CUBOS MARCHANTES: UNA IMPLEMENTACIÓN EFICIENTE

Carmona Rhadamés. Rodríguez Omaira.

Laboratorio de Computación Gráfica
Facultad de Ciencias
Universidad Central de Venezuela
Apartado 47002. Los Chaguaramos 1041-A.
Caracas-Venezuela.
e-mail: {rhadames,omaira}@jade.ciens.ucv.ve

RESUMEN

En este trabajo se presenta una forma de mejorar la eficiencia del algoritmo de Cubos Marchantes, eliminando el cálculo redundante, y facilitando la manipulación de la información volumétrica. Se muestran aspectos de implementación del mismo, así como resultados experimentales haciendo medición de tiempo para algunos datos de prueba sobre una estación Indigo 2 y otra INDY.

Palabras Claves: Reconstrucción 3-D, Cubos Marchantes, Estructuras de Datos.

1. INTRODUCCIÓN

La Visualización Volumétrica es un área dentro del campo de la Visualización Científica y su objetivo básico es permitir la manipulación de información referente a volúmenes y fenómenos u objetos complejos para producir el despliegue de los mismos. Debido a su gran poder de permitir profundizar en la estructura interna de objetos y fenómenos, la Visualización Volumétrica es ampliamente usada en el campo de la Medicina, Astrofísica, Química, Microscopía, y otras áreas.

Distintos enfoques de despliegue se han desarrollado para la Visualización Volumétrica, los cuales pueden ser divididos en dos categorías: Despliegue por Superficies y Despliegue Volumétrico. La diferencia básica entre los dos métodos radica en el tipo de primitiva que utilizan para definir la representación del volumen. Para el depliegue por superficies, la imagen del volumen se construye a partir de superficies formadas por polígonos o contornos unidos por triángulos. El Despliegue Volumétrico manipula la data volumétrica directamente sin determinar una representación intermedia y la información a desplegar se forma a partir de la manipulación directa sobre los elementos volumétricos.

El algoritmo de Cubos Marchantes (*Marching Cubes*), se ubica en la clasificación de Despliegue por Superficies. Este algoritmo reconstruye una superficie mediante triángulos, a partir de los cortes planos de un objeto y según un umbral definido por el usuario, correspondiente a la superficie que desea visualizar [1,2]. En general, la data volumétrica que maneja el algoritmo es extensa y la cantidad de triángulos que se generan puede sobrepasar el medio millón [1,3], creando un problema de espacio de almacenamiento, tiempo requerido para su generación y despliegue. Para mejorar el tiempo de despliegue, se han propuesto métodos que reducen el número de triángulos de la superficie [3,7].

En este trabajo consideramos una forma alterna de mejorar la eficiencia de este algoritmo. minimizando la redundancia en cálculo y almacenamiento. Proponemos una estructura de datos eficiente que optimiza el manejo de la información, y por ende mejora el tiempo de respuesta en la reconstrucción. Se muestran mediciones de tiempo de reconstrucción con distintas fuentes de entrada de datos: Resonancia Magnética, Discretización de una función implícita, y Tomografía Computarizada.

2. ENTRADA DE DATOS PARA EL ALGORITMO

La data volumétrica de entrada consta de k cortes paralelos, alineados y uniformemente espaciados que corresponden en principio a un sólo objeto. Estos cortes pueden obtenerse de distintas fuentes: la discretización automatizada de una función implícita, Tomografía Computarizada, Resonancia Magnética, etc. Otras fuentes de datos no necesariamente generan cortes alineados, por ejemplo, cortes microscópicos de tejidos orgánicos. En estos casos, es necesario un preprocesamiento de los cortes para ser alineados [4, 5, 6].

Cada corte es una matriz bidimensional $n \times m$, y cada elemento de la misma corresponde (dependiendo de la fuente) a la evaluación de una función implícita, o al valor que el equipo registró sobre un punto del objeto.

El umbral U determina la superficie que se va a reconstruir. Este umbral generalmente es cero cuando la fuente de datos es la discretización 3D de una función implícita, ya que en cero es donde se anula dicha función, y define la superficie que queremos visualizar. De la misma forma, cuando se trabaja con Resonancia Magnética o Tomografía Computarizada se escoge el valor asociado al tejido que se desea reconstruir.

3. EL ALGORITMO DE CUBOS MARCHANTES

La ubicación de la superficie que se desea visualizar a través de los cortes planos se lleva a cabo procesando la entidad volumétrica: el ***voxel*** (***volume pixel***). Un voxel es un cubo lógico formado por 8 pixeles de la data, tal como se muestra en la figura 3.1. Por cada par de cortes consecutivos se tienen $(n-1)\cdot(m-1)$ voxeles, y en total para la data volumétrica de k cortes se tienen $(n-1)\cdot(m-1)\cdot(k-1)$ voxeles.

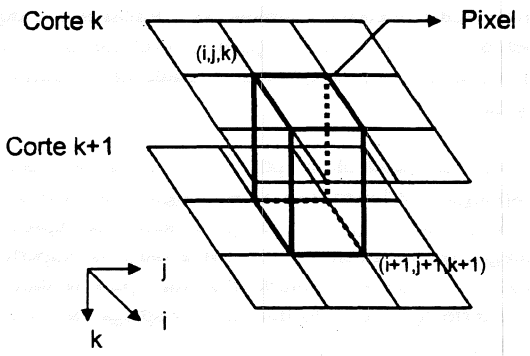

Figura 3.1: Voxel

Este algoritmo se basa en el principio "divide y conquista": generar todos los triángulos de la superficie se reduce a generar los triángulos de cada voxel. En otras palabras, se determina como la superficie corta cada voxel, y usando interpolación lineal se generan los vértices de estos triángulos. También se puede determinar la normal de la superficie en cada vértice de cada triángulo, para su futura visualización con *Gouraud-Shading*. Los siguientes pasos muestran el algoritmo de alto nivel:

```
Para c=1 hasta k-1 hacer
  Para i=1 hasta n-1 hacer
    Para j=1 hasta m-1 hacer
      Para el Voxel delimitado por (i,j,c) e (i+1,j+1,c+1):
        a) Clasificar cada vértice del voxel
        b) Determinar el caso
        c) Obtener los vértices de los triángulos
        d) Determinar la normal de la superficie en cada vértice de cada
           triángulo
        e) Guardar los triángulos
```

a.- Clasificar cada vértice del voxel. A partir del valor umbral U dado por el usuario, se clasifica cada vértice del voxel dependiendo si está dentro o fuera de la superficie. Un vértice está dentro de la superficie si su valor es estrictamente mayor al valor umbral U y está fuera o sobre la superficie, si su valor es menor o igual a U. Los vértices de cada voxel se pueden etiquetar tal como se muestra en la Fig. 3.2.

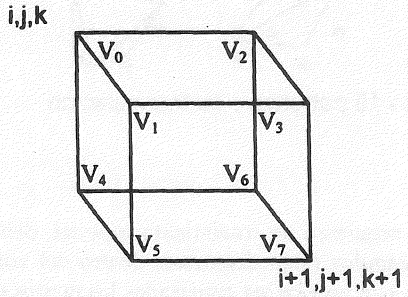

Figura 3.2: Los 8 vértices de un voxel

b.- Determinar el caso. Una arista a_{ij} (arista cuyos vértices son v_i y v_j) del voxel es intersecada por la superficie si uno de sus vértices está adentro, y el otro fuera de la superficie. Hay en total 2^8=256 casos posibles de como la superficie interseca el voxel. Esto corresponde a los 8 vértices del voxel, y las dos posibilidades en que cada vértice puede ser clasificado.

El número de casos a estudiar puede ser reducido a 15. Por un lado, la topología de la superficie triangulada es la misma para el caso i y j, si j=255-i. Note que el caso j se obtiene complementando los bits del valor i. De aquí se deriva el nombre de casos complementarios (Fig. 3.3), reduciendo el número de casos a 128.

Figura 3.3: Un ejemplo de dos casos complementarios. Los pixeles dentro de la superficie están marcados en negro.

Debido a la simetría rotacional de los cubos, el número de casos puede ser reducido a 15, tal como se muestra en la Fig. 3.4. Sin embargo, para facilitar la programación y evitar la tarea de determinar la correspondencia entre un caso $i \in 0..127$ y un caso $i' \in 0..14$, se usa una tabla con la triangulación correspondiente a los 128 casos y su respectivo índice i' por cada caso.

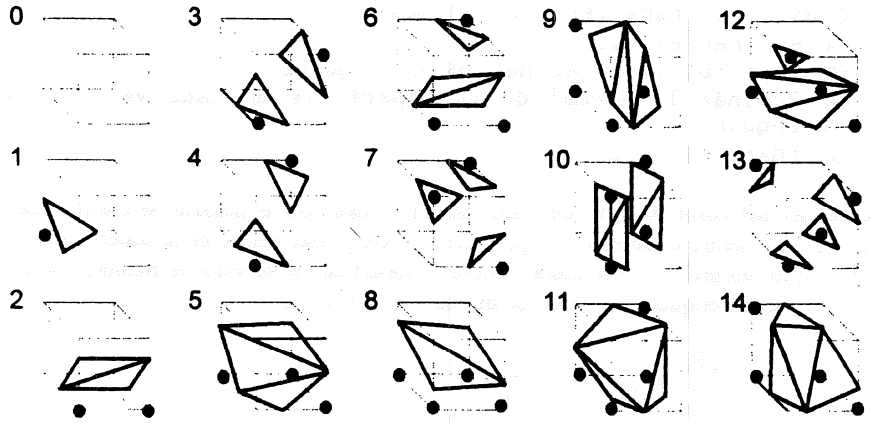

Figura 3.4: Los 15 patrones y su triangulación

De los 15 patrones, el 3, 6, 7, 10, 12, 13 requieren un tratamiento especial, debido a que para los mismo no existe una única topología en que los triángulos están dispuestos dentro del voxel [7, 2]. En estos patrones, se pueden distinguir dos maneras distintas en que pueden ser conectados los puntos donde la superficie interseca a las aristas del voxel para formar triángulos. Estos patrones son denominados casos o patrones ambiguos (Fig. 3.5).

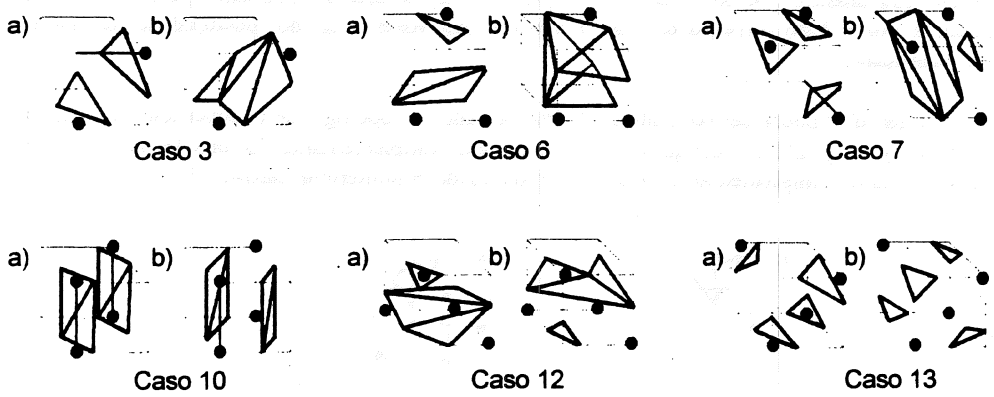

Figura 3.5: Los seis patrones ambiguos

Si la condición ambigua de estos patrones no es tomada en cuenta, se corre el riesgo de producir superficies con huecos [2,7], por lo que para cada patrón se debe determinar un criterio de selección para la triangulación adecuada. Por ejemplo, para el caso i'=3, particularizando con i=40, utilizamos la heurística de promediar las densidades de los vértices de la cara frontal (lo que aproxima la densidad del centro de la misma); si dicho promedio está dentro de la superficie, la triangulación adecuada sería la mostrada en 3(b), y en caso contrario la 3(a).

c.- **Obtener los vértices de los triángulos**: Usando la tabla de referencia que contiene los 128 casos, se determinan cuales son las aristas del voxel que intervienen en cada triángulo. Para cada arista intersecada por la superficie, el valor umbral U pertenece al intervalo formado por las densidades de los vértices de la misma. El punto de intersección de una arista a_{ij} con la superficie se determina usando interpolación lineal. Sea $P(x,y,z)$ el punto de intersección, y $P_k(x_k,y_k,z_k)$ las coordenadas del vértice v_k, P puede ser obtenido como

$$P = (1-t)P_i + tP_j \text{, donde } t = \frac{U - v_i}{v_j - v_i}$$

Debido a que P_i y P_j son vértices de la arista de un cubo, sus coordenadas son iguales salvo en una de ellas. Luego, la interpolación lineal debe realizarse sólo sobre la coordenada desigual, evitando cálculo redundante.

d.- **Determinar la normal de la superficie en cada vértice de cada triángulo**. Se determina la normal de la superficie en los vértices del voxel, cuyas aristas son intersecadas por la superficie. Sea $D[i,j,k]$ el valor de densidad del pixel (i,j) del corte k-ésimo, y d_x, d_y, d_z las longitudes respectivas a los ejes del voxel cúbico. La normal $N(n_x,n_y,n_z)$ para el vértice i,j,k se calcula como las diferencias divididas que se muestran a continuación:

$$n_x = (D[i+1, j, k]-D[i-1, j, k])/d_x$$
$$n_y = (D[i, j+1, k]-D[i, j-1, k])/d_x$$
$$n_z = (D[i, j, k+1]-D[i, j, k-1])/d_x$$

Finalmente, la normal en el punto $P(x,y,z)$ se calcula usando interpolación lineal entre las normales del P_i y P_j, utilizando el mismo valor de t en en paso anterior.

e.- **Guardar los triángulos en una estructura**. Se guardan los triángulos generados en una estructura que soporte su almacenamiento y manipulación.

4. ASPECTOS DE IMPLEMENTACIÓN

Sólo 4 cortes consecutivos son necesarios mantener en memoria en el caso de que nos interese calcular las normales de los vértices de los triángulos, o dos cortes consecutivos en caso contrario. La estructura de datos a utilizar para almacenar los 4 cortes es una matriz tridimensional $nxmx4$, tal como se muestra:

```
A: Array [1..n, 1..m, 1..4] of real
```

Hasta ahora hemos denotado un pixel por una posición (i,j,k) relativa a la data volumétrica. Como se tienen 4 cortes en memoria y debido a la estructura de A, de ahora en lo adelante nos referiremos indistintamente a dicho pixel como $A[i,j,c]$, en donde $c \in 1..4$.

Para guardar los triángulos se usa una lista de vértices Lv y una lista de triángulos Lt. La información de cada nodo de la lista de triángulos va a consistir de tres índices (i_1,i_2,i_3) que representan las posiciones relativas de sus vértices asociados en Lv.

La data es procesada corte a corte, línea a línea y pixel a pixel. Denotemos $V_{i,j,k}$ el voxel actual delimitado por los pixeles (i,j,k) e $(i+1,j+1,k+1)$. Por el orden en que se tratan los voxeles, el voxel $V_{i,j,k}$ $(i,j,k>1)$ ya tiene clasificado 7 de sus vértices en pasos anteriores, faltando sólo por clasificar el vértice v_7. Similarmente, ya se han identificado las aristas intersecadas por la superficie, y el punto de intersección en sí, salvo aquellas que tienen en un extremo el vértice v_7. Para evitar chequeos y cálculos redundantes, se usa la estructura de datos IV que permite localizar de manera directa el índice relativo en Lv de los puntos de intersección de las aristas ya tratadas. Así, en general se determina sólo la intersección (si la hay) de las aristas a_{57}, a_{67}, a_{37} del voxel con la superficie.

```
Type TrioIndices = Array [1..3] of Integer
IV = Array [1..n, 1..m, 1..4] of TrioIndices
```

Un elemento $IV[i+1, j+1, c+1]$ consta de tres enteros (un trío de índices) correspondientes a los índices de los puntos que intersecan a las aristas a_{57}, a_{67}, a_{37} del voxel $V_{i,j,k}$ (Fig. 4.1). En otras palabras, contiene el punto de intersección de la aristas cuyos extremos son:

1. $A[i, j+1, c+1]$ y $A[i+1, j+1, c+1]$
2. $A[i+1, j, c+1]$ y $A[i+1, j+1, c+1]$
3. $A[i+1, j+1, c]$ y $A[i+1, j+1, c+1]$

En caso de que no haya intersección en cualquiera de las aristas, el índice respectivo será -1.

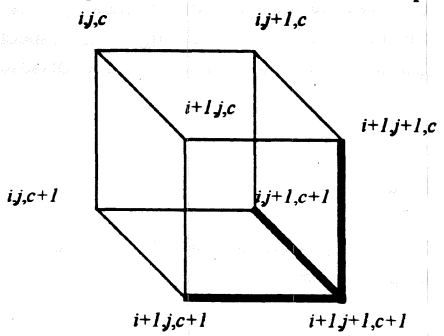

Figura 4.1: Aristas asociadas a $IV[c,i,j]$

Para generar los triángulos asociados al voxel $V_{i,j,k}$, en general ya se conoce la información de la intersección (o no intersección) de 9 de las 12 aristas del voxel con la superficie. Por ejemplo, si la superficie interseca a la arista a_{45}, en $IV[i+1, j, c+1,1]$ tenemos el índice relativo en Lv del punto de intersección. Salvo casos de frontera, en cada paso de Cubos Marchantes, se debe actualizar los tres índices de $IV[i+1, j+1, c+1]$ si hay intersección de la superficie en sus aristas respectivas, y en dado caso, agregar en Lv los puntos de intersección. En los casos de frontera (i=1 o j=1 o k=1) hay que determinar otras intersecciones. El algoritmo general para determinar las intersecciones faltantes en el voxel $V_{i,j,k}$ está dado por:

```
DET(5,7); DET(6,7); DET(3,7);
if  k=1 then
  DET(2,3);   DET(1,3);
  if  i=1 then
      DET(4,6); DET(2,6); DET(0,2);
      if j=1 then
          DET(0,1); DET(4,5); DET(0,4); DET(1,5);
  else
      if j=1 then
          DET(0,1); DET(4,5); DET(1,5);
else
  if i=1 then
      DET(4,6); DET(2,6);
      if j=1 then
          DET(0,4); DET(1,5); DET(4,5);
  else
      if j=1 then
          DET(1,5); DET(4,5);
```

5. RESULTADOS

Las pruebas se realizaron sobre dos estaciones Silicon Graphics. Las características relevantes de estas máquinas se muestran en la tabla 5.1.

Característica \ máquina	M_1: Indigo 2	M_2: INDY
Sistema Operativo	IRIX 5.3	IRIX 5.3
Velocidad	100 MHz	132 MHz
CPU	MIPS R4000	MIPS R4600
FPU	MIPS R4010	MIPS R4610
Cache de datos	8 KBytes	16 KBytes
Cache de Instrucciones	8 KBytes	16 KBytes
Memoria Principal	32 MBytes	32 MBytes
Cache secundario	1 MB	512 KBytes

Tabla 5.1: Características principales de las máquinas

La data volumétrica de prueba proviene de tres fuentes: discretización de funciones implícitas de tres variables. Resonancia Magnética y Tomografía Computarizada. Para determinar los cortes planos de una función implícita $f(x,y,z)$, se evaluó la función en un conjunto discreto de puntos (x_i,y_j,z_k) dentro de un cubo delimitado por los puntos (x_1,y_1,z_1) y (x_n,y_m,z_s), en donde s denota la cantidad de cortes, y (n,m) la cantidad de filas y columnas de cada corte. De los cortes de Resonancia Magnética y Tomografía Computarizada se conocen los parámetros (n,m,s) y la longitud real de cada eje x,y,z de un voxel de la data. Sin embargo, para ser consistente, se analizarán estos datos de manera similar a los obtenidos de una función implícita, con la salvedad de que el umbral que corresponde a la superficie a reconstruir no será el valor cero necesariamente.

Se realizaron pruebas con data de una espina dorsal (ED) proveniente de una Resonancia Magnética, y cortes de un cráneo humano (CH) proveniente de una Tomografía Computarizada. También se tomaron cuatro funciones implícitas: jacky (F_1), toroide (F_2), tres toros (F_3) y esfera al cuadrado (F_4). Los datos de entrada y parámetros de prueba se muestran en la tabla 5.2. Las dimensiones del cubo usado para la función F_4 fueron escogidas más reducidas que la dimensión de la esfera externa. Esto refleja los huecos de esta esfera producto del corte con el cubo (Fig. 5.1.d). Como F_4 es siempre positiva, se tomó un umbral mayor que cero para visualizar lo que se quiere.

Data\item	n	m	s	x_1	y_1	z_1	x_n	y_m	z_s	U
F_1	100	100	100	-1.2	-1.2	-1.2	1.2	1.2	1.2	0.0
F_2	100	100	100	-3.9	-3.9	-3.1	3.9	3.9	3.1	0.0
F_3	100	100	100	-5.7	-4.7	-2.7	3.7	4.7	2.7	0.0
F_4	100	100	100	-1.1	-1.1	-1.1	1.1	1.1	1.1	0.5
ED	64	64	64	-1	-1	-1	1	1	1	15.5
CH	256	256	113	-1.92	-2.56	-1.7	1.92	2.56	1.7	85.5

Tabla 5.2: Parámetros de entrada para 5 datos de prueba

Se realizaron mediciones de tiempo para los diferentes datos de entrada, en ambas máquinas. Por cada dato de prueba se realizaron 20 corridas, para promediar el tiempo de procesamiento. En la tabla 5.3 se muestra por cada dato su tiempo promedio de procesamiento (sin contar el tiempo de lectura de los cortes y de escritura de los triángulos), así como el número de vértices y triángulos generados.

Datos	T.M_1 (seg.)	T.M_2 (seg.)	Vértices	Triángulos
F_1	6.351	3.914	16136	32268
F_2	8.773	5.174	58008	116016
F_3	8.262	4.865	48720	97608
F_4	9.041	4.770	48456	95380
ED	1.902	1.153	9036	18092
CH	i	38.600	285992	573994

Tabla 5.3: Resultados experimentales

Vemos en los resultados experimentales que en promedio la máquina M_2 triangula la superficie en aproximadamente el 55% del tiempo en que la triangula M_1. A pesar de que esta proporción no se ve reflejada en la diferencia de velocidad de las máquinas, M_2 duplica a M_1 en el tamaño de la memoria caché para datos y para instrucciones, lo cual justifica la diferencia notoria en el tiempo de triangulación. La visualización fue hecha con la técnica de Gouraud-Shading (Fig. 5.1).

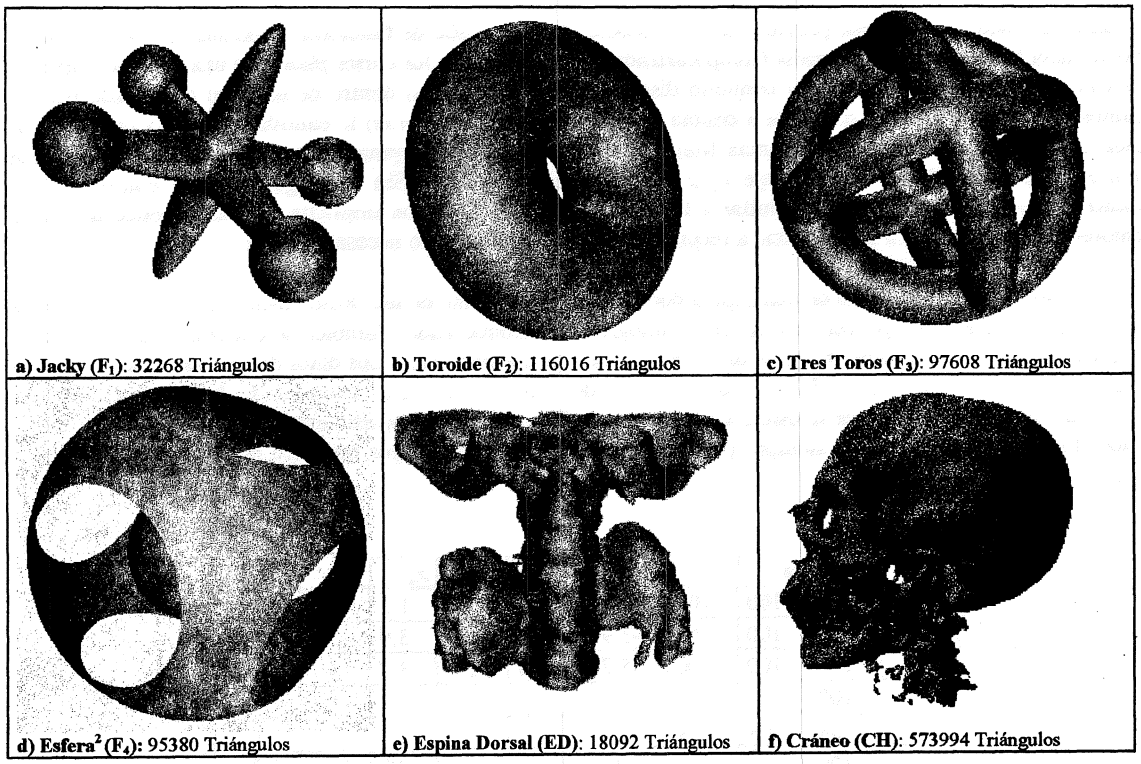

Figura 5.1: Imágenes resultantes de la triangulación

6. CONCLUSIONES

Se ha presentado un análisis de la implementación del algoritmo de Cubos Marchantes, mostrando que el uso de una estructura de datos apropiada al problema determina un tiempo de procesamiento bastante aceptable, tomando en cuenta el gran volumen de información que es manejada.

7. TRABAJOS A FUTURO

En el caso particular de las funciones implícitas, se puede hallar la intersección de la superficie en cada arista por un método de ceros de funciones (Bisección, Newton, etc.). Esto mejoraría la calidad del mallado, ya que el error cometido al hallar un vértice del mallado por interpolación lineal podría reducirse tanto como se quiera. Similarmente se puede obtener una mejor aproximación de la normal en cada vértice.

8. REFERENCIAS

1. Willian E. Lorensen, Harvey E. Cline. "Marching Cubes: A high resolution 3D surface construction algorithm". Computer Graphics, Volumen 21, Número 4, Julio 1987.

2. Ana María Wessolosky. "Recosntrucción de Superficies a partir de cortes planos de un objeto". Trabajo Especial de Grado. Escuela de Computación. Universidad Central de Venezuela. 1995.

3. William J. Schroeder, Jonathan A. Zarge, William E. Lorensen, "Decimation of Triangle Meshes". Computer G&raphics. Julio 1992.

4. Pranab K Banerjee, Arthur W toga. "Image Alignment by Integrated rotational ans traslational transformation matrix". Phys. Med. Biol. 39. 1994.

5. Radcliffe. "Pseudocorrelation: A fast, robust, absolute, grey-level image alignment algorithm". Medilac Physics. Vol. 21. No. 6. Junio 1994.

6. S. Hibbard, R. A. Grothe, T. L. Arnicar-Sulze. "Computed three-dimensional recosntruction of median-eminence capillary modules: image alignment and correlation". Journal of Microscopy. Vol. 171, Pt 1, Julio 1993, pp. 39-56.

7. Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, Wernen Stuetzle. "Mesh Optimization". Computer Graphics Annual Conference Series, 1993.