DE DERIVADAS MULTIORDEN

Muchos algoritmos han sido propuestos para calcular o estimar las derivadas de una imagen digital para extraer sus características locales. El diseño de estos algoritmos cae en alguna de estas tres categorías:

- Extensión de operadores diferenciales,
- Diferenciación de un polinomio adecuado
- Filtrado satisfaciendo condiciones o criterios específicos.

A pesar de existir varios procedimientos, la mayoría de los operadores derivativos son representados por mascaras de convolución en el caso discreto.
Los operadores del primer tipo (*extensión de operadores diferenciales*) como el de Roberts, Prewitt y Sobel, se caracterizan por un tamaño pequeño, lo cual causa que sea sensitivo al ruido y que no trabajen bien en imágenes ruidosas. Para poder incrementar su inmunidad al ruido, la mascaras pueden ser expandidas usando una gran vecindad o el tamaño efectivo de la mascara se puede incrementar obteniendo un promedio de la imagen antes del procesamiento.
En los operadores del segundo tipo (*diferenciación de polinomios adecuados*) una vez que se tiene un rango adecuado y el grado del polinomio es determinado, las derivadas se expresan como combinaciones lineales de datos dentro del rango deseado, los operadores se expresan como mascaras de convolución, debido a que las derivadas de un polinomio se determinan por sus coeficientes y estos son combinaciones lineales de los datos. Cuando se tiene un rango cuadrado o rectangular en el caso de dos dimensiones, la mascara se puede separar en dos de una dimensión. La desventaja de un rango cuadrado o rectangular es que los cálculos no son invariantes a la rotación y este efecto se incrementa con el tamaño de la mascara. Un rango circular puede utilizarse pero en este caso las mascaras no son separables. Procedimientos para hallar la mejor combinación de tamaño del rango y grado del polinomio todavía no son conocidos.
Los operadores del ultimo tipo (*filtrado satisfaciendo un criterio en particular*) son parecidos a los del primer tipo. Sin embargo, se establecen criterios para lograr una operación optima en vez de una simple extensión o promedio. El filtro de Wiener y el filtro Gaussiano para estimar el Laplaciano (una forma del operador derivativo de segundo orden) son ejemplos de este tipo.

1.2.1 Análisis de imágenes

Una imagen digital se define como un arreglo de valores de intensidades. Para poder hacer una interpretación inteligente de la imagen se debe obtener la información más importante de ésta. Esto generalmente implica determinar las relaciones entre estas intensidades y de hecho se requiere de un procesamiento local de los datos visuales.

El procesamiento requerido involucra dos decisiones importantes: Primero, para hacer el procesamiento local, la imagen se multiplica por una función ventana. El tamaño de la ventana establece el conjunto de puntos de la imagen que contribuyen en un paso básico del procesamiento. La forma de la ventana determina el peso relativo, con el que contribuye cada punto de la imagen. Para poder describir la imagen completamente se requiere que el procesamiento local se repita en un número suficiente de posiciones de la ventana.

Segundo, para cada posición de la ventana, se debe realizar un procesamiento especifico. Como un procesamiento especifico implica la búsqueda de patrones particulares, seleccionar el proceso es equivalente a fijar los patrones que son considerados más importantes *a priori.*

Es muy difícil escoger valores adecuados para la función ventana y el procesamiento con solo argumentos teóricos. Por lo cual se ocupa a menudo el sistema de visión humana como referencia.

Nos enfocaremos a un nuevo esquema de procesamiento local de información visual, llamada la transformada Hermitiana. Nos interesa el problema desde el punto de vista de codificación de la imagen y, por lo tanto, el esquema propuesto es un sistema de análisis/síntesis. Los objetivos, sin embargo, no se restringen solo a la codificación. Primero, la parte de análisis es diseñada de tal forma que pueda servir en aplicaciones de visión por computadora. De hecho, las derivadas de la Gaussiana, las cuales tienen grandes aplicaciones en la detección de características en los tiempos, juegan un papel central en el análisis Hermitiano. Segundo, además de integrar las ideas de las distintas (pero relacionadas) áreas de la codificación de imágenes y visión por computadora, esta argumentado que el esquema propuesto de procesamiento esta en amplia concordancia con la idea que tenemos de cómo se lleva a cabo el procesamiento de imágenes por el sistema de visión humano.

1.2.2 Wavelets

Las wavelets son una herramienta matemática para descomponer funciones jerárquicamente. Permiten que una función sea descrita en terminos de una forma general, mas datos que incrementan el detalle. Sin importar si la función de interés sea una imagen, una curva o una superficie, las wavelets ofrecen una técnica elegante de representar los niveles de detalles presentes.

A pesar que las wavelets tienen su origen en la teoría de aproximaciones y procesamiento de señales, recientemente han sido aplicados a una gran variedad de problemas de gráficos por computadora. Estas aplicaciones incluyen edición de imágenes, compresión de imagen, control del nivel de detalle para editar e interpretar de curvas y superficies., reconstrucción de superficies a partir de contornos, métodos rápidos para resolver problemas de simulaciones en animaciones e iluminación global, etc.

1.2.3 Wavelets en una dimensión

La base Haar es la base más simple de wavelets. Primero se vera cómo una función de una dimensión puede ser descompuesta usando wavelets Haar, y entonces veremos las funciones bases actuales.

Para tener una idea de como funcionan los wavelets, supongamos una imagen de una dimensión, con una resolución de 4 pixeles, teniendo los valores:

9 7 3 5

se puede representar esa imagen en las bases Haar, calculando una transformada wavelet. Para hacer esto, se promedian los pixeles contiguos, por pares, obteniéndose una nueva imagen de menor resolución con los siguientes valores:

8 4

Como se puede ver se ha perdido información en el proceso. Para regresar a los valores originales, se requieren *coeficientes de detalle,* los cuales guardan la información perdida. Para nuestro ejemplo los coeficientes de detalle son 1 y -1. Ya que 8 es 1 menor que 9 y uno mayor que 7, y 4 es -1 menor que 3 y -1 mayor que 5.

Si seguimos este procedimiento recursivamente obtenemos una descomposición completa, como se muestra en la siguiente tabla:

Resolución	Promedio	Coeficientes de detalle
4	9 7 3 5	
2	8 4	1 -1
1	6	2

Finalmente, se define la ***transformada wavelet*** (también ***llamada descomposición wavelet***) de la imagen original como un solo coeficiente representando el promedio de la imagen original, seguido de los coeficientes de detalle en orden de incremento de resolución. Así para nuestro ejemplo, la transformada wavelet de la imagen original sería:

6 2 1 -1

La forma de obtener la transformada wavelet, por medio de promedios y coeficientes diferenciales, es llamado un ***banco de filtros.*** Hay que destacar que ninguna información ha sido ganada ni perdida en este proceso. La imagen original tiene 4 coeficientes así como la

transformada. También cabe destacar que dada la transformada, se puede reconstruir la imagen a una resolución cualquiera por adiciones y sustracciones recurridas de los coeficientes de detalle sobre versiones de menor detalle.

Guardar la imagen generada por la transformada wavelet, tiene un gran numero de ventajas. Una ventaja es que a menudo un gran numero de coeficientes de detalle tienen muy poca magnitud, como se ve en la siguiente figura. Así si eliminamos estos coeficientes de la representación se producirán errores muy pequeños en la reconstrucción de la imagen, dándonos una forma de compresión de imagen "aceptable".

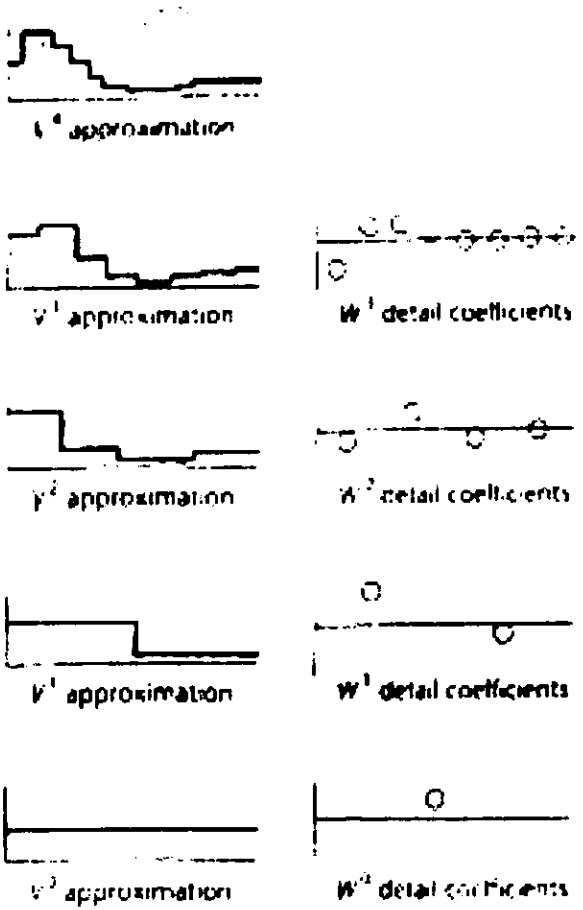


Imagen tomada de [Stollnitz, DeRose, Salesin, 1995]

1.2.4 Filtro Gaussiano

La respuesta impulso de un filtro Gaussiano G(x) esta dado por

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

con su transformada de Fourier

$$G(\omega) = \exp\left(-\sigma^2\omega^2/2\right)$$

Uno de los motivos de suavizar la imagen es limitar el índice con el cual las intensidades pueden cambiar así que una parte del espectro de la imagen pueda ser examinado. El filtro suavizante debe ser suave y de banda limitada en el dominio de la frecuencia para reducir el rango de frecuencias, además de suave y de banda limitada en el dominio del espacio, ya que la mayoría de los objetos tienen extensión de espacio limitado. La función Gaussiana da un equilibrio entre estos requerimientos en conflicto.

La función Gaussiana es infinitamente diferenciable. La respuesta impulso de la estimación del filtro derivativo k-esimo es:

$$G_k(x) = \frac{d^k}{dx^k} G(x) = G^{(k)}(x),$$

y la función de transferencia es:

$$G(\omega) = (j\omega)^k G(\omega)$$

1.2.5 Derivadas en 2 Dimensiones

Uno de los propósitos de la estimación de derivadas es extraer características locales para operaciones subsecuentes, como detección de bordes, detección de líneas, clasificación de texturas, etc. Puesto que una imagen cualquiera no tiene una dirección predeterminada para los bordes, líneas o texturas, filtros independientes a la dirección son deseables. Un estimador derivativo optimo, esta compuesto de un filtro para estimar la imagen en si y un operador derivativo ideal. Para ser independiente a la dirección el filtro suavizante y el operador derivativo deben ser isotrópicos. Los filtros Gaussianos tienen esta característica cuando se usan en dos dimensiones.

El filtro suavizante Gaussiano en 2 dimensiones:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\{-(x^2+y^2)/2\sigma^2\}$$

es isotrópico y separable.

Como el filtro suavizante es separable, el filtro derivativo parcial estimador es también separable.

$$G^{(k,l)}(x,y) = \frac{\partial^{k+l}}{\partial x^k \partial y^l} G(x,y)$$
$$= G^{(k)}(x) G^{(l)}(y)$$

Supongamos $\hat{f}^{(k,l)}(x,y)$ sea la derivada parcial estimada de orden k en x y de orden 1 en y. La derivada parcial se estima como:

$$\hat{f}^{(k,l)}(x,y) = G^{(k)}(x) G^{(l)}(y) * f_o(x,y)$$

donde $f_o(x,y)$ es la imagen observada con ruido aditivo y * denota la convolución en dos dimensiones. Cuando el filtro se implementa en el dominio del espacio, la convolución de dos dimensiones puede calcularse por dos convoluciones de una dimensión y economizar el tiempo de computo.
Una derivada direccional en una dirección arbitraria puede ser calculada por derivadas parciales del mismo orden en un sistema coordenado fijo (x,y). Supongamos un sistema coordenado (u,v), rotado θ grados del eje x. la derivada parcial de primer orden en la dirección u, esta dada por una combinación lineal como:

$$\frac{\partial \hat{f}(x,y)}{\partial u} = \frac{\partial x}{\partial u} \hat{f}^{(1,0)}(x,y) + \frac{\partial y}{\partial u} \hat{f}^{(0,1)}(x,y)$$
$$= \cos\theta \hat{f}^{(1,0)}(x,y) + sin\theta \hat{f}^{(0,1)}(x,y)$$

Similarmente una derivada de mayor orden puede ser obtenida como:

$$\frac{\partial^n \hat{f}(x,y)}{\partial u^n} = \sum_{i=0}^{n} \binom{n}{i} (\cos\theta)^{n-i} sin^i \theta \hat{f}^{(n-i,i)}(x,y)$$

Por otro lado una derivada direccional también se estimada por la estimación del filtro derivativo direccional como:

$$\frac{\partial^n \hat{f}(x,y)}{\partial u^n} = \frac{\partial^n}{\partial u^n} G(x,y) * f_o(x,y)$$

1.2.6 Paralelización

Las ventajas de la paralelización son bien conocidas, pero debemos tener presentes las siguientes:

- Permite resolver problemas matemáticos mas rápidamente
- Permite realizar un mayor análisis en un periodo de tiempo dado,

- Permite realizar tareas independientes simultáneamente,
- Permite incrementar la respuesta de los programas en tiempo real, etc.

Existen tres niveles de paralelización:

- Paralelización del algoritmo,
- Paralelización del software y,
- Paralelización del hardware

Aquí nos enfocaremos a la paralelizacion pero desde un punto de vista de programación (el algoritmo y el software), debido a que es más simple de implementar, da mayores facilidades para modificar e involucra un menor costo que la paralelización del hardware.
Veamos una breve descripción de las características que implica cada uno de los diversos niveles de paralelización.

1.3 PARALELIZACIÓN DEL ALGORITMO

El principal requisito para ejecutar un código es tener un buen algoritmo. Se pueden hacer pocas cosas para ayudar a que el optimizador trabaje, pero si no se comienza por un buen algoritmo, no se pude obtener la ejecución ideal.
Tener un "buen" algoritmo puede ser un tanto subjetivo, pero lo más deseable es que el algoritmo utilizado en si nos facilite la utilización de un equipo paralelo, es decir, un algoritmo que permita ser implementado en un equipo paralelo en forma natural.

1.4 PARALELIZACIÓN DEL SOFTWARE

El concepto de "paralelizar el software" hace referencia a afinar el código de los programas para obtener el mayor provecho de un hardware en paralelo.
Hay veces que la velocidad hace la diferencia entre el fracaso y el éxito. Las aplicaciones de tiempo real tienen requerimientos fijos de la velocidad a la que se deben ejecutar, velocidades mas bajas no son aceptables.
Por otra parte, para un sistema interactivo grandes intervalos de tiempo entre la entrada de datos por parte de los usuarios y las actualizaciones del sistema hacen a los usuarios menos productivos. Los usuarios pierden tiempo esperando para las actualizaciones del sistema, lo cual pueden afectar su concentración, por lo tanto causan retrasos entre pasos.
Hacer el código más rápido y pequeño puede hacer que se pueda migrar la aplicación a plataformas más pequeñas y obtener resultados aceptables. Esto permite que pueda servir a un amplio rango de clientes, es decir, proveen una familia de soluciones.
Los limites de la ejecución del código depende de las características del hardware donde se va a ejecutar y los requerimientos que impone el algoritmo que se va a implementar.

Antes de optimizar el código, hay que hacer un análisis para determinar donde nos debemos enfocar para optimizar, es decir, se debe de hallar dónde el código gasta la mayor parte del tiempo y donde es posible paralelizar el código.

1.5 PARALELIZACIÓN DEL HARDWARE

El concepto de un hardware con arquitectura en paralelo, es la de un equipo con dos o más unidades de procesamiento funcionando simultáneamente, donde cada unidad de procesamiento puede tener desde un solo un procesador hasta ser un computadora independiente (CPU,FPU, buses de E/S, memoria, subsistema gráfico, etc.).

CAPITULO II
TRANSFORMADA POLINOMIAL DE 2 DIMENSIONES.

2.1 ANÁLISIS: Teoría de la Transformada Polinomial de 2 Dimensiones y Transformada Hermitiana.

2.1.1. Análisis Local (Importancia).

En la Transformada Polinomial se realiza un procesamiento de la imagen para obtener los componentes de la imagen.

Con la Transformada Polinomial es posible detectar la posición y orientación de bordes relevantes de la imagen. Este procesamiento tiene como fin principal, extraer la información más relevante para el análisis de la imagen, siendo esta información la localización, orientación y el contraste de las transiciones de luminiscencia. Para extraer esta información es necesario el procesamiento local con ventanas o campos de distintos tamaños a diferentes resoluciones[Escalante, Martens,1992].

Para el procesamiento local es necesario tomar en cuenta:

- El tamaño de la ventana, que se usara para procesar la imagen es muy importante, porque usualmente la imagen es multiplicada por la ventana, y ese procesamiento, determinará el peso relativo que tendrán cada uno de los puntos de la imagen. Además de que este procesamiento local en la imagen es repetido en cada una de las posiciones de la ventana sobre la imagen.
- Como en cada posición se realiza el procesamiento, también se realiza la búsqueda de un patrón o característica especifica. Por lo cual es necesario considerar, cuál será la característica más importante [Martens,1990].

Si en la selección del tamaño de la ventana, se quiere obtener una alta reducción de los datos, es necesario que la ventana sea grande, pero esto implica que a mayor tamaño de la ventana se tenga una mayor complejidad del análisis. Por lo cual, se puede actuar de dos maneras:

- Seleccionar un tamaño de la ventana y realizar el análisis dentro de cada ventana, siendo lo suficientemente compleja para que pueda incluir todos las características que sean de interés.
- Limitar la complejidad del análisis en cada ventana y después determinar el tamaño de ventana necesario para volver a analizar la sección de la imagen con suficiente exactitud. [Martens,1990].

La Transformada Polinomial se puede realizar a múltiples escalas, utilizando ventanas de distintos tamaños, por lo cual, dependiendo del tamaño de la ventana será el espacio que tengan una de otra, siendo este espacio proporcional al tamaño de la ventana.

2.1.2. Relación con la Percepción Visual.

Para muchas aplicaciones de codificación de imágenes y *visión computacional*, así como también en la visión humana, es necesario que los datos de la imagen se encuentren como un arreglo de valores de intensidad, para que puedan ser interpretados en patrones visuales tales como bordes, líneas, texturas, etc. Es difícil conseguir un óptimo procesamiento de la imagen por medio de ventanas rectangulares, teóricamente se deberían usar ventanas hexagonales como en la visión humana, pero es más práctico el uso de ventanas rectangulares.

Con la Transformada Polinomial se puede obtener una descripción de los cambios de intensidad de la imagen, además de que incorpora propiedades de la visión, como el modelo de derivadas gausianas.

Para implementarse se usan ventanas rectangulares aunque sería más conveniente usar ventanas hexagonales pues son más acordes a la visión humana.

En la Transformada Polinomial se puede realizar un procesamiento a múltiples escalas y a su vez, se utilizan los componentes obtenidos para seleccionar la escala óptima en cada posición. Este es una análisis "espacio escala" similar al que realiza el sistema de visión humana mediante campos receptivos de distintos tamaños, que están ubicados en la retina.

2.1.3. Ventanas Gaussianas y Polinomios Hermitianos (Casos).

Usando la Transformada Polinomial, se verá el caso especial en que la ventana es una función gausiana:

$$V(x,y;\sigma) = \frac{1}{\sqrt{\pi\sigma}} \exp\left[\frac{-(x^2+y^2)}{2\sigma^2}\right]$$

donde el factor de normalización es tal que $V^2(x, y)$ tiene energía unitaria. Los polinomios ortogonales que están asociados con $V^2(x, y)$ son conocidos como polinomios Hermitianos, por lo cual, el proceso que se realiza sobre la muestra de imagen es una Transformada Hermitiana [Martens,1990].

Las razones por la que son importantes las ventanas gausianas, son las siguientes:

- Las propiedades de la Transformada Hermitiana pueden ser fácilmente derivadas y evaluadas.
- Las ventanas gausianas traslapadas y separadas entre si por una distancia σ son buenos modelos de los campos receptivos de la visión humana encontrados en experimentos fisiológicos.
- Con la Transformada Hermitiana se pueden usar filtros cuyas funciones sean derivadas de funciones gausianas.

Estas funciones gausianas son extensamente encontradas en *visión computacional* y en los modelos fisiológicos del sistema de visión humana. Además según la Transformada Hermitiana todas las derivadas gausianas son importantes, dependiendo de la característica o patrón contenido [Martens,1990]. Por lo cual la Transformada Hermitiana proporciona un marco teórico más amplio y mejor, para los modelos de visión humana.

2.1.4. Expansión a dos dimensiones (mediante polinomios).

La Transformada Polinomial puede ser generalizada a dos dimensiones. Como la ventana es colocada en varias posiciones dentro de la imagen, estas posiciones constituyen muestras dentro de una retícula, que llamaremos *S*. Y dentro de cada muestra se realizará un procesamiento de la imagen, el cual nos dará una descripción de ella, por medio de pesos. El mapeo que se realiza de la imagen a los pesos es llamado Transformada Polinomial y los pesos son los coeficientes de los polinomios. La interpolación de los coeficientes de los polinomios con funciones especificas nos permiten obtener la imagen original, este proceso es llamado Transformada Polinomial Inversa.

Los coeficientes polinomiales $L_{m,n-m}(p, q)$ que se obtienen de los polinomios $G_{m,n-m}(x-p, y-q)$, por todas las posiciones $(p, q) \in S$ que toma la ventana $V(x-p, y-q)$, se obtienen mediante la convolución de la imagen $L(x, y)$ con un filtro:

$$D_{m,n-m}(x,y) = G_{m,n-m}(-x,-y)V^2(-x,-y)$$

y los coeficientes de las salidas se obtienen mediante:

$$L_{m,n-m}(p,q) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} L(x,y)D_{m-n-m}(p-x,q-y)dxdy$$

evaluando en las posiciones (p, q) de la retícula S, para $m=0,...,n$ y $n=0,1,...N$.

$G_{m,n-m}(x,y)$ es el polinomio de orden m en x y $n-m$ en y. Además $G_{m,n-m}(x,y)$ son polinomios ortogonales, llamados polinomios *Krawtchouck* en el caso de ventanas binomiales y Hermitianos en las ventanas gausianas.

Se debe escoger una ventana gausiana para una aproximación continua, por sus propiedades matemáticas:

$$V(x,y;\sigma) = \frac{1}{\sqrt{\pi\sigma}}\exp\left[\frac{-(x^2+y^2)}{2\sigma^2}\right]$$

pues es separable, simétrica al rotamiento y porque está más acorde con la psicofisiología de la visión, formalizada en la teoría de "espacio escala", por lo cual modela eficientemente los campos receptivos de la retina. Por lo que, esto implicaría que la Transformada Polinomial sea una Transformada Hermitiana.

Como se ha mencionado anteriormente, el tamaño de la ventana también es importante, pues se relaciona con la escala en la que se realizará el análisis. Pues si nosotros deseamos detectar tan sólo cambios finos, se tiene que usar ventanas de tamaño pequeño. Pero si se desea una eficiente detección de objetos que tengan una baja resolución, entonces se tendría que utilizar una ventana grande. Por lo cual, el uso de una simple ventana, no resulta eficiente. Como una alternativa, se puede realizar el análisis con múltiples resoluciones, siendo las estructuras piramidales las que cumplen este objetivo [Viveros, Escalante, 1997].

2.1.5. Discretización

La aproximación discreta de los filtros, siendo estos derivadas de gausianas, en el dominio del espacio, mantienen las propiedades de los filtros continuos asintóticamente [M.Hashimoto, J. Sklansky, 1987].

Siendo $d_{N,k}$ un vector cuya dimensión es $(N+1)$, definido como:

$$d_{N,k} = 2^{N-k} \begin{bmatrix} d_{N,k}(N) \\ d_{N,k}(N-1) \\ \cdot \\ \cdot \\ \cdot \\ d_{N,k}(0) \end{bmatrix}, k = 0,1,...N.$$

En una dimensión, las estimaciones de las derivadas pueden ser:

$$\hat{f}^{(k)} = \frac{1}{2^{N-k}} d_{N,k}^T f_O, k = 0,1,...,N.$$

Donde f_o es un vector de datos de dimensión (N+1) y las derivadas son calculadas en el centro de la ventana. Se define la matriz $[P_N]$ de $N+1$ por $N+1$ tal que sea $k+1$-esimo renglón es $d^T_{N,k}$. F es una matriz de datos conteniendo todas las derivadas de f_o:

$$F = (F_0, F_1, ..., F_N)^T$$

$$F = [P_N] f_o$$

La matriz $[P_N]$ para $N=2$ es la siguiente:

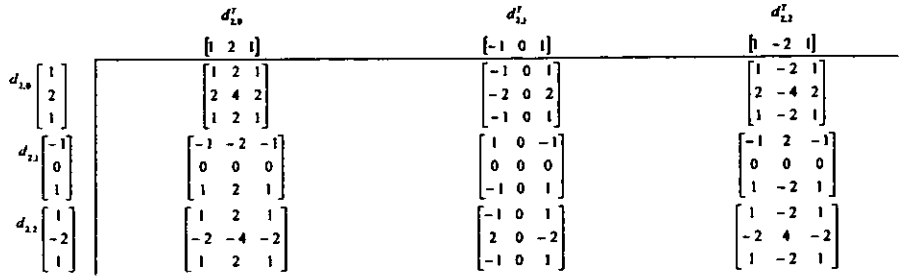
$$\begin{bmatrix} 1 & 2 & 1 \\ -1 & 0 & 1 \\ 1 & -2 & 1 \end{bmatrix}$$

En dos dimensiones, las derivadas parciales son estimadas usando un operador de una dimensión para cada dirección. Una matriz $[F]$ cuyas derivadas parciales son:

$$[F] = \begin{bmatrix} F_{00} & F_{01} & ...F_{0N} \\ F_{10} & F_{11} & ...F_{1N} \\ ... & ... & ... \\ F_{N0} & F_{N1} & ...F_{NN} \end{bmatrix}$$

$$[F] = [P_N][f_o][P_N]^T,$$

Donde $[f_o]$ es una matriz de datos en una ventana de $(N+1)$ por $(N+1)$. Cada elemento de $[F]$ es calculado, con una correlación de la imagen con un filtro $d_{N,k}\ d^T_{N,k}$ [M.Hashimoto, J. Sklansky,1987]. Nueve filtros de *3 x 3* para $N=2$ son mostrados a continuación:



Los filtros de la primera derivada son iguales a los filtros de Sobel.
Esta representación de la matriz, muestra que la expresión de múltiples derivadas de múltiples órdenes es una transformación discreta [M.Hashimoto, J. Sklansky,1987].

2.2 SÍNTESIS: Transformada Polinomial de 2 Dimensiones Inversa

2.2.1. Ecuaciones de la Interpolación.

Cuando se realiza la Transformada Polinomial, se obtienen unos coeficientes, los cuales se pueden interpolar para obtener la imagen resintetizada. A esto se le llama Transformada Polinomial Inversa.

La imagen resintetizada $\hat{L}(x,y)$ es obtenida con:

$$\hat{L}(x,y) = \sum_{n=0}^{N} \sum_{m=0}^{n} \sum_{(p,q)\in S} L_{m,n-m}(p,q) P_{m,n-m}(x-p, y-q)$$

La función de interpolación esta definida por:

$$P_{m,n-m}(x,y) = G_{m,n-m}(x,y) V(-x,-y) / W(x,y),$$

para $m = 0,\ldots, n$ y $n = 0,\ldots, N$ donde:

$$W(x,y) = \sum_{(p,q)\in S} V(x-p, y-q)$$

es llamada la función de pesos, la cual es definida por la retícula S, y la única condición para que exista para la Transformada Polinomial es que esta función de pesos sea diferente de cero, para todas las coordenadas (x, y).

2.3 PROYECCIÓN A LA TRANSFORMADA POLINOMIAL DE UNA DIMENSIÓN: ¿Para qué proyectar de dos Dimensiones a una Dimensión?

Como se mostrará más adelante, las ventajas de la proyección a la transformada polinomial de una dimensión se encontrará en las aplicaciones de codificación y restauración.

2.3.1. Proyección de 2 dimensiones a 1 dimensión.

Con la Transformada Polinomial de dos dimensiones se puede hacer una aproximación a la Transformada Polinomial de una dimensión [Martens,1990]. Por otra parte, los coeficientes del polinomio de una dimensión se aproximan a la detección del óptimo borde. Los coeficientes K del polinomio de una dimensión de orden k en el ángulo θ son obtenidos de los coeficientes del polinomio de dos dimensiones, por:

$$K_{k,\theta} = \sum_{n=0}^{k} \sum_{m=0}^{n} L_{m,n-m} h_{k,\theta}(m, n-m),$$

donde $L_{m,n-m}$ es el coeficiente polinomial en $(m,\ n\text{-}m)$, $h_{k,\theta}$ es una función de ángulos, proyecta los coeficientes en la dirección θ.

En cada posición donde un borde es encontrado, una Transformada Polinomial de una dimensión es calculada en diferentes direcciones, de acuerdo a la ecuación anterior, y se guardan los coeficientes de una dimensión en un vector $K_{N,\theta i} = [K_{0,\theta i}\ K_{1,\theta i}\ \ldots\ K_{N,\theta i}]$.

En donde se encuentra que no hay bordes, es decir, es regiones homogéneas los coeficientes de orden cero son guardados, estos son los coeficientes que representan el promedio local de una región suavizada de la imagen. Todos los coeficientes de orden superior son fijados a cero en estas regiones homogéneas.

Para encontrar la óptima orientación del borde, se encuentra el máximo de la función de contraste direccional definida por:

$$C_\theta = \log\left[1 + \sum_{k=1}^{N} K_{k,\theta}^2\right]$$

en todos los diferentes ángulos θ.

En cada posición donde un borde fuera encontrado, solamente los coeficientes de una dimensión que tienen un mayor contraste son mantenidos en el vector $K_{N,\theta opt}$. Este paso de discrimanción representa en sí mismo un proceso de restauración, pues únicamente los patrones que conforman un borde son preservados. Todos los patrones que conforman estructuras de la imagen orientadas a diferentes direcciones, tales como el ruido, son descartadas. Por otra parte, los coeficientes de orden mayor o igual a uno, que pertenecen a regiones donde no se encontraron bordes son fijados a cero [Escalante, López, 1996].

2.3.2. Proyección de 1 dimensión a 2 dimensiones.

De los coeficientes de los polinomios de una dimensión (con una orientación óptima), sus correspondientes coeficientes de los polinomios de dos dimensiones pueden ser encontrados con:

$$L_{m,n-m} = \sum_{k=n}^{\infty} K_{k,\theta} \cdot h_{k,\theta}(m, n-m).$$

donde $n = 0,1, \ldots, N$ y $m = 0, \ldots, n$. Para los casos en que $k > N$, los coeficientes $k_{k,\theta}$ son desconocidos.

Se le puede realizar la Transformada Polinomial Inversa de dos dimensiones, sobre los coeficientes obtenidos, en la Transformada Inversa de una dimensión, reconstruyendo la imagen, la cual restaura las regiones donde los bordes fueron encontrados (por la Transformada de una dimensión), mientras que suaviza las regiones donde no se encontraron bordes [Escalante, López, 1996].

2.4 APLICACIONES. Aplicaciones de Transformada de 2 Dimensiones proyectada en la Transformada de 1 Dimensión

2.4.1 Codificación (reducción de datos, Expansión).

La Transformada Polinomial puede representar una imagen, así como también imita algunas de las propiedades del sistema de visión humana. Basados en ella, es posible construir un esquema jerárquico piramidal para la codificación de imágenes, siendo las características que se podrán codificar, las siguientes: promedio local, orientación del borde, posición del borde y magnitud del borde [Viveros, Escalante, 1997].

Se sabe que los bordes de una imagen, cuando tienen una degradación, si son altamente percibidos por el ojo humano. Esto es ignorado por algunas técnicas de compresión porque están enfocados a almacenar tanta energía como sea posible, sin considerar, si la energía en realidad representa las características más importantes para la percepción humana. Sin embargo, los métodos de multiresolución, están inspirados en los múltiples canales espaciales del sistema de visión humana, por lo cual, muestran ser útiles en la codificación de estructuras que ayudan a obtener altos rangos de compresión. Cuando se realiza un análisis en la imagen por medio de filtros, es deseable detectar los finos cambios que se detectan con filtros pequeños, pero por otro lado, es deseable detectar los objetos que tienen una baja resolución, los cuales se detectan con filtros grandes. Para lograr ambas cosas una de las mejores alternativas es el uso de un análisis a múltiples resoluciones, por ejemplo uno de las estructuras que cumplen dicho objetivo, es el uso de estructuras piramidales [Viveros, Escalante, 1997].

Cuando se realiza el análisis (Transformada Polinomial), la información obtenida es redundante, sin embargo, esta nos permite tener las características más importantes y

relevantes de la imagen, tales como los bordes, de los cuáles, podemos obtener su orientación, amplitud y posición. Esta información es posible proyectarla a una dimensión, donde lo más redundante desaparece [Viveros, Escalante, 1997].
Una estructura de análisis piramidal es más eficiente para detectar estructuras a diferentes dimensiones espaciales con operadores a diferentes escalas.
La imagen original es procesada con la Transformada Polinomial de 2 dimensiones, después se realiza la proyección a 1 dimensión y se extraen los siguientes parámetros:

- Promedio Local (de la proyección a 2 dimensiones).
- Orientación de la imagen (de la proyección de 2 dimensiones a 1 dimensión).
- Posición del borde (de la proyección a 1 dimensión).
- Magnitud de la posición (de la proyección a 1 dimensión).

Estos parámetros son altamente cuantificables. Después de la cuantificación la imagen puede ser estimada por medio de una serie de operaciones inversas, reconstruyendo con los coeficientes de 1 dimensión y proyectarlos a coeficientes polinomiales 2 dimensiones, y haciendo una transformada de 2 dimensiones inversa.
Como es posible la eliminación de redundancia, este es el primer paso para un esquema de codificación de la imagen, que caracteriza los bordes en múltiples escalas. Esto reduciría la entropía sin que se tenga una perdida en la calidad perceptiva de una imagen [Viveros, Escalante, 1997].

2.4.2 Suavizado del ruido.

Es posible hacer la restauración de imágenes con la Transformada Polinomial, pues se puede realizar un análisis y direccional que reduce el ruido de la imagen [Camarillo, Varela, Escalante, 1998].
Con la Transformada Polinomial se obtiene una aproximación de la imagen usando una ventana, y con diferentes tamaños de la ventana toma una modalidad de multiresolución. Haciendo la proyección a 1 dimensión, se obtendrá la dirección óptima de los bordes, esto tiene como fin, descartar los objetos que tienen diferentes orientaciones pues se les considera ruido.
Debe tomarse en cuenta que al realizarlo, es posible que el ruido que se encuentra cercano a los bordes no pueda ser eliminado del todo, pero sí el que se encuentra en superficies lisas.

CAPITULO III
PARALELIZACIÓN DE LA TRANSFORMADA HERMITIANA Y RESULTADOS.

Analizaremos los distintos niveles de paralelización (algoritmo, hardware, software) aplicados al cálculo de la transformada Hermitiana.

3.1 TEORÍA.

3.1.1 Paralelización del Algoritmo.

Como se menciono anteriormente la base de una buena paralelización se basa en el algoritmo, no se puede hacer grandes cosas sin no se cuenta con un buen algoritmo. Así que lo primero que debemos hallar es el algoritmo para implementar la transformada Hermitiana.

Para estimar las derivadas de múltiple orden se pueden implementar un conjunto de filtros discretos de varias maneras:

- Convoluciones con un conjunto de mascaras de 2 dimensiones
- Multiplicación de matrices para cada ventana de datos
- Convoluciones por un conjunto de mascaras de una dimensión en ambas direcciones
- Una serie de convoluciones por dos secuencias {1,1} y {1,-1} en ambas direcciones.

De estos algoritmos, el repetir las convoluciones por las dos secuencias de números, como veremos, es la que puede ser implementada con la menor cantidad de operaciones. Si la repetición de convoluciones es implementada directamente, se requiere de una gran cantidad de localidades de memoria extra, para guardar resultados intermedios. M. Hashimoto y J. Sklansky han desarrollado algoritmos rápidos que calculan las convoluciones de las dos secuencias mientras que la cantidad de localidades de memoria extra es muy pequeña. Estos algoritmos pueden fácilmente implementarse en software o hardware. Comenzaremos con el caso en una dimensión.

3.1.2 Algoritmo rápido en una dimensión.

Como la función de transferencia del filtro estimador de la k-esíma derivada es:

$$D_{N,k}(z) = \frac{z^{N/2}}{z^{N-k}}(1+z^{-1})^{N-k}(1-z^{-1})^{k},$$

puede ser implementada poniendo en cascada dos tipos de filtros de primer orden cuya función de transferencia sea (1 + z -1) y (1 - z -1). Esta implementación en cascada es equivalente a convoluciones repetidas por las secuencias {1,1} y {1,-1}.

El algoritmo para la estimación de derivadas de múltiple orden se obtiene por usando la implementación en cascada para cada filtro en paralelo. Debido a que la función de transferencia de los filtros estimadores derivativos tiene términos comunes, las operaciones redundantes se pueden ser eliminados compartiendo resultados intermedios. El número de operaciones redundantes incrementa a medida que N se hace mayor. A continuación se muestra el algoritmo para N=1, N=2 y N=3, donde cada filtro de primer orden se implementa por una operación de retraso y una suma.

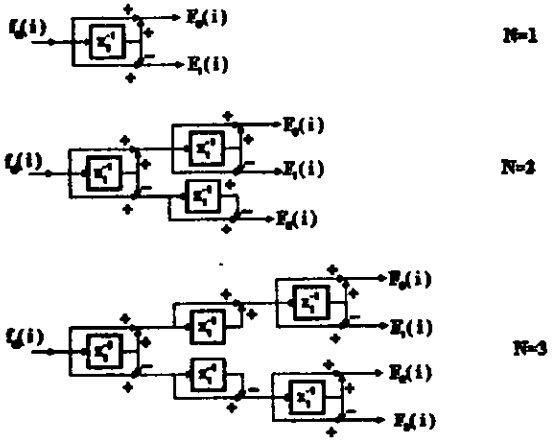


Imagen tomada de [M.Hashimoto, J. Sklansky,1987]

Se pueden construir un algoritmo rápido para grandes valores de N, poniendo en cascada los filtros de primer orden y algoritmo rápidos para tamaños de N pequeños, como se ve en la figura:

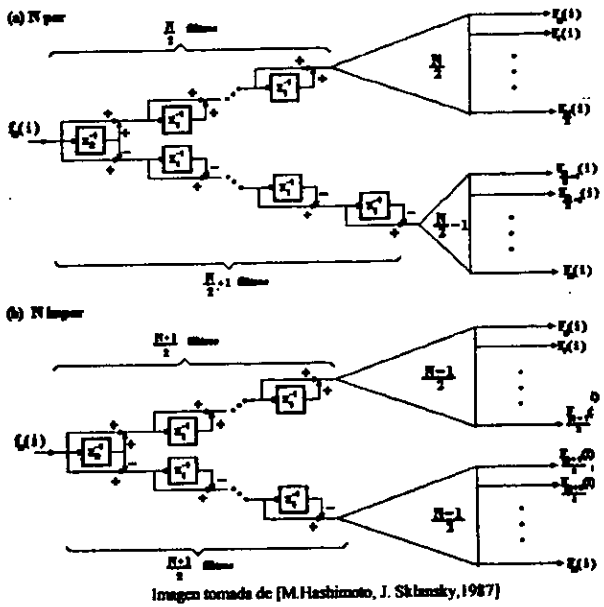


Imagen tomada de [M.Hashimoto, J. Sklansky,1987]

Cada triángulo representa un algoritmo rápido para el tamaño especificado. Puesto que la función de transferencia de la salida Fk(i) es (1 - z -1)k (1 + z -1) N - k , tenemos que modificar la salida como:

$$\hat{f}^{(k)}(i) = \frac{1}{2^{N-k}} F_k(i + N/2)$$

Como el algoritmo propuesto para valores grandes de N, esta compuesto del calculo del algoritmo para pequeños valores y una serie de filtros de primer orden, las ecuaciones para el número requerido de sumas y retrasos puede expresarse en forma recursiva:

- Cuando N+1 es una potencia de 2 el número de sumas por muestra esta dado por (N+1) $\log_2$ (N+1), debido a que cada estructura se divide en 2 subestructuras y cada una de estas en otras 2 y así sucesivamente.
- Para N+1 que no es potencia de 2 el más pequeño entero mayor o igual a (N+1)$\log_2$ (N+1), dará el número exacto de sumas.
- El número de retardos es el número de sumas menos N.
- Por ejemplo, para N=1,2,3 obtenemos:

N	*Numero de sumas*	*Número de retardos*
1	2	1
2	5	3
3	8	5

3.1.3 Algoritmo rápido para dos dimensiones.

Como los filtros estimadores derivativos en dos dimensiones son separables, las derivadas parciales pueden ser estimadas aplicando un algoritmo rápido en una dimensión a cada columna(o renglón) de la imagen de entrada y entonces a cada renglón (o columna) de los resultados. En una dimensión los filtros son implementados sin multiplicaciones utilizando la relación recursiva de resultados intermedios entre muestras vecinas. Debido a esta propiedad recursiva la entrada debe dirigirse a un conjunto de filtros continuamente. En dos dimensiones no es posible dirigir la imagen de entrada continuamente en ambas direcciones al menos que use un algoritmo en paralelo para cada columna o para cada renglón. La siguiente figura muestra el algoritmo rápido en dos dimensiones donde cada triángulo representa el algoritmo rápido en una dimensión de tamaño N.

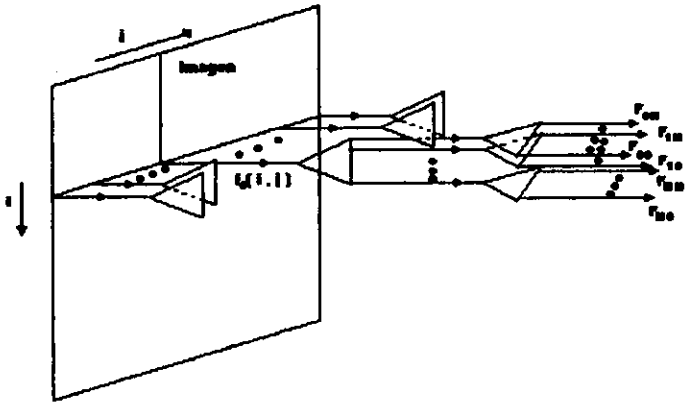


Imagen tomada de [M.Hashimoto, J. Sklansky,1987]

Los filtros de columnas son paralelos a todas las columnas de la imagen de entrada y los filtros de renglones son paralelos para N=1 salidas de los filtros columna. La estimación de las derivadas parciales se obtiene de la modificación:

$$\hat{f}^{(k,l)}(i,j) = \frac{1}{2^{2N-k-l}} F_{kl}(i + N/2, j + N/2)$$

En este caso no se utilizan las (N+1)2 derivadas parciales como características locales, dado que se pueden borrar las partes innecesarias del algoritmo o redundantes. Reacomodando el orden de los filtros de primer orden podemos simplificar el algoritmo.

Veamos casos sencillos para ver la implementación de los algoritmos en el caso de dos dimensiones:

Caso 1 *Algoritmo estimador de derivadas parciales de primer orden para N=2 (Operador rápido de Sobel)*

Con N=2 las mascaras para las derivadas parciales de primer orden son las mismas que las mascaras del gradiente de Sobel. El operador rápido de Sobel se ilustra en la figura: donde z1-1 y z2-1 denotan operaciones de retraso para los filtros columnas y filtros renglones respectivamente.

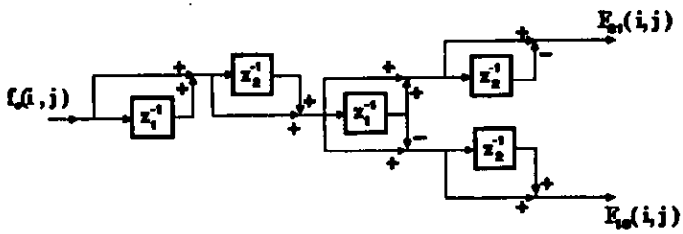


Imagen tomada de [M.Hashimoto, J. Sklansky,1987]

Por cada operación de retraso denotado por z1-1, un renglón de localidades de memoria es requerido. Este algoritmo se genera eliminando las partes innecesarias del algoritmo para N=2 y revirtiendo el orden del ultimo subfiltro para la operación columna y el primer subfiltro para la operación renglón.

Caso 2 *Algoritmo de estimación de derivadas parciales de segundo orden para N> 2 (Operador rápido Laplaciano)*

La siguiente figura muestra el operador Laplaciano el cual es la aproximación discreta al Laplaciano del filtro Gaussiano.

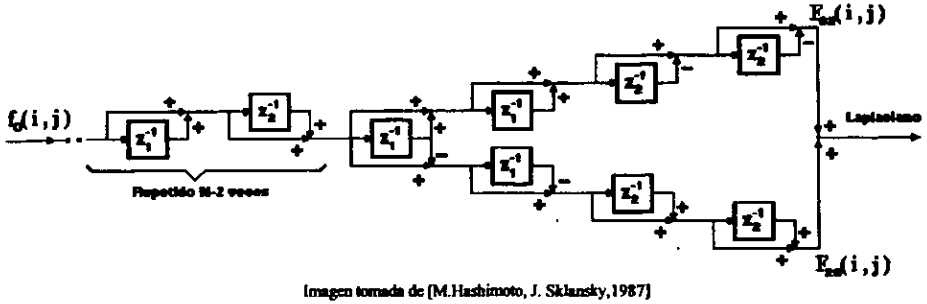


Imagen tomada de [M.Hashimoto, J. Sklansky,1987]

Este algoritmo requiere 2N+5 sumas por pixel con N+1 renglones extras de localidades de memoria.

Caso 3 *Algoritmo para estimación de derivadas de primer y segundo orden para N=2*

En la figura se muestra como estimar 5 derivadas parciales.

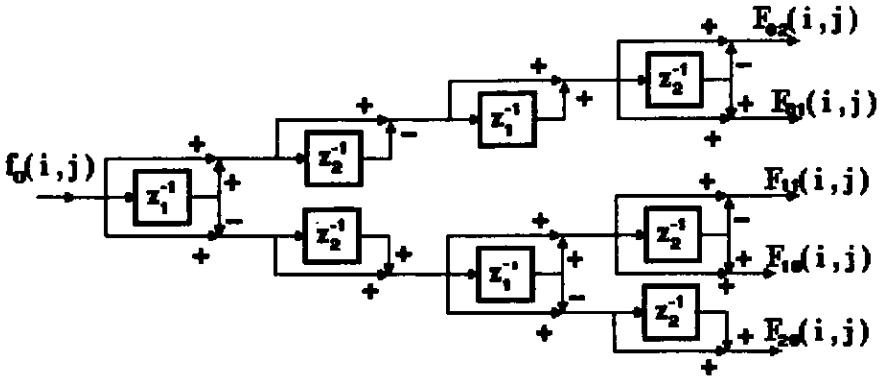


Imagen tomada de [M.Hashimoto, J. Sklansky,1987]

La información obtenida es la misma que la obtenida con la convolución por las mascaras de 3 x 3. Este algoritmo solo requiere de 12 sumas por pixel y tres localidades de memoria extra.

3.2 CASO GENERAL

3.2.1 El caso general como algoritmo propuesto.

Un algoritmo rápido de tamaño arbitrario y alguna opción de derivadas parciales pueden ser implementado de manera parecida.

El algoritmo rápido propuesto para el cálculo de todas las derivadas parciales se ilustra en la siguiente figura:

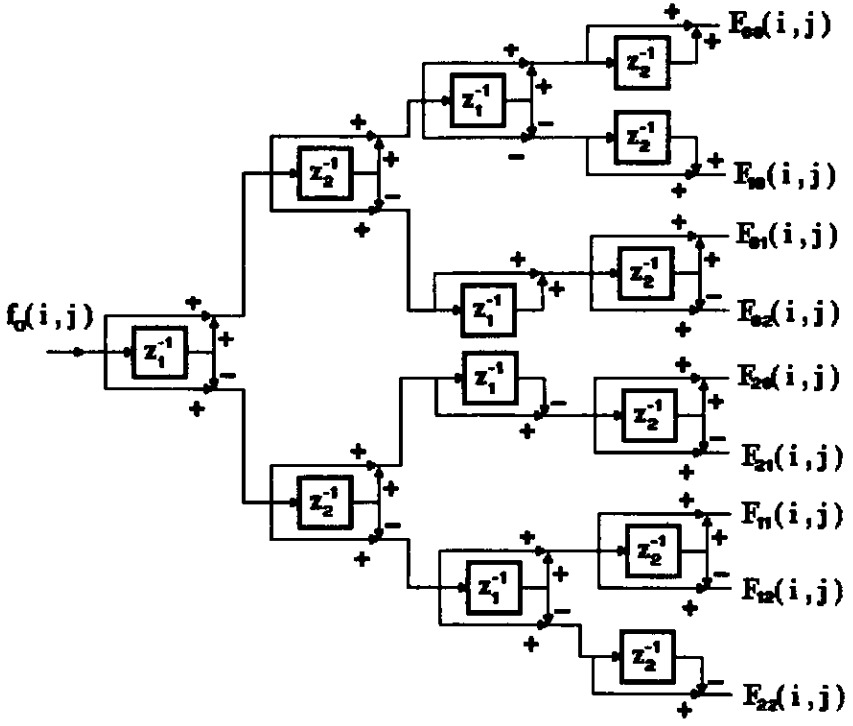
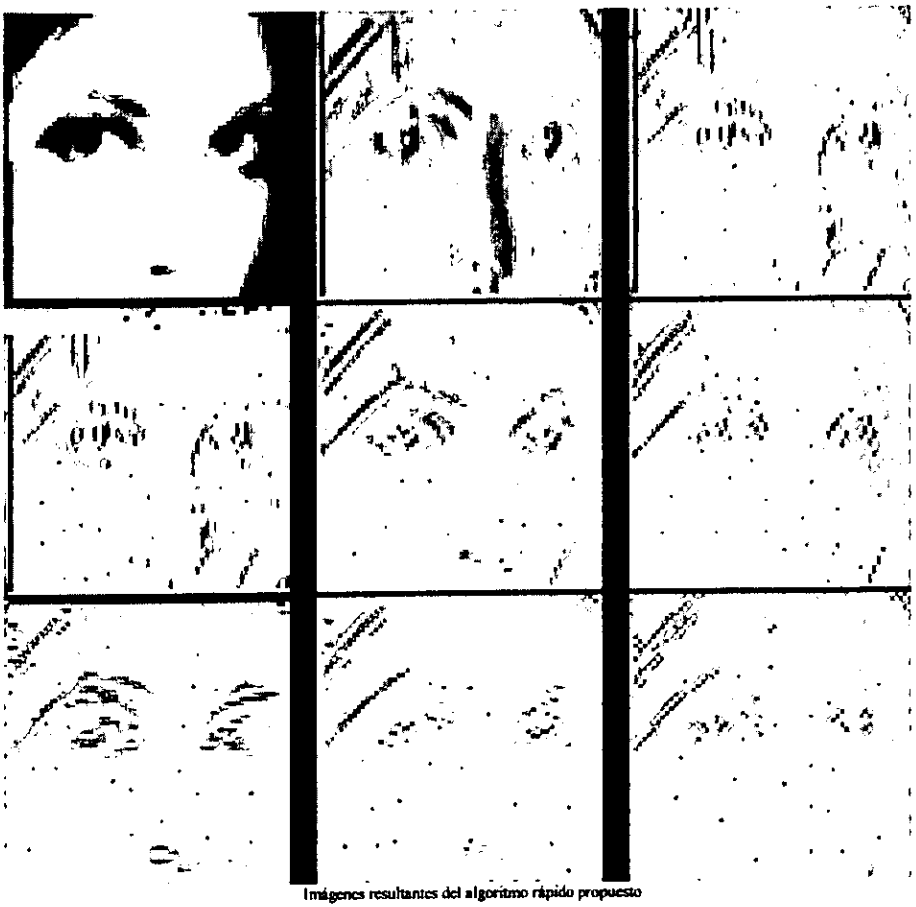


Imagen del algoritmo rápido propuesto

La siguiente figura muestra las derivadas parciales obtenidas de aplicar el algoritmo del caso general a una imagen. Por lo cual, son las imágenes resultantes de la Transformada Polinomial de 2 Dimensiones.



Imágenes resultantes del algoritmo rápido propuesto

Haremos referencia a cada una de las imágenes de arriba, usando como subíndices los números de la tabla siguiente, relacionándolos con la posición de cada imagen de la figura anterior.

00	01	02
10	11	12
20	21	22

3.3 RESULTADO DE COMPARACIONES

3.3.1 Comparación de diferentes algoritmos.

La siguiente tabla muestra el número de ciclos de reloj que ocupan diversos microprocesadores en realizar las operaciones de adicción y multiplicación, requeridas por el algoritmo propuesto:

Procesador	Adición	Multiplicacion
HC11	3	10
8086	9	83
80x286	7	16
80x386	6	17
80x486	2	18

La siguiente tabla realiza una comparación de los recursos de computo necesarios para realizar diversas operaciones, utilizando el método tradicional de convolución y el método propuesto.

Operación	Tradicional	Algoritmo rápido o propuesto
Unidimensional	(N+1) multiplicaciones N sumas	(N+1) $\log_2$ (N+1) sumas (N+1) $\log_2$ (N+1)-N localidades de memoria
Sobel	$(N+1)^2$ multiplicaciones N(N+1) sumas	6 Sumas 2 Localidades de memoria
Laplaciano	$(N+1)^2$ multiplicaciones N(N+1)sumas	2N+5 Sumas N+1 Loclidades de memoria
9 Derivadas parciales (N=3)	81 multiplicaciones 54 sumas	12 Sumas 8 Localidades de memoria

Comparación de algoritmos.

La siguiente tabla muestra, para N=2, la cantidad de operaciones necesarias para realizar una operación, las 9 operaciones, y la cantidad de ciclos de reloj que llevaría realizar las 9 operaciones (calculo de todas las derivadas parciales), con ambos métodos, el tradicional y el propuesto.

	Una operaciones	Las 9 operaciones	total de ciclos
Una dimensión	5 sumas	20 sumas	40
2 dimensiones	----------	21 sumas	42
Tradicional	9 multiplicaciones 6 sumas	81 multiplicaciones 54 sumas	1566

Comparación de cantidad de operaciones requeridas

Para tener una idea suponiendo una maquina que trabaje a 50 Mhz, 135Mhz y 166 Mhz.

Velocidad	Método	Tiempos (en segundos)			
		Un pixel	Imagen de 128x128	Imagen de 256 x256	Imagen de 512x 512
50 Mhz	*Rápido*	8e-7	0.0131072	0.0524288	0.2097152
	Directo	3.132e-5	0.51314688	2.05258752	8.21035008
133 MHz	*Rápido*	3e-7	0.004927	0.01971	0.07884
	Directo	1.177e-5	0.19291	0.77165	3.0866
166 MHz	*Rápido*	2.41e-7	0.00394	0.01579	0.06317
	Directo	9.433e-6	0.15456	0.618249	2.473

Como se puede observar hay un gran beneficio de tiempo al aplicar el método propuesto. Si tomamos como ejemplo el procesador de 133 Mhz, para la imagen de 512 x 512, nos indica que podemos procesar 12.6 imágenes por segundo y si las imágenes fueran de 256 x 256 se podrían procesar 50 imágenes por segundo, lo que nos lleva a pensar en procesar vídeo en tiempo real.

Para poder mostrar una secuencia a 30 cuadros por segundo requerimos que cada cuadro a mostrar se tarde no mas de 0.0333333 segundos, para 24 cuadros por segundo el tiempo requerido debe ser de 0.0416666 segundos máximo. Continuando con nuestro ejemplo del procesador de 133 Mhz, los tiempos que tenemos son de:

	Dimensiones de las imágenes		
	160x120 pixeles	320x240 pixeles	640x480 pixeles
133Mhz	0.0001156 seg	0.0185 seg	0.0740 seg

Algoritmo propuesto con imágenes no cuadradas.

Por lo que podemos observar podemos trabajar en tiempo real con imágenes de hasta 320x240 pixeles. Si deseáramos trabajar con imágenes de 640x480 requeriríamos de un procesador de al menos 295 MHz.

3.4 PARALELIZACIÓN DEL SOFTWARE

3.4.1 Código del algoritmo propuesto.

El algoritmo para implementar la transformada Hermitiana, se codificó en lenguaje C (Apéndice IV del Código Fuente). Y el código del software obtenido tiene opciones de optimización, para cuando se ejecutar en una arquitectura en paralelo.

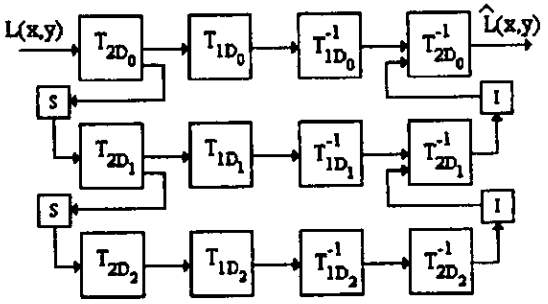
Aunque la implementación del procesamiento en paralelo del software se pueden hacer de diversas formas, en particular el procesamiento de las imagenes con el algoritmo

propuesto de la Transformada Polinomial en tiempo real, nos es posible paralelizar en diversas formas, tales como:

- Por imagen obtenida.
- Por cada nivel de procesamiento de la imagen.
- Por cada transformación y antitransformación

Pero un paralelización entre segmentos de imagenes es dificil de implementar además de que modificaría el algoritmo para ese tipo de adaptación, esto se debe a que en el mismo algoritmo hay una dependencia entre los resultados obtenidos del proceso anterior. Y por otro lado, se encontró que la paralelización del código por imagen obtenida resulta ser la más fácil de implementar y no afecta al algoritmo en el procesamiento de la imagen. Además de que el tiempo de procesamiento de las imagenes en esta forma resulto ser muy bueno.

En el siguiente diagrama se puede ver el flujo del procesamiento de la imagen con 3 niveles, implementado en el software con el algoritmo propuesto.



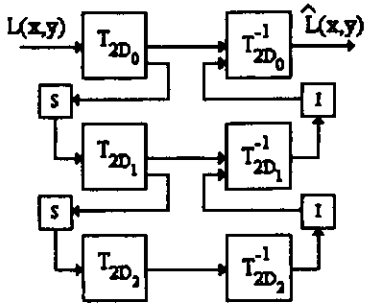
Disgrama de bloques, del procesamineto de 3 niveles con Transformada Polinotial de dos dimensiones y una dimension.

Siendo los elementos de este diagrama los siguientes:

- L(x,y) es la imagen original de entrada y $\hat{L}(x,y)$ es la imagen resultante del procesamiento.
- T_{2D} es la Transformada Polinomial de 2 Dimensiones.
- T_{1D} es la Transformada Polinomial de 1 Dimensión.
- T^{-1}_{2D} es la Antitransformada Polinomial de 2 Dimensiones.
- T^{-1}_{1D} es la Antitransformada Polinomial de 1 Dimensión.
- S tiene como entrada la imagen a procesar en el nivel, la cual muestrea y dá como salida una imagen de ¼ en relación a la imagen que tiene de entrada.
- I tiene como entrada la imagen resultante del procesamiento del nivel, la cual interpola para dar como salida una imagen 4 veces más grande.

- Los subindicesse se refieren a los niveles de procesamiento los cuales varian de nivel a nivel por ¼ de la imagen procesada en el nivel anterior.

Como el código de procesamiento se encuentra organizado en secciones de código independientes, se puede tener un procesamiento de la imagen sin la Tranformada Polinomial de 1 Dimensión, y procesar la imagen con sólo la Transformada Polinomial de 2 Dimensiones, quedando el diagrama de bloques como se muestra a continuación:



Disgrama de bloques, del procesamineto de 3 niveles con Transformada Polinolial de dos dimensiones.

Siendo los mismos elementos que se tienen en el diagrama de bloques anterior y también en este caso se pueden tener los 3 niveles de procesamiento.

3.5 CONSIDERACIONES DEL HARDWARE

3.5.1 Hardware utilizado.

El software se probó en un equipo con arquitectura O2 (Ver Apéndice II de la WorkStation O2), y es factible su ejecución en ella, cumpliendo una muy buena perspectiva, es decir, el tiempo de ejecución del algoritmo rápido (algoritmo propuesto) puede procesar cerca de 30 imágenes por segundo, de 320 x 240 pixeles cada imagen.

Para su ejecución en una computadora como la Origin2000 (Ver Apéndice V de la Origin) sería necesario, hacer un procesamiento por lotes, por lo cual, los resultados no se tendrían en ese momento. Aunque se puede considerar que se justificaría su uso si las imágenes a procesar fueran de un tamaño bastante grande, y las cuales serían no apropiadas para ser procesadas en una computadora como la O2.

3.6 IMÁGENES OBTENIDAS

3.6.1 Imágenes obtenidas (Barbara procesada)

A continuación se muestra la imagen de entrada y las imágenes resultantes de un procesamiento tal como los mostrados, en los diagramas de bloques del punto de Paralelización del Software, de este capítulo.

Imagen Original	
Imagen resultante del procesamiento de la Transformada Polinomial 2 Dimensiones.	
Imagen resultante del procesamiento de la Transformada Polinomial de 2 Dimensiones y 1 Dimensión.	

A las imagenes resultates se les aplico los 3 niveles de procesamiento.

Las primeras dos imágenes son iguales si al procesar la imagen original se le aplica la Transformada y la Antitransforma Polinomial de 2 Dimensiones, siempre y cuando se usen las 9 imágenes resultantes, en caso de que no se usen las imágenes de las posiciones 12, 21 y 22. No se tendrá una importante perdida de información, como se puede ver en la imagen. Y en comparación, la última imagen es la resultante de aplicar la Transformada Polinomial de 1 y 2 Dimensiones con sus correspondientes Antitransformadas. En la cual se tiene perdida de información, pero se elimina redundancia y realce de la información más

importante, es decir, se encuentran los ángulos óptimos de los bordes para su mejor apreciación a la vista humana, así como también una disminución en la entropía, y con ello tener la imagen con un tamaño de palabra menor.

3.6.2 Cantidad de información (Entropía) y otros.

Es importante tomar en cuenta la Cantidad de Información, dado que al hacer los diferentes niveles de transformación, podemos ver si se tiene perdida de la información que se procesa. Aunque con la Transformada Polinomial, tenemos la posibilidad de agregar otros procesamientos, los cuales pueden mejorar la imagen respecto a la original, comprimirla, etc.

La Cantidad de Información o Entropía, se calcula con la siguiente fórmula:

$$\varepsilon = \sum_{i=0}^{255} P_i \log_2 \frac{1}{P_i}$$

Siendo P_i la probabilidad de cada uno de los pixeles desde 0 a 255, siendo la imagen en 255 niveles de gris.

Para calcular la entropía de cada uno de los niveles, se calcula con las cuatro imágenes resultantes de la Transformada Polinomial de 1 Dimensión, siendo esta últimas obtenidas del calculo resultante de las nueve imágenes resultantes de la Transformada Polinomial de 2 Dimensiones, del nivel a calcular. En este caso se realiza con tres niveles.

La Entropía para cada uno de los niveles, se muestra a continuación:

$$\varepsilon_N = \sum_{n=1}^{N} \frac{\varepsilon_{n0} + \varepsilon_{n1} + \varepsilon_{n2} + \varepsilon_{n3}}{4^n} - \sum_{n=1}^{N-1} \frac{\varepsilon_{n0}}{4^n}$$

Siendo *n* el nivel que a procesar. Por lo tanto, para cada uno de los niveles las ecuaciones son las siguientes:

$$\varepsilon_1 = \frac{\varepsilon_{10} + \varepsilon_{11} + \varepsilon_{12} + \varepsilon_{13}}{4}$$

$$\varepsilon_2 = \frac{\varepsilon_{11} + \varepsilon_{12} + \varepsilon_{13}}{4} + \frac{\varepsilon_{20} + \varepsilon_{21} + \varepsilon_{22} + \varepsilon_{23}}{16}$$

$$\varepsilon_3 = \frac{\varepsilon_{11} + \varepsilon_{12} + \varepsilon_{13}}{4} + \frac{\varepsilon_{21} + \varepsilon_{22} + \varepsilon_{23}}{16} + \frac{\varepsilon_{30} + \varepsilon_{31} + \varepsilon_{32} + \varepsilon_{33}}{64}$$

Para obtener la Tasa de Compresión de cada uno los niveles, se calcula con la siguiente ecuación:

$$t_{C_n} = \frac{\varepsilon_o}{\varepsilon_n}$$

Siendo t_{C_n} la Tasa de Compresión en el nivel n, ε_0 es la Entropía en la imagen original, y ε_n es la Entropía del nivel a calcular. Por lo cual se obtiene primero los valores de la entropía de cada uno de los niveles.

Y la Eficiencia de cada uno de los niveles se calcula con la siguiente ecuación:

$$E_n = 1 - \frac{1}{t_{C_n}}$$

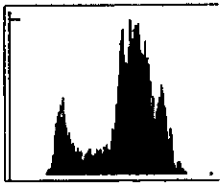
Como se puede ver en la ecuación, se necesita haberse calculado la Tasa de Compresión del nivel con anterioridad.

La siguiente tabla muestra los valores obtenidos para la entropía, la eficiencia y la tasa de compresión, para la imagen original y las imágenes obtenidas de aplicar la transformada Hermitiana a distintos niveles.

	Entropía ε_N	Eficiencia	Tasa de compresión
Imagen original	7.070268	0	1
Imágenes resultantes de la Transformada Polinomial en 1er nivel.	ε_1 = 3.769540	0.466846	1.875634
Imágenes resultantes de la Transformada Polinomial en el 2o nivel.	ε_2 = 3.0016363125	0.575456	2.355475
Imágenes resultantes de la Transformada Polinomial en el 3er nivel.	ε_3 = 2.816552078125	0.601635	2.510260

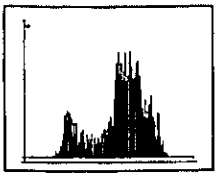
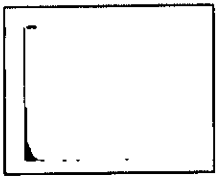
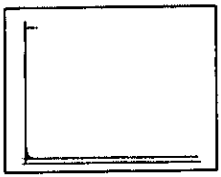
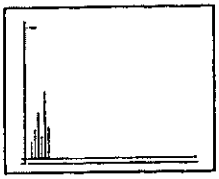
3.6.3 Histograma y Cantidad de información (Entropía)

Como la imagen a procesar, se encuentra en 255 niveles de gris se puede hacer el histograma de ella, la cual muestra en que tonalidades dominan en la imagen. La siguiente gráfica muestra el histograma de los niveles de grises de la imagen original.

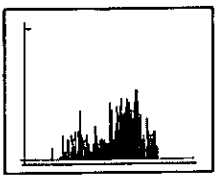
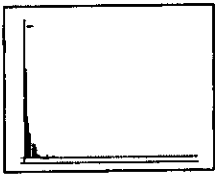
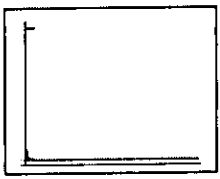
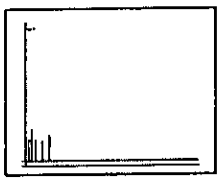


Las siguientes gráficas muestran el histograma en niveles de grises, de las imágenes resultantes de aplicar la Transformada Polinomial de 1 Dimensión, pasando por la Transformada Polinomial de 2 Dimensiones, en cada uno de los 3 niveles.

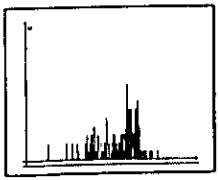
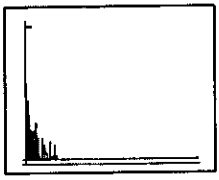
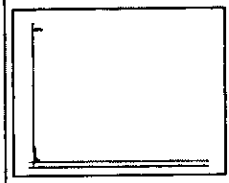
El primer subíndice nos indica el nivel de procesamiento, y el segundo subíndice se refiere a la imagen de ese nivel, siendo en particular la **0** una muestra de la imagen original y la imagen **4** que sólo tiene 8 valores, son los ángulos óptimos de la imagen en las posiciones que ocupan respecto a la imagen.

1er Nivel			
Imagen $_{10}$	Imagen $_{11}$	Imagen $_{12}$	Imagen $_{13}$
$\varepsilon_{10} = 7.043668$	$\varepsilon_{11} = 3.381758$	$\varepsilon_{12} = 1.834584$	$\varepsilon_{13} = 2.818150$
			

La entropía de imágenes resultante en el 1er. Nivel es $\varepsilon_{10} + \varepsilon_{11} + \varepsilon_{12} + \varepsilon_{13} = 15.07816$
Y la entropía de ε_1 es: $\varepsilon_1 = 3.76954$

2er Nivel			
Imagen $_{20}$	Imagen $_{21}$	Imagen $_{22}$	Imagen $_{23}$
$\varepsilon_{20} = 6.921928$	$\varepsilon_{21} = 4.208489$	$\varepsilon_{22} = 1.949825$	$\varepsilon_{23} = 2.807971$
			

La entropía de imágenes resultantes en el 2o. Nivel es $\varepsilon_{20} + \varepsilon_{21} + \varepsilon_{22} + \varepsilon_{23} = 15.888213$
Y la entropía de ε_2 es: $\varepsilon_2 = 3.0016363125$

3er Nivel			
Imagen $_{30}$	Imagen $_{31}$	Imagen $_{32}$	Imagen $_{33}$
$\varepsilon_{30} = 6.275361$	$\varepsilon_{31} = 4.695192$	$\varepsilon_{32} = 1.968882$	$\varepsilon_{32} = 2.902886$
			

La entropía de imágenes resultantes en el 3er. Nivel es $\varepsilon_{30} + \varepsilon_{31} + \varepsilon_{32} + \varepsilon_{33} = 15.842321$
Y la entropía de ε_3 es: $\varepsilon_3 = 2.816552078125$

3.7 EFICIENCIA EN LA EJECUCIÓN

La siguiente tabla muestra los valores obtenidos de la cantidad de instrucciones necesarias para la realización la transformada polinomial a distintos niveles, para una imagen de 240 x 320 pixeles (76,800 pixeles).

	Un nivel	Dos niveles	Tres niveles
Transformada de 2 Dimensiones, número de instrucciones.	2,906,652	3,635,224	3,818,356
Transformada de 1 Dimensión, número de instrucciones.	4,838,499	6,048,198	6,653,109
Anti –Transformada de 2 Dimensiones, número de instrucciones.	5,473,687	6,832,994	7,168,311
Anti –Transformada de 1 Dimensión, número de instrucciones.	2,803,284	3,504,168	3,679,452
Total de instrucciones usados en la transformación y anti-transformación.	16,022,122	20,020,584	21,319,228
Total del número de ciclos	16,259,262	20,262,997	21,260,714
Total de tiempo de ejecución	0.0903s	0.11257s	0.11812s
Razón de ciclos/instrucción	1.000	1.000	1.000
Razón del reloj	180	180	180
Cantidad de instrucciones entre la cantidad de pixeles de la imagen original	208.62	263.84	277.59

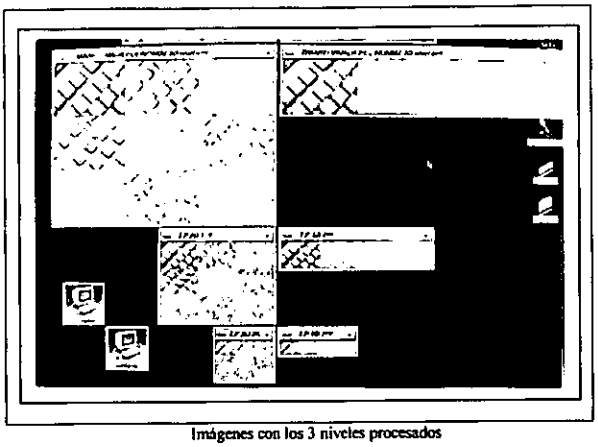
Lo cual nos permite observar que la cantidad de instrucciones necesarias, para cada pixel, no variará drásticamente en la práctica. Además se comprobó en otras pruebas que estos valores se mantiene prácticamente para imágenes de diferentes tamaños. Lo cual implica que tiende a ser lineal.

Estas pruebas se realizaron en una WS Silicon Graphics a 180MHz con sistema operativo IRIX 6.3, con una memoria de 64 MB.

3.8 RESULTADOS EN TIEMPO REAL

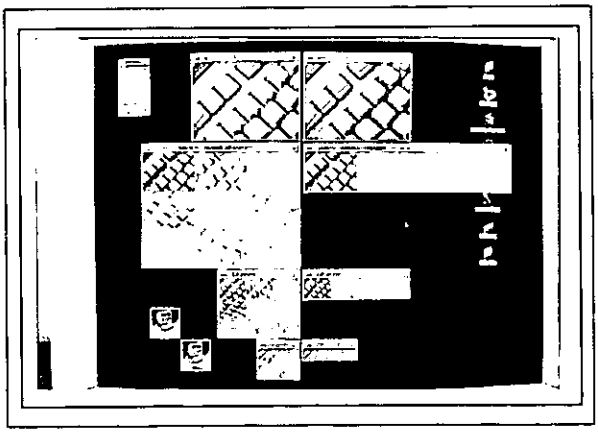
Como el software programado se ejecuta en una WS Silicon Graphics, se puede apreciar por las siguientes imágenes, los resultados desplegados en tiempo real por dicho equipo de computo, haciendo uso de la misma cámara de vídeo que cuenta.

En la siguiente imagen se puede ver el teclado de la computadora, del cual se puede ver las imágenes resultantes de los 3 niveles procesados. A la izquierda se encuentra los resultados de la Transformada Polinomial de 2 dimensiones y a la derecha los resultados de la Transformada Polinomial de 1 dimensión. Y de arriba abajo, el primer, segundo y tercer nivel de procesamiento.



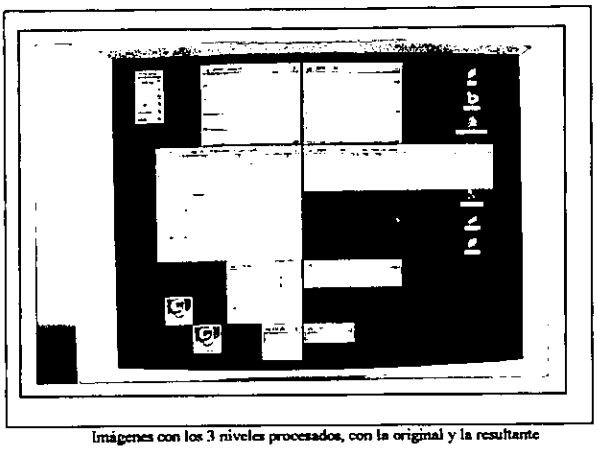
Imágenes con los 3 niveles procesados

Como se puede ver en la imagen anterior, las líneas diagonales se detectan según el filtro que este aplicando la imagen. Dichas características se pueden apreciar mejor, en la siguiente imagen, donde también se puede apreciar las letras del teclado.



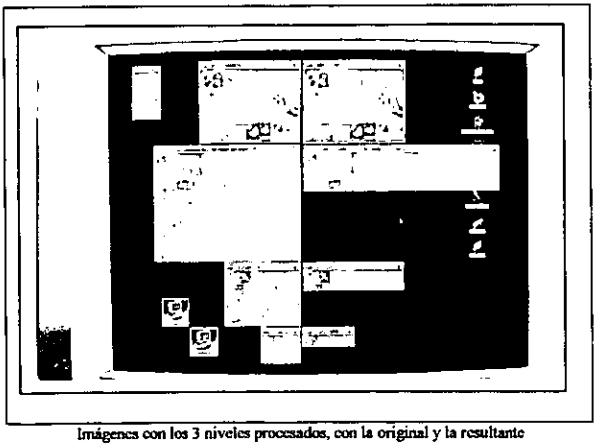
La imagen superior izquierda es la captada por la cámara de vídeo de la computadora y la imagen superior derecha es la imagen resultante de realizar los 3 niveles de procesamiento de la Transformada Polinomial.

Es interesante ver el procesamiento resultante de líneas horizontales y verticales, como se puede ver en la siguiente imagen. Donde se puede ver la detección de varias características de la imagen.



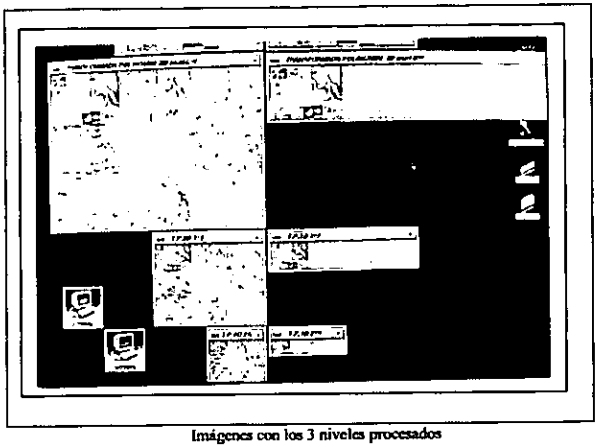
Imágenes con los 3 niveles procesados, con la original y la resultante

La siguiente imagen muestra el procesamiento de la imagen superior izquierda y la imagen superior derecha que es la resultante. En esta imagen se está procesando es el tapete del ratón de la computadora, teniendo esta varias formas y figuras.



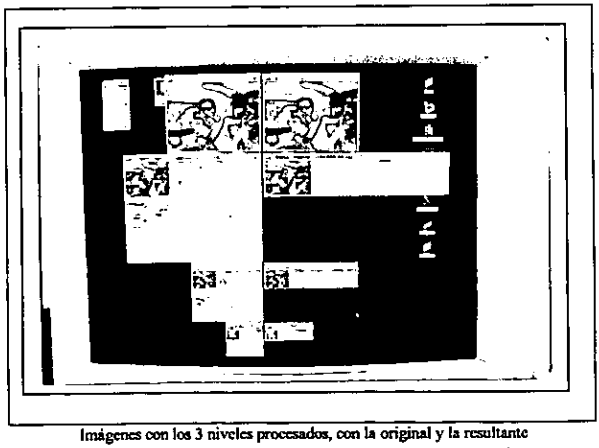
Imágenes con los 3 niveles procesados, con la original y la resultante

Un acercamiento de la imagen anterior nos muestra mejor las imágenes resultantes del procesamiento realizado con la Transformada Polinomial de 2 Dimensiones y por la Transformada Polinomial de 2 Dimensiones.



Imágenes con los 3 niveles procesados

En la siguiente imagen se puede apreciar la cámara fotográfica que se utilizó para estas fotos manejada por el Dr. Boris Escalante. La cual nos permite corroborar que el procesamiento se está realizando en tiempo real. Además en la imágenes resultantes del procesamiento, se pueden ver ciertas características del laboratorio, las cuales no resaltan a simple vista, pero se puede ver que el procesamiento distingue los bordes de ciertas figuras que no son nítidas en su totalidad.



Imágenes con los 3 niveles procesados, con la original y la resultante

Y en la siguiente imagen también se corrobora que el procesamiento se está realizando en tiempo real. A un lado de la pantalla, sobre el CPU de la computadora se encuentra la cámara de vídeo, la cual capta las imágenes, luego son procesadas por la computadora, y finalmente se despliegan en la pantalla.

Imágenes con los 3 niveles procesados, con la original tomada de la cámara de vídeo y la resultante

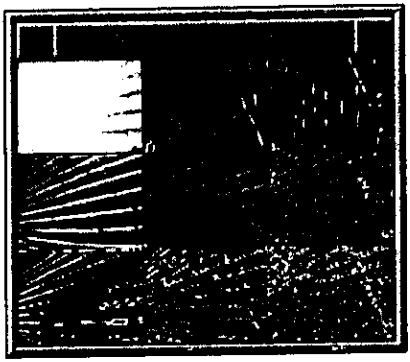
El tiempo de procesamiento es satisfactorio en dicho equipo de computo, pero el desplegar varias imágenes producen ciertos tiempos de despliegue, dependiendo de la cantidad de imágenes que se desplieguen, dado que estamos utilizando más recursos, en sólo ver las imágenes.

Se ha probado el algoritmo sin tener que desplegar ninguna imagen, lo cual ha permitido ver que el procesamiento es de alrededor de 20 a 30 imágenes por segundo, dicho procesamiento es considerando solamente una imagen, siendo esta en teoría monocromática.

Pero al desplegar se tienen que correr varios algoritmos, dado que la imagen capturada con la cámara es de color y se encuentra en formato RGB, es por ello que se encontramos las siguientes particularidades para lograr un despliegue de las imágenes resultantes del procesamiento y de sus resultados:

- Transformar la imagen en formato RGB al formato YIQ (Apéndice III de Video Library), esto se realiza porque las componentes RGB tienen información de interés las 3 componentes, implicando un procesamiento de las 3, es por ello que al pasarlas al dominio YIQ, es sólo necesario procesar la componente "Y" dado que tiene la mayor cantidad de información respecto a las otras componente, y es por ello las componentes "I" e "Q" se guardan para su uso posterior. Por lo cual la componente "Y" se guarda en un vector para su procesamiento.
- Preparación de imágenes para su despliegue, como al procesar se obtienen varias imágenes resultantes, que se encuentran cada una de ellas en una en un vector, es necesario pasar estos vectores una matriz para su despliegue, como si se tratase de una sola imagen. Esto se realiza tanto para cada nivel como para cada Dimensión que se desee visualizar en pantalla. Además de que en el caso particular de la última imagen resultante de la Trasformada Polinomial de 1 Dimensión se multiplican sus valores para que se puedan ver en la pantalla, dado que desde el punto de vista práctico no es necesario dicha multiplicación.

- Transformar la imagen procesada al formato RGB (Apéndice III de Video Library), esto se realiza con la componente "Y" resultante del procesamiento, y usando las componentes almacenadas "I" e "Q" para construir la imagen en el formato RGB, dado que para desplegar los colores es necesario que se encuentre en dicho formato.
- Despliegue de las resultantes en tonalidades de gris, la cual se logra retomando los valores de los vectores resultado de las cada nivel de transformación y después de arreglan en matrices, para que se tengan ordenados. Como estos valores corresponden a valores en el dominio de YIQ, en particular, la componente "Y", se usan los valores "I" e "Q" para pasarlos al dominio RGB, en los cuales, las componentes "R" y "B" son los valores obtenidos en "G", para que se tenga la imagen en tonalidades de gris.



Imágenes de un nivel de procesamiento de la Transformada Polinomial de 2 Dimensions

Imágenes de un nivel de procesamiento de la Transformada Polinomial de 1 Dimensión

CAPITULO IV
CONCLUSIONES

4.1 ANÁLISIS

4.1.1 La factibilidad del procesamiento de vídeo con la Transformada Polinomial en tiempo real.

En un despliegue de vídeo es deseable tener un rango de 20 a 30 imágenes o frames por segundo para ofrecer una buena calidad de movimiento continuo. En la aplicación que se desarrolló, es posible tener este rango de imágenes procesadas por segundo en la computadora O2 (Apéndice II de la WorkStation O2), aprovechando el despliegue en la pantalla, para apreciar los resultados intermedios y finales del procesamiento. Existe un retardo en el tiempo al desplegar en el monitor las imágenes. Esto se debe, a que la parte de despliegue se realiza con X Window (Apéndice I sobre X Window) y la captura de las imágenes con librerías de vídeo (Apéndice III de Video Library), y no directamente como lo hace la aplicación de Silicon, la cual es proporcionada con la cámara de vídeo de la computadora, y que no realiza ningún procesamiento de las imágenes.

Al considerar que se toman las imágenes de la cámara de vídeo, para luego procesarlas y después desplegarlas, se puede intuir que se necesita gastar más tiempo en ello. Esto se corrobora al hacer pruebas con respecto al tiempo de procesamiento, en las cuales se verifica, que el hecho de desplegar las imágenes provoca una perdida de tiempo. Apreciando directamente el procesamiento ante la cámara es posible tener una mejor idea de todo el procesamiento, pues se puede ver tanto la imagen tomada por la cámara, como la imagen ya procesada, así como también se pueden apreciar todos los niveles del procesamiento, en una dimensión y dos dimensiones al mismo tiempo. Con esto desplegamos la información importante de la imagen, que puede ser necesaria para la realización de otras aplicaciones.

4.1.2 Paralelización en tiempo real.

En nuestro caso particular, el código de la aplicación no está paralelizada, dado que no fue necesario por la velocidad en la que se realiza el procesamiento. Si se realizara un procesamiento en paralelo, consideramos que sería posible hacer una paralelización por imágenes, es decir, procesar cada imagen como un todo y no por segmentos de ella. El código ha sido optimizado y probado, en el sentido de que es lo suficientemente rápido y pequeño para que pueda ser ejecutado por un solo procesador. Con el cual, se puede llegar a procesar de 15 a 20 imágenes por segundo, siempre y cuando sea nuestra aplicación la única que está utilizando el procesador, y no se esté desplegando después de procesar la imagen. En el caso, de que contara con varios procesadores obtendríamos un procesamiento de las imágenes dentro del rango de 20 a 30 imágenes por segundo, aunque la cámara no nos podría proporcionar más de 30 y también tendríamos la misma limitación del despliegue de las imágenes. Así también, es importante mencionar que no se está haciendo una lectura o escritura en el disco, pues todo es realizado en la memoria de la computadora, y es por ello que es posible, hacer este procesamiento en

un buen tiempo. Se podría considerar la posibilidad de paralelizar mediante fragmentación de las imágenes si éstas fueran mayores que las utilizadas en la presente aplicación, pues el tiempo de procesamiento depende linealmente también del tamaño de la imagen.

4.1.3 Procesamiento en el dominio del espacio.

El procesamiento es realizado imagen por imagen, sin tomar en cuenta la siguiente imagen proporcionada por la cámara, dado que esto implicaría hacer un estudio respecto a la correlación que existe entre una y otra imagen, con el fin de procesar únicamente los segmentos de la imagen que cambian con respecto a la anterior o anteriores, sin descartar el resultado del procesamiento anteriormente obtenido.

Dado el tiempo de procesamiento que hemos logrado en este trabajo, para cada imagen de la secuencia, no es necesario explotar la correlación temporal entre imágenes. Sin embargo, en aplicaciones más complejas esta característica permitiría reducir el tiempo de procesamiento mediante algoritmos de estimación y compensación de movimiento.

4.2 APLICACIONES

4.2.1 Compresión y realce de vídeo en tiempo real.

El procesamiento que se realiza en la aplicación es similar al necesario para poder hacer aplicaciones de compresión y realce, que usan la transformada polinomial. Por lo tanto, esto nos induce a la reflexión de que la compresión y realce de imágenes de vídeo en tiempo real, es factible, al igual que otras posibles aplicaciones que usen la transformada polinomial. Es por ello que esta aplicación, a un nivel de software está logrando lo que en otros casos nos haría pensar que sólo con hardware se lograría. Aunque hay que considerar que se está haciendo uso de una computadora O2 (Apéndice II de la WorkStation O2), en la que la entrada de vídeo en estado normal es muy buena y no se trata de una PC, donde se les integra comúnmente una tarjeta y una cámara para apreciar una imagen capturada por la cámara en tiempos "grandes" y de tamaño pequeño.

Consideramos que esta aplicación es la mejor demostración de que el uso de la transformada polinomial, en su versión rápida y optimizada, permite muy buenos resultados y que nos abre la posibilidad de hacer análisis de imágenes en el tiempo. Este trabajo es, sin embargo, el inicio de una investigación que permitirá a corto plazo alcanzar metas más ambiciosas. Por lo cual consideramos que este es una de las partes importantes que lograrán despegar al procesamiento de imágenes en movimiento, de las imágenes estáticas.

REFERENCIAS

[Anil, 1981] Anil K. Jain, ***Advances in Mathematical Models for Image Processing.*** Proceedings of the IEEE, Vol. 69, No 5. pp. 122-148 May 1981.

[M.Hashimoto, J. Sklansky,1987] M. Hashimoto and J. Sklansky. ***Multiple-Order derivaties for detecting local Image Characteristics.*** Computer vision, graphics and image processing, 39,1987,28-55

[Rosenfeld, Kak, 1982] A. Rosenfeld and A. C. Kak, ***Digital Picture Processing*** (2nd. Ed.), Academic Press, New York.1982

[M.Hashimoto, Sankar, J. Sklansky,1982] M. Hashimoto, P.V. Sankar, and J. Sklansky, ***Detecting the edges of lung tumors by classification techinques,*** in Proceedings, 6th International Conference on Pattern Recognition, Oct 1982, 276-279

[Modestino, Fries, 1977] J. W. Modestino and R. W. Fries, ***Edge detection in noisy images using recursive digital filtering.*** Comput Graphics Image Process. 6, 1977, 409-433.

[Marr, Hildreth, 1980] D. Marr and E. Hildreth, ***Theory of edge detection,*** Proc. Roy. Soc. London Ser. B 207, 1980, 187-217

[Martens. 1990] Jean Bernard Martens, ***The Hermite Transform-Theory.*** Transaction on acoustics, speech and signal processing, vol,38, No 9, Sep 1990. pp 1595-1606

[Stollnitz, DeRose, Salessin, 1995] E. J. Stollnitz, T. D. DeRose and D. H. Salesin. ***Wavelets for Computer Graphics: A Primer, Part 1.*** IEEE Computer Graphics and Applications, May 1995. pp 76-84

[Escalante, Martens, 1992] Boris Escalante Ramírez, Jean Bernard Martens. ***Noise Reduction in Computerized Tomography Images by Means of Polynomial Transforms.*** Journal of Visual Communication and Image Representation, vol. 3, No 3, Sep 1992, pp 272-285.

[Viveros, Escalante, 1997] Oscar Viveros Cancino and Boris Escalante Ramírez. ***Parametric Image Coding by means of Plynomial Transforms.*** SPIE, Visual Communications and Image Processing '97, Proceedings, vol 3024, Feb 12-14 1997 pp 863-872.

[Escalante, López, 1996] Boris Escalante Ramírez and Juan R. López Miranda. ***Edge-orientation-based noise-reduction with polynomial trransforms.*** SPIE, Applications of Digital Image Processing XIX, vol 2847, Ago 7-9 1996 pp 280-291.

[Camarillo, Varela, Escalante, 1998] Patricia Camarillo Sandoval, Alma Varela López and Boris Escalante Ramírez. ***Adaptive multiplicative-noise reduction in SAR images with Polinomial Transforms***. IGARSS '98, IEEE Geoscience and Remote sensing society, Proceedings, Jul 1998 pp 1171-1173.

[O2 WorkStation] Silicon Graphics [en línea]: *O2 **WorkStation Hardware Reference Guide.*** <http:// techpubs.sgi.com/ library/dynaweb_bin/0630/ bin/ nph-dynaweb.cgi/ dynaweb/ SGI_EndUser/ 02_0G/ @Generic_BookTextView/ 95> [Consulta: 2000].

[Programming Guide]. Silicon Graphics [en línea]: *IRIS Digital Media Programming Guide.* <http:// techpubs.sgi.com/ library/dynaweb_bin/0530/ bin/ nph-dynaweb.cgi/ dynaweb/ SGI_Developer/ DmediaDev_PG/ @Generic_BookTextView/ 14729> [Consulta: 2000].

APÉNDICE

A. X Window

A.1 INTRODUCCIÓN

El sistema X Window (o simplemente X), proporciona una plataforma de desarrollo de Interfaces Gráficas de Usuario (GUI's) basadas en ventanas. Es un estándar aceptado por la industria para las estaciones de trabajo, lo que quiere decir que no pueda correr en otros sistemas operativos y deferentes plataformas.

Comencemos por hacer un poco de historia acerca de X Window:

1984	El Laboratotio para Ciencias de la Computación del Instituto Tecnológico de Massachusetts (MIT) en Cambridge, con el apoyo de Digital Equipment Corporation y otras compañias inició el proyecto Athena encabezado por Robert Scheifler y Jim Gettys basándose en el sistema de ventanas de la Universidad de Stanford denominada W. El objetivo concreto consistía en hacer que los programas estuvieran disponibles para los usuarios sobre cualquier estación en cualquier lugar del campus.
1986	La primera versión disponible para los diseñadores hace su aparición denominándose X Window 10.4.
1987 septiembre	Se libera la versión 11 en su release 1. X11 ofrece mejoras importantes en los aspectos de soporte de características de despliegue, estilos de manejo de ventanas, soporte de ventanas nultiples y por supuesto mejor rendimiento.
1988 marzo	Se libera X11R2. En esta versión el control de X se pasa del MIT al Consorcio de X, una asociación de los principales fabricantes de computadoras que soportan el estándar X Window, casas de desarrollo de software, universidades y miembros asociados.
1989 febrero	Se libera la versión X11R3.
1990 enero	Se libera la versión X11R4.
1991 agosto	Se libera la versión X11R5.

X fue diseñado para resolver cierta problemática. A continuación se presentan los objetivos de diseño de X Window.

- X fué diseñada para proveer de ventanas a las terminales *bitmapped* (cada pixel corresponde a un conjunto de bits en memoria), con gran portabilidad entre diferentes sistemas operativos sin sacrificar eficiencia.
- X fué diseñado para permitir la cooperación de diferentes arquitecturas en una red. Para ello se diseño el protocolo X que establece las reglas de comunicación a través de la red.
- X no implica un estilo particular de interfase de usuario. Para ello X crea la "libre politica" para diseñar la interfase de usuario según lo requiera la aplicación.
- Para los usuarios y las aplicaciones la red es transparente. Cuando se establece una conexión hay programas corriendo en ambos lados estableciendo un modelo cliente servidor.

A.2 CARACTERÍSTICAS DE X WINDOW

Ahora que conocemos un poco de su historia y sus objetivos, podemos imaginarnos la importancia y complejidad del Sietma X Window. Sin embargo se basa en unas cuantas premisas básicas fáciles de entender que se describen a continuación.

A.2.1 Despliegues y pantallas

El sistema X Window es un sistema de ventanas para sistemas gráficos *bitmmaped* que son aquellos que manejan memoria de video y en ella cada pixel se representa por un conjunto de bits.

Un *despliegue* se define como una estación de trabajo con teclado, un dispositivo apuntador como lo es un ratón y una o más *pantallas*. Varias pantallas pueden trabajar juntas, con el puntero del ratón cruzando la frontera de una pantalla para hacerse visible en la otra trabajando con el mismo teclado.

A.2.2 Modelo cliente-servidor

Tradicionalmente, el modelo Cliente-Servidor representa al cliente como estado local y el servidor como estado remoto. X Window toma de manera inversa esta notación, por ejemplo podemos tener varios clientes corriendo en diferentes máquinas, uno es una supercomputadora, otros en varias estaciones de trabajo y algún otro tal vez es una

minicomputadora, pero todos con una salida gráfica a una sóla pantalla que representaría al servidor.

Servidor X Window.

- Se describe al servidor como el programa que controla el despliegue.
- Un Servidor X Window puede ser una estación de trabajo sin disco, una estación de trabajo completa o un periférico conectado a un sistema de grandes recursos.
- El monitor del servidor de X puede ser monocromático, de niveles de gris o a color.

Tareas del servidor.

- Permite el acceso al despliegue a los clientes.
- Interpreta los mensajes de los clientes a través de la red.
- Manda las entradas de usuarios a los clientes mediante mensajes a través de la red.
- Encargarse del proceso de graficación y dibujo en el despliegue, en vez de que lo haga el cliente.
- Mantiene las estructuras de datos complejas como lo son ventanas, cursores, fuentes, contextos gráficos como recursos que comparten los clientes refiriéndose a ellos por medio de ID's (Identificadores).

Cliente X Window

- Un cliente X Window es un programa de aplicación que necesita un servidor X para interactuar con un usuario.
- Un cliente X Window puede ser ejecutado en la misma computadora que el servidor, o en otra máquina. Esta carácterística permite el proceso distribuido.
- Lo principal de la comunicación entre clientes y servidores en máquinas separadas es la red. El consorcio de X soporta redes TCP/IP o DECnet.
- Los clientes y servidores en la misma máquina típicamente utilizan un sistema de IPC's locales (Interprocess Communication Channels), memoria compartida o sockets en UNIX para comunicarse, pero siempre usando el protocolo X.

A.2.3 Manejadores de ventanas

Los clientes no controlan directamente variables como el caso donde aparece una ventana o el tamaño de esta. Dado que puede existir multiprocesamiento y que muchos clientes pueden accesar el mismo despliegue, los clientes no pueden depender de una configuración particular. En su lugar los clientes solamente dan indicaciones indirectas de donde y de que tamaño desean su ventana. La política sobre el manejo de ventanas, el estilo de interacción de los usuarios con el sistema se dejan a programas llamados manejadores de ventanas (windows managers).

El *manejador de ventanas* es un cliente en el cual tiene autoridad sobre la distribución de las ventanas sobre la pantalla y puede ser visto como cualquier otro cliente. Dicho manejador de ventanas tiene derechos pero no responsabilidades, todos los clientes deben estar preparados para cooperar con cualquier tipo de manejador de ventanas y aun para funcionar sin la existencia de uno.

Existen muchos manejadores de ventanas y cada uno de ellos definen la ofrma de decorar las ventanas, de interactuar con los usuarios. Sin embargo la comunicación con los clientes debe ser igual para que las aplicaciones sean portables. Entre los más famosos en la actualidad están el de OSF/Motif *nwm*, el de OPEN-LOOK *olwm* y el manejador *twm* que siempre se trae con la distribución de X Window.

A.2.4 Eventos.

Como en cualquier sistema de ventanas manejado con ratón, un cliente X debe estar preparado para responder a cualquier tipo de eventos, que pueden ocurrir en cualquier orden y tiempo. Esta es la diferencia principal entre la programación X Window y la tradicional en UNIX o PC, ya que los clientes reciben eventos y toman una acción de acuerdo al evento generado.

Los eventos se generan como
- Presión de alguna tecla.
- Presión de algún botón del ratón.
- Movimiento del ratón.
- Etc...

A.3 PROTOCOLO X

A.3.1 Capas funcionales en X Window

X Window tiene varias capas de programación, cada una con cierta complejidad y metodología, pero en sí, todas pretenden el intercambio de datos entre el servidor y el cliente vía el protocolo X. Veamos un poco de cada una de estas capas comenzando desde la más baja y básica: El protocolo X.

A.3.2 El Protocolo X

Características.

El protocolo X especifica qué representa cada paquete de información que es transferido entre el servidor y el cliente en cualquier dirección. Si ambos se encuentran corriendo en la misma máquina la comunicación se realizará por medio de algún canal

interno. Permite la construcción de cualquier tipo de interfase de usuario basándose en un protocolo de red asíncrono. Esto nos da algunas ventajas:

- El protocolo X se puede implementar en una gran variedad de lenguajes y sistemas operativos.
- Las conexiones locales y de red pueden operar usando el mismo protocolo, haciendo la red transparente al usuarioy las plicaciones.
- No afecta al rendimiento de la aplicación. El rendimiento es limitado por el tiempo requerido para el dibujo y no por la carga del protocolo.
- Se puede usar sobre cualquier medio que permita el flujo confiable de bytes.
- Define completamente al sistema X Window por lo tanto se puede implementar en cualquier lenguaje que se apegue totalmente a las normas de comunicación que dicta entre cliente y servidor. Cuando el cliente y el servidor se encuentran sobre la misma máquina, la conexión se basa en canales locales de comunicación interprocesos (IPC), memoria compartida o sockets en el caso de UNIX.
- Normalmente, los clientes implementan el protocolo X usando librerías de programación que interactúan con un protocolo de red que cubra las capas bajas del protocolo de comunicación entre maquinas de una red, tipicamente TCP/IP o DECnet. El MIT provee estas librerías escritas en lenguaje C con el nombre de Xlib.

A.3.3 Tipos de mensajes.

El protocolo X especifica cuatro tipos de mensajes los cuales pueden ser transferidos por la red. Las solicitudes son enviadas desde el cliente al servidor. Respuestas, eventos y errores son enviados desde el servidor al cliente.

Una **petición** (request) es generada por el cliente y enviada al servidor. Esta puede acarrear una amplia variedad de información, como son una especificación para dibujar una línea, cambiar el valor del color de una celda en un mapa de colores o talvez el tamaño de una ventana.

Una **respuesta** (reply) es enviada del servidor al cliente como resultado de ciertas solicitudes. No todas las solicitudes son contestadas como respuestas, sólo aquéllas que solicitan información.

Un **evento** es enviado desde el servidor al cliente y contiene información acerca de un dispositivo accionado o acerca del efecto de una solicitud previa. Los datos contenidos en eventos son bastante variados porque este es el principapal método por el cual los clientes obtienen información. Todos son almacenados en estructuras de 32 bytes para simplificar su encolamiento y manejo.

Un **error** es parecido a un evento, pero es manejado de forma diferente por los clientes. Los errores son enviados a una rutina de manejo de errores de las librerías de programación

del lado del cliente. Los mensajes de errores son del mismo tamaño que los eventos para simplificar su manejo.

Las peticiones que requieren de una respuesta inmediata son llamadas peticiones de viaje redondo y tienen que ser minimizadas en los programas de los clientes puesto que degrada el rendimiento cuando hay retardos en la red.

A.3.4 División de responsabilidades.

En el protocolo X, las responsabilidades están divididas.

Primero, el servidor es diseñado, tanto como sea posible, para ocultar diferencias en el hardware subyacente a las aplicaciones clientes. El servidor controla ventanas, todos los dibujos, y las interfaces de los controladores de dispositivos para obtener entradas desde el teclado o ratón. El código del servidor escrito por el MIT contiene una parte dependiente y una independiente del dispositivo. La parte dependiente desde ser optimizada por cada configuración de hardware, yes aquí donde las características desl hardware son traducidas a abstracciones usadas por X como por ejemplo el mapa de colores (*colormaps*).

El hecho de que el servidor tenga la responsabilidad del manejo de la jerarquía y superposición de ventanas, tiene algunas desventajas. En X, es responsabilidad del cliente redibujar el contenido de sus ventanas en el momento en que llega un evento Expose por ejemplo.

El servidor maneja el teclado pero los clientes pueden solicitar el mapeo del teclado de acuerdo a sus necesidades y capacidades del servidor. También maneja abstracciones como son los contextos gráficos, fuentes, cursores, mapas de colores, etc; que comparten los clientes por medio de identificadores (ID's), como una forma de reducir el tráfico en la red al mantener toda esa información en el servidor y no en cada maáquina en donde se ejecute el cliente.

Ejemplo de una sesión.

A continuación se proporciona una explicación de qué es lo que pasa en la red con una pequeña aplicación que crea una ventana, reserva un color, espera por algunos eventos, dibuja dentro de las ventanas y sale. Estos son los eventos que tomarán lugar durante una sesión exitosa del cliente:

- El cliente abre una conexión al servidor y envía información describiéndose a sí mismo.
- El servidor envía de regreso al cliente datos describiendo el servidor o rechazando la conexión solicitada.

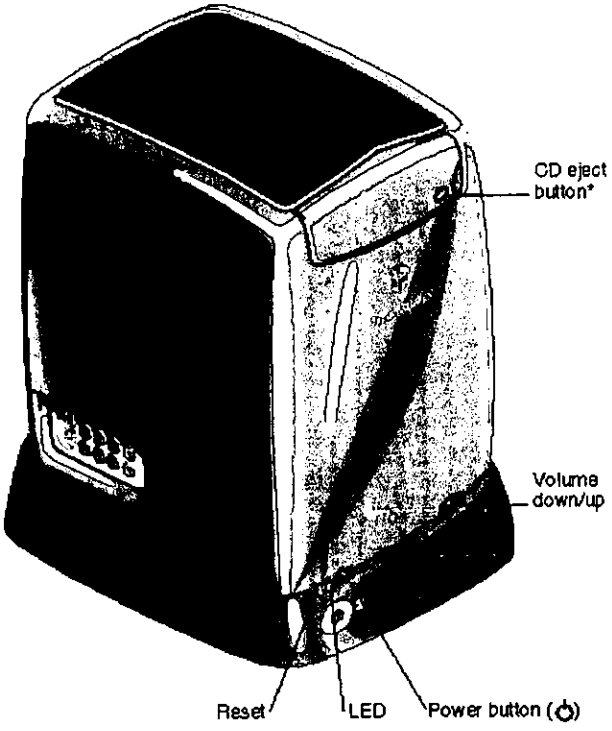
- El cliente hace una solicitud para crear una ventana.
- El cliente hace una solicitud para reservar color.
- El servidor devuelve una respuesta describiendo el color reservado.
- El cliente hace solicitudes para crear el contexto gráfico que será usado más tarde en las solicitudes de dibujo.
- El cliente hace una solicitud identificando los tipos de eventos que este requiere. En este caso, los eventos **Expose** y **ButtonPress**.
- El cliente hace solicitudes para mapear la ventana creada.
- El cliente espera por un evento **Expose** antes de continuar. Este envía las solicitudes acumuladas hacia el servidor.
- El servidor envía al cliente un evento **Expose** indicando que la ventana acaba de ser desplegada.
- El cliente hace una solicitud para dibujar una gráfica, usando contextos gráficos.
- El ciclo de procesamiento de eventos espera por un evento **Expose**.

Muchas de las acciones tomadas por el cliente en la sesión, debieron ser hechas por la aplicación en el orden mostrado para trabajar adecuadamente. Por ejemplo, los eventos **Expose** deben ser seleccionados antes de mapear la ventana, porque en otro caso ningún evento llegaría para notificar al cliente cuando dibujar. Esto es aún más importante cuando el manejador de ventanas controla la pantalla.

B. Hardware de la WorkStation O2.

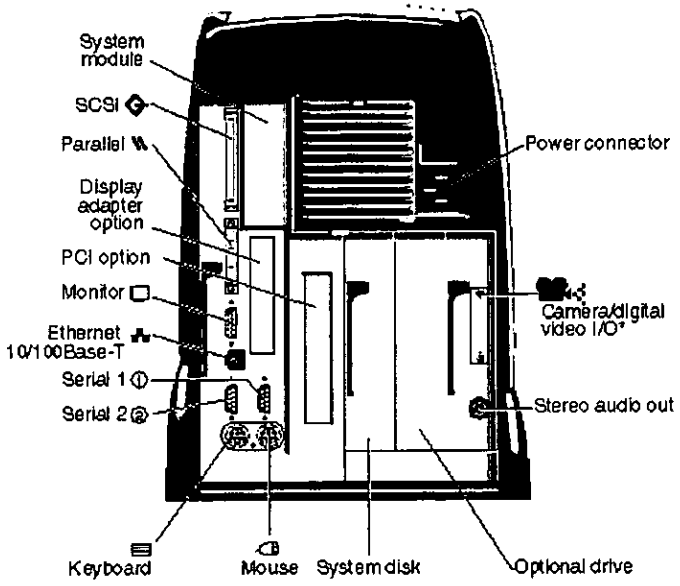
Esto es un resumen de la vista exterior de la estación de trabajo.

Se muestra los controles al frente de la estación de trabajo.



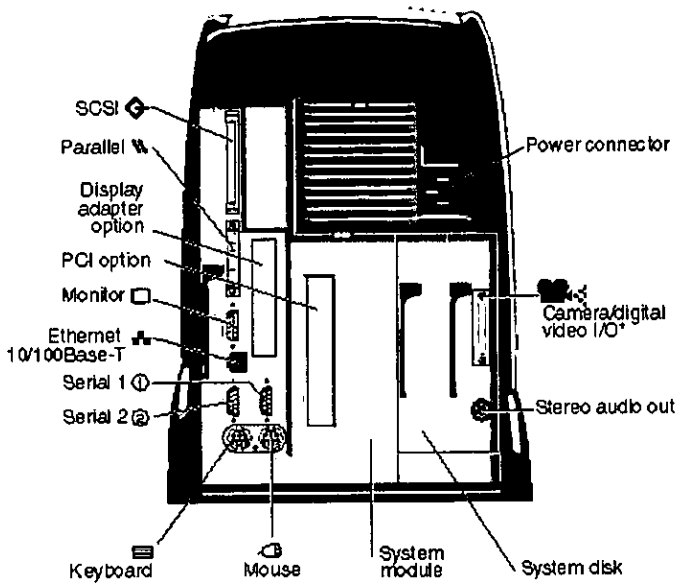
* Not all models

Se muestra el posterior de la estación de trabajo R5000.



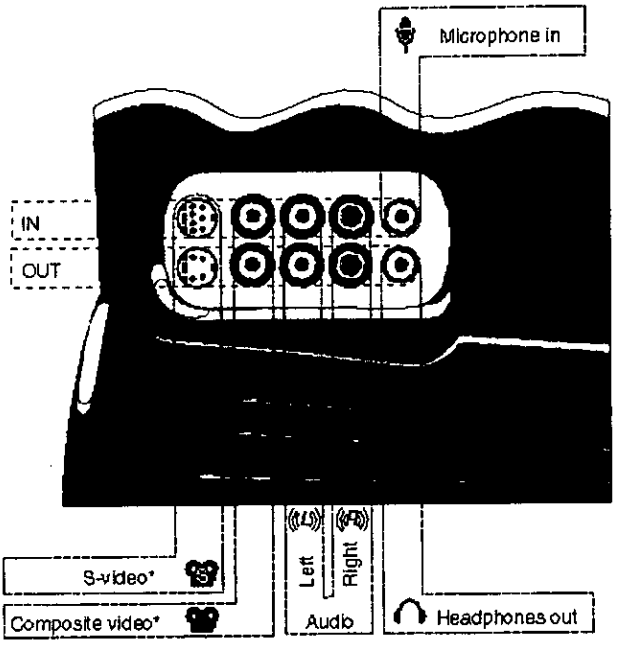
* Not all models

Se muestra el posterior de la estación de trabajo R10000.



* Not all models

Se muestran los puertos de audio y de vídeo, que se encuentran en uno de sus lados.

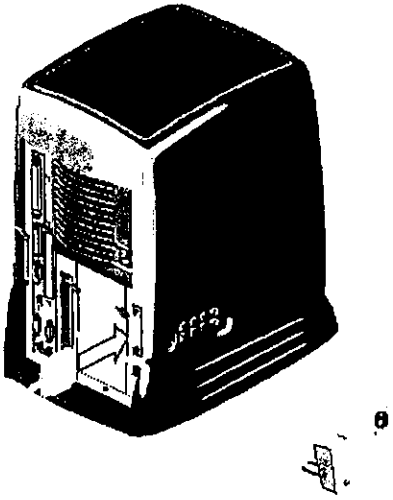


*Not all models

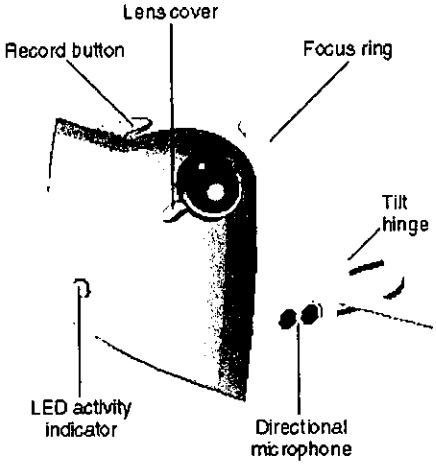
B.1. LA CÁMARA DE VÍDEO DIGITAL O2CAM

La cámara de vídeo digital O2Cam captura imágenes y registra vídeo. Esta tiene internamente un micrófono que apunta en la misma dirección que el lente. Para hablar frente a la cámara.

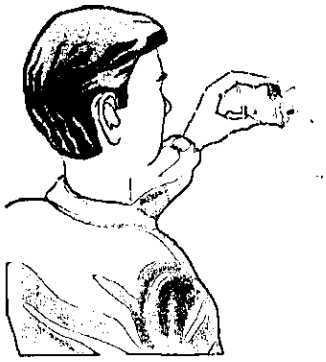
Se muestra donde se conecta la O2Cam a la estación de trabajo. Se debe conectar antes de encender la estación de trabajo para que pueda reconocer el dispositivo.



Se muestran las características de la O2Cam.



Se muestran el uso del micrófono de la cámara.



La información fue tomada del sitio de Silicon Graphics, de Publicaciones Técnicas [O2 WorkStation].

C. Video Library

C.1 INTRODUCCIÓN

Las computadoras gráficas y de vídeo difieren en un número de formas; entendiendo que las diferencias pueden ayudar para producir un mejor resultado con el Vl y la opción de vídeo de una Silicon Graphics. Introduciremos algunos importantes términos en conjunción con el vídeo.

Las diferencias de vídeo de las computadoras gráficas son:

- Interlacing o Entrelazado.
- Broadcast standards.
- Codificación de color.
- Señales de vídeo.
- Formatos de vídeo cintas.

C.2 INTERLACING

A pesar de la forma en que la pantalla es dibujada típicamente por computadoras gráficas, más señales de vídeo son *interlaced* (entrelazadas); cada vez que la pantalla de vídeo es refrescada, únicamente la mitad de las líneas horizontales son dibujadas. Esto es, cada *frame*, es compuesto por dos *fields*.

Durante un refresco de la pantalla, el monitor de vídeo dibuja el primer campo, que contiene todos las líneas numeradas impar; y durante el siguiente refresco, este dibuja el segundo *field*, que contiene todos las líneas numeradas par. Por lo tanto, dos ciclos de refresco son requeridos para dibujar un *frame*.

El rango de despliegue de las señales de vídeo entrelazadas, pueden ser medidas cualquiera de las dos en términos del rango del *field*, o del rango del refresco, o en términos del rango del *frame*, que es igual a la mitas del rango del *field*, porque cada *frame* contiene dos *frames*.

En la sig. Figura se muestra un *frame* y sus dos *fields* para NTSC, utilizado en Norte América y algunas partes del mundo.

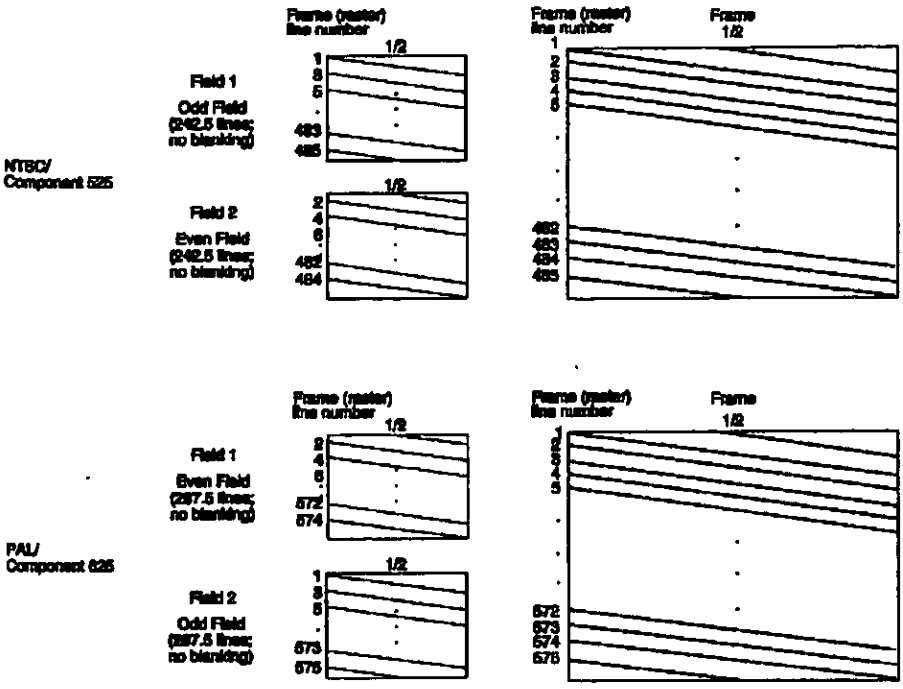


Figura 1: *Fields* y *Frame*.

C.3 BROADCAST STANDARDS

Broadcast standards, o formatos de tiempo de vídeo, son formas de codificación de información de vídeo para radiodifusión de receptores de televisión. Estos estándares son igualmente usados para describir las capacidades de despliegue de vídeo de monitores de vídeo y son de este modo al igual llamadas formatos de tiempo de vídeo o formatos de salida de vídeo (VOFs). Los tres estándares son:

NTSC

Llamado después National Television Systems Committee, que lo desarrollo, este estándar es usado en todo el norte y sur de América, excepto Brazil y también en el Este de Asia.

PAL

(Phase Alternated by Line) Este estandar es usado en el Oeste de Europa, incluyendo Reino Unido pero excluyendo Francia y en el Este de Asia, incluyendo Australia.

SECAM
(Sequential Couleur avec Memoire) Este estándar es usado en Francia, Este de Europa, el Cercano Este y Medio-Este y partes de Africa y el Caribe.

Nota: Las implementaciones de NTSC pueden variar ligeramente por países; las implantaciones de PAL y SECAM pueden variar considerablemente.

NTSC emplea un total de 525 líneas horizontales por *frame*, con dos *fields* por *frame* de 262.5 lineas cada una. Cada *field* se refresca cada 60Hz (actualmente 59.84Hz). NTSC codifica brillantes, color y sincronización de información en una señal.

PAL emplea un total de 625 líneas horizontales por *frame*, con dos *fields* de 312.5 líneas por *frame*. Cada *field* es refrecado en 50Hz. PAL codifica brillantes, color y sincronización de información en una señal, pero en diferente forma al NTSC.

SECAM transmite el mismo número de líneas en un rango de tiempo igual al PAL, pero transmite cada señal de diferencia de color en líneas que se alternan, usando la modulación de frecuencia del sub-portador.

Estos números de líneas horizontales - 525 y 625, respectivamente – son una descripción taquigráfica de lo que sucede actualmente. Para NTSC, el primer (impar) *field* empieza con una línea entera y termina con una media línea; el segundo (par) *field* comienza con la mitad de la línea y termina con la línea entera. Cada *field* NTSC contiene 242.5 líneas activas y 20 líneas de verticales blancas.

Similarmente, para PAL, el primero (par) *field* comienza con una media línea y termina con una línea completa; la segunda (impar) *field* comienza con una línea completa y termina con una media línea. Cada *field* PAL contiene 287.5 líneas activas y 25 líneas de verticales blancas.

En cada caso, los números 525 y 625 se refiere a líneas transmitidas; las líneas de vídeo activo son pocas – típicamente, 485 para NTSC y 575 para PAL. Las líneas restantes son usadas para la delimitación de lindes del *frame* y para la sincronización de otra información.

Para minimizar la fluctuación y reducir el ancho de banda de la señal de vídeo, las líneas activas son entrelazadas.

NTSC y PAL pueden ser registradas digitalmente; estas técnicas de registro son referidas como D2 525 (NTSC digital) y D2 625 (PAL digital).

C.4 CODIFICACIÓN DE COLOR

Los métodos de codificación de color son:

- RGB (componente).
- YUV (componente).
- YIQ (componente).
- YC (separa luminosidad (Y) y color (C)), YC-358, YC-443.
- Mezcla de vídeo.

C.4.1 RGB

RGB es un método de codificación de color usado por muchas computadoras gráficas, además algunas cámaras de calidad profesional. Los tres colores rojo, verde, y azul son generadas separadamente; cada una es portada en un cable separado.

C.4.2 YUV

YUV, una forma que es usada por el estándar de vídeo PAL y por Betacam y cámaras D1 y VCRs, es igual un componente de codificación de color, pero en una forma difiere del RGB. En este caso, brillantes, o luminosidad, es portada en una señal conocida como Y, y el color es llevado en las señales U y V. Las señales de color U y V son dos fases de amplitud-modulada: los componentes U modulados están en el sub-portador en un ángulo de 0 grados. Pero los componentes V modulados están a 90 o 180 grados en líneas alternantes. Los rompimientos de color son al igual líneas alternadas en +135 y –135 grados relativamente a la señal U.

La matriz de multiplicadores derivados de color en YUV del RGB son de las siguientes formulas:

$Y = .299R + .587G + .114B$

$CR = R - Y$

$CB = B - Y$

En cada Y se representa la luminosidad y R-Y, y B-Y representan las diferentes señales de color usadas por este formato. En este sistema, que es algunas veces es referido como Y/R-Y/B-Y, R-Y correspondiendo a CR y V, y B-Y correspondiendo a CB y U. R-Y y B-Y son obtenidos por substraer la luminancia (Y) del rojo (R) y azul (B) señales de cámara, respectivamente. CR, CB, V y U son derivados a travez de diferentes métodos de normalización, dependiendo al formato de vídeo usado. Las señales U y V son portadas en la misma señal.

El componente de color YUV codificado puede ser registrado digitalmente, según al estándar CCIR 601; esta técnica de registro es referida como D1.

C.4.3 YIQ

La codificación de colorYIQ, que es típicamente usada en el formato de vídeo NTSC, codificando el color en dos señales llamadas I y Q (para intermodulación y quadratura, respectivamente). Estas dos señales tienen fase diferente de modulación en la transmisión NTSC. De otro modo los componentes U y V de YUV, I y Q son portados en diferentes anchos de banda.

La fórmula YIQ es la siguiente:

Y = .299R + .587G +.114B (igual como la YUV)

I = .596R - .275G - .321B

Q = .212R - .523G + .311B

C.4.4 YC, YC-358, YC-443, o S-Video

YC, una señal de dos-líneas, resultan cuando I y Q son combinadas dentro de una señal, llamada color (cromanancia). YC-358 es la versión más común de NTSC de este formato de luminancia/color; YC-443 es la versión de PAL más común. Estos formatos son igualmente conocidos como S-Video; S-Video es uno de los formatos usados por S-VHS (TM) en reproductores de cintas de vídeo.

C.5 MEZCLA DE VÍDEO

La mezcla de la codificación del color en esquemas combina la brillantes y las señales de color dentro de una señal para radiodifusión. NTSC y PAL combinan ambos brillantes y color pero usan diferentes métodos.

La Figura 2 muestra las relaciones entre los métodos de codificación de color y los formatos de vídeo.

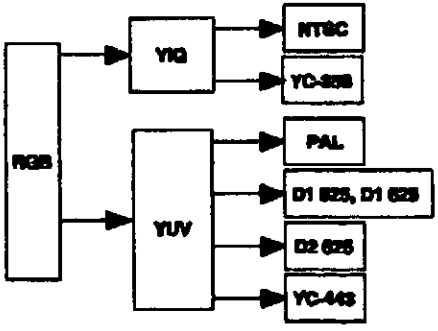


Figura 2: Relación entre los métodos de codificación de color y los formatos de vídeo.

C.6 SEÑALES DE VÍDEO

Las señales de vídeo, cualquiera que sea el *broadcast standard* que sea usado, porta otra información junto con el vídeo (luminancia y color) y audio. Por ejemplo, sincronización horizontal y vertical de la información es requerida, además un color de referencia a fase, que es llamado *color sync burst*. La figura 3 muestra una mezcla de la forma de onda de la señal de vídeo.

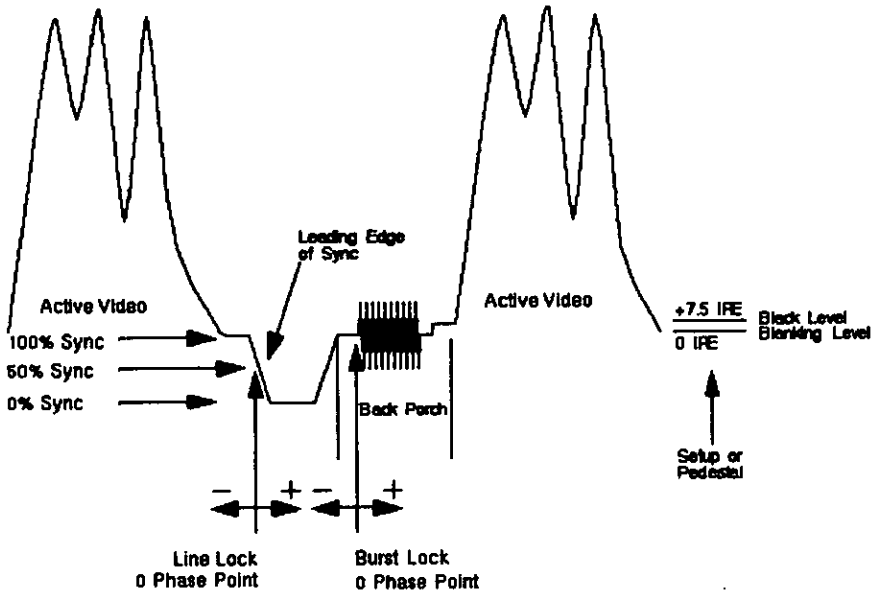


Figura 3: Mezcla de la forma de onda del vídeo.

C.7 FORMATOS DE VÍDEO CINTAS

Los vídeo cintas son disponibles para formatos analógicos y digitales registradas en varios formatos. Ellos son por lo tanto clasificados por el nivel de ejecución, o mercado: consumidor, profesional, y *broadcast*. Además, durante la postproducción (edición, incluyendo adición de gráficos), el original puede ser transferido a medios digitales, formatos digitales de vídeo cintas que son disponibles para mezclas y formatos de componentes de vídeo. No hay un estandar para la clasificación de vídeo cintas.

Tabla 1. Formatos de cintas y formatos de vídeo.

Electronica	Consumidos	Profesional	Broadcast	Postproducción
Analógico	Casete VHS (mezcla)	Casete U-Matic (SP) ¾" (mezcla)	Tipo C carrete a carrete 1" (mezcla)	
	S-VHS (YC, mezcla)		Tipo B (Europa) (mezcla)	
	S-Video (YC-358)	S-Video (YC-358)		
	Beta (mezcla) 8mm (mezcla)	Hi-8mm (YC)	Betacam (mezcla) Betacam SP (YUV, YIQ, mezcla) MII (TM) (YUV, YIQ, mezcla)	
Digital				D1 525 (YUV) D1 625 (YUV) D2 525 (NTSC) D2 625 (PAL)

Aunque el VL y otras opciones de software para vídeo de Silicon Graphics no distinguen entre los formatos de vídeo, se necesita saber el tipo de conector de vídeo. Por ejemplo, la tarjeta Galileo tiene conector para mezcla y S-Video.

Mas conectores de mezcla VCR se usan. S-Video, por otro lado, portan los componentes de color y brillo de la imagen en cables separados; por lo tanto, los conectores S-Video son igualmente llamados conectores Y/C. Más S-VHS y VCRs Hi-8mm carácterizan a los conectores S-Video.

C.8 INICIANDO CON LAS LIBRERÍAS DE VÍDEO

Las librerías de vídeo (VL) son una colección de dipositivos-independientes de llamadas en lenguaje C para estaciones de trabajo Silicon Graphics equipadas con opciones de vídeo, como son Sirius Video (TM), Indigo2 Video (TM), Indy Video (TM), o Galileo Video (TM), o equipados en estaciones de trabajo con tarjetas de vídeo (VINO (TM): vídeo in, no out), como Indy (TM). El VL comprende genéricas herramientas de vídeo, comprendiendo simples herramientas para importación y exportación digital de datos hacia y desde, los actuales y futuros productos Silicon Graphics, además desde y hacia un tercer dispositivo de vídeo que se incluya al modelo de arquitectura de Silicon Graphics para dispositivos de vídeo.

Las llamadas a VL habilitan el desarrollo de teleconferencia en plataformas que lo soporten, mezclar gráficas de computadoras que las generen con *frames* de vídeo cintas o alguna fuente de vídeo, y para presentar vídeo en una ventana de la pantalla de la estación de trabajo y para digitalizar los datos del vídeo.

Nota: El rango de las capacidades del VL puede depender de las capacidades de la estación de trabajo y de las opciones de vídeo instaladas en ella.

La información fue tomada del sitio de Silicon Graphics, de Publicaciones Técnicas [Programming Guide].

D. Código Fuente

D.1 TRANSFORMADA POLINOMIAL DE 2 DIMENSIONES

El código en lenguaje C para implementar la Transformada de 2 Dimensiones es:

```
/*
        dimx y dimy         son las dimensiones de la imagen.
        f                           es la imagen de entrada
        n                           es el nivel que se esta procesando
*/
void transco(int dimx, int dimy,unsigned char *f,int n)
{
        int i,j,k;      /* Contadores */
        int L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,MC[8];  /* Localidades de memoria temporales */

        /* Arreglo para guardar los 9 resultados de la transformada para cada
           pixel
        */
        signed char F[3][3];

        /* Apuntadores a cada una de las imágenes de salida de la transformada
           polinomial
        */
        unsigned char d,*c1;
        signed char *c2, *c3,*c4,*c5,*c6,*c7,*c8,*c9;


        /*
                Las matrices B tienen las imágenes de salida
        */
        c1=B1[n];
        c2=B2[n];
        c3=B3[n];
        c4=B4[n];
        c5=B5[n];
        c6=B6[n];
        c7=B7[n];
        c8=B8[n];
        c9=B9[n];


        /*
                Rutina  para el calcula de la transformada polinomial 2D
        */
        for(j=0;j<dimy;j++)
        {
                MR(0,j)=MR(1,j)=MR(2,j)=MR(3,j)=MR(4,j)=0;
        }
        for(i=0;i<dimx;i+=2)
        {
                for(k=0;k<8;k++)
                        MC[k]=0;
                for(j=0;j<dimy;j+=2)
                {
                        L1=f(i,j)+MR(0,j);
                        L2=f(i,j)-MR(0,j);
                        MR(0,j)=f(i,j);
                        L3=L1+MC(0);
                        L4=L1-MC(0);
                        MC(0)=L1;
                        L5=L2+MC(1);
                        L6=L2-MC(1);
```

```
                    MC(1)=L2;
                    L7=L3+MR(1,j);
                    L8=L3-MR(1,j);
                    MR(1,j)=L3;
                    L9=L4+MR(2,j);
                    MR(2,j)=L4;
                    L10=L5-MR(3,j);
                    MR(3,j)=L5;
                    L11=L6+MR(4,j);
                    L12=L6-MR(4,j);
                    MR(4,j)=L6;

                    F(0,0)=d=(L7+MC(2))/16;
                    MC(2)=L7;
                    F(1,0)=(L8+MC(3))/(16);
                    MC(3)=L8;
                    F(0,1)=(L9+MC(4))/(16);
                    F(0,2)=(L9-MC(4))/(16);
                    MC(4)=L9;
                    F(2,0)=(L10+MC(5))/(16);
                    F(2,1)=(L10-MC(5))/(16);
                    MC(5)=L10;
                    F(1,1)=(L11+MC(6))/16;
                    F(1,2)=(L11-MC(6))/(16);
                    MC(6)=L11;
                    F(2,2)=(L12-MC(7))/16;
                    MC(7)=L12;

                    if((i==0)||(j==0)) continue;

                    *(c1)++=d;
                    *(c2)++=F(0,1);
                    *(c3)++=F(0,2);
                    *(c4)++=F(1,0);
                    *(c5)++=F(1,1);
                    *(c6)++=F(1,2);
                    *(c7)++=F(2,0);
                    *(c8)++=F(2,1);
                    *(c9)++=F(2,2);
                }
        }
```

D.2 TRANSFORMADA POLINOMIAL DE 1 DIMENSIÓN

El código en lenguaje C para implementar la Transformada de 1 Dimension es:

```
void trans1d(int dimx, int dimy, int n)
{
  unsigned char o1t;
  int ang1, ang2, ang3;
  double o2t1, o3t1;
  double o2t2, o3t2;
  double o2t3, o3t3;
  unsigned int cont;
  signed char *c2,*c3,*c4,*c5,*c7;
  signed char *cb,*cc,*cd;
  int num_ang=NUM_ANG;
  long tam,i;
  tam=dimx*dimy;
  c2=B2[n];
  c3=B3[n];
  c4=B4[n];
  c5=B5[n];
  c7=B7[n];
  cb=BB[n];
  cc=BC[n];
  cd=BD[n];
```

```
for(i=0;i<tam; i++,c2++,c3++,c4++,c5++,c7++,cb++,cc++,cd++)
{
  ang1=0;
  ang2=num_ang-1;
  ang3=ang2/2;

  o2t1=(*c2*q_1_0[ang1])+(*c4*q_0_1[ang1]);
  o3t1=(*c3*q_2_0[ang1])+(*c5*q_1_1[ang1])+(*c7*q_0_2[ang1]);
  o2t2=(*c2*q_1_0[ang2])+(*c4*q_0_1[ang2]);
  o3t2=(*c3*q_2_0[ang2])+(*c5*q_1_1[ang2])+(*c7*q_0_2[ang2]);
  o2t3=(*c2*q_1_0[ang3])+(*c4*q_0_1[ang3]);
  o3t3=(*c3*q_2_0[ang3])+(*c5*q_1_1[ang3])+(*c7*q_0_2[ang3]);

  do
  {
   if( (o2t1+o3t1) > (o2t2+o3t2) )
   {
          ang2=ang3;
          o2t2=o2t3;
          o3t2=o3t3;
          if(ang2>(ang1+1))
          {
           ang3=(ang3+ang1)/2;
          }
          else { break; }
   }
   else
   {
          ang1=ang3;
          o2t1=o2t3;
          o3t1=o3t3;
          if(ang2>(ang1+1))
          {
           ang3=(ang2+ang3)/2;
          }
          else { break; }
   }
   o2t3=(*c2*q_1_0[ang3])+(*c4*q_0_1[ang3]);
   o3t3=(*c3*q_2_0[ang3])+(*c5*q_1_1[ang3])+(*c7*q_0_2[ang3]);

   if( (ang2==(ang1+2))&&(ang3==(ang1+1)) )
        {
          while( ((o2t3+o3t3)<(o2t1+o3t1)) && (ang1!=0) )
          {
            ang2=ang3;
            o2t2=o2t3;
            o3t2=o3t3;
            ang3=ang1;
            o2t3=o2t1;
            o3t3=o3t1;
            ang1--;
            o2t1=(*c2*q_1_0[ang1])+(*c4*q_0_1[ang1]);
            o3t1=(*c3*q_2_0[ang1])+(*c5*q_1_1[ang1])+(*c7*q_0_2[ang1]);
          }

          while( ((o2t3+o3t3)<(o2t2+o3t2)) && (ang2!=num_ang) )
          {
            ang1=ang3;
            o2t1=o2t3;
            o3t1=o3t3;
            ang3=ang2;
            o2t3=o2t2;
            o3t3=o3t2;
            ang2++;
            o2t2=(*c2*q_1_0[ang2])+(*c4*q_0_1[ang2]);
            o3t2=(*c3*q_2_0[ang2])+(*c5*q_1_1[ang2])+(*c7*q_0_2[ang2]);
          }

        }

  }while(1);

  if( (o2t1+o3t1) < (o2t2+o3t2) )
```

```
    {
     ang1=ang2;
     o2t1=o2t2;
     o3t1=o3t2;
    }

    if(o2t1>255.0) o2t1=255.0;
    if(o2t1<0.0)   o2t1=0.0;
    if(o3t1>255.0) o3t1=255.0;
    if(o3t1<0.0)   o3t1=0.0;


    cont=o2t1;
    if((o2t1-cont)>0.5)cont++;
    *cb=(unsigned char)cont;

    cont=o3t1;
    if((o3t1-cont)>0.5)cont++;
    *cc=(unsigned char)cont;

    *cd=(unsigned char)(ang1*MUL_DIV);

  }
        memcpy(BA[n],B1[n],tam);
}
```

D.3 ANTITRANSFORMADA POLINOMIAL DE 2 DIMENSIONES

El código en lenguaje C para implementar la Antitransformada de 2 Dimensiones es:

```
void atrans2d(int dimx , int dimy, unsigned char *ima, int  n)
{
/* Funcion para aplicar la antitransformada polinomial y recuperar
  la imagen  */
        int i,j;
        unsigned char *t,*p1;
        signed char *p2,*p3,*p4,*p5,*p6,*p7,*p8,*p9;
        int c[9];

        t=ima;
        p1=B1[n];

        p2=B2[n];
        p3=B3[n];
        p4=B4[n];
        p5=B5[n];
        p6=B6[n];
        p7=B7[n];
        p8=B8[n];
        p9=B9[n];

        for(i=0;i<dimx;i++)
        {
                for(j=0;j<dimy;j++)
                {
                /* Se calculan los renglones pares */
                        /* Se calculan las columnas pares*/
                        c[0]= *p1;
                        c[2]=-(*p3);
                        c[6]=-(*p7);
                        c[8]= *p9;
                        *t=((c[0]+c[2]+c[6]));
                        t++;
                        if((j+1)==(dimy))
                                break;
                        /*Se calculan las columnas impares*/
                        /*c[0]= B1(i,j,n)+B1(i,j+1,n);
                        c[1]= B2(i,j,n)-B2(i,j+1,n);
                        c[2]= B3(i,j,n)+B3(i,j+1,n);
```

```
                c[6]=-B7(i,j,n)-B7(i,j+1,n);
                c[7]=-B8(i,j,n)+B8(i,j+1,n);
                c[8]=-B9(i,j,n)-B9(i,j+1,n);*/
                c[0]= (*p1)+(*(p1+1));
                c[1]= (*p2)-(*(p2+1));
                c[2]= (*p3)+(*(p3+1));
                c[6]=-(*p7)-(*(p7+1));
                c[7]=-(*p8)+(*(p8+1));
                c[8]=-(*p9)-(*(p9+1));

                *t=(((((long)(c[0]+c[2]+c[6]))>>1) + (c[1])*K3));
                t++;
                p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
        }
        *t=*(t-1);t++;
        p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
        p1-=dimy;p2-=dimy;p3-=dimy;p4-=dimy;p5-=dimy;
        p6-=dimy;p7-=dimy;p8-=dimy;p9-=dimy;
        if((i+1)==(dimx))
                break;
        for(j=0;j<dimy;j++)
        {
        /*Se calculan los renglones impares  */
                /*Se calculan las columnas pares*/

                /*c[0]= B1(i,j,n)+B1(i+1,j,n);
                c[2]=-B3(i,j,n)-B3(i+1,j,n);
                c[3]= B4(i,j,n)-B4(i+1,j,n);
                c[5]=-B6(i,j,n)+B6(i+1,j,n);
                c[6]= B7(i,j,n)+B7(i+1,j,n);
                c[8]=-B9(i,j,n)-B9(i+1,j,n);*/
                c[0]= (*p1)+(*(p1+dimy));
                c[2]=-(*p3)-(*(p3+dimy));
                c[3]= (*p4)-(*(p4+dimy));
                c[5]=-(*p6)+(*(p6+dimy));
                c[6]= (*p7)+(*(p7+dimy));
                c[8]=-(*p9)-(*(p9+dimy));

                *t=((((c[0]+c[2]+c[6])>>1)+(c[3])*K3));
                t++;
                if((j+1)==(dimy))
                        break;

                /*Se calculan las columnas impares*/
                c[0]= (*p1)+(*(p1+dimy))+(*(p1+1))+(*(p1+dimy+1));
                c[1]= (*p2)+(*(p2+dimy))-(*(p2+1))-(*(p2+dimy+1));
                c[2]= (*p3)+(*(p3+dimy))+(*(p3+1))+(*(p3+dimy+1));
                c[3]= (*p4)-(*(p4+dimy))+(*(p4+1))-(*(p4+dimy+1));
                c[4]= (*p5)-(*(p5+dimy))-(*(p5+1))+(*(p5+dimy+1));
                c[5]= (*p6)+(*(p6+dimy))-(*(p6+1))-(*(p6+dimy+1));
                c[6]= (*p7)+(*(p7+dimy))+(*(p7+1))+(*(p7+dimy+1));
                c[7]= (*p8)+(*(p8+dimy))-(*(p8+1))-(*(p8+dimy+1));
                c[8]= (*p9)+(*(p9+dimy))+(*(p9+1))+(*(p9+dimy+1));
                *t=((((c[0]+c[2]+c[6])>>2)+(c[1]+c[3])*K5+(c[4]>>1)));
                t++;
                p1++;
                p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
        }
        *t=*(t-1);t++;
        p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
}
        for(j=0;j<dimy;j++)
        {
        /* Se calculan los renglones pares */
                /* Se calculan las columnas pares*/
                /*c[0]= B1(i,j,n);
                c[2]=-B3(i,j,n);
                c[6]=-B7(i,j,n);
                c[8]= B9(i,j,n);*/
                c[0]= *p1;
                c[2]=-(*p3);
                c[6]=-(*p7);
                c[8]= *p9;
                *t=((c[0]+c[2]+c[6]));
```

```
                        t++;
                        if((j+1)==(dimy))
                                break;
                        /*Se calculan las columnas impares*/
                        c[1]= B2(i,j,n)-B2(i,j+1,n);
                        c[2]= B3(i,j,n)+B3(i,j+1,n);
                        c[6]=-B7(i,j,n)-B7(i,j+1,n);
                        c[7]=-B8(i,j,n)+B8(i,j+1,n);
                        c[8]=-B9(i,j,n)-B9(i,j+1,n);*/
                        c[0]= (*p1)+(*(p1+1));
                        c[1]= (*p2)-(*(p2+1));
                        c[2]= (*p3)+(*(p3+1));
                        c[6]=-(*p7)-(*(p7+1));
                        c[7]=-(*p8)+(*(p8+1));
                        c[8]=-(*p9)-(*(p9+1));

                        *t=((((c[0]+c[2]+c[6])>>1) + (c[1]*K3);
                        t++;
                        p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
                }
                *t=*(t-1);t++;
}
```

D.4 ANTITRANSFORMADA POLINOMIAL DE 1 DIMENSIÓN

El código en lenguaje C para implementar la Antitransformada de 1 Dimension es:

```
void atrans1d(int dimx, int dimy, int n)
{
  unsigned int ind;
  double tv2, tv3, tv4, tv5, tv6;
  long i=0, tam;
  signed char *c2, *c3,*c4, *c5,*c7,*cb,*cc,*cd;
  c2=B2[n];
  c3=B3[n];
  c4=B4[n];
  c5=B5[n];
  c7=B7[n];
  cb=B8[n];
  cc=BC[n];
  cd=BD[n];
  tam=dimx*dimy;

  for(i=0;i<tam; i++,c2++,c3++,c4++,c5++,c7++,cb++,cc++,cd++)
  {
     ind=*cd/MUL_DIV;
tv2=*cb*q_1_0[ind];
     tv4=*cb*q_0_1[ind];
     tv3=*cc*q_2_0[ind];
     tv5=*cc*q_1_1[ind];
     tv6=*cc*q_0_2[ind];
     if(tv2>255){tv2=255;}
     if(tv2<0)  {tv2=0.0;}
     if(tv3>255){tv3=255.0;}
     if(tv3<0)  {tv3=0.0;}
     if(tv4>255){tv4=255.0;}
     if(tv4<0)  {tv4=0.0;}
     if(tv5>255){tv5=255.0;}
     if(tv5<0)  {tv5=0.0;}
     if(tv6>255){tv6=255.0;}
     if(tv6<0)  {tv6=0.0;}
     *c2=( char)tv2;
     *c4=( char)tv4;
     *c3=( char)tv3;
     *c5=( char)tv5;
     *c7=( char)tv6;
}
}
```

D.5 CÓDIGO FUENTE COMPLETO DE LA APLICACIÓN

El código fuente completo de la aplicación es:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "params.h"
#include "define.h"
#include "utils.h"

static double q_1_0[NUM_ANG+1],q_0_1[NUM_ANG+1],q_2_0[NUM_ANG+1],q_1_1[NUM_ANG+1],q_0_2[NUM_ANG+1];
static unsigned char *B1[NUM_NIVELES];
static signed char *B2[NUM_NIVELES], *B3[NUM_NIVELES], *B4[NUM_NIVELES];
static signed char *B5[NUM_NIVELES], *B6[NUM_NIVELES], *B7[NUM_NIVELES];
static signed char *B8[NUM_NIVELES], *B9[NUM_NIVELES], *BA[NUM_NIVELES];
static signed char *BB[NUM_NIVELES], *BC[NUM_NIVELES], *BD[NUM_NIVELES];
static int *MR;
static double KK;

void trans2d(int dimx, int dimy,unsigned char *f,int n)
{
        int i,j,k;
        int L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,MC[8];
        unsigned char d,*c1;
        signed char F[3][3];
        signed char *c2, *c3,*c4,*c5,*c6,*c7,*c8,*c9;
        c1=B1[n];
        c2=B2[n];
        c3=B3[n];
        c4=B4[n];
        c5=B5[n];
        c6=B6[n];
        c7=B7[n];
        c8=B8[n];
        c9=B9[n];

        for(j=0;j<dimy;j++)
        {
                MR(0,j)=MR(1,j)=MR(2,j)=MR(3,j)=MR(4,j)=0;
        }
        for(i=0;i<dimx;i+=2)
        {
                for(k=0;k<8;k++)
                        MC[k]=0;
                for(j=0;j<dimy;j+=2)
                {
                        L1=f(i,j)+MR(0,j);
                        L2=f(i,j)-MR(0,j);
                        MR(0,j)=f(i,j);
                        L3=L1+MC(0);
                        L4=L1-MC(0);
                        MC(0)=L1;
                        L5=L2+MC(1);
                        L6=L2-MC(1);
                        MC(1)=L2;
                        L7=L3+MR(1,j);
                        L8=L3-MR(1,j);
                        MR(1,j)=L3;
                        L9=L4+MR(2,j);
                        MR(2,j)=L4;
                        L10=L5-MR(3,j);
                        MR(3,j)=L5;
                        L11=L6+MR(4,j);
                        L12=L6-MR(4,j);
                        MR(4,j)=L6;

                        F(0,0)=d=(L7+MC(2))/16;
                        MC(2)=L7;
                        F(1,0)=(L8+MC(3))/(16);
```

```
                        MC(3)=L8;
                        F(0,1)=(L9+MC(4))/(16);
                        F(0,2)=(L9-MC(4))/(16);
                        MC(4)=L9;
                        F(2,0)=(L10+MC(5))/(16);
                        F(2,1)=(L10-MC(5))/(16);
                        MC(5)=L10;
                        F(1,1)=(L11+MC(6))/16;
                        F(1,2)=(L11-MC(6))/(16);
                        MC(6)=L11;
                        F(2,2)=(L12-MC(7))/16;
                        MC(7)=L12;

                        if((i==0)||(j==0)) continue;

                        *(c1)++=d;
                        *(c2)++=F(0,1);
                        *(c3)++=F(0,2);
                        *(c4)++=F(1,0);
                        *(c5)++=F(1,1);
                        *(c6)++=F(1,2);
                        *(c7)++=F(2,0);
                        *(c8)++=F(2,1);
                        *(c9)++=F(2,2);
                }
                if(i)
                {
                        *(c1++)=*(c1-1),*(c2++)=*(c2-1),*(c3++)=*(c3-1);
                        *(c4++)=*(c4-1),*(c5++)=*(c5-1),*(c6++)=*(c1-1);
                        *(c7++)=*(c7-1),*(c8++)=*(c8-1),*(c9++)=*(c9-1);
                }
        }

}


void muestrea(int dimx, int dimy,unsigned char *f, int n)
{
        unsigned char *uc1, *uc2;
        uc1=B1[n];
        uc2=f;
        memcpy(uc2,uc1,dimx*dimy);
}

void atrans2d(int dimx , int dimy, unsigned char *ima, int  n)
{
/* Funcion para aplicar la antitransformada polinomial y recuperar
  la imagen  */
        int i,j;
        unsigned char *t,*p1;
        signed char *p2,*p3,*p4,*p5,*p6,*p7,*p8,*p9;
        int c[9];

        t=ima;
        p1=B1[n];

        p2=B2[n];
        p3=B3[n];
        p4=B4[n];
        p5=B5[n];
        p6=B6[n];
        p7=B7[n];
        p8=B8[n];
        p9=B9[n];

        for(i=0;i<dimx;i++)
        {
                for(j=0;j<dimy;j++)
                {
                /* Se calculan los renglones pares */
                        /* Se calculan las columnas pares*/
                        c[0]= *p1;
                        c[2]=-(*p3);
                        c[6]=-(*p7);
```

```
                c[8]= *p9;
                *t=((c[0]+c[2]+c[6]));
                t++;
                if((j+1)==(dimy))
                        break;
                /*Se calculan las columnas impares*/
                /*c[0]= B1(i,j,n)+B1(i,j+1,n);
                c[1]= B2(i,j,n)-B2(i,j+1,n);
                c[2]= B3(i,j,n)+B3(i,j+1,n);
                c[6]=-B7(i,j,n)-B7(i,j+1,n);
                c[7]=-B8(i,j,n)+B8(i,j+1,n);
                c[8]=-B9(i,j,n)-B9(i,j+1,n);*/
                c[0]= (*p1)+(*(p1+1));
                c[1]= (*p2)-(*(p2+1));
                c[2]= (*p3)+(*(p3+1));
                c[6]=-(*p7)-(*(p7+1));
                c[7]=-(*p8)+(*(p8+1));
                c[8]=-(*p9)-(*(p9+1));

                *t=(((((long)(c[0]+c[2]+c[6]))>>1) + (c[1]*K3));
                t++;
                p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
        }
        *t=*(t-1);t++;
        p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
        p1-=dimy;p2-=dimy;p3-=dimy;p4-=dimy;p5-=dimy;
        p6-=dimy;p7-=dimy;p8-=dimy;p9-=dimy;
        if((i+1)==(dimx))
                break;
        for(j=0;j<dimy;j++)
        {
        /*Se calculan los renglones impares  */
                /*Se calculan las columnas pares*/

                /*c[0]= B1(i,j,n)+B1(i+1,j,n);
                c[2]=-B3(i,j,n)-B3(i+1,j,n);
                c[3]= B4(i,j,n)-B4(i+1,j,n);
                c[5]=-B6(i,j,n)+B6(i+1,j,n);
                c[6]= B7(i,j,n)+B7(i+1,j,n);
                c[8]=-B9(i,j,n)-B9(i+1,j,n);*/
                c[0]= (*p1)+(*(p1+dimy));
                c[2]=-(*p3)-(*(p3+dimy));
                c[3]= (*p4)-(*(p4+dimy));
                c[5]=-(*p6)+(*(p6+dimy));
                c[6]= (*p7)+(*(p7+dimy));
                c[8]=-(*p9)-(*(p9+dimy));

                *t=((((c[0]+c[2]+c[6])>>1)+(c[3])*K3));
                t++;
                if((j+1)==(dimy))
                        break;

                /*Se calculan las columnas impares*/
                c[0]= (*p1)+(*(p1+dimy))+(*(p1+1))+(*(p1+dimy+1));
                c[1]= (*p2)+(*(p2+dimy))-(*(p2+1))-(*(p2+dimy+1));
                c[2]= (*p3)+(*(p3+dimy))+(*(p3+1))+(*(p3+dimy+1));
                c[3]= (*p4)-(*(p4+dimy))+(*(p4+1))-(*(p4+dimy+1));
                c[4]= (*p5)-(*(p5+dimy))-(*(p5+1))+(*(p5+dimy+1));
                c[5]= (*p6)+(*(p6+dimy))-(*(p6+1))-(*(p6+dimy+1));
                c[6]= (*p7)+(*(p7+dimy))+(*(p7+1))+(*(p7+dimy+1));
                c[7]= (*p8)+(*(p8+dimy))-(*(p8+1))-(*(p8+dimy+1));
                c[8]= (*p9)+(*(p9+dimy))+(*(p9+1))+(*(p9+dimy+1));
                *t=((((c[0]+c[2]+c[6])>>2)+(c[1]+c[3])*K5+(c[4]>>1)));
                t++;
                p1++;
                p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
        }
        *t=*(t-1);t++;
        p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
}
        for(j=0;j<dimy;j++)
        {
        /* Se calculan los renglones pares */
                /* Se calculan las columnas pares*/
```

```
                        /*c[0]= B1(i,j,n);
                        c[2]=-B3(i,j,n);
                        c[6]=-B7(i,j,n);
                        c[8]= B9(i,j,n);*/
                        c[0]= *p1;
                        c[2]=-(*p3);
                        c[6]=-(*p7);
                        c[8]= *p9;
                        *t=((c[0]+c[2]+c[6]));
                        t++;
                        if((j+1)==(dimy))
                                break;
                        /*Se calculan las columnas impares*/
                        c[1]= B2(i,j,n)-B2(i,j+1,n);
                        c[2]= B3(i,j,n)+B3(i,j+1,n);
                        c[6]=-B7(i,j,n)-B7(i,j+1,n);
                        c[7]=-B8(i,j,n)+B8(i,j+1,n);
                        c[8]=-B9(i,j,n)-B9(i,j+1,n);*/
                        c[0]= (*p1)+(*(p1+1));
                        c[1]= (*p2)-(*(p2+1));
                        c[2]= (*p3)+(*(p3+1));
                        c[6]=-(*p7)-(*(p7+1));
                        c[7]=-(*p8)+(*(p8+1));
                        c[8]=-(*p9)-(*(p9+1));

                        *t=((((c[0]+c[2]+c[6]))>>1) + (c[1])*K3);
                        t++;
                        p1++,p2++,p3++,p4++,p5++,p6++,p7++,p8++,p9++;
                }
                *t=*(t-1);t++;
}

/*
 Se llenan los arreglos de con los valores de las ecuaciones para
 cada uno de los archivos de imagen.
*/
void ecuaciones(void)
{
 double incremento=PI/NUM_ANG, ang;
 int cont;

   for(ang=0.0,cont=0; cont<=NUM_ANG; ang+=incremento,cont++)
   {
    q_1_0[cont]=cos(ang);
    q_0_1[cont]=sin(ang);
    q_1_1[cont]=sqrt(1-((1/ORDEN_M)*(pow(cos(ang),4.0)+pow(sin(ang),4.0))));
    q_0_2[cont]=q_2_0[cont]=sqrt(1-(1/ORDEN_M))*q_1_1[cont];
    q_1_1[cont]*=sqrt(2)*cos(ang)*sin(ang);
    q_2_0[cont]*=pow(cos(ang),2.0);
    q_0_2[cont]*=pow(sin(ang),2.0);
   }
   KK=cos(M_PI_4);
}

void trans1d(int dimx, int dimy, int n)
{
 unsigned char o1t;
 int ang1, ang2, ang3;
 double o2t1, o3t1;
 double o2t2, o3t2;
 double o2t3, o3t3;
 unsigned int cont;
 signed char *c2,*c3,*c4,*c5,*c7;
 signed char *cb,*cc,*cd;
 int num_ang=NUM_ANG;
 long tam,l;
 tam=dimx*dimy;
 c2=B2[n];
 c3=B3[n];
 c4=B4[n];
 c5=B5[n];
 c7=B7[n];
 cb=B8[n];
 cc=B9[n];
```

```
cd=BD[n];


for(i=0;i<tam; i++,c2++,c3++,c4++,c5++,c7++,cb++,cc++,cd++)
{
  ang1=0;
  ang2=num_ang-1;
  ang3=ang2/2;

  o2t1=(*c2*q_1_0[ang1])+(*c4*q_0_1[ang1]);
  o3t1=(*c3*q_2_0[ang1])+(*c5*q_1_1[ang1])+(*c7*q_0_2[ang1]);
  o2t2=(*c2*q_1_0[ang2])+(*c4*q_0_1[ang2]);
  o3t2=(*c3*q_2_0[ang2])+(*c5*q_1_1[ang2])+(*c7*q_0_2[ang2]);
  o2t3=(*c2*q_1_0[ang3])+(*c4*q_0_1[ang3]);
  o3t3=(*c3*q_2_0[ang3])+(*c5*q_1_1[ang3])+(*c7*q_0_2[ang3]);

  do
  {
   if( (o2t1+o3t1) > (o2t2+o3t2) )
   {
          ang2=ang3;
          o2t2=o2t3;
          o3t2=o3t3;
          if(ang2>(ang1+1))
          {
           ang3=(ang3+ang1)/2;
          }
          else { break; }
   }
   else
   {
          ang1=ang3;
          o2t1=o2t3;
          o3t1=o3t3;
          if(ang2>(ang1+1))
          {
           ang3=(ang2+ang3)/2;
          }
          else { break; }
   }
   o2t3=(*c2*q_1_0[ang3])+(*c4*q_0_1[ang3]);
   o3t3=(*c3*q_2_0[ang3])+(*c5*q_1_1[ang3])+(*c7*q_0_2[ang3]);

   if( (ang2==(ang1+2))&&(ang3==(ang1+1)) )
        {
          while( ((o2t3+o3t3)<(o2t1+o3t1)) && (ang1!=0) )
          {
            ang2=ang3;
            o2t2=o2t3;
            o3t2=o3t3;
            ang3=ang1;
            o2t3=o2t1;
            o3t3=o3t1;
            ang1--;
            o2t1=(*c2*q_1_0[ang1])+(*c4*q_0_1[ang1]);
            o3t1=(*c3*q_2_0[ang1])+(*c5*q_1_1[ang1])+(*c7*q_0_2[ang1]);
          }

          while( ((o2t3+o3t3)<(o2t2+o3t2)) && (ang2!=num_ang) )
          {
            ang1=ang3;
            o2t1=o2t3;
            o3t1=o3t3;
            ang3=ang2;
            o2t3=o2t2;
            o3t3=o3t2;
            ang2++;
            o2t2=(*c2*q_1_0[ang2])+(*c4*q_0_1[ang2]);
            o3t2=(*c3*q_2_0[ang2])+(*c5*q_1_1[ang2])+(*c7*q_0_2[ang2]);
          }

        }

}while(1);
```

```
    if( (o2t1+o3t1) < (o2t2+o3t2) )
    {
     ang1=ang2;
     o2t1=o2t2;
     o3t1=o3t2;
    }

    if(o2t1>255.0) o2t1=255.0;
    if(o2t1<0.0)   o2t1=0.0;
    if(o3t1>255.0) o3t1=255.0;
    if(o3t1<0.0)   o3t1=0.0;


    cont=o2t1;
    if((o2t1-cont)>0.5)cont++;
    *cb=(unsigned char)cont;

    cont=o3t1;
    if((o3t1-cont)>0.5)cont++;
    *cc=(unsigned char)cont;

    *cd=(unsigned char)(ang1*MUL_DIV);

  }
          memcpy(BA[n],B1[n],tam);
}

/*
 ************************************************************** */

/*
  Procesa los vectores de entrada para generar llenar los 3 de salida
*/

void atrans1d(int dimx, int dimy, int n)
{
  unsigned int ind;
  double tv2, tv3, tv4, tv5, tv6;
  long i=0, tam;
  signed char *c2, *c3,*c4, *c5,*c7,*cb,*cc,*cd;
  c2=B2[n];
  c3=B3[n];
  c4=B4[n];
  c5=B5[n];
  c7=B7[n];
  cb=B8[n];
  cc=BC[n];
  cd=BD[n];
  tam=dimx*dimy;

  for(i=0;i<tam; i++,c2++,c3++,c4++,c5++,c7++,cb++,cc++,cd++)
  {
    ind=*cd/MUL_DIV;
tv2=*cb*q_1_0[ind];
    tv4=*cb*q_0_1[ind];
    tv3=*cc*q_2_0[ind];
    tv5=*cc*q_1_1[ind];
    tv6=*cc*q_0_2[ind];
    if(tv2>255){tv2=255;}
    if(tv2<0)  {tv2=0.0;}
    if(tv3>255){tv3=255.0;}
    if(tv3<0)  {tv3=0.0;}
    if(tv4>255){tv4=255.0;}
    if(tv4<0)  {tv4=0.0;}
    if(tv5>255){tv5=255.0;}
    if(tv5<0)  {tv5=0.0;}
    if(tv6>255){tv6=255.0;}
    if(tv6<0)  {tv6=0.0;}
    *c2=( char)tv2;
    *c4=( char)tv4;
    *c3=( char)tv3;
    *c5=( char)tv5;
    *c7=( char)tv6;
```

```
}
}

int lee_imagen(int dimx,int dimy, unsigned char *f,char *img)
{
/*Funcion para leer el archivo de la imagen y pasarla a memoria */
        FILE  *fp;
        if(!(fp=fopen(img,"rb")))
                return 0;
        fread(f,sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        return 1;
}

int escribe_imagen(int dimx, int dimy,unsigned char *f, char *img)
{
/*Funcion para grabar a disco, la imagen */
        FILE *fp1;
        if(!(fp1=fopen(img,"wb")))
                return 0;
        fwrite(f,sizeof(unsigned char),dimx*dimy,fp1);
        fclose(fp1);
        return 1;
}

int buffer2disk(int dimx,int dimy,int n)
{
        FILE *fp;

        puts("ARCH_SAL1");
        if(!(fp=fopen(ARCH_SAL1,"wb")))
                return 0;
        fwrite(B1[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL2,"wb")))
                return 0;
        fwrite(B2[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL3,"wb")))
                return 0;
        fwrite(B3[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL4,"wb")))
                return 0;
        fwrite(B4[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL5,"wb")))
                return 0;
        fwrite(B5[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL6,"wb")))
                return 0;
        fwrite(B6[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL7,"wb")))
                return 0;
        fwrite(B7[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL8,"wb")))
                return 0;
        fwrite(B8[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SAL9,"wb")))
                return 0;
        fwrite(B9[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SALA,"wb")))
                return 0;
        fwrite(BA[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SALB,"wb")))
                return 0;
        fwrite(BB[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
```

```
        if(!(fp=fopen(ARCH_SALC,"wb")))
                return 0;
        fwrite(BC[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        if(!(fp=fopen(ARCH_SALD,"wb")))
                return 0;
        fwrite(BD[n],sizeof(unsigned char),dimx*dimy,fp);
        fclose(fp);
        return 1;
}

int disk2buffer(int dimx,int dimy, int n)
{
        FILE *fp1,*fp2,*fp3,*fp4,*fp5,*fp6,*fp7,*fp8,*fp9;
        FILE *fpa,*fpb,*fpc,*fpd;

        if(!(fp1=fopen(ARCH_ENT1,"rb")))
                return 0;
        if(!(fp2=fopen(ARCH_ENT2,"rb")))
                return 0;
        if(!(fp3=fopen(ARCH_ENT3,"rb")))
                return 0;
        if(!(fp4=fopen(ARCH_ENT4,"rb")))
                return 0;
        if(!(fp5=fopen(ARCH_ENT5,"rb")))
                return 0;
        if(!(fp6=fopen(ARCH_ENT6,"rb")))
                return 0;
        if(!(fp7=fopen(ARCH_ENT7,"rb")))
                return 0;
        if(!(fp8=fopen(ARCH_ENT8,"rb")))
                return 0;
        if(!(fp9=fopen(ARCH_ENT9,"rb")))
                return 0;
        if(!(fpa=fopen(ARCH_ENTA,"rb")))
                return 0;
        if(!(fpb=fopen(ARCH_ENTB,"rb")))
                return 0;
        if(!(fpc=fopen(ARCH_ENTC,"rb")))
                return 0;
        if(!(fpd=fopen(ARCH_ENTD,"rb")))
                return 0;
        fread(B1[n],sizeof(unsigned char),dimx*dimy,fp1);
        fread(B2[n],sizeof(unsigned char),dimx*dimy,fp2);
        fread(B3[n],sizeof(unsigned char),dimx*dimy,fp3);
        fread(B4[n],sizeof(unsigned char),dimx*dimy,fp4);
        fread(B5[n],sizeof(unsigned char),dimx*dimy,fp5);
        fread(B6[n],sizeof(unsigned char),dimx*dimy,fp6);
        fread(B7[n],sizeof(unsigned char),dimx*dimy,fp7);
        fread(B8[n],sizeof(unsigned char),dimx*dimy,fp8);
        fread(B9[n],sizeof(unsigned char),dimx*dimy,fp9);
        fread(BA[n],sizeof(unsigned char),dimx*dimy,fpa);
        fread(BB[n],sizeof(unsigned char),dimx*dimy,fpb);
        fread(BC[n],sizeof(unsigned char),dimx*dimy,fpc);
        fread(BD[n],sizeof(unsigned char),dimx*dimy,fpd);
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        fclose(fp4);
        fclose(fp5);
        fclose(fp6);
        fclose(fp7);
        fclose(fp8);
        fclose(fp9);
        fclose(fpa);
        fclose(fpb);
        fclose(fpc);
        fclose(fpd);
        return 1;
}

int crea_buffers(int dimx, int dimy)
{
        int i;
```

```
        MR=reserva_memoria(int,5*dimy);
        if(!MR) return 0;
        for(i=0;i<NUM_NIVELES;i++)
        {
          dimx>>=1;dimy>>=1;
          if(!crea_nivel(dimx,dimy,i)) return 0;
        }
        return 1;
}

void libera_memoria(void)
{
        int i;
        for(i=0;i<NUM_NIVELES;i++)
        {
          borra_nivel(i);
        }
        free(MR);
}

int crea_nivel(int dimx , int dimy, int n)
{
        B1[n]=reserva_memoria(unsigned char,dimx*dimy);
        if(!B1[n]) return 0;

        B2[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B2[n]) return 0;

        B3[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B3[n]) return 0;

        B4[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B4[n]) return 0;

        B5[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B5[n]) return 0;

        B6[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B6[n]) return 0;

        B7[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B7[n]) return 0;

        B8[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B8[n]) return 0;

        B9[n]=reserva_memoria(signed char,dimx*dimy);
        if(!B9[n]) return 0;

        BA[n]=reserva_memoria(signed char,dimx*dimy);
        if(!BA[n]) return 0;

        BB[n]=reserva_memoria(signed char,dimx*dimy);
        if(!BB[n]) return 0;

        BC[n]=reserva_memoria(signed char,dimx*dimy);
        if(!BC[n]) return 0;

        BD[n]=reserva_memoria(signed char,dimx*dimy);
        if(!BD[n]) return 0;

        return 1;
}

void  borra_nivel(int n)
{
        free(B1[n]);
        free(B2[n]);
        free(B3[n]);
        free(B4[n]);
        free(B5[n]);
        free(B6[n]);
        free(B7[n]);
        free(B8[n]);
```

```
        free(B9[n]);
        free(BA[n]);
        free(BB[n]);
        free(BC[n]);
        free(BD[n]);
}

void buf2img(int dimx, int dimy, unsigned char * imagen, int nivel)
{
        signed char *c2,*c3,*c4,*c5,*c6,*c7,*c8,*c9;
        unsigned char *c1,*t;
        int i;

        t=imagen;
        c1=getB1(nivel);
        c2=getB2(nivel);
        c3=getB3(nivel);
        c4=getB4(nivel);
        c5=getB5(nivel);
        c6=getB6(nivel);
        c7=getB7(nivel);
        c8=getB8(nivel);
        c9=getB9(nivel);

        dimx>>=(nivel+1);
    dimy>>=(nivel+1);

        gray1(dimx,dimy,c2);
        gray2(dimx,dimy,c3);
        gray1(dimx,dimy,c4);
        gray2(dimx,dimy,c5);
        gray3(dimx,dimy,c6);
        gray2(dimx,dimy,c7);
        gray3(dimx,dimy,c8);
        gray4(dimx,dimy,c9);

        for(i=0;i<dimx;i++)
        {
                memcpy(t,c1,dimy);
                t+=dimy;c1+=dimy;
                memcpy(t,c2,dimy);
                t+=dimy;c2+=dimy;
                memcpy(t,c3,dimy);
                t+=dimy;c3+=dimy;
        }
        for(i=0;i<dimx;i++)
        {
                memcpy(t,c4,dimy);
                t+=dimy;c4+=dimy;
                memcpy(t,c5,dimy);
                t+=dimy;c5+=dimy;
                memcpy(t,c6,dimy);
                t+=dimy;c6+=dimy;
        }
        for(i=0;i<dimx;i++)
        {
                memcpy(t,c7,dimy);
                t+=dimy;c7+=dimy;
                memcpy(t,c8,dimy);
                t+=dimy;c8+=dimy;
                memcpy(t,c9,dimy);
                t+=dimy;c9+=dimy;
        }
}

void buf1img(int dimx, int dimy, unsigned char * imagen, int nivel)
{
        signed char *cb,*cc,*cd;
        unsigned char *c1,*t;
        int i;

        t=imagen;
        c1=getB1(nivel);
        cb=getBB(nivel);
```

```
        cc=getBC(nivel);
        cd=getBD(nivel);

    dimx>>=(nivel+1);
    dimy>>=(nivel+1);


        gray1(dimx,dimy,cb);
        gray2(dimx,dimy,cc);
        gray1(dimx,dimy,cd);

        for(i=0;i<dimx;i++)
        {
                memcpy(t,c1,dimy);
                t+=dimy;c1+=dimy;
                memcpy(t,cb,dimy);
                t+=dimy;cb+=dimy;
                memcpy(t,cc,dimy);
                t+=dimy;cc+=dimy;
                memcpy(t,cd,dimy);
                t+=dimy;cd+=dimy;
        }
}

unsigned char * getB1(int n)
{
        return B1[n];
}

signed char * getB2(int n)
{
        return B2[n];
}

signed char * getB3(int n)
{
        return B3[n];
}

signed char * getB4(int n)
{
        return B4[n];
}

signed char * getB5(int n)
{
        return B5[n];
}

signed char * getB6(int n)
{
        return B6[n];
}

signed char * getB7(int n)
{
        return B7[n];
}

signed char * getB8(int n)
{
        return B8[n];
}

signed char * getB9(int n)
{
        return B9[n];
}

signed char * getBB(int n)
{
        return BB[n];
}
```

```
signed char * getBC(int n)
{
	return BC[n];
}

signed char * getBD(int n)
{
	return BD[n];
}

void gray(int dimx, int dimy, signed char *img)
{
	long i;
	signed char *t;

	t=img;

	for(i=0; i<(dimx*dimy); i++, t++)
	{
		*t+=128;
	}
}

void gray1(int dimx, int dimy, signed char *img)
{
	long i;
	signed char *t;

	t=img;

	for(i=0; i<(dimx*dimy); i++, t++)
	{
		*t^=8;
		*t+=128;
	}
}
void gray2(int dimx, int dimy, signed char *img)
{
	long i;
	signed char *t;

	t=img;

	for(i=0; i<(dimx*dimy); i++, t++)
	{
		*t^=12;
		*t+=128;
	}
}
void gray3(int dimx, int dimy, signed char *img)
{
	long i;
	signed char *t;

	t=img;

	for(i=0; i<(dimx*dimy); i++, t++)
	{
		*t^=64;
		*t+=128;
	}
}
void gray4(int dimx, int dimy, signed char *img)
{
	long i;
	signed char *t;

	t=img;

	for(i=0; i<(dimx*dimy); i++, t++)
	{
		*t^=100;
		*t+=128;
	}}
```