Capítulo I
Introducción

1.1 Antecedentes

Uno de los esquemas más conocidos que es utilizado para la representación de objetos sólidos mediante la descomposición recursiva del espacio es el de los OctTrees. Un OctTree es una estructura jerárquica de árbol octal generada mediante la descomposición recursiva de un universo cúbico finito. Este cubo es dividido en ocho subespacios iguales llamados octantes. Si un octante se encuentra parcialmente dentro del sólido, es dividido otra vez en ocho cubos, y este proceso continúa recursivamente hasta que todos los octantes obtenidos se encuentren completamente afuera o adentro del sólido, o cuando un nivel máximo de profundidad predefinido es alcanzado [Samet90].

La representación mediante OctTrees clásicos permite realizar operaciones Booleanas entre dos objetos sólidos de una manera muy sencilla, ya que los algoritmos recorren la estructura de los árboles de ambos objetos al mismo tiempo. Además, ya que por la propia naturaleza del modelo se conserva un ordenamiento espacial entre los nodos del árbol, para poder visualizar un objeto representado mediante este modelo eliminando al mismo tiempo sus partes ocultas se puede utilizar un algoritmo que simplemente recorra el árbol nivel por nivel visitando recursivamente cada uno de los hijos en un orden que depende únicamente de la posición relativa del observador respecto al nodo padre [Foley92]. Sin embargo, las representaciones obtenidas nunca son exactas, y los niveles de profundidad requeridos para reducir los errores de aproximación producen inmensos consumos de memoria [Samet90].

Por otro lado, el modelo conocido como PM-OctTrees [Samet90], también llamado OctTrees Extendidos [Ayala85],[Brunet85],[Navazo86],[Navazo89] fue desarrollado para resolver los ya mencionados problemas del modelo clásico. Originalmente, agregaron tres nuevos tipos de nodo (nodos Cara, Arista y Vértice, o C, A, V), además de los nodos clásicos. Posteriormente, un cuarto tipo fue añadido (nodos Casi-Vértice) [Ayala91]. Estos nuevos nodos permitieron obtener representaciones mucho más compactas y exactas de objetos poliédricos, heredando al mismo tiempo la mayoría de las ventajas de los OctTrees clásicos.

1.2 Problemas existentes

A pesar de que los PM-OctTrees resolvieron la mayoría de los problemas encontrados en el modelo clásico y heredaron la mayoría de sus beneficios, algunos problemas persistieron o algunos nuevos aparecieron:

- Primero, existen objetos poliédricos para los cuales es imposible obtener una representación exacta a menos que el proceso recursivo de descomposición vaya más allá de vóxels cuyo tamaño sea de una unidad cúbica, aún si se utilizan los nodos Casi-Vértice en la representación. Esto se debe principalmente a que pueden existir caras vecinas en un sólido intersectando un mismo octante muy pequeño pero cuya arista común se encuentre ligeramente afuera del octante, o también cuando se encuentran ángulos muy agudos entre dos caras de un sólido **[Figura1.1]**.

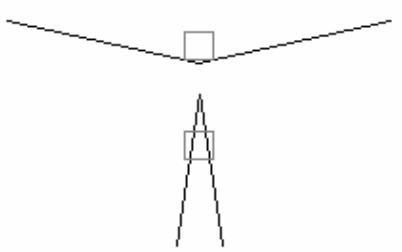

FIGURA 1.1
Dos ejemplos mostrando casos donde aparecen nodos Grises usando los PM-OctTrees (elaboración propia).

- Segundo, aunque los algoritmos para realizar operaciones Booleanas entre dos objetos heredaron los beneficios de ser capaces de recorrer los árboles de ambos sólidos al mismo tiempo, originalmente se requería un procedimiento especializado para manejar cada una de las posibles combinaciones entre nodos Extendidos (C-C, C-A, C-V, A-A, A-V, V-V) [Navazo86]. La complejidad para algunos de dichos procedimientos resultó ser extremadamente alta. Peor aún, para operar nodos Extendidos cualquiera con nodos Vértice de mayor tamaño pueden aparecer configuraciones de caras que no es posible adaptar a ninguno de los procedimientos especializados considerados para las combinaciones mencionadas anteriormente, por lo que más casos especiales aparecen. Posteriormente [Navazo89], [Ayala91], estos algoritmos fueron simplificados, aunque aún requiriendo de varios casos especiales. De cualquier forma, todas las estrategias requerían convertir desde o hacia modelos de fronteras o árboles CSG, lo que no es difícil de realizar, pero aún así representa un costo extra en cada nodo Extendido.
- Tercero, en cualquier vértice de un objeto poliédrico *manifold* puede haber a lo más dos caras incidentes que sean no convexas en ese vértice en particular **[Figura1.2]**. Los nodos Vértice y Casi-Vértice que almacenan vértices de esta categoría deben ser manipulados de manera diferente a los nodos que almacenan únicamente caras

convexas cuando se intenta reconstruir su representación en modelo de fronteras al momento de visualizarlos. Sin embargo, de acuerdo a la experiencia empírica de las pruebas realizadas, la información almacenada en un nodo Extendido no es suficiente para distinguir de qué categoría es el vértice almacenado, por lo que el proceso de reconstruir el modelo de fronteras de estos nodos se encuentra indefinido. Inclusive, en [Navazo86] no se encuentra ninguna mención a este problema ni a su forma de resolverlo, debido a que los aspectos relativos a la visualización del modelo no fueron incluidos.

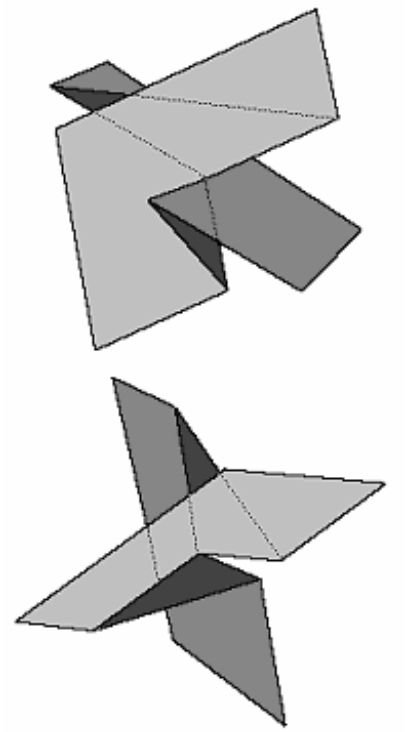

FIGURA 1.2
Vértice mostrando dos caras incidentes no convexas, visto desde dos posiciones diferentes (elaboración propia).

- Cuarto, originalmente los nodos Vértice fueron restringidos a vértices que tuvieran a lo más tres o cuatro caras incidentes [Navazo86], principalmente porque se necesitaban

clasificar ciertos puntos con respecto a un nodo terminal en el algoritmo que obtiene el PM-OctTree a partir de un modelo de fronteras, y este tópico no tenía solución para vértices más complejos. Posteriormente [Navazo89], [Ayala91], el modelo fue generalizado para soportar vértices con cualquier número de caras incidentes. Sin embargo, el método para resolver esto no fue tan simple.

- Quinto, los nodos Vértice y Casi-Vértice requieren de algún tipo de ordenamiento espacial para poder visualizarlos eliminando sus partes ocultas. El algoritmo de visualización del modelo clásico únicamente permite detectar el orden en que los nodos deben pintarse, pero no es suficiente para determinar el orden de pintado de las caras internas de un nodo Vértice o Casi-Vértice **[Figura1.3]**. Además, existen objetos poliédricos con vértices para los cuales, a determinadas posiciones del observador, no existe ningún ordenamiento posible para pintar las caras incidentes con una eliminación correcta de partes ocultas **[Figura1.4]**.

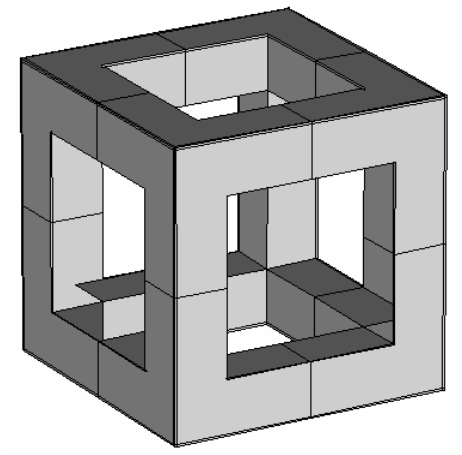

FIGURA 1.3
Objeto poliédrico representado mediante el modelo Extendido, con nodos Vértice siendo visualizados sin una eliminación correcta de partes ocultas (elaboración propia).

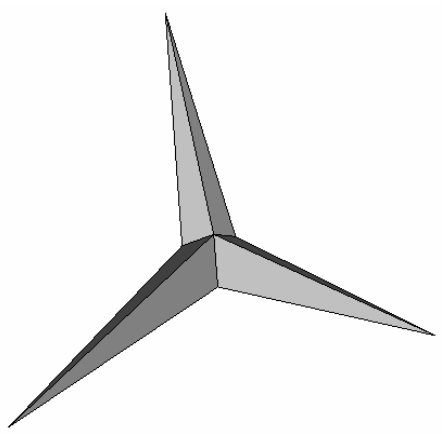

FIGURA 1.4
Vértice para el que sin importar qué cara fuera elegida primero para construir un árbol BSP, forzosamente habría al menos una partición, lo que significa que no existe ningún ordenamiento posible para visualizar las caras con eliminación de partes ocultas (elaboración propia).

Existen diversas estrategias que pueden resolver el quinto problema encontrado en los PM-OctTrees, incluyendo al *Z-Buffer*, *Ray-Tracing* y árboles BSP, pero todos ellos requerirían un gran costo extra al momento de visualización. Los dos primeros operan directamente sobre los píxeles resultantes (espacio imagen), calculando el color de cada uno y decidiendo si debe o no ser pintando sobre un píxel pintado con anterioridad. El último opera sobre el espacio objeto directamente.

Un árbol BSP (partición binaria del espacio, por sus siglas en inglés) representa una subdivisión recursiva y jerárquica del espacio d-dimensional. Se entiende mejor como el proceso que toma un subespacio y lo divide por medio de cualquier hiperplano que intersecte el interior de dicho subespacio. Esto produce dos nuevos subespacios que pueden ser divididos recursivamente. Se pueden obtener representaciones exactas de objetos poliédricos por medio de ellos, además de que las operaciones Booleanas entre ellos son muy sencillas de realizar, mucho más que en el modelo de fronteras, aún cuando en realidad son ligeramente más

complejas que en los OctTrees clásicos. Además, su algoritmo para clasificar un punto respecto a un sólido representado mediante este esquema es muy simple también. Sin embargo, su principal desventaja radica en que, para poliedros complejos, los árboles BSP resultantes pueden ser extremadamente grandes, aún después de aplicar algunas optimizaciones [Paterson90].

1.3 Propuesta

En este trabajo se propone un nuevo tipo de nodo terminal como una extensión al modelo clásico de los OctTrees, la cual permitirá resolver los cinco mayores problemas encontrados en los PM-OctTrees conservando al mismo tiempo los beneficios del modelo clásico así como los problemas que ya habían sido resueltos con los PM-OctTrees. Este nuevo tipo de nodo terminal almacenará la representación del árbol BSP de aquellas caras que lo intersecten, en vez de almacenar simplemente una lista con los apuntadores a las ecuaciones de dichas caras y sus bits de configuración.

1.4 Organización

En el siguiente capítulo se realiza un pequeño estudio comparativo de las principales ventajas y desventajas existentes entre los modelos de representación geométrica más utilizados y relacionados con el modelo propuesto. En el Capítulo III se describe a profundidad el modelo propuesto, sus requerimientos, y sus principales ventajas y desventajas. En el Capítulo IV se

plantean los algoritmos para visualizar correctamente el modelo propuesto, incluyendo la conversión al modelo de fronteras y la eliminación de partes ocultas. En el Capítulo V se plantean los algoritmos para la realización de operaciones Booleanas en el nuevo modelo. En el Capítulo VI se muestran los resultados obtenidos con la implementación del modelo utilizando como entrada desde objetos simples hasta escenas complejas, creadas tanto algorítmicamente [Haines88], [Vort01], como tomadas a partir de datos geográficos reales, como lo es la cartografía del volcán Popocatépetl. También se presentan las conclusiones del presente trabajo. Por último, en el Apéndice A, se incluyen todos los detalles técnicos de la implementación, tanto las tecnologías utilizadas como las plataformas soportadas y las capacidades del sistema programado.