03063
16

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

UNIDAD ACADÉMICA DE LOS CICLOS
PROFESIONALES Y DE POSGRADO

Cómputo Paralelo en la Visualización de Imágenes Tridimensionales

T E S I S

QUE PARA OBTENER EL GRADO DE:
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

RANULFO RODRÍGUEZ SOBREYRA

DIRECTOR DE TESIS:
DR. FABIÁN GARCÍA NOCETTI

MÉXICO, D.F. ENERO DE 2002

TESIS CON
FALLA DE ORIGEN

Universidad Nacional
Autónoma de México

Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos:

Quiero agradecer en primer lugar al Dr. Fabián García Nocetti, por el asesoramiento, por la paciencia y su apoyo constantes durante el desarrollo del presente trabajo. También agradezco al Ing. Francisco Javier Cárdenas F., por el apoyo técnico en el funcionamiento de la plataforma de desarrollo, y a todos los integrantes del Departamento de Ingeniería de Sistemas Computacionales y Automatización (Sección de Ingeniería de Sistemas Computacionales). Y en especial, a la Mat. Ana Luisa Solis González por proporcionar el conjunto de imágenes ultrasónicas, empleadas para la evaluación de la arquitectura propuesta (Laboratorio de Visualización, Facultad de Ciencias).

A la UNAM, que me dio la oportunidad de continuar con mi enriquecimiento académico, a través de su programa de posgrado impartido por la U.A.C.P.y P. También, agradezco a los académicos y administrativos de dicho programa, por su dedicación y apoyo.

Este trabajo fue realizado con el apoyo de los siguientes proyectos: de CONACYT (No.27982A y No. 7350-858), Arquitecturas y Algoritmos de Alto Desempeño en Imagenología Ultrasónica y Red de Desarrollo e Investigación en Informática REDII 1998-1999. Y de la DGAPA-UNAM (PAPIIT-IN117999), Arquitecturas y Algoritmos de Alto Desempeño para la Generación y Procesamiento de Imágenes Acústicas.

Y a todos mis amigos que de algún modo colaboraron en la realización de esta tesis, mil gracias.

Resumen

En el presente trabajo se plantea el diseño y desarrollo de un sistema de visualización de imágenes tridimensionales a partir de un conjunto de imágenes bidimensionales, empleando cómputo paralelo.

De un estudio de los diversos métodos empleados para la visualización tridimensional, se seleccionó el algoritmo "volume rendering", que permite un manejo e interpretación eficiente del volumen de datos asociados al conjunto de imágenes bidimensionales.

Debido a que la complejidad computacional del método, se hace factible la utilización de técnicas de cómputo paralelo para reducir el tiempo de ejecución, al implementar el algoritmo y distribuir los datos en un número creciente de procesadores. Se empleó una plataforma de desarrollo basada en una arquitectura de memoria distribuida, los procesadores utilizados fueron "transputers" (INMOS T805).

A partir de un caso de estudio, se obtuvieron una imagen tridimensional y los tiempos promedios de ejecución, para configuraciones de 1, 2, 4, 8 y 16 procesadores.

Se realizó un análisis de métricas de desempeño ("speedup", eficiencia y fracción serial). El cual muestra un desempeño limitado del sistema, debido a un intenso nivel de comunicaciones en las etapas de distribución del volumen de datos y en la integración de la imagen tridimensional (se prevé que al incrementar la granularidad del sistema, será mayor el aprovechamiento de los recurso de la paralelización).

La plataforma resultó ser la adecuada para llevar a cabo los desarrollos preliminares de estrategias de paralelización de algoritmos, que pueden ser transportables a otras plataformas paralelas (de mayor velocidad), manteniendo la misma filosofía de diseño.

Contenido

Capítulo 1

Pag.

Introducción 1
1.1 Introducción General 1
1.2 Objetivos 1
1.3 Descripción General de los Capítulos 2

Capítulo 2

Visualización de Datos Tridimensionales 3
2.1 Introducción 3
2.2 Conceptos Básicos de la Visualización Tridimensional 4
2.3 Métodos de Despliegue Tridimensional 6
2.3.1 Proyección paralela 6
2.3.2 Proyección de perspectiva 7
2.3.3 Indicación de la intensidad 7
2.3.4 Identificación de superficies y línea visible 8
2.3.5 Representación de superficies 8
2.3.6 Vistas separadas y de recortado 8
2.3.7 Vistas tridimensional y estereoscópicas 8
2.4 Representaciones Tridimensionales de Objetos 9
2.4.1 Representación y modelación de objetos geométricos 10
2.4.2 Representación poligonal 11
2.4.2.1 Modelación de objetos mediante polígonos 13
2.4.2.2 Modelación manual de objetos mediante polígonos 13
2.4.2.3 Generación automática de objetos mediante polígonos 14
2.4.2.4 Modelación matemática de objetos mediante polígonos 14
2.4.2.5 Generación por barrido de objetos mediante polígonos 15
2.4.3 Mallas de parches con parámetros bicúbicos 15
2.3.1 Modelación de objetos mediante redes de parches 16
2.4.4 Geometría Sólida Constructiva 18
2.4.5 Técnicas de subdivisión espacial 20
2.4.5.1 Árboles octantes 20

2.4.6 Estrategias de Interprestación ("rendering") 21
2.4.6.1 Interpretación de objetos formados por polígonos.......... 21
2.4.6.2 Interpretación de una malla de parches con parámetros.. 24
2.4.6.3 Interpretación con una descripción en CSG..................... 24
2.4.7 "Volume rendering" 26
2.4 Discusión 28

Capítulo 3

Estudio del Algoritmo de Visualización "Volume Rendering" 29
3.1 Introducción 29
3.2 Preprocesamiento del Volumen de Datos 29
3.2.1 Otsu un método de Segmentación 32
3.3 Interpretación del Volumen de Datos 34
3.3.1 Teoría "Volume Rendering" empleando "Ray Casting" 36
3.3.2 Definición del Algoritmo Típico del "Volume Rendering" 37
3.3.2.1 Cálculo de la Densidad del voxel 39
3.3.2.2 Proceso de composición 40
3.3.2.3 Interpretación en volúmenes binarios 41
3.3.2.4 Interpretación en volúmenes en tonos de gris 43
3.4 Discusión 46

Capítulo 4

Procesamiento Paralelo y Lenguajes de Programación Paralela 47
4.1 Introducción 47
4.2 Conceptos Básicos del Procesamiento Paralelo 47
4.2.1 Terminología del procesamiento paralelo 47
4.3 Arreglos de Procesadores, Multiprocesadores y Multicomputadoras 51
4.3.1 Arreglos de procesadores 51
4.3.1.1 Malla de la red 52
4.3.1.2 Red árbol binario 53
4.3.1.3 Red hiper-árbol 53
4.3.1.4 Red piramidal 54
4.3.1.5 Red Butterfly 55

4.3.1.6 Red hipercubo(cubo-conectado) 56
4.3.1.7 Red de ciclos de cubos-conectados 56
4.3.1.8 Red cambio-Intercambio 57
4.3.1.9 Red Bruijn 58
4.3.2 Organización del procesador 59
4.3.3 Multiprocesadores 60
4.3.3.1 Multiprocesadores con Memoria de Acceso Uniforme 60
4.3.3.2 Multiprocesadores con Memoria de Acceso No Uniforme 61
4.3.4 Multicomputadoras 62
4.4 Taxonomía de Flynn 63
4.5 Lenguajes de Programación Paralela 63
4.5.1 FORTRAN 90 64
4.5.2 Modelo de Programadores en C 64
4.5.3 C Secuencial 64
4.5.4 Programación nCUBE 65
4.5.5 OCCAM 66
4.5.6 C-LINDA 66
4.5.7 Multi-Pascal 67
4.5 Discusión 67

Capítulo 5

Implementación Paralela del Algoritmo de Visualización "Volume Rendering" 69
5.1 Introducción 69
5.2 Etapas de la Implementación del Algoritmo para Obtener una Imagen Tridimensional 69
5.3 Descripción de la Estrategia de Paralelización 72
5.4 Implantación del Programa 77
5.5 Caso de Estudio 77
5.5.1 Organización de imágenes 83
5.5.2 Descripción del ciclo de procesamiento 84
5.6 Discusión 87

Capítulo 6

Resultados 89
6.1 Introducción 89
6.2 Tiempo de Ejecución para cada Configuración de los Transputers 90
6.3 Métricas de Desempeño "Speedup", Eficiencia y Fracción Serial 91
6.4 Análisis de los Resultados 92
6.5 Discusión 94

Capítulo 7

Conclusiones 97

Bibliografía 101

Anexo A 107

Anexo B 111

Capítulo 1

Introducción

1.1 Introducción General

En el Departamento de Ingeniería de Sistemas Computacionales y Automatización (DISCA) del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS), se desarrolla investigación en el área de cómputo paralelo, aplicado al procesamiento de señales e imágenes ultrasónicas. Una línea de investigación de gran interés ha sido la de la imagenología ultrasónica, en la que se ha logrado desarrollos que permiten adquirir, procesar y desplegar imágenes ultrasónicas de alta resolución. Dentro de esta línea y como continuación del trabajo previamente descrito, se plantea el desarrollo de un sistema de visualización de imágenes tridimensionales, a partir de un conjunto de imágenes bidimensionales, el cual esta basado en técnicas de interpretación volumétrica. Un sistema de cómputo paralelo, basado en una arquitectura de memoria distribuida, es propuesto como plataforma para la implementación de los algoritmos computacionalmente intensivos, que son utilizados en el proceso de formación de la imagen tridimensional.

Este trabajo presenta estrategias de paralelización, desarrollados específicamente para la implementación de los diversos procesos que componen la técnica de visualización volumétrica utilizada. Además, de un estudio del desempeño del sistema al implementarlo en diversas configuraciones de procesadores, realizando al mismo tiempo un análisis de los resultados obtenidos. Finalmente se proponen líneas de investigación en las que se plantea desarrollar trabajo a futuro.

1.2 Objetivos

El objetivo general del presente trabajo es:
Diseñar y desarrollar un sistema de visualización de imágenes tridimensionales a partir de un conjunto de imágenes bidimensionales, utilizando técnicas de procesamiento en paralelo.

A partir del objetivo general se derivan los siguientes objetivos específicos:

- *Realizar un estudio de los diversos métodos de visualización tridimensional para seleccionar aquel que permita un manejo e interpretación eficiente del volumen de datos asociados al conjunto de imágenes bidimensionales.*
- *Desarrollar estrategias de paralelización que permitan distribuir el volumen de datos en diversas configuraciones de procesadores en una arquitectura de memoria distribuida.*
- *Implementar el algoritmo de visualización seleccionado utilizando técnicas de cómputo paralelo en una plataforma de desarrollo basada en una arquitectura de memoria distribuida.*

- *Realizar un estudio de desempeño de la implementación paralela del algoritmo utilizando diversas configuraciones de procesadores.*

1.3 Descripción general de los capítulos

Para cumplir con los objetivos planteados en la presente tesis, el trabajo se divide en los siguientes capítulos:

Capítulo 1: Se presenta una introducción general del trabajo realizado, los objetivos y el contenido por capítulos del mismo.

Capítulo 2: Se presenta una introducción a la visualización tridimensional, que incluye conceptos y definiciones asociados con esta área de estudio. Se estudian diversos métodos que intervienen en el proceso de visualización tridimensional, presentando las características de cada uno de ellos.

Capítulo 3: Se aborda el estudio del algoritmo de visualización "volume rendering"[1], el cual presenta características adecuadas para obtener una representación tridimensional a partir de una conjunto de imágenes bidimensionales.

Capítulo 4: Se hace una introducción al procesamiento paralelo, que incluye los conceptos y definiciones dentro de ésta área. Se estudian diversas organizaciones de procesadores, multiprocesadores y multicomputadoras, empleados en el procesamiento paralelo. Además se presentan las características de algunos lenguajes de programación empleados en las diversas arquitecturas del procesamiento paralelo. Este estudio brinda el marco teórico para el planteamiento de la arquitectura a desarrollar.

Capítulo 5: Presenta la descripción de las etapas definidas en el desarrollo del programa de computación (lenguaje C ANSI paralelo) para la implantación del algoritmo "volume rendering". Se define la organización de la arquitectura de procesadores a utilizar en una plataforma de desarrollo compuesta por procesadores tipo transputer[2] (INMOS T805), en configuraciones de 1, 2, 4, 8 y 16 procesadores. También, se hace referencia a un caso de estudio en donde se plantean las características del conjunto de imágenes a emplear para obtener una imagen tridimensional. Además se presenta la definición del ciclo de procesamiento que se utilizará para evaluar el desempeño de la implantación del algoritmo, en la arquitectura propuesta.

Capítulos 6 y 7: Se presentan los resultados que consisten en la imagen tridimensional generada, con la implantación del algoritmo, utilizando arreglos de 1, 2, 4, 8 y 16 transputers. Se incluyen los tiempos de procesamiento en cada caso, lo que permite calcular diversas métricas para evaluar el desempeño de la aplicación en el sistema. Con estos resultados, finalmente se presentan las conclusiones, y las propuestas de trabajos futuros.

---

[1] "volume rendering" : interpretación de un volumen de información.

[2] "transputer" : circuito integrado que reúne varias unidades de cálculo que operan simultáneamente (procesamiento paralelo), una memoria y múltiples conexiones que permiten un intercambio rápido con otros transputers.

Capítulo 2

Visualización de Datos Tridimensionales

2.1 Introducción

La visualización es un método de computación. Este transforma modelos abstractos dentro de la geometría, permitiendo a los investigadores observar sus simulaciones y cálculos. La visualización ofrece un método para ver lo no visible. Esto enriquece los procesos de los descubrimientos científicos, realzando su profunda y sorprendente agudeza. En muchos campos, esto es ya, una revolución en el camino de los científicos en el forjamiento de la ciencia [McCormick, 1987]. Visualizar es el proceso para crear una escena de manera gráfica de "algo", y ese "algo" puede ser un concepto, una idea, un grupo de datos, un objeto pequeño o grande, etc. Otra definición de visualización se refiere a "Representar con imágenes ópticas fenómenos de otro carácter" [Larousse,1991].

La visualización abarca la compresión y síntesis de la imagen. Esto es, la visualización es una herramienta para la interpretación de la información contenida en la imagen, al ser proporcionada a la computadora, y para la generación de imágenes a partir de conjuntos complejos de datos de tipo multi-dimensional [McCormick, 1987]. Estos estudios de los mecanismos en los humanos y computadoras los cuales permiten a ellos en acuerdo percibir, usar y comunicar la información visible. La visualización unifica extensamente los campos independientes pero convergentes de: la graficación por computadora, el procesamiento digital de imágenes, la observación por computadora, el diseño asistido por computadora, el procesamiento de señales y estudios de interfaz para usuarios.

Richard Hamming observó "el propósito de la computación científica es el discernimiento, no sólo números" [McCormick, 1987]. El objetivo de la visualización es la existencia de métodos científicos para proveer nuevos discernimientos a través de los métodos de visualización. Se estima que un 50 por ciento de las neuronas del cerebro están asociadas con la vista. La visualización en la computación científica intenta identificar, cual es la maquinaria neurológica que actúa.

A continuación se presenta un panorama general en el área de la visualización, en cuanto a conceptos y los principios en cual se basan las diversas técnicas empleadas.

2.2 Conceptos Básicos de la Visualización Tridimensional

1) El campo de la visualización

Las fuentes de datos en la actualidad son como canales que descargan información, algunas de gran volumen se encuentran en: las supercomputadoras, los satélites orbitando alrededor de la tierra, los servicios de inteligencia militar, los datos relacionados a la climatología y astronomía, las naves aeroespaciales que envían datos interplanetarios o planetarios, los escáneres médicos empleados en imágenes de varias modalidades (tales como, de tomografías o de resonancia magnética), etc.

Al existir la posibilidad de que el número de fuentes podría multiplicarse, tanto como la densidad de estas fuentes. Estableciéndose necesidades relacionadas al tratamiento de muchos datos. Por ejemplo, en la definición de una supercomputadora esta el cambio significante de 0.1 a 1 "Gigaflops"[3] a 1-10 "Gigaflops".

Nosotros hablamos y por 5000 años hemos preservado nuestras palabras, pero, nosotros no podemos compartir la observación. Por lo que la comunicación compartida en la visualización podría presentarse si cada uno de nosotros tuviera en la frente un tubo de rayos catódicos (CRT). Este descuido de la evolución en nosotros provoca un atraso en la comunicación visual comparada con la del lenguaje. Para superar la carencia, los científicos se han visto en la necesidad de improvisar la interacción visual tanto con los datos como cada otro participante [McCormick, 1987].

Los científicos podrán aprender a comunicarse visualmente con algún otro. Mucho de la ciencia moderna no puede ser expresada por escrito. Las secuencias de DNA, modelos moleculares, escáneres de imágenes médicas, mapas del cerebro, simulación de vuelos a través de un terreno, simulaciones de flujos de fluidos, y más, todos con la necesidad de estar comunicados visualmente.

Actualmente, analizar los resultados de los cálculos no basta, se requiere el poder interpretarlos e interactuar con ellos. Además, de realizar las operaciones en tiempo real.

2) Visualización en la Computación Científica (ViSC)

Es la producción de representaciones gráficas para un conjunto de datos y procesos científicos de ingeniería y de medicina [Hearn,1995]. Los inicios de ViSC se encuentran al explorar el desarrollo de la ciencia y la visualización, y hace recomendaciones para alimentar este crecimiento y desarrollo. Este propósito inicial para sentar las herramientas de visualización de alta calidad en manos y mentes de investigadores e ingenieros. La ViSC inicialmente recomendaba fundir ambos desarrollos tanto el de investigación como el de la tecnología. Los desarrollos en investigación son responsabilidad de los usuarios de herramientas, los expertos desde la ingeniería y las ciencias cuya disciplina se basa en la

[3] "Gigaflops": billones de operaciones punto flotante por segundo.

computación para sus investigaciones. Los desarrollos tecnológicos están manejados por los hacedores de herramientas, los investigadores en visualización son quienes pueden desarrollar el hardware, software y sistemas requeridos [McCormick, 1987].

Por ejemplo, en cuanto a las modalidades de imágenes calculadas en la trasmisión y emisión de tomografías, imágenes de resonancia magnética y de ultrasonido, actualmente mejoradas por el uso de agentes de contraste, son encaminadas por los nuevos conocimientos sobre el diagnóstico clínico. Las técnicas de visualización tridimensional son esenciales para la comprensión de espacios complejos y, en algunos casos, relaciones temporales entre la características anatómicas con y a través de esta modalidad de imágenes. La computación juega un papel central en el diagnóstico clínico como una información que esta integrada en múltiples imágenes.

El entendimiento de los diseños básicos principales de mallas de elementos finitos bidimensionales es reciente (1984). Ahora, se tiene la tecnología de integración de elementos finitos con la modelación de sólidos más requerida para el entendimiento de procedimientos de diseño de mallas y que clase de técnicas de mapeo de superficies pueden ser incorporadas dentro de los módulos de elementos finitos.

III) Dispositivos de Despliegue en video

El principal dispositivo de salida en un sistema de gráficas es un monitor de video. La operación de la mayor parte de los monitores de video se basa en el diseño estándar de CRT, pero existen otras tecnologías como los monitores de estado sólido.

IV) Dispositivos de vista tridimensional

Los monitores gráficos para desplegar escenas tridimensionales se diseñan utilizando una técnica que refleja una imagen de CRT de un espejo flexible vibrante. Estas vibraciones están sincronizadas con el despliegue de un objeto en un CRT, de manera que cada punto del objeto se refleje del espejo a una distancia espacial correspondiente a la distancia de ese punto desde una posición de vista específica [Hearn,1995].

V) Visión estereoscópica y realidad virtual

El despliegue de vistas estereoscópicas, en un método que no produce imágenes tridimensionales reales, pero ofrece un efecto tridimensional al presentar una vista diferente para cada ojo del observador, de manera que las escenas parecen tener profundidad [Overington,1992]. La vista estereoscópica es un componente de los sistemas de realidad virtual, donde los usuarios pueden entrar en una escena e interactuar con el entorno.

VI) Software de Graficación

Se tiene dos tipos de clasificación para el software de graficación: a) paquetes generales de programación, ofrecen un amplio conjunto de funciones gráficas que se pueden utilizar en un lenguaje de programación de alto nivel, como C o FORTRAN; y b) paquetes de aplicaciones para propósitos especiales, están diseñados para usuarios que no son programadores, de modo que puedan generar despliegues sin preocuparse por el desarrollo de las operaciones gráficas, como el diseño asistido por computadora (Computer Asisted Drawing [CAD]) [Hearn,1995].

2.3 Métodos de Despliegue Tridimensional

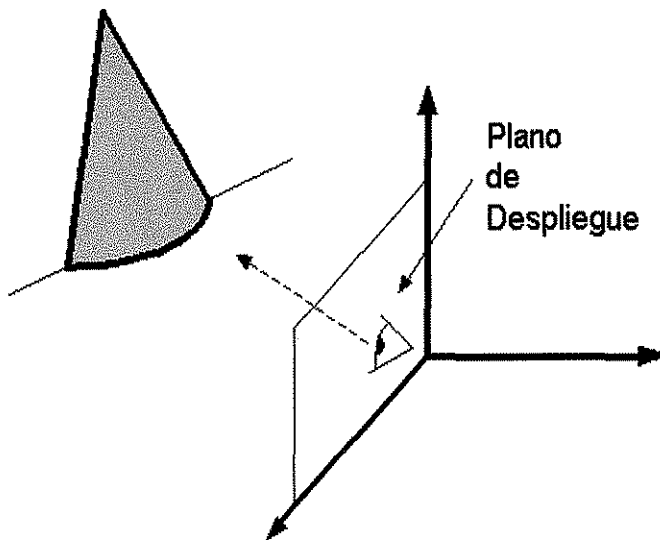


Fig. 2.1 Plano de Despliegue y sistema coordenado de referencia.

En los métodos de despliegue tridimensional, primero se debe establecer un sistema de coordenadas de referencia, que defina la posición y orientación para el plano de proyección (ver figura 2.1). De esta manera se transfieren las descripciones de los objetos a las coordenadas de referencia y se proyectan sobre el plano de despliegue seleccionado. Así, los objetos se pueden desplegar en forma de contorno o aplicar técnicas de iluminación [Watt, 1993].

2.3.1 Proyección paralela

Un método para generar una vista de un objeto sólido consiste en proyectar puntos en la superficie del objeto a lo largo de líneas paralelas sobre el plano de despliegue, con la posibilidad de seleccionar diferentes posiciones de dichas vistas, como se muestra en la figura 2.2(A). En una proyección paralela, las líneas paralelas en la escena están en coordenadas convencionales y se proyectan en las líneas paralelas del plano de despliegue bidimensional. De esta manera, se puede reconstruir la apariencia del objeto sólido a partir de las vistas principales [Berger, 1991].

2.3.2 Proyección de perspectiva

Otro método para generar una vista de una escena tridimensional es proyectar puntos hacia el plano de despliegue a lo largo de trayectorias convergentes. Este hace que los objetos que están más lejos de la posición de vista se desplieguen más pequeños que aquellos del mismo tamaño que se encuentran más cerca de la posición de vista (ver figura 2.2(B)). En una proyección de perspectiva, las líneas paralelas en una escena que no son paralelas al plano de despliegue se proyectan en líneas convergentes. Las escenas que se despliegan utilizando proyecciones de perspectiva parecen más reales, como se muestra en la figura 2.2(C), ya que ésta es la manera en que los ojos y el lente de una cámara forman las imágenes [Berger, 1991].

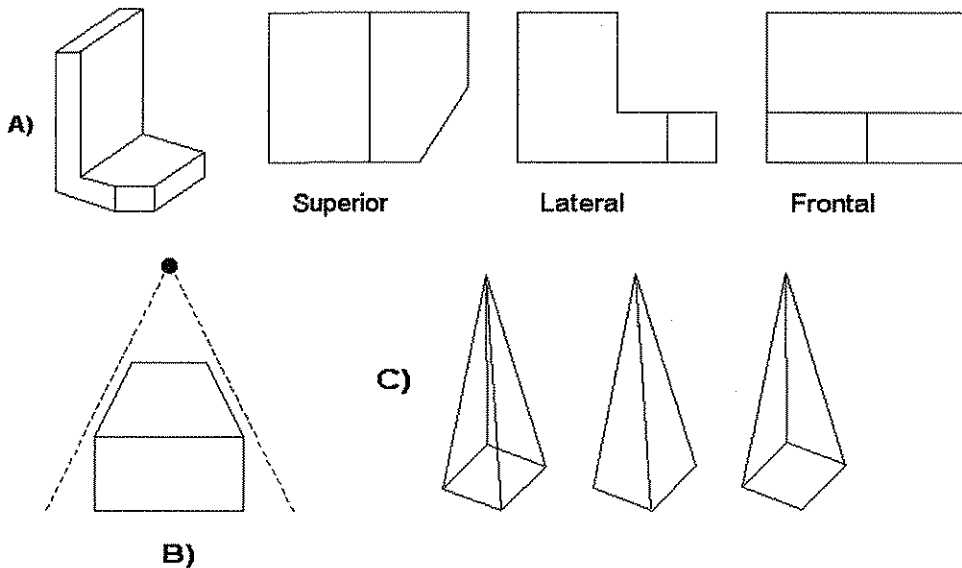


Fig. 2.2 A) Proyección paralela, B) Proyección de perspectiva y C) Representación en perspectiva de una pirámide.

2.3.3 Indicación de la intensidad

La información de la intensidad es importante para poder identificar con facilidad, para una dirección de vista en particular, el frente y la parte de atrás de los objetos desplegados. Se puede presentar una ambigüedad cuando un objeto es desplegado sin la información de la intensidad. Un método sencillo para indicar la intensidad consiste en variar la intensidad de los objetos de acuerdo con su distancia dentro de la escena al plano visual [Hearn,1995].

Una aplicación de la indicación de la intensidad es en el modelado del efecto de la atmósfera en la intensidad que se percibe de los objetos. Los objetos más distantes tienen una apariencia más tenue que los objetos más cercanos debido a la dispersión de la luz por partículas de polvo, neblina y humo, entre otros factores.

2.3.4 Identificación de superficies y línea visible

Las relaciones de profundidad se identifican de alguna manera en las líneas visibles. El método más sencillo consiste en realzar las líneas visibles o desplegarlas en un color diferente. Otra técnica, es el despliegue de las áreas no visibles como líneas de rayas, o bien otro planteamiento consiste en eliminar sólo las líneas no visibles. Al eliminar las líneas ocultas, también se elimina información acerca de la forma de las superficies traseras del objeto. Estos métodos de línea visible también identifican las superficies visibles de los objetos.

2.3.5 Representación de superficies

Se logra un mayor realismo en los despliegues al establecer la intensidad de la superficie de los objetos de acuerdo con las condiciones de iluminación en la escena y según las características de superficie asignadas. Las especificaciones de iluminación incluyen la intensidad y posiciones de fuentes de luz y la iluminación de fondo general que se requiere para una escena. Las propiedades de la superficie de los objetos incluyen el grado de transparencia y cuán ásperas o lisas deben ser las superficies. Así se pueden aplicar procedimientos para generar la iluminación y regiones de sombreado correctas para la escena. Al combinar los métodos de presentación de superficie con identificación de perspectiva y de la superficie visible generan un mayor grado de realismo en una escena desplegada [Hearn, 1995].

2.3.6 Vistas separadas y de recortado

Las vistas separadas y de recortado de objetos se pueden utilizar entonces para mostrar la estructura interna y la relación con las partes del objeto. Una alternativa para separar un objeto en sus componentes es la vista de recortes, que elimina parte de las superficies visibles para mostrar la estructura interna.

2.3.7 Vistas tridimensional y estereoscópicas

Otro método para agregar un sentido de realismo a una escena generada por computadora consiste en desplegar los objetos, utilizando vistas tridimensionales o estereoscópicas. Los dispositivos estereoscópicos presentan dos vistas de una escena: una para el ojo izquierdo y la otra para el derecho. Se generan las dos vistas al seleccionar posiciones de referencia que corresponden a las posiciones de los dos ojos de un observador individual. Ambas vistas se pueden desplegar en ciclos de actualización alternativa de un monitor de rastreo y es posible verlos a través de lentes que oscurecen de manera alternativa, primero un lente y luego el otro en sincronía con los ciclos de actualización del monitor [Castleman, 1979].

2.4 Representaciones Tridimensionales de Objetos

Las escenas de gráficas pueden contener clases diferentes de objetos (árboles, flores, nubes, rocas, agua, ladrillos, paneles de madera, hule, papel, mármol, acero, cristal, plástico y tela, por mencionar sólo unas cuantas). Es sorprendente que no exista ningún método que se pueda utilizar para describir los objetos que incluya todas las características de los materiales presentes.

Las superficies cuadráticas y de polígonos proporcionan descripciones exactas de los objetos euclidianos sencillos, como poliedros y elipsoides, las superficies de "spline"[4]. Las técnicas de construcción son útiles para diseñar las alas de las aeronaves, engranes y otras estructuras de ingeniería con superficies curvas, los métodos de procedimientos, como las construcciones de fractales y los sistemas de partículas, nos permiten obtener representaciones exactas para las nubes, de hierba y otros objetos naturales; las imágenes médicas de tomografías computarizadas y los despliegues de isosuperficies, las representaciones de volumen y otras técnicas de visualización se aplican a los conjuntos de datos discretos tridimensionales para obtener representaciones visuales de los datos [Berger, 1991].

Modelación de sólidos

La necesidad de modelar objetos como sólidos ha dado lugar al desarrollo de diversas técnicas para representarlos, debido a la existencia de una demanda de las aplicaciones del modelado de sólidos. Por ejemplo, a partir de la representación satisfactoria de un objeto sólido se pueden generar instrucciones de manera automática para que con máquinas controladas por computadora se produzcan el objeto o los prototipos, ya sea, usando una técnica como la estereolitografía, que es un proceso en el cual se emplea un rayo láser para formar un objeto sólido a partir de un "baño" de plástico fundido. Además de emplear técnicas de graficación, como el modelado de la opacidad con refracción, que depende de la posibilidad de determinar por dónde entra y sale la luz de un objeto sólido [Foley, 1996].

La representación de sólidos

En teoría, un esquema de representación debe hacer imposible la creación de una representación inválida (es decir, una que no corresponda a un sólido). Además, debe ser fácil crear una representación válida, generalmente con ayuda de un sistema interactivo de modelado de sólidos [Foley, 1996]. Una representación debe ser compacta (ahorro de espacio), y en el caso de un sistema distribuido se ahorra tiempo de comunicación. Por último, debe permitir la utilización de algoritmos eficientes para calcular las propiedades físicas deseadas, y lo principal crear imágenes.

---

[4] "spline": Es una *función* que está formada por varios polinomios, cada uno definido sobre un subintervalo, que se unen entre sí obedeciendo a ciertas condiciones de continuidad, generando una trayectoria.

Los esquemas de representación para los objetos sólidos con frecuencia se dividen en dos amplias categorías:

a) Las representaciones de frontera describen un objeto tridimensional como un conjunto de superficies que separan del entorno al interior del objeto. Algunos ejemplos típicos de las representaciones de frontera son las facetas de polígonos y los paneles de "spline".
b) Las representaciones de división de espacio se utilizan para describir las propiedades interiores, al dividir la región espacial que contiene un objeto de sólidos contiguos, pequeños y que no se superponen. Una descripción común de división de espacio para un objeto tridimensional es una representación de árbol octal.

2.4.1 Representación y modelación de objetos geométricos

Un número de formas de representación de objetos tridimensionales han sido desarrolladas en la graficación por computadora. Algunos de estas surgen de las aplicaciones y de las estructuras de datos que determinan por completo la estrategia de modelación. Por ejemplo, en la modelación de sólidos, un método conocido como CSG (Computer Solid Geometric), es una forma de representación y un método que facilitan una forma de interacción gráfica que habilita la ingeniería de partes a ser construidas (algunas formas son determinadas por los algoritmos de interpretación).

Los factores que se consideran en la representación son:

a) La estructura de los datos, la forma de los algoritmos de procesamiento y el diseño del hardware con programas fijos.
b) El costo del procesamiento de un objeto a través de un "pipeline"[5] tridimensional.
c) La apariencia final de un objeto, algunas formas son más aproximadas que otras.
d) La facilidad o diferente edición de la figura de un objeto.

Se presenta una lista de las formas de representación en el orden de importancia y frecuencia de uso.

1) Poligonal: Los objetos son aproximados por medio de una red de trazas poligonales planas.

2) Parches con parámetros bicúbicos: Los objetos son representados por redes de elementos llamados parches. Estos son polinomios con dos variables paramétricas (usualmente cúbicos).

3) CSG: Usados en la modelación de sólidos, un objeto es representado por una colección de objetos "elementales" tales como esferas, cilindros y cubos.

---

[5] "pipeline": Proceso vectorial, en donde se ejecutan de las operaciones repetidamente a un flujo de datos.

4) Técnicas de subdivisión espacial: Es la incrustación de un objeto en un espacio en donde los puntos en el espacio son etiquetados de acuerdo a la colocación del objeto.

Las cuatro representaciones son esquemas que aproxima la forma de un objeto. Por otro lado, las formas 2 y 3 son esencialmente representaciones. Otra categorización que es importante es la representación de superficies o fronteras del objeto, ya sea que el volumen por completo del objeto sea representado. Las formas 1 y 2 son representaciones de fronteras, y las 3 y 4 son representaciones volumétricas [Watt, 1993].

2.4.2 Representación poligonal

Esta es una forma de representación clásica en la graficación tridimensional. Un objeto es representado por una malla. En el caso general, un objeto posee superficies curvas y las trazas son una aproximación a tal superficie. Una representación en malla de polígonos es llamada formalmente una representación de fronteras o la conocida representación B (Binaria) [Foley, 1996], porque es esta una descripción geométrica y topológica de las fronteras o superficies del objeto.

Las representaciones poligonales son inequívocas en la graficación por computadora. La modelación o creación poligonal de objetos es directa, sin embargo, hay ciertas dificultades prácticas. La calidad de la modelación, o la diferencia entre la representación por trazas y la superficie curva de un objeto, son usualmente arbitrarias. Depende de que tanto la calidad de la imagen final es concebida, el tamaño de los polígonos individuales podría idealmente depender de la curvatura local espacial. De tal forma que donde la curvatura cambia abruptamente, más polígonos son requeridos por unidad de área de la superficie, como se muestra en la figura 2.3. Estos factores tienden a hacer relacionadas con el método usado en la creación de los polígonos, por ejemplo, en una malla esta comenzando a construirse a partir de un objeto existente, mediante el empleo de un digitalizador tridimensional, para determinar las coordenadas espaciales de los vértices del polígono, el operador podrá decidir el base a la experiencia cual debe ser el tamaño de cada polígono [Watt, 1993]. Algunos polígonos son extraídos algorítmicamente (por ejemplo, la creación de un objeto por barrido o con un algoritmo de subdivisión de parches bicúbicos), y es posible con un aprovechamiento riguroso de la relación de polígonos por unidad de área de la superficie.

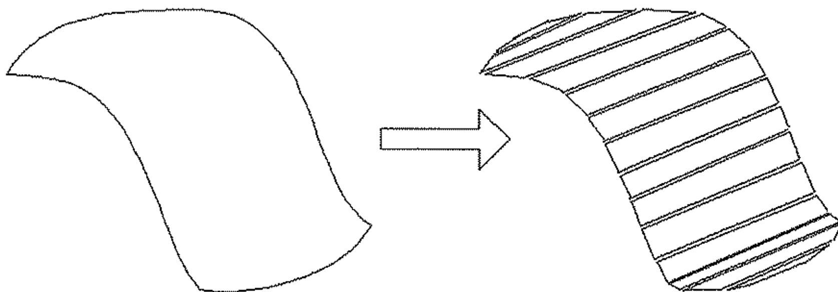


Fig. 2.3 Aproximación de una superficie curva usando polígonos.

Uno de los desarrollos más significativos en la graficación tridimensional fue el surgimiento de los algoritmos para sombrear, que en hacen un eficiente tratamiento con los objetos poligonales, y al mismo tiempo, a través de la esquema de interpolación disminuyen el efecto visual con una adecuada linealización en la representación de la pieza. Este factor, junto con los recientes desarrollos en la programación a nivel hardware de la interpretación, tiene asegurado el arraigo de la estructura de la malla de polígonos. De hecho, una estructura de malla de polígonos es usada, no solamente como una modelación de la estructura de datos, sino como una forma intermedia para muchas otras estructuras de datos. Una estructura de datos fuente que es convertida a polígonos como un todo, en base a la estrategia de interpretación obteniendo como resultado una baja complejidad en la decodificación requerida para interpretar directamente la descripción a partir de los parches. Este mismo enfoque es tomado en la estructuras CSG.

En el caso más simple de una malla de polígonos, consiste de polígonos representados por una lista de coordenadas (x,y,z) que son los vértices de los polígonos. La información almacenada para describir un objeto finalmente es una lista de puntos o vértices. También puede almacenar, como parte de la representación del objeto, información geométrica que es usada en subsecuentes procesos. Toda esta información es conveniente almacenarla dentro de una estructura de datos y algunos de ellos pueden padecer transformaciones lineales que son aplicadas al objeto. El orden jerárquico simple que tienen los polígonos dentro de una estructura es conveniente, ya que los polígonos son agrupados en superficies y las superficies son agrupadas en objetos (ver figura 2.4) [Watt, 1993]. Por ejemplo, un cilindro tiene tres superficies, que corresponden a: la tapa, la base y una curva. La razón de esta agrupación es que puede distinguirse entre los bordes que son parte de la aproximación de los que existen en la realidad.

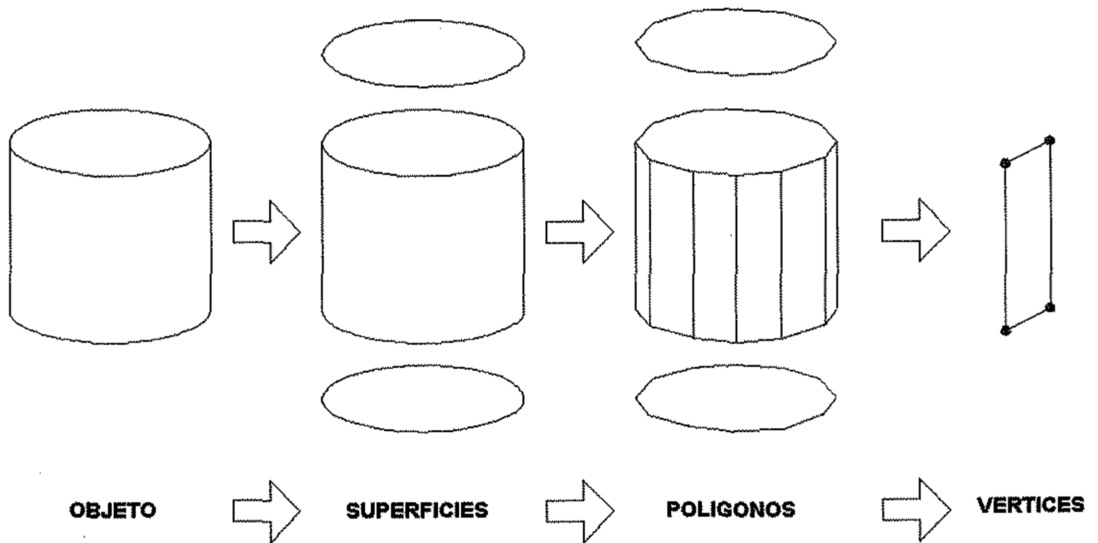


Fig. 2.4 Un objeto de acuerdo a un orden jerárquico.

Una característica importante de esta representación es que los polígonos son entidades independientes tratadas como tales por el interpretador. Esto tiene sus implicaciones dentro del diseño del interpretador, la más popular de las estrategias de interpretación es tener polígonos independientemente del procesamiento, que entran en el marco de la memoria usada para superficies ocultas con el algoritmo de "buffer-Z"[6], junto con el sombreado por interpolación.

2.4.2.1 Modelación de objetos mediante polígonos

Aunque la malla de polígonos es la forma mas común de representación en la graficación por computadora, la modelación resulta ser tediosa. La popularidad de ésta deriva de la facilidad de la modelación, y del empuje de la estrategia de interpretación, a nivel software y hardware, para procesar los polígonos de los objetos. Lo más importante es que nos hay restricción en la forma y complejidad del objeto que es modelado.

El desarrollo interactivo de un modelo es posible por el muestreo de los vértices con un dispositivo localizador tridimensional, pero intervienen factores importantes (transparentes al usuarios) al ser colocado el objeto, como tal, en el sistema. El principal requerimiento es asegurar una adecuada aproximación al ser representado un objeto [Watt, 1993]. Por ejemplo, si el usuario hace un cambio en la curvatura espacial de la aproximación del objeto en la actual malla de polígonos (moviendo uno o más vértices), entonces, las nuevas trazas deben ser viables para mantener una aproximación adecuada.

Los tres ejemplos comunes de la estrategia de modelación con polígonos, son:

1) Emplear un digitalizador tridimensional o adoptar una tarea manual similar.
2) Usar un dispositivo automatizado basado en un láser.
3) Generar un objeto a partir de una descripción matemática.
4) Generar un objeto por barrido.

Los dos primeros métodos de modelación convierten un objeto real a una malla de polígonos, los siguientes dos son a partir de definiciones.

2.4.2.2 Modelación manual de objetos mediante polígonos

El camino más fácil para modelar un objeto real es usando manualmente un digitalizador tridimensional. El operador hace uso de su experiencia y juzga al emplazar puntos sobre un objeto, los cuales son transformados a vértices. Las coordenadas tridimensionales son la entrada al sistema de una forma simple. Una estrategia común para asegurar una adecuada representación, es dibujar una red sobre la superficie del objeto, donde las líneas de intersección de la red curvada definen la posición de los vértices de los polígonos [Watt, 1993].

---

[6] "buffer-Z": Técnica de búfer que mantiene una lista ordenada de los objetos que son visibles para la luz.

2.4.2.3 Generación automática de objetos mediante polígonos

Un dispositivo basado en el láser tiene la capacidad de crear objetos de alta precisión o alta resolución, con mallas de polígonos a partir de objetos reales. El objeto es colocado en una tabla de rotación en la ruta del haz, esta se mueve verticalmente de arriba abajo. El láser genera un conjunto de contornos (la intersección del objeto y un grupo de cerrado de planos paralelos espaciados), a partir de la medición de la distancia que se tiene a la superficie del objeto. En un algoritmo tipo "skin"[7], la operación en los pares de contornos (ver figura 2.5), convierte el dato de la frontera a un número extenso de polígonos [Watt, 1993].

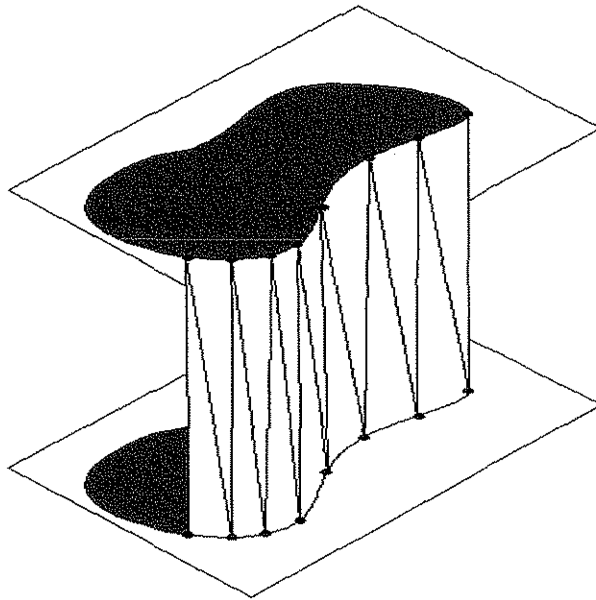


Fig. 2.5 Algoritmo tipo "skin" une puntos de contornos consecutivos para crear un objeto tridimensional en base a polígonos.

2.4.2.4 Modelación matemática de objetos mediante polígonos

La generación de un objeto a partir de una descripción matemática puede ser la más fácil de obtener, como lo es, el barrido de una sección transversal [Foley, 1996]. La resolución de los polígonos es fácilmente controlada por el algoritmo de generación, pero los problemas en la resolución de la forma pueden presentarse. En el caso de un toroide, generado por el barrido de una sección transversal circular, alrededor de una trayectoria circular, los polígonos serán amplios en la cara frontal del toroide comparadas con la de la cara posterior, como se muestra en siguiente figura.

[7] "skin": capa superficial tipo piel.

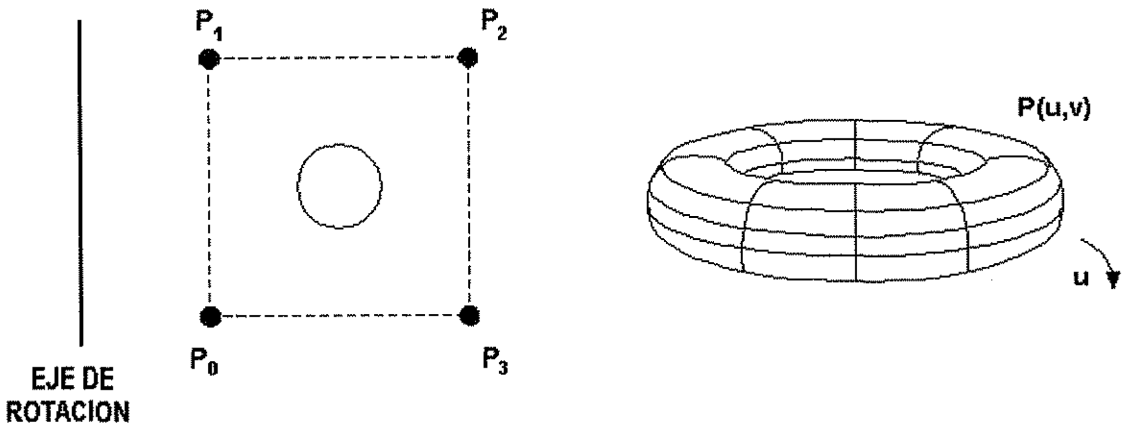


Fig. 2.6 Modelación por barrido.

2.4.2.5 Generación por barrido de objetos mediante polígonos

La idea previa puede ser generalizada con algunas extensiones. La primera, se refiere a poder barrer una sección transversal a lo largo de una trayectoria curva. La trayectoria a través de la cual la sección transversal es barrida puede ser una curva arbitraria, es decir, generada interactivamente por el usuario al aplicar la técnica. En segundo término, la sección transversal puede estar variando su forma en el momento de ser barrida. Las generalizaciones habilitan la producción de objetos llamados ductos sólidos o cilindros generalizados.

2.4.3 Mallas de parches con parámetros bicúbicos

A partir de las mallas de polígonos de manera sencilla se tienen las mallas de parches. Considerando una malla de polígonos de cuatro lados en la aproximación a una superficie curva, en una malla de parches con parámetros se puede considerar un conjunto de polígonos curvilíneos que representan a la superficie. Se nota que un parche es una superficie curva y que cada punto en el parche esta definido. Esta definición Q(u,v), es en términos de dos parámetros u y v, donde u >=0, v =<1, y la función Q es un polinomio cúbico. Para los valores de los coeficientes en los términos cúbicos de Q(u,v), se propone un especial y conveniente camino en la definición de éstos, al usar 16 puntos tridimensionales conocidos como puntos de control. Cuatro de ellos son los puntos de la esquina de un parche, tal como es usada la definición de una predefinida forma de un polinomio, se tiene una función básica, donde los 16 puntos de control son considerados dentro de esta definición y una única Q(u,v), es obtenida. La forma del parche es determinada completamente a partir de la posición de los puntos de control [Watt, 1993]. Un ejemplo, es un simple parche Bézier y la relación entre su forma y los puntos de control, ver figura 2.7.

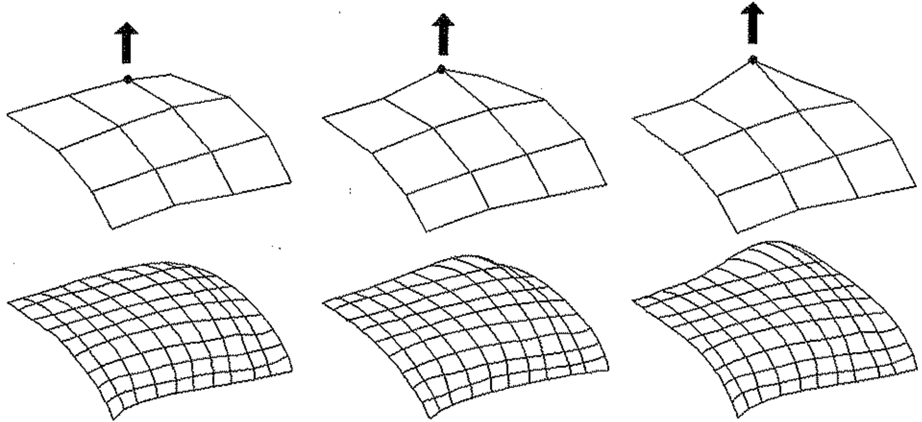


Fig. 2.7 Relación entre la posición de los puntos de control y el forma de los parches.

Para más aplicaciones, la modelación de los objetos o en la construcción de una representación desde una estructura de datos de un objeto tridimensional, es más difícil cuando son empleados los parches bicúbicos (objetos reales muy complejos son fácilmente convertidos a polígonos usando un digitalizador tridimensional y el software de operación). Los 16 puntos de control tienden a ser especificados para cada parche y hay un problema práctico más significante, para el mantenimiento de la integridad de la representación, las restricciones de continuidad requieren ser mantenidas allí en todas las fronteras. Un parche no puede estar en un conjunto sin observar a sus vecinos, y las descripciones de los parches pueden ser generadas de manera semi-automática.

Algunas de las ventajas de esta modelación, se tienen en que la representación es "fluida" y el empleo de software que ajusta la posición de los puntos de control, de tal manera que el objeto puede ser ajustado. Sin embargo, esto no es tan fácil, porque se requiere de la continuidad entre los parches. Otra, esta en la forma analítica de las propiedades de la masa (tales como el volumen, el área de la superficie y momentos de inercia) que pueden ser extraídas a partir de la descripción. Esta propiedad es totalmente explotada en los sistemas CAD.

Finalmente, la representación de objetos con lata resolución tridimensional puede demandar volúmenes de memoria extensos. Esto significa que tanto el costo de la memoria es alto, como las fallas generadas en el tiempo de transferencia en la base de datos.

2.4.3.1 Modelación de objetos mediante mallas de parches

Existen dos estrategias de modelación práctica que pueden ser empleadas para crear modelos de parches con parámetros. Ambas, requieren de algunos conocimientos detallados de la teoría de la representación de parches con parámetros bicúbicos.

El primer método, llamado adaptación de superficie, es básicamente un método particular de hacer la interpolación entre superficies. De un conjunto de puntos se tienen disponibles cuales de estos muestran la superficie del objeto al ser representado, y una descripción del parche es creada a partir de los mismos. La adaptación de una línea es por

medio de un número de puntos en un plano, análogamente adaptar una superficie es a través de un número de puntos en el espacio tridimensional. Así, el método es adecuado para la modelación de objetos a partir de alguna abstracción o adaptación de una superficie a puntos obtenidos por la digitalización de un objeto real. La diferencia, entre esta técnica y la obtención de una representación de malla de polígonos, es que una superficie continua es obtenida a partir de los puntos digitalizados [Watt, 1993].

Sin embargo, en el caso de un objeto real, la superficie del parche no corresponderá exactamente a la superficie de la cual los puntos fueron digitalizados. La exactitud de la representación depende del número de puntos a través de los cuales la superficie es adaptada o la magnitud espacial de los parches contenidos en la red.

El principio del procesos se muestra en la figura 2.8. Se inicia con un conjunto de puntos en el espacio tridimensional. El siguiente paso adapta una curva a través de los puntos en dos direcciones de los parámetros, u y v. Esta curva de la red es divida en conjuntos de cuadriláteros curvilíneos. De cada cuadrilátero, los puntos de control para un parche individual son obtenidos, la curva de la red esta "contenida dentro" de los parches de las superficies.

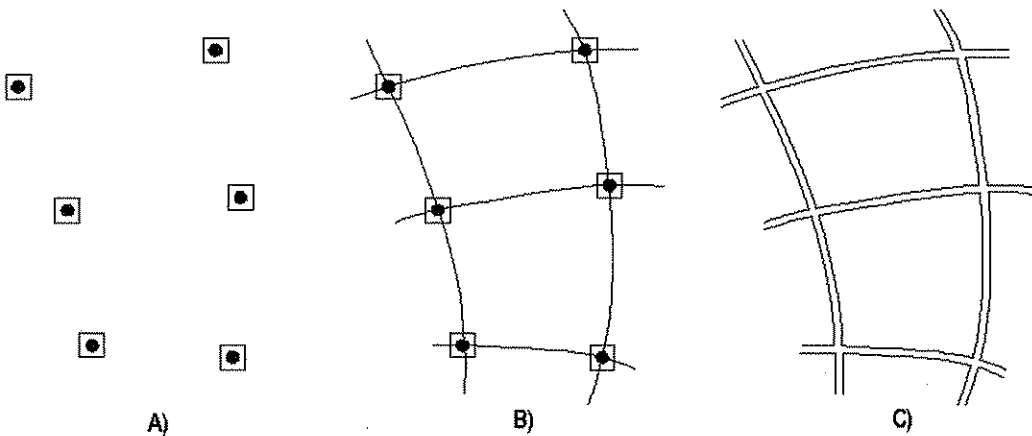


Fig. 2.8 Representación esquemática de la adaptación de superficie A) Conjunto de puntos tridimensionales, B) Adaptación de las curvas a través de los puntos con dos parámetros de direcciones, C) Cuadricula de curvas a partir de las fronteras de los parches.

El segundo método en la modelación de parches con parámetros, es el barrido de la sección transversal. Tal como la modelación de malla de polígonos, se define un eje o curva de barrido, esta será una curva cúbica. La sección transversal es definida a partir de una curva cúbica simple, como la mostrada en la figura 2.9, o como un conjunto de segmentos de curvas cúbicas. La sección transversal curva es colocada en intervalos apropiados, usando las mismas técnicas de la modelación de objetos mediante polígonos. Respecto a la figura 2.9, esta muestra las secciones transversales curvas contenidas en dos puntos consecutivos de muestro a lo largo de la curva de barrido. Otro par de curvas cúbicas son barridas fuera de los puntos finales del segmento de la sección transversal. Estas cuatro curvas forman la frontera de un parche y una descripción del parche es obtenida.

Incluyendo otra sección transversal en el siguiente punto de muestreo en la curva de barrido, habilitará a otro parche para ser definido y de esta forma continuar [Watt, 1993].

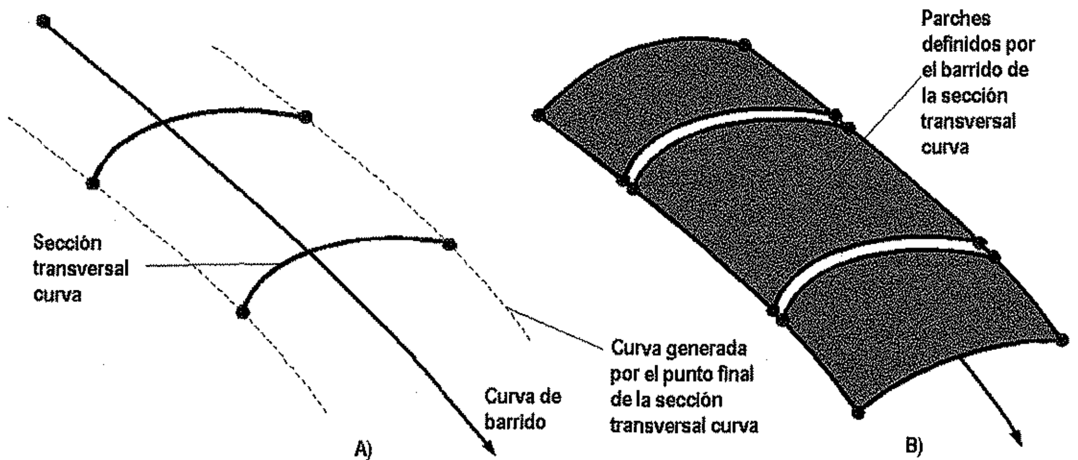


Fig. 2.9 Generación de parches con parámetros bicúbicos por barrido de una sección transversal curva a lo largo de una trayectoria curva.

2.4.4 Geometría Sólida Constructiva

La motivación para este tipo de representación es que proporciona un modo interactivo para la modelación de sólidos. La idea es que los objetos son partes que serían eventualmente manufacturadas por repartición o maquinación, y estos pueden ser construidos sobre encimados por la combinación simple de elementales objetos llamados primitivos geométricos. Los primitivos son, por ejemplo, esferas, conos, cilindros o sólidos regulares, estos son combinados usando un conjunto de operadores booleanos y transformaciones lineales [Foley, 1996]. Un objeto es almacenado en una estructura tipo árbol. Los niveles contienen primitivos simples y los nodos almacenan operadores o transformaciones lineales. La representación no sólo define la forma de un objeto sino también la modelación histórica es posible en una tipo de edición. Por ejemplo, al incrementar el diámetro de un agujero en un sólido rectangular, significa una alteración trivial del radio de primitivo cilindro cuando es incrementado. Esto contrasta con la malla de polígonos de representación B, donde la misma operación distintamente no-trivial. Aunque cada polígono constituyente de la superficie cilíndrica es fácilmente accesible en un esquema jerárquico, generar un nuevo conjunto de polígonos significa la reactivación de cualquier procedimiento de modelación que fue usado para crear los polígonos originales.

El conjunto de operadores booleanos son usados tanto en la representación y como en una técnica de la interfaz del usuario. El usuario específica los sólidos primitivos y combina éstos usando el conjunto de operadores booleanos, y la representación del objeto es un reflejo de las operaciones definidas. Se puede decir que la modelación y la representación no están separadas, ya que la actividad de modelación inicia con la representación.

La figura 2.10 muestra una representación usando geometría sólida constructiva, que presenta la construcción de un simple objeto. Tres originales sólidos aparecen en los niveles del árbol, dos cajas y un cilindro. Los cajas son combinadas usando la operación unión y un orificio es perforado en una de las cajas, con la definición de la representación un cilindro es sustraído del ensamble de la cajas [Watt, 1993].

Aunque existen ventajas substanciales en la representación CSG, esta presenta ciertos problemas. Uno práctico, es el tiempo de cálculo requerido para producir una imagen como resultado de la interpretación del modelo. Otro se refiere a que el método impone restricciones en la operaciones disponibles para crear o modificar un sólido.

Las operaciones booleanas son globales (afectan totalmente al sólido). En cuanto a las locales, al tener una detallada modificación compleja para una de las secciones de un objeto, no es fácilmente implementada al emplear el conjunto de operaciones.

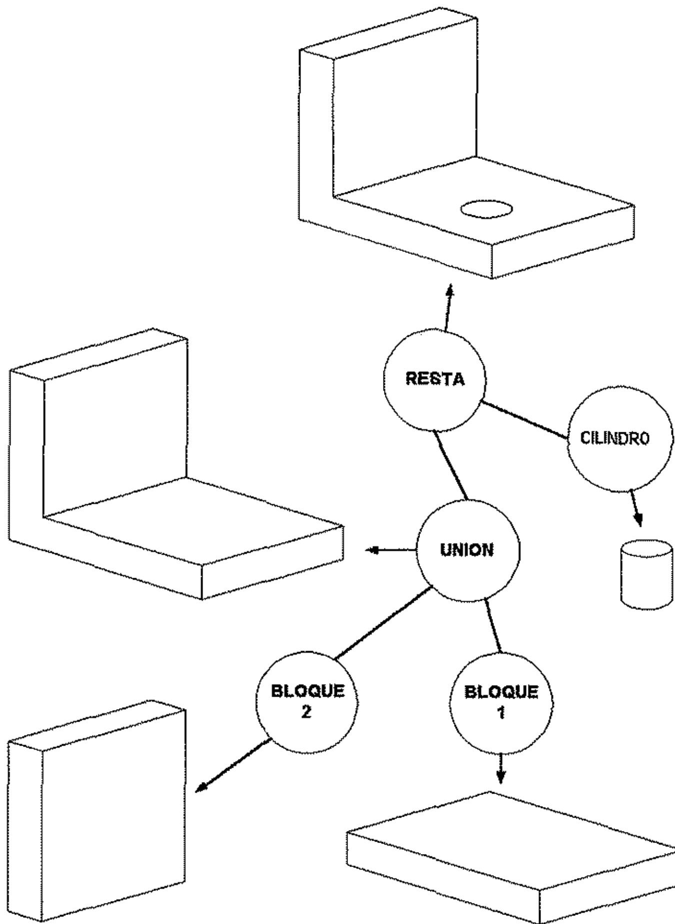


Fig. 2.10 Árbol CSG para la construcción de un objeto simple a partir de tres primitivos.

2.4.5 Técnicas de subdivisión espacial

Las técnicas de subdivisión espacial son métodos que consideran todo el espacio del objeto y etiquetan cada punto del espacio de acuerdo al espacio ocupado por objeto. A través de esta técnica se tiene un esquema, que puede dividir a todo el espacio de trabajo en elementos cúbicos o regulares, y cada uno etiquetarlo. Estos elementos con frecuencia se les denominan elementos de volumen (o voxeles, por analogía con los pixeles) [Foley, 1996]. Esto resulta costoso, en términos de consumo de memoria, en relación a los diversos esquemas disponibles que implementen una organización estructural básica en la asignación de las etiquetas a los voxeles.

En la graficación tridimensional por computadora, este tipo de representaciones de datos son usadas con frecuencia como una estructura secundaria o auxiliara de datos. Por ejemplo, la interpretación de objetos que están representados por un árbol de CSG no resulta sencilla, puesto que, primero se convierte el árbol de CSG a una estructura de datos intermedia a un tipo de subdivisión espacial a partir de ella se interpretan los datos.

Otro ejemplo, es el uso de la subdivisión espacial en el empleo de un haz de moldeador. Lo que implica un examen en la expansión de las intersecciones, que arrastran al espacio del voxel una característica del haz dentro de un intervalo. Esto requiere, de una búsqueda exhaustiva de la estructura primaria de datos para las posibles intersecciones.

2.4.5.1 Árboles octantes

Los árboles octantes, son estructuras jerárquicas que se utilizan para representar objetos sólidos en algunos sistemas gráficos. Las imágenes médicas y otras aplicaciones que requieren del despliegue de cortes transversales de los objetos por lo general utilizan representaciones de árboles octantes. En la estructura del árbol cada nodo corresponde a una región de espacio tridimensional, además de reducir los requerimientos de almacenamiento para los objetos tridimensionales [Watt, 1993]. El procedimiento de codificación de árboles octantes se basa en la llamada codificación del árbol de cuadrantes (quadtrees). Estos árboles cuadráticos se generan al dividir, en forma sucesiva una región bidimensional, en cuadrantes y cada nodo en el árbol cuadrático tiene cuatro elementos de datos, uno para cada uno de los cuadrantes de la región. En un esquema de codificación, el árbol octante se divide en regiones del espacio tridimensional (por lo general en cubos) en octantes y almacena a ocho voxeles (ver figura 2.11). Si todos los voxeles en un árbol octante son del mismo tipo, este valor se almacena en el elemento de datos del nodo que corresponde, en regiones vacías se representan mediante el tipo de voxel "vacío".

Una vez que se ha obtenido la representación del árbol octante para un objeto sólido, se pueden aplicar a el sólido diferentes rutinas de manipulación. Un algoritmo para llevar a cabo operaciones de especificación, se puede aplicar a dos representaciones de árbol octante de la misma región de espacio.

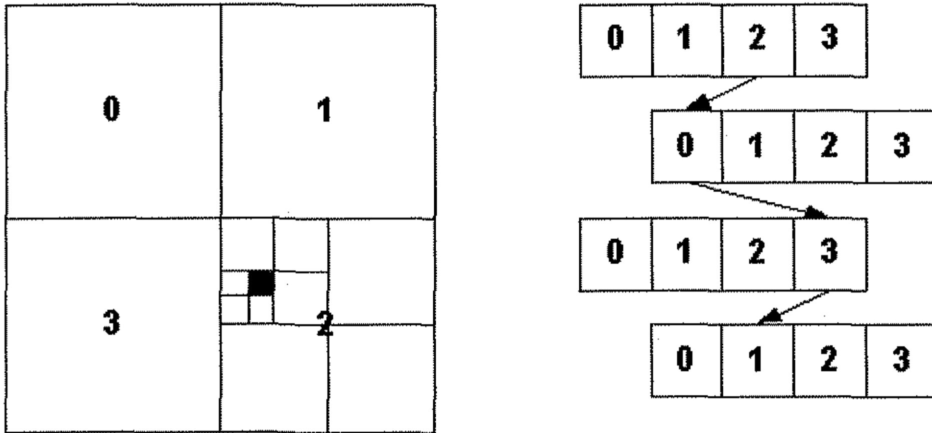


Fig. 2.11 Árboles octantes.

2.4.6 Estrategias de interpretación ("rendering"[8])

La interpretación es una palabra que ha venido a significar "el conjunto de operaciones necesarias para proyectar una vista de un objeto o una escena a un plano visual que las contenga" [Watt, 1993]. Por ejemplo, un objeto al ser iluminado se calcula su interacción con la fuente de luz para producir el sombreado de la escena. Las mejores estrategias de interpretación están determinadas por la representación del objeto y las opciones disponibles de algoritmos para los procesos internos.

2.4.6.1 Interpretación de objeto formados por polígonos

Los objetos formados por polígonos son la forma más común en la graficación por computadora. Ahora, se encuentran hardware (estaciones de trabajo para graficación), que tienen grabados los programas que interpretan uno o más objetos a partir de una base de datos donde se tiene la información de los polígonos. La entrada a un interpretador de polígonos es una lista de polígonos y la salida es un color para cada píxel de la pantalla. La mayor ventaja estos interpretadores de polígonos es que los algoritmos que emplean en su la manipulación, los consideran como unidades o entidades simples. Esto hace al procesamiento simple y rápido, aunque pueden ser dispensadas cuando se tienen objetos más y más complejos.

En el proceso de interpretación, considerar una descripción del objeto a través de un número de espacios coordenados rompe con el esquema tradicional. Ahora, en cada espacio, las operaciones son agregadas en la salida, los diferentes espacios coordenados facilitan ciertos procesos y especificaciones, como se muestra en la figura 2.12. Un objeto almacenado en una base de datos, usando para los vértices coordenadas que son representadas en un sistema coordenado de modelación. El origen puede ser un punto convenientemente localizado en el objeto mismo, por ejemplo, la esquina de un cubo.

[8] "rendering": interpretación.

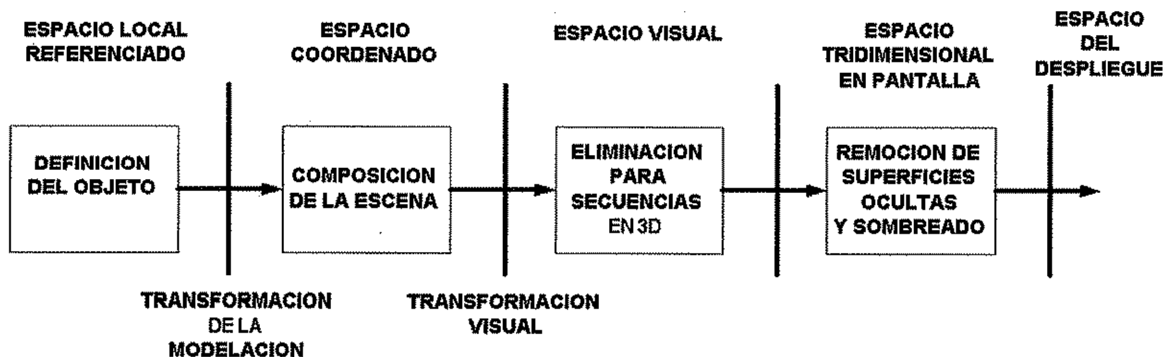


Fig. 2.12 Proceso de “rendering”.

Construir una escena, con objetos especificados dentro de un sistema coordenado de modelación (ver figura 2.13), puede incluir transformaciones tridimensionales aplicables a estos objetos. Un objeto esta empotrado en un espacio coordenado, que es común para todos los demás objetos de la escena, figura 2.13(B). En este espacio se establece la posición de la fuente de luz, o fuentes, que iluminan la escena, además, se tiene una referencia que determina el origen del sistema coordenado. Al variar las especificaciones relacionadas a la visualización, el objeto es transformado dentro de este espacio. La necesidad de establecer, una dirección en la cual este el observador, conduce a determinar que clase de proyección se requiere.

En el campo de visión del observador, una operación llamada “eliminación” esta en funcionamiento. Esta remueve totalmente los polígonos que no tienen la posibilidad de ser visibles desde el punto de observación. La diferencia entre la eliminación y la renovación de la superficie oculta puede ser mostrada al comparar las figuras 2.13(D) y 2.13(E). La cara frontal del polígono del objeto esta parcialmente oculto debido al pequeño cubo. La eliminación remueve las tres caras del cada cubo, pero todos las caras frontales de un sólido son visibles para el observador. Las dos operaciones finales que son mostradas a la salida en los polígonos que están sombreados y referenciados. En el sombreado se compara la orientación de cada polígono con la dirección de la fuente de luz y se asigna una sombrea en el interior del polígono. Para la referencia esta reparte a cada pixel a través de la proyección, (ver figura 2.13(F)), este es proceso no trivial, ya que involucra la conversión a partir de la especificación geométrica del polígono a su aproximación en el espacio [Watt, 1993].

Un simple interpretador asigna el sombreado a un pixel de una cara “a”, como se muestra en la figura 2.13(G). Una estrategia más elaborada puede discernir que el pixel mostrado esta sobrepuesto por dos caras y reparte el sombreado intermedio entre las caras “a” y “b”, cada consideración esta en la competitividad de los algoritmos anti-alias.

Finalmente, un proceso mostrado en la figura 2.13(E) es el de recorte del volumen tridimensional visible. Un sistema de graficación por computadora difiere de una cámara de video en un aspecto importante, un volumen tridimensional visible puede ser definido. Una región del volumen en el espacio puede ser colocada y la información del objeto fuera de este espacio puede ser removida [Watt, 1993]. Así, por ejemplo, se pueden renovar todos los objetos o partes de ellos que tienen cierta distancia a la cámara. La operación de recorte tridimensional implicada es tangible a la salida en el espacio visible para el observador.

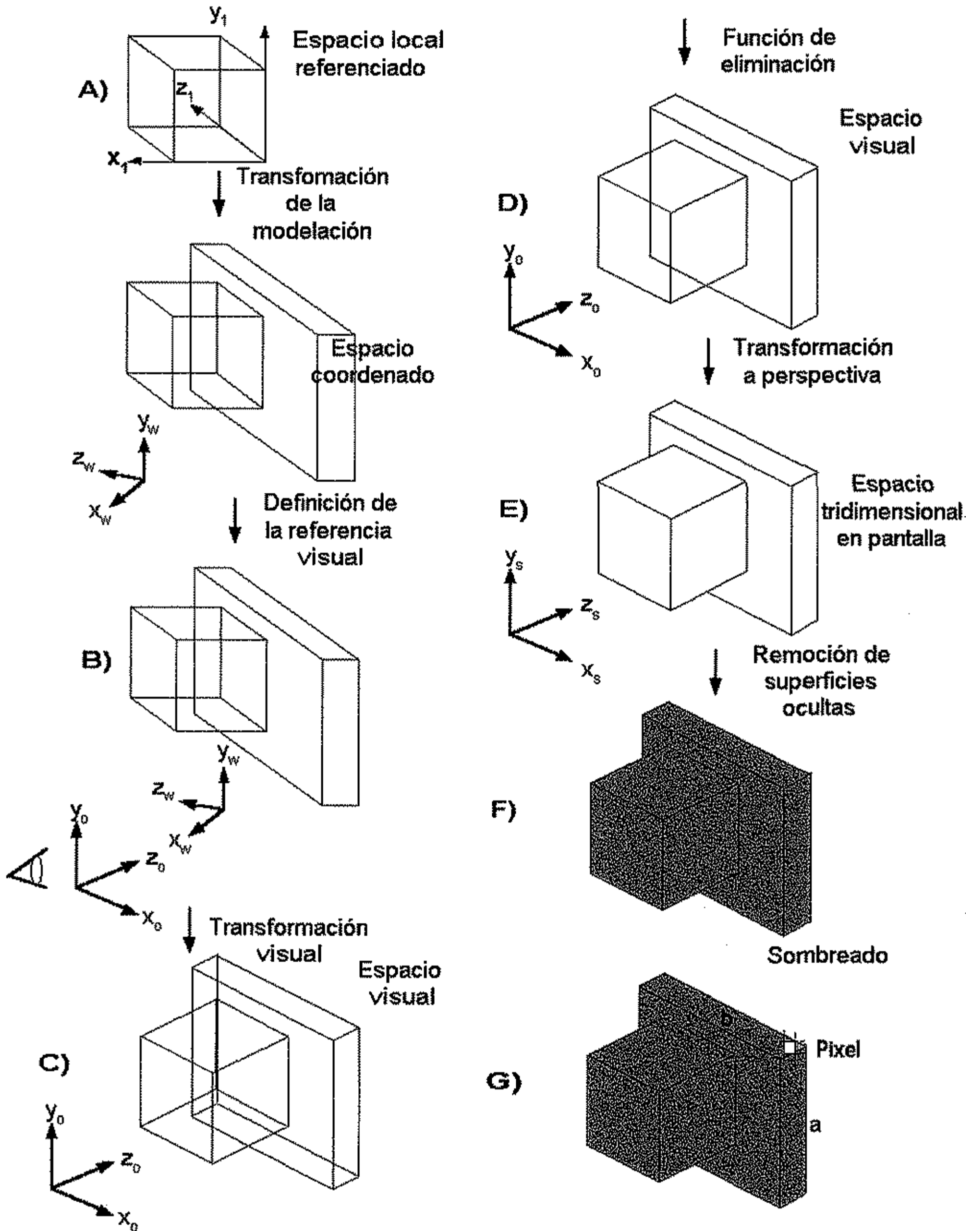


Fig. 2.13 Ejemplo del proceso "Pipeline" de una interpretación tridimensional.

2.4.6.2 Interpretación de una malla de parches con parámetros

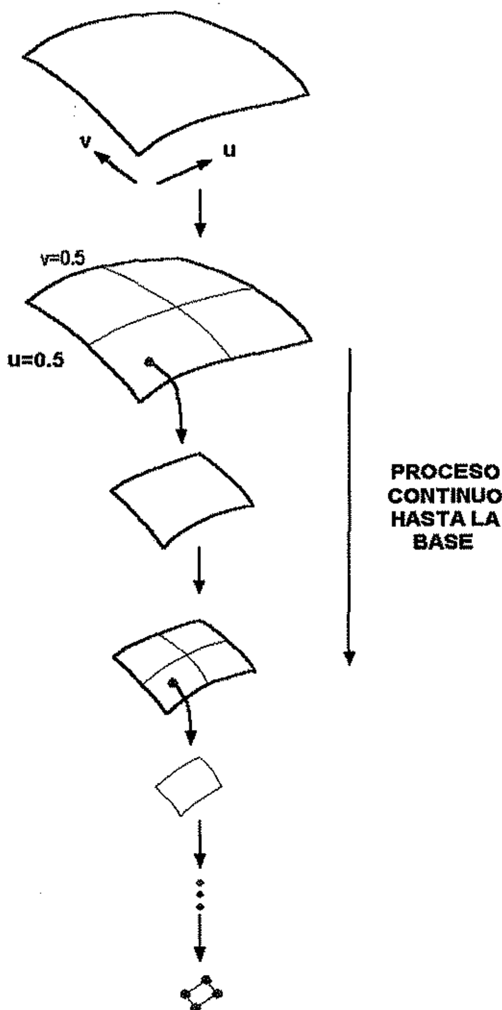


Fig. 2.14 Representación del proceso de división de parches.

La mayor aproximación de la interpretación de parches con parámetros bicúbicos es el preproceso de la representación y conversión de los parches en polígonos planos (ver figura 2.14). Este resulta en una baja complejidad del código. Sin embargo, se tiene un cuestionamiento debido a que en la modelación un objeto mediante el empleo de la más alta exactitud de la representación por parches, el recurso de aproximación esta en base a los polígonos. La respuesta está en que la representación original por parches puede ser necesaria porque en un ambiente interactivo es más simple, en cuanto a la exactitud requerida, la representación por parches es convertida a polígonos mediante una subdivisión o proceso de fraccionamiento y se tiene el completo control sobre el perímetro de la subdivisión y el tamaño final de los polígonos. Incluso se puede establecer una dependencia del tamaño del polígono con la curvatura local del parche [Watt, 1993]. En estas actividades son una unificación de las mostradas en la interpretación con malla de polígonos, actualmente se encuentra disponibles tanto el software como el hardware con programas grabados este tipo de interpretación.

2.4.6.3 Interpretación con una descripción en CSG

Las estrategias de interpretación de modelos CSG son un desatino. Porque la característica distintiva de la representación con CSG esta es que no es una representación de fronteras. En las otras dos formas, la superficie representada es una frontera del objeto donde se divide en dos regiones el espacio tridimensional del propio objeto, en el interno y el externo. En la representación CSG, la base de datos del objeto es una estructura de árbol que relaciona objetos de un conjunto de objetos primitivos, mediante operaciones booleanas [Foley, 1996]. Destacando que esta representación se rige por la potencialidad de las facilidades de la interactividad, y el costo de una interpretación resulta en una estrategia compleja y extensiva.

El principal problema implicado en la representación de los objetos mediante CSG es que deriva en la representación de fronteras a partir de una base de datos de CSG. Tres técnicas están involucradas:

1) CSG con "ray casting"[9]
2) Conversión a un a representación de voxels seguida por la interpretación volumétrica.
3) Empleando una versión del algoritmo "Z-buffer".

La evaluación de un objeto a partir de una descripción puede ser realizada por la reducción del problema a una dimensión, al enviar el haz de un rayo a partir de cada pixel al plano de visualización. En el caso más simple (proyección paralela) se explora el espacio del objeto con un conjunto de rayos paralelos.

Se considera un haz simple, y cada instancia primitiva es comparada en relación con la intersección de este haz. Algunas intersecciones son ordenadas con una profundidad Z, además, se tiene una clasificación de haz/primitivo para cada haz (ver figura 2.15).

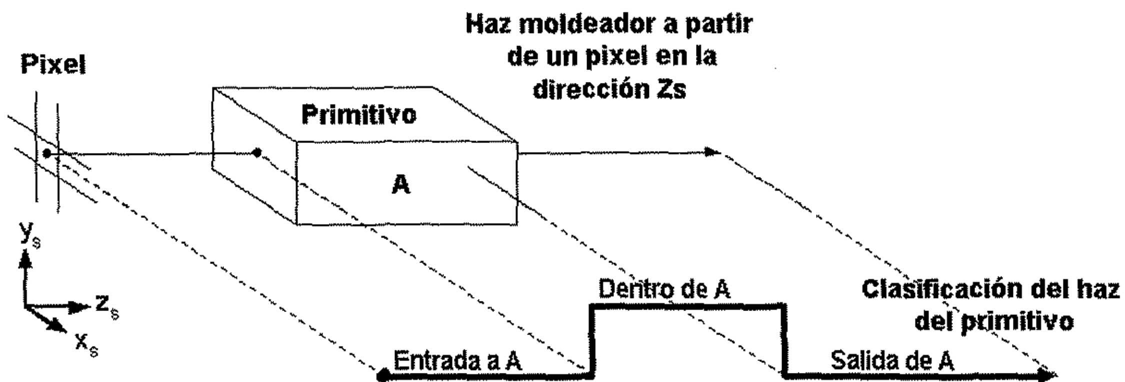


Fig. 2.15 Derivación de una clasificación haz/primitivo.

A partir de ahora se tienen algunas combinaciones booleanas entre las primeras primitivas encontradas a lo largo del haz. En la figura 2.16 se muestra que la evaluación de las operaciones booleanas con las primitivas a lo largo del haz, es simple. Un valor de sombreado puede ser asignado al pixel con modelo simple de reflexión aplicado desde la primera intersección a lo largo del haz. Los puntos de intersección varían de acuerdo a las operaciones booleanas entre las primitivas [Watt, 1993].

[9] "Ray casting" : Método de modelación basado en un haz de modelación.

2.5 Discusión.

En este capítulo, se hace una presentación general del área de visualización de datos tridimensionales, describiendo los conceptos y principios en los que se basan las diversas técnicas empleadas.

Se destacan los estrategias de interpretación ("rendering") y en particular "volume rendering". La cual permite crear una imagen tridimensional a partir de un conjunto de imágenes bidimensionales. Además, de permitir un manejo e interpretación eficiente del volumen de datos asociados al conjunto de imágenes.

Capítulo 3

Estudio del Algoritmo de Visualización "Volume Rendering"

3.1 Introducción

En capítulo 2 se presentaron diversos métodos de visualización, uno de los cuales considera la interpretación volumétrica, proporcionando una imagen tridimensional a partir de un conjunto de imágenes bidimensionales.

En este capítulo se estudia "volume rendering" como una técnica de visualización, y a la vez las características de las transformaciones directamente posibles desde la escena del espacio al plano visual. Existen dos aspectos básicos ha considerar durante el proceso de la interpretación volumétrica: el preprocesamiento del volumen de datos y la interpretación misma del volumen de información [Udopa, 1986]. Esta es sólo una división conceptual para destacar dos aspectos importantes de este método.

3.2 Preprocesamiento del Volumen de Datos

La interpretación volumétrica puede ser descrita como un proceso en el cual se tiene una proyección de una imagen. En este proceso, se tiene como base un conjunto de imágenes, que representa rebanadas ("slices"[10]) del volumen a analizar (escena). Cada imagen esta compuesta por objetos, los cuales presentan propiedades que pueden ser de interés en el estudio de dicho volumen. Los valores asociados a las propiedades con relación al objeto, como son conocidos, es posible crear la proyección dentro de una variedad de caminos sofisticados, así como para poder dar énfasis en los aspectos seleccionados de la escena.

En el modelo conceptual, empleado en la interpretación volumétrica, la escena está considerada como la representación de un volumen con un color y una opacidad asociados, que son diferentes en otras regiones de la escena. La meta de la interpretación es calcular imágenes que presenten la apariencia de este volumen, desde varios ángulos simultáneamente a la transmisión de luz a través del volumen, también como la reflexión en las interfaces de los objetos a través de la cuidadosa selección de color y opacidad. Después es necesario hacer la especificación de cuáles aspectos podrían ser enfatizados (grado de opacidad) y de cuáles podrían no ser enfatizados (alto grado de opacidad). Así como su color, que es imperativo en la identificación de aspectos de alguna estructura, de voxel en voxel a través del uso de un método de procesamiento de la escena, incluyendo las operaciones geométricas de discriminación, de filtrado, de interpolación y de segmentación [Udopa, 1991].

---

[10] "slices": secciones transversales de un objeto.

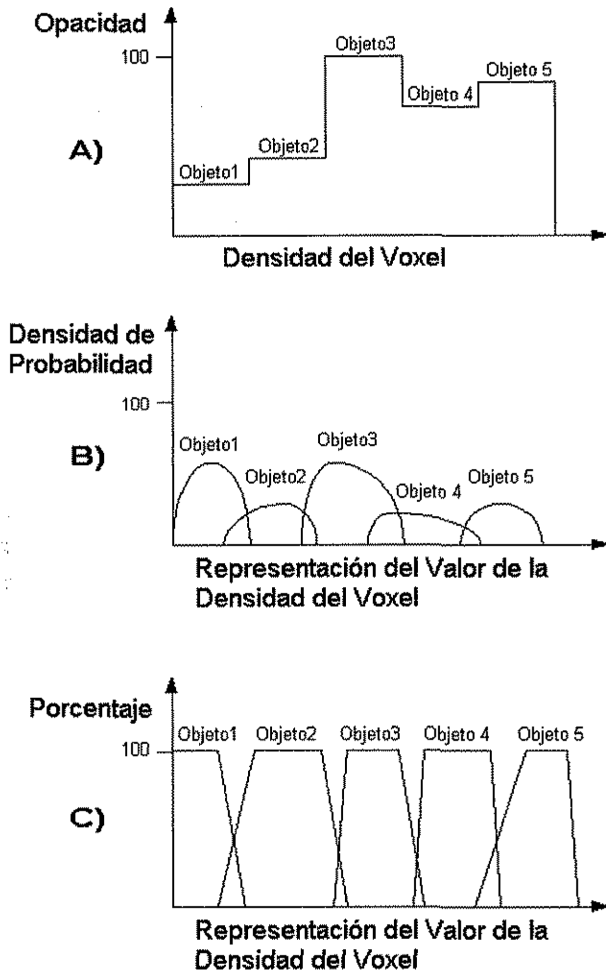
Las técnicas para la realización de estas operaciones descritas en el área de "procesamiento de escenas", son aplicadas y han sido usadas en la interpretación volumétrica. La segmentación también llamada clasificación, es la más frecuentemente aplicada. El propósito de la segmentación es asignar a cada voxel que forma parte de la escena una opacidad y un color. Si se asigna, el mismo color a todos los voxels que pertenecen a un objeto de una escena con una opacidad del 100%, y una opacidad del 0% a todo el resto de los voxels de la escena, entonces el resultado es esencialmente una escena binaria. En la mejor situación generalmente se pueden considerar múltiples objetos, con un color diferente para cada objeto y una opacidad entre 0 y 100 % (basándose en la textura del material de cada objeto contenido en la escena).

Un método simple, es la asignación del mismo color a todos los voxels, pero variando la opacidad basada en la densidad de voxel. Con una opacidad fijada entre 0 - 100 % es asignada a cada rango de densidad de voxel, con la premisa de que en cada voxel se pueden tener diferentes tipos de objetos, mismos que se identifican basándose en el intervalo en el cual esta contenida su densidad (ver fig. 3.1 (A)). En la figura 3.1(B) se muestra otro método más sofisticado para la asignación de opacidad y color a cada voxel, basado en un estimado de la mezcla de los objetos en el voxel [Udopa, 1991].

Alrededor de cada objeto se tienen otros objetos, como se muestra en la figura 3.1 (C), entonces se puede suponer para cada voxel una estructura mixta (en porcentajes P_3 y P_4), donde es conocida tanto la densidad del voxel como su distribución para los diferentes objetos. De hecho, esta densidad se puede calcular a partir de la gráfica porcentaje versus densidad mostrada en la figura 3.1(C).

Ahora se supone que se asume una cierta opacidad fija y un color para cada tipo de objeto, basado en el tipo de despliegue que se desea crear (por ejemplo, para un hueso que es opaco, si está dentro de una estructura semitransparente del 25% de opacidad correspondería a la piel). Conociendo los porcentajes de mezcla del objeto para cada voxel y el color efectivo, la opacidad del voxel puede ser calculada. Si se supone que un voxel contiene dos tipos de objetos con opacidades α_3 y α_4 y colores C_3 (R_3, G_3, B_3) y C_4=(R_4, G_4, B_4), el porcentaje de estos objetos en el voxel tiene los valores P_1 y P_2, respectivamente, entonces el color efectivo asociado con el voxel es: $[(P_3\alpha_3R_3 + P_4\alpha_4R_4), (P_3\alpha_3G_3 + P_4\alpha_4G_4), (P_3\alpha_3B_3 + P_4\alpha_4B_4)]$ y su opacidad efectiva es $(P_3\alpha_3 + P_4\alpha_4)$ [Udopa, 1991]. Si existe alguna superficie que pasa a través del voxel, entonces es posible mantener la opacidad y el color asociado a cada objeto contenido en el voxel, con el propósito de determinar la contribución de este voxel en la interpretación final.

Una alternativa en el método de segmentación, basada en la idea de la interpretación volumétrica, es que solamente para la regiones donde los objetos contenidos en una estructura de un volumen, se supone el empleo de una magnitud de un gradiente evaluado para cada voxel que forma parte de la escena. En la estructura donde se tiene una fuerte evidencia de una región mixta, la asignación de una opacidad será proporcional a la magnitud del gradiente. Entonces, la región mixta con un alto contraste podría tener una alta opacidad, así como, para un bajo contraste podría tenerse una baja opacidad.



Se pueden asignar opacidades diferentes, para las diversas estructuras representadas en las múltiples superficies, en donde cada región de la estructura se tiene especificado un valor típico de la propiedad del objeto, como se ilustra en la figura 3.2. El color de un voxel esta determinado por sus *características de difusión y reflexión* especular, y para cada componente de color (R,G,B) de la luz incidente. A través de este método no se determinan cuanto de estas características podrían ser escogidas, los valores típicamente usados para la determinación de las regiones mixtas de los objetos, pueden también ser usados para especificar *cuales características son consideradas* en cada región mixta.

Fig. 3.1 A) Representación de opacidad basado en el intervalo de la densidad del voxel, B) Representación gráfica del valor de la distribución del valor de la propiedad de cada tipo de objeto, C) Representación gráfica basada en B), indicando la mezcla de los objetos dado por el valor de la propiedad.

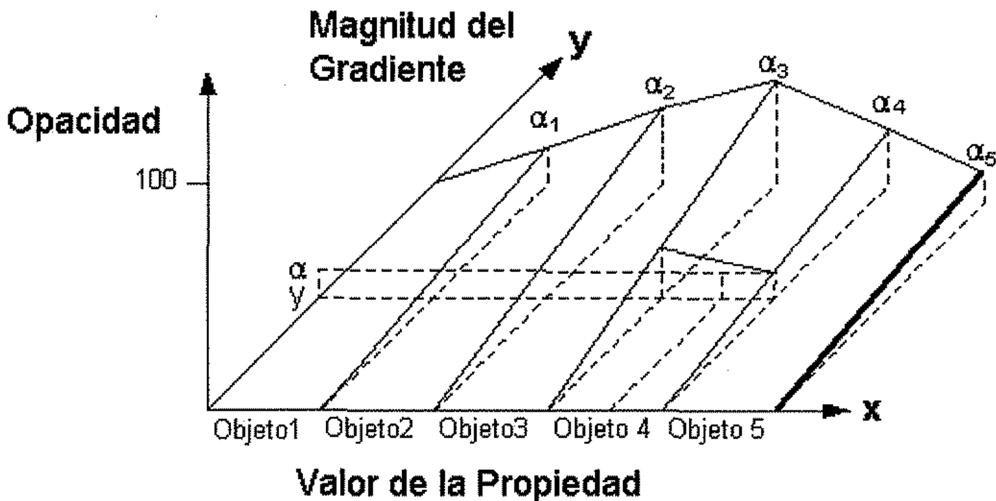


Fig. 3.2 Representación gráfica basada en valores típicos de objeto y asumiendo opacidades máximas de varios tipos de objetos con la magnitud del máximo gradiente.

El enmascaramiento, es otro preprocesamiento que frecuentemente se hace necesario en las interpretaciones volumétricas. Los voxels constituyen estructuras las cuales en el despliegue pueden no ser incluidas, debido a las altas opacidades asignadas (y/o color inapropiado). Al considerar un proceso automatizado de segmentación se tiene, como consecuencia, la presencia de estas estructuras en la interpretación. También, tales voxels pueden ser suprimidos por la asignación de un valor bajo de opacidad (cero), y para ello se usa la operación de enmascaramiento. Este preprocesamiento es requerido si se desea tratar en forma distinta del resto de la misma estructura, por ejemplo, se puede desear desplegar la superficie de la piel en una mitad de la cara con alta opacidad y la otra mitad con baja opacidad, de tal manera que se visualizan las estructuras óseas debajo de capas de piel.

La interpretación volumétrica, no es la excepción al tedioso requerimiento de la caracterización de una subregión de una escena (frecuentemente: "slice" por "slice"), en la que la segmentación podría ser restringida. Podemos enfatizar que estas restricciones son aplicadas igualmente a las técnicas de interpretación de superficie, y no hay nada inherente de una u otra técnica que garanticen una disminución significativa en las limitaciones en estas técnicas de segmentación.

3.2.1 El método de Segmentación Otsu

Existen tres tipos métodos para realizar el proceso de segmentación a una imagen. Los basados en pixeles, en cuales se considera sólo el valor de gris de un pixel, para decidir si el mismo pertenece o no al objeto de interés. El objetivo de este método, es el de encontrar de una manera óptima los valores característicos de la imagen que establecen la separación del objeto de interés, con respecto a las regiones que no pertenecen al mismo.

Otro tipo es el método basado en contornos que puede ser usado para evitar la variación del tamaño del objeto. Este método se basa en realizar la búsqueda del valor máximo del gradiente, sobre cada línea que forma la imagen. Cuando un máximo es encontrado, un algoritmo de trazado trata de seguir el máximo del gradiente alrededor del objeto, hasta encontrar de nuevo el punto inicial, para luego buscar el próximo máximo en el gradiente [Bravo, 1998].

Por último los basados en regiones, toman en cuenta un conjunto de puntos de la imagen, a los cuales se les analiza características como, la posición en el espacio de intensidades, las relaciones topológicas (conectividad) y las característica de las fronteras entre dos conjuntos. Dependiendo de como sea analizada la posición en el espacio y las relaciones espaciales existentes entre los pixeles, se pueden encontrar métodos de Clasificación y métodos por Crecimiento de Regiones [Bravo, 1998]..

Entre los métodos basados en pixeles se encuentra el método segmentación Otsu [Otsu, 1979], el cual presenta características que permiten realizar una adecuada segmentación de manera sencilla y simple, además de poder determinar N clases contenidas en un imagen. A continuación se describe el método:

Sea una imagen con L tonos de gris [1,2,....,L]. Se denota al número de pixeles con un tono de gris i como n_i, y el total de pixeles como $N=n_1+n_2+....+n_L$.

Normalizando el histograma para tratarlo como una distribución de probabilidad, se tiene:

$$P_i = \frac{n_i}{N} \tag{3.1}$$

Con $P_i \geq 0$

$$\sum_{i=1}^{L} P_i = 1 \tag{3.2}$$

Se define K, como un umbral que separa la imagen en dos clases: C_0 que corresponde a los pixeles con valores dentro del rango [1,2,...,K] y C_1 que corresponde a los pixeles con valores dentro del rango[K+1, K+2,...,L]. Para nuestros fines se debe determinar el valor de K.

La probabilidad de ocurrencia de cada clase se obtiene como

$$W_0 = P_R(C_0) = \sum_{i=1}^{K} P_i = w(K) \tag{3.3}$$

$$W_1 = P_R(C_1) = \sum_{i=K+1}^{L} P_i = 1 - w(K) \tag{3.4}$$

El valor promedio de cada clase es

$$\mu_0 = \sum_{i=1}^{K} i * P_R(i \mid C_0) = \sum_{i=1}^{K} i * \frac{P_i}{W_0} = \frac{\mu(K)}{w(K)} \tag{3.5}$$

considerando

$$\mu(K) = \sum_{i=1}^{K} i * P_i \tag{3.6}$$

entonces

$$\mu_1 = \sum_{i=K+1}^{L} i * P_R(i \mid C_1) = \sum_{i=K+1}^{L} i * \frac{P_i}{W_1} = \frac{\mu_T - \mu(K)}{1 - w(K)} \tag{3.7}$$

donde

$$\mu_T = \mu(L) = \sum_{i=1}^{L} i * P_i \tag{3.8}$$

Se verifica que $W_0\mu_0 + W_1\mu_1 = \mu_{T_i}$ y $W_0 + W_1 = 1$

Para maximizar la varianza entre las clases, por medio de análisis discriminante [Otsu, 1979], se tiene:

$$\sigma_B^2 = W_0(\mu_0 - \mu_T)^2 + W_1(\mu_1 - \mu_T)^2 \qquad 3.9$$

$$\sigma_B^2 = W_0 W_1 (\mu_1 - \mu_0)^2 \qquad 3.10$$

Expresando ésta última ecuación en términos de K se tiene:

$$\sigma_B^2(K) = \frac{[\mu_T w(K) - \mu(K)]^2}{w(K)[1 - w(K)]} \qquad 3.11$$

El valor de K que maximice la anterior relación, determina el umbral que identifique a las dos clases.

A partir de estos nuevos intervalos se puede aplicar nuevamente el método logrando con ello obtener dos clases más. De esta forma se pueden determinar los umbrales para clasificar N clases contenidas en la imagen.

3.3 Interpretación del Volumen de Datos

La meta computacional en la interpretación volumétrica es determinar el color que es asignado a cada pixel en el espacio visual (la pantalla), para alguna orientación dada de la escena en la imagen espacial.

Un paso computacional fundamental en esta meta es determinar el conjunto de voxels que contribuye a cada píxel en la pantalla. Dos técnicas comúnmente usadas para este propósito son: proyección de voxel y el "ray casting" (haz moldeador). Otras son el "splatting"[11], "shear-warp"[12] y "3D Texture Mapping"[13] [Meiβner, 2000].

En el caso de la proyección de voxels, éstos son proyectados en la pantalla con un cierto orden de acuerdo a su distancia en la pantalla (del más lejano al más cercano o viceversa). Existen muchos caminos para la proyección de voxels uno a uno, asegurando que los voxels sean proyectados dentro de algún pixel dado en la pantalla, en uno de estos ordenes [Udopa, 1986]. Por ejemplo, en una escena rotada como la mostrada en la figura 3.3(A), puede estar almacenada como un arreglo tridimensional en un orden de renglón por renglón y "slice" por "slice".

[11] "splatting": Es un tipo de interpretación en donde los voxeles del conjunto son obtenidos del frente a atrás, formando una rebanda de voxeles, su contribución a la imagen final es mediante la utilización de un flitro.
[12] "shear-warp": Es un tipo de interprestación en donde se emplea una factorización entre el volume y las estructuras de datos de las imágenes intermedias.
[13] "3D Texture Mapping": es un método tener objetos geométricos simples en tres dimensiones mostrándose más complejo y realista, en base a la generación de mapas de textura.

Si se proyectan los voxels, iniciando desde el primer "slice" y finalizando con el último "slice", entonces los voxels proyectados dentro de un pixel dado son considerados en el orden del más lejano al más cercano. La proyección de voxels en el orden inverso, corresponde al orden del más cercano al más lejano. En algunas situaciones, se puede tener el orden inverso en los renglones (por ejemplo del último renglón al primero) y a la par, el orden de los voxels dentro del renglón (por ejemplo del último voxel al primer voxel), ver figura 3.3(B), para garantizar una proyección ordenada.

Dada una orientación de una escena en una imagen espacial, es posible determinar el orden de las "slices", de los renglones dentro de un "slice", y de los voxels dentro de un renglón en el cual los voxels podrían ser proyectados, así como garantizar que el criterio del orden seleccionado de su distancia desde la pantalla este satisfecho.

En el "ray casting" los voxels que influyen el color de un pixel son identificados por la traza de un rayo desde el pixel dentro de la escena (ver fig. 3.3 (A)) y notando que voxels pasan a través de la vecindad del rayo en la escena. La vecindad puede ser definida en una serie de caminos diferentes. El método más simple es considerar solo por los voxels que son interceptados por el rayo.

Un método más sofisticado, es tomar un número de puntos igualmente espaciados entre el punto de entrada y el punto de salida del haz a través de la región de la escena, y estimar la opacidad y el color en cada uno de estos puntos por interpolación.

En las siguientes líneas se expondrá como el color del pixel es determinado. Para ello es conveniente usar paradigmas, cómo el "ray casting" o la proyección del voxel dependiendo de los resultados en una simple descripción. Por lo menos en una de las instancias de estas estrategias pueden ser intercambiadas entre ellas, y se obtienen diferencias con un refinado cálculo en una imagen con calidad subjetiva .

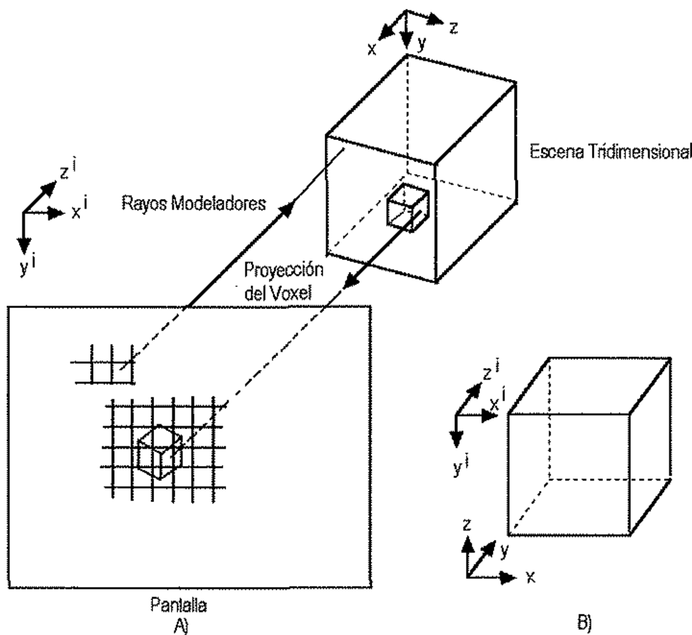


Fig. 3.3 A) Ilustración de la de la proyección del voxel y el rayo director y B) Orientación de la escena

3.3.1 Teoría "Volume Rendering" empleando "Ray Casting"

El volumen interpretado que emplea "ray casting" esta basado en el modelo Blinn/Kajiya [Watt, 1992]. En el cual se tiene un volumen que exhibe una densidad D(x,y,z), penetrado por un rayo R, en cada punto a lo largo del rayo se tiene una Iluminación I(x,y,z), alcanzando en el punto p(x,y,z) de la fuente o fuentes de luz. El valor de la densidad dispersada a lo largo del rayo al punto visual, depende de una función de reflexión o función de fase P y la densidad local D(x,y,z). La dependencia de la densidad expresa el hecho de que pocas partículas dispersadas brillan menos a la luz al punto visual, en la dirección de un número de partículas diminutas. La función de densidad esta parametrizada a lo largo del rayo como:

$$D(x(t), y(t), z(t)) = D(t) \qquad 3.12$$

La función de una fuente es:

$$I(x(t), y(t), z(t)) = I(t) \qquad 3.13$$

la iluminación dispersada a lo largo de R a un punto de distancia t a lo largo del rayo es:

$$I(t)D(t)P(\cos\theta) \qquad 3.14$$

donde: θ es el ángulo entre R y L (el vector de iluminación desde el punto de interés).

Si existe más de una fuente de luz entonces se tiene:

$$\sum_n I_n(t)D(t)P(Cos\theta_n) \qquad 3.15$$

La determinación de I(t) no es trivial, ya que significa calcular la atenuación de la fuente de luz en su viaje a través del volumen al punto de interés. Esto es idéntico al cálculo de la cantidad de la dispersión de luz en el punto p(x, y, z) que esta afectado a lo largo de R al punto visual, en el algoritmo éste es ignorado, e I(x, y, z) es considerado constante a través del volumen. Esta consideración puede ser una desventaja, pero se enfatiza una mejor aplicación práctica en cuanto al interés en una cierta visualización, incluyendo la línea de la integral del punto p(x, y, z) a la fuente de luz. El cálculo de la atenuación debida a la función de densidad a lo largo del rayo es [Pawasauskas, 1997]:

$$atenuación = \exp\left(-\mathrm{T}\int_{t_1}^{t_2} D(s)ds \right) \qquad 3.16$$

siendo T es una constante que convierte densidad en atenuación.

La intensidad de la luz que llega al punto visual a lo largo de la dirección de R, debido a todos los elementos a lo largo del rayo esta dada por:

$$B = \int_{t_1}^{t_2} \left(\exp\left[-\mathrm{T}\int_{t_1}^{t_2} D(s)ds \right] \right) \left[I(t)D(t)P(Cos\theta) \right] dt \qquad 3.17$$

3.3.2 Definición del Algoritmo Típico del "Volume Rendering"

Los algoritmos que están basados en esta teoría, generalmente involucran una simplificación de la Ecuación 3.17. La técnica que Drebin refiere como una "reproyección aditiva" ha sido explorada por varios autores [Watt, 1992]. El método en efecto colapsa o proyecta voxels a lo largo de una cierta dirección visual. Intensifica a los voxels a lo largo de los rayos visuales paralelos, proyectando una intensidad para un pixel en el plano visual. Un espectador puede elegir los voxels de cierta profundidad particular para dar una máxima opacidad, así se tiene que la profundidad en el campo del volumen es visualizado, y el número de planos de los sobreplanos en el plano visual, pueden ser controlados. Los grados de libertad son importantes en la computación gráfica, los datos proporcionados permiten que el volumen sea visualizado desde cualquier dirección y la superficie oculta pueda ser implementada. Finalmente, el color puede ser usado para mejorar la interpretación. Básicamente, las opciones concernientes están disponibles en el proceso, en el cual la información es integrada a lo largo del rayo emitido.

En suma, todos los métodos de reproyección hacen uso de un modelo de iluminación del voxel, que involucra una simple combinación de la luz reflejada y transmitida desde el voxel. Todos los enfoques son un subconjunto del modelo mostrado en la figura 3.3, en ésta se muestra un simple voxel iluminado por las fuentes de luz. Una fuente de luz es una fuente direccional para la cual el voxel reacciona en función de sus propiedades de reflexión, la otra es la luz entrante del voxel vecino a lo largo de la trayectoria del rayo. La luz saliente puede tener las siguientes contribuciones:

1. La luz puede ser reflejada a lo largo de la trayectoria del rayo, debido a la reflexión desde la fuente de luz direccional, al interactuar con un fragmento de la superficie contenida en el voxel.
2. La luz entrante puede ser atenuada por la existencia de una superficie opaca.
3. Si el voxel es homogéneo entonces actuaría como un objeto parcialmente opaco y filtrando la luz incidente.
4. El voxel puede ser semi-luminoso.
5. Una combinación de los anteriores factores.

La luz incidente puede resultar de la iluminación de la fuente o de la luz que está siendo transmitida o reflejada a lo largo de la dirección de la trayectoria del rayo. Una implementación típica de este enfoque está descrito por Levoy [Watt, 1992], llamada técnica de "visualización volumétrica directa" y las etapas del proceso están enfocadas para hacer el mejoramiento de los datos originales dentro de la interpretación. Levoy describe la técnica como la consistencia de dos "pipeline"; uno de visualización y otro de clasificación [Watt,1992]. La salida de estos dos "pipelines" son combinados mediante la composición volumétrica para producir la imagen final.

En el "pipeline" de visualización, la adquisición de datos en la forma de voxels es matizada. Cada voxel, en los datos, es localizado y tonalizado empleando una aproximación del gradiente local para obtener un voxel normalizado. Es sustituido dentro del modelo estándar de reflexión Phong, para obtener una intensidad a la salida de este "pipeline",

como un color con tres componentes de intensidad para cada voxel del conjunto de datos. El hecho de que cada voxel, en el volumen es interpretado, explica el uso del término "volume rendering" [Watt,1992].

En regiones homogéneas el gradiente es cero, el matiz o color de las regiones es constante. El método Levoy consiste en determinar básicamente la opacidad para cada voxel de acuerdo al gradiente. Este gradiente es multiplicado por la opacidad, en regiones homogéneas se tiene una opacidad cero y el color no hace efecto en la imagen final. El propósito del "pipeline" de clasificación está en asociar una opacidad a cada voxel, y puede asignar los parámetros de color, que son usados en el modelo de tonalización. La clasificación de opacidad es, dependiente del contexto y la aplicación de la técnica Levoy, por ejemplo en las imágenes CT, cada valor de voxel en el dato original es un coeficiente de absorción de rayos X. En este "pipeline", Levoy emplea una técnica que está destinada para retener "mechones" o regiones aisladas de un objeto con una densidad particular, que puede ser descartada por los métodos de clasificación. Por ejemplo el simple uso de umbrales, como:

$$X \in V_a \quad \text{Si } V(X) < T$$
$$X \in V_b \quad \text{Si } V(X) > T$$

donde V_a y V_b son volúmenes de diferentes tipos de objetos a y b. Entonces esto es posible para regiones estrechas de un tipo para ser representadas por voxels $V(X) > T$

Empleando las siguientes restricciones dentro de la naturaleza de los datos, Levoy presenta un esquema de clasificación, en donde los valores de V(X) entre dos tipos de valores n y n+1 están asignados a una opacidad entre los valores α_n y α_{n+1}. Las restricciones son tales que los datos originales contienen valores que caen dentro de una pequeña vecindad de algún valor conocido, tal que, el objeto de cada tipo toca, a lo más, a otros dos tipos, si los tipos son ordenados por valores, y cada tipo solamente toma los tipos adyacentes a éste, se tiene:

$$f_{V_n} \quad n=1,2,.....,N$$

tal que:

$$f_{V_m} < f_{V_{m+1}} \quad m=1,2,....,N-1$$

El objeto que no es de un valor $f_{V_{n1}}$ toma algún valor del correspondiente objeto.

$$f_{V_{n2}} \quad |n_1 - n_2| > 1$$

La opacidad se puede escribir como:

$$\alpha(X) = \begin{cases} \alpha_{v_{n+1}} \left\{ \dfrac{V(X) - f_{v_n}}{f_{v_{n+1}} - f_{v_n}} \right\} + \alpha_{v_n} \left\{ \dfrac{f_{v_{n+1}} - V(X)}{f_{v_{n+1}} - f_{v_n}} \right\} & si \{ f_{v_n} \le V(X) \le f_{v_{n+1}} \} \\ 0 & \{otros\} \end{cases} \quad 3.18$$

En esta clasificación, si la opacidad de los objetos considerados es enfatizada junto con realce los bordes de los mismos, al final la visualización es mejor. Esto se obtiene multiplicando del valor de la opacidad por el gradiente local:

$$\alpha'(X) = |\nabla V(x)|\, \alpha(X) \quad 3.19$$

Ahora se tienen dos valores asociados a cada voxel: C(X) un tono calculado a partir de un modelo de reflexión usando el gradiente local, y $\alpha(X)$ una opacidad de acorde al tipo de objeto.

El próximo paso, es la llamada "composición volumétrica" que consiste en una proyección bidimensional de estos valores en un plano visual. Los rayos son moldeadores a partir del punto visual al arreglo de voxels, tanto los valores de C(X) y $\alpha(X)$ son "combinados" o proyectados para producir un valor de intensidad final (valor del pixel). Para un cualquier voxel a lo largo del rayo, la fórmula estándar de transparencia es:

$$C_{out} = C_{in}(1-\alpha(X))+C(X)\alpha(X) \quad 3.20$$

donde:
C_{out} es la intensidad saliente de color/intensidad para el voxel X a lo largo del rayo.
C_{in} es la intensidad entrante para el voxel.

Esto puede ser considerado como un proceso de remuestreo con varias opciones disponibles. Se podría simplemente interpolar desde lo valores vértices del voxel que pasan a través del rayo, pero es más correcto considerar los voxels vecinos y la interpolación trilinear del campo de valores. La intensidad debida del conjunto de voxels que intercepta el rayo esta dado por:

$$C(R) = \sum_{k=0}^{k} \left\{ C(R,k)\alpha(R,k) \prod_{i=k+1}^{k} (1-\alpha(R,i)) \right\} \quad 3.21$$

donde:
(R,k) es el k-ésimo voxel a lo largo del rayo R.
$C(R,0) = C_{fondo}$
$\alpha(R,0) = 1$

3.3.2.1 Cálculo de la Densidad del voxel

Todos lo datos a lo largo del rayo, que están al frente de algún voxel con opacidad=1 no contribuyen a la visualización. La ecuación 3.21 de intensidad podría ser evaluada en las tres bandas de colores para una imagen de color estándar. En esta etapa esto es útil para comparar la ecuación 3.21 con la integral general del rayo moldeador de intensidades de volumen (Ecuación 3.17). Primero la atenuación tiende ha ser reemplazada por la opacidad, donde Opacidad = 1– atenuación.

El valor a lo largo del rayo es multiplicado y acumulado, si se consideran los factores remanentes en la ecuación 3.10. Un número de extrapolaciones físicas son implementadas: primero, la fuente de luz se asume como uniforme desde cada voxel, y cada voxel ve una fuente de luz externa a través de un objeto perfectamente transparente. En

otras palabras, se considera la atenuación cuando la luz está viajando a través del volumen desde el voxel al punto donde se encuentra el observador, pero no cuando está saliendo de la fuente de luz al voxel. El producto de la densidad de volumen y la función de fase en la ecuación 3.17, es reemplazado por el gradiente calculado de los valores para un simple voxel. La densidad de volumen de peso es reemplazada, y la función isotrópica de fase es considerada en la superficie tonalizada mediante un modelo de reflexión (es este caso el modelo Phong).

Un enfoque similar, adoptado por Drebin, donde los voxels son clasificados con una etiqueta específica, usando un clasificador probabilístico [Watt, 1992]. Las distribuciones en cada tipo son conocidas a priori y, además con el hecho de que no más de dos tipos de distribuciones de objetos sobre la capa, por ejemplo, considerando una composición anatómica en donde el aire puede distribuirse sobre una capa de grasa pero no con una que represente el hueso, y a cada voxel le es asignado a un porcentaje del material. Esto se refleja en el hecho de que un voxel pueda resaltarse en los bordes de la regiones que contienen diferentes materiales. El histograma de las intensidades de voxels, que son los datos originales, forma parte de esta decisión. La diferencia entre este enfoque y el previamente explicado, es que más de una etiqueta puede ser asociada con cada voxel. Las etiquetas color y opacidad asociadas con un voxel son:

$$C = \sum_{i=1}^{n} P_i C_i \qquad 3.22$$

donde:
n es el número de materiales en el voxel.
P_i es el porcentaje del material en el voxel.
C_i es el color del material multiplicado por su opacidad que es $C_i=(\alpha_i R_i, \alpha_i G_i, \alpha_i B_i, \alpha_i)$, α_i se asume independiente del ancho de onda.

La superficie normal calculada está basada en el gradiente del campo de densidad D del voxel:

$$D = \sum_{i=1}^{n} P_i \mu_i \qquad 3.23$$

donde:
μ_i es la densidad asignada al material *i*.

3.3.2.2 Proceso de composición

Drebin define el proceso de composición en términos de un operador de composición, llamado ***over*** [Watt, 1992]. La ecuación de transparencia estándar para un simple voxel es:

$$C_{out}=C_{in}(1-\alpha_z) + C_z\alpha_z$$

utilizando el operador:

$$C_{out}= C \textbf{\textit{ over }} C_{in}$$

donde $C = C_z\alpha_z$

Drebin también generaliza el proceso para distinguir las contribuciones a la cantidad de la luz saliente de un voxel a lo largo del rayo visual. En este método, un voxel, es considerado como la contribución de tres parámetros de "regiones de color", el parámetro asociado con la región del volumen al frente de una superficie considerada, nombrándole C_f, C_s está asociado con la superficie y el C_b ligado a la región entre la superficie. Por lo que C_{out} queda como:

$$C_{out}=(C_f \, \textbf{\textit{over}}(C_s \, \textbf{\textit{over}}(C_b \, \textbf{\textit{over}} C_{in}))) \qquad 3.24$$

Considerando una superficie del volumen, $S=|N|$, donde $|N|$ es la magnitud del gradiente (está definido y usado en la contribución de C_S). Se tiene de este modo, si S es bajo, entonces el voxel está considerado como un gel homogéneo semi-opaco que no refleja la luz de alguna fuente de luz, pero solamente modula la luz entrante. No modula, la luz reflejada directamente emanada desde el interior de un voxel homogéneo. En contraste la técnica Levoy, simplemente usa: $C=C_S$. Los atributos de tal técnica, se asume la visualización de un campo escalar tridimensional, usando la mejor técnica de realce del pseudocolor, con las siguientes consideraciones :

1. La interacción con una fuente de luz hace posible que la percepción del tono de la isosuperficies sea realzada.
2. La transparencia es incorporada para que las regiones dentro de las estructuras puedan ser visualizadas.
3. La profundidad para la cual el volumen esta penetrado es infinitamente ajustable en la asignación apropiada al campo de valores de opacidad. Aunque en ocasiones se tenga la existencia de una isosuperficie.
4. La posibilidad de ver el conjunto de datos desde un punto visual arbitrario.

Otro autor, Sabella, generaliza la reflexión o el algoritmo de la tonalización, en el cual, las implementaciones son un simple modelo de dispersión. La luz es directamente reflejada desde la fuente de luz al punto visual vía la superficie visualizada. En este método, el campo total del volumen es considerado, un emisor de densidad variable y la luz reflejada en la dirección visual. Es calculado usando un modelo de dispersión, en conjunto con el modelo de Kajiya [Watt, 1992].

3.3.2.3 Interpretación en volúmenes binarios

Si se asigna un valor simple de opacidad (100%) y de color a todos los voxels contenidos en los objetos a ser interpretados, se pueden descartar ambas entidades y esencialmente tener una escena de tipo binario. Una serie de elegantes técnicas están disponibles para interpretar tales volúmenes binarios. El más directo de éstos emplea una representación en un arreglo rectangular (por ejemplo, en el orden renglón por renglón y "slice" por "slice" de los voxels de datos almacenados) para el volumen binario.

Si se proyecta solamente el primer voxel del volumen en la pantalla, de acuerdo al orden del más lejano al más cercano, se identifican los pixeles considerados por la proyección de cada voxel, con un valor asignado de tono en cada voxel. Sobrescribiendo los valores del pixel por del voxel, cuando todos los voxels hayan sido proyectados se desearía haber creado una recapitulación del volumen proyectado, el cual se asemeja a una interpretación la superficie de un objeto.

Algunas de las técnicas de descripción de tonalidades (mejor conocidas como "interpretación superficial"), pueden ser usadas para determinar el valor de tono en los voxels proyectados. En este método, una considerable cantidad de tiempo de cálculo es desperdiciado en los voxels que están detrás de los voxels más cercanos al punto de visión, partiendo del valor del tono de tales voxels y siempre sobrescribiendo en los voxels más cercanos. Un método alternativo es usar una proyección del más cercano al más lejano, y considerar solamente los pixeles que requieren ser presentados. Algunas veces este método es más eficiente, que él que está conformado cuando se puede pasar o no pasar el valor calculado de tono, en el proceso de sobrescritura. Una vez que se encuentran todos los pixeles que cubren la proyección de un voxel anterior tendrían que ser presentados.

Una técnica más eficiente es retener solamente los voxels de los bordes del volumen binario, en la proyección tal que solamente los voxels que están a partir de la mitad al frente de un objeto, se incluyen en la proyección para el tono, y el pixel solamente se sobrescribe para un número mínimo de voxels.

Dentro de estas técnicas alternativas sobre la organización de los datos, se emplea una representación conocida como árboles octantes ("octrees"[14]) (ver figura 3.4(C)), en lugar de usar arreglos rectangulares convencionales. El uso de árbol octantes es apropiado ya que el caso de realizar alguna rotación de la escena, ésta se puede determinar con un simple cálculo. Para la organización, en la cual las subregiones podrían ser consideradas, del más lejano al más cercano (muy apropiado y similar al de los arreglos rectangulares descritos anteriormente) [Udopa, 1986].

Por ejemplo, en las condiciones de visualización se asume, el orden del más lejano al más cercano de las subregiones de una escena no rotada (Fig. 3.4(A)), es 5, 6, 7, 8, 1, 2, 3, 4; el mismo orden es aplicado a todas las subregiones de cada una de estas subregiones. La proyección solamente de estas subregiones es representada por todos los nodos en el árbol dentro de la pantalla en este orden. Desde muchas subregiones tales bloques son grandes, conteniendo muchos voxels, y se puede calcular la proyección del bloque por entero sin requerir del cálculo de la proyección de los voxels individuales, ésta es ventaja del método del árbol octante. Usando esta regla entonces las subregiones en nuestro ejemplo son proyectadas en el siguiente orden de nodos: 85, 86, 87, 88, 81, 83, 84, 48, 438. El valor de tono asociado con la subregión podría sobrescribir los valores de pixel en esta proyección.

[14] "octrees": Organización basada en un árbol octante.

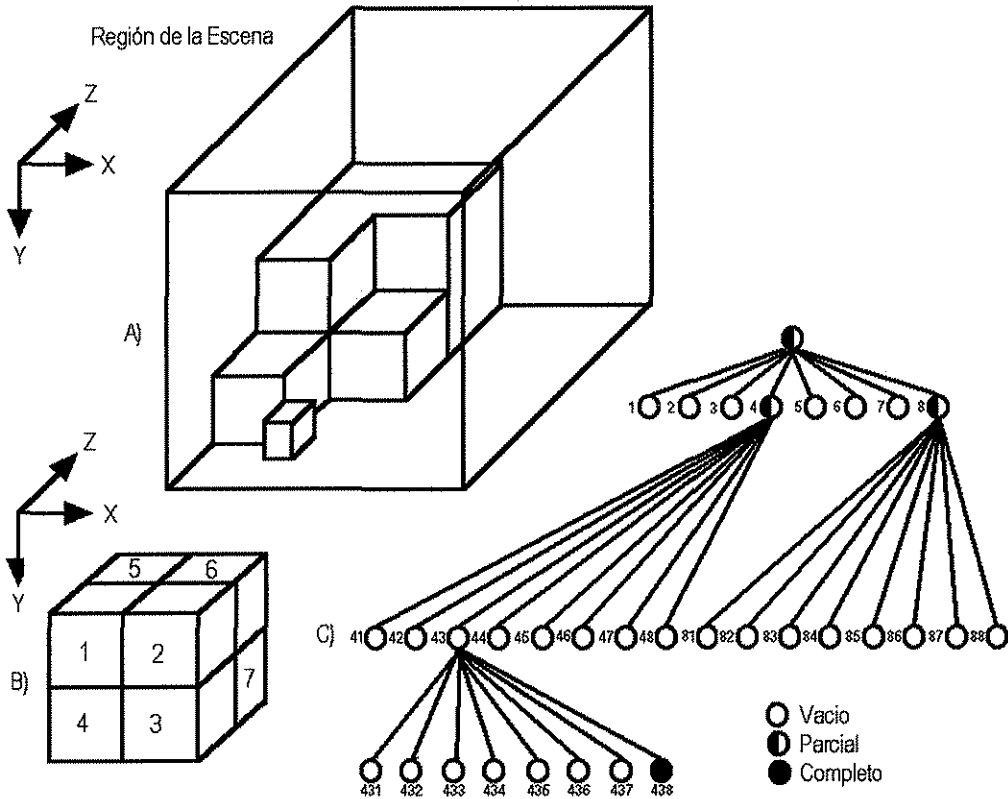


Fig.3.4 A) Escena en 3D, B) Representación de las subregiones, y C) Árbol octante

3.3.2.4 Interpretación en volúmenes en tonos de gris

La técnica más simple para interpretar volúmenes en tonos de grises es asumir que todos los voxels tienen el mismo color, que sus opacidades son determinadas en base de los intervalos de la densidad del voxel como se muestra en la figura 3.2. Se simula una reproyección usando las opacidades asignadas para el control relativo de la opacidad de cada voxel. Se asume que $d_1, d_2, ..., d_n$ representan los valores estimados uniformemente de las densidades de los voxels a lo largo de la trayectoria del rayo, a partir de un pixel contenido en la escena (fig 3.5(A)). Si $\alpha_1, \alpha_2, ..., \alpha_n$ son las opacidades correspondientes a sus densidades ($\delta_1, \delta_2, ..., \delta_n$), y son determinadas a partir de una opacidad asumida con la ayuda del esquema de la figura 3.2, entonces el pixel asignado es un valor de gris (o color fijo de intensidad) [Udopa, 1986]. La densidad resultante es:

$$(\Sigma\ \alpha i\ \delta i) / (\Sigma\ \alpha i) \qquad 3.25$$

En la asignación de altas opacidades, en los intervalos que contiene la densidad de interés, y al resto de los intervalos con las bajas opacidades, se podrían simular "objetos de disolución" y zonas de realce, manteniendo fijo el contexto de las regiones de objetos de menor interés. Un método más sofisticado (desde el punto de vista de la complejidad en la creación de la interpretación) es imponer un modelo óptico de comportamiento, en las

interfaces de los objetos de la escena y asignar un color a un pixel basado en la luz reflejada hacia el pixel.

Aunque el método originalmente usa la proyección del voxel, se emplea el "ray casting" para la simplificación de la descripción de la idea principal. Se supone un muestreo de puntos igualmente espaciados a lo largo de la trayectoria del rayo a partir de un pixel. Esto es equivalente a la determinación de una secuencia de voxels limitados con V_1, V_2, ..., V_n a lo largo de la trayectoria del rayo en el centro de los voxels muestreados (ver fig 3.5(A)). Si α_1, α_2, ...,α_n y C_1, C_2, ..., C_n son las opacidades y los colores asignados a estos voxels respectivamente, se estiman los valores a partir de las opacidades y colores asociados a los voxels de la escena.

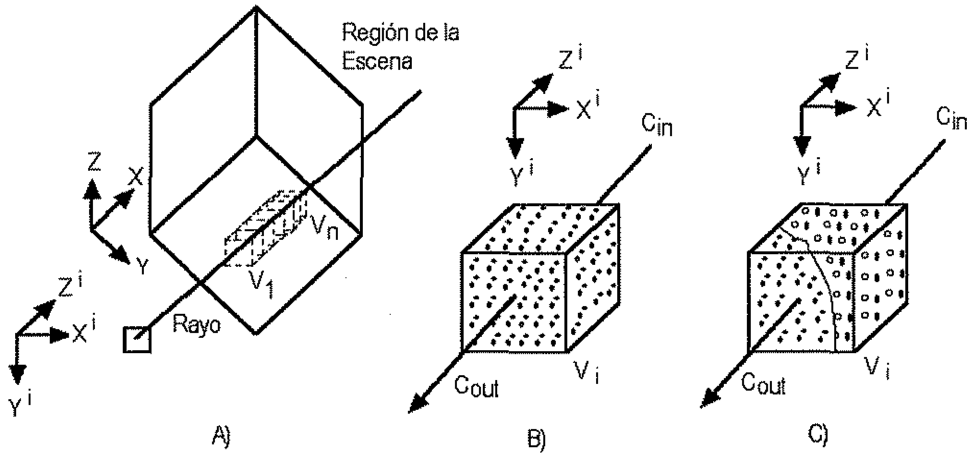


Fig.3.5 A) Puntos muestrados a lo largo de la trayectoria del rayo, B) Medio homogéneo y C) Medio no heterogéneo.

Al considerar que los voxels son luminosos y que emiten luz, cuya naturaleza depende del color de los voxels, y éstos agrupados son visibles en unos cuantos volúmenes. También si se desean ver superficies, mediante el uso de una fuente de luz externa, de tal forma que la luz reflejada desde las superficies podrían ilustrar sus sombras. Además, se estima el color alcanzado del pixel como el resultado de una combinación de estos comportamientos ópticos. Si un voxel V_i a lo largo de la trayectoria del rayo no tiene alguna mezcla entre objetos (por ejemplo de una superficie) pasando a través de éste (fig. 3.5 B), entonces el objeto contenido es mas o menos homogéneo, y el color C_{out} de la luz saliente de éste puede ser determinado como:

$$C_{out} = C_i + (1 - \alpha_i)C_{in} \qquad 3.26$$

El primer término en la suma representa el color emitido por V_i y el segundo el color transmitido por V_i en relación a la parte de color proveniente de C_{in} (se asume que las opacidades son las mismas para las componentes R, G, B de color). Para determinar en todo caso si una superficie pasa a través de V_i, se puede estimar un vector gradiente N en el centro de V_i. La magnitud, $|N|$, de este vector indica la intensidad de la superficie y su

dirección es normal a la superficie del voxel. En la práctica, |N| es raramente igual a cero a la par para los voxels de una región homogénea, pero en las mezclas de objetos, ésta tiene un valor grande.

Cuando una superficie pasa a través de un voxel (fig. 3.5 (C)), C_{out} es determinado por el color C_i^B del objeto detrás de la superficie, el color C_i^F del objeto en el frente de la superficie y el color C_i^S de la luz reflejada desde la superficie. Si todas estas entidades son conocidas se puede aplicar la ecuación 3.26 en tres pasos: primero calcular el color C^B_{out} de la luz de salida del objeto detrás de la superficie, segundo calcular el color C^S_{out} de la luz de salida de la superficie, y finalmente calcular el color C_{out} de la luz de salida del objeto en el frente [Udopa, 1986].

$$C^B_{out} = C_i^B + (1 - \alpha_i^B)C_{in} \qquad 3.27$$

$$C^S_{out} = C_i^S + (1 - \alpha_i^S)^S C^B_{out} \qquad 3.28$$

$$C_{out} = C_i^F + (1 - \alpha_i^F) C^S_{out} \qquad 3.29$$

Empleando un modelo de un material mezclado, descrito anteriormente (fig. 3.5 (B) y 3.5 (C)), se conoce C_i^B, C_i^F, α_i^B y α_i^F. Se puede asumir que la opacidad α_i^S de la superficie es idéntica a la opacidad α_i^B del objeto en la parte posterior. Se pueden determinar cuáles de estos tipos de objetos en la mezcla están en la parte posterior y cuáles en el frente mediante el chequeo del signo de la componente de N. Si este es positivo, entonces la densidad del objeto es mayor en la parte posterior, de otra manera este es el objeto con menor densidad. Restaría conocer solamente de las ecuaciones a C_i^S que depende de las propiedades de reflexión asumidas en la mezcla entre objetos. Algún modelo de reflexión de la superficie puede ser usado para calcular C_i^S. En particular, la ecuación 3.28 puede ser usada con las modificaciones apropiadas para determinar el color y la intensidad de la superficie influenciada por las reflexiones de superficie.

La fórmula general para intensidad de la superficie puede ser dada de la siguiente forma:

$$C_i^S = [f_d(N, L)\, C_d + f_s(N, L)\, C_l]\, |N| + A \qquad 3.30$$

donde C_d es el color de la luz difusamente reflejada, la cual se puede asumir como idéntica a C_i^B, C_l es el color de la reflexión especular asumiendo que es idéntico al color de la fuente de luz. L es un vector, que indica la dirección de los rayos de luz (asumiendo que la dirección de L y el punto de visión son idénticos), f_d y f_s son funciones apropiadas, desinhibiendo la reflexión difusa y especular. Con lo anterior se tiene descrito como es calculado C_{out} conociendo C_{in} para un voxel particular V_i. Para determinar el color a ser asignado al pixel en la fig. 3.5 (A), se inicia desde V_n, con C_{in} = (0,0,0) (negro) para V_n, y calcula este C_{out}, el cual por supuesto es C_{in} para V_{n-1}. Trabajando hacia V_l en esta forma para finalmente calcular C_{out} para V_l, el cual es el color asignado al pixel bajo consideración [Udopa, 1986]. El proceso se podría describir como la repetición de este para todos los pixeles en la pantalla en la creación completa de la interpretación.

Si se deja fuera la emisión y consideramos solamente la reflexión y la absorción para cada voxel, se pueden describir las mezclas de los objetos (superficies) a través de la interpretación volumétrica. Se consideraría la asignación de opacidades a los voxels en base en ambas densidades y la magnitud del gradiente como en la figura 3.2. Entonces, solamente los voxels en las interfaces podrán tener una opacidad significante y éstas junto con las regiones homogéneas podrán tener una opacidad despreciable. C_{out} y C_{in} para cada voxel a lo largo de la trayectoria del rayo y están relacionados como sigue:

$$C_{out} = \alpha_i^{\,S}\, C_i^S + (1 + \alpha_i^{\,S})\, C_{in} \qquad 3.31$$

donde $\alpha_i^{\,S}$, es la opacidad de la superficie a través de V_i, se asume que es igual a la opacidad asignada a V_i, C_i^S como antes, es el color de la luz reflejada desde la superficie a través de V_i.

$$C_i^S = [\, F_d(N,L) + F_s(N,L) + A \qquad 3.32$$

F_d y F_s son las funciones que determinan el color reflejado desde la superficie a través de la reflexión difusa y especular, respectivamente. (En este método no se específica la dependencia del color con estas funciones). El color asignado al pixel bajo consideración es determinado como en el método previo, iniciando desde V_n considerando C_{in} = (R, V, A), como color y trabajando hacia V_1 tomando C_{out} de V_i para el C_{in} de V_{i-1}.

3.4 Discusión.

A lo largo de este capítulo se ha descrito el algoritmo de visualización "volume rendering", detallando las características de las transformaciones posibles desde la escena en el espacio al plano visual. Se han destacado dos aspectos básicos durante el proceso de la interpretación volumétrica, el preprocesamiento de los datos y la interpretación del volumen de datos.

También, se presenta la teoría de "volume rendering" empleando "ray casting", en donde se define una expresión general para determinar la intensidad de luz en un punto del plano visual. Esta intensidad depende de todos los elementos que intervienen a lo largo de una trayectoria descrita por un rayo modelador. Finalmente, se define el algoritmo típico del "volume rendering", el cual es empleado en la implementación del sistema.

Capítulo 4

Procesamiento Paralelo y Lenguajes de Programación Paralela

4.1 Introducción

Los sistemas convencionales de cómputo operan en una forma secuencial, en donde las instrucciones de un programa son ejecutadas una a la vez. Esta característica ha sido forzada por la arquitectura secuencial de las computadoras comunes (arquitectura Von Neumann), en la cual un procesador central es conectado a un banco de memoria por medio de un bus. Desde entonces, la mayoría de las generaciones sucesivas de computadoras disponibles han seguido este diseño. Sin embargo, una gran variedad de problemas asociados a las áreas de visión, simulación, procesamiento de voz, imágenes y de señales digitales poseen un paralelismo intrínseco. De hecho, el resolver este tipo de problemas en forma secuencial se ha reflejado en un gran número de casos computacionalmente intensivos y restringidos. Destacándose sobre todo, cuando se trata con aplicaciones en tiempo real, en donde se manejan intervalos de muestreo muy cortos del orden de milisegundos.

En ciertas aplicaciones reales, se sobrepasan los límites de desempeño de las arquitecturas convencionales. Aunque el uso de la tecnología de integración ha traído como resultado un incremento en la velocidad de los procesadores y en consecuencia de los sistemas de cómputo. La velocidad máxima a la que los componentes electrónicos operan ha marcado un límite en el diseño de procesadores. La alternativa ha sido entonces modificar la arquitectura típica de los sistemas de cómputo.

El procesamiento paralelo ha sido una las alternativas mas viables. La disponibilidad actual de arquitecturas de procesamiento paralelo, que permiten distribuir tanto algoritmos como información sobre un número de procesadores, ha creado nuevas oportunidades para el diseño e implementación de sistemas más rápidos y complejos. El procesamiento paralelo está siendo cada vez más atractivo como un medio para construir sistemas de alto desempeño y confiabilidad.

4.2 Conceptos Básicos del Procesamiento Paralelo

4.2.1 Terminología del Procesamiento Paralelo

I) Concurrencia

La concurrencia es invisible para el usuario, ya que ésta se presenta cuando muchas máquinas en un mismo tiempo requieren acceder a un recurso. Esto implica compartir los recursos de un sistema como lo son la memoria, el tiempo de CPU, los dispositivos de

entrada/salida, etc., por diversos usuarios. En el multiprocesamiento se tiene un método empleado para alcanzar concurrencia en los niveles de trabajo o de programa. Así, la pre-búsqueda de una instrucción es un método para lograr la concurrencia a nivel inter-instrucción. Bajo esta consideración se hace necesario crear una serie de reglas para los usuarios de un mismo recurso que evite la generación de algún conflicto [Haore, 1978], como el caso de un servidor de una base de datos.

II) Procesamiento Paralelo

El procesamiento paralelo enfatiza la manipulación concurrente de los elementos llamados datos, propiedad de uno o varios procesos para la solución de un problema simple. Una computadora paralela es una computadora de múltiples-procesadores con la capacidad de realizar procesamiento paralelo (ver figura 4.1 (C)).

III) Supercomputadora

Una supercomputadora, es una computadora de propósito general capaz de resolver problemas individuales a alta velocidad de cálculo, comparada con otras computadoras empleadas que emplean más tiempo. Las supercomputadoras contemporáneas son computadoras con una arquitectura con características propias del procesamiento paralelos. Algunas tienen un número pequeño de procesadores de gran capacidad de procesamiento, otras están hechas en base a un número extenso de procesadores.

IV) "Throughput"

El "throughput" de un dispositivo es la cantidad de resultados que se producen por unidad de tiempo. Existen muchos caminos para mejorar el "throughput" de un dispositivo; la velocidad del dispositivo puede ser incrementada con ello la concurrencia de procesos se incrementaría, esto es, la cantidad de operaciones que se inician en relación al funcionamiento de varios procesos en un instante de tiempo, puede incrementarse.

V) Pipeline y el Paralelismo de datos

Estos son dos caminos que incrementan la concurrencia en un cálculo. En la figura 4.1 (B), se muestra un cálculo "pipeline" el cual primero se divide en un número de pasos llamados segmentos o etapas (w1, w2, w3,...). Cada etapa trabaja a velocidad limite con una parte particular del cálculo, la salida de una etapa es la entrada a la próxima etapa. Si todos las etapas trabajan a la misma velocidad, entonces se dice que el "pipeline" es total. La razón de trabajo del "pipeline" es igual a la suma de las razones de cada línea de ensamblado: el flujo de resultados es simple y fijo, procedente de restricciones que pueden ser "honoradam" y esto toma tiempo para llenar y drenar el flujo de datos [Quinn, 1994].

El paralelismo de datos, es el usado en una unidad de funcionalidad múltiple para aplicar la misma operación simultáneamente a los elementos de un conjunto de datos. Una k-división incrementa el número de unidades funcionales conduciendo a una k-división que incrementa el "throughput" de un sistema, si no hay un "overhead"[15] asociado con el incremento del paralelismo.

---

[15] "overhead": Es la cantidad de tiempo usado por el software del sistema para procesar, de acuerdo como opera el sistema supervisor tareas de procesos.

En la grafica mostrada en la figura 4.2, se presenta el comportamiento de "speedup"[16] en relación al incremento del número de procesadores, en los casos de emplear una arquitectura paralela basada en el paralelismo de datos y una arquitectura tipo "pipeline".

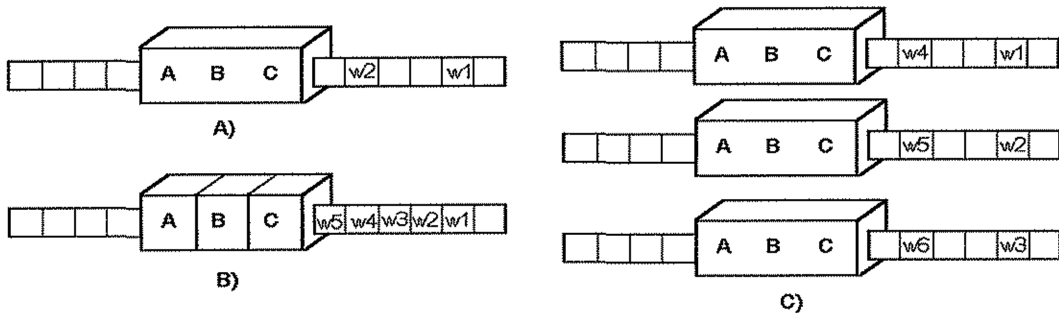


Fig.4.1 A) Proceso secuencial B)Proceso Pipeline y C)Proceso Paralelo.

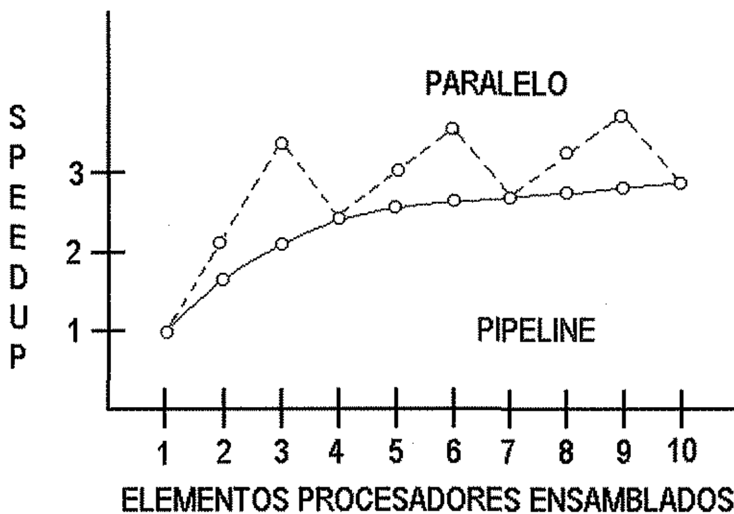


Fig. 4.2 Comparación de "speedup" entre procesos en base a "pipeline" y paralelismo de datos.

VI) Métricas de Desempeño

a) **"Speedup"(s)**: El "speedup" es medido generalmente durante la ejecución del mismo programa, en una variación del número de procesadores. Se define como el tiempo de ejecución empleado por un solo procesador respecto al tiempo de ejecución empleado *n* procesadores [Karp, 1990].

$$\mathbf{s(n)=T(1)/T(n)} \qquad 4.1$$

donde:

T(1): es el tiempo de ejecución empleando un solo procesador.
T(n): es el tiempo de ejecución empleando n procesadores.

[16] "speedup": es una métrica usada para analizar desempeño de procesos paralelos.

b) **Eficiencia(e)**: *Es la relación entre costo/funcionamiento, el significado que encierra* esta unidad es que tan eficientemente se usa el hardware [Karp, 1990].

$$\mathbf{e(n)=T(1)/T(n)n} \quad 4.2$$

en términos de "speedup":

$$\mathbf{e(n)=s(n)/n} \quad 4.3$$

donde:

s(n): es el "speedup" obtenido en n procesadores.
n: es el número de procesadores.

c) **Fracción serial(f)**: Para definir esta métrica, primero se recurre a la Ley de Amdahl [Karp, 1990], la cual en su forma más simple se tiene:

$$\mathbf{T(n)=T_s+(T_p/n)} \quad 4.4$$

donde $\mathbf{T_s}$ es el tiempo que toma la parte del programa que debe ser ejecutado serialmente, y $\mathbf{T_p}$ es el tiempo que toma la parte paralelizable de dicho programa. Obviamente para un procesador, $T(1)=T_s+T_p$, y ahora si se define la fracción serial, $f=T_s/T(1)$, entonces la ecuación puede ser escrita como:

$$T(n)=T(1)f+(T(1)-(1-f))/n$$

en términos de "SpeedUp":

$$\mathbf{f(n) = ((1/s(n)) - (1/n)) / (1 - (1/n))} \quad 4.5$$

La interpretación del valor de f, según la ecuación 4.1, se enfoca en tres aspectos:

a) Los efectos del balance de carga, que son probablemente el resultado de un irregular cambio de f cuando n se incrementa. Se asume que todos procesadores emplean una misma cantidad de tiempo (balance perfecto de carga), pero si algunos procesadores toman mas tiempo que otros, la medición del **s** podría ser reducido dando una medida alta en la fracción serial.

b) El "overhead" de comunicación entre procesadores, el cual es una función incremento monolíticamente de n (típicamente se asume el incremento linealmente ya sea de n o de log n). El incremento del "overhead" se manifiesta en el decremento del "speedup", lo que provoca un ligero incremento en la fracción serial f cuando n se incrementa. El incremento de f denota que la granularidad en las tareas paralelas es fina.

c) La reducción del tamaño del vector en las paralelizaciones de un particular algoritmo. Si la paralelización divide a un extenso vector en pequeños vectores, el tiempo de ejecución puede incrementarse. Este efecto se nota en un ligero incremento en la medición de la función serial cuando se incrementa el número de procesadores [Karp, 1990].

VII) Control en el Paralelismo

El "pipeline" es actualmente un caso especial, de una de las clases más generales de algoritmos paralelos, llamado algoritmo de control paralelo. En contraste, el paralelismo en datos, en el cual el paralelismo es alcanzado por la aplicación de una simple operación a un conjunto de datos, el control en el paralelismo es logrado por la aplicación de diferentes operaciones a diferentes elementos de datos simultáneamente. El flujo de datos es la suma de estos procesos y puede ser arbitrariamente complejo. Si el flujo de datos grafica formas

en una simple ruta directa, entonces se tiene que el algoritmo es del tipo "pipeline" [Quinn, 1994]. Los problemas más realistas pueden explotar ambos paralelismo en datos y el control en el paralelismo.

VIII) Escalabilidad

Un algoritmo es escalable si el nivel de paralelismo intrínseco incrementa en lo más mínimo la linealidad con relación al tamaño del problema. Una arquitectura es escalable si esta mantiene las mismas características de funcionamiento del procesador empleado con relación al incremento del número de procesadores, aunque se use en un problema de mayor tamaño. La escalabilidad del algoritmo y de la arquitectura de un computador son importantes porque esta siempre se usa para resolver problemas extensos en un mismo lapso de tiempo, con sólo emplear un computador con más procesadores. Los algoritmos de paralelismo en datos son más escalables que los algoritmos de control en el paralelismo, porque el nivel de control paralelo es usualmente constante, independiente del tamaño del problema, mientras el nivel del paralelismo en datos es una función dependiente del tamaño del problema.

Quizá los estudiosos de hoy en día de algoritmos y programación sólo estarían viendo una computadora secuencial con algunas excepciones. Hoy en día, más estudiosos acarrean nociones preconcebidas sobre algoritmos y estructuras de datos desde sus experiencias en una máquina secuencial. Nuevas técnicas de resolución de problemas son requeridas para tomar una ventaja total de la potencialidad del hardware paralelo.

4.3 Arreglos de Procesadores, Multiprocesadores y Multicomputadoras

Existen tres modelos importantes del computo paralelo y una serie de diseños de computadoras paralelas asociadas. Los tres modelos son arreglos de procesadores, multiprocesadores y multicomputadoras.

4.3.1 Arreglos de Procesadores

Una organización de procesadores puede ser representada por una gráfica en la cual cada nodo (vértice) representan los procesadores y las aristas representan la ruta de comunicación entre los pares de procesadores. Para evaluar esta organización de procesadores de acuerdo a los criterios que ayuden a entender su efectividad en la implementación eficiente de los algoritmos paralelos en el hardware real [Quinn, 1994]. Estos criterios son:

A) Diámetro: El diámetro de una red es la distancia mas larga entre dos nodos. Un bajo diámetro es mejor, porque el diámetro presenta un límite bajo en relación a la complejidad de la comunicación arbitraria entre los pares de nodos al algoritmos paralelos.

B) Bisección del ancho de una red: La bisección del ancho de una red es el número mínimo de bordes que pueden ser recorridos con un orden para dividir la red en dos mitades. Una alta bisección del ancho de una red es mejor, ya que algunos algoritmos

requieren de cantidades extensas de movimientos de datos, al dividir el tamaño del conjunto de datos por la bisección del ancho de la red, se presenta un límite bajo dada la complejidad del algoritmo paralelo.

C) Número de bordes por nodo: Este número es mejor, si el número de bordes por nodo es constante independiente del tamaño de la red, porque la organización del escalamiento del procesador a sistemas con un número extenso de nodos, es más fácil.

D) Máximo tamaño del borde: Por razones de escalabilidad, éste es mejor si los nodos y los bordes de la red pueden ser extendidos fuera en el espacio tridimensional, así que el tamaño máximo de los bordes es constante independiente del tamaño de la red.

4.3.1.1 Malla de la red

En una malla de la red, los nodos son ubicados dentro de un enrejado de q-dimensiones. La comunicación esta permitida solamente entre los nodos vecinos, aunque en el interior los nodos se comunican con otros 2q procesadores. Una malla en dos dimensiones se muestra en la figura 4.3(A). Algunas variantes de ésta malla, es el modelo que envuelve una conexión entre los procesadores en el mismo borde de la malla, estas conexiones conectan procesadores del mismo renglón o columna (fig. 4.3(B)), renglones o columnas adyacentes (fig.4.3.(C)).

Para evaluar una malla de red de acuerdo a los cuatro criterios descritos antes. Se asume que en la malla, se tienen conexiones del tipo envoltura-alrededor. El diámetro de una malla q-dimensiones con k^9 nodos es q(k-1). Desde el punto de vista teórico, la malla de la red tiene la desventaja de que el direccionamiento de los datos requeridos frecuentemente retienen el desarrollo del tiempo poli-logarítmico del algoritmo paralelo.

La bisección del ancho de una malla de q-dimensión con k^9 nodos es k^{q-1}. El número máximo de bordes por nodo es 2q. La máxima longitud del borde es una constante, independiente del número de nodos, para mallas de 2 y 3 dimensiones. La malla bi-dimensional ha sido una topología popular para arreglos de procesadores [Quinn, 1994].

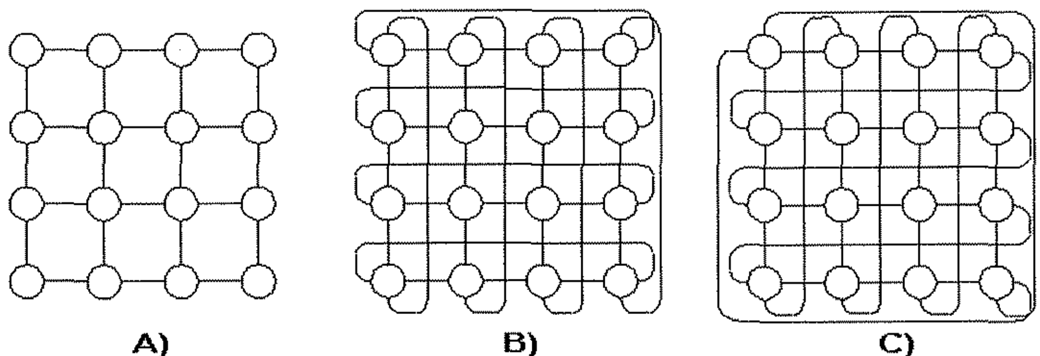


Fig. 4.3 A) Malla de la red en dos dimensiones, B) con conexión en la misma columna o renglón y C) con conexión a columnas o renglones adyacentes.

4.3.1.2 Red árbol binario

En una red árbol binario de 2^k-1 nodos, estos nodos son colocados dentro de un árbol binario completo de profundidad k-1. Un nodo tiene al menos tres ligas, cada nodo interno puede comunicarse con estos dos hijos y cada nodo a otro que tiene raíz puede comunicarse con su padre. El árbol binario tiene un bajo diámetro 2(k-1), pero tiene una baja bisección del ancho de la red (igual a uno). Asumiendo, que los nodos tienen un volumen, es imposible de colocar los nodos en un árbol binario en el espacio tri-dimensional, tal que con el número de nodos se incremente la longitud del borde más extenso y sea siempre menor que una constante específica [Almasi, 1989].

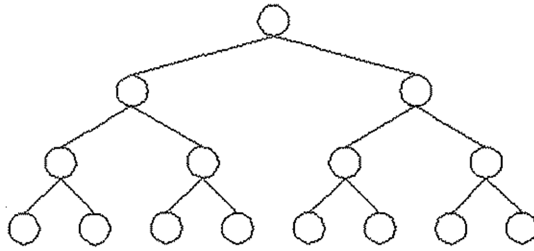


Fig.4.4 Red árbol binario.

4.3.1.3 Red hiper-árbol

Un hiper-árbol representa una aproximación a la construcción de una red con un árbol binario con bajo diámetro, pero una bisección del ancho de la red mejorada. Pensar en la facilidad del camino en una red hiper-árbol de grado k y profundidad d, es considerar la red desde dos diferentes ángulos, (ver figura 4.5). A partir de la vista lateral de una red de hiper-árbol de grado k y profundidad d, se considera a esta como un k-ésimo árbol completo de peso 8 como se muestra en la figura 4.5(B). O bien, la vista frontal de la red hiper-árbol con un árbol binario de arriba abajo de peso d (fig. 4.5 (A)). Juntando las dos vistas (frontal y lateral) se tiene la red completa como se muestra en la figura 4.6. Se muestra un hiper-árbol de grado 4 y peso 2. Un hiper-árbol cuaternario con profundidad d tiene 4^k niveles y $[2^d (2^{d+1}-1)]$ nodos en total. El diámetro de esta red es 2d y su bisección del ancho de la red es 2^{d+1}. El número de bordes por nodo es no más que 6 y la máxima longitud del borde esta en función del incremento del tamaño del problema [Quinn, 1994].

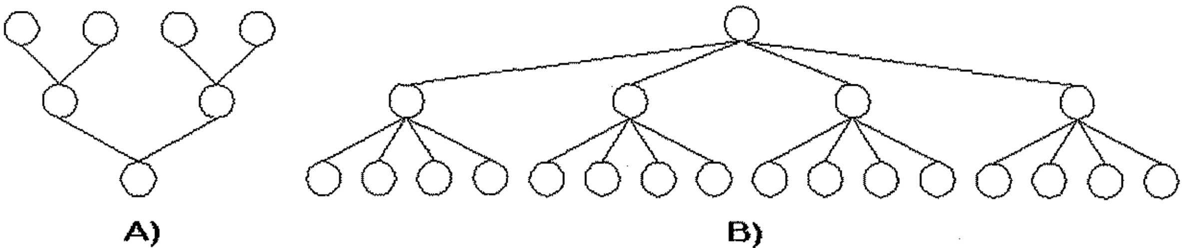


Fig. 4.5 Red Hiper-árbol de grado 4 y profundidad 2. A)Vista lateral y B)Vista frontal.

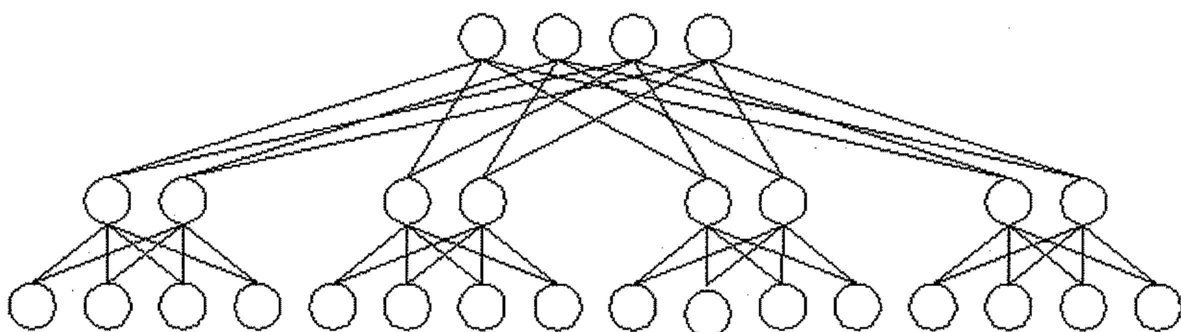


Fig. 4.6 Red Hiper-árbol completa.

4.3.1.4 Red piramidal

La red piramidal combina las ventajas de una malla de red con las de un árbol de red. Una red piramidal de tamaño k^2 es un árbol cuaternario completo con raíz de peso $Log_2(k)$ aumentado con las ligas inter-procesadores, así que los procesadores en cada nivel del árbol forman una malla de red de 2-dimensiones. Una pirámide de tamaño k^2 tiene en la base una malla de red de 2-dimensiones conteniendo k^2 procesadores, la figura 4.7 se muestra una red piramidal de tamaño 16. El número total de procesadores en una pirámide de tamaño k^2 es $(4/3)\ k^2 - (1/3)$. Los niveles de la pirámide son enumerados en orden ascendente tal que la base tiene el nivel 0, con un simple procesador en la cima de la pirámide, con el número de nivel $Log_2(k)$. Cada procesador interno esta conectado a otros nueve procesadores, uno padre cuatro de la malla vecina y 4 hijos.

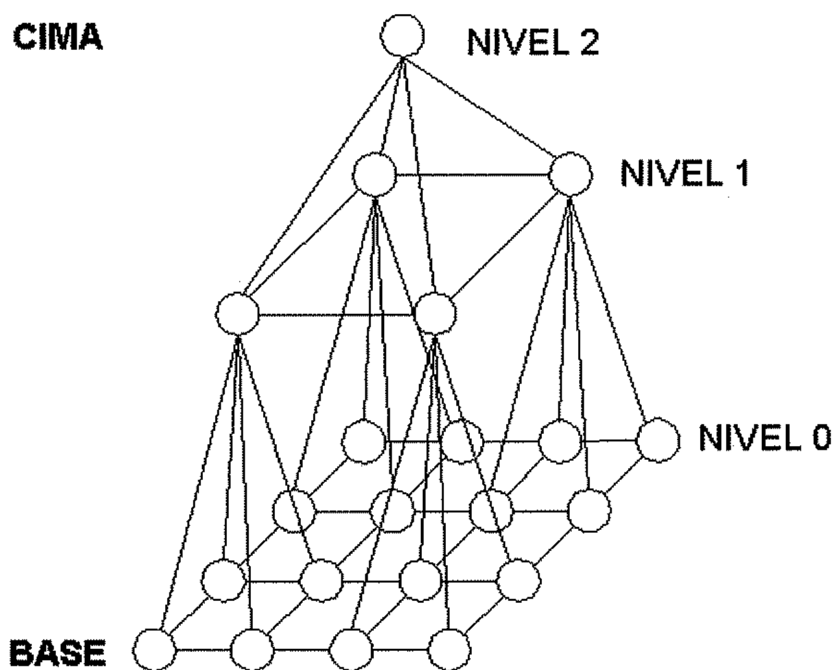


Fig. 4.7 Red Piramidal.

Las ventajas de la pirámide sobre la malla de 2 dimensiones son que la pirámide reduce el diámetro de la red, el cual para una pirámide de tamaño k^2 es 2Log(k). La suma de las ligas del árbol proporciona a la pirámide, una bisección del ancho de la red piramidal más significante, por ejemplo, para una de tamaño k^2 la bisección es 2^k.

El máximo número de ligas por nodo no es más grande que 9, no obstante, del tamaño de la red. Diferente a una malla de 2 dimensiones, sin embargo, la longitud del borde más largo es la red piramidal esta en función del incremento del tamaño de la red [Quinn, 1994].

4.3.1.5 Red "Butterfly"

Una red "Butterfly" consiste de $(k+1)2^k$ nodos divididos entre (k+1) rangos, renglones o filas, y cada uno contiene $n=2^k$ nodos. Los rangos son etiquetados de 0 a k, y a través de estos son algunas veces combinados, al designar a cada nodo 4 conexiones a otros nodos. Dejando al nodo (i,j) de referencia, el j-ésimo nodo en el rango de i>0 es conectado a los nodos dentro del rango i-1, nodo (i-1,j) y nodo(1-i,m), donde m es el entero encontrado por la inversión de i-ésimo bit más significativo en la representación binaria de j. Si el nodo(i,j) es conectado al nodo(i-1,m), entonces el nodo(i,m) es conectado al nodo(i-1,j). La red entera esta echa sobre el patrón de la forma de una mariposa, de ahí su nombre, en la figura 4.8 se muestra una red con 32 nodos.

Con el decremento del número de rangos, el ancho de las alas de la mariposa se incrementa exponencialmente. Por esta razón la longitud del borde de la red más extenso es el número de los nodos de la red incrementada. El diámetro de una red butterfly con $(k+1)2^k$ nodos es 2k y la bisección del ancho de una red del mismo tamaño es 2^{k-1} [Leighton, 1992].

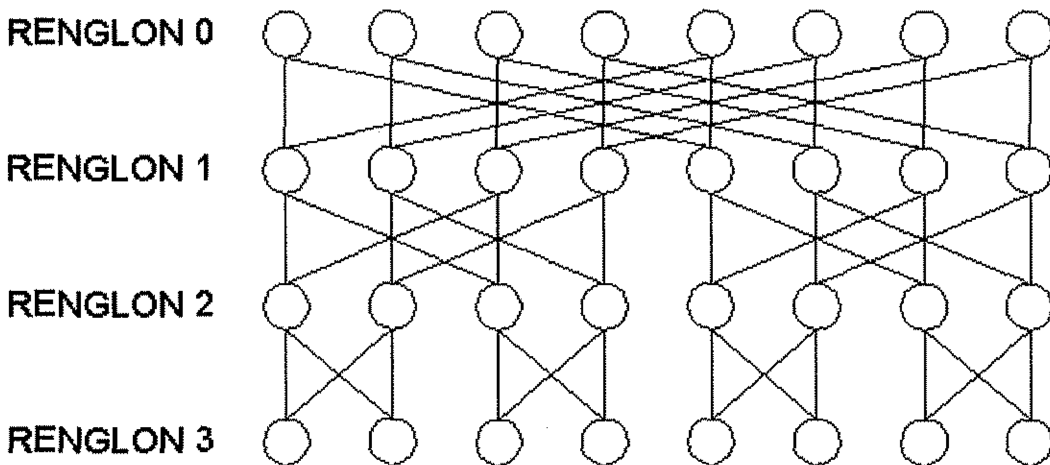


Fig. 4.8 Red Butterfly con 32 nodos.

4.3.1.6 Red hipercubo (cubo-conectado)

Una red cubo-conectado, también llamada una red binaria de n-cubo, es una red "butterfly" con sus columnas colapsadas dentro de los simples nodos. Formalmente esta red consiste de 2^k nodos formando un hipercubo de k-dimensiones. Los nodos son etiquetados 0,1, 2, . , 2^k-1; dos nodos son adyacentes, sus etiquetas difieren exactamente en un bit de la misma posición. Para un red hipercubo (ver fig. 4.9) se tiene, el diámetro de un hipercubo de 2^k nodos es k, y la bisección del ancho del mismo tamaño de la red es 2^{k-1}. La organización del hipercubo presenta un diámetro bajo y una bisección del ancho de la red alta de acuerdo a la extensión del número de bordes por nodo y la longitud del borde más extenso. El número de bordes para un nodo es k veces el logaritmo del número de nodos en la red. La longitud del borde más extenso en una red hipercubo se incrementa con el número incrementado de nodos en la red [Leighton, 1992].

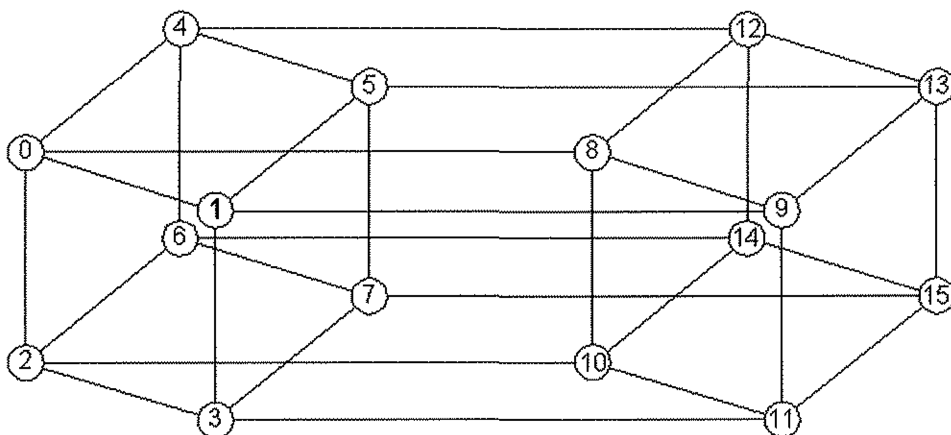


Fig. 4.9 Red Hipercubo cuaternario (16 nodos).

4.3.1.7 Red de ciclos de cubos-conectados

La red de ciclos de cubos-conectados es un hipercubo de k-dimensional cuyos 2^k vértices son ahora ciclos de k nodos formados por las columnas de una red "butterfly" cuyo rango de 0 a k han sido combinados. Para cada dimensión, cada ciclo tiene un nodo conectado a un nodo en la vecindad del ciclo en esa dimensión.

Formalmente, el nodo(i,j) esta conectado al nodo(i,m) sí y sólo sí m es el resultado de la inversión del i-ésimo bit más significativo de la representación binaria de j. Estas conexiones son ligeramente diferentes a las de una red "butterfly", tal que el nodo(i,j) esta conectado al nodo(i-1,m) en la red "butterfly", donde j≠m, y el nodo(i,j) esta conectado al nodo(i,m) en la red de ciclos de cubos-conectados. Sin embargo, en la red de ciclos de cubos-conectados, el nodo(i,j) puede amortiguar la comunicación con el nodo(i-1,m) a las siguientes dos ligas, desde allí esta una ruta directa a partir del nodo(i,m) al nodo(i-1,m). En la figura 4.10 se muestra una red de ciclos de cubos interconectados con 24 nodos.

Comparando con el hipercubo, la organización de los procesadores de ciclo de cubos-conectados tiene la ventaja de que el número de bordes por nodo es tres, una constante independiente del tamaño de la red. Pero, la red de ciclos de cubos-conectados tiene la desventaja de que el diámetro de la red es el doble que él de un hipercubo y la bisección del ancho es mas baja. Dada una red de ciclos de cubos-conectados de tamaño $k2^{k}$, su diámetro es 2k y su bisección del ancho de la red es 2^{k-1} [Leighton, 1992].

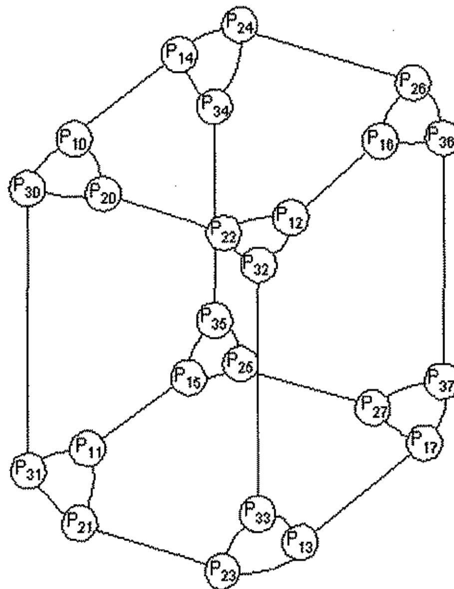


Fig. 4.10 Red de ciclos de cubos interconectados con 24 nodos.

4.3.1.8 Red cambio-intercambio

Una red cambio-intercambio consiste de $n=2^{k}$ nodos numerados 0,1,2,3,4,..,n-1 y dos tipo de conexiones, llamadas cambio e intercambio. Las conexiones intercambio liga pares de nodos cuyos números difieren en su bit menos significativo. La conexión cambio perfecto liga nodos i con nodos 2i módulo (n-1), con la excepción del nodo n-1 es conectado a sí mismo. Las conexiones cambio son indicadas por la flecha de salida y las ligas intercambio son representadas por flechas discontinuas (ver figura 4.11).

Para entender la derivación del nombre cambio perfecto, considere el cambio de una baraja de 8 cartas, numeradas 0,1,2,3,4,5,6,7. Si la baraja es dividida en 2 mitades exactamente y perfectamente cambiada, entonces el resultado es el orden siguiente: 0,4,1,5,2,6,3,7. Ahora, reexaminando la figura 4.11 se tiene que la posición final de la carta que comienza en el índice i puede ser determinado por el siguiente cambio de liga a partir del nodo i.

Asignando A_{k-1}, A_{k-2}, ..., A_1, A_0 son las direcciones de un nodo en la red cambio perfecto expresado en binario. Un datum en esta dirección podría estar en la dirección A_{k-2}, ..., A_1, A_0, A_{k-1}, siguiente en una operación cambio, En otras palabras, el cambio en la dirección de una pieza del dato después de una operación cambio, corresponde a la rotación cíclica izquierda de la dirección por un bit. Si $n=2^{k}$, entonces k operaciones cambio mueven el datum a su localidad original. Los nodos, a través de los cuales un dato idéntico

comienza el viaje en la dirección i como respuesta a la secuencia de cambios, se les llamado el "collar" de i. El "collar" no es más extenso que k y un "collar" más corto que k es llamado un "collar" corto [Quinn, 1994].

Cada nodo en una red cambio-intercambio tiene dos ligas de salida y dos de entrada. La longitud de la liga más extensa se incrementa como una función del tamaño de la red. Un número extenso de ligas tiene ventajas con respecto al diámetro de la red y la bisección del ancho de la red, el diámetro de una red cambio-intercambio es el logaritmo del número de nodos, en una red con 2^k nodos se tiene que el diámetro es 2^{k-1}, y la bisección del ancho de la red es menor a $(2^{k-1})/k$.

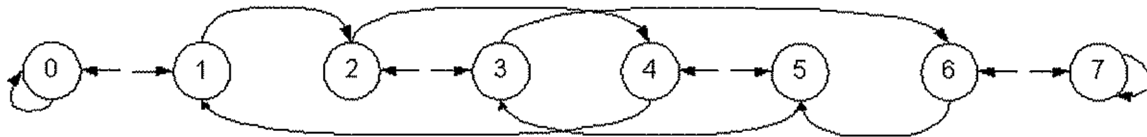


Fig. 4.11 Red Cambio-Intercambio con 8 nodos.

4.3.1.9 Red Bruijn

Una red Bruijn consiste de $n=2^k$ nodos. Asignando $A_{k-1}A_{k-2}...A_1A_0$ como la dirección de cada nodo en la red Bruijn . Los dos nodos que pueden realimentarse vía los bordes dirigidos a partir de que nodo son:

$A_{k-2}A_{k-3}, ... A_1, A_0$ 0

$A_{k-2}A_{k-3}, ... A_1, A_0$ 1

En la figura 4.12 se muestra una red Bruijn. El número de bordes por nodo es una constante independiente del tamaño de la red. La bisección del ancho de una red con 2^k nodos es $2^k/k$, y el tamaño del borde más extenso se incrementa con el tamaño de la red [Leighton, 1992]. Así como las redes cambio-intercambio, las de Bruijn contienen conexiones tipo cambio.

El diámetro de una red Bruijn con 2^k nodos en k, la cual esta cerca de la mitad del diámetro de una red cambio-intercambio con el mismo número de nodos [Quinn, 1994].

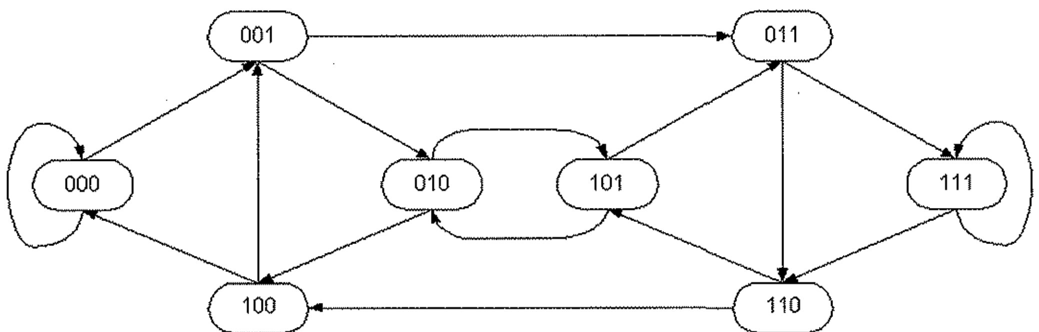


Fig. 4.12 Red Bruijn con 8 procesadores.

4.3.2 Organización del procesador

Una computadora vectorial, es una computadora cuyo conjunto de instrucciones incluye operaciones tanto vectoriales como escalares. Existen dos caminos para la implementación de una computadora vectorial. En un procesador vectorial tipo "pipeline", los vectores fluyen de la memoria a la Unidad de Procesamiento Central (CPU), las unidades aritméticas tipo "pipeline" manipulan a estos vectores. En un arreglo de procesadores se implementa una computadora vectorial, como una computadora secuencial conectada a un conjunto de elementos procesadores idénticos sincronizados, capaces de ejecutar simultáneamente la misma instrucción en diferentes datos (ver figura 4.13). Una computadora secuencial es un CPU de propósito general que almacena el programa y los datos que no están siendo manipulados en paralelo, también ejecuta por partes la secuencia del programa [Quinn, 1994].

Cada elemento procesador tiene una pequeña memoria local que puede ser accesada directamente. Colectivamente, las memorias locales individuales de los elementos procesadores almacenan el vector de datos que son manipulados en paralelo. Cuando la computadora secuencial encuentra una instrucción cuyo operando es un vector, ésta emite un comando a los elementos procesadores para ejecutar la instrucción en paralelo [Lester, 1993]. Así, los elementos procesadores operan en paralelo y las unidades pueden ser programadas para ignorar una instrucción en particular. Esta habilidad para enmascarar los elementos de procesamiento, permite mantener la sincronización a través de las diversas rutas de las estructuras de control.

En el caso de un cálculo normal, el flujo de datos es de la computadora secuencial al arreglo de procesadores, entre los elementos de procesamiento de acuerdo del arreglo de procesadores, y de éstos a la misma computadora. Los elementos de procesamiento envían los valores a cada uno por medio de una red de intercomunicación, la cual esta basada en una configuración de los diversos arreglos de procesadores. Por años, la malla bi-dimensional ha sido el más popular arreglo de procesadores. Los arreglos de procesadores que tienden a ser recomendados, son los hipercubos, las redes cambio-intercambio y las redes de ciclos de cubos-conectados.

Los arreglo de procesadores tienen un mecanismo eficiente para el frente-final para las instrucciones de búsqueda y datos idénticos para los elementos de procesamiento individual. En suma los arreglos de procesadores también soportan el acceso eficiente de una localidad de memoria particular en la memoria de un elemento de procesamiento arbitrario por el de frente-final. La existencia de operaciones rápidas para exploración y la pre-búsqueda de memorias arbitrariamente juegan un rol importante en muchos algoritmos paralelos.

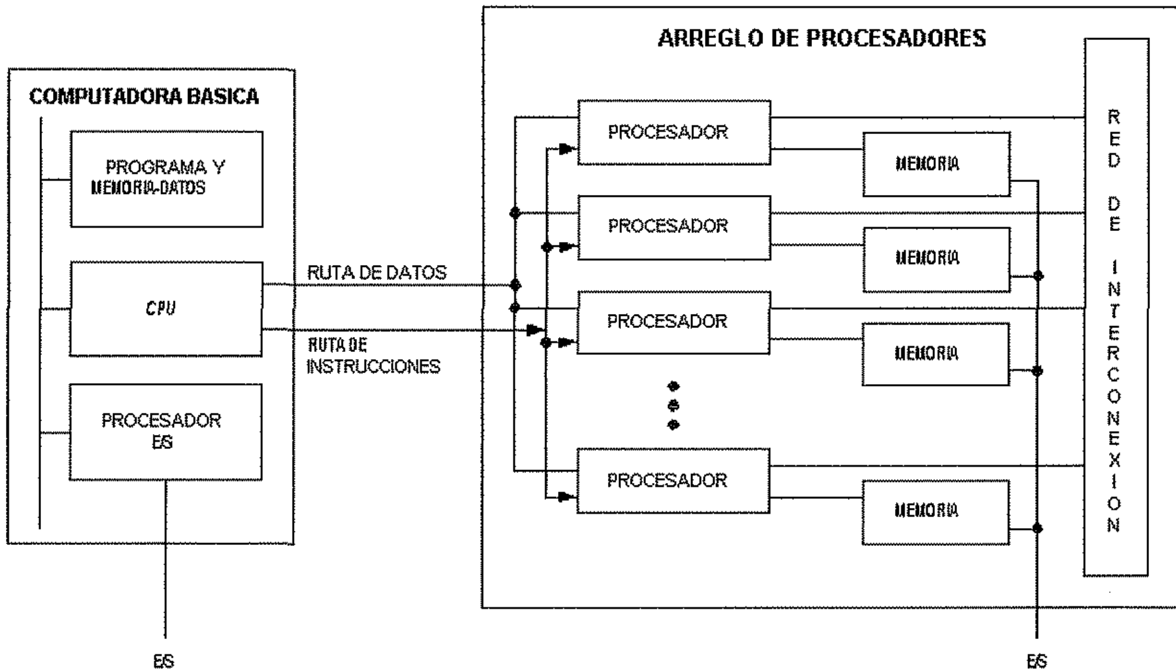


Fig. 4.13 Arreglo de procesadores.

4.3.3 Multiprocesadores

Una computadora basada en multiprocesadores, esta integrada por más de un procesador programable totalmente, y cada uno capaz de ejecutar su propio programa. Estas computadoras con múltiples CPU presenta una memoria compartida [Lester, 1993], que de acuerdo a su configuración se clasifican en: Memoria de Acceso Uniforme (MAU) (la memoria compartida es centralizada) y Memoria de Acceso No Uniforme (MANU) (la memoria compartida es distribuida).

4.3.3.1 Multiprocesadores con Memoria de Acceso Uniforme

El patrón más simple de intercomunicación del procesador asume que todos los procesadores trabajan a través de un mecanismo compartido centralizado. Existe una variedad de caminos para la implementación de este mecanismo de interruptores, incluyendo un bus común a la memoria global, un panel de interruptores y una red del paquete de interruptores (ver figura 4.14). Los sistemas usan un bus, están limitados en el tamaño, solamente así algunos procesadores pueden compartir el bus antes que éste comience a saturarse. En el caso de los sistemas, usando un panel de interruptores, el costo del uso del interruptor pronto se manifiesta como un factor dominante al limitar el número de procesadores que pueden ser conectados. Los multiprocesadores basados en redes de interruptores pueden concebiblemente contener un número extenso de procesadores, a través del surgimiento de computadoras no comerciales usando esta arquitectura [Quinn, 1994].

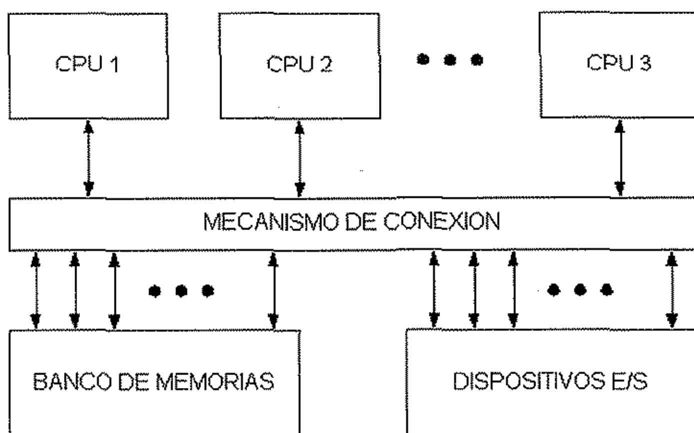


Fig. 4.14 Multiprocesadores con memoria de acceso uniforme.

4.3.3.2 Multiprocesadores con Memoria de Acceso No Uniforme

Estos multiprocesadores están caracterizados por un espacio de direccionamiento compartido, la memoria del multiprocesador MANU es distribuida, como se muestra en la figura 4.15. Cada procesador tiene algo de memoria y el espacio de direccionamiento compartido dentro de un procesador MANU esta formada por la combinación de estas memorias locales. El tiempo requerido para accesar una localidad de memoria compartida dentro de un multiprocesador MANU depende de que tanto la localidad es local para el procesador [Quinn, 1994].

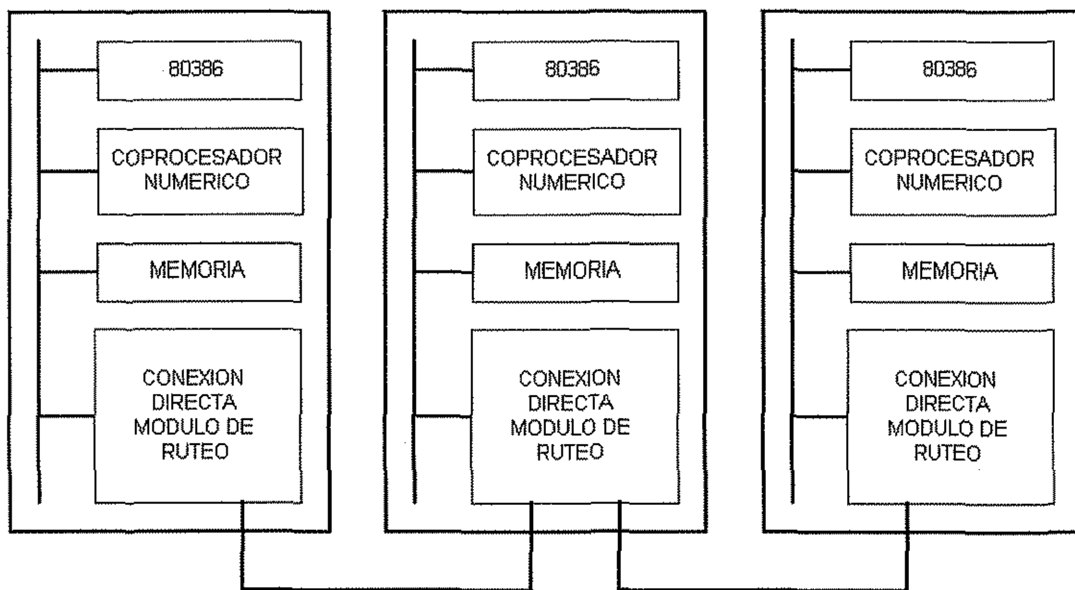


Fig. 4.15 Multiprocesadores con memoria de acceso no uniforme.

4.3.4 Multicomputadoras

Otra arquitectura múltiple CPU, es la multicomputadora como la mostrada en la figura 4.16, en la cual no se tiene una memoria compartida sino que cada procesador tiene memoria privada y los procesos de interacción ocurren a través del envío de mensajes [Lester, 1993]. Dentro de las multicomputadoras comerciales se encuentran los sistemas basados en transputer T800, los cuales están caracterizados por la utilización del envió de mensajes del tipo almacena-envía por el software empleado. Envía un mensaje desde un procesador a otro procesador no adyacente, cada procesador intermedio a lo largo de la ruta del mensaje puede almacenar el mensaje completo y entonces enviar el mensaje al próximo procesador debajo de la línea. Aunque los datos transferidos son hechos a través del canal de Acceso Directo a la Memoria (DMA), el CPU es interrumpido cada vez que una transferencia DMA es iniciada [Quinn, 1994].

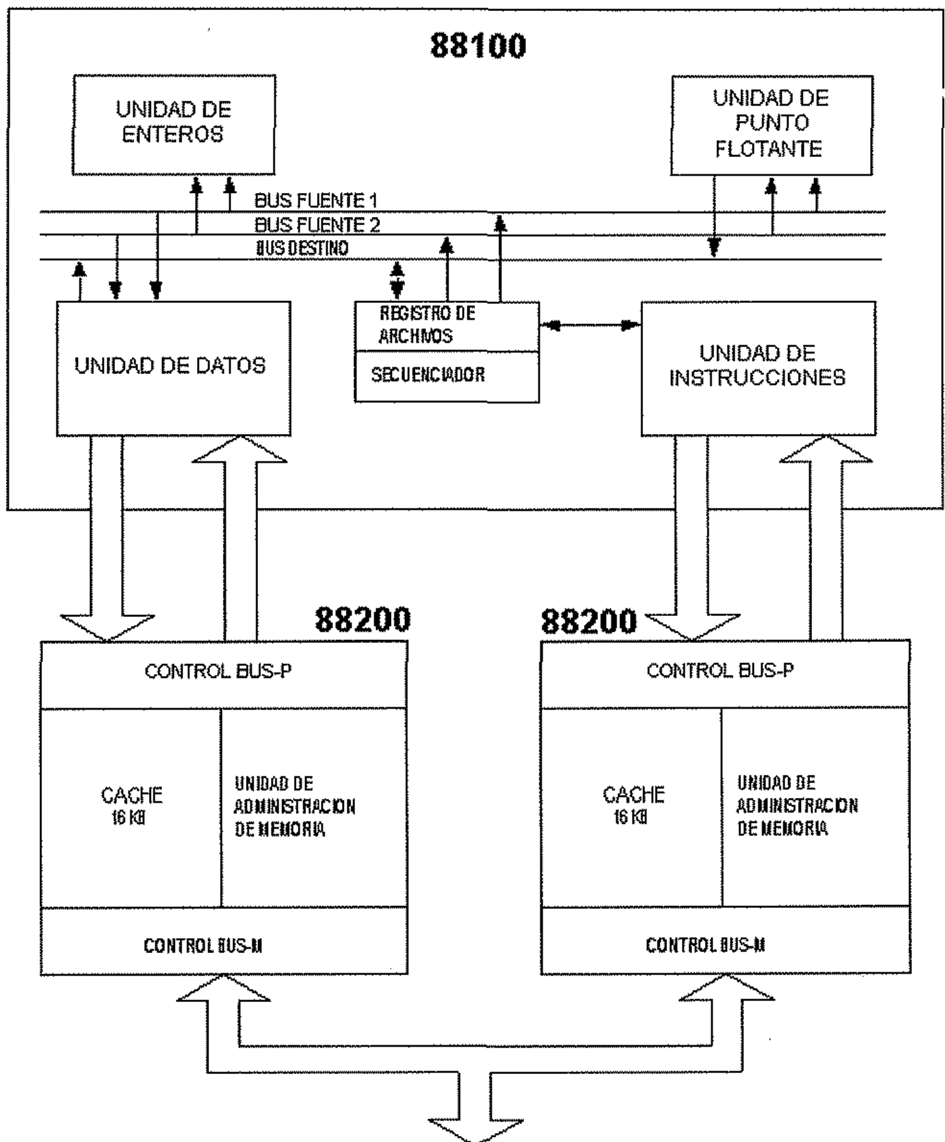


Fig. 4.16 Multicomputadora.

4.4 Taxonomía de Flynn

La taxonomía de Flynn [Flynn, 1996] es la clasificación esquemática más conocida para arquitecturas de computadoras seriales o paralelas. Flynn basa su taxonomía de arquitecturas de computadoras en los conceptos duales de flujo de instrucciones y flujo de datos. Un flujo de instrucción es una secuencia de instrucciones procesadas por una computadora. Un flujo de datos es una secuencia de datos manipulados por un flujo de instrucciones. Flynn categoriza una arquitectura por la multiplicidad del hardware usado para manipular una instrucción y el flujo de datos. La multiplicidad es tomar como el máximo número posible de operaciones simultáneas (instrucciones) u operando (datos) comenzando en la misma fase de ejecución en la más restringida componente de la organización. Existen cuatro clases de computadoras que resultan dados la multiplicidad de la instrucción y el flujo de datos [Hockney, 1981].

A) **SISD:** Simple Instrucción, Simple Dato. Representa el sistema uniproceso convencional de computadora, la clásica arquitectura de Von Neumann.

B) **SIMD:** Simple Instrucción, Múltiple Dato. Incluye más de un sistema de múltiples procesadores, donde las operaciones ocurren sincrónicamente y usualmente comparten espacio común de memoria.

C) **MIMD:** Múltiple Instrucción, Múltiple Dato, incluye más de un sistema de múltiples procesadores, las operaciones ocurren asincrónicamente y usualmente comparten espacio común de memoria.

D) **MISD:** Múltiple Instrucción, Simple Dato, esta es la menos intuitiva, no tiene sentido.

4.5 Lenguajes de Programación Paralela

Cada Lenguaje Paralelo puede tratar con certeza el tema explícitamente o implícitamente. Se tiene un camino para crear procesos paralelos y otro para coordinar las actividades de esos procesos [Quinn, 1994]. Algunas veces los procesos trabajan en sus datos propios y no interactúan. Pero cuando los procesos intercambian resultados, ellos pueden comunicarse y sincronizarse con cada uno de los otros. La comunicación y sincronización puede ser acompañado por variables compartidas o por el paso de mensajes.

Existen dos tipos de métodos de sincronización: sincronización por precedencia y sincronización por mutua exclusión. La sincronización precedente garantiza que un evento no comience hasta que otro evento haya finalizado. La sincronización de mutua exclusión garantiza que solamente un proceso en un tiempo entra la sección crítica del código donde una estructura de datos esta compartida y es manipulada [Almasi, 1989].

4.5.1 FORTRAN 90

En 1978 el comite técnico acreditado de la ANSI, X3J3, comenzó trabajando en una nueva versión del Lenguaje Fortran. En los 90's el lenguaje resultado, Fortran 90, es el superconjunto de Fortran 77. Este incluye todas las características de Fortran 77, más: operaciones de arreglos, mejora de facilidades para cálculos numéricos, la síntesis permite a los procesadores soportar enteros, paquetes lógicos, conjunto de cadenas extensas, alta precisión de números reales y complejos; el usuario define el tipo de datos, estructuras y apuntadores; localización de almacenamiento dinámico; los módulos soportan tipo de datos abstractos; procedimientos internos y procedimientos recursivos; mejora de facilidades de entrada-salida; nuevas estructuras de control; nuevos procedimientos intrínsecos; forma de fuente orientado-terminal [Almasi, 1994].

En Fortran 90 se programa en un modelo de computación paralela similar a una máquina paralela de acceso aleatorio. Un CPU y una unidad vectorial comparten una memoria simple. El CPU ejecuta instrucciones secuenciales, accesa a variables almacenadas en la memoria compartida. Ejecuta operaciones paralelas, el CPU controla la unidad vectorial, la cual también almacena y busca datos en y desde la memoria compartida [Quinn, 1994].

4.5.2 Modelo de programación en C

Los programadores en C programan en una computadora tipo SIMD, que consiste de un procesador central a cual se agrega un procesador paralelo adaptable. Este procesador central almacena variables secuenciales y ejecuta código secuencial. Mientras que los procesadores agregados almacenan variables paralelas y ejecuta las porciones paralelas del programa. Cada elemento de procesamiento de la parte agregada tiene su propia memoria local.

Existe un flujo simple de control, en algún instante de tiempo cualquiera el procesador de la parte de enfrente esta ejecutando una operación secuencial, o los procesadores de la parte posterior están ejecutando una operación paralela. El arreglo del procesador de la parte posterior es adaptable, esto es, los programadores pueden seleccionar el tamaño y forma de los elementos de procesamiento que se deseen activar. Estos parámetros son independientes del tamaño y la topología de la máquina física que soportan a la computadora. Por esta razón, algunas veces se refiere a los elementos de procesamiento como procesadores virtuales. Además, la configuración del arreglo del procesador de la parte posterior es adaptable entre las diferentes partes del mismo programa [Quinn, 1994].

4.5.3 C secuencial

En programación paralela bajo DYNIX, las computadoras secuenciales corren el Sistema Operativo DYNIX, una versión de UNIX^{TM}, estructurado para el ambiente de multiprocesadores. Las peticiones a un sistema-operativo típicamente se encontraban el sistema UNIX, ahora DYNIX provee un conjunto de rutinas para facilitar el procesamiento paralelo. Los lenguajes de programación paralela convencionales emplean un hardware

secuencial, y son simples extensiones de lenguajes secuenciales que permiten a los programadores declarar variables compartidas que interactúan vía mutua exclusión y por barrido de sincronización, siendo los lenguajes resultados primitivos. Para lo cual se defines procesos paralelos en al secuencia coordinada con sus actividades para el acceso a estructuras de datos compartidos [Quinn, 1994].

El modelo de programación paralela estándar en el C secuencial es expansivo e intuitivo. Un programa comienza la ejecución como un simple proceso. Este proceso tiene la responsabilidad de la ejecución de las partes del programa que son inherentemente secuenciales. Cuando el control en una parte del cálculo pueda funcionar en paralelo de un proceso original, éste la desvía a otros nuevos procesos y cada proceso funciona con su esquema de trabajo. El total de los nuevos procesos que accesan a los datos compartidos no pueden exceder al número de procesadores físicos no menor a uno. Porque existe en algunos CPU's un proceso activo, cada proceso puede ejecutar en su propio CPU. Esto permite una mayor reducción en el tiempo de ejecución, asumiendo que la computadora no esta ejecutando otros trabajos. Cuando el control alcanza una porción inherentemente secuencial del cálculo, solamente el proceso original ejecuta el código, el resto de los procesos esperan hasta que el control alcance otra porción del cálculo que pueda ser dividido en piezas y ejecutarlas convencionalmente. Los ciclos de programa a través de estos dos modos son considerados hasta la terminación.

Monitores:

Los algoritmos paralelos implementados en los multiprocesadores requieren de un proceso para ejecutar una serie de operaciones en una estructura de datos compartida, como si esta fuera una operación atómica. En la parte del código en la cual la mutua exclusión pueda ser forzada es llamada región crítica, y en estas es fácil cometer algún error, cuando se comparten recursos en un camino sistemático. Un camino para estructurar el acceso a los recursos compartidos es mediante el empleo de monitores. Un monitor consiste de la representación de variables del estado de algún recurso, los procedimientos que implementan operaciones del recurso son inicializadas en el código. Los valores de las variables son inicializadas en algún procedimiento cuando el monitor es llamado, estos valores son retenidos entre el procedimiento de invocación y pueden ser accesados solamente por los procedimientos en el monitor. Los procedimientos monitores reensamblan procedimientos ordinarios en el lenguaje de programación con un significado de excepción. La ejecución de los procedimientos en el mismo monitor esta garantizado para ser mutuamente exclusivo. Los monitores son estructurados en el camino de implementación de la mutua exclusión.

4.5.4 Programación nCUBE

En la programación nCUBE usualmente se escribe un programa simple que se ejecuta en cada nodo procesador. En cada nodo el programa comienza la ejecución tan pronto como el sistema operativo se carga dentro del nodo. Para implementar una aplicación de datos paralelos, el programador asigna a cada procesador la responsabilidad del almacenamiento y la manipulación de su forma de la estructura de datos. Esto es llamado programación en estilo SPMD. Los procesadores trabajan en sus propios datos

locales hasta que ellos alcancen un punto en el cálculo cuando ellos requieran interactuar con otros procesadores a: "swap"[17] de datos idénticos, comunicación de resultados, ejecución de una operación combinada en resultados parciales, etc. Si un procesador inicia una comunicación con un vecino que no tiene finalizado su propio cálculo, entonces el inicializador puede esperar por su pareja y así actualizar la información. Una vez que los procesadores han manifestado la necesidad de comunicación, ellos pueden resumir el trabajo en datos locales. Un camino para ver el cálculo total es considerar las actividades de los procesadores con periodos ocasionales de espera, que incluyan un ciclo entre la computación y la comunicación [Quinn, 1994].

Un programa SPMD ejecutado en un multicomputador como el nCUBE consiste de segmentos alternativos en los cuales los procesadores trabajan independientemente con datos locales, entonces el intercambio de valores con otros procesadores es a través de las llamadas de rutinas de comunicación.

4.5.5 OCCAM

La compañía Inmos de Gran Bretaña desarrolló OCCAM como un lenguaje de programación para sus series de procesadores llamados transputer. El diseño de OCCAM fue fuertemente influenciado por el trabajo de Hoare en la comunicación de procesos secuenciales. El lenguaje OCCAM esta asociado con la evolución del chip transputer, por esta razón el hardware de los transputers sucesivos es más sofisticado. En el nivel macro, el programador en OCCAM ve un cálculo paralelo como una colección de procesos ejecutándose asincrónicamente mediante un protocolo de envío de mensaje en sincronía. Un programa en OCCAM esta construido en base a tres tipos de procesos primitivos: asignación, entrada y salida. El programador puede ensamblar más procesos complicados por la especificación cuando ellos puedan ejecutar secuencialmente y cuando ellos puedan ejecutar en paralelo.

4.5.6 C-LINDA

Linda consiste de una serie de operaciones que trabajan en un espacio "tuple", una memoria compartida asociada. La incorporación de operaciones Linda dentro de los campos del lenguaje base secuencial y un lenguaje de programación paralela. Linda es un modelo MIMD de computación paralela. El programador Linda prevé un grupo de procesos ejecutándose asincrónicamente que interactúan por medios de una memoria compartida asociada, espacio "tuple"[18]. El espacio "tuple" consiste de una colección de "tuples" lógicos. El paralelismo es alcanzado por la creación de procesos "tuples", los cuales son ejecutados en los procesadores requeridos para hacer el trabajo. Los procesos paralelos interactúan por datos "tuples" compartidos. Después que un proceso "tuple" ha finalizado

---

[17] "swap": es un espacio reservado en tu disco duro para poder usarse como una extensión de memoria virtual del sistema.

[18] "tuple": modelo de memoria virtual compartida que provee una comunicación y sincronización interprocesos que es independiente de la plataforma.

la ejecución, éste retorna el resultado al espacio "tuple" como un dato "tuple". En muchos lenguajes MIMD, como el OCAM, los procesos pueden interactuar con cada uno de los otros por caminos complicados. Los proponentes de Linda dicen que, forzando todos los procesos las interacciones ocurren a través de un espacio "tuple" que simplifica la programación paralela [Quinn, 1994].

4.5.7 Multi-Pascal

El Multi-Pascal es una extensión de Pascal, con el incremento de características para la creación e interacción de procesos paralelos. Multi-Pascal tiene muchas de las características estándares del paralelismo que son encontrados en una variedad de otros lenguajes de programación paralela. Posee un alto nivel de abstracción del programación paralela que ayudan a simplificar los procesos de programación. Esta diseñado para ser una máquina independiente y puede ejecutar en una amplia variedad de computadoras paralelas incluyendo multiprocesadores con memoria compartida y multicomputadoras basadas en envió de mensajes entre procesadores con memoria local.

Multi-Pascal tiene características que permiten la creación dinámica de procesos paralelos para ejecutarlos en procesadores. Los datos pueden ser compartidos por los procesos paralelos a través del uso de memorias compartidas, las cuales están definidas en la memoria compartida de la abstracción del software y localizada en el hardware de la computadora multiprocesadores. Además tiene variables de canal, que son usadas para transmitir datos desde un procesador a otro, y las características estándares de los "spin locks" que son especialmente empleados en la operación de operaciones atómicas [Lester, 1993].

4.5 Discusión.

En este capítulo se han presentado conceptos generales y definiciones básicas del procesamiento en paralelo.

Modelos importantes de cómputo paralelo han sido descritos (arreglos de procesadores, multiprocesadores y multicomputadoras), así como, los lenguajes de programación paralela mas comúnmente utilizados.

Cabe destacar, que en el siguiente capítulo se presenta la implementación del algoritmo de visualización "volume rendering", empleando técnicas de procesamiento en paralelo.

Capítulo 5

Implementación Paralela del Algoritmo de Visualización "Volume Rendering"

5.1 Introducción

El procesamiento paralelo, como se revisó en el capítulo anterior, se relaciona con la forma de ejecutar una actividad en un arreglo de procesadores (hardware) y al mismo software empleado para que esta actividad se subdivida en tareas a realizar por cada elemento procesador. Uno de los propósitos de este trabajo es seleccionar un algoritmo que presente características que permitan aplicar las técnicas de computo paralelo. El algoritmo típico "volume rendering" (descrito en el capítulo 3), es el seleccionado para la implantación, en donde interviene más de un elemento procesador y se distribuyan las tareas o datos a los procesadores. En el último caso, la paralelización de datos se puede llevar a cabo distribuyendo la información a más de un procesador, de esta forma todos ejecutan un mismo proceso al mismo tiempo sobre la parte de datos que les fue asignada. Existiendo una etapa de recolección de los resultados y con ello tener el total de la información procesada. En éste capítulo, se presenta la implementación del algoritmo "volume rendering" considerando la paralelización de los datos, empleando el lenguaje C ANSI paralelo y una plataforma computacional basada en una arquitectura de memoria distribuida, compuesta por procesadores tipo "transputer" IMST805 [INMOSa, 1992]. Por último, se presenta un caso de estudio empleando la implementación propuesta.

5.2 Etapas de la Implementación del Algoritmo para Obtener una Imagen Tridimensional

Para la implementación del algoritmo típico "volume rendering", se define un proceso mediante el cual se tiene como información de entrada un volumen de datos, en este caso un conjunto de imágenes bidimensionales (secciones transversales de algún objeto), y como resultado se obtiene una representación tridimensional de dicho volumen. El proceso esta compuesto por etapas, la cuales representan preprocesamientos definidos en la estructura del algoritmo. Se definen cuatro etapas importantes: histograma, segmentación, opacidad e integración.

Cada una de estas etapas, realiza una tarea que permite obtener características del conjunto de datos, al efectuar operaciones sobre dichos datos. Además de presentar una secuencia entre las mismas de tal forma que los resultados de una son los datos de entrada de la siguiente (histograma-segmentación-opacidad). Finalmente la etapa de integración, como su mismo nombre la define, tiene como función integrar los resultados de la etapa de opacidad con los datos del volumen de información del conjunto de imágenes, para obtener

una representación tridimensional (imagen tridimensional). En la figura 5.1 se muestra a manera de bloques la implementación del algoritmo, considerando las cuatro etapas antes mencionadas.

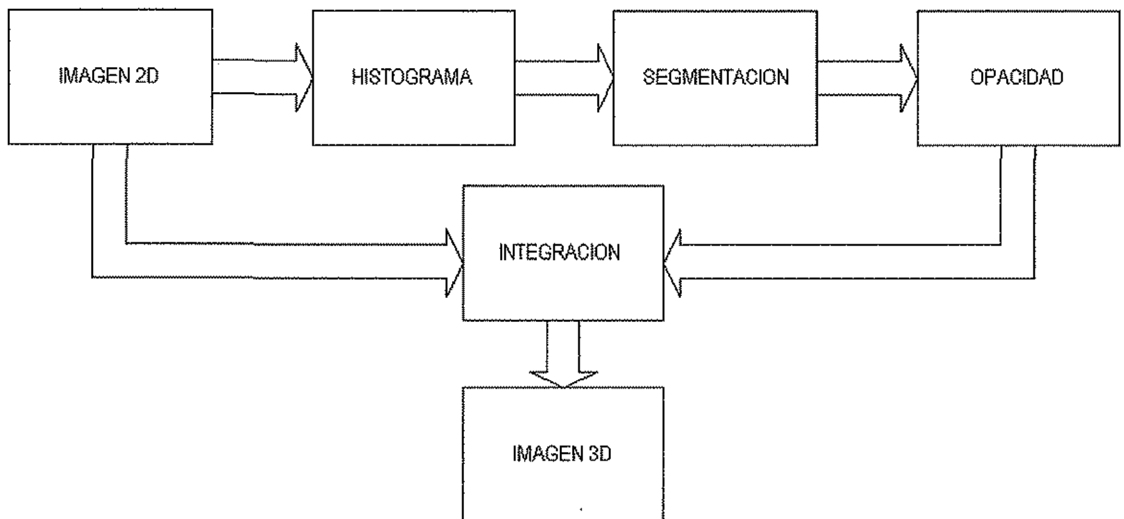


Fig. 5.1 Diagrama de bloques del proceso para obtener una imagen 3D.

A partir del diagrama a bloques de las etapas que constituyen la implementación del algoritmo. Se plantea la estructura del programa a desarrollar, empleando pseudcódigo, de tal modo que se tiene lo siguiente:

For imagen_bidimensional

- En la primera etapa, a cada imagen del conjunto, se le aplican las siguientes funciones:
  Histograma(imagen_bidimensional);
  Segmentación(imagen_bidimensional);
  Tabla_Opacidad(imagen_bidimensional);

For imagen_bidimensional

- En la segunda etapa, que corresponde a la aplicación de la función integración, en donde interviene el conjunto de imágenes bidimensionales, el cual dependiendo del número de procesadores se divide al mismo, y junto con las tablas de opacidad generadas se obtiene como resultado una imagen tridimensional.
  Imagen_tridimensional=INTEGRACION(imagen_bidimensional,Tabla_Opacidad(imagen_bidimensional));

A continuación se hace una descripción de cada una de éstas etapas:

A) *Etapa Histograma*

Como resultado de esta etapa, se obtiene los histogramas de cada imagen, cuya información es usada en los siguientes pasos. Un histograma es la representación gráfica de la frecuencia de ocurrencia de cada intensidad de luz (tono de gris) en una imagen. Esta información no determina la localización o distribución de la imagen, esto es, si consideramos dos imágenes con un fondo único que contenga un objeto, y en una de ellas se cambia la posición del objeto y obtenemos los histogramas, al compararlos se debe concluir que son iguales. En resumen el histograma para una imagen específica es único, sin embargo la imagen para un histograma específico no es única [Castleman, 1979].

B) *Etapa Segmentación*

La clasificación o segmentación consiste en la descomposición de una imagen en grupos de datos que poseen una característica, y ésta consiste en que cada grupo pertenece a un objeto contenido en la imagen. En este caso se hace necesario emplear algún método que permita obtener una segmentación de cada imagen para clasificar la información contenida en la misma. Para esta tarea de segmentación se empleará el Método de Otsu [Otsu, 1979], descrito en el capítulo 3. Se pueden definir "n" grupos de objetos, el número depende del tipo de información contenida en la imagen.

C) *Etapa Opacidad*

En esta sección se le asigna a cada valor de voxel de la imagen una opacidad que esta en función de su ubicación, dicho valor está dentro de las clases identificadas en la imagen. Drebin [Watt,1992] emplea un clasificador probabilístico, a partir del conocimiento de las distribuciones de clases en una imagen que define la opacidad asociada a un voxel como:

$$C = \sum_{i=1}^{n} P_i C_i$$

donde:
n; es el número de materiales en el voxel.
P_i; es el porcentaje del material en el voxel.
C_i; es el color del material multiplicado por su opacidad que es $C_i = (\alpha_i R_i, \alpha_i G_i, \alpha_i B_i, \alpha_i)$, donde α_i se asume como independiente del ancho de onda.

A partir de la anterior definición se plantea el empleo del siguiente procedimiento, en la designación de la opacidad α para los valores de los voxeles de la imagen:

1) Para el rango de valores de voxel, se puede identificar, mediante la segmentación, la clase al que pertenece cada uno de éstos valores.
2) Se determina un porcentaje, en base a la relación de la frecuencia del valor del voxel, entre el valor máximo de frecuencia dentro de la clase correspondiente.
3) Este porcentaje representa la opacidad del voxel, de tal forma que, en el caso de que el valor de frecuencia sea el valor máximo, se le asigna una opacidad de valor 0 (voxel transparente). Si el valor de la frecuencia es el mínimo, a éste le corresponde una opacidad máxima 1 (voxel opaco).
4) Finalmente se crea una tabla de opacidad para cada uno de las imágenes del conjunto de datos, que son empleadas en la etapa de integración.

D) *Etapa de Integración*

En esta etapa se realiza el proceso de la composición volumétrica al aplicar la ecuación 3.26, de acuerdo a los planteamientos de Drebin [Watt, 1992], y como resultado se obtendrá la representación de una imagen tridimensional.

$$C_{out} = C_{in}(1 - \alpha_i) + C_i\alpha_i$$

5.3 Descripción de la Estrategia de Paralelización

En la implementación del algoritmo se tienen los siguientes elementos: los procesadores (de uno hasta dieciséis procesadores tipo "transputer" IMST805) y el lenguaje de programación paralela C ANSI (con bibliotecas de funciones de procesamiento en paralelo), proporcionado en la plataforma de desarrollo MCP100 [INMOSa, 1992].

En base a las características del algoritmo (descriptas anteriormente) se plantean las siguientes consideraciones:

A) Para determinar la opacidad del rango de valores de los voxels de una imagen, es necesario disponer de las características de toda la escena, como lo es el histograma de la imagen. La información original del conjunto de imágenes bidimensionales, debe ser la misma para todos los procesadores que intervienen en la ejecución proceso. Al considerar la división de la escena bidimensional original, y que cada procesador puede ejecutar las operaciones de histograma, clasificación, opacidad e integración, en la primera etapa del proceso se generarían histogramas que sólo correspondería a la información contenida en la sección de la imagen bidimensional. Emplear, estos histogramas en la obtención de las tablas de opacidades, proporcionaría como resultado el tener opacidades diferentes para un mismo valor de voxel, y la opacidad dependería de donde se encuentra localizado el voxel. El resultado esperado sería una imagen que presentaría contrastes entres las secciones en la cual fue dividida.

B) Si se divide el conjunto de imágenes entonces a cada procesador le corresponde un número "n" de esas imágenes bidimensionales, y pueda procesar a las mismas ejecutando las operaciones histograma, clasificación, opacidad e integración. Se presenta la siguiente situación, en la etapa de integración, al integrar los valores de los voxels [voxel(i,j,k)], es condición para cada localidad de la imagen resultado, disponer de todos los valores de los voxels y opacidades a lo largo del conjunto de las K imágenes. El funcionamiento de esta etapa, sería una operación tipo "pipeline" en donde los procesadores esperarían los resultados del procesador antecesor. Con ello, se presupone, como resultado final, un incremento en el tiempo de procesamiento.

En base a las consideraciones arriba mencionadas, se propone la siguiente estrategia de paralelización:

Primero, la información del conjunto de imágenes se distribuirá a cada uno de los "transputers", y ejecutarán cada uno las siguientes operaciones: histograma y clasificación. Esto significa que una vez que todos los procesadores tengan almacenada el conjunto de imágenes en sus respectivas memorias, obtendrán la información del histograma y lo aplicarán en el proceso de segmentación para cada una de las imágenes. Con la información obtenida de la segmentación se designa la tabla de opacidades para cada valor de voxel de la imagen.

Segundo, en la siguiente tarea se plantea realizar una paralalización de datos, esto es, cada "transputer" ejecutará las operaciones integración sólo a una área o sección del conjunto de imágenes bidimensionales, y los resultados serán enviados al "transputer raíz" que sirve de enlace entre la plataforma de trabajo y la estación de trabajo.

Una vez planteada la paralelización de datos, si se tiene un conjunto de K imágenes de tamaño IxJ para ser evaluado el algoritmo "volume rendering" y considerando configuraciones de 1, 2, 4, 8 y 16 procesadores. Resultará que cada procesador tendrá almacenado el conjunto de las K imágenes, el cual aplicará las operaciones histograma, segmentación y opacidad; y para la de integración, el procesador sólo aplicará a una sección de conjunto de imágenes. En este caso si se cuenta con imágenes de I columnas por J renglones, la división por secciones se hará de acuerdo al número de columnas, por lo que se presenta la siguiente tabla:

Tamaño de la Sección A Procesar	Número de Transputers
IxJ	1
(I/2)xJ	2
(I/4)xJ	4
(I/8)xJ	8
(I/16)xJ	16

Tabla 5.1 División de acuerdo a las columnas, según el número de procesadores.

Finalmente, para evaluar el desempeño de los procesadores, se proponen los siguientes arreglos de procesadores:

I) La distribución del trabajo equitativamente, estableciendo para ello una comunicación de manera eficiente entre 1, 2, 4, 8 y 16 procesadores, además de considerar que los procesadores tiene cuatro ligas de comunicación. En el caso de 16 procesadores (ver figura 5.2), la distribución de los procesadores es en forma de "árbol", siendo la "raíz" el procesador que establece la comunicación con el "host"[19], y para los diferentes niveles se emplean las cuatro ligas de comunicación de cada procesador. Con esta configuración se evaluaría del desempeño del procesamiento paralelo, de una manera simple y de acuerdo a los propósitos de este trabajo.

Sin embargo, este arreglo para el caso de 8 procesadores presenta un doble flujo de datos en la distribución entre el procesador (T_1) y el procesador (T_3), ya que este último envía la información a dos procesadores (T_6 y T_8). Esta actividad representa un doble trabajo comparado con el realizado por los procesadores (T_2 y T_4), como se muestra en la figura 5.3.

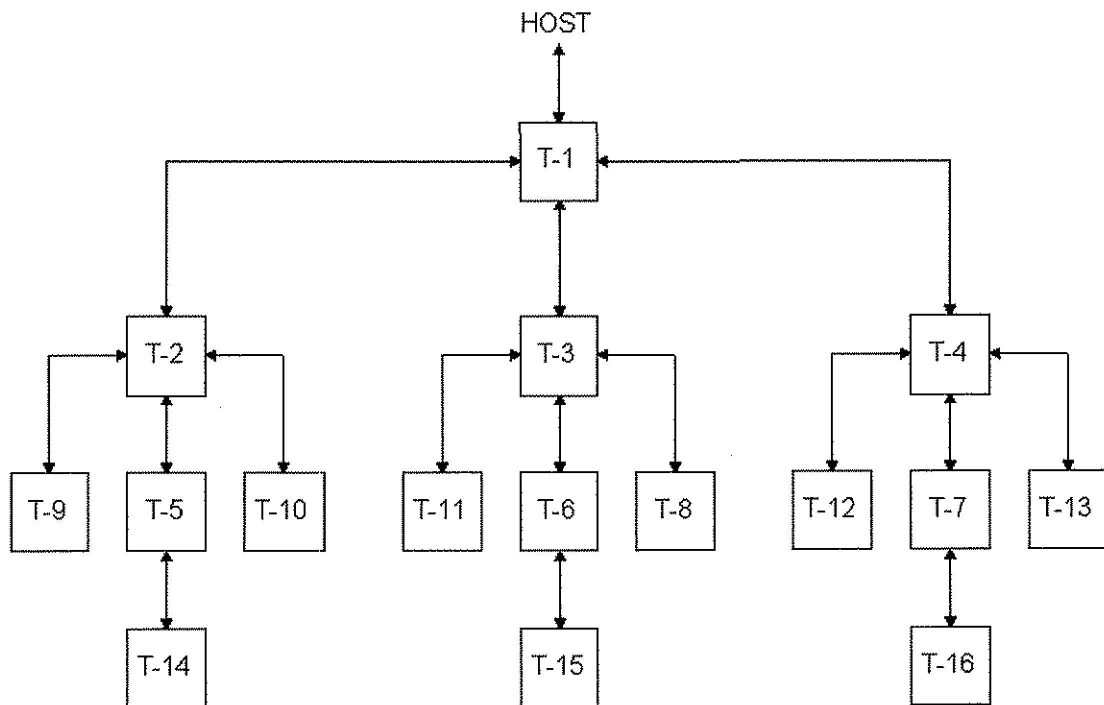


Fig. 5.2 Configuración de los 16 procesadores para evaluar el procesamiento empleando cuatro ligas de comunicación.

[19] "host": máquina organizadora.

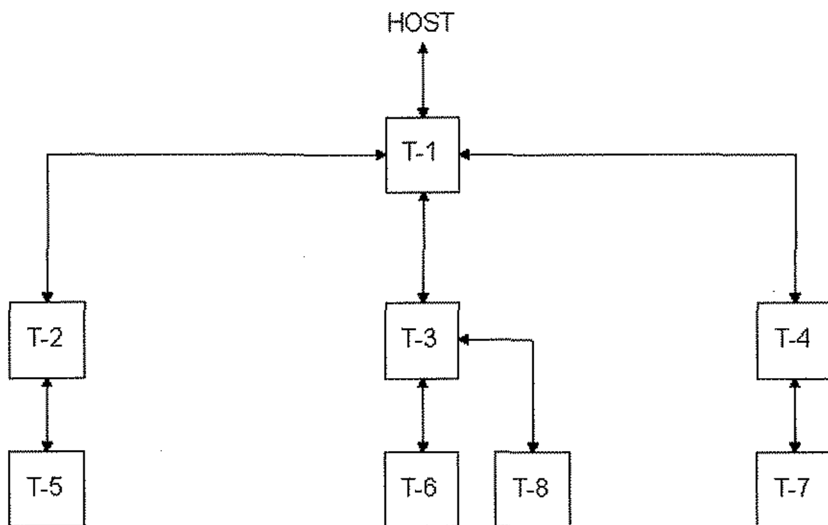


Fig. 5.3
Configuración de los 8 procesadores para evaluar el procesamiento empleando cuatro ligas de comunicación.

II) De acuerdo a las características de las tarjetas de plataforma empleada (ver anexo A), se tiene una limitante, ya que los "transputers" solo pueden utilizar 3 de sus 4 ligas de comunicación, y por esta razón, para el presente trabajo se proponen las siguientes configuraciones de los "transputers".

Para 1, 2 y 4 "transputers", se presentan en la siguiente figura:

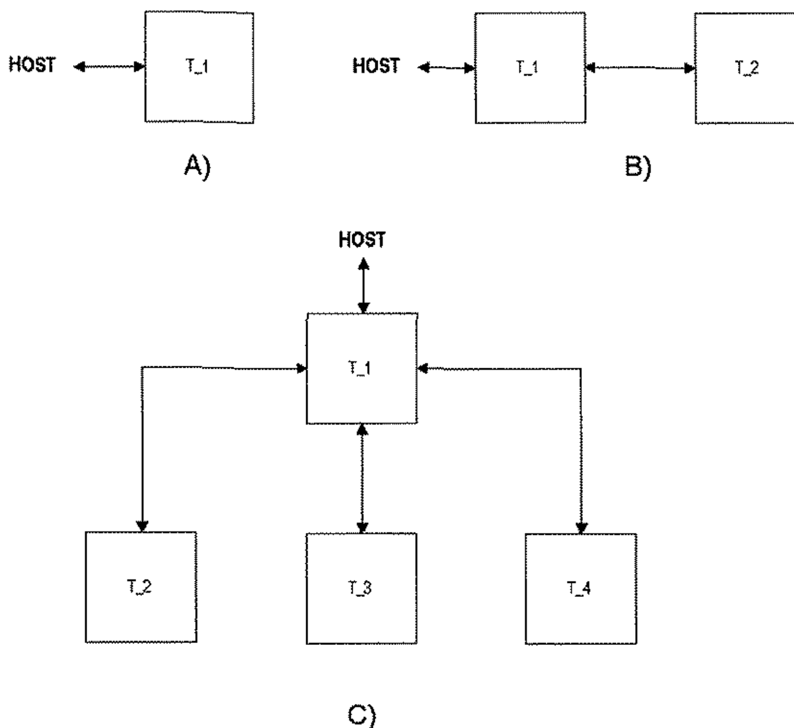


Fig.5.4 Configuración de los transputers para 1, 2 y 4.

Para 8 y 16 “transputers”, se tiene la configuración en las siguientes figuras:

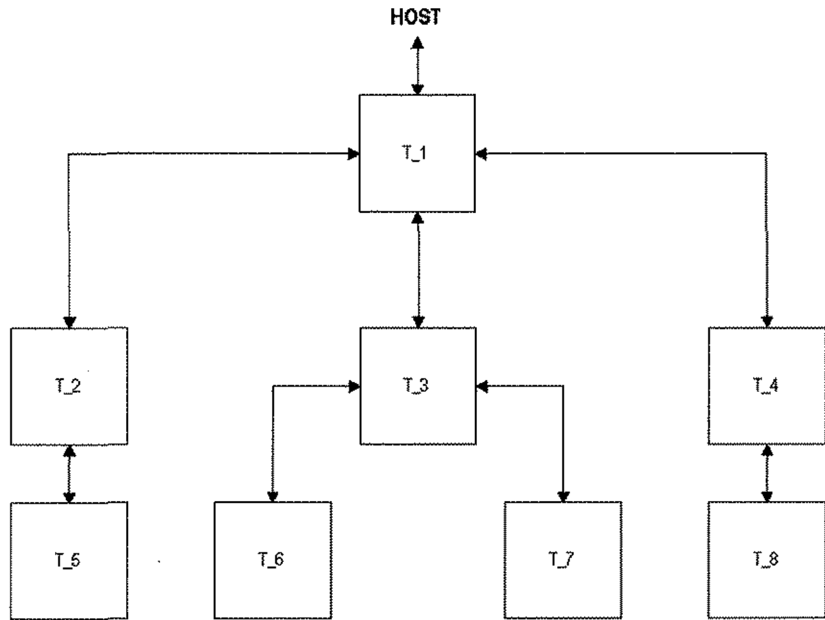


Fig.5.5 Configuración 8 transputers.

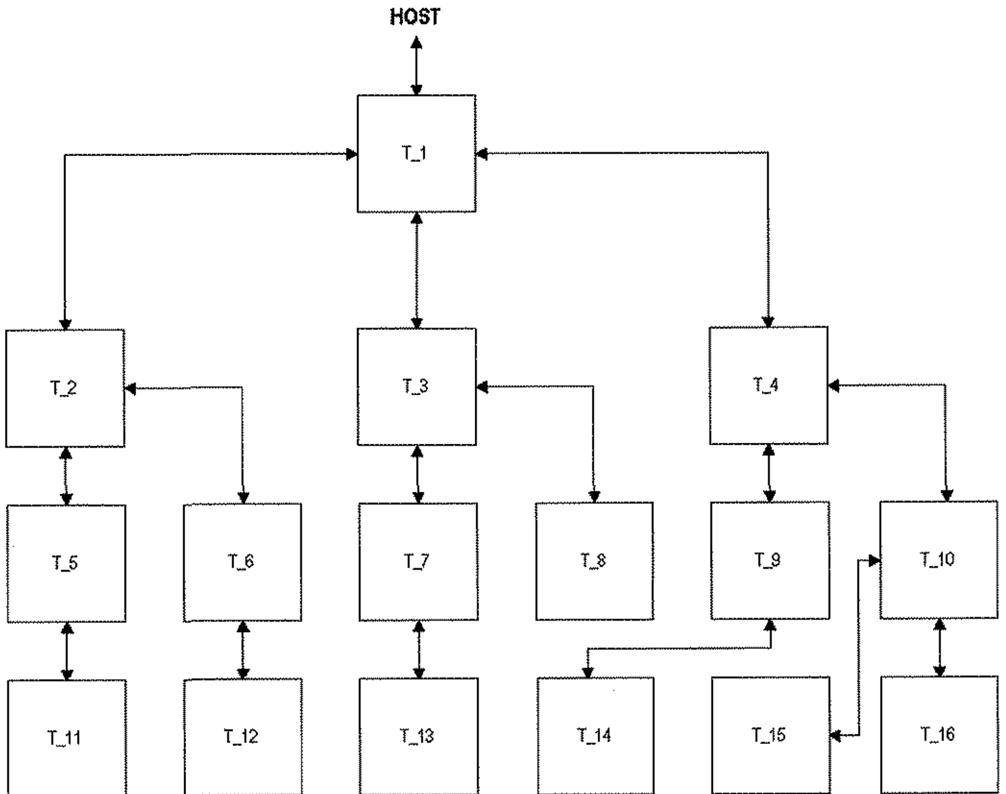


Fig. 5.6 Configuración 16 transputers.

5.4 Implantación del Programa

El programa desarrollado, empleando el lenguaje C ANSI (paralelo), tiene la siguiente estructura:

1) Definición de Variables
2) Lectura del conjunto de imágenes y su almacenamiento en un arreglo cubo(i, j, k).
3) Distribución del cubo a los diferentes procesadores que intervienen en el proceso.
4) Obtención del histograma de la imagen.
5) Segmentación de la imagen.
6) Determinación de la tabla de opacidades opacidad (valor del voxel, k) de cada k imagen del cubo(i, j, k).
7) Ejecución de la integración de los datos.
8) Envío de los resultado de cada procesador al arreglo que contiene la imagen tridimensional.
9) Almacenamiento en un archivo de la imagen resultado.

Con las tareas antes definidas se determinan las tareas a ejecutar en cada "transputer", de la siguiente forma:

a) En los puntos 4, 5 y 6 será al todo el conjunto de imágenes.
b) En el punto 7 solo será a la sección que se le asigne.
c) Enviará los resultados de la sección asignada.

En el Anexo B, se presenta los listados completos de los programas desarrollados para la configuración de 1, 2 , 4, 8 y 16 procesadores.

5.5 Caso de Estudio

La información a procesar es un conjunto de 16 imágenes ultrasónicas de 264 renglones por 512 columnas (figura 5.7). Las imágenes fueron proporcionadas por el Laboratorio de Graficación de la Facultad de Ciencias UNAM, la información corresponde a un huevo de dinosaurio. En las figuras 5.8 y 5.9 se muestran las 16 imágenes empleadas.

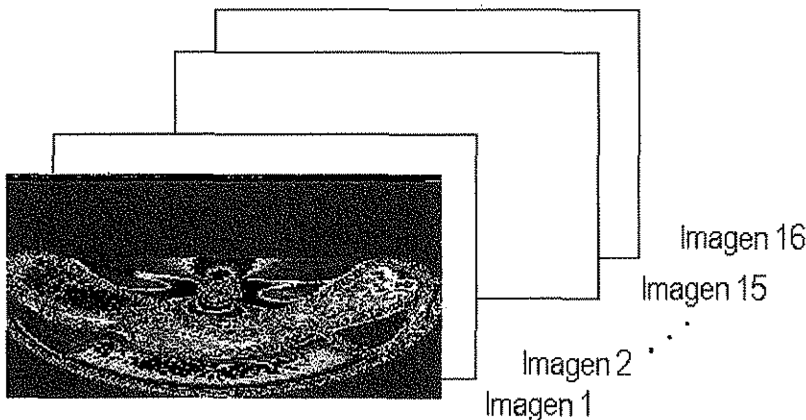


Fig. 5.7 Conjunto de imágenes a emplear en la evaluación del algoritmo "volume redering".

TESIS CON
FALLA DE ORIGEN

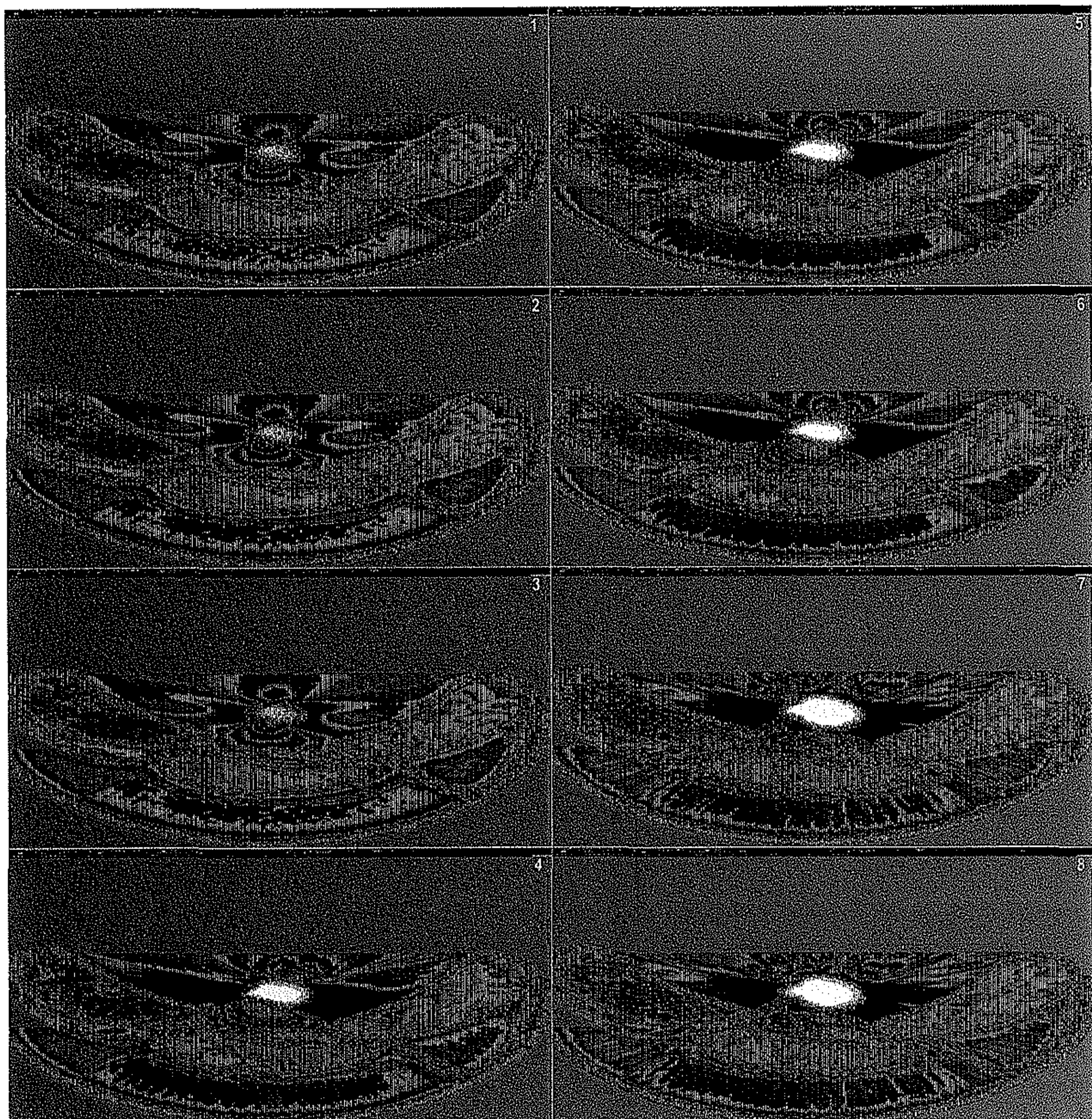


Fig. 5.8 Imágenes 1a la 8 del conjunto.

ESTA TESIS NO SALE
DE LA BIBLIOTECA

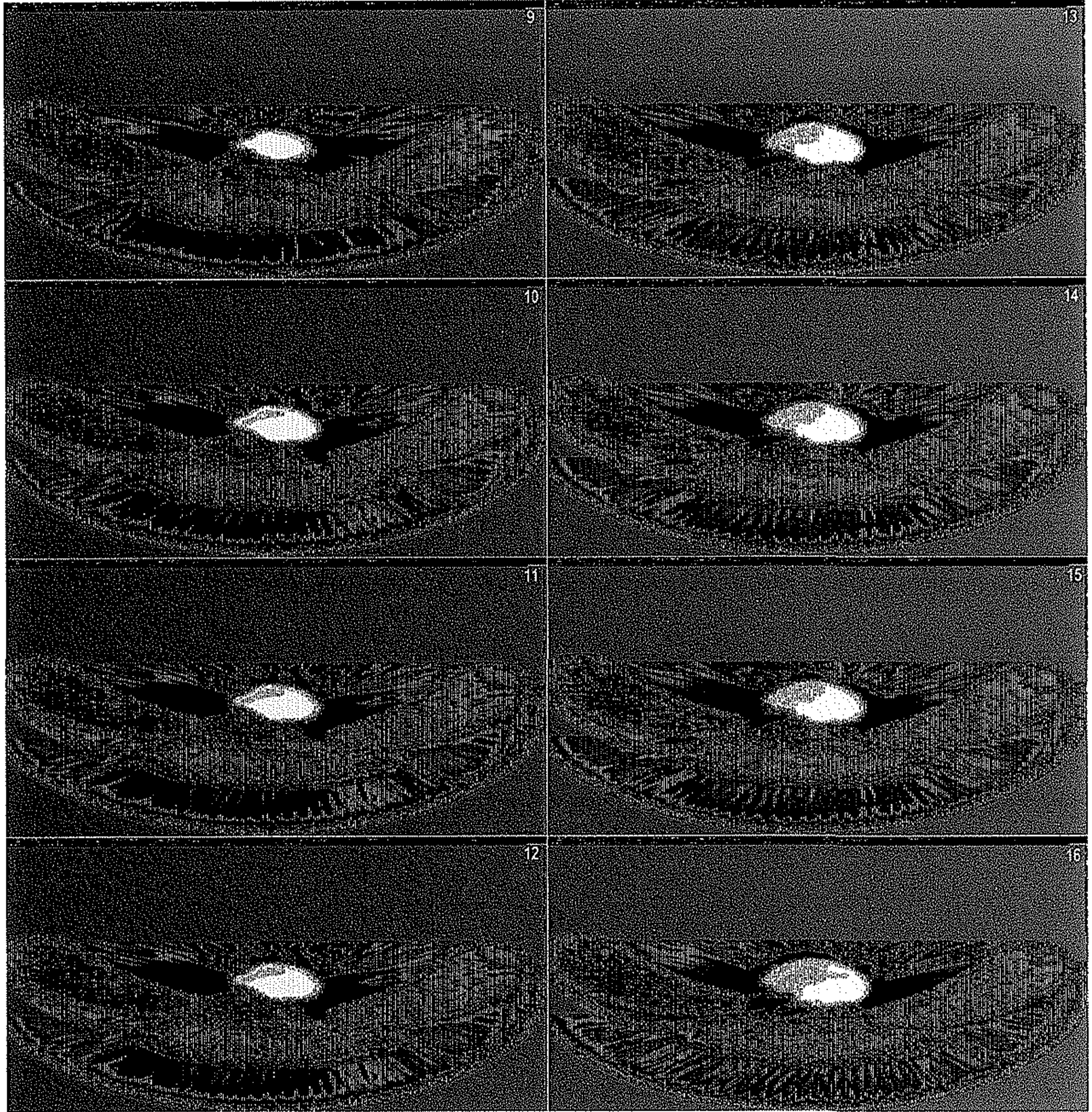


Fig. 5.9 Imágenes 9 a la 16 del conjunto.

TESIS CON
FALLA DE ORIGEN

5.5.1 Organización de imágenes

El conjunto de imágenes se debe organizar de acuerdo a las características planteadas en cada uno de los programas desarrollados, y obviamente con base en las configuraciones propuestas de los "transputers" para el procesamiento en paralelo (paralelización de datos). Para nuestro trabajo se platea el usar 1, 2, 4, 8 y 16 "transputers", y esto se traduce en la división del conjunto de datos. Con la información de las imágenes a procesar y de las configuraciones planteadas, se define la siguiente tabla:

Número de Transputers	Tamaño de la Sección a Procesar en Base al Número de Transputers
1	264x512
2	264x256
4	264x128
8	264x64
16	264x32

Tabla 5.2. Seccionamiento de la información de acuerdo al número de procesadores.

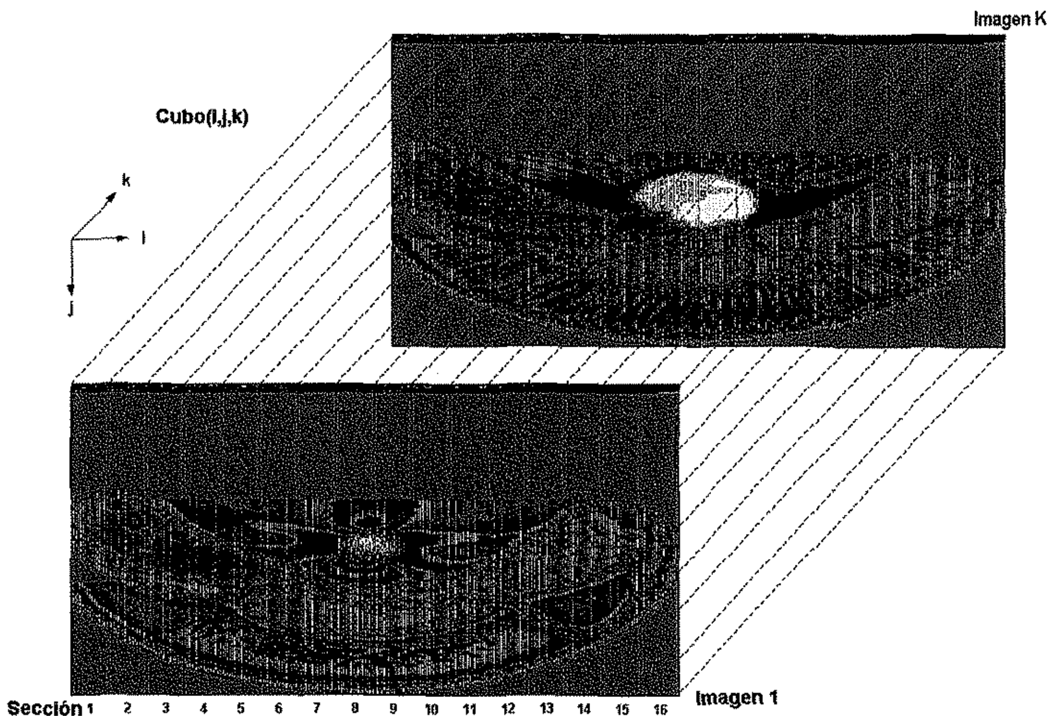


Fig.5.10 Arreglo matricial de las imágenes y división del conjunto de datos.

En la figura 5.10, se muestra la división propuesta del conjunto de imágenes a procesar. Con esta división se obtienen 16 secciones, las cuales se distribuyen a los

"transputers" de acuerdo a la configuración empleada. Se plantea un total de 5 configuraciones, que corresponden al empleo de 1, 2, 4, 8 y 16 "transputers". En el caso de emplear un solo "transputer", éste procesará las 16 secciones, mientras que al usar 16 "transputers", cada uno éstos sólo procesará una sección. En la siguiente tabla se define para cada configuración, cual(es) sección(es) le(s) corresponde(n) procesar a cada "transputer".

Número de Transputer	**Secciones a procesar por cada transputer con base en el número de transputers**				
	1	**2**	**4**	**8**	**16**
T_1	1-16	1-8	1-4	1-2	1
T_2	-	9-16	5-8	3-4	2
T_3	-	-	9-12	5-6	3
T_4	-	-	13-16	7-8	4
T_5	-	-	-	9-10	5
T_6	-	-	-	11-12	6
T_7	-	-	-	13-14	7
T_8	-	-	-	15-16	8
T_9	-	-	-	-	9
T_10	-	-	-	-	10
T_11	-	-	-	-	11
T_12	-	-	-	-	12
T_13	-	-	-	-	13
T_14	-	-	-	-	14
T_15	-	-	-	-	15
T_16	-	-	-	-	16

Tabla 5.3. Asignación de las secciones a procesar, de acuerdo al número de procesadores.

5.5.2 Descripción del ciclo de procesamiento

Una vez revisada la información a procesar, ahora se deben de determinar el ciclo de procesamiento, y cuales etapas nos permitirán evaluar el desempeño del procesamiento. En el programa desarrollado se definen dos etapas principales para implementar el algoritmo "volume rendering"; además de dos tareas que corresponden al almacenamiento del conjunto de imágenes en el arreglo matricial definido como cubo(i,j,k), figura 5.11, y otra a la generación del archivo que contiene la imagen resultado (ver figura 5.15).

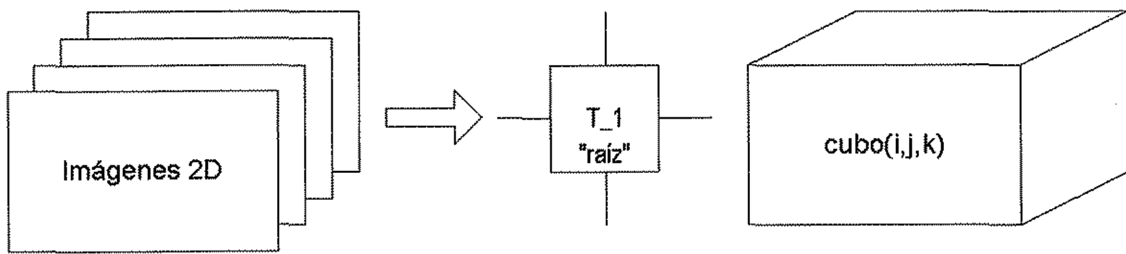


Fig. 5.11 Almacenamiento del conjunto de imágenes en un arreglo matricial cubo(i,j,k).

A continuación se describen las actividades que componen el ciclo de procesamiento.

I) La primera tarea consiste básicamente en distribuir el conjunto de imágenes almacenadas en el arreglo matricial cubo(i,j,k), a cada uno de los "transputers" que intervienen en el proceso (2, 4, 6 y 16 "transputers"), ver figura 5.12. El envío de la información tiene como punto de partida el "transputer" "raíz" (el cual establece la comunicación entre la plataforma de trabajo y la estación de trabajo).

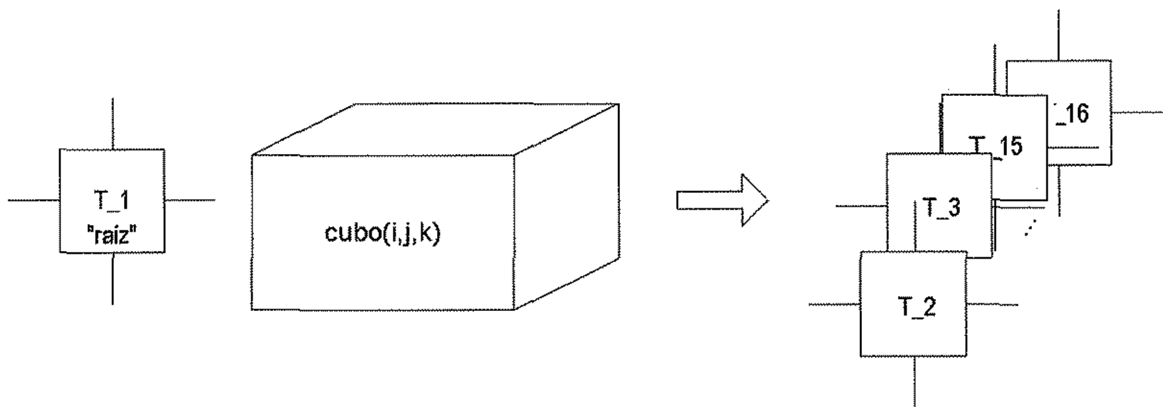


Fig. 5.12 Distribución del arreglo matricial cubo(i,j.k) a los procesadores.

II) Una vez concluida la primera tarea, se ejecutan las operaciones: histograma y clasificación para cada una de las imágenes. Cada procesador que participa en proceso genera primero un histograma, mismo que se requiere en el proceso de segmentación de las imágenes, y con los resultados de estas segmentaciones se está en condiciones de ejecutar la operación de opacidad. Esta operación consiste en la generación de una tabla de opacidades en donde se asigna una opacidad a cada valor de voxel (en el rango de 0 a 256), para cada una de las imágenes del conjunto. Una vez determinadas estas tablas, se integran los datos de la sección que le corresponde a cada procesador (figura 5.13).

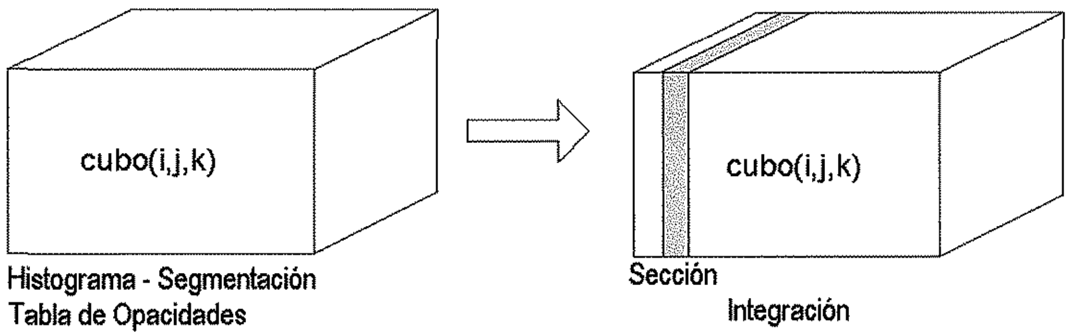


Fig. 5.13 Cada procesador ejecuta las operaciones histograma, segmentación tabla de opacidad a todo arreglo matricial, y la operación integración sólo a su sección asignada.

Finalmente, cada procesador envía el resultado de la integración al procesador "raíz" de la configuración de procesadores, figura 5.14.

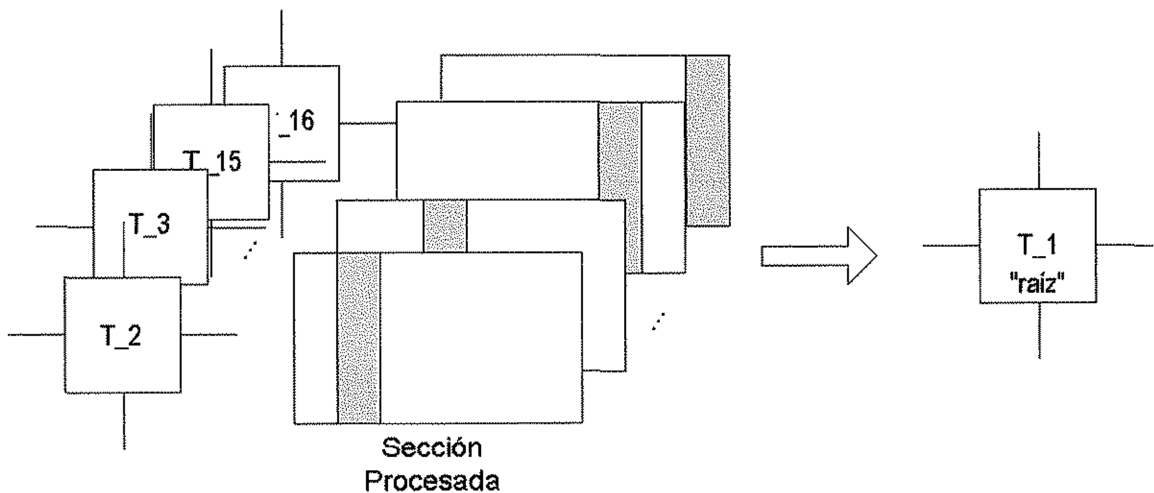


Fig. 5.14 Envío de los resultados al procesador "raíz".

Estas dos etapas, que forman parte del ciclo de procesamiento, involucran actividades que definen la implantación del algoritmo "volume rendering". El tiempo de ejecución de la mismas será nuestro tiempo de ejecución del ciclo de procesamiento, y es el principal parámetro para evaluar el desempeño del sistema propuesto en el presente trabajo.

Como última tarea el "transputer raíz", realiza un proceso mediante el cual genera el archivo que contiene como resultado la imagen tridimensional (figura 5.15).

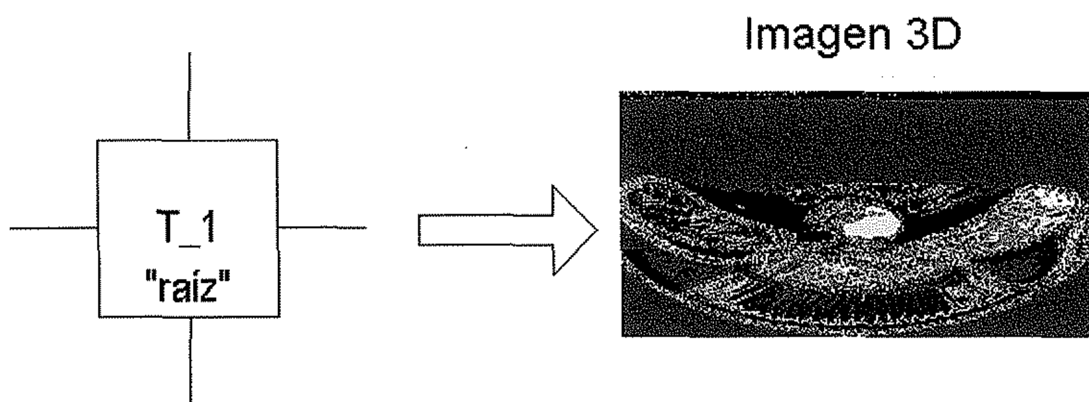


Fig. 5.15 Generación del archivo que contiene la imagen 3D.

5.5 Discusión.

En el desarrollo de este capítulo, primero se hizo un análisis del proceso de visualización empleando el algoritmo "volume rendering", determinando una serie de etapas, en donde cada una de ellas realiza alguna de las tareas que se llevan a cabo durante dicho proceso.

Posteriormente, con esta información se describió la estrategia de paralelización, acorde a las condiciones tanto del proceso (dividiendo la información a procesar) como de los arreglos de procesadores (arquitectura de procesadores con memoria distribuida). Se planteó una estrategia que permite implementar el algoritmo acorde a las etapas del proceso, junto con los diferentes arreglos de procesadores que se pueden configurar en la plataforma de desarrollo.

Una vez definida la estrategia, el paso siguiente fue implementarla en la plataforma de desarrollo disponible (MCP100 [INMOSa, 1992]), así como, definir el lenguaje de programación paralela a emplear, el cual es el lenguaje C ANSI paralelo (que incluye bibliotecas de funciones de cómputo paralelo), proporcionado por la misma plataforma de desarrollo [INMOSb,1992].

En el caso de estudio se dispone de un conjunto de 16 imágenes bidimensionales y se implementaron configuraciones de 1, 2, 4, 8 y 16 procesadores. En cada una de estas configuraciones, a cada procesador se le proporciona el conjunto de imágenes y se le asigna las tareas histograma, segmentación y tablas de opacidades. La tarea más intensiva que corresponde a la integración de los datos es dividida de acuerdo al número de procesadores empleados. Finalmente, los resultados son recolectados y enviados a un archivo, el cual contiene la imagen tridimensional.

Capítulo 6

Resultados

TESIS CON
FALLA DE ORIGEN

6.1 Introducción

En este capítulo se presentan los resultados de la implementación paralela del algoritmo de visualización "volume rendering" que se describió en el capítulo 5. La figura 6.1 muestra la imagen resultado, obtenida al utilizar este algoritmo tomando como base un conjunto de 16 imágenes bidimensionales. Cabe mencionar que la imagen tridimensional presentada (ver fig. 6.1), es la idéntica a las que se obtienen al ejecutar el algoritmo en un número creciente de procesadores, con diversas configuraciones desde 1 hasta 16 procesadores. También, se presenta un estudio de desempeño de la implementación, partiendo de la medición del tiempo de ejecución del algoritmo en las diferentes configuraciones definidas.

Para obtener el tiempo de ejecución del algoritmo se utilizaron funciones de la biblioteca "time.h", disponible en el sistema de desarrollo del transputer, las cuales nos permiten monitorear dicho tiempo de ejecución, a partir del uso de temporizadores [INMOSb,1992]. Los monitores de tiempo se ubicaron dentro de la rutina del procesador "raíz", y a partir de la distribución inicial de los datos al resto de los procesadores, se toma la referencia inicial. La referencia final se toma cuando el procesador "raíz" concluye la recolección de los resultados. Con las mediciones obtenidas se determinan las siguientes métricas de desempeño: "Speedup", Eficiencia y Fracción Serial, presentándose también un análisis de las mismas.

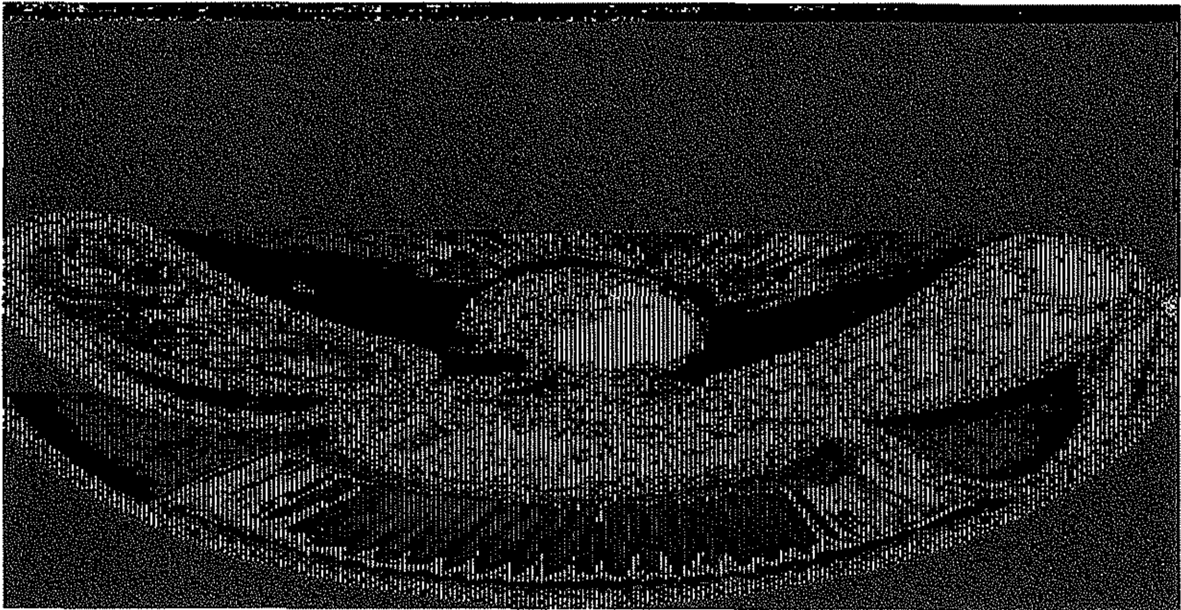


Fig. 6.1 Imagen 3D

6.2 Tiempo de Ejecución para cada Configuración de los Transputers

El tiempo de ejecución, como se revisó en el capítulo anterior, corresponde al tiempo en el cual se procesa el conjunto de 16 imágenes, utilizando para ello la implementación del algoritmo en las diferentes configuraciones propuestas (1, 2, 4, 8 y 16 transputers), para generar una imagen tridimensional.

En la siguiente tabla 6.1, se muestran los resultados obtenidos en donde se indica el número de transputers empleados, el tiempo de ejecución para tres diferentes eventos, y también se calcula el tiempo promedio de ejecución derivado de estos tres eventos. Este tiempo promedio será el empleado en el cálculo de nuestras métricas de desempeño, como son el "Speedup", Eficiencia y Fracción Serial.

Num Transp.	1a. ejecucion	2a. ejecucion	3a. de ejecucion	Promedio (seg)
1	68.311104	68.311168	68.31104	68.311104
2	63.932736	63.932864	63.932608	63.932736
4	54.865664	54.865792	54.865856	54.86577067
8	48.716992	48.717056	48.716992	48.71701333
16	45.993216	45.993664	45.9992	45.99536

Tabla 6.1 Tiempos de ejecución para tres eventos, tiempo promedio de ejecución para cada configuración de transputers empleados.

En la figura 6.2 se muestra la gráfica del tiempo promedio de ejecución de acuerdo al número de procesadores (1, 2, 4, 8 y 16 transputers). Se puede observar que este tiempo tiende a decrementar su valor, presentando un valor máximo de 68.31 segs. (empleando un transputer), y un valor mínimo de 45.99 segs. (empleando 16 transputers). Esto nos indica una reducción de 22.32 segs., que representa el 32.67 % del tiempo empleado por un transputer. Esta observación, no permite hacer una evaluación del desempeño de la implementación, ya que no aporta más que lo observado. Para analizar con más detalle estos resultados, a continuación se determinarán las métricas de desempeño.

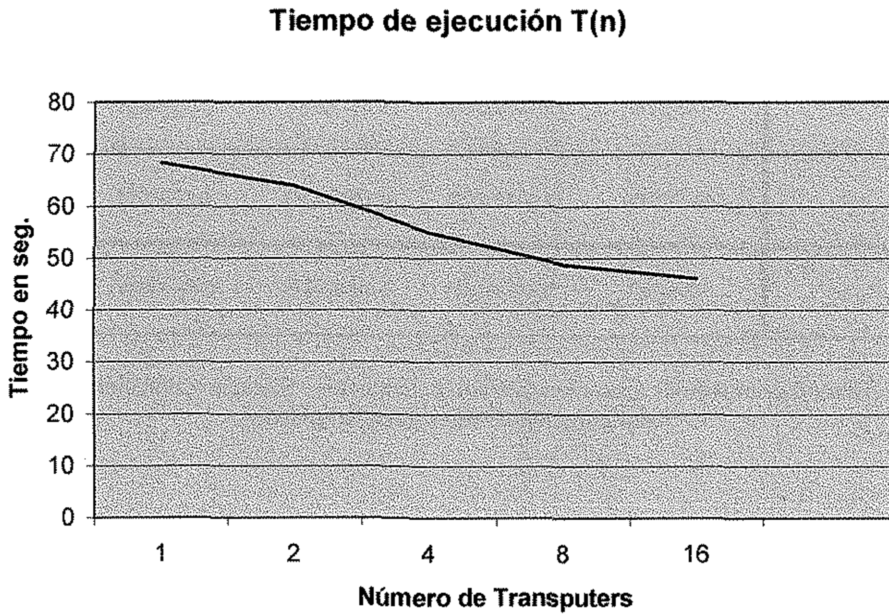


Fig. 6.2 Gráfica del tiempo promedio de ejecución, para 1, 2, 4, 8 y 16 transputers.

6.3 Métricas de Desempeño "SpeedUp", Eficiencia y Fracción serial

En el capítulo 4 se definieron estas métricas, la cuales serán usadas para analizar los resultados. A continuación se presentan las ecuaciones empleadas en el cálculo de las métricas. Los valores obtenidos están en función del tiempo promedio de ejecución para las diferentes configuraciones del sistema implementado.

A)"Speedup":

$$s(n) = T(1)/T(n) \tag{6.1}$$

B) Eficencia:

$$e(n) = s(n)/n \tag{6.2}$$

C) Fracción serial:

$$f(n) = ((1/s) - (1/n)) / (1 - (1/n)) \tag{6.3}$$

Num Transp.	Promedio (seg)	Speedup(s)	Eficiencia(e)	Fracción Serial(f)
1	68.311104	1	1	-
2	63.932736	1.068483977	0.534241988	0.8718109
4	54.8657707	1.245058679	0.31126467	0.73756666
8	48.717013	1.402202225	0.175275278	0.6721873
16	45.9936	1.485230641	0.092826915	0.65151584

Tabla 6.2 Tiempo promedio de ejecución, SpeedUp, Eficiencia, Eficacia y Fracción serial para cada configuración de transputers empleados.

6.4 Análisis de los Resultados

Speedup(s) : El "Speedup" obtenido, como se muestra en la curva de la figura 6.3, presenta valores más bajos de los que se esperarían tener idealmente, esto es indicio que las tareas que ejecutan los procesadores podrían no estar balanceadas o bien se presenta un "overhead" en la comunicación entre los procesos.

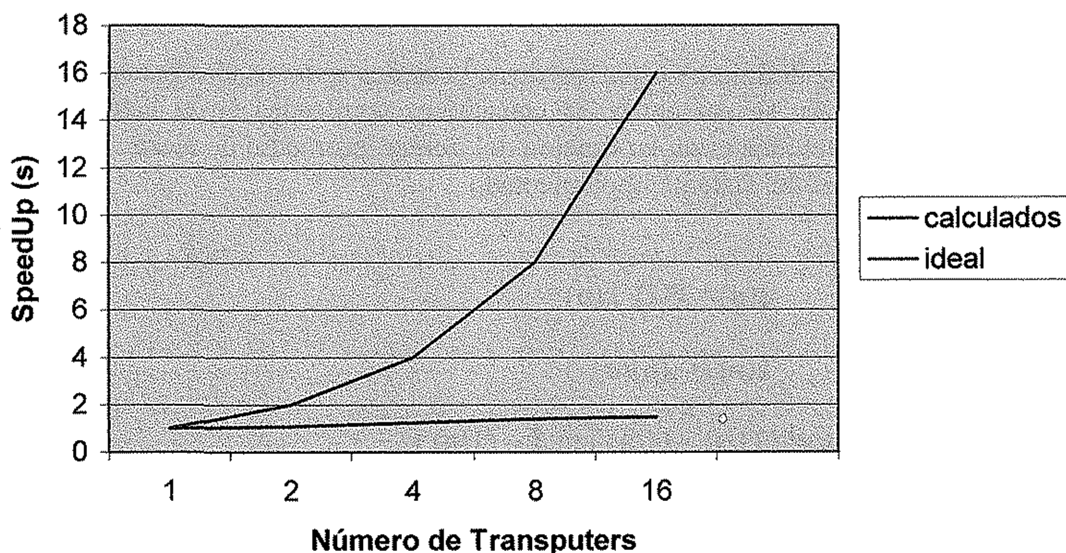
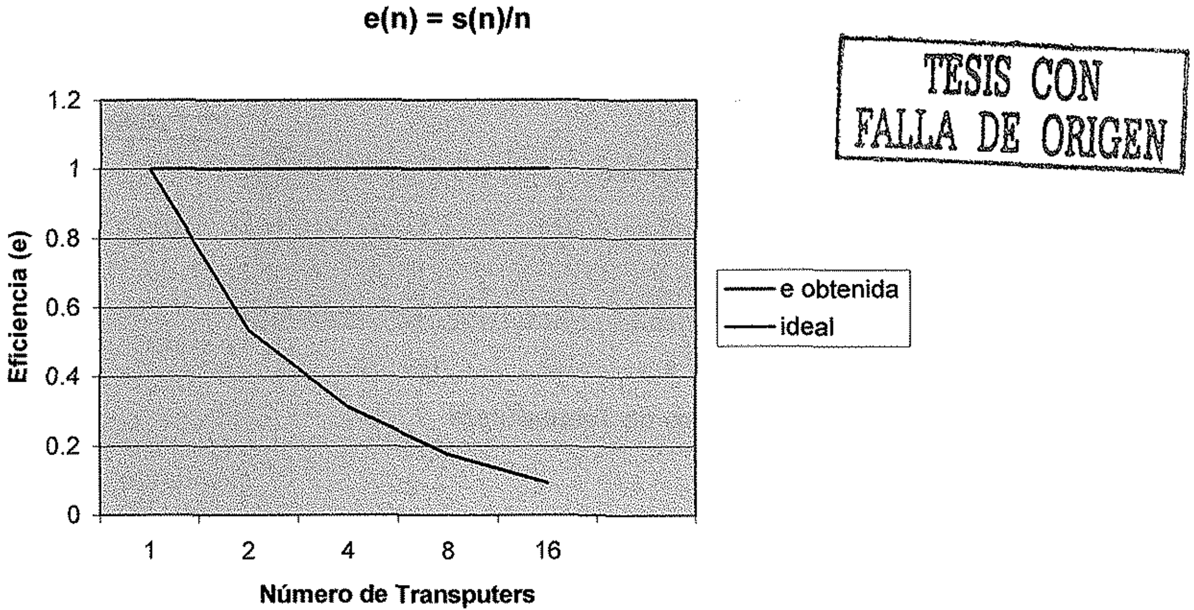


Fig. 6.3 Gráfica del SpeedUp para 1, 2, 4, 8 y 16 transputers

Eficiencia(e): La gráfica mostrada en la figura 6.4, presenta la curva de la eficiencia para las configuraciones de 1, 2, 4, 8 y 16 transputers. En ésta se puede observar que la eficiencia máxima es de 53.42 % (para un transputer), y una mínima de 9.28 % (para 16 transputers). Esto nos indica que nuestra aplicación no está aprovechando de manera eficiente los recursos disponibles en la plataforma de desarrollo.



TESIS CON
FALLA DE ORIGEN

Fig. 6.4 Gráfica de Eficiencia para 1, 2, 4, 8 y 16 transputers.

Fracción serial(f): El comportamiento de la fracción serial es mostrado en la figura 6.5. En la tabla 6.2, se puede observar que para el caso de dos procesadores se tiene un valor alto de f =0.8718, el cual representa por sí mismo un intenso nivel de comunicaciones entre los procesadores. Sin embargo, el valor de f tiende a disminuir al incrementarse el número de procesadores siendo de 0.6515 el valor mínimo para el caso de 16 procesadores. Este valor decreciente de f, que además tiende a estabilizarse en un valor de f cercano al 0.6, nos puede indicar que la paralelización del problema es más eficiente al incrementarse el número de procesadores. Aunque los valores de eficiencia nos indican que no se están aprovechando al máximo los recursos computacionales disponibles.

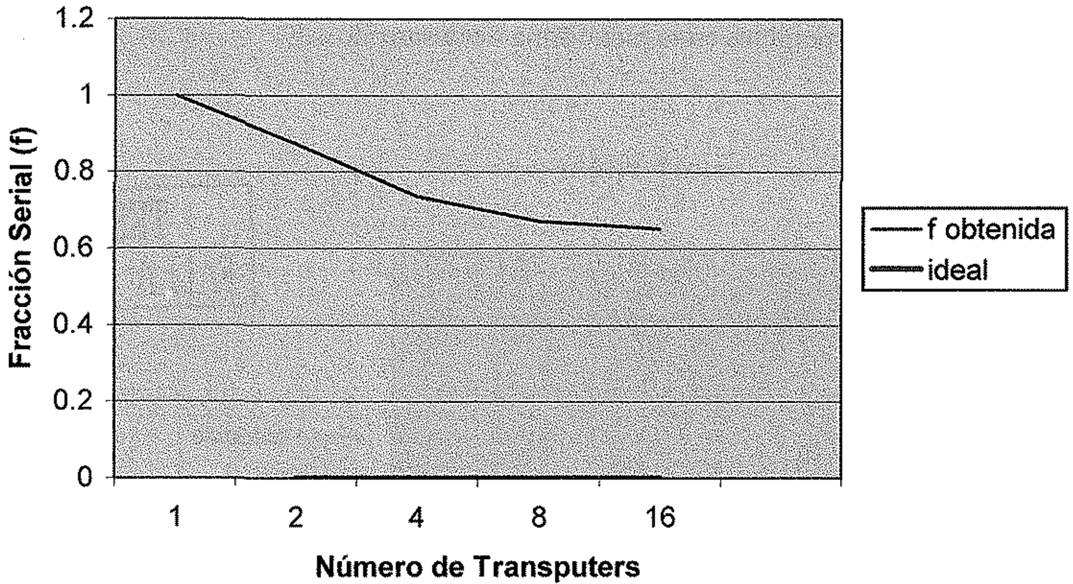


Fig. 6.5 Gráfica de la Fracción serial para 1, 2, 4, 8 y 16 transputers

Neumann define la transferencia entre dos nodos, de un volumen de datos o imagen requerida por un algoritmo paralelo es definida como una redistribución. El costo de esta redistribución es medido como la cantidad de datos transferidos por el tiempo consumido en el traslado sobre la red. En un sistema una parte importante de tiempo dentro del proceso de interpretación es consumido en la reconstrucción y re-muestreo del volumen. En trabajos realizados se muestra que el tiempo requerido para la comunicación se decrementa como los datos y tamaño del sistema se incrementan [Neumann, 1994].

6.5 Discusión

A partir del análisis de los resultados presentados, se puede observar que el desempeño de la plataforma de procesamiento paralelo es modesto. Esto debido a que los procesadores y canales de comunicación utilizados en dicha plataforma son de velocidad reducida.

Sin embargo, el énfasis, que se debe dar al trabajo desarrollado se centra en la estrategia de paralelización utilizada. Cabe hacer mención, que la arquitectura del transputer actualmente presenta limitantes en su desempeño, pero es una plataforma flexible que permite estudiar e implementar estrategias de paralelización de manera formal y segura. Por lo que, ha resultado ser una plataforma adecuada para llevar a cabo desarrollos preliminares de estrategias de paralelización de algoritmos. Estas propuestas pueden ser fácilmente "transportable" a otras plataformas de cómputo paralelo, manteniendo la misma filosofía de diseño.

Capítulo 7

Conclusiones

Para el desarrollo de la esta tesis se plantearon una serie de objetivos, mismos que han sido logrados en sus respectivos niveles.

Se ha realizado un estudio de los diversos métodos de visualización tridimensional, lo que permitió seleccionar una representación de modelación de sólidos basado en la técnica de subdivisión espacial. Dentro de este tipo de técnica se seleccionó el algoritmo de "volume rendering", el cual presenta características eficientes para manejar, generar e interpretar el volumen de datos asociados al conjunto de imágenes bidimensionales que el proceso utiliza como entrada.

Debido a la complejidad computacional del método seleccionado para obtener la visualización tridimensional, se utilizaron técnicas de cómputo paralelo para lograr reducir el tiempo de ejecución, al implementar el algoritmo y distribuir los datos en un número creciente de procesadores, dentro de una arquitectura paralela de memoria distribuida. De esta forma, las técnicas basadas en el paralelismo de datos, han sido utilizadas para reducir los tiempos de ejecución en la obtención de la imagen tridimensional, a partir de un conjunto de datos en la forma de imágenes bidimensionales. En particular se desarrollaron estrategias de paralelización que permiten distribuir el volumen de datos asociados al conjunto de imágenes de manera balanceada en diversas configuraciones de procesadores.

Se llevó a cabo un caso de estudio de desempeño del sistema, monitoreando el tiempo de ejecución del algoritmo al integrar 16 imágenes. Este proceso se implementó en un número creciente de procesadores con diversas configuraciones, desde 1 procesador hasta 16 procesadores, partiendo de los resultados obtenidos, se determinaron diversas métricas de desempeño ("speed-up", eficiencia y fracción serial).

Estos resultados mostraron un desempeño limitado del sistema, debido a un intenso nivel de comunicaciones en las etapas de distribución del volumen de datos y en la integración de la imagen tridimensional. Además, de la velocidad reducida de los procesadores disponibles y sus canales de comunicación. Cabe señalar que, para el caso de estudio se utilizaron solamente 16 imágenes, pero se prevé que al utilizar un mayor número de imágenes, o sea incrementar la granularidad del sistema y con misma velocidad en los canales de comunicaciones, se obtendrá un mayor aprovechamiento de los recurso de la paralelización.

Finalmente, aunque la arquitectura del transputer actualmente tiene limitantes en su desempeño, es una plataforma flexible que permite estudiar e implementar estrategias de paralelización de una manera formal y segura. Por lo tanto, ha resultado una plataforma adecuada para llevar a cabo desarrollos preliminares de estrategias de paralelización de

algoritmos, que pueden ser transportables a otras plataformas paralelas (de mayor velocidad), manteniendo la misma filosofía de diseño.

Trabajo futuro

Durante el desarrollo de la últimas etapas de esta investigación, se detectaron diversos aspectos y en los que se puede desarrollar trabajo futuro.

Uno de los primeros aspectos se enfoca en el uso de nuevas tecnologías de procesamiento para llevar a cabo la implementación de las estrategias desarrolladas en este trabajo, tal es el caso de la arquitectura SHARC de Analog Devices (AD21060-62) conformada por DSPs paralelos. Esta tecnología integra procesadores de alto desempeño junto con canales de comunicación de mayor velocidad, lo cual ofrece una alternativa a la solución del problema de comunicaciones presente en la actual implementación. Y se tendría como objetivo tener un sistema de procesamiento en tiempo real

Para reducir el problema de comunicación entre procesadores, también se plantea el uso de una arquitectura de memoria compartida, en donde todos los procesadores que la conforman tengan acceso al mapa de memoria que alberga los datos de entrada y a la base de datos de la imagen tridimensional resultante del proceso de integración.

Otro aspecto de interés, esta asociado con el estudio del paralelismo intrínseco en las diversas etapas del algoritmo "volume rendering" (histograma, segmentación, opacidad e integración), donde se han detectado procesos independientes que pueden ser implementados de manera paralela y en donde una arquitectura de alto desempeño como la del SHARC puede ser muy eficiente.

Por último, con la disponibilidad de nuevas arquitecturas de mayor desempeño permitirán desarrollar y usar algoritmos computacionales mas intensivos, en aplicaciones de visualización tridimensional en tiempo real.

Bibliografía

Bibliografía

[Almasi, 1989] Almasi G., Gottlieb A., 1989. **Highly Parallel Computing**. The Benjamin/Cummings Publishing Company. U.S.A., 519 p.

[Bravo, 1998] Bravo A.J., 1998. **Tutorial de Procesamiento Digital de Imágenes**. Grupo de Ingeniería Biomédica de la Universidad de Los Andes. (http://www.ing.ula.ve/~abravo/document/tutorial/imagenes/indice.html)

[Castleman, 1979] Castleman K., 1979. **Digital Image Processing**. Prentice-Hall. U.S.A., 429 p.

[Drebin, 1988] Drebin R.A., Carpenter L., Hanrahem P., **Volume Rendering**. Computer Graphics. Volume 22, Number 4. August 1988. U.S.A., pp.65-74.

[Foley, 1996] Foley J.D. 1996. **Introducción a la Graficación por Computador**. Addison-Weslwy. México, 649 p.

[Flynn, 1996] Flynn M.J., Rudd K.W. **Parallel Architectures**. ACM Computer Surveys. Volume 28, Number 1. March 1996. U.S.A., pp. 67-70.

[Hearn, 1995] Hearn D., Baker M.P., 1995. **Gráficas por Computadora**. Prentice-Hall. México, 685 p.

[Hoare, 1978] Hoare C.A.R., **Communicating Sequential Processes**. Communications of the ACM. Volume 21, Number 8, August 1978, U.S.A., pp. 666-677.

[Hockney, 1981] Hockney R.W., 1981. **Parallel Computers**. Adam Hilger Ltd. Great Britain, 423 p.

[INMOSa,1992] INMOS, 1992. **MCP1000 Technical Reference Manual**. INMOS Limited. U.S.A.

[INMOSb,1992] INMOS, 1992. **ANSI C Language and Libraries Reference Manual**. INMOS Limited. U.S.A.

[INMOSc,1992] INMOS, 1992. **ANSI C Toolset User Guide**. INMOS Limited. U.S.A.

[INMOSd,1992] INMOS, 1992. **ANSI C Toolset Reference Manual**. INMOS Limited. U.S.A.

[INMOSe,1992] INMOS, 1992. **MCP1000 Technical Reference Manual**. INMOS Limited. U.S.A.

[Karp, 1990] Karp A.H., Flatt H.P., **Measuring Parallel Processor Performance.** Communications of the ACM. Volume 33, Number 5, May, 1990. U.S.A., pp. 539-543.

[Leighton, 1992] Leighton, T.F., 1992. **Introduction to Parallel Algorithms and Architectures.** Morgan Kaufmann Publishers. U.S.A., 446 p.

[Lester, 1993] Lester B.P., 1993. **The Art of Parallel Programming.** Prentice-Hall. U.S.A., 375 p.

[McCormick, 1987] McCormick B.H., DeFanti T.A., Brown M.D., **Visualization on Scientific Computing.** Computer Graphics. Volume 21, Number 6. November 1987. U.S.A., pp.1-14.

[Meiβner, 2000] Meiβner M., Huang J., Bartz D., Mueller K., Crawfis R., **A Practical Evalution of Popular Volume Rendering Algorithms.** ACM Symposium on Volume Visualization 2000. ACM Press. U.S.A., pp. 81-90.

[Meinzer, 1991] Meinzer H.P., Meetz K., Scheppelmann D., Engelmann U., Baur H. J., **The Heidelberg Ray Tracing Model.** IEEE Computer Graphics & Applications. Volume 11, Number 6. November, 1991. U.S.A., pp. 34-43.

[Nelson, 1993] Nelson T.R., Elvins T.T., **Visualization of 3D Ultrasound** Data. IEEE Computer Graphics & Applications. Volume 13. Number 6. November. 1993. U.S.A., pp. 50-57.

[Neumann, 1994] Neumann U., **Communication Costs for Parallel Volume-Rendering Algorithms.** IEEE Computer Graphics and Applications. Colume 14, Number 4. July, 1994. U.S.A., pp. 49-58.

[Ney, 1990] Ney D.R., Fishman E.K., Magid D., Drebin R.A., **Volumetric Rendering, Volumetric Rendering of Computed Tomography Data: Principles and Techniques.** IEEE Computer Graphics & Applications. Volume 10, Number 2. March, 1990. U.S.A., pp. 24-32.

[Ney, 1992] Ney D.R., Fishman E., **3D Visualization in Medicine.** SIGGRAPH 1992, 19 th International Conference on Computer Graphics and Interactive Techniques. Volume 181, Number 2. November, 1992. U.S.A., pp. 16-18.

[Offen, 1985] Offen R.J., 1985. **VLSI Image Processing.** Collins. Great Britain, 326 p.

[Otsu, 1979] Otsu N., **A Threshold Selecction Meted from Gray-Level Histograms.** IEEE Transactions on System, Man, and Cybernetics, Vol. 9, January 1979. U.S.A., pp. 62-67.

[Overington, 1992] Overington I., 1992. **Computer Vision.** Elsevier. Nederland, 423 p.

[Pitas, 1993] Pitas I., 1993. **Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks.** Jonh Wiley & Sons. Great Britain, 395 p.

[Quinn, 1994] Quinn M.J., 1994. **Parallel Computing: Theory and Practice**. McGraw-Hill. Singapore, 446 p.

[Russ,1990] Russ J., 1990. **The Image Processing Handbook**. CRC Press. U.S.A.

[Sakas, 1995] Sakas G., Schreyer L., Grimm M., **Preprocessing and Volume Rendering of 3D Ultrasonic Data**. IEEE Computer Graphics & Applications. Volume 15, Number 4. July, 1995. pp.47-54.

[Thiébaut,1995] Thiébaut D., 1995. **Parallel Programming in C for the Transputers.** (http://www.cs.smith.edu/~thiebaut/transputer/descript.html)

[TUDelft, 2000] Laboratory Control. 2000. **Transputer System Manual**. Faculty of Information Technology & Systems, Delft University. (http://dutera.et.tudelft.nl/index_old.html).

[Udupa, 1991] Udupa J. K., Hernan G., 1991. **3D Imaging in Medicine**. CRC Press. U.S.A., 347 p.

[Watt, 1992] Watt A., 1992. **Advanced Animation and Rendering Techniques. Theory and Practice.** Addisson-Wesley. Great Britain, 455 p.

[Watt, 1993] Watt A., 1993. **3D Computer Graphics**. Addison-Wesley. Great Britain, 500 p.

[Webber, 1992] Webber H.C., 1992. **Image Processing and Transputers**. IOS Press. Nederland, 186 p.

Anexos

Anexo A

A.1 Características del transputer

Los transputers son microprocesadores de alto desempeño que soportan procesamiento paralelo a través del hardware interno del mismo (on-chip). Estos transputers pueden ser conectados por medio de sus ligas de comunicación seriales (links) dentro de un modo específico de aplicación, y con esto pueden construirse configuraciones de sistemas complejos de procesamiento paralelo (ver figura a.1). Además, se pueden construir de manera muy simple sistemas multi-transputers, ya que el transputer cuenta con 4 ligas de comunicación de alta velocidad para ser conectadas a otros en configuraciones diversas (arreglos, árboles, etc). Todo el manejo del circuito de las ligas esta dentro del mismo transputer y solamente dos cables son necesarios para conectar dos transputers a un mismo tiempo. En suma, el equipamiento de una comunicación y una ruta sincronizada entre procesadores de las ligas del transputer, permiten que la memoria sea examinada directamente por programas de depuración y permite que los programas sean cargados dentro de las redes de transputers a través de la liga del transputer. Cada transputer tiene un controlador de tiempos ejecución altamente eficiente para la ejecución de procesos en paralelo dentro del mismo transputer y soporta el canal de comunicación a través de simples localidades de memoria en memoria. Los procesamientos esperan para la entrada o salida, o esperan sobre un temporizador, no consume recursos del CPU, y en el contexto de los intercambios en los procesos el tiempo puede se tan pequeño como un microsegundo. Las ligas comunicación entre los procesadores operan concurrentemente con la unidad de procesamiento y puede transferir datos simultáneamente en todos las ligas sin la intervención del CPU [TUDelft, 2000].

El diseño de una parte del procesador es muy interesante. No dispone de registros de datos, pero si registros de "stack", los cuales permiten una selección implícita de los registros. El resultado es un formato de instrucción mas pequeño. El transputer adopta la filosofía RISC y soporta un pequeño conjunto de instrucciones ejecutadas cada una en unos pocos ciclos. El microcódigo soporta multitareas. Las acciones necesarias para que el transputer cambie de una tarea a otra son ejecutadas a nivel de hardware, liberando al programador del sistema de esta tarea, y el resultado en la rapidez del cambio de operaciones. Estas características están basadas en el INMOS T805, procesador de 32-bits, emplean un procesador de punto flotante (on-chip) para el funcionamiento de la aritmética REAL [Thiébaut,1995].

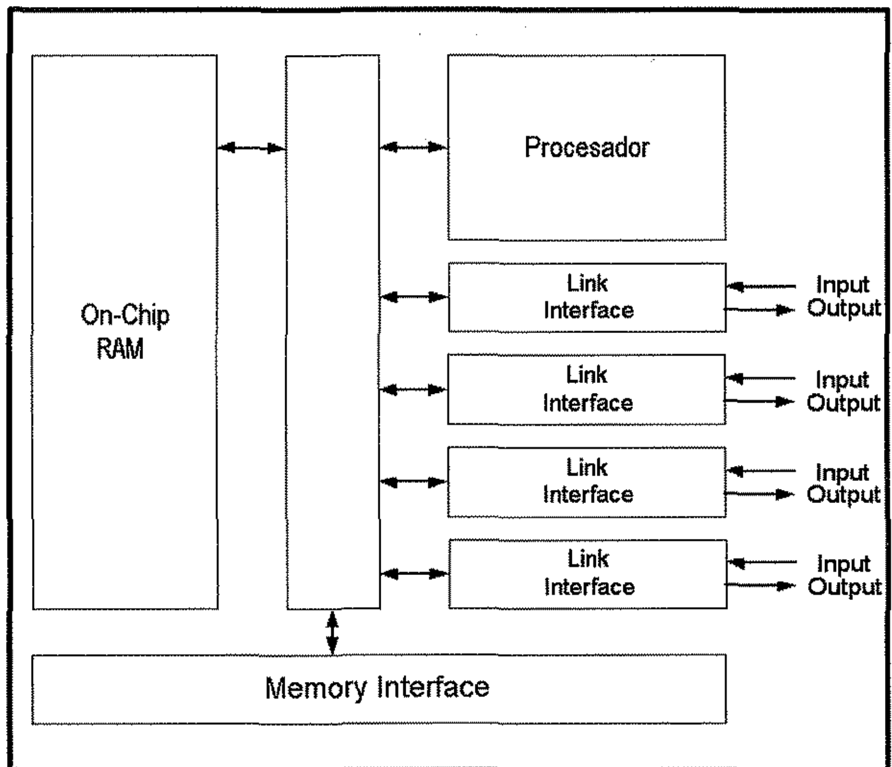


Fig.a.1 Diagrama a bloques del transputer.

A.2 Características Tarjeta MCP100

La tarjeta MCP1000 fue diseñada específicamente para proveer las siguientes características, ver figura a.3:

En cada nodo se puede integrar un rango variado de módulos de procesamiento, que incluyen a los módulos de transputers TRAM de Transtech y de otros proveedores tales como INMOS Corp. En el caso de los TRAMS, se tienen transputers IMST805 INMOS a 30 Mhz., de 32 bits, 8 Mbytes de RAM, y como mínimo 4 ligas de comunicación serial full-duplex de 1.7 Mbytes/s [INMOSe,1992].

La intercomunicación a las estaciones de trabajo (Sun 3 y Sun 4) esta basada en VME. Proporciona manejadores y una interfaz suficientemente inteligente para asegurar un alto "troughputs" con un mínimo costo de carga, sin una sincronización fina por parte de los usuarios.

Esta es una plataforma altamente configurable para una variedad de posibilidades de servicios en la ejecución de programas. La mayoría de los parámetros de configuración son accesibles bajo el software controlador.

En el modo multi-usuario se habilitan una o más tarjetas para ser compartidas entre muchos usuarios en una red. Cada plataforma puede ser simultáneamente usada por arriba de 4 usuarios concurrentemente, o los recursos combinados bajo el software controlador. Para formar una enorme máquina, se tiene la facilidad de poder ser extendida hasta con 8 tarjetas, y da la posibilidad de tener 32 usuarios independientes.

El hardware se escala, para que los usuarios puedan beneficiarse con una nueva tecnología sin sacrificar su inversión original. Transtech esta comprometida a preservar independiente el software de los mismos procesadores, lenguajes y otros hardware o software de desarrollo.

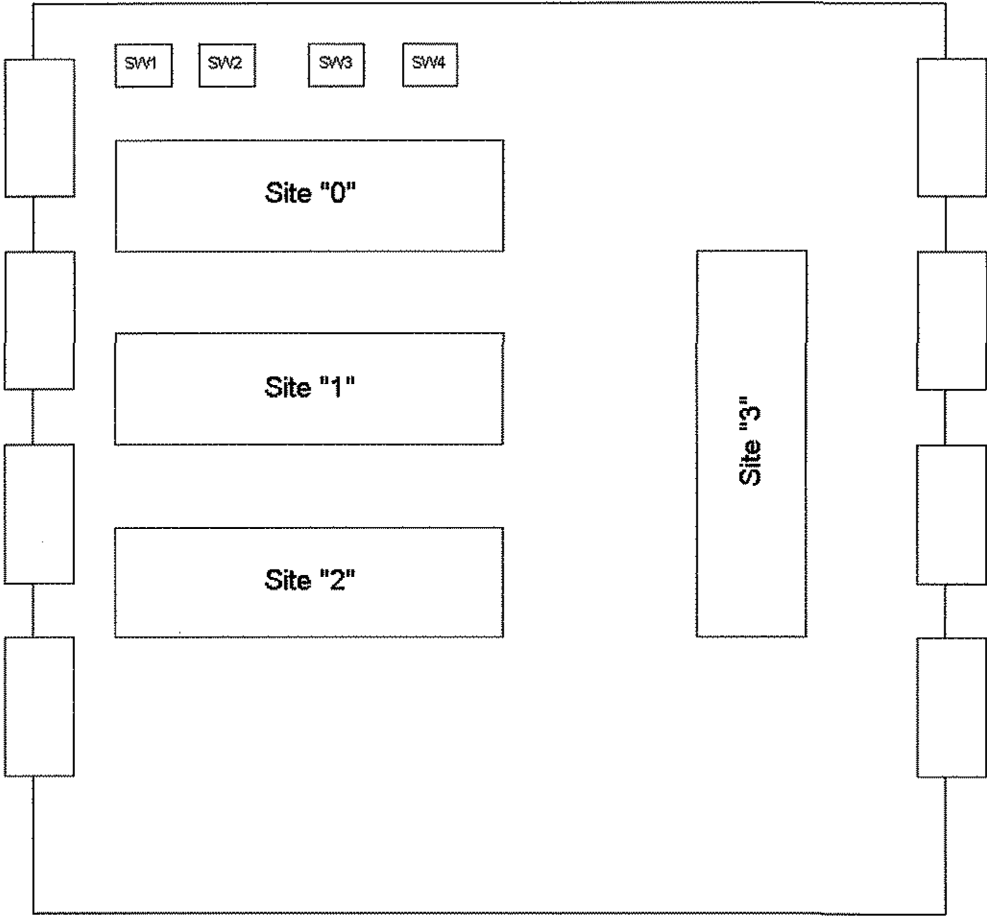


Fig.a.2 Diagrama a bloques de los componentes internos de la tarjeta MCP1000. Los principales dispositivos configurables por el usuario están definidos con sus ID.

Los nodos procesadores son agrupados en 4 "sites", cada uno contiene 8 módulos en "slots". Cada "site" es completamente semi-contenido y abarca:

Los servicios del "site" (reset, análisis, y señales de error)
Liga directa del MCP1000 con VME interface.
Arreglo de interruptores con una interfaz de control exclusiva (figura a.3).
Conector tipo D de 32 vías.

Cada "site" puede ser usado por procesos separados desde Unix en la red de Sun. Alternativamente 2 o más "sites" pueden ser unidos en un mismo tiempo para formar una simple unidad de alta capacidad de procesamiento [INMOSa,1992].

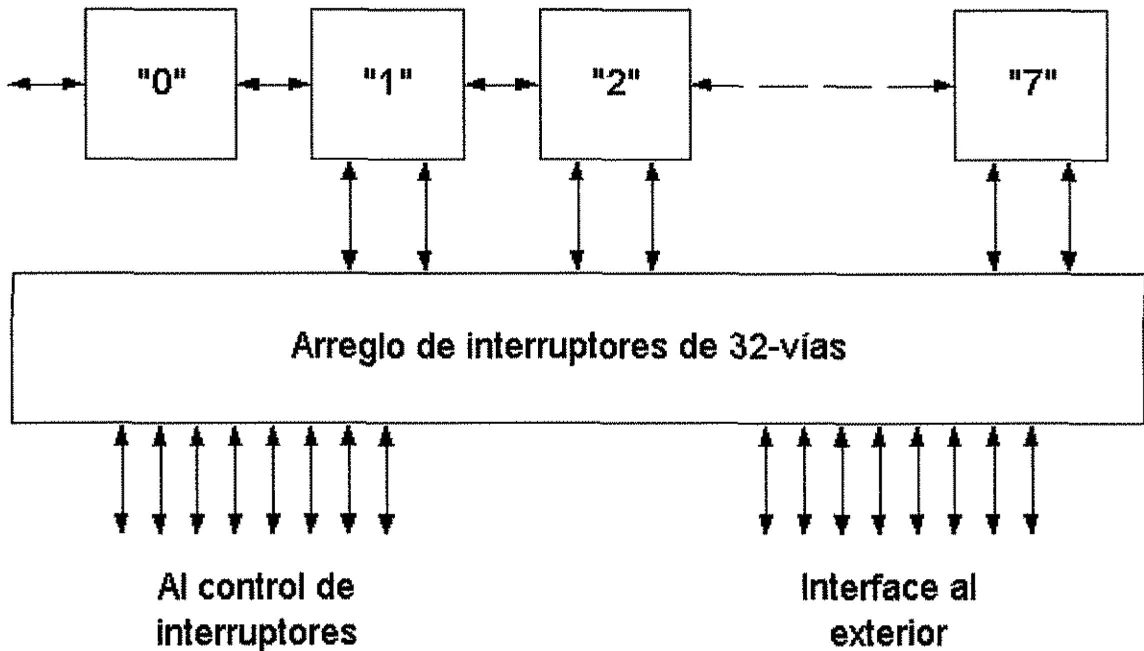
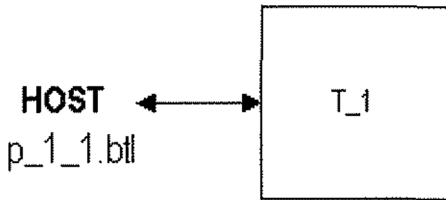


Fig.a.3 Diagrama a bloques de arreglo de interruptores empleado en la tarjeta MCP1000.

Anexo B

B. Programas

B.1 Programa para un solo transputer

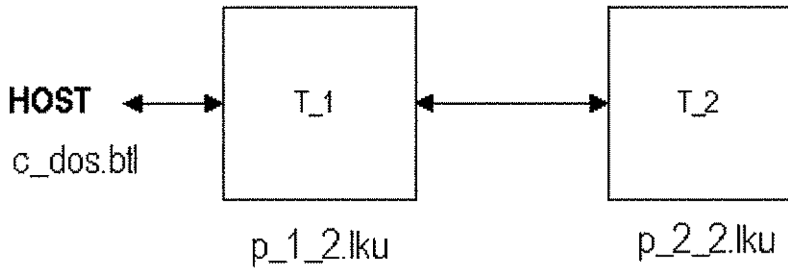



```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* programa de principal */
int main()
{
 /*Designación del archivo resultado */
  if ( (imagen3dfd = open("i_r1.98.1", O_WRONLY |
O_TRUNC )) < 0)
    printf("Error al  Crear Archivo");
/* Proceso de almacenaje del conjunto de imágenes en
e arreglo matricial */
 for(k=0;k<16;k++)
  {
   if (k<9)
    {
     nombre[4]=numero[k+1];
    }
   if ( (k>8)&(k<25) )
    {
     aux=(k+1)/10;
     nombre[3]=numero[aux];
     nombre[4]=numero[k-9-((aux-1)*10)];
    }
   if ( ( imagenfd = open(nombre, O_RDONLY) ) < 0)
    {
     printf(" Error abriendo archivo");
    }
   p_linea=linea;
   for(j=0;j<264;j++)
    {
     m = read(imagenfd, p_linea, sizeof(linea));
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
   close(imagenfd);
  }
 printf("Imagenes cargadas \n");
 /*Inicio del procesamiento del conjunto de imágenes */
 hora1=ProcTime();
 for(k=0;k<16;k++)
  {
   /* Etapa Histograma  */
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
 /* Etapa Segmentación */
   otsu();
  /* Etapa Opacidad */
   opaco();
  }
  /* Etapa de Integración */
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       Cin=0;
       for(k=0;k<16;k++)
        {
         if (cubo[j][i][k]>0)
          {
           aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
           fval1=(float)(aux1*0.01);
           aux1=(unsigned int)(Cin*fval1);
           aux=(char)aux1;
           aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
           fval1=(float)(aux1*0.01);
           aux1=(unsigned int)(fval1);
           Cout=(char)(aux1+aux);
```

```
        } else Cout=0;
       Cin=Cout;
       }
      ima_resul[j][i]=Cout;
    }
  }
hora2=ProcTime();
tiempo1=ProcTimeMinus(hora2,hora1);
printf(" \n numero de ciclos en procesar:  %d
",tiempo1);
/* Almacenamiento de la imagen resultado */
 p_linea=&ima_resul[0][0];
 for(j=0;j<264;j++)
   {
     if ( (write(imagen3dfd, p_linea,m)) != m)
perror("ERROR al escribir");
     p_linea=p_linea+512;
   }
 close(imagen3dfd);
 printf(" Termino \n ");
}
```

B.2 Programas para dos transputers



B.2.1 PROGRAMA DE CONFIGURACIÓN (c_dos8.cfs)

```
T805 (memory=8M) T_1;
T805 (memory=8M) T_2;
connect host to T_1.link[1];
connect root.link[2] to T_2.link[1];
process(stacksize=10K, heapsize=100K,
      interface(input host_in, output host_out,
      input data_in, output data_out)
     ) raiz_1;
process(stacksize=10K, heapsize=100k,
           interface(input up_in, output up_out)
     ) nivel_1_2;
input HostInput;
output HostOutput;
connect raiz_1.host_in to HostInput;
connect raiz_1.host_out to HostOutput;
connect raiz_1.data_in to nivel_1_2.up_out;
connect raiz_1.data_out to nivel_1_2.up_in;
use "p_1_2.8.lku" for raiz_1;
use "p_2_2.8.lku" for nivel_1_2;
place controller on T_1;
place subordinate on T_2;
place HostInput on host;
place HostOutput on host;
```

B.2.2 PROGRAMA TRANSPUTER T_1 (p_1_2.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
```

```
/* PROGRAMA PRINCIPAL */
int main()
{
Channel* in_chan = (Channel*) get_param(3);
Channel* out_chan = (Channel*) get_param(4);
if ( (imagen3dfd = open("i_r2.8.0", O_WRONLY |
O_TRUNC )) < 0)
    printf("Error al  Crear Archivo");
bandera=12;
p_linea=linea;
for(k=0;k<16;k++)
 {
  if (k<9)
   {
    nombre[4]=numero[k+1];
   }
  if ( (k>8)&(k<25) )
   {
    aux=(k+1)/10;
    nombre[3]=numero[aux];
    nombre[4]=numero[k-9-((aux-1)*10)];
   }
  if ( ( imagenfd = open(nombre, O_RDONLY) ) < 0)
   {
    printf(" Error abriendo archivo");
   }
  for(j=0;j<264;j++)
   {
    m = read(imagenfd, p_linea, sizeof(linea));
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
  close(imagenfd);
 }
```

```
printf("imagenes en memoria");
hora1=ProcTime();
for(k=0;k<16;k++)
 {
 for(j=0;j<264;j++)
  {
  for(i=0;i<512;i++)
   {
    linea[i]=cubo[j][i][k];
   }
   ChanOut(out_chan, p_linea, 512);
  }
 }
for(k=0;k<16;k++)
 {
 for(j=0;j<256;j++) gris[j]=0;
 for(j=0;j<264;j++)
  {
  for(i=0;i<512;i++)
   {
    his(cubo[j][i][k]);
   }
  }
 otsu();
 opaco();
 }
for(j=0;j<264;j++)
 {
 for(i=0;i<256;i++)
  {
   Cin=0;
   for(k=0;k<16;k++)
    {
    if(cubo[j][i][k]>0)
     {
      aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
      fval1=(float)(aux1*0.01);
     aux1=(unsigned int)(Cin*fval1);
      aux=(char)aux1;
      aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
      fval1=(float)(aux1*0.01);
      aux1=(unsigned int)(fval1);
      Cout=(char)(aux1+aux);
     } else Cout=0;
     Cin=Cout;
    }
   ima_resul[j][i]=Cout;
  }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
  ChanIn(in_chan, p_linea, 256);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
tiempo1=ProcTimeMinus(hora2,hora1);
printf(" \n numero de ciclos en procesar:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan);
printf(" \n num. ciclos en procesar T_2:  %d ",tiempo1);
  p_linea=&ima_resul[0][0];
   for(j=0;j<264;j++)
   {
    if ( (write(imagen3dfd, p_linea,m)) != m)
       perror("ERROR al escribir");
       p_linea=p_linea+512;
   }
close(imagen3dfd);
printf(" \n Termino de ejecucion \n ");
}
```

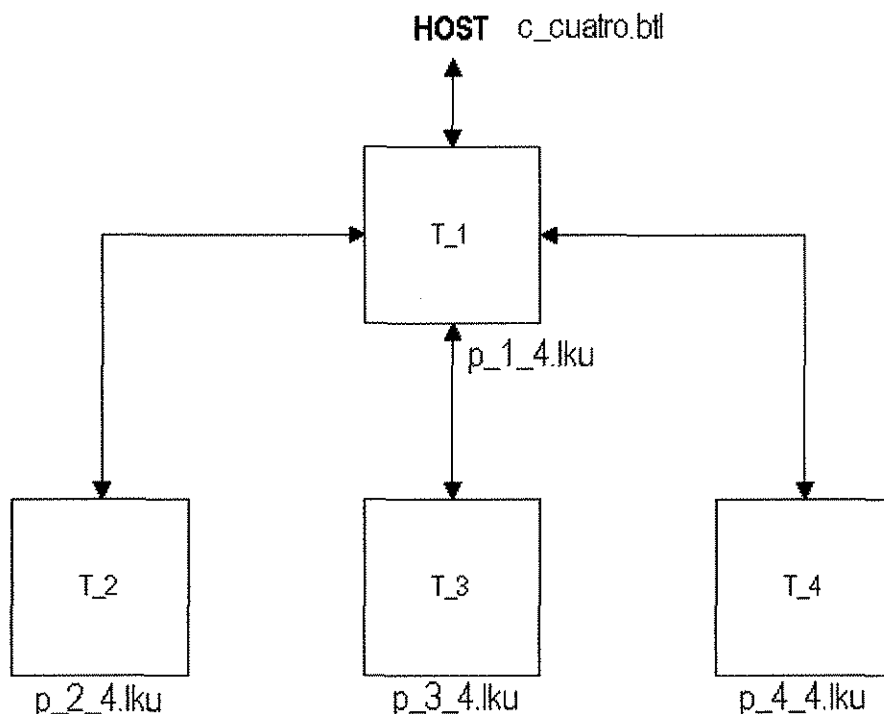
B.2.3 PROGRAMA TRANSPUTER T_2 (p_2_2.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */

int main()
{
Channel * in_chan = (Channel*) get_param(1);
Channel * out_chan = (Channel*) get_param(2);
p_linea=linea;
listo=0;
for(k=0;k<16;k++)
 {
 for(j=0;j<264;j++)
  {
   ChanIn(in_chan, p_linea, 512);
   if(listo==0)
    {
     hora1=ProcTime();
     listo=1;
    }
   for(i=0;i<512;i++)
    {
     cubo[j][i][k]=linea[i];
    }
  }
 }
for(k=0;k<16;k++)
 {
 for(j=0;j<256;j++) gris[j]=0;
 for(j=0;j<264;j++)
  {
   for(i=0;i<512;i++)
    {
```

```
      his(cubo[j][i][k]);
     }
    }
   otsu();
   opaco();
  }
for(j=0;j<264;j++)
 {
  for(i=256;i<512;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k])
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
      ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan, p_linea, 256);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

B.3 Programas para cuatro transputers



B..3.1 PROGRAMA DE CONFIGURACIÓN (c_cuatro.cfs)

```
T805 (memory=8M) T_1, T_2, T_3, T_4;
connect host to T_1.link[1];
connect T_1.link[0] to T_2.link[0];
connect T_1.link[2] to T_3.link[1];
connect T_1.link[3] to T_4.link[0];
process(stacksize=10K, heapsize=100K,
      interface(input host_in, output host_out,
      input data_in_1, output data_out_1, input
data_in_2, output data_out_2,
      input data_in_3, output data_out_3)
     ) raiz_1;
process(stacksize=10K, heapsize=100k,
           interface(input up_in_1, output up_out_1)
     ) nivel_1_2;
process(stacksize=10K, heapsize=100k,
           interface(input up_in_2, output up_out_2)
     ) nivel_1_3;
process(stacksize=10K, heapsize=100k,
           interface(input up_in_3, output up_out_3)
     ) nivel_1_4;
input HostInput;
output HostOutput;
connect raiz_1.host_in to HostInput;
connect raiz_1.host_out to HostOutput;
connect raiz_1.data_in_1 to nivel_1_2.up_out_1;
connect raiz_1.data_out_1 to nivel_1_2.up_in_1;
connect raiz_1.data_in_2 to nivel_1_3.up_out_2;
connect raiz_1.data_out_2 to nivel_1_3.up_in_2;
connect raiz_1.data_in_3 to nivel_1_4.up_out_3;
connect raiz_1.data_out_3 to nivel_1_4.up_in_3;
use "p_1_4.8.lku" for raiz_1;
use "p_2_4.8.lku" for nivel_1_2;
use "p_3_4.8.lku" for nivel_1_3;
use "p_4_4.8.lku" for nivel_1_4;
place raiz_1 on T_1;
place nivel_1_2 on T_2;
place nivel_1_3 on T_3;
place nivel_1_4 on T_4;
place HostInput on host;
place HostOutput on host;
```

B.3.2 PROGRAMA TRANSPUTER T_1 (p_1_4.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
```

```c
/* PROGRAMA PRINCIPAL */
int main()
{
Channel* in_chan1 = (Channel*) get_param(3);
Channel* out_chan1 = (Channel*) get_param(4);
Channel* in_chan2 = (Channel*) get_param(5);
Channel* out_chan2 = (Channel*) get_param(6);
Channel* in_chan3 = (Channel*) get_param(7);
Channel* out_chan3 = (Channel*) get_param(8);
if ( (imagen3dfd = open("i_r4.8.0", O_WRONLY |
O_TRUNC )) < 0)
    printf("Error al  Crear Archivo");
p_linea=linea;
for(k=0;k<16;k++)
 {
  if (k<9)
   {
    nombre[4]=numero[k+1];
   }
  if ( (k>8)&(k<25) )
   {
    aux=(k+1)/10;
    nombre[3]=numero[aux];
    nombre[4]=numero[k-9-((aux-1)*10)];
   }
  if ( ( imagenfd = open(nombre, O_RDONLY) ) < 0)
   {
    printf(" Error abriendo archivo");
   }
  for(j=0;j<264;j++)
   {
    m = read(imagenfd, p_linea, sizeof(linea));
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
  close(imagenfd);
 }
printf("imagenes en memoria");
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
   for(i=0;i<512;i++)
    {
     linea[i]=cubo[j][i][k];
    }
    ChanOut(out_chan1, p_linea, 512);
    ChanOut(out_chan2, p_linea, 512);
    ChanOut(out_chan3, p_linea, 512);
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
   otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=0;i<128;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan1, p_linea,128);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
  ChanIn(in_chan2, p_linea,128);
  p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
  ChanIn(in_chan3, p_linea,128);
  p_linea=p_linea+512;
 }
```

```
hora2=ProcTime();
tiempo1=ProcTimeMinus(hora2,hora1);
printf(" \n num. ciclos en procesar:  %d  \n",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_2:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_3:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_4:  %d ",tiempo1);
p_linea=&ima_resul[0][0];
for(j=0;j<264;j++)
  {
    if ( (write(imagen3dfd, p_linea,m)) != m)
      perror("ERROR al escribir");
    p_linea=p_linea+512;
  }
close(imagen3dfd);
printf("\n Termino de ejecucion \n ");
}
```

B.3.3 PROGRAMA TRANSPUTER T_2 (p_2_4.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
Channel * in_chan = (Channel*) get_param(1);
Channel * out_chan = (Channel*) get_param(2);
p_linea=linea;
hora1=ProcTime();
listo=0;
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=128;i<256;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k])
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan, p_linea, 128);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

B.3.4 PROGRAMA TRANSPUTER T_3 (p_3_4.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
Channel * in_chan = (Channel*) get_param(1);
Channel * out_chan = (Channel*) get_param(2);
p_linea=linea;
```

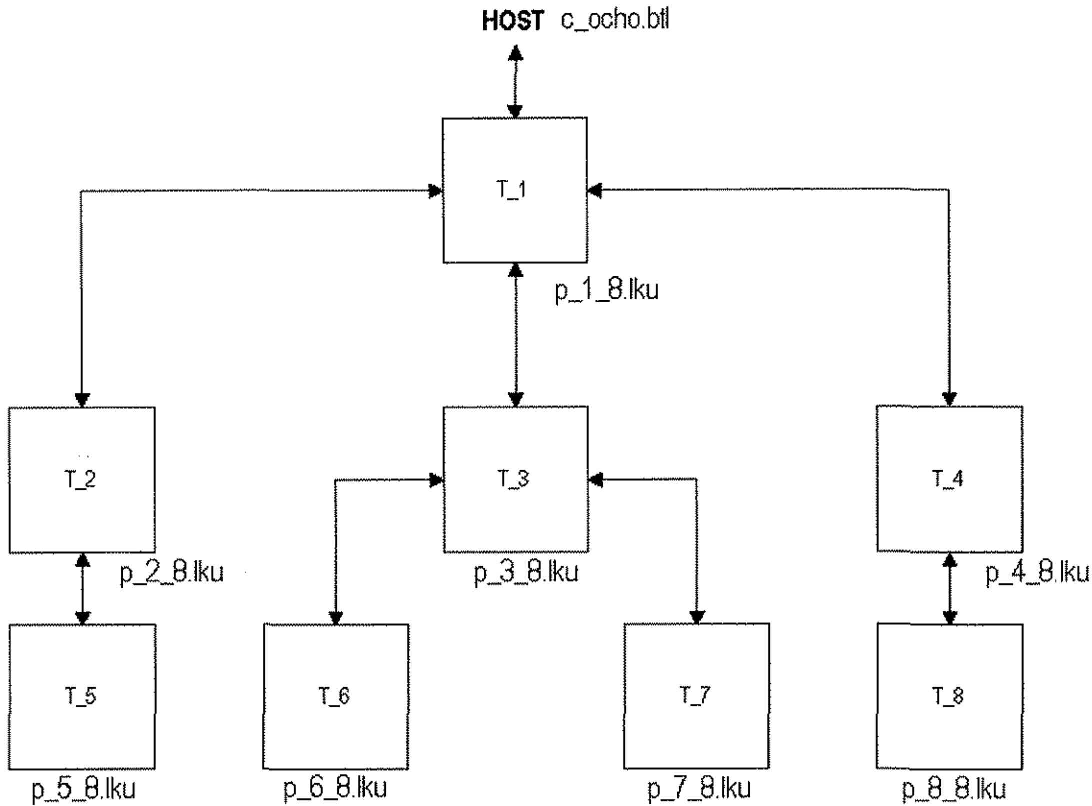
```
hora1=ProcTime();
listo=0;
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan, p_linea, 512);
    if(listo==0)
     {
      hora1=ProcTime();
      listo=1;
     }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=256;i<384;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k])
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan, p_linea, 128);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

B.3.5 PROGRAMA TRANSPUTER T_4 (p_4_4.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    ChanOut(out_chan2, p_linea, 512);
    if(listo==0)
     {
      hora1=ProcTime();
      listo=1;
     }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
```

```
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=192;i<256;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan2, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,64);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,64);
   p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
exit_terminate(EXIT_SUCCESS);
}
```

B.4 Programas para ocho transputers



B.4.1 PROGRAMA DE CONFIGURACIÓN (c_ocho.cfs)

```
T805 (memory=8M) T_1, T_2, T_3, T_4, T_5, T_6,
T_7, T_8;
connect host to T_1.link[1];
connect T_1.link[0] to T_2.link[0];
connect T_1.link[2] to T_3.link[1];
connect T_1.link[3] to T_4.link[0];
connect T_3.link[0] to T_6.link[0];
connect T_3.link[3] to T_7.link[0];
connect T_2.link[3] to T_5.link[0];
connect T_4.link[3] to T_8.link[0];
process(stacksize=10K, heapsize=100K,
    interface(input host_in, output host_out,
    input data_in_1, output data_out_1, input
data_in_2, output data_out_2,
    input data_in_3, output data_out_3)
   ) raiz_1;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
    output  up_out_2)
   ) nivel_1_2;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
    output up_out_2, input up_in_3, output up_out_3)
   ) nivel_1_3;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
    output up_out_2)
   ) nivel_1_4;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
   ) nivel_2_5;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
   ) nivel_2_6;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
   ) nivel_2_7;
process(stacksize=10K, heapsize=100k,
```

```
            interface(input up_in_1, output up_out_1)
      ) nivel_2_8;
input HostInput;
output HostOutput;
connect raiz_1.host_in to HostInput;
connect raiz_1.host_out to HostOutput;
connect raiz_1.data_in_1 to nivel_1_2.up_out_1;
connect raiz_1.data_out_1 to nivel_1_2.up_in_1;
connect raiz_1.data_in_2 to nivel_1_3.up_out_1;
connect raiz_1.data_out_2 to nivel_1_3.up_in_1;
connect raiz_1.data_in_3 to nivel_1_4.up_out_1;
connect raiz_1.data_out_3 to nivel_1_4.up_in_1;
connect nivel_1_2.up_in_2 to nivel_2_5.up_out_1;
connect nivel_1_2.up_out_2 to nivel_2_5.up_in_1;
connect nivel_1_3.up_in_2 to nivel_2_6.up_out_1;
connect nivel_1_3.up_out_2 to nivel_2_6.up_in_1;
connect nivel_1_3.up_in_3 to nivel_2_7.up_out_1;
connect nivel_1_3.up_out_3 to nivel_2_7.up_in_1;
connect nivel_1_4.up_in_2 to nivel_2_8.up_out_1;
connect nivel_1_4.up_out_2 to nivel_2_8.up_in_1;

use "p_1_8.8.lku" for raiz_1;
use "p_2_8.8.lku" for nivel_1_2;
use "p_3_8.8.lku" for nivel_1_3;
use "p_4_8.8.lku" for nivel_1_4;
use "p_5_8.8.lku" for nivel_2_5;
use "p_6_8.8.lku" for nivel_2_6;
use "p_7_8.8.lku" for nivel_2_7;
use "p_8_8.8.lku" for nivel_2_8;
place raiz_1 on T_1;
place nivel_1_2 on T_2;
place nivel_1_3 on T_3;
place nivel_1_4 on T_4;
place nivel_2_5 on T_5;
place nivel_2_6 on T_6;
place nivel_2_7 on T_7;
place nivel_2_8 on T_8;
place HostInput on host;
place HostOutput on host;
```

B.4.2 PROGRAMA TRANSPUTER T_1 (p_1_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel* in_chan1 = (Channel*) get_param(3);
 Channel* out_chan1 = (Channel*) get_param(4);
 Channel* in_chan2 = (Channel*) get_param(5);
 Channel* out_chan2 = (Channel*) get_param(6);
 Channel* in_chan3 = (Channel*) get_param(7);
 Channel* out_chan3 = (Channel*) get_param(8);
 if ( (imagen3dfd = open("i_r8.8.0", O_WRONLY |
O_TRUNC )) < 0)
   printf("Error al  Crear Archivo");
p_linea=linea;
 for(k=0;k<16;k++)
  {
   if (k<9)
    {
     nombre[4]=numero[k+1];
    }
   if ( (k>8)&(k<25) )
    {
     aux=(k+1)/10;
     nombre[3]=numero[aux];
     nombre[4]=numero[k-9-((aux-1)*10)];
    }
   if ( ( imagenfd = open(nombre, O_RDONLY) ) < 0)
    {
     printf(" Error abriendo archivo");
    }
   for(j=0;j<264;j++)
    {
     m = read(imagenfd, p_linea, sizeof(linea));
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
   close(imagenfd);
  }
printf("imagenes en memoria");
hora1=ProcTime();
for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       linea[i]=cubo[j][i][k];
      }
     ChanOut(out_chan1, p_linea, 512);
     ChanOut(out_chan2, p_linea, 512);
     ChanOut(out_chan3, p_linea, 512);
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
     {
      for(i=0;i<512;i++)
        {
         his(cubo[j][i][k]);
        }
     }
```

```
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
  {
  for(i=0;i<64;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
       {
        if(cubo[j][i][k]>0)
        {
          aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
          fval1=(float)(aux1*0.01);
          aux1=(unsigned int)(Cin*fval1);
          aux=(char)aux1;
          aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
          fval1=(float)(aux1*0.01);
          aux1=(unsigned int)(fval1);
          Cout=(char)(aux1+aux);
        } else Cout=0;
        Cin=Cout;
       }
      ima_resul[j][i]=Cout;
    }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+64;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan1, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan2, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan3, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan1, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,128);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan3, p_linea,64);
    p_linea=p_linea+512;
  }
hora2=ProcTime();
tiempo1=ProcTimeMinus(hora2,hora1);
printf(" \n num. ciclos en procesar T_1:  %d
\n",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_2:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_3:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_4:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_5:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_6:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_7:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_8:  %d ",tiempo1);
p_linea=&ima_resul[0][0];
 for(j=0;j<264;j++)
   {
    if ( (write(imagen3dfd, p_linea,m)) != m)
      perror("ERROR al escribir");
    p_linea=p_linea+512;
   }
 close(imagen3dfd);
 printf(" Termino \n ");
}
```

B.4.3 PROGRAMA TRANSPUTER T_2 (p_2_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
```

```
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
p_linea=linea;
hora1=ProcTime();
listo=0;
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    ChanOut(out_chan2, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=64;i<128;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,64);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+64;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
exit_terminate(EXIT_SUCCESS);
}
```

B.4.4 PROGRAMA TRANSPUTER T_3 (p_3_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
 Channel * in_chan3 = (Channel*) get_param(5);
 Channel * out_chan3 = (Channel*) get_param(6);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    ChanOut(out_chan2, p_linea, 512);
```

```
    ChanOut(out_chan3, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=128;i<192;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[i][j][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan2, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan3, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,64);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,128);
   p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
exit_terminate(EXIT_SUCCESS);
}
```

B.4.5 PROGRAMA TRANSPUTER T_4 (p_4_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    ChanOut(out_chan2, p_linea, 512);
    if(listo==0)
      {
```

```
        hora1=ProcTime();
        listo=1;
       }
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=192;i<256;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan2, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
 {
    ChanOut(out_chan1, p_linea,64);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
    ChanOut(out_chan1, p_linea,64);
    p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
exit_terminate(EXIT_SUCCESS);
}
```

B.4.6 PROGRAMA TRANSPUTER T_5 (p_5_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
```

```
      his(cubo[j][i][k]);
     }
    }
   otsu();
   opaco();
  }
for(j=0;j<264;j++)
 {
  for(i=256;i<320;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,64);
   p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

B.4.7 PROGRAMA TRANSPUTER T_6 (p_6_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
   otsu();
   opaco();
  }
for(j=0;j<264;j++)
  {
   for(i=320;i<384;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
        {
         aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(Cin*fval1);
         aux=(char)aux1;
         aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(fval1);
         Cout=(char)(aux1+aux);
        } else Cout=0;
       Cin=Cout;
      }
     ima_resul[j][i]=Cout;
    }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
```

```
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

B.4.8 PROGRAMA TRANSPUTER T_7 (p_7_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
     {
      hora1=ProcTime();
      listo=1;
     }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=384;i<448;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

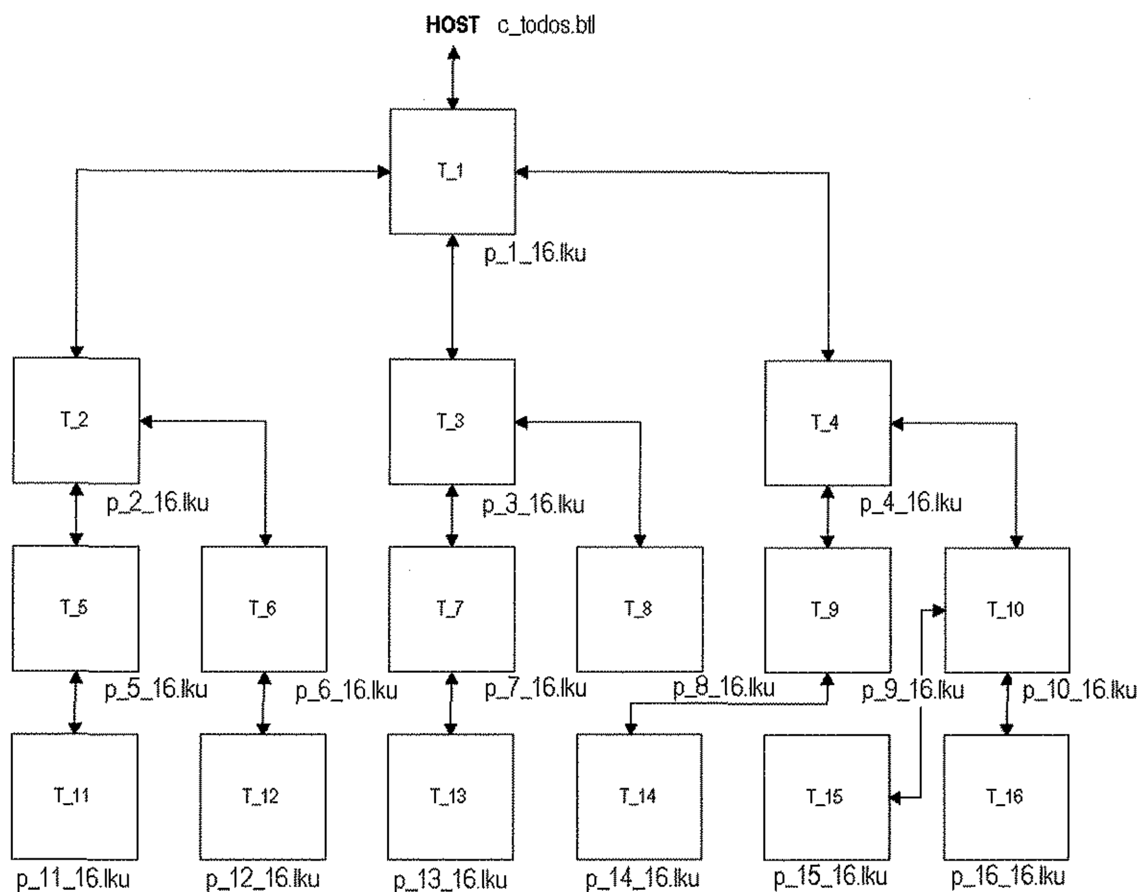
B.4.9 PROGRAMA TRANSPUTER T_8 (p_8_8.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"


/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
listo=0;
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
```

```
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
for(j=0;j<264;j++)
 {
  for(i=448;i<512;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
exit_terminate(EXIT_SUCCESS);
}
```

A.5 Programas para dieciséis transputers



B.5.1 PROGRAMA DE CONFIGURACIÓN (c_todos.cfs)

```
T805 (memory=8M) T_1, T_2, T_3, T_4, T_5, T_6,
T_7, T_8, T_9, T_10, T_11, T_12,
T_13, T_14, T_15, T_16;
connect host to T_1.link[1];
connect T_1.link[0] to T_2.link[3];
connect T_1.link[2] to T_3.link[1];
connect T_1.link[3] to T_4.link[3];
connect T_2.link[0] to T_5.link[0];
connect T_2.link[2] to T_6.link[1];
connect T_3.link[0] to T_7.link[3];
connect T_3.link[3] to T_8.link[0];
connect T_4.link[0] to T_9.link[3];
connect T_4.link[2] to T_10.link[1];
connect T_5.link[2] to T_11.link[1];
connect T_6.link[0] to T_12.link[3];
connect T_7.link[1] to T_13.link[2];
connect T_9.link[2] to T_14.link[1];
connect T_10.link[0] to T_15.link[0];
connect T_10.link[3] to T_16.link[3];
process(stacksize=10K, heapsize=100K,
    interface(input host_in, output host_out,
    input data_in_1, output data_out_1, input
data_in_2, output data_out_2,
    input data_in_3, output data_out_3)
   ) raiz_1;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
    output up_out_2, input up_in_3, output up_out_3)
   ) nivel_1_2;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
    output up_out_2, input up_in_3, output up_out_3)
   ) nivel_1_3;
process(stacksize=10K, heapsize=100k,
```

```
        interface(input up_in_1, output up_out_1,
input up_in_2,
      output up_out_2, input up_in_3, output up_out_3)
    ) nivel_1_4;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
      output up_out_2)
    ) nivel_2_5;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
      output up_out_2)
    ) nivel_2_6;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
      output up_out_2)
    ) nivel_2_7;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_2_8;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
      output up_out_2)
    ) nivel_2_9;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1,
input up_in_2,
      output up_out_2, input up_in_3, output up_out_3)
    ) nivel_2_10;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_3_11;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_3_12;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_3_13;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_3_14;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_3_15;
process(stacksize=10K, heapsize=100k,
        interface(input up_in_1, output up_out_1)
    ) nivel_3_16;
input HostInput;
output HostOutput;
connect raiz_1.host_in to HostInput;
connect raiz_1.host_out to HostOutput;
connect raiz_1.data_in_1 to nivel_1_2.up_out_1;
connect raiz_1.data_out_1 to nivel_1_2.up_in_1;
connect raiz_1.data_in_2 to nivel_1_3.up_out_1;
connect raiz_1.data_out_2 to nivel_1_3.up_in_1;
connect raiz_1.data_in_3 to nivel_1_4.up_out_1;
connect raiz_1.data_out_3 to nivel_1_4.up_in_1;
connect nivel_1_2.up_in_2 to nivel_2_5.up_out_1;
connect nivel_1_2.up_out_2 to nivel_2_5.up_in_1;
connect nivel_1_2.up_in_3 to nivel_2_6.up_out_1;
connect nivel_1_2.up_out_3 to nivel_2_6.up_in_1;
connect nivel_1_3.up_in_2 to nivel_2_7.up_out_1;
connect nivel_1_3.up_out_2 to nivel_2_7.up_in_1;
connect nivel_1_3.up_in_3 to nivel_2_8.up_out_1;
connect nivel_1_3.up_out_3 to nivel_2_8.up_in_1;
connect nivel_1_4.up_in_2 to nivel_2_9.up_out_1;
connect nivel_1_4.up_out_2 to nivel_2_9.up_in_1;
connect nivel_1_4.up_in_3 to nivel_2_10.up_out_1;
connect nivel_1_4.up_out_3 to nivel_2_10.up_in_1;
connect nivel_2_5.up_in_2 to nivel_3_11.up_out_1;
connect nivel_2_5.up_out_2 to nivel_3_11.up_in_1;
connect nivel_2_6.up_in_2 to nivel_3_12.up_out_1;
connect nivel_2_6.up_out_2 to nivel_3_12.up_in_1;
connect nivel_2_7.up_in_2 to nivel_3_13.up_out_1;
connect nivel_2_7.up_out_2 to nivel_3_13.up_in_1;
connect nivel_2_9.up_in_2 to nivel_3_14.up_out_1;
connect nivel_2_9.up_out_2 to nivel_3_14.up_in_1;
connect nivel_2_10.up_in_2 to nivel_3_15.up_out_1;
connect nivel_2_10.up_out_2 to nivel_3_15.up_in_1;
connect nivel_2_10.up_in_3 to nivel_3_16.up_out_1;
connect nivel_2_10.up_out_3 to nivel_3_16.up_in_1;

use "p_1_16.8.lku" for raiz_1;
use "p_2_16.8.lku" for nivel_1_2;
use "p_3_16.8.lku" for nivel_1_3;
use "p_4_16.8.lku" for nivel_1_4;
use "p_5_16.8.lku" for nivel_2_5;
use "p_6_16.8.lku" for nivel_2_6;
use "p_7_16.8.lku" for nivel_2_7;
use "p_8_16.8.lku" for nivel_2_8;
use "p_9_16.8.lku" for nivel_2_9;
use "p_10_16.8.lku" for nivel_2_10;
use "p_11_16.8.lku" for nivel_3_11;
use "p_12_16.8.lku" for nivel_3_12;
use "p_13_16.8.lku" for nivel_3_13;
use "p_14_16.8.lku" for nivel_3_14;
use "p_15_16.8.lku" for nivel_3_15;
use "p_16_16.8.lku" for nivel_3_16;

place raiz_1 on T_1;
place nivel_1_2 on T_2;
place nivel_1_3 on T_3;
place nivel_1_4 on T_4;
place nivel_2_5 on T_5;
place nivel_2_6 on T_6;
place nivel_2_7 on T_7;
place nivel_2_8 on T_8;
place nivel_2_9 on T_9;
place nivel_2_10 on T_10;
```

```
place nivel_3_11 on T_11;
place nivel_3_12 on T_12;
place nivel_3_13 on T_13;
place nivel_3_14 on T_14;
place nivel_3_15 on T_15;
place nivel_3_16 on T_16;
place HostInput on host;
place HostOutput on host;
```

B.5.2 PROGRAMA TRANSPUTER T_1 (p_1_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(3);
 Channel * out_chan1 = (Channel*) get_param(4);
 Channel * in_chan2 = (Channel*) get_param(5);
 Channel * out_chan2 = (Channel*) get_param(6);
 Channel * in_chan3 = (Channel*) get_param(7);
 Channel * out_chan3 = (Channel*) get_param(8);
  if ( (imagen3dfd = open("i_r16.8.0", O_WRONLY |
O_TRUNC )) < 0)
    printf("Error al  Crear Archivo");
p_linea=linea;
bandera=116;
for(k=0;k<16;k++)
 {
  if (k<9)
   {
    nombre[4]=numero[k+1];
   }
  if ( (k>8)&(k<25) )
   {
    aux=(k+1)/10;
    nombre[3]=numero[aux];
    nombre[4]=numero[k-9-((aux-1)*10)];
   }
  if ( ( imagenfd = open(nombre, O_RDONLY) ) < 0)
   {
    printf(" Error abriendo archivo");
   }
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    m = read(imagenfd, p_linea, sizeof(linea));
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
  close(imagenfd);
 }
printf("imagenes en memoria");
hora1=ProcTime();
for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      linea[i]=cubo[j][i][k];
     }
    ChanOut(out_chan1, p_linea, 512);
    ChanOut(out_chan2, p_linea, 512);
    ChanOut(out_chan3, p_linea, 512);
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
   for(i=0;i<32;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
        {
         aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(Cin*fval1);
         aux=(char)aux1;
         aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(fval1);
         Cout=(char)(aux1+aux);
        } else Cout=0;
       Cin=Cout;
      }
     ima_resul[j][i]=Cout;
    }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+32;
for(j=0;j<264;j++)
 {
```

```
    ChanIn(in_chan1, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan1, p_linea,64);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan1, p_linea,64);
    p_linea=p_linea+512;
  }

p_linea=&ima_resul[0][0];
p_linea=p_linea+64;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,64);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+96;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan3, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan3, p_linea,64);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+416;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan3, p_linea,96);
    p_linea=p_linea+512;
  }
hora2=ProcTime();
tiempo1=ProcTimeMinus(hora2,hora1);
printf(" \n num. ciclos en procesar T_1:  %d
\n",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_2:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_3:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_4:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_5:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_6:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_7:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_8:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_9:  %d ",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_10:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_11:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan1);
printf(" \n num. ciclos en procesar T_12:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan2);
printf(" \n num. ciclos en procesar T_13:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_14:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_15:  %d
",tiempo1);
tiempo1=ChanInInt(in_chan3);
printf(" \n num. ciclos en procesar T_16:  %d
",tiempo1);
  p_linea=&ima_resul[0][0];
  for(j=0;j<264;j++)
    {
      if ( (write(imagen3dfd, p_linea,m)) != m)
        perror("ERROR al escribir");
      p_linea=p_linea+512;
    }
  close(imagen3dfd);
  printf(" \n Termino \n ");
}
```

B.5.3 PROGRAMA TRANSPUTER T_2 (p_2_16.c)

```
#include "bibliote.h
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
 Channel * in_chan3 = (Channel*) get_param(5);
 Channel * out_chan3 = (Channel*) get_param(6);
 p_linea=linea;
 for(k=0;k<16;k++)
  {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
     {
      hora1=ProcTime();
      listo=1;
     }
    ChanOut(out_chan2, p_linea, 512);
    ChanOut(out_chan3, p_linea, 512);
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
  }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
  for(i=32;i<64;i++)
   {

    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
      {
       aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(Cin*fval1);
       aux=(char)aux1;
       aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(fval1);
       Cout=(char)(aux1+aux);
      } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+160;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan3, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+352;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan3, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+32;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }

p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
```

```
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.4 PROGRAMA TRANSPUTER T_3 (p_3_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
 Channel * in_chan3 = (Channel*) get_param(5);
 Channel * out_chan3 = (Channel*) get_param(6);
p_linea=linea;
 for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
     {
      hora1=ProcTime();
      listo=1;
     }
    ChanOut(out_chan2, p_linea, 512);
    ChanOut(out_chan3, p_linea, 512);
    for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
   for(i=64;i<96;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
       Cin=Cout;
      }
     ima_resul[j][i]=Cout;
    }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+224;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan3, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
```

```
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+64;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.5 PROGRAMA TRANSPUTER T_4 (p_4_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
 Channel * in_chan3 = (Channel*) get_param(5);
 Channel * out_chan3 = (Channel*) get_param(6);
```

```
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     ChanOut(out_chan2, p_linea, 512);
     ChanOut(out_chan3, p_linea, 512);
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
   otsu();
   opaco();
  }
 for(j=0;j<264;j++)
  {
   for(i=96;i<128;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
        {
         aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(Cin*fval1);
         aux=(char)aux1;
         aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(fval1);
         Cout=(char)(aux1+aux);
        } else Cout=0;
       Cin=Cout;
      }

     ima_resul[j][i]=Cout;
    }
  }
```

```
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+288;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan3, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+416;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan3, p_linea,64);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+96;
for(j=0;j<264;j++)
  {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
  {
   ChanOut(out_chan1, p_linea,64);
   p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+416;
for(j=0;j<264;j++)
  {
   ChanOut(out_chan1, p_linea,96);
   p_linea=p_linea+512;
  }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);

tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.6 PROGRAMA TRANSPUTER T_5 (p_5_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     ChanOut(out_chan2, p_linea, 512);
     for(i=0;i<512;i++)
       {
        cubo[j][i][k]=linea[i];
       }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
       {
        his(cubo[j][i][k]);
       }
    }
   otsu();
   opaco();
  }
 for(j=0;j<264;j++)
```

```
{
for(i=128;i<160;i++)
 {
  Cin=0;
  for(k=0;k<16;k++)
   {
    if(cubo[j][i][k]>0)
    {
     aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
     fval1=(float)(aux1*0.01);
     aux1=(unsigned int)(Cin*fval1);
     aux=(char)aux1;
     aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
     fval1=(float)(aux1*0.01);
     aux1=(unsigned int)(fval1);
     Cout=(char)(aux1+aux);
    } else Cout=0;
    Cin=Cout;
   }
  ima_resul[j][i]=Cout;
 }
}
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+128;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.7 PROGRAMA TRANSPUTER T_6 (p_6_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     ChanOut(out_chan2, p_linea, 512);
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
   otsu();
   opaco();
  }
 for(j=0;j<264;j++)
  {
   for(i=160;i<192;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
       {
```

```
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
      } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+352;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+160;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+352;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.8 PROGRAMA TRANSPUTER T_7 (p_7_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
  p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
     ChanOut(out_chan2, p_linea, 512);
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
   otsu();
   opaco();
  }
  for(j=0;j<264;j++)
   {
    for(i=192;i<224;i++)
     {
      Cin=0;
      for(k=0;k<16;k++)
       {
        if(cubo[j][i][k]>0)
        {
          aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
          fval1=(float)(aux1*0.01);
          aux1=(unsigned int)(Cin*fval1);
          aux=(char)aux1;
          aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
          fval1=(float)(aux1*0.01);
          aux1=(unsigned int)(fval1);
          Cout=(char)(aux1+aux);
```

```
        } else Cout=0;
        Cin=Cout;
       }
      ima_resul[j][i]=Cout;
     }
   }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+192;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.9 PROGRAMA TRANSPUTER T_8 (p_8_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */
int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
   for(i=224;i<256;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
       Cin=Cout;
      }
     ima_resul[j][i]=Cout;
    }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+224;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
```

```
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.10 PROGRAMA TRANSPUTER T_9 (p_9_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */

int main()
{
  Channel * in_chan1 = (Channel*) get_param(1);
  Channel * out_chan1 = (Channel*) get_param(2);
  Channel * in_chan2 = (Channel*) get_param(3);
  Channel * out_chan2 = (Channel*) get_param(4);
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     ChanOut(out_chan2, p_linea, 512);
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
   otsu();
   opaco();
  }
  for(j=0;j<264;j++)
   {
    for(i=256;i<288;i++)
     {
      Cin=0;
      for(k=0;k<16;k++)
       {
        if(cubo[j][i][k]>0)
        {
         aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(Cin*fval1);
         aux=(char)aux1;
         aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
         fval1=(float)(aux1*0.01);
         aux1=(unsigned int)(fval1);
         Cout=(char)(aux1+aux);
        } else Cout=0;
        Cin=Cout;
       }
      ima_resul[j][i]=Cout;
     }
   }
p_linea=&ima_resul[0][0];
p_linea=p_linea+416;
for(j=0;j<264;j++)
  {
    ChanIn(in_chan2, p_linea,32);
    p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+256;
for(j=0;j<264;j++)
  {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+416;
for(j=0;j<264;j++)
  {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
  }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.11 PROGRAMA TRANSPUTER T_10 (p_10_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
```

```
/* PROGRAMA PRINCIPAL */

int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
 Channel * in_chan2 = (Channel*) get_param(3);
 Channel * out_chan2 = (Channel*) get_param(4);
 Channel * in_chan3 = (Channel*) get_param(5);
 Channel * out_chan3 = (Channel*) get_param(6);
p_linea=linea;
 for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
  {
   ChanIn(in_chan1, p_linea, 512);
   if(listo==0)
    {
     hora1=ProcTime();
     listo=1;
    }
   ChanOut(out_chan2, p_linea, 512);
   ChanOut(out_chan3, p_linea, 512);
   for(i=0;i<512;i++)
    {
     cubo[j][i][k]=linea[i];
    }
  }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
   for(i=0;i<512;i++)
    {
     his(cubo[j][i][k]);
    }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
  for(i=288;i<320;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
     if(cubo[j][i][k]>0)
     {
      aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
      fval1=(float)(aux1*0.01);
      aux1=(unsigned int)(Cin*fval1);
      aux=(char)aux1;
      aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
      fval1=(float)(aux1*0.01);
      aux1=(unsigned int)(fval1);
      Cout=(char)(aux1+aux);
     } else Cout=0;
     Cin=Cout;
    }
   ima_resul[j][i]=Cout;
  }
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan2, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+480;
for(j=0;j<264;j++)
 {
   ChanIn(in_chan3, p_linea,32);
   p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+288;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,64);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
tiempo1=ChanInInt(in_chan2);
ChanOutInt(out_chan1, tiempo1);
tiempo1=ChanInInt(in_chan3);
ChanOutInt(out_chan1, tiempo1);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.12 PROGRAMA TRANSPUTER T_11 (p_11_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"
```

```
/* PROGRAMA PRINCIPAL */

int main()
{
  Channel * in_chan1 = (Channel*) get_param(1);
  Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
       his(cubo[j][i][k]);
      }
    }
   otsu();
   opaco();
  }
 for(j=0;j<264;j++)
  {
   for(i=320;i<352;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
       Cin=Cout;
      }
        ima_resul[j][i]=Cout;
     }
   }
p_linea=&ima_resul[0][0];
p_linea=p_linea+320;
for(j=0;j<264;j++)
  {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
  }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.13 PROGRAMA TRANSPUTER T_12 (p_12_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */

int main()
{
  Channel * in_chan1 = (Channel*) get_param(1);
  Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
 for(k=0;k<16;k++)
  {
   for(j=0;j<264;j++)
    {
     ChanIn(in_chan1, p_linea, 512);
     if(listo==0)
       {
        hora1=ProcTime();
        listo=1;
       }
     for(i=0;i<512;i++)
      {
       cubo[j][i][k]=linea[i];
      }
    }
  }
for(k=0;k<16;k++)
  {
   for(j=0;j<256;j++) gris[j]=0;
   for(j=0;j<264;j++)
    {
     for(i=0;i<512;i++)
      {
```

```
        his(cubo[j][i][k]);
       }
     }
   otsu();
   opaco();
  }
  for(j=0;j<264;j++)
   {
   for(i=352;i<384;i++)
    {
     Cin=0;
     for(k=0;k<16;k++)
      {
       if(cubo[j][i][k]>0)
       {
        aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(Cin*fval1);
        aux=(char)aux1;
        aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
        fval1=(float)(aux1*0.01);
        aux1=(unsigned int)(fval1);
        Cout=(char)(aux1+aux);
       } else Cout=0;
       Cin=Cout;
      }
     ima_resul[j][i]=Cout;
    }
   }
p_linea=&ima_resul[0][0];
p_linea=p_linea+352;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.14 PROGRAMA TRANSPUTER T_13 (p_13_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */

int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
 for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
  {
   ChanIn(in_chan1, p_linea, 512);
   if(listo==0)
    {
     hora1=ProcTime();
     listo=1;
    }
   for(i=0;i<512;i++)
    {
     cubo[j][i][k]=linea[i];
    }
  }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
  for(i=384;i<416;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
      {
       aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(Cin*fval1);
       aux=(char)aux1;
       aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(fval1);
       Cout=(char)(aux1+aux);
      } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
  }
```

```
p_linea=&ima_resul[0][0];
p_linea=p_linea+384;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.15 PROGRAMA TRANSPUTER T_14 (p_14_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */

int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
 for(k=0;k<16;k++)
 {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
 }
for(k=0;k<16;k++)
 {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
 }
 for(j=0;j<264;j++)
  {
  for(i=416;i<448;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
      {
       aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(Cin*fval1);
       aux=(char)aux1;
       aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(fval1);
       Cout=(char)(aux1+aux);
      } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+416;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.16 PROGRAMA TRANSPUTER T_15 (p_15_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */

int main()
{
 Channel * in_chan1 = (Channel*) get_param(1);
 Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
```

```
 for(k=0;k<16;k++)
  {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
      {
      cubo[j][i][k]=linea[i];
      }
   }
  }
for(k=0;k<16;k++)
  {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
  }
 for(j=0;j<264;j++)
  {
  for(i=448;i<480;i++)
   {
    Cin=0;
    for(k=0;k<16;k++)
     {
      if(cubo[j][i][k]>0)
      {
       aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(Cin*fval1);
       aux=(char)aux1;
       aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
       fval1=(float)(aux1*0.01);
       aux1=(unsigned int)(fval1);
       Cout=(char)(aux1+aux);
      } else Cout=0;
      Cin=Cout;
     }
    ima_resul[j][i]=Cout;
   }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+448;
for(j=0;j<264;j++)
 {
   ChanOut(out_chan1, p_linea,32);
   p_linea=p_linea+512;
  }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.17 PROGRAMA TRANSPUTER T_16 (p_16_16.c)

```
#include "bibliote.h"
#include "variablesf.h"
#include "histograma.h"
#include "segmenta9.h"
#include "t_opacidad1.h"

/* PROGRAMA PRINCIPAL */

int main()
{
  Channel * in_chan1 = (Channel*) get_param(1);
  Channel * out_chan1 = (Channel*) get_param(2);
p_linea=linea;
 for(k=0;k<16;k++)
  {
  for(j=0;j<264;j++)
   {
    ChanIn(in_chan1, p_linea, 512);
    if(listo==0)
      {
       hora1=ProcTime();
       listo=1;
      }
    for(i=0;i<512;i++)
     {
      cubo[j][i][k]=linea[i];
     }
   }
  }
for(k=0;k<16;k++)
  {
  for(j=0;j<256;j++) gris[j]=0;
  for(j=0;j<264;j++)
   {
    for(i=0;i<512;i++)
     {
      his(cubo[j][i][k]);
     }
   }
  otsu();
  opaco();
  }
```

```
for(j=0;j<264;j++)
 {
 for(i=480;i<512;i++)
  {
   Cin=0;
   for(k=0;k<16;k++)
    {
     if(cubo[j][i][k]>0)
     {
      aux1=( (100-opacidad[(cubo[j][i][k])][k]) );
      fval1=(float)(aux1*0.01);
      aux1=(unsigned int)(Cin*fval1);
      aux=(char)aux1;
      aux1=(cubo[j][i][k]*opacidad[(cubo[j][i][k])][k]);
      fval1=(float)(aux1*0.01);
      aux1=(unsigned int)(fval1);
      Cout=(char)(aux1+aux);
     } else Cout=0;
     Cin=Cout;
    }
    ima_resul[j][i]=Cout;
   }
  }
p_linea=&ima_resul[0][0];
p_linea=p_linea+480;
for(j=0;j<264;j++)
 {
  ChanOut(out_chan1, p_linea,32);
  p_linea=p_linea+512;
 }
hora2=ProcTime();
t1_seg=ProcTimeMinus(hora2,hora1);
ChanOutInt(out_chan1, t1_seg);
 exit_terminate(EXIT_SUCCESS);
}
```

B.5.18 Archivo bibliote.h

```
#include <stdio.h>
#include <channel.h>
#include <misc.h>
#include <string.h>
#include <stdlib.h>
#include <iocntrl.h>
#include <math.h>
#include <time.h>
#include <process.h>
```

B.5.19 Archivo variablesf.h

```
int         imagenfd,         /* fd  de imagen a cargar al
arreglo */
            imagen3dfd;       /* file de salida */

char        nombre[9]={'0','0','0','0','0','.','i','m','a'};
            /* nombre de base de la imagenes a
procesar */
char        numero[10]={'0','1','2','3','4','5','6','7','8','9'};
            /* numero de secuencia de la imagen a leer
*/
unsigned int  i,j,k,        /* indices del arreglo cubico */
            ii,jj,l,
            dato;

char        aux;   /*  variable auxiliar para lectura de
imagenes */


char                 linea[512]; /* arreglo para lectura de
renglones de la imagen */
char                 *p_linea;  /* apuntador para lectura
de imagenes */
char        resultado[512]; /* arreglo de renglo de
imagen resultado */

int         n,
            m;  /* numero de bytes leidos y a escribir */

unsigned int  gris[256]={0,0}; /* arreglo para
histograma */

char      cubo[512][264][15],     /* arreglo cubico de
imagenes */
            opacidad[256][15];      /* tabla de opacidades
*/

int         limites[6]; /* arreglo de umbrales de
segmentacion */
int                  m_o;        /* numero de umbral a
calcular      */
int                  r1,r2;      /* umbrales para el calculo
del umbral */
char                 max_k[5];   /* arreglo para
almacenar umbrales    */

int                  bandera; /* bandera identificadora
de arreglo de procesadores */
char                 cte, cte1,  /*  variable auxiliar
*/
                     mm;   /* indice para clases en
imagen */
```

```
int                  lim_i,  /* limite inferior para calculo
de opacidades */
                     lim_s;  /* limite superior para calculo
de opacidades */

struct
{
unsigned int suma;
char indice;
unsigned int max;} clase[5]={0,0}; /* estructura de
caracteristicas de clases */

unsigned int a_suma;
unsigned int a_aux[5];

float                valor1;  /* valor maximo de gris
dentro de la clase */
float                valor2;  /* valor decimal de la
opacidad  */
double         valor3;
double               valor4;
float                valor11;

char           Cout,  /* valor de salida en la integral del
algoritmo  */
               Cin;   /* valor de entrada en la integral del
algoritmo */

char        Oout,
                     Oin;

float                fval1; /* variable auxiliar en la
evaluacion de la integral */
float          fval2,
               fval3;

double         fval4;

unsigned int         aux1;  /* variable auxiliar en cambio
de tipo de variable  */
unsigned int    aux2;

int          hora1,   /* hora inicial de un proceso */
             hora2,   /* hora final del un proceso  */
                     tiempo1; /* tiempo en ejecutar un
proceso */

int                  t1_seg,  /* Numero de ciclos en
almacenar datos */
                     t2_seg,  /* Numero de ciclos en
integrar datos  */
                     t3_seg;  /* Numero de ciclos en
enviar datos    */
char                 listo;
```

B.5.20 Archivo histograma.h

```
void his(int linea1)

{
  gris[linea1]=gris[linea1]+1;
  return;
}
```

B.5.20 Archivo segmenta9.h

```
void metodo()
{

float W[256];
float P[256];
float D[256];
float M[256];

int indice;
float D_2;
float D_1;
float mT;
float maximo;
long N;
float Igris;
int k_o;

 N=0;
  for (indice=r1; indice <= r2; indice++)
   {
    N=N+gris[indice];
   }
  mT=0;
  for (indice=r1; indice <= r2; indice++)
   {
    Igris=(float)gris[indice];
    P[indice]=(Igris/N);
    mT=(mT+(indice*P[indice]));
   }

  maximo=0;
  max_k[m_o]=0;
  for (indice=r1; indice <= r2; indice++)
   {
    W[indice]=0;
    M[indice]=0;

    for (k_o=r1; k_o <= indice; k_o++)
     {
      W[indice]=W[indice]+P[k_o];
      M[indice]=( M[indice] + ((k_o)*P[k_o]) );
     }

    D_2 = ( (mT*W[indice])-M[indice] )*(
(mT*W[indice])-M[indice] );
    D_1 = ( W[indice]*(1-W[indice]) );

   if (D_1 > 0.0)
    {
     D[indice]=(float)(D_2/D_1);
    }
    else
    {
     D[indice]=0;
    }
   if ( D[indice] > maximo )
    {
     maximo=D[indice];
     max_k[m_o]=indice;
    }
   }
}

void otsu()
{
max_k[0]=0;
max_k[1]=255;

r1=max_k[0];
r2=max_k[1];
m_o=2;
metodo();

r1=max_k[0];
r2=max_k[m_o];
m_o=3;
metodo();

r1=max_k[m_o]+1;
r2=max_k[m_o-1];
m_o=4;
metodo();

r1=max_k[m_o-2]+1;
r2=max_k[m_o-3];
m_o=5;
metodo();

 limites[0]=0;
 limites[1]=max_k[3];
 limites[2]=max_k[4];
 limites[3]=max_k[2];
 limites[4]=max_k[5];
 limites[5]=256;
}
```

B.5.21 Archivo t_opacidad1.h

```
void opaco()
{
cte=0;
for(mm=0;mm<5;mm++)
 {
  clase[mm].suma=0;
  clase[mm].indice=0;
  clase[mm].max=0;
 }
for(mm=0;mm<5;mm++)
 {
  for(i=limites[mm];i<limites[mm+1];i++)
   {
    clase[mm].suma=clase[mm].suma+gris[i];
    if(gris[i]>=clase[mm].max) clase[mm].max=gris[i];
   }
  clase[mm].indice=(4-mm);
 }
cte=0;
   for(i=limites[0];i<limites[1];i++)
    {
     valor4=(double)(clase[0].max);
     valor3=(double)(1.0- ((gris[i]/valor4)) )*100;
     aux=(unsigned int)(valor3);
     opacidad[i][k]=(char)aux;
    }

   for(i=limites[1];i<limites[2];i++)
    {
     valor4=(double)(clase[1].max);
     valor3=(double)(1.0- ((gris[i]/valor4)) )*100;
     aux=(unsigned int)(valor3);
     opacidad[i][k]=(char)aux;
    }
   for(i=limites[2];i<limites[3];i++)
    {
     valor4=(double)(clase[2].max);
     valor3=(double)(1.0- ((gris[i]/valor4)) )*100;
     aux=(unsigned int)(valor3);
     opacidad[i][k]=(char)aux;
    }

   for(i=limites[3];i<limites[4];i++)
    {
     valor4=(double)(clase[3].max);
     valor3=(double)(1.0- ((gris[i]/valor4)) )*100;
     aux=(unsigned int)(valor3);
     opacidad[i][k]=(char)aux;
    }

   for(i=limites[4];i<limites[5];i++)
    {
     valor4=(double)(clase[4].max);
     valor3=(double)(1.0- ((gris[i]/valor4)) )*100;
     aux=(unsigned int)(valor3);
     opacidad[i][k]=(char)aux;
    }
}
```