SHEAR-WARP:
UNA IMPLEMENTACIÓN EFICIENTE

Rhadamés Carmona
Laboratorio de Computación Gráfica. Escuela de Computación. Facultad de Ciencias
Universidad Central de Venezuela. Apartado 47002. Los Chaguaramos 1041-A.
Caracas-Venezuela. e-mail: rcarmona@strix.ciens.ucv.ve.
Proyecto No. PI03-13-4503-1999 financiado por el CDCH, UCV.

RESUMEN

Philippe Lacroate en 1995 logró diseñar e implementar el primer algoritmo de visualización volumétrica en tiempo real, que generaba una imagen por segundo en una plataforma Silicon Graphics, utilizando la factorización *Shear-Warp*. En este trabajo se presentan detalles de implementación del Shear-Warp, que permiten mejorar la calidad de la imagen en tonos de grises, y obtener un tiempo de respuesta adecuado, generando hasta diez imágenes por segundo en una estación NT, y hasta tres en un PC de bajo costo. Entre los detalles merecen citarse la cuantización de normales vía subdivisión de un icosahedro, mejoras en la estructura de datos de la imagen intermedia, e implementación del *warping* en dos versiones: usando el *hardware* gráfico vía *texture mapping*, y en *software* mediante un algoritmo incremental.

Palabras Claves: visualización volumétrica, visualización científica, Shear-Warp, cuantización de vectores.

1. INTRODUCCIÓN

La visualización volumétrica es una técnica para visualizar arreglos volumétricos. Los datos de entrada suelen provenir de resonancia magnética, tomografía computarizada, microscopía electrónica, información 3D de yacimientos, suelos, etc. Sus aplicaciones en el área médica y visualización científica son claras, pero se ven limitadas por el alto costo computacional. Implementaciones convencionales del conocido algoritmo de Ray-Casting suelen generar cada imagen en alrededor de 100 segundos[1]. Investigadores se han dado la tarea de mejorar el tiempo de respuesta explotando la forma en que están almacenados los datos, reduciendo el tiempo en hasta 10 segundos. Finalmente, Philippe Lacroate en el año 1995, explota la coherencia en el espacio objeto e imagen, logrando un tiempo de respuesta de un segundo por imagen[1]. Lacroute recopila detalles importantes de implementación para acelerar la ejecución del visualizador volumétrico, en la que se destacan: la factorización Shear-Warp, la forma de almacenar los datos, y la cuantización de vectores para el cálculo del modelo de iluminación *Phong*.

En este trabajo, se proponen cambios en la tres direcciones. Por un lado, se presenta una forma alterna de almacenar y actualizar los datos en la imagen intermedia, facilitando el acceso rápido y secuencial de los *pixels* transparentes. Esto se logró mejorando la idea de compresión de camino[1], por mantener doblemente enlazados los pixels transparentes. Se mejora además la técnica para cuantizar vectores, a

través de la subdivisión recursiva de un icosahedro. Esto hace más uniforme la cuantización, lo que redunda en mejora de la calidad de la imagen, sin degradar el tiempo de respuesta. Por último, el warping fue implementado tanto en software mediante un algoritmo incremental, como usando el hardware gráfico mediante texture mapping (aplicación de textura). El warping en hardware se implementó definiendo un polígono, con la textura de la imagen intermedia. Los vértices del polígono son obtenidos aplicando la matriz de warping a los cuatro vértices de la imagen intermedia.

Se hicieron mediciones de tiempo de respuesta del algoritmo, probando arreglos volumétricos de 94, 100 y 113 cortes (cada corte de 256x256). La rata de despliegue obtenida fue de hasta diez imágenes por segundo en una estación de trabajo Intergraph con sistema operativo NT. Adicionalmente, se hicieron pruebas en un PC de bajo costo, en donde se obtuvo hasta casi tres imágenes por segundo.

Se compararon los tiempos de respuesta del warping en las dos implementaciones. En la estación de trabajo, los resultados en tiempo usando texture mapping resultaron inferiores que el implementado por software, cuando la imagen intermedia era de 512x512, y la de salida 256x256. Esto se debió a que la transferencia de la imagen intermedia a la memoria de textura cuadruplica en tiempo a la transferencia de la imagen de salida. Al escalar la imagen de salida a 512x512, los tiempos de transferencia se igualaron en ambos casos, pero a diferencia del uso de texture mapping, el tiempo de warping vía software aumentó drásticamente, en cuyo caso resultó muchos más eficiente el texture mapping.

El trabajó es presentado en varias secciones. En la sección 2 se muestra un resumen de la matemática envuelta en la factorización Shear-Warp. Luego en la sección 3, se explica una de las técnicas usadas para mejorar la eficiencia del visualizador volumétrico: la cuantización de normales. En la sección 4 se resume el algoritmo de visualización volumétrica usando Shear-Warp en nueve pasos. En la sección 5 se muestra el aporte principal de este trabajo, que son los detalles de esta implementación particular. La sección 6 muestra los resultados obtenidos principalmente en una estación de trabajo Intergraph, en donde se hacen mediciones de tiempo con distintos datos de prueba, y distintos parámetros del algoritmo. Finalmente, se presentan las conclusiones y trabajos a futuro.

2. FACTORIZACIÓN SHEAR-WARP

La factorización Shear-Warp consiste en transformar el volumen a un sistema de coordenadas intermedio (espacio objeto desplazado, o *sheared object space*) en el que todos los rayos visuales son paralelos al tercer eje de coordenadas. Esto permite la generación de una imagen intermedia, proyectando los cortes desplazados sobre un plano, accediendo la data volumétrica corte por corte desde el más cercano al más lejano. Luego, se aplica un warping a la imagen desplazada para generar la imagen final (ver Fig. 2.1).

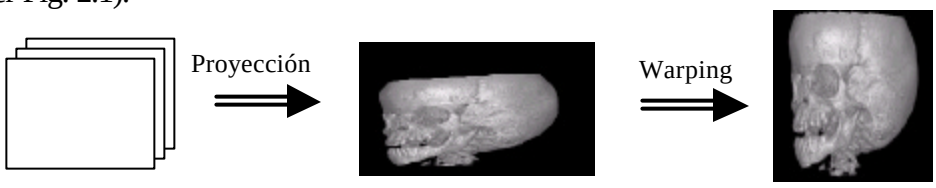


Figura 2.1: Visualización volumétrica usando la factorización Shear-Warp

Matemáticamente, la factorización Shear-Warp consiste en expresar la matriz de visualización M_{view} como un producto de dos matrices $M_{warp} * M_{shear}$. La matriz M_{shear} es usada para desplazar los cortes y generar la imagen intermedia, y M_{warp} es la matriz de warping. En la Fig. 2.2 se muestran los sistemas de coordenadas involucrados en el proceso.

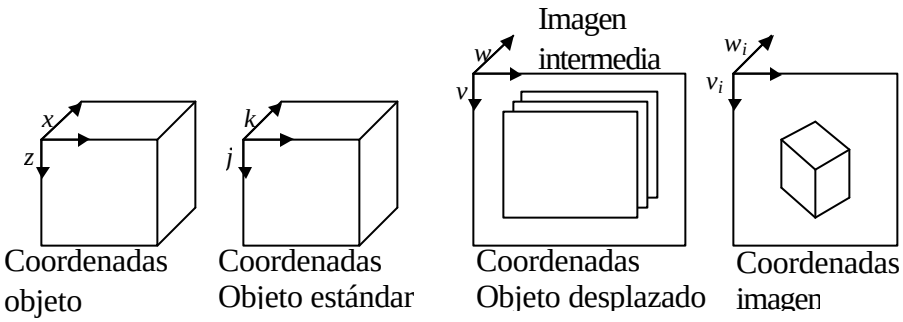


Figura 2.2: Sistemas de coordenadas involucrados en la derivación de la factorización Shear-Warp

El algoritmo de visualización volumétrica mediante Shear-Warp, para el caso de proyección paralela, puede resumirse en los siguientes pasos:

a. Encontrar el eje principal: este es el eje en el espacio objeto que forma el menor ángulo con la dirección del vector de visualización. El vector de visualización v_o en el espacio objeto puede expresarse como

$$v_o = \begin{pmatrix} m_{12} * m_{23} - m_{22} * m_{13} \\ m_{21} * m_{13} - m_{11} * m_{23} \\ m_{11} * m_{22} - m_{21} * m_{12} \end{pmatrix} \qquad \{1\}$$

en donde los m_{ij} son lo elementos de la matriz M_{view}. Así, la determinación del eje principal c es resuelto mediante la siguiente expresión:

$$c = \underset{x,y,z}{Max}\left(v_{o,x}, v_{o,y}, v_{o,z}\right). \qquad \{2\}$$

b. Transformación al sistema de coordenadas estándar: una vez conocido el eje principal, la idea es transformar el sistema de coordenadas objeto al sistema de coordenadas estándar, en donde el eje principal es siempre el tercer eje de coordenadas (eje k). En la Tabla 2.1 se muestra la matriz de permutación que transforma coordenadas objeto a coordenadas objeto estándar en cada caso. Se muestra además la equivalencia entre ambos sistemas. Así, la transformación del sistema de coordenadas objeto al sistema de coordenadas objeto estándar, puede ser expresado en general como $[i,j,k,0]^t = P*[x,y,z,0]^t$.

Igualmente, la matriz de transformación M_{view} permutada se expresa como $M_{view}' = M_{view} * P^{-1}$. El vector de visualización v_{so} en el sistema de coordenadas estándar se deduce como:

$$v_{so} = P * v_o = \begin{pmatrix} m'_{12} * m'_{23} - m'_{22} * m'_{13} \\ m'_{21} * m'_{13} - m'_{11} * m'_{23} \\ m'_{11} * m'_{22} - m'_{21} * m'_{12} \end{pmatrix}$$

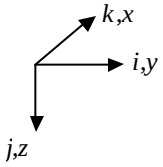
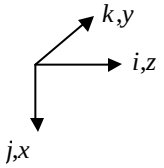
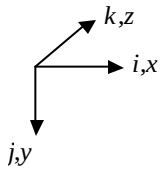
	Eje principal x	Eje principal y	Eje principal z
Matriz de Permutación P	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
Equivalencia entre los dos sistemas de coordenadas			

Tabla 2.1: Correspondencia entre el sistema de coordenadas objeto y el sistema de coordenadas objeto estándar.

c. Transformación al sistema de coordenadas desplazado: los factores de shear s_i, s_j en los ejes i, j respectivamente, son expresados como $s_i=-v_{so,i}/v_{so,k}$ y $s_j=-v_{so,j}/v_{so,k}$. En la Fig. 2.3 se muestra gráficamente cómo se determina el factor de shear s_i (Fig. 2.3a), así como el efecto que la transfomación M_{shear} en el volumen (Fig. 2.3b).

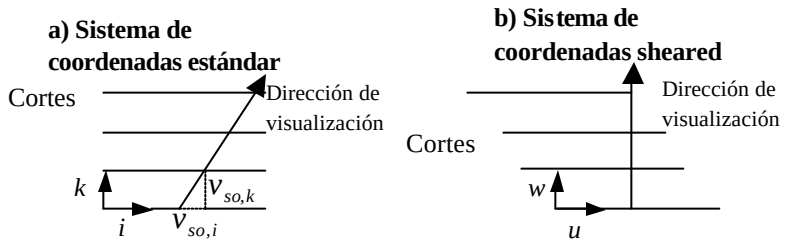


Figura 2.3: Determinación de los factores de shear (a), y efecto de la transformación al sistema de coordenadas objeto desplazado (b).

El factor s_j es calculado de manera similar. Luego, la transformación M_{shear} puede ser escrita como:

$$M_{shear} = \begin{pmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

d. Proyección a la imagen intermedia: los cortes en el sistema de coordenadas objeto desplazado son compuestos para dar origen a la imagen intermedia. Sin embargo, el sistema de coordenadas en el cual yace esta imagen no es conveniente, debido a que su origen (i=0,j=0) no está ubicado en una esquina de la imagen intermedia (u=0.v=0). Por ello, se realiza previamente una traslación del sistema de coordenadas objeto desplazado, para que el origen de este sistema sea siempre aplicado a la esquina superior izquierda de la imagen intermedia. Los coeficientes t_i y t_j de la traslación pueden obtenerse según los siguientes casos:

- $s_i \geq 0$ y $s_j \geq 0 \Rightarrow t_i = 0$ y $t_j = 0$
- $s_i \geq 0$ y $s_j < 0 \Rightarrow t_i = 0$ y $t_j = -s_j*(k_{Max}-1)$
- $s_i < 0$ y $s_j \geq 0 \Rightarrow t_i = -s_i*(k_{Max}-1)$ y $t_j = 0$
- $s_i < 0$ y $s_j < 0 \Rightarrow t_i = -s_i*(k_{Max}-1)$ y $t_j = -s_j*(k_{Max}-1)$

Así, la transformación M_{shear} seguida a la traslación nos da origen a un sistema de coordenadas intermedio. La composición de ambas transformaciones la denominaremos M_{shearT}. Los cortes en este sistema de coordenadas son compuestos desde el corte más cercano al ojo, hasta el más lejano. El signo de $v_{so,k}$ nos indicará el orden en el cual los cortes son recorridos; i.e. si $v_{so,k}>0$ entonces el corte $k=0$ es el primero por estar al frente, en caso contrario sería $k=k_{max}-1$.

e. Warping: a continuación se deduce la descomposición de la matriz M_{view} como el producto de dos matrices: $M_{warp}*M_{viewT}$. Debido a que el warping es realizado sobre la imagen intermedia (2D), la tercera fila y columna de la matriz M_{warp} pueden obviarse en el momento de hacer efectivo el warping.

$$M_{shearT} = \begin{pmatrix} 1 & 0 & 0 & t_i \\ 0 & 1 & 0 & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & s_i & t_i \\ 0 & 1 & s_j & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \{3\}$$

$$M'_{view} = M_{warp} * M_{shearT} \Rightarrow M_{warp} = M'_{view} * M^{-1}_{shearT} \quad \{4\}$$

Sustituyendo la Ec. {3} en Ec. {4} nos queda:

$$M_{warp} = \begin{pmatrix} m'_{11} & m'_{12} & m'_{13} - s_i * m'_{11} - s_j * m'_{12} & m'_{14} - t_i * m'_{11} - t_j * m'_{12} \\ m'_{21} & m'_{22} & m'_{23} - s_i * m'_{21} - s_j * m'_{22} & m'_{24} - t_i * m'_{21} - t_j * m'_{22} \\ m'_{31} & m'_{32} & m'_{33} - s_i * m'_{31} - s_j * m'_{32} & m'_{34} - t_i * m'_{31} - t_j * m'_{32} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. CUANTIZACIÓN DE VECTORES

Para generar la imagen intermedia, es necesario hacer la composición de los voxels que se solapan sobre cada pixel de dicha imagen en la proyección. Por cada voxel se aplica un modelo de iluminación. En nuestro caso se aplica el modelo Phong[2]. Este tiene la siguiente fórmula:

$La*Ka + Ld*Kd*<N,v_0> + Ls*Ks*<N,Vs>^{sh}$, en donde:

- *La*: Coeficiente de luz ambiental en [0,1]
- *Ka*: Coeficiente ambiental del objeto en [0,1]
- *Ld*: Coeficiente difuso de la luz en [0,1]

- *Kd*:Coeficiente difuso del objeto en [0,1]
- *N*: Gradiente del voxel
- v_0: Dirección de visualización normalizado
- *Ls*: Coeficiente especular de la luz en [0,1]
- *Ks*: Coeficiente especular del objeto en [0,1]
- *Vs*: Vector de máxima especularidad, que en proyección paralela es $(Li+v_o)/\|Li+v_o\|$
- *sh*: Coeficiente de *Shininess* (brillo) en [0,∞)

Por simplicidad, se asumirá que se tiene una sola luz ubicada en el infinito en sistema de coordenadas desplazado, con dirección *Li*(0,0,1), osea, igual a v_o. Si se calcula el modelo Phong en el sistema de coordenadas desplazado, habría que calcular el gradiente del voxel cada vez que el objeto sea rotado. Esto acarrearía una costo excesivo para la generación de una imagen. Es por ello que se calcula el modelo en el sistema de coordenadas objeto. Tanto v_o como *Li* son obtenidos mediate la Ec.[1]. Para mejorar aún más el tiempo de respuesta, se precomputa el modelo de iluminación para un subconjunto de vectores gradientes, distribuidas pseudo uniformementes entre todas las direcciones posibles[1]. El método general es el siguiente:

- Escoger un subconjunto de vectores unitarios distribuidos en forma pseudo uniforme entre todas las direcciones posibles.
- Asignar un entero único *Vu* a cada vector *u* en el conjunto.
- Calcular el gradiente de cada voxel y encontrar el vector unitario *u* más cercano al mismo en el conjunto. Así, el gradiente del voxel estará representado por el entero *Vu*.

Debido a que esto es realizado una sóla vez, la eficiencia del algoritmo mejora notablemente. El modelo Phong sólo se computa para el subconjunto vectores unitarios (por ejemplo, 8192 vectores), cada vez que cambia el punto de vista, lo cual es bastante inferior en tiempo que realizarlo para los gradientes de todos los voxels (cientos de miles o millones).

Lacroute[1], realiza la cuantización de vectores mediante una discretización pseudo-uniforme de un octahedro, con el fin de obtener una fácil asociación entre un gradiente y su índice respectivo. En algunas zonas del octahedro Lacroute explica que hay más vectores que en otras, por lo que la aproximación de un vector gradiente *N* es mejor o peor en promedio según la zona en donde se encuentre el vector unitario más cercano a él.

4. ALGORITMO A ALTO NIVEL

Al momento de cargar los datos, se debe definir qué tejidos visualizar. Esto se logra especificando cuáles densidades de voxels aportarán información relevante en el proceso. Por ejemplo, se podría limitar la visualización a sólo huesos, si se observa por ejemplo que éstos comprenden la banda entre 100 y 255. El resto de los voxels son nulos, y no deben procesarse. A continuación se muestra a alto nivel el algoritmo de visualización volumétrica pseudo-formal utilizando la factorización Shear-Warp.

(a) Calcular el vector de visualización en coordenadas objeto utilizando la Ec.{1}.
(b) Obtener el eje principal mediante la Ec.{2}.

(c) Según el eje principal, obtener la permutación P, y aplicarla sobre M_{view}.
(d) Obtener los valores s_i, s_j, t_i, t_j necesarios para expresar las coordenadas de los cortes en el sistema de coordenadas desplazado.
(e) Calcular el modelo de iluminación para el conjunto reducido de vectores.
(f) Generar la imagen intermedia proyectando los cortes en el plano k=0 en el sistema de coordenadas desplazado. En este paso, se explota coherencia en los datos volumétricos, tratando sólo los voxels no nulos, y la coherencia en la imagen, tratando los pixels transparentes (i.e. su opacidad no es inferior a 0.05).
(g) Con los valores calculados en (d), obtener la matriz M_{warp}.
(h) Aplicar la transformación M_{warp} a la imagen intermedia para generar la imagen final.
(i) Desplegar la imagen.

En el caso de utilizar texture mapping, los pasos (h) e (i) corresponden a realizar la transferencia de la imagen intermedia a la memoria de textura, y realizar la texturización del polígono respectivamente.

5. DETALLES DE IMPLEMENTACIÓN

5.1 Cuantización de Vectores: la cuantización de vectores propuesta por Lacroute[1], parte de la división de un octahedro de una forma muy particular, poco uniforme, con la idea de establecer una rápida asociación entre un vector y su respectivo índice. En nuestro caso, como este proceso se realiza una sola vez (al cargar los datos), se plantea mejorar la uniformidad de la cuantización mediante la subdivisión de un icosahedro (ver Fig. 5.1a). Cada triángulo del icosahedro es dividido en cuatro triángulos equiláteros (ver Fig. 5.1b). Cada vértice nuevo es normalizado, y el proceso vuelve a repetirse sobre todos los triángulos. En la Fig. 5.1c se muestran distintos niveles de subdivisión.

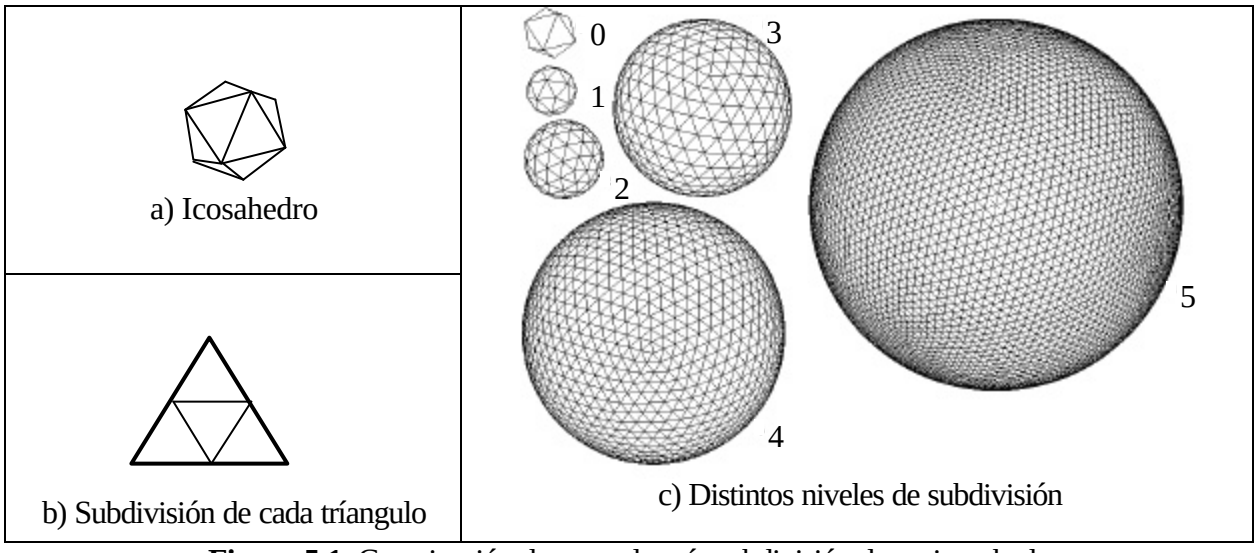


Figura 5.1: Cuantización de normales vía subdivisión de un icosahedro

Los vectores que parten del origen y pasan por el centro de un triángulo constituyen los vectores unitarios de la cuantización. El número de triángulos se cuadruplica por cada nivel de subdivisión. Así

para el nivel 5 se tienen 8192 vectores. La uniformidad de la cuantización mediante este método, mejora la calidad de la imagen, debido a que el modelo de iluminación se calcula en forma más precisa.

En la Fig. 5.2 se muestra una imagen generada con distintos niveles de subdivisión del icosahedro en la cuantización de vectores. Del nivel de subdivisión cinco (5) en adelante, no hay diferencia visual en la calidad de la imagen, por lo cual el nivel fue fijado en cinco en las pruebas realizadas.

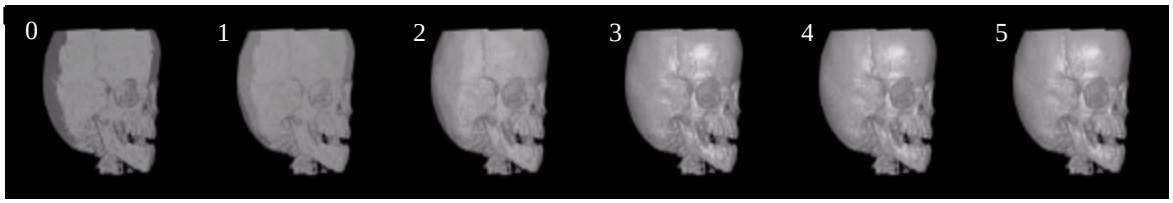


Figura 5.2: Imágenes obtenidas usando distintos niveles de subdivisión (0,1,...5) del icosahedro, en la cuantización de vectores.

5.2 Almacenamiento: los datos volumétricos fueron almacenados usando compresión RLE[1]. Las bondades de almacenar los datos de esta manera, es que se explota la localidad espacial, ya que los voxels no nulos se almacenan en el mismo orden en que son accedidos (fila tras fila por cada corte). Además, los voxels nulos no son almacenados, por lo que son fáciles de obviar.

Para la generación de la imagen intermedia mediante el proceso shear y proyección, es importante considerar el equivalente (en un algoritmo de Ray Casting) a la terminación temprana de rayo, i. e. no considerar el resto del rayo que atraviesa la data volumétrica cuando la opacidad acumulada por el mismo excede un umbral de opacidad[1] (0.05 en la práctica). Así, se reduce el cálculo no sólo cuando el voxel es no nulo, sino además cuando los pixels que el voxel solapa en la proyección no sean opacos. Es necesario entonces definir una estructura de datos que permita recorrer y actualizar los pixels transparentes de la imagen intermedia en una forma eficiente. Lacroute[1], propuso una en donde por cada pixel opaco se tiene un índice que puede apuntar al próximo pixel opaco del mismo run, o al primer pixel transparente del próximo run. La estructura es una matriz, de las mismas dimensiones de la imagen intermedia, donde cada celda contiene la opacidad del pixel y un índice. El índice es cero si el pixel es transparente, y mayor que cero si es opaco. Con esta estructura se puede encontrar el próximo pixel transparente a partir de un pixel opaco en pocas iteraciones. Cuando un pixel cambia de transparente a opaco, se pone a apuntar al próximo inmediato, es decir, el índice se coloca en uno. Para evitar largas búsquedas, se utiliza compresión de camino[1]. Este consiste en que luego de realizar una búsqueda de un pixel transparente, se vuelven a recorrer los pixels visitados para ponerlos a apuntar al pixel transparente encontrado (Ver Fig. 5.3), reduciendo iteraciones en futuras búsquedas.

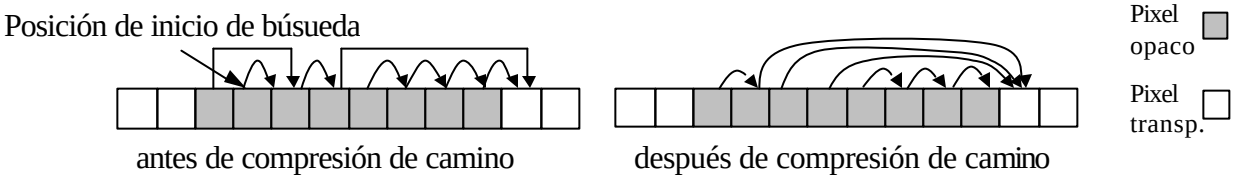


Figura 5.3: Compresión de camino para reducir futuras búsquedas de pixels transparentes

La estructura de datos presenta un inconveniente; cuando se termina de recorrer un run de pixels transparentes es necesario realizar una búsqueda a partir del primer pixel opaco del run siguiente, para encontrar el próximo. Por ello, planteo otra estructura que permita además enlazar los pixels transparentes. Esta consiste en tener por cada pixel transparente un apuntador (desplazamiento) al próximo y previo pixel transparente. Si el pixel es opaco, el apuntador previo se ignora, y el próximo apuntaría a un pixel opaco del mismo run, o al próximo pixel transparente. La matriz, que denominaremos A, es inicializada con pixels transparentes, opacidad 1.0, donde cada elemento $a_{i,j}$ apunta al próximo transparente $a_{i,j+1}$ y al previo $a_{i,j-1}$. Cuando un pixel (i,j) se hace opaco, se realiza el siguiente algoritmo, para mantener enlazados los transparentes:

```
Si (A[i,j].opacidad < 0.05) entonces
      prev ← A[i, j].prev;
      prox ← A[i, j].prox;
      A[i, j-prev].prox ← A[i, j-prev].prox + A[i,j].prox;
      A[i, j+prox].prev ← A[i, j+prox].prev + A[i,j].prev;
FinSi
```

La operación de buscar el primer pixel transparente sólo es realizada una vez por fila. En vez de comenzar a realizar la búsqueda desde el primer pixel de la fila, se comienza desde la posición p correspondiente a la proyección del primer voxel no nulo que solapa con la fila. De esta manera se evita recorrer los primeros pixels de cada fila, que suelen ser transparentes, y que son solapados por voxels no nulo. Para reducir largas búsquedas futuras a partir de p, se usa el algoritmo de compresión de camino sólo en este caso.

5.3 Warping: el warping consiste en obtener la imagen correcta C en el sistema de coordenadas imagen, a partir de la imagen intermedia I, obtenida de aplicar la composición de los cortes. El warping se implementó de dos maneras. La primera (SoftWarp) fue mediante la transformada inversa. Para ello se calcula M_{warp}^{-1} y luego para cada centro de pixel (x,y) de la imagen C, se resuelve:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = M_{warp}^{-1} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Debido a que (x',y') no necesariamente son coordenadas enteras (centros de pixels en I), se utiliza interpolación bilineal para obtener el color correspondiente en (x,y). Dado que en el algoritmo de warping, las variables x e y varían iterativamente de 1 en 1, se redujeron los cálculos del producto matriz-vector en cada iteración en sólo sumas, haciendo que el algoritmo sea incremental.

En la segunda implementación del warping (HardWarp) se utilizó el hardware gráfico, mediante texture mapping. Para ello, se toma como textura la imagen intermedia, y como vértices del polígono las esquinas de ésta, transformadas por M_{warp}.

6. RESULTADOS

El programa se implementó usando VisualC++ 5.0. Se probó en una estación de trabajo Intergraph, con sistema operativo NT, procesador Pentium III 600 MHz, 256 MB de RAM y tarjeta gráfica *Wildcat*

4105. Se hicieron mediciones de tiempo con tres datos de prueba, promediando el tiempo de generación de 6000 imágenes. Las 6000 imágenes generadas fueron obtenidas de variar uniformemente 10 ángulos de rotación para cada eje (10^3=1000 imágenes). Como se realizan tres ciclos anidados (uno por cada eje), se generaron las mismas imágenes seis veces, permutando el orden de los tres ciclos. En la tabla 6.1 se muestran los parámetros de los datos de prueba.

Data \ item	*xMax*	*yMax*	*zMax*	Voxels no nulos	% de voxels no nulos
BabyCTHead	256	256	94	246068	3,99
MRBrain	256	256	110	1416666	19,65
CTHead	256	256	113	474443	6,41

Tabla 6.1: arreglos volumétricos de entrada

El tiempo se dividió en cuatro partes: tiempo de actualización Tact (pasos a, .., e), tiempo de shear Tshear (paso f), tiempo de warping Twarp, y tiempo de despliegue Ttrans. En el caso de warping por software, Twarp consiste en los pasos g y h, y Ttrans en el paso i (ver sección 4). En el caso de warping por hardware, Ttrans es el tiempo del paso g más transferir la imagen intermedia a la memoria de textura, y Twarp el tiempo de texturización del polígono. En la cuantización de normales, el nivel de subdivisión del icosahedro se fijó en cinco (8192 vectores). Para esta prueba, la imagen final se limitó a 256x256 pixels. En la tabla 6.2 se muestran los resultados de tiempo (segundos) para los distintos datos de prueba, utilizando la estructura de datos propuesta en la sección 5.2. De los resultados se puede decir que con las condiciones dadas, el uso de Softwarp es recomendable. Por otro lado, es interesante ver que la transferencia de la imagen final al buffer de despliegue (Ttrans en Softwarp) es inferior a la transferencia de la imagen intermedia a la memoria de textura (Ttrans en Hardwarp). Esto se debe a que las dimensiones de la imagen intermedia (textura) es de 512x512, i.e. el cuádruple en memoria de la imagen final.

Data\Tiempo	Tact	Tshear	Softwarp			Hardwarp		
			Twarp	Ttrans	Total	Twarp	Ttrans	Total
BabyCTHead	0,0055	0,0499	0.0243	0,0162	0,0960	0,0001	0,0564	0,1119
MRBrain		0,0777	0,0246		0,1240			0,1397
CTHead		0,0737	0,0251		0,1206			0,1357

Tabla 6.2: Tiempos (seg.) involucrados en la generación y despliegue de imágenes 256x256. El tiempo total es igual a Tact+Tshear+Twarp+Ttrans.

Por otro lado, podemos notar que el tiempo de respuesta aumenta suavemente al aumentar el número de voxels no nulos. De hecho, MRBrain tiene 475% más voxels no nulos que BabyCTHead; sin embargo, el tiempo Tshear (que es el que depende del número de voxels no nulos) aumentó apenas en un 55%. Esto se debe a la aplicación de la técnica de terminación temprana de rayo, que evita procesar gran cantidad de voxels, que no tienen influencia significante en el color final de los pixels de la imagen.

Al escalar la imagen de salida a 512x512, Hardwarp tiene un mejor rendimiento que Softwarp. Esto se debe a que usando Hardwarp, Ttrans se mantiene inalterable, porque el tamaño de la imagen intermedia se mantiene. Similarmente, Twarp es prácticamente el mismo debido a que no hay diferencia notable en la texturización de un polígono pequeño respecto a uno más grande. En cambio, usando Softwarp, el

tiempo Ttrans y Twarp son afectados drásticamente al aumentar el tamaño de la imagen de salida. En la tabla 6.3 se muestra un ejemplo, en donde el tiempo Hardwarp es casi un tercio de Softwarp.

	Softwarp			Hardwarp		
	Ttrans	Twarp	Ttrans+Twarp	Ttrans	Twarp	Ttrans+Twarp
BabyCTHead	0,0657	0,0962	0,1619	0,0550	0,0001	0,0551

Tabla 6.3: Tiempo de Softwarp y Hardwarp en seg. con la imagen final escalada a 512x512

Se compararon los tiempos de Tshear (mostrados en la Tabla 6.2) usando la estructura de datos planteada, respecto al uso de sólo compresión de camino[1]. Los tiempos son mostrados en la tabla 6.4. Se puede notar que hay una mejora cuantificable en la estructura de datos planteada en este trabajo. Se estima que para un volumen de datos con dimensiones superiores, la diferencia sería más notoria.

	$Tshear_1$	$Tshear_2$	Mejora en tiempo de Tshear1 respecto a Tshear2
BabyCTHead	0,0499	0,0521	0,0022 seg. (4,4%)
MRBrain	0,0777	0,0816	0,0039 seg. (5.0%)
CTHead	0,0737	0,0755	0,0018 seg. (2,4%)

Tabla 6.4: Comparación de el tiempo Tshear en segundos usando la estructura de datos propuesta en este trabajo ($Tshear_1$) versus la propuesta por Lacroute ($Tshear_2$).

El programa fue probado además en un PC con procesador Pentium III de 600 Mhz, 64MB de RAM, sistema operativo Windows NT y tarjeta gráfica *Diamond Fire GL 1000 Pro*. Los tiempos fueron prácticamente los mismos, salvo el tiempo Ttrans, que en esta arquitectura fue notablemente mayor, i.e. 0,0831 segundos para Softwarp, y 0,2393 para Hardwarp con imagen de salida de 256x256 pixels.

También fue probado en un PC de muy bajo costo, con procesador Pentium I de 166MHz, no MMX, 64MB de RAM, sistema operativo Windows 95 OSR2, y tarjeta de video *Diamond Stealth 2500*. En este caso, el tiempo de generación de imágenes 256x256 osciló entre 0.6354 (MRBrain) y 0.3888 (BabyCTHead) segundos. En este último caso, el tiempo equivale a casi tres imágenes por segundo.

7. CONCLUSIONES

El número de voxels no nulos del arreglo volumétrico no influye proporcionalmente en el tiempo de generación de la imagen intermedia (Tshear), debido al uso de la técnica de terminación temprana de rayo.

Cuando el tamaño de la imagen intermedia es inferior o incluso igual al tamaño de la imagen de salida, se recomienda el uso de texture mapping en hardware para realizar el warping, por generar un mejor tiempo de respuesta. Esto es debido a varias razones. Por un lado, si la imagen intermedia es más pequeña, se transfiere menos información a la tarjeta gráfica. Por otro, el tiempo de texturización de polígonos vía hardware es despreciable independientemente del tamaño del polígono, mientras que el tiempo de warping vía software aumenta proporcionalmente con el tamaño de la imagen de salida.

La cuantización de vectores unitarios vía subdivisión de un icosahedro, mejora la calidad de la imagen generada, respecto a la propuesta por Lacroute[1]. Esto se debe a que al hacer la cuantización más uniforme, el error cometido (en promedio) al aproximar una normal en un voxel es inferior, redundando en mejora de la precisión de la aplicación del modelo de iluminación.

En la Fig. 5.3 se muestra una imagen obtenida con cada arreglo volumétrico de prueba. La imagen de salida en estas pruebas fue de 256x256 pixels.

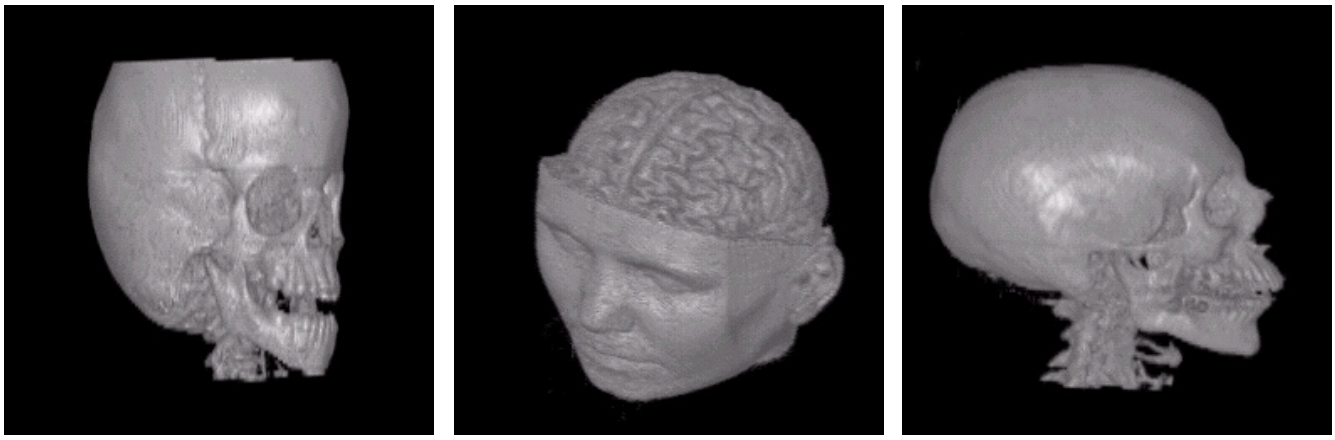


Figura 5.3: Imágenes generadas por el programa. La imagen izquierda corresponde a BabyCTHead, la del centro a MRBrain, y derecha a CTHead.

8. TRABAJOS A FUTURO

A futuro se implementará la versión estereoscópica del visualizador volumétrico. Será enriquecido además con una interfaz gráfica, que permita interactivamente cambiar el color de los distintos tejidos o densidades del volumen, cambiar la opacidad de los tejidos, y cortar el volumen con un plano arbitrario o una esfera para visualizar interiores del objeto. Se implementará el proceso de shear y composición de cortes usando texture mapping 3D, texturizando polígonos perpendiculares a la dirección de visualización que cortan la data volumétrica, y haciendo la composición de dichos polígonos mediante la técnica de alfa blending[2], en OpenGL 1.2. El programa será probado en Linux y en estaciones Silicon Graphics para comparar rendimiento. Además, será adaptado para funcionar con volúmenes de entrada con dimensiones superiores a 256^3 voxels.

9. BIBLIOGRAFÍA

[1] Philippe G. Lacroute. "*Fast Volume Rendering Using Shear-Warp Factorization of the Viewing Transformation*". Reporte técnico: CSL-TR-95-678. Septiembre 1995.

[2] James Foley, Andries van Dam, Steven Feiner, John Hughes. "*Computer Graphics: Principles and Practice*". Addison-Wesley. Segunda edición. 1990.