Capítulo 2. Modelado

En este capítulo se presentan diferentes modelos para representar objetos tridimensionales en memoria, describiendo sus principales ventajas, desventajas y usos. Como ya se mencionó, el navegador implementado en este trabajo está limitado a objetos poliédricos (cuerpos compuestos por superficies planas), por lo que los modelos y algoritmos aquí descritos están enfocados a este tipo de estructuras.

Cuando se habla de mundos tridimensionales tres tipos de modelados pueden observarse:

+*Modelado de escenas*.- el mundo virtual es visto como un conjunto de objetos o primitivas tridimensionales con alguna posición y orientación en el espacio 3D.

+*Modelado de sólidos*.- se utiliza para representar objetos tridimensionales considerados como sólidos.

+*Modelado de superficies*.- es utilizado para definir superficies que no necesariamente pertenecen a objetos sólidos y sirve de base para el modelado de sólidos.

2.1 Modelado de superficies

Como ya se mencionó, para presentar en pantalla un objeto es necesario contar con una representación interna de él en la computadora, y para que éste luzca real, es necesario que esta representación sea lo más cercana posible al objeto real. Si un objeto es poliédrico,

sus superficies son planas y puede modelarse almacenando los vértices y aristas de sus superficies. Sin embargo, la mayoría de los cuerpos reales presentan superficies curvas, las cuales se representarían mediante las coordenadas de sus puntos infinitos, lo cual es imposible en una computadora con un límite de almacenamiento [FOLE90].

Para representar superficies curvas existen algunas técnicas como:

+*Curvas paramétricas polinomiales*: las superficies son definidas por tres polinomios con un parámetro *t* en común: un polinomio para *x*, otro para *y* y otro para *z*. Los coeficientes de los polinomios son seleccionados de manera que la curva siga el camino deseado.

+*Mallas de polígonos*: las superficies curvas son aproximadas por un conjunto de superficies planas poligonales. Aunque ésta es sólo una aproximación de la superficie, el error puede disminuir aumentando el número de polígonos para representar la curva.

2.1.1 Mallas de polígonos

Formalmente una malla de polígono es una colección de aristas, vértices y polígonos conectados de manera que cada arista es compartida por al menos dos polígonos[ZECH94]. Los vértices son descritos por sus coordenadas *x*, *y*, *z*. Las aristas se definen como la conexión entre dos vértices. Un polígono es definido como una secuencia

cerrada de aristas. Existen diferentes esquemas que pueden ser usados para describir la malla, cada uno con ventajas y desventajas.

Para evaluar estos esquemas, se debe tomar en cuenta el espacio que requieren para almacenar un objeto y el grado de facilidad que permiten para realizar ciertas operaciones con ellos, lo que repercute en tiempo. Aunque la cantidad de memoria ocupada es importante, la elección del esquema adecuado depende principalmente de las operaciones que se necesitan realizar con los objetos, ya que éstas repercuten en el tiempo de respuesta del navegador. A continuación se presentará una descripción de los principales esquemas: enumeración explicita de vértices, apuntadores a una tabla de vértices y apuntadores a una tabla de aristas.

2.1.2 Enumeración explícita de vértices

Bajo este esquema, la malla es almacenada como un conjunto de polígonos, donde cada polígono es representado por una lista de vértices ordenados, de manera que sea posible recorrer su perímetro viajando de vértice a vértice. Si los vértices se encuentran en el sentido de las manecillas del reloj, entonces se dice que utiliza un sistema de mano derecha, y en caso contrario de mano izquierda. Formalmente, sea *P* el polígono a representar y *n* el número de vértices de *P*, entonces

$P = \{ (x_1,y_1,z_1),...,(x_n,y_n,z_n) \}$

en donde el primer y último vértice están conectados.

Este esquema es muy sencillo pero tiene varias desventajas en cuanto a espacio y tiempo para realizar cálculos:

+Los vértices compartidos entre dos o más polígonos están duplicados, ya que se encuentran en la lista de vértices de cada uno de estos polígono. Para un solo polígono esta representación es adecuada, sin embargo, para una malla de polígonos, en la cual un vértice es compartido por varios polígonos, es muy ineficiente. En la figura 2.1 izquierda es posible observar que cada vértice del cubo pertenece a tres polígonos, por lo que la información está triplicada. En objetos más complejos el problema es aún mayor. Véase la figura 2.1 derecha, donde cada vértice de la esfera pertenece a seis polígonos y el punto *A* del cono a diez.

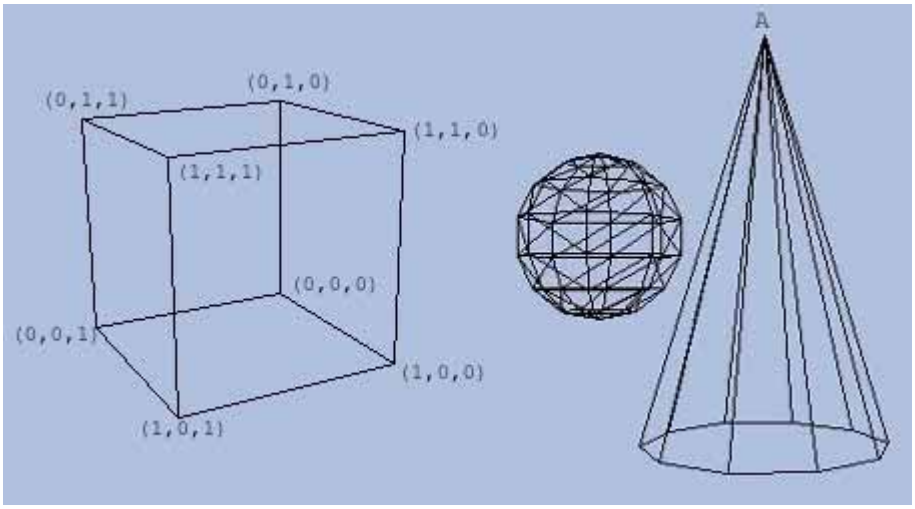

Fig. 2.1. Objetos primitivos en modelo de alambres.
Izquierda: cubo tridimensional con coordenadas en sus vértices.
Derecha: esfera y cilindro (nótese que el punto *A* del cono es compartido por diez polígonos y cada vértices de la esfera por seis).

+Aunque el problema del almacenamiento de vértices es serio, si se tiene la suficiente memoria el sistema funcionará correctamente. Para un navegador, el problema

más grave con este esquema es la repetición de cálculos al momento de realizar transformaciones sobre los objetos, como rotarlos o trasladarlos. Debido a que no se cuenta con una estructura que especifique qué vértices o aristas están compartidas por polígonos, se deben realizar los cálculos necesarios para cada vértice de la lista de cada polígono. Esto implica una repetición de cálculos (que de por sí son complejos) para vértices compartidos que afecta el tiempo de respuesta del navegador.

+También tiene inconvenientes al momento de dibujar los polígonos en pantalla, debido a que cada arista compartida por dos polígonos es dibujada dos veces.

La estructura utilizada para representar el cubo de la figura 2.1 izquierda se presenta en la siguiente tabla:

Tabla de polígonos:

	X1	Y1	Z1	X2	Y2	Z2	X3	Y3	Z3	X4	Y4	Z4
Polígono1	0	1	0	0	1	1	1	1	1	1	1	0
Polígono2	1	1	1	1	0	1	1	0	0	1	1	0
Polígono3	1	1	0	1	0	0	0	0	0	0	1	0
Polígono4	0	1	1	0	1	0	0	0	0	0	0	1
Polígono5	0	1	1	0	0	1	1	0	1	1	1	1
Polígono6	1	0	1	0	0	1	0	0	0	1	0	0

2.1.3 Apuntadores a una tabla de vértices

Una malla de polígonos puede ser descrita empleando para cada polígono, una lista de apuntadores a una tabla de vértices, en donde se encuentren todos los vértices de la

malla una sola vez. Esta estructura hace un uso más eficiente de la memoria ya que no presenta vértices repetidos, lo que acelera el tiempo de respuesta del navegador, debido a que para rotar o mover los polígonos, basta con aplicar los cálculos necesarios a la tabla de vértices, evitando cálculos repetitivos.

Aunque esta estructura es mucho más eficiente que la enumeración explícita de vértices, todavía presenta el problema de que aristas compartidas por dos polígonos son dibujadas dos veces.

La estructura utilizada para representar el cubo de la figura 2.1 izquierda se puede observar en las siguientes tablas:

Tabla de polígonos

	V1	V2	V3	V4
Polígono1	6	5	1	2
Polígono2	1	3	4	2
Polígono3	2	3	8	6
Polígono4	5	7	8	6
Polígono5	5	7	3	1
Polígono6	3	7	8	4

Tabla de vértices

	X	Y	Z
Vértice1	1	1	1
Vértice2	1	1	0
Vértice3	1	0	1
Vértice4	1	0	0
Vértice5	0	1	1
Vértice6	0	1	0
Vértice7	0	0	1
Vértice8	0	0	0

2.1.4 Apuntadores a una tabla de aristas

Una malla de polígonos también puede ser representada por una tabla de vértices y una tabla de aristas, donde cada elemento en esta tabla contiene dos apuntadores a la tabla de vértices que definen esa arista. Bajo este esquema, un polígono puede representarse utilizando una lista de apuntadores a la tabla de aristas que definen al polígono.

Este esquema es muy eficiente debido a que no hay vértices repetidos y las aristas no son dibujadas dos veces debido a que las aristas compartidas por dos vértices se encuentran definidas sólo una vez, sin embargo requiere del uso de más memoria que el esquema anterior para almacenar las aristas.

Las siguientes tablas muestran la estructura de datos para representar el cubo de la figura 2.1 izquierda:

Tabla de polígonos

	A1	A2	A3	A4
Polígono1	1	2	3	4
Polígono2	8	9	5	4
Polígono3	5	10	6	1
Polígono4	7	2	4	8
Polígono5	8	3	7	12
Polígono6	12	11	10	9

Tabla de aristas

	V1	V2
Arista1	6	2
Arista2	5	6
Arista3	5	5
Arista4	1	2
Arista5	2	4
Arista6	6	8
Arista7	5	7
Arista8	1	3
Arista9	3	4
Arista10	4	8
Arista11	8	7
Arista12	7	3

Tabla de vértices

	X	Y	Z
Vértice1	1	1	1
Vértice2	1	1	0
Vértice3	1	0	1
Vértice4	1	0	0
Vértice5	0	1	1
Vértice6	0	1	0
Vértice7	0	0	1
Vértice8	0	0	0

2.2 Modelado de sólidos

El modelado de sólidos es un área central de búsqueda y desarrollo en diversas aplicaciones como la ingeniería y diseño de productos, Computer-Aided Manufacturing (CAM), prototipos electrónicos y planeación de animaciones[AGUI98]. Todos estos sistemas utilizan cuerpos sólidos que necesitan ser representados en la computadora y la elección del tipo de modelado a utilizar es muy importante.

Un objeto sólido es aquel cuyas superficies limitan un volumen cerrado. Generalmente los objetos sólidos cumplen con la propiedad de que dado un punto de vista del observador, la caras frontales del objeto obscurecen e impiden ver sus caras traseras. Aunque en la sección anterior se presentaron esquemas con los que se pueden representar cuerpos en 2D y 3D, éstas no contienen la información necesaria para considerar esos objetos como sólidos, sin embargo sirven como base para representarlos.

Una representación de sólidos debe contener las siguientes propiedades[REQU80]:

+*Dominio*: el conjunto de entidades representables en el esquema. Debe ser lo suficientemente extenso para permitir representar un conjunto útil de objetos físicos.

+*Validez*: el conjunto de representaciones válidas producidas por el esquema. La base de datos del esquema no debe contener ninguna estructura que corresponda a objetos no reales.

+*Ser único*: una representación es única si puede ser usada para codificar cualquier sólido de una sola manera.

+*Ser completo*: un esquema es completo si cada representación válida producida por el esquema no es ambigua. Una representación no es ambigua si no hay duda de lo que se está representando y ésta corresponde a un solo sólido.

2.2.1 Representación por barrido

Un objeto sólido puede representarse con una superficie plana y una trayectoria en la que esta superficie es barrida. La superficie plana se conoce como generador y la trayectoria a través de la cuál es barrida, director[MORT89]. Las trayectorias más utilizadas para el barrido son la traslación y rotación, aunque trayectorias más complejas pueden utilizarse como espirales y curvas.

Este tipo de representación de sólidos es adecuado para objetos que tienen secciones constantes como varios objetos de la industria (Fig. 2.2). Sin embargo, los objetos barridos son difíciles de modelar eficientemente y de realizar cálculos con ellos. Por ejemplo, cálculos de volumen pueden ser muy complicados debido a que la trayectoria de barrido puede provocar que el objeto se intersecte a sí mismo (Fig. 2.2.C).

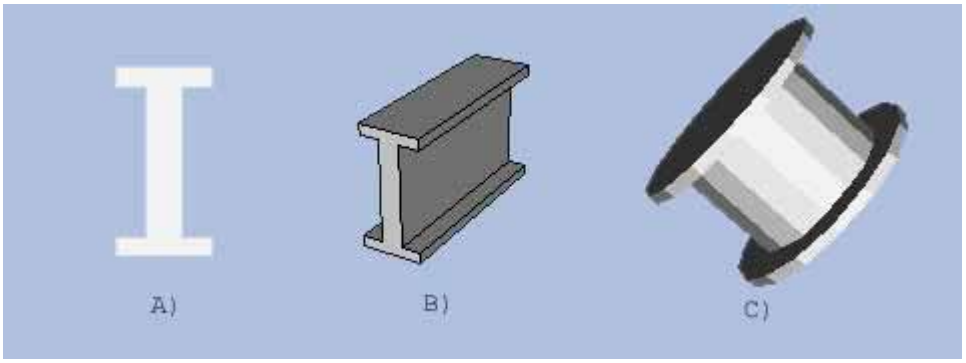

Fig. 2.2 Barridos. A)Superficie plana usada como generador. B)Barrido por traslación. C)Barrido por rotación.

2.2.2 Representación de fronteras

Este esquema también conocido como *boundary representation* o simplemente *b-rep*, es una manera más eficiente de representar objetos sólidos utilizando los elementos en sus fronteras, esto es, aquellos elementos que separan el interior del exterior del objeto. Estos elementos son sus vértices, aristas y caras. Los vértices contienen la información base del objeto y las aristas y caras se encuentran ordenadas por relaciones topológicas. Este modelado puede verse como un grafo dirigido donde los nodos pueden ser los vértices, aristas o caras de los cuerpos.

Existen varias representaciones utilizando *b-rep*; algunas limitan los cuerpos a caras planas y poligonales, incluso polígono convexos o triángulos. La manera más sencilla de representar un sólido utilizando este modelo es con una lista de sus caras o polígonos, utilizando algún tipo de modelado de superficies (ver sección 2.1) pero para distinguir la parte frontal de cada polígono se deben listar los vértices de cada polígono en el mismo sentido viendo cada uno de ellos de frente.

2.2.3 Representación por partición del espacio

Utilizando representaciones por partición del espacio, un sólido es dividido en una colección de primitivas sólidas que no se intersectan entre sí[HARR92]. Estas primitivas pueden diferir en tamaño, parametrización, posición, tipo y orientación. Las representaciones más comunes son:

+Descomposición en celdas
+Árboles octales.
+Árboles de partición binaria del espacio(BSP-Trees)

La idea general de estas representaciones es dividir recursivamente el sólido original en sólidos más simples que puedan ser manipulados de manera más eficiente.

2.2.3.1 Descomposición en celdas

Este esquema es una de las formas más generales de partición del espacio. Básicamente consiste en definir sólidos utilizando un conjunto de primitivas parametrizables e incluso curvas definidas en el sistema. El sólido es formado uniendo cada una de estas primitivas, las cuales no deben intersectarse entre sí. Aunque esta representación de sólidos no es ambigua, un mismo cuerpo puede representarse de varias formas (Fig. 2.3). Una desventaja de esta representación es que es difícil de validar dado que cada primitiva que forma parte del objeto debe ser probada para calcular intersecciones u otras operaciones[FOLE90].

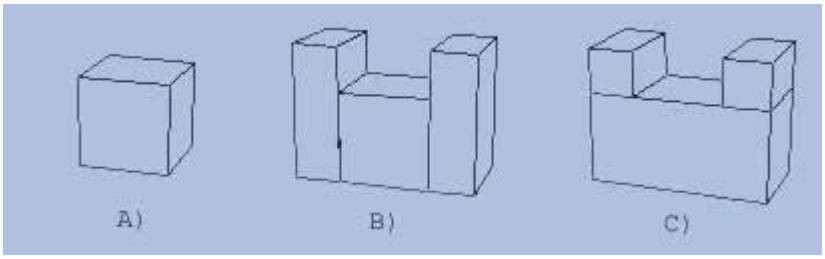

Fig. 2.3- A)Primitiva utilizada para construir el mismo objeto presentado en las figuras B) y C).

2.2.3.2 Enumeración del espacio ocupado

Este es un caso particular de la descomposición en celdas. La idea de la enumeración del espacio ocupado consiste en dividir el sólido en celdas del mismo tamaño colocadas en una cuadrícula fija. Estas celdas son llamadas *voxels* (volume cells) en analogía a pixels (picture cells). El tipo de celda más utilizado es el cubo, donde cada celda tiene seis caras vecinas, doce aristas vecinas y ocho vértices vecinos[KONG89].

Este tipo de representación de sólidos permite saber la presencia o ausencia de una celda en la cuadrícula, por lo que facilita operaciones como saber si una celda pertenece o no a un polígono o determinar si dos objetos son adyacentes. Por ejemplo, si representamos una cuerpo como un arreglo tridimensional a_{ijk}, de datos binarios, donde la celda $a_{ijk}=1$ significa que esa celda es cubierta por el sólido y en otro caso está vacía, para realizar operaciones binarias sobre el objeto basta con aplicar estas operaciones a los elementos del arreglo *a*.

Una desventaja de este esquema es que objetos curvos o con superficies no paralelas a la cuadrícula tienen que ser aproximadas, obteniéndose una mejor representación aumentando la resolución de la cuadrícula (empleando celdas más pequeñas), a costa de un aumento en la cantidad de memoria para almacenar al objeto.

A diferencia de la descomposición en celdas, la enumeración del espacio ocupado no es ambigua y es única (para una resolución utilizada). Sin embargo tiene la desventaja de que necesita mucho espacio para almacenar la estructura de datos. Por ejemplo, a una

resolución de 256^3, la cuál no es totalmente adecuada, necesita 2 Mega bytes para almacenarla. Además, si los cuerpos tiene colores, es necesario almacenar para cada *voxel* el color que le corresponde y, suponiendo que la resolución fuera de 256^3 y cada *voxel* pueda tener 256 colores diferentes, se necesitaría 16 Mega bytes para almacenarla[AGUI98].

2.2.3.3 Árboles octales

Los árboles octales también conocidos como octrees son una variación de la enumeración del espacio ocupado donde las celdas (que pueden variar en tamaño) son organizadas de una manera jerárquica. Su mayor ventaja contra el método anterior es que no necesita tanta memoria para almacenar cuerpos.

Los árboles octales son una derivación de los quadtrees, estructuras que son utilizadas para representar imágenes en 2D. Un quadtree es producido dividiendo recursivamente el plano en cuatro cuadrantes, donde cada cuadrante puede estar lleno, parcialmente lleno o vacío, dependiendo de la cantidad de área que contenga ocupada por el objeto. Los cuadrantes que se encuentra parcialmente llenos son subdivididos de nuevo en cuatro cuadrantes hasta que cada cuadrante está lleno o vacío o hasta que se alcance algún límite de aproximación.

El quadtree es representado utilizando un árbol, en donde las áreas parcialmente llenas son nodos intermedios y las áreas llenas o vacías son las hojas del árbol. El mejoramiento en el uso de memoria se logra gracias a que cada cuatro cuadrantes vecinos

del mismo tamaño son eliminados y substituidos por un solo nodo lleno o vacío según sea el caso (Fig. 2.4).

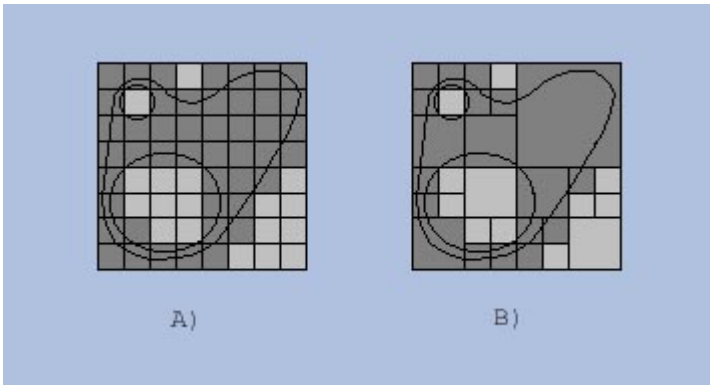

Fig. 2.4.- Una imagen representada usando A)Enumeración del espacio ocupado y B)Quadtree.

Los árboles octales funcionan de la misma manera pero en tres dimensiones. El procedimiento para construir un árbol octal es como sigue: dado un cubo que cubre completamente al espacio del cuerpo a modelar, si éste es muy complejo es dividido en 8 octantes y cada uno de éstos se recorre recursivamente en un orden establecido hasta que cada octante esté lleno o vacío o se haya alcanzado la profundidad máxima (Fig. 2.5). Analógicamente a los quadtrees, un octante está lleno si todo su volumen se encuentra dentro del sólido y vacío si todo su volumen está fuera de él.

Es posible demostrar que la cantidad de nodos en el árbol octal es proporcional a la superficie del objeto[HUNT78]. Esta relación se mantiene debido a que la subdivisión de nodos ocurre sólo de la necesidad de representar los límites del objeto siendo codificado y los únicos nodos internos que son divididos son aquellos que pasan por los límites del objeto[FOLE90].

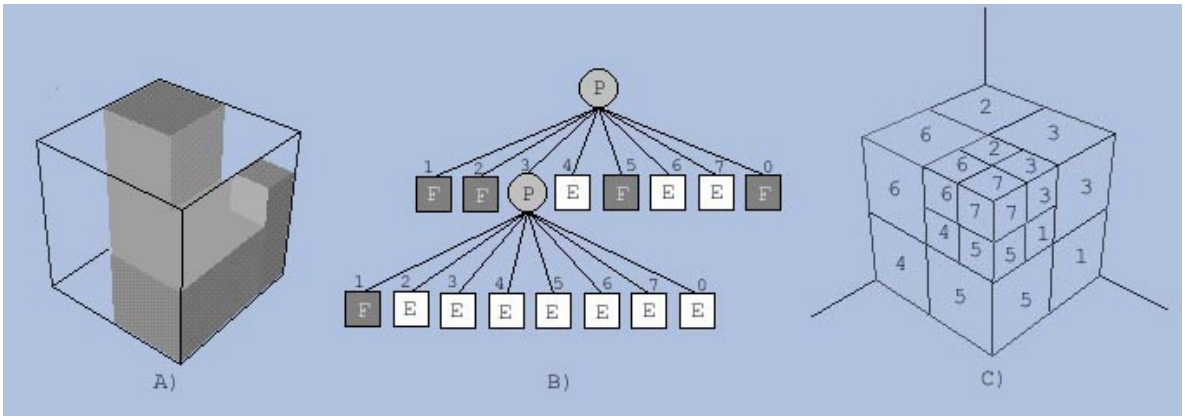

Fig. 2.5.- A)Un objeto tridimensional, B)Su representación como árbol octal, C)Numeración para los octantes (El octante 0 no es visible).

Esta estructura se basa en la idea de dividir para conquistar, la cual permite algoritmos muy sencillos para operaciones como rotaciones, escalamientos con potencia de dos y encontrar vecinos de un nodo realizando un procedimiento recursivo sobre cada nodo hijo[FOLE90].

Sus propiedades son muy parecidas al esquema de enumeración del espacio ocupado: a una resolución determinada, su representación es única y no es ambigua. Sólo objetos ortogonales cuyos vértices caigan en la cuadrícula pueden representarse exactamente. Sin embargo, debido a que son muy versátiles, son muy utilizados para aplicaciones como gráficos por computadora, procesamiento de imágenes, sistemas de información geográfica (GIS), reconocimiento de patrones, modelado de sólidos y volúmenes y otras áreas[AGUI98].

Una desventaja de los árboles octales es que debido a que no existe el concepto de octante parcialmente lleno, si se quiere representar el cuerpo lo mejor posible, los nodos

cercanos a los límites deben ser lo suficientemente pequeños y por la tanto la cantidad de memoria para almacenarlo aumenta en consideración. Los *árboles octales extendidos* son una estructura basada en este esquema que elimina esta desventaja.

2.2.3.4 Árboles de partición binaria del espacio

Conocidos como BSP-trees dividen el espacio recursivamente en dos subespacios divididos por un plano de arbitraria posición y orientación[FOLE90].

Este esquema presenta las ventaja de que es independiente de la dimensión: superficies planas, sólidos o líneas pueden representarse con este modelo. Además, facilita operaciones como determinar si un punto está dentro o fuera de un polígono o sólido y es muy eficiente para desplegar objetos.

Para una mayor descripción de la estructura y algoritmos para la construcción de árboles BSP vea la sección 5.2.

2.3 Modelado de escenas

Las escenas pueden representarse internamente en memoria en el sistema y externamente en archivos para luego ser leídos por el navegador. Los modelos descritos en las secciones anteriores son utilizados para la representación interna de objetos. En esta sección se hablará del modelado de escenas externo.

El modelado de escenas externo es una representación del mundo virtual visto como un conjunto de objetos o primitivas tridimensionales con ciertas propiedades. El modelado de escenas debe contener una serie de elementos:

+*Ser compacta*: el objetivo es poder representar objetos complejos y escenas grandes en la menor cantidad posible de espacio.

+*Ser simple*: el lenguaje utilizado para modelar las escenas debe ser lo más sencillo posible, de manera que estos archivos no solo sean generados por computadoras, sino también creador por personas.

+*Ser completa*: con la especificación del modelado, se debe poder representar cualquier objeto por complejo que este sea.

+*Ser independiente*: tanto de plataforma como de dispositivos especiales como tarjetas aceleradoras de video. El lenguaje debe representar sólo una escena u objeto tridimensional, siendo responsabilidad del programador del navegador, transformar el mundo a pantalla utilizando cualquier herramienta que tenga disponible.

Aunque éstas son las características principales que debe tener un modelado de escenas externo, también es posible que contengan información que no represente un objeto de la escena en sí, sino la interacción de la misma con el usuario, por ejemplo, instrucciones que le indiquen a un navegador que cuando el usuario señale con el puntero un objeto, éste cambie de color.

Generalmente estos archivos son archivos de texto. El formato más conocido para representar escenas es el lenguaje VRML descrito en la siguiente sección. En este trabajo se

utilizó tanto el formato NFF (también explicado más adelante) como el leguaje VRML para ser utilizado en el navegador.

2.3.1 VRML

Virtual Reality Modeling Languaje es un formato para archivos de texto que describen mundos y objetos en 3D para ser presentados por un navegador, el cual interpreta todas estas instrucciones de acuerda a la especificación VRML. Este formato es utilizado principalmente para mostrar escenas tridimensionales a través del internet, de manera similar a HTML para mostrar texto. Los navegadores VRML pueden ser sistemas independientes pero en general son aplicaciones dependientes de un navegador de Web, como Netscape o Intenet Explorer.

La primera versión liberada fue la especificación VRML 1.0 creada por Silicon Graphics, Inc. y fue basada en el formato de archivos de Open Inventor[HART96]. La segunda versión tuvo mejoras significativas y mayor capacidad. Fue desarrollada por el equipo de Silicon Graphics con la contribución de Sony Research y Mitra.

La principal diferencia entre la versión 1.0 y 2.0 es que la segunda tiene más información sobre la interacción del mundo con el usuario así como animación, aunque también hubo mejoras en la representación de objetos, permitiendo efectos complejos como torsiones. Sin embargo, dado que ésta es demasiado extensa y compleja, y la versión 1.0 contiene las especificaciones necesarios para mostrar mundos relativamente realistas y complejos, en este trabajo se utilizó la versión 1.0.

Debido a que VRML se utiliza principalmente en Internet, fue diseñado para satisfacer los siguientes requerimientos:[MUDE96]

+*Independencia de plataformas*.- VRML define únicamente la escena como objetos en 3D y algunas interacciones de la escena con el usuario.

+*Extensibilidad*.- VRML define la escena a través de nodos con propiedades, donde cada nodo puede ser una esfera, un cono, una transformación, etc. Para agregar nuevos objetos a la especificación basta con agregar nodos. También es posible agregar propiedades a nodos ya existentes.

+*Capacidad de trabajar bien en conexiones de poco ancho de banda*.- VRML permite ir presentando los objetos conforme son leídos del archivo, por lo que no es necesario esperar que todo el archivo sea leído para luego presentarlo.

El siguiente texto en lenguaje VRML 1.0 define un objeto sencillo ilustrado en la figura 2.6:

```
#VRML V1.0 ascii
Material{
       diffuseColor 0.29 0.68 1
}
Cylinder {
       height 1.5
       radius .4
}
Cylinder {
       height 3
       radius .05
}
Translation{
       translation 1 -1 0
}
Scale {
       scaleFactor 4  1 1
}
Sphere {
       radius 1
}
```

```
Translation{
       translation 0 1 0
}
Cylinder {
       height .5
       radius .5
}
```

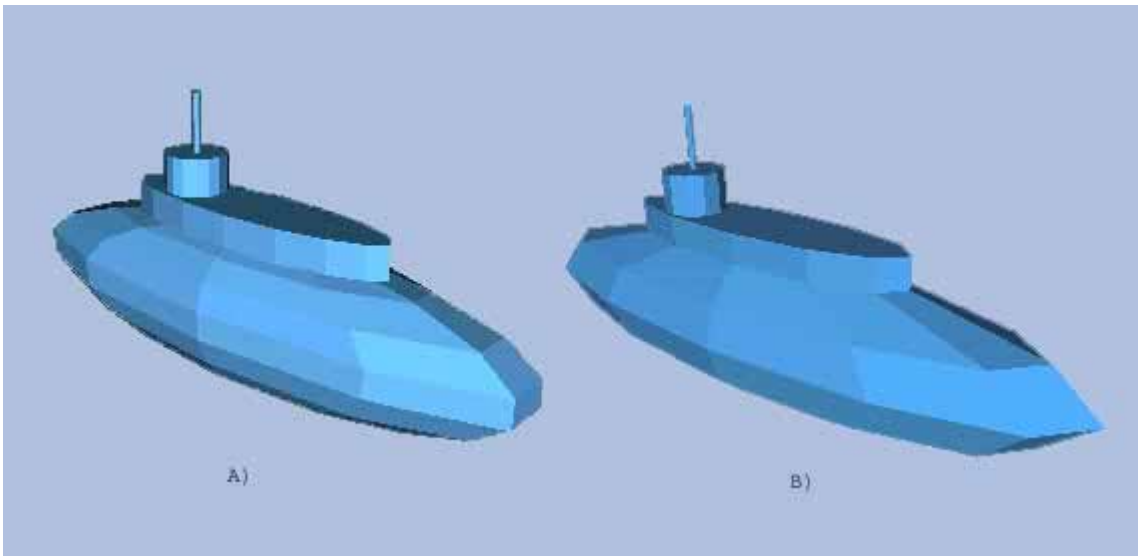

Fig. 2.6 Imagen del mundo descrito en el texto VRML A)Con el navegador implementado en este trabajo, B)Con el navegador comercial VRWeb Viewer 0.54 Beta.

Para una descripción más detallada de la especificación VRML 1.0 ver apéndice A.

2.3.2 Neutral File Format (NFF)

Otro tipo de representación externa de escenas, conocido como *Neutral File Format (NFF),* fue diseñada por Eric Haines en 1988. Este formato está pensado como un lenguaje mínimo para la descripción de escenarios tridimensionales, con el objetivo de probar varios algoritmos y esquemas de presentación de objetos 3D. Describe la geometría y las características básicas de las superficies de los objetos, la posición de luces y la visión del observador.

Basado en este formato, Eric Haines desarrolló varios programas en C para generar una serie de archivos de texto que representaban escenas complejas como montañas fractales, esferas recursivas, anillos, engranes, una estructura de tetraedros recursiva y árboles formados por cilindros y esferas.

Este software tenía la intención de funcionar como una serie de pruebas básicas de imágenes para *ray traicing* y algoritmos de ocultamiento de superficies [HAIN88] mediante la lectura de estos archivos generados y el estudio detallado del tiempo que le tomaba a la máquina presentar las escenas. Por ejemplo, una de las pruebas a realizar era cuántos vectores o polígonos por segundo podía generar una máquina en particular.

El siguiente texto representa una balanza construida con objetos primitivos utilizando el formato NFF. La figura 2.7 muestra la imagen de esta escena.

```
v
from -1.0 -2.61 0.5
at -1.0 -1.61 0.5
up 0.0 0.0 1.0
angle 45.0
hither 1.0
resolution 512 512
b 0.078 0.361 0.753
l 0 0 100.0
f 1.0 0.75 0.33 1.0 0.0 0.0 0.0 0.0
p 4
20 20 -20
-20 20 -20
-20 -20 -20
20 -20 -20
f 1.0 0.9 0.7 0.5 0.5 3.0 0.0 0.0
c
0.0 1.5 0.0 0
0.0 -1.5 0.0 1.5.0
c
-4 1.75 0.0 0.5
4 1.75 0.0 0.5
s -3.35 3 0 0.7
s 3.35 3 0 0.7
```

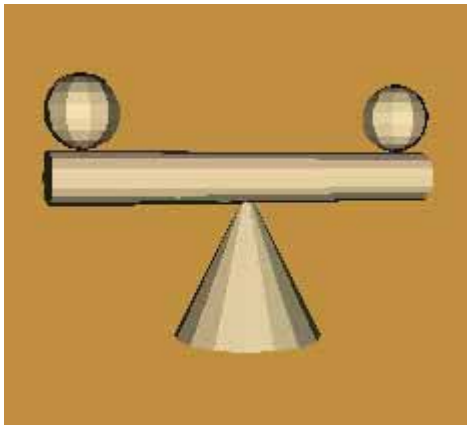

Fig. 2.7-Balanza compuesta por primitivas según el archivo en formato NFF.

Para una descripción más detallada del formato NFF ver apéndice B.