Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Lecturas en Ciencias de la Computación
ISSN 1316-6239

Introducción al Rendering Directo de Volúmenes

Rhadamés Carmona

RT 2015-01

Centro de Computación Gráfica
Caracas, febrero 2015

Introducción al Rendering Directo de Volúmenes

Prof. Rhadamés Carmona
Centro de Computación Gráfica, Universidad Central de Venezuela
Escuela de Computación, Caracas, Venezuela, 1041-A
rhadames.carmona@ciens.ucv.ve

Resumen

Este trabajo presenta en detalle la teoría necesaria para comprender e implementar la técnica de *rendering* directo de volúmenes (*direct volume rendering*). Esto incluye la derivación paso a paso de la ecuación de composición volumétrica, y su implementación en los algoritmos de *ray casting*, planos alineados, conchas esféricas y *shear-warp*. Aborda también la clasificación pre-integrada, la cual mejora la calidad del *rendering* con poco impacto en el tiempo de respuesta, así como la inclusión de una luz externa para la iluminación.

Palabras claves: *ray casting*, *rendering* directo de volúmenes, *rendering*, planos alineados al *viewport*, planos alineados al objeto, conchas esféricas, pre-integración.

1 INTRODUCCIÓN

El *rendering* directo de volúmenes o *direct volume rendering* es una simulación aproximada de la propagación de la luz a través de un medio participante representado por el volumen [LAC95]. Con esta técnica, se produce una imagen mediante el cómputo de la cantidad de luz alcanzada en cada píxel. Físicamente, cuando la luz fluye a través del volumen, ésta puede ser absorbida, esparcida (*scattering*) y emitida; adicionalmente, se pueden producir otros tipos de interacción tales como fosforescencia (absorción y reemisión de energía luego de un pequeño retardo), y fluorescencia (absorción y reemisión de la energía en una frecuencia distinta). Sin embargo, el *rendering* de volúmenes no tiene como objetivo simular todos estos fenómenos. Sólo interesa el resultado óptico del proceso de propagación y avance de la luz a través del volumen, quedando así un modelo óptico basado únicamente en emisión y absorción [SAB88], [WIL92], [LAC95].

Por lo general, el volumen viene dado como un conjunto de muestras de una función escalar continua, representado por una malla regular, y almacenado en un arreglo tridimensional de escalares [MOR04]. Existen diversas técnicas para visualizar la información presente en un volumen. En una etapa previa, se pueden extraer los contornos de interés de cada corte del volumen [MAX95], [CAR01], o reconstruir una aproximación poligonal de las superficies de interés [CLI87], [CAR99]. Sin embargo, mediante el *rendering* directo del volumen, podemos evitar etapas previas de reconstrucción, y observar información adicional que no está presente en contornos y superficies.

En el *rendering* directo de volúmenes se genera una imagen tridimensional a partir de la proyección de las muestras discretas de un volumen, por lo que no necesita de la reconstrucción de una iso-superficie intermedia como en el caso de *Marching Cubes* [CLI87]. Por comodidad, en este trabajo nos referimos al *rendering* de volúmenes como *rendering* directo de volúmenes indistintamente.

Desde finales del siglo pasado, este *rendering* puede ser acelerado mediante la utilización del *hardware* gráfico [AKE93]. Este *hardware* está optimizado para el *rendering* de triángulos, los cuales son transformados, discretizados y desplegados con el *pipeline* gráfico [WOO99]. Comúnmente el *rendering* acelerado por *hardware* gráfico es realizado mezclando polígonos texturizados con cortes del volumen. Sin embargo, en la actualidad, el *hardware* gráfico cuenta adicionalmente con unidades programables, tanto para vértices como para fragmentos, lo que permite realizar algoritmos que operen directamente con el volumen para obtener el color de cada píxel de la imagen [OPE07], [NVI07].

Este trabajo estudia conceptos relacionados con el *rendering* de volúmenes, y diversas implementaciones, haciendo énfasis en aquellas que utilizan la aceleración por *hardware* gráfico.

2 LA ECUACIÓN

En esta sección del trabajo se describe el modelo óptico utilizado, del cual se deriva la ecuación de *rendering* de volúmenes. Debido a que esta ecuación es básicamente una integral que no tiene anti-derivada para funciones lineales a trozos o de orden superior, se detalla cómo la misma es aproximada para su posterior evaluación.

2.1 Modelo Óptico

Para visualizar el volumen se requiere de un modelo óptico para representar la interacción de la luz con el volumen, y determinar la cantidad de luz alcanzada en cada píxel de la imagen. La modelación de esta interacción es relativamente compleja, y requiere el uso de la Teoría del Transporte Radioactiva[1] [CHA60], [KRU90]. Sin embargo, en el contexto de visualización científica se puede realizar una simulación menos compleja, al no considerar algunos fenómenos que ocurren en la interacción.

Para derivar el modelo óptico más utilizado en el área de *rendering* de volúmenes, es necesario primero comprender y relacionar algunos conceptos asociados con el fenómeno del transporte de luz: *scattering*, absorción, emisión, reflexión, refracción y extinción.

Gran parte de la luz que llega a nuestros ojos es luz indirecta, proveniente del reflejo de la luz en objetos. Si observamos la hoja de un árbol, vemos que esta es verde. Esto significa que la hoja absorbe más la luz que viaja en el espectro de la frecuencia del azul y el rojo, y menos la de verde. La energía absorbida es transformada en otra energía (por ejemplo en calor); el resto de la energía es esparcida (*scattered*), y parte de esta energía llega hasta nuestros ojos. El *scattering* consiste es el

[1] Transferencia Radiactiva: el término de transferencia radiactiva se refiere al fenómeno físico de la transferencia de energía en la forma de radiación electromagnética. La propagación de la radiación a través de un medio es afectado por los procesos de absorción, emisión y *scattering*. La ecuación de transferencia radiactiva describe estas interacciones matemáticamente. Las ecuaciones de transferencia radiactiva tiene aplicaciones en una amplia variedad situaciones; entre ellos, la detección de satélites en movimiento, astronomía, óptica, etc. Usualmente la solución analítica de la ecuación de transferencia radiactiva no existe en un medio 3D no homogéneo, y debe ser resuelta por métodos numéricos.

desvío de la luz causado por partículas encontradas en su trayectoria. La colisión de la luz con estas partículas puede ser estudiada a distintos niveles, según el tamaño de las partículas.

Consideremos partículas más grandes que la longitud de onda de la luz. Así, se puede pensar en gotas de agua, polvo, burbujas, células, etc. En estos casos, el estudio del *scattering* de la luz puede ser comprendido a través de los conceptos de óptica geométrica [HAN74], y puede utilizarse *Ray Tracing* o trazado de rayos para obtener resultados numéricos aceptables. El *scattering* puede ser estudiado mediante los fenómenos de reflexión y refracción. Cuando la luz colisiona con una partícula, parte de la luz se refleja, otra se refracta, y otra se absorbe. La reflexión cambia la dirección de la luz, pero la devuelve al medio de donde provenía; la refracción también cambia la dirección de la luz, pero en este caso la luz viaja dentro de la partícula. El ángulo de refracción depende de las densidades de ambos medios, y puede calcularse por la Ley de Snell [FOL90]. El rayo refractado viaja dentro de la partícula, hasta volver a colisionar con la superficie de la partícula, en donde en rayo puede volver al medio original por refracción, o puede reflejarse, en cuyo caso sigue su trayectoria de vuelta al interior de partícula (reflexión interna). Este proceso continúa, y si pensamos en la partícula como una caja negra colisionada por una rayo de luz, notamos que la luz se esparce (*scattering*) al incidir en la partícula (ver rayos a,b,c,d en la Fig. 2.1). Como se ha podido estudiar, una partícula puede absorber, y esparcir luz, pero adicionalmente, la luz puede ser emanada desde ella, como si fuese a su vez una fuente de luz. Tal es el caso de las partículas incandescentes.

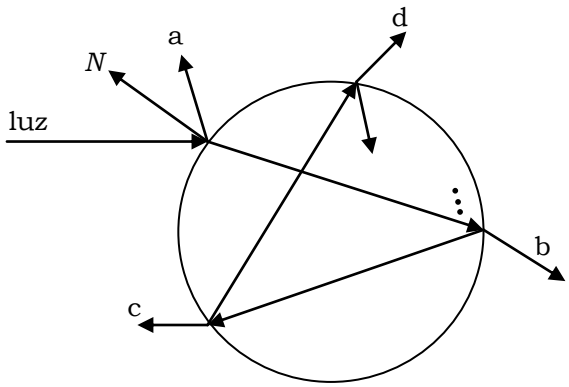


Figura 2.1: *scattering*, considerando que las partículas son más grandes que la longitud de onda de la luz. Los rayos resultantes de la interacción de la luz con la partícula mostrada en la figura son producto de: (a) reflexión externa, (b) dos refracciones, (c) una reflexión interna, (d) dos refracciones internas.

Tanto el *scattering* como la absorción reducen la energía del rayo de luz, que atraviesa un medio o volumen colmado de partículas. Esta atenuación de la luz se denomina extinción [HAN74]. Como se estudiará posteriormente, la extinción quedará sólo en función de la absorción, al ignorar el fenómeno de *scattering* del modelo.

Para modelar la interacción de la luz con el volumen, también se pueden considerar partículas infinitesimalmente pequeñas (i.e. inferiores a la longitud de onda de la luz), estudiando la interacción de la luz con la nube a nivel atómico o molecular. En este caso, el *scattering* puede verse

como el esparcimiento de la luz en todas las direcciones, cuando esta colisiona con la partícula. Uno de los primeros modelos ópticos en esta área describe un método para sintetizar imágenes de los anillos del planeta Saturno, los cuales consisten de nubes de partículas de hielo reflectivo [BLI82]. Este modelo considera el *scattering* simple, *shadowing*[2] y propagación de la luz a través de la nube. El *scattering* múltiple es posteriormente considerado por, Kajira y Von Herzen [KAJ84]. Estas interacciones de la luz con volumen pueden resumirse en la Fig. 2.2.

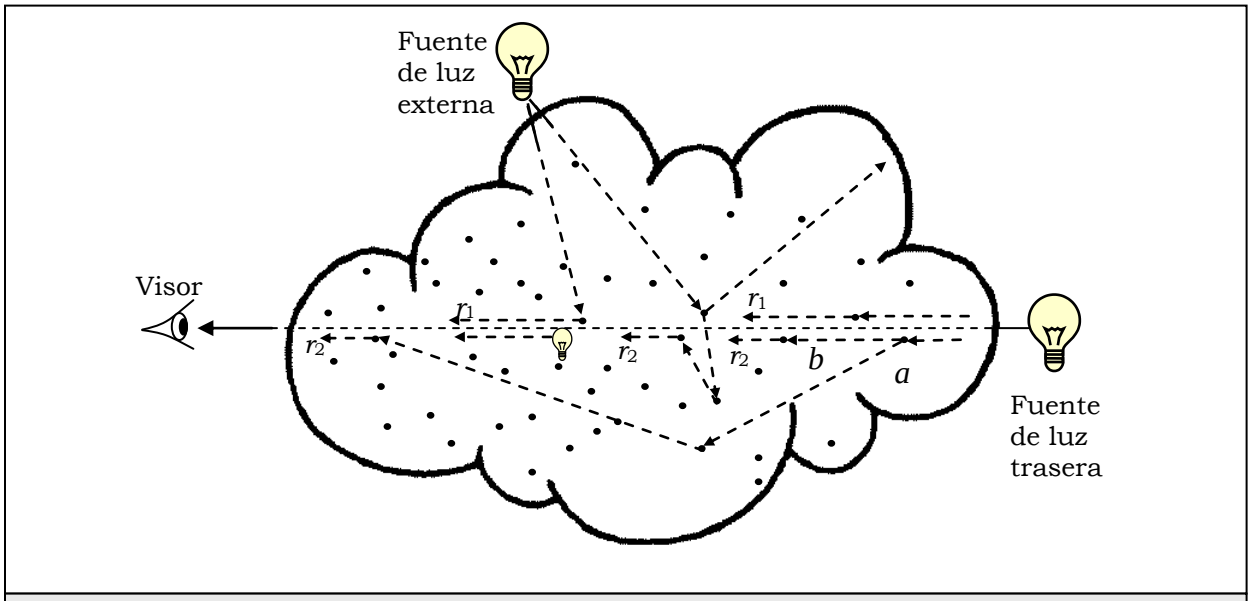


Figura 2.2: interacciones de la luz con el volumen. En el modelo se asume que hay una fuente de luz trasera que emite energía en dirección al visor. Un rayo de luz que viaja hacia el visor puede ser atenuado (por absorción y *scattering*) o incluso obstruido totalmente por partículas encontradas en la travesía del rayo, según el modelo óptico utilizado. Dentro de la nube pueden existir fuentes de luz, que suelen ser las mismas partículas emanando energía en todas las direcciones, y particularmente en la dirección al visor. También pueden existir una o más fuentes de luz externas. Mediante *scattering*, la luz puede llegar al visor de manera indirecta, mediante la interacción de un rayo de luz con una única partícula (*scattering* simple, ver rayos r_1), o con más de una partícula (*scattering* múltiple, ver rayos r_2), ya sea considerando la fuente de luz trasera o la externa. Adicionalmente, una partícula *a* puede producir *shadowing* a otra partícula *b*, al interponerse en la trayectoria de la luz.

Otros trabajos basados en estas investigaciones fueron publicados en la década de los 80's [MAX86], [DRE88], [SAB88]. En particular, el trabajo presentado por Paolo Sabella [SAB88] describe un modelo de partículas llamado "*the density emitter model*". Se basa en el trabajo de Blinn, pero asume que las partículas emiten su propia luz, en vez de reflejarlas de la fuente de luz

[2] **Shadowing**: opacamiento o sombra. En el contexto de modelos ópticos, significa la atenuación de la luz por parte de una partícula previa a la partícula en estudio. En el campo de la visualización científica, el *shadowing* por lo general es ignorado, al considerar al volumen como una nube de partículas dispersas infinitesimalmente pequeñas, en donde la probabilidad de que un mismo rayo de luz alcance a dos partículas en su trayectoria es prácticamente cero.

vía *scattering*, y no considera el *shadowing*. El resultado de este modelo simplificado ha sido ampliamente aceptado y refinado en trabajos siguientes, pues permite visualizar volúmenes en tiempo real. Cabe señalar que los efectos de *shadowing* y *scattering* no necesariamente revelan información importante de un volumen en el área de visualización científica.

Posteriormente, Peter Williams y Nelson Max [WIL92] refinaron estos trabajos, y especificaron dos modelos con resultados idénticos. El primero de ellos es llamado Modelo de Partículas, en el cual el volumen es modelado como una nube de partículas que absorben y emiten luz, y está basado en el trabajo de Sabella [SAB88]. El segundo modelo lo presentan los autores por primera vez, y se denomina el Modelo Continuo, en donde el volumen es concebido como un medio continuo incandescente. La derivación de este modelo se basa en una ecuación diferencial, el cual será mostrado posteriormente. Ambos modelos resultan en la misma fórmula matemática para obtener la intensidad final en un píxel de la imagen.

Muchos trabajos subsiguientes se han basado en el modelo continuo, y en particular el trabajo de Nelson Max [MAX95], el cual consiste en una revisión de diferentes modelos de interacción de la luz con el volumen. Los diferentes modelos son presentados por Nelson Max por orden de realismo: sólo absorción, sólo emisión, emisión y absorción; luego a este último se le incluye *scattering* simple de iluminación externa sin *shadowing*, *scattering* simple de iluminación externa con *shadowing*, y finalmente *scattering* múltiple.

Basados en estos trabajos, y principalmente los trabajos de Williams y Nelson Max, a continuación se presenta la derivación de la ecuación de visualización volumétrica, considerando únicamente la emisión y absorción.

El volumen o *medio participante*[3] suele ser pensado como un material semitransparente, compuesto de partículas infinitesimalmente pequeñas que absorben y emiten la luz que viaja en dirección al visor. Para determinar la cantidad de luz alcanzada se puede considerar un simple cilindro centrado en el rayo de visualización que pasa a través de volumen, pues las partículas externas al cilindro no influyen en el resultado final debido a la eliminación del *scattering* del modelo. Se asume que el cilindro es lo suficientemente delgado para que las propiedades del volumen no cambien a lo ancho, pero sí pueden cambiar a lo largo.

La intensidad de luz I_0 incide en la parte trasera del cilindro, en dirección al visor. La luz I_D que sale del otro extremo del cilindro y llega al ojo, determina la intensidad de un píxel (ver Fig. 2.3). Consideremos una rebanada o *slab* del cilindro, al igual que el cilindro con base de área E (área del disco), y longitud Δ_s. Como se muestra a la derecha de la Fig. 2.3, algunos rayos son absorbidos por partículas, otros logran pasar al otro lado del *slab*, y otros son emitidos por partículas dentro del *slab*. La cantidad de partículas por *slab* viene dada por $N=\rho E\Delta_s$, donde ρ es la densidad de partículas por unidad de volumen, y $E\Delta_s$ el volumen del *slab*. El valor de ρ puede variar en el volumen, pues hay zonas con mayor densidad que otras, pero se asume constante dentro de un *slab*.

[3] Medio participante: es un término de la óptica que significa un material cuya propiedad afecta el transporte de la luz a través de su volumen. Un medio participante influye sobre la travesía de la luz por su coeficiente de extinción.

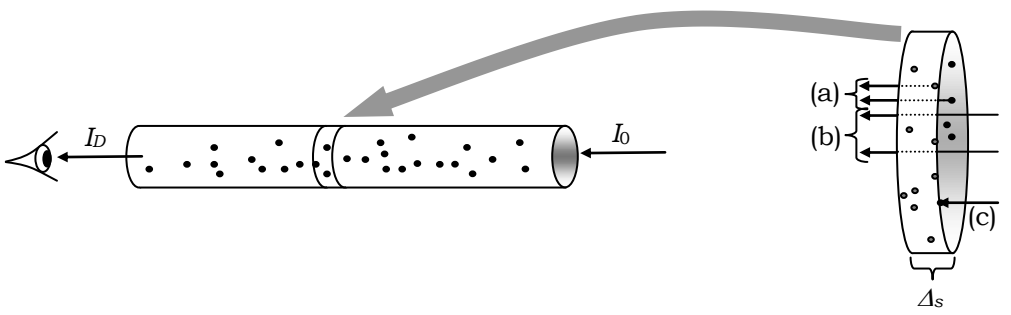


Figura 2.3: interacción de la luz con el volumen, considerando sólo absorción y emisión. Los rayos etiquetados con: (a) representan partículas emitiendo luz, (b) rayos de luz que atraviesan el *slab* (rebanada), (c) rayo de luz absorbido por una partícula. La intensidad inicial de la luz al entra al volumen es I_0, y la intensidad resultante al atravesar todo el volumen con una distancia D, es I_D.

Supongamos que todas las partículas son esferas de radio r. Entonces, un corte seccional de la partícula, y equivalentemente el área proyectada de la partícula es $A=\pi*r^2$. Conforme Δ_s se acerca a cero, la superposición de las partículas tiende a cero. Bajo esta premisa, el área proyectada del *slab* oscurecida por las partículas es el número de partículas multiplicada por el área de una partícula (i.e. $AN=A\rho E\Delta_s$). Si dividimos esta área por el área proyectada total de toda la rebanada (área del disco), obtenemos la fracción de luz obstruida cuando esta pasa a través de la rebanada de longitud Δ_s, equivalente a $AN/E=A\rho\Delta_s$. Así se puede definir el coeficiente de atenuación por unidad de longitud como $\tau=A\rho$. A este coeficiente también se le conoce como la absorción, o coeficiente de extinción[4].

Sea L la emisión de luz de las partículas por unidad de área proyectada. Debido que el área proyectada de una partícula es A, la emisión de una partícula es LA. Conociendo que hay N partículas en un *slab*, su emisión es $LAN = LA\rho E\Delta_s$. Al dividir este resultado por E, obtenemos la luz emitida por unidad de área, la cual puede ser expresada en función de τ como $L\,\tau\Delta_s$.

Con estos parámetros podemos expresar el cambio de intensidad de un rayo de luz $I(\lambda)$ en el disco s como la cantidad de luz emitida en dicho disco menos la cantidad de luz entrante que es atenuada. Esto puede expresarse fácilmente mediante la siguiente ecuación diferencial:

$$\frac{dI(\lambda)}{d\lambda} = L(\lambda)\tau(\lambda) - I(\lambda)\tau(\lambda). \qquad \text{[Ec. 2.1]}$$

Esta ecuación diferencial puede ser instanciada una vez por cada una de las tres componentes de la luz: rojo, verde y azul. Así la ecuación se transforma en tres ecuaciones diferenciales lineales de primer orden, que pueden ser resueltas numéricamente por métodos como Runge-Kutta de cuarto orden [WIL92]. Alternativamente, para obtener la intensidad de luz $I(D)$ resultante de la interacción

[4] Coeficiente de extinción: es un término utilizado en óptica que significa la fracción de luz que se pierde debido a la absorción y dispersión por unidad de distancia en un medio participante. En el caso de visualización de volúmenes, el medio participante es el volumen en sí, considerándolo como un material semitransparente.

del rayo con el volumen, podemos manipular la ecuación, empezando por multiplicar por conveniencia ambos lados de la igualdad por el factor $\exp(\int_0^{\lambda} \tau(\lambda')d\lambda')$ y obtenemos:

$$\frac{dI(\lambda)}{d\lambda} e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} = L(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} - I(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}$$

$$\Rightarrow \quad \frac{dI(\lambda)}{d\lambda} e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} + I(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} = L(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}. \qquad \text{[Ec. 2.2]}$$

Sabiendo que $\frac{d}{d\lambda}\left(\int_0^{\lambda} \tau(\lambda')d\lambda'\right) = \tau(\lambda)$, podemos rescribir la ecuación como:

$$\frac{dI(\lambda)}{d\lambda} e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} + I(\lambda)\frac{d}{d\lambda}\left(e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}\right) = L(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}$$

$$\Rightarrow \quad \frac{d}{d\lambda}\left(I(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}\right) = L(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}. \qquad \text{[Ec. 2.3]}$$

Luego integramos respecto de λ en [0,D] para obtener

$$\left[I(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'}\right]_0^D = \int_0^D L(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} d\lambda$$

$$\Rightarrow \quad I(D)e^{\int_0^{D} \tau(\lambda')d\lambda'} - I(0) = \int_0^D L(\lambda)\tau(\lambda)e^{\int_0^{\lambda} \tau(\lambda')d\lambda'} d\lambda. \qquad \text{[Ec. 2.4]}$$

Finalmente, renombrando $I(0)=I_0$ y despejando $I(D)$ obtenemos

$$I(D) = I_0 e^{-\int_0^{D} \tau(\lambda')d\lambda'} + \int_0^D L(\lambda)\tau(\lambda)e^{-\int_{\lambda}^{D} \tau(\lambda')d\lambda'} d\lambda. \qquad \text{[Ec. 2.5]}$$

La Ec. 2.5 modela la interacción de la luz con el volumen, considerando sólo absorción y emisión. El primer término de la ecuación, calcula la cantidad de luz I_0 entrante al volumen, atenuada exponencialmente según la absorción de las partículas en la travesía del rayo. El segundo término agrega la cantidad de luz emitida de cada punto a lo largo del rayo, considerando la extinción desde el punto de emisión (a una distancia λ del origen del rayo) hasta el final de rayo. Debido a que los factores de extinción no tienen antiderivada, esta ecuación es evaluada numéricamente, en donde suele utilizarse la aproximación más simple basada en Series de Riemann.

El rayo de luz puede ser reparametrizado, desde el ojo (λ=0) hasta el punto de entrada de la luz (λ=D). Así, la ecuación puede ser escrita como:

$$I_0 = I_D e^{-\int_0^D \tau(\lambda')d\lambda'} + \int_0^D L(\lambda)\tau(\lambda)e^{-\int_0^\lambda \tau(\lambda')d\lambda'} d\lambda \,. \qquad \text{[Ec. 2.6]}$$

Muchos trabajos siguientes presentan esta versión de la ecuación de *volume rendering* [MOR04]; adicionalmente, otros eliminan el primer factor [LAC95], [ENG01], [SCH03], cuya aproximación es válida considerando despreciable la intensidad de luz entrante a la nube, o bien porque es muy atenuada por la nube. Esta simplificación será igualmente adoptada en este trabajo.

$$I_0 = \int_0^D L(\lambda)\tau(\lambda)e^{-\int_0^\lambda \tau(\lambda')d\lambda'} d\lambda \,. \qquad \text{[Ec. 2.7]}$$

El término de atenuación exponencial (extinción) es también llamada la transparencia: cantidad de luz que no es bloqueada por partículas entre el ojo y λ [BLI82]. Finalmente, la Ec. 2.7 halla la contribución de luz emitida $L(\lambda)$ a cada distancia λ, atenuada por su absorción $\tau(\lambda)$ y adicionalmente por la transparencia acumulada hasta λ, a lo largo del rayo que atraviesa la nube.

La Ec. 2.7 genera una imagen monocromática, en donde la intensidad $L(\lambda)$ representa sólo la luminosidad. Para extenderla a imágenes de color real, se puede expresar esta ecuación para las longitudes de onda del rojo, verde y azul, correspondientes al espacio de color RGB [MOR04]. Considerando que en el modelo de emisión-absorción, una partícula absorbe totalmente la luz que colisiona con la misma, la absorción $\tau(\lambda)$ será igual para todas las longitudes de onda. De esta forma, la ecuación puede re-escribirse como:

$$C = \int_0^D c(\lambda)\tau(\lambda)e^{-\int_0^\lambda \tau(\lambda')d\lambda'} d\lambda \,, \qquad \text{[Ec.2.8]}$$

en donde $c(\lambda)$ y C son tuplas RGB. La extinción $e^{-\int_0^\lambda \tau(\lambda')d\lambda'}$ se puede interpretar como la transparencia $T(\lambda)$ de la nube hasta la distancia λ. Basada en la transparencia, se puede calcular la opacidad α acumulada en la travesía del rayo a cualquier distancia λ, como:

$$\alpha(\lambda) = 1 - T(\lambda)$$
$$\Rightarrow \alpha(\lambda) = 1 - e^{-\int_0^\lambda t(s(x(\lambda')))d\lambda'} \,. \qquad \text{[Ec. 2.9]}$$

2.2 Evaluación de la ecuación

La asignación de un color y absorción (lo cual es llamado clasificación) en cada punto de intersección del rayo con el volumen se puede realizar de distintas formas. Una de ellas es mediante

la segmentación. En este caso, el volumen puede ser particionado en sus estructuras específicas que lo conforman; así, se le asigna distintas absorciones y colores a las distintas partes del volumen. Otra alternativa para la clasificación es la manipulación de funciones, que en el contexto de visualización de volúmenes se denominan funciones de transferencia, que pueden ser unidimensionales o multidimensionales [KNI02B]. Típicamente las funciones de transferencia unidimensionales, asignan las propiedades ópticas a cada vóxel basándose en el valor escalar s del vóxel; por ejemplo, las muestras con valores altos en una tomografía de un cuerpo humano, por lo general representan la estructura ósea, y se le asocia el color blanco. Adicionalmente, se puede incluir más información para asignarle las propiedades ópticas a un vóxel (funciones de transferencias multidimensionales), como el histograma de frecuencias, las derivadas de primer y segundo orden, la normal, la longitud del gradiente, la posición de la muestra, producto punto entre la dirección de la luz y la normal, etc. En este trabajo nos basaremos únicamente en el valor puntual del vóxel para asignarle sus propiedades ópticas.

Las funciones de transferencia unidimensionales suelen ser funciones lineales definidas a trozos (*piece-wise linear*), en donde cada muestra s es aplicada a un color $c(s)$=RGB y una absorción $\tau(s)$=A, para conformar una tupla RGBA [ENG01].

Existen dos tipos básicos de clasificación; estos son, pre-clasificación y post-clasificación. La diferencia fundamental entre ellas es el orden en que se llevan a cabo los procesos de clasificación e interpolación de muestras. La pre-clasificación ocurre cuando se aplica la función de transferencia a cada vóxel antes de que estos sean interpolados. Así, en una etapa de pre-procesamiento, se almacena el volumen como un arreglo tridimensional de valores RGBA [FAN96], y la interpolación es realizada sobre estos valores y no sobre muestras escalares. Contrariamente, post-clasificación ocurre cuando se aplica la función de transferencia al resultado de la interpolación de las muestras escalares del volumen. Los procesos de pre-clasificación y post-clasificación producen resultados diferentes si las funciones de transferencia no son constantes o la identidad [ENG01].

Desafortunadamente en la ecuación 2.8 no está reflejado el proceso de muestreo del volumen y clasificación, puesto que el color y absorción están en función del parámetro $\lambda \in [0,D]$ de la ecuación paramétrica de la recta que atraviesa el volumen. Centrando el análisis en post-clasificación, esta ecuación puede ser re-escrita como:

$$C = \int_0^D c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_0^{\lambda}\tau(s(x(\lambda')))d\lambda'}d\lambda\,, \qquad \text{[Ec. 2.10]}$$

donde, $x(\lambda)$ es la parametrización del rayo evaluada en λ, que da como resultado un punto (x,y,z). Note que en cada (x,y,z) se evalúa el campo escalar $s(x,y,z)$ del volumen para obtener el valor interpolado o muestra s, y a esta se aplica la función de transferencia para obtener su color y absorción.

Para calcular el color de salida C de un píxel particular, se integra la emisión de la luz representada por $c(s)$ auto-atenuada por su respectiva absorción $\tau(s)$, en el intervalo de intersección del rayo con el volumen $[0,D]$ (Ver Fig. 2.4). En cada punto $x(\lambda)$ del rayo, la emisión es adicionalmente atenuada por la transparencia en el segmento definido en $\lambda' \in [0,\lambda)$.

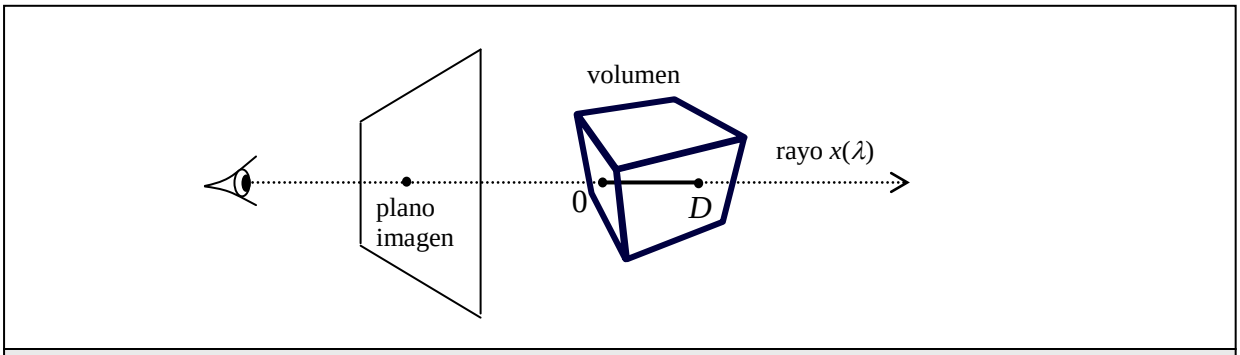


Figura 2.4: determinación de un píxel de la imagen. Se calcula la acumulación de color y opacidad a lo largo del rayo, desde que este entra al volumen, hasta que sale. El rayo es parametrizado mediante $x(\lambda)$. Así, barriendo λ desde 0 hasta D, se considera todo el segmento de intersección con el volumen.

Para facilitar la evaluación de la ecuación, el rayo es dividido en $n=\lceil D/h \rceil$ segmentos, con un paso fijo de longitud h. Dada que la división D/h no necesariamente es exacta, la ecuación puede escribirse de manera aproximada como:

$$C \approx \sum_{i=1}^{n} \int_{(i-1)h}^{ih} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{0}^{\lambda}\tau(s(x(\lambda')))d\lambda'} d\lambda . \qquad \text{[Ec. 2.11]}$$

La integral del término de transparencia puede separarse en dos: la transparencia acumulada hasta el segmento anterior, y la transparencia hasta el punto $x(\lambda)$ del segmento i-ésimo:

$$C \approx \sum_{i=1}^{n} \int_{(i-1)h}^{ih} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{0}^{(i-1)h}\tau(s(x(\lambda')))d\lambda' - \int_{(i-1)h}^{\lambda}\tau(s(x(\lambda')))d\lambda'} d\lambda . \qquad \text{[Ec. 2.12]}$$

Dado que $\exp\left(-\int_{0}^{(i-1)h}\tau(s(x(\lambda')))d\lambda'\right)$ no depende de λ, nos queda:

$$C \approx \sum_{i=1}^{n} e^{-\int_{0}^{(i-1)h}\tau(s(x(\lambda')))d\lambda'} \int_{(i-1)h}^{ih} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{(i-1)h}^{\lambda}\tau(s(x(\lambda')))d\lambda'} d\lambda . \qquad \text{[Ec. 2.13]}$$

La integral presente en el término de transparencia del intervalo [0,(i-1)h] puede a su vez ser expresado como una suma de integrales:

$$C \approx \sum_{i=1}^{n} e^{-\sum_{j=1}^{i-1} \int_{(j-1)h}^{jh} \tau(s(x(\lambda')))\,d\lambda'} \int_{(i-1)h}^{ih} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{(i-1)h}^{\lambda} \tau(s(x(\lambda')))\,d\lambda'}\,d\lambda$$

$$\Rightarrow \quad C \approx \sum_{i=1}^{n} \prod_{j=1}^{i-1} e^{-\int_{(j-1)h}^{jh} \tau(s(x(\lambda')))\,d\lambda'} \int_{(i-1)h}^{ih} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{(i-1)h}^{\lambda} \tau(s(x(\lambda')))\,d\lambda'}\,d\lambda$$

$$\Rightarrow \quad C \approx \sum_{i=1}^{n} \prod_{j=0}^{i-2} e^{-\int_{jh}^{(j+1)h} \tau(s(x(\lambda')))\,d\lambda'} \int_{(i-1)h}^{ih} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{(i-1)h}^{\lambda} \tau(s(x(\lambda')))\,d\lambda'}\,d\lambda$$

$$\Rightarrow \quad C \approx \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} e^{-\int_{jh}^{(j+1)h} \tau(s(x(\lambda')))\,d\lambda'} \int_{ih}^{(i+1)h} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{ih}^{\lambda} \tau(s(x(\lambda')))\,d\lambda'}\,d\lambda\,. \qquad \text{[Ec. 2.14]}$$

Note que las integrales resultantes en la Ec. 2.14 barren una pequeña área de longitud h o inferior, y generalmente son aproximadas al considerar que h es pequeño. Se utiliza una aproximación numérica denominada Aproximación de la Composición Volumétrica [LAC95] (ver Fig. 2.5).

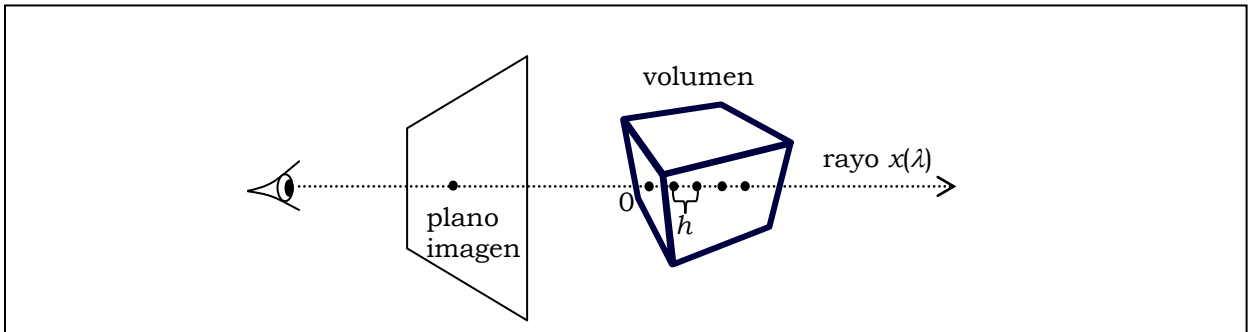


Figura 2.5: determinación de un píxel de la imagen a partir de las contribuciones de color y opacidad de las muestras equidistantes que se encuentran a lo largo del rayo que atraviesa el volumen. En cada segmento de longitud h, se asume constantes la emisión y la absorción. La distancia h entre muestras comúnmente es constante.

Para evaluar la ecuación numéricamente, típicamente la emisión y la absorción se asumen constantes dentro de cada intervalo. Así, podemos renombrar $s(x(\lambda))$ como s_i, en el segmento delimitado por $[ih, (i+1)h]$.

$$C \approx \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} e^{-\int_{jh}^{(j+1)h} \tau(s_j)\,d\lambda'} \int_{ih}^{(i+1)h} c(s_i)\tau(s_i)e^{-\int_{ih}^{\lambda} \tau(s_i)\,d\lambda'}\,d\lambda \qquad \text{[Ec. 2.15]}$$

En este contexto, $\tau(s_i)$ y $c(s_i)$ no dependen de la variable de integración en cada integral. Así, podemos escribir la ecuación como:

$$C \approx \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} e^{-h\tau(s_j)} c(s_i)\tau(s_i) \int_{ih}^{(i+1)h} e^{-\tau(s_i)\int_{ih}^{\lambda} d\lambda'} d\lambda \,, \qquad \text{[Ec. 2.16]}$$

en donde la integral remanente puede escribirse como:

$$\int_{ih}^{(i+1)h} e^{-\tau(s_i)\int_{ih}^{\lambda} d\lambda'} d\lambda = \int_{0}^{h} e^{-\tau(s_i)\int_{0}^{\lambda} d\lambda'} d\lambda = \int_{0}^{h} e^{-\lambda\tau(s_i)} d\lambda = \frac{1 - e^{-h\tau(s_i)}}{\tau(s_i)}. \qquad \text{[Ec. 2.17]}$$

Sustituyendo Ec. 2.17 en la Ec. 2.16 se obtiene

$$C \approx \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} e^{-h\tau(s_j)} c(s_i)(1 - e^{-h\tau(s_i)}). \qquad \text{[Ec. 2.18]}$$

Sea el color $c_i = c(s_i)$, y la opacidad $\alpha_i = 1 - e^{-h\tau(s_i)}$. La ecuación es escrita en su forma más simple como:

$$C \approx \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} (1 - \alpha_j)\alpha_i c_i$$

$$C \approx \alpha_0 c_0 + (1-\alpha_0)\alpha_1 c_1 + (1-\alpha_0)(1-\alpha_1)\alpha_2 c_2 + \ldots + (1-\alpha_0)\ldots(1-\alpha_{n-2})\alpha_{n-1} c_{n-1}. \qquad \text{[Ec. 2.19]}$$

Para evaluar la Ec. 2.19 se puede utilizar el operador digital *under* iterativamente, mezclando las muestras clasificadas en el orden *front to back*, es decir, desde la muestra más cercana hasta la más lejana al ojo [RUI06]. Este operador se define como:

$$c := (1-\alpha)\alpha_i c_i + c,$$
$$\alpha := (1-\alpha)\alpha_i + \alpha, \qquad \text{[Ec. 2.20]}$$

en donde c y α son inicializados en 0. Similarmente, si componemos las muestras en sentido *back to front* (desde la más lejana hasta la más cercana al ojo), se puede utilizar el operador digital *over* [LAC95] que se define como:

$$c := \alpha_i c_i + (1-\alpha_i)c,$$
$$\alpha := \alpha_i + (1-\alpha_i)\alpha, \qquad \text{[Ec. 2.21]}$$

e igualmente c y α son inicializados en 0, aunque en este caso el valor acumulado en α no es utilizado en la composición del color.

A continuación se estudian los algoritmos clásicos de visualización de volúmenes, tanto aquellos basados en *software*, como los acelerados por *hardware*.

3 RAY CASTING

El *ray casting* consiste en el lanzamiento de un rayo desde el ojo por cada píxel de la imagen, para evaluar directamente la Ec. 2.19 (ver Fig. 3.1). Cada rayo es discretizado una vez que entra al volumen, utilizando un paso h constante (típicamente h=1 ó h=0.5). Por cada posición discreta del rayo, se muestrea el volumen, se clasifica la muestra y se compone con las muestras que la preceden.

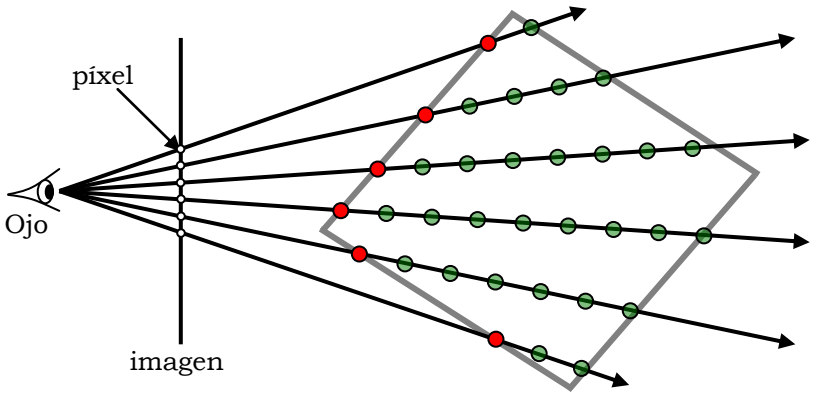


Figura 3.1: el *ray casting*. Por cada centro de píxel de la imagen se lanza un rayo. Aquellos rayos que atraviesan el volumen acumulan un color y opacidad. El paso entre muestras es constante. Un filtro suele ser aplicado para obtener las muestras requeridas. Los puntos rojos son los puntos de entrada de cada rayo en el volumen.

En esta guía se presentan dos algoritmos de *ray casting*, que difieren únicamente en la forma en que se calcula la intersección de los rayos con el volumen. El primer algoritmo calcula individualmente la intersección de un rayo con el *bounding box* (caja delimitadora) del volumen, mediante un algoritmo general de intersección rayo-caja. El segundo calcula la intersección de los rayos con el volumen mediante la rasterización por software del *bounding box* del volumen, utilizando interpolación lineal con corrección perspectiva. El algoritmo 3.1 presenta al algoritmo de *ray casting* en su primera versión.

A continuación se describe cada uno de los sub problemas mencionados en el algoritmo a alto nivel.

3.1 Definir el rayo $r(t)$ que parte del ojo hacia el píxel (x,y)

Basándonos en la terminología de openGL, asumimos que está definida una matriz de *modelview M* que lleva al volumen de coordenadas objeto a coordenadas de ojo. Igualmente, asumiremos que el ojo en coordenadas homogéneas está en la posición [0,0,0,1] mirando hacia $-z$. Si el *field of view* o *fov* es de 60^0, y colocamos el *near clipping plane* en $z = -1$, el ancho y alto del *near clipping plane*

es 2*tan(30^0). Considerando una corrección del ancho por el aspect ratio *ar* de la ventana de despliegue (*ar* = *width* / *height*), tenemos que el ancho del *near clipping plane* es W=*ar**2*tan(30^0), mientras que la altura viene dada por H=2*tan(30^0). Este rectángulo de W*H lo dividimos uniformemente en *width*height* puntos; así, cada píxel (x,y) tendrá coordenadas dentro del rango $x \in$[-*ar**tan(30), *ar**tan(30^0)] y $y \in$[-tan(30^0), tan(30^0)].

```
for each pixel (x,y)
   Definir el rayo r(t) que parte del ojo hacia el píxel (x,y)
   if (el rayo intersecta al volumen)
      Obtener puntos de entrada y salida (a y b) del rayo en el volumen
      A = 1
      C = 0
      D = length(b-a)
      Redefinir el rayo como r(t)=a+(t/D)(b-a), en el volumen
      for (t = 0; t <= D; t = t+h)
            Muestrear el volumen en la posición r(t), obteniendo s = s(r(t))
            Clasificar la muestra s, obteniendo c(s) y α(s)
            C = C + c(s) * α(s) * A
            A = A*(1-α(s))
      Pixel(x,y) = C
```

Algoritmo 3.1: algoritmo pseudo-formal de *ray casting*

Un rayo que parte del ojo [0,0,0,1] y pasa por el punto [x,y,–1,1] lo podemos definir paramétricamente en coordenadas homogéneas como $r(t)$ = [0,0,0,1]+t*[x,y,–1,0]. Para realizar la intersección volumen con rayo, convenientemente llevamos el rayo al espacio de volumen, pues en este espacio el volumen (y en consecuencia su *bounding box*) está alineado a los ejes, y la intersección se puede hallar mediante cálculos sencillos.

Para llevar el rayo $r(t)$ al espacio de volumen, basta llevar tanto su origen [0,0,0,1] como su vector director a dicho espacio utilizando la inversa del *modelview* M^{-1}. Así, el rayo se puede obtener mediante $r(t)=M^{-1}$*[0,0,0,1]+ t* M^{-1}*[x,y,–1,0] = p+tv. Note que el valor de $p=M^{-1}$*[0,0,0,1] es constante para todos los píxeles, y debe calcularse sólo una vez.

3.2 Obtener punto de entrada y salida del rayo en el volumen

La idea consiste en intersectar el rayo $r(t)$ en coordenadas de volumen con cada *slab* (rebanada) del *bounding box*. Un *slab* está delimitado con un par de planos paralelos de la caja. Si la caja está delimitada por $P_{min}(x_{min},y_{min},z_{min})$ y $P_{max}(x_{max},y_{max},z_{max})$, entonces el *slab* relativo al eje x está comprendido por los planos $x=x_{min}$ y $x=x_{max}$.

La idea del algoritmo es primero hallar la intersección del rayo con el *slab* de las x's; luego con el *slab* de las y's; y finalmente con el *slab* de las z's. Por cada *slab*, calculamos el valor de t para (a lo sumo) 2 intersecciones: t_0 y t_1. Al considerar los 3 *slabs*, de todos los valores de t_0 nos interesa el valor mayor, para ignorar intersecciones fuera de la caja delimitadora. Similarmente, para todos los valores de t_1 nos interesa el de menor valor. En el algoritmo 3.2 se muestra un código en C++ para

intersectar un rayo $r(t)=p+t.v$ con los *slabs* del volumen, obteniendo como resultado el punto de entrada del rayo en el volumen (*a*) y el punto de salida del rayo del volumen (*b*).

```
bool Intersection(vec3D &min, vec3D &max, vec3D &p, vec3D &v, vec3D &a, vec3D &b)
{
	float t0, t1, tnear = -INFINITE, tfar = INFINITE;
	for (int i=0; i<3; i++)
	{
		if (v[i] == 0)
		{
			// el rayo es paralelo al slab ⊥ al eje i-ésimo
			if (p[i] < min[i] || v[i] > max[i])
				return false; // rayo fuera del slab
			else
			{} // no intersecta los planos de este slab… ignorar este slab!
		}
		else
		{
			// el rayo no es paralelo al plano
			// p[i]+t.v[i] = min[i] ==> t0 = (min[i]-p[i])/v[i]
			t0 = (min[i]-p[i])/v[i];
			t1 = (max[i]-p[i])/v[i];
			// queremos que t0 sea el "t" de intersección con el plano más cercano
			if (t0> t1) swap(t0, t1);
			// queremos el tnear más lejano
			if (t0> tnear) tnear = t0;
			// queremos el tfar más cercano
			if (t1< tfar) tfar = t1;
			// no intersecta la caja
			if (tnear > tfar) return false;
			// la caja está totalmente detrás del rayo
			if (tfar < 0) return false;

		}
	}
	a = p + tnear * v;
	b = p + tfar * v;
	return true;
}
```

Algoritmo 3.2: intersección rayo-caja.

Si para los 3 *slabs* el algoritmo sobrevive los *tests* sin retornar, entonces hay intersección del rayo con el volumen, por lo que retorna *true*, junto los puntos de intersección *a* y *b*.

3.3 Muestrear el volumen

En general, el rayo atraviesa el volumen en una dirección arbitraria, y las muestras requeridas durante la travesía del rayo no coinciden con las muestras originales del volumen. Por lo tanto, se suele utilizar un filtro para el remuestreo. Comúnmente se emplea un filtro tri-lineal entre las 8 muestras más cercanas a la muestra a reconstruir. Sin embargo pueden aplicarse filtros más sofisticados que involucren más cantidad de muestras, como filtros gaussianos y bi-cúbicos [DRE88].

Al hacer interpolación tri-lineal, se realizan 7 interpolaciones lineales sobre las 8 muestras más cercanas del volumen al valor a reconstruir. Al hacer cada interpolación lineal entre dos vóxeles s_1 y s_2, preferiblemente utilice la fórmula $f(t)=s_1+t(s_2-s_1)$ en vez de $(1-t)s_1+ts_2$, para reducir el número de productos. El algoritmo 3.3 muestra cómo implementar la función de muestreo tri-lineal sobre el volumen. Este realiza 4 interpolaciones lineales en el eje *x*, 2 interpolaciones lineales en el eje *y*, y la última en el eje *z*.

```
float Sample(const GenType ***v, const vec3D &p) const
{
  // se assume que p está estrictamente dentro de los límites del volumen
  int x = int(p[0]), y = int(p[1]), z = int(p[2]);
  float tx = p[0] - x, ty = p[1] - y, tz = p[2] - z;
  float x1, x2, y1, y2, s;
  x1 = v[z][y][x]   + (v[z][y][x+1]   - v[z][y][x])   * tx;
  x2 = v[z][y+1][x] + (v[z][y+1][x+1] - v[z][y+1][x]) * tx;
  y1 = x1 + (x2-x1)*ty;
  x1 = v[z+1][y][x]   + (v[z+1][y][x+1]   - v[z+1][y][x]  ) * tx;
  x2 = v[z+1][y+1][x] + (v[z+1][y+1][x+1] - v[z+1][y+1][x]) * tx;
  y2 = x1 + (x2-x1)*ty;
  s  = y1 + (y2-y1)*tz;
  return s; // puede ser dividido por MAXSAMPLEVALUE para normalizar a [0,1]
}
```

Algoritmo 3.3: interpolación tri-lineal.

3.4 Clasificar la muestra *s*

Para aplicar la función de transferencia, en esta guía se asume que está definida como una función lineal a trozos, por cada canal RGBA. Esta función de transferencia permite clasificar las muestras escalares *s* del volumen, asignándoles un color $c(s)$ y una absorción $\tau(s)$. La edición puede hacerse canal a canal (Fig. 3.2a), o asignarle un color a cada punto de control de la función de absorción (Fig. 3.2b). En ambos casos, el valor RGBA para cada vóxel reconstruido *s* puede obtenerse por interpolación lineal en cada componente.

Por simplicidad, se puede discretizar la función de transferencia (definida por puntos de control) en una paleta o textura 1D de tipo RGBA, en donde el color y la absorción se obtienen por interpolación lineal de los puntos de control para un número finito de vóxeles. Se suele utilizar 2^8, o 2^{12} o 2^{16} valores en la paleta. Se recomienda que el número de valores cuantizados coincida con el rango de valores para un vóxel *s*. Por ejemplo, si los vóxeles están cuantizados a 8 bits, la paleta podrá contener 2^8 valores RGBA. Los valores RGBA deben estar normalizados en punto flotante, de manera que clasificar una muestra se reduzca a muestrear en esta paleta 1D. Para reducir el número de invocaciones a la función exponencial, el canal A puede almacenar la opacidad α la cual se obtiene de la absorción τ mediante Ec. 3.1.

$$\alpha(s) = 1 - \exp(-h\,\tau(s)). \qquad \text{[Ec. 3.1]}$$

Note que al cambiar el espaciado entre las muestras (h) se debe recalcular la opacidad de todas las muestras cuantizadas de la función de transferencia.

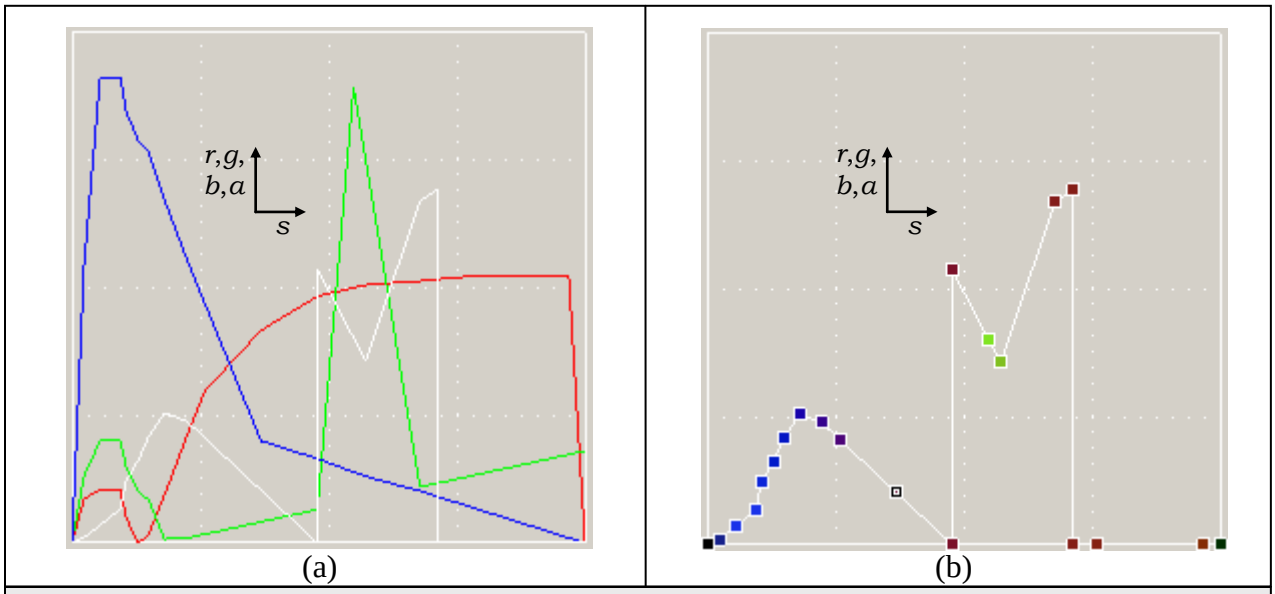


Figura 3.2: edición de la función de transferencia unidimensional. (a) Por canal; el eje *x* representa los valores posibles para un vóxel *s*, comúnmente normalizados en [0,1], y el eje *y* es cada componente de la tupla RGBA, también normalizadas en [0,1]. (b) Función de transferencia equivalente asignando colores RGB en los nodos de la función absorción.

3.5 Ray casting con rasterización

Para implementar el *ray casting* en el CPU, se debe contar con un método eficiente para intersectar cada rayo con el volumen, y deducir así el punto de entrada y el punto de salida del rayo en el volumen. Una manera eficiente es rasterizar las caras visibles del *bounding box* del volumen, similar a la solución basada en GPU propuesta en [KRU03]. Durante la rasterización, se deben interpolar por cada fragmento las coordenadas del volumen presentes en los vértices de las caras visibles. Así, por cada fragmento generado contamos con las coordenadas de volumen en dicho fragmento. Esas coordenadas de volumen interpoladas $a(x_a,y_a,z_a)$ no son más que la entrada del rayo en el volumen para un determinado píxel de la imagen. En la Fig. 3.3 se pueden observar las coordenadas de volumen en los vértices del *bounding box*. Aún cuando el *bounding box* se muestra en coordenadas de ojo, las coordenadas a interpolar están en coordenadas de volumen.

Para desplegar las caras visibles del volumen, primero creamos una matriz *modelview M* que lleve el *bounding box* del volumen de coordenadas de volumen (coordenadas objeto en este caso) a coordenadas de ojo. Conociendo la normal de cada cara del *bounding box* en coordenadas de volumen, y conociendo la posición del ojo en el mismo espacio, se puede determinar cuáles caras son visibles, y descartar las no visibles (*back face culling*). Al rasterizar las caras visibles del *bounding box*, e interpolar las coordenadas de volumen de sus vértices, obtenernos por interpolación las coordenadas de volumen del punto de entrada *a* de cada rayo en el volumen. También es posible conocer el punto de salida *b* de cada rayo en el volumen, mediante la rasterización de las caras no visibles del *bounding box*. Una alternativa a realizar *rendering* de las caras no visibles es determinar

la salida del rayo del volumen durante su travesía, cuando al menos una de las coordenadas ($x(t)$,$y(t)$,$z(t)$) de $r(t)$ esté fuera del rango del volumen.

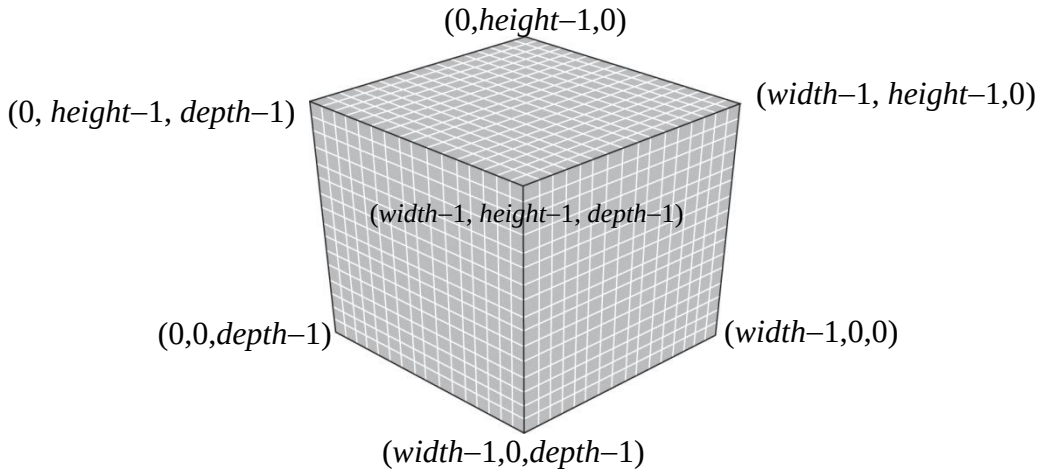


Figura 3.3: caras frontales del volumen en coordenadas de ojo. Hay que llevar traza de las coordenadas originales de volumen para hacer la interpolación.

Típicamente, al interpolar dos coordenadas (en nuestro caso coordenadas de volumen) v_0 y v_1 con un peso $\beta \in [0,1]$, el resultado es $v_\beta = (1-\beta)v_0 + \beta v_1$. Esta es una transformación afín, que trae notables defectos para *rendering* de volúmenes, pues no toma en cuenta la profundidad del fragmento en la interpolación. La corrección perspectiva [FAR01] solventa este problema, ponderando los pesos de interpolación según el inverso de la profundidad de los puntos interpolados [PER08]. Supongamos que v_0 está a una profundidad z_0 en coordenadas de ojo, y que v_1 a una profundidad z_1. La coordenada de textura corregida viene dada por:

$$v_\beta = \frac{((1-\beta)/z_0)v_0 + (\beta/z_1)v_1}{(1-\beta)1/z_0 + \beta/z_1}. \qquad \text{[Ec. 3.2]}$$

Puesto que un cuadrilátero del *bounding box* puede dividirse en dos triángulos, la ec. 3.2 puede extenderse con un término adicional para interpolar los valores de 3 vértices utilizando coordenadas baricéntricas. Así, si contamos con 3 vértices con coordenadas de volumen v_0, v_1, v_2, las cuales van a ser ponderadas con los pesos o coordenadas baricéntricas β_0, β_1, β_2, las coordenadas de volumen resultantes de la combinación convexa con corrección perspectiva vienen dadas por:

$$v_\beta = \frac{(\beta_0/z_0)v_0 + (\beta_1/z_1)v_1 + (\beta_2/z_2)v_2}{\beta_0/z_0 + \beta_1/z_1 + \beta_2/z_2}. \qquad \text{[Ec. 3.3]}$$

3.6 Consideraciones finales

Dado que el algoritmo de *ray casting* requiere de mucho cómputo, y aunado a problemas de localidad espacial, se dificulta la respuesta en tiempo real. El *rendering* se puede acelerar mediante la utilización de las técnicas de terminación temprana del rayo y salto de espacios vacíos [LEV90],

[DAN92]. La terminación temprana del rayo consiste en truncar la travesía del rayo cuando se acumule un umbral de opacidad α definido por el usuario (comúnmente de 0.95 a 0.99). El resto del rayo puede ignorarse, puesto que el aporte de las muestras remanentes es insignificante en el color final del píxel.

La estrategia de saltar espacios vacíos puede implementarse dividiendo el volumen en bloques. Por cada bloque se puede pre calcular el valor de muestra mínimo y el valor máximo (s_{min},s_{max}), y si la función de transferencia indica que todas esas muestras son transparentes, el bloque se considera vacío. Cuando un rayo entra a un bloque vacío, se determina el punto de salida del bloque para continuar la travesía del rayo [KRU03].

Los algoritmos de *ray casting* recorren la imagen en el mismo orden en que los píxeles están almacenados en memoria principal (algoritmo *image order* u orden de imagen) [LAC95]. Sin embargo, el problema principal es que no se accede al volumen en el orden en que está almacenado, ya que los rayos de visualización lo atraviesan en cualquier dirección. Como resultado, los algoritmos de *ray casting* ocupan .considerable tiempo en cálculos de posición de las muestras, y no explota la localidad espacial de datos volumétricos, limitando el rendimiento en computadores convencionales.

3.7 Ray Casting en GPU

Hoy en día, con la flexibilidad de hacer programas que se ejecutan en los procesadores de fragmentos, el *ray casting* puede ser implementado con aceleración en *hardware* en tiempo real [KRU03]. Es este caso, en cada programa de fragmento se evalúa la ecuación de composición volumétrica para un píxel de la imagen. Similar al caso de la implementación en CPU, se considera el volumen como un cubo (6 cuadriláteros), pero en este caso, un cubo unitario. Rasterizando las caras visibles de dicho cubo, e interpolando las coordenadas de los vértices de dichas caras, se disparan los programas de fragmentos con el punto de entrada de cada rayo en el volumen. El punto de salida del rayo puede hallarse de manera análoga al punto de entrada, pero esta vez rasterizando las caras no visibles del cubo (ver Fig. 3.4). En este caso, el *rendering* de las caras traseras puede realizarse sobre otro buffer (e.g. un *frame buffer object*), que luego podrá ser consultado por los programas de fragmentos disparados por la rasterización de las caras frontales, para así conocer también el punto de salida y calcular la longitud del rayo. Como alternativa, el punto de salida puede determinarse durante la travesía del rayo, cuando el punto del rayo a considerar se encuentre fuera del volumen. Similar a la implementación en CPU, el ojo es llevado al espacio de textura, para definir el rayo en este espacio, en donde la textura está alineada a los ejes.

Cabe destacar que el espacio de textura es $[0,1]^3$. El espacio de textura 3D y el espacio de color RGB tienen el mismo dominio: $[0,1]^3$, lo que hace posible que el rasterizador interpole los colores extremos del cubo RGB como si estuviese interpolando las coordenadas de textura (ver Fig. 3.5). Por ejemplo, al vértice (0,0,0) del *bounding box* de la textura 3D se le puede asignar el color (0,0,0) del cubo RGB, que representa análogamente la coordenada (0,0,0) de textura 3D. Similarmente, al vértice (1,1,1) del *bounding box* del volumen le asignamos el color (1,1,1) que equivale a la coordenada de textura (1,1,1).

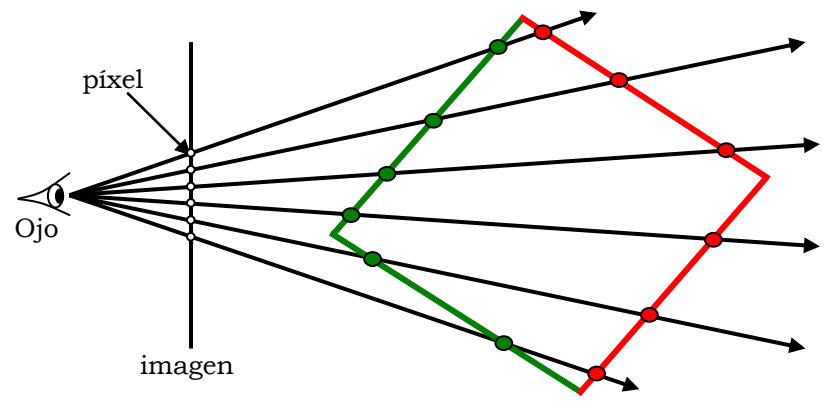


Figura 3.4: obtención del punto de entrada y punto de salida de cada rayo por interpolación lineal. Las líneas verdes corresponden a las caras frontales del volumen, y las líneas rojas a las traseras. Al rasterizar las caras frontales y traseras interpolando las posiciones de los vértices en coordenadas de textura, obtenemos por cada fragmento el punto de entrada y salida de cada rayo en la textura 3D que representa al volumen.

Para desplegar de las caras visibles del cubo, OpenGL® provee un mecanismo para remover las caras traseras de una geometría cualquiera mediante dos instrucciones:

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

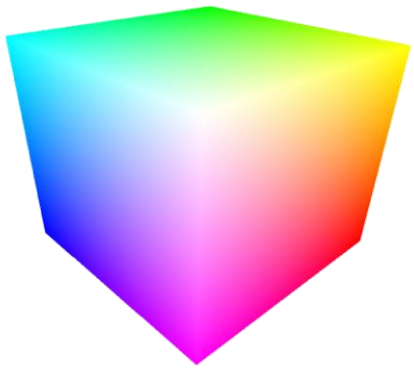


Figura 3.5: despliegue de las caras frontales de la caja delimitadora del volumen, interpolando los colores extremos del cubo unitario RGB. Cada color interpolado por fragmento representa igualmente las coordenadas de textura de la intersección de un rayo de visión con el volumen en coordenadas de volumen. Se debe utilizar la corrección perspectiva para obtener una adecuada interpolación.

Luego de habilitar el *back face culling*, se envía la geometría del cubo a la tarjeta gráfica como 12 triángulos, y el pipeline gráfico se encargará de remover los triángulos correspondientes a las caras traseras y rasterizar los triángulos correspondientes a las caras frontales. Los programas de

fragmento que se disparan recibirán como entrada un color RGB del cubo de color (ver Fig. 3.6) que representa la coordenada de textura del punto de entrada de un rayo en el volumen. Aún cuando las tarjetas gráficas actuales hacen corrección perspectiva por defecto durante la rasterización, se recomienda habilitar esta opción por compatibilidad.

```
glHint(GL_PERSPECTIVE_CORRECTION, GL_NICEST);
```

4 PLANOS ALINEADOS

La visualización de volúmenes basada en planos alineados busca explotar el poder de rasterización con texturizado de las tarjetas gráficas, para mezclar un conjunto de cortes del volumen. Para ello se utiliza una geometría intermedia formada por una pila de cuadriláteros o planos. Cada plano se va a texturizar con un corte particular del volumen, y se va a mezclar con el resultado parcial de *rendering* que se acumula en el *frame buffer* o búfer de color. De esta manera, cada píxel del *frame buffer* va acumulando un color y una opacidad que se va actualizando con el operador *over* (o el operador *under*) conforme el rasterizador vaya generando los fragmentos de cada corte. Luego de rasterizar y combinar todos los cortes del volumen, cada píxel contendrá una aproximación de la Ec. 2.19.

Los polígonos texturizados se despliegan típicamente desde el más lejano hasta el más cercano, utilizando el operador *over*. El operador *over* es especificado con la librería gráfica OpenGL® de la siguiente manera:

```
glEnable(GL_BLEND);
glBlend(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

El *blending* o mezcla actúa por píxel, y hace que el color c acumulado en el búfer de color se combine con el color arrojado por el procesador de fragmentos. La constante `GL_SRC_ALPHA` en el primer parámetro del procedimiento `glBlend` indica que el color RGB=c_i arrojado por el procesador de fragmento es multiplicado por su respectiva opacidad A=α_i, y la constante `GL_ONE_MINUS_SRC_ALPHA` indica que el color c acumulado en el búfer de color es multiplicado por $(1-\alpha_i)$. En caso que el color RGB de salida del procesador de fragmento ya esté pre-multiplicado por su respectiva opacidad (i.e. RGB=$\alpha_i c_i$), se puede configurar la mezcla como:

```
glBlend(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
```

Los polígonos también pueden ser desplegados desde el más cercano hasta el más lejano. En este caso, hay que configurar la mezcla para el operador *under*, pero necesariamente el color c_i y la opacidad α_i tienen que estar ya multiplicados (i.e. RGB=$\alpha_i c_i$). La configuración adecuada de la mezcla se realiza mediante la siguiente invocación:

```
glBlend(GL_ONE_MINUS_DST_ALPHA, GL_ONE);
```

en donde la constante `GL_ONE_MINUS_DST_ALPHA` especificada en el primer parámetro indica que el valor $\alpha_i c_i$ es a su vez multiplicado por $(1-\alpha)$ durante la mezcla (ver Ec. 2.20), y el resultado se suma con el color c. Los valores c y α son respectivamente el color y la opacidad del píxel correspondiente acumulados en el búfer de color.

Típicamente el conjunto de polígonos a texturizar pueden estar alineados al objeto (en cuyo caso se suele utilizar texturas 2D), o alineados al *viewport* (en cuyo caso se utiliza texturas 3D). A continuación se describe cada caso y su implementación.

4.1 Planos alineados al objeto

Cuando se utilizan planos alineados a los ejes del objeto, el volumen puede ser pensado como una pila de cortes que se almacenan en texturas 2D. Al rasterizar estos cortes (desde el más lejano al más cercano al ojo) haciendo la mezcla con el operador *over*, se obtiene en cada píxel del *frame buffer* una aproximación de la Ec. 2.19 (ver Fig. 4.1).

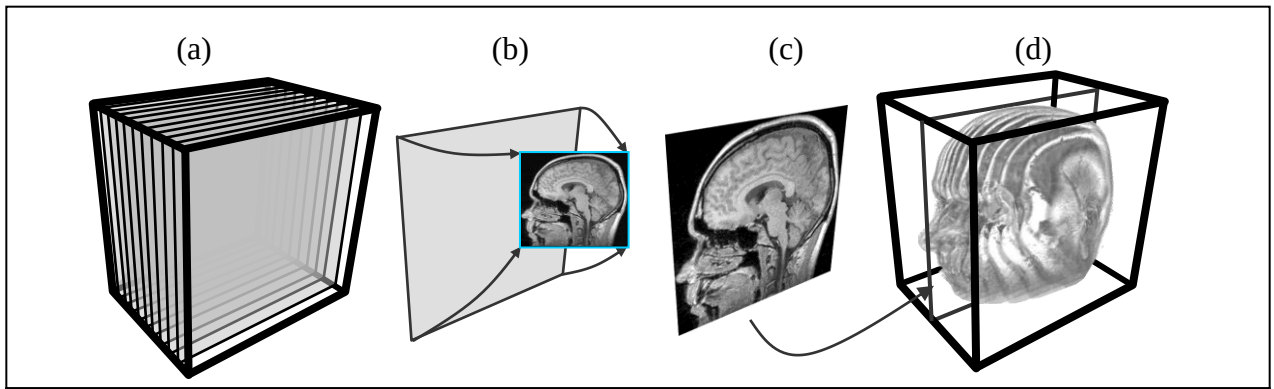


Figura 4.1: despliegue de polígonos texturizados alineados al objeto. Los polígonos alineados (a) son texturizados uno por uno, asignándole coordenadas de textura a cada vértice (b). Cada polígono texturizado (c) es mezclado mediante el operador *over*, para obtener la imagen resultante (d). Note que se pueden distinguir los cortes del volumen por insuficiencia en el número de cortes, y ausencia de interpolación espacial.

Al observar los cortes perpendicularmente, los artefactos presente en la Fig. 3.7d son minimizados. A medida que los cortes son menos perpendiculares a la vista, los artefactos se hacen más notables. El peor caso es cuando los cortes son paralelos a la vista, en donde se podría apreciar una imagen vacía o algunas líneas en el *rendering*. Para reducir estos artefactos, se utilizan 3 copias del volumen (una por cada eje), y tres pilas de texturas 2D (ver Fig. 4.2). Igualmente, se generan 3 pilas de polígonos, una por cada eje. Durante el *rendering*, se elije la pila de polígonos (y así las texturas) más perpendiculares a la dirección de visualización [LAM99] (ver sección 6.1). Así, los artefactos más notorios se obtendrán en ángulos de 45 grados. La Fig. 4.3 muestra como al pasar de los 45 grados, se elije otra pila de polígonos.

Para utilizar el operador *over*, es necesario desplegar los cortes desde el más lejano al más cercano. En la Fig. 4.2c se seleccionó la copia asociada al eje *y*. Pero cuál es el orden en que deben desplegarse los polígonos?. Debido a que el coeficiente y=0.885 es positivo, eso indica que el corte más lejano es $y=y_{max}$, y el más cercano es y=0. Si el coeficiente hubiese sido negativo, el más lejano sería y=0 y el más cercano $y=y_{max}$. En términos generales, el signo del término que corresponde al eje principal en el vector de visualización indica el orden en que deben ser desplegados los cortes. Por consiguiente, hay 3 ejes principales posibles, y por cada eje dos posibilidades en el orden de *rendering*, generando en total 3*2=6 casos.

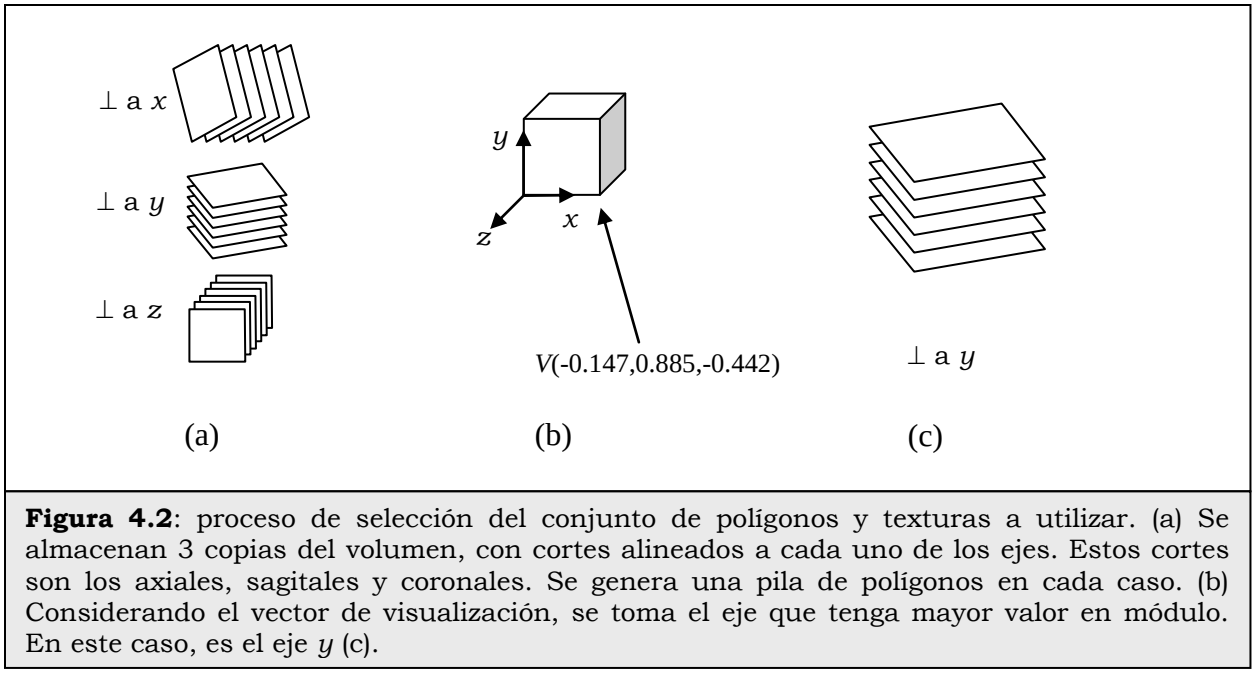


Figura 4.2: proceso de selección del conjunto de polígonos y texturas a utilizar. (a) Se almacenan 3 copias del volumen, con cortes alineados a cada uno de los ejes. Estos cortes son los axiales, sagitales y coronales. Se genera una pila de polígonos en cada caso. (b) Considerando el vector de visualización, se toma el eje que tenga mayor valor en módulo. En este caso, es el eje *y* (c).

El número de polígonos a texturizar está limitado en principio a las dimensiones del volumen. Sin embargo, utilizando multi-texturas, se puede interpolar cortes durante el *rendering* vía *register combiners* (combinadores de registros) para aproximar la interpolación tri-lineal [ENG01], [REZ00]. En la actualidad se puede realizar la interpolación tri-lineal entre dos cortes del volumen programando los procesadores de fragmentos.

Entre las bondades de este método, se considera su fácil implementación y excelente tiempo de respuesta. La portabilidad de esta técnica es otro punto a favor, ya que la aplicación de texturas 2D es soportada por la mayoría de las tarjetas gráficas. Para volúmenes relativamente grandes, en donde una copia del volumen excede la capacidad de la tarjeta gráfica (pero no los cortes individuales), el *rendering* puede efectuarse paginando los cortes del volumen. Esta paginación es típicamente realizada en forma automática por el manejador o *driver* gráfico.

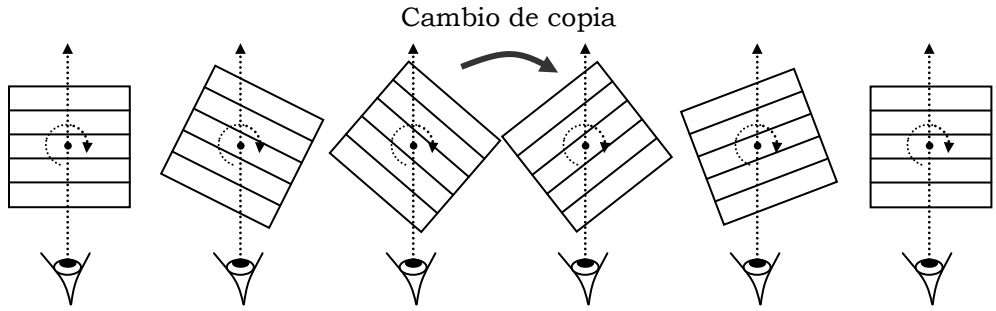


Figura 4.3: visualización de volúmenes utilizando texturas 2D. Se almacenan 3 copias del volumen, y se selecciona la pila de cortes más perpendicular a la dirección de visualización.

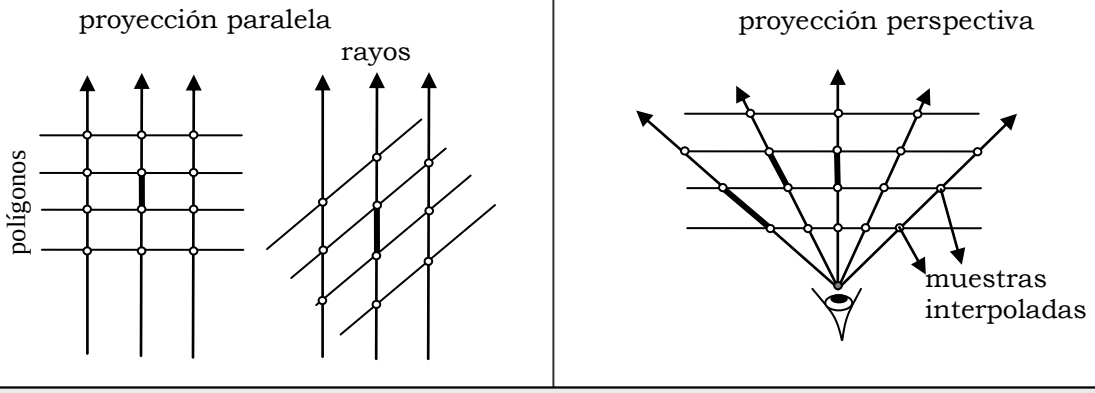


Figura 4.4: distancia entre muestras. Al texturizar polígonos alineados al objeto, la distancia entre muestras cambia por rotación (proyección paralela) o por rayo (proyección perspectiva).

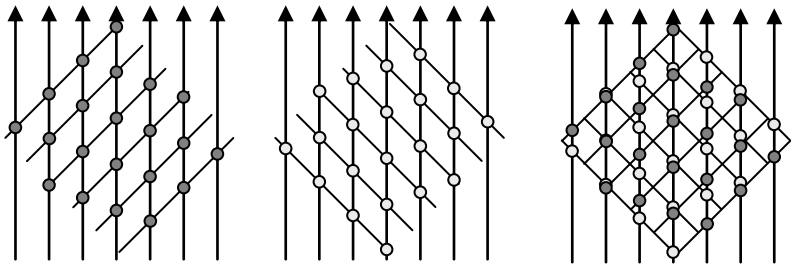


Figura 4.5: despliegue alrededor de los 45 grados para el caso de proyección paralela. Note que las muestras utilizadas entre una copia y otra difieren, por lo que el brillo en la imagen final puede cambiar ligeramente entre par de fotogramas.

Cabe destacar que al rotar los polígonos, cambia la distancia entre las muestras. Para obtener consistencia entre *frames* (cuadros de imagen o fotogramas[5]) hay que corregir las opacidades de las

[5] Fotograma: también llamado cuadro de imagen, o *frame* en inglés, es una imagen particular dentro de un conjunto de

muestras, ya que a mayor distancia recorrida por el rayo entre dos muestras, mayor es la absorción o atenuación (ver Fig. 4.4). Las desventajas de este método son la necesidad de mantener 3 copias del volumen y la ausencia de interpolación espacial. En ángulos cercanos a 45 grados, justo en el momento de cambiar la copia activa, puede notarse un pequeño cambio en el brillo de la imagen, debido a que las muestras utilizadas en la composición volumétrica suelen diferir (ver Fig. 4.5).

Detalles para la implementación:
Para implementar esta técnica, se almacenan los cortes del volumen por cada eje. Específicamente, si el volumen tiene dimensiones *n*m*k* vóxeles, entonces se tendrán *n* texturas 2D para los cortes perpendiculares a *x*, *m* texturas 2D para los perpendiculares a *y*, y *k* texturas 2D para los perpendiculares a *z*. Los polígonos (cuadriláteros) pueden ser definidos en el espacio de volumen; es decir, entre las esquinas de dicho volumen: (0,0,0) y (*n*,*m*,*k*). Luego, estos cortes pueden ser escalados según las dimensiones reales del volumen, si se conocen. Algunos aparatos de muestreo definen el tamaño de un vóxel en sus tres dimensiones, y puede utilizarse para determinar el tamaño físico del volumen, sabiendo que tiene *n*m*k* vóxeles.

A cada vértice a de un polígono cuadrilátero se le asigna una coordenada de textura, la cual suele ser (0,0), (0,1), (1,0) ó (1,1), como se puede apreciar en la Fig. 4.6. Sin embargo, en las áreas mapeadas fuera del área del rectángulo blanco segmentado, no están presentes las 4 muestras para la interpolación bilineal, sino tan solo dos muestras para una interpolación lineal; más aún, en las esquinas de la textura no se realiza ninguna interpolación, pues se cuenta con un único vóxel. Esto puede adicionar artefactos en el *rendering*, puesto que se puede notar una diferencia de calidad en las fronteras del volumen.

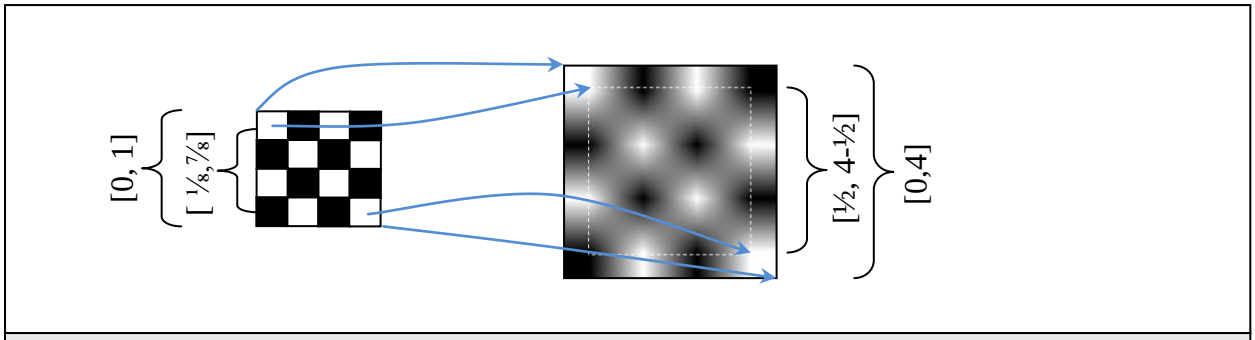


Figura 4.6: asignación de coordenadas de textura a los polígonos. (a) El espacio de textura. (b) Un polígono que representa a la textura. Tanto el tamaño del polígono como el dominio de textura son reducidos adecuadamente para una mejor calidad visual en las fronteras del volumen. En este ejemplo, el polígono de 3x3 ocupa un área de aprox. 100x100 píxeles en coordenadas imagen, por lo que se puede apreciar los valores interpolados del tablero de ajedrez.

Para reducir estos artefactos, se suele restringir el área del volumen, y así el espacio de textura, de manera tal de garantizar homogeneidad en el *rendering*, aún en las áreas de borde del volumen. Los

---

imágenes que componen una animación.

polígonos que se generen estarán restringidos al sub volumen delimitado por (½,½,½) y (*n*–½, *m*–½, *k*–½), y las coordenadas de textura estarán reducidas a medio píxel en ambos extremos (ver Fig. 4.6). Por ejemplo, para la copia asociada al eje *x*, las coordenadas de textura de los cuatro vértices de un polígono serían:

$$\left(\frac{1}{2m},\frac{1}{2k}\right),\left(\frac{1}{2m},1-\frac{1}{2k}\right),\left(1-\frac{1}{2m},\frac{1}{2k}\right)\ y\ \left(1-\frac{1}{2m},1-\frac{1}{2k}\right)$$

Para construir los polígonos, otra alternativa válida es pensar que la coordenada objeto (0,0,0) está en el centro del primer vóxel del volumen. Bajo esta premisa, los polígonos a generar están delimitados por las esquinas (0,0,0) y (*n*–1, *m*–1, *k*–1).

Los vértices de los polígonos son transformados por la matriz de modelación y vista, mientras que las coordenadas de volumen son interpoladas por fragmento. El *fragment shader* recibe las coordenadas de textura interpoladas, así como la textura en sí donde realizar el muestreo. El téxel muestreado va a servir luego como coordenadas de textura para muestrear la textura 1D correspondiente a la función de transferencia discretizada. La salida RGBA del fragmento será entonces la muestra clasificada, que será combinada con el correspondiente píxel del *frame buffer* mediante el operador *over*.

En algunas tarjetas gráficas, las dimensiones de las texturas 2D deben ser potencias de dos. En este caso, si las dimensiones del volumen no son todas potencias de dos, hay que completar las texturas con vóxeles de holgura, pero restringir el dominio de cada coordenadas a [0,*p*/(*p*+*holgura*)] en vez de [0,1], donde $p \in \{n,m,k\}$. Eliminando medio píxel en los extremos de la textura, el espacio de textura para la coordenada *p* quedará reducido a:

$$\left(\frac{1}{2p},\frac{p}{p+holgura}-\frac{1}{2p}\right)$$

4.2 Planos alineados al *viewport*

El volumen puede ser almacenado en una textura 3D, en cuyo caso se requiere una sola copia en la memoria de GPU para su *rendering*. Akeley [AKE93] en 1993 fue el primero en mencionar la posibilidad de utilizar una textura 3D para visualización de volúmenes. Seguidamente, esta idea fue utilizada y mejorada en otros trabajos del año siguiente [CAB94], [GUA94], [WIL94]. Con la técnica de planos alineados al *viewport*, se texturizan y despliegan una serie de polígonos cuadriláteros, paralelos al *viewport*, en sentido *back to front* (desde el más lejano hasta el más cercano) y se mezclan con el operador *over* (ver Fig. 4.7).

Este método utiliza comúnmente interpolación tri-lineal para reconstruir las muestras requeridas. Para el caso de proyección paralela, la distancia entre muestras interpoladas es siempre la misma, independientemente del ángulo de rotación. Para la proyección perspectiva, la distancia entre muestras varía rayo por rayo (ver Fig. 4.7).

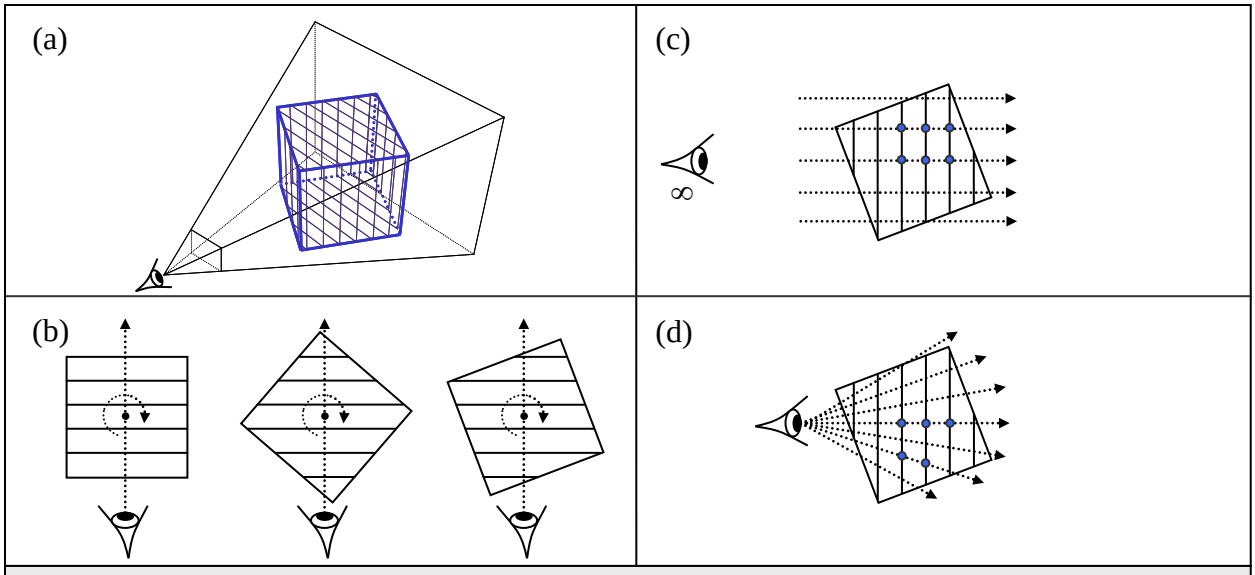


Figura 4.7: visualización de volúmenes utilizando texturas 3D y planos alineados al *viewport*. (a) Planos alineados al *viewport* ajustados al volumen. (b) Se muestran los planos alineados para 3 ángulos distintos de rotación. (c) Proyección paralela; la distancia entre muestras no varía. (d) Proyección perspectiva; la distancia entre muestras varía rayo a rayo.

Para implementar esta técnica de visualización de volúmenes, podemos notar que el número de polígonos requeridos para la visualización puede variar según el ángulo de rotación. Por ejemplo, si el volumen contiene *n*m*k* muestras, bastarían *p=k/h* polígonos si visualizamos perpendicularmente los cortes *xy*. En cambio, si hacemos una visualización oblicua desde el vóxel de centro (0,0,0) al vóxel de centro (*n*–1,*m*–1,*k*–1), necesitaríamos $p=\left\lceil \sqrt{n^2+m^2+k^2} \right\rceil / h$ polígonos. Por simplicidad, se envía a la tarjeta gráfica siempre $p=\left\lceil \sqrt{n^2+m+k^2} \right\rceil / h$ polígonos, y mediante alguna técnica de *clipping*, se puede remover los polígonos o fragmentos de ellos que estén fuera del dominio de la textura (ver Fig. 4.7a). Similarmente, debemos conocer las dimensiones del volumen en coordenadas de ojo, para establecer el tamaño (ancho y alto) de los polígonos a desplegar. Podemos así considerar los puntos extremos de la textura 3D (e.g. los puntos *min*[0,0,0,1] y *max*[1,1,1,1] de la textura) y llevarlos a coordenadas de ojo mediante la transformación dada por la matriz *modelview M*. Así, la distancia $d = length(M.max - M.min)$ entre estos dos puntos en coordenadas de ojo nos indicará el alto y ancho de los polígonos a desplegar. Note que mientras *d* nos indica el ancho y el alto de los polígonos, este valor nos va a indicar también en qué espacio se van a distribuir los *p* polígonos a lo largo del eje *z*.

Sea $c = (M.max - M.min) * 0.5$ el centro del volumen en coordenadas de ojo, el *i*-ésimo cuadrilátero a desplegar (con *i*=0..*p*–1) en coordenadas de ojo viene dado por las 4 esquinas:

$$\left(c_x \pm \frac{d}{2}, c_y \pm \frac{d}{2}, c_z + \frac{p}{2} + ih\right)$$

Finalmente, hay que asignarle una coordenada de textura a cada vértice de cada polígono. Contando con la matriz *modelview* que transforma el espacio de textura al espacio ojo, podemos simplemente

utilizar su inversa para obtener la coordenada de textura de un vértice dado en el espacio ojo. Así, para un vértice $[x,y,z,1]$, su coordenada de textura $[s,t,r,1]$ es simplemente $[s,t,r,1]^T = M^{-1}[x,y,z,1]^T$. Las coordenadas de textura asignadas a los vértices se interpolan por fragmento, de manera tal que el *fragment shader* podría simplemente muestrear la textura 3D en la coordenada interpolada para obtener la muestra (vóxel) normalizada del volumen en [0,1]. Este valor de vóxel normalizado es luego clasificado, resultando un color RGB y una opacidad A. Similar a las técnicas previas estudiadas, para clasificar el vóxel se utiliza su propio valor como coordenada de textura de la función de transferencia, la cual ha sido discretizada en una textura uni-dimensional de tipo RGBA.

5 CONCHAS ESFÉRICAS

Las técnicas de planos alineados al objeto (basados en texturas 2D) y al *viewport* (basados en texturas 3D) generan ligeros artefactos bajo proyección perspectiva, si la variación de la distancia entre muestras por rayo no es considerada. LaMar et al. [LAM99] proponen una solución utilizando conchas esféricas (*spherical shells*) texturizadas con la textura 3D del volumen (ver Fig. 5.1). Debido a que las conchas son concéntricas respecto al ojo, la distancia entre muestras entre pares de conchas es constante, para todos los rayos que parten desde el centro.

Cada concha o semiesfera es aproximada por un conjunto de triángulos, para poder explotar el *hardware* gráfico. Si la triangulación es muy burda (pocos triángulos), también lo será la aproximación del mallado a la esfera, agudizando la variación de la distancia entre cada par de muestras. Una triangulación muy fina, requiere de muchos vértices a ser enviados al sistema gráfico, en detrimento del tiempo de respuesta. Un balance entre calidad y velocidad de *rendering* debe ser considerado al construir la triangulación. Al igual que con planos alineados al *viewport*, un cómputo adicional debe ser efectuado para descartar geometría fuera del volumen, el cual puede efectuarse por *software* en coordenadas objeto [PLA02], [BEN05], o por *hardware* configurando los planos de corte [BOA01].

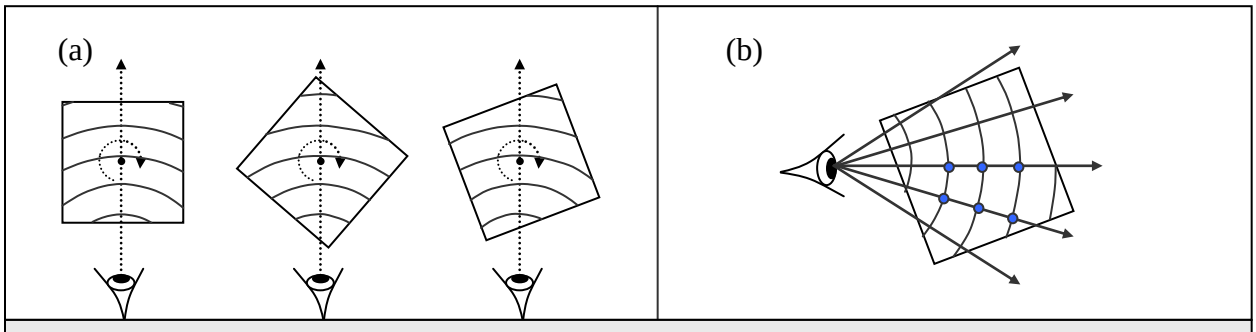


Figura 5.1: visualización de volúmenes utilizando texturas 3D y *spherical shells*. (a) Se muestran las conchas esféricas para 3 ángulos distintos de rotación. (b) Proyección perspectiva. Observe que la distancia entre muestras es siempre la misma.

Madero et al. [MAD02] muestran en la práctica que la calidad de *rendering* utilizando planos alineados al *viewport* es superior que conchas esféricas al visualizar al volumen perpendicularmente respecto a alguna de sus caras. Las conchas esféricas en cambio generan mejores resultados en ángulos cercanos a los ±45 y ±135 grados (ver Fig. 5.2). Debido a la interpolación tri-lineal en la

reconstrucción de las muestras, se reducen los artefactos que generan algoritmos basados en texturas 2D y *Shear-Warp*. Sin embargo, la interpolación tri-lineal suele ser más lenta que la interpolación bi-lineal, puesto que requiere de más operaciones aritméticas y accesos al volumen. Si el volumen es muy grande, y la textura no puede almacenarse completamente en memoria de textura, el *rendering* del volumen no puede efectuarse, a menos que se utilice una técnica más sofisticada conocida como *bricking* [LAM99].

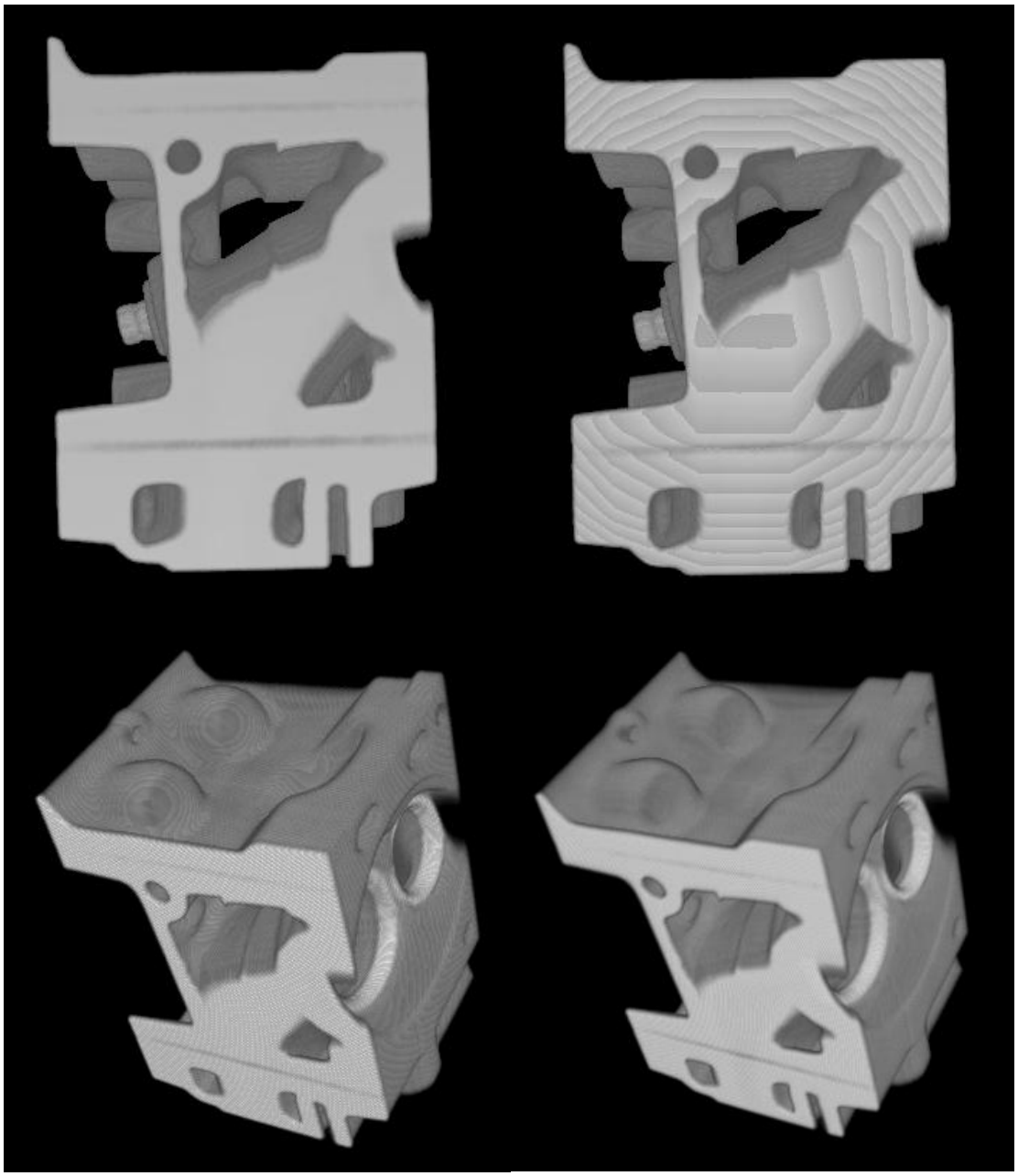


Figura 5.2: planos alineados al *viewport* (izquierda) versus *sphericall shells* (derecha).

Los algoritmos de planos alineados y conchas esféricas utilizan los píxeles del búfer de color para almacenar la evaluación parcial de la ecuación de *rendering*, y se accede y actualiza mediante el operador de *blending* al desplegar cada polígono. La profundidad (número de bits por canal) del búfer de color juega en este caso un papel muy importante en la precisión del *rendering*, puesto que en cada actualización de un píxel se pierde precisión al momento de ser almacenado en dicho búfer. Esto se hace evidente cuando cada componente es de tan solo 8 bits, mientras que el resultado del *blending* podría estar en punto flotante.

6 SHEAR-WARP

La factorización *Shear-Warp* [LAC95] consiste en transformar el volumen a un sistema de coordenadas intermedio (espacio objeto desplazado, o *sheared object space*) en el que todos los rayos de visualización son paralelos al tercer eje de coordenadas. Esto permite la generación de una imagen intermedia deformada, proyectando corte a corte del volumen sobre la imagen, explotando la localidad tanto en el volumen como en la imagen. Luego, la deformación de la imagen intermedia es corregida mediante una transformación 2D de *warping*, obteniendo así la imagen final (ver Fig. 6.1).

Figura 6.1: visualización volumétrica usando la factorización *Shear-Warp*.

Matemáticamente, la factorización *Shear-Warp* consiste en expresar la matriz de visualización *M* (la cual en este caso es el producto de 4 matrices: *viewport*, proyección, vista y modelo) 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*. Al igual que en el caso de utilizar planos alineados con texturas 2D, se requieren de tres copias del volumen (una por cada eje), y durante el *rendering* se elije la copia más perpendicular a la dirección de visualización. En la Fig. 6.2 se muestran los sistemas de coordenadas involucrados en el proceso.

El algoritmo de visualización de volúmenes basado *Shear-Warp* con proyección paralela puede resumirse en los siguientes pasos:

- Determinar el eje principal
- Transformar al sistema de coordenadas estándar
- Transformar al sistema de coordenadas desplazado o *sheared*
- Proyectar a la imagen intermedia
- Aplicar *Warping*

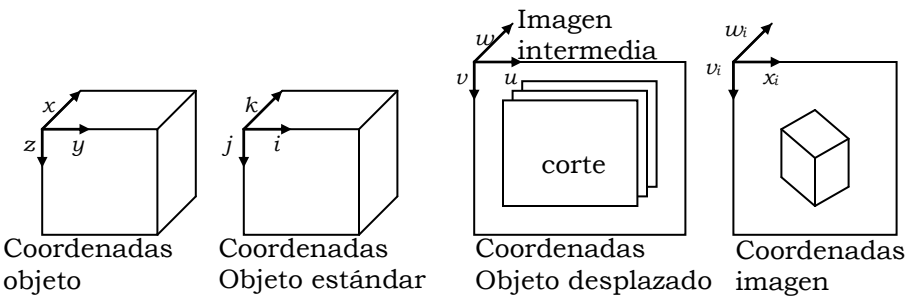


Figura 6.2: sistemas de coordenadas involucrados en la derivación de la factorización *Shear-Warp*.

6.1 Determinar 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_{obj} en el espacio objeto puede obtenerse de la última columna de la matriz M^{-1}, la cual a su vez puede obtenerse de los coeficientes de M de la siguiente forma:

$$v_{obj} = \begin{pmatrix} m_{12} * m_{23} - m_{22} * m_{13} \\ m_{21} * m_{13} - m_{11} * m_{23} \\ m_{11} * m_{22} - m_{21} * m_{12} \end{pmatrix} \qquad \text{[Ec. 6.1]}$$

Así, la determinación del eje principal c es resuelto mediante la siguiente expresión:

$$c = \underset{x,y,z}{Max}\left(| v_{obj,x} |, | v_{obj,y} |, | v_{obj,z} | \right) \qquad \text{[Ec. 6.2]}$$

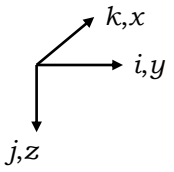
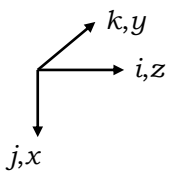
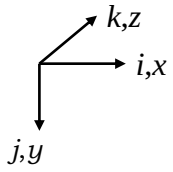
	Eje principal x	Eje principal y	Eje principal z
Matriz de Permutación P	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0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

Figura 6.3: correspondencia entre el sistema de coordenadas objeto y el sistema de coordenadas objeto estándar.

6.2 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 Fig. 6.3 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 $[i,j,k,1]^t = P^*[x,y,z,1]^t$.

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

$$v_{so} = P^* v_{obj} \qquad \text{[Ec. 6.3]}$$

6.3 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. 6.4 se muestra gráficamente cómo se determina en particular el factor de *shear* s_i (Fig. 6.4a), así como el efecto que la transfomación M_{shear} en el volumen (Fig. 6.4b). Luego, la transformación M_{shear} puede ser escrita como:

$$M_{shear} = \begin{pmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{[Ec. 6.4]}$$

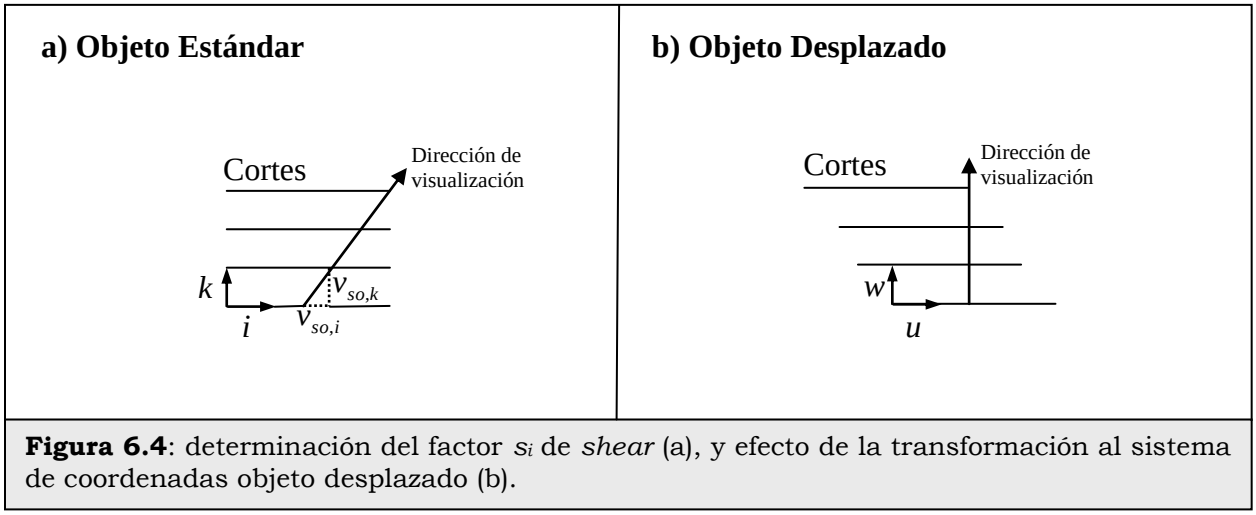


Figura 6.4: determinación del factor s_i de *shear* (a), y efecto de la transformación al sistema de coordenadas objeto desplazado (b).

6.4 Proyección a la imagen intermedia

Los cortes del volumen en el sistema de coordenadas objeto desplazado son proyectados y mezclados sobre la imagen para dar origen a la imagen intermedia. Sin embargo, el sistema de coordenadas en el cual yace esta imagen no es conveniente, ya que los factores de *shear* negativos harían proyectar porciones del volumen en coordenadas negativas de la imagen. Por ello, se realiza previamente una traslación del sistema de coordenadas objeto desplazado, para que el origen de este sistema sea 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 $t_j = -s_j * k_{Max}$
- $s_i < 0$ y $s_j \geq 0 \Rightarrow t_i = -s_i * k_{Max}$ y $t_j = 0$
- $s_i < 0$ y $s_j < 0 \Rightarrow t_i = -s_i * k_{Max}$ y $t_j = -s_j * k_{Max}$

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}.

$$M_{shearT} = \begin{pmatrix} 1 & 0 & 0 & t_i \\ 0 & 1 & 0 & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & s_i & 0 \\ 0 & 1 & s_j & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & s_i & t_i \\ 0 & 1 & s_j & t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{[Ec. 6.4]}$$

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 $k=k_{max}$.

El tamaño de la imagen intermedia se calcula a partir de las coordenadas máximas en *u* y en *v* del volumen en el sistema de coordenadas desplazado. Durante la composición de los cortes, se recorren los vóxeles y píxeles en el mismo orden en que están almacenados. Al proyectar el primer vóxel sobre la imagen, se puede determinar un filtro que indicará el aporte del vóxel a los píxeles correspondientes. Debido a que los factores de escala del volumen no juegan ningún papel en el espacio desplazado, cada vóxel afecta en general a 4 píxeles de la imagen intermedia. Los pesos de un filtro de 2x2 pueden calcularse simplemente por la intersección del vóxel proyectado con los 4 píxeles. Debido a que la proyección es paralela, y los cortes son paralelos a la imagen en el sistema de coordenadas desplazado, los pesos del filtro son los mismos para cualquier vóxel proyectado del mismo corte, lo cual genera un ahorro sustancial en cálculo.

6.5 Warping

A continuación se deduce la descomposición de la matriz $M' = P*M$ como el producto de dos matrices: $M_{warp}*M_{viewT}$.

$$M' = M_{warp} * M_{shearT} \Rightarrow M_{warp} = M' * M_{shearT}^{-1} \quad \text{[Ec. 6.4]}$$

$$M_{shearT}^{-1} = \begin{pmatrix} 1 & 0 & -s_i & -t_i \\ 0 & 1 & -s_j & -t_j \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[Ec. 6.5]}$$

Sustituyendo la Ec. 6.5 en Ec. 6.4 nos queda:

$$M_{warp} = \begin{pmatrix} m'_{11} & m'_{12} & m'_{13} - s_i * m'_{11} - s_j * m'_{12} & m'_{14} - t_i * m'_{11} - t_j * m'_{12} \\ m'_{21} & m'_{22} & m'_{23} - s_i * m'_{21} - s_j * m'_{22} & m'_{24} - t_i * m'_{21} - t_j * m'_{22} \\ m'_{31} & m'_{32} & m'_{33} - s_i * m'_{31} - s_j * m'_{32} & m'_{34} - t_i * m'_{31} - t_j * m'_{32} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[Ec. 6.6]}$$

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*. El *warping* puede implementarse tanto por software [LAC95] como acelerado por hardware [CAR00]. Una posible implementación por software recorre los píxeles de la imagen destino. Por cada píxel (u_i,v_i) se determina la posición (u,v) en la imagen intermedia mediante la transformada inversa (M_{warp}^{-1}). Mediante un filtro (típicamente bilineal) se muestrea la imagen intermedia en (u,v) para obtener el píxel resultante. Dado que en el algoritmo de *warping*, las coordenadas u_i, v_i varían iterativamente de 1 en 1, se pueden reducir los cálculos del producto matriz-vector en cada iteración en sólo sumas, haciendo que el algoritmo sea incremental.

La implementación acelerada por hardware considera la imagen intermedia como una textura aplicada sobre un polígono del mismo tamaño de la imagen intermedia [CAR00]. Al transformar los vértices del polígono mediante la matriz de *warping* M_{warp}, nos dará como resultado la corrección deseada.

6.6 Consideraciones adicionales de implementación

El algoritmo de visualización de volúmenes basado en *Shear-Warp* es *object order* como el *splatting* [LAC95], e *image order* como el *ray casting*, por lo que su rendimiento es superior. Debido a que es *object order*, se puede implementar la técnica de aceleración basada en salto de espacios vacíos mediante la compresión sin pérdida RLE [LAC95].

Actualmente no se está haciendo investigación sobre *Shear-Warp*, debido a la capacidad de cómputo que hay en el GPU. Tampoco encontramos investigación con implementaciones basadas en aplicación de texturas 2D y 3D.

7 LUZ EXTERNA

En cualquiera de las técnicas de *rendering* de volúmenes mencionadas, es necesario hacer la composición de los vóxeles que se solapan sobre cada píxel de la imagen en la proyección. Por cada vóxel se aplica un modelo de iluminación; hasta ahora la iluminación se ha basado en emisión y absorción, asumiendo una luz trasera en el volumen, acorde al modelo óptico utilizado. Tal y como lo hemos venido describiendo, el valor final de un vóxel se determina a partir de la función de transferencia. Sin embargo, al considerar una luz externa en el modelo, se puede aplicar adicionalmente un modelo de iluminación en cada vóxel como Phong [Ec. 7.1] o Blinn-Phong [Ec. 7.2]. Esto va a permitir mostrar reflexión especular y difusa en las superficies del volumen, sin la necesidad de la reconstrucción de dichas superficies. Los valores involucrados los modelos de iluminación de Phong y Blinn-Phong son:

- e: posición del ojo
- p: las coordenadas del vóxel a iluminar
- La: intensidad de luz ambiental
- Ld: componente difusa de la luz
- Ls: componente especular de la luz
- Ka: rata de reflexión de la luz ambiental
- Kd: rata de reflexión de la luz difusa entrante
- Ks: rata de reflexión de la luz especular entrante
- N: vector normal en el vóxel p
- L: vector de luz. Si la luz está en la posición l, se calcula como $L=(l-p)$. Si la luz está en el infinito, L es un vector constante para todos los puntos p.
- V: vector de visualización. Se calcula como $V=(e-p)/\|e-p\|$ para el caso de proyección perspectiva. Para proyección paralela, V es constante para todo p, constituyendo la dirección de máxima especularidad para el modelo Phong.
- R: vector de luz reflejado en el punto p. Se calcula como $R=2\langle L.N\rangle.N-L$
- H: vector medio entre L y V. $H=(L+V)/\|L+V\|$. Es la dirección de máxima especularidad para el modelo de Blinn-Phong. Se utiliza para evitar el cálculo del vector reflejado R del modelo de Phong. Note que cuando si la luz está en el infinito, y la proyección es paralela, entonces H es constante para todos los vóxeles del volumen, reduciendo notablemente los cálculos.
- s: coeficiente de *Shininess* (brillo) en $[0,\infty)$

$$I(p)=La*Ka + Ld*Kd*\langle N,L\rangle + Ls*Ks*\langle R,V\rangle^{s} \qquad \text{[Ec. 7.1]}$$

$$I(p)=La*Ka + Ld*Kd*\langle N,L\rangle + Ls*Ks*\langle N,H\rangle^{s} \qquad \text{[Ec. 7.2]}$$

Los valores de La, Ld, Ls, Ka, Kd, Ks son típicamente valores RGB, donde cada componente está en el rango de [0,1]. El vector gradiente en un vóxel (x,y,z) se calcula mediante diferencias finitas. Así, $grad(x,y,z) = (v_{x+1,y,z} - v_{x-1,y,z},\ v_{x,y+1,z} - v_{x,y-1,z},\ v_{x,y,z+1} - v_{x,y,z-1})$. Este vector se normaliza, obteniendo $N(x,y,z) = grad(x,y,z)/|grad(x,y,z)|$.

Si se calcula el modelo de iluminación en el sistema de coordenadas de ojo, habría que llevar la normal de cada vóxel a este sistema de coordenadas, lo cual es un producto matriz-vector por vóxel.

Preferiblemente se calcula el modelo de iluminación en el sistema de coordenadas objeto. Así, la luz y el ojo son llevados al sistema de coordenadas objeto mediante la inversa de la matriz de modelación y vista.

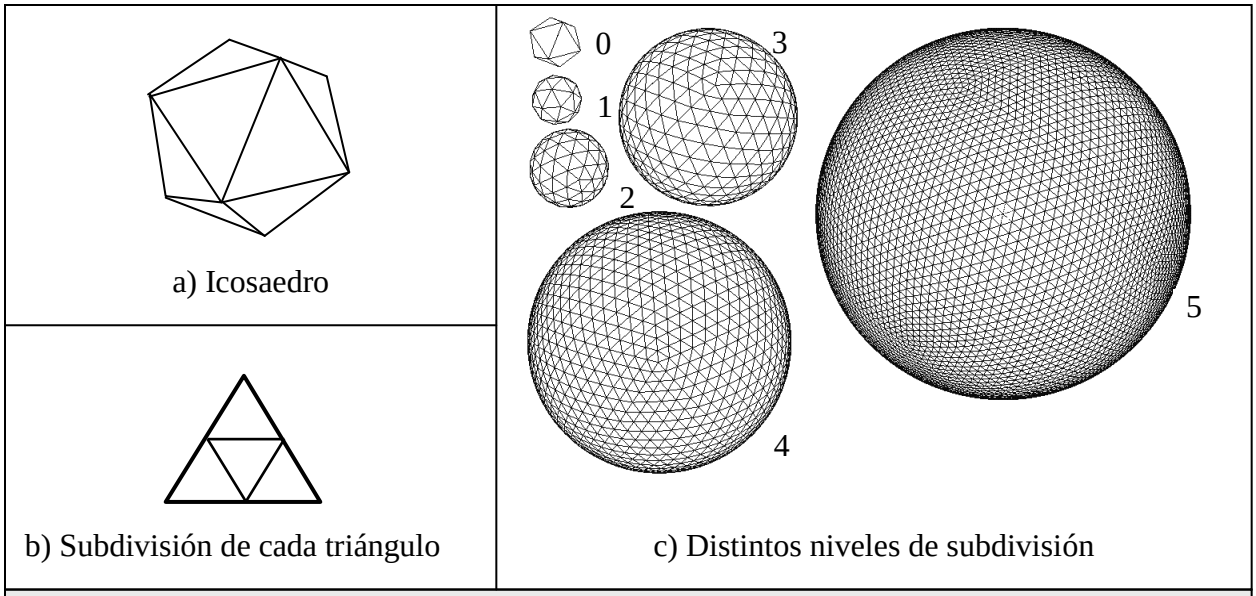
Una optimización importante se pude realizar cuando la proyección es paralela y la luz es direccional (está en el infinito) con el modelo Blinn-Phong. En este caso, se puede precalcular el modelo de iluminación para un subconjunto de vectores normales, distribuidas pseudo uniformemente entre todas las direcciones posibles. El método general es el siguiente:

- Escoger un subconjunto de n vectores unitarios $\{N_1,\ldots, N_n\}$ distribuidos en forma pseudo uniforme en el espacio 3D.
- Asignar un entero único u a cada vector N en el conjunto (su índice).
- Para cada vector normal N en el volumen, encontrar el vector unitario u más cercano al mismo en el conjunto. Así, la normal del vóxel estará representado por el entero u.
- El modelo de iluminación de Blinn-Phong se simplifica a $LaKa+K*(Ld*<N,L> + Ls* <N,H>^s)$, en donde $K=Kd=Ks$, y $LaKa=La*Ka$. Para cada Nu del conjunto de normales cuantizados pre-calcular $Table[u]=(Ld*<Nu,L> + Ls* <Nu,H>^s)$.
- Durante el *rendering*, para un vóxel con normal aproximada por Nu, simplemente se calcula su intensidad como $I(p)= LaKa+K*Table[u]$, donde K es el color RGB resultante de aplicar la función de transferencia en el vóxel.

La eficiencia del algoritmo de *rendering* de volúmenes con luz externa mejora notablemente al considerar la luz en el infinito, y proyección paralela, puesto que el cálculo de productos escalares y potencia necesarios en la iluminación es realizado solo para los n vectores cuantizados (en vez de realizarlo para cada vóxel).

Lacroute [LAC95], realiza la cuantización de vectores mediante una discretización pseudo uniforme de un octaedro, con el fin de obtener una fácil asociación entre una normal y su índice respectivo. En algunas zonas del octaedro 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. En nuestro caso, como la cuantización de vectores se realiza una sola vez (al cargar los datos), se plantea mejorar la uniformidad de la cuantización mediante la subdivisión de un icosaedro (ver Fig. 7.1a). Cada uno de sus 20 triángulos es dividido en cuatro triángulos equiláteros (ver Fig. 7.1b). Cada vértice nuevo es normalizado, y el proceso vuelve a repetirse recursivamente sobre todos los triángulos. En la Fig. 7.1c se muestran distintos niveles de subdivisión. Para asociarle un índice un vector normal N arbitrario, simplemente buscamos el triángulo del icosaedro por donde pasa ese vector N. El triángulo se subdivide y determinamos recursivamente en cuál triángulo va quedando el vector.

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 1 se tienen 80 vectores, y para el nivel 3, 1280 vectores. En la Fig. 7.2 se muestra una imagen generada con distintos niveles de subdivisión del icosaedro en la cuantización de vectores. En nuestras pruebas, del nivel de subdivisión 4 en adelante la diferencia visual es imperceptible.



a) Icosaedro

b) Subdivisión de cada triángulo

c) Distintos niveles de subdivisión

Figura 7.1: cuantización de normales vía subdivisión de un icosaedro.

Figura 7.2: imágenes obtenidas utilizando distintos niveles de subdivisión (0,2 y 4) del icosaedro en la cuantización de vectores.

8 PRE-INTEGRACIÓN

Acorde al teorema de muestreo, una reconstrucción correcta de un campo escalar se logra si esta es muestreada a una tasa superior o igual a la frecuencia de Nyquist. Sin embargo, utilizando post-clasificación, el campo escalar es muestreado antes de aplicar la función de transferencia, función que puede requerir una frecuencia de muestreo superior para capturar todos los detalles [LUM04].

Considere una función de transferencia definida por un pulso delgado como se muestra en la Fig. 8.1. Si el espaciado entre las muestras del rayo es superior a la longitud de este pulso, algunas muestras interpoladas pueden capturar este detalle, mientras otras no, resultando en una imagen compuesta de bandas y puntos, y no una superficie continua (ver Fig. 8.1a). Estos artefactos visuales

pueden ser reducidos si se aumenta el muestreo a una tasa muy alta (i.e. reducir la distancia entre muestras). Sin embargo, esto limita el rendimiento del sistema.

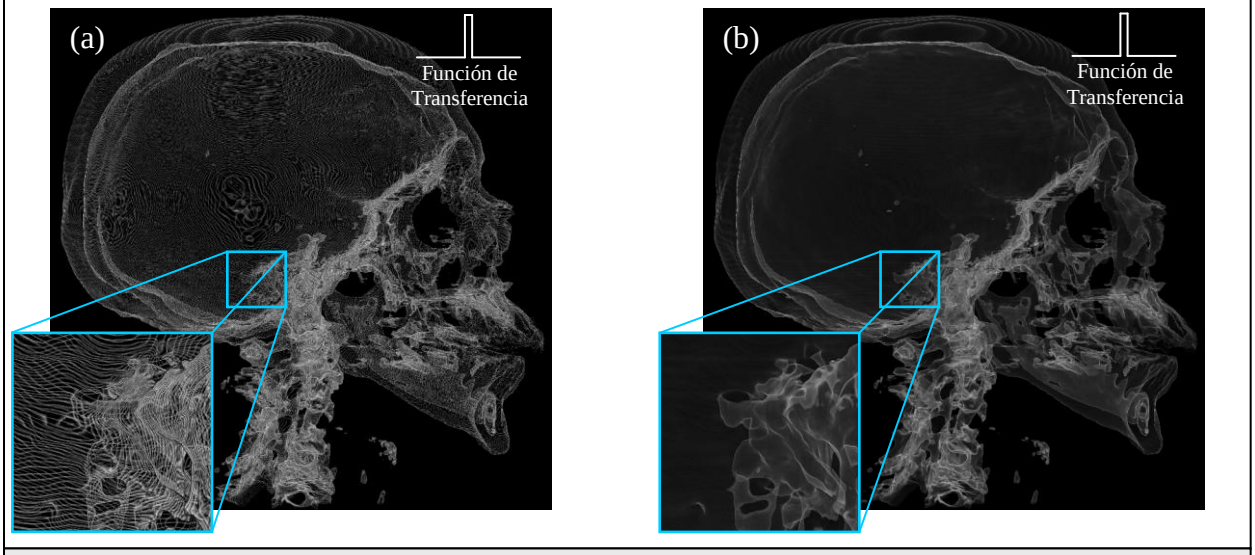


Figura 8.1: despliegue de una superficie no poligonal. Se define una función de transferencia en forma de pulso delgado, para $c(s)$ y $\tau(s)$ que define la superficie a visualizar. La imagen (a) ha sido generada utilizando post-clasificación, y la imagen (b) pre-integración.

La pre-integración tiene sus orígenes en el *rendering* de volúmenes y superficies sin realizar reconstrucción poligonal, a partir de mallas tetraédricas [ROE00]. Seguidamente fue utilizado para visualizar volúmenes representados en mallas regulares o *grid* [ENG01]. Para el caso que nos ocupa, mallas regulares, el objetivo de la pre-integración es ofrecer la solución numérica de la integral de *rendering* de volúmenes en cada segmento $[ih,(i+1)h]$ del rayo de la Ec. 2.14, y específicamente el color y opacidad dados por:

$$c_i = \int_{ih}^{(i+1)h} c(s(x(\lambda)))\tau(s(x(\lambda)))e^{-\int_{ih}^{\lambda}\tau(s(x(\lambda')))d\lambda'}d\lambda \qquad \text{[Ec. 8.1]}$$

$$\alpha_i = 1 - e^{-\int_{ih}^{(i+1)h}\tau(s(x(\lambda)))d\lambda}.$$

Para ello, se cuantizan los valores de s en n valores en [0,1], separados por una distancia $\Delta=1/n$. El valor de n determina la resolución de la tabla. Típicamente, se considera n=256 o n=4096, pero puede aumentar o disminuir según la calidad deseada, o la profundidad de las muestras del volumen. La tabla de integrales se utiliza para almacenar las integrales precalculadas entre cada par de muestras potenciales (s_f,s_b) que delimitan un segmento (ver Fig. 7.2). De esta manera, al realizar el *ray casting*, por cada par de muestras consecutivas (s_f,s_b) se acceden a las integrales pre-calculadas

en la entrada correspondiente en la tabla, y se componen con las integrales de los segmentos siguientes, utilizando el operador "*over*".

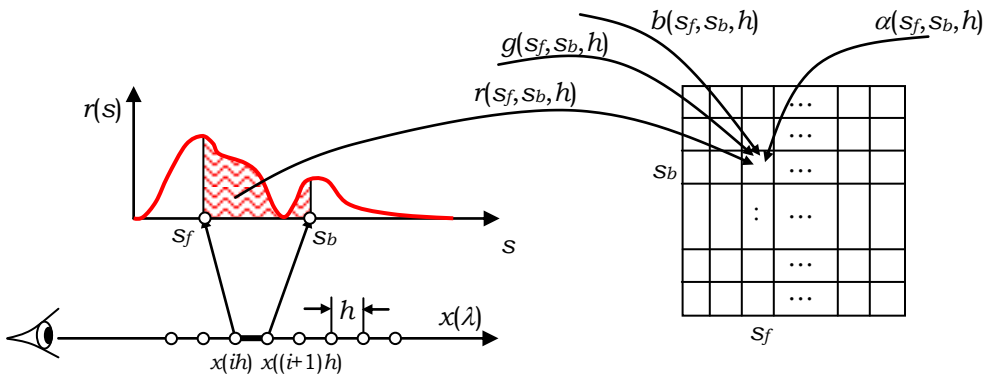


Figura 8.2: construcción de la tabla 2D de integrales. Se precalcula la integral entre pares de muestras (s_f, s_b), por cada canal de color $r(s)$, $g(s)$ y $b(s)$, y opacidad $\alpha(s)$. En la gráfica se muestra el segmento considerado para el canal $r(s)$, donde $s_f=s(x(ih))$ y $s_b=s(x((i+1)h))$.

Sea $s_f = s(ih)$, $s_b = s((i+1)h)$. La opacidad entre las muestras s_f y s_b puede escribirse como

$$\alpha_i = 1 - e^{-\int_0^h \tau(s_f + \lambda\frac{s_b - s_f}{h})d\lambda}. \qquad \text{[Ec. 8.2]}$$

Haciendo el cambio de variable, $\beta=s_f+\lambda(s_b-s_f)/h$, obtenemos

$$\alpha_i = \alpha(s_f,s_b,h) = 1 - e^{-\frac{h}{s_b - s_f}\int_{s_f}^{s_b}\tau(\beta)d\beta}. \qquad \text{[Ec.8.3]}$$

Haciendo la sustitución para el color, se obtiene

$$c_i = \int_0^h c\left(s_f + \frac{\lambda}{h}(s_b - s_f)\right)\tau\left(s_f + \frac{\lambda}{h}(s_b - s_f)\right)e^{-\int_0^\lambda \tau\left(s_f + \frac{\lambda'}{\lambda}(s_\lambda - s_f)\right)d\lambda'}d\lambda. \qquad \text{[Ec. 8.4]}$$

Por triángulos semejantes:

$$\frac{1}{\lambda}(s_\lambda - s_f) = \frac{1}{h}(s_b - s_f)$$

$$\Rightarrow c_i = \int_0^h c\left(s_f + \frac{\lambda}{h}(s_b - s_f)\right)\tau\left(s_f + \frac{\lambda}{h}(s_b - s_f)\right) e^{-\int_0^{\lambda} \tau\left(s_f + \frac{\lambda'}{h}(s_b - s_f)\right) d\lambda'} d\lambda . \quad \text{[Ec. 8.5]}$$

Haciendo el cambio de variable $\beta = s_f + \frac{\lambda}{h}(s_b - s_f)$ obtenemos

$$c_i = \frac{h}{s_b - s_f} \int_{s_f}^{s_b} c(\beta)\tau(\beta) e^{-\int_{sf}^{\frac{\beta - s_f}{s_b - s_f} h} \tau\left(s_f + \frac{\lambda'}{h}(s_b - s_f)\right) d\lambda'} d\beta . \quad \text{[Ec. 8.6]}$$

Haciendo un cambio de variable $\beta' = s_f + \frac{\lambda'}{h}(s_b - s_f)$ en la integral interna, obtenemos finalmente:

$$c_i = c(s_f, s_b, h) = \frac{h}{s_b - s_f} \int_{s_f}^{s_b} c(\beta)\tau(\beta) e^{-\frac{h}{s_b - s_f}\int_{sf}^{\beta} \tau(\beta) d\beta'} d\beta . \quad \text{[Ec. 8.7]}$$

La integral de color $c(s_f, s_b, h)$ no puede ser resuelta analíticamente, dado que la integral interna resulta en términos cuadráticos para una función de transferencia lineal a trozos, i.e. $\exp(-\lambda^2)$ no tiene antiderivada. Sin embargo, puede ser resuelta numéricamente en tiempo lineal, utilizando por ejemplo el método de *Simpson Composite* [BUR00]. Suponiendo que *s* ha sido cuantizada en *n* valores, y que *h* es constante, la complejidad para calcular esta tabla es $O(n^3)$. Esto es debido a que cada una de las n^2 integrales requiere en el peor de los casos integrar sobre los *n* sub intervalos de longitud $1/n$ de la función de transferencia, limitando la edición de la función de transferencia en tiempo real.

Diversos trabajos se han realizado para mejorar el tiempo de respuesta en la actualización de esta tabla. Engels et al. [ENG01] reducen efectivamente el número de integrales a calcular, al eliminar auto-extinción del color en cada segmento del rayo. Así, las integrales a pre calcular se reducen a:

$$c_i = \int_{ih}^{(i+1)h} c(s(x(\lambda)))\tau(s(x(\lambda)))d\lambda , \quad \text{[Ec. 8.8]}$$

$$\alpha_i = 1 - e^{-\int_{ih}^{(i+1)h} \tau(s(x(\lambda)))d\lambda} .$$

Luego de hacer una simplificación de estas ecuaciones, se obtiene:

$$c_i = c(s_f, s_b, h) = \frac{h}{s_b - s_f} \int_{s_f}^{s_b} c(\beta)\tau(\beta) d\beta ,$$

$$\alpha_i = \alpha(s_f, s_b, h) = 1 - e^{-\frac{h}{s_b - s_f}\int_{s_f}^{s_b}\tau(\beta)d\beta}. \qquad \text{[Ec. 8.9]}$$

Basándose en la siguiente observación,

$$\int_{s_f}^{s_b} c(\beta)\tau(\beta)d\beta = \int_{0}^{s_b} c(\beta)\tau(\beta)d\beta - \int_{0}^{s_f} c(\beta)\tau(\beta)d\beta,$$

$$\int_{s_f}^{s_b} \tau(\beta)d\beta = \int_{0}^{s_b} \tau(\beta)d\beta - \int_{0}^{s_f} \tau(\beta)d\beta, \qquad \text{[Ec. 8.10]}$$

basta calcular únicamente las integrales en (0,*s*) para los valores cuantizados de *s*, haciendo que el número de integrales a computar sea O(*n*).

$$C(0, s, h) = C(s) = \int_{0}^{s} c(\beta)\tau(\beta)d\beta, \qquad \text{[Ec. 8.11]}$$

$$T(0, s, h) = T(s) = \int_{0}^{s} \tau(\beta)d\beta.$$

Calcular las *n* posibles integrales para *C*(*s*) y *T*(*s*) puede hacerse incrementalmente. Considerando los valores cuantizados de *s*

$$s \in \{0, 1/n, 2/n, \ldots k/n, \ldots, n/n\},$$

los colores *C*(*s*) pueden calcularse incrementalmente como

$$C(0,k/n,h)=C(0,(k-1)/n,h) + C((k-1)/n, k/n,h). \qquad \text{[Ec. 8.12]}$$

Asumiendo que el soporte de los trozos de la función de transferencia son al menos de longitud 1/*n*, *C*((*k*–1)/*n*, *k*/*n*,*h*) puede calcularse en un tiempo constante. El mismo análisis puede hacerse para el cálculo de *T*(*s*).

Una vez pre-calculado todas las integrales *C*(*s*) y *T*(*s*) en tiempo lineal, el cálculo de color y opacidad para cada una de las n^2 entradas en la tabla se obtienen mediante:

$$c(s_f,s_b,h) = \frac{h}{s_b - s_f}\left[C(s_b) - C(s_f)\right],$$

$$\alpha(s_f,s_b,h) = 1 - e^{-\frac{h}{s_b - s_f}\left[T(s_b) - T(s_f)\right]}. \qquad \text{[Ec. 8.13]}$$

Por lo tanto, la actualización de la tabla es O(n^2). Trabajos siguientes muestran que la eliminación de la auto-extinción en cada segmento de rayo genera artefactos para funciones de transferencias que

revelan múltiples superficies a la vez [ROE02], [LUM04]. Por ejemplo, si varias superficies intersectan el mismo segmento de rayo, la auto-extinción dentro del segmento juega un papel importante en la determinación del color. Así, la investigación se ha basado desde entonces en mejorar el tiempo de respuesta sin la eliminación de la auto-extinción.

Eric Lum et al. [LUM04] proponen un algoritmo eficiente, considerando la auto-extinción por cada segmento de rayo y *h* constante. Se basan en la observación inicial que por cada diagonal de la tabla de integrales, se repite el cálculo de pequeñas integrales. Calculando estas pequeñas integrales una sola vez por diagonal, y combinándolas adecuadamente, se logra reducir la complejidad de $O(n^3)$ a $O(n^2)$. La adaptación de esta técnica para tablas 3D (que consideran variable el paso *h*) puede encontrarse en [CAR10], [CAR11].

9 CONCLUSIONES Y TRABAJOS A FUTURO

En este trabajo se ha presentado en detalle la teoría necesaria para comprender e implementar la técnica de *rendering* directo de volúmenes. El modelo óptimo utilizado surge de una ecuación diferencial que resulta en una integral, la cual puede ser evaluada numéricamente por píxel durante los distintos algoritmos de *rendering*. Entre ellos de describen el *ray casting*, planos alineados, conchas esféricas y *Shear-Warp*. En cada uno de estos algoritmos se han presentado detalles de implementación, incluyendo la utilización del hardware gráfico.

Cuando se desea visualizar superficies sin la necesidad de la reconstrucción de iso-superficies intermedias, consideramos la posibilidad de agregar una luz externa al modelo, de manera de ver reflexión especular sobre dichas superficies. Se abordó también la clasificación pre-integrada, la cual mejora la calidad del *rendering*, y debe tener poco impacto en el tiempo de respuesta.

Gracias al poder de cómputo de los GPUs, la visualización de volúmenes ha evolucionado en la última década, incorporando aspectos de iluminación global como sombras y *scattering* durante el *rendering*. Por consiguiente, se desea realizar próximamente un estudio del estado del arte en iluminación global para *rendering* de volúmenes, y su implementación en GPU.

9 REFERENCIAS

[AKE93] Akeley Kurt. "*Reality Engine Graphics*". Computer Graphics, Vol. 27, pp. 109-116, 1993.

[BOA01] Boada Imma; Navazo Isabel y Sopigno Roberto. "*A 3D texture-based octree volume visualization algorithm*". The Visual Computer, Vol. 17, pp. 185–197, 2001.

[BEN05] Benassarou A., Bittar E., John N. W. y Lucas L. "*MC Slicing for Volume Rendering Applications*". Lecture Notes in Computer Science, Vol. 3515, pp. 314–321. Enero 2005.

[BLI82] Blinn James. "*Light reflection functions for simulation of clouds and dusty surfaces*". Computer Graphics (ACM SIGGRAPH 82), Vol. 16, pp. 21–29, Julio 1982.

[BUR00] Burden Richard y Faires Douglas. "*Numerical Analysis*". Brooks/Cole, 7ma. edición, ISBN 0-534-38216-9, 2000.

[CAB94] B. Cabral, N. Cam and J. Foran, "*Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware*". En Proc. Symposium on Volume Visualization '94, pp. 91-98, 1994.

[CAR00] Carmona Rhadamés. "*Shear-Warp: Una implementación Eficiente*". En Proc. Clei2000: XXVI Conferencia Latinoamericana de Informática, Ciudad de México – México, Septiembre 2000.

[CAR10] Carmona Rhadamés. "*Visualización Multi-Resolución de Volúmenes de Gran Tamaño*". Tesis Doctoral. Universidad Central de Venezuela, Junio 2010.

[CAR11] Carmona Rhadamés, Froehlich Bernd. "*Error-controlled real-time cut updates for multi-resolution volume rendering*". Computer & Graphics, 35(4): 931-944. Agosto 2011.

[CHA60] Chandrasekhar Subrahmanyan. "*Radiative Transfer*". Dover, New York, 1960.

[CLI87] Cline Harvey y Lorensen Willian. "*Marching Cubes: A high resolution 3D surface construction algorithm*". Computer Graphics, Vol. 21 – No. 4, Julio 1987.

[DAN92] Danskin John y Narran Pat. "*Fast Algorithms for Volume Ray Tracing*". En Proc. Workshop on Volume Visualization '02, pp. 91–98, Boston-Massachussets-USA, 1992.

[DRE88] Drebin Robert, Carpenter Loren y Hanrahan Pat. "*Volume rendering*". Computer Graphics (ACM SIGGRAPH 88), Vol. 22, pp. 65–74, Agosto 1988.

[ENG01] Engel Klaus, Kraus Martin y Ertl Thomas. "*High Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading*". En Proc. Siggraph/Eurographics Workshop on Graphics *Hardware*, pp. 9–16, California–USA, 2001.

[FAR01] Farin Gerald. "*Curves and Surfaces for CAGD*". Academic Press, Morgan Kaufmann Publishers, 5ta. edición, Octubre 2001.

[FOL90] Foley James, van Dam Andries, Feiner Seteven y Hughes John. "*Computer Graphics: Principles and Practice*". Addison-Wesley, 2da. edición, 1990.

[GUA94] Guan Shen-Yih y Lipes Richard. "*Innovative volume rendering using 3D texture mapping*". SPIE Medical Imaging – Image Capture, formatting and Display, Vol. 2164, pp. 382–392, 1994.

[HAN74] Hansen James y Travis Larry. "*Light Scattering in Planetary Atmospheres*". Space Science Reviews 16, pp. 527–610, 1974.

[KAJ84] Kajiya James y Von Herzen Brian. "*Ray tracing volume densities*". Computer Graphics (ACM SIGGRAPH '84), Vol. 18 – No. 3, pp. 165–174, Julio 1984.

[KRU90] Krueger Wolfang. "*The application of Transport Theory to Visualization of 3D Scalar Data Fields*". En Proc. IEEE Visualization '90, pp. 272–280, Octubre 1990.

[KRU03] Krüger J. y Westermann R. "*Acceleration techniques for GPU-base Voume Rendering*". En Proc. Visualization '03, pp. 287–292, Washington-USA, 2003.

[LAC95] Lacroute Phillippe G. "*Fast Volume Rendering Using Shear-Warp Factorization of the Viewing Transformation*". Reporte Técnico CSL-TR-95-678, Universidad de Standford, USA, 1995.

[LAM99] LaMar Ericm Hamann Bernd y Joy Kenneth I. "*Multiresolution Techniques for Interactive Texture-based Volume Visualization*". En Proc. Visualization '99, pp. 355–361, California-USA, 1999.

[LEV90] Levoy M. "*Efficient Ray Tracing of Volume Data*". ACM Transactions on Graphics, Vol. 9 – No. 3, pp. 245–261, 1990.

[LUM04] Lum Eric B., Wilson Brett y Ma Kwan-Liu. "*High-Quality Lighting and Efficient Pre-Integration for Volume Rendering*". En Proc. Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization, pp. 25–34, 2004.

[MAD02] Madero, A. "*Visualización Volumétrica Estereoscópica en Tiempo Real*". Proyecto de Grado, Biblioteca Alonso Gamero, Universidad Central de Venezuela, 2001.

[MAX86] Max Nelson. "*Light diffusion through clouds and haze*". Computer Vision, Graphics, and Image Processing, Vol. 33, pp. 280–292, 1986.

[MAX95] Max Nelson. "*Optical Models for Direct Volume Rendering*". Visualization in Scientific Computing, Springer, pp. 35–40, 1995.

[MOR04] Moreland Kenneth. "*Fast High Accuracy Volume Rendering"*. Tesis Doctoral, Universidad de New Mexico, Julio 2004.

[PER08] "*Perspective Correction"*. http://en.wikipedia.org/wiki/Texture_mapping#Perspective_correctness. Ultimo acceso 15/02/2015, Enero 2008.

[PLA02] Plate John, Tirsana Michael, Carmona Rhadamés y Froehlich Bernd. "*Octreemizer: A Hierarchical Approach for Interactive Roaming Through Very Large Volumes*". En Proc. Joint Eurographics - IEEE TCVG Symposium on Visualization '02, pp. 53–60, 2002.

[REZ00] Rezk-Salama C., Engel K., Bauer M., Greiner G. y Ertl T. "*Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stages Rasterization*". En Proc. Eurographics / SIGGRAPH Workshop on Graphics *Hardware*, pp. 109–119, 2000.

[ROE00] Roettger S., Kraus M. y Erlt T. "*Hardware Accelerated Volume and Isosurface Rendering Based on Cell Projection*". En Proc. IEEE Visualization '00, pp. 109–116, 2000.

[ROE02] Roettger Stefan y Ertl Thomas. "*A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids*". En Proc. VolVis '02, ACM Press, pp. 23–28, 2002.

[RUI06] Ruijters D. y Vilanova A. "*Optimizing GPU Volume Rendering*". Journal of *WSCG - Winter School of Computer Graphics*, Vol. 14, pp. 9–16, 2006.

[SAB88] Sabella Paolo. "*A rendering algorithm for visualizing 3D scalar fields*". En. Proc. ACM SIGGRAPH '88, Computer Graphics, Vol. 22, pp. 51–58, Agosto 1988.

[SCH03] Schulze J. P., Kraus M., Lang U. y Ertl T. "*Integrating Pre-Integration into the Shear-Warp Algorithm*". En Proc. Eurographics/IEEE TVCG Workshop on Volume Graphics, pp. 109–118, Tokio-Japón, 2003.

[WIL92] Williams Peter y Max Nelson. "*A volume density optical model*". En Proc. Workshop on Volume Visualization '92, Computer Graphics, pp. 61–68, Octubre 1992.

[WIL94] Wilson O., Van Gelder A. y Wilhelms J. "*Direct Volume Rendering via 3D Textures*". Reporte técnico UCSC-CRL-94-19. Universidad de California, Santa Cruz, 1994.