
# Coherencia de objetos, coherencia de rayos y paralelismo, en la aceleración del trazado de rayos.

TESIS DOCTORAL
PRESENTADA AL DEPARTAMENTO DE
LENGUAJES Y SISTEMAS INFORMÁTICOS E INGENIERÍA DEL SOFTWARE
PARA LA OBTENCIÓN DEL TÍTULO DE
DOCTOR EN INFORMÁTICA

PRESENTADA POR

Pascual González López

DIRIGIDA POR

DR. FRANCISCO GISBERT CANTÓ

Madrid, 1999

# Dedicatoria

A toda mi familia y especialmente a mi mujer Carmen y a mis hijos Enrique, Javier y Cristina.

# Agradecimientos

Si cuando se comenzó este trabajo el instante de redactar estas líneas era tan lejano y remoto que uno pensaba que no se le ocurriría que escribir. Llegado el momento han sucedido tantas cosas, ha habido momentos de zozobra e instantes de duda que no habrían podido salvarse sin el apoyo, aunque a veces fuese testimonial, de algunas personas que me ayudaron a terminar lo que un día comencé con mucho entusiasmo.

En primer lugar debo recordar a Paco Gisbert, mi tutor, que siempre me dio ánimos y sugerencias que han permitido concluir este trabajo. Pero junto a él debo recordar a otros investigadores que trabajan dentro de la informática gráfica, con los que tuve el placer de impartir una escuela de verano en 1997 (J. Carlos Torres, Xavier Pueyo, Pere Brunet, Roberto Vivó, Francisco Serón, Dolors Ayala, Alvar Vinacua, Isabel Navazo, Daniela Tost, Ramón Mas, Alejandro García-Alonso) y que en un instante de desanimo fueron capaces de hacer que no desfalleciera y continuara con el trabajo comenzado. Pero entre todos ellos resaltar la ayuda prestada por Juan Carlos Torres que no sólo me apoyó para que continuara mi trabajo sino que dedicó algo de su tiempo a escuchar mis propuestas y sugerir algunas alternativas o nuevos enfoques al trabajo que estaba realizando.

También quiero agradecer el apoyo prestado por el Departamento de Informática de la Universidad de Castilla-La Mancha y en especial el del grupo de investigación "Redes y Arquitecturas de Altas Prestaciones" (RAAP), en el que me he integrado con el objetivo de diseñar algoritmos paralelos que puedan ser

utilizados como benchmark para evaluar las nuevas propuestas arquitectónicas que surjan dentro de los trabajos de investigación que se están llevando a cabo. Junto a las personas que me han apoyado con alguna sugerencia y muchos ánimos, quiero agradecer especialmente la colaboración a Francisco J. Alfaro, José P. Molina y Eduardo Collado, cuyas aportaciones han facilitado que ahora me encuentre redactando estas líneas.

Pero tal vez, los pilares fundamentales que han permitido concluir el trabajo un día comenzado debo buscarlos en el seno de mi familia que aunque a veces no comprendían la importancia y la dedicación que este trabajo requería siempre han estado ahí. Por ello quiero recordar especialmente a mis padres y mi tía a los que ya podré ver sin necesidad de salir corriendo para seguir con el trabajo, y como no a Carmen y a mis hijos Enrique, Javier y Cristina que pensaban que esto nunca iba a terminar y con los que podré recuperar los fines de semana como tiempo de diversión y no de trabajo.

# Índice general

# Índice de Figuras

# Índice de Gráficos

# Índice de Tablas

# Motivación.

Uno de los campos de investigación más activos en la actualidad está relacionado con la generación de imágenes con alto grado de realismo a partir de escenas sintéticas existentes sólo dentro de un mundo simulado, que en algunos casos pretende simular al mundo real pero que en otros casos tiene como objetivo crear mundos irreales. En la actualidad la aplicación de técnicas de síntesis de imágenes (*rendering*) es algo muy importante en multitud de áreas de trabajo. El ámbito de aplicación va desde la publicidad o el entretenimiento, hasta el diseño industrial o arquitectónico, pasando por la educación o el entrenamiento de expertos en el manejo de ciertos aparatos. En definitiva, podemos ver que este tipo de técnicas se están acercando cada vez más a los usuarios finales de los diferentes ámbitos de actuación de la informática gráfica.

Este creciente interés por las técnicas que obtienen imágenes de alta calidad (foto-realistas) se puede encuadrar en dos grandes corrientes de investigación: búsqueda de algoritmos que incorporen nuevos efectos que mejoren la calidad de la imagen final y optimización de los algoritmos existentes para que se reduzca el tiempo necesario para procesar una escena. Dentro de estas dos grandes alternativas la Tesis Doctoral que aquí se presenta se encuadra en la segunda corriente,

planteándose como meta principal conseguir reducir el tiempo total de procesado de una escena cualquiera. En nuestro caso, dentro de las diferentes técnicas que permiten generar imágenes de alta calidad se ha seleccionado el algoritmo de trazado de rayos (*Ray Tracing*). Su elección viene motivada por su amplia utilización no sólo en ámbitos científicos sino también dentro del entorno de las aplicaciones comerciales existentes actualmente.

El principal problema de este algoritmo es el elevado coste computacional que posee, debido especialmente al cálculo de la intersección rayo-objeto [WHIT80]. Este elevado coste computacional ha motivado que un gran número de investigadores se hayan dedicado a definir alternativas de optimización que permitan reducir dicho coste. En la definición de dichas alternativas la mayoría de los investigadores se han apoyado en las características de coherencia que el trazado de rayos exhibe y en las posibilidades de paralelismo que éste ofrece, al poder considerar el análisis del color de un punto de la pantalla (pixel) independiente del realizado para el resto de ellos.

Estas dos fuentes principales de aceleración serán muy tenidas en cuenta dentro de este trabajo, intentando definir una solución que obtenga una mayor optimización al integrar diferentes alternativas individuales presentadas previamente.

Por otra parte, el algoritmo de trazado de rayos se ha convertido en una técnica tan utilizada a la hora de analizar el comportamiento de una determinada arquitectura hardware que ha hecho que éste se convierta en uno de los Benchmark [KEAT94] (test de prueba) más utilizados para analizar su eficiencia. Esto ha motivado que el trabajo que aquí se presenta pueda considerarse el comienzo de una nueva labor investigadora encuadrada dentro del grupo de investigación de *"Redes y Arquitecturas de Altas Prestaciones"* (*RAAP*) de la Universidad de Castilla-La Mancha y que a la vez que permite evaluar las prestaciones de un nuevo diseño arquitectónico, consigue generar una solución más eficiente en el ámbito de la

síntesis de imágenes tridimensionales. Para ello, dentro de la investigación del grupo *RAAP*, este trabajo cuenta con la subvención de la Comisión Interministerial de Ciencia y Tecnología, CICYT, bajo el contrato TIC97-0897-C04-02.

## *Objetivos de la Tesis Doctoral.*

Dentro de la línea de investigación anteriormente comentada, esta Tesis Doctoral se ha propuesto como principal objetivo: estudiar los principales métodos de aceleración presentados hasta este momento y proponer una nueva alternativa que, teniendo presentes los beneficios esperados de la aplicación individual de cada una de ellas, ofrezca además nuevas ventajas fruto de la unión, en una alternativa conjunta, de dichas propuestas individuales. En particular, los objetivos de específicos de la misma son los siguientes:

1. Estudiar las principales ideas utilizadas en la aceleración del trazado de rayos y, a partir de ellas, plantear una nueva propuesta que integre en un nuevo algoritmo varias de ellas.

2. Definir y desarrollar un nuevo algoritmo que permite acelerar el trazado de rayos, apoyándose para ello en las características de coherencia que el trazado de rayos exhibe.

3. Analizar el comportamiento individual del nuevo algoritmo y compararlo con otras propuestas previas, para de este modo poder apreciar mejor las bondades nuestro algoritmo.

4. Estudiar las posibilidades de paralelización del algoritmo secuencial propuesto anteriormente y diseñar una nueva versión paralela del mismo.

## *Desarrollo de la Tesis Doctoral.*

Para cubrir estos objetivos, esta memoria se ha estructurado en ocho capítulos y dos anexos:

El **capítulo 1**, pretende encuadrar el problema de la síntesis de imágenes desde una perspectiva amplia, para finalmente centrarse en las alternativas básicas para la obtención de imágenes realistas o foto-realistas. En él, tras una pequeña introducción, se describirán las técnicas clásicas dentro del ámbito de la visualización realista (trazado de rayos, radiosidad y métodos híbridos).

El **capítulo 2**, muestra las principales líneas de optimización del trazado de rayos. En este caso ante la imposibilidad de describir, aunque fuese de manera somera, todas las alternativas presentadas hasta el momento, debido a la abundantísima bibliografía existente, se ha optado por buscar las fuentes básicas de dichas optimizaciones, realizando una tarea de clasificación de los diferentes trabajos realizados hasta el momento. Por ello, se han presentado las principales propuestas encuadradas dentro de la aplicación de la idea de coherencia y del aprovechamiento del paralelismo que el trazado de rayos exhibe. Se ha prestado especial atención al estudio de las propuestas previamente realizadas sobre las bases en las que se apoya la optimización presentada en esta Tesis: coherencia de objetos, coherencia de rayos y paralelismo.

El **capítulo 3**, presenta las principales ideas del algoritmo de trazado de haces que aquí se expone. En él se explican de manera detallada las diferentes tareas que componen el trazado de haces: obtención de la estructura de subdivisión; creación del haz inicial; generación de la trayectoria de un haz dentro de la estructura de árbol octal y análisis de los pixel asociados a un haz.

El **capítulo 4**, muestra las características básicas del modelo OO asociado al trazado de haces. Presenta la estructura básica de la librería utilizada como base

(OORT) y las modificaciones necesarias para incluir las clases y servicios básicos del trazado de haces.

El **capítulo 5**, se encarga de mostrar los resultados de la evaluación del trazado de haces. Para ello, tras describir las escenas utilizadas como casos de prueba, este capítulo se ha dividido en tres grandes apartados: análisis individual del trazado de haces; comparación del comportamiento del trazado de haces con respecto a otras técnicas que utilizan algoritmos de trazado individual de rayos sobre árboles octales; y, por último, comparación del trazado de haces con técnicas que utilizan otras estructuras de descomposición del espacio.

El **capítulo 6**, expone un nuevo algoritmo, en este caso una versión paralela del trazado de haces. Muestra en primer lugar diferentes alternativas de diseño de un algoritmo paralelo, para, a partir de ellas, presentar el algoritmo elegido. Tras su diseño se describe la implementación de los principales servicios encargados de la comunicación entre los diferentes procesos. Finalmente se realiza una evaluación de los resultados obtenidos por este algoritmo.

El **capítulo 7**, es el encargado de recoger las principales conclusiones de este estudio y de proponer nuevas líneas de trabajo a las que las ideas presentadas en esta Tesis pueden dar lugar.

El **capítulo 8**, recoge las diferentes referencias bibliográficas utilizadas durante este trabajo. Junto a esta recopilación, se incluyen una serie de direcciones de Internet muy útiles para aquellos investigadores interesados en profundizar en el estudio de las diferentes propuestas sobre síntesis de imágenes en general y trazado de rayos en particular.

Finalmente, en los dos **anexos** últimos, se presentan de manera más detallada tanto el lenguaje utilizado para describir las diferentes escenas (NFF), como el lenguaje (MPI) que ofrece los diferentes mecanismos de comunicación utilizados dentro de la implementación del algoritmo paralelo.

# Capítulo 1. Visualización realista.

## 1.1 Introducción.

Desde sus comienzos, una de las metas perseguidas por los investigadores que estudiaban la generación de imágenes sintéticas a partir de un ordenador ha sido la obtención de imágenes de alto realismo. Aunque no existe unanimidad a la hora de definir lo que se entiende por imagen realista [HAGE86], en nuestro trabajo vamos a referirnos a ella como aquella imagen que es capaz de captar o representar la mayoría de los efectos que la luz provoca al interaccionar con los objetos de una escena. Dentro de este continuo acercamiento a la realidad podemos intentar plasmarla como una visión fotográfica de la misma, pudiendo hablar entonces de imágenes 'foto-realistas'. Estas imágenes pretenden sintetizar las intensidades de luz que pueden recogerse dentro de la cámara en el plano de la película asociadas a las emisiones lumínicas de los objetos de la escena.

En todo caso, debemos tener en mente que no necesariamente es más útil o deseable una imagen con un mayor nivel de realismo. Por una parte el realismo implica un mayor tiempo de cálculo y por tanto no siempre es necesario que la

imagen refleje todos los posibles efectos del mundo real (sombreados, reflexiones, etc.). Por otra parte, algunas veces, los diseñadores o creadores de las mismas desean obtener imágenes imposibles en la realidad, realizando alteraciones intencionadas sobre el comportamiento real de la luz, los objetos, etc.

La consecución de imágenes realistas no es un proceso simple ya que el fenómeno que se quiere modelar es de naturaleza compleja. Nuestro entorno está rodeado de objetos que poseen ciertas características, como pequeñas alteraciones de su superficie, gradaciones de color, características lumínicas de reflexión y refracción, etc. Estas características del proceso complican su simulación en un ordenador y, en el mejor de los casos, implican un elevado coste computacional, o lo que es lo mismo un tiempo de simulación, en algunos casos, excesivo (algunas imágenes pueden necesitar minutos e incluso horas para su generación).

Por tanto, una submeta, tal vez más fácil de alcanzar, en la búsqueda del realismo es la de obtener información suficiente como para permitir al observador comprender las relaciones espaciales en 3D de los objetos de la escena. Esta submeta puede alcanzarse en la mayoría de los casos a través de un coste computacional más bajo. Una simple visión en perspectiva que oculte las líneas no visibles puede ser suficiente para percibir la estructura de la escena, la dimensión o la posición de cada uno de los objetos dentro de la misma. En este caso, no es necesario representar las características de las superficies de los objetos o las sombras que éstos producen. En efecto, en algunos contextos, la inclusión de mayores detalles puede distraer la atención del observador de la información realmente importante.

Una dificultad asociada a la representación de relaciones espaciales es que la mayoría de los dispositivos de visualización son bidimensionales. Por tanto, los objetos dentro de una escena 3D deben proyectarse a 2D, con considerable pérdida de información, que en algunos casos puede crear ambigüedad en la imagen obtenida. En estos casos debemos incluir efectos que se pueden encontrar en nuestro

entorno visual y que pueden ayudar a que los mecanismos de la percepción humana resuelvan dichas ambigüedades e interpreten la escena correctamente. Aunque, como hemos visto anteriormente en algunos casos, bien por restricciones de tiempo en su visualización o bien por deseo expreso de sus creadores, las imágenes no plasman la realidad con gran precisión.

Actualmente la creación de imágenes realistas es algo básico para multitud de campos. Así podemos encontrar aplicaciones de estas técnicas dentro del diseño industrial y arquitectónico, en las creaciones publicitarias y de entretenimiento (juegos, películas, etc.), en la investigación, la educación o la formación y en el entrenamiento de expertos en el manejo de ciertos aparatos (simuladores de vuelo, de navegación, etc).

En la mayoría de los casos se pretende crear un prototipo de la realidad a un coste reducido que permita simular ciertos aspectos de la misma, haciendo que el espectador o usuario del sistema piense que lo que está percibiendo es algo real y tangible, aun en el caso de que dicho objeto todavía no exista en la realidad.

Dentro del proceso de visualización de imágenes en 3D podemos establecer dos objetivos o submetas: (1) mostrar las relaciones de profundidad de las escenas y objetos 3D en una superficie 2D; (2) mostrar relaciones de sombreado y otros efectos lumínicos.

El primer objetivo pretende plasmar las relaciones existentes entre los objetos de una escena tridimensional mediante la realización de las siguientes tareas:

- Especificación del tipo de proyección: En general una proyección transforma puntos de un sistema de coordenadas de dimensión *n* a otro de dimensión menor. En nuestro caso las transformaciones son de un sistema de coordenadas en 3D a otro en 2D. Estas son proyecciones geométricas planas ya que la proyección se realiza sobre un plano en vez de sobre una superficie

curva y los proyectores son rectos en vez de curvos. Podemos dividirlas en dos clases básicas: en perspectiva y paralelas. Estas dos clases se diferencian en la relación existente entre el centro de proyección y el plano de proyección. Si la distancia entre ambos es finita entonces podemos hablar de proyección en perspectiva. Por contra, si la distancia se considera infinita los proyectores son paralelos y estamos en el otro caso de proyección.

• Especificación de los parámetros de la vista: En este caso establecemos la posición de los ojos del observador y la localización del plano de la imagen (superficie sobre la cual se va a visualizar la escena). Se establecen así dos sistemas de coordenadas, el asociado a la escena y el asociado al observador.

• Seleccionar y recortar (clipping) el área de la escena visible por el observador (volumen visible): Esta tarea es compleja y pretende obtener un proceso normalizado que transforme un "volumen visible" arbitrario, definido mediante proyección en perspectiva o paralela, en uno denominado canónico que facilite su manipulación para determinar el área visible. Con ello se pretende establecer el conjunto de puntos 3D que deben ser transformados para definir la imagen 2D.

• Proyectar y visualizar: Finalmente el contenido de la proyección del volumen visible en el plano de proyección, llamado la ventana del mundo real, se transforma en la región de visualización de la pantalla (viewport) para ser finalmente representado en el ordenador.

Con esta primera aproximación, el usuario de este tipo de sistemas puede comprender la situación de los objetos en la escena pero es consciente de que se trata de algo que no es real, sino tan sólo de un proceso que pretende simular dicha realidad. En cualquier caso, este tipo de técnicas ofrece la suficiente información para que el usuario pueda interpretar la escena o el objeto con la precisión y detalle

suficientes como para poder realizar una evaluación de lo que en ella se representa. La rapidez con que se pueden realizar modificaciones en los objetos hace que este tipo de técnicas sean muy útiles en las primeras etapas del diseño o en la creación de un objeto o escena. Por otra parte, aun en el caso de que se les añada color, los objetos no dejan de ser visiones planas en las que no es fácil añadir gradaciones de color que permitan plasmar los efectos de profundidad o de volumen. Esta escasa calidad de las imágenes obtenidas hace que no sean válidas como representación final de una escena u objeto, pues, en la mayoría de los casos, esta imagen final debe tener una apariencia que pueda confundirse con una fotografía, o con una escena obtenida en una cámara de vídeo.

Para llegar a este nivel de realismo las técnicas utilizadas deben intentar alcanzar la segunda submeta que pretende plasmar efectos como sombreado, que permitan percibir mejor la sensación de profundidad o de volumen, y otros efectos asociados a las condiciones lumínicas de la escena y de los objetos de la misma.

En este caso podemos llegar a obtener imágenes muy parecidas a las que se consiguen mediante una fotografía o una cámara de vídeo, aunque cuanto mayor sea su parecido a las imágenes reales mayor será el tiempo de procesamiento necesario para obtenerlas. Por tanto, es aquí donde se encuentran las principales líneas de investigación en el campo de la creación de imágenes sintéticas a partir de modelos 3D (rendering) las cuales pretenden por una parte aumentar el grado de realismo y por otra reducir el tiempo necesario para la obtención de una imagen.

La complejidad y tiempo de computación de las técnicas que pueden utilizarse en este caso va aumentando al mismo tiempo que lo hace la calidad de la imagen final. Así podemos hablar, entre otras, de las siguientes técnicas:

- Determinación de superficies visibles: Permiten visualizar solamente aquellas partes de la superficie de los objetos que son visibles por el observador. Esta tarea es básica para que tengan sentido las escenas que

vamos a representar.

• Sombreado e iluminación para polígonos: Este tipo de técnicas permiten plasmar efectos básicos de sombreado e iluminación sobre objetos representados mediante polígonos o mallas poligonales. Estas técnicas permiten obtener imágenes de calidad con una rapidez suficiente como para que muchos sistemas de visualización tridimensional los utilicen. Pero, como apuntan J. Foley et al [FOLE90], tienen algunos problemas que hacen que no sea la solución definitiva al problema de la visualización foto-realista, ya que son incapaces de tratar efectos como reflexiones, refracciones, etc.

• Representación de texturas y otras características de las superficies de los objetos: Para conseguir un mayor realismo debemos tener en cuenta ciertas características de la superficie de los objetos que permiten plasmar perturbaciones de la superficie (bump mapping) o características de coloración de la misma (texture mapping). El primero de los efectos permite simular las rugosidades o imperfecciones de las superficies de los objetos. El segundo posibilita la asignación de patrones de color a la superficie de los objetos simulando coloraciones asociadas a objetos reales, como por ejemplo madera, mármol, etc., o incorporando la posibilidad de establecer nuevos patrones que faciliten la representación de objetos de superficie uniforme pero de coloración no constante como por ejemplo un tablero de ajedrez, una caja forrada con un papel en el que aparece una figura, etc.

• Modelos de iluminación y sombreado globales: En este apartado agrupamos aquellas técnicas que obtienen imágenes de mayor calidad al tener en cuenta los efectos que provocan las interacciones existentes entre las superficies de los objetos de la escena, es decir, los efectos de la luz indirecta. En este tipo de modelos para calcular la luz de un punto se debe tener en cuenta no sólo la luz emitida por las fuentes de luz, sino también aquella que alcanza dicho

punto después de reflejarse o transmitirse a través de los distintos objetos de la escena. Para la manipulación de la iluminación global se han definido dos clases de algoritmos: 'trazado de rayos' (ray-tracing) y 'radiosidad' (radiosity). En el primero de los algoritmos las tareas de determinación de las superficies visibles y el sombreado se encuentran unidas. En cambio el segundo tipo de algoritmos separa completamente los procesos de sombreado y determinación de las superficies visibles.

Estos dos últimos algoritmos se verán en mayor detalle en la siguiente sección. En ella se realizará una descripción muy somera de cada uno de ellos y se intentará plasmar las diferencias esenciales entre ambos algoritmos.

## 1.2 Métodos de alto nivel de realismo.

En el mundo real que deseamos modelar, a las superficies de los objetos llegan rayos luminosos tanto directamente desde los focos o fuentes de luz, como indirectamente tras interaccionar con otras superficies de la escena. Por tanto, para poder conseguir imágenes de gran calidad debemos utilizar técnicas que simulen lo mas correctamente dichos efectos (métodos basados en ciertos modelos de la física).

El transporte de la luz está gobernado por un caso especial de la ecuación de Boltzmann [GLAS95] obtenida de la teoría del transporte, en la cual se estudia cómo una distribución estadística de partículas (como pueden ser los fotones) fluyen a través de un entorno. En los trabajos de J. Arvo [ARVO93], A. Glassner [GLAS95] y M. Cohen y J.R.Wallace [COHE93] se puede encontrar un análisis más profundo de la *teoría del transporte de la luz* y su aplicación a la informática gráfica, y el trabajo de R. Siegel y J. Howell [SIEG92] ofrece un tratamiento más detallado desde la perspectiva de la física. La importancia de estas bases teóricas nos obligan a realizar un pequeño repaso a la denominada ecuación de visualización ("rendering

equation").

El flujo de las partículas de luz (fotones) en un entorno estable, rápidamente alcanza el equilibrio. Por tanto, el número de fotones que fluyen a través de una región del espacio en un rango de direcciones dado, debe ser constante a través del tiempo. Pero en vez de realizar un análisis en función de los fotones vamos a realizar el estudio en términos de *radianza*, es decir, del flujo saliente de cada punto de las superficies para una longitud de onda dada.

Para simplificar, vamos a suponer que el entorno está formado por superficies continuas, en las cuales se ignoran los efectos provocados por la fluorescencia y fosforescencia, a la vez que se considera que los fotones se transmiten a través del vacío. La eliminación de la fluorescencia permite asumir que las soluciones obtenidas para una longitud de onda $\lambda$ son independientes de las obtenidas para otra longitud de onda $\lambda'$. Por otra parte, si eliminamos la fosforescencia asumimos que la solución obtenida es independiente del instante de tiempo elegido para su generación. Finalmente, tampoco tenemos en consideración la posible dispersión de las partículas debido al efecto de los gases presentes entre las superficies, es decir consideramos que el proceso se desarrolla en el vacío. Al realizar todas estas adaptaciones la **ecuación completa de la radianza** definida por A. Glassner [GLAS95] se simplifica enormemente y se asemeja bastante a la definida por J.T. Kajiya [KAJI86] que éste denomina "***rendering equation***" (ecuación de la síntesis de imágenes) y que modela el comportamiento de la radianza en entornos en los que se han asumido las simplificaciones anteriores.

La radianza describe la densidad de energía transmitida por un fotón que parte de un punto de la escena y se propaga en una dirección dada. También es posible definirla como la energía emitida por unidad de área y por unidad de ángulo sólido y su unidad es $W/m^2\text{-}sr$ (siendo "*sr*" la unidad de ángulo sólido).

Para llegar a la ecuación 1.1 vamos a partir de una serie de definiciones previas. Supongamos que nuestra escena está formada por un conjunto de superficies $S \subset \Re^3$, de tal forma que para cada punto $p$ de $S$ existe un vector normal a la superficie en dicho punto. Además se asume que las superficies son continuas y diferenciables, lo que permite definir $dA(p)$ como el área diferencial de superficie alrededor de un punto $p$. Sea $L_\lambda(p,\omega)$ la radianza total (emitida más reflejada) desde el punto $p$ en la dirección $\omega$. A su vez denominaremos $L_{e\lambda}$ a una función con las mismas caracterísitcas que $L_\lambda$, pero que expresa la parte de $L_\lambda$ debida a la emisión del punto $p$ de la superficie. Por tanto podemos decir que la radianza saliente $L_\lambda$ (ecuación 1.1) se obtiene como la suma de la radianza emitida $(L_{e\lambda})$ más la reflejada en cada punto $q$ de la superficie proveniente del resto de puntos del entorno.

$$L_\lambda(p,\omega) = L_{e\lambda}(p,\omega) + \int_{q \in S} K_\lambda(p,q,\omega) L_\lambda(q,\omega_{pq}) dA(q) \qquad (1.1)$$

donde $\omega_{pq}$ es el vector director de la recta que une los puntos $q$ y $p$. Igualmente $K_\lambda(p,q,\omega)$ es una función que modela la fracción de la radianza reflejada en $p$ en la dirección $\omega$, debida a la radianza incidente en $p$ que proviene de $q$. Al mismo tiempo, $K_\lambda(p,q,\omega)$ puede expresarse como: $f_r(p,\omega, \omega_{pq})\ G(p,q)$, donde $f_r$ es conocida como la función bidireccional de distribución de la reflectancia *(BRDF Bidirectional reflection distribution function)* y $G$ es un término puramente geométrico que depende básicamente de los puntos $p$, $q$ y de las superficies $S$.

La meta de las diferentes técnicas de síntesis de imágenes es encontrar una función que satisfaga la ecuación de la radianza en su versión completa o simplificada [GLAS95]. Esto generalmente implica situar la superficie que contendrá la imagen dentro de la escena. El resultado de la síntesis es una imagen que representa la luz que llega a dicha superficie.

Existen dos formas típicas de resolver la ecuación de la radianza (1.1), a las que podemos denominar, siguiendo la definición de Glassner [GLAS95],

***aproximación explícita*** y ***muestreo implícito***.

En la *aproximación explícita* se pretende encontrar una función explícita que se aproxime a la distribución de la radianza y posteriormente, a partir de ella, calcular los valores de la función en el plano imagen. Esta aproximación está ejemplificada por los *algoritmos denominados de radiosidad (radiosity)*. En este caso al calcular la función de la radianza para cualquier punto de la escena, podemos mover el plano imagen y obtener la imagen en él representada sin necesidad de recalcular la función. Dicha función se dice que es una solución independiente de la vista y por tanto a los algoritmos que la implementan se les denomina algoritmos independientes de la vista. Este tipo de algoritmos modela eficientemente los fenómenos difusos, pero tiene un peor comportamiento a la hora de operar con fenómenos de tipo especular.

En el *muestreo implícito* deseamos encontrar el valor de la radianza tan solo en un conjunto de puntos (muestras) asociados con al plano imagen, condicionando dichos cálculos por la geometría de la vista. Esta aproximación se denomina "dependiente de la vista" y está representada por los *algoritmos de trazado de rayos* (ray tracing). El hecho de que el proceso de generación de la imagen dependa de la situación del observador facilita la simulación de ciertos efectos lumínicos de tipo especular. En cambio, la simulación de efectos difusos, que pueden considerarse estables mientras no cambie la escena, requieren trabajo extra.

Las características de ambos métodos en vez de hacerlos excluyentes parece hacerlos complementarios. La utilización conjunta de ambos se refleja en numerosos algoritmos que se encuadran dentro de los denominados "algoritmos híbridos".

En las siguientes secciones realizaremos un pequeño repaso de las ideas básicas de dichos métodos.

### 1.2.1 Trazado de rayos.

Las ideas que subyacen a este algoritmo no son nuevas y para determinar sus orígenes podríamos remontarnos, como hace G. Hofmann [HOFM90], a algunos de los estudios que se realizaron durante el Renacimiento europeo. En cualquier caso las fuentes de inspiración pueden encontrarse en algunos trabajos de física en los que se estudian los efectos de la luz sobre objetos, lentes, etc. En ellos se analizaban los caminos que seguían dentro de la escena los rayos que partían de las fuentes de luz. A este proceso de seguimiento de rayos luminosos se le denominó 'trazado de rayos' o *ray-tracing*.

En nuestro caso este algoritmo no solo permite determinar los efectos que la luz provoca en una escena sino que también permite realizar una proyección en dos dimensiones de escenas tridimensionales.

En 1968 A. Appel [APPE68] publica un artículo en el que se presenta la idea de realizar un seguimiento de los rayos de luz para determinar las sombras producidas por los objetos de una escena, considerándose la primera referencia a este método dentro del campo de los gráficos por ordenador. Los trabajos posteriores de T. Whitted [WHIT80] sobre modelos de iluminación permitieron extender el algoritmo trazador de rayos incluyendo los efectos de reflexión y refracción especular. Este estudio propone un algoritmo recursivo que realiza un seguimiento de los rayos reflejados y transmitidos a que da lugar un rayo primario al interaccionar con los objetos de la escena. Los rayos que se van generando durante el proceso se almacenan en una estructura jerárquica, *árbol de rayos*. Finalmente para determinar el color de un rayo primario se recorre en sentido inverso el árbol de rayos y se pondera la participación del color de cada rayo en el color final del rayo primario.

El objetivo de este algoritmo es calcular el color de cada punto de la pantalla (pixel). Éste vendrá determinado básicamente por los colores de los rayos de luz que

inciden o afectan a dicho punto, por lo cual debemos determinar el camino seguido por cada rayo de luz, considerando un rayo de luz como un estrecho camino seguido por una partícula de luz (fotón) dentro de la escena a tratar.

La estrategia más natural, si deseamos simular el efecto real, es suponer que las fuentes de luz emiten un gran número de rayos luminosos (fotones) y lo que debemos hacer es realizar un seguimiento de los distintos rayos desde la fuente de luz hasta que alcancen al observador o se salgan fuera de los límites de la escena, como se indica en la figura 1.1. A este tratamiento se le denominó "*forward ray tracing*"[GLAS89] o también "*photon tracing*" [GLAS95].

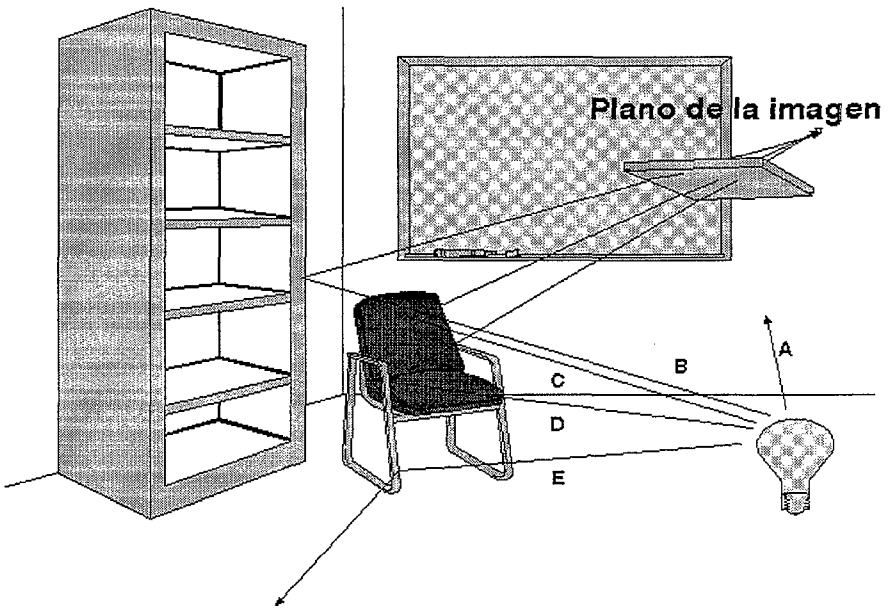



**Figura 1.1 Esquema general del trazado de una partícula de luz.**

Como podemos apreciar en la figura 1.1, el problema que se nos presenta utilizando este algoritmo es que realizamos el seguimiento de un número de rayos que nunca alcanzarán al observador (en la figura los rayos A y E se salen de la escena sin pasar por el plano de la imagen), utilizando, por tanto, parte del tiempo en seguir rayos "inútiles" desde el punto de vista de la imagen a generar. Aunque es

menos frecuente encontrar algoritmos que se apoyen en esta idea, el primero de los trabajos sobre el trazado de rayos, publicado por A. Appel [APPE68], se basaba en este tipo de trazado. Más recientemente han surgido otros trabajos ([PATT92], [PATT93], [DUTR95], [HEIR97], [UREÑ98]...) que se engloban dentro los métodos denominados *"Particle Tracing"* (trazado de partículas). Estos métodos se basan en la aplicación de las técnicas de simulación de Monte Carlo para resolver directamente la ecuación de visualización (1.1). En este modelo la luz es tratada como un conjunto de partículas que son enviadas desde los focos luminosos hacia la escena. La dirección con la cual dichas partículas se envían a la escena se determina estocásticamente intentando emular el comportamiento de los focos luminosos. Tras ello, una vez que se emite una partícula el tratamiento es semejante al descrito anteriormente y que ha sido denominado como "*forward ray tracing*".

Para evitar este problema y centrar nuestra atención tan solo en los rayos útiles para la generación de la escena, en la literatura se propone un algoritmo alternativo, que se puede llamar "*backward ray tracing*" [GLAS89] o "*visibility tracing*" [GLAS95] pero al que generalmente se le conoce simplemente como *"ray tracing*". Este nuevo algoritmo realiza un seguimiento de los rayos en sentido inverso, partiendo por tanto del observador, para posteriormente calcular las posibles intersecciones de éstos con los objetos de la escena y finalmente determinar cuáles de ellos alcanzan una fuente de luz. De este modo, los rayos estudiados serán tan sólo los que el observador percibe y por tanto los relevantes para la generación de la escena.

El planteamiento es, por tanto, determinar el color de la luz que viene desde la escena a través de los rayos trazados desde el observador. Pero dicho color no viene dado tan solo por el color del objeto con el que choca el rayo lanzado desde el observador (a este tipo de rayos los denominaremos a partir de *ahora rayos primarios*), sino que deben tenerse en cuenta los efectos de reflexión o refracción que puedan provocarse al chocar el rayo con la superficie del objeto e incluso efectos de sombreado provocados por otros objetos que hacen que las fuentes de luz sean no

visibles desde el punto en consideración. De este modo, dentro del proceso de trazado surgen nuevos rayos producto de reflexiones o refracciones a los que denominaremos *rayos secundarios*. Por último, la determinación del color debe tener en cuenta los efectos de las luces sobre los objetos surgiendo el último tipo de rayos denominado *rayos de sombra*.

Por tanto, para llegar a obtener el color final del punto hay que realizar dos tipos de estudios: uno geométrico y otro colorimétrico. El primero nos permite determinar el punto de intersección entre una superficie y un rayo, y a la vez calcular las direcciones de salida de los diferentes rayos secundarios y de sombra. Por otra parte, el estudio colorimétrico, nos permite determinar como inciden los diferentes rayos en el color final a obtener. El punto básico para llevar a cabo ambos tipos de estudios se apoya en la determinación del vector normal a la superficie en el punto en el que se produce la intersección, vector que determinará, en cierta medida, las características colorimétricas y la dirección de los nuevos rayos generados.

Para conseguir una simulación fiel a la realidad, nuestro modelo debe contemplar los efectos físicos básicos que se pueden producir al chocar un rayo de luz con un objeto: reflexión y transmisión, tanto especular como difusa, efectos que, en cualquier caso, dependerán de las características de la superficie del objeto. Junto a estos efectos, derivados de la intersección de un rayo de luz con un objeto, es interesante que este tipo de aplicaciones trate también de forma integrada el sombreado.

El algoritmo trazador de rayos, para simular de manera lo más precisa posible esta compleja realidad, se apoya generalmente en la utilización de cuatro tipos de rayos básicos: (1) rayos primarios; (2) rayos de sombra; (3) rayos reflejados; y (4) rayos transmitidos.

Los rayos primarios son aquellos que parten del observador hacia la escena y

a partir de los cuales se derivarán el resto de rayos necesarios para aplicar el modelo de iluminación y obtener finalmente el color del rayo primario.

Una vez que un rayo primario alcanza la superficie de un objeto de la escena debemos determinar si el punto de intersección dado está en sombra respecto a los distintos focos de luz. Para ello el algoritmo genera tantos rayos de sombra como puntos de luz existan en la escena, dirigiendo cada uno de ellos hacia un foco, comprobando finalmente si alcanzan el foco de luz al que se dirigían. Si dicho rayo de sombra alcanza el foco de luz, entonces el rayo primario se verá influenciado, teniendo en cuenta las características difusas de la superficie del objeto, por la luz proveniente de dicho foco. Pero si encuentra en su camino otro objeto opaco antes de alcanzar el foco luminoso podremos decir que el punto de intersección entre el rayo primario y la superficie del objeto está en sombra respecto a dicho foco y, por tanto, la luz final de dicho rayo primario no se verá afectada por el foco de luz en cuestión.

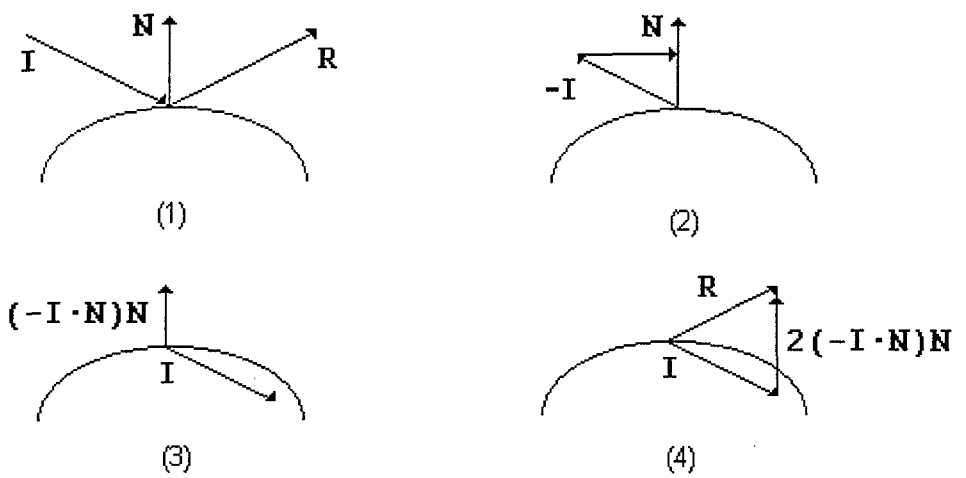
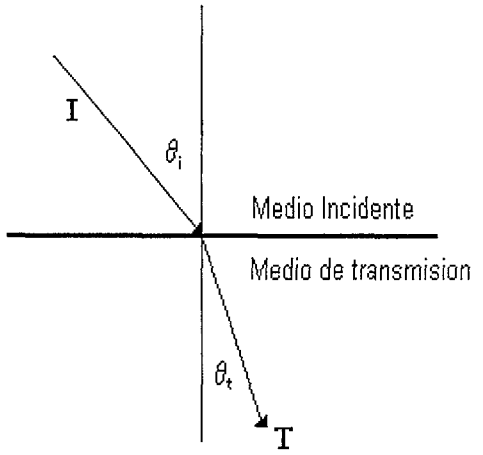



**Figura 1.2 Interpretación geométrica del cálculo de la dirección de reflexión.**

Del mismo modo, una vez que un rayo primario ha impactado con la superficie de un objeto de la escena, dependiendo de las características especulares

de la superficie del objeto podemos crear un nuevo rayo reflejado o transmitido, o incluso uno de cada tipo, los cuales deberán tratarse como si fueran nuevos rayos primarios. Si la superficie es reflectante, el color del punto de intersección del rayo primario y la superficie se verá influenciado por el color que llega desde la dirección de reflexión, dirección hacia donde parte el rayo reflectante. Igualmente, si el objeto tiene características de transmisión, el color final del punto de intersección dependerá en cierta medida del rayo (rayo transmitido) que atraviesa la superficie en la dirección de refracción.




*Ley de Snell*

$$\frac{\text{sen}\theta_i}{\text{sen}\theta_t} = \eta_{ti} = \frac{\eta_t}{\eta_i}, T = \alpha I + \beta N$$

Donde

$\eta_i$ es el índice de refracción del medio incidente con respecto al vacío.

$\eta_t$ es el índice de refracción del medio de transmisión con respecto al vacío.

$\eta_{ii}$ es el índice de refracción del medio incidente con respecto al de transmisión.

**Figura 1.3 Interpretación geométrica del cálculo de la dirección de transmisión.**

Por tanto, finalmente el color del rayo primario se verá influenciado por la combinación de la luz asociada a los distintos rayos emitidos desde el punto de intersección del rayo primario con la superficie. Pero la determinación de la luz asociada a un rayo reflejado o transmitido vendrá dada por el camino inverso

recorrido por cada uno de dichos rayos, siendo necesario estudiar dicho camino como si se trataran de nuevos rayos primarios.

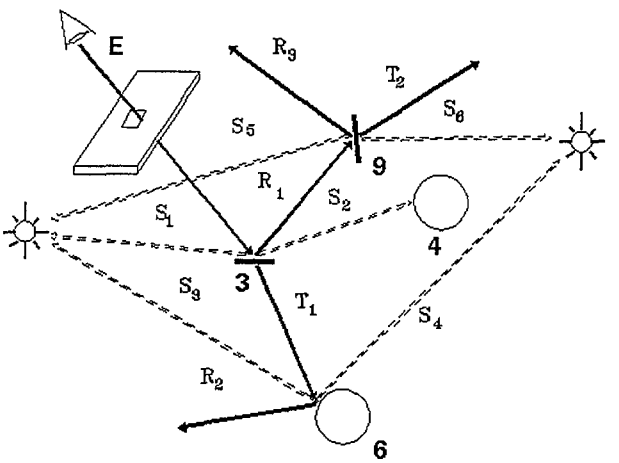



**Figura 1.4 Proceso de obtención del color de un pixel.**

Este proceso de descomposición o generación de nuevos rayos a partir de rayos primarios obliga a un planteamiento recursivo del problema (ver figura 1.4). En este caso el color del rayo primario E vendrá determinado por las características del resto de rayos que aparecen en la figura. De esta forma, suponiendo que el plano 1 pasa un 40% de luz transmitida (rayo $T_1$) y refleja un 20% de luz (rayo $R_1$), podemos decir que $T_1$ contribuye un 40% y $R_1$ un 20% al color de E. Pero a su vez vemos que el rayo $T_1$ se ve reflejado (rayo $R_2$) en la esfera 2, la cual es un 30% reflectante, pudiendo asegurar finalmente que el nuevo rayo $R_2$ contribuirá un 12% en el color final de E.

El proceso anterior puede expresarse utilizando una estructura en árbol (figura 1.5) que nos muestra los distintos rayos generados asociados a un rayo primario dado y que, por tanto, contribuirán al color final de éste. Este proceso de descomposición finalizará cuando todos los rayos secundarios se salgan de la escena, o bien alcancen un foco de luz o una superficie opaca y mate. Sin embargo, este

proceso en escenas con muchos objetos puede ser muy largo haciendo que se consuma mucho tiempo siguiendo y descomponiendo rayos secundarios cuando la incidencia de éstos en el color final del rayo primario puede ser casi insignificante. Por ello en este tipo de escenas con un gran número de objetos se les asocia un nivel máximo de descomposición. De este modo, si se llega a dicho nivel máximo el proceso de traza de rayos para un pixel termina.

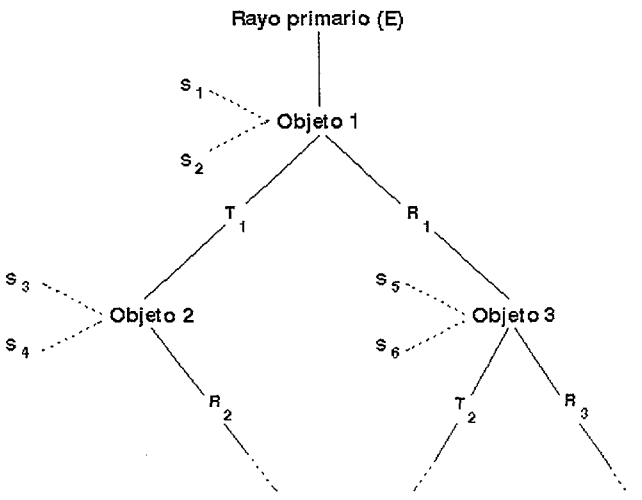



**Figura 1.5 Árbol de descomposición asociado a un rayo primario.**

## 1.2.2 Radiosidad.

Este método es más reciente que el anterior y se basa en los modelos de ingeniería térmica. La teoría en la que se apoya puede encontrarse en la mayoría de los textos que tratan la transferencia de calor mediante la radiación ([SPAR63], [SPAR66], [SIEG92],...). La idea básica que subyace detrás de la radiación es que cuando una parte o la totalidad de un cuerpo está más caliente que los objetos circundantes éste tiende a enfriarse con el tiempo emitiendo energía en todas direcciones. Parte de esta energía es reflejada, transmitida o absorbida. La tasa a la cual la energía abandona una superficie, denominada *radiosidad*, incluye tanto la

energía que emite la superficie como la que refleja o transmite. De este modo aquellas técnicas que calculan las emisiones de energía de las superficies dentro de un entorno se han denominado métodos de *radiosidad* (*radiosity*).

En este caso la emisión y reflexión de radiación elimina la necesidad de utilizar el término de *luz ambiente* aportando un mejor tratamiento de las reflexiones interobjetos. La introducción de esta idea dentro del campo de gráficos es debida los trabajos de Goral, Torrance, Greenberg y Battaile [GORA84] y de Nishita, Nakamae [NISH85]. Estos métodos primero determinan todas las interacciones de la luz en el entorno. Tras esto, crean una o más imágenes con la única sobrecarga de la determinación de las superficies visibles.

En los métodos de radiosidad se permite que las superficies emitan luz y por tanto todas las fuentes luminosas son modeladas de tal manera que tienen asociada un área. Por otra parte, se asume que todas las superficies de los objetos de la escena tienen un comportamiento ideal en lo referente a los fenómenos difusos, es decir, reflejan luz con igual intensidad de radiación en todas las direcciones, expresando la transferencia de energía entre dichas superficies a través de un *factor de forma*. Estas superficies son divididas en pedazos o áreas planares (*patches*) en los cuales se asume que la energía emitida (radiosidad) en cada punto es constante. Para cada patch podemos formular la ecuación de radiosidad como sigue:

$$B_i = E_i + \rho_i \sum_{j=1}^{j=N} F_{ji} B_j \qquad (1.2)$$

donde

$B_i$ = energía emitida (radiosidad) por el *área i*.

$E_i$ = tasa a la cual la luz es emitida desde el *área i*.

$\rho_i$ = cantidad de radiación incidente al *área i* que es reflejada por él (índice de reflexión del área).

$F_{ji}$ = Factor de forma o de configuración, el cual expresa la fracción de

energía que abandona el *área i* y llega al *área j*.

$B_j$ = energía emitida (radiosidad) por el *área j*.

$N$ = número de áreas.

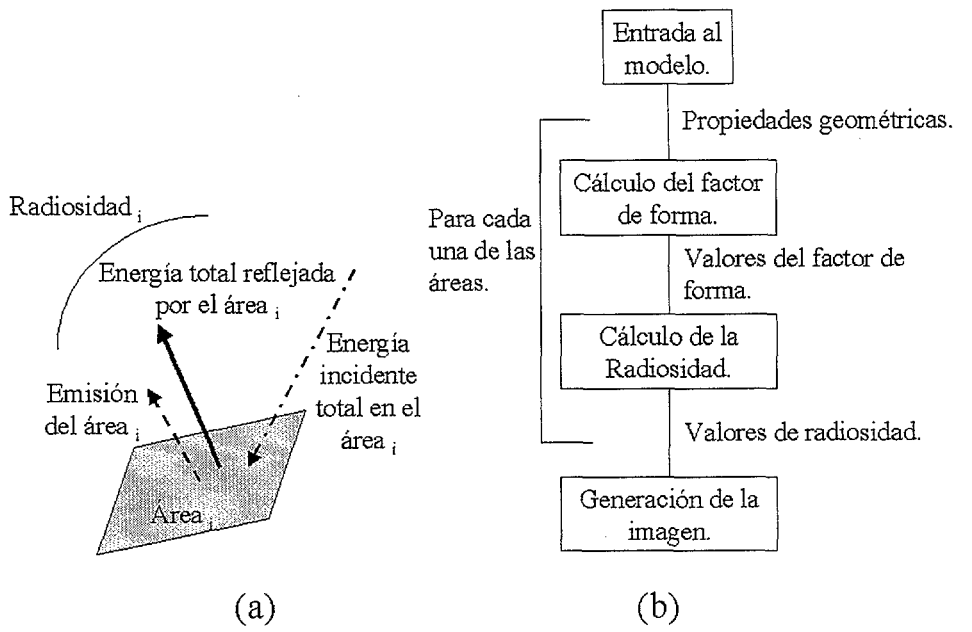



**Figura 1.6 a) Geometría de la radiosidad de una superficie "*i*"; b) Diagrama de flujo del método básico de radiosidad.**

Esta ecuación indica que la energía que abandona una unidad de área (*patch*) de la superficie es igual a la emisión original de dicha área mas la suma de cualquier energía de entrada al área, proveniente del resto de objetos de la escena, que es reflejada por ésta (como aparece en la figura 1.6 a). La luz reflejada se calcula ponderando, mediante el índice de reflexión, la suma de la luz incidente en dicha área. Por último, la luz incidente a un área depende de la energía emitida por cada una de las áreas restantes y del factor de forma, que determinará la fracción de la energía emitida por un área que llega al área que estamos estudiando.

Estudiando la ecuación anterior vemos que la radiosidad de un área *i* depende

de la radiosidad del resto de áreas. De este modo se obtiene un sistema de ecuaciones lineales en el que se tendrán tantas ecuaciones como áreas en las que se hayan descompuesto las superficies de la escena. Por tanto, para la escena completa tenemos un sistema de ecuaciones simultáneas que representa el intercambio de energía a través de múltiples interreflexiones y emisiones en el entorno.

$$\begin{pmatrix} 1-\rho_1F_{11} & -\rho_1F_{21} & \cdots & -\rho_1F_{n1} \\ -\rho_2F_{12} & 1-\rho_2F_{22} & \cdots & -\rho_2F_{n2} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ -\rho_nF_{1n} & -\rho_nF_{2n} & \cdots & 1-\rho_nF_{nn} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_1 \\ \cdot \\ \cdot \\ \cdot \\ E_n \end{pmatrix}$$

**Gráfico 1.1 Ecuaciones del intercambio de energía entre áreas planares (*patches*).**

Esta ecuación (gráfico 1.1) puede resolverse para cada longitud de onda considerada en el modelo luminoso, ya que $\rho_i$ y $E_i$ dependen de la longitud de onda. El factor de forma, sin embargo, es independiente de dicha longitud de onda y puede calcularse solamente en función de la geometría del objeto. Por tanto, no necesitará ser recalculado si las luces o las características de reflexión de las superficies cambian. La tarea de calcular la iluminación global se centra en la resolución del factor de forma entre cada par de áreas, y tras ello en la obtención del resultado del conjunto de ecuaciones lineales que permitirán determinar los valores de la radiosidad para cada una de las áreas. Una vez calculados estos valores las imágenes pueden ser generadas. Un esquema básico de este método se puede observar en la figura 1.6 b.

El modo de resolución de este conjunto de ecuaciones determina diferentes tipos de algoritmos de radiosidad: algoritmos convencionales y de refinamiento progresivo.

El primer tipo resuelve simultáneamente todas las ecuaciones, para lo cual generalmente utiliza el método iterativo de Gauss-Siedel, partiendo de un valor inicial aproximado para la emisión de cada una de las áreas. Todos estos algoritmos ([GORA84], [COHE85], [COHE86], [RUSH86],...) tienen el problema del coste computacional y de memoria para la matriz que suponen el que las soluciones no pueden ser refinadas adaptativamente [BERG86].

En el segundo tipo, se realiza un refinamiento sucesivo de la solución. La iluminación debida a una sola área es distribuida al resto de áreas dentro de la escena. En la primera etapa se eligen las áreas asociadas a las fuentes de luz para lanzar su energía a la escena. En las siguientes se seleccionarán fuentes secundarias, comenzando con aquellas áreas que recibieron la mayor parte de la luz emitida por las fuentes. Cada etapa incrementa la precisión de los resultados que se muestran en la pantalla. Debido al reducido coste de estos métodos, se pueden generar imágenes más complejas, obteniendo una solución inicial más rápidamente. Dentro de este grupo podemos encontrar también numerosos trabajos ([CHEN89], [CHEN90], [AIRE89], [BAUM91],...).

Todos estos métodos anteriores tienen el problema de que el mecanismo para calcular el transporte difuso lo extienden para capturar los efectos especulares. Esta traslación del método hace que el tratamiento de las superficies especulares no sea óptimo, ya que la reflexión especular de una superficie depende fuertemente del ángulo con el cual el observador mira la superficie. Por tanto, debemos calcular una gran cantidad de información extra, ya que no tenemos información sobre el punto de vista. A su vez esta información direccional debe ser interpolada para acomodarse

a un punto de vista específico.

Por tanto, estos nuevos métodos aportan una solución muy interesante para reflejar complejos efectos de naturaleza difusa, pero no son muy eficientes en el tratamiento de efectos de naturaleza especular, por lo que una solución optima de la visualización realista pasa por una combinación de los algoritmos de trazado de rayos y de radiosidad. A estos nuevos métodos se les suele denominar métodos híbridos y serán analizados en el siguiente apartado.

### 1.2.3 Métodos híbridos.

Los primeros resultados de los métodos de radiosidad asociados a escenas en las que predominaban las características de naturaleza difusa y en las que los algoritmos de trazado de rayos no aportaban soluciones muy válidas hicieron que un gran número de investigaciones se centraran en el estudio del comportamiento de estos nuevos métodos a la hora de reflejar efectos imposibles o muy costosos de conseguir con el trazado de rayos.

Algunos investigadores llegaron a pensar inicialmente que los métodos de radiosidad desbancarían y convertirían en una técnica olvidada al trazado de rayos. Pronto se observaron las deficiencias que los métodos basados en radiosidad tienen a la hora de reflejar efectos de naturaleza especular. En estos casos, la posición del observador condiciona los resultados del proceso de representación de la escena y para ellos el trazado de rayos es un algoritmo eficiente.

Para resaltar las diferencias entre ambas técnicas, en la figura 1.7 [JANS93] se muestran algunos de los caminos que puede recorrer la luz desde un foco lumínico hasta el observador. Para simplificar, se ha supuesto que las superficies de los objetos de la escena son puramente difusas o especulares. El camino 1 representa la reflexión difusa directa, el segundo la reflexión difusa-especular, el tercero la

reflexión difusa-difusa, el cuarto la reflexión especular-difusa, el quinto la reflexión especular-especular y, por último, el sexto la simplemente especular.

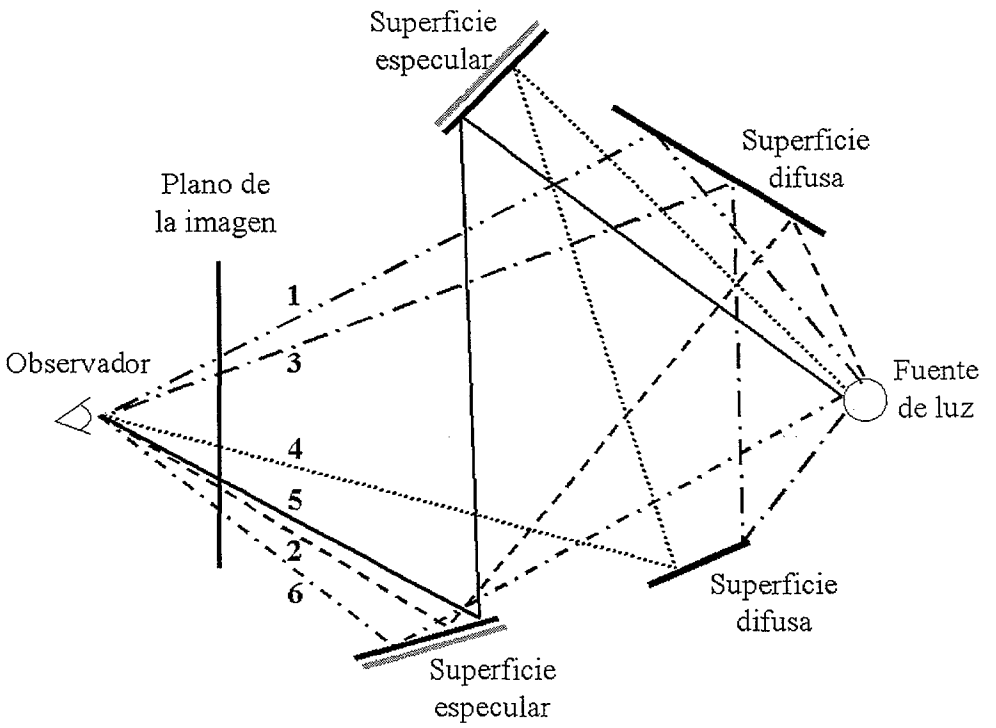



**Figura 1. 7 Posibles caminos seguidos por la luz.**

El trazado de rayos estándar [WHIT80] sólo contempla, junto al proceso de detección de sombras, las luces que siguen los caminos 1, 2, 5 y 6 que aparecen reflejados en la figura 1.7. En cambio, no tiene en cuenta la reflexión indirecta de la luz obtenida como resultado de la interreflexión entre distintas superficies con características o propiedades difusas (camino 3). Del mismo modo tampoco tiene en cuenta la luz que primero es reflejada por una superficie especular y después ilumina a una superficie de tipo difuso visible desde el observador (camino 4). Para capturar estos rayos de luz sería necesario trazar rayos secundarios en todas las direcciones cada vez que un rayo choca con la superficie de un objeto ([KAJI86], [SHIR91], [ARVO90a],...).

Por otra parte el algoritmo de radiosidad clásico no trata de modo correcto los efectos de reflexión especular (caminos 5 y 6). Para tratar dichos efectos se han propuesto algunas extensiones al algoritmo clásico. Unas lo consiguen mediante la utilización de una mayor cantidad de memoria [IMME86], otras tienen un crecimiento exponencial en función del número de espejos de la escena [RUSH87], etc. Un análisis más profundo de las extensiones al algoritmo de radiosidad clásico se puede encontrar en [GLAS95].

Como vemos los puntos fuertes y débiles de ambos métodos son complementarios y parece razonable suponer que un método que los utilice de modo conjunto debe mejorar los resultados obtenidos por cada uno individualmente. Esta es la filosofía que está detrás de los denominados métodos híbridos. Estos generalmente implementan una secuencia de etapas de radiosidad y trazado de rayos y por ello son llamados también algoritmos multipasadas. Cuando se utiliza tan solo una etapa de radiosidad y otra de trazado de rayos se denominan algoritmos de dos pasadas.

La mayoría de los algoritmos de dos pasadas primero utilizan el método de radiosidad para generar el resultado de las múltiples interreflexiones difusas en la escena. Debido a que, como ya hemos indicado, estos cálculos son independientes de la situación del observador (independientes de la vista) pueden almacenarse con el modelo y ser utilizados repetidas veces si lo único que cambia en la escena es la situación del observador. Tras esta primera etapa, la segunda pasada es el trazado de rayos, que añade las características dependientes de la situación del observador, como por ejemplo la reflexión especular.

Podemos encontrar numerosos ejemplos de algoritmos de multipasadas que, como indican F. Jansen Y A. Chalmers [JANS93], presentan algunas diferencias básicas entre ellos. La primera propuesta [WALL87] calcula los valores de la radiosidad, del mismo modo que lo hacen los algoritmos de buffer de profundidad.

El segundo de ellos [SILL89] toma el cálculo previo de la radiosidad como intensidad difusa de cada trozo (patch) y añade los componentes de reflexión especular mediante el trazado de rayos secundarios. Esta versión todavía requiere un cálculo costoso de la radiosidad ya que las sombras desde las fuentes de luz se incluyen en los cálculos de la radiosidad. El tercer grupo ([SHIR90], [CHEN91],[KOK91],...) utiliza el cálculo previo de la radiosidad para obtener una mejor aproximación del término luz ambiente y determinar qué objetos o parte de ellos pueden considerarse también fuentes de luz, acercándose de este modo mejor a los efectos de sombreado de la escena.

# Capítulo 2. Hacia una optimización del Trazado de Rayos.

## 2.1 Introducción.

Como hemos visto, la idea básica del algoritmo trazador de rayos es lanzar rayos desde el observador hacia la escena y determinar para cada uno de ellos la intersección con los diferentes objetos de la misma. El algoritmo inicial propone para ello la realización de dos tareas: (1) calcular la intersección de un rayo con cada uno de los objetos de la escena; (2) si el rayo choca con más de un objeto, entonces seleccionar aquel objeto que esté más próximo al observador pues será el primero que el rayo encontrará en su camino.

Como vemos este método propone realizar de forma exhaustiva los cálculos de todas las intersecciones de un rayo con cada objeto de la escena, por lo que la mayoría de los cálculos realizados no son útiles. Su coste, como apunta Whitted [WHIT80], depende casi exclusivamente de una sola operación: *calcular para cada rayo el punto de intersección entre éste y una entidad geométrica atómica en el*

*espacio tridimensional*, llegando a suponer un 95% del tiempo total en la representación de escenas complejas. Por ello, este algoritmo, aunque muy simple de implementar, es muy costoso computacionalmente. El tiempo de generación de una imagen depende fundamentalmente de la complejidad de la escena (número de objetos) y la resolución de la pantalla (número de rayos a trazar).

Por otra parte, la generación de imágenes mediante un ordenador, se comporta de modo un tanto diferente a como lo hace una película fotográfica al exponerla a una escena real. La diferencia básica consiste en el hecho de que debemos representar una señal continua en un dispositivo de visualización de un ordenador que es de tipo discreto. El modo de operar con una señal continua, en este tipo de dispositivos, consiste en recoger un conjunto de muestras de dicha señal, a partir de las cuales podamos generar una representación de la señal original. Si tomamos un número elevado de muestras entonces la representación de la señal original a partir de las muestras obtenidas se comporta de modo semejante a la señal a la que representan, aumentando la calidad. Pero si el número de muestras es insuficiente la apariencia de la representación discreta no se corresponde con la señal original. A este problema de conversión se le denomina "aliasing" (con dicho nombre se quiere resaltar que el problema es hacer que una señal se parezca a otra, alias).

Este problema no es propio del trazado de rayos, sino que está presente siempre que se desea representar una imagen mediante un ordenador, independientemente de la técnica utilizada para ello. Para reducir este problema se han aportado una serie de soluciones todas ellas encuadradas dentro del término "anti-alising".

La mayoría de las soluciones de anti-aliasing aportadas se apoyan en el aumento de la frecuencia de muestreo, consiguiendo de ese modo una mejor aproximación a la señal original. Las distintas alternativas propuestas se basan, por

tanto, en la idea de trazar por cada punto de la pantalla varios rayos, bien mediante técnicas de sobremuestreo uniforme, adaptativo, estadístico o estocástico [CAMA95], [WHIT80], [LEE85], [DIPP85], [COOK86],... (una completa descripción de las diferencias entre dichas alternativas y su aplicación al trazado de rayos, la podemos encontrar en el libro de A. Glassner [GLAS89]). Un hecho en común a la mayoría de las propuestas de antialiasing anteriores es que se apoyan en el denominado *espacio de la imagen*, es decir, que utilizan únicamente la información de la imagen para abordar las soluciones al problema del aliasing. En cambio, como apuntan J. Genetti et al. [GENE98] existen otras alternativas, a las que denominan basadas en el *espacio de los objetos* (THOM89], [OHTA90], [GENE98],...), que obtienen mejores resultados ya que utilizan información generada durante el proceso de intersección rayo-objeto. Esta información les permite no lanzar rayos adicionales en las zonas desocupadas ya que en ellas no se producen realmente intersecciones.

El hecho de necesitar trazar un número elevado de rayos para conseguir imágenes de calidad, junto al modo en que se determina el color de cada uno de los rayos, provoca que los tiempos necesarios para la generación de una imagen sean excesivamente altos, pudiendo llegar a durar minutos e incluso horas.

Este excesivo coste computacional y tiempo de generación, ha motivado la creación de una importante línea de investigación, todavía no cerrada, en la que, a través de diferentes soluciones, se pretenden solventar o reducir los problemas anteriormente expuestos.

Según J. Arvo y D. Kirk [ARVO89], se poden encontrar tres grandes estrategias a la hora de buscar optimizaciones del trazado de rayos: (1) cálculo más rápido de intersecciones; (2) menor número de rayos; (3) rayos generalizados. En gráfico 2.1 se muestra una clasificación de las diferentes alternativas de aceleración en función de los criterios anteriores.

La primera de ellas se puede alcanzar de varias maneras: calculo más eficiente de la intersección de rayo-objeto; menor número de intersecciones rayo-objeto; paralelización del algoritmo.

❶ Cálculo más rápido de intersecciones.
- Intersección rayo-objeto más eficiente.
- Menor número de intersecciones rayo-objeto.
  - Jerarquías de volúmenes envolventes.
  - Subdivisión del espacio (uniforme y no uniforme).
  - Técnicas direccionales.
- Tratamiento simultaneo de varios rayos (paralelismo).

❷ Menor número de rayos.
- Control adaptativo de la profundidad del árbol de recursión.
- Optimizaciones estadísticas para el tratamiento del aliasing.

❸ Rayos generalizados.
- Trazado de rayos agrupados en estructuras más complejas.

**Gráfico 2.1 Clasificación de los diferentes métodos de aceleración.**

La segunda de ellas, menor número de intersecciones, se pretende conseguir básicamente mediante: reducción del nivel de recursión a la hora de generar rayos secundarios; utilización de métodos estadísticos para reducir el número de rayos a trazar para mitigar el problema de aliasing.

Por último, la utilización de rayos generalizados se basa en la creación de estructuras más generales que permitan trazar de modo simultáneo un conjunto de rayos.

Para llevar a cabo la mayoría de las optimizaciones anteriormente expuestas existen dos grandes propuestas: utilizar las características básicas de la representación de imágenes 3D (coherencia) y realizar una implementación paralela

del algoritmo. A continuación se describen mas detalladamente ambas alternativas.

## 2.2 Características básicas de la representación de imágenes tridimensionales: tipos de coherencia.

Antes de revisar algunas alternativas de aceleración vamos a repasar ciertas características atribuibles a la generación de imágenes. Estas son propiedades que pretenden reflejar cierta homogeneidad en la escena a representar o en el proceso de generación. Dicha homogeneidad se expresa mediante ciertos tipos de "coherencia".

### 2.2.1 Tipos de Coherencia.

Sutherland et al. [SUTH74b] identificaron y clasificaron diferentes manifestaciones o tipos de coherencia. Aunque ellos los aplicaron para mejorar el comportamiento de los algoritmos de eliminación de superficies ocultas, estos tipos son ampliamente utilizados en otros entornos. De ellos, dentro del trazado de rayos, debido a la naturaleza del proceso, podemos utilizar básicamente cuatro tipos de coherencia: de *objetos*, de *imagen*, de *rayos* y de *fotograma*.

Junto a los niveles de coherencia anteriores, Green y Paddon [GREE89] proponen un quinto modo de coherencia que denominan *coherencia de datos*. Este modo se puede interpretar como una forma general de coherencia de rayos y coherencia de imagen. Este tipo de coherencia se apoya en la idea de que la mayoría de las peticiones a la base de datos de objetos se realizan a un subconjunto reducido de ella y tienden a mostrar una gran localidad espacial. Esta forma de coherencia depende fuertemente del orden en que se realice el algoritmo, en particular, del orden en que se recorren los árboles de rayos.

En el gráfico 2.2 puede apreciarse un resumen de los diferentes tipos de coherencia y las alternativas utilizadas para llevarlos a la práctica. A continuación

vamos a analizar dichos tipos de coherencia, indicando algunos de los trabajos que basan su alternativa de aceleración en cada uno de ellos.

**Coherencia de Objetos**

- Descomposición guiada por los objetos de la escena.
  - Jerarquías de volúmenes envolventes.
    - Diferentes tipos de primitivas envolventes.
- Descomposición guiada por el espacio de la escena.
  - Descomposición uniforme.
    - Matrices tridimensionales de cubos o *voxel.*
  - Descomposición no uniforme.
    - Árboles octales u *octrees.*
    - Árboles binarios o *BSP.*
- Alternativas mixtas de descomposición.

**Coherencia de Rayos**

- Técnicas Direccionales.
- Técnicas de Rayos Generalizados.

**Coherencia de imagen**

**Coherencia de fotograma o temporal**

**Gráfico 2.2 Clasificación de los diferentes modos de coherencia.**

### 2.2.1.1 Coherencia de Objetos.

Esta viene expresada por el hecho de que *posiciones próximas en el espacio tienden a estar ocupadas por un solo objeto de la escena o un conjunto reducido de ellos.* Tal vez sea el más importante y más utilizado tipo de coherencia. La idea de las diferentes aplicaciones de la coherencia de objetos es descomponer el espacio de la escena en áreas atendiendo a diferentes criterios de subdivisión.

Existen dos grandes alternativas de descomposición de la escena: ***guiada por los objetos*** o ***guiada por el espacio.***

La primera alternativa, **guiada por los objetos**, para descomponer la escena se apoya en la situación de los objetos, agrupando objetos próximos y creando una jerarquía de volúmenes envolventes que los contienen ([WHIT80], [RUBI80], [WEGH84], [KAY86], [GOLD87], [CHAR90], [WILT94],...). Una de las grandes diferencias entre las diferentes alternativas se producen en la elección del volumen envolvente (ver figura 2.1-a), siendo los más frecuentemente utilizados las esferas [WHIT80] o los paralelepípedos rectangulares, también denominados cajas envolventes "bounding boxes" ([RUBI80], [GOLD87], [WILT94]...). En todo caso, también podemos encontrar trabajos, como los de Weghorst et al. [WEGH84], en los que se realiza un estudio previo que permite ajustar el volumen que mejor se adapta al objeto u objetos a contener.

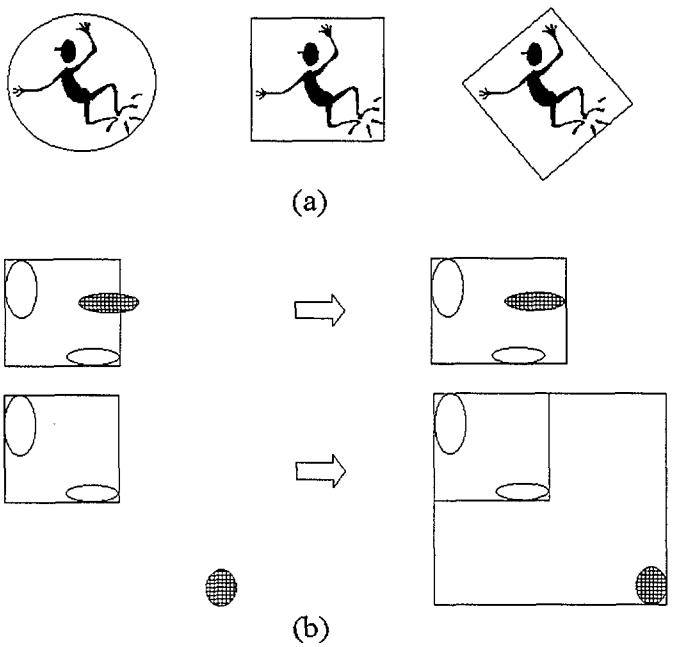



**Figura 2.1 a) Diferentes primitivas utilizadas como envolventes de un objeto. b) Generación automática de la jerarquía de volúmenes envolventes.**

Por otra parte dentro de estos métodos es interesante analizar el criterio utilizado para construir la jerarquía de volúmenes envolventes, pues será un factor muy importante a la hora de optimizar el trazado de rayos. En este caso se debe pretender obtener una jerarquía con un número reducido de niveles y, a su vez, que los objetos contenidos en un volumen envolvente estén muy próximos.

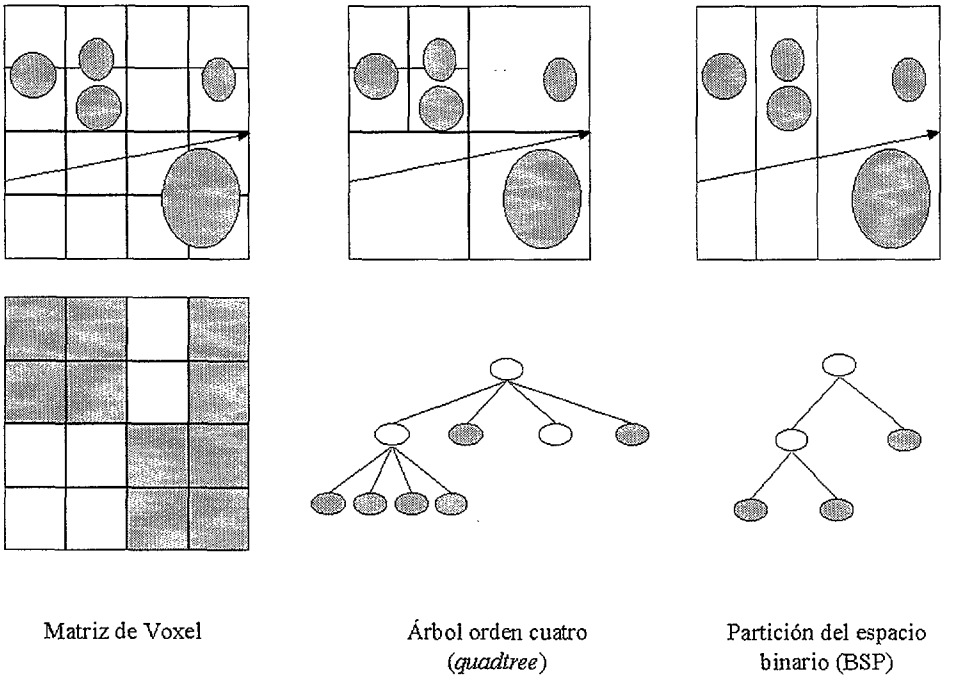



**Figura 2.2 Diferentes criterios de descomposición espacial.**

Tal vez uno de los trabajos más interesantes sobre construcción automática de una jerarquía es el desarrollado por J. Goldsmith y J. Salmon [GOLD87]. En éste se define una heurística que permita estimar el coste la jerarquía que se está construyendo. En su estudio la jerarquía se construye de modo incremental. El proceso de inserción de un objeto comienza desde la raíz de la jerarquía, seleccionado a partir de ella el subárbol que implique, si el objeto finalmente se incorporara como un objeto contenido en el volumen, un menor aumento en el área

superficial del volumen asociado. En la figura 2.1-b puede apreciarse un ejemplo de este criterio.

Junto a las alternativas anteriores, encontramos otras propuestas que se apoyan en criterios de descomposición espacial para obtener la estructura que se asocia a la escena, esta alternativa la hemos denominado **guiada por el espacio**. En este caso la escena se divide mediante un proceso de descomposición que permite obtener celdas del mismo (descomposición uniforme) o de diferente tamaño (descomposición no uniforme).

En el proceso de **descomposición uniforme** ([FUJI86], [AMAN87], [CLEA88], [MÜLL88], [SLAT92], [KUZM94], [ZALI97],...) se obtiene una matriz o malla tridimensional formada por cubos del mismo tamaño "voxel" (ver figura 2.2). La eficiencia de este tipo de descomposición viene dada básicamente por la rapidez con que un rayo puede recorrer esta estructura. Esta rapidez se debe esencialmente a la utilización de métodos incrementales que permite determinar el voxel de salida sin necesidad de calcular intersecciones complicadas con el voxel que está atravesando. Para ello, los procesos de recorrido de la matriz de voxel se comportan de modo semejante a como lo hacen los algoritmos de generación de líneas propuestos en la literatura de gráficos 2D [GONZ98c].

En todo caso, los algoritmos que utilizan este tipo de estructuras se ven fuertemente afectados por el nivel de subdivisión de la matriz. Así, escenas que contienen objetos muy pequeños respecto al volumen global de la escena y dichos objetos están fuertemente concentrados en pocas regiones de dicha escena, este tipo de técnicas producen matrices con un gran número de voxel de reducido tamaño, lo cual dificulta y hace poco eficiente su recorrido a la vez que requiere una gran cantidad de memoria para su almacenamiento.

Para solventar este problema han surgido nuevas alternativas que asocian a la

escena un conjunto de matrices de diferente tamaño (más tupida en aquellas áreas con una mayor concentración de objetos) relacionadas entre sí mediante una jerarquía ([JEVA89], [CAZA95], [KLIM97],...). Con estas nuevas estructuras se reduce el tiempo necesario para atravesar la escena a la vez que disminuye la cantidad de memoria necesaria para almacenar la estructura de la escena.

Por último, las técnicas de **descomposición no uniforme** estructuran la escena en regiones de diferente tamaño, asociándole a ésta un árbol de orden *n*, en el cual los nodos terminales referencian a una de las regiones en que se ha descompuesto dicha escena (ver figura 2.2). Para descomponer la escena se suelen utilizar tanto árboles binarios ([KAPL85], [JANS86], [KAPL87], [SUNG92],...), como octales ([GLAS84], [SAME89], [MACD90], [SPAC91], [MCNE92], [GARG93], [ENDL94], [GONZ98a],...). La principal diferencia se obtiene del tipo de descomposición que realizan y por tanto de la estructura necesaria para almacenar la escena. En la figura 2.2 pueden verse las dos estrategias de descomposición utilizadas.

Estos tipos de descomposición intentan reducir los problemas que se presentan en la descomposición uniforme en aquellas escenas en las que los objetos están fuertemente concentrados en ciertas regiones, dejando otras vacías. Dicha optimización la consiguen gracias a que permiten que el nivel de descomposición sea mayor en las regiones más pobladas. En cualquier caso, el recorrido de estas nuevas estructuras se complica notablemente, lo que hace que no siempre sea rentable utilizarlas como sustitución de las matrices de voxel.

Para amortiguar los problemas de la utilización aislada de las técnicas de descomposición no uniforme se han propuesto algunos algoritmos que combinan su utilización con técnicas de descomposición uniforme ([STOL95], [CUON97], [LAST98], ...) y que facilitan el recorrido por las zonas que no contienen objetos.

Junto a los algoritmos anteriores que guían el proceso de descomposición bien en función de los objetos o del espacio, encontramos otros que mezclan ambas alternativas ([SCHE87], [KORN87],...) y por tanto pueden considerarse como **alternativas mixtas**. En estos casos, existen diferentes propuestas, así Scherson y Caspary [SCHE87] realizan una definición de los niveles altos de la jerarquía mediante árboles octales mientras que los más bajos están formados por volúmenes envolventes. De este modo disminuye el nivel de descomposición necesario para el árbol octal, al incluir un mayor número de objetos en los nodos terminales, reduciendo, por una parte, el tiempo necesario para recorrer esta estructura. Con el proceso anterior al incrementarse el número de objetos contenidos en un nodo el tiempo necesario para calcular la intersección de un rayo con dichos objetos aumenta. Es por ello que los objetos contenidos en un nodo se agrupan nuevamente formando una jerarquía de volúmenes envolventes que permite acelerar el cálculo de intersecciones.

Otro ejemplo lo encontramos en el trabajo de A. Glassner [GLAS88]. En éste se crea una jerarquía de volúmenes envolventes partiendo de una descomposición del espacio mediante un árbol octal para de ese modo garantizar que los volúmenes envolventes de la jerarquía no se solapan.

En cualquier caso podemos ver que la idea en que se basan todas las técnicas de aceleración anteriores (guiadas por los objetos o por el espacio), consiste en posibilitar la realización de un seguimiento de la trayectoria de cada rayo dentro de la escena. Así, para determinar el objeto con el que choca un rayo no es necesario calcular la intersección de éste con todos los objetos de la escena sino tan solo con aquellos contenidos en las celdas que atraviesa, como puede verse en la figura 2.2.

Esta idea de coherencia también ha sido utilizada en numerosos trabajos que proponen la paralelización del trazado de rayos. Su utilización es más frecuente en aquellos sistemas que realizan una distribución de la escena entre los distintos procesadores (MIMD de memoria distribuida). Dentro de estos trabajos, como

indican D. Badouel, K. Bouatouch y T. Priol [BADO94], podemos encontrar algoritmos que realizan una división de la escena guiada por los objetos ([CASP89] [GOLD85],[POTM89],...) o guiada por el espacio ([DIPP84] [CLEA86] [KOBA88a] [CAUB88] [PRIO89] [ISLE91] [AYKA94], [GRIM94],...). Esta estructura, a la vez que optimiza el proceso de trazado de un rayo, sirve para realizar el reparto de los objetos de la misma entre los diferentes procesadores.

#### 2.2.1.2 Coherencia de Rayos.

Este tipo de coherencia es más difícil de explotar que la anterior, aunque es la que permite una mayor independencia entre el tiempo de generación y la calidad de la imagen, es decir, entre el tiempo y el número de rayos a trazar. Para conseguir imágenes de alta calidad la resolución de ésta (número de pixel de la pantalla) debe ser elevada (p.e. 1280x1024) y además debemos aplicar técnicas que mitiguen el problema de aliasing, con lo que, en general, por cada pixel debemos trazar varios rayos. En la generación de estas escenas nos encontramos con un gran número de rayos a trazar dentro de la escena, pero todos ellos muy próximos entre sí. En definitiva la coherencia de rayos es elevada y por tanto es muy recomendable aplicar algoritmos que la utilicen.

La idea básica de este tipo de coherencia es la siguiente: *rayos muy próximos entre sí, es decir, aquellos cuyo origen y dirección son semejantes, probablemente chocan con los mismos objetos dentro de la escena.*

Basadas en la coherencia de rayos podemos encontrar dos grandes grupos de algoritmos [ARVO89], aquellos agrupados dentro de las denominadas *técnicas de* ***rayos generalizados*** y aquellos encuadrados dentro de las ***técnicas direccionales***.

Las **técnicas de rayos generalizados** se basan en la idea de agrupar rayos próximos entre sí en un manojo o haz, lo que les permite trazarlos en la escena de modo conjunto dentro del haz. Por tanto, la idea básica es no tratar los rayos de

manera individual, sino intentar determinar el color de un conjunto de pixels a través del comportamiento de dichas agrupaciones de rayos. Las diferentes alternativas que se apoyan en rayos generalizados se diferencian básicamente en el modo en que se agrupan los rayos. Así, entre otros, podemos encontrar agrupaciones en troncos de pirámide de sección poligonal (Beam Tracing [HECK84]), en conos (Ray Tracing with Cones [AMAN84]) o que se apoyan en un tipo de rayos especiales denominados rayos axiales, que agrupan rayos próximos a dicho rayo axial (Pencil Tracing [SHIN87]).

En cualquier caso no todas las alternativas propuestas que pretenden agrupar rayos en haces producen resultados satisfactorios, como sucede en la propuesta realizada por L.R. Speer et al [SPEE86]. Esto se debe a que, en algunos casos [SPEE86], el coste de calcular los haces es tan elevado que hace que el coste final del trazado sea incluso mayor que en versiones en las que se realiza un trazado de rayos individuales. Por tanto, la tarea más costosa y complicada es la determinación de los nuevos haces conforme estos van avanzando o propagándose por la escena.

Por otra parte, las **técnicas direccionales** tienen en cuenta la dirección de los rayos individuales y a partir de ella construyen una estructura de datos que asocia a un conjunto de rayos con direcciones semejantes una lista de objetos candidatos. Dentro de ellas encontramos tres tipos básicos: "Light buffer" [HAIN86], "Ray Coherence"[OHTA87], "Ray Clasification" [ARVO87]. Estas propuestas utilizan generalmente un *cubo dirección* (ver figura 2.3), que permite discretizar las direcciones de salida en un conjunto finito de regiones rectangulares (celdas) a las que se les asocia una lista de objetos. Las principales diferencias entre las distintas propuestas residen en: el criterio utilizado para asociar los objetos a las celdas; el criterio de subdivisión (uniforme o no uniforme); el tipo de rayos a los que puede aplicarse (cualquier tipo de rayo o tan sólo a rayos de sombra).

Como podemos apreciar este tipo de técnicas, al igual que las que se basan en la coherencia de objetos, utilizan una estructura de descomposición que asocian a la

escena. La idea que subyace en la creación de esta estructura es obtener una descomposición de la escena siguiendo criterios direccionales y que, en base a ella, se facilite la determinación de los objetos a analizar en la tarea del cálculo de intersecciones. Pero como puede suponerse dicha estructura creada con criterios direccionales depende fuertemente de los potenciales orígenes de los rayos (el observador en el caso de rayos primarios) que la atraviesen y el resultado final tras el proceso de construcción podría variar si se desplaza el origen de dichos rayos. En cambio, la estructura creada con la idea de la coherencia de objetos es totalmente independiente del origen de los diferentes rayos y, por tanto, no se ve afectada por el desplazamiento de dicho origen. Por tanto, vemos que el modo de creación de la estructura a la que se hace referencia en las técnicas direccionales (basada en la coherencia de rayos) es diferente al utilizado con un criterio de coherencia de objetos.

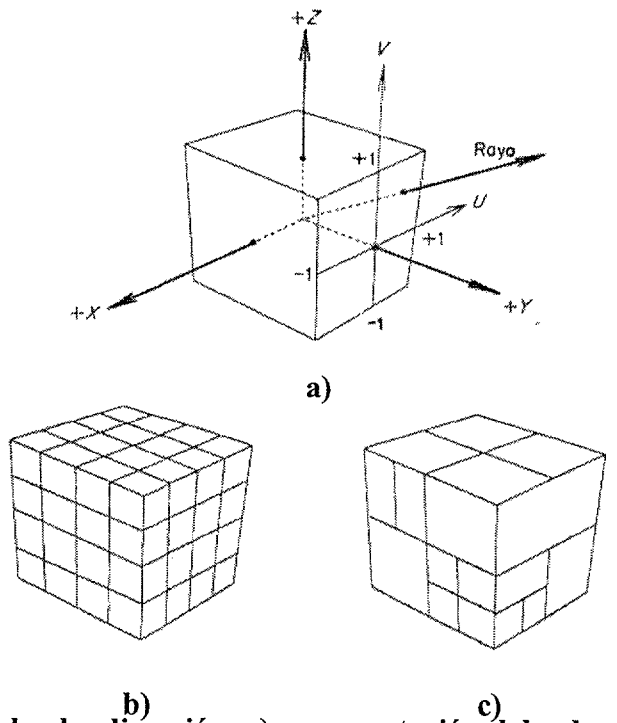



**Figura 2.3 Utilización del cubo dirección. a) representación del cubo dirección; b) aplicación de subdivisión uniforme; c) aplicación de subdivisión no uniforme**

Este diferente criterio de construcción también se ve reflejado en la propia estructura a la que este tipo de técnicas dan lugar. Como puede apreciarse, la idea del "cubo dirección" es asociar a cada cara de dicho cubo una estructura de descomposición, pudiéndose dar el caso, como se muestra en la figura 2.3c para el caso de utilizar una descomposición no uniforme, de que cada cara del cubo puede tener diferente tipo de descomposición, de ese modo, la descomposición de la cara superior no se corresponde con la realizada para las caras frontales. Por tanto, vemos que la idea no es descomponer el cubo que almacena la escena de manera recursiva, sino realizar una descomposición de las caras de salida del "cubo dirección" de manera que se ajusten al criterio de direccionalidad, tras lo cual se puede obtener una descomposición diferente en cada cara. A su vez generalmente asocian a la escena varios "cubos dirección" en función de la técnica concreta utilizada (por ejemplo, la técnica denominada "light buffer" crea tantos cubos dirección como fuentes de luz existen en la escena)

Por otra parte, la diferencia básica entre las técnicas direccionales y las de rayos generalizados, reside en que en el caso de las técnicas direccionales tras generar la estructura de descomposición se realiza un procesado de cada rayo de manera individual apoyándose para su trazado en la estructura direccional previamente calculada, mientras que en el caso de las técnicas de rayos generalizados el tratamiento es conjunto para los rayos contenidos en un haz.

De todas las propuestas de técnicas direccionales, tal vez la más utilizada es la primera de ellas (light buffer) que pretende optimizar el cálculo de los rayos de sombra. Para ello, asocia a cada fuente de luz un cubo subdividido de manera uniforme y cada celda tiene asociada un conjunto de objetos que son visibles desde la fuente de luz. Esto es, si situamos un observador en la fuente de luz y consideramos la celda como una ventana por la que el observador mira la escena, dicho observador tan solo verá los objetos contenidos en la lista. De este modo un rayo de sombra que al dirigirse a la fuente luz choque con una celda dada tan solo deberá comprobar los objetos de la lista asociada a la celda pues son los únicos que

pueden hacer que el punto al que está asociado el rayo se encuentre en sombra.

Al igual que sucede con el resto de tipos de coherencia éste también es utilizado para optimizar ciertos procesos dentro del diseño de algoritmos paralelos. En este campo, como apuntan D. Badouel,K. Bouatouch y T. Priol [BADO94], su mayor utilización se encuentra en la posibilidad de realizar un balanceo de la carga. Si trazamos un número reducido de rayos en la escena, analizamos su comportamiento y suponemos que rayos próximos a estos tendrán un comportamiento similar, podemos dividir y distribuir los objetos de la escena entre los diferentes procesadores de tal modo que la carga de trabajo de cada uno sea semejante.

#### 2.2.1.3 Coherencia de Imagen.

*Un objeto 3D tiende a ocupar después de su transformación a dos dimensiones posiciones contiguas en la pantalla, mientras que diferentes objetos suelen ocupar posiciones no contiguas*. De este modo puntos de la pantalla (pixel) contiguos probablemente tengan un color semejante. Este modo de coherencia no es fácil de aplicar al algoritmo trazador de rayos, aunque podemos encontrar algunas implementaciones [WEGH84] que realizan un preprocesado que permite explotar después esta característica en el trazador de rayos.

Tal vez donde más se tenga en consideración este tipo de coherencia es en algunas versiones paralelas del trazado de rayos donde se realiza una distribución del cálculo de la imagen entre distintos procesadores ([NISH83], [NARU87], [GREE90],...). En este caso, se puede suponer que pixel contiguos tengan una carga de trabajo semejante. La coherencia de imagen se puede utilizar para realizar un mejor balanceo de la carga. Una alternativa de balanceo de la carga estático, puede consistir en asociar a distintos procesadores pixel contiguos consiguiendo que el trabajo asignado a cada uno de ellos sea semejante.

#### 2.2.1.4 Coherencia de fotograma o temporal.

Esta forma de coherencia es una ampliación del modo anterior al que se ha añadido la dimensión temporal y puede ser utilizada en secuencias de imágenes en procesos de animación. *La proyección (visión) de una escena tiende a cambiar de modo continuo a través del tiempo, es decir, dos imágenes consecutivas (fotogramas) en una secuencia de animación tienden a ser semejantes si la diferencia de tiempo entre ambas es pequeña.*

La mayoría de las soluciones propuestas se apoyan en estructuras utilizadas para la aceleración del trazado de un fotograma aislado y le añaden una nueva dimensión temporal ([GLAS88], [BADT88], [MURA90], [GRÖL91], [JEVA92], [GRIM94], [DAVI98]...). De este modo, aprovechan la información que no ha cambiado y tan solo vuelven a procesar las zonas de la escena que se ven afectada por la nueva situación. Su aplicación cada vez es mayor y generalmente está acompañada de soluciones que aprovechan también las ventajas de otros tipos de coherencia o del paralelismo.

### 2.2.2 Coherencia de objetos, mediante la utilización de árboles octales (octrees).

Tal vez de todos los tipos de coherencia la más utilizada sea la coherencia de objetos y dentro de ella la utilización de árboles octales es una alternativa que ofrece buenas expectativas y que ha sido analizada por numerosos investigadores. En este apartado vamos a realizar un pequeño repaso a los trabajos más importantes que utilizan este tipo de estructuras para descomponer la escena.

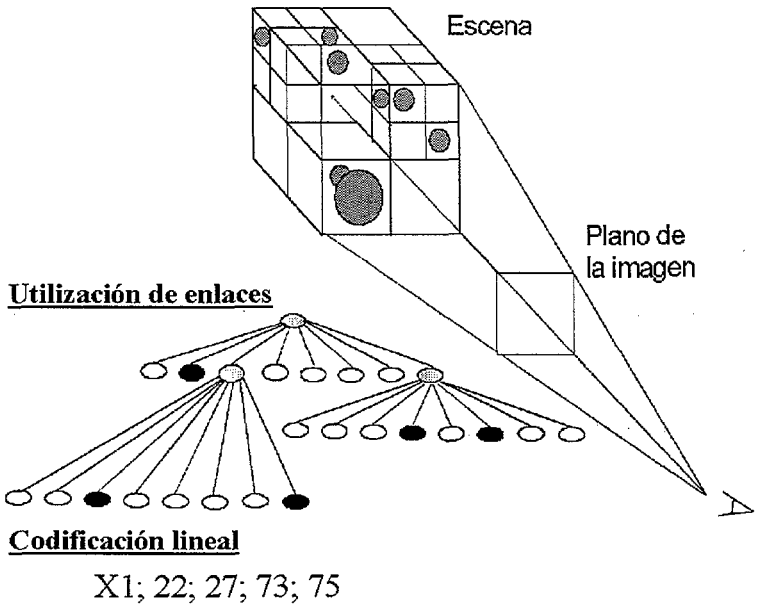



**Figura 2.4 Descomposición de la escena mediante un árbol octal.**

Este tipo de técnicas discretizan el espacio en regiones de diferente tamaño para, de este modo, ajustarse mejor a las características de la escena. Esta descomposición no uniforme permite realizar una subdivisión mayor en aquellas áreas más densamente pobladas y definir cubos de gran tamaño asociadas a zonas casi desocupadas o totalmente vacías. Para realizar este proceso de descomposición no uniforme se apoya en una estructura jerárquica denominada *árbol octal* u *"octree"*. Este tipo de estructuras se construyen mediante la subdivisión del cubo original asociado a la totalidad de la escena, en ocho octantes (nodos) de igual tamaño. Este proceso de subdivisión se realiza de modo recursivo sobre los diferentes cubos que se van generando, hasta alcanzar cubos que cumplen un determinado criterio de homogeneidad (en este caso el número de objetos contenidos en el cubo no supera un cierto umbral). En la figura 2.4 podemos apreciar el proceso de subdivisión de una escena y el árbol octal resultante.

***Diferencias básicas entre los algoritmos que utilizan árboles octales:***

- Modo de almacenamiento de la estructura.
  - Enlaces.
  - Linear octree.
- Instante de construcción del árbol octal.
  - Descomposición a priori.
  - Descomposición dinámica.
- Modo en que se calcula el punto de entrada/salida.
  - Métodos explícitos.
  - Métodos iterativos.
- Proceso de búsqueda del nodo que contiene a un determinado punto.
  - Métodos raíz.
  - Métodos padre.
  - Métodos enlace.

**Gráfico 2.3 Principales diferencias entre los algorimos que utilizan los árboles octales como estructura de descomposición.**

El modo más común de **representar y almacenar la estructura de descomposición** obtenida se apoya en la utilización de enlaces desde el nodo padre hasta cada uno de sus ocho hijos ([SAME84], [SAME90a], [SAME90b],...). En este caso, como podemos apreciar en la figura 2.4, dentro de la estructura de árbol tenemos dos tipos de nodos: *nodos intermedios* que se descomponen en un nivel inferior, y *nodos terminales* que, a su vez, pueden estar ocupados o vacíos.

Aunque ésta es la alternativa de almacenamiento más usual, existen otras propuestas de almacenamiento de la estructura obtenida, como la denominada *"linear octree"* ([GARG82a], [GARG82b], [GARG86], [GARG89], [GARG93],...). En este caso, se almacenan tan sólo los nodos ocupados y para cada uno de ellos utiliza una secuencia de dígitos que indican su situación. El número de dígitos asociado a cada nodo es igual al número máximo de niveles del árbol. Cuando un

nodo terminal se encuentra en un nivel del árbol inferior al nivel máximo se le añaden tantas *X* como sea necesario hasta completar el número de dígitos establecido. En la figura 2.4 podemos apreciar la descomposición de la escena y las representaciones de dicha estructura tanto con punteros como utilizando la codificación lineal. Aunque bastante menos usual, esta representación también ha sido utilizada para optimizar el algoritmo de trazado de rayos [GARG93].

Junto a la diferencia en el modo de representar internamente la estructura de descomposición de la escena, también podemos encontrar otra gran diferencia en el propio **proceso de descomposición**: *descomposición a priori*, *descomposición dinámica*. La mayoría de los trabajos se basan en la primera alternativa que realiza un proceso de descomposición previo al proceso de trazado ([GLAS84], [SAME89], [MACD90], [SPAC91], [GARG93], [ENDL94], [GONZ98a],...).

En este caso, se establecen dos criterios básicos de descomposición: número de objetos contenidos en un nodo y nivel máximo de descomposición. Con el primero se pretende reducir el tiempo necesario para calcular la intersección de un rayo que llega a un nodo ocupado por objetos. En cambio la segunda regla pretende reducir el nivel de subdivisión del árbol y por tanto el tiempo necesario para recorrer la escena. En este caso cada cubo o nodo contiene una lista de los objetos que están total o parcialmente contenidos dentro de él y por tanto serán candidatos a chocar con los rayos que lleguen a dicho cubo. Para determinar los objetos asociados a un nodo se propone realizar una comprobación de la superficie del objeto con las seis caras que delimitan el cubo. Si la superficie interseca con alguna de las caras, el objeto se asocia a la lista de objetos del nodo. Para aquellos objetos que no cumplen la condición anterior es necesario realizar una nueva comprobación para determinar si éstos están totalmente contenidos dentro del cubo. En este último caso, sí cualquier punto del objeto está dentro de cubo, el propio objeto también lo estará y por tanto deberá añadirse a la lista asociada al nodo.

La segunda alternativa propone realizar una descomposición dinámica de la escena conforme sea necesario ([HEBE90], [MCNE92],...). Estos estudios se apoyan en la idea de que cuanto menor es el tamaño de un nodo menor es la probabilidad de que un rayo llegue a él. Con ello reduce el tiempo previo de descomposición de la escena, al crear tan solo los niveles altos del árbol octal. En este caso, a los nodos de tipos no terminal e intermedio, se añaden aquellos parcialmente subdivididos. Estos últimos están asociados con nodos que contienen un número de objetos superior al umbral de subdivisión pero que al tener un nivel muy bajo dentro del árbol (son de reducido tamaño y con una menor probabilidad de que sean visitados por algún rayo) no se ha descompuesto en el proceso de subdivisión previo. Durante el proceso de trazado si un rayo llega a un nodo de estas características, dicho nodo se subdivide y se continúa el proceso de trazado estándar.

En cualquiera de los casos anteriores tras descomponer la escena se debe proceder a trazar rayos dentro de este tipo de estructuras. El proceso básico de trazado de rayos en este tipo de estructuras puede verse en el gráfico 2.4.

```
Trazar_Rayo_en_Octree (rayo)
{
        Q = rayo.puntoEntrada;
        Repetir /* recorrido del rayo en el octree */
                Buscar el nodo que contiene al punto Q;
                Para cada objeto asociado con el nodo hacer
                        CalcularInterccion (rayo, objeto);
                Sí no se encuentre intersección entonces
```

**Gráfico 2.4 Proceso básico de trazado de rayos en un árbol octal.**

En este algoritmo existen dos tareas básicas: (1) determinar el punto por el que un rayo entra al nodo, y (2) buscar el nodo que contiene a un determinado punto. Como indican R. Endl y M Sommer [ENDL94], el modo en el que se realizan dichas

tareas permite obtener una clasificación de los diferentes algoritmos que utilizan los árboles octales. Si nos fijamos en el **modo de determinar el punto de entrada/salida** podemos clasificar a los algoritmos en: *método explícitos* ([GLAS84], [SAND85], [SAME89], [ROTH91], [FRÖH88], [ROTH89], [GARG93], [ENDL94], [GONZ98a],...) y *métodos iterativos* ([FUJI85], [SPAC91],[SUNG91],...).

Métodos explícitos:

Realizan explícitamente el cálculo de la intersección del rayo con los planos que delimitan el nodo, determinando de ese modo el punto de salida. Por tanto, para ello debe conocerse la situación y el tamaño del nodo en el que se encuentra el rayo.

Métodos iterativos:

Determinan el punto de salida y el nodo que visitarán de manera iterativa a partir del conocimiento del punto de entrada y varios parámetros definidos por el propio algoritmo.

Esta clasificación anterior la podemos completar observando el **modo utilizado para determinar el nodo que contiene a un determinado punto**. En este caso podemos clasificar a los diferentes algoritmos en tres: *métodos raíz* ([GLAS84],...), *métodos padre* ([SAND85], [SAME89], [ROTH91],...) *métodos enlace* ([FRÖH88], [ROTH89], [ENDL94],...).

Métodos raíz:

Determinan el nodo que contiene un determinado punto siempre partiendo desde la raíz del árbol hasta llegar mediante una búsqueda descendente a dicho nodo.

Métodos padre:

Se apoyan en la idea de que dos nodos vecinos siempre tienen un antecesor común. En este caso es necesario incluir dentro de la estructura que representa al árbol octal un enlace al nodo padre dentro de dicha jerarquía. En este caso el proceso tiene dos subtareas: ascender en el árbol hasta encontrar el primer antecesor común; realizar un descenso en el árbol hasta encontrar el nodo buscado. Este último proceso tiene en cuenta el camino seguido para encontrar el primer antecesor común y realiza un descenso en sentido contrario.

Métodos enlace:

En este caso se puede encontrar el nodo vecino de igual o superior tamaño de modo directo. Si en la dirección dada los vecinos son de menor tamaño que el del nodo actual entonces es necesario realizar un descenso en el árbol hasta alcanzar el nodo buscado.

### 2.2.3 Coherencia de rayos, mediante la utilización de rayos generalizados.

Como ya hemos indicado anteriormente la coherencia de rayos puede aprovecharse desde dos perspectivas: rayos generalizados y técnicas direccionales. En este apartado vamos a estudiar de manera más detallada algunos trabajos previos que van a servir de base al algoritmo que aquí se propone, y, por tanto, nos centraremos básicamente en las técnicas de rayos generalizados.

La idea en que se apoyan estas técnicas para realizar la aceleración del algoritmo es crear estructuras tridimensionales que contengan en su interior una conjunto de rayos y que por tanto permitan trazarlos en la escena de modo simultáneo. La gran diferencia estriba en la estructura utilizada para contener a los

rayos. Así podemos encontrar tres alternativas básicas: haces (beams) [HECK84] [DADO85], conos [AMAN84] [KIRK87] y lápices[SHIN87]. En cualquier caso, como apuntan J. Arvo y D. Kirk [ARVO89] todas estas alternativas imponen algún tipo de restricción sobre la escena o al proceso de representación. Por ejemplo, en algunos casos se deben imponer restricciones sobre el tipo de objetos que se pueden representar, o los cálculos del punto de intersección deben ser siempre aproximados. En cualquier caso, los beneficios obtenidos pueden hacer que el tiempo de generación de la imagen se reduzca sensiblemente, siendo especialmente interesante para reducir optimizar las soluciones aportadas para reducir el efecto de aliasing.

La propuesta realizada por J. Amanatides [AMAN84] se basa en la utilización de conos. Un cono está representado por un vértice, un eje central que define la dirección y un ángulo. En el caso de que el cono choque con algún objeto se calculan las direcciones de reflexión y refracción (eje central de los nuevos conos) utilizando las técnicas convencionales del trazado de rayos. El cálculo del vértice y del ángulo dependen de la superficie del objeto. Esta propuesta no solo pretende acelerar el trazado de rayos clásico, sino que amplia los efectos lumínicos que es capaz de presentar (añade básicamente el tratamiento de penumbras). Debido ala dificultad de calcular la intersección entre un cono y un objeto cualquiera, el entorno solamente está compuesto e esferas, planos y polígonos.

Otra alternativa de rayo generalizado es el denominado "Pencil Tracing" [SHIN87]. En este caso se agrupan los rayos que se encuentran próximos a un rayo denominado "*rayo axial*". Un lápiz (pencil) esta formado por un rayo axial y un manojo de rayos ("rayo paraxial") situados próximos a él. Matemáticamente un lápiz se define por un vector 4D que representa las desviaciones en posición y dirección de los rayos paraxiales respecto al rayo axial. Las transformaciones de un lápiz pueden representarse mediante un sistema de matrices 4x4. Por tanto, en este caso la propagación de los rayos agrupados en un lápiz puede llevarse a cabo combinando el sistema de matrices con la definición matemática de las superficies. Esta

aproximación sólo es válida para superficies suaves, por tanto el pencil tracing solo es aplicable en aquellas partes de la escena donde no hay ejes o superficies discontinuas. En estos casos es necesario trazar rayos individuales.

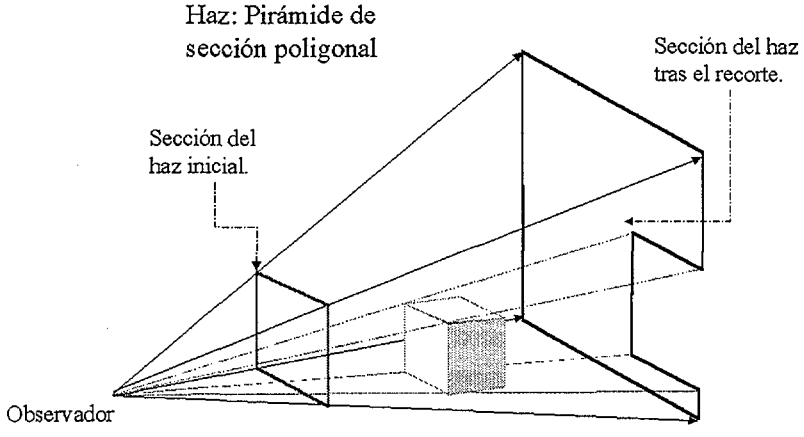



**Figura 2.5 Recorte de un haz al chocar con un objeto.**

La ultima gran alternativa sobre creación de rayos generalizados es la aportada por P. Heckbert y P. Hanrahan [HECK84]. Ellos proponen el denominado algoritmo de "Beam Tracing". En esta propuesta los rayos individuales son agrupados en haces. Un haz está formado por un conjunto de rayos que tienen un origen común y pasan a través de un polígono. Por tanto, la forma de un haz es la de una pirámide de sección poligonal.

La aplicación de este algoritmo se restringe a escenas en las que los objetos están definidos mediante polígonos. De este modo, un haz tras chocar con un objeto produce como resultado otro nuevo haz con una sección poligonal diferente (ver figura 2.5), resultado de recortar la sección del haz inicial con la silueta proyectada del objeto con el que choca. Esta restricción impuesta a la escena permite que los haces también puedan utilizarse para los rayos reflejados, ya que la reflexión en un plano es una transformación lineal. En este caso, el haz reflejado tiene como vértice

un nuevo observador virtual (ver figura 2.6). Por desgracia la refracción no se comporta generalmente como una transformación lineal. Por ello, este algoritmo impone una nueva restricción al no considerar el efecto de refracción puro sino realizar una aproximación de dicho efecto.

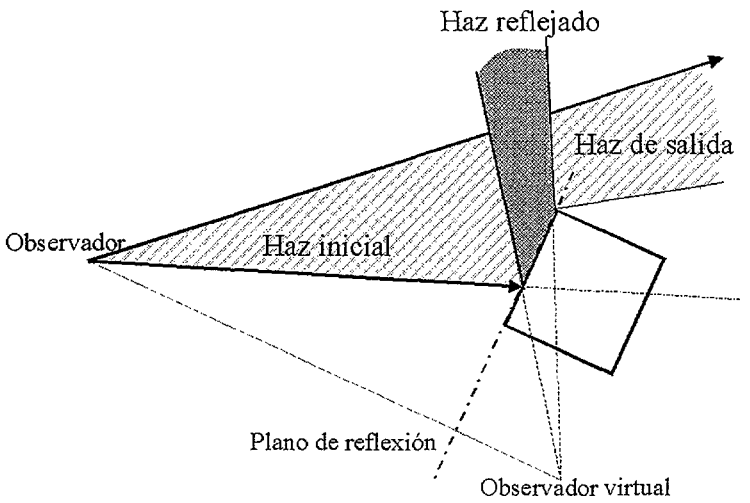



**Figura 2.6 Trazado de un haz.**

Teniendo en cuenta las restricciones impuestas anteriormente, este algoritmo traza los haces como si se tratarán de rayos, obteniendo un árbol de haces similar al árbol de rayos obtenido por el algoritmo estándar. En este proceso junto al cálculo de la intersección, al recorte y a la determinación de los nuevos haces, es necesario realizar una ordenación de los objetos que asegure que los cálculos se están realizando sobre el objeto más próximo al observador. En el cuadro siguiente se indican las líneas básicas del algoritmo propuestos por P. Heckbert y P. Hanrahan [HECK84]. En el podemos ver que necesita realizar operaciones sobre todos los polígonos de la escena lo que reduce su eficiencia cuando el número de polígonos que contiene la escena crece.

```
Beam-Trace
    Transformar la escena con respecto al sistema de coordenadas del haz (beam).
    Ordenar los polígonos de la escena atendiendo a criterios de proximidad.
    Para cada polígono de la lista de objetos.
        Calcular la intersección del haz con el polígono.
        Si existe intersección.
            Si límite de recursión no ha sido alcanzado.
                Si superficie del polígono es reflectante.
                    Crear el nuevo haz en la dirección de reflexión y trazarlo.
                Si superficie del polígono es refractante.
                    Crear el nuevo haz en la dirección de refracción y trazarlo.
        Añadir el polígono a la lista de polígonos intersecados.
        Calcular la diferencia entre el haz y el polígono.
    Trazar el nuevo haz resultado de los sucesivos recortes realizados.
```

**Gráfico 2.5 Pseudocódigo del algoritmo de *Beam-Tracing*.**

## 2.3 El paralelismo y el trazador de rayos.

La aparición de los primeros procesadores paralelos abrió una nueva expectativa de aceleración de este algoritmo. Las tremendas necesidades de cálculo que este algoritmo impone, incitan a pensar en distribuir la carga en diferentes máquinas o procesadores, reduciendo, de ese modo, el tiempo global ejecución.

Esta idea fue rápidamente utilizada por aquellos que pretendían, desde la perspectiva de los gráficos generados mediante ordenador, diseñar algoritmos de síntesis de imágenes que en "tiempo razonable" obtuvieran imágenes de gran calidad. Pero la paralelización de este algoritmo no solo ha servido para acelerar el proceso de síntesis sino que este algoritmo se ha convertido, como apuntan Keates y Hubbold [KEAT94], en un benchmark o test de prueba frecuentemente utilizado para demostrar las ventajas de una determinada arquitectura. Todo esto hace que se

encuentren bastantes referencias de trabajos que implementan una nueva versión paralela o que analizan su comportamiento en una determinada arquitectura.

Todos estos trabajos pretenden aprovechar la independencia existente entre los diferentes rayos a trazar dentro del proceso de generación de una imagen. Así, cada rayo puede procesarse de modo independiente al resto de los que componen la imagen.

Como podemos apreciar esta idea de paralelismo choca frontalmente con la idea de coherencia anteriormente expuesta, ya que en este caso se pretende explotar la independencia de los rayos y no las dependencias existentes entre ellos. Como indica S. Whitman [WHIT92], esta perdida de coherencia se agudiza cuando crece el número de procesadores utilizados para generar la imagen. En cualquier caso, la idea de coherencia debe tenerse en cuenta en las diferentes tareas asociadas al diseño del nuevo algoritmo paralelo. Es especialmente importante a la hora de plantearse la distribución de la carga, tanto si se plantea solamente una distribución de la imagen, como sí además se realiza una distribución de la escena.

Antes de entrar de lleno en el estudio de las diferentes propuestas de paralelización vamos a realizar un rápido repaso a los distintos tipos de arquitecturas paralelas, analizando su aplicación dentro de la paralelización del algoritmo de trazado de rayos.

## 2.3.1 Tipos de arquitecturas paralelas y su aplicación en el trazado de rayos.

### 2.3.1.1 Clasificación de las arquitecturas paralelas.

A la hora de abordar el estudio de las máquinas paralelas debemos partir de algún tipo de clasificación que facilite la obtención de grupos mas o menos

homogéneos y nos permita poder estudiar las diferentes soluciones de modo conjunto.

Se han propuesto diferentes tipos de clasificaciones de las arquitecturas [HWAN88], pero las más ampliamente aceptada es la propuesta por M.J. Flynn a mediados de los sesenta. Esta clasificación se basa en la multiplicidad de los flujos de instrucciones y datos en un sistema computador. Partiendo de esta clasificación encontramos cuatro grandes grupos de arquitecturas: SISD, SIMD, MISD, MIMD.

SISD (Simple flujo de instrucciones-simple flujo de datos):

Dentro de este grupo se encuadran las arquitecturas más antiguas asociadas al modelo propuesto por Von Neumann. Estas máquinas están compuestas por una memoria central y un único procesador, por tanto en ellas los algoritmos se ejecutan de modo secuencial.

SIMD (Simple flujo de instrucciones-múltiples flujos de datos):

En este tipo de arquitecturas existe una única unidad de control y múltiples unidades de tratamiento. En ellas todas las unidades de tratamiento efectúan exactamente la misma operación en el mismo instante pero sobre datos diferentes. Por tanto, todas las operaciones paralelas de una instrucción se ejecutan de manera perfectamente síncrona bajo la supervisión de la unidad de control.

MISD (Múltiples flujos de instrucciones-simple flujo de datos ).

En este tipo de arquitecturas existen *"n"* unidades de procesamiento, cada una de las cuales recibe distintas instrucciones que operan sobre el mismo flujo de datos derivados. En este grupo, como indican C. Germain-Renaud y J.P. Sansonnet [GERM93], se encontrarían las máquinas que implementan el modelo pipe-line y los supercomputadores vectoriales. Aunque como ellos proponen el efecto pipe-line

puede asemejarse al efecto *múltiple flujo de datos* y clasificarse este tipo de arquitecturas dentro del grupo anterior SIMD.

MIMD (Múltiples flujos de instrucciones-múltiples flujos de datos ).

En este tipo de arquitecturas no solo están repetidas las unidades de tratamiento sino también las unidades control, como lo cual los procesadores son capaces de ejecutar programas independientes. Los procesos que se ejecutan en distintos procesadores se comportan como si se tratara de ordenadores independientes funcionando de modo asíncrono.

En esta clasificación Flynn considera que todos procesadores acceden a una memoria central. Esta consideración realizada por Flynn a finales de los sesenta hoy en día no es totalmente correcta ya que los avances tecnológicos han permitido diferentes modos de realizar la gestión y el acceso a memoria. En este caso, podemos hablar de sistemas con *memoria compartida* (multiprocesadores) en el esquema clásico de Flynn o de *memoria distribuida* (multicomputadores) en cuyo caso cada procesador dispone de una memoria local y se comunica con otros procesadores mediante el envío de mensajes a través de una red de interconexión que interrelaciona a los diferentes procesadores. Según la topología de la red utilizada para el intercambio de información, podemos encontrar diferentes propuestas arquitectónicas: bus, anillo, malla, toro o hipercubo. Por último, en este tipo de arquitecturas con memoria distribuida existen nuevas implementaciones, denominadas *MIMD con memoria compartida virtual,* que consideran un único espacio de direccionamientos y por tanto hacen transparente para el programador la existencia de distribución de los datos en diferentes memorias aisladas.

Pero estas ideas de paralelismo no solo pueden implementarse en máquinas paralelas específicas, sino que se puede pensar también en utilizar redes de estaciones de trabajo para llevar a cabo dicho paralelismo. Esta propuesta es más

interesante si pensamos en que en la actualidad están surgiendo *redes de ordenadores de altas prestaciones* ([ANDE95], [BODE95], [LANG98],...) que aceleran notablemente el envío de información entre diferentes estaciones de la red, facilitando de este modo la incorporación de las redes al procesamiento paralelo.

#### 2.3.1.2 Aplicación de las diferentes arquitecturas paralelas al trazado de rayos.

Como hemos indicado anteriormente, el algoritmo de trazado de rayos es un algoritmo fácilmente paralelizable. Este hecho ha motivado que existan soluciones paralelas asociadas a los diferentes tipos de arquitecturas descritas anteriormente. Podemos encontrar trabajos referentes a la paralelización del trazado de rayos en todos los tipos de sistemas paralelos definidos hasta el momento. Junto al estudio realizado por Arvo y Kirk [ARVO89], en el trabajo de Reinhard, Chalmers y Jansen [REIN98] puede encontrarse un interesante análisis de las diferentes propuestas realizadas hasta el momento sobre paralelización del algoritmo de trazado de rayos.

De este modo, podemos encontrar soluciones en arquitecturas de tipo vectorial ([MAX81], [PLUN85], etc.) o en otras de tipo SIMD ([MOSH85], [DORB86], [WILL88], [LIN91], etc.). Aunque, como apuntan Foley et. al [FOLE90], estas arquitecturas no parecen las más indicadas para paralelizar el trazado de rayos. Esto ha motivado que la mayoría de las soluciones aportadas se desarrollen sobre arquitecturas de tipo MIMD ([ULLN83], [DIPP84], [GOLD85], [CLEA86], [AKTK87] [KOBA87], [BOUA88], [GAUD88], [CHAL89], [GREE89], [JENS89], [PRIO89], [BADO90], [CART90], [GREE90], [PADD92], [AYKA93], [LEFE93], [BADO94], [KEAT94], [MARI94], [VERD96a]...). Dentro de las implementaciones sobre arquitecturas tipo MIMD, existen propuestas tanto en sistemas de memoria compartida, como distribuida. A su vez, en ellas la red de interconexión tiene diferente topología encontrándose implementaciones en hipercubo, malla, etc.

Junto a las propuestas más comunes sobre multiprocesadores o multicomputadores, en la actualidad han surgido nuevos sistemas paralelos en los que se implementa memoria compartida aunque ésta está físicamente distribuida entre los diferentes procesadores. Estas nuevas arquitecturas son bastante interesantes pues permiten tener accesible desde cualquier procesador toda la infomación de la escena a la vez que ofrecen la posibilidad de trabajar con escenas muy complejas formadas por cientos de miles de objetos al disponer de mas memoria para su almacenamiento y gestión. Estos beneficios potenciales de estas nuevas arquitecturas están haciendo que algunas de las últimas implementaciones paralelas del algoritmo exploren las ventajas de su utilización [BADO94], [KEAT94], [MARI94].

Al mismo tiempo, junto a los anteriores entornos de trabajo más clásicos, más recientemente están surgiendo implementaciones sobre redes de estaciones de trabajo ([REIN97] [DAVI98], [ALFA98], [BART98]...). Estos nuevos desarrollos pretenden mostrar las posibilidades que las redes de estaciones de trabajo tradicionales o las nuevas redes de altas prestaciones ([ANDE95], [LANG98],...) pueden introducir en el desarrollo de versiones eficientes y de reducido coste de este algoritmo. Estas nuevas versiones se apoyan en las grandes expectativas en cuanto al ratio coste-redimiento que estas nuevas arquitecturas ofrecen y que pueden permitir que las soluciones paralelas finalmente se difundan a un amplio numero de usuarios. En cualquier caso estos entornos pueden esconder nuevos problemas a la hora de realizar el balanceo de la carga ya que la potencia de cálculo de las estaciones asociadas a la red puede ser diferente, o bien algunas de ellas pueden estar ejecutando al mismo tiempo otras aplicaciones. Como veremos en el siguiente apartado, en este nuevo entorno los algoritmos de balanceo de la carga deben adaptarse para tener en cuenta el comportamiento del algoritmo en sistemas heterogéneos y con carga.

### 2.3.2 Alternativas de paralelización del algoritmo y balanceo de la carga.

Al igual que otros algoritmos de visualización o síntesis de escenas tridimensionales (rendering) [WHIT92], dentro del algoritmo de trazado de rayos se pueden definir dos grandes alternativas de paralelización: *subdivisión de la imagen* (image space subdivision), *subdivisión de la escena* (object space subdivisión). La primera de las alternativas realiza una distribución de los diferentes pixel que componen la imagen entre los diferentes procesos. La segunda, en cambio, realiza una distribución de los objetos que componen la escena entre los diferentes procesos. Esta última alternativa es la más común en los algoritmos que se desarrollan para architecturas de memoria distribuida con paso de mensajes (multicomputadores).

En cualquiera de los dos casos siempre que se plantea distribuir el procesamiento entre diferentes procesadores es importante estudiar dicho reparto con el objetivo de que se produzca un eficiente *balanceo de la carga*. Un balanceo eficiente pretende que la carga asignada a cada proceso sea tal que todos ellos terminen en el mismo instante. De este modo, el objetivo fundamental de todo análisis de distribución de la carga pretende asignar una carga de trabajo semejante a cada uno de los procesos.

Este tipo de análisis puede complicarse si el sistema está compuesto por procesadores de diferentes características o alguno de ellos está siendo utilizado de modo simultáneo por otros procesos [GONZ99]. Como veremos a continuación el balanceo de la carga puede hacerse de dos modos, surgiendo el balanceo *estático* y el *dinámico*. En el primero de ellos, el proceso que distribuye la carga (host) realiza una distribución completa antes de que ningún procesador comience con el procesado de su carga. En cambio en el balanceo dinámico, el proceso host realiza una primera distribución parcial de la carga y el resto lo va asignando a los procesos que le notifican que han terminado de procesar la carga previa.

En el gráfico 2.6 se describen las principales propuestas de paralelización del algoritmo de trazado de rayos. A su vez, en cada caso se indican las alternativas que deben tenerse más presentes en cada supuesto. Como podemos apreciar en el caso de la subdivisión de la imagen el factor más importante es el tipo de balanceo estático o dinámico ya que todos los procesos tienen accesible la totalidad de la escena y por tanto tan solo podemos manejar el número de pixel que analizará cada proceso.

- Subdivisión de la imagen.
  - Estrategias de balanceo.
    - Estático.
    - Dinámico.
- Subdivisión de la escena.
  - Paso de información entre procesos.
    - Comunicación de rayos.
    - Comunicación de objetos (aproximación a memoria compartida virtual).
  - Criterio de descomposición.
    - Descomposición guiada por los objetos.
    - Descomposición guiada por el espacio.
  - Estrategias de balanceo.
    - Estático.

**Gráfico 2.6 Principales alternativas de paralelización del trazado de rayos.**

En cambio en las propuestas basadas en la subdivisión de la escena las características a tener en cuenta aumentan. En este caso la distribución de los objetos obliga a pensar el modo de repartir los objetos entre los diferentes procesadores. A su vez, el desconocimiento de la totalidad de la escena por parte de un proceso obliga a que exista la necesidad de solicitar información a otros procesos o enviarles datos para que ellos se encarguen de continuar son el trazado. Por último, aunque también aparecen en la bibliografía estrategias de balanceo tanto estático como

dinámico, la mayoría de las soluciones aportadas se apoyan en esta segunda alternativa.

A continuación, vamos a analizar los dos tipos de distribución y para cada uno de ellos algunas de las principales alternativas propuestas hasta el momento.

### 2.3.2.1 Subdivisión de la imagen.

En el primer esquema, *subdivisión de la imagen*, la totalidad de los pixel que forman el dominio del espacio de la imagen se descomponen en subdominios. Entonces cada subdominio es asignado a un proceso diferente el cual se encarga de realizar todos los cálculos asociados a los pixel de dicho subdominio. Para ello, cada proceso debe tener acceso a la totalidad de los objetos de la escena, ya que de otro modo no podría determinar de modo completo el color de los pixel de los cuales es responsable. Si el sistema tiene una única memoria compartida por todos los procesadores (es de tipo multiprocesador) el problema está resuelto, pero en caso contrario (es de tipo multicomputador) el problema se complica ya que cada procesador tiene asociada una memoria local propia. En este caso, el modo más simple de facilitar el acceso a la totalidad de la escena es duplicar completamente la estructura de datos que almacena la escena en la memoria local de cada procesador ([NISH83], [NARU87], [ISLE91],...). Pero esta alternativa puede limitar el tipo de escenas que el sistema es capaz de procesar ya que la memoria de los diferentes procesadores debe ser suficientemente grande como para almacenar la totalidad de la escena.

Actualmente los nuevos sistemas distribuidos de memoria compartida virtual ([LI89], [BADO94], [KEAT94], [MARI94]...) solventan el problema de la accesibilidad de la escena y del tamaño de la memoria local, ya que cualquier procesador conectado al sistema puede direccionar zonas de memoria situadas en cualquiera de los otros procesadores. Esta nueva alternativa puede asegurar una distribución de los objetos y un acceso eficiente de todos los nodos a la totalidad de la escena, siendo, por tanto, una alternativa intermedia entre la subdivisión de la

imagen y la subdivisión de la escena.

La asignación de los rayos a los diferentes procesos es una tarea delicada ya que debemos intentar que la carga esté lo más balanceada posible. Esta asignación puede realizarse de modo estático o dinámico. En el esquema estático los diferentes pixel son distribuidos antes de comenzar la ejecución del algoritmo. En cambio en el esquema dinámico los pixel se van asignando a los diferentes procesos conforme van acabando su trabajo previo

En el **esquema estático** se debe predecir a priori la carga de cada zona de la imagen. Sin embargo, una descomposición y asignación uniforme de la totalidad de los pixel entre los diferentes procesos no garantiza una distribución uniforme de la carga. Esto es debido a que el tiempo de procesamiento de cada rayo puede ser bastante diferente, dependiendo de la localización y la configuración de los objetos de la escena.

El modo más simple de realizar una distribución estática de la imagen se denomina *método contiguo* [HU85]. La imagen es dividida en áreas contiguas de igual tamaño y cada área se asigna a un procesador diferente. En este caso, lo más probable es que cada área requiera un tiempo de procesamiento diferente. Esto puede provocar que algunos procesos terminen su procesamiento antes que otros y, por tanto, un mal balanceo de la carga. En la figura 2.7 (a) se muestra la distribución utilizando el método contiguo para una imagen que ha sido distribuida entre 8 procesos. Como podemos apreciar el área asignada al proceso $P_1$ no contiene ningún objeto luego dicho proceso finalizará rápidamente. En cambio el área asignada a los procesos $P_6$ y $P_7$ tiene una alta concentración de objetos, con lo cual dichos procesos necesitarán bastante más tiempo para procesar las filas asignadas.

Como apuntan Isler y Ozguc [ISLE91] el problema anterior puede resolverse aplicando la distribución denominada *método entrelazado*. Este modo de

distribución tiene en cuenta, como apuntan los autores, la coherencia de imagen por medio de la cual se puede suponer que pixel vecinos necesitarán un tiempo de procesamiento similar. Propone dividir la imagen entre los *P* procesos de tal manera que al proceso *i* se le asignen las líneas *i, i + p, i + 2p, i + 3p*, etc. (figura 2.7 (b)). Esta distribución intenta asegurar que a cada proceso se le asignarán líneas distribuidas uniformemente en la imagen y por tanto la probabilidad de que la carga asignada a cada proceso sea similar es bastante alta. Esta alternativa obtiene en la mayoría de los casos un buen balanceo de la carga.

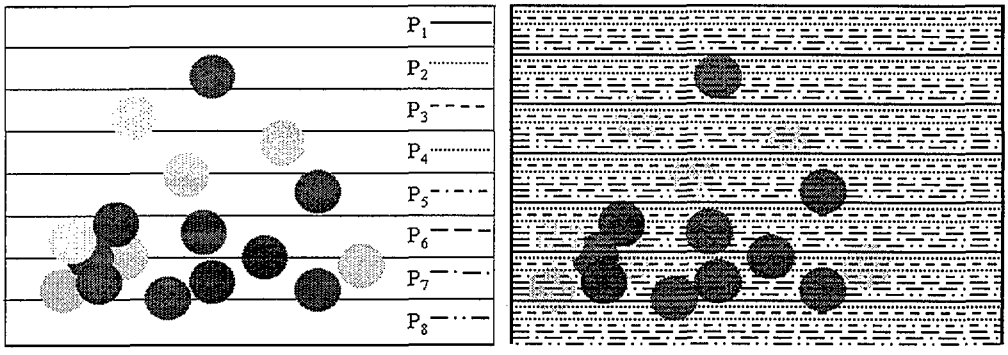



**Figura 2.7 Diferentes modos de balanceo de carga dentro de la alternativa de paralelización de subdivisión de la imagen. a) método contiguo; b) método entrelazado.**

En cualquier caso, esta última estrategia de balanceo que en entornos homogéneos (procesadores con las mismas características y dedicados en exclusiva al proceso de trazado) funciona perfectamente cuando el número de procesadores es inferior a 128, no lo hace igualmente bien para mayor número de procesadores [HEIR98] o cuando se analizan sistemas heterogéneos [GONZ99]. En el primer supuesto, la utilizan de un gran número de procesadores provoca que la aceleración (*speed-up*) no sea progresiva sino que se produzcan picos a intervalos constantes, intervalos que están asociados al número de pixel a procesar. Por otra parte, la utilización de este tipo de balanceo en entornos heterogéneos (entornos en los que los procesadores son de diferente tipo o están atendiendo a otros programas al

mismo tiempo) provoca que al tratar cada procesador un número distinto de pixel la asignación no esté correctamente balanceada, al no asociar pixel de la totalidad de la escena (ver figura 2.8).

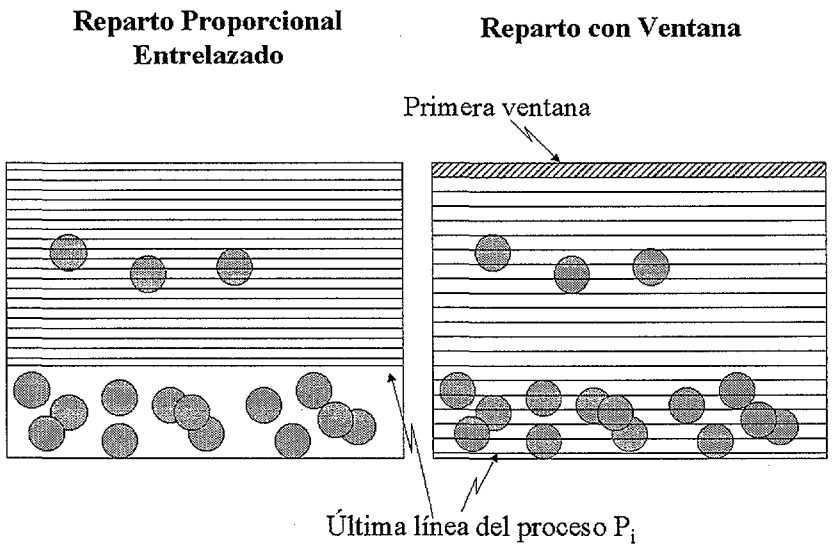



**Figura 2.8 Diferencias entre el reparto entrelazado y mediante la utilización de ventanas en entornos heterogéneos.**

Los métodos anteriores parten de una estimación apriorística de cómo será la distribución de los objetos en la escena, pero desconocen cual es la distribución real de la misma. Esto puede dar lugar, en ciertas escenas, a pequeños problemas en el balanceo de la carga. En este caso si se desea partir de una estimación de la carga real de la escena se puede utilizar otro nuevo método que propone realizar un *preprocesado de la escena* con una imagen de baja calidad ([SALM88], [PRIO89],...), lo que les permite observar la distribución del tiempo de procesado en las diferentes partes de la escena. Este método propone por tanto trazar un número reducido de rayos y apoyándose en la coherencia de rayos suponer que rayos próximos seguirán caminos semejantes dentro de la escena y por tanto necesitarán un tiempo de procesado similar.

Junto a los métodos estáticos, dentro de la literatura podemos encontrar también referencias a **métodos dinámicos** ([KOBA88a], [ISLE91], [BADO94],...).

En ellos el proceso host, encargado de la distribución, realiza una primera asignación de un número reducido de pixel entre los diferentes procesos, los cuales se encargan de procesarlos. Cada vez que uno termina de procesar los pixel que le habían asignado, se comunica con el host enviando el resultado y solicitando otro conjunto de pixel. En este caso aunque el balanceo puede considerarse perfecto, ya que cada proceso solicita pixel cada vez que termina de procesar los anteriores, tiene el inconveniente de que pueden crecer bastante las comunicaciones entre el proceso host que distribuye y los procesos trazadores. Por tanto, hace necesario que en vez de enviar un pixel cada vez se envíe un paquete conteniendo un número dado de pixel.

En este tipo de algoritmos lo más importante es determinar el tamaño del área a asignar a cada proceso cuando realizan una solicitud de nuevos pixel. La definición de bloques de gran tamaño incrementa el rendimiento del algoritmo ya que reduce el número de comunicaciones necesarias entre los nodos y el host. Por otra parte, esta asignación de bloques de gran tamaño lleva consigo el mismo problema que se ha apuntado para el método contiguo, provocando que en este caso el balanceo de la carga sea generalmente bastante bajo. D. Badouel, K. Bouatouch y T. Priol [BADO94] consideran que un área de tamaño 3x3 es una propuesta que consigue un buen compromiso entre los dos criterios anteriores para un sistema MIMD con memoria compartida virtual.

Por otra parte, este tipo de alternativas de balanceo dinámico se comportan bastante bien ante situaciones en las que se desconocen las características del sistema donde se va a procesar el algoritmo o están cambian o pueden hacerlo durante su ejecución, esto es, lo que hemos denominado entornos heterogéneos. Este correcto comportamiento se debe a que cada proceso se autoregula y solo solicita los pixel que es capaz de atender, ya que hasta que no termina de procesar los pixel anteriores no solicita más.

### 2.3.2.2 Subdivisión de la escena.

En la segunda alternativa, *subdivisión de la escena*, el espacio que contiene a los objetos es subdividido y almacenado en la memoria local de cada procesador. Este tipo de alternativa necesita, por tanto, comunicación entre los diferentes procesadores, ya que un procesador por sí solo no puede determinar el color asociado a un pixel. En este caso podemos encontrar dos grandes alternativas: *movimiento de objetos* o *movimiento de rayos*.

En la primera alternativa la estructura que contienen a la escena es distribuida entre la memoria local de los diferente procesadores ([GREE89], [GREE90], [BADO89], [POTM89],...). Cada procesador genera un conjunto de rayos asociados a los pixel que le han sido asignados. Cuando un procesador, para completar el trazado de un rayo, necesita una parte de la escena que no está almacena en su memoria local realiza una petición al procesador donde está almacenada dicha parte. Entonces la información solicitada es enviada y almacenada en la memoria local del nodo que lo ha solicitado.

Como podemos apreciar, esta propuesta puede considerarse como una alternativa intermedia entre los sistemas de memoria distribuida puros y los sistemas de memoria distribuida con memoria compartida virtual. En este caso, la memoria local se convierte es una especie de cache y contiene los objetos utilizados últimamente por los rayos que el procesador está encargado de procesar. En este caso la escena está almacenada de modo completo en el procesador host el cual la descompone utilizando alguna de las técnicas de subdivisión espacial (volúmenes envolventes, árboles octales, etc.) y envía al resto de procesadores la estructura general de la escena y un conjunto de objetos asociados a un área de la misma.

La segunda alternativa, **movimiento de rayos**, propone realizar una comunicación de rayos entre los diferentes procesadores. Aquí, al igual que en el caso anterior, se propone realizar una división de la escena de tal modo cada

procesador contenga una parte de los objetos. Las dos principales propuestas para distribuir los objetos entre los diferentes procesadores dependen de la estructura de datos seleccionada para ello. Las estructuras a utilizar son las mismas que se utilizan en la coherencia de objetos para estructurar la escena. La descomposición puede realizarse: *guiada por los objetos* ([SALM88], [CASP89],...) o *guiada por el espacio* ([DIPP84],[CLEA86],[KOBA88b], [CAUB88], [PRIO89], [ISLE91], [AYKA94],...)

Si el proceso de descomposición es **guiada por los objetos** se obtiene una jerarquía de volúmenes envolventes que contienen a los objetos de la escena. En este caso la estructura resultante es tal que un objeto estará contenido en un único procesador. En cambio, si se utiliza una descomposición guiada por el espacio, mediante cualquier técnica de subdivisión espacial (uniforme o no uniforme), es posible que un mismo objeto esté compartido por dos partes de la estructura y por tanto replicado en varios procesadores. En cualquiera de los dos casos todos los nodos conocen la estructura general de descomposición de la escena y a su vez almacenan una parte de los objetos.

En estas alternativas, cada vez que un rayo abandona la parte de la escena almacenada en la memoria del procesador encargado de trazarlo, dicho procesador envía un mensaje al procesador que contiene el punto de salida del rayo. Ahora el procesador que recibe el mensaje es el encargado de trazarlo. Este proceso implica que se establezca un algoritmo de finalizado que determine cuando se ha recibido toda la información para calcular el color final de un pixel. Esto es debido a que para calcular el color final de un pixel necesitemos componer datos de rayos analizados por diferentes procesadores ya que durante el trazado se pueden generar nuevos rayos debidos a las características de reflexión y refracción de las superficies que generan caminos independientes entre sí.

En todos las propuestas que implican una distribución de la escena también se realiza un reparto y asignación de los pixel que componen la imagen entre los diferentes procesadores. Dicho reparto de pixel suele venir guiado por la asignación

previa de una zona de la escena. Por tanto, aunque se pueden tener presente las alternativas analizadas en los algoritmos de subdivisión de la imagen, en este caso el balanceo de la carga se conseguirá básicamente mediante una correcta distribución de los objetos entre los diferentes procesadores.

En este tipo de algoritmos la tarea más difícil, y no siempre tratada en todas las alternativas propuestas, es la obtención de un correcto balanceo de la carga. En la mayoría de los casos el balanceo propuesto es de tipo estático y las soluciones son muy variadas y generalmente adaptadas a la estructura de descomposición de la escena. Así H. Kobayashi et al. realizan la descomposición de la escena mediante árboles octales y asignan a cada procesador volúmenes que contienen un número semejante de objetos [KOBA87]. Una alternativa similar a la anterior es la aportada por C. Aykanat et al. [AYKA94] en la cual el propio proceso de descomposición pretende obtener una subdivisión balanceada de la escena. En este caso la estructura utilizada es una generalización de los BSP que denomina BBSP ("balanced binary space partitioning").

Tal vez la solución más utilizada es aquella que propone realizar un preprocesado que permita determinar cual es la distribución de los objetos dentro de la escena ([GREE89], [PRIO89], [GREE90], [BADO90],...). Para ello trazan sobre la escena un número reducido de rayos que permitan, apoyándonos en la coherencia de rayos, estimar el coste asociado a un área de la escena.

Otros trabajos realizan una distribución del espacio semejante al modo en que distribuye los pixel el método entrelazado. En este caso, se realiza una distribución del espacio en zonas pequeñas y varias de ellas se asignan al mismo procesador, pero teniendo en cuenta que las zonas asignadas no sean contiguas ([CAUB88], [KOBA88b],...)

Aunque menos numerosos, también podemos encontrar trabajos que realizan

un balanceo dinámico ([DIPP84], [NEMO86],...). En estos casos la redistribución de los mensajes implica un movimiento de los límites de las regiones asignadas a cada procesador. Pero estas alternativas junto al aumento del tráfico requieren de heurísticas complicadas de implementar y no muy eficientes.

En cualquier caso, como indican D. Badouel, K. Bouatouch y T. Priol [BADO94], es interesante resaltar que las técnicas de subdivisión de la escena tienen dos grandes problemas: (1) dificultad en el diseño de los algoritmos y el balanceo de la carga, (2) eficiencia que decrece al aumentar el número de procesadores. Estos problemas hacen que su aplicación sea cada vez menor, máxime con la aparición de MIMD de memoria compartida virtual y el aumento en las posibilidades de memoria a bajo costo que podemos encontrar en una red de estaciones de trabajo.

# Capítulo 3. Trazado de haces de rayos en escenas estructuradas espacialmente mediante árboles octales.

## *3.1 Introducción.*

En el capítulo anterior, hemos analizado de manera detallada las diferentes alternativas propuestas hasta el momento para acelerar el algoritmo de trazado de rayos. Como hemos podido apreciar, las principales ideas de aceleración provienen de la utilización de las características de coherencia y de la paralelización del algoritmo. Por otra parte, si nos marcamos como objetivo la optimización de la generación de una imagen mediante procesamiento secuencial, las alternativas de aceleración del algoritmo más utilizadas y que, por tanto, ofrecen unas mayores ventajas potenciales son la coherencia de objetos y la coherencia de rayos. La primera de ellas permite guiar el trazado de un rayo dentro de la escena, mientras

que las segunda facilita el trazado de varios rayos a la vez. En cualquier caso, como ya hemos apuntado anteriormente, aunque es posible encontrar abundantes referencias que utilicen de modo aislado uno u otro tipo de coherencia, no sucede lo mismo si buscamos alternativas que las utilicen de modo conjunto y, por tanto, aprovechen las ventajas que ambas aportan.

La unión de ambas técnicas ofrece una alternativa de aceleración que aprovecha las ventajas de las tres grandes corrientes de aceleración propuestas por J. Arvo y D. Kirk [ARVO89] (ver grafico 3.1): (1) cálculo más rápido de intersecciones; (2) menor número de rayos; (3) rayos generalizados. El "cálculo más rápido de intersecciones", se consigue utilizando la estructura de árbol octal para descomponer la escena, facilitando, de ese modo, el seguimiento de la trayectoria de un rayo. La utilización de "rayos generalizados", se logra al definir haces de rayos que permiten mover de modo simultáneo a un conjunto de rayos dentro de la escena. Por último, la idea de "menor número de rayos" se lleva a cabo mediante la generación de un número de rayos diferente en cada zona de la imagen en función de la concentración de objetos, facilitando, de este modo, la resolución del problema de aliasing.

❶ Cálculo más rápido de intersecciones.
- Subdivisión del espacio mediante árboles octales.

❷ Menor número de rayos.
- Generar más rayos en zonas densamente pobladas o lejanas (optimización de los problemas de aliasing).

❸ Rayos generalizados.
- Trazado de rayos agrupados en haces.

**Gráfico 3.1 Características básicas del trazado de haces.**

La idea básica del trazado de haces es descomponer la escena mediante un árbol octal y utilizar esta estructura de descomposición para guiar el trazado de los haces dentro de ésta. Para ello, el algoritmo que aquí se propone se apoya básicamente en los trabajos de H. Samet sobre la definición y utilización de los árboles octales dentro del trazado de rayos ([SAME89], [SAME90b]) y en la propuesta de P. Heckbert y P. Hanrahan [HECK84] sobre el trazado de haces (beam tracing).

En el algoritmo que aquí se presenta se eliminan las restricciones impuestas por el algoritmo "beam tracing" a la hora de definir la escena (escenas formadas tan sólo por objetos modelados mediante mallas de polígonos). En este caso, a diferencia de la propuesta previa, los objetos de la escena pueden definirse utilizando cualquier tipo de primitivas. A cambio de esta ventaja, el tipo de rayos a los que se pueden aplicar los haces se reduce tan sólo a rayos primarios. Esto que inicialmente puede parecer un gran inconveniente se ve mitigado ya que, por una parte, el número de rayos reflejados o transmitidos es generalmente menor que el de rayos primarios, y por otra parte el trazado individual de los rayos secundarios se está acelerando al utilizar el árbol octal para facilitar el seguimiento de su trayectoria. En cualquier caso es interesante resaltar que, si se impusiese la misma restricción en cuanto al modelado de los objetos de la escena, los haces de rayos se podrían utilizar también para rayos reflejados y transmitidos, aplicando las misma técnicas propuestas por P. Heckbert y P. Hanrahan [HECK84].

En los siguientes apartados se describirá de manera detallada el algoritmo propuesto y los principales procesos que éste incluye.

## 3.2 Análisis del algoritmo propuesto.

Este algoritmo realiza de modo simultáneo el trazado de un conjunto de rayos

agrupados en un haz. Para ello se definen una serie de haces que van descomponiéndose en otros de menor tamaño conforme avanzan dentro de la escena. Este proceso de subdivisión de los haces está controlado por la estructura utilizada para descomponer la escena.

Como ya hemos indicado anteriormente, la agrupación de rayos en haces se limita tan sólo a los rayos primarios. Esta limitación está motivada por el hecho de no imponer que los objetos de la escena deban definirse obligatoriamente mediante polígonos. De este modo al aumentar la libertad en la definición de los objetos de la escena (esferas, cilindros, conos, superficies algebraicas, CSG, polígonos, etc.) las ideas de P. Heckbert y P. Hanrahan [HECK84] sobre el trazado de haces para rayos reflejados y transmitidos no son válidas. En cualquier caso, como ya apuntamos, esta limitación se ve amortiguada por dos hechos básicos: el número de rayos primarios suele ser generalmente mayor que el de rayos secundarios (muchos rayos primarios no chocan con objetos de la escena, o estos no tienen características de reflexión o de refracción) y por otra parte, el trazado de los rayos secundarios también está siendo acelerado al utilizar árboles octales [SAME89]. A su vez el algoritmo que nosotros presentamos no necesita analizar todos los polígonos de la escena en cada iteración (como sucede en la propuesta de Heckbert y Hanrahan, ver gráfico 2.5), sino tan sólo aquellos objetos contenidos en un nodo ocupado cuando un haz llega a dicho nodo.

Las estructuras básicas utilizadas en el algoritmo son los *árboles octales* como estructura de subdivisión de la escena y los *haces* como estructura de trazado dentro de la escena.

Un **árbol octal** es una representación jerárquica a través de la cual el espacio es subdividido en cada paso en ocho voxels (cubos) o nodos de igual tamaño, llamadas octantes (ver figura 2.3). Los nodos del árbol podrán ser de tres tipos: (1) *nodos intermedios*, es decir, aquellos que están divididos en otros de nivel inferior; (2) *nodos terminales vacíos*, es decir, aquellos nodos que no contiene en su interior ningún objeto; (3) *nodos terminales ocupados*, es decir, aquellos nodos que

contienen un número, generalmente reducido, de objetos de la escena.

Como vimos en el capítulo anterior existen varias alternativas para llevar a cabo las implementaciones que utilizan árboles octales. En el gráfico 3.2 aparecen detallados los diferentes criterios utilizados para implementar el trazado sobre árboles octales.

***Criterios utilizados en la implementación del trazado de haces en árboles octales:***

- Modo de almacenamiento de la estructura.
  - Enlaces.
- Instante de construcción del árbol octal.
  - Descomposición a priori.
- Modo en que se calcula el punto de entrada/salida (cálculo de la dirección de salida).
  - Métodos explícitos.
- Proceso de búsqueda del nodo que contiene a un punto dado (búsqueda de vecinos).
  - Métodos padre.

**Gráfico 3.2 Criterios utilizados en la implementación del trazado en árboles octales.**

La representación interna seleccionada para almacenar el árbol octal es una representación mediante enlaces. Para ello, cada nodo intermedio de nivel superior (nodo padre) contiene ocho enlaces a sus descendientes inmediatos (nodos hijo). Por otra parte, el método elegido para realizar la búsqueda de un nodo vecino (método padre) impone la inclusión de un nuevo enlace que nos permita acceder para cada nodo a su antecesor (nodo padre) dentro de la estructura de árbol.

A su vez el proceso de creación del árbol octal se propone realizarlo antes de comenzar su trazado, considerando, por tanto, una descomposición a priori. Este criterio de construcción permitiría utilizar la misma estructura para generar una

secuencia de imágenes en la que los objetos no se mueven y tan sólo cambia la situación del observador.

Por último, como veremos más adelante cuando analicemos el proceso de propagación de los haces, la determinación de la dirección de salida se apoya en técnicas explícitas. La utilización de este tipo de técnicas implica la necesidad de conocer la posición real de cada nodo del árbol. Para acelerar este proceso, cada nodo almacena como atributos la posición de uno de sus vértices y la longitud de sus aristas.

Un **haz** consta de una colección de rayos que tienen un origen común y pasan a través de un polígono plano que representa una sección del tronco de pirámide asociado al haz. Por tanto en este caso para definir un haz basta con indicar su *origen* y una *lista de vectores normalizados* que determinan su dirección y están asociados a los extremos del haz.

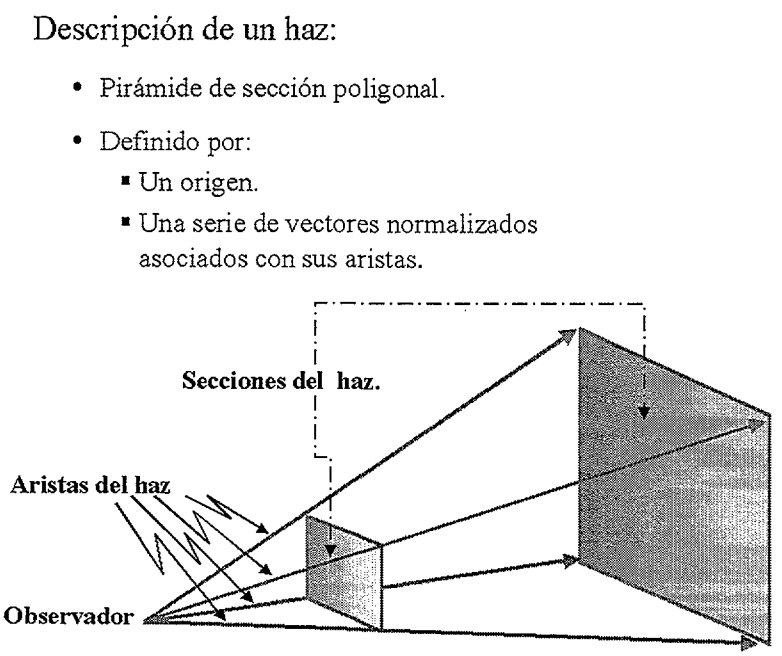



**Figura 3.1 Definición de haz.**

El proceso de trazado propuesto depende fuertemente del tipo de nodo que un haz se encuentre en su camino. Si el nodo está vacío la tarea básica es continuar el trazado del haz y obtener, en función de la estructura de subdivisión, los haces de salida que llegarán a los nodos vecinos. En cambio, si un haz se encuentra con un nodo ocupado por objetos se deberán determinar los pixel contenidos en el haz, generar los rayos asociados y calcular su intersección con todos los objetos incluidos en el nodo.

Podemos definir, por tanto, dos tareas básicas asociada cada una a un tipo de nodo: (1) generar la trayectoria de un haz dentro de la estructura del árbol octal; (2) analizar el comportamiento (el color) de los pixel asociados a un haz. El hecho de que en la mayoría de los casos se genere un único rayo por pixel hará que en el resto del estudio ambos términos puedan utilizarse de modo equivalente. En el cuadro 3.3 se describe de manera general el algoritmo propuesto.

**Algoritmo:**

**nodo vacío: (Generar la trayectoria del haz en la escena)**

Determinar las direcciones de salida dado un haz y un nodo.
Para cada dirección
    Obtener el haz o haces de salida y procesarlos de modo recursivo.

**nodo ocupado o bien el haz abandona la escena: (Analizar el comportamiento de los pixel asociados al haz)**

Determinar los pixel contenidos.
Para cada pixel
    Obtener el color final.

**Gráfico 3.3 Algoritmo general del trazado de haces.**

En las siguientes secciones se estudiarán de modo más detallado las tareas

principales del algoritmo. Este análisis se centrará en: la obtención de la estructura de subdivisión de la escena, la creación del haz inicial, la generación de la trayectoria de un haz y el análisis del comportamiento de los pixel asociados a un haz.

## 3.3 Obtención de la estructura de subdivisión de la escena.

Mediante el proceso de descomposición se pretende dividir y estructurar la escena asociando los diferentes objetos a los cubos que componen los nodos terminales del árbol octal. De las diferentes alternativas de generación posibles (estática, dinámica) en este caso se ha seleccionado la propuesta estática pero se han tenido en cuenta las ideas del algoritmo de generación dinámica [MCNE92] para limitar el nivel de profundidad del árbol generado. De este modo, la escena es analizada y subdividida antes de comenzar el propio proceso de trazado.

Para llevar a cabo la subdivisión de la escena los criterios aplicados generalmente son: (1) reducir el número de objetos contenidos en un nodo; (2) limitar el nivel de subdivisión. El primero de ellos pretende reducir el número de intersecciones a calcular cada vez que un rayo llega a un nodo. El segundo disminuye la cantidad de memoria necesaria para almacenar el árbol, a la vez que optimiza el recorrido de dicha estructura. De este modo dentro del proceso general de descomposición se suelen definir dos umbrales: número máximo de objetos contenidos y máximo nivel de subdivisión.

En nuestro caso, es interesante resaltar que las mayores ventajas del trazado de haces se obtienen cuando el haz va encontrando nodos vacíos, ya que en este caso no se analiza el comportamiento individual de los rayos contenidos en él, sino que se continúa con el trazado de dicho haz por la escena. Por tanto, el criterio básico

aplicado en el proceso de subdivisión será definir el mayor número de nodos vacíos.

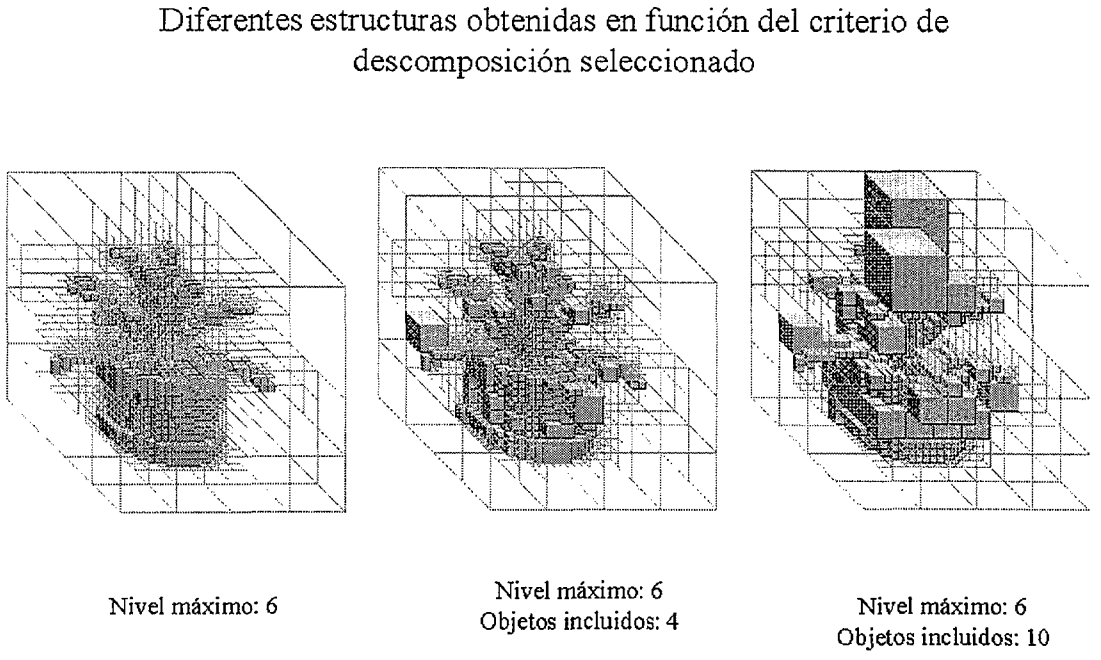



**Figura 3.2 Estructuración de una escena utilizando diferentes criterios de descomposición.**

En la figura 3.2 podemos ver la descomposición de una escena en la que se han utilizado diferentes criterios de subdivisión. En ella puede verse que al incluir el parámetro que considera el número mínimo de objetos contenidos en un nodo el tamaño de los nodos ocupados crece, con lo que el comportamiento de nuestro algoritmo será peor, como podremos comprobar en el capítulo de análisis de resultados.

En cualquier caso, en el proceso de generación de la estructura que nosotros proponemos seguimos manteniendo el umbral asociado al nivel máximo de descomposición. Esta última restricción viene impuesta por el hecho de que cuando un haz es de tamaño muy reducido, el número de rayos que contendrá será tan

pequeño que tal vez sea más rentable trazarlos de modo individual.

Para establecer el nivel máximo de descomposición nos vamos a apoyar en el trabajo de McNeill et al. [MCNE92]. En este se estudia el comportamiento de los árboles octales como estructura de descomposición y se analizan tanto las necesidades de memoria como la frecuencia con que se accede a cada nodo del árbol en función de su nivel. Si observamos algunas de las conclusiones a las que llegan, vemos que a partir del sexto nivel de descomposición el número de accesos a los nodos de dichos niveles (número de rayos que llegan a dichos nodos) es bastante bajo. Esta reducida frecuencia de acceso a los nodos de niveles inferiores hace pensar que si un haz llegase a dichos nodos la probabilidad de que contuviera un número reducido de rayos es alta y, por tanto, la rentabilidad de dicho haz sería escasa.

El proceso de descomposición parte de la definición de un cubo alineado con los ejes que contendrá a los objetos de la escena a representar. Para ello analizamos los diferentes objetos de la escena y calculamos la longitud de la arista mayor. Tras definir este cubo máximo inicial, comienza el proceso de subdivisión siguiendo el criterio anteriormente establecido. Este criterio establece que un nodo ocupado por objetos debe seguir descomponiéndose mientras no hayamos alcanzado el máximo nivel de subdivisión establecido. Cada vez que se crea un nuevo cubo de nivel inferior se debe comprobar si los objetos asociados al nodo padre están total o parcialmente contenidos dentro del nuevo cubo. Por tanto, la tarea básica necesaria dentro de este proceso de descomposición, es determinar si un objeto está contenido o interseca con el cubo asociado a un nodo. Esta tarea depende fuertemente de la superficie del objeto a analizar, por ello, es necesario asociar a cada clase de objetos una función que permita determinar esta pertenencia.

Dentro de este trabajo se han implementado tan sólo dos funciones asociadas a dos tipos de objetos: esferas y polígonos. Estos dos tipos de figuras permitirán probar la potencia del trazado de haces y su aplicación a figuras de diferente tipo. En

cualquier caso, es interesante resaltar que los algoritmos utilizados para estas dos figuras pueden ser extendidos a otro tipo de objetos.

A continuación se describirán las características básicas de los algoritmos utilizados para determinar la intersección entre el cubo asociado al nodo y una esfera o un polígono (una explicación más completa de los algoritmos de test intersección y algunas comparaciones de eficiencia entre ellos, puede encontrarse en [MOLI98] o en los artículos de los propios autores [ARVO90b], [RATS94], [VOOR92], [GREE95]).

### 3.3.1 Test de intersección cubo-esfera.

Estrictamente hablando, un cubo del árbol octal sólo debería hacer referencia a una esfera de nuestra escena si la región del espacio delimitada por dicho cubo contiene una parte o la totalidad de la superficie de esa esfera. Esto debe ser así dado que, durante el proceso de trazado de rayos, los cálculos de intersección que se realizan entre rayos y objetos se centran exclusivamente en la superficie de los propios objetos. Esto significa que el test de intersección debe considerar el cubo como un objeto sólido y la esfera como un objeto hueco. Un algoritmo que resuelve este problema lo podemos encontrar dentro de la familia de tests caja-esfera propuestos por Jim Arvo [ARVO90b]. En particular, el algoritmo que nos interesa está basado en el cálculo de las distancias mínima y máxima de los vértices de la caja al centro de la esfera, las cuales se comparan después con el radio de la esfera para resolver si hay o no intersección.

Ratschek y Rokne [RATS94], por su parte, ofrecen un algoritmo diferente inspirado en el análisis de intervalos, el cual unifica y realiza una unificación de los propios algoritmos de J. Arvo [ARVO90b]. Este algoritmo considera únicamente cajas cuyas aristas sean paralelas a los ejes de coordenadas, como es nuestro caso, de manera que pueda definirse cualquier caja por medio de tres intervalos, uno por cada

eje. El test consiste básicamente en aplicar una función que calcula la mínima y la máxima distancia desde cualquier punto de la caja a la superficie de la esfera, de forma que si el 0 queda entre ambos valores se deduce que hay intersección. La función de distancia que se toma como base para el algoritmo es la siguiente:

$$F(x) = \|x - c\|^2 - r^2 = \sum_{i=1}^{3} \left(x_i^2 - 2x_i c_i\right) + c \cdot c - r^2 = \sum_{i=1}^{3} f_i(x_i) + c \cdot c - r^2 \qquad (3.1)$$

en donde:

$x \in \mathrm{R}^3$ es un punto y $x_i$ es su coordenada $i$-ésima.

$c \in \mathrm{R}^3$ es el centro de la esfera y $c_i$ es su coordenada $i$-ésima.

$r$ es el radio de la esfera.

Calcular entonces el valor mínimo y el máximo de esa expresión para el conjunto de puntos que forman una caja, definida por sus tres intervalos, depende únicamente del valor mínimo y máximo de cada $f_i$ sobre el intervalo correspondiente de la caja. Y como cada función $f_i$ es una sencilla parábola, calcular el mínimo y el máximo sobre un intervalo dado resulta casi inmediato.

### 3.3.2 Test de intersección cubo-polígono.

La idea intuitiva es que un polígono intersecará un cubo siempre que uno de sus lados lo interseque o, en su defecto, el interior del polígono interseque al cubo. En primer lugar se comprobaría si uno de los extremos de los lados del polígono (vértices del mismo) está dentro del cubo o bien uno de sus lados atraviesa una de las caras del cubo. Si no es así, sólo podrá haber intersección si el polígono divide en dos el cubo. De este modo, en segundo lugar se calcularía la intersección de las cuatro diagonales del cubo con la superficie del polígono.

Una función que implementase el procedimiento anterior sería, aunque eficaz, costosa en cálculos. Al igual que sucedía con la esfera y el cubo, hay

situaciones en las que unas pocas comparaciones nos bastan para determinar que la intersección es imposible. La idea que se persigue es comprobar primero los casos más frecuentes y que menos cálculos requieran, y dejar en última instancia las situaciones menos probables y más costosas. Voorhies [VOOR92], Green y Hatch [GREE95] dan las claves para construir una rutina mucho más eficiente y que sirva tanto para polígonos convexos como cóncavos (con "intrusiones").

Si, por ejemplo, determinamos que todos los vértices quedan a la derecha del plano que contiene la cara derecha del cubo, es lógico inferir que no es posible la intersección. Así, una versión más elaborada empezaría con unos sencillos tests que tratarían de encontrar un plano que separe el cubo del polígono. Los tests triviales de Voorhies [VOOR92] prueban con los seis planos que contienen las caras del cubo, doce planos que pasan por las aristas y ocho que se apoyan en los vértices. Green y Hatch [GREE95] mejora el código eliminando comprobaciones innecesarias mediante el uso de máscaras de bits. Por ejemplo, si un vértice se encuentra a la derecha del cubo no es preciso comprobar si está a la izquierda de dicho cubo. O si un vértice está por encima del cubo y otro por debajo, entonces es inútil seguir comprobando si el resto de los vértices están por encima o por debajo, pues ya no podrán quedar todos a uno de esos lados del cubo.

En general el método propuesto se basa en tres comprobaciones básicas: (1) comprobar la posición de los vértices con respecto a ciertos planos asociados con caras, aristas y vértices del cubo; (2) comprobar si alguno de los lados interseca con el cubo; (3) comprobar si el cubo se halla en el interior del polígono.

## 3.4 Creación del haz inicial.

Hasta ahora se ha descrito la estructura de descomposición que guiará el trazado de los haces. En este apartado se analiza el proceso de obtención del haz

inicial, a partir del cual se crearán el resto de los haces en el proceso de recorrido de la escena.

La creación del haz inicial tiene algunas diferencias respecto al resto de haces. En la determinación de este haz inicial influirán tanto la ventana por la que el observador mira a la escena (tamaño y situación de la pantalla), como la situación y descomposición del cubo que contiene la escena.

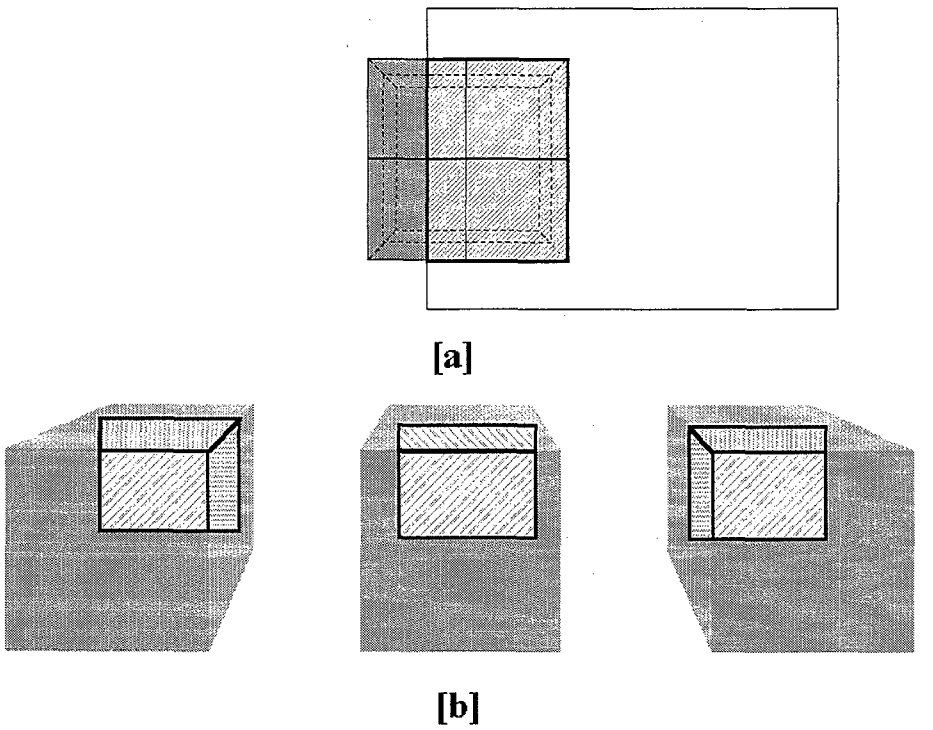



**Figura 3.3 Determinación de los haces iniciales.**

En la figura 3.3 se muestran diferentes situaciones que se pueden producir en el supuesto tal vez más complejo que puede darse, el cual se corresponde con el hecho de que el observador esté fuera del cubo que contiene a la escena. En este caso, al estar el observador fuera del cubo que contiene a la escena, debe realizarse una proyección en perspectiva de la pantalla (que representa la ventana de visión que tiene el observador sobre la escena) sobre las caras visibles del cubo para integrarla dentro de la estructura que describe a la escena. En caso contrario, es decir, si el

observador se encuentra dentro del cubo, dicha proyección ya no es necesaria. En este caso, al considerar el origen del haz dentro de uno de los nodos del árbol el proceso será similar al realizado cuando un haz llega a un nodo vacío (como veremos posteriormente con mayor nivel de detalle, se generan haces de salida en función de las direcciones de salida y de los vecinos en dichas direcciones). La principal diferencia será que ahora el haz que debemos considerar como entrada (haz que llega a un nodo vacío y que puede descomponerse en otros de salida para seguir su camino), se corresponde con el haz que engloba a toda la pantalla y, por tanto, es probable que el tamaño de dicho haz haga que aumenten las direcciones por las que éste abandona el nodo (en algunos casos es posible que el haz salga por todas las direcciones menos por la opuesta a la dirección principal de propagación). Junto a ello, se debe incluir un proceso que permita determinar el nodo terminal en el que se encuentra situado el observador, para lo cual se realiza un recorrido descendente desde la raíz del árbol hasta alcanzar dicho nodo.

Como podemos observar en la figura 3.3, las diferentes posiciones relativas de la pantalla respecto al cubo inicial y la subdivisión del mismo, condicionarán el número y el tamaño del haz o haces iniciales. Como puede apreciarse si la ventana es mayor que el cubo (figura 3.3 [a]) el haz debe ajustarse al tamaño de la cara del cubo por la que entra. En este caso, los rayos asociados a dicha zona no cubierta por ningún haz podemos asegurar que no encontrarán en su camino ningún objeto y, por tanto, los pixel de esta zona pueden etiquetarse sin más con un color de fondo. Por otra parte, si desde la pantalla se ven varias caras del cubo (figura 3.3 [b]) se formaran varios haces iniciales. En este caso, cada haz entrará a la escena por una cara diferente del cubo.

La determinación del primer haz se realiza proyectando la pantalla sobre los planos asociados a las caras visibles del cubo y recortando el polígono resultante con respecto a dichas caras. La gran diferencia de este proceso con el que se analizará posteriormente y que tiene por objetivo determinar los haces de salida, viene dada por los planos que se utilizan para realizar la proyección. En este caso en vez de

utilizar el plano asociado a la cara posterior de la dirección de propagación se utiliza el plano asociado a la cara anterior, es decir la cara por la que entrará el haz.

Los haces iniciales creados con este proceso deben sufrir un refinamiento, pues el cubo inicial (raíz del árbol) que contiene la escena suele estar subdividido en otros de menor tamaño siguiendo una estructura de árbol octal (figura 3.3[a]). Por tanto, debemos ajustar el tamaño de los haces al de los cubos asociados a nodos terminales que se encuentren en la dirección de propagación seleccionada. Como vemos en la figura 3.3[a] el haz inicial que entra al cubo que representa la totalidad de la escena debe dividirse en función de los cubos de la cara de entrada, produciéndose cuatro haces de menor tamaño. Este proceso de ajuste al tamaño de la cara de entrada se realiza mediante el recorte, con respecto a las caras del nodo por las que entrará el haz, del polígono obtenido tras la proyección de dicho haz sobre el plano asociado a la cara de entrada.

Con el proceso anterior se han obtenido una serie de haces asociados a partes de la pantalla para las cuales los pixel correspondientes representen previsiblemente a algún objeto de la escena, siendo necesario, por tanto, realizar un seguimiento de su trayectoria. En el siguiente apartado se explicará cómo se realiza el trazado de estos haces iniciales dentro de la escena.

## 3.5 Generación de la trayectoria de un haz dentro de la estructura de árbol octal.

Este proceso se realiza cuando un haz se encuentra un nodo vacío y tal vez sea el más complicado de todo el algoritmo que aquí proponemos. Como hemos indicado en el gráfico 3.3, se pueden identificar dos grandes tareas: (1) determinar las direcciones de salida; (2) obtener el haz de salida asociado a cada una de estas direcciones. En el cuadro 3.4 se detalla el algoritmo seguido para realizar la generación de la trayectoria de un haz dentro de la escena. Durante este proceso

como puede apreciarse, un haz inicial se va descomponiendo en otros de manera sucesiva conforme avanza en la estructura de subdivisión de la escena. Cada haz de salida generado en una etapa se analiza como haz de entrada en la etapa siguiente.

### 3.5.1 Determinación de las direcciones de salida.

Una vez que un haz llega a un nodo vacío, la primera de las tareas consiste en determinar las direcciones de salida del haz. Es importante resaltar la diferencia entre este proceso y el realizado en el trazado de rayos individuales, ya que en este caso un haz puede salir por varias direcciones a diferencia de lo que sucede con un rayo que sale por una única dirección.

**Algoritmo:**

**nodo vacío: (Generar la trayectoria del haz en la escena)**

```
Determinar las direcciones de salida dado un haz y un nodo.
Para cada dirección
    Obtener el haz de salida estándar en función de la cara del nodo por la que sale.
    Buscar el nodo o nodos terminales vecinos en dicha dirección.
    Si en dicha dirección tiene vecinos.
        Si tiene un solo vecino cuyo tamaño es igual o mayor que el del nodo origen.
            El haz estándar no se modifica.
        En otro caso, (tiene varios vecinos de menor tamaño)
            Debemos dividir el haz de salida estándar en función del tamaño de sus
            vecinos.
        Procesar el o los nuevos haces de modo recursivo.
```

**Gráfico 3.4 Detalle del proceso de generación de la trayectoria de un haz.**

La determinación de las diferentes direcciones de salida se apoya en la

proyección del haz sobre uno de los planos del cubo, en este caso el plano asociado a la dirección de propagación principal. Para determinar dicha dirección se analiza el valor de las coordenadas de los vectores (vectores normalizados) asociados a las aristas que delimitan el haz. Aquella que tenga un mayor valor absoluto será la que indique la dirección de propagación buscada. Si consideramos que el observador mira al cubo que contiene la escena en la dirección de la coordenada Z el plano sobre el que debemos proyectar el haz es el asociado a la cara posterior. A continuación se detalla el proceso seguido teniendo en cuenta la consideración anterior. En cualquier

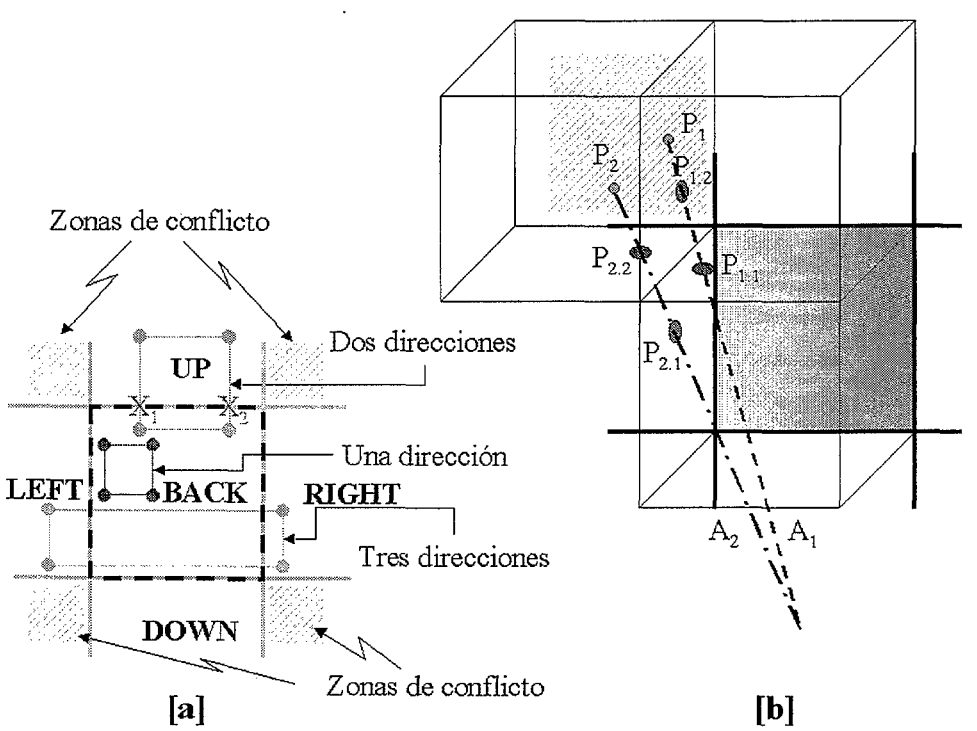



**Figura 3.4 Determinación de las direcciones de salida de un haz.**

caso, la generalización para el resto de direcciones es trivial.

Una vez proyectado el haz, el problema se reduce a determinar la situación de los vértices del polígono resultante con respecto al cuadrado asociado a la cara

posterior del cubo. Como puede apreciarse en la figura 3.4[a] se definen cinco zonas principales (*BACK, UP, DOWN, LEFT, RIGHT*) que permiten asegurar una dirección de salida del haz. A su vez, junto a estas existen zonas conflictivas asociadas a las áreas de intersección de dos zonas principales (en la figura 3.4[a] aparecen rayadas).

Si un punto se encuentra dentro de estas áreas de conflicto no se sabe a priori la dirección o direcciones de salida. En este caso, se necesita proyectar el haz sobre otro plano correspondiente a una de las caras asociadas a las direcciones de conflicto. Por ejemplo, si la proyección de una arista del haz se sitúa en la esquina superior izquierda (puntos $P_1$ y $P_2$ de la figura 3.4 [b]), para llegar hasta dicha zona es necesario que el haz atraviese bien la cara izquierda o bien la cara superior, pero de momento desconocemos cuál de ellas. Para eliminar esta incertidumbre, se debe proyectar el haz sobre el plano asociado a una de las caras (*LEFT* o *UP*) y en función de la situación del punto correspondiente a la proyección de la arista problemática se puede determinar la dirección de salida real. En el ejemplo de la figura 3.4[b] si utilizamos el plano asociado a la cara superior obtenemos como puntos de corte los puntos $P_{2.2}$ y $P_{1.1}$. Como podemos apreciar, en el caso de la arista $A_1$ el punto de corte ($P_{1.1}$) se encuentra situado dentro de la cara superior del cubo y por tanto dicha arista saldrá primero por la cara superior. En cambio el punto de corte de la arista $A_2$ ($P_{2.2}$) se encuentra situado fuera de la cara superior y por tanto la arista atravesará primero la cara izquierda ($P_{2.1}$). Como resultado podemos decir que el haz que contenga a las aristas $A_1$ y $A_2$ saldrá por las direcciones *UP* y *LEFT*.

## 3.5.2 Obtención de los haces de salida.

La obtención de los haces de salida puede ser considerada como el proceso más complejo de todos los necesarios para realizar el trazado de los haces dentro de la escena. Este proceso implica la realización de tres tareas básicas: (1) determinar el *haz de salida estándar* asociado a cada dirección de salida; (2) buscar dentro del

árbol octal nodos vecinos en una dirección dada; y por último (3) calcular los haces de salida finales.

El proceso final de obtención de los vectores que delimitan los nuevos haces aparece descrito en la figura 3.5. En ella se indica, para un haz de entrada cualquiera, cómo se realiza el proceso de determinación de los haces de salida considerando que el haz se propaga en la dirección Z y, por tanto, la dirección principal de salida es la etiquetada con BACK. Si se considerase otra dirección principal de propagación el proceso sería semejante, produciéndose tan solo un cambio en la situación de las direcciones de salida.

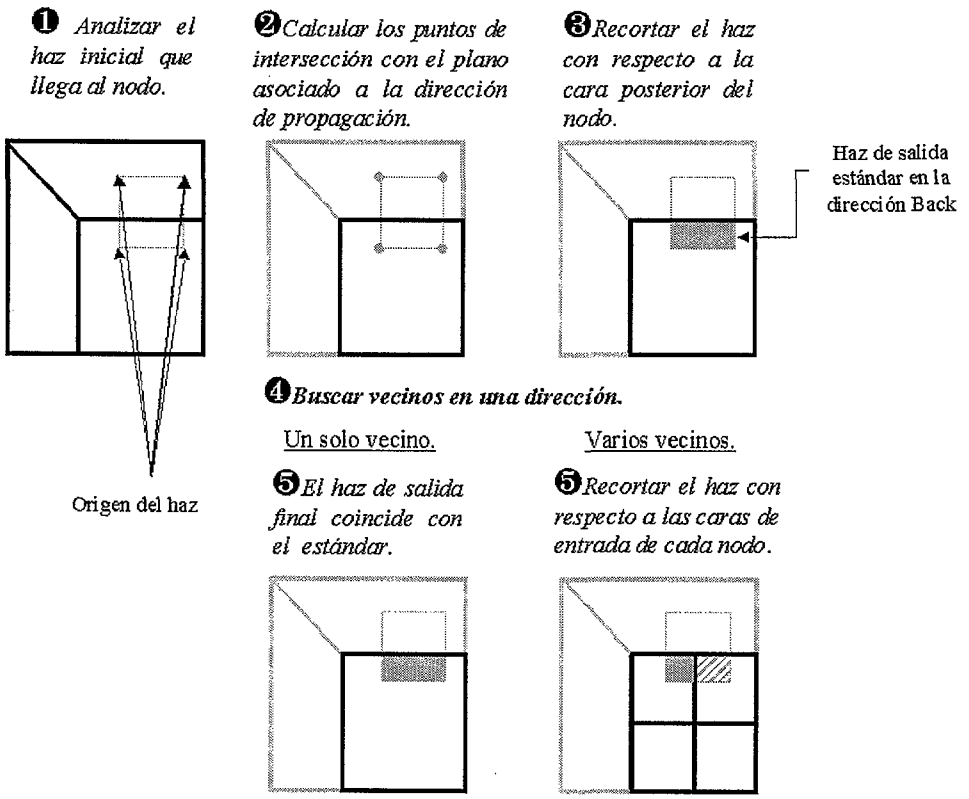



**Figura 3.5 Proceso general de descomposición de los haces conforme avanzan en el árbol octal.**

Como podemos apreciar en la figura, primeramente se calculan los puntos de corte del haz inicial con el plano asociado a la cara posterior del nodo (asociada a la dirección principal de propagación para este ejemplo), obteniendo el denominado *haz de salida estándar*. Tras ello se buscan vecinos en la dirección de propagación. Si se encuentra un solo vecino habrá un único haz de salida igual al estándar. En cambio si se encuentran varios vecinos el *haz de salida estándar* debe dividirse para ajustar los haces al tamaño de la cara de entrada de cada uno de dichos vecinos (en la figura 3.5 puede apreciarse cómo el *haz de salida estándar* se divide en dos).

### 3.5.2.1 Determinación de los haces de salida estándar.

Tras conocer las direcciones de salida se deben calcular para cada una de ellas el denominado *haz de salida estándar*. Este haz es el que se obtiene como resultado de recortar con respecto a la cara de salida la proyección del haz sobre el plano asociado a dicha cara.

De manera general, este proceso debería realizarse para cada una de las direcciones de salida. Pero por otra parte podemos ver que los diferentes haces a generar comparten algunas aristas con los haces vecinos. Por ello, se puede utilizar datos de los haces previamente calculados para establecer los nuevos haces.

Si analizamos en la figura 3.4 [a] el haz de la parte superior que sale por las direcciones *BACK* y *UP,* podemos ver que los puntos de corte con la arista superior de la cara posterior (BACK) $X_1$ y $X_2$ también serán los puntos de corte con la arista de la cara superior (UP) ya que dicha arista es compartida. Esta característica permitirá aprovechar el conocimiento de los puntos que componen un haz para determinar, sin necesidad de cálculo alguno, los puntos que delimitan a otro nuevo haz que comparte con el anterior una arista. De este modo no siempre es necesario proyectar el haz sobre el plano asociado a cada nueva cara de salida y realizar el recorte posterior.

Por tanto el proceso es siempre el mismo:

- Recortar el haz respecto a la cara del cubo asociada a la dirección de propagación principal. Con ello se obtiene el *haz de salida estándar* para esa dirección.

- Generar los nuevos haces de salida estándar en el resto de direcciones aprovechando los puntos calculados anteriormente para las direcciones previas.

En el haz que sale por *UP* dentro de la figura 3.4 [a], vemos que para realizar el proceso que permita obtener el haz de salida en dicha dirección, tan sólo es necesario proyectar las aristas superiores del haz inicial y unir dichos puntos con los obtenidos previamente (puntos $X_1$ y $X_2$) en el proceso de recorte con respecto a la cara posterior. Este modo de operar, aunque complicará su implementación, nos permitirá optimizar el algoritmo de trazado.

Por último, dentro del proceso anteriormente descrito siempre debe realizarse el recorte de la proyección del haz con respecto a la cara de salida. Este problema se reduce a calcular el recorte de un polígono con respecto a un cuadrado. En este caso, el recorte se realiza utilizando el algoritmo clásico de recorte de un polígono respecto a una región rectangular propuesto por Sutherland y Hodgman [SUTH74a] (ver figura 3.6). Este algoritmo permite obtener los nuevos puntos que delimitarán el haz de salida.

Este algoritmo [SUTH74a], es válido para recortar un polígono (cóncavo o convexo) con respecto a otro polígono convexo. Para ello, utiliza una estrategia del tipo *"divide y vencerás"*, es decir, reduce un problema complejo a una serie de problemas más simples de tipo semejante. En este caso el problema se reduce a calcular el recorte de un polígono con un eje. Realizando este proceso para los cuatro ejes que definen la ventana obtenemos finalmente el polígono recortado.

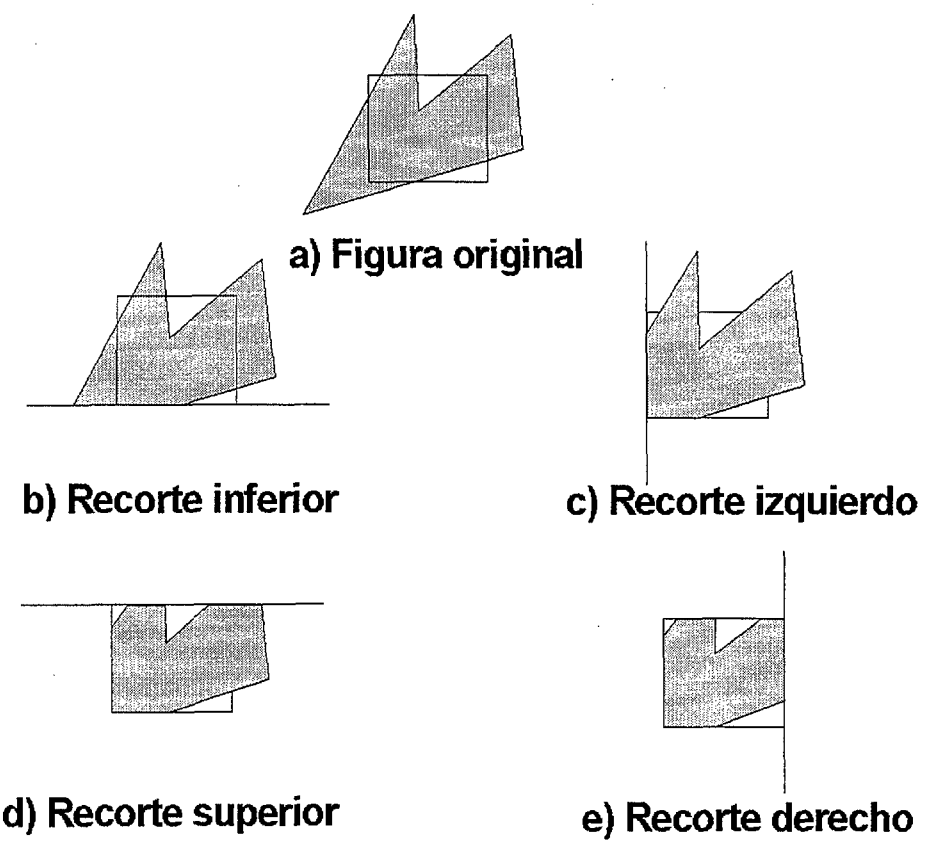



**Figura 3.6 Algoritmo de recorte de polígonos.**

Este algoritmo utiliza como entrada un conjunto ordenado de vértices que definen el polígono y realiza un recorte con respecto a un eje, obteniendo como salida otra serie de vértices que definen el nuevo polígono recortado. Este conjunto de vértices de salida se pasan de nuevo como entrada al algoritmo y se realiza el recorte con el siguiente eje, repitiendo el proceso hasta que se realice con todos los ejes de la ventana de recorte. Con ello se obtienen los nuevos vértices que definirán el polígono recortado y que en nuestro caso se convertirán en las aristas del nuevo haz.

Con el proceso anterior hemos definido unos haces de salida que tan sólo tiene en cuenta la cara del nodo por la que salen. A estos haces de salida los hemos denominado ***haces de salida estándar***. Pero, como podemos apreciar en la figura

3.7, la determinación de los haces de salida finales no sólo depende de la cara por la que abandona el haz el nodo en cuestión, sino también de los nodos a los que llega.

Si analizamos detenidamente la figura 3.7 vemos que si en la dirección de propagación el haz encuentra un nodo vecino de igual o mayor tamaño al actual (como sucede con los haces B, C o E), el haz de salida tan sólo tiene en cuenta el haz inicial y los límites de salida del nodo en dicha dirección (este haz lo hemos denominado *haz de salida estándar*). Por otra parte si en la dirección dada encontramos varios vecinos, el *haz de salida estándar* debe dividirse tantas veces como vecinos hayamos encontrado. De ese modo, como podemos apreciar en la figura, el *haz de salida estándar D* se divide en dos nuevos haces *D'* y *D''*, cada uno de los cuales llegará a un nodo vecino diferente. Por tanto, para obtener los haces de salida definitivos junto a la cara de salida se deben de tener en cuenta las caras de entrada de los nodos vecinos.

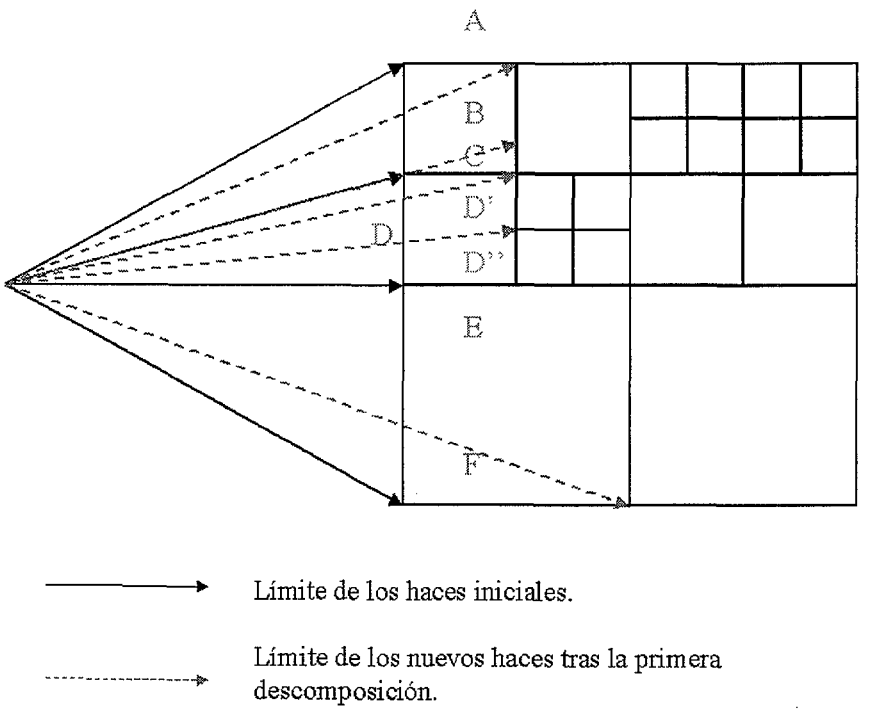



**Figura 3.7 Descomposición del haz de salida en una escena 2D.**

### 3.5.2.2 Búsqueda de vecinos en una determinada dirección.

El proceso de búsqueda de los nodos vecinos en una determinada dirección se apoya en los trabajos de H. Samet [SAME89] que propone un algoritmo que se encuadra dentro de los denominados *"métodos padre"*. El nombre de estos métodos se debe a que para realizar el proceso necesitan que cada nodo almacene un nuevo enlace que apunte al nodo anterior (denominado *padre*). El proceso implica un ascenso en el árbol hasta encontrar el primer antecesor común al nodo actual y al vecino que estamos buscando. Tras determinar dicho antecesor común realiza un descenso en el árbol buscando por las direcciones opuestas a las seguidas en el proceso de ascenso. Con este proceso se accede al vecino de igual o mayor tamaño en dicha dirección. Por último si el nodo al que llegamos es un nodo no terminal debemos seguir el proceso de descenso en el árbol hasta alcanzar los nodos terminales en la dirección deseada. A continuación detallamos el algoritmo propuesto para el caso de una escena en dos dimensiones, el cual será fácilmente transladable a un entorno tridimensional simplemente añadiendo la característica de la profundidad.

En la figura 3.8 puede apreciarse el proceso de búsqueda del vecino situado a la derecha (dirección *RIGHT*) del nodo rayado y que comparte el punto *P*. En esta figura se explica el proceso de búsqueda del vecino para el caso de dos dimensiones. Como puede apreciarse cada nodo se etiqueta en función de su situación con dos caracteres (*LU*-> superior izquierda; *RU*-> superior derecha; *LD*-> inferior izquierda; *RD*-> inferior derecha), el primero de los cuales referencia a la dirección horizontal (izquierda "*L*" y derecha "*R*") y el segundo referencia a la dirección vertical (arriba "*U*" y abajo "*D*").

El proceso de ascenso viene guiado por la dirección en la que se desea buscar el vecino (en el caso descrito en la figura la dirección derecha o *RIGHT*). El proceso de ascenso termina cuando se encuentra un nodo etiquetado con una letra distinta a la dirección de búsqueda. En el ejemplo de la figura el proceso de ascenso termina

cuando encontramos un nodo etiquetado con *"L"* ya que la dirección de búsqueda era *"R"* (derecha), determinando que el primer antecesor común es el padre del nodo etiquetado con *"L"* (en el ejemplo el nodo *"LD"*).

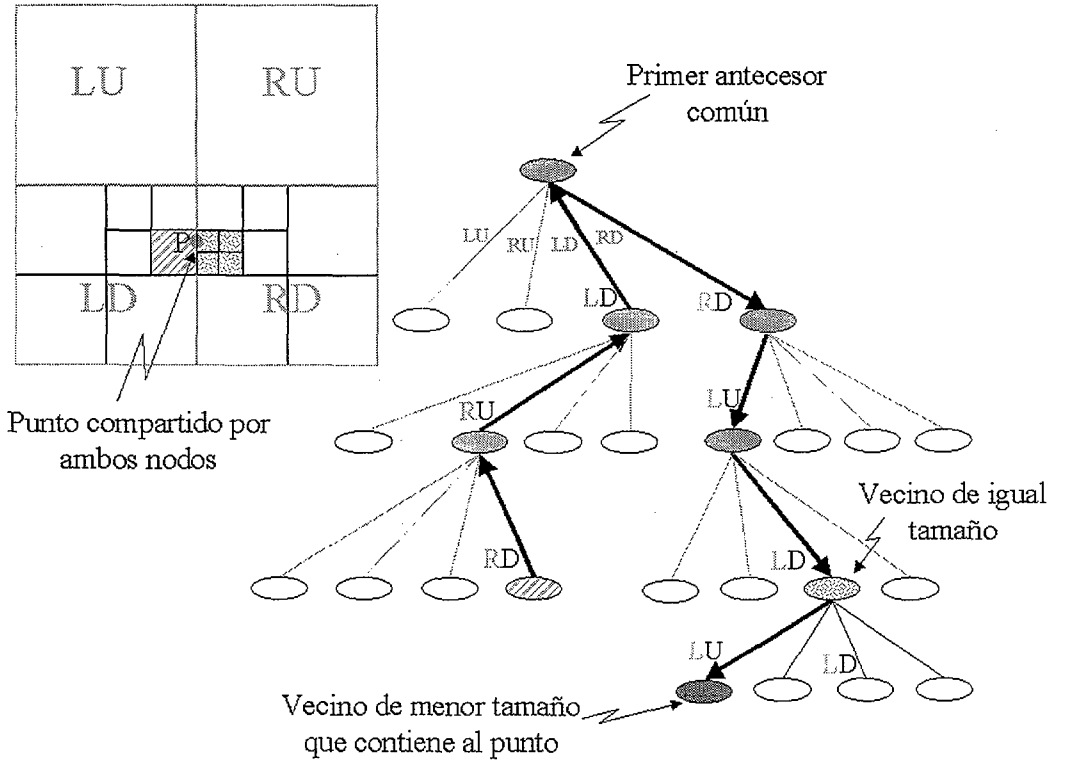



**Figura 3.8 Proceso de búsqueda de un nodo vecino en una dirección.**

Una vez encontrado el primer antecesor que tanto el nodo actual como su vecino en la dirección indicada tienen en común, se debe descender en el árbol para encontrar el vecino buscado. Este proceso de descenso se realiza utilizando el camino inverso al seguido durante el ascenso. En el ejemplo el camino seguido durante el ascenso fue: "*RD-RU-LD*" y por tanto el camino de descenso será "*RD-LU-LD*". Como podemos apreciar, en este caso, se intercambia el valor del primer carácter de la etiqueta ya que la dirección de búsqueda hace referencia a una dirección horizontal.

Con todo este proceso se localiza un nodo vecino que tendrá un tamaño mayor o igual al del nodo desde donde se inició la búsqueda. Por tanto, si el nodo al que se llega tiene descendientes, se deberá seguir el proceso de descenso pero esta vez se debe acceder a los nodos en el sentido contrario a la dirección de búsqueda. En nuestro caso sería buscar los nodos descendientes siempre en la dirección "*L*" ya que serán los que compartan la arista con el nodo inicial. De este modo, del nodo punteado tan sólo necesitamos analizar los hijos etiquetados con "*LD*" y "*LU*", comprobándose que el nodo que contiene realmente el punto buscado es el etiquetado con "*LU*".

### 3.5.2.3 Cálculo de los haces de salida finales.

El último de los procesos necesarios para obtener los haces de salida generados por el trazado, está asociado con los nodos a los que llegará cada *haz de salida estándar*. Por tanto, este último proceso vendrá condicionado por los vecinos encontrados en la tarea anterior.

Cada *haz de salida estándar* deberá dividirse en tantos haces como vecinos se hayan encontrado en la dirección de salida. Como puede verse en la figura 3.7, el *haz de salida estándar D* encuentra en la dirección de propagación dos nodos vecinos, lo que provoca que deba dividirse y se creen dos nuevos haces *D'* y *D''*. Este proceso de generación de los haces finales de entrada implica el recorte del *haz de salida estándar* con respecto al cuadrado asociado a la cara de entrada del nodo vecino al que llegará dicho haz tras abandonar el nodo en el que se encuentra.

Como indicamos anteriormente, el proceso de recorte que nos permite obtener los límites del nuevo haz se realiza utilizando el algoritmo de recorte de polígonos respecto a una región rectangular propuesto por Sutherland y Hodgman [SUTH74a]. De este modo, el tamaño del haz que entra a un nodo se ajusta el tamaño de la cara por la que entra, pudiendo considerar tras esto que hemos encontrado los haces de salida reales.

## 3.6 Análisis del comportamiento de los pixel asociados a un haz.

El comportamiento del algoritmo propuesto, cuando el haz llega a un nodo ocupado o cuando éste abandona el cubo inicial que contiene a la escena es totalmente distinto al analizado anteriormente. En ambos casos es necesario analizar el comportamiento de los diferentes pixel asociados al haz. En primer lugar hay que determinar el número de pixel a los que afecta el haz y tras ello determinar el número de rayos que se desean trazar para determinar el color de un pixel concreto. Con este proceso se permite que se asigne un mayor número de rayos a aquellos pixel que están asociados a zonas con una mayor concentración de objetos.

El mayor ahorro se obtiene cuando un haz abandona la escena, sin visitar previamente ningún nodo ocupado. En este caso todos los pixel asociados a dicho haz tendrán el mismo color (el correspondiente al color del fondo) y, por tanto, no será necesario generar ningún rayo, obteniéndose una gran ganancia de tiempo.

Por otra parte, cuando el haz llega a un nodo ocupado, la determinación del color de los pixel no es tan fácil. En esta situación es necesario generar al menos un rayo por pixel, pudiendo producirse el denominado problema o efecto de aliasing. Este problema se produce debido al proceso de muestreo asociado al modo en que se genera la imagen. En cualquier caso, con el trazado de haces este efecto de aliasing puede reducirse aplicando un nuevo criterio, ya que en el instante de la generación de los rayos se dispone de información que en las técncas tradicionales de antialiasing, técnicas basadas en el *espacio de la imagen*, no está accesible.

La alternativa de tratamiento del aliasing que aquí se propone se apoya, en cambio, en técnicas basadas en el *espacio de los objetos* [GENE98], al utilizar, en la

determinación de los rayos que se generarán por cada pixel, información sobre los objetos de la escena. En este caso, en el instante de generación de los rayos se conocen dos datos importantes: el número de objetos contenidos en el nodo al que llega el haz y la profundidad a la que se encuentran los objetos. Esta información puede ser útil a la hora de decidir el número de rayos a generar, creando y lanzando un mayor número de rayos en las zonas que contienen más objetos que en aquellas asociadas a espacios vacíos. Por otra parte la profundidad a la que se encuentran los objetos puede ser útil si deseamos que no se produzca el efecto de aparición y desaparición de objetos entre distintos fotogramas. Con esta información sobre la profundidad se pueden lanzar más rayos si el nodo que contiene a los objetos está lejos que si éste se encuentra muy próximo.

**Algoritmo:**

**nodo ocupado o bien el haz abandona la escena: (Analizar el comportamiento de los pixel asociados a un haz)**

```
Determinar los pixel asociados al haz.
Si el haz abandona la escena.
    Generar un rayo por pixel.
    Etiquetar cada rayo con el color del fondo.
En otro caso,
    Generar tantos rayos por pixel como sea necesario para reducir el problema de aliasing.
    Para cada uno de ellos calcular su intersección con los objetos contenidos en el nodo.
        Si el rayo choca con uno de los objetos asociados al nodo y el punto de intersección
          se encuentra dentro del nodo.
            Generar los diferentes rayos secundarios (sombra, reflejados o transmitidos) y
              trazarlos de modo individual a través del árbol octal.
        En otro caso,
            Trazar de modo individual cada rayo primario.
```

**Gráfico 3.5 Detalle del proceso de análisis del comportamiento de los rayos contenidos en un haz.**

En el gráfico 3.5 se describe el algoritmo utilizado para analizar el comportamiento de los pixel asociados a un haz. Como podemos apreciar el proceso de generación de rayos por pixel, que proponemos inicialmente, depende de sí el haz llega a una zona poblada o libre de objeto. En el caso de que un haz abandone la escena, la parte de la imagen asociada a dicho haz tendrá un color homogéneo y, por tanto, para determinar el color de un pixel no se necesitaría lanzar ningún rayo. En cambio, si la zona está poblada de objetos, puede aparecer el problema de aliasing asociado a zonas en las que el color cambie bruscamente, por ejemplo en la representación de las siluetas de los objetos o en las intersecciones entre las proyecciones de dos objetos.

Dentro de este proceso existen varias tareas que pueden considerarse comunes a cualquier trazador de rayos basado en árboles octales [SAME89]: (1) calcular la intersección entre un rayo y un objeto; (2) generar y trazar de modo individual los diferentes rayos secundarios. Junto a ellas, se encuentran tareas específicas del trazado de haces: (a) determinar pixel asociados a un haz; (b) analizar el tratamiento más conveniente para los rayos que no chocan con ninguno de los objetos asociados al nodo. A continuación, se analizarán de manera más detallada las tareas propias del trazado de haces. Al mismo tiempo se invita al lector interesado en conocer con detalle cómo se realiza el trazado de rayos individuales en escenas estructuradas mediante árboles octales a revisar los trabajos de H. Samet [SAME89], [SAME90b] en los que se basa esta implementación.

### 3.6.1 Determinación de los pixel asociados a un haz.

La determinación de los pixel asociados a un haz se apoya en las ideas de rellenado de polígonos mediante líneas de rastreo [FOLE90]. Para ello se realizan dos tareas básicas: (1) proyectar el haz sobre el plano de la pantalla; (2) determinar los pixel contenidos dentro de un polígono.

La proyección implica el cálculo de la intersección de los vectores que delimitan el haz con el plano de la pantalla. Esta proyección genera un polígono dentro del cual se situarán una serie de pixel. En este proceso debe tenerse en cuenta que los diferentes haces compartirán aristas o vértices con otros haces vecinos. Por tanto, si un pixel se corresponde con una arista o vértice del polígono proyectado dicho pixel estará asociado a varios haces. Esta circunstancia también se produce cuando se plantea el proceso de rellenado de un polígono en 2D. El proceso general de rellenado de polígonos puede resumirse del siguiente modo:

1.- Encontrar las intersecciones de la línea de barrido con las aristas del polígono.

2.- Ordenar los puntos de intersección en orden creciente de la coordenada *X*.

3.- Seleccionar todos los puntos entre pares de intersecciones utilizando para ello una variable de estado que indica si el punto es interior o no. Inicialmente el estado se define como *"no interior"* y cada intersección invierte el valor de la variable de estado. Si el estado es *"interior"* el punto se rellena y si es *"no interior"* el punto se deja como está.

En nuestra propuesta, se parte de una ventaja añadida que simplifica el proceso descrito anteriormente y es que los polígonos son siempre convexos y por tanto una línea de barrido sólo cortará a dos aristas del polígono. Por otra parte, en el proceso anterior se deben establecer ciertas reglas de asignación que aseguren que un pixel sólo se asociará a un determinado polígono. En este caso, el algoritmo que aquí se propone se apoya en las mismas ideas que permiten seleccionar el polígono correcto cuando el punto de corte de la línea de rastreo y una arista coincide con el valor exacto de un pixel [GONZ98c].

En cualquier caso, podría pensarse que la aplicación del criterio de asignación establecido que asocia un pixel a un haz, podría hacer variar el resultado final. Si analizamos el problema con detenimiento podemos ver que el supuesto peor sería cuando la arista del haz que se corresponde con un pixel coincidiera con una arista o vértices del nodo (ver figura 3.9). Si esto es así, se podría suponer que en función de que se asociase a un haz o a otro la trayectoria (nodos por los que pasarían) de dichos rayos asociados a una arista o vértice variaría. Esto puede ser cierto pero tal vez el problema no es si la trayectoria variará sino si el resultado final será el mismo, es decir, si los rayos chocarán con los mismos objetos.

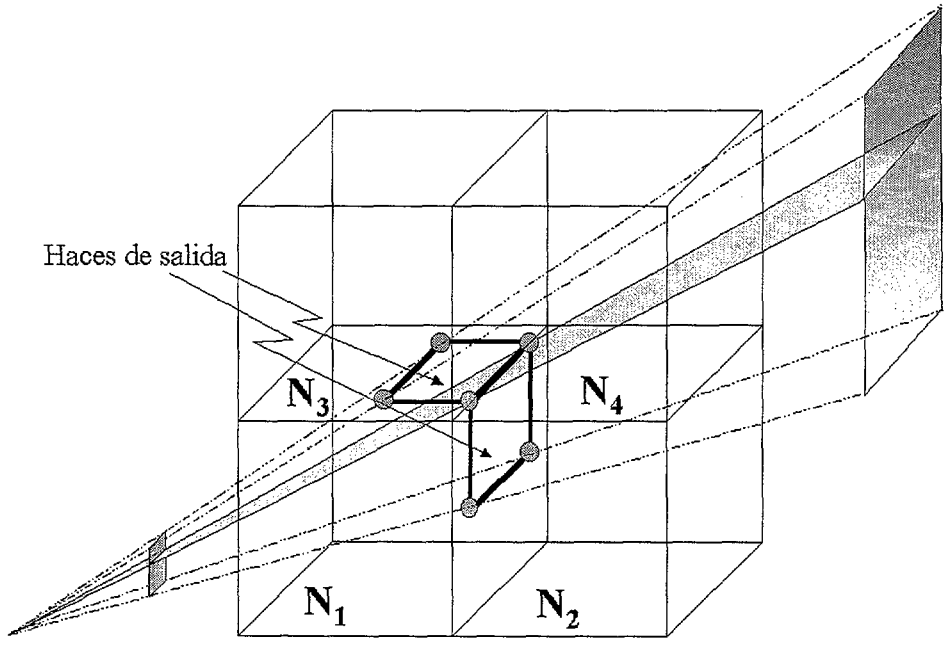



**Figura 3.9 Análisis de los rayos que atraviesan las aristas o vérices de un nodo.**

Como podemos ver en la figura 3.9, los rayos asociados a una arista deberían pasar directamente al nodo vecino en la dirección de propagación establecida que comparte dicha arista (en la figura el nodo $N_4$). El problema surgiría si al pasar el

haz previamente por otro nodo ($N_2$ o $N_3$) alguno de estos rayos encontrara un objeto en su camino. Si esto ocurriera el objeto que chocara con el rayo debería estar tan próximo a la arista que generalmente intersecaría con ella. En este último caso, si el objeto tiene algún punto de intersección con la arista, dicho objeto no sólo estaría asociado a este nodo, sino a todos los nodos vecinos que comparten dicha arista y, por tanto, dicha intersección ya habría sido analizada en el nodo inicial ($N_1$).

Junto a este posible problema de falta de precisión, anteriormente despejado, puede pensarse que existe algún problema de ineficiencia, al hacer que un rayo atraviese un nodo innecesario. Si analizamos esta situación con detenimiento, podemos ver que este supuesto se puede producir mientras el rayo está contenido en un haz o mientras el rayo se traza individualmente. En el primer caso, el rayo se propaga dentro del haz junto al resto sin necesidad de consumir tiempo. Por otra parte, en el segundo supuesto, el problema desaparece al utilizar el algoritmo propuesto por H. Samet [SAME90b] para rayos individuales, ya que éste tiene en cuenta el hecho de que un rayo atraviese una arista o un vértice.

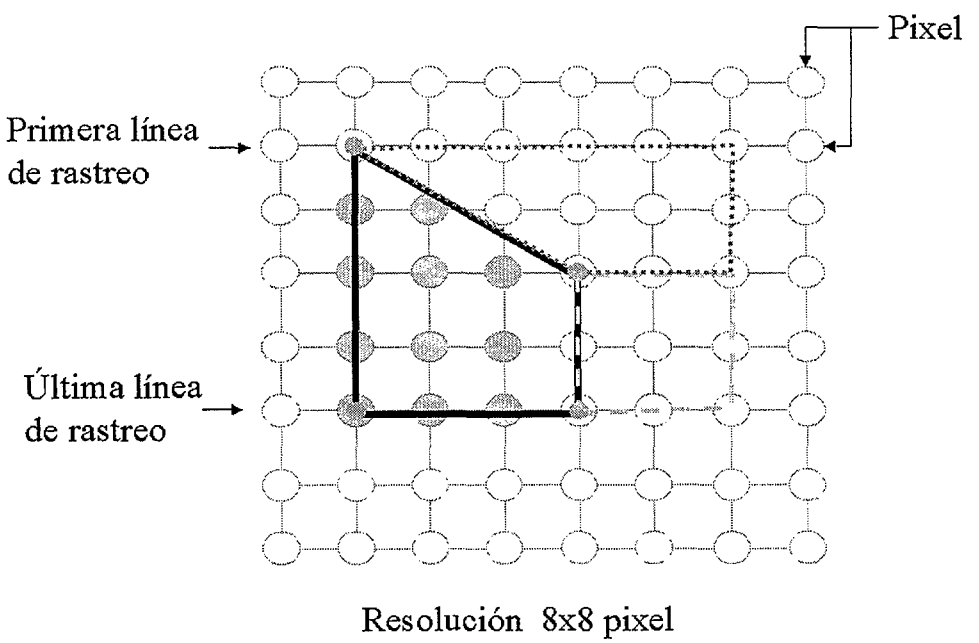



**Figura 3.10 Esquema del proceso de rellenado de polígonos mediante líneas de rastreo.**

Por tanto, cuando un pixel se corresponda con un vértice o arista el tratamiento a realizar puede ser el mismo que el que se propone dentro del algoritmo típico de rellenado de polígonos mediante líneas de rastreo. En éste, se establecen ciertas condiciones que determinan la pertenencia de una arista o vértice compartido entre varios polígonos a uno en concreto (ver figura 3.10). En nuestro caso, para asegurar que todos los pixel se tratan al menos una vez, si el haz que se está analizando es uno que abandona la escena, dicho haz no tendrá vecinos en la dirección principal de salida y, por tanto, deberá tratar los pixel que le corresponderían al vecino, si es que existiese dicho vecino.

### 3.6.2 Generación y análisis de los rayos asociados a un pixel.

Tras la determinación de los pixels asociados a un haz, se deben generar los rayos correspondientes a cada uno de dichos pixels, para de este modo poder determinar su color final. En este caso, nuestro algoritmo también aporta ciertas mejoras respecto a otras alternativas al tratar el problema del aliasing. Como puede apreciarse el análisis del color de un pixel se realiza bien cuando un haz abandona la escena o cuando llega a un nodo ocupado por objetos.

Por ello, si el haz ha llegado a un nodo ocupado por objetos o el haz abandona la escena puede pensarse que es necesario generar al menos un rayo por cada pixel para poder determinar su color. El número de rayos a generar aumentaría si se desean utilizar técnicas de sobremuestreo (uniforme, adaptativo, estadístico o estocástico) para reducir el problema de aliasing. El inconveniente que tiene los métodos tradicionales que utiliza las técnicas de sobremuestreo para reducir el aliasing es que la determinación del número mínimo de rayos a trazar por cada pixel se realiza antes de comenzar el trazado y solo se aprovecha la información obtenida durante el trazado para determinar el número máximo de rayos que deben trazarse

por cada pixel para conseguir una mayor precisión en la determinación del color final del pixel dado. Por ello, Genetti et al. [GENE98] definen a dichas alternativas como *basadas en el espacio de la imagen.*

En nuestro caso se retrasa la tarea de definir tanto el número mínimo como máximo de rayos necesarios para determinar con cierta precisión el color de un pixel. Este retraso es posible debido a que los rayos que pudiesen ser generados viajan juntos dentro de los haces y mientras que atraviesan la escena no debemos preocupar por su número. Con este retraso en el instante de generación de los rayos se obtiene una gran ventaja, pues en dicho instante se conoce cierta información de la que carecen las técnicas de antialiasing *basadas en el espacio de la imagen.* Nuestro algoritmo permite disponer de información sobre cuál es el comportamiento previsible de los posibles rayos a generar y de ciertas características de la parte de la escena con la que probablemente pudiesen chocar. La utilización de esta información adicional obtenida durante el proceso de trazado hace que nuestro algoritmo se asome a la utilización de técnicas de antialiasing desde una nueva perspectiva a los que Genetti et al. [GENE98] denominan *basada en el espacio de los objetos.*

Así, en un caso, el hecho de que la generación de rayos se realice cuando un haz se encuentra en un nodo ocupado ofrece la posibilidad de conocer información referente al número y la situación aproximada de los objetos asociados a dicho nodo. Esta información puede utilizarse para determinar, en función de los criterios previamente establecidos sobre calidad de la imagen final, el número de rayos a trazar por cada uno de los pixels asociados al haz que ha llegado a dicho nodo. A su vez, el conocimiento aproximado de la profundidad a la que se encuentran los objetos podría permitir simular efectos de enfocado y desenfocado de objetos situados a una determinada distancia desde el observador. Para ello, se podrían generar un mayor número de rayos cuando se alcance la profundidad de enfoque y un número mucho menor cuando nos encontremos a profundidades diferentes.

Pero sin duda la mayor ventaja se obtiene en el supuesto de que encontremos

un haz que abandona la escena. En este caso, también tenemos que determinar el número de rayos a generar para cada pixel de la pantalla. Pero en esta situación el color de los pixel que se corresponden con un haz que abandona la escena será uniforme e igual al color de fondo establecido. Con ello, podemos asegurar el color de los pixel asociados a un haz sin necesidad de trazar ningún rayo. En este caso, la información obtenida no sólo permite establecer un criterio de sobremuestreo, sino que posibilita asegurar lo innecesario que es, en este caso, utilizar cualquier sobremuestreo. De este modo, nuestro algoritmo evita la generación de rayos asociados a zonas de la imagen que no contienen objetos, algo que la mayoría de las propuestas de antialiasing (basadas en el espacio de la imagen) no pueden garantizar ya que desconocen por dónde atravesará la escena un rayo cualquiera. Nuestro algoritmo, en cambio, lo hace posible ya que conoce información sobre las características finales de la escena y de los objetos que la componen, antes de generar los rayos asociados a un determinado pixel.

En el supuesto de que el haz haya llegado a un nodo ocupado, una vez que se conocen las direcciones de los rayos contenidos en el haz, se realiza el cálculo de la intersección de cada rayo con los objetos asociados a dicho nodo. Este cálculo es similar al realizado por cualquier trazador de rayos, con la principal diferencia de que debe comprobarse que el punto de intersección hallado esté contenido en el nodo que se está analizando. Este diferencia se debe al proceso de descomposición de la escena que permite que un objeto esté contenido en varios nodos y, por tanto, no podemos asegurar que el punto de intersección encontrado pertenezca al nodo en el que actualmente se encuentra el rayo.

Tras el proceso de cálculo de intersecciones entre rayos y objetos nos encontramos con otro problema: ¿qué se hace con los rayos que no chocan con ninguno de los objetos contenidos en el nodo?.

En este caso existen dos grandes alternativas: seguir trazando los rayos dentro de un haz o trazar los distintos rayos de modo individual. La primera

alternativa requiere almacenar los rayos ya procesados, es decir, aquellos rayos que chocaron previamente con algún objeto y por tanto han finalizado su trazado. Por ello, un haz junto con los vectores que definen sus aristas debería mantener una lista con los rayos ya finalizados. Por otra parte, la rentabilidad de trazar los rayos dentro de un haz está fuertemente ligada al número de rayos que viajan dentro de él y, en este caso, el real número de rayos asociados a dicho haz sería inferior ya que algunos ya han finalizado su trazado. Con lo cual sólo sería rentable esta opción si pensáramos que el número de rayos no finalizados contenidos en dichos haces fuera considerable, lo cual no es así en nuestro caso.

Junto a lo anterior debemos tener en cuenta que el criterio de subdivisión establecido, no tenía en consideración el número de objetos contenidos en un nodo, sino tan sólo el máximo nivel de descomposición establecido. De este modo, se consigue que los nodos terminales que contienen objetos se correspondan siempre a nodos de dicho nivel de descomposición y, por tanto, es de esperar que su tamaño sea reducido. Como podemos ver en la primera descomposición que se muestra en la figura 3.2, o más adelante en las figuras 5.4 y 5.5, el criterio de descomposición es tal, que los nodos ocupados son de pequeño tamaño y se ajustan bastante a la silueta de los objetos de la escena, dejando en general poco espacio libre dentro de cada uno de dichos nodos ocupados. Por otra parte, si el tamaño de los haces viene determinado por los nodos que un haz atraviesa es de esperar que su tamaño al alcanzar un nodo ocupado sea reducido e incluso inferior al tamaño de los nodos terminales a los que llega. De este modo, se puede suponer que el número de rayos contenidos en un haz que llega a un nodo ocupado y no chocan con ningún objeto asociado a dicho nodo sea tan pequeño que no sea rentable continuar con su trazado dentro del haz.

Por todo lo anteriormente expuesto, el criterio finalmente establecido para tratar los rayos contenidos en un haz que llega a un nodo ocupado y que no chocan con ninguno de los objetos asociados a dicho nodo es trazarlos de modo individual desde ese punto. Por tanto, a partir de este momento dichos rayos se trazarán de

modo individual, utilizando para ello el algoritmo propuesto por H. Samet [SAME89] para trazado de rayos sobre árboles octales.

# Capítulo 4. Definición y desarrollo del trazado de haces.

## 4.1 Introducción.

Como todo proyecto software dentro de esta tesis se ha utilizado una metodología de desarrollo que ha cubierto la totalidad de las fases del ciclo de vida del sistema. En este caso la metodología utilizada ha sido la definida por P. Coad y E. Yourdon [COAD91a],[COAD91b], [COAD97], denominada *Object Oriented Analysis and Design*.

Existen dos razones básicas que nos inducen a utilizar una metodología orientada a objetos. La primera de ellas es la facilidad con que el *dominio del problema* analizado se ajusta a los conceptos de clases-objetos, generalización-especialización, comunicación mediante mensajes, etc. Estas características han hecho que en algunos casos la generación de imágenes tridimensionales se utilice como ejemplo para explicar algunas metodologías orientadas a objetos [RUMB96]. La otra razón fundamental se deriva de las amplias posibilidades que la orientación

a objetos ofrece en cuanto a reutilización y extensibilidad del software apoyado sobre todo en la explotación de la herencia [GONZ94b].

La reutilización no se ha limitado a las fases de diseño o implementación sino que también se han reutilizado las clases básicas que deben estar presentes en todo proceso de generación de imágenes: la escena, los objetos, las luces, el observador, la pantalla, etc. En la figura 4.1 puede verse una primera aproximación al modelo de objetos asociado al dominio del problema que estamos analizando.

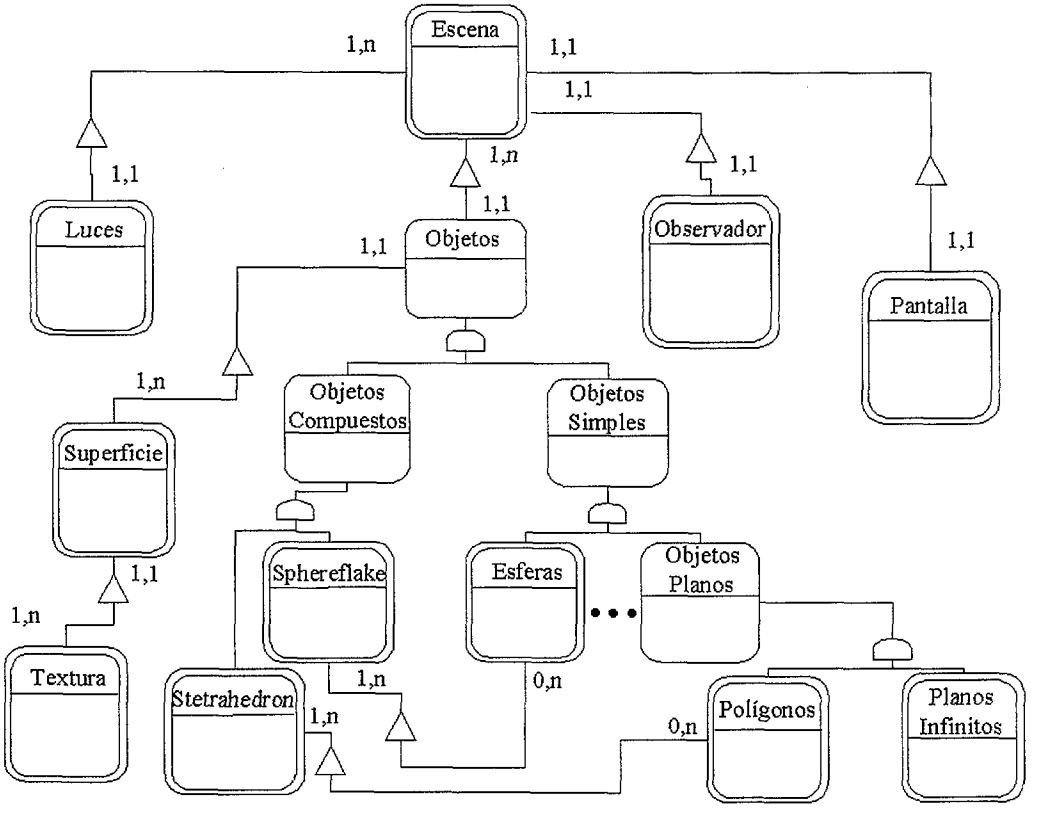



**Figura 4.1 Modelo de objetos de un trazador de rayos genérico.**

Por una parte la escena está compuesta básicamente de objetos y luces situadas en puntos del espacio. Los objetos pueden especializarse en diferentes clases en función de su modo de representación. Podemos tener objetos simples, como esferas o planos, y objetos compuestos como *Sphereflake* o *Stetrahedron*

(figuras recursivas compuestas de esferas o pirámides definidas por E. Haines [HAIN87]). En cualquier caso todo objeto debe tener asociada una superficie que define su color, textura, índices de reflexión y refracción, etc. Junto a las características intrínsecas de la escena, para poder realizar la generación de la imagen bidimensional es necesario añadir información sobre la posición desde donde se observa la escena y las características (resolución, tamaño, etc.) y posición de la pantalla donde ésta será finalmente representada.

La reutilización realizada en las diferentes etapas del ciclo de vida se ha basado en el trabajo de N. Wilt denominado *"Object Oriented Ray Tracing"* [WILT94]. En este trabajo, junto a un estudio teórico del trazado de rayos, N. Wilt aporta una librería donde se implementan en C++ las clases básicas que permiten determinar el color de un pixel de la pantalla mediante el algoritmo de trazado de rayos. Esta librería permitirá la reutilización de la implementación de las clases y funciones básicas y, a su vez, facilitará la extensibilidad del software al posibilitar la inclusión de nuevas funcionalidades. En este caso, nuevas estructuras de descomposición de la escena (árboles octales o matrices de voxel ), o nuevas estructuras a trazar dentro de la escena (haces).

Antes de describir las nuevas clases que soportan el trazado de haces, vamos a realizar un pequeño estudio de la librería OORT en la que se apoya nuestra implementación y de la cual, el algoritmo aquí propuesto, heredará multitud de características.

## 4.2 Estructura de la librería OORT.

OORT es una librería de clases desarrollada en C++ que implementa las clases necesarias para obtener imágenes mediante el algoritmo de trazado de rayos. En este caso la definición de la escena puede realizarse de dos modos diferentes, bien

mediante el lenguaje NFF propuesto por E. Haines [HAIN87], o bien utilizando el propio lenguaje C++ para crear los diferentes objetos de las clases participantes en el proceso de trazado.

El modelo de clases es similar al propuesto en la figura 4.1 aunque en él se han realizado ciertas optimizaciones, dando lugar finalmente al modelo que aparece en la figura 4.2.

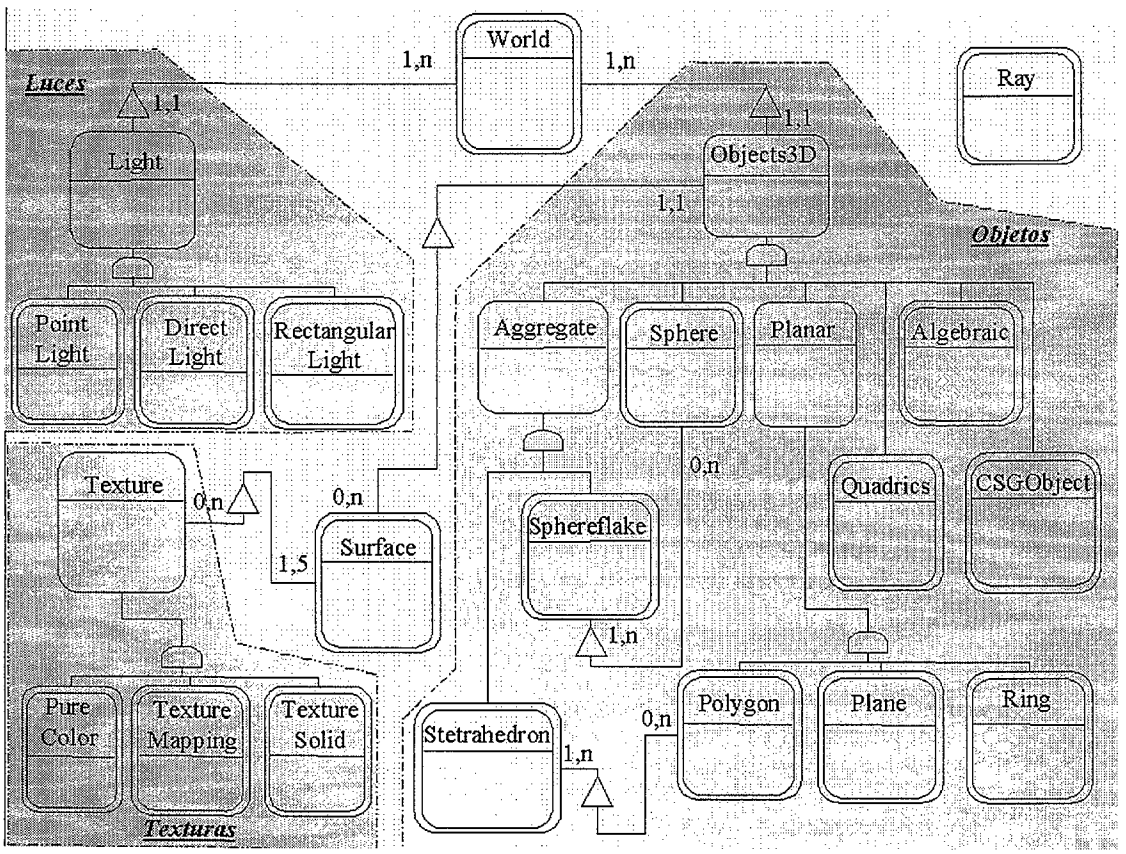



**Figura 4.2 Modelo de objetos del OORT.**

Como podemos apreciar en la figura 4.2, la clase básica de todas las que aparecen en el modelo es la clase *World*, la cual recoge todos los datos y servicios fundamentales que permiten realizar el trazado de los diferentes rayos y por tanto la obtención de la imagen final. Esta clase, junto a los enlaces que le permiten saber los objetos y las luces que contiene la escena, incluye como atributos la información

referente al observador y a la pantalla. A partir de toda esta información la clase *World* es capaz de generar los diferentes rayos primarios (implementados dentro de la clase *Ray*) que permitirán componer la imagen final.

Si se observa la figura 4.2 se puede apreciar que existen tres grandes jerarquías con estructuras de tipo generalización-especialización: objetos (*Object3D*), luces (*Light*) y texturas (*Texture*). La mayor y más compleja de ellas es la que permite definir diferentes tipos de objetos. Como puede apreciarse se pueden definir una amplia variedad de objetos, dese objetos simples formados por una única primitiva, hasta objetos complejos que requieren de la utilización conjunta de varios objetos simples. El modelado de objetos complejos puede realizarse aponyándose tanto en técnicas de representación de contorno (*B-rep*), como de Constructive Solid Geometry (*CSG*), lo cual amplía los posibles tipos de escenas a representar. A su vez, al estar todos asociados a una jerarquía permite aprovechar las características del enlace dinámico propias de los lenguajes OO, facilitando así la comprensión de los programas que manipulan los diferentes objetos.

Como hemos comentado anteriormente, junto a esta jerarquía encontramos otras dos asociadas a las luces y las texturas. La primera de ellas permite definir varios tipos de focos de luz que provocarán distintos efectos sobre la escena. La segunda, asociada a las texturas, nos ofrece diversas alternativas a la hora de establecer el color de la superficie de un objeto (clase *Surface*). Como podemos apreciar en la figura 4.2, una superficie tiene asociadas cinco texturas que le permiten calcular los coeficientes de los cinco componentes (ambiente, difuso, especular, reflexión y refracción) necesarios para evaluar la ecuación de iluminación.

Para implementar finalmente la totalidad de la librería existen otra serie de clases, que no aparecen en el gráfico de la figura 4.2. Dentro de ellas, podemos encontrar algunas que permiten realizar ciertas transformaciones (Matrix), manejar puntos en tres dimensiones (Vector3D), implementar algunas estructuras de datos

necesarias (SimpleList, SimpleStack, Heap), generar las estadísticas de resultados (Statistics), etc.

El proceso más importante es aquel que se encarga de calcular el color de un pixel. En la figura 4.3 se muestra el *escenario*[1] asociado a dicho proceso. Como puede apreciarse en la figura la clase *World*, en la que están almacenadas las características tanto de la pantalla como del observador, es la encargada de controlar el proceso de trazado de un rayo, pero necesita de otras clases para completar su servicio.

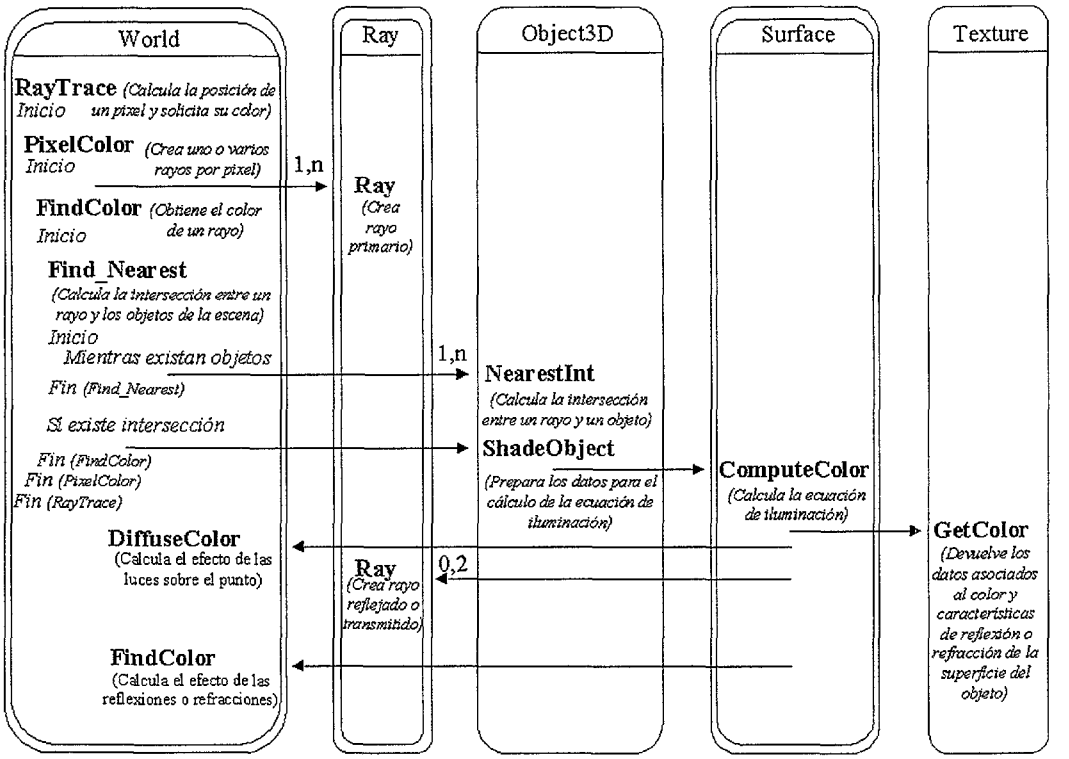



**Figura 4.3 Escenario del servicio encargado de realizar el trazado de los diferentes rayos.**

[1] Un *escenario* es la notación utilizada por P. Coad et al. [COAD97] para describir el funcionamiento de ciertos servicios complejos que necesitan de la participación de objetos de diferentes clases para completar su tratamiento.

El algoritmo de trazado comienza con la llamada del servicio *RayTrace*, el cual se encarga de calcular las coordenadas del pixel a analizar. Para cada uno de los pixel de la pantalla llama a la función *PixelColor* que será la encargada de devolver el color del pixel analizado. Este nuevo servicio primeramente genera uno o varios rayos en función del tratamiento del problema de aliasing, y cada rayo llama a la función *FindColor* que será la que trazará el rayo y devolverá el color final de dicho rayo. El color de este rayo, junto con el del resto de rayos que se han generado para un mismo pixel se utilizan para determinar el color final correspondiente a dicho pixel.

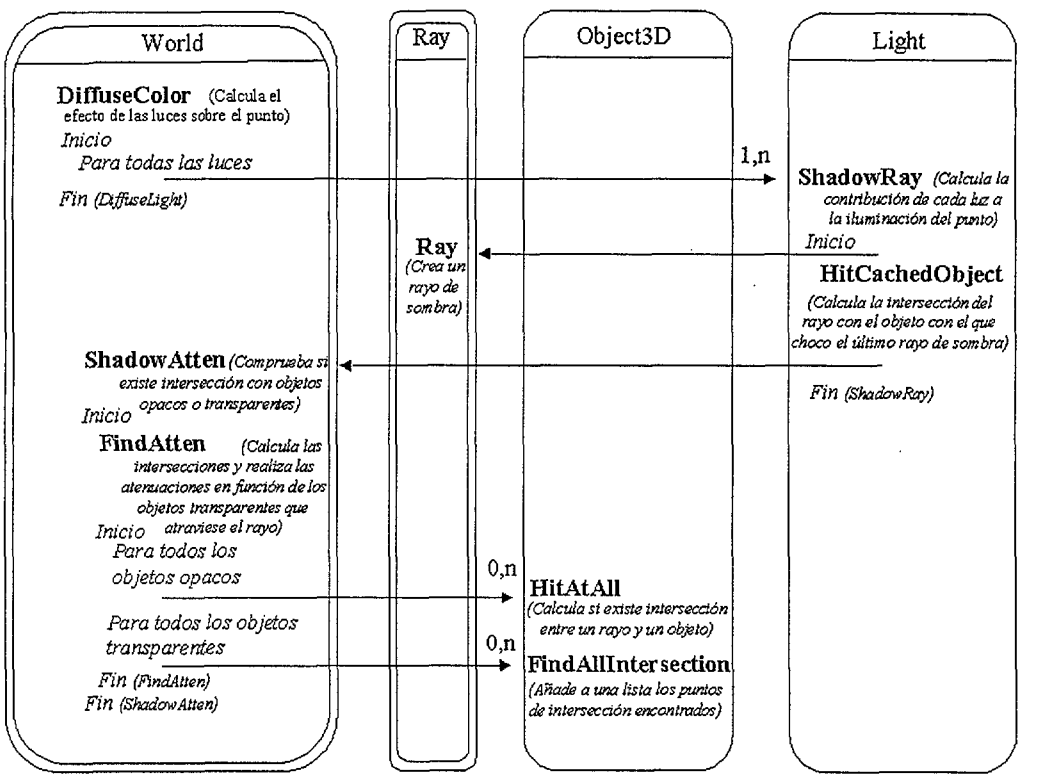



**Figura 4. 4 Escenario del servicio encargado de ver la aportación de las luces a la iluminación del punto de intersección entre un rayo y un objeto.**

El proceso de trazado de un rayo dentro de la escena es realizado, por tanto, por el servicio *FindColor*. Éste primero busca el objeto más próximo con el que choca el rayo (*Find_Nearest*) y si encuentra un objeto realiza el cálculo de la ecuación de iluminación para el punto de intersección encontrado, llamando al servicio *ShadeObject* de la clase abstracta *Object3D*. Por su parte el servicio *Find_Nearest* recorre los diferentes objetos contenidos dentro de la escena y para cada uno de ellos llama al servicio de *Object3D* denominado *NearestInt*. Este servicio se encuentra redefinido en cada clase asociada a un objeto o figura diferente, ya que el cálculo de la intersección entre un rayo y cada tipo de figura es distinto. *NearestInt* será pues uno de los servicios básicos que deberá tener cada nueva clase asociada a un tipo de figura que se desee incorporar a nuestro sistema, ya que será el encargado de determinar si un rayo choca con un objeto de dicha clase o no.

El servicio *ShadeObject* es el que realiza el cálculo de la ecuación de iluminación para cada punto de choque entre un rayo y un objeto. Éste tras calcular algunos datos sobre el punto de intersección encontrado llama a *ComputeColor*. El servicio anterior, *ComputeColor*, pertenece a la superficie del objeto con el que ha chocado el rayo, y será el encargado realmente de calcular la ecuación de iluminación para dicho rayo. Para ello, necesita saber los datos de las características difusa, especular, ambiente, de reflexión o de transmisión de la superficie (cada una de esas características las obtiene llamando al servicio *GetColor* de la clase *Texture*). Junto a ello, genera y traza los rayos de sombra necesarios para saber si el punto está iluminado o en sombra, para ello llama al servicio de *DiffuseLight* de la clase *World* (ver la figura 4.4). Finalmente, si la superficie es capaz de reflejar o transmitir luz debe generarse el rayo reflejado o transmitido y trazarlo dentro de la escena. Para realizar su trazado estos rayos pueden considerarse como rayos primarios y por tanto el servicio encargado de su trazado será el mismo (*FindColor*).

Por tanto, encontramos dos servicios encargados de trazar rayos: *FindColor* (para los rayos primarios, reflejados o transmitidos) y *DiffuseLight* (para los rayos de

sombra). Esta separación en dos servicios se debe principalmente a la diferencia existente en el tratamiento de los rayos de sombra. Por una parte, a los rayos primarios, reflejados o transmitidos les interesa determinar el punto de intersección más próximo entre un rayo y los objetos de la escena. En cambio, los rayos de sombra tan solo deben determinar si existe algún objeto opaco, no necesariamente el más próximo, en el camino del rayo. Por otra parte, si un rayo primario choca con un objeto se debe evaluar la ecuación de iluminación nuevamente, produciéndose un tratamiento recursivo del trazado (*FindColor*).

## 4.3 Métodos de aceleración incluidos en el OORT.

El proceso anteriormente descrito puede aplicarse al algoritmo de trazado de rayos estándar, el cual, como comentamos en los primeros capítulos, tiene asociado un alto coste computacional y por tanto un elevado tiempo de ejecución. Por tanto es interesante introducir técnicas que aceleren este proceso.

Dentro de la librería OORT se incluyen algunas técnicas de aceleración que permiten reducir el tiempo de ejecución. Dichas técnicas pretenden básicamente reducir el tiempo necesario para trazar un rayo dentro de la escena, al reducir el número de objetos con los que un rayo debe calcular su intersección. La primera de las técnicas se apoya en la coherencia de objetos y para ello asocia a la escena una estructura que facilita el seguimiento de la trayectoria de un rayo. La segunda se corresponde con el tratamiento de los rayos de sombra y se apoya en la idea de la coherencia de rayos.

La estructura seleccionada para descomponer la escena es una jerarquía de volúmenes envolventes (Bounding Volume). Para ello, descompone la escena guiándose por los objetos contenidos en ella. El criterio de descomposición se basa en los trabajos de Goldsmith y Salmon [GOLD87]. Estos definen una heurística para

estimar el coste de la jerarquía que se está construyendo y pretenden generar la jerarquía que, en función de dicha heurística, tenga coste mínimo. Para definir su heurística ellos parten de una serie de premisas. Suponen que todos los objetos en la jerarquía requieren el mismo tiempo para calcular su intersección con un rayo. Por otra parte, asumen que los volúmenes envolventes son convexos y, por tanto, la probabilidad de que un rayo choque con dicho volumen es proporcional al área de su superficie [STON75]. Por último, solamente se debe prestar atención a aquellos rayos que chocan con el volumen que envuelve a la totalidad de la escena (volumen asociado a la raíz de la jerarquía).

De este modo, si un rayo choca con el volumen que envuelve a la raíz $BV_r$, la probabilidad de que también choque con un volumen interior $BV_i$ es proporcional a $A_i / A_r$, donde $A_i$ y $A_r$ son el área de la superficie de $BV_i$ y $BV_r$ respectivamente. Debido a que es necesario calcular la intersección entre el rayo y todos los objetos contenidos en el volumen $BV_i$, si un rayo choca con este volumen, la contribución de $BV_i$ al coste total estimado para la jerarquía será $kA_i / A_r$, donde $k$ es el número de objetos contenidos en el volumen $BV_i$. El coste total de la jerarquía es la suma de los costes parciales de cada volumen envolvente creado. Este coste es, por tanto, el coste estimado del número de intersecciones que debemos calcular si un rayo choca con el volumen envolvente inicial (asociado a la raíz de la jerarquía).

El algoritmo que construye la jerarquía analiza uno a uno cada objeto. El proceso de inserción de cada objeto comienza siempre por la raíz de la jerarquía, seleccionando el subárbol cuyo volumen incrementaría menos el área de su superficie si se le añadiese dicho objeto. Este proceso continúa hasta encontrar un nodo terminal dentro del árbol. En ese caso, la decisión de añadir un nuevo objeto al nodo terminal o crear un nuevo volumen envolvente, se basa en el análisis del coste que supone cada alternativa con respecto al coste total de la jerarquía, seleccionándose aquella que suponga un menor coste (ver figura 2.1).

Para envolver a los objetos se ha elegido una caja alineada con los ejes ya que es una figura que permite calcular rápidamente la intersección entre ella y un rayo. En este caso, el cálculo del área de la superficie viene dado por la siguiente expresión $2wh + 2wd + 2hd$, donde $w$, $h$ y $d$ son respectivamente el ancho, alto y profundo de la caja. Dentro de esta librería se ha utilizado una expresión similar a la anterior pero más fácil de calcular: $w(h + d) + hd$.

Por otra parte, para optimizar el recorrido de la jerarquía OORT se basa en el algoritmo propuesto por Kay y Kajiya [KAY86]. Este método requiere una ordenación de la jerarquía con respecto a cada rayo basado en la distancia de los puntos de intersección con los volúmenes envolventes. Se utiliza una estimación de dicha distancia como clave de ordenación, almacenándose los resultados en una cola con prioridades ("*CP*"). En el gráfico 4.1 se incluye el pseudocódigo del algoritmo de recorrido de la jerarquía propuesto por Kay y Kajiya.

```
Find_Nearest (rayo, raíz)
Si (el rayo choca con el volumen inicial asociado a la raíz)
        Insertar todos los miembros de la raíz en CP, utilizando como clave el
        punto de intersección con el volumen inicial más próximo.
Mientras (CP no esté vacía) hacer
        Extraer de CP el objeto mínimo obj.
        Si (la clave de obj es mayor que la intersección más cercana encontrada).
                No existe ningún objeto más cercano. Terminar.
        Si no,
                Actualizar la intersección más cercana con el nuevo valor
                calculado.
Devolver la intersección más próxima.
```

**Gráfico 4.1 Pseudocódigo del algoritmo de recorrido de la jerarquía de volúmenes propuesto por Kay y Kajiya.**

Este algoritmo de recorrido de la jerarquía es útil cuando se desea obtener el punto de intersección más próximo y, por tanto, es interesante utilizarlo cuando se desea trazar un rayo primario, o aquellos que se comportan como tales (rayos reflejados o transmitidos), dentro del servicio *Find_Nearest*. En cambio, cuando no es necesario encontrar el punto de intersección más próximo, sino tan solo si existe alguna intersección este método no parece el más indicado. Por ello, el trazado de los rayos de sombra utiliza el algoritmo estándar de recorrido de este tipo de jerarquías (ver gráfico 4.2), implementado dentro del servicio *FindAtten*.

El hecho de que cada rayo que choque con un objeto, ya sea primario o secundario, genera tantos rayos de sombra como luces haya en la escena, hace especialmente interesante estudiar alguna posible optimización en el trazado de dichos rayos. En general las optimizaciones propuestas para acelerar el trazado de este tipo de rayos se apoyan en las ideas aportadas por la coherencia de rayos. Parten, por tanto, de la base de que rayos próximos entre si suelen chocar con los mismos objetos de la escena. Por otra parte en este caso, no interesa determinar cual es el primer punto de intersección entre un rayo y los objetos de la escena, sino que interesa conocer un rayo en su camino hacia una fuente de luz encontrará al menos un objeto opaco.

```
FindAtten (rayo,nodo)
Si (nodo es terminal)
    Calcular intersección con objetos del nodo.
Si no,
    Si (el rayo choca con un volumen envolvente)
        Para cada hijo del nodo.
            FindAtten(rayo,hijo)
```

**Gráfico 4.2 Pseudocódigo del algoritmo básico de recorrido de una jerarquía de volúmenes.**

La técnica de aceleración de los rayos de sombra más conocida es la aportada por Haines y Greenberg [HAIN86] denominada "*light buffer*". En este caso, como ya apuntamos en el capítulo segundo, cada fuente de luz se encierra en una caja alineada con los ejes, cuyas caras son divididas en rectángulos. Cada uno de los rectángulos tiene una lista de objetos que pueden verse desde la fuente de luz si se mira a través de un rectángulo. En este caso, cada rayo de sombra choca con un rectángulo dado y por tanto tan solo es necesario comprobar el comportamiento del rayo para los objetos contenidos en el rectángulo dado.

En cualquier caso, existen otras alternativas de aceleración más simples pero en algunos casos bastante efectivas. Dentro de OORT se implementa una técnica que denominan "*shadow cache*" y que consiste en asociar a cada fuente de luz un enlace al último objeto opaco que un rayo encontró en su camino. De este modo, si un rayo choca con un objeto se considera que los rayos vecinos es muy probable que también choquen con él. Por tanto, cada vez que se genera un rayo de sombra se calcula primero la intersección con el objeto opaco con el que chocó el rayo de sombra anterior y sólo si este nuevo rayo no choca con dicho objeto se analiza su trayectoria, para lo cual se sigue el proceso descrito anteriormente. Este método de aceleración del trazado de rayos de sombra se ha incorporado a la librería OORT y se incluye dentro del servicio *HitCachedObject* como una comprobación previa (figura 4.4).

Para implementar dentro de OORT las diferentes técnicas de aceleración se han incluido las clases que aparecen en la figura 4.5. En la figura aparecen sombreadas con diferente color las modificaciones correspondientes a cada optimización: *shadow cache* y jerarquía de volúmenes envolventes. La primera de ellas, tan solo necesita la inclusión de un enlace que permita a cada luz conocer el objeto con el que chocó el último rayo de sombra que se dirigía hacia dicha fuente de luz. Por otra parte, la segunda optimización requiere de un tratamiento más complejo. Para su implementación necesita la inclusión de varias clases que permitan definir y manejar la jerarquía de volúmenes envolventes deseada. Como puede apreciarse el volumen elegido dentro de OORT para envolver a un objeto

cualquiera es una caja alineada con los ejes (*Bounding Box*). Esta figura hereda ciertas características de dos clases abstractas: *Bounding Volume* y *AxisAligned Box*. La primera implementa la idea de volumen envolvente y la segunda incluye las características asociadas a una caja alineada con los ejes definida mediante dos puntos en 3D. De este modo, la inclusión de nuevas figuras envolventes es fácil de realizar ya que la clase genérica *Bounding Volume* incluye las funciones necesarias para utilizar la jerarquía. Por otra parte, para obtener la caja mínima que envuelve a cada objeto la mayoría de las clases derivadas de *Object3D* incluye un servicio denominado *Bbox* que se encarga de ello. Pero no todos los objetos (por ejemplo un plano infinito) pueden estar contenidos en una caja envolvente y por tanto la clase abstracta *Object3D* necesita incluir una implementación por defecto del servicio *BBox*. Junto a ello, dentro de los servicios de la clase *World* hay que incluir uno nuevo (*BuildHierarchy*) que se encargue de construir la jerarquía de cajas envolventes.

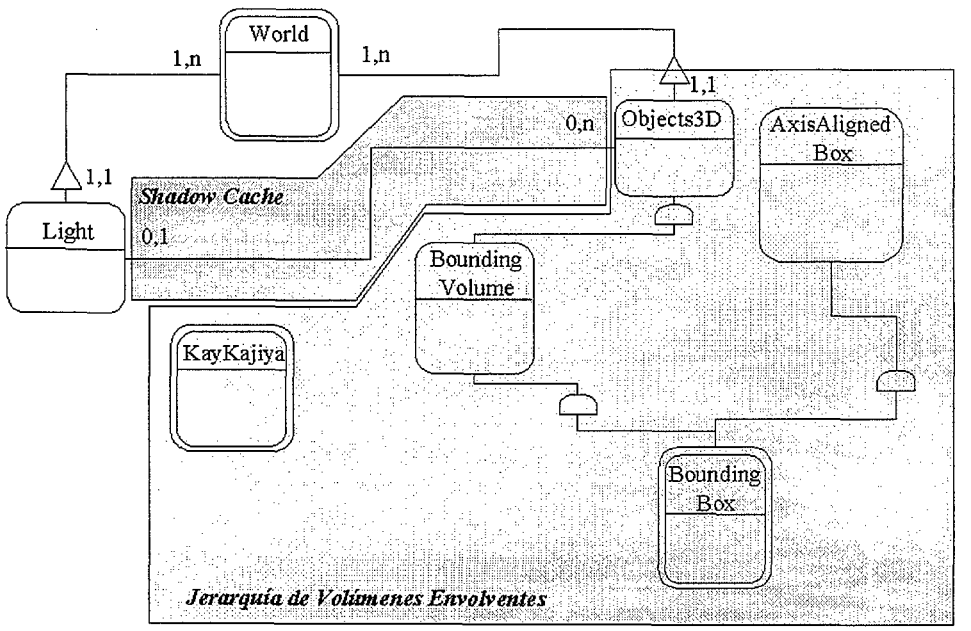



**Figura 4.5 Clases y relaciones necesarias para implementar las diferentes técnicas de aceleración incluidas en la librería OORT.**

Por otra parte, se incluye la clase KayKajiya que se utilizará para implementar los servicios necesarios para realizar el recorrido de la estructura jerárquica obtenida.

## 4.4 Clases adicionales para el soporte del trazado de haces.

Tras comentar de manera detallada las principales características de la librería OORT que ha servido de base para desarrollar los nuevos algoritmos, vamos a analizar la estructura de las nuevas clases y sus principales servicios.

En este caso, se propone otra alternativa de optimización que incluye, por una parte, una nueva estructura de descomposición de la escena (árbol octal) y, por otra, un nuevo tipo de elemento a trazar dentro de ella (haz de rayos).

Comenzaremos explicando la implementación de la nueva estructura de descomposición de la escena. Las clases y servicios necesarios en este caso son bastante similares a los que se han incluido para introducir las optimizaciones descritas en el apartado anterior, aunque evidentemente difieren en la implementación final de las clases y servicios.

En la figura 4.6 se indica el modelo Clase&Objeto asociado a las nuevas clases. En él puede verse como aparecen básicamente dos nuevas clases: *Haz* y *OctreeNodo*. La primera de ellas, incluye los atributos y servicios que describen a un haz. Esta clase será la encargada tanto del trazado de los diferentes haces, como de la determinación de los pixel contenidos en cada uno de ellos. Por otra parte, la clase *OctreeNodo* implementa la estructura de descomposición de árbol octal. Ésta clase hereda características de la clase *AxisAlignedBox* ya que cada nodo del árbol tiene asociado un cubo, que en este caso, debido a las restricciones impuestas en el proceso de descomposición, es un cubo alineado con los ejes de coordenadas. Por

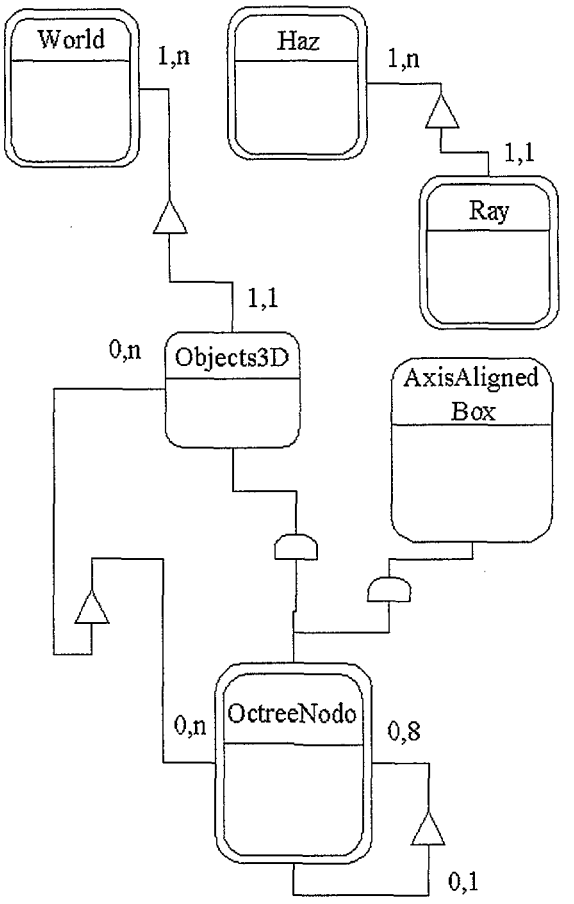



**Figura 4.6 Modelo de objetos asociado a las nuevas clases.**

otra parte, al igual que ocurría con los volúmenes envolventes podemos encontrar objetos que no puedan incluirse dentro de un cubo y por tanto no puedan asociarse a un árbol octal. Por ello el mundo mantiene una lista de objetos que incluye a los objetos que no pertenecen al árbol octal. Por último, como sucedía con la clase *Bounding Volume* la clase *OctreeNodo* puede considerarse también como un nuevo tipo de objeto y por tanto heredar las características de la clase base *Objects3D* (raíz del árbol que contiene al resto de objetos de la escena).

Finalmente debido al modo de realizar el proceso de descomposición (*descomposición guiada por el espacio),* pueden obtenerse, y generalmente se obtienen, estructuras en las que un objeto está asociado a más de un nodo. Esto último obliga a añadir una nueva comprobación cada vez que un rayo choca con un objeto, pues debemos asegurarnos de que el punto de intersección se encuentra dentro del nodo en el que está actualmente el rayo.

### 4.4.1 Creación de la estructura de descomposición.

Para manipular y crear la clase *OctreeNodo* es necesario incluir dos servicios básicos: determinar la inclusión de un objeto dentro de un nodo y construir la estructura de descomposición final de la escena.

El primero debe asociarse a cada uno de los diferentes tipos de objetos o figuras que el sistema es capaz de generar. Este servicio se encargará de determinar si un objeto está total o parcialmente incluido dentro del cubo asociado a un nodo dado (servicio *IntersecCubo*). En este caso, dentro de este trabajo, se ha incluido este nuevo servicio tan sólo a las clases asociadas a las esferas y los polígonos (algoritmos descritos en detalle en el capítulo anterior). El resto de objetos a generar se consideran inicialmente como objetos a los que no se les puede calcular dicha función y por tanto no formarán parte de los objetos contenidos dentro del árbol octal, sino que se analizarán siempre a parte como sucede con los planos infinitos dentro de las jerarquías de volúmenes.

Junto al servicio que determina la inclusión de un objeto en un cubo, es necesario incluir dentro de la clase *World* un servicio que permita construir la estructura de árbol octal que descomponga la escena (*CrearOctree*). En la figura 4.7 puede verse el escenario que describe el funcionamiento de este servicio.

Al servicio *CrearOctree* se le pasa una lista con los objetos contenidos en la escena y, a partir de ella, genera una lista auxiliar con los objetos a los que el sistema es incapaz de determinar su inclusión total o parcial dentro de un cubo (servicio *ObetoInc*). Con el resto de objetos determina el tamaño mínimo del cubo que los contendrá. Para ello, calcula para cada objeto la caja envolvente asociada (determinada por el servicio *Bbox*). La situación de los extremos de las diferentes cajas obtenidas se utiliza para determinar el tamaño y posición del cubo mínimo inicial que es capaz de contener a los objetos de la escena y que, por tanto, constituirá la raíz del árbol octal.

Tras determinar el cubo mínimo inicial se llama al constructor de la clase *OctreeNodo*, al cual se le pasan como argumentos su posición y una lista con los objetos que posiblemente estén contenidos dentro de él. Para cada uno de estos objetos se debe comprobar si realmente están total o parcialmente contenidos en el nodo (*IntersecCubo*).

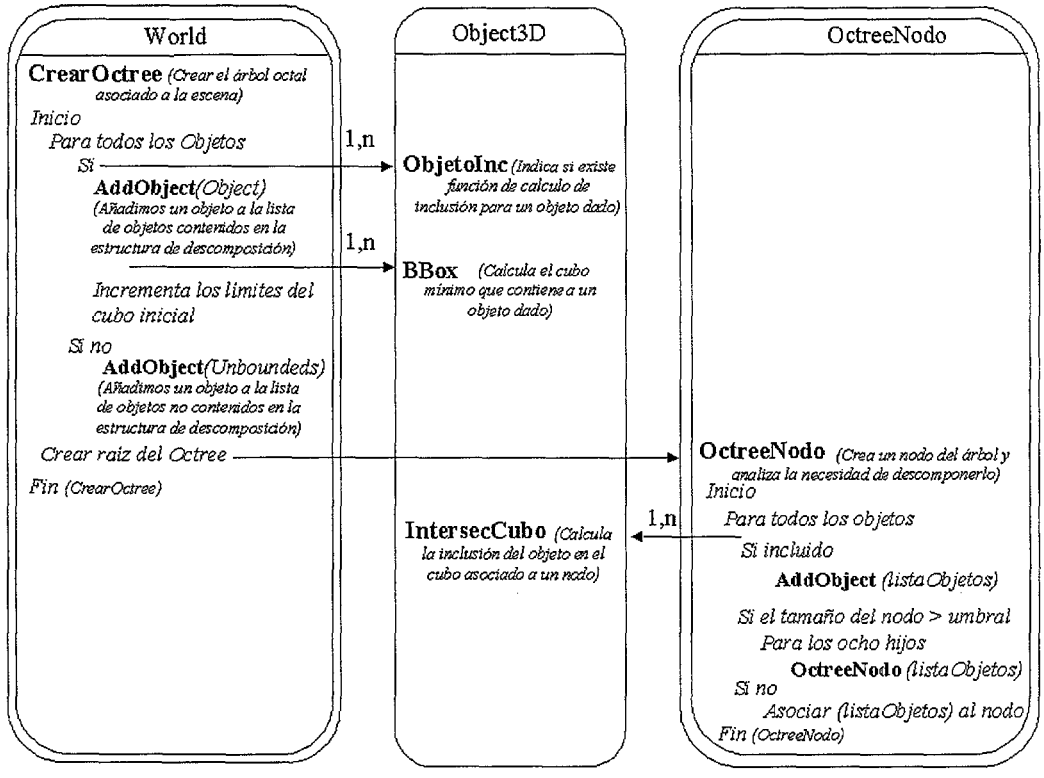



**Figura 4.7 Escenario correspondiente al proceso de creación de la estructura de árbol octal asociada a la escena.**

Este proceso de descomposición es un proceso recursivo que finaliza según el criterio mencionado en el capítulo anterior cuando el tamaño del nodo es inferior a un determinado umbral. De este modo se aprovecha al máximo el hecho de que los rayos se propagan de manera conjunta dentro de un haz por las zonas de la escena que no contienen objetos.

Al final del proceso de descomposición se obtiene una estructura de árbol octal en la cual cada nodo terminal etiquetado como "ocupado" (nodo que contiene objetos en su interior) tiene asociado una lista de los objetos contenidos total o parcialmente en él.

## 4.4.2 Trazado de haces.

Una vez creada la estructura de descomposición de la escena se puede proceder a crear y trazar los diferentes haces. Aunque la idea general del proceso de trazado de haces ya ha sido descrita en el capítulo anterior, en este apartado se describen algunas consideraciones de la implementación de este proceso de trazado. En las figuras 4.8, 4.9 y 4.10 se muestran los escenarios asociados a los principales servicios del proceso de trazado de haces.

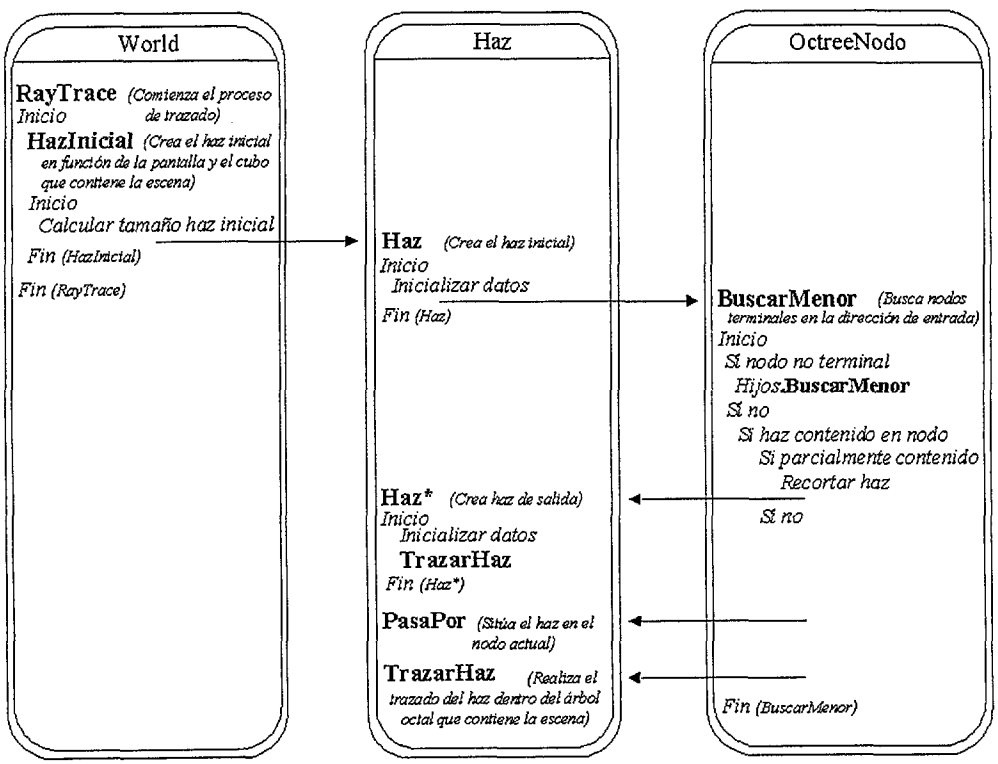



**Figura 4.8 Escenario asociado a la creación de los haces iniciales con los que comienza el trazado de la escena.**

El proceso de trazado comienza con la llamada al servicio *RayTrace* de la clase *World* (ver figura 4.8). Éste básicamente realiza una llamada al servicio encargado de definir el haz inicial a trazar dentro de la escena. Para crear el haz inicial tiene en cuenta tanto la pantalla como el cubo que contiene a la escena. Por tanto, primeramente proyecta los extremos de la pantalla sobre el plano asociado a la cara principal de propagación. Tras ello, en función de dicha proyección y de los extremos del cubo inicial que contiene la escena, ajusta el tamaño del haz inicial y lo crea.

Pero, como indicamos en el capítulo anterior, este haz creado inicialmente será de mayor tamaño que los nodos que se encuentran situados en la cara principal de propagación (figura 3.7), con lo cual deberá subdividirse antes de que pueda entrar en un nodo. Para ello el proceso de creación de un haz llama al servicio *BuscarMenor* de la clase *OctreeNodo*. Este servicio realiza un recorrido del árbol buscando nodos terminales en una determinada dirección.

En este caso fruto de la búsqueda podemos encontrar nodos por los que no pasará el haz y otros por los que si lo hará, al estar la proyección del haz total o parcialmente contenida dentro de la cara de entrada del nodo. En los casos en los que el haz entra en el nodo dado, podemos encontrarnos con dos supuestos: (1) el haz está totalmente contenido en el nodo y por tanto dicho haz solo pasará por el nodo encontrado; (2) el haz está parcialmente contenido en la cara de entrada del nodo y debe reajustarse su tamaño. En el primer caso tan solo debemos situar el haz en el nodo de llegada (servicio *PasaPor* de la clase *Haz*) y continuar el trazado a partir de dicho nodo (servicio *TrazarHaz* de la clase *Haz*). En el segundo caso, al estar el haz parcialmente contenido en el nodo, se debe proceder previamente a un proceso de recorte que ajuste el tamaño del haz a la cara de entrada del nodo al que ha llegado. En este caso el haz inicial dará lugar a varios haces de entrada asociados a los diferentes nodos que contiene en su interior. Tras esta definición de tamaño del nuevo haz se procede a su creación llamando a una redefinición del constructor de la clase *Haz** que crea un haz y procede a su trazado.

Una vez que hemos determinado el haz que entra en un nodo se procede a su trazado, para lo cual se llama al servicio *TrazarHaz* de la clase *Haz* (ver figura 4.9). Si el nodo al que llega el haz está ocupado por objetos se deberán determinar los pixel contenidos en él y analizar si los rayos individuales asociados a cada uno de dichos pixel choca o no con los objetos contenidos en el nodo (servicio *BuscarPixel* de la clase *Haz*, ver figura 4.10). Por otra parte, si el nodo al que llega está vacío el haz debe propagarse por el árbol octal y en su trazado podrá dar lugar a otros haces de salida de menor tamaño. En la figura 4.9 se detalla la implementación del algoritmo general descrito en el gráfico 3.4. Este algoritmo tiene cuatro tareas básicas: (1) buscar direcciones de salida; (2) recortar el haz respecto a las caras de salida (obtener el haz de salida estándar); (3) buscar vecinos en una determinada dirección; (4) recortar, si fuese necesario, el haz de salida con respecto a las caras por las que el haz entra al nodo vecino.

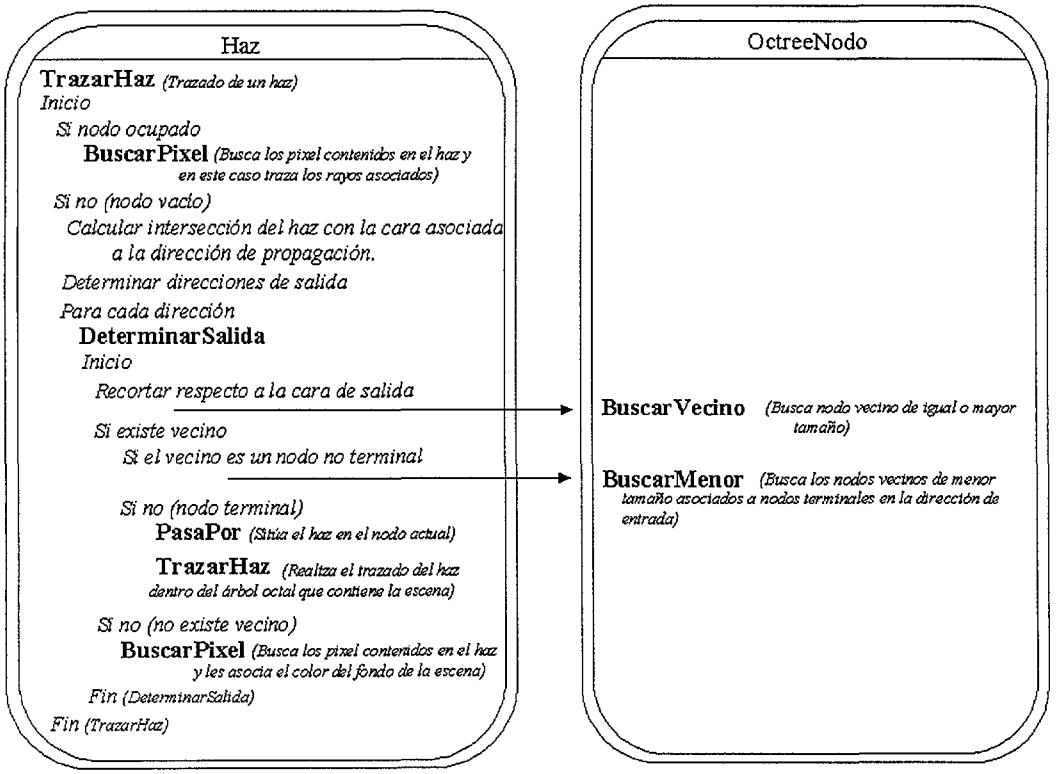



**Figura 4.9 Escenario asociado al trazado de un haz dentro de la estructura de árbol octal.**

Para finalizar el estudio de los principales servicios utilizados en la implementación del algoritmo de trazado de haces, vamos a describir el escenario asociado a la determinación de los pixel contenidos en un haz (ver figura 4.10).

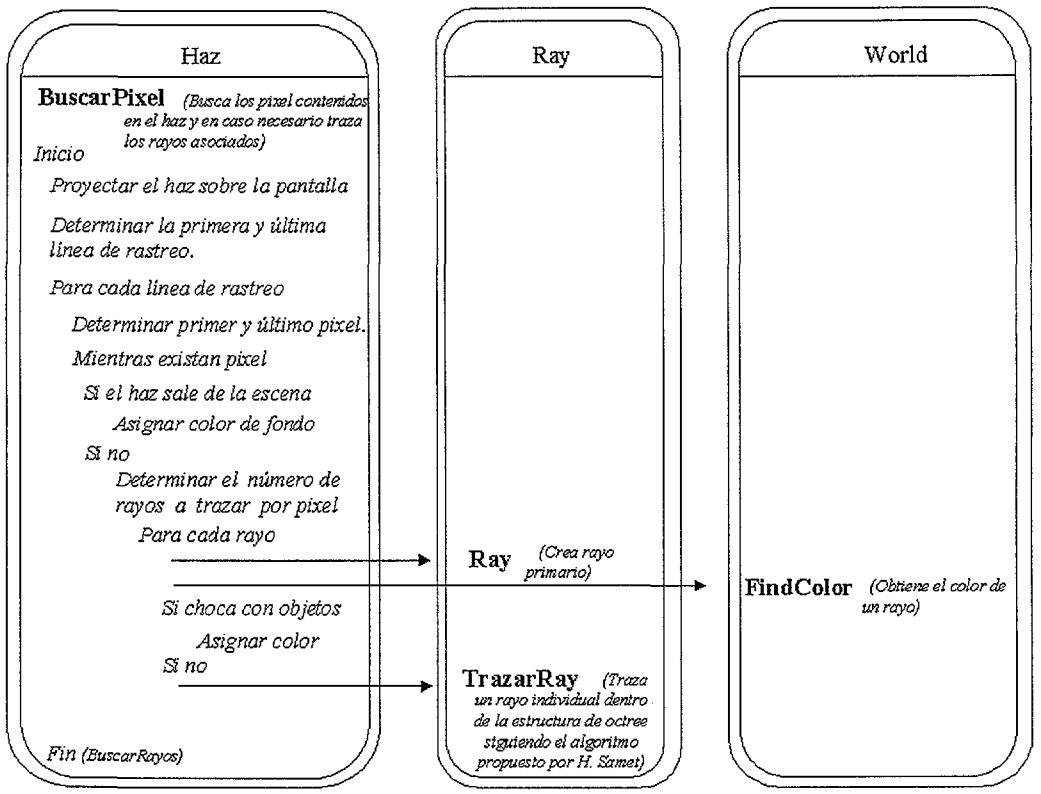



**Figura 4.10 Escenario asociado a la determinación de los pixel contenidos en un haz.**

Si analizamos la figura 4.10, la determinación de los pixel contenidos se apoya en un algoritmo clásico de rellenado de polígonos, ya que nos encontramos ante un problema semejante "determinar los pixel contenidos dentro de una figura poligonal". Para conseguir llegar a esta situación de semejanza, debemos reducir el problema a 2D. Para ello, realizamos la proyección del haz sobre el plano de la pantalla, consiguiendo como resultado un polígono que se corresponde con la sección del propio haz.

Tras la generación de la sección poligonal asociada al haz, la búsqueda de los pixel contenidos se apoya, como ya hemos indicado, en el algoritmo clásico de rellenado de polígonos mediante líneas de rastreo (ver figura 3.14). Después de determinar los pixel contenidos el siguiente problema es definir el número de rayos que se trazarán para finalmente obtener el color de cada pixel. En la implementación realizada como soporte a este trabajo, se han considerados sólo dos situaciones: generación de un único rayo por pixel y utilización de sobremuestreo uniforme mediante la generación de un número constante de rayos por pixel.

Para cada rayo lanzado desde el observador nuestro algoritmo debe analizar las intersecciones con los diferentes objetos contenidos en el nodo en el que el haz y, por tanto el rayo, se encuentra (llamada al servicio *FindColor* de la clase *World*). Si como resultado de dicha comprobación el rayo choca con un objeto el color calculado servirá para determinar el color final del pixel. En otro caso, es decir, si el rayo no choca con ningún objeto, dicho rayo continúa su trazado de modo individual utilizando para ello el algoritmo propuesto por H. Samet [SAME89].

# Capítulo 5. Evaluación de resultados de la versión secuencial.

## 5.1 Introducción.

En este capítulo vamos a analizar el comportamiento del algoritmo descrito anteriormente utilizando para ello diferentes escenas. Para evaluar su eficiencia, realizaremos una comparación de los resultados obtenidos por nuestro algoritmo con los obtenidos por otras alternativas que realizan un trazado individual de los diferentes rayos que formarán la imagen final. En todos los casos, las pruebas se han realizado considerando que las superficies no reflejan, ni transmiten rayos y que no existen focos de luz, con lo que nos aseguramos que en las escenas sólo se trazan rayos primarios. Para procesar las diferentes escenas y obtener los valores del tiempo de procesado (medido siempre en segundos), se ha utilizado un ordenador Pentium a 350 Mhz, con 64 Mb de memoria principal.

En un primer acercamiento veremos el comportamiento individual del trazado de haces, analizando como le afecta la estructura de descomposición utilizada, la distribución de los objetos en la escena o el incremento en la calidad de la imagen a

generar. A su vez se estudiaran las tareas básicas de este algoritmo con el objetivo de determinar cuál o cuáles de ellas son las más costosas y, por tanto, las que serían el principal objetivo en un estudio sobre paralelización de este algoritmo. Dicho análisis nos permitirá introducir el proceso de paralelización que se describirá en el siguiente capítulo.

Junto al estudio anterior, nuestro algoritmo se comparará con otros de trazado individual de rayos que utilizan como estructura de descomposición tanto árboles octales, como otro tipo estructuras (matrices de cubos o voxel y jerarquías de volúmenes envolventes). Esta comparación con métodos que utilizan otras estructuras de descomposición permitirá demostrar la amplia aplicación del método descrito en esta tesis y abundará en los beneficios que se pueden esperar de su aplicación.

El hecho de que la complejidad de los algoritmos de trazado de rayos y, por tanto, sus tiempos de respuesta, dependan básicamente de la calidad de la imagen y la complejidad de la escena (distribución, número y características de los objetos contenidos en ella), ha condicionado los diferentes experimentos realizados.

Es por ello que en todos los casos anteriores se estudiarán la evolución de los resultados obtenidos por los diferentes algoritmos conforme incrementamos la resolución de la imagen. Con ello se pretende mostrar la evolución de los tiempos de respuesta al aumentar la calidad de la imagen, calidad que, en general, va unida a altas resoluciones y a la utilización de sobremuestreo para reducir los efectos del aliasing.

En general, a la hora de analizar la eficiencia de los algoritmos de síntesis de imágenes siempre nos encontramos con el mismo problema: seleccionar escenas que permitan analizar su comportamiento lo más ampliamente posible. Esto se debe a que, como hemos indicado, la eficiencia de los diferentes algoritmos está fuertemente ligada a la escena que se está representando. Por ello, dentro de nuestro

trabajo hemos creado dos escenas simples pero en las que se reflejan dos situaciones un tanto extremas en cuanto a distribución de los objetos dentro de la propia escena. Estas escenas junto con otras con apariencia más real serán las utilizadas como casos de prueba durante el siguiente estudio.

## 5.2 Descripción de las escenas utilizadas.

Para analizar el comportamiento del algoritmo que aquí se propone, es necesario seleccionar algunas escenas que sirvan de banco de prueba. La elección de estas escenas no es trivial pues, como sucede con el resto de implementaciones del algoritmo trazador de rayos, su eficiencia se ve altamente influenciada por la escena utilizada para su análisis. Por ello, junto a algunas escenas que pueden ser consideradas como *"reales"*, ya que representan objetos que podemos considerar como tales (*delfines, tetera, habitación, torre Eiffel,* etc.), hemos querido definir otras de apariencia menos real, a las que hemos denominado "*sintéticas*", pero que permitan reflejar situaciones que pueden considerarse un tanto extremas por la distribución de los objetos que en ellas se muestran (ver figuras 5.1 y 5.2). Todas estas escenas (*reales* y *sintéticas*) nos permitirán analizar el comportamiento individual del trazado de haces, así como realizar un estudio comparativo de su eficiencia con respectos a diferentes alternativas de aceleración que utilizan tanto árboles octales como otras estructuras de descomposición de la escena.

La primera de las escenas creadas con el propósito de reflejar situaciones extremas aparece en la figura 5.1. En ella encontramos un conjunto de objetos que se distribuyen de modo aleatorio por el espacio de la escena. Ésta la forman unas 376 esferas de diferente tamaño distribuidas de manera aleatoria y en ella, como puede apreciarse, existen pocas áreas vacías o que presentan una gran concentración de objetos. Esta escena permitirá analizar el comportamiento de los árboles octales cuando dentro de la escena los objetos se distribuyen de manera casi uniforme.

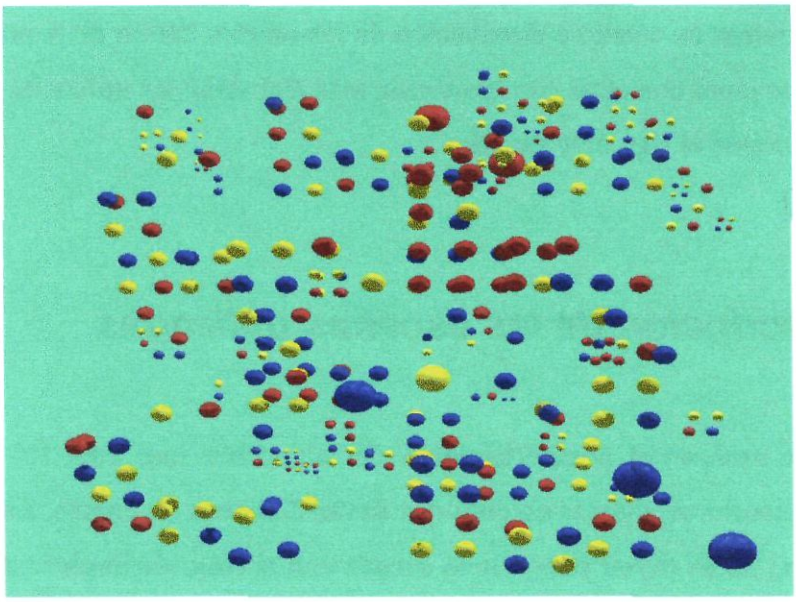


**Figura 5.1 Escena 1. Distribución aleatoria de esferas.**

La otra escena, refleja otra situación también un tanto extrema en la que existe una gran área de trabajo y una fuertísima concentración de los objetos en la parte central de la misma (ver figura 5.2). Esta escena está formada por un polígono, situado en el fondo de la escena, y 91 esferas definidas como una "Sphereflake" (composición recursiva de esferas unidas, definida inicialmente por E. Haines [HAIN87] para analizar el comportamiento del trazado de rayos). El tamaño y situación del polígono hace que el tamaño del cubo que contendrá a la escena sea muy grande respecto al tamaño de las esferas. Con ello se pretende simular el efecto que se provoca en escenas de tipo real en las que el espacio que ocupa ésta es muy grande y en ella existen un número pequeño de zonas con gran concentración de objetos (en algunos casos este efecto se denomina "tetera en un campo de fútbol").

El resto de escenas utilizadas están definidas mediante ficheros de tipo NFF (Neutral File Format, en el anexo A aparece una descripción más detallada del formato de este tipo de ficheros) definido por E. Haines [HAIN87]. La mayoría de ellas están modeladas mediante mallas de polígonos.

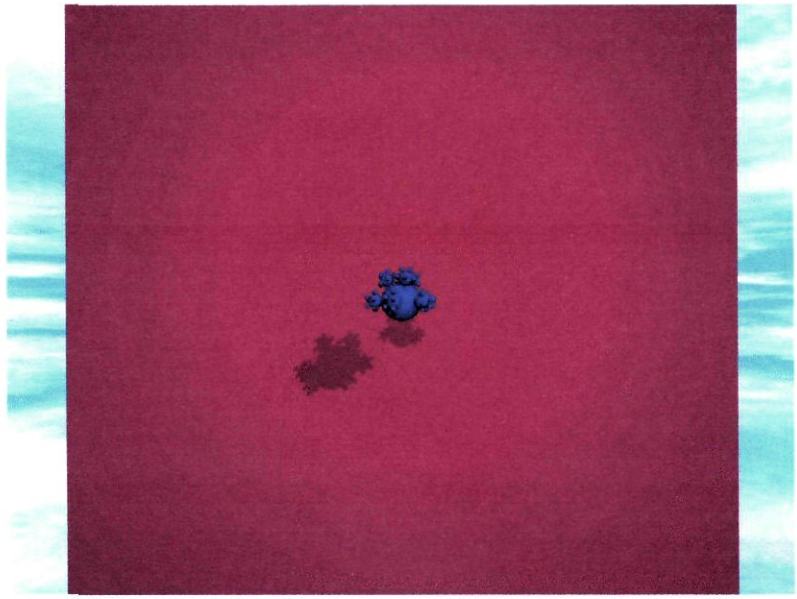


**Figura 5.2 Escena 2. Fuerte concentración de esferas en el centro de la escena.**

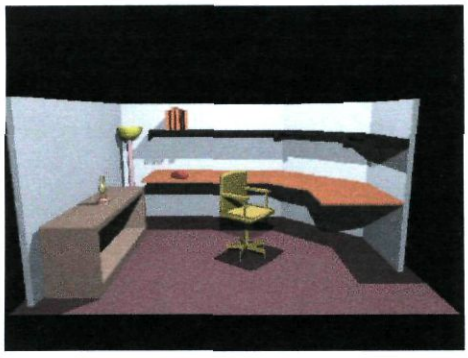
La posibilidad que ofrece nuestro entorno de procesar escenas definidas mediante formato NFF, hace que éste sea capaz de analizar un amplio abanico de escena, como podremos ver en las pruebas realizadas.

No todas las escenas utilizadas han sido definidas inicialmente mediante fichero NFF. Para su creación se han utilizado programas que facilitan su modelado, como 3D Studio Max. Estas escenas han sido posteriormente transformadas a un fichero con formato NFF a través de un programa conversor. La disponibilidad de dicho programa, que convierte ficheros de diferentes formatos al de entrada de nuestro sistema, amplía todavía más las posibilidades de utilización de este sistema.

Aunque a lo largo del desarrollo de esta tesis se han analizado un gran número de escenas con resultados muy semejantes, en la figura 5.3 pueden verse algunas de las que han sido seleccionadas para comprobar la eficiencia de los diferentes algoritmos estudiados. Estas escenas junto a las anteriores (ver figuras 5.1 y 5.2) serán utilizadas en los apartados posteriores para comprobar las cualidades del trazado de haces y compararlas con otras alternativas de aceleración.

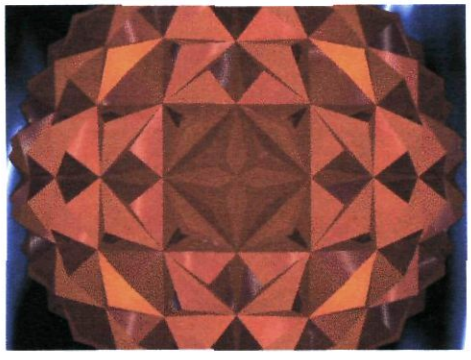
**Delfines**

**Maceta**

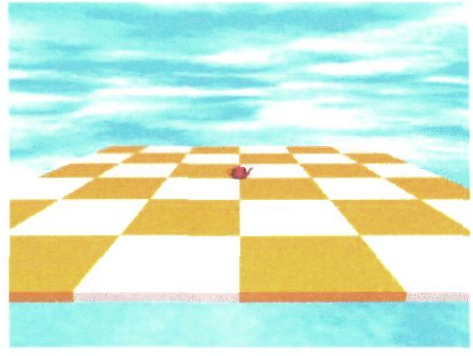


**Habitación**

**Misil**

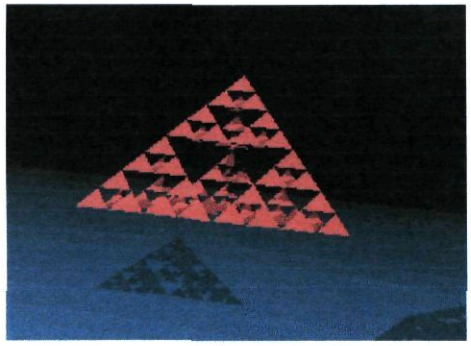


**Bola espacial**

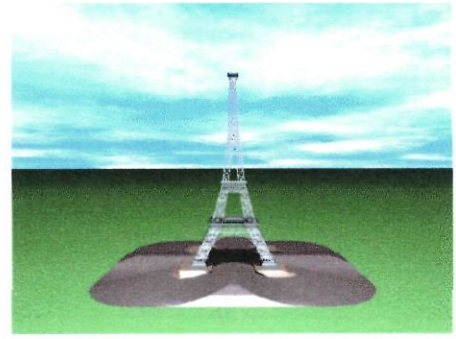


**Tetera**

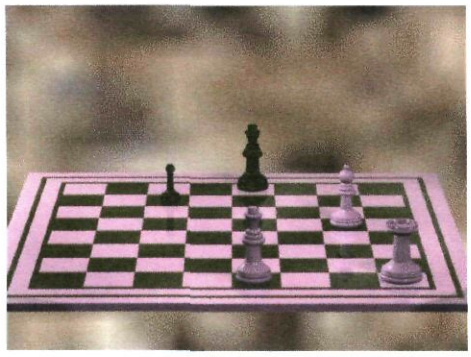
**Figura 5.3 Algunas de las escenas utilizadas para el análisis de los diferentes algoritmos.**



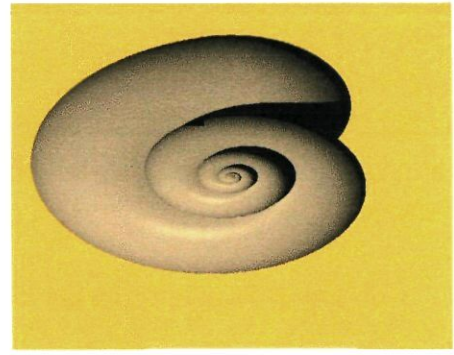
**Stetra**



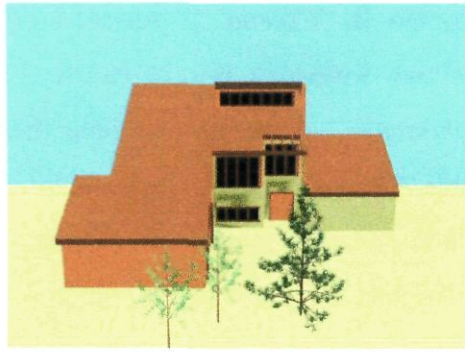
**Torre Eiffel**



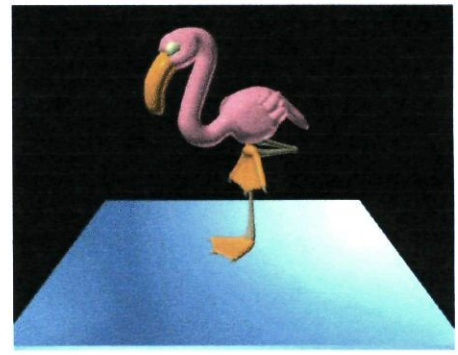
**Ajedrez**



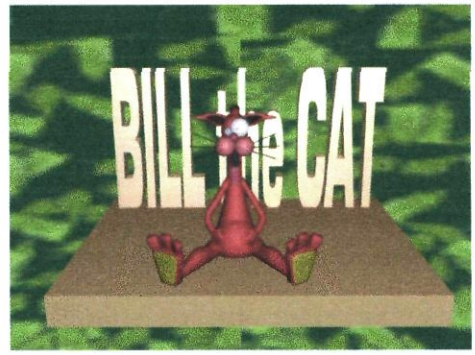
**Caracol**



**Casa de Campo**



**Flamenco**




**Gato-Bill**

**Figura 5.3 (cont.)** **Algunas de las escenas utilizadas para el análisis de los diferentes algoritmos.**

| | *Escena 1* | *Escena2* | *Delfines* | *Maceta* | *Habitación* | *Misil* | *Bola* | *Tetera* |
|---|---|---|---|---|---|---|---|---|
| ***Polig.*** | | 1 | 1.692 | 1.284 | 34.110 | 956 | 1.536 | 1.004 |
| ***Esfer.*** | 430 | 91 | | | | | | |

| | *Stetra* | *Torre Eiffell* | *Ajedrez* | *Caracol* | *Casa Campo* | *Flamenco* | *Gato-Bill* |
|---|---|---|---|---|---|---|---|
| ***Polig.*** | 256 | 16.648 | 19.734 | | 12.172 | 12.874 | 15.128 |
| ***Esfer.*** | | | | 5.671 | | | |

**Tabla 5.1 Descripción de las diferentes escenas utilizadas.**

Como podemos apreciar en las imágenes (ver figuras 5.1, 5.2 y 5.3) y en la descripción (ver tabla 5.1) de las escenas analizadas, el número de primitivas utilizadas para su definición y su distribución espacial es bastante diferente. Podemos encontrar escenas muy simples como la *"Escena 2"* formada por 92 objetos, junto a otras bastante más complejas formadas por cerca de 35.000 polígonos con es el caso de la escena *"Habitación"*. A su vez, también tenemos figuras definidas únicamente mediante esferas como es el caso de la denominada *"Caracol"*, que contiene 5.671, y otras que mezclan ambas primitivas como *"Escena 2"*. Por otra parte, encontramos algunas en las que los objetos están fuertemente concentrados y, por tanto, existen grandes zonas libres (*"Tetera"*, *"Torre Eiffel"*, etc.) u otras con una distribución más uniforme de los objetos por toda la escena (*"Maceta"*, *"Bola espacial"*, etc.)

Esta diversidad en las escenas utilizadas para evaluar los resultados del trazado de haces asigna mayor validez al análisis de su comportamiento. Aunque como en todas las evaluaciones realizadas sobre algoritmos de *rendering* es difícil, o tal vez imposible, asegurar que los casos de prueba utilizados recogen todos los casos posibles y, por tanto, que el comportamiento del algoritmo será siempre el observado en las pruebas realizadas.

## 5.3 Análisis del Trazado de Haces.

En este punto vamos a revisar las características más importantes del trazado de rayos. Analizaremos, por tanto, el comportamiento del algoritmo ante diferentes criterios de subdivisión del árbol octal y ante escenas con distinta distribución espacial de los objetos. Junto a esto, se comprobará qué ventajas ofrece el trazado de haces para optimizar los diferentes algoritmos propuestos hasta el momento con el objetivo de reducir el problema de aliasing. Por último, estudiaremos cuál es la tarea más costosa en tiempo, ya que ésta influirá decisivamente en el posterior diseño de una versión paralela del algoritmo.

### 5.3.1 Comportamiento del trazado de rayos ante diferentes criterios de subdivisión del árbol octal.

Como ya apuntamos en el estudio teórico del trazado de haces, uno de los problemas con los que nos encontramos al asociar estructuras de descomposición a las escenas, es definir un criterio de subdivisión que permita obtener estructuras eficientes. Aunque los distintos criterios están muy ligados a la distribución espacial y al número de objetos de las escenas a tratar, en este apartado pretendemos mostrar cómo reacciona el trazado de haces ante diferentes alternativas de descomposición. Podría considerarse este estudio como punto de partida de otro más amplio que pretenda establecer una formulación matemática que permita definir un criterio de subdivisión óptimo para una escena dada.

El estudio que vamos a realizar a continuación va a tomar como base el trabajo de McNeill et al. [MCNE92], en el que se analiza el comportamiento del trazado de rayos en árboles octales en función del nivel de subdivisión del árbol. Como ya indicamos, en este trabajo se concluye finalmente que a partir del sexto

nivel de subdivisión la probabilidad de que un rayo visite un nodo es muy baja y, por otra parte, las necesidades de memoria aumentan notablemente. Este trabajo permite pensar que los niveles de descomposición más eficientes podrían estar próximos al nivel seis ya que asegurarán que los nodos terminales serán visitados por un número suficientemente elevado de rayos y por tanto el tamaño de los haces en los que viajarán será mayor, lo que provocará un aumento en la rentabilidad del trazado de haces.

Para analizar el comportamiento del trazado de haces vamos a observar cómo evoluciona el tiempo total de trazado conforme aumentamos el nivel de subdivisión del árbol. Junto a ello, se estudiará cómo afecta la utilización del criterio "número máximo de objetos contenidos en un nodo" (*nmax*) dentro de la eficiencia del algoritmo. Para ello, se observarán los tiempos de trazado del algoritmo para diferentes valores del parámetro *nmax,* considerando que si se supera dicho valor, y no se ha alcanzado todavía el máximo nivel de subdivisión preestablecido, el nodo debe dividirse.

El hecho de que la composición de la escena pueda condicionar los resultados obtenidos, aconseja utilizar varias escenas dentro del estudio realizado. Por ello, se han seleccionado dos escenas un tanto diferentes, en la primera ("*Maceta*") los objetos están fuertemente distribuidos por toda la escena, en cambio en la segunda ("*Delfines*") los objetos se concentran en tres zonas claramente diferenciadas, asociadas cada una a un delfín. En este caso, se han realizado dos tipos de pruebas diferentes. Las primeras parten de establecer el máximo nivel de descomposición en seis y analizan el comportamiento del trazado de haces al aumentar el número máximo de objetos contenidos en un nodo. A continuación, en un segundo lugar, se han realizado diferentes pruebas aumentando progresivamente el nivel de descomposición comenzando con un nivel de cuatro hasta llegar a uno de siete.

En las figuras 5.4 y 5.5 pueden apreciarse los resultados de la aplicación de los diferentes criterios de descomposición sobre las dos escenas utilizadas para esta prueba ("*Maceta*" y "*Delfines*").

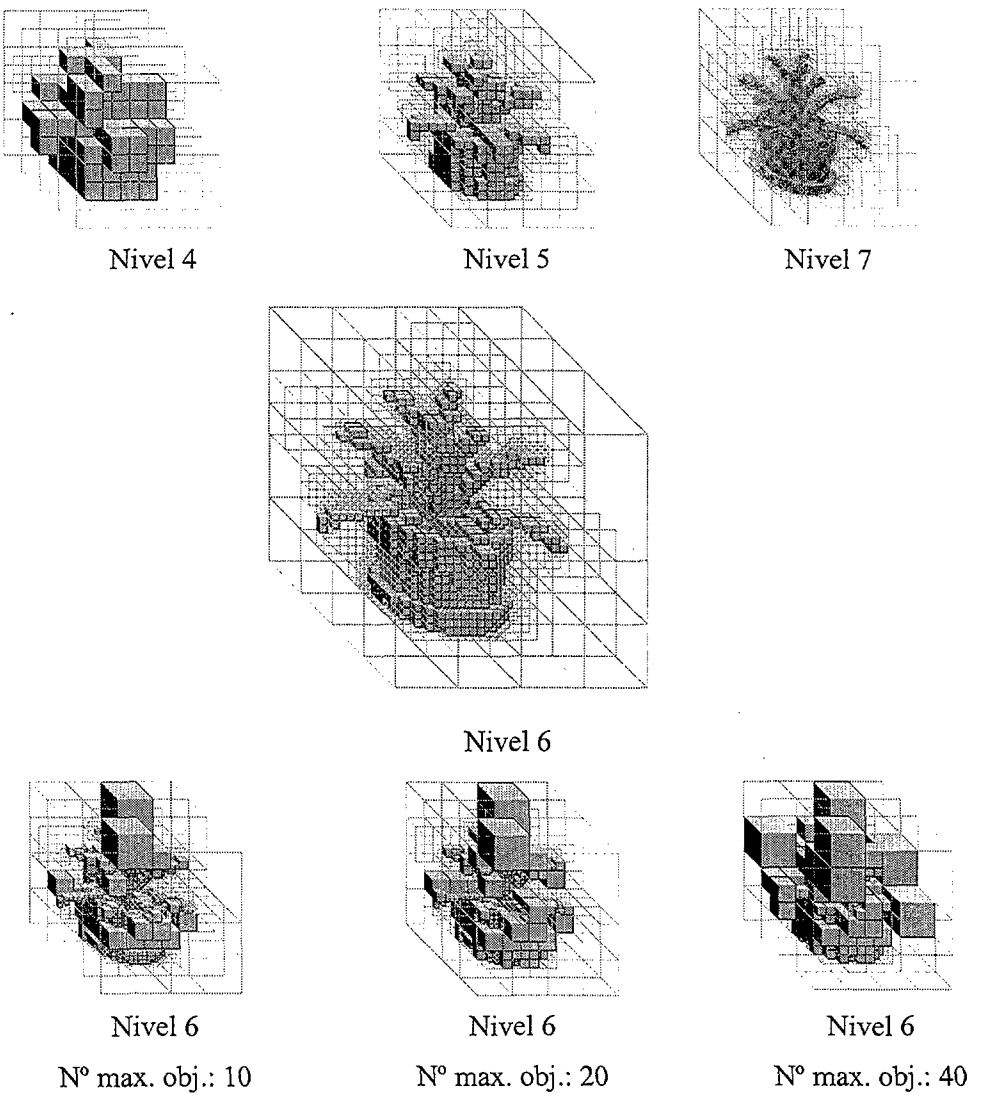
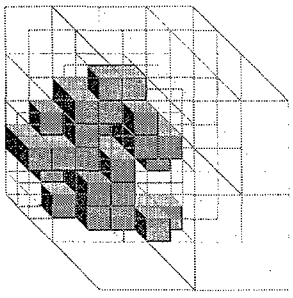
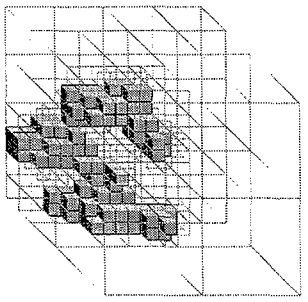



**Figura 5.4 Resultado obtenido tras utilizar diferentes criterios de descomposición sobre la escena "*Maceta*".**

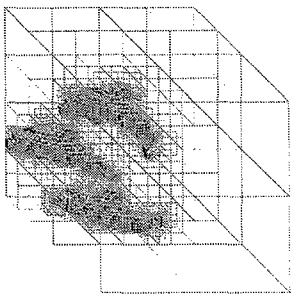
Primeramente analizaremos el comportamiento del trazado de haces al establecer un segundo parámetro en el proceso de descomposición (número máximo de objetos contenidos).



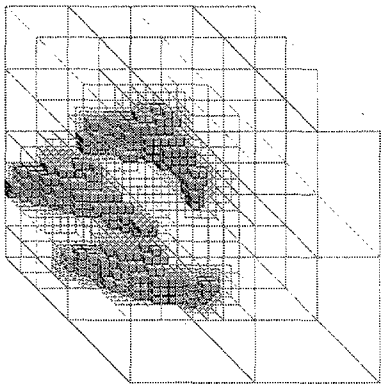
Nivel 4



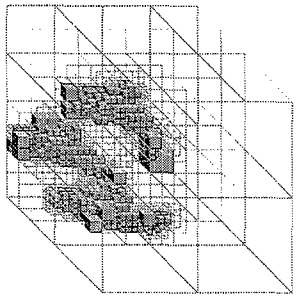
Nivel 5



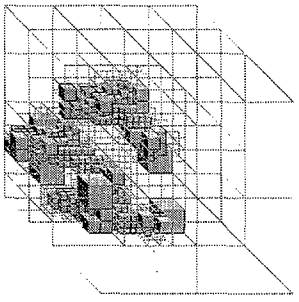
Nivel 7



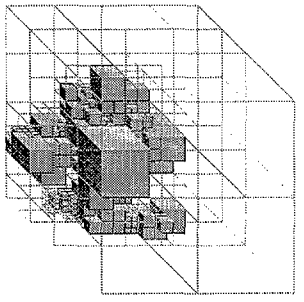
Nivel 6



Nivel 6
N° max. obj.: 10



Nivel 6
N° max. obj.: 20



Nivel 6
N° max. obj.: 40

**Figura 5.5 Resultado obtenido tras utilizar diferentes criterios de descomposición sobre la escena "*Delfines*".**

En la tabla 5.2 y en la figura 5.6 se muestran los valores del tiempo total de procesado para las diferentes pruebas realizadas. Como podemos apreciar el tiempo

aumenta al incrementar el número máximo de objetos que pueden estar contenidos en un nodo.

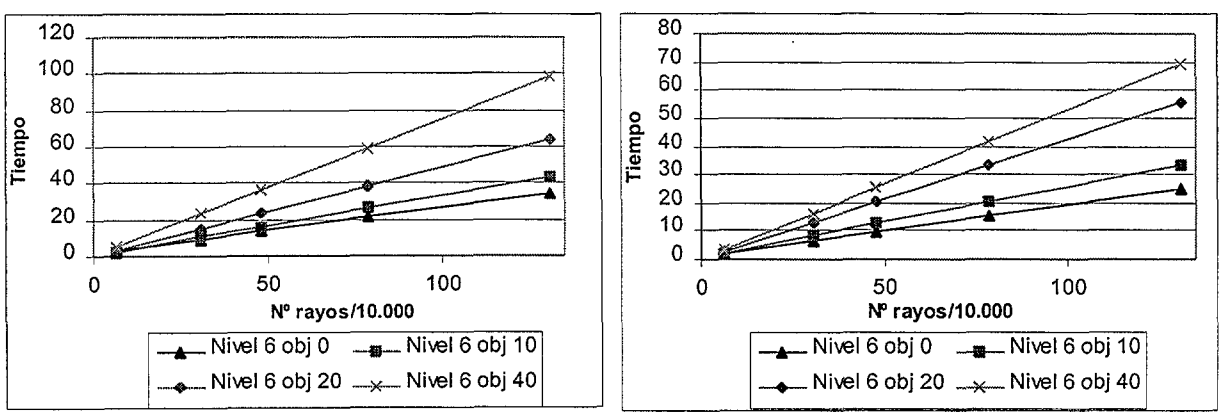
| | | | *Nº rayos / 10.000* | | | *Nº nodos ocupados* | *Pixel de haces que salen* |
|---|---|---|---|---|---|---|---|
| | *6,4* | *30,72* | *48* | *78,6432* | *131,072* | | *131,072* |
| ***Nivel 6 obj. 0*** | 3,25 | 9,29 | 13,63 | 21,18 | 34,45 | 1.529 | 212.142 |
| ***Nivel 6 obj. 10*** | 2,45 | 10,46 | 16,15 | 26,17 | 43,57 | 956 | 48. 319 |
| ***Nivel 6 obj. 20*** | 3,14 | 15 | 23,43 | 38,34 | 63,91 | 465 | 1.350 |
| ***Nivel 6 obj. 40*** | 4,85 | 23,18 | 36,25 | 59,33 | 98,84 | 214 | 108 |

**(a)**

| | | | *Nº Rayos / 10.000* | | | *Nº nodos ocupados* | *Pixel de haces que salen* |
|---|---|---|---|---|---|---|---|
| | *6,4* | *30,72* | *48* | *78,6432* | *131,072* | | *131,072* |
| ***Nivel 6 obj 0*** | 2,14 | 6,63 | 9,8 | 15,36 | 25,02 | 601 | 612.706 |
| ***Nivel 6 obj 10*** | 2 | 8,19 | 12,65 | 20,45 | 33,23 | 566 | 273.354 |
| ***Nivel 6 obj 20*** | 2,74 | 13,03 | 20,36 | 33,31 | 55,53 | 420 | 91.711 |
| ***Nivel 6 obj 40*** | 3,4 | 16,2 | 25,32 | 41,46 | 69,06 | 222 | 3.031 |

**(b)**

**Tabla 5.2 Resultados obtenidos al aumentar el número máximo de objetos contenidos en un nodo. a) *Maceta*; b) *Delfines***




**(a)** **(b)**

**Figura 5.6 Evolución del tiempo total de procesamiento al utilizar diferentes criterios de descomposición sobre las escenas "*Maceta*" y "*Delfines*".**

Este aumento del tiempo de procesado se debe a que el tamaño de los nodos que contiene objetos crece, lo que implica que el número y tamaño de los haces que llegan a nodos ocupados sea mayor. Este incremento del número y tamaño de los haces que llegan a un nodo ocupado puede comprobarse si observamos en la tabla 5.2 la evolución, para cada una de las estructuras, del número de pixel que contienen los haces que salen de la escena sin alcanzar ningún nodo ocupado (los valores que aparecen se corresponden con una resolución de 1280 x 1024). Como podemos apreciar, cada vez que aumenta el número de objetos se incrementa el número de nodos de mayor tamaño lo que hace más difícil que los haces salgan de la escena sin alcanzar ningún nodo ocupado. Esto provoca que se reduzca el número de pixel asociados a los haces que salen, haces que son responsables, en gran medida, de los beneficios que el trazado de haces ofrece. A su vez, el número de objetos contenidos en un nodo crece, ralentizando el proceso de determinación de la intersección rayo-objeto, al incrementarse el número de cálculos que se deben realizar en dicho nodo.

Del análisis anterior parece deducirse que cuanto menor sea el tamaño de los nodos ocupados del árbol mejores resultados podríamos obtener. De ser totalmente cierta esta suposición, y llevándola hasta sus últimas consecuencias, se debería asociar a cada escena un árbol con infinitos niveles de subdivisión para, de este modo, asegurar que el tamaño del nodo es el menor posible. Como puede suponerse esto no es así, pues junto a la conclusión anterior debemos tener en cuenta que tras cierto nivel de subdivisión los haces que viajan por la escena son tan pequeños que su coste de trazado es superior al coste que supondría el trazado individual de los rayos asociados a ellos.

Para analizar cómo influye el nivel de subdivisión dentro del algoritmo de trazado de haces se han realizado nuevas pruebas que permitan mostrar la evolución del tiempo de trazado. En la tabla 5.3 pueden verse los resultados obtenidos tanto en tiempo de procesado, como en número de nodos ocupados o pixel asociados a haces

que abandonan la escena sin pasar por ningún nodo ocupado. A su vez, la figura 5.7 muestra gráficamente la evolución de los tiempos de ejecución.

En este caso, el tiempo de procesado va disminuyendo conforme aumentamos el nivel de descomposición, pero llega un momento (nivel siete) en el cual dicho tiempo no sólo no disminuye sino que aumenta nuevamente, y en algunos casos, como sucede con la escena denominada "*Maceta*", el tiempo obtenido es superior al conseguido con un nivel cinco.

| | | | *Nº rayos / 10.000* | | | *Nº nodos ocupados* | *Pixel de haces que salen* |
|---|---|---|---|---|---|---|---|
| | *6,4* | *30,72* | *48* | *78,6432* | *131,072* | | *131,072* |
| ***Nivel 4*** | 5,46 | 26,42 | 41,31 | 67,6 | 112,72 | 97 | 28 |
| ***Nivel 5*** | 2,69 | 12,12 | 18,77 | 30,55 | 51,17 | 398 | 90.104 |
| ***Nivel 6*** | 3,25 | 9,29 | 13,63 | 21,18 | 34,45 | 1.529 | 212.142 |
| ***Nivel 7*** | 39,6 | 43,74 | 46,91 | 53,34 | 62,66 | 6.287 | 342.544 |

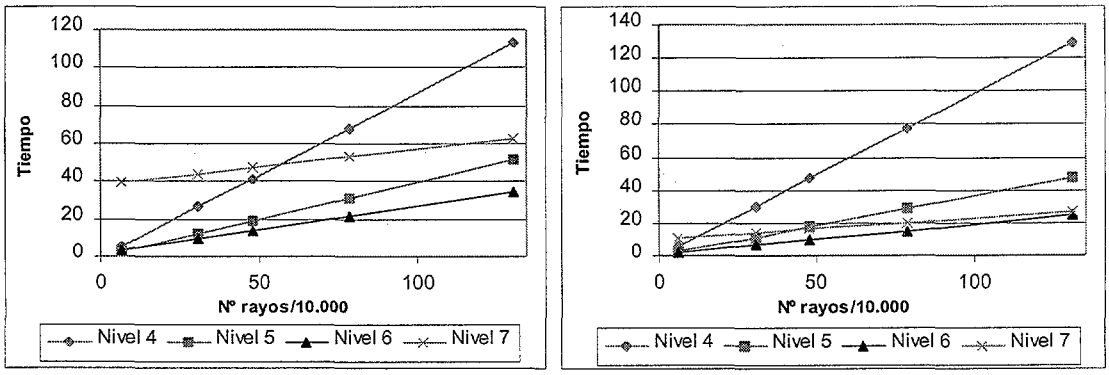
**(a)**

| | | | *Nº rayos / 10.000* | | | *Nº nodos ocupados* | *Pixel de haces que salen* |
|---|---|---|---|---|---|---|---|
| | *6,4* | *30,72* | *48* | *78,6432* | *131,072* | | *131,072* |
| ***Nivel 4*** | 6,34 | 30,28 | 47,4 | 77,56 | 129,23 | 38 | 67. 973 |
| ***Nivel 5*** | 2,52 | 11,38 | 17,68 | 28,85 | 47,85 | 144 | 327.777 |
| ***Nivel 6*** | 2,14 | 6,63 | 9,8 | 15,36 | 25,02 | 601 | 612.706 |
| ***Nivel 7*** | 11,28 | 14,1 | 16,4 | 20,3 | 27,05 | 2.385 | 786.570 |

**(b)**

**Tabla 5.3 Resultados obtenidos al aumentar el nivel máximo de descomposición. a) *Maceta*; b) *Delfines***

Si analizamos más detenidamente los resultados obtenidos para el nivel siete (figura 5.7) vemos que la pendiente de la recta que representa la evolución del tiempo de trazado es menor que la pendiente del resto de rectas correspondientes a la evolución del tiempo de procesado en los otros niveles. Esto nos lleva a pensar que el problema que nos estamos encontrando es que al aumentar el nivel de descomposición el tamaño de los haces disminuye y sólo pueden considerarse rentable su utilización cuando se incrementa la resolución de la imagen.




**(a)** **(b)**

**Figura 5.7 Evolución del tiempo total de procesamiento al utilizar diferentes niveles de descomposición sobre las escenas "*Maceta*" y "*Delfines*".**

Por otra parte, vemos que la evolución de los tiempos de trazado no es igual en las dos escenas analizadas. Este comportamiento diferente se debe a que en escenas con una mayor distribución de objetos, el tamaño de los haces se reduce más rápidamente que en el otro tipo de escenas y, por tanto, es más difícil que un haz alcance la salida de la escena sin antes pasar por un nodo ocupado. Esta afirmación anterior puede comprobarse si observamos, para las dos escenas, la evolución del número de pixel contenidos en haces que salen de la escena sin encontrar en su camino ningún nodo ocupado. Como podemos apreciar en la tabla 5.3 el número de pixel es mucho mayor en la escena denominada "*Delfines*" que en la denominada "*Maceta*" (con mayor distribución de objetos y menor numero de espacios libres).

Por ello, la cantidad de rayos a trazar junto con el número y la distribución de los objetos en la escena serán dos parámetros muy importantes para determinar el máximo nivel de descomposición que permite obtener los mayores beneficios en la utilización del trazado de haces. En nuestro caso, todas las escenas analizadas tienen un buen comportamiento considerando como máximo nivel de descomposición el nivel seis, por ello, todas las pruebas que vamos a realizar a continuación tienen en cuenta dicho criterio de subdivisión.

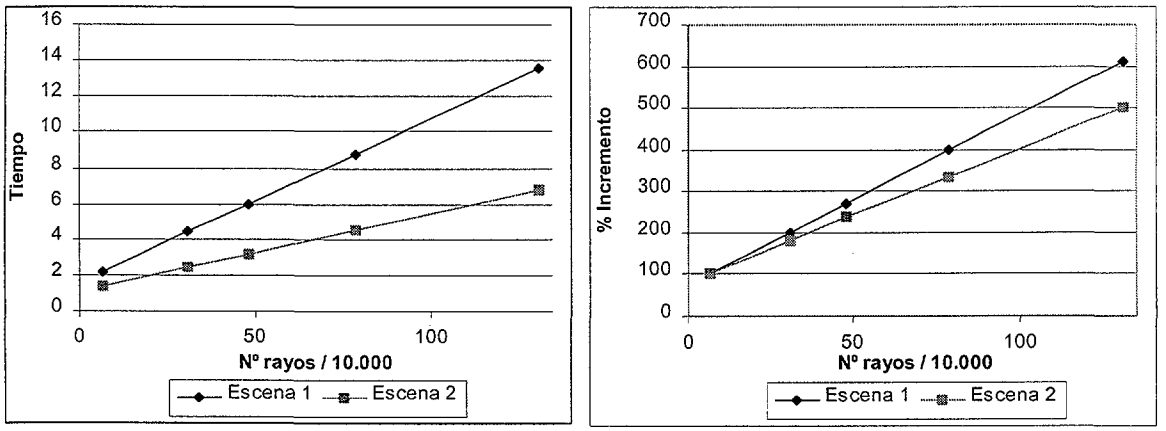
### 5.3.2 Análisis del comportamiento del trazado de haces ante el aumento de la calidad de la imagen final.

Una vez analizado el efecto que la estructura de descomposición tiene sobre el trazado de haces vamos a comprobar cómo afecta el aumento en la calidad de la imagen obtenida, en el tiempo de procesado. A su vez, vamos estudiar cuál es su comportamiento ante escenas con diferente distribución espacial.

Primeramente nos centraremos en estudiar el efecto que el aumento de la calidad de la imagen tiene sobre el tiempo de procesado. Dicha calidad va ligada a dos ideas: generar imágenes con una alta resolución y aplicar técnicas que reduzcan los problemas de aliasing que todo proceso de muestreo tiene asociados. Para ello, realizaremos dos tipos de pruebas: incrementar la resolución de la imagen (desde 320x200 hasta 1280x1024) y analizar el efecto de aplicar sobremuestreo uniforme sobre la imagen de máxima resolución.

Centrémonos en analizar el comportamiento de las denominadas escenas "*sintéticas*" ("*Escena 1*" y "*Escena 2*"). En la figura 5.8, podemos apreciar la evolución del tiempo de trazado. Como era de esperar dicha evolución (figura 5.8.a) es lineal, incrementándose conforme aumenta la resolución de la imagen. Pero, como podemos apreciar, dicho incremento no sólo depende de la resolución de la imagen, sino que juega un papel muy importante la distribución de los objetos dentro de la escena. En la figura 5.8.b podemos ver el porcentaje de incremento del tiempo

total de procesado tomando como base el tiempo obtenido para una resolución de 320x200. Con este estudio pretendemos aislar, en cierta medida, el efecto que la complejidad de la escena provoca en el tiempo final de ejecución. En este caso vemos que, tanto a nivel absoluto como relativo, el incremento del tiempo es mayor para la *"Escena 1"* que para la *"Escena 2"*. Este mayor incremento se debe básicamente a la mayor complejidad de la *"Escena 1"*, en la cual los objetos están distribuidos de manera aleatoria dentro del espacio de la escena, lo que reduce en cierta medida la eficiencia del trazado de haces, ya que es más difícil que los rayos recorran gran parte de la escena contenidos dentro de algún haz.




**(a** **(b)**

**Figura 5.8 Evolución del tiempo de trazado en *"Escena 1"* y *"Escena 2"*; (a) tiempo total de procesado; b) % de incremento del tiempo con respecto al tiempo utilizado por la resolución 320x200.**

Veamos ahora cuál es el comportamiento de algunas de las escenas denominadas *"reales"*. En este caso vamos a seleccionar cuatro escenas con una clara diferencia en cuanto a distribución de los objetos en la escena: *"Bola espacial"*, *"Caracol"*, *"Tetera "* y *"Torre Eiffel""*. Si analizamos el árbol octal al que dan lugar cada una de ellas (figura 5.9) podemos ver que las dos últimas tienen grandes espacios libres de objetos por los que podrán propagarse los haces sin dificultad, en cambio las otras dos escenas casi no dejan espacio libres y, por tanto,

es difícil conseguir que los haces puedan salir de la escena sin pasar previamente por un nodo ocupado.

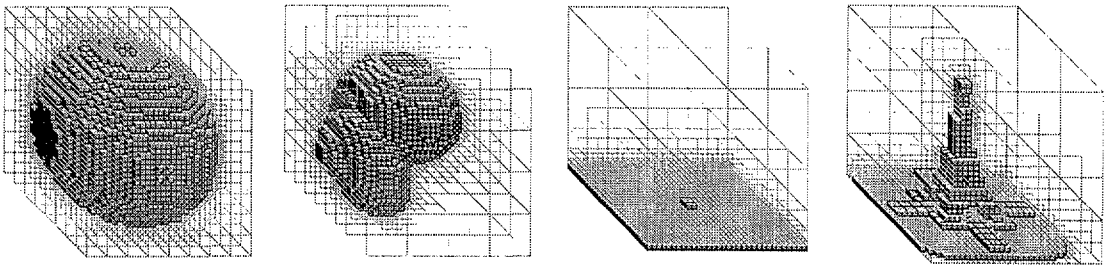
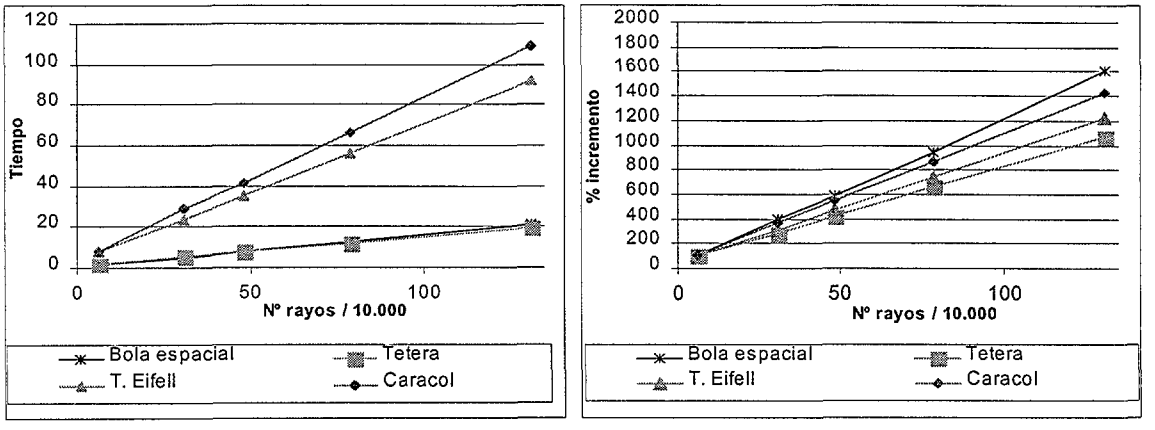



**Figura 5.9 Estructura de descomposición de las escenas "*Bola espacial*", "*Caracol*", "*Tetera*" y "*Torre Eiffel*".**

Estas diferencias en cuanto a la estructura de descomposición a la que dan lugar, tendrán un fiel reflejo en los resultados del trazado de haces. Como se muestra en la figura 5.10.a, el tiempo de procesado depende fuertemente de la complejidad de la escena (número y distribución espacial).




**(a)** **(b)**

**Figura 5.10 Evolución del tiempo de trazado en las escenas "*Bola espacial*", "*Tetera*", "*T. Eiffel*" y "*Caracol*"; (a) tiempo total de procesado; b) % de incremento del tiempo con respecto al tiempo utilizado por la resolución 320x200.**

Si analizamos cómo se incrementa dicho tiempo conforme aumentamos la resolución los resultados son tal vez más interesantes. En la figura 5.10.b, se muestra la evolución del incremento del tiempo conforme aumentamos la resolución, tomando como base el valor obtenido al generar una imagen de 320x200. Si observamos los resultados de esta gráfica, podemos ver que la recta con mayor pendiente se corresponde con la de la escena denominada "*Bola espacial*", escena con un valor bajo en cuanto a tiempo total. Junto a la escena anterior, la denominada "*Caracol*" que en valor absoluto supone el mayor tiempo de procesado, vemos que también provoca un gran incremento relativo. La causa de que las dos escenas anteriormente mencionadas sean las que tengan un mayor incremento relativo es la estructura de descomposición a la que dan lugar, estructura que, como ya hemos indicado, deja pocos nodos libres y, por tanto, es poco probable que los haces puedan salir de la escena sin pasar por un nodo ocupado. Esto provoca que el número y tamaño de los haces que llegan a nodos ocupados sea mayor y, por tanto, el número de intersecciones rayo-objeto a calcular crece más rápidamente conforme se incrementa la resolución (número de pixel) y dicho efecto se deja notar claramente en el tiempo final de procesado.

Analicemos ahora el efecto de utilizar técnicas de antialiasing sobre las escenas anteriores. Las técnicas de antialiasing se basan en la utilización, para cada pixel, de varias muestras en vez de una única. Este sobremuestreo implica el trazado de un mayor número de rayos, lo que lleva asociado un coste añadido que requiere de propuestas que permitan reducirlo sin por ello perder las ventajas esperadas en cuanto a calidad de la imagen final. Como ya apuntamos en los primeros capítulos, la mayoría de las técnicas de sobremuestreo tienen como base el "espacio de la imagen" y, por tanto, no aprovechan información que está accesible en el propio instante de trazado. En nuestro caso, se propone utilizar otra información adicional que en el trazado de haces puede estar accesible en el instante de la determinación y generación de los rayos correspondientes a un pixel concreto, pudiendo ser considerada esta propuesta como "basada en el espacio de los objetos" [Gene98].

En el trazado de haces los rayos se generan cuando un haz alcanza un nodo ocupado o cuando éste sale de los límites de la escena, por tanto, en el instante de generación de los diferentes rayos se dispone de información adicional: número de objetos contenidos y distancia a la que se encuentran. En este caso, en las pruebas realizadas tan sólo hemos aprovechado la información de "*número de objetos contenidos*". Con esta información somos capaces de ahorrarnos el trazado de rayos adicionales para aquellas zonas que no contienen objetos y que, por tanto, han sido atravesadas por haces que finalmente salen de la escena. A su vez, al llegar a un nodo ocupado, podríamos establecer un número diferente de rayos en función de los objetos contenidos dentro de dicho nodo.

En las pruebas realizadas tan solo hemos utilizado la técnica de sobremuestreo uniforme, que es la que implementa la librería que hemos tomado como base (OORT). En este caso hemos supuesto que por cada pixel se lanzarán cinco rayos, cuatro asociados a las esquinas del pixel y uno al centro. En la implementación realizada tan solo se generará un número diferente de rayos si el haz sale de la escena (cero rayos en vez de cinco) pues estamos seguros que en este caso el color del pixel asociado será uniforme.

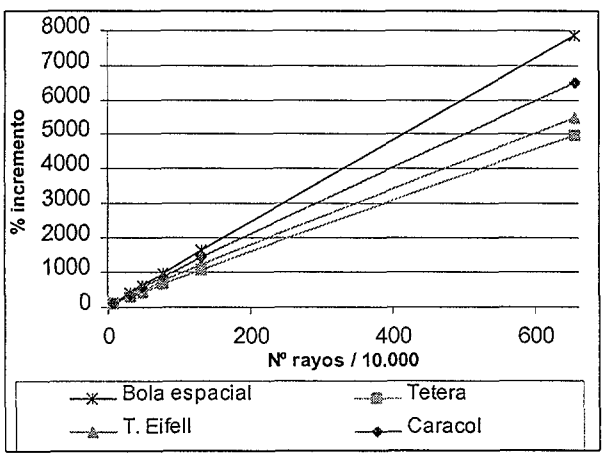



**Figura 5.11 Evolución del % de incremento del tiempo al introducir el tratamiento del antialiasing en las escenas *"Bola espacial", "Tetera", "T. Eiffel"* y *"Caracol".***

En la figura 5.11 se muestra la evolución del incremento del tiempo tomando como base el tiempo necesario para obtener una imagen de 320x200 pixel. Como podemos apreciar las rectas con menores pendientes se corresponden a las escenas con una mayor espacio libre, ya que son las que contiene un mayor número de haces que salen de la escena sin pasar por ningún nodo ocupado. A su vez podemos ver si comparamos la figura 5.11 con la 5.10.b que la separación entre las rectas se agudiza, de tal modo que las escenas con mayor espacio libre tienen una ganancia proporcionalmente mayor.

En cualquier caso, aunque en la implementación de la versión actual del trazado de haces tan sólo se ha incluido el criterio que permite generar diferente número de rayos en función del tipo de haz (haz que sale o haz que llega a un nodo ocupado) al que pertenece el pixel, como ya hemos apuntado en el instante de generación disponemos de más información que en futuras versiones del trazado de haces podrá ser utilizada para mejorar aun más este proceso.

### 5.3.3 Determinación de las tareas que más afectan al tiempo total de procesado.

Para finalizar el análisis del trazado de haces, vamos a estudiar el comportamiento de las diferentes tareas que éste conlleva con el objetivo de determinar cuál de ellas es la que influye más en el tiempo total de procesado y la que se ve afectada en mayor medida por el aumento en la calidad de la imagen final. Este estudio servirá de base para el diseño de la versión paralela que propondremos en el siguiente capítulo.

Si analizamos el algoritmo propuesto en el capítulo 3, para realizar el trazado de haces, podemos encontrar dos grandes tareas: (1) trazar un haz por la estructura de árbol octal; (2) determinar el color final de un pixel. La primera se realiza

siempre que el haz llega a un nodo vacío y la segunda se realizará cuando se llega a un nodo ocupado o cuando el haz sale de la escena.

El trazado de un haz dentro del árbol octal implica básicamente la determinación de las direcciones de salida y la generación de los haces de salida en cada una de dichas direcciones. Estas tareas deberán realizarse siempre que un haz avanza por la estructura de árbol y, por tanto, estará asociada a haces no terminales. Esta tarea podría suponer un tiempo de procesado alto si el nivel de descomposición del árbol fuese elevado ya que dicha subdivisión implicaría un incremento sustancial en el número de haces no terminales. En términos generales el tiempo de procesado de uno de estos haces viene dado por la siguiente expresión:

$$n_a * I_{r\text{-}p} + C_d + n_d * [R_{pol\text{-}cuad} + B_v + (n_v * R_{pol\text{-}cuad})]$$

donde

$n_a$ es el número de aristas del haz.

$I_{r\text{-}p}$ tiempo de cálculo de la intersección entre una recta y un plano.

$C_d$ tiempo de cálculo de las direcciones de salida.

$n_d$ número de direcciones de salida.

$R_{pol\text{-}cuad}$ tiempo de recorte de un polígono respecto a un cuadrado.

$B_v$ tiempo asociado a la búsqueda de vecinos en una determinada dirección.

$n_v$ número de vecinos.

Como vemos, el tratamiento a realizar es complejo y, por tanto, si el número de haces que viajan por la red sin encontrar nodos ocupados es alto, puede provocar una pérdida de eficiencia en el trazado de haces. Este es el motivo de que una excesiva subdivisión no resulte tan ventajosa como cabría esperar y, por tanto, una de las justificaciones de que debamos establecer un nivel máximo de subdivisión a la hora de estructurar la escena.

Así, en la escena denominada "*Maceta*" con un nivel de subdivisión 6 se generan 8.701 haces de los cuales 4.071 son no terminales, mientras que en la descomposición de nivel 7 se generan 37.718 haces de los cuales 17.751 son no terminales. En este caso el tiempo de procesado total aumenta considerablemente (para resoluciones de 320x200 el tiempo pasa de 3'25 a 39'6) convirtiéndose el trazado de estos haces en la tarea crítica (para 320x200 pasa de suponer el 46% del tiempo total para llegar hasta un 96%). Pero hay que tener en cuenta que lo anterior se ha producido por partir de una estructura de descomposición no adaptada correctamente y además que el tiempo de trazado de los haces no terminales no depende de la resolución y, por tanto, es de suponer que para resoluciones altas dicha diferencia se reduzca (en la resolución de 1280x1024 el trazado de haces no terminales, utilizando una estructura de nivel 7, supone ahora el 60'8% del tiempo total de procesado).

El nivel de descomposición utilizado afecta, por tanto, decisivamente a la hora de determinar la tarea más relevante en cuanto a tiempo consumido. Pero si partimos de una estructura de descomposición adecuada, como sucede en las diferentes pruebas realizadas, podemos pensar, en base a dichas pruebas, que el tiempo que representa el trazado de los haces con respecto al tiempo total es siempre sensiblemente menor que el que representa la tarea asociada a los haces terminales. Por otra parte, la tarea de trazado en este caso no depende del número de rayos necesarios para obtener la imagen final, sino que se mantiene constante independientemente de la resolución aplicada o la utilización de técnicas de sobremuestreo. Esto hace que en algunos casos, como veremos a continuación al compararlo con otras técnicas de trazado individual, el tiempo de procesado del trazado de haces sea superior al de las técnicas de trazado de rayos individuales para resoluciones bajas (320x200) y sin embargo, obtenga excelentes resultados comparados con los mismos algoritmos cuando la resolución aumenta.

El tiempo asociado a los haces terminales depende fuertemente de que el haz abandone la escena o haya llegado a un nodo ocupado. En el primer caso, la única

tarea a realizar es determinar los pixel contenidos en un haz y asociarles el color del fondo. Esta tarea, como vimos en el tercer capítulo, implica la proyección del haz sobre la pantalla y la aplicación del algoritmo de rellenado de polígonos mediante líneas de rastreo. Por lo general el tiempo utilizado para procesar haces terminales que salen de la escena puede considerarse pequeño y, por tanto, nunca se convertirá en la tarea crítica.

En cambio el segundo supuesto, es decir, cuando un haz llega a un nodo ocupado, lleva implícito un tratamiento más complejo. Este proceso puede expresarse mediante la siguiente expresión:

$$B_{pixel} + (n_{pixel} * n_{rayos\text{-}pixel}) * [(I_{rayo\text{-}objeto} * n_{obj\text{-}nodo}) + (n_{rayos\text{-}noFin} * T_{indiv})]$$

donde

$B_{pixel}$ tiempo asociado a la búsqueda de los pixel contenidos en el haz.

$n_{pixel}$ número de pixel contenidos dentro del haz.

$n_{rayos\text{-}pixel}$ número de rayos generados por pixel.

$I_{rayo\text{-}objeto}$ tiempo asociado al cálculo de la intersección rayo–objeto.

$n_{obj\text{-}nodo}$ número de objetos contenidos en un nodo.

$n_{rayos\text{-}noFin}$ número de rayos que no chocan con ningún objeto contenido en el nodo.

$T_{indiv}$ tiempo asociado al trazado individual de un rayo.

Como podemos apreciar, dentro de la expresión anterior se incluye el tiempo, generalmente nada despreciable, asociado a los cálculos intersección rayo-objeto. A su vez, en el caso de que un rayo no choque con ninguno de los objetos contenidos en dicho nodo se realiza un trazado individual de dicho rayo (utilizando el algoritmo propuesto por H. Samet [SAME89]) hasta que finalmente se determina su color. A su vez, como puede apreciarse en la expresión anterior, el tiempo que se necesitará para realizar esta tarea depende fuertemente del número de rayos necesarios para generar la imagen final. En las pruebas realizadas anteriormente, esta tarea, es la

causante de la pendiente observada en la recta que representa el tiempo de procesado al incrementar la resolución. Por ello generalmente si deseamos encontrar la tarea crítica del algoritmo de trazado de haces que aquí se propone debemos dirigir nuestra mirada a la tarea encargada de procesar haces que llegan a nodos ocupados.

En la mayoría de las escenas analizadas, utilizando una resolución de 1280x1024, el tiempo asociado a dicha tarea representa generalmente más del 85% del tiempo total de procesado, llegando al 98% en escenas como la denominada "*Bola espacial*" en las que la proporción de haces terminales que llegan a un nodo ocupado respecto al total de haces (terminales y no terminales) es de un 63% aproximadamente.

## 5.4 Comparación con otras alternativas de aceleración basada en el trazado individual de cada rayo.

En este apartado nos dedicaremos a analizar el comportamiento del trazado de haces comparándolo con otras alternativas de aceleración que utilizan un trazado individual de los diferentes rayos que permitirán generar la imagen final. Dentro de este apartado se estudiarán dos situaciones un tanto diferentes. En primer lugar, se comprobará cuál es el comportamiento del trazado de haces con respecto a otras alternativas de aceleración que utilizan también árboles octales como estructura de descomposición. Tras esto, se estudiará su comportamiento con respecto a otros algoritmos que utilizan otras estructuras de descomposición, tanto basadas en descomposición *guiada por los objetos* (jerarquías de volúmenes envolventes), como *guiadas por el espacio* (descomposición uniforme). Con este último estudio pretendemos mostrar las ventajas que el trazado de haces aporta incluso en el

supuesto de que la estructura de árbol octal no sea la más indicada para descomponer una escena.

Para seleccionar los algoritmos basados en subdivisión *guiada por el espacio*, que se utilizarán en las pruebas, nos hemos apoyado en el trabajo de R. Endl y M. Sommer [ENDL94], en el cual se lleva a cabo un estudio minucioso de numerosas propuestas realizadas por diferentes autores sobre trazado de rayos utilizando tanto estructuras de descomposición uniforme como no uniforme, basadas en este último caso en árboles octales. Dentro de este trabajo se muestra una primera clasificación de los diferentes algoritmos y posteriormente se realiza una comprobación de la eficiencia de dichas alternativas.

En este caso, las diferentes pruebas se realizan sobre una estructura equivalente, es decir, si unas técnicas utilizan un árbol octal de nivel 5 los algoritmos que emplean matrices también realizan una descomposición hasta dicho nivel ($2^5$ x $2^5$ x $2^5$). Tras las pruebas realizadas puede verse que es difícil establecer a priori cuál es la mejor forma de descomponer el espacio y, por tanto, qué algoritmo utilizar para realizar el trazado, pues existe una gran dependencia de la escena utilizada. En cualquier caso, puede comprobarse que dentro de los algoritmos basados en árboles octales la mejor alternativa es la que se denomina "*Samet-Net*" [ENDL94] (optimización del algoritmo inicial propuesto por H. Samet [SAME89]) y dentro de las alternativas que se basan en descomposición uniforme la propuesta por Amanatides y Woo [AMAN87]. Por ello en las pruebas posteriores serán estas las alternativas que utilizaremos para analizar la eficiencia del trazado de haces.

Junto a los algoritmos anteriores que se apoyan en subdivisión guiada por el espacio, también se analizará el comportamiento del trazado de haces con respecto a otras alternativas que se encuadran dentro de las técnicas de descomposición denominadas *guiadas por los objetos*. En este caso, se utiliza la implementación que la librería OORT [WILT94] incluye como soporte de las jerarquías de volúmenes envolventes. Como ya indicamos en el capítulo anterior, esta implementación se

basa en los algoritmos propuestos por Goldsmith y Salmon [GOLD87] (utilizado para generar la estructura de subdivisión) y Kay y Kajiya [KAY86] (válido para trazar los rayos primarios dentro de la jerarquía previamente formada).

En las pruebas que vamos a realizar a continuación, los tiempos de procesado incluyen también los tiempos de generación de la estructura de descomposición. Esto se debe básicamente a que, para optimizar el algoritmo de Amanatides y Woo [AMAN87], la implementación que se ha llevado a cabo [MOLI98] realiza una construcción dinámica de la matriz de voxel de tal modo que aquellos voxel no visitados por ningún rayo no sean evaluados. Con esta optimización se obtienen mejoras en cuanto al espacio necesario para almacenar la matriz y en cuanto al tiempo requerido para construir la estructura, ya que no hace falta evaluar los tests de inclusión objeto-voxel para los voxel que no son visitados por ningún rayo. Por todo ello, aunque los demás algoritmos realizan un preprocesado que les permite generar la estructura, dicho tiempo debe incluirse para que puedan ser comparables los resultados obtenidos.

### 5.4.1 Comparación con otras técnicas de trazado individual sobre árboles octales.

Tal vez éste sea el estudio más importante pues una de las ideas básicas del trazado de haces es demostrar las ventajas del trazado simultáneo de varios rayos, con respecto a las técnicas de trazado individual. Por tanto, es fundamental comprobar si esta propuesta se comporta mejor que otras previas que realizan un trazado individual sobre el mismo tipo de estructura.

Como hemos indicado anteriormente, los algoritmos que utilizaremos para comprobar la eficiencia del trazado de haces son el algoritmo de trazado de rayos sobre árboles octales desarrollado por H. Samet [SAME89] y una modificación a éste propuesta posteriormente por Endl y Sommer [ENDL94]. A la primera técnica

la denominaremos durante este estudio *Samet* y a la segunda *Samet-Net.* Al igual que sucede en el trabajo de Endl y Sommer, las pruebas se realizan utilizando como base la misma estructura de descomposición, es decir, todos los algoritmos deben recorrer el mismo árbol octal. Por otra parte, las pruebas se llevarán a cabo aumentando la resolución de la imagen de tal modo que podamos apreciar la evolución de las diferentes alternativas.

En este caso, al igual que sucederá en las pruebas posteriores, primero se analizará el comportamiento de los distintos algoritmos sobre las escenas "*sintéticas*" y posteriormente se comprobará su eficiencia utilizando como base algunas de las escenas "*reales*" que aparecen en la figura 5.3.

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 2'21 | 4'44 | 5'98 | 8'81 | 13'52 |
| ***Samet*** | 2'26 | 10'79 | 16'82 | 27'53 | 45'87 |
| ***Samet-Net*** | 2'12 | 10'19 | 15'84 | 25'92 | 43'3 |

**(a)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 1'35 | 2'44 | 3'2 | 4'51 | 6'78 |
| ***Samet*** | 1'32 | 6'4 | 10 | 16'42 | 25'08 |
| ***Samet-Net*** | 1'26 | 6'3 | 9'8 | 16 | 24'6 |

**(b)**

**Tabla 5.4 Resultados del tiempo total de procesamiento. a) *Escena 1*; b) *Escena 2.***

Vayamos, por tanto, a estudiar qué sucede con las dos primeras escenas (*Escena1* figura 5.1 y *Escena2* figura 5.2). En la tabla 5.4 y en la figura 5.11 puede apreciarse el comportamiento de los diferentes algoritmos para estas escenas. Como

era de esperar, en función del estudio realizado por Endl y Sommer [ENDL94], el algoritmo denominado *Samet-Net* tiene un mejor comportamiento que el denominado *Samet*, y dicha mejora aumenta conforme lo hace la resolución de la imagen. Esto es debido a que el número de rayos que se aprovechan de las ventajas de los enlaces a los vecinos es mayor.

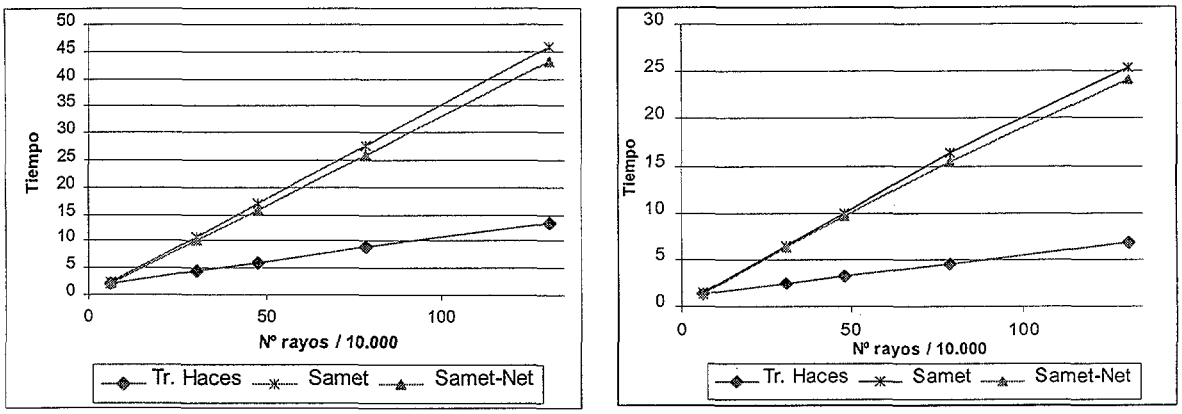



**Figura 5.12 Evolución del tiempo de trazado en las escenas: (a) *Escena 1*; b) *Escena 2*.**

Aunque tal vez lo que más llama la atención y, por otra parte, más interesa dentro de este estudio es el excelente comportamiento que tiene el trazado de haces si lo comparamos con los otros dos métodos. En este caso, para pequeñas resoluciones en las que los haces contienen muy pocos rayos es posible que los otros métodos tengan un comportamiento ligeramente mejor (como sucede a una resolución de 320x200). Pero en cuanto incrementamos la resolución y, por tanto la calidad de la imagen, las ventajas del trazado de haces quedan patentes. Podemos observar que dicha tendencia se mantiene (en la figura 5.12 se puede ver que la pendiente de la recta que representa la evolución del trazado de haces es mucho menor que la de las otras alternativas), por lo que es de esperar que conforme se desee mejorar la calidad de la imagen las ventajas del trazado de haces con respecto a otras alternativas de trazado aumente. Por otra parte, en este caso, el comportamiento del trazado de haces es semejante en las dos escenas, lo que hace

pensar, si partimos de la gran diferencia existente entre ambas escenas, que dicho comportamiento no se ve muy afectado por la escena que estemos representando y, por tanto, su aplicación se amplía.

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 8'61 | 38'55 | 59'69 | 97'24 | 161'88 |
| ***Samet*** | 9'09 | 43'87 | 68'25 | 112'23 | 186'86 |
| ***Samet-Net*** | 8'88 | 42'94 | 66'96 | 109'88 | 182'84 |

**(a)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 10'8 | 18'87 | 24'36 | 34'08 | 50'82 |
| ***Samet*** | 4'17 | 19'79 | 30'95 | 50'63 | 84'37 |
| ***Samet-Net*** | 4'06 | 19'25 | 30'11 | 49'2 | 82'18 |

**(b)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 1'81 | 7'08 | 10'8 | 18 | 31'6 |
| ***Samet*** | 2'29 | 11 | 18 | 28'4 | 49 |
| ***Samet-Net*** | 2'19 | 10'73 | 16'71 | 27 | 45'92 |

**(c)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 1'26 | 2'91 | 4'1 | 6'15 | 9'71 |
| ***Samet*** | 1'13 | 5'34 | 8'35 | 13'63 | 22'67 |
| ***Samet-Net*** | 1'07 | 4'95 | 7'77 | 12'69 | 21'39 |

**(d)**

**Tabla 5.5 Resultados del tiempo total de procesamiento. a) *Casa de Campo*; b) *Gato Bill*; c) *Misil* ; d) *Stetra* .**

Para reafirmar todavía más las conclusiones a las que hemos llegado anteriormente, vamos a analizar el comportamiento de las diferentes alternativas al procesar algunas de las escenas denominadas "*reales*". Aunque los resultados obtenidos para las diferentes escenas analizadas son muy similares, para acompañar este estudio se han seleccionado cuatro escenas: "*Casa de campo*", "*Gato Bill*", "*Misil*" y "*Stetra*".

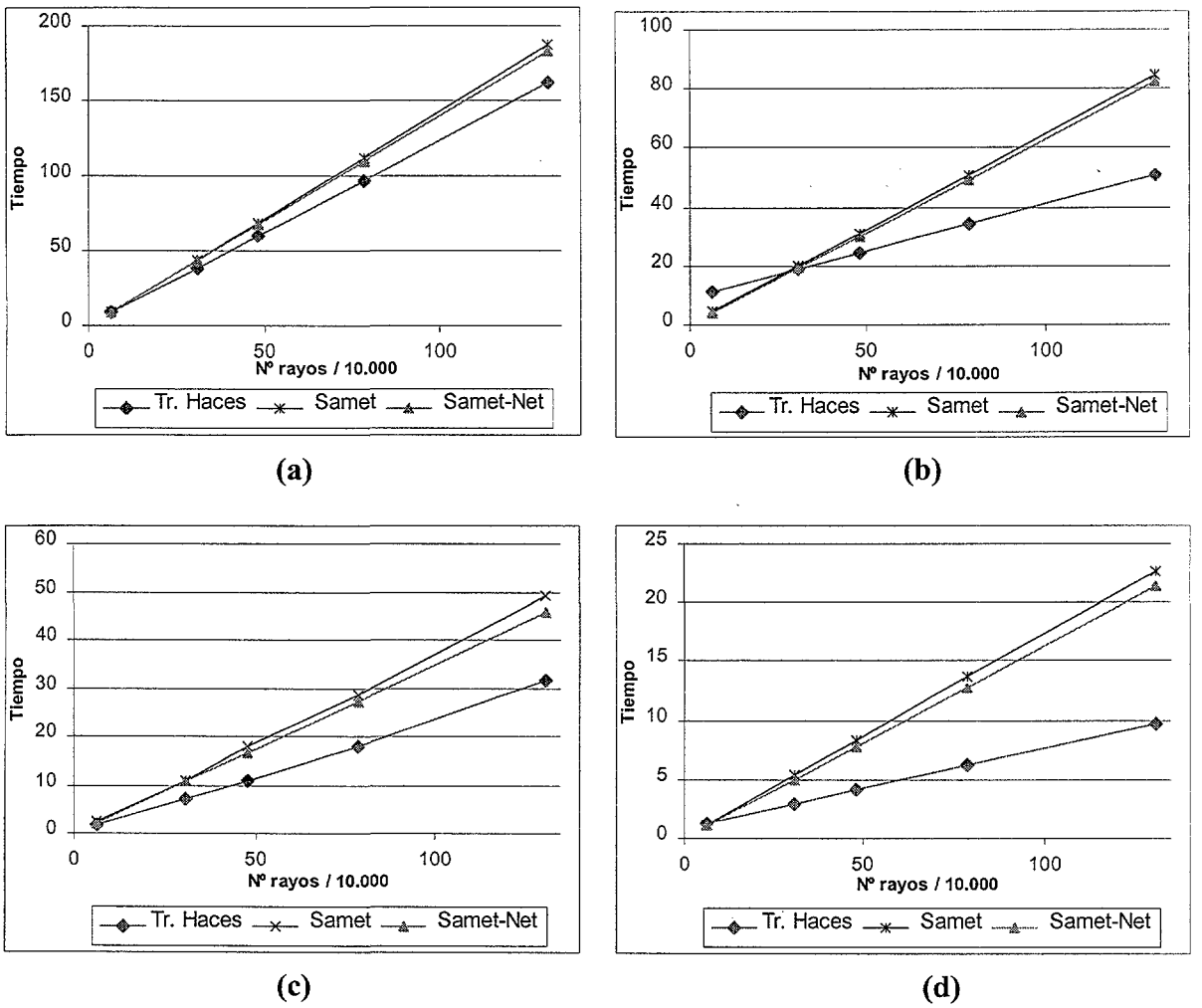


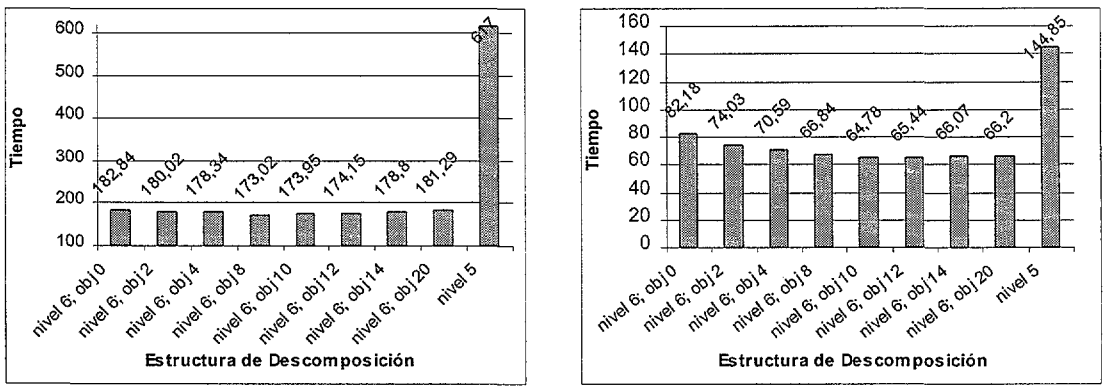

**Figura 5.13 Evolución del tiempo de trazado en las escenas: a) *Casa de Campo*; b) *Gato Bill*; c) *Misil* ; d) *Stetra* .**

En la tabla 5.5 se incluyen los resultados de las diferentes pruebas realizadas sobre las escenas anteriores. A su vez en la figura 5.13 se muestra gráficamente su evolución al aumentar la resolución de la imagen final. En líneas generales el comportamiento observado es muy similar al de las escenas denominadas *sintéticas,* procesadas anteriormente. Tal vez lo que más llame la atención es el resultado de la escena denominada "*Gato Bill*", pues, como podemos observar en la figura 5.13.b, el trazado de rayos individuales para resoluciones bajas (320x200 o 640x480) tiene un menor tiempo de trazado. Esto es debido a que la estructura de descomposición aplicada a la escena es un tanto elevada y, por tanto, los haces son de reducido tamaño y sólo se rentabiliza su trazado para altas resoluciones.

Como indicamos al principio de este apartado, siguiendo el criterio utilizado por Endl y Sommer [ENDL94], las pruebas anteriores se han realizado tomando como estructura de descomposición el mismo árbol octal tanto para el trazado de haces como para las otras dos alternativas de trazado individual. La utilización de la misma estructura nos permite ver que sobre dichas estructuras, y para altas resoluciones, el trazado de haces es más rápido. Pero debemos tener en cuenta que la estructura de descomposición utilizada está adaptada en cierta medida al trazado de haces y esto puede influir decisivamente en los ventajosos resultados observados para el trazado de haces. Por ello, a continuación vamos a ver qué resultados se obtendrían si procesáramos los algoritmos de trazado individual sobre estructuras más adaptadas a su forma de operar. En este caso vamos a analizar las escenas "*Casa de campo*" en la que la diferencia de tiempo del trazado de haces con las otras alternativas no es tan grande, y, junto a ella, vamos a estudiar también la escena denominada "*Gato Bill*" en la que los algoritmos de trazado individual tienen un comportamiento mejor que el trazado de haces para resoluciones bajas.

En la figura 5.14, puede apreciarse la evolución del tiempo de procesado al aplicar distintos criterios de descomposición y, por tanto, utilizar un árbol con estructura diferente. Si comparamos los valores obtenidos para la estructura más eficiente vemos que todavía el trazado de haces produce mejores resultados. Por

tanto, tras esta nueva comprobación podemos asegurar con mayor certeza que el trazado de haces tiene un comportamiento mejor que otras alternativas de trazado individual, incluso en el caso de que la estructura de descomposición utilizada por las técnicas de trazado individual esté adaptada a las características de dicho tipo de trazado.




**(a)** **(b)**

**Figura 5.14 Evolución del tiempo de trazado en las escenas: a) *Casa de Campo*; b) *Gato Bill.***

## 5.4.2 Comparación con técnicas que utilizan otras estructuras de descomposición del espacio.

Tras comprobar cuál es el comportamiento del trazado de haces con respecto a otras propuestas que utilizan también como estructura de descomposición los árboles octales, vamos a compararlo con otras alternativas que se apoyan en estructuras de descomposición diferente. En este caso, se han seleccionado dos algoritmos. El primero utiliza criterios de descomposición uniforme implementando un trazado de rayos sobre matrices de voxel. Por otra parte, el segundo utiliza un criterio de descomposición guiada por los objetos e implementa un trazado de rayos que se apoya en una jerarquía de volúmenes envolventes. Dentro de los diferentes algoritmos que utilizan las estructuras anteriores se han seleccionado las propuestas de Amanatides y Woo [AMAN87] para la descomposición uniforme y las

propuestas de Goldsmith y Salmon [GOLD87] y Kay y Kajiya [KAY86] para las jerarquías de volúmenes. Un mayor detalle de las implementaciones de ambos algoritmos los podemos encontrar en el trabajo de Molina y González [MOLI98] y en la descripción de la librería OORT [WILT94].

Como en las comparaciones realizadas anteriormente, primero analizaremos el comportamiento de las diferentes propuestas para las escenas "*sintéticas*" y posteriormente se estudiarán otras de las que hemos denominado "*reales*".

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 2'21 | 4'44 | 5'98 | 8'81 | 13'52 |
| *Matriz Voxel* | 7'12 | 9'01 | 12'3 | 15'15 | 22'18 |
| *Bounding Box* | 14'2 | 63'16 | 96'1 | 153'45 | 275'3 |

**(a)**

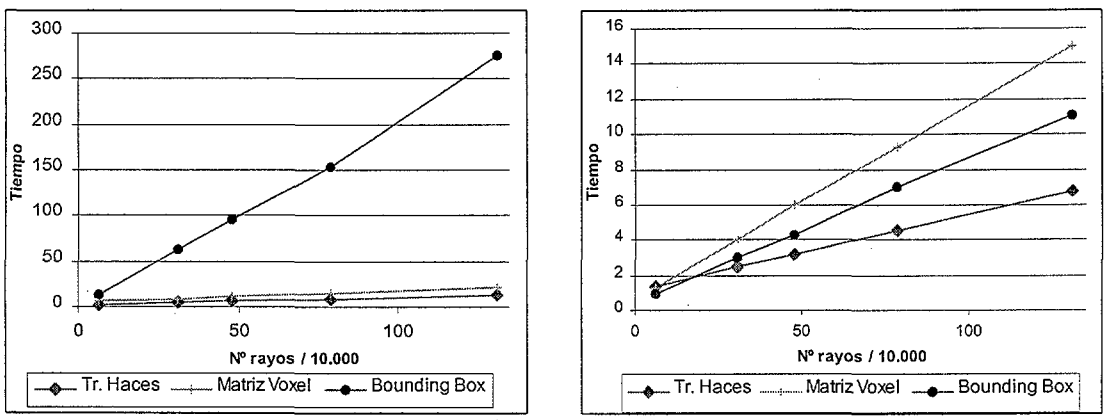
| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 1'35 | 2'44 | 3'2 | 4'51 | 6'78 |
| *Matriz Voxel* | 1'3 | 4'03 | 6'1 | 9'2 | 15'23 |
| *Bounding Box* | 0'9 | 3'1 | 4'3 | 7'06 | 11'3 |

**(b)**

**Tabla 5.6 Resultados del tiempo total de procesamiento. a) *Escena 1*; b) *Escena 2.***

Analicemos primero las escenas *sintéticas*. En la tabla 5.6 y en la figura 5.15 podemos observar el comportamiento de los diferentes algoritmos. Como podemos ver, en este caso la distribución de los objetos por la escena afecta decisivamente al comportamiento de los distintos algoritmos. Así en la primera escena, en la que existe una distribución más uniforme de los objetos, el algoritmo que utiliza una matriz de voxel se comporta sensiblemente mejor que el algoritmo que implementa

jerarquías de volúmenes. Esto es debido principalmente a que la fuerte distribución de los objetos produce una jerarquía difícil de recorrer y, por tanto, poco eficiente. En cambio, las matrices de voxel se adaptan bastante bien a este tipo de escenas, ya que existen pocas zonas libres. Analicemos ahora la segunda escena, en la que los objetos están fuertemente concentrados en el centro de la escena y la inclusión del plano posterior obliga a utilizar una matriz de gran tamaño. En este caso, vemos que las jerarquías de volúmenes tienen un mejor comportamiento que las matrices de voxel, ya que la matriz contendrá muchos voxel, la mayoría de ellos vacíos, por los que deberán pasar los diferentes rayos trazados. En cambio la jerarquía de volúmenes se adapta muy bien a estas situaciones, ya que rápidamente aísla la zona ocupada del resto de la escena, con lo que los rayos se trazan rápidamente.




**(a)** **(b)**

**Figura 5.15 Evolución del tiempo de trazado en las escenas: (a) *Escena 1*; b) *Escena 2.***

Pero, tal vez vuelve a sorprender el excelente comportamiento del trazado de haces en ambos casos. Incluso cuando la estructura de árbol octal parece no ser la más indicada, como sucede claramente en la *Escena 1*, el trazado de haces obtiene mejores resultados que el algoritmo que utiliza matrices de voxel, aun cuando podría considerarse la matriz de voxel como la estructura más indicada para este tipo de escenas. Esto mismo sucede con la *Escena 2,* en la que el espacio que ocupa la escena es muy grande con respecto a los objetos contenidos en ella, objetos que se

sitúan en el centro de la misma, dejando, por tanto, grandes espacios libres. Teóricamente, para este tipo de escenas, las jerarquías de volúmenes envolventes son una alternativa mejor que las matrices de voxel o los árboles octales. Pero, como podemos apreciar en la figura 5.15, el algoritmo de trazado de haces reduce los problemas derivados de una estructura de descomposición no muy apropiada y obtiene mejores resultados que el algoritmo que se apoya en las jerarquías de volúmenes (Bounding Box).

Tras este análisis basado en el comportamiento de los diferentes algoritmos sobre las escenas encuadradas dentro de las denominadas "*sintéticas*", vamos analizar algunas de las escenas del otro grupo. En este caso vamos a utilizar cinco escenas para estudiar el tiempo de procesado de los algoritmos seleccionados. Las dos primeras ("*Bola espacial*" y "*Tetera*") son escenas ya utilizadas anteriormente, pero que, por sus características, pueden ejemplificar situaciones interesantes. Las tres restantes ("*Ajedrez*", "*Flamenco*" y "*Habitación*") son escenas con distinta distribución de objetos, pero que tiene en común un notable incremento en el número de objetos, llegando en el caso de la escena "*Habitación*" hasta los cerca de 35.000 polígonos.

En la tabla 5.7 y en la figura 5.16 se muestran los resultados correspondientes a las dos primeras escenas. Lo primero que nos llama la atención es la gran diferencia de tiempos entre el método basado en jerarquía de volúmenes y el resto. Esta diferencia todavía se ve más acusada en la escena denominada "Bola espacial", ya que los diferentes objetos están muy próximos y ocupan casi la totalidad del espacio de la escena lo que provoca que la jerarquía de volúmenes asociada sea poco eficiente (la escena contiene 1.536 objetos y la jerarquía está formada por 1.154 cubos, indicando que el coste de la jerarquía es de 372). En la otra escena el incremento del tiempo de procesado para las jerarquías de volúmenes, no es tan grande debido a que la escena tiene grandes espacios libres por los que pasan gran cantidad de rayos de manera rápida y, por tanto, en este caso la eficiencia de la estructura es algo mayor (la escena contiene 1.004 objetos y la jerarquía la forma

731 cubos, asociándose un coste a dicha jerarquía de 130, sensiblemente menor que en el caso anterior aunque todavía superior al coste de la *Escena 1*, 4'64). En cualquier caso podemos ver que el incremento en el número de objetos hace poco interesante la utilización de las jerarquías de volúmenes tal cual se implementan en la librería OORT. Por ello, a partir de aquí, centraremos el análisis en el comportamiento de nuestro algoritmo y el de matrices de voxel.

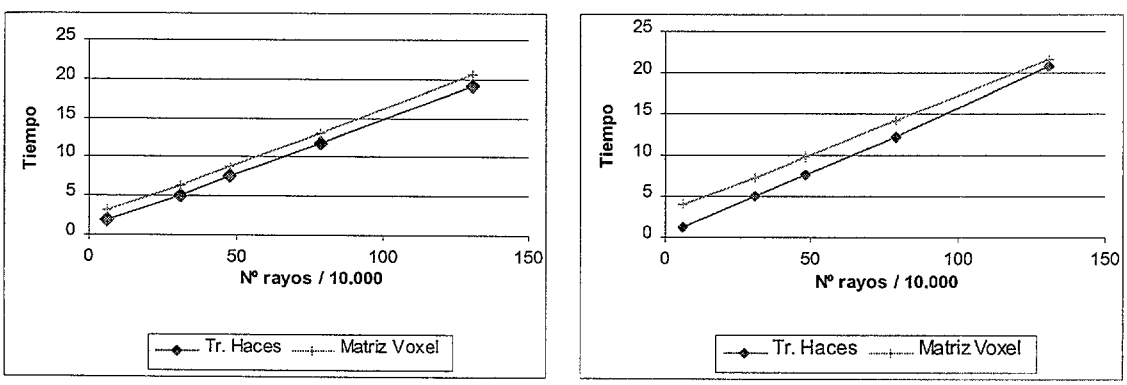
| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| *Tr. Haces* | 1'79 | 5'13 | 7'53 | 11'79 | 19'15 |
| *Matriz Voxel* | 3'05 | 6'32 | 8'75 | 13'11 | 20'5 |
| *Bounding Box* | 11 | 43 | 63 | 96 | 145 |

**(a)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 1'29 | 5'04 | 7'58 | 12'19 | 20'72 |
| *Matriz Voxel* | 4'06 | 7'25 | 9'85 | 14'2 | 21'65 |
| *Bounding Box* | 43 | 213 | 317 | 532 | 850 |

**(b)**

**Tabla 5.7 Resultados del tiempo total de procesamiento. a) *"Tetera"*; b) *"Bola espacial"*.**




**(a)** **(b)**

**Figura 5.16 Evolución del tiempo de trazado en las escenas: (a) *"Tetera"*; b) *"Bola espacial"*.**

Si observamos los resultados obtenidos por las matrices de voxel y el trazado de haces (figura 5.16), podemos apreciar que en el caso de la "*Tetera*", como podía esperarse, las técnicas que se basan en árboles octales producen mejores resultados que las matrices de voxel ya que hay muchas zonas libres de objetos y, como puede verse en la figura 5.9, el tamaño de los nodos terminales es bastante grande. En este caso la existencia de zonas libres permitía pensar que el trazado de haces tendría un buen comportamiento respecto a otras alternativas. Pero en cambio, el resultado de la segunda escena ("*Bola espacial*") no es tan esperado, ya que, como puede verse en la figura 5.9, la escena está ocupada de manera uniforme dejando muy pocos espacios libres. En esta situación lo normal es que las técnicas de trazado basadas en matrices de voxel tengan unos resultados mucho mejores que las basadas en árboles octales, pero al observar los resultados podemos ver que el trazado de haces obtiene un tiempo de procesado menor que las matrices de voxel. Esto es debido a que los rayos viajan en el haz y cuando éste llega a un nodo ocupado y se generan los rayos casi todos ellos chocan con objetos de la escena, no siendo necesario seguir trazándolos de modo individual. En este caso podemos ver que el proceso de trazado de haces es más rápido que el trazado sobre la matriz de voxel.

En las pruebas anteriores se han utilizado escenas con pocos objetos (no más de 1.536) y hemos visto que en el caso de las jerarquías de volúmenes el número de objetos repercutía considerablemente, haciendo poco recomendada su utilización. En cambio las otras dos alternativas no se veían afectadas por dicho incremento de manera tan importante. Las escenas que van a ser analizadas a continuación, son escenas bastante más complejas que las utilizadas hasta ahora y permitirán ver cómo afecta el número de objetos a la eficiencia de las diferentes alternativas. En la tabla 5.8 se incluyen los resultados obtenidos para las escenas "*Ajedrez*", "*Flamenco*" y "*Habitación*".

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| *Tr. Haces* | 18 | 34'59 | 46'59 | 67'88 | 103'12 |
| *Matriz Voxel* | 146 | 159 | 169 | 185 | 211 |
| *Bounding Box* | 130 | | | | 2.578 |

**(a)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 12'56 | 15'25 | 17'04 | 20'17 | 25'63 |
| *Matriz Voxel* | 478 | 483 | 486 | 498 | 516 |
| *Bounding Box* | 143 | | | | 2.692 |

**(b)**

| *Resolución* | *320x200* | *640x480* | *800x600* | *1024x768* | *1280x1024* |
|---|---|---|---|---|---|
| | *6'4* | *30'72* | *48* | *78'6432* | *131'072* |
| ***Tr.Haces*** | 18'64 | 51'78 | 74'38 | 115' 7 | 188'18 |
| *Matriz Voxel* | 162 | 194 | 214 | 253 | 316 |
| *Bounding Box* | 800 | | | | 17.456 |

**(c)**

**Tabla 5.8 Resultados del tiempo total de procesamiento. a) *"Ajedrez"*; b) *"Flamenco"*; c) *"Habitación"*.**

Tal vez lo primero que llama la atención, si analizamos los resultados, es que el tiempo de procesamiento del algoritmo que utiliza las jerarquías de volúmenes es elevadísimo en comparación con el resto. Este incremento en el tiempo de procesado de esta alternativa no es nuevo, pues ya se había detectado para escenas más simples, pero aquí se muestra en mayor grado y produce un descarte total de esta técnica o mejor dicho de la implementación que de ella se incluye en la librería OORT.

Centremos ahora nuestro estudio en el comportamiento de los otros dos algoritmos. En este caso, vemos que el algoritmo que utiliza matrices de voxel parte inicialmente, al procesar la escena sobre una resolución baja (320x200), de unos tiempos muy superiores a los del trazado de haces, aunque el incremento que se produce con el aumento de la resolución no es muy grande.

Si observamos los tiempos de procesado de la escena denominada "*Flamenco*" para una resolución de 320x200, vemos que partimos de un tiempo inicial de 478 segundos, mientras que el trazado de haces tarda menos de 13 segundos. Para comprender mejor este resultado debemos tener en cuenta que dentro del tiempo de procesado se incluye, en ambos casos, el tiempo necesario para generar la estructura de descomposición. Estructura que en el caso de los árboles octales se crea al principio, mientras que en el caso de la matriz de voxel se crea de modo dinámico. Por otra parte, el problema asociado a la creación de la matriz (ya sea dinámica o estáticamente) es que cada vez que deseamos ver los objetos contenidos en un voxel debemos analizar todos los objetos de la escena por lo que esta tarea puede convertirse en muy pesada si el número de objetos y el nivel de descomposición es elevado. Por ello, aunque hasta ahora hemos utilizado un nivel de descomposición similar tanto en el árbol octal como en la matriz, vamos a analizar qué sucederá si comparamos los resultados del trazado de haces con los obtenidos por la matriz de voxel pero, en este caso, sobre una estructura más eficiente.

Estudiemos cuál es el nivel de descomposición de la matriz que mejor se adapta a dos de las escenas: "*Ajedrez*" y "*Flamenco*". Para ello, en la figura 5.17 puede apreciarse la evolución del tiempo de procesado al modificar el nivel de subdivisión de la matriz. Como vemos en la escena "*Flamenco*" se produce un fuerte crecimiento a partir del nivel 16, es decir, una matriz de $2^{16}$ x $2^{16}$ x $2^{16}$, y para el valor más eficiente, que se corresponde con el nivel 13, el tiempo de procesado ha bajado notablemente situándose en 93 segundos. A la otra escena ("*Ajedrez*"), en cambio, le afecta más la aplicación de un nivel de descomposición más bajo que el propuesto inicialmente de $2^{32}$ x $2^{32}$ x $2^{32}$. Esto se debe principalmente a que un

reducido nivel de descomposición hace que el tiempo de determinación de la intersección de un rayo con los objetos contenidos en un voxel aumente notablemente. En todo caso vemos que para un nivel de 15 el resultado obtenido es el mejor de todos.

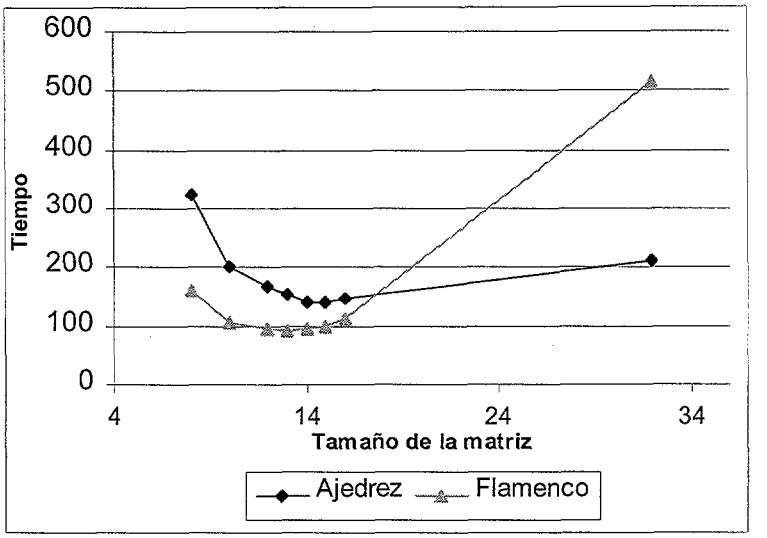
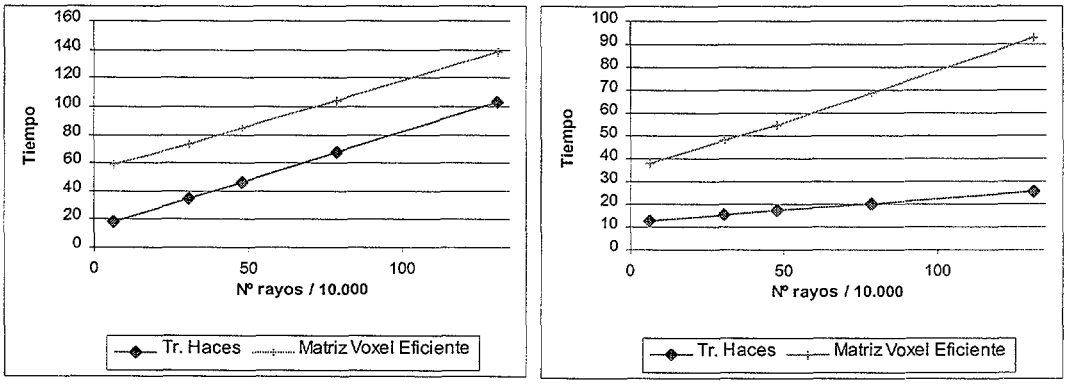



**Figura 5.17 Evolución del tiempo de trazado conforme aumentamos el nivel de subdivisión.**




**(a)** **(b)**

**Figura 5.18 Comparación del trazado de haces sobre árboles octales y el trazado utilizando matrices de voxel con un nivel de subdivisión eficiente. a) "*Ajedrez*"; b) "*Flamenco*".**

Una vez determinado el nivel de descomposición que mejor se adapta a cada escena procedamos a comparar los resultados que se obtendrían con dichas estructuras con respecto al trazado de haces. En la figura 5.18 se muestra la evolución del tiempo de respuesta del trazado de haces y del trazado sobre una matriz de voxel cuyo tamaño se ajusta al nivel considerado más eficiente (nivel 15 para "*Ajedrez*" y nivel 13 para "*Flamenco*"). En este caso vemos que los resultados del algoritmo de trazado de haces siguen siendo mejores que los obtenidos por el algoritmo basado en el recorrido de la matriz de voxel. La diferencia en el comportamiento de ambas escenas debemos buscarlo en la propia composición de las mismas.

Por tanto, tras las pruebas realizadas vemos que el algoritmo de trazado de haces es una alternativa muy a tener en cuenta a la hora de plantearse un método que pretenda acelerar el proceso de trazado de rayos, ya que no sólo mejora los resultados de los algoritmos que realizan un trazado individual de los distintos rayos sobre árboles octales, sino que, como acabamos de ver, la adaptación de la estructura aplicada a la escena a procesar no afecta de manera tan significativa como cabría esperar.

# Capítulo 6. Diseño y evaluación de una versión paralela del trazado de haces.

## 6.1 Introducción.

Como indican muchos de los investigadores que intentan acelerar el trazado de rayos ([WHIT92], [JANS93], [BADO94] etc.) toda propuesta de optimización de éste que pretenda alcanzar las máximas cotas de aceleración, pasa por tener presente la idea de realizar un procesamiento paralelo que sea capaz de reducir el tiempo total. Estas propuestas paralelas, permitirían teóricamente conseguir optimizaciones sucesivas con tan sólo incrementar el número de procesadores. Por desgracia esta situación ideal no puede producirse, pues, como definió Amdahl [AMDA67] (ley de Amdahl), existe un límite o cota que la incorporación al sistema de nuevos procesadores no puede superar. Aunque esta cota ha sufrido algunas rectificaciones [GUST88] más optimistas que tienen en cuenta el tamaño del problema, ausente en la formulación de Amdahl, debemos tener presente que dicha cota existirá. En todo caso, aunque la incorporación progresiva de procesadores al sistema no sea capaz de

reducir de manera continuada el tiempo de ejecución de un determinado proceso, se ha demostrado que en algoritmos que necesitan realizar gran cantidad de cálculos, como le sucede al trazado de rayos, la paralelización es la alternativa más eficiente.

Por ello, como ya hemos indicado, siempre que nos planteamos nuevas alternativas de optimización del trazado de rayos debemos tener presente que para alcanzar las máximas cotas de aceleración dicha propuesta debe ser fácilmente paralelizable. En este capítulo vamos a mostrar las posibilidades de paralelización que el trazado de haces ofrece y, en base a las pruebas realizadas en el capítulo anterior, seleccionaremos una alternativa concreta. Tras la selección de dicha alternativa procederemos a su diseño e implementación. La implementación se apoyará en la utilización de una librería (MPI Message-Passing Interface) que ofrece un conjunto de primitivas de comunicación que permiten, mediante las técnicas típicas de paso de mensajes, comunicar diferentes procesos (en el anexo B aparece una amplia descripción de las funciones básicas de esta librería).

## 6.2 Paralelización del trazado de haces.

En este apartado se va a proceder a describir las principales características del diseño y la implementación de la versión paralela. Para ello, vamos a partir de un primer estudio que nos permita ver las diferentes alternativas existentes y centre el marco de trabajo que guiará todo el proceso de diseño posterior.

### 6.2.1 Alternativas de paralelización y definición del marco de trabajo.

Como vimos en el capítulo segundo, existen multitud de alternativas de paralelización del trazado de rayos, basadas en diferentes arquitecturas y distintos criterios de distribución. Pero todas ellas parten de la independencia existente entre

los diferentes rayos utilizados para generar la imagen final. Esta independencia hace a este algoritmo fácilmente paralelizable.

La aplicación del paralelismo a diferentes campos ha dado lugar a distintas alternativas de diseño adaptadas a cada problema en concreto. Si atendemos a la clasificación aportada por B. Lester [LEST93] podemos encontrar las siguientes categorías de paralelismo: *paralelismo de datos, partición de datos, algoritmos autosuficientes, iteración síncrona, trabajadores replicados* y *procesado en pipeline*. Esta clasificación nos muestra las alternativas básicas que debemos tener presentes a la hora de diseñar una versión paralela de un algoritmo secuencial. En el gráfico 6.1 se incluye una pequeña descripción de las diferentes categorías. Estas categorías no son excluyentes sino que generalmente los diferentes algoritmos se podrán encuadrar dentro de varias de ellas.

Junto a las categorías definidas anteriormente, a la hora de diseñar una versión paralela de un algoritmo debemos tener presentes los problemas típicos que pueden provocar una reducción en la eficiencia de dicha versión. Estos problemas que limitan el rendimiento de los programas paralelos son los siguientes [LEST93]: *contención de memoria, excesivo código secuencial, tiempo de creación de un proceso, demora en la comunicación, demora en la sincronización* y *desbalanceo de la carga* (en el gráfico 6.2 se incluye una explicación básica de todos ellos).

En cualquier caso, dentro del ámbito de los algoritmos de síntesis de imágenes encontramos alternativas de tratamiento específicas que debemos tener igualmente presentes cuando nos proponemos diseñar una versión paralela de uno ellos. Como vimos en el capítulo segundo, dentro de la paralelización de este tipo de algoritmos podemos encontrar dos corrientes básicas: paralelización por *subdivisión de la imagen* o por *subdivisión de la escena* (ver gráfico 2.6).

***Paralelismo de datos.-***

Realización, en paralelo, de la misma operación sobre cada componente de la estructura de datos. El paralelismo de datos significa simplemente que la estructura de paralelismo se corresponde con la estructura de datos.

***Partición de datos.-***

Es un tipo especial de paralelismo de datos en el que el espacio de datos es dividido en regiones adyacentes, cada una de las cuales se procesa en paralelo por diferentes procesadores. Con ello, es posible que se generen nuevas necesidades de comunicación entre los distintos procesos.

***Algoritmos autosuficientes.-***

Cada proceso paralelo ejecuta por sí solo la parte de programa que se le ha asignado, sin necesidad de sincronización o comunicación entre procesos.

***Iteración síncrona.-***

Cada procesador realiza el mismo cálculo iterativo sobre una porción de datos diferente. Sin embargo, los procesadores deben sincronizarse al final de cada iteración.

***Trabajadores replicados.-***

Se mantiene una bolsa central con las tareas a distribuir entre los diferentes procesadores y el reparto se realiza de manera dinámica.

***Procesado en pipeline.-***

Los procesos son organizados en una estructura regular y los datos fluyen a través de dicha estructura de tal modo que cada proceso se encargue de una parte del tratamiento (en cada etapa se introducen los datos resultantes de la anterior y se preparan los que entrarán en la siguiente).

**Gráfico 6.1 Diferentes categorías de paralelismo [LEST93].**

Por tanto, teniendo en cuenta todo lo anterior vamos a intentar definir cuál es la alternativa de paralización que mejor se adapte al algoritmo de trazado de haces. Para ello debemos considerar también el estudio realizado sobre la versión secuencial de este algoritmo pues, en cierta medida, debe servir de referencia dentro del proceso de diseño de la versión paralela. En cualquier caso, debemos tener

presente que nuestra propuesta está ligada a la generación de imágenes de alta calidad asociadas siempre a altas resoluciones y a la utilización de sobremuestreo para reducir el problema de aliasing. Por lo que tal vez, el factor más importante será la gran cantidad de rayos a trazar dentro de las escenas.

***Contención de memoria.-***

La ejecución de un procesador se demora debido a que debe esperar a acceder a una posición de memoria pues otro proceso está accediendo en ese instante. Este problema se produce en sistemas con memoria compartida.

***Excesivo código secuencial.-***

En toda implementación paralela existe una porción de código puramente secuencial, pero si ésta es elevada puede afectar muy negativamente a las mejoras en el rendimiento esperado de esta versión respecto a la versión secuencial (*aceleración* o *speedup*).

***Tiempo de creación de un proceso.-***

En cualquier sistema real, la creación de un proceso requiere de un cierto tiempo de ejecución. Si el periodo en el que un proceso está activo es muy pequeño es posible que el tiempo de creación influya muy negativamente en su eficiencia.

***Demora en la comunicación.-***

Este problema sólo se produce en sistemas en los que existe comunicación entre los diferentes procesos. En muchos casos para enviar un mensaje entre dos procesadores es necesario que dicho mensaje pase por un conjunto de procesadores intermedios dentro de la red por la que se establece físicamente la comunicación.

***Demora en la sincronización.-***

Siempre que se impone una sincronización entre diferentes procesos esto significa que alguno de ellos deberá esperar a otros más lentos.

***Desbalanceo de la carga.-***

Cuando a priori no se puede establecer una distribución homogénea de la carga entre los diferentes procesos es posible que alguno termine antes que el resto, produciéndose un incorrecto balanceo o distribución de la carga.

**Gráfico 6.2 Problemas básicos del procesamiento paralelo [LEST93].**

Como hemos visto a lo largo de este trabajo, el algoritmo que proponemos tiene dos tareas básicas: 1) trazar los haces dentro de la estructura de descomposición de la escena; 2) obtener y determinar el color final de los pixels contenidos en un haz cuando éste llega a un nodo ocupado o abandona la escena. Ambas tareas pueden ser paralelizables ya que tanto los rayos como los haces pueden procesarse de modo independiente.

La primera tarea depende básicamente de la escena y la estructura de descomposición que tiene asociada, no viéndose afectada por la resolución utilizada. Por otra parte, como vimos en el capítulo anterior, si se ha utilizado un criterio de subdivisión adecuado, esta tarea no supone más del 10% del tiempo total de procesado. Por tanto, el trazado de un haz dentro de la estructura de descomposición no parece ser la tarea más crítica y, por ello, la que debe centrar el estudio de paralelización que acabamos de comenzar.

Por otra parte, la segunda tarea sí está ligada a la calidad de la imagen, aunque dentro de ella podemos distinguir dos situaciones claramente diferentes en base a la carga de trabajo que ambas suponen. La primera situación se produce cuando un haz sale de la escena. En este caso, la determinación del color de los pixel es trivial pues al abandonar la escena todos ellos tendrán el mismo color y éste se corresponderá con el color asociado al fondo de la escena. Por ello, para este primer tratamiento la única tarea que depende de la calidad de la imagen, es la búsqueda de los pixels contenidos dentro de un haz, ya que, como hemos indicado anteriormente, el color de cada uno se conoce a priori.

En cambio, si pensamos en el proceso anterior cuando el haz llega a un nodo ocupado vemos que éste es bastante más complicado y en él si afecta de manera importante la calidad de la imagen final. Como vimos en los capítulos anteriores, en este caso, se desconoce a priori el color de los pixels y, por tanto, la calidad de la imagen no sólo afectará al proceso de obtención de los pixel contenidos, sino que también influirá decisivamente en la tarea de determinación de su color. Por ello,

nuestra propuesta de paralelización se centrará en intentar realizar esta tarea de modo simultáneo y, por tanto, solapado en el tiempo.

Otro aspecto importante del diseño de todo algoritmo paralelo es seleccionar el entorno hardware donde se ejecutará, pues, como vimos en el capítulo segundo, existen diferentes alternativas arquitectónicas que afectarán a las decisiones de diseño. En este caso, nuestro algoritmo lo vamos a diseñar para un entorno de redes de estaciones de trabajo. La elección de esta alternativa se debe a varios motivos: su amplia utilización (en cualquier entorno de trabajo es muy usual disponer de varias estaciones de trabajo conectadas mediante una red), los grandes avances en el ratio coste-rendimiento de las estaciones de trabajo, y las grandes perspectivas de mejora en los tiempos de comunicación dentro de los nuevos diseños de redes de altas prestaciones ([ANDE95], [BODE95], [LANG98],...).

### 6.2.2 Diseño de la versión paralela.

Tras centrar el entorno en que se implantará el algoritmo y las tareas candidatas a distribuir, y, por tanto, tener definido nuestro marco de trabajo, podemos comenzar el proceso de diseño.

La primera gran elección consiste en la elección de la principal característica que condicionará todas las decisiones de diseño posteriores. Se trata, en este caso, de elegir el tipo de distribución que vamos a utilizar (*subdivisión de la escena* o *subdivisión de la imagen*). Para tomar esta decisión vamos a apoyarnos en los trabajos de D. Badouel, K. Bouatouch y T. Priol [BADO94]. Estos autores indican que las técnicas de *subdivisión de la escena* tienen dos grandes problemas: (1) dificultad en el diseño de los algoritmos y el balanceo de la carga, (2) decrecimiento de la eficiencia al aumentar el número de procesadores debido básicamente al incremento notable en las comunicaciones. Estos problemas hacen que la utilización de estas técnicas sea cada vez menor, máxime con la aparición de MIMD de memoria compartida virtual y el aumento en las posibilidades de memoria a bajo costo que podemos encontrar en una red de estaciones de trabajo. Por todo ello, en

nuestro caso suponemos que las capacidades de las estaciones de trabajo son tales que disponen de suficiente memoria como para almacenar toda la escena. Por tanto, partiendo de esa hipótesis nuestro entorno de trabajo sería semejante al descrito dentro del tipo "*paralelismo de datos*" y "*subdivisión de la imagen*", ya que cada procesador tiene accesible la totalidad de la escena y la subdivisión está guiada por la distribución de los pixel para los que deseamos conocer su color, entre los diferentes procesadores. En este caso, cada proceso es capaz de resolver el cálculo del color de un pixel por sí solo.

Pero nuestro algoritmo de trazado de haces introduce ciertas diferencias con el proceso de trazado de rayos individuales que afecta básicamente al diseño de la versión paralela. En nuestro caso, la determinación del número de rayos a generar por cada uno de los pixel de pantalla no se realiza antes de comenzar el trazado sino que se resuelve dentro del propio proceso de trazado de los diferentes haces. Por tanto, no es posible aplicar una distribución apriorística de los rayos entre los diferentes procesadores de tal modo que la carga que suministremos a cada uno sea equivalente. De este modo, de los problemas citados anteriormente (ver gráfico 6.2) uno de los más relevantes en este entorno es la necesidad de establecer un reparto que asegure un correcto balanceo de la carga. Por otra parte, siempre que se pretende paralelizar una versión del trazado de rayos existe el peligro de perder la coherencia existente en este tipo de algoritmos (ver capítulo segundo). En nuestro caso, el envío de los pixel agrupados en haces permite pensar que dichos pixel tendrán una fuerte relación entre sí y, por tanto, los niveles de coherencia deberán ser altos. Con ello, se permitirá que técnicas, como el *ShadowCache*, basadas en la coherencia de rayos, puedan aplicarse con un mayor éxito.

Como hemos visto, dentro de nuestro algoritmo, es necesario que un haz, después de un recorrido por la escena, llegue a un nodo ocupado antes de generar un rayo y determinar su color mediante el proceso de trazado de rayos más o memos clásico. Por tanto, nuestro esquema de distribución consiste en un proceso encargado de trazar haces y cuando estos llegan a un nodo ocupado enviar un mensaje a uno de

los procesadores libres para que se encargue de determinar los pixel contenidos en él y obtener finalmente su color.

El esquema de distribución (ver figura 6.1) se corresponde con un esquema del tipo *productor-consumidor*. En este caso, el *productor* (también denominado *host*) realizará el trazado de los diferentes haces y cuando un haz llega a un nodo ocupado generará la información necesaria para que el *consumidor* (denominado en alguna ocasión *server*) se encargue de calcular los pixel contenidos y determinar finalmente su color. En este caso, el proceso *productor* se encarga no sólo de generar el trabajo que deberán procesar los *consumidores*, sino que deberá realizar la distribución de la carga entre los diferentes procesos *consumidores* disponibles, de manera que consigamos un correcto balanceo de la misma.

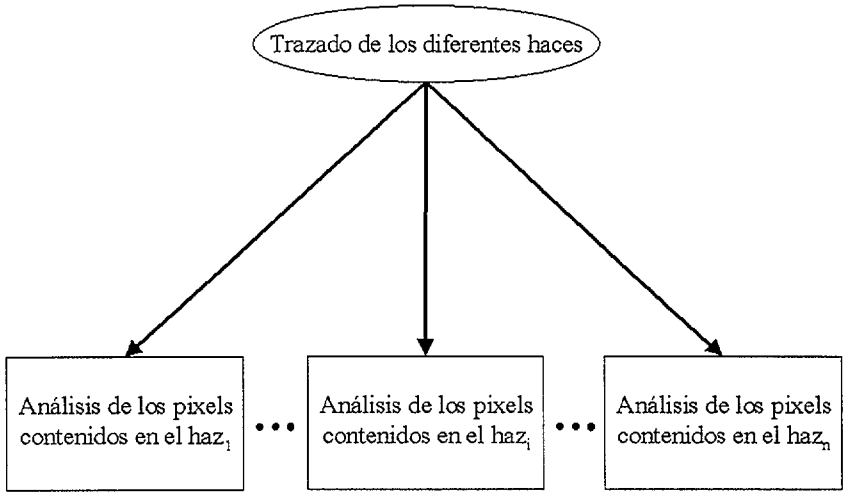



**Figura 6.1 Esquema de distribución de tareas.**

Este esquema de trabajo que acabamos de describir se encuadra dentro del tipo "*trabajadores replicados*" según la clasificación propuesta por B. Lester [LEST93]

(ver cuadro 6.1). Siguiendo dicho esquema, en este caso se propone la realización de un balanceo de la carga de modo dinámico, ya que cada *trabajador* (*consumidor*) tras procesar la carga asignada previamente solicitará más a la *bolsa de trabajos pendientes*, gestionada por el proceso *productor*. Sin embargo, en el esquema que aquí se propone los *trabajadores* no incorporan nuevas tareas a la *bolsa de trabajos pendientes,* siendo el *proceso* productor el único capaz de generar nuevas entradas en dicha bolsa. El balanceo dinámico de la carga propuesto en este caso, implica que cada vez que un proceso de tipo *consumidor* finaliza su tarea éste deba comunicarse con el proceso *productor* para solicitar nuevas tareas. Siguiendo el esquema anterior, debemos tener presente que las necesidades de comunicación crecen y, por tanto, el problema asociado a la *demora en la comunicación* debe tenerse en cuenta, pues puede afectar sensiblemente al rendimiento del algoritmo paralelo. Por ello, debemos enviar a los procesadores *consumidores* tareas con un coste previsiblemente superior al que conllevan las tareas de comunicación.

**•Alternativas de paralelización.**

- •Paralelismo de datos.
- •Algoritmos autosuficientes.
- •Trabajadores replicados.

- •Subdivisión del espacio de la imagen.

**•Principales problemas a tener en cuenta.**

- •Desbalanceo de la carga ⇨ Balanceo dinámico.
- •Demora en la comunicación ⇨ Distribución sólo de haces que llegan a nodos ocupados.

**Gráfico 6.3 Criterios de paralelización utilizados en el diseño del algoritmo.**

Tras esta explicación de las características básicas del modelo asociado al algoritmo paralelo que se propone (ver gráfico 6.3), vamos a profundizar en el modo en que éste afecta al modelo OO descrito en el capítulo cuatro. Si analizamos el esquema general de distribución de tareas descrito anteriormente, podemos ver que el servicio que se verá afectado en mayor medida por la paralelización será el denominado "*BuscarPixel*" (ver figura 4.10) perteneciente a la clase *Haz*. Este servicio es el encargado de dado un haz, buscar los pixel contenidos y determinar el color final de cada uno de ellos. Pero si analizamos más detenidamente este servicio podemos ver que el tratamiento propuesto difiere sensiblemente en función de que el haz que estemos analizando se corresponda con uno que abandona la escena o con uno que llegue a un nodo ocupado. En el primer caso, el tratamiento tan sólo consiste en buscar los pixel contenidos ya que el color de todos ellos se conoce a priori y coincide con el color asignado al fondo de la escena (si se considera que no existe ningún foco de luz en la dirección de salida). Por tanto, el tiempo asociado al tratamiento de estos haces que salen se reduce a proyectar el haz sobre la pantalla y buscar los pixel contenidos en el polígono resultante tras dicha proyección, utilizando, para ello, el algoritmo de rellenado de polígonos mediante líneas de rastreo. El tiempo necesario para procesar estos haces es tan reducido que no parece interesante enviar esta tarea a los procesos *consumidores*, ya que es posible que el tiempo de comunicación fuese mayor que el tiempo propio de procesado. Por ello, esta tarea se asigna al proceso denominado *productor*, el cual no sólo se encarga de generar tareas para que sean atendidas por los *consumidores,* sino que también procesa las tareas más simples que él mismo genera.

Esta distribución de las tareas asignadas inicialmente al servicio *BuscarPixel* entre distintos procesadores conlleva la división de dicho servicio en dos, uno asociado a haces que salen y otro asociado a haces que llegan a un nodo ocupado. Esta separación puede realizarse fácilmente si tenemos en cuenta que el tipo de haz al que se corresponde cada uno de los servicios anteriores es diferente. Por tanto, si aplicamos la idea de la herencia podemos crear dos nuevas clases derivadas de la

clase *Haz* (*HazSale* y *HazNodoOcupado*), en las cuales la principal diferencia con respecto a la clase genérica reside en la implementación del servicio *BuscarPixel*.

Por otra parte, debemos tener presente que se necesita un servicio que reciba la información del haz (clase genérica *Haz*) enviada por el proceso *productor* y cree un haz (clase derivada *HazNodoOcupado)* con las mismas características en el proceso *consumidor* (servicio *EsperarHaz*). La creación de un haz con las mismas características implica asignar las aristas y situarse dentro del árbol en el mismo nodo en el que estaba situado el haz dentro del proceso *productor* (utilizamos el servicio *AsignarValoresHaz*). Para ello, primeramente se necesita asegurar que la escena y la estructura de descomposición asociada será igual en el *productor* y *consumidor*. A su vez, al enviar la información del haz, en vez de mandar una variable que permita direccionar una posición de memoria donde se encuentra situado el nodo, debemos enviar algún código que permita al proceso *consumidor* determinar, de forma única, el nodo al que hace referencia y crear de manera precisa el mismo haz que generó el *productor*. Esto implica que es necesario añadir un servicio a la clase *Octree* que busque el nodo del árbol correspondiente a un código dado (*BuscarNodoHaz*) y permita enlazar dicho nodo con el nuevo haz creado.

Junto a las modificaciones anteriores, debemos incluir nuevos servicios encargados de gestionar las comunicaciones entre los diferentes procesadores que intervienen en el sistema. De este modo, se crean dos nuevos servicios, uno asignado al proceso *productor*, asociado a la gestión de la *bolsa de trabajos pendientes* y otro asignado a los procesos *consumidores* que les permite recibir los trabajos (haces a procesar) y enviar los resultados. Pero, en este caso, para explicar el funcionamiento de estos nuevos servicios tal vez sea necesario conocer algunas de las características de los mecanismos de comunicación que ofrece la librería (MPI) utilizada para su implementación.

### 6.2.3 Implementación del algoritmo paralelo mediante MPI.

MPI [GROP96], [PACH97] es una librería que incluye los mecanismos básicos para establecer y controlar las comunicaciones entre diferentes procesadores. En este caso, su diseño está guiado por tres principios: eficiencia, funcionalidad y portabilidad. Por tanto, debe ser tan eficiente como el propio lenguaje suministrado por el vendedor de una determinada arquitectura hardware y debe ser capaz de realizar las mismas funciones que dicho lenguaje. Pero tal vez la mayor ventaja que ofrece esta plataforma de programación de algoritmos paralelos es su portabilidad que hace que una misma implementación pueda ser ejecutada en diferentes arquitecturas (p.e. máquinas de tipo MIMD de diferentes fabricantes, redes de estaciones de trabajo tanto heterogéneas como homogéneas, etc.). Por tanto, el programador puede escribir programas portables que puedan ser ejecutados de modo eficiente en arquitecturas suministradas por diferentes vendedores.

En el anexo B aparece una descripción más completa de las características de esta librería y de las principales funciones que permiten establecer y controlar las comunicaciones entre los diferentes procesadores. En todo caso, a continuación se describirán las características básicas de los principales mecanismos de comunicación disponibles.

MPI ofrece la posibilidad de realizar tanto un envío *punto a punto* (*MPI_Send, MPI_Recv*) como una *comunicación colectiva* (*MPI_Broadcast, MPI_Gather, MPI_Reduce,* etc). Por otra parte, permite solapar en la medida de lo posible el procesamiento de las tareas asignadas a cada procesador y el control de la comunicación, mediante la inclusión de comunicación no bloqueante (*MPI_Isend, MPI_Irecv,* etc.). A su vez, ofrece la posibilidad de agrupar los procesos creando la noción de *comunicador* (el comunicador estándar es *MPI_COMM_WORLD*). Por último, es capaz de realizar la exploración de una línea de comunicación (*MPI_Probe, MPI_Iprobe*) para analizar la existencia de un mensaje que cumpla unas determinadas características (destino, etiqueta y comunicador establecido)

Por lo general, dentro de MPI, al lanzar la ejecución del programa paralelo es cuando se definen el número de procesos a utilizar en la distribución. En este punto cada uno de los procesos generados puede procesar un mismo programa o varios programas diferentes, facilitando así un tipo de programación tanto *SPMD* (*Simple Program Multiple Data*) como *MPMD* (*Multiple Program Multiple Data*). En nuestro caso, la situación que proponemos se ajusta al esquema *MPMD*, ya que el procesamiento asignado al *productor* es totalmente diferente del que se asocia a los procesos *consumidores*.

El hecho de que la escena esté totalmente accesible para todos los procesadores facilita la implementación del programa paralelo a la vez que reduce sustancialmente las comunicaciones. En este caso, por tanto, nuestro algoritmo se encuadra dentro de los denominados de *subdivisión del espacio de la imagen*. Pero, como hemos indicado, ahora nos encontramos con una situación diferente, ya que se desconoce a priori el número exacto de rayos a generar por cada pixel, lo que hace que las propuestas hasta ahora definidas (ver capítulo segundo), sobre todo desde la perspectiva del balanceo de la carga, no puedan aplicarse con éxito en este entorno.

Por otra parte, las agrupaciones de pixel en haces se realizan en función de la estructura de la escena pero no tienen en cuenta el tiempo necesario para procesar cada agrupación. De este modo, es muy probable que el tiempo de procesado de cada haz (grupo de pixel asociado al haz) sea diferente y en algunos casos las diferencias de tiempo pueden ser notables. A su vez, se desconoce a priori el número de haces que se generarán y llegarán a un nodo ocupado, pues el modo de creación es dinámico. Todo ello, nos lleva a proponer un tipo de balanceo dinámico que garantice que a un proceso no se le asigne más carga hasta que no termine de procesar la que le ha sido previamente asignada.

El esquema del algoritmo de paralelización propuesto es el siguiente: el proceso *productor* traza los distintos haces y cuando llega a un nodo ocupado comienza el proceso de comunicación con uno de los procesos *consumidores*. Para

ello, el *productor* analiza cuál de dichos proceso está libre y, siguiendo un criterio de asignación rotativo (*round robin*), le asigna el haz al proceso que más tiempo lleva esperando. En el caso de que no haya ninguno libre entra en un bucle de llamadas al servicio *VerPendientes* que finalizará cuando llegue la notificación de que uno de los *consumidores* ha enviado los resultados de un haz previo y ya está disponible para procesar uno nuevo. Tras ello, inicia la operación de comunicación (*MPI_Isend*) y se continúa con el procesado sin necesidad de esperar a que dicho envío se haya completado (utiliza un envío no bloqueante). La utilización de esta función de comunicación no bloqueante, aunque permite seguir trazando otro haz mientras se realiza el envío, puede dar lugar a algunos problemas si antes de volver a enviar el siguiente haz no comprobamos que la operación anterior ya ha sido completada. Para ello, primero se comprueba si existe algún envío pendiente (MPI_Test) y, tras ello, si fruto de esta comprobación se llega a la conclusión de que el envío previo no ha sido completado, se llama a la función (*MPI_Wait*) que detendrá el procesamiento hasta que dicha comunicación no finalizada termine. En la figura 6.2 se muestra el escenario asociado al servicio encargado de la comunicación entre los procesos *productor* y *consumidor*.

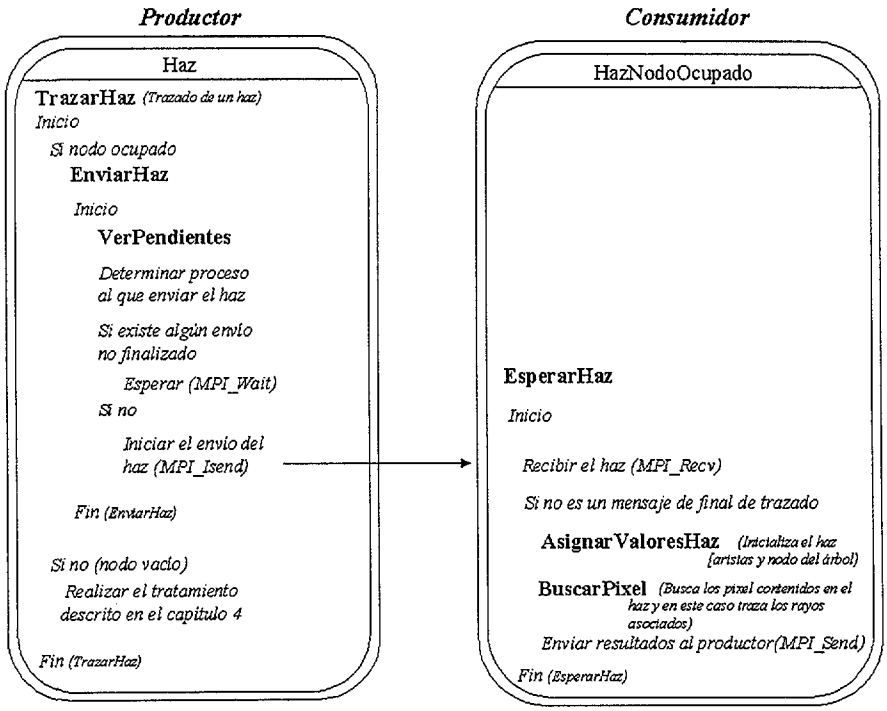



**Figura 6.2 Escenario del servicio encargado de la comunicación productor-consumidor.**

En el gráfico 6.4 se muestra la implementación del servicio encargado de mirar si está pendiente de recepción algún mensaje enviado por uno de los procesos *consumidores*. Este servicio (*VerPendientes*) se procesa siempre antes de realizar el envío de un nuevo haz. De este modo, condicionamos la frecuencia con la que miramos y atendemos a los resultados enviados por los procesos *consumidores,* a la frecuencia con que se generan nuevos haces que llegan a nodos ocupados. Esto podría ser una fuente de problemas si el tiempo entre dos llegadas consecutivas de mensajes enviados desde los diferentes procesos *consumidores* fuese mucho menor que el tiempo entre la generación de dos haces asociados a nodos ocupados, lo cual no sucede en la práctica. Por otra parte, el tratamiento anterior evita la necesidad de incluir, dentro del sistema, un proceso que esté continuamente comprobando el estado de las colas que reciben los envíos provenientes de los *consumidores*.

```
VerPendientes(MPI_Comm comm,...)
{
   int emisor, flag, numero,tag;
   MPI_Datatype tipo_emitido;
   MPI_Aint tamanno;
   MPI_Status estado;
   MPI_Estadisticas estadisticas_proc; //Creación de una variable del tipo MPI_Estadisticas definido por nosotros
   pixel *matriz_recibida;
   do{
      // Comprueba si tiene mensajes pendientes y obtiene el estado
      MPI_Iprobe (MPI_ANY_SOURCE,tag,comm,&flag,&estado);
      // Si existen mensajes pendiente, es decir, flag=TRUE
      if (flag)
      {
        emisor = estado.MPI_SOURCE;   // Obtiene el emisor del mensaje
        MPI_Get_Count(&estado, tipo_emitido, &numero) //Obtiene el tipo de dato y el número de elementos
        MPI_Type_extent(tipo_emitido, &tamanno); // Obtiene el tamaño del tipo de dato
        matriz_recibida = malloc (tamanno * numero); // Reserva espacio para recibir la matriz de resultados
        MPI_Recv(matriz_recibida, numero, tipo_emitido, emisor, tag,comm); // Recibe el mensaje de
                                                                           resultados
        TratarResultados(matriz_recibida);
        MPI_Recv(estadisticas_proc,1,MPI_Estadisticas,emisor,tag,comm); // Recibe el mensaje con las
                                                                        //estadísticas parciales
        GuardarEstadisticas(estadisticas_proc)
      }
   } while (flag);
}
```

**Gráfico 6.4 Comprobación de la existencia de mensajes de los procesos *consumidores* pendientes de ser recibidos.**

El funcionamiento del servicio *VerPendientes* es bastante interesante pues está pensado con un doble objetivo: solapar computación y comunicación, y evitar la necesidad de emplear diferentes canales de comunicación para los distintos procesos *consumidores*. El primero de ellos se consigue al hacer que la comprobación de la recepción de un mensaje no sea bloqueante (*MPI_Iprobe*) y, por tanto, si en el momento de la comprobación no hay ningún mensaje pendiente que cumpla las condiciones establecidas en la sentencia *MPI_Iprobe* (flag=0), pueda continuarse con el tratamiento de nuevos haces. Por otra parte, la recepción de un mensaje (*MPI_Recv*) impone conocer a priori el origen del mensaje y su tamaño, pero, en este caso, pueden llegar mensajes desde diferentes procesadores y el tamaño del paquete puede ser distinto en cada envío (su tamaño depende del número de pixel contenidos dentro del haz que ha procesado). Para solventar el primer problema, podríamos establecer tantas líneas de comunicación como procesos, pero esto complicaría la ejecución de nuestro algoritmo al considerar un número diferente de procesos *consumidores* (aumentar el número de procesadores), a la vez que no solventaríamos el problema del desconocimiento del tamaño. Para resolver el problema del desconocimiento del origen y el tamaño del mensaje, antes de realizar la recepción real del dicho mensaje se necesita realizar una serie de operaciones a partir de la información que la función *MPI_Iprobe* ha recogido. Estas operaciones permiten reservar el espacio necesario para el buffer de entrada de datos y determinar el proceso que envía el mensaje, como sucede antes de realizar el primer *MPI_Recv* que aparece en el gráfico 6.4. Tras ello, recibe un mensaje con las estadísticas pero, en este caso, ya conoce tanto el origen como el tamaño del paquete (*MPI_Estadisticas*) que va a recibir.

En cualquier caso, cuando finaliza el trazado del último haz es posible que queden pendientes envíos de los procesos *consumidores*. Por ello, para evitar problemas en la recepción de dichos mensajes pendientes, el proceso *productor,* antes de finalizar totalmente la ejecución de su programa, debe explorar la cola de entrada y cuando reciba un envío, tras tratarlo, mandar al proceso *consumidor* correspondiente un mensaje que le indique a éste que ya ha finalizado

completamente el trazado. Finalmente, una vez recibidos de los procesos *consumidores* todos los mensajes pendientes, se debe comprobar que a todos ellos se les ha notificado que el trazado ha finalizado y en caso de que quede alguno sin notificar se le debe enviar un mensaje de finalizado.

Junto al tratamiento anterior, asociado al proceso que hemos denominado *productor*, cada proceso *consumidor* debe estar preparado para realizar el procesamiento de los haces que se le envíen. Pero antes, cada uno de ellos debe cargar la escena y preparar la estructura de descomposición asociada. Para esto se necesita que el proceso *productor* les envíe el nombre del fichero NFF que contiene la escena y las características de descomposición seleccionadas, para que así todos los procesos dispongan de la misma escena y la misma estructura de descomposición. Esta comunicación se realiza mediante una función de comunicación colectiva (*MPI_Broadcast*), ya que todos los procesos deben recibir la misma información. Tras este tratamiento de creación de la estructura asociada a la escena, el programa correspondiente a los procesos *consumidores* debe entrar en un bucle de llamadas a la función encargada de procesar los haces que el *productor* le envíe (*EsperarHaz*). Este bucle de llamadas terminará cuando dicha función reciba un *mensaje de final*, en cuyo caso el proceso *consumidor* debe terminar su procesamiento, pues ya no se le enviarán más haces.

En el gráfico 6.5 se describe el funcionamiento general del tratamiento básico del servicio *EsperarHaz* que debe realizar cada proceso *consumidor*. Como vemos el tratamiento comienza con una operación de recepción de un mensaje, pero en este caso dicha operación es bloqueante, lo que significa que hasta que no se haya completado no se continuará con las siguientes instrucciones. Una vez recibido el mensaje si éste no indica el final del tratamiento (*mensfinal*), sino que se corresponde con un haz que debe ser procesado, lo primero que se hace es inicializar los atributos de un objeto de la clase *HazNodoOcupado* (clase derivada de *Haz*) y, tras ello, llamar al servicio *BuscarPixel* (servicio que ha sido redefinido y que realiza un tratamiento distinto al de la clase genérica *Haz*). Una vez finalizado el

procesamiento del haz en cuestión, el servicio *BuscarPixel* devuelve una lista de pixel, la cual tras situarla en la estructura de emisión se envía al proceso *productor*, junto con las estadísticas parciales generadas hasta el momento (tiempo de procesamiento del haz, número de rayos que intersecan o salen, etc.). Una vez que se ha terminado el tratamiento de un haz y, por tanto se ha solicitado otro, el proceso *consumidor* vuelve a llamar al servicio *EsperarHaz* para estar preparado ante una posible asignación de un nuevo haz a dicho proceso (se queda esperando en la instrucción *MPI_Recv*).

```
EsperarHaz(MPI_Comm comm, unsigned char final,...)
{
   MPI_Haz haz; //Creación de una variable del tipo MPI_Haz que contiene el mensaje enviado por el productor
   pixel *matriz_enviada;
   MPI_Estadisticas estadisticas;
   // Inicio de la definición de otras variables
   ...
   // Final de la definición de otras variables

   MPI_Recv(haz, 1, MPI_Haz, 0, tag,comm); // Espera hasta recibir un haz del proceso 0 (productor)
   if (haz = mensFinal) // Si el mensaje recibido indica que no hay más haces que procesar
      final = TRUE;
   else
   {
      // Procesamos el haz que hemos recibido.
      hazProcesar->AsignarValoresHaz(haz);
      hazProcesar->BuscarPixel(listaPixel,numPixel); // devuelve una lista de pixel y el número total de ellos

      // Ha finalizado el tratamiento del haz y comunicamos los resultados
      matriz_enviada = malloc (sizeof (pixel) * numPixel ); // Reserva espacio para enviar la matriz de resultados
      AsignarValoresMatriz(listaPixel,matriz_enviada); // Asigna los valores de los pixel a la matriz a enviar
      MPI_Send(matriz_enviada, numPixel, MPI_Pixel, 0, tag, comm); // Enviar resultados al proceso 0
      PrepararEstadisticas(estadisticas);
      MPI_Send(estadisticas, 1, MPI_Estadisticas, 0, tag, comm); // Enviar estadisticas parciales al proceso 0
   }
}
```

**Gráfico 6.5 Tratamiento realizado por los diferentes procesos *consumidores*.**

## 6.3 Análisis de resultados.

Tras describir las características más importantes del algoritmo paralelo que se propone, vamos a analizar a continuación los resultados de éste. Para ello hemos

seleccionado varias escenas que servirán como casos de prueba (*Gato Bi*ll y *Torre Eiffel*). En todos los casos la resolución seleccionada ha sido 1280x1024, pero considerando a su vez la aplicación de un sobremuestreo uniforme de 5 rayos por pixel. Para realizar las diferentes pruebas se ha utilizado una red de estaciones de trabajo compuesta por 15 estaciones homogéneas. Cada una de ellas está formada por un ordenador con un procesador Pentium II a 330 Mh y 32Mb de memoria principal. La red utilizada para conectar las diferentes estaciones es de tipo Ethernet.

El análisis de un programa paralelo es diferente al que se puede realizar sobre uno secuencial. En este caso el estudio del tiempo de ejecución debe realizarse en función de dos variables: tamaño de la entrada y número de procesos. Por tanto, en vez de utilizar *T(n)* para denominar al rendimiento, debemos usar una función de dos variables *T(n,p)*. El tiempo al que aquí estamos haciendo referencia es el correspondiente al que transcurre desde el instante en que el primer proceso comienza su ejecución hasta el instante en que el último proceso completa la ejecución de su última sentencia.

La definición anterior de la variable que nos permite estudiar el comportamiento de un programa paralelo es un tanto errónea, ya que realmente dicho tiempo no depende del número de procesos. Como podemos suponer, el tiempo de ejecutar *n* procesos en un único procesador es mayor que el tiempo que se necesitaría para ejecutar dichos procesos cada uno en un procesador diferente. En cualquier caso, siempre que se toma esta función para analizar el comportamiento del tiempo se supone que cada proceso se ejecuta en un procesador diferente.

Generalmente cuando se desea analizar el comportamiento de la versión paralela de un programa secuencial, se debe establecer alguna función que permita comparar el funcionamiento de ambas versiones (secuencial y paralela). Para ello, las funciones más comúnmente utilizadas son las denominadas *aceleración* (*speedup*) y *eficiencia*. La primera de ellas, se utiliza para comprobar la relación existente entre el tiempo de procesamiento de la versión secuencial y el de la

paralela. Por tanto, la expresión correspondiente al concepto de *aceleración A(n,p)* es la siguiente:

$$A(n,p) = \frac{T_{vs}(n)}{T_{vp}(n,p)} \qquad (6.1)$$

donde,

$T_{vs}(n)$ es el tiempo de procesado utilizado por la versión secuencial.

$T_{vp}(n,p)$ es el tiempo de procesado utilizado por la versión paralela.

Para un valor constante de $p$, se cumple que $0 < A(n,p) \leq p$. Si $A(n,p)=p$, se dice que el programa tienen una *aceleración lineal o teórica*. Esto es desde luego algo poco habitual ya que debemos tener en cuenta, la sobrecarga producida por las comunicaciones o la generada por las tareas necesarias para controlar y distribuir la carga de manera balanceada. A su vez, se debe tener en consideración que la parte secuencial del algoritmo paralelo diseñado afecta considerablemente en la evolución de la aceleración al incluir nuevos procesadores en el sistema (ley de Amdahl).

Una alternativa a la medida de la *aceleración* es la utilización de la *eficiencia*. Esta nueva medida refleja la utilización de un proceso dentro de un algoritmo paralelo. Se define la *eficiencia* como:

$$E(n,p) = \frac{A(n,p)}{p} \qquad (6.2)$$

En este caso, tenemos la siguiente relación $0 < E(n,p) \leq 1$. Por otra parte, si $E(n,p) = 1$ decimos que el programa paralelo exhibe aceleración lineal, mientras que si $E(n,p) = 1/p$ el programa tiene una eficiencia menor a la teórica, situación casi habitual en los programas paralelos.

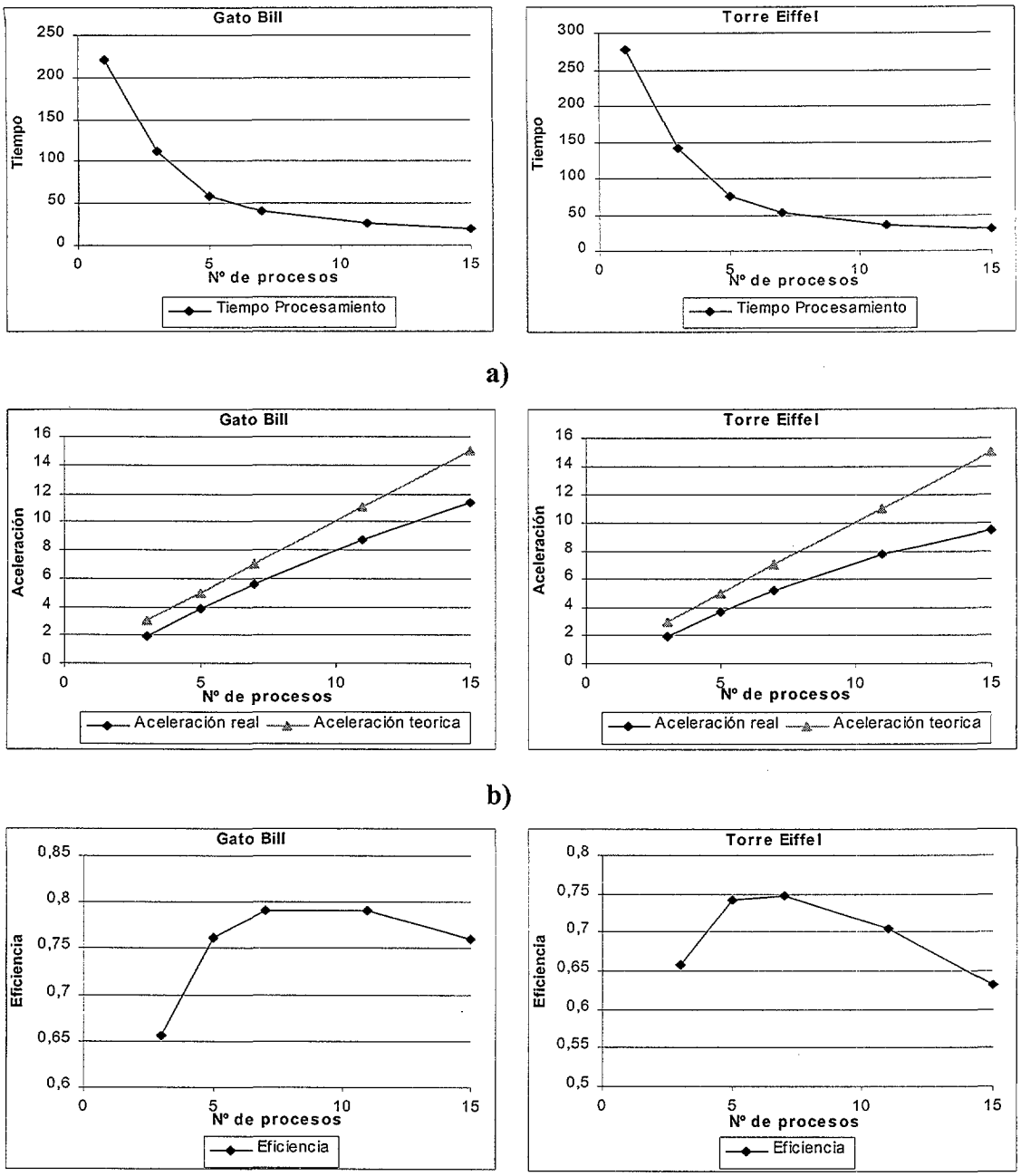



**Figura 6.3 Resultados obtenidos por la versión paralela para las escenas *Gato Bill* y *Torre Eiffel*. a) Tiempo total de procesamiento; b) Aceleración; c) Eficiencia.**

Tras indicar las variables necesarias para comprobar las características del algoritmo paralelo vamos a proceder a realizar el análisis de los resultados obtenidos. Como ya hemos indicado, dentro de las pruebas realizadas se han utilizado dos escenas (*Gato Bill* y *Torre Eiffel*) con diferente distribución espacial.

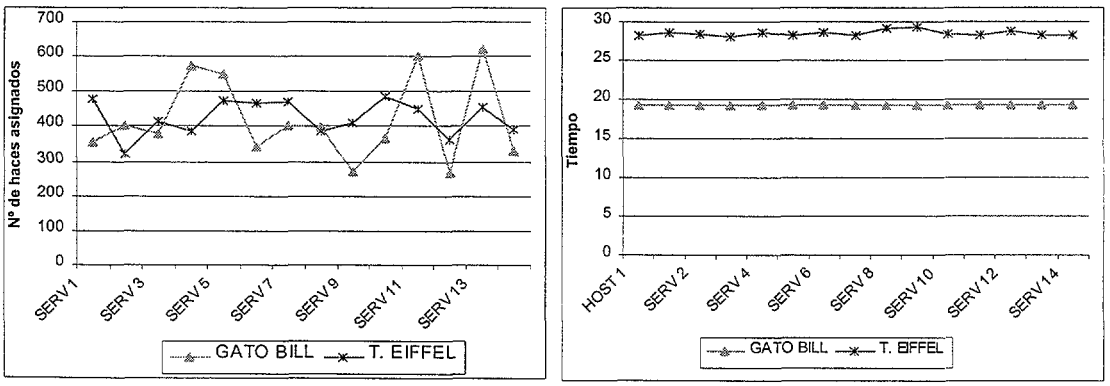
La primera tiene una figura en el centro de la escena, un gran polígono en la base y unas letras al fondo que hace que mucho rayos choquen con alguno de los objetos de la misma. La segunda en cambio tiene grandes zonas libres, pues los polígonos están todos fuertemente concentrados en el centro de la escena.

En la figura 6.3 se muestran los resultados obtenidos tanto para el tiempo total de procesamiento, como para las otras variables seleccionadas (*aceleración* y *eficiencia*). Vemos que el aumento en el número de procesadores de tipo *consumidor* implica, como era de esperar, una bajada en el tiempo total de procesamiento, pero, como podemos apreciar en la figura 6.3.a, dicha bajada no es totalmente proporcional al número de procesadores incorporados al sistema. La incorporación de los primeros procesadores hace que la reducción en el tiempo total de ejecución sea muy acusada, pero llega un momento en que la inclusión de otros nuevos casi no afecta a la disminución de dicho tiempo.

Esta situación puede observarse más claramente en el comportamiento de las otras dos variables. En el caso de la aceleración, aunque nunca se alcanza la *aceleración teórica*, al principio, cuando el número de procesadores es bajo, los valores obtenidos se aproximan más a ella, pero a partir de 7 procesadores comienzan a separarse del valor teórico.

En cualquier caso, esta tendencia se aprecia más claramente en las gráficas que representan la evolución de la eficiencia. La variación de la eficiencia muestra ciertas diferencias en el comportamiento de las dos escenas. En la primera escena, el período de incremento en la eficiencia es mayor y hasta que no se superan los once procesadores no cambia claramente la tendencia. En cualquier caso, para esta escena la eficiencia está siempre entorno al 0'75, es decir, el aprovechamiento de los procesadores es alto. En la segunda escena dicho cambio de tendencia se percibe antes, al superar los siete procesadores, y, a su vez, en este caso los valores de la eficiencia son algo menores, aunque siguen siendo superiores a 0'6 y, por tanto, el aprovechamiento de los procesadores todavía puede considerarse aceptable.

Estas variaciones en la tendencia de la aceleración y sobre todo en la eficiencia al incrementar el número de procesadores, son debidas, primeramente, a que al disponer de un mayor número de procesadores es más difícil realizar una correcta distribución de la carga. Al incrementarse el número de procesadores disponibles, la existencia de tareas más pesadas que otras puede hacer que a un procesador se le asigne una tarea muy costosa en tiempo que provoque que dicho procesador necesite tanto tiempo para ejecutarla que el resto terminen mucho antes que él. Por otra parte, al incrementarse el número de procesadores los efectos de la parte secuencial de todo algoritmo paralelo, enunciados por la ley de Amdahl, se aprecian mejor. En este caso, la disponibilidad de un gran número de procesadores provoca que el proceso *productor (host)*, encargado de realizar el reparto de las tareas entre los *consumidores (server)*, comience a tener un mayor peso en el tiempo total de procesado y, por tanto, condicione la evolución de las mejoras posteriores. Esta situación es más clara en la escena denominada "*Torre Eiffel*", ya que se genera un mayor número de haces, lo que motiva un aumento en el tiempo de trazado y provoca que el proceso *productor* (*HOST1*) necesite tanto tiempo como el resto para procesar sus tareas, haciendo que los demás procesos que dependen de él retrasen el instante en el que comienzan a ejecutar las tareas que les son asignadas.




**a)** **b)**

**Figura 6.4 Análisis de la distribución de la carga entre los diferentes procesadores (escenas Gato Bill y Torre Eiffel). a) Nº de haces asignados; b) Tiempo total de procesamiento.**

Si analizamos el reparto de trabajo realizado por nuestro algoritmo (figura 6.4.b), podemos observar que es bastante bueno. En este caso aunque el número de haces que atiende cada procesador no es homogéneo, como puede apreciarse en la figura 6.4.a, el instante de finalización de la última tarea asignada a cada uno de ellos es similar. Esto se debe básicamente a la utilización de un balanceo dinámico. Este tipo de distribución dinámica provoca que a cada procesador no se le asigne un nuevo haz hasta que no finalice de tratar el haz que se le envió previamente. Aquellos procesadores a los que se les envían haces que requieren un tiempo de procesamiento alto solicitan menos haces que aquellos otros a los que se les asignan haces menos costosos en tiempo. El desconocimiento a priori del número de haces a distribuir y de su coste hacen imposible, o poco útil, la utilización de técnicas de balanceo estático (descritas en el capítulo segundo), opciones que, sin embargo, obtienen excelentes resultados en la paralelización de otros algoritmos que realizan un trazado de rayos individuales. Por otra parte, la aplicación de una estrategia de balanceo dinámica, permite pensar que este algoritmo se adaptará correctamente a situaciones bastante habituales en las redes de estaciones de trabajo, como son la posibilidad de disponer de estaciones de trabajo heterogéneas (con diferente capacidad de procesamiento), o el hecho de que alguna de las estaciones esté siendo utilizada al mismo tiempo por otros usuarios y, por tanto, la capacidad disponible varíe en el tiempo.

Tras este estudio podemos ver que el algoritmo paralelo que se propone consigue una reducción de tiempo considerable en escenas con una gran cantidad de objetos y para las cuales deseamos obtener imágenes de gran calidad. Por otra parte hemos podido comprobar que, aunque no se consiguen las máximas cotas de paralelización (*aceleración lineal*), los valores obtenidos tanto para la aceleración como para la eficiencia son bastante buenos llegando a aprovechar los diferentes procesadores entorno al 75% o 65% de media. Finalmente hemos podido comprobar que la estrategia de balanceo seleccionada obtiene excelentes resultados, haciendo

que cada procesador analice un número diferente de haces pero que la carga que suponen de dichos haces sea homogénea.

# Capítulo 7. Conclusiones y trabajo futuro.

## 7.1 Introducción.

Hasta el momento hemos expuesto lo más relevante del trabajo desarrollado en esta Tesis Doctoral. Se han presentado las ideas más importantes del algoritmo de trazado de haces, que permiten apreciar las bases teóricas en las que se apoya. Por otra parte, se ha incluido una versión paralela de dicho algoritmo que ofrece la posibilidad de alcanzar cotas de aceleración todavía mayores, en escenas con un número considerable de objetos y una alta calidad de la imagen final. Todos estos algoritmos han sido validados mediante la aplicación a un conjunto de escenas (casos de prueba) lo que ha permitido analizar sus puntos fuertes, y a la vez ha facilitado la comparación del algoritmo aquí presentado con otras alternativas de aceleración propuestas previamente por diferentes autores.

En este último capítulo resumiremos las principales conclusiones que se pueden extraer del estudio realizado y presentaremos algunas de las líneas de trabajo que su elaboración nos ha sugerido para realizar en el futuro.

## 7.2 Conclusiones.

El problema central que se ha abordado en esta tesis consiste en el análisis, desarrollo e implementación de algoritmos que permitan acelerar el método clásico de generación de imágenes foto-realistas a través de las técnicas de trazado de rayos. Para ello, se ha pretendido estudiar los principales métodos de aceleración presentados hasta este momento y proponer una nueva alternativa que, teniendo presentes los beneficios esperados de la aplicación individual de cada propuesta, ofrezca además nuevas ventajas fruto de la unión, en una alternativa conjunta, de las diferentes propuestas individuales.

### 7.2.1 Estudio de trabajos de aceleración previos

En la primera parte de este trabajo se han explorado las diferentes alternativas presentadas hasta el momento, lo que nos ha permitido comprobar la gran cantidad de trabajos existentes sobre técnicas de aceleración del trazado de rayos. Hemos podido apreciar que la mayor parte de los trabajos se centran en la búsqueda de una estructura de descomposición de la escena adecuada y en algoritmos eficientes que la recorran, apoyándose para ello en la coherencia de objetos. Junto a ello se ha constatado el escaso número de trabajos que aprovechan la coherencia de rayos, incluso podemos encontrar algunos [SPEE86] que obtienen peores resultados que el algoritmo de trazado de rayos original. Esto es debido básicamente a la dificultad de definir y trazar estructuras más generales (haces, conos, lápices, etc.) que contengan a los rayos. A su vez, este estudio nos ha mostrado la escasez de trabajos que analicen el comportamiento conjunto de la *coherencia de objetos* y la *coherencia de*

*rayos*, y, por tanto, que aprovechen la estructura de descomposición de la escena para optimizar el trazado de los diferentes haces.

Por ello en esta tesis, se ha definido un nuevo algoritmo que permite conseguir las ventajas de la aplicación conjunta de los principales criterios de coherencia. En su definición se han utilizado como principales bases teóricas otros trabajos que abordan de manera individual la coherencia de objetos, mediante la utilización de árboles octales [SAME89], y la coherencia de rayos, mediante la definición y trazado de haces que contienen a un cierto número de rayos [HECK84]. Aunque la propuesta implementada en nuestro trabajo se restringe a la agrupación de rayos primarios en haces y, por tanto, deberíamos hablar de *RayCasting* en vez de *RayTracing*, si aplicamos las mismas restricciones impuestas por P. Heckbert y P. Hanrahan [HECK84], el algoritmo que proponemos podría adaptarse fácilmente a los rayos secundarios.

### 7.2.2 Selección de la metodología y entorno de desarrollo.

Tanto para la definición como para el desarrollo del software que implementa el trazado de haces se ha utilizado la Orientación a Objetos, se han aprovechado así al máximo las posibilidades que esta tecnología nos ofrece para la reutilización y extensibilidad del software.

La utilización de esta librería, suministrada por N. Wilt, denominada OORT [WILT94], nos ha permitido centrarnos en la definición e implementación de las clases propias del algoritmo que aquí proponemos, reutilizando el resto de clases necesarias para obtener finalmente la imagen de la escena. A su vez, el uso del mismo entorno de programación (OORT) para la implementación de los diferentes algoritmos, ha otorgado mayor significación a las pruebas realizadas pues se asegura que la implementación de dichos algoritmos comparte la misma base y, por tanto, las mismas posibles eficiencias e ineficiencias.

### 7.2.3 Análisis del comportamiento del trazado de haces.

En los tests de prueba realizados en el capítulo quinto, hemos podido apreciar que los beneficios obtenidos por nuestro algoritmo se derivan básicamente del trazado de haces que salen sin encontrar ningún nodo ocupado en su camino y, por tanto, los rayos en ellos contenidos nunca deben trazarse de modo individual dentro de la escena. Hemos comprobado que las ventajas del trazado de haces aumentan en escenas con grandes zonas libres que permiten la propagación de numerosos haces sin que estos encuentren nodos ocupados en su camino. Por ello, es interesante que en la creación de la estructura de descomposición se utilice básicamente el criterio de máximo nivel de descomposición, pues permite que los nodos ocupados sean de menor tamaño y, por tanto, se ajusten mejor a la silueta de los objetos de la escena, dejando mayor cantidad de espacios libres.

Por otra parte, el tiempo necesario para trazar los diferentes haces no depende de la resolución de la imagen, sino tan sólo de la estructura de descomposición utilizada, lo que permite que su coste se vea amortiguado conforme se incrementa la resolución de la imagen a obtener.

El hecho de que la generación de los rayos se posponga (se resuelve dentro del propio proceso de trazado) y no se realice antes de comenzar el trazado, ofrece también ciertas ventajas a la hora de aplicar algoritmos que reduzcan los problemas de aliasing. En este caso, en el instante de la generación de los rayos asociados a un pixel se dispone de información sobre el número de objetos con los que pueden chocar inicialmente los rayos (objetos contenidos en el nodo al que llega el haz) y la profundidad a la que se encuentran. Esta información puede utilizarse para generar un número diferente de rayos según las características de la escena y, por tanto, podemos considerar a nuestra propuesta de anti-aliasing como basada en el espacio de los objetos [GENE98].

## 7.2.4 Comparación con otras alternativas de aceleración.

Las pruebas realizadas sobre la versión secuencial del algoritmo de trazado de haces han permitido mostrar las bondades de éste frente a otras propuestas de trazado individual de rayos que utilizan tanto árboles octales como otras estructuras para la descomposición de la escena. Las primeras comprobaciones sobre algoritmos que utilizan también árboles octales como estructura de descomposición, pero que realizan un trazado individual [SAME89] [ENDL94], han permitido apreciar las ventajas del trazado de haces sobre este tipo de técnicas, en escenas de diferentes características. Estas pruebas han demostrado que, para todas las escenas, independientemente del número y distribución de los objetos, el trazado de haces obtiene mejores resultados que las técnicas de trazado individual de rayos y, que dichas mejoras se incrementan conforme lo hace la resolución o la calidad de imagen al aplicar técnicas de sobremuestreo.

Por otra parte, se ha comparado el trazado de haces sobre árboles octales con algoritmos de trazado individual que utilizan otras estructuras de descomposición (matrices de voxel [AMAN87] y jerarquías de volúmenes envolventes [GOLD87] [KAY86] [WILT94]). Estas pruebas nos han permitido apreciar que las características de la escena influyen decisivamente en el comportamiento de los diferentes algoritmos que realizan un trazado individual, lo que nos permite comprobar que no se puede considerar una estructura de descomposición mejor que otra. Pero, a su vez, dichas pruebas nos llevan a concluir que el trazado de haces reduce los problemas derivados de una inadecuada estructura de descomposición y obtiene mejores resultados que otras técnicas que utilizan estructuras más adaptadas a las características de dichas escenas.

### 7.2.5 Análisis de la propuesta de paralelización del trazado de haces.

Como hemos indicado en varias ocasiones, cualquier idea de aceleración que pretenda obtener altas cotas de optimización debe apoyarse en la explotación del paralelismo que el trazado de rayos lleva consigo. Por ello, hemos presentado una propuesta de paralelización del trazado de haces que realiza una distribución dinámica del *espacio de la imagen*. Se ha considerado un esquema *productor-consumidor* o también denominado *trabajadores replicados* [LEST93]. Las tareas se generan de manera dinámica y el tiempo de procesado de cada una de ellas es diferente, lo que invalida la utilización de balanceo estático y obliga a utilizar un balanceo dinámico. Este tipo de balanceo permite que el algoritmo pueda utilizarse en entornos con procesadores heterogéneos y previamente cargados. A su vez, la distribución de los pixel agrupados en haces permite que no se pierda toda la coherencia existente entre los rayos vecinos, algo muy difícil de conseguir con técnicas de distribución estática. De este modo se pueden aprovechar las optimizaciones que producen técnicas como el "*shadow cache*", que se basa en el hecho de que rayos de sombra generados por rayos vecinos chocarán previsiblemente con los mismos objetos de la escena, facilitando la determinación de posibles objetos opacos en el camino de un rayo de sombra hacia una fuente de luz.

Para disminuir las comunicaciones tan sólo se distribuyen las tareas teóricamente más costosas, evitando que se envíen aquellas cuyo coste de comunicación sea mayor que el que supone el procesado. Por otra parte para incrementar el solapamiento entre comunicación y procesado, el proceso que hemos denominado *productor* (*host*) realiza comunicaciones no bloqueantes que permiten comenzar el proceso de comunicación y mientras éste se completa seguir tratando otros haces. Con este algoritmo hemos conseguido reducir sensiblemente el tiempo

de ejecución, a la vez que hemos obtenido un correcto balanceo de la carga y cotas más que aceptables de aceleración y eficiencia.

## 7.3 Líneas de trabajo futuro.

Como consecuencia de los estudios desarrollados dentro de esta Tesis Doctoral, han surgido ideas que podrían explorarse en futuros trabajos y que contribuirán a aumentar el ámbito de aplicación del algoritmo propuesto y tal vez a obtener mayores cotas de optimización. A continuación se incluyen varias de estas líneas de trabajo.

- **Ampliar el tipo de haces a trazar.**

  Incluyendo las mismas restricciones impuestas por P. Heckbert y P. Hanrahan [HECK84], ampliar la aplicación del trazado de haces a rayos reflejados y transmitidos.

- **Rediseñar el algoritmo paralelo desde la perspectiva de *subdivisión de la escena.***

  Diseñar un nuevo algoritmo que posibilite procesar escenas con tal cantidad de objetos que su definición no quepa en la memoria de una de las estaciones de trabajo. En este caso la división de la escena podría dar lugar a una división de la tarea de trazar haces no terminales ya que cada procesador sólo debería trazar los haces correspondientes a su parcela de la escena. A su vez, en este caso, se produciría un aumento notable en las comunicaciones que debería ser analizado.

- **Estudiar posibles optimizaciones de las técnicas de anti-aliasing.**

  El hecho de disponer de información sobre la escena en el instante de generación de los diferentes rayos a trazar, abre nuevas alternativas dentro de las técnicas tradicionales de anti-aliasing.

- **Adaptar la técnica de trazado de haces a otras estructuras de descomposición.**

  Las ventajas obtenidas en el trazado de haces sobre árboles octales podrían ser aplicables a otras estructuras, como por ejemplo jerarquías de matrices de voxel en las que los haces podrían servir para recorrer de manera más eficiente la matriz del primer nivel de la jerarquía.

- **Adaptar la técnica de trazado de haces a otros campos.**

  Ver las adaptaciones necesarias y analizar cuál es el comportamiento del trazado de haces en el campo de la *visualización de volumen* ("*Volume Rendering*") dentro de la visualización de estructuras anatómicas en medicina, ya que en este campo una de las técnicas utilizadas hasta el momento es el algoritmo de trazado de rayos individuales sobre árboles octales.

# Capítulo 8. Bibliografía.

En este capítulo junto a la descripción completa de las referencias bibliográficas a las que se hace referencia en el texto, se incluyen algunas direcciones de Internet que pueden ser de gran utilidad para aquellas personas interesadas en el algoritmo de trazado de rayos.

## *8.1 Direcciones www de interés.*

http://liinwww.ira.uka.de/bibliography/Graphics/index.html

Contiene numerosas referencias sobre gráficos en general y trazado de rayos en particular.

ftp://ftp.eye.com

http://liinwww.ira.uka.de/bibliography/Graphics/ray.html

Fichero con más de 700 referencias recogidas y actualizadas hasta 1998 por E. Haines.

http://www.cm.cf.ac.uk/Ray.Tracing/

http://www.cm.cf.ac.uk/Ray.Tracing/RT.Bibligraphy.html

Contiene gran cantidad de información sobre el algoritmo de trazado de rayos y numerosos enlaces a otras páginas donde puede encontrarse más información. Dentro existe un subapartado que contiene mas de 400 referencias sobre este algoritmo.

http://www.acm.org/tog/resources/RTNews/html/rtn_index.html

http://povray.org/rtn/rtn

Incluye toda la información referente a una revista electrónica especializada en el trazado de rayos ("*Ray Tracing News*"), editada por E. Haines.

ftp://ftp.princeton.edu/pub/Graphics/Papers/speer.raytrace.bib.ps.Z

Fichero generado por R. Speer con bastantes referencias bibliográficas ordenadas por temas.

http://www.acm.org/pubs/tog/Software.html

Información sobre gran cantidad de programas sobre gráficos disponibles en la red, entre ellos programas que implementan el algoritmo de trazado de rayos.

## 8.2 Referencias bibliográficas.

[AIRE89] AIREY, J.M., OUH-YOUNG, M. Two Adaptative Techniques Let Progressive Radiosity Outperform the Traditional Radiosity Algorithm". *Tech. Report TR89-020, University of North Carolina Department of Computer Science*, 1989.

[ALFA98] ALFARO, F. J., GONZALEZ, P., SANCHEZ, J.L. "Trazado de Rayos en Redes de Estaciones de Trabajo". *Proc. Jornadas de Informática Gráfica, JIG'98*. Granada, Septiembre, 1998, pp. 131-140.

[AMAN84] AMANATIDES, J. "ray tracing with cones". *Computer Graphics*, Vol. 18, No. 3, July, 1984, pp.129-135.

[AMAN87] AMANATIDES, J.; WOO, A.; "A fast voxel traversal algorithm for ray tracing". *EUROGRAPHICS '87,* MARECHAL, G. (ed.), 1987, pp. 3-10.

[AMDA67] AMDAHL, G. "Validity of the single-processor approach to achieving large scale computing capabilities". *AFIPS Conference Procc.*, Vol.30, April 1967, pp. 483-485.

[ANDE95] ANDERSON,T.E., CULLER,D.E., PATTERSON,D.A. "A Case for NOW (Networks of Workstations)". *IEEE Micro.* Vol. 15, No. 1, February 1995, pp.54-64.

[AKTK87] AKTKIN, P., GHEE,S., PACKER, J. "Transputer Architectures for Ray Tracing". *Procc. of Computer Graphics '87*, 1987, pp. 157-172.

[APPE68] APPEL, A. "Some Techniques for Shading Machine Rendering of Solids". *Procc. of the Spring Joint Computer Conference*, 1968, pp. 37-45.

[ARVO87] ARVO, J. KIRK, D. "Fast Ray Tracing by Ray Classification". *Computer Graphics (SIGGRAPH'87).* Vol. 21, No. 4, July 1987, pp. 55-64.

[ARVO89] ARVO, J. KIRK, D. "A survey of Ray Tracing Acceleration Techniques". *Introduction to Ray Tracing,* GLASSNER A.S. (ed.) Academic Press, 1989, pp. 201-262.

[ARVO90a] ARVO, J., KIRK, D. "Particle Transport and Image Synthesis". *Computer Graphics (Siggraph'90),* Vol. 24, No. 4, 1990, pp. 63-66.

[ARVO90b] ARVO, J. "A simple method for box-sphere intersection testing". *Graphics Gems I.* GLASSNER, A. (ed) Academic Press, 1990, pp. 335-339.

[ARVO93] ARVO, J. "Transfer equations in global illumination". *Siggraph 1993 Global Illumination course notes,* chapter 1, HECKBERT, P (ed.) ACM Siggraph, August 1993.

[AYKA93] AYKANAT, C., OZGÜC B. "An Efficient Parallel Spatial Subdivision Algorithm for Parallel Ray Tracing Complex Scenes". *First Bilkent Computer Graphics Conference*, ATARV-93, Ankara, July1993.

[AYKA94] AYKANAT, C., ISLER, V., OZGÜC B. "Efficient Parallel Spatial Subdivision Algorithm for Object-Based Parallel Ray Tracing". *Computer-Aided Design.* Vol. 26, No. 12, Dec. 1994, pp.883-890.

[BADO89] BADOUEL, D., PRIOL,T. "An Efficient Parallel Ray Tracing Scheme for Highly Parallel". *Proc. Eurographics Hardware Workshop*, Springer-Verlag, berlin, 1989, pp. 93-106.

[BADO90] BADOUEL, D., BOUATOUCH, K., PRIOL,T. "Ray Tracing on distributed Memory Parallel Computers: Strategies for Distributing Computations and Data". *In Parallel Algorithms and Architectures for 3D Image Generation*. S. Whitman (ed). *ACM Siggraph'90 Course 28*, Aug. 1990, pp.185-198.

[BADO94] BADOUEL, D., BOUATOUCH, K., PRIOL,T. "Distributing Data and Control for Ray Tracing in Parallel". *IEEE Computer Graphics and Applications*, Vol. 14, July 1994, pp. 69-77.

[BADT88] BADT,S. "Two algorithms for taking advantage of temporal coherence in ray tracing". *The Visual Computer*, Vol. 4, 1988, pp. 123-132.

[BART98] BARTZ, D., SILVA, C., SCHNEIDER, B. "Rendering and Visualization in Affordable Parallel Environments". *Tutorial, EUROGRAPHICS'98*. Lisboa 1998.

[BAUM91] BAUM, D., WINGET, J., MANN, S., SMITH, K. "Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for Generation of Accurate Radiosity Solutions" *Computer Graphics (SIGGRAPH'91)*. Vol. 25, No. 4, August 1991, pp. 51-60.

[BERG86] BERGMAN, L., FUCHS, H., GRANT, E., SPACH, S. "Image Rendering by Adaptative Refinement". SIGGRAPH 86, July 1986, pp.29-37.

[BODE95] BODEN,N., COHEN,D., FELDERMAN,R., KULAWIK, A., SEITZ,C., SEIZOVIC, J., SU,W. "Myrinet: A Gigabit-per-second Local Area Network". *IEEE Micro*. Vol. 15, No. 1, February 1995, pp. 29-36.

[BOUA88] BOUATOUCH, K., PRIOL, T. "Parallel Space Tracing: An Experience on an iPSC Hypercube". *New Trends in Computer Graphics (Proceedings of CG International '88)*. Springer-Verlag, 1988, pp. 170-87.

[BOUA89] BOUATOUCH, K., SAOUTER, Y., CANDELA, J.C. "A VLSI Chip for Ray Tracing Bicubic Patches". *Proc. of Eurographics'89*, W. Hansmann, F.R.A. Hopgood y W. Strasser (eds.), 1989, pp.107-124.

[CAMA95] CAMAHORT, E., QUINTANA, E., QUINTANA, G. "Sobremuestreo y paralelización; dos formas de mejorar el Trazado de Rayos". *Novatica*, nº 118, 1995, pp. 110-117.

[CART90] CARTER, M., TEAGUE, K. "Distributed Object Database Ray Tracing on the Intel iPSC/2 Hypercube". *Procc. of the 5th distributed Memory Computing Conference*. 1990, pp. 217-222.

[CASP89] CASPARY, P.E., SCHERSON, I.D. "A Self-Balanced Parallel Ray Tracing Algorithm". *Parallel Processing for Computer Vision and Display*, Dew, P.M., Heywood,T.R., Earnshaw R.A.(ed). Addison-Wesley, 1989, pp.408-419.

[CAUB88] CAUBET, R., DUTHEN, Y., GAILDRAT-INGUIMBERT, V. "VOXAR: A Tridimensional Architecture for Fast Realistic Image Synthesis". *Proc. Computer Graphics International*, Berlin 1988, pp. 135-149.

[CAZA95] CAZALS, F., DRETTAKIS, G., PUECH, C. "Filtering Clustering and Hierarchy Construction: a New Soluction for Ray-Tracing Complex Scenes". *EUROGRAPHICS'95*. Post,F., Göbel, M. (ed). Vol 14, No.3, 1995, pp.371-382.

[CHAL89] CHALMERS, M. "On the Design and Implementation of a Multiprocessor Ray Tracer". *Parallel Processing for Computer Vision and Display*, Addison Wesley, 1989. pp. 420-30.

[CHAR90] CHARNEY, M.J., SCHERSON, I.D. "Efficient Traversal of Well-Behaved Hierarchicial Trees of Extents for Ray-Tracing Complex Scenes". *The Visual Computer*, Vol. 9, 1990, pp.167-178.

[CHEN89] CHEN, S.E. "A Progressive Radiosity Method and its Implementation in a Disributed Processing Environment". *Master Thesis, Cornell University*, January 1989.

[CHEN90] CHEN, S.E. "Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System". *Computer Graphics (SIGGRAPH'90)*, Vol. 24, No. 4, 1990, pp. 134-144.

[CHEN91] CHEN, S.E., RUSHMEIER, H., MILLER, G., TURNER, D. "A Progressive Multi-Pass Method for Global Illumination". *Computer Graphics (SIGGRAPH'91)* , Vol. 25, No. 4, 1991,pp. 165-174.

[CLEA86] CLEARY, J.G., WYVILL, B., BIRTWISTLE, G., VATTI, R. "Multiprocessor Ray Tracing". *Computer Graphics Forum*. Vol. 5, 1986, pp.3-12.

[CLEA88] CLEARY, J.G., WYVILL,G. "Analysis of an algorithm for fast ray tracing using uniform space subdivision". *The Visual Computer*. Vol. 4, 1988, pp.65-83.

[COAD91a] COAD, P., YOURDON, E. *Object-Oriented Analysis*. Yourdon Press, Prentice-Hall. 1991.

[COAD91b] COAD, P., YOURDON, E. *Object-Oriented Design*. Yourdon Press, Prentice-Hall. 1991.

[COAD97] COAD, P., NOTH,D., MAYFIELD, M. *Object Models. Straategies, Patterns, & Applications*. Yourdon Press Computing Series, Prentice-Hall. 1997.

[COHE85] COHEN, M., GREENBERG, D.P. "The Hemi-cube: a Radiosity Solution for Complex Environments". *Computer Graphics*, Vol. 19, No. 3, 1985, pp. 31-41.

[COHE86] COHEN, M., GREENBERG, D.P., IMMEL,D.S., BROCK, P.J. "An Efficient Radiosity Approach for realistic Image Synthesis". *IEEE Computer Graphics & Applications*, Vol. 6, No. 3, March 1986, pp. 26-35.

[COHE93] COHEN, M., WALLACE, J.R. *Radiosity and Realistic Image Synthesis*. Academic Press, San Diego, 1993.

[COOK86] COOK, R.L. "Stochastic Sampling in Computer Graphics". *ACM Trans. On Graphics*, Vol.5, No. 1, January 1986, pp. 51-72.

[CUON97] CUONG, N.D. "An Exploration of Coherence-based Acceleration Methods Using the Ray Tracing Kernel G/GX". *TU-Dresden*, Germany, March 1997.

[DADO85] DADOUN, N., KIRKPATRICK, D.G. "The Geometry of Beam Tracing". *Proc. of the Symposium on Computational Geometry*, June 1985, pp. 55-61.

[DAVI98] DAVIS, T.A., DAVIS, E.W. "Rendering Computer Animations on Network of Workstations". *Proc of the IPPS/SPDP*, April 1998, pp. 726-730.

[DIPP84] DIPPÉ, M., SWENSEN, J. " An Adaptative Subdivision Algorithm and Parallel Architectture for Realistic Image Synthesis". *ACM Computer Graphics*. Vol.18, No. 3, July 1984, pp. 149-158.

[DIPP85] DIPPE, M., WOLD, E.H. "Antialiasing through stocastic sampling". *Computer Graphics*, Vol. 19, No. 3, 1995, pp. 69-78.

[DORB86] DORBAND,J. "Ray Tracing on the MPP". *Frontiers of Massively Parallel Scientific Computation, Proc, os the 1st Symposium at Goddard Space Flight Center, NASA,* Conference publication 2478, 1986, pp.211-215.

[DUTR95] DUTRE, P., WILLEMS, Y.D. "Potential-driven Monte Carlo Particle Tracing for Diffuse Environments with Adaptive Probability Density Functions". *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering),* Springer-Verlag, 1995, pp. 306-315.

[ENDL94] ENDL, R. SOMMER, M. "Classification of Ray-Generators in Uniform Subdivisions and Octrees for Ray Tracing". *Computer Graphics Forum*, Vol.13, No.1,1994, pp. 3-19.

[FOLE90] FOLEY, J.D., VAN DAM, A., FEINER, S.K., HUGHES, J.F. *Computer Graphics: Principles and Practice.* Addison-Wesley, MA, 2nd ed., 1990.

[FRÖH88] FRÖHLICH, B. JOHANNSEN, A. "Beschleunigung des Ray-Tracing Algorithmus: Drei Vorschläge zum verbesserten Einsatz der Octree-Raumteilung". *Informatik Fachberichte 183, Austrographics'88*, Springer- Verlag, 1988, pp.51-69.

[FUJI85] FUJIMOTO, A., IWATA, K. "Accelerated Ray Tracing". *Computer Graphics, Visual Technology and Art, Procc. Of Computer Graphics Tokyo'85.* Springer- Verlag, 1985, pp.41-65.

[FUJI86] FUJIMOTO, A., TANAK, T., IWATA, K. "ARTS: Accelerated Ray Tracing System". *IEEE Computer Graphics and Applications*, Vol. 6. No.4, April 1986, pp. 16-26.

[GARG82a] GARGANTINI, I. "Linear Octrees for Fast Processing of Three-Dimensional Objects". *Computer Graphics and Image Processing*, Vol.20, No. 4, 1982, pp.265-274.

[GARG82b] GARGANTINI, I. "An Effective Way to Store Quadtrees". *Comm. ACM*, Vol.25, 1982, pp. 905-910.

[GARG86] GARGANTINI, I., WALSH, T.R., WU, O.L. "Viewing Transformations of Voxel-Based Objects via Linear Octrees". *IEEE Computer Graphics & Applications*, Vol. 6, No. 10, 1986, pp. 12-21.

[GARG89] GARGANTINI, I., SCHRACK, G., ATKINSON, H.H. "Adaptive Display of Linear Octrees". *Computer & Graphics*, Vol. 13, No. 3, 1989, pp.337-343.

[GARG93] GARGANTINI, I., ATKINSON, H.H. "Ray Tracing an Octree: Numerical Evaluation of the First Intersection". *Computer Graphics Forum*, Vol.12, No. 4, 1993, pp.199-210.

[GAUD88] GAUDET, S., HOBSON, R., CHILKA, P. CALVERT, T. "Multiprocessor Experiments for Higj-Speed Ray Tracing". *ACM Transactions on Graphics*, Vol. 7, No. 3, July, 1988, pp.151-179.

[GENE98] GENETTI, J., GORGON, D., WILLIAMS, G. "Adaptive Supersampling in Object Space Using Pyramidal Rays". *Computer Graphics Forum.* Vol. 17, No. 1, 1998, pp. 29-54.

[GERM93] GERMAIN-RENAUD, C., SANSONNET, J.P. *Ordenadores Masivamente Paralelos.* Editorial Paraninfo, 1993.

[GLAS84] GLASSNER, A.S. "Space subdivision for fast ray tracing". *IEEE Computer Graphics & Applications,* Vol. 4, No.10, October 1984, pp.15-22.

[GLAS88] GLASSNER A.S. "Spacetime ray tracing for animation". *IEEE Computer Graphics & Applications,* Vol. 8, No. 3, March 1988, pp. 60-70.

[GLAS89] GLASSNER A.S. *Introduction to Ray Tracing.* Academic Press, 1989.

[GLAS95] GLASSNER, A.S. *Principles of Digital Image Synthesis.* Morgan Kaufmann Publishers, San Francisco,California, 1995.

[GOLD85] GOLDSMITH, J., SALMON, J. "A ray tracing system for hypercube. *Caltech Concurrent Computing Project Memorandum HM154,* California Institute of Technology, 1985.

[GOLD87] GOLDSMITH, J. SALMON, J. "Automatic creation of object hierarchies for ray tracing". *IEEE Computer Graphics & Applications,* Vol.7, No. 5, 1987, pp. 14-20.

[GONZ94a] GONZÁLEZ, P."Una versión paralela del algoritmo trazador de rayos para la generación de imágenes fotorealistas". *Nuevas Tendencias en la Informática: Arquitecturas Paralelas y Programación Declarativa.* FERNANADEZ, M.A. et al. (ed.) Colección estudios Univ. Castilla-La Mancha, 1994, pp. 201-217.

[GONZ94b] GONZÁLEZ, P. "La reutilización: un camino hacia la industrialización del desarrollo del software". *Ensayos.* Univ Castilla-La Mancha, Diciembre 1994, pp.267-281.

[GONZ98a] GONZÁLEZ, P., GISBERT, F. "Trazado de haces de Rayos en Escenas Estructuradas Espacialmente Mediante Árboles Octales". *Proc of the 8EPCG,* Coimbra, Febrero 1998, pp.201-214.

[GONZ98b] GONZÁLEZ, P., GISBERT, F."Object and Ray Coherence in the Acceleration of the Ray Tracing Algorithm". *Proc. of the Computer Graphics International, CGI'98,* IEEE Computer Society Press, Hannover, Junio 1998, pp.264-267.

[GONZ98c] GONZÁLEZ, P., GISBERT, F., GARCÍA-CONSUEGRA, J. "Tratamiento de gráficos en 2D". *Informática Gráfica,* Colección Ciencia y Técnica. Univ. Castilla-La Mancha, Junio 1998, pp.43-70.

[GONZ99] GONZÁLEZ, P., ALFARO, F.J., SÁNCHEZ, J.L. "An Efficient Load Balancing Method for Parallel Ray Tracing on Heterogeneous Workstation Networks". A publicar en *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99).* Las Vegas, USA, Junio 1999.

[GORA84] GORAL, C.M., TORRANCE,K.E., GREENBERG,D.P., BATTAILE, B. "Modeling the Interaction of Light Between Diffuse Surfaces". *Computer Graphics (Procc. Siggraph'84).* Vol. 18, No. 3, July 1984, pp. 213-222.

[GREE89] GREEN, S.A. PADDON, D.J. "Exploiting Coherence for Multiprocessor Ray Tracing". *IEEE Computer Graphics & Applications,* Vol.4, No. 10, 1989, pp. 12-26.

[GREE90] GREEN, S.A. PADDON, D.J. "A Highly Flexible Multiprocessor Solution for Ray Tracing". *The Visual Computer*, Vol. 6, 1990, pp. 62-73.

[GREE95] GREEN, D., HATCH, D. "Fast Polygon-Cube Intersection Testing". *Graphics Gems V,* PAETH,A. W. (ed.), AP Profesional, Boston, 1995, pp. 375-379.

[GRIM94] GRIMSTEAD, I.J., HURLEY, S. "Animation Using Accelerated Ray Tracing on a Hypercube". *IEEE Colloquium on High Performance Applications of Parallel Architectures,* Feb. 1994, pp. 2/1-2/4.

[GRÖL91] GRÖLLER, E., PURGATHOFER, W. "Using temporal and spatial coherence for accelerating the calculation of animation sequences". EUROGRAPHICS'91, Post, F.H., Barth, W. (ed.), 1991, pp.103-113.

[GROP96] GROPP, W., LUSK,E., SKJELLUM, A. *Using MPI. Portable Parallel Programming with the Message-Passing Interface.* The MIT Press, 1996.

[GUST88] GUSTAFSON, A.N. "Reevaluating Amdahl's law". *Comm. of the ACM, Technical Note*, Vol. 31, N°. 5, May 1988, pp. 532-533.

[HAGE86] HAGEN, M. *Varieties of Realism.* Cambridge University Press, Cambridge, England, 1986.

[HAIM98] HAIMES, R., JORDAN, K.E. "Using PVM and MPI for Co-processed Distributed and Parallel Scientific Visualization". *Parallel Distributed Processing* ROLIM, J. (ed.) Lectures Notes in Computer Science, Springer 1998, pp. 1098-1105.

[HAIN86] HAINES, E., GREENBERG, D. " The light buffer: a shadow-testing accelerator". *IEEE Computer Graphics & Applications*, Vol. 6, No. 9, Sep 1986, pp. 6-16.

[HAIN87] HAINES, E. "A Proposal for Standard Graphics Environment". *IEEE Computer Graphics & Applications*, Vol. 7, No. 11,Nov 1987, pp.3-5.

[HEBE90] HEBERT, M., McNEILL, M.D.,SHAH,B.C., GRIMSDALE,R.L., LISTER, P.F. "MARTI- a Multiprocessor Architecture for Ray Tracing Images". *Proc. of Advances in Computer Graphics Hardware V*, 1990.

[HECK84] HECKBERT, P., HANRAHAN, P. "Beam Tracing Polygonal Objects". *Computer Graphics*, Vol. 18, No.3, 1984, pp.119-127.

[HEIR97] HEIRICH, A., ARVO, J. "Scalable monte carlo image syntesis". *Parallel Computing,* Vol. 23, No. 7, July 1997, pp. 845-859.

[HEIR98] HEIRICH, A., ARVO, J. "A Competitive Analysis of Load Balancing Strategies for Parallel Ray Tracing". *The Journal of Supercomputing,* Vol. 12, No.1/2, January 1998, pp. 57-68.

[HOFM90] HOFMANN, G.R. "Who Invented Ray Tracing?". *The Visual Computer.* Vol.6, 1990, pp.120-124.

[HU85] HU, M., FOLEY,J.D. "Parallel Processing Approaches to Hidden-Surface Removal in Image Space". *Computer & Graphics*, Vol. 9, No.3, 1985, pp.303-317.

[HWAN88] HWANG, K., BRIGGS, F.A. *Arquitectura de Computadoras y Procesamiento Paralelo.* McGraw-Hill, Mexico,1988.

[IMME86] IMMEL, D.S., COHEN, M.F., GREENBERG, D.P. "A Radiosity Methods for Non-Diffuse Environments". Siggraph'86, 1986, pp. 133-142.

[ISLE91] ISLER, V. ÖZGÜC, B. "Fast Ray Tracing 3D Models". *Computer & Graphics.* Vol.15, No. 2, 1991, pp.205-216.

[JANS86] JANSEN, F.W. "Data Structures for Ray Tracing". En *Data Structures for Raster Graphics.* KESSENER, L., PETERS, F., LIEROP . M. (ed.) Springer-Verlag,1986, pp.57-73.

[JANS93] JANSEN, F.W., CHALMERS, A. "Realism in Real Time?". *Fourth Eurographics Workshop on Rendering,* Paris , France, June 1993, pp. 27-46.

[JENS89] JENSEN, D., REED, D. "Ray Tracing on Distributed Memory Parallel Systems". *Tech. Report UIUCDCS-R-89-1551, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign,* October 1989.

[JEVA89] JEVANS, D, WYVILL, B. "Adaptive voxel subdivision for ray tracing". *Proc. of Graphics Interface'89*, June 1989, pp.164-172.

[JEVA92] JEVANS,D.A. "Object Space Temporal Coherence for Ray Tracing". *Graphics Interface'92*, 1992, pp.176-183.

[KAJI86] KAJIYA, J.T. "The Rendering Equation". *Computer Graphics*, Vol.20, No.4, 1986, pp.143-150.

[KAPL85] KAPLAN, M.R. "Space tracing a constant time ray tracer". *State of the Art in Image Synthesis. SIGGRAPH '85*, Vol.11, 1985.

[KAPL87] KAPLAN, M.R. "The Use of Spatial Coherence in Ray Tracing". *Techniques for Computer Graphics*, Springer Verlag, 1987, pp. 173-193.

[KAY86] KAY, T.; KAJIYA, J.; "Raytracing complex scenes". *Computer Graphics*, Vol. 20,No. 4, 1986, pp. 269-278.

[KEAT94] KEATES, M.J., HUBBOLD, R.J."Accelerated Ray Tracing on the KSR1 Virtual Shared-Memory Parallel Computer". *Tech. Report UMCS-94-2-2. Computer Science, Univ. of Manchester*, 1994.

[KIRK87] KIRK, D., ARVO,J. "The Simulation of Natural Features Using Cone Tracing". *The Visual Computer.* Vol. 3, No. 2, 1987, pp. 63-71.

[KLIM97] KLIMASZEWSKI, K.S., SEDERBERG, T.W. "Fast Ray Tracing Using Adaptive Grids". *IEEE Computer Graphics & Applications,* January-February 1997, pp. 42-51.

[KOBA87] KOBAYASHI, H., NAKAMURA, T., SHIGEI, Y. "Parallel processing of an object space for image synthesis using ray tracing". *The Visual Computer.* Springer-Verlag, Vol. 3, No. 1, 1987, pp.13-22.

[KOBA88a] KOBAYASHI, H., NISHIMURA, S., KUBOTA, H., NAKAMURA, T., SHIGEI, Y. "Load balancing strategies for parallel ray tracing system based on constant division". *The Visual Computer.* Springer-Verlag, Vol. 4, 1988, pp.197-209.

[KOBA88b] KOBAYASHI, H., NAKAMURA, T., SHIGEI, Y. "A Strategy for Mapping Paralle Ray-Tracing into Hypercube Multiprocessor System". *Proc Computer Graphics International*, Springer-Vrelag, Berlin, 1988, pp. 160-169.

[KOK91] KOK, A.J.F., JANSEN, F.W. "Source Selection for the Direct Lighting Computation in Global Illumination". *Proc. Of the 2nd Eurographics Workshop on Rendering.*

[KORN87] KORNGOLD, P., ELAINE, V. "Optimizing Ray Tracing Performance with Object and Spatial Coherence". *TR-87047, Resensselaer Polytechnic Institute*, Troy NY, August 1987.

[KUZM94] KUZMIN, Y.P. "Ray Traversal of Spatial Structures". *Computer Graphics Forum.* Vol.13, No. 4, 1994, pp. 223-227.

[LANG98] LANGENDOEN, K., HOFMAN, R., BAL, H. "Challenging Applications on Fast Networks". *Fourth International Symposium on High-*

*Performance Computer Architectures. IEEE Computer Society*, Las Vegas, Nevada, January 1998, pp. 68-79.

[LAST98] LASTRA, M., REVELLES, J. "Métodos Híbridos para la Mejora del test de Intersección Rayo-Escena". *Proc. VIII Congreso Español de Informática Gráfica.* Ourense, 1998, pp. 191-200.

[LEE85] LEE, M.E., REDNER, R.A., UPSELTON, S.P. "Statistically Optimized Sampling for Distributed Ray Tracing" *SIGGRAPH'85, Computer Graphics*, Vol.19, No. 3, 1985, pp. 61-67.

[LEFE93] LEFER,W. "An Efficient Parallel Ray Tracing Scheme for Distributed Memory Parallel Computers". Procc. of IEEE Visualization'93 Parallel Rendering Symposium. ACM Siggraph, October 1993, pp. 77-80.

[LEST93] LESTER, B.P. *The Art of Parallel Programming*. Prentice-Hall Inter., 1993.

[LI89] LI, K. SCHAEFER, R. "A hypercube shared virtual memory system". *International Conference on Parallel Processing*, 1989, pp. 125-132.

[LIN91] LIN, T. , SLATER, M. "Stochastic Ray Tracing Using SIMD Processor Arrays". *The Visual Computer*, Vol. 7, July 1991, pp. 187-199.

[MACD90] MACDONALD, J.D., BOOTH, K.S. "Heuristics for ray tracing using space subdivision". *The Visual Computer*, Vol. 6, No. 3, 1990, pp. 153-166.

[MARI94] MARINI, D., CANESI, A., GATTI, C., ROSSI, M. "Parallelising Accelerated Ray Tracing for High Quality Image Synthesis". *Transputer Applications and Systems'94. IOS Press,* 1994, pp. 40-53.

[MAX81] MAX, N.L. "Vectorized procedural models for natural terrain:waves and islands in the sunset". *Computer Graphics,* Vol. 15, No.3,August 1981, pp. 317-324.

[MCNE92] McNEILL, M.D., SHAH, B.C., HEBERT, M., LISTER, P.F., GRIMSDALE, R.L. "Performance of Space Subdivision Techniques in Ray Tracing". *Computer Graphics Forum*, Vol. 11, No. 4, 1992, pp. 213-220.

[MOLI98] MOLINA, J.P., GONZALEZ, P. "Aceleración del trazado de rayos mediante matriz de voxels" *Tech. Report DIAB-98-02-03. Dpto. Informática, Univ. Castilla-La Mancha,* 1998.

[MOSH85] MOSHELL, J.M., CULLIP, T.J. "An Array Processor for Ray Tracing". *Department of Computer Science, Univ. of Central Florida, Orlando,* November 1985.

[MÜLL88] MÜLLER, H. "Realistische Computergraphik". *Informatik Fachberichte*, Springer Verlag, 1988, pp.37-41.

[MURA90] MURAKAMI, K., HIROTA, K. "Incremental Ray Tracing". *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, June 1990, pp. 15-29.

[NARU87] NARUSE et al. "Sihgt: A Dedicated Computer Graphics Machine". *Computer Graphics Forum*, Vol. 6, No. 4, Dec. 1987, pp. 327-334.

[NEMO86] NEMOTO, K., OMACHI, T. "An Adaptive Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing". Proc Graphics Interface, Computer Graphics Soc. Montreal, 1986, pp.43-48.

[NISH83] NISHIMURA, H. et al. "LINKS-1: a parallel pipelined multimicrocomputer system for image creation". *Proc. of the 10th Symposium on Computer Architecture, SIGARCH*, 1983, pp.387-394.

[NISH85] NISHITA, T., NAKAMAE, E. "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and interreflection". *Computer Graphics (Procc. Siggraph'85)*. Vol. 19, No. 3, July 1985, pp. 23-30.

[OHTA87] OHTA, M., MAEKAWA, M. "Ray Coherence Theorem and Constant Time Ray Tracing Algoritm". *Computer Graphics*, 1887, pp. 303-314.

[OHTA90] OHTA, M., MAEKAWA, M. "Ray-bound tracing for perfect and efficient anti-aliasing". *The Visual Computer*, Vol. 6, No. 3, 1990, pp. 125-133.

[PACH97] PACHECO, P. S. *Parallel Programming with MPI.* Morgan Kaufmann Publishers, 1997.

[PADD92] PADDON, P., DEREK, J., CHALMERS, A. "Parallel Processing for Rendering". *Third Eurographics Workshop on Rendering.* Bristol, UK, May 1992, pp. 1-8.

[PATT92] PATTANAIK, M.J., MUDUR, S.P. "Computation of Global Illumination by Monte carlo Simulation of the Particle Model of Light". CHALMERS, A., PADDON, G., SILLION, F. (ed.). *Third Eurographics Workshop on Rendering.* Elsevier, Amsterdam, 1992, pp. 71-83.

[PATT93] PATTANAIK, M.J., MUDUR, S.P. "The Potential Equation and Importance in Illumination Computations". *Computer Graphics Forum*, Vol.12, No. 2, 1993, pp.131-136.

[PLUN85] PLUNKETT, D.J., BAILEY, M.J. "The vectorization of a ray tracing algorithm for improved execution speed", *IEEE Computer Graphics and Applications,* Vol. 5, No.8, August 1985, pp 52-60.

[POTM89] POTMESIL, M., HOFFERT, E.M. "The Pixel Machine: A Parallel Image Computer". *Computer Graphics, Proc. of Siggraph,* Vol. 23, No. 3, July 1989, pp.69-78.

[PRIO89] PRIOL, T., BOUATOUCH, K. "Static Load Balancing for a Parallel Ray Tracing on a MIMD Hypercube". *The Visual Computer.* Vol. 4, No. 12, March 1989, pp.109-119.

[PULL87] PULLEYBLANK, R., KAPENGA, J. "The Feasibility of a Vlsi Chip for Ray Tracing Bicubic Patches". *IEEE Computer Graphics and Applications,* March 1987, pp.33-44.

[RAMO90] RAMOS, R. *Acelerador del Algoritmo Trazador de Rayos por Métodos Incrementales, para la Síntesis de Imágenes en Tres Dimensiones.* Tesis Doctoral. Universidad Politécnica de Madrid, Facultad de Informática. 1990.

[RATS94] RATSCHEK, H., ROKNE, J. "Box-Sphere intersection test". *Computer-Aided Design.* Vol. 16, No. 7, July 1994, pp. 579-584;

[REIN97] REINHARD, E., CHALMERS, A. "Message Handling in Parallel Radiance". *Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proc 4th European PVM/MPI Users' Group Meeting,* BUBAK, M., DONGARRA, J., WASNIEWSKI, J. (ed.) Lecture Notes in Computer Science. Cracow, Poland, Nov. 1997, pp.486-493.

[REIN98] REINHARD, E., CHALMERS, A., JANSEN, F. "Overview of Parallel Photo-realistic Graphics". *State of The Art Report, EUROGRAPHICS'98.* Lisboa 1998, pp 1-25.

[ROTH89] ROTHE, C. "Ein Vektorgenerator fur hierarchische Raumraster". CAD und Computergraphik, Vol 12, No. 5, 1989, pp. 139-150.

[ROTH91] ROTHE, C. "Untersuchung von Raumrastern zur Beschleunigung der Strahlanfrage in der photo-realistischen Bildsynthese". Universität Karlsruhe, 1991.

[RUBI80] RUBIN, S., WHITTED, T. "A three-dimensional representation for fast redering of complex scenes". *Computer Graphics,* Vol.14, No. 3, July 1980, pp.110-116.

[RUMB96] RUMBAUGH,J., BLAHA, M., PREMERLANI, W., EDDY, F., LORENSEN, W. *Modelado y diseño orientados a objetos.* Prentice Hall, 1996.

[RUSH86] RUSHMEIER, H.E. *Exending the Radiosity Methods to Transmitting and Specularly Reflecting Surfaces.* M.S. Thesis, Mechanical Engineering Department, Cornell University, Ithaca, 1986.

[RUSH87] RUSHMEIER, H.E, TORRANCE, K. "The Zonal Methods for Calculating Light Intensities in the Presence of a Participating Medium". *Siggraph'87*, 1987, pp. 293-302.

[SALM88] SALMON, J., GOLDSMITH,J. "A Hypercube Ray-Tracer". *Proc. Third Conf. Hypercube Concurrent Computers and Applications,* Vol.2, Applications,ACM Press, New York , 1988, pp. 1194-1206.

[SAME84] SAMET, H. " The Quadtree and Related Hierarchical Data Structures". *ACM Computing Surveys*, Vol. 16, No. 12, June 1984, pp.187-260.

[SAME89] SAMET, H. "Implementing Ray Tracing with Octrees and Neighbor Finding". *Computer & Graphics*, Vol. 13, No. 4, 1989, pp. 445-460.

[SAME90a] SAMET, H. *The Design and Analisys of Spatial Data Structures.* Computer Series, Addison Wesley, 1994

[SAME90b] SAMET, H. *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS.* Computer Series, Addison Wesley, 1990.

[SAND85] SANDOR, J. "Octree Data Structures and Perpective Imagery". *Computer & Graphics*, Vol. 9, No. 4, 1985, pp. 393-405.

[SCHE87] SCHERSON, I.D., CASPARY, E. "Data Structures and the Complexity of Ray Tracing". *The Visual Computer*, Vol. 3, 1987, pp.201-213.

[SHIN87] SHINYA, M., TAKAHASHI, T., NAITO, S. "Principles and Applications of Pencil Tracing". *Computer Graphics*, Vol. 21, No. 4, July 1987, pp. 45-54.

[SHIR90] SHIRLEY, P. "A Ray Tracing Method for Illumination Calculation in Diffese Specular Scenes". *Proc. Computer Graphics Interface'90*, 1990, pp.205-212.

[SHIR91] SHIRLEY, P., WANG, C. "Direct Lighting Calculation by Monte Carlo Integration". *Procc. of the Second Eurographics Workshop on Rendering*, Barcelons 1991.

[SIEG92] SIEGEL, R., HOWELL, J.R. *Thermal Radiation Heat Transfer.* Hemisphere Publishing, Bristol, PA, 3rd ed., 1992.

[SILL89] SILLION, F., PUECH, C. "A General Two Pass Methods Integrating Specular and Diffuse Reflection". *Computer Graphics (Siggraph'89)*, Vol. 23, No. 3, 1989, pp. 335-344.

[SLAT92] SLATER, M. "Tracing a ray through uniformly subdivided n-dimensional space". *The Visual Computer*. Vol. 9, 1992 pp. 39-46.

[SPAC91] SPACKMAN, J. WILLIS, P. "The SMART Navigation of a Ray Through an Oct-tree". *Computer & Graphics*. Vol. 15, No. 2, 1991, pp.185-194.

[SPAR63] SPARROW, M.E. "A new and simpler formulation for radiative angle factors". *Journal of Heat Transfer*. May 1963, pp. 81-88.

[SPAR66] SPARROW, M.E., CESS, R. D. *Radiation Heat Transfer*. Wadsworth, Belmont, CA,1966.

[SPEE86] SPEER, L.R. et al. "A Theoretical and Empirical Analysis of Coherent Ray-Tracing". *Computer-Generated Images (Proc. Of Graphics Interface '85)*, 1986, pp. 11-25.

[STOL95] STOLTE, N., CAUBET, R. "Discrete Ray-Tracing of Huge Voxel Spaces". *Proc. of the Eurographics '95*, 1995, pp. 383-394.

[STON75] STONE, L. *Theory of Optimal Search*. Academic Press, 1975, pp. 27-28.

[SUNG91] SUNG, K. "A DDA Octree Traversal Algorithm for Ray Tracing". *Eurographics '91*, 1991, pp. 11-23.

[SUNG92] SUNG, K., SHIRLEY, P." Ray Tracing with the BSP tree". Graphics Gems III, 1992, pp. 271-274.

[SUTH74a] SUTHERLAND, I.E., HODGMAN, G.W. "Reentrant Polygon Clipping". *Comm of the ACM*, Vol. 17, No.1,1974, pp. 32-42.

[SUTH74b] SUTHERLAND, I.E., SPROULL, R.F., SCHUMACKER, A.R. "A characterization of ten hidden-surface algorithms. *Computing Surveys*, Vol. 6, No. 1, 1974, pp. 1-55.

[THOM89] THOMAS, D., NETRAVALI, A., FOX, D. "Antialiased ray tracing with covers". *Computer Graphics Forum*, Vol. 8, No. 4, 1989, pp.325-336.

[ULLN83] ULLNER, M.K. "Parallel Machines for Computer Graphics". *Tech. Report 5112:TR:83, California Institute of Technology,* 1983.

[UREÑ98] UREÑA, C. *Métodos de Monte-Carlo Eficientes para Iluminación Global*. Tesis Doctoral. Universidad de Granada. Depto. Lenguajes y Sistemas Informáticos. 1998.

[VERD96a] VERDÚ,I., GIMÉNEZ,D., TORRES,J.C. "Ray Tracing for Natural Scenes in Parallel Processors". *High-Performance Computing and Networking, Lecture Notes in Computer Science*. COLBROOK, A., HERTZBERGER, B., SLOOT, P (ed.). Springer-Verlag, 1996, pp. 297-305.

[VERD96b] VERDÚ,I. *Optimización y Paralelización en Síntesis de Escenas Naturales por Trazado de Rayos*. Tesis Doctoral. Universidad de Murcia. Depto. Informática y Sistemas. 1996.

[VOOR92] VOORHIES, D. "Triangle-Cube Intersection". *Graphics Gems III,* KIRK, D. (ed.), AP Profesional, Boston, 1992, pp. 236-239.

[WALL87] WALLANCE, J.R., COHEN, M.F., GREENBERG, D.P. "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods". *Computer Graphics (Siggraph'87)*, Vol. 21, No. 4, 1987, pp. 311-320.

[WALL89] WALLANCE, J.R., ELMQUIST, K.A., HAINES, E.A "A Ray Tracing Algorithm for Progressive Radiosity". *Computer Graphics (Siggraph'89)*, Vol. 23, No. 2, 1989, pp. 315-324.

[WEGH84] WEGHORST, H., HOOPER, G., GREENBERG, D. "Improved Computational Methods for Ray Tracing". *ACM Trans. on Graphics*, Vol.3, No. 1, January 1984, pp. 52-69.

[WHIT80] WHITTED, T.; "An Improved Illumination Model for Shaded Display". *Comm of the ACM*, Vol. 23, No. 6, 1980, pp. 343-349.

[WHIT92] WHITMAN, S. *Multiprocessor Methods for Computer Graphics Rendering.* Jones and Bartlett Publishers, 1992.

[WHIT94] WHITMAN, S., HANSEN, C.D., CROCKETT, T. "Recent Development in Parallel Rendering". *IEEE Computer Graphics and Applications.* July 1994, pp 21-22.

[WILL88] WILLIAMS, N.S., BUXTON, B.F., BUXTON, H. "Distributed Ray Tracing Using a SIMD Processor Array". *Theoretical Foundations of Computer Graphics and CAD*, Springer-Verlag, 1988, pp. 703-25.

[WILT94] WILT, N. *Object-Oriented Ray Tracing in C++*. John Wiley & Sons, Inc. 1994.

[ZALI97] ZALIK,B. CLAPWORTHY, G. OBLONSEK, C. "An Efficient Code-Based Voxel-Traversing Algorithm". *Computer Graphics Forum*, Vol. 16, No. 2, 1997, pp. 119-128.

# Anexos.

# *Anexo A: Lenguaje NFF*

El lenguaje NFF (Neutral File Format) fue creado por Eric Haines [HAIN87] como un medio para comparar las velocidades entre distintas implementaciones del algoritmo de traza de rayos. Los programas incluidos en su paquete de software SPD (Standard Procedural Database), generan ficheros NFF como salida. El paquete SPD puede ser descargado vía FTP desde el sitio de Internet ftp.princenton.edu (directorio /pub/Graphics).

Los elementos de un fichero NFF son:

- Situación del observador.
- Color de fondo.
- Fuentes de luz puntuales.
- Descripción de superficies con texturas lisas.
- Objetos (polígonos, polígonos suavizados, cilindros, conos y esferas).

Nuestra implementación de un intérprete de ficheros NFF, incluye la mayoría de estas características, únicamente no se han contemplado los objetos de tipo cilindro y cono. Esto es debido a que inicialmente para este tipo de objetos no se ha determinado la función que calcula su inclusión dentro de un cubo, con lo cual este tipo de objetos no han formado parte de las escenas utilizadas para nuestra pruebas. Pero este interprete además de las características básicas del lenguaje NFF, lo amplía para que los objetos de las escenas puedan definir superficies con texturas más complejas. Las nuevas texturas que se pueden utilizar corresponden a todas las que implementa *OORT* en la clase *Texture*; granito, madera, cristal, etc.

Un fichero NFF consta de una serie de líneas de texto. Cada una de las líneas corresponde a un comando y unos ciertos argumentos para ese parámetro. A continuación se detallan estos comandos.

**Parámetros del observador**

| | |
|---|---|
| *V* | Cabecera |
| ***from*** Fx Fy Fz | Localización del ojo |
| ***at*** Ax Ay Az | Hacia donde mira el ojo |
| **up** Ux Uy Uz | Tope de la imagen |
| ***angle*** ángulo | Amplitud del ángulo de visión |
| ***hither*** distancia | Distancia de la pantalla |
| ***Resolution*** xres yres | Resolución de la pantalla |

Los parámetros del visor deben establecerse antes de definir cualquier tipo de objeto.

**Color de fondo**

| | |
|---|---|
| ***b*** R G B | Establece el color de fondo |

Este comando es opcional y si no se indica en el fichero, se toma por defecto un color de fondo negro (0,0,0).

**Luces**

| | |
|---|---|
| *l* X Y Z [R G B] | Localización de una luz puntual, y opcionalmente su color. |

Si el color de la luz no se define, se toma por defecto un color blanco para ella (1,1,1).

**Parámetros de superficie**

| | |
|---|---|
| *f* red green blue Kd Ks Shine T index | Establece superficie lisa |

*red*, *green*, y *blue* especifican el color reflejado por el objeto, *Kd* y *Ks* dan los coeficientes de reflexión especular y difusa, *Shine* es el brillo de la superficie, *T* es el coeficiente de transmisión, *index* es el índice de refracción del objeto. Estos parámetros correspondientes a la superficie de un objeto se asignan a todos los objetos que se definen a continuación hasta encontrar otra nueva definición de superficie. Con este parámetro se define una superficie lisa. Posteriormente veremos las extensiones que se han añadido para otro tipo de superficies.

**Esfera**

| | |
|---|---|
| *s* center.x center.y center.z radius | Define una esfera dando su centro y radio |

**Polígono**

| | |
|---|---|
| *p* total_vértices<br>vert1.x vert1.y vert1.z<br>[...] (hasta total_vértices) | Define un polígono dando el número de vértices que tiene, y la lista de los vértices. |

**Polígono suavizado**

| | |
|---|---|
| *pp* total_vértices<br>vert1.x vert1.y vert1.z norm1.x norm1.y norm1.z<br>[...] (hasta total_vértices) | Define un polígono suavizado dando el número de vértices que tiene, y la lista de los vértices con sus vectores normales. |

**Conos y cilindros**

| | |
|---|---|
| *c* base.x base.y base.z base_radius<br>apex.x apex.y apex.z apex_radius | Define un cono o cilindro, dando el centro de la base superior e inferior y sus radios. |

**Comentario**

| # [cadena de caracteres] |
|---|

Tan pronto como se encuentra el carácter "#", el resto de la línea es considerada como un comentario.

**Extensiones**

A parte del formato estándar NFF definido por Eric Haines, el interprete implementado se ha dotado de unas extensiones, para poder aprovechar cierto tipo de texturas que ofrece *OORT*.

| | |
|---|---|
| ***surfreflect*** | Superficie reflectante (espejo) |
| ***surfglass*** | Superficie transparente (cristal) |
| ***surfgranite*** | Superficie de granito |
| ***surfshiny*** | Superficie brillante |
| ***texblackmarble*** | Textura marfil negro |
| ***texbluemarble*** | Textura marfil azul |
| ***texcherrywood*** | Textura madera normal |
| ***texclouds*** | Textura cielo con nubes |
| ***texdarkwood*** | Textura madera oscura |
| ***texpinewood*** | Textura madera de pino |
| ***texredmarble*** | Textura marfil rojo |
| ***texwhitemarble*** | Textura marfil blanco |

Si uno de estos comandos aparece en una de las líneas del fichero NFF, los objetos que le siguen utilizan la superficie y/o textura que corresponda al comando, hasta encontrar otra definición de textura o superficie.

# *Anexo B: MPI*

## B.1 Introducción

MPI es una librería que incluye un conjunto de primitivas necesarias para realizar y controlar la comunicación entre diferentes procesadores. Es, por tanto, una librería que pretende resolver los problemas derivados del paradigma de ***paso de mensajes***. Su orientación es convertirse en un estándar dentro de este entorno de trabajo que permita asegurar la portabilidad, eficiencia y funcionalidad máxima esperada. Aunque este paradigma está especialmente diseñado para sistemas con arquitectura MIMD de memoria distribuida (multicomputadores de memoria distribuida) es especialmente válido para las nuevas aplicaciones paralelas desarrolladas sobre redes de estaciones de trabajo.

Al diseñarse MPI, en vez de seleccionar un lenguaje y adoptarlo como el estándar, se tuvieron en cuenta las características más atractivas de los sistemas para el *paso de mensajes* existentes en aquel momento. En su desarrollo tuvieron especial influencia los trabajos realizados por fabricantes de este tipo de sistemas como: IBM, INTEL NX/2, Express, nCUBE's Vernex, p4 y PARMACS. Otras contribuciones importantes provienen de las implementaciones de lenguajes de paso de mensajes existentes hasta ese momento: Zipcode, Chimp, PVM, Chameleon y PICL.

El esfuerzo para estandarizar MPI involucró a cerca de 60 personas de 40 organizaciones diferentes principalmente de U.S.A y Europa. La mayoría de los vendedores de computadoras concurrentes estaban involucrados con MPI, así como con investigadores de diferentes universidades, laboratorios del gobierno e

industrias. El proceso de estandarización comenzó en un congreso denominado "E*stándares para el paso de mensajes en un ambiente con memoria distribuida*", patrocinado por el *Centro de Investigación en Computación Paralela* en Williamsbur Virginia (abril 29-30 de 1992). En éste se llegó a una propuesta preliminar conocida como **MPI1**, enfocada principalmente a comunicaciones punto a punto, pero en la que no se incluían rutinas para comunicación colectiva. El estándar final MPI fue presentado en la conferencia de Supercomputación en Nov. de 1993, constituyéndose así el foro MPI (http://www.mcs.anl.gov/mpi/index.html).

En un ambiente de comunicación con memoria distribuida en el cual las rutinas de nivel mas alto y/o las abstracciones son construidas sobre rutinas de paso de mensajes de nivel bajo, los beneficios de la estandarización son muy notorios. La principal ventaja al establecer un estándar para el paso de mensajes es la portabilidad y la facilidad de utilización.

MPI es un sistema complejo, que comprende 129 funciones, de las cuales la mayoría tienen muchos parámetros y variantes.

**Objetivos del estándar MPI:**

El objetivo básico de MPI o Message Passing Interface (Interfaz de Paso de Mensajes), es desarrollar un estándar válido para escribir programas que implementen el *paso de mensajes*. Intenta establecer un estándar práctico, portable, eficiente y flexible. A continuación se detallan las metas fundamentales que dirigen su desarrollo.

- Diseñar una interfaz de programación aplicable a diferentes entornos.
- Permitir una comunicación eficiente, evitando el copiar de memoria a memoria y permitiendo (donde sea posible) el solapamiento de computación y comunicación, además de reducir la comunicación con el procesador.

- Permitir implementaciones que puedan ser utilizadas en un ambiente heterogéneo.
- Permitir enlaces fáciles con lenguajes como C, C++, Fortran, etc.
- Asumir un interfaz de comunicación seguro. El usuario no debe lidiar con errores de comunicación, tales errores son controladas por el subsistema de comunicación interior.
- Definir un interfaz que no sea muy diferente a los actuales, tales como PVM, NX, Express, p4, etc., y proveer de extensiones para permitir mayor flexibilidad.
- Definir un interfaz que pueda ser implementado en diferentes plataformas, sin cambios significativos en el software y las funciones internas de comunicación.
- La semántica del interfaz debe ser independiente del lenguaje.
- La interfaz debe ser diseñada para producir tareas seguras.

## B.2 Modelo de programación

En el modelo de programación MPI, una ejecución comprende uno o más procesos comunicados a través de llamadas a funciones de librería para mandar (*send*) y recibir (*receive*) mensajes a otros procesos. En la mayoría de las implementaciones de MPI, se parte de un conjunto fijo de procesos creados al lanzar la ejecución del programa principal, mediante el comando *mpirun*. Sin embargo, estos procesos pueden ejecutar diferente código. De ahí que, el modelo de programación MPI es algunas veces referido como **MPMD** (*Multiple Program Multiple Data*) para distinguirlo del modelo **SPMD** (*Simple Program Multiple Data*), en el cual todos los procesadores ejecutan el mismo programa.

Para comunicar los diferentes procesos existen primitivas básicas que pueden utilizarse como operaciones de *comunicación punto a punto.* Estas operaciones pueden servir para implementar comunicaciones locales y no estructuradas. Pero

junto a las *comunicaciones punto a punto*, existen otras que implican a un mayor número de procesos (p.e. *broadcast*), denominadas funciones primitivas de *comunicación colectiva.* A su vez MPI ofrece la posibilidad de analizar el tipo de mensaje que se está recibiendo antes de realizar la propia operación de recepción, permitiendo y dando soporte, de este modo, a *comunicaciones asíncronas.* Probablemente una de las características más importantes del MPI es el soporte que ofrece a la programación modular. Para ello utiliza un mecanismo llamado *comunicador,* el cual permite al programador definir módulos que encapsulan estructuras internas de comunicación (estos módulos pueden ser combinados secuencial y paralelamente).

Aunque MPI es un sistema complejo, podemos resolver un amplio rango de problemas usando seis funciones básicas. Estas funciones inician y terminan una ejecución, identifican procesos o envían y reciben mensajes.

| | |
|---|---|
| **MPI_INIT** | *Inicia una ejecución.*<br>**MPI_INIT( int *argc, char ***argv )**<br>argc, argv son requeridos solo por el contexto del lenguaje C (son los argumentos del programa principal). |
| **MPI_FINALIZE** | *Termina una ejecución.*<br>**MPI_FINALIZE()** |
| **MPI_COMM_SIZE** | *Determina el número de procesos en una ejecución.*<br>**MPI_COMM_SIZE( comm, size )**<br>**IN *comm*** comunicador (manejador [handle])<br>**OUT *size*** número de procesos en el grupo del comunicador (entero). |
| **MPI_COMM_RANK** | *Determina el identificador del proceso actual "mi proceso".*<br>**MPI_COMM_RANK( comm, pid )** |

| | |
|---|---|
| | **IN *comm*** comunicador (manejador [handle])<br>**OUT *pid*** identificador del proceso dentro del comunicador . |
| **MPI_Send** | *Envía un mensaje.*<br>**MPI_Send( buf, count, datatype, dest, tag, comm )**<br>**IN *buf*** dirección del buffer a enviar (tipo x)<br>**IN *count*** número de elementos del buffer a enviar (entero >= 0)<br>**IN *datatype*** tipo de datos del buffer a enviar (*handle*)<br>**IN *dest*** identificador del proceso destino (entero)<br>**IN *tag*** message tag (entero)<br>**IN *comm*** comunicador (handle) |
| **MPI_Recv** | *Recibe un mensaje.*<br>**MPI_Recv( buf, count, datatype, source, tag, comm, status )**<br>**OUT *buf*** dirección del buffer a recibir (tipo x)<br>**IN *count*** número de elementos a recibir del buffer (entero >= 0)<br>**IN *datatype*** tipo de datos a recibir del buffer (*handle*)<br>**IN *source*** identificador del proceso fuente, o<br>MPI_ANY_SOURCE(entero)<br>**IN *tag*** message tag, o MPI_ANY_TAG(entero)<br>**IN *comm*** comunicador (handle)<br>**OUT *status*** estado del objeto (estado) |

**IN**: Parámetro de entrada.

**OUT**: Parámetro de salida.

**INOUT**: Parámetro de entrada/salida.

Excepto las dos primeras, el resto de las funciones necesitan indicar un "*comunicador*" como argumento. El *comunicador* identifica el grupo de procesos y el contexto en el cual la operación se debe realizar. Como se mencionó anteriormente, los *comunicadores* proveen de un mecanismo para identificar subconjuntos de procesos durante el desarrollo de programas modulares y para garantizar que los mensajes generados con diferentes propósitos no sean

confundidos. Por defecto se puede utilizar el *comunicador MPI_COMM_WORLD*, el cual identifica todos los procesos en una ejecución.

Las funciones *MPI_INIT* y *MPI_FINALIZE* se utilizan para iniciar y terminar una ejecución MPI respectivamente. *MPI_INIT* debe ser llamada antes que cualquier otra función MPI y solamente una vez por proceso. Ninguna función MPI puede ser llamada después de *MPI_FINALIZE*.

Las funciones *MPI_COMM_SIZE* y *MPI_COMM_RANK* determinan, respectivamente, el número de procesos en la ejecución actual y el identificador (entero) asignado al proceso actual. De este modo, los procesos pertenecientes a un grupo son identificados con un único número (entero), que se asigna de modo secuencial comenzando en 0.

Consideremos el siguiente programa (pseudocódigo):

```
program main

 begin
       MPI_INIT()        // Iniciar la ejecución.
       MPI_COMM_SIZE(MPI_COMM_WORLD, count) // Obtener el n° de procesos.
       MPI_COMM_RANK(MPI_COMM_WORLD, myid)// Obtener el id de mi proceso.
       print("Yo soy el proceso ", myid, " de ", count)
       MPI_FINALIZE()                          // Finalizar.
 end
```

El estándar MPI no especifica cómo se debe iniciar una ejecución paralela. Un mecanismo típico podría ser el especificar desde la línea de comandos el número de procesos a crear, por ejemplo ***miPrograma -n 4***, donde *miPrograma* es el nombre del ejecutable. Se pueden especificar argumentos adicionales tales como los nombres de los procesos en un ambiente de red o los nombres de los ejecutables para una ejecución MPMD.

Consideremos el siguiente programa (pseudo código):

```
program main
begin
     MPI_INIT()
     MPI_COMM_SIZE(MPI_COMM_WORLD, count)

     // deben ser dos procesos
     if count != 2 then exit

     MPI_COMM_RANK(MPI_COMM_WORLD, myid)

     if myid = 0 then
        funcionA(100)
     else
        funcionB()
     endif

     MPI_FINALIZE()
end


// para el proceso 0
procedure funcionA( num )
begin
   // mandar mensajes
   for i=1 to num
       MPI_Send(i, 1, MPI_INT, 1, 0, MPI_COMM_WORLD)
   endfor
   i=-1  // mandar mensaje de terminación
   MPI_Send(i, 1, MPI_INT, 1, 0, MPI_COMM_WORLD)
end


// para el proceso 1
procedure funcionB
begin
   // recivir los mensajes
   MPI_Recv(msg, 1,MPI_INT, 0,0,MPI_COMM_WORLD,status)
   while msg != -1 do
     usa_mensaje(msg) // usarlo
     MPI_Recv(msg,1,MPI_INT,0,0,MPI_COMM_WORLD, status)
   enddo
end
```

Este programa está diseñado para ejecutar dos procesos. El primero llama al procedimiento "*funcionA*" y el segundo a "*funcionB*", creando dos tareas diferentes. El primer proceso hace una serie de llamadas *MPI_Send* para enviar 100 mensajes (con datos tipo entero) al segundo proceso, terminando la secuencia con un

número negativo. El segundo proceso recibe estos mensajes utilizando *MPI_Recv*.

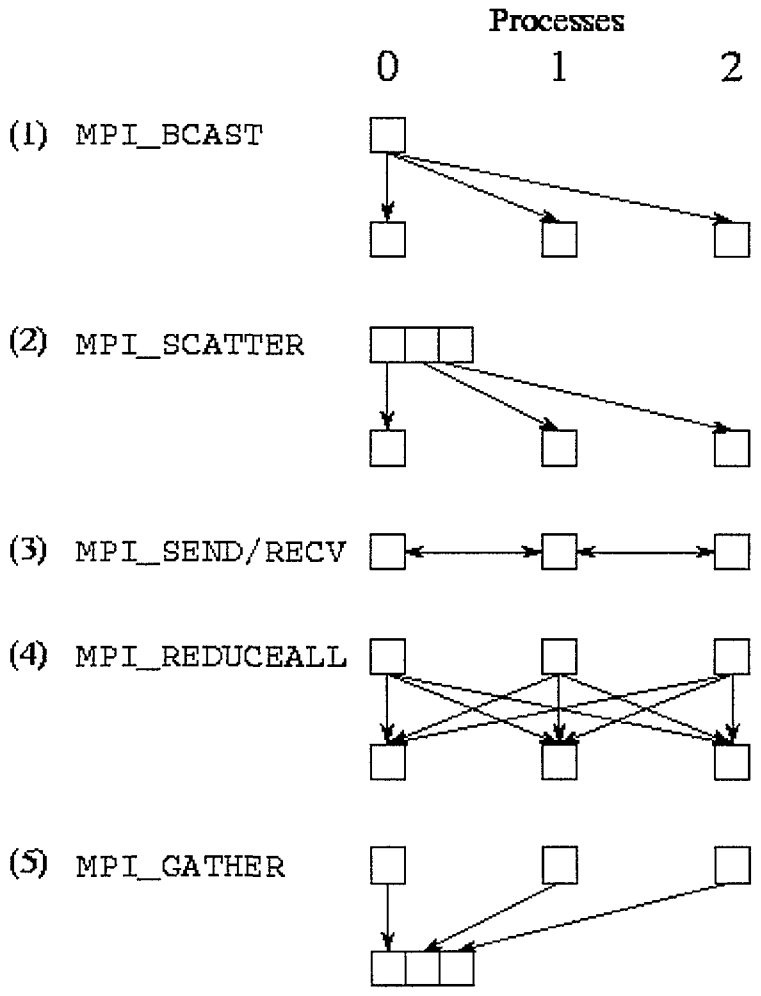
El *paso de mensajes* en módulos de programación es, por defecto, no determinista. El orden de llegada de los mensajes enviados desde dos procesos *A* y *B* hacia un tercer proceso *C*, no está definido. Pero, MPI garantiza que dos mensajes enviados desde un proceso *A*, hacia otro proceso *B*, llegarán en el orden en que fueron enviados.

En el modelo de programación Tarea/Canal, el determinismo está garantizado al definir canales separados para diferentes comunicaciones y al asegurar que cada canal tiene solamente un escritor y un lector. Por lo cual, un proceso *C* puede distinguir mensajes recibidos de *A* o *B*, ya que estos llegan en diferentes canales. MPI no soporta canales directos, pero provee mecanismos similares. En particular, permite una operación de recepción para especificar una fuente, etiqueta, y/o contexto.

Como ya se ha explicado, los algoritmos paralelos ejecutan llamadas a funciones que coordinan la comunicación entre múltiples procesos. Por ejemplo, todos los procesos pueden necesitar cooperar para invertir una matriz distribuida o para sumar un conjunto de números distribuidos entre los distintos procesos. Claramente, estas operaciones globales pueden ser implementadas por un programador usando las funciones *send* y *receive*. Pero en algunos entornos la utilización de estas funciones elementales no es recomendable para solucionar este tipo de comunicaciones en las que participan varios procesos a la vez. Para permitir la optimización de estas tareas, MPI provee un conjunto especializado de funciones denominadas *de comunicación colectiva* que involucran a un conjunto de procesos a la vez.

En la siguiente figura se representa el funcionamiento de las principales

funciones de comunicación suministradas por MPI. En este caso, se supone que el sistema está formado por tres procesos (cada proceso aparece en una columna separada).




Los cinco tipos de comunicación son: (1) *broadcast*, (2) *scatter*, (3) *nearest-neighbor exchange*, (4) *reduction*, y (5) *gather*.

1. *MPI_Bcast* hace una transmisión (*broadcast*) del proceso 0 al resto de procesos existentes.

2. *MPI_Scatter* distribuye la información del proceso 0 a otros procesos.

3. *MPI_Send* y *MPI_Recv* intercambian datos entre vecinos.
4. *MPI_Allreduce* determina los valores calculados en diferentes procesos y los distribuye al resto de procesos.
5. *MPI_Gather* acumula los valores de otros procesos en un solo proceso.

La comunicación entre vecinos puede realizarse mediante funciones que bloquean el proceso hasta que se complete la operación de comunicación iniciada (*MPI_Send, MPI_Recv*) o bien solapando en cierta medida la tarea de comunicación iniciada y el procesamiento de otras partes del programa. En este último caso, se habla de comunicación no bloqueante y las funciones de envío y recepción son diferentes (*MPI_Isend, MPI_Irecv*). Estas funciones inician el envío o recepción de la información y dejan que el proceso siga ejecutando otras líneas de código y más adelante, o tal vez antes de realizar otra operación sobre el mismo buffer, comprobará si la operación iniciada previamente ha finalizado con éxito.

Junto a las funciones anteriores MPI ofrece primitivas que permiten explorar los mensajes de entrada a un determinado proceso antes de realizar la operación de recepción. Con estas funciones podemos implementar tareas que periódicamente exploran la situación de un cierto canal de comunicación a la espera de determinar la procedencia y características del mensaje que se recibirá a continuación. Para ello MPI presenta tres funciones *MPI_Iprobe, MPI_Probe, MPI_Get_count.*

*MPI_Iprobe* comprueba la existencia de mensajes pendientes, pero no realiza su recepción, para ello necesita realizar posteriormente *MPI_Recv*. Esta función analiza un canal de comunicación pero no bloquea el proceso, permitiendo por tanto la realización de otras tareas.

*MPI_Probe* bloquea el proceso hasta que llegue un mensaje de unas determinadas características.

El siguiente fragmento de código hace uso de estas funciones para recibir un

mensaje de una fuente desconocida y con un número de enteros desconocidos como contenido. Primero detecta la llegada del mensaje utilizando *MPI_Probe*. Después, determina la fuente del mensaje y utiliza *MPI_Get_count* para conocer su tamaño. Finalmente, crea un buffer para recibir el mensaje.

```
int count, *buf, source;

MPI_Probe(MPI_ANY_SOURCE,0,comm,&status);
source= status.MPI_SOURCE;
MPI_Get_count(status,MPI_INT,&count);
buf= malloc(count*sizeof(int));
MPI_Recv(buf, cout, MPI_INT, source, 0, comm, &status);
```

MPI soporta la programación modular a través de su mecanismo de *comunicador* (*comm*), al permitir la especificación de componentes de un programa, los cuales encapsulan las operaciones internas de comunicación y proveen un espacio para el nombre local de los procesos.

Una operación de comunicación MPI siempre especifica un comunicador. Éste identifica el grupo de procesos que están comprometidos en la comunicación y el contexto en el cual ocurre. El grupo de procesos permite a un subconjunto el comunicarse entre ellos mismos usando identificadores locales y el ejecutar operaciones de comunicación colectiva sin meter a otros procesos. El contexto forma parte del paquete asociado con el mensaje. Una operación *receive* puede recibir un mensaje sólo si éste fue enviado en el mismo contexto. Si dos rutinas usan diferentes contextos para su comunicación interna, no puede existir peligro alguno en confundir sus comunicaciones.

En ejemplos anteriores, todas las operaciones de comunicación utilizaron el *comunicador* por defecto MPI_COMM_WORLD, el cual incorpora todos los procesos relacionados en una ejecución MPI y define un contexto por defecto. A continuación se describen las funciones que permiten utilizar de modo más flexible diferentes *comunicadores*.

- *MPI_COMM_DUP*: Un programa puede crear un nuevo comunicador, conteniendo el mismo grupo de procesos pero con un nuevo contexto para asegurar que las comunicaciones generadas para diferentes propósitos no sean confundidas. Este mecanismo soporta la composición secuencial.
- *MPI_COMM_SPLIT*: Un programa puede crear un nuevo comunicador, conteniendo sólo un subconjunto del grupo de procesos. Estos procesos pueden comunicarse entre ellos sin riesgo de tener conflictos con otras ejecuciones concurrentes. Este mecanismo soporta la composición paralela.
- *MPI_INTERCOMM_CREATE*: Un programa puede construir un intercomunicador, el cual enlaza procesos en dos grupos. Soporta la composición paralela.
- *MPI_COMM_FREE*: Esta función puede ser utilizada para liberar el comunicador creado al usar las funciones anteriores.

## B.3 Especificaciones en el contexto del lenguaje C o C++

La utilización dentro de un programa en C o C++ de las primitivas suministradas dentro de la librería MPI tiene ciertas peculiaridades que a continuación se indican:

- Los valores de los estados son devueltos como valores enteros.
- Los nombres de las funciones son tal y como se presentan en la definición del MPI pero solo con el prefijo de MPI y la primera letra del nombre de la función en mayúsculas.
- El valor devuelto para una ejecución exitosa es MPI_SUCCESS. También está definido un conjunto de valores de error.
- Las constantes están en mayúsculas y están definidas en el archivo *mpi.h*
- Los manejadores son representados por tipos definidos en *mpi.h*

- Los parámetros de las funciones del tipo *IN* son pasados por valor, mientras que los parámetros *OUT* y *INOUT* son pasados por referencia (como punteros).
- La variable de estado (*status*) tiene el tipo *MPI_Status* y es una estructura que contiene los campos: *status.MPI_SOURCE* y *status.MPI_TAG*.

- Existen tipos de datos MPI para cada tipo de datos de C: *MPI_CHAR, MPI_INT, MPI_LONG_INT, MPI_UNSIGNED_CHAR*, etc.
- Pueden definirse datos complejos mediante la creación de tipos de datos derivados: *MPI_Type_contiguous, MPI_Type_Extend, MPI_Type_Size, MPI_Type_Count, MPI_Type_ib, MPI_Type_ub*, etc.