Una Implementación Eficiente del Algoritmo de *Marching Cubes*

Andrea Silvetti[1], Claudio Delrieux[2] y Silvia Castro[1]

[1]Dpto. de Ciencias de la Computación
[2]Dpto. de Ingeniería Eléctrica
Universidad Nacional del Sur
Avda. Alem 1253 - 8000 - Bahía Blanca
0291-4595135 - FAX 0291-4595136
e-mail: {silvetti,usdelrie,uscastro}@criba.edu.ar

RESUMEN

Uno de los algoritmos de rendering de volúmenes más difundido y utilizado es el denominado *marching cubes*, propuesto por Lorensen y Cline en 1987. En el mismo se busca extraer una superficie umbral a partir de una matriz volumétrica de datos escalares. Una *celda* en el espacio está delimitada por los ocho valores de sus vértices. Cada celda se clasifica según los valores de sus vértices respecto al valor umbral. Una celda contiene un trozo de la superficie umbral si por lo menos uno de sus vértices está por debajo del valor umbral y por lo menos otro está por encima. En este caso, cada uno de los ocho vértices de una celda puede asumir un valor por debajo o por encima del umbral. El total de todos los casos posibles es 2^8=256, pero por consideraciones de simetría se reducen en principio a solo 15.

Un problema que pronto fue detectado es la existencia de ambigüedades en la construcción de la superficie umbral entre celdas vecinas. Sin modificaciones al algoritmo original, algunos casos resultan en superficies con 'agujeros'. Esto se produce fundamentalmente cuando dos celdas adyacentes comparten una cara, pero la conexión de los cuatro puntos que dividen las aristas se realiza en una de ellas de forma tal que los puntos con valor superior al umbral queden separados mientras que en la otra cara esos puntos queda unidos. Hasta el momento, se han resuelto varios casos en lo que esto ocurre, pero ninguna solución ha sido exhaustiva, proponiéndose una proliferación de casos especiales para las estas situaciones. Este trabajo presenta una solución definitiva para el buen funcionamiento del algoritmo y algunas pautas para la implementación eficiente del mismo con sólo 30 casos.

PALABRAS CLAVE: VISUALIZACIÓN CIENTÍFICA. RENDERING DE VOLÚMENES. *MARCHING CUBES*

1. Introducción

La *visualización científica* es una de las tecnologías derivadas de las Ciencias de la Computación que actualmente están revolucionando con mayor fuerza las metodologías de investigación científica en todos sus campos [Cunigham90, Defanti90, Rosenblum89]. Por visualización se entiende el empleo de técnicas derivadas de la computación gráfica utilizadas para la representación de datos científicos de diverso tipo [Avila90, Nielson90]. Dentro de la investigación en visualización científica, la representación de datos volumétricos se destaca por las dificultades computacionales que plantea, pero al mismo tiempo concentra la mayor atención en la investigación actual [Ranjan94, Sakas92, Williams92]. Esto es así porque constituye una de las innovaciones más importantes y de mayor aplicabilidad al producir una adecuada representación gráfica computacional de datos que por una u otra razón no pueden representarse en términos de gráficos convencionales. La representación volumétrica de datos es actualmente de gran utlildad en la investigación científica en temas tan diversos como en matemática, medicina, ciencias naturales e ingeniería [Fuchs90, Hibbard90] y es utilizada para representar datos que pueden provenir de sensores, como en el caso de tomógrafos o de satélites, o bien pueden provenir de tareas computacionales anteriores, como por ejemplo de simulaciones o de análisis por elemento finito. Al mismo tiempo, los resultados de la visualización volumétrica de estos datos no son meramente una representación cuantitativa de los mismos, es decir, no se busca necesariamente la presentación fiel de valores. Por el contrario, se busca un entendimiento global de determinadas propiedades del modelo o de la simulación que produjo los datos.

Para encontrar representaciones adecuadas de conjuntos volumétricos de datos, se utilizan representaciones intermedias que puedan ser procesadas para producir una imagen. Estas representaciones intermedias son arbitrarias, y por lo tanto normalmente se eligen de manera que facilite el proceso intuitivo de la percepción. Las técnicas usuales de rendering de volúmenes están normalmente asociadas a una representación de los mismos en estructuras de *celdas* volumétricas o en *voxels* [Sakas92, Wright92]. Una celda es el espacio del volumen ocupado por ocho muestras vecinas, las cuales son interpretadas como los vértices de la celda. Un voxel, en cambio, representa el factor de ocupación que el sólido posee en una determinada fracción del espacio tridimensional. En gran parte de las aplicaciones usuales (por ejemplo en medicina, meteorología, etc.) la representación con voxels es el resultado natural de la adquisición o computación misma de los datos. Estos datos son eventualmente preprocesados para extraer y enfatizar adecuadamente las características intuitivamente adecuadas en una determinada aplicación y para un determinado propósito en su visualización (por ejemplo, destacar determinadas áreas en la representación visual de una tomografía). Una vez que los datos volumétricos están adecuadamente preparados para el rendering, el mismo procede según algoritmos de mayor o menor sofisticación.

Una de las primeras técnicas de rendering de volúmenes [Farrel83] consiste en graficar por capas el volumen de datos. Normalmente el volumen de datos se hace coincidir con los ejes del sistema de coordenadas del mundo, de modo que el eje *z* (hacia donde mira el observador) coincida con uno de los ejes del volumen de datos. Planos perpendiculares a dicho eje son entonces procesados de adelante hacia atrás. El procesamiento es sencillo, consistiendo en una proyección paralela de los datos al buffer de pantalla, utilizando alguna técnica de pseudocoloring [Rheigans92, Ware88] para asociar los valores a representar con colores de una paleta predeterminada (por ejemplo, asociar un determinado color a un determinado tejido). Cada voxel, en función de su valor, tiene a su vez una determinada transparencia, es decir que no es necesariamente opaco, permitiendo que se visualize parcialmente las partes del volumen que se encuentran detrás. La transparencia en cada dirección visual se computa acumulándola en un *alpha*-buffer de pantalla [Blinn94]. Para emular una proyección tridimensional, los datos de las capas se van desplazando una determinada distancia en *x e y* a medida que éstas son más distantes en el eje *z*. Esta técnica es bastante primaria, pero por esa misma razón es implementable directamente con hardware específico. Su mayor limitación consiste en que, al no existir un sólido propiamente dicho en ningún momento del procesamiento, no es

posible una representación con realismo, por ejemplo, la interacción con iluminantes o con otros objetos.

Otros métodos más sofisticados buscan extraer la *representación* de un objeto tridimensional a partir del volumen de datos. Una de las primeras técnicas [Ganapahty83] consiste en procesar capa por capa al volumen de datos, en función de un determinado valor umbral. De esa manera, es posible identificar en una capa dada aquellas celdas en las cuales ocurre una transición cercana al valor umbral. Entre dos capas adyacentes, entonces, es posible vincular los contornos para determinar un esqueleto de polígonos. El conjunto de polígonos encontrado entre todas las capas procesadas de esta manera constituye una representación del sólido con una estructura "intermedia", en este caso, una superficie. Esta estructura de polígonos permite la visualización de los datos originarios, y tiene la ventaja de ser una estructura "tradicional" en el sentido de la computación gráfica, es decir, es un conjunto de polígonos, el cual puede graficarse con los algoritmos usuales, utilizando cara oculta, sombreado, iluminación, etc. Sin embargo, esta técnica encuentra problemas cuando no es directo encontrar el esqueleto de polígonos entre dos capas sucesivas (por ejemplo si ocurren discontinuidades topológicas).

2. El algoritmo de Marching Cubes y trabajos relacionados

Otra solución, más estable con respecto a este tipo de problemas, es la denominada "*marching cubes"* [Lorensen87], en la cual se clasifican las celdas que pertenecen a una superficie umbral. Dado un valor umbral y una grilla cúbica conteniendo los datos del objeto a modelar, el algoritmo original la procesa considerando cada celda en un orden determinado, generando localmente una isosuperficie. Esta isosuperficie queda representada a través de triángulos cuyos vértices están en las aristas de las celdas que procesa. Cada arista de la celda puede tener a lo sumo una intersección con la isosuperficie, la cual se produce cuando el valor umbral queda acotado inferior y superiormente por los valores en los vértices de la arista. La forma de la superficie dentro de cada celda depende solamente de la combinación de puntos con valor mayor o menor al umbral en dicho cubo (puntos marcados o no marcados). Son posibles 2^8=256 combinaciones, pero este número se reduce a 128 asumiendo que dos configuraciones son iguales si se invierten los puntos marcados y no marcados, y se invierten las normales de los triángulos generados. Esta situación se denomina complementaria. Por lo tanto, sólo se consideran los casos con a lo sumo cuatro puntos marcados. Además, aplicando equivalencia por rotaciones, los casos posibles se reducen a 15 (Ver Figura 1). Esta idea original de Lorensen lleva a la implementación de un algoritmo veloz y sencillo [Watt92] que, sin embargo, en muchos casos produce resultados no deseados.

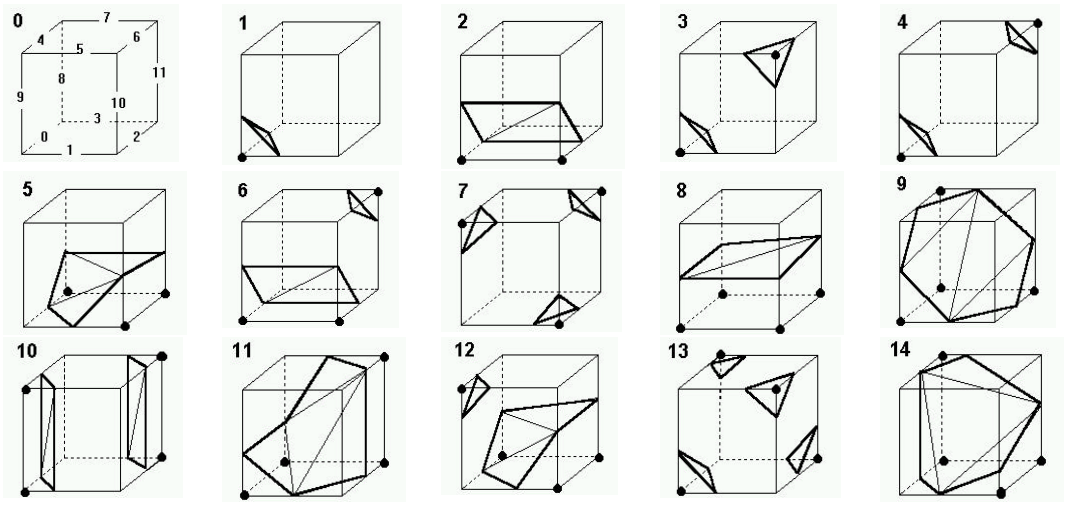


Figura 1: 15 configuraciones originales.

3. Agujeros: ¿Cuándo se producen? ¿Cómo solucionarlos?

Sin sin modificaciones del algoritmo original, algunos casos resultan en superficies con “agujeros” [Nielson91]. Llamaremos *cara ambigua* a una cara con puntos marcados en los extremos de una de sus diagonales, y puntos no marcados en los extremos de la otra diagonal. Por ejemplo, la cara superior de la configuración 10 (ver Figura 1) puede ser como se muestra en la Figura 2. Se ve entonces la necesidad de distinguir entre una cara ambigua en la cual la isosuperficie separa localmente los puntos marcados, de una cara ambigua donde dichos puntos estan dentro de la isosuperficie.

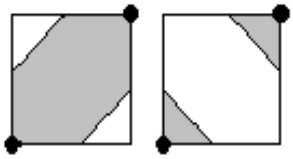


Figura 2: Conexiones posibles en una cara ambigua.

Sean dos celdas adyacentes que comparten una cara ambigua y tales que una de las celdas tiene a lo sumo cuatro puntos marcados mientras que la otra tiene al menos cuatro. Es decir, una de ellas es complementada y debe invertir los puntos marcados antes de ser procesada. De este modo, la cara compartida es tal que en una de las celdas queda con los dos puntos marcados separados mientras que en el otro quedan unidos. Dicha situación es conocida como “agujero” (Ver Figura 3).

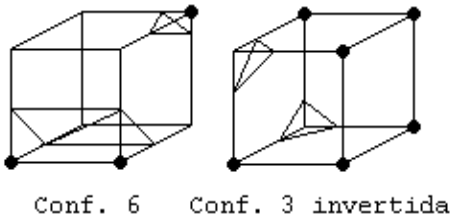


Figura 3: agujero.

Para lograr una superficie topológicamente correcta, las dos celdas en cuestión deben optar por la misma conexión y en función de esta decisión elegir la triangulación correcta en cada caso. Una forma de evitarlo es trabajar con las 256 configuraciones posibles. Sin embargo, es evidente que el algoritmo de obtención de la isosuperficie se vería seriamente comprometido en su velocidad y complejidad. Por lo tanto, estamos frente a la necesidad de agregar nuevas configuraciones a las 15 originales, sin que exista una proliferación de casos particulares.

En ese sentido, Nielson y Hamann [Nielson91] presentan una forma correcta de optar por una conexión para una cara ambigua. Se usa una variación bilineal de los valores escalares en direcciones paramétricas r y s sobre la cara ambigua Para obtener el valor escalar en cualquier punto de la cara con parámetro r y s (r,s): $0 \le r \le 1$, $0 \le s \le 1$ se usa la función de interpolación bilineal

$$\begin{bmatrix} \text{Valor}(0,0) & \text{Valor}(0,1) \end{bmatrix} \begin{bmatrix} 1-s \end{bmatrix}$$

$$Valor(r,s) = [1-r,r] \begin{bmatrix} Valor(1,0) & Valor(1,1) \end{bmatrix} \begin{bmatrix} s \end{bmatrix}$$

donde Valor(0,0), Valor(0,1), Valor(1,0) y Valor(1,1) son los valores en los cuatro vértices de la cara ambigua.

El punto (r,s) podría tomarse como el centro de la cara y por lo tanto, calcular el Valor(r,s) como ¼ (Valor(0,0)+Valor(0,1)+Valor(1,0)+Valor(1,1)). Si este valor es mayor o igual al umbral, es una cara 'Unida' (también referenciada como U), de lo contrario es 'Separada' (también referenciada como S). Los parámetros r_0, r_1, s_0 y s_1, todos pertenecientes al intervalo abierto 0..1 se obtienen partiendo de las coordenadas (x,y,z) de los puntos R_0, R_1, S_0 y S_1 por donde pasa la isosuperficie y de los vértices v_0, v_1, v_2 y v_3 de la cara (Ver Figura 4).

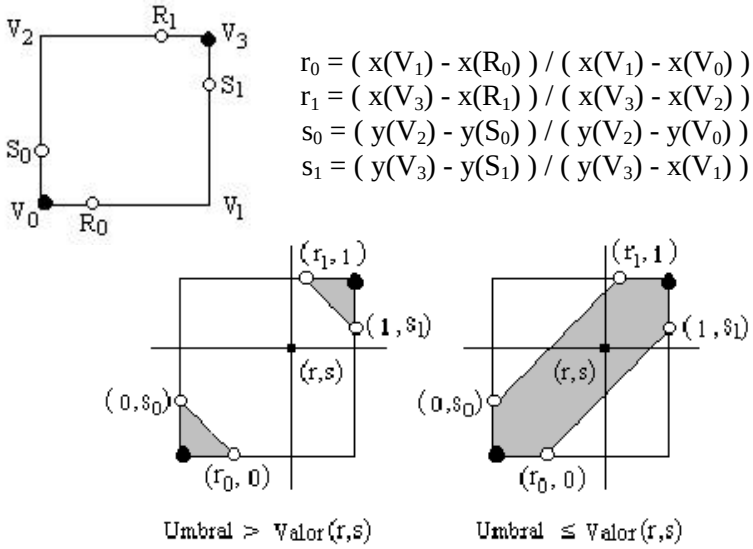


Figura 4: Cara Separada si Umbral > Valor(r,s) y cara Unida si Umbral = Valor(r,s).

Pese a que esta determinación permite solucionar casos simples de agujeros por caras ambiguas, cuando una celda tiene dos o más caras ambiguas (por ejemplo, las confiuraciones 10, 12 y 13), la idea no es directamente aplicable, por lo que surge la necesidad de agregar variantes particulares para cada una de estas configuraciones. De esa manera surge una proliferación de configuraciones que no solo no evita completamente la aparición de agujeros, sino que además hace que el algoritmo sea más lento y complejo.

4. Un nuevo acercamiento al problema

Como vimos, se han resuelto varias situaciones ambiguas en el algoritmo, pero como veremos aún no todas. Este trabajo presenta nuevas configuraciones, necesarias para el buen funcionamiento del algoritmo, y algunas pautas para la implementación eficiente del mismo con la menor cantidad posible de configuraciones. La base de este nuevo acercamiento consiste en determinar que, en algunos casos, además de las seis caras que delimitan a un voxel, necesitamos considerar una cara `diagonal' para evitar obtener superficies no esperadas. También es preciso distinguir entre caras ambiguas orientadas hacia la derecha y caras ambiguas orientadas hacia la izquierda (ver Figura 5).

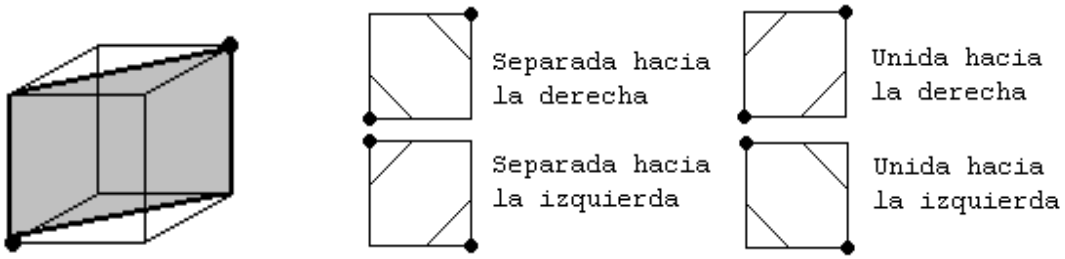
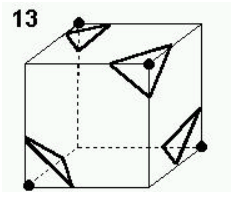


Figura 5: Cara diagonal ambigua y las distintas orientaciones y conexiones para caras ambiguas - S/,U/, S\ y U\

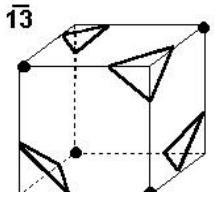
La Figura 6 muestra la configuración 13 que tiene las seis caras ambiguas y por tanto, es la configuración ideal para mostrar con cada cara, cómo mirarlas para decidir la orientación que tienen. Las caras del frente y del fondo están hacia la derecha y las de la izquierda, derecha, superior e inferior están hacia la izquierda. En el complemento, es exactamente al revés.

De las 15 configuraciones del algoritmo original, las configuraciones 3, 4, 6, 7, 10, 12 y 13 tienen caras ambiguas. Tal vez la configuración 4 parece no poseerla pero si se observa bien, tiene una cara ambigua “diagonal” formada por los extremos superior-frente-izquierdo, inferior-frente-izquierdo, superior-fondo-derecho e inferior-fondo-derecho. Una isosuperficie que intersecta una celda así puede ser un tubo que pasa por los extremos mencionados, o simplemente, dos partes aisladas que involucran sólo los extremos (ver en la
Figur las triangulaciones neceasarias para las configuraciones 4A y 4B). En otras configuraciones en las que está presente este tipo de cara, no es necesario tenerla en cuenta, pues es suficiente con analizar sólo las caras ambiguas que delimitan a la celda.

Todas las configuraciones con sólo cuatro puntos marcados y caras ambiguas, tienen la misma cantidad de caras orientadas hacia la derecha y caras orientadas hacia la izquierda que sus complementos salvo la configuración 13 que tiene dos caras hacia la derecha y cuatro hacia la izquierda mientras que su complemento posee cuatro caras hacia la derecha y dos hacia la izquierda. Esto nos da la pauta de la posible necesidad de analizar a ambos como casos diferentes, sobre todo, para lograr una implementación más eficiente con menos triangulaciones necesarias. Es precisamente este el motivo por el cual para la configuración 13 no son suficientes las triangulaciones que agregan Nielson y Hamann en su trabajo [Nielson91]; son necesarias dos más. Por otro lado, si analizamos los complementos como casos diferentes, por rotaciones veremos que cuatro de esas triangulaciones son innecesarias. Del mismo modo, que son innecesarias dos triangulaciones para la configuración 10 y una para la 12.



6 caras ambiguas:
2 hacia la derecha
4 hacia la izquierda



6 caras ambiguas:
4 hacia la derecha
2 hacia la izquierda

Figura 6

Para seguir con la misma línea de trabajo que en el caso de la configuración 13, podemos hacer lo mismo con las configuraciones 8, 9, 10, 11, 12 y 14 que también tienen igual cantidad de vértices

con valor superior o igual al umbral que sus complementos; sin embargo, sólo lo haremos con las configuraciones 10 y 12 que son las que tienen caras ambiguas. Analizaremos cuántas triangulaciones diferentes admite cada elemento en función de la cantidad de caras problemáticas que posee.

Config.	Orientación	Combinaciones posibles						
3, 4, 6	1 /	1S	1U					
7	2/ + 1\	3S	3U	2S y 1U	2U y 1S			
10, 12	1/ + 1\	2S	2U	1S y 1U	1U y 1S			
13	2/ + 4\	6S	6U	5S y 1U	1S y 5U	4S y 2U	2S y 4U	3S y 3U

No sólo es suficiente con analizar las combinaciones posibles con un número determinado de caras, sino que cuando en una configuración dos o más caras ambiguas se unen, es preciso analizar si la unión es del mismo modo en el complemento. Ver en
Figura8 y Figura9 las triangulaciones mencionadas de aquí en adelante.

Para los casos 3,4 y 6 no tenemos problemas pues sólo hay una cara ambigua.

3 - FRENTE ambigua	
3A	S/
3B	U/
3A complemento	S\
3B complemento	U\

4 - DIAGONAL ambigua	
4A	S/
4B	U/
4A complemento	S\
4B complemento	U\

6 – DERECHA ambigua	
6A	S/
6B	U/
6A complemento	S\
6B complemento	U\

En el caso 7, tenemos tres caras ambiguas que se unen en un único punto no marcado, o marcado en el complemento. Esto no ocasiona problemas pues es una configuración con distinta cantidad de puntos marcados que su complemento.

	SUP	DER	FRE
7A	S/	S/	S
7B	S/	U/	U
7C	U/	S/	S
7D	U/	U/	U

	SUP	DER	FRE
7A complemento	U	U	U/
7B complemento	U	S	S/
7C complemento	S	U	U/
7D complemento	S	S	S/

En el caso 10, con dos caras ambiguas que no se unen, es suficiente trabajar con dos triangulaciones. Las triangulaciones 10a y10b no son necesarias pues son las mismas que 10A y 10B aplicadas convenientemente, esto es, rotamos hasta obtener el complemento de 10A o de 10B, triangulamos y le cambiamos las normales a los triángulos obtenidos. Ante la combinación que representa el complemento de 10A o de 10B triangulamos y cambiamos las normales. Finalmente cuando se presenta la combinación que representa el complemento de 10a o de 10b rotamos hasta obtener el 10A o 10B, triangulamos pero no cambiamos las normales de los triángulos obtenidos.

	SUPERIOR	INFERIOR
10A	S/	S\
10B	S/	U\
10a	U/	U\
10b	U/	S\

	SUPERIOR	INFERIOR
10A complemento	U\	U/
10B complemento	U\	S/
10a complemento	S\	S/
10a complemento	S\	U/

En el caso 12, con dos caras ambiguas que se unen en una arista, debemos tener cuidado cuando las conexiones se eligen separadas (S) en una cara y unidas (U) en la otra ya que la arista que las une puede quedar con un punto marcado en un extremo o en el contrario. En el caso de conexiones iguales en ambas caras, el análisis a realizar es más simple. La triangulación 12a es una rotación adecuada de 12A tal como la explicada con 10A y 10a (la rotación deja como cara izquierda la del frente y viceversa); sin embargo, no podemos decir lo mismo con respecto a 12B y 12C, ya que si a 12B le aplicamos el mismo procedimiento, no llegamos a 12C pues no sólo cambia la orientación de las caras, sino también el tipo de conexión.

	IZQUIERDA	FRENTE
12A	S/	S
12B	S/	U
12C	U/	S
12a	U/	U

	IZQUIERDA	FRENTE
12A complemento	U	U/
12B complemento	U	S/
12C complemento	S	U/
12a complemento	S	S/

Finalmente, en el caso 13 tenemos seis caras ambiguas, cuatro orientadas hacia un lado y dos hacia el otro. Cuando la conexión deja tres unidas (U) y tres separadas (S), hay que ver muy bien todas las combinaciones posibles ya que tenemos cuatro caras hacia la izquierda y dos hacia la derecha mientras que en el complemento tenemos cuatro hacia la derecha y dos hacia la izquierda. Debemos entonces distinguir entre los casos en que las tres caras se unen en un punto marcado o en un punto no marcado, o en una cara ambigua hacia la derecha o en una cara ambigua hacia la izquierda, ver Figura 7).

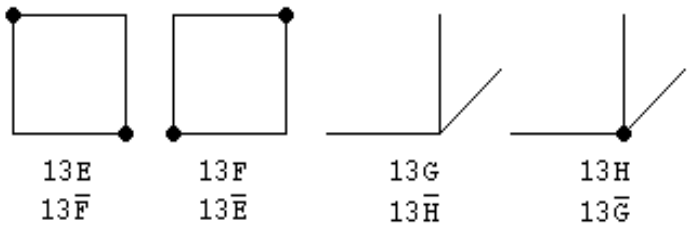


Figura 7

En 13A todas las caras se conectan de igual modo por lo tanto con una triangulación y rotaciones y/o cambios de normales adecuados es suficiente para lograr tanto 13A como 13a y sus complementos. De manera similar resolvemos 13B que sólo tiene una cara ambigua. 13C y 13D se distinguen ya que las caras con conexión unida (U) quedan en una caso totalmente distanciadas y en el otro se juntan a través de una arista; 13c y 13d se obtienen a partir de 13C y 13D aplicando el procedimiento ya conocido.

13E, 13F, 13G y 13H son todas las combinaciones que involucran tres caras con conexión separada (S) y tres caras con conexión unida (U). De manera similar a lo ocurrido con 12B y 12C, ninguno de estos casos puede obtenerse a partir de otro aplicando el procedimiento dado. Veamos por qué no es posible esto.

En 13E se unen las caras (superior\ - derecha\ - inferior\) y las caras (frente/ - izquierda\ - fondo/); en el primer caso, todas las caras están orientadas hacia la izquierda mientras que en el segundo, sólo la cara que las une está orientada hacia la izquierda. En el complemento, (superior/ - derecha/ - inferior/) están todas orientadas hacia la derecha mientras que (frente\ - izquierda/ - fondo\) sólo tiene la cara que las une orientada hacia la derecha. Son cuatro casos totalmente diferentes.

La triangulación 13E considera tres caras (U) todas orientadas hacia la izquierda y 13F considera tres caras (U) pero dos orientadas hacia la derecha que se unen en una orientada hacia la izquierda.

Sus complementos consideran tres caras (S) orientadas todas hacia derecha y tres caras (S) dos hacia la izquierda y la que las une hacia la derecha respectivamente.

En 13 G se unen las caras (superior\ - izquierda\ - fondo/) y las caras (inferior\ - derecha\ - frente/); en ambos caso hay dos caras orientadas hacia la izquierda y una hacia la derecha pero en el primer caso, se unen en un punto marcado y en el segundo, lo hacen en un punto no marcado. En el complemento, si se unían en un punto marcado se unen en uno no marcado y viceversa; además, ahora tenemos dos caras hacia la derecha y una hacia la iquierda. Otra vez son cuatro casos bien marcados.

La triangulación 13G considera tres caras (U) unidas en un punto marcado y 13H considera tres caras (U) pero unidas en un punto no marcado. Sus complementos consideran tres caras (S) unidas en un punto no marcado y tres caras (S) unidas en un punto marcado respectivamente.

	SUP	INF	IZQ	DER	FRE	FON
13A	S\	S\	S\	S\	S/	S/
13B	S\	S\	S\	U\	S/	S/
13C	S\	S\	U\	U\	S/	S/
13D	S\	S\	S\	U\	U/	S/
13E	U\	U\	S\	U\	S/	S/
13F	S\	S\	U\	U\	U/	S/
13G	U\	S\	U\	S\	S/	U/
13H	U\	S\	U\	S\	U/	S/
13a	U\	U\	U\	U\	U/	U/
13b	U\	U\	U\	S\	U/	U/
13c	U\	U\	S\	S\	U/	U/
13d	U\	U\	U\	S\	S/	U/

	SUP	INF	IZQ	DER	FRE	FON
13A complemento	U/	U/	U/	U/	U\	U\
13B complemento	U/	U/	U/	S/	U\	U\
13C complemento	U/	U/	S/	S/	U\	U\
13D complemento	U/	U/	U/	S/	S\	U\
13E complemento	S/	S/	U/	S/	U\	U\
13F complemento	U/	U/	S/	S/	S\	U\
13G complemento	S/	U/	S/	U/	U\	S\
13H complemento	S/	U/	S/	U/	S\	U\
13a complemento	S/	S/	S/	S/	S\	S\
13b complemento	S/	S/	S/	U/	S\	S\
13c complemento	S/	S/	U/	U/	S\	S\
13d complemento	S/	S/	S/	U/	U\	S\

La Figura 8 muestra las 30 triangulaciones necesarias para el buen funcionarmiento del algoritmo de Marching Cubes y la Figura muestra las 7 triangulaciones que parecen ser necesarias pero que pueden resolverse por rotaciones y cambios de normales de algunas de las 30 triangulaciones necesarias.

5. Algoritmo Marching Cubes ([Silvetti99])

ALGORITMO MARCHINGCUBES(Datos,Umbral,ArchivoDeTriángulos)

Por cada celda
número= armar número binario
si *número* pertenece al intervalo cerrado [1..254]
entonces
si *número* pertenece al *Grupo de Complementos*
entonces
número=255-*número*
InvertirNormales=true
sino
Grupo de Equivalenciaia= grupo en el que figura el número (entre 1 y 14)
Posición= posición en la que debe analizarse el número (entre 1 y 24)
índice= IndiceDeTabla(*GrupoDeEquivalencia*, *Posición,MatrizNudos)*
ObtenerTriangulaciónDeTabla(índice)
Por cada triángulo
Adaptarlo a la *Posición* usando MatrizPosiciones
si *InvertirNormales*=true entonces Invertirlas
Buscar las coordenadas correctas de los puntos de los triángulos
Guardarlo en archivo de Triangulos
Fin MARCHINGCUBES

FUNCIÓN INDICE DE TABLA(GrupoDeEquivalencia, Posición,MatrizNudos)

Si el *Grupo de Equivalenciaia* es 1,2,5,9,11 o14
entonces
la asignación es simple porque sólo hay una posible triangulación a usar
sino {grupos de equivalencia con caras ambiguas}
PasarAPosicion1(Densidades,Posicion,MatrizDeNudos,DensidadesEn1);
si *Grupo de Equivalenciaia* es
3: si Cara separada('frente')
ent 3A
sino 3B
4: si Cara separada('diagonal')
ent 4A
sino 4B
6: si Cara separada('derecha')
ent 6A
sino 6B
7: *contadorCaras*=CaraSeparada('frente')+CaraSseparada('derecha')+
Cara Separada('superior')
si *contadorCara* es
3: 7A
2: 7B modificando la posición para que la cara unida quede como cara superior
1: 7C modificando la posición para que la cara separada quede como cara superior
0: 7D

10: si CaraSeparada('inferior')
entonces
si CaraSeparada('superior')
entonces 10A
sino 10B
sino
modificar posición según lo explicado en sección 6
si CaraSeparada('superior')
entonces 10B
sino 10A
invertir normales

12: *contadorCaras*=CaraSeparada('frente')+CaraSseparada('izquierda)
si *contadorCara* es
2: 12A
1: si CaraSseparada('izquierda)
ent 12B
sino 12C
0: modificar posición según lo explicado en sección 6 pero para el grupo 12
entonces 12A
invertir normales

13: *contadorCaras*=
CaraSeparada('Superior')+CaraSeparada('Inferior')+CaraSeparada('Frente')+
CaraSeparada('Fondo')+CaraSeparada('Izquierda')+CaraSeparada('Derecha')
si *contadorCara* es
0,6: 13A
si cont = 0 ent modificar la posición e invertir normales
1,5: 13B
si la cara con conexión diferente al las otras cinco es separada
entonces invertir las normales
modificar la posición para que la cara con conexión diferente al las otras cinco quede hacia la derecha
2,4: si la dos caras con igual conexión son
opuestas: 13C modificar la posición para que las dos caras queden como izquierda y derecha
adyacentes:13D modificar la posición para que las dos caras queden como frente y derecha
si las dos caras están separadas
entonces invertir normales
3: si 3 de las caras
son tal que dos de ellas son opuestas:
si la tercera de esas caras está orientada hacia la izquierda
entonces 13E
sino 13F
concurren las tres a un punto:
si el punto es no marcado (inferior al umbral)
entonces 13G
sino 13H
sino 13H
si la posición original es 2
entonces invertir normales
modificar posiciones usando los arreglos respectivos para que las caras analizadas queden orientadas como en 13E, 13F, 13G y 13H.

Fin

Procedimiento PasarAPosicion1(Densidades,Posicion,MatrizDeNudos,DensidadesEn1);
{Densidades: tiene las densidades en los 8 nudos del voxel}
{DensidadesEn1: sale con las densidades en los 8 nudos como si fuese en posición 1}
{el cambio solo se hace para consultar por caras ambiguas pero no al triangular}
Desde k=1 hasta 8
DensidadesEn1[k]=[Densidades[1,MatrizDeNudos[Posicion,k]]
Fin

Observemos que 'modificar la posición…' implica usar un procedimiento muy simple que asigna la nueva posición del siguiente modo: en el caso del grupo de equivalencia 10 son necesarios mantener dos arreglos; el primero, con las seis posiciones que pueden usarse originalmente para cada una de las combinaciones posibles, y el segundo, con las seis posiciones a asignar respectivamente. El procedimiento entonces busca la posición original en el primer arreglo devolviendo el índice que le permitirá acceder al segundo arreglo para obtener la posición final.

Trabajando de este modo, necesitamos tres arreglos de 8 componentes para el grupo de equivalencia 7; dos arreglos de 6 componentes para el grupo 10, dos arreglos de 12 componentes para el grupo y 12 y para el grupo 13 necesitamos los arreglos que se muestran a continuación

Posición7Frente	= [1 3 5 7 9 11 13 15]	con las posiciones originales
Posición7Superior	= [8 23 16 17 4 21 12 19]	estas posiciones ponen al frente la cara superior
Posición7Derecha	= 18 10 20 2 24 14 22 6]	estas posiciones ponen al frente la cara derecha
Posición10	= [1 6 7 10 18 24]	con las posiciones originales
Posición10Comp	=[7 10 1 6 24 18]	en esta posición vemos lo que veríamos en la posición original invirtiendo puntos marcados por no marcados y viceversa.
Posición12	= [1 7 21 20 5 24 16 8 22 10 12 9]	con las posiciones originales
Posición12Comp	=[11 13 18 19 15 23 6 14 17 4 2 3]	en esta posición vemos lo que veríamos en la posición original invirtiendo puntos marcados por no marcados y viceversa.

En los arreglos usados para el grupo de equivalencia 13, el orden de los elementos en cada uno de ellos está dado según el orden secuencial en que se analizan las distintas combinaciones posibles para conexiones de caras ambiguas en el algoritmo.

TreceB = [2 6 3 7 21 17] se usa cuando hay 5 caras separadas y la posición original es 2 o hay una cara separada y la posición original es es 1
TreceB = [18 22 8 4 5 1] se usa cuando hay 5 caras separadas y la posición original es 1 o hay una cara separada y la posición original es es 2 (como un complemento)
TreceD = [3 15 10 6 11 7 2 14 23 17 21 19] y TreceC=[2 3 17] se usan cuando hay 4 caras separadas y la posición original es 2 o hay dos caras separadas y la posición original es es 1
TreceD = [13 1 8 12 5 9 16 4 20 18 22 24] y
TreceC = [18 4 1] se usan cuando hay 4 caras separadas y la posición original es 1 o hay dos caras separadas y la posición original es es 2
TreceE = [1 5 4 8 3 7] se usa cuando tres de las caras se juntan en una de ellas y posición original es 1
TreceF = [1 5 4 8 3 7] se usa cuando tres de las caras se juntan en una de ellas y posición original es 2
TreceG = [7 3 1 4] se usa cuando tres de las caras se juntan en un nudo y posición original es 1
TreceH = [1 5 3 6] se usa cuando tres de las caras se juntan en un nudo y posición original es 1

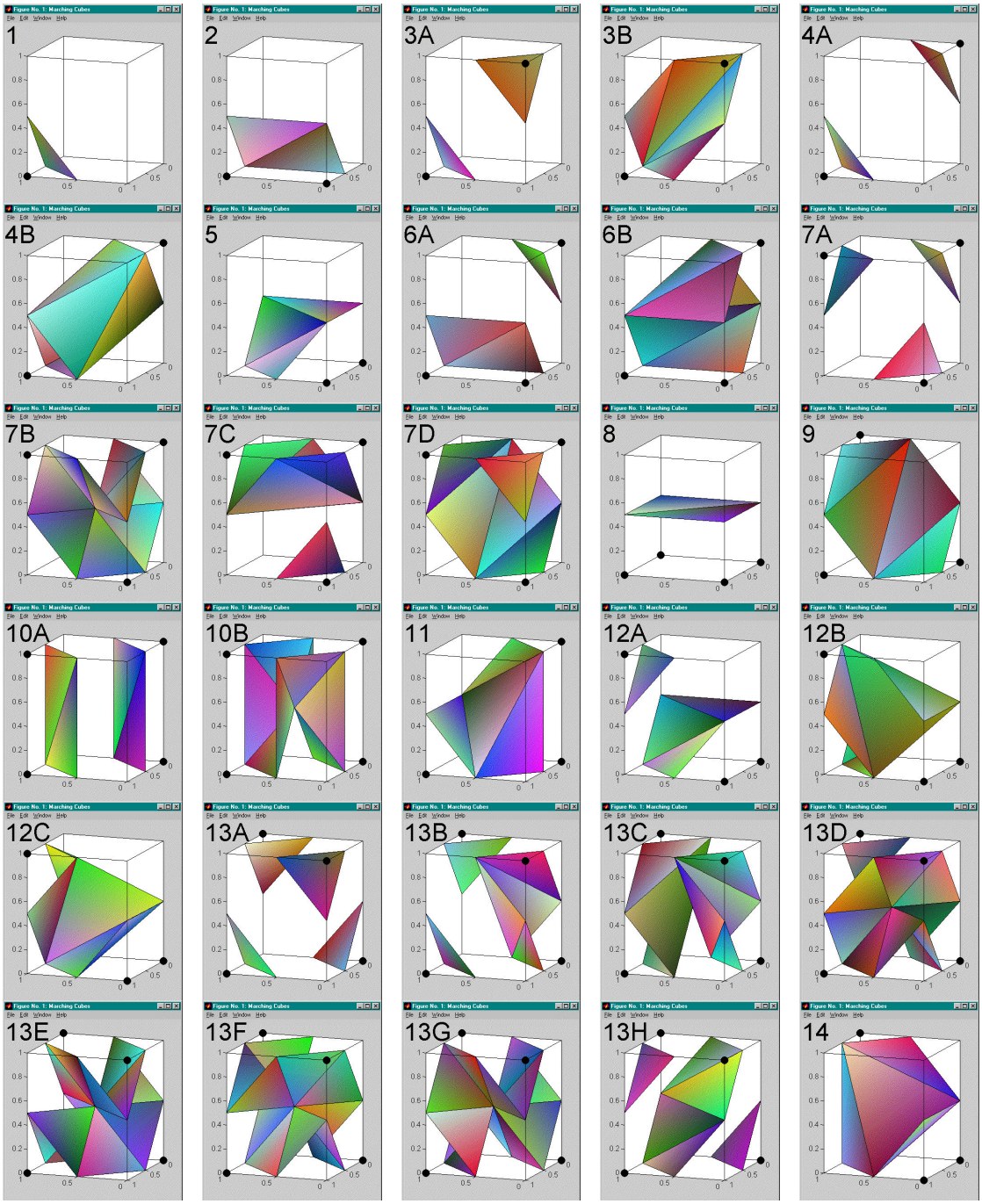


Figura 8: 30 triangulaciones para el algoritmo de Marching Cubes

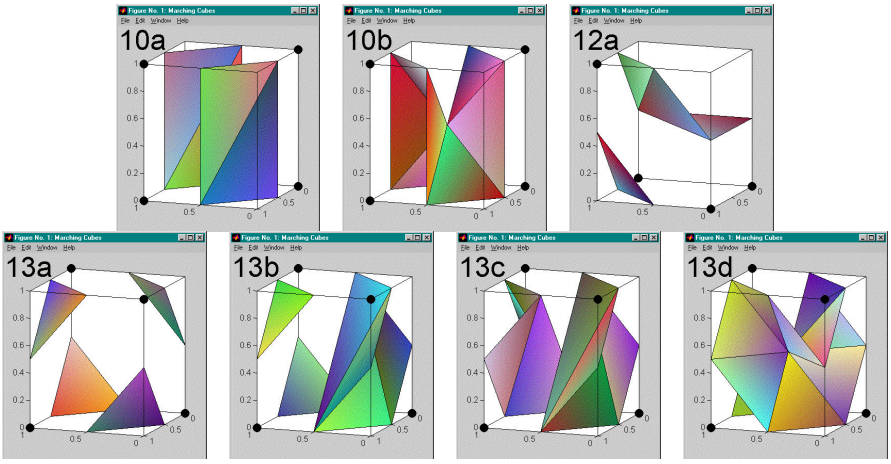


Figura 9: 7 triangulaciones innecesarias

6. Conclusiones

Se presentó una solución exhaustiva al problema de los “agujeros” y las ambigüedades en el algoritmo de marching cubes. La misma contempla todos los casos posibles de celdas vecinas, las cuales, teniendo en cuenta todas las rotaciones y simetrías especulares, se reducen a 30 casos posibles. Esta cantidad, si bien es mayor que la solución original de Lorensen y Cline, es en cambio menor a otras soluciones propuestas recientemente. En particular se muestra que algunos casos considerados por Nielsen y Hamman son innecesarios, pero que su algoritmo tampoco está libre de ambigüedades. Al mismo tiempo, se desarrolló un algoritmo de gran eficiencia que implementa la solución presentada.

Uno de los problemas remanentes en la filosofía general del algoritmo de marching cubes es que el resultado del mismo es una colección de triángulos, los cuales en general no producen un resultado geométricamente adecuado si la resolución del conjunto volumétrico de datos es baja. Por lo tanto, estamos investigando la posibilidad de que el algoritmo genere *parches triangulares* en vez de triángulos. Los mismos son superficies de segundo orden, es decir, superficies curvadas, las cuales producen un resultado visual de mucha mayor calidad geométrica, y, al mismo tiempo, se les puede aplicar un modelo de iluminación y sombreado más rico que en el caso de triángulos planos. Para encontrar parches triangulares a partir de los datos generados en el algoritmo de marching cubes se requiere conocer no solo la posición sino también la dirección de cambio geométrico en las celdas vecinas, lo cual genera en principio una cantidad no trivial de nuevos casos a considerar.

7. Bibliografía

[Avila92] R. S. Avila, l. M. Sobierajski, and A. E. Kaufman. Towards a Comprehensive Volume Visualization System. In Visualization `92 Proceedings, pags. 13--20, Los Alamitos, CA, 1992. IEEE Technical Committee on Computer Graphics, IEEE Computer Society Press.

[Blinn94] James F. Blinn. Compositing I-Theory. IEEE Comp Graphics and Applications, 14(5):83--87, 1994.

[Cuningham90] S. Cunningham, J. R. Brown, and M. McGrath. Visualization in Science and Engineering Education, en [Nielson90] pags. 48--58.

[Defanti90] T. A. Defanti, M. D. Brown, and B. H. McCormick. Visualization: Expanding Scientific and Engineering Research Opportunities. En [Nielson90], pags. 32--47.

[Farrel83] J. Farrell. Colour Display and Interactive Interpretation of Three-Dimensional Data. IBM Journal of Research and Development, 27(4):356--366, 1983.

[Fuchs90] H. Fuchs, M. Levoy, and J. K. Lam. Interactive Visualization of 3D Medical Data. En [Nielson90], pags. 140--146. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Ganapathy83] S. Ganapathy and T. Dennehy. A New General Triangulation Method for Planar Contours. ACM Computer Graphics, 16(3):69--75, 1983.

[Hibbard90] W. Hibbard and D. Santek. Visualizating Large Meteorological Data. En [Nielson90], pags. 147--152. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Lorensen87] Lorensen W. E., and Cline H. E., 'Marching Cubes: A Hight-Resolution 3D Surface Construction Algorithm', *Computer Graphics*, Vol. 21, Nº 4, July 1987, pp.163-169.

[Nielson90] G. M. Nielson and B. D. Shriver (editores). Visualization in Scientific Computing. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Nielson91] Nielson, G. M., and Hamann B., 'The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes', Proceedings of the *IEEE* Visualization '91 Conference, October 1991, 83-91.

[Ranjan94] V. Ranjan and A. Fournier. Volume Models for Volumetric Data. IEEE Comp, 27(7):28--36, 1994.

[Rheingans92] P. Rheingans. Color, Change and Control for Quantitative Data Display. In Visualization `92 Proceedings, pages 252--259, Los Alamitos, CA, 1992. IEEE Technical Committee on Computer Graphics, IEEE Computer Society Press.

[Rosenblum89] L. Rosenblum. Scientific Visualization at Research Laboratories. IEEE Comp, 22(8):68--100, 1989.

[Sakas92] G. Sakas and J. Hartig. Interactive Visualization of Large Scalar Voxel Fields. In Visualization `92 Proceedings, pags. 29--38, Los Alamitos, CA, 1992. IEEE Technical Committee on Computer Graphics, IEEE Computer Society Press.

[Silvetti99] Silvetti, A., Castro, S y Delrieux, C, 'Marching Cubes: una implementación eficiente que contempla todos los casos'. WICC 99, San Juan, Mayo de 1999.

[Ware88] Colin Ware. Color Sequences for Univariate Maps: Theory, Experiments and Principles. IEEE Computer Graphics and Applications, 8(5):41--49, 1988.

[Watt92] A. Watt y M. Watt, 'Advanced Animation and Rendering Techniques', Addison-Wesley, London, 1992.

[Williams92] R. D. Williams, F. L. Wefer, and T. E. Clifton. Direct Volumertic Visualization. In Visualization `92 Proceedings, pags. 99--106, Los Alamitos, CA, 1992. IEEE Technical Committee on Computer Graphics, IEEE Computer Society Press.

[Wright92] J. R. Wright and J. C. Hsieh. A Voxel-based, Forward-Projection Algorithm for Rendering Surface and Volumetric Data. In Visualization `92 Proceedings, pages 340--348, Los Alamitos, CA, 1992. IEEE Technical Committee on Computer Graphics, IEEE Computer Society Press.