Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 7053-7066 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

IMPROVING THE LOW RANK RADIOSITY METHOD USING SPARSE MATRICES

Eduardo Fernández, Pablo Ezzatti y Sergio Nesmachnow

Centro de Cálculo, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, http://www.fing.edu.uy/inco/grupos/cecal, {eduardof,pezzatti, sergion}@fing.edu.uy

Palabras Clave: Radiosidad de rango bajo, tiempo real, matrices dispersas, GPU.

Resumen. El problema de radiosidad consiste en el cálculo de la distribución de la radiación lumínica en escenas compuestas por superficies con reflexión lambertiana. Su modelación matemática se realiza a través de la ecuación de radiosidad, ecuación integral de Fredholm de segunda especie. Para la resolución de la ecuación de radiosidad se utiliza el método de elementos finitos, a través del cual la ecuación de radiosidad se aproxima por un sistema lineal con una matriz F densa, de dimensión cuadrática en la cantidad de elementos considerados en la discretización de la escena. Para obtener soluciones visualmente realistas es necesario que las superficies de la escena contengan al menos decenas de miles de elementos. Con el propósito de resolver el sistema lineal en tiempo real contemplando las limitaciones de memoria y capacidad de cómputo del hardware actual, en un trabajo anterior se propuso aproximar la matriz F por un producto matricial de rango bajo.

En este artículo se plantea la construcción de una nueva propuesta de aproximación de rango bajo utilizando una matriz dispersa con un único elemento distinto de cero por fila. El ahorro de memoria inherente al uso de matrices dispersas posibilita, para un hardware determinado, desarrollar discretizaciones con mayor cantidad de elementos y la construcción de aproximaciones a F de mayor rango que los permitidos con las aproximaciones que utilizan matrices densas. A su vez, explotar la estructura particular de la matriz dispersa permite acelerar notoriamente los cálculos, posibilitando el procesamiento de imágenes más complejas.

Para evaluar el uso de matrices dispersas en la resolución del problema de radiosidad con geometría fija en tiempo real, en este trabajo se presentan experimentos donde se compara para diversas dimensiones y rangos el uso de memoria y los tiempos de ejecución de la etapa de tiempo real cuando la matriz de rango bajo es densa y dispersa. El análisis experimental se realiza sobre la arquitectura tradicional basada en CPU y sobre la moderna alternativa que utiliza unidades de procesamiento gráfico (GPU). Los resultados experimentales obtenidos permiten concluir que el uso de matrices dispersas mejora la eficiencia en comparación al uso de matrices densas y posibilita el cálculo de radiosidad en tiempo real para escenas discretizadas en millones de elementos, así como el desarrollo de metodologías híbridas, donde algunas operaciones se realicen en CPU y otras en GPU.

1. INTRODUCCIÓN

El método de radiosidad es una de las técnicas pioneras de iluminación global, que surgió como una variante de las técnicas empleadas para el estudio de la transferencia de calor a través de la radiación térmica. En el método de radiosidad se propone una formulación general de la iluminación global de la escena, donde las superficies poseen reflexión difusa lambertiana y se excluyen aquellas superficies que tienen componentes de reflexión especular. Esta restricción no ha invalidado su uso en diversas áreas del diseño y la animación (Dutre et al., 2006).

A partir de la formulación clásica del método de radiosidad en la década de 1980, se han implementado diversas variantes basadas en la resolución de sistemas lineales por métodos iterativos. Los trabajos de Cohen et al. (1993) son excelentes referencias de los métodos surgidos durante la primer década de desarrollo. Los métodos allí mostrados tienen los inconvenientes de ser iterativos y de procesar grandes volúmenes de información, requiriendo un tiempo de cómputo del orden de minutos en la generación de imágenes.

Como consecuencia del diseño de tarjetas gráficas con capacidades de cómputo superiores a las CPU, comenzaron a desarrollarse métodos alternativos con el objetivo de calcular la iluminación global de una escena en tiempo real. Entre los métodos más conocidos se encuentran *Instant Radiosity* (Keller, 1997), el precómputo de la transferencia de radiancia (*precomputed radiance transfer*) (Sloan et al., 2002), y recientemente el trabajo de Wang et al. (2009) sobre la implementación de la técnica de mapas de fotones (*photon mapping*) en GPU. Una reseña de estos métodos y otras técnicas relacionadas puede encontrarse en la tesis de Fernández (2010).

Recientemente se presentó la novedosa técnica de **radiosidad de rango bajo** (RRB), que permite abordar en tiempo real y utilizando cantidades relativamente bajas de memoria el problema de radiosidad en escenas con cientos de miles de elementos (Fernández, 2009).

La técnica de radiosidad de rango bajo se basa en las ideas de reducción de modelos, que en su implementación original utiliza matrices densas. En el trabajo previo (Fernández et al., 2009) se explicita que los métodos desarrollados pueden generar matrices dispersas, pero no se implementa un código que saque provecho de esa situación. La utilización de matrices dispersas ha permitido la implementación de algoritmos más eficientes en diversas áreas de aplicación, debido a la reducción del volumen de memoria utilizado y la aceleración o *speedup* en la realización de algunas operaciones algebraicas.

Basados en los argumentos expuestos anteriormente, en este trabajo se explora la utilización de matrices dispersas en RRB, presentando algoritmos que resuelven el problema de radiosidad en CPU y en GPU. El resto del artículo se estructura de la manera que se describe a continuación. En la siguiente sección se plantea el problema de radiosidad en su versión continua y discreta. Luego se desarrolla la técnica RRB, que incluye el cálculo de la aproximación de rango bajo a la principal matriz del sistema. En la sección 4 se introduce el uso de matrices dispersas en RRB. La sección 5 expone el contexto del análisis experimental realizado y se presentan y discuten los principales resultados obtenidos. Finalmente, la última sección resume las principales conclusiones del trabajo y las posibles líneas para trabajo futuro.

2. EL PROBLEMA DE RADIOSIDAD

Formulado como un problema matemático, el problema de radiosidad para una escena implica hallar la función solución de una ecuación integral de Fredholm de segunda especie, denominada ecuación de radiosidad (Ecuación 1).

$$B(x) = E(x) + \rho(x) \int_S B(x')G(x,x')dA' \tag{1}$$

En la ecuación de radiosidad, la función $B(x)$ es la radiosidad del punto x, $E(x)$ es la emisión en x y $\rho(x)$ es la reflectividad difusa de x. La función $G(x,x')$ mide la intensidad luminosa radiada entre dos puntos x y x', de acuerdo a la Ecuación 2, donde la función $V(x,x')$ vale uno si los puntos x y x' son mutuamente visibles y vale cero si no lo son. El resto de la expresión en la Ecuación 2 está en función de los ángulos entre la recta que une los puntos (x y x'), de las normales de las superficies a los que los puntos pertenecen, y de la distancia existente entre los puntos.

$$G(x,x') = G(x',x) = V(x,x')\frac{\cos\theta' \cos\theta}{\|x-x'\|^2} \tag{2}$$

Una ecuación de Fredholm puede ser transformada en una ecuación con operadores lineales, que en su versión discreta es un sistema de ecuaciones lineales. Con estas transformaciones, el cálculo de la radiosidad de una escena implica la resolución de un sistema lineal.

Al discretizar la ecuación de radiosidad, se obtiene la ecuación matricial de radiosidad (Ecuación 3), donde $\mathbf{I}$ es la matriz identidad de dimensión $n \times n$ (siendo n la cantidad de elementos, parches o polígonos en que se divide la escena), $\mathbf{R}$ es una matriz diagonal que en la celda (i,i) tiene el índice de reflectividad ρ_i correspondiente al parche i, $\mathbf{F}$ es una matriz con los factores de forma $\mathbf{F}_{ij}$ (que indican la fracción de la energía lumínica reflejada o emitida por el parche i y que llega al parche j), B es un vector con los valores de radiosidad de cada parche y E es un vector que contiene la emisión de cada parche.

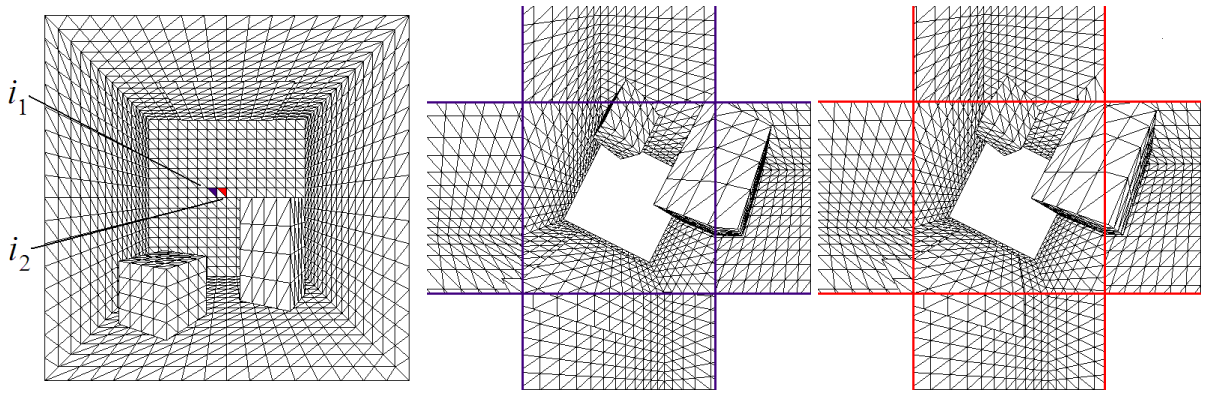
$$(\mathbf{I} - \mathbf{RF})B = E \tag{3}$$

3. RADIOSIDAD DE RANGO BAJO

Björck y Dalhquist (1999) establecen que si el *kernel* de una ecuación integral de Fredholm de segunda especie varía suavemente, entonces su discretización está mal condicionada y posee un rango numérico bajo. Para el caso de la ecuación de radiosidad el *kernel* es $\rho(x)G(x,x')$ y su discretización es la matriz $\mathbf{RF}$. La matriz $\mathbf{RF}$ tiene grandes posibilidades de tener un rango numérico bajo, debido a que cada fila i se calcula a partir de la vista de la escena desde el centro del parche i. En la Figura 1 se aprecia cómo dos parches cercanos poseen una vista muy similar del resto de la escena. Como a partir de cada vista se genera una fila de la matriz $\mathbf{RF}$, se deduce que en $\mathbf{RF}$ existirán muchos pares de filas parecidas y por lo tanto su rango numérico será bajo (Fernández, 2010).

Al poseer la matriz $\mathbf{RF}$ un rango numérico k bajo, es posible aproximarla mediante un producto de matrices $\mathbf{U}_k\mathbf{V}_k^T$ de dimensiones $n \times k$, con pérdida de poca información.

El producto $\mathbf{U}_k\mathbf{V}_k^T$ genera una matriz de dimensiones $n \times n$ y rango k, y la memoria total ocupada por ambas matrices tiene $O(nk)$. La memoria ocupada por ambas matrices es muy conveniente en comparación con la memoria ocupada por la matriz $\mathbf{RF}$ (cuyo volumen posee $O(n^2)$), especialmente cuando el valor de n es grande y $n \gg k$. Este ahorro implícito posibilita el almacenamiento en memoria temporal de información correspondiente a escenas con cientos de miles de elementos.



(a) Parches cercanos i_1 e i_2. (b) Vista desde i_1. (c) Vista desde i_2.

Figura 1: Dos parches cercanos tienen vistas de la escena similares y generan filas parecidas en **RF**.

Al sustituir **RF** por su aproximación de rango bajo en la ecuación matricial de radiosidad se obtiene la **ecuación de radiosidad de rango bajo** (RRB), presentada en la Ecuación 4), donde la incógnita ya no es B sino una aproximación $\widetilde{B}$.

$$\left(\mathbf{I} - \mathbf{U}_k \mathbf{V}_k^T\right) \widetilde{B} = E \tag{4}$$

La matriz $\left(\mathbf{I} - \mathbf{U}_k \mathbf{V}_k^T\right)$ es fácilmente invertible utilizando la fórmula de Sherman-Morrison-Woodbury (SMW). Luego de aplicar SMW en la ecuación RRB, se puede despejar $\widetilde{B}$ obteniéndose la Ecuación 5.

$$\widetilde{B} = E + \mathbf{U}_k \left(\left(\mathbf{I} - \mathbf{V}_k^T \mathbf{U}_k\right)^{-1} \left(\mathbf{V}_k^T E\right) \right) \tag{5}$$

El cálculo de $\widetilde{B}$ utilizando la Ecuación 5 tiene una complejidad $O(nk^2)$ y utiliza una memoria $O(nk)$. Cuando la geometría de la escena es estática y sólo varían las fuentes de luz (cuando sólo varía el término independiente E de las Ecuaciones 3 y 4, entonces se puede transformar la Ecuación 5 en la **ecuación RRB con geometría fija** (Ecuación 6). En esta nueva ecuación las operaciones que posibilitan el cálculo de $\mathbf{Y}_k$ se realizan sólo una vez. De esta forma la complejidad final del cálculo de $\widetilde{B}$ para escenas con geometría estática se reduce a $O(nk)$.

$$\widetilde{B} = E - \mathbf{Y}_k \left(\mathbf{V}_k^T E\right), \text{ con } \mathbf{Y}_k = -\mathbf{U}_k \left(\mathbf{I}_k - \mathbf{V}_k^T \mathbf{U}_k\right)^{-1} \tag{6}$$

Este último resultado resulta atractivo para intentar procesar en tiempo real el cálculo de la radiosidad de una escena. En un trabajo previo de los autores (Fernández et al., 2009) se utilizaron estas ideas para desarrollar implementaciones sobre CPU y GPU para el cálculo de radiosidad en escenas con geometría constante, logrando en todos los casos evaluados para GPU resultados superiores a las 20 imágenes por segundo.

3.1. Cálculo de $\mathbf{U}_k$ y $\mathbf{V}_k$

Para el cálculo de las matrices $\mathbf{U}_k$ y $\mathbf{V}_k$ se pueden emplear diversas técnicas de factorización, como por ejemplo la descomposición en valores singulares (SVD) (Golub y Loan, 1996), la factorización CUR (Goreinov et al., 1997; Goreinov y Tyrtyshnikov, 2001)), y las transformaciones discretas basadas en Fourier y en wavelets (Press et al., 2007). Entre estos métodos, SVD es el único que genera buenas aproximaciones a **RF** para valores bajos de k, con el inconveniente de que posee una complejidad $O(n^3)$. Los métodos CUR y los basados en Fourier

y wavelets poseen órdenes menores pero generan aproximaciones a $\mathbf{RF}$ de mayor error. Resultados comparativos al utilizar estas técnicas para el cálculo de aproximaciones a la matriz $\mathbf{RF}$ fueron presentados en el trabajo de Fernández y Nesmachnow (2010).

Para superar las debilidades de las técnicas de factorización mencionadas, Fernández (2009) presentó una técnica de factorización que luego se extendió para incluir el concepto de *coherencia espacial* (Sutherland et al., 1974). La nueva técnica se sintetiza en dos algoritmos, denominados *2MF* (dos Mallas Fijas) y *2MCE* (dos Mallas con Coherencia Espacial). Ambos algoritmos discretizan la escena en polígonos utilizando dos mallas de distinto nivel de granularidad (una *malla gruesa* con k parches y otra *malla fina* con n elementos), donde cada elemento de la malla fina está incluido (o se corresponde) con un único parche de la malla gruesa.

En el algoritmo 2MF se establece que ambas mallas están definidas previamente, siendo conveniente que sus elementos (o parches) posean tamaño uniforme. A partir de ellas se construyen las matrices $\mathbf{U}_k$ y $\mathbf{V}_k$ de dimensiones $n \times k$, con algoritmos de complejidad $O(nk)$ u $O(n^2)$, según la forma de generación empleada y el error de $\widetilde{B}$ buscado.

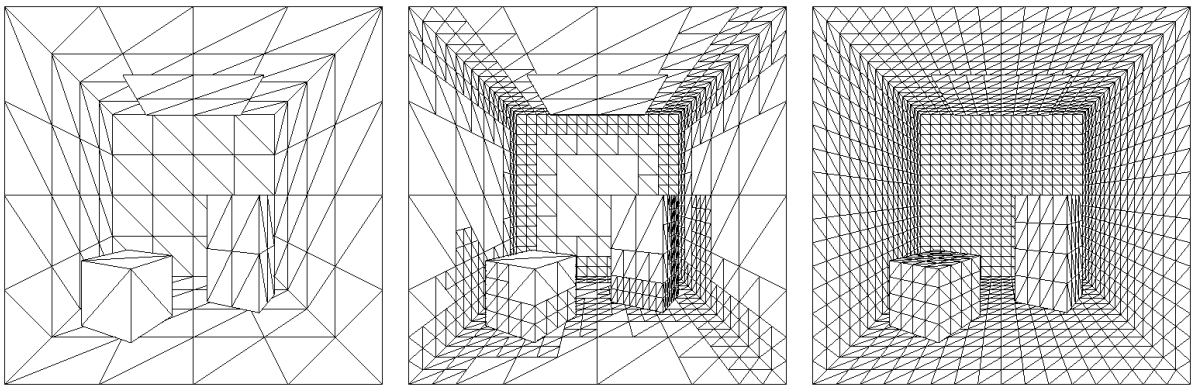
En el algoritmo 2MCE la malla fina está predeterminada (constituida por elementos de igual tamaño) y se construye la malla gruesa donde se busca que cada parche posea *coherencia espacial* interna, implicando que al interior de cada parche todas las vistas de la escena deben ser muy parecidas (véase la Figura 1). El algoritmo 2MCE genera una malla irregular, donde el tamaño de los parches es menor en aquellas áreas donde hay menor coherencia espacial, o donde las vistas de la escena presentan mayor variabilidad. El Algoritmo 1 muestra un esquema *top-down* para el cálculo de la malla gruesa que parte de una malla formada por pocos parches. Los parches se ingresan a una lista L, y se van sacando de a uno, evaluándose su coherencia espacial. Si la coherencia espacial del parche es mala (si la variabilidad de las vistas dentro del parche supera cierto umbral según alguna métrica), se divide el parche y los nuevos parches creados se ingresan a L. El algoritmo se detiene cuando la lista L se vacía, esto es, cuando los parches resultantes del algoritmo tienen coherencia espacial, o cuando aquellos que no tienen coherencia espacial poseen un área menor acierto valor prefijado. Conviene que L siga un esquema *FIFO* (*First In, First Out*), para que los nuevos parches ingresados no se evalúen (ni eventualmente se dividan) antes de haber evaluado el resto de los parches ya contenidos en L.

Algoritmo 1 Algoritmo 2MCE.

```
1 Cargar los parches p de una malla gruesa en una lista FIFO L.
2 Mientras (L≠∅) hacer
3   Quitar de L un parche p.
5   Si ¬(Coherencia Espacial en p) ∧ (Area(p) > Amin)
6     Divido p en subparches y los ingreso a L.
7   Fin.
8 Fin.
```

En el algoritmo 2MCE la cantidad de parches k no es un parámetro de entrada, sino que se determina indirectamente, ajustando los valores de $Amin$ y la forma de cálculo de la coherencia espacial. En la Figura 2 (b) se muestra una malla gruesa generada a partir del algoritmo 2MCE.

Luego de que se dispone de una malla gruesa y una malla fina, ya sea por definición (2MF) o construcción (2MCE), se pasa a la etapa de generación de las matrices $\mathbf{U}_k$ y $\mathbf{V}_k$. Como primer paso, se debe establecer la relación de pertenencia de cada elemento e de la malla fina en algún parche p de la malla gruesa. Para definir esta relación conviene que la superficie de cada elemento de la malla fina está incluido en la superficie de un único parche de malla gruesa. Si



(a) Malla gruesa (2MF). (b) Malla gruesa (2MCE). (c) Malla fina.

Figura 2: Dos versiones de mallas gruesas y una malla fina, para el cálculo de $\mathbf{U}_k$ y $\mathbf{V}_k$.

esto no es posible, ya sea por la metodología de construcción y división de los parches en 2MCE o por la forma en que se definieron las mallas utilizadas en 2MF, entonces una solución práctica es asignar a cada elemento el parche más cercano. Esta asignación se define como una función $P : e \rightarrow p$ que relaciona el espacio de índices de los elementos con el espacio de índices de los parches. La función $P(e)$ establece para cada elemento e de la malla fina, a qué parche p de la malla gruesa está asociado, ya sea por cercanía o por inclusión total de e en p.

Por último, se construyen las matrices $\mathbf{U}_k$ y $\mathbf{V}_k$ empleando las Ecuaciones 7, donde $\mathbf{F}_{ep}$ es el *factor de forma* entre el elemento e y el parche p y c_p es la cantidad de elementos asociados al parche p (se asume que los elementos contenidos en un parche poseen área uniforme). Los detalles del cálculo se pueden encontrar en el trabajo de Fernández (2010).

$$\mathbf{U}_k(e,p) = \mathbf{F}_{ep}/c_p \ , \ \mathbf{V}_k(e,P(e)) = 1 \ , \ \mathbf{V}_k(i,j) = 0 \forall j \neq P(i) \tag{7}$$

4. UTILIZACIÓN DE MATRICES DISPERAS EN RRB

La matrices $\mathbf{V}_k$ generadas con los métodos 2MCE y 2MF poseen como características comunes ser matrices dispersas de dimensiones $n \times k$, todos sus elementos distintos de cero valen uno y poseer un único elemento distinto de cero por fila. En las Ecuaciones 7 se observa que en cualquier fila e de $\mathbf{V}_k$ el único elemento distinto de cero es $P(e)$.

Las filas de $\mathbf{V}_k$ se pueden reordenar agrupándolas según la columna en la que esté el elemento distinto de cero, formando así k grupos disjuntos (véase la Figura 3). Con este agrupamiento y usando ideas del almacenamiento comprimido de matrices dispersas (por ejemplo, los formatos estándares CCS y CRS (Barrett et al., 1994)) se puede construir un vector de índices I y establecer que entre $I_1 = 1$ e $I_2 - 1$ se encuentren las filas de $\mathbf{V}_k$ con un uno en la primer columna, de I_2 a $I_3 - 1$ están las filas de de $\mathbf{V}_k$ con un uno en la segunda columna, y así sucesivamente hasta que de I_k a $I_{k+1} - 1 = n$ están las filas de $\mathbf{V}_k$ con un uno en la k-ésima columna. Por tanto, el vector de índices I tiene dimensión $k+1$ (ocupa memoria $O(k)$) y alcanza para definir la matriz $\mathbf{V}_k$. En el ejemplo de la Figura 3, la sucesión de índices en I están equiespaciados. Esto sucede cuando cada parche de la malla gruesa contiene una cantidad constante de elementos de la malla fina, lo que ocurre frecuentemente al utilizar el algoritmo 2MF, donde cada parche de la malla gruesa contiene una misma cantidad de elementos de la malla fina. Difícilmente este fenómeno ocurra al emplear el algoritmo 2MCE, dado que los parches poseen diferente tamaño y la malla fina ya fue generada.

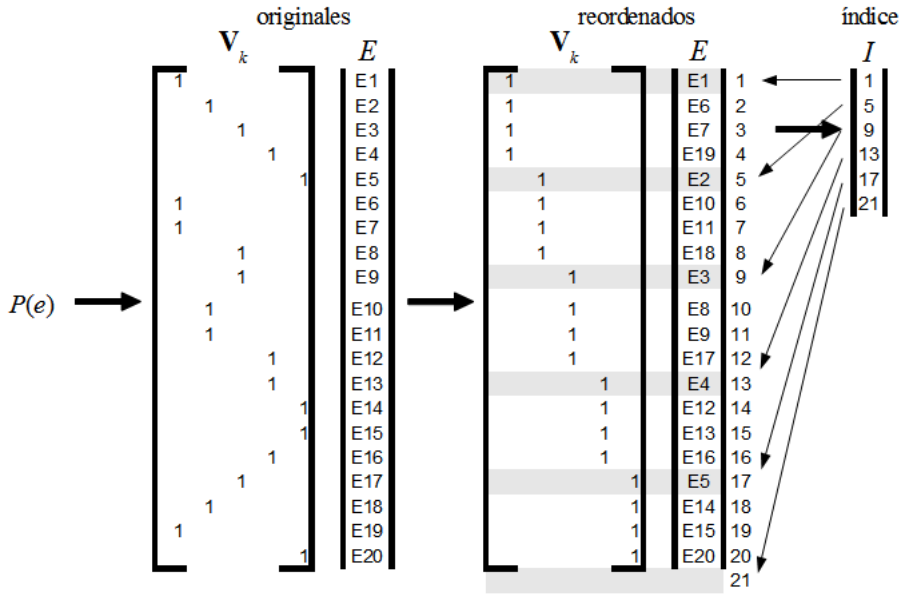


Figura 3: Construcción de $\mathbf{V}_k$ e I a partir de la función P.

Una vez construido el vector I y reordenados los elementos del vector E de igual manera que las filas de $\mathbf{V}_k$, se puede establecer que el producto $\mathbf{V}_k^T E$ es simplificable a un conjunto de sumatorias, según lo expresado en la Ecuación 8. El i-ésimo valor del vector producto es igual a la suma de los valores de E que se encuentran entre las posiciones I_i e $I_{i+1} - 1$.

$$\left(\mathbf{V}_k^T E\right)_i = \sum_{s=I_i}^{I_{i+1}-1} E_s, \forall i / 1 \leq i \leq k \tag{8}$$

Con este resultado se concluye que en el caso en que $\mathbf{V}_k$ sea una matriz dispersa con las características mencionadas, el producto matriz-vector $\mathbf{V}_k^T E$ tiene complejidad $O(n)$. En comparación, el mismo producto cuando la matriz $\mathbf{V}_k$ es densa tiene complejidad $O(nk)$.

4.1. Implementación de RRB dispersa

Para validar los resultados teóricos presentados anteriormente, se realizaron dos implementaciones sobre diferentes arquitecturas para la resolución de la Ecuación 6, donde $\mathbf{V}_k$ es una matriz dispersa que cumple con la definición de la Ecuación 7. En una implementación las operaciones matriciales se realizan en CPU y en la otra en GPU, utilizando la versión correspondiente de la función `XGEMV` de BLAS/CUBLAS para realizar el producto matriz-vector en cada caso. En ambas implementaciones el producto $\mathbf{V}_k^T E$ se realiza utilizando la sumatoria definida en la Ecuación 8.

La implementación en CPU utiliza la biblioteca BLAS (Lawson et al., 1979) y la estrategia de resolución de la Ecuación 8 se realiza en forma secuencial. El seudocódigo del algoritmo implementado se muestra en el Algoritmo 2.

Algoritmo 2 RRB dispersa en CPU.

1 Cargar la matriz $\mathbf{Y}_k$ y los vectores I y E en memoria RAM.
2 **Para cada** (j=1:$CantE$) **hacer**
3 Tomar el vector E_j.
4 Calcular X_i=$\left(\mathbf{V}_k^T E_i\right)_i, \forall i$ con $\sum_{s=I_i}^{I_{i+1}-1} E_j(s)$
5 Aplicar XGEMV para realizar $\widetilde{B}_j = E_j - \mathbf{Y}_k^T X$.
6 Almacenar el vector $\widetilde{B}_j$ en la memoria RAM.
7 **Fin.**

La implementación en GPU utiliza la versión de BLAS desarrollada sobre arquitectura CUDA denominada CUBLAS (NVIDIA, 2008). El seudocódigo del algoritmo implementado se muestra en el Algoritmo 3. Para la resolución de la Ecuación 8 se utilizan solamente las capacidades ofrecidas por CUDA, creando un conjunto de hilos que ejecutan en paralelo (tantos como el rango de la matriz reducida) para computar los valores del nuevo vector. Todos los hilos realizan un mismo proceso iterativo, calculando las sumatorias $\sum_{s=I_i}^{I_{i+1}-1} E_s$ con un valor diferente de i por hilo. Para esta implementación se realizó una etapa de calibración para encontrar la mejor configuración de la grilla de bloques de threads para resolver los distintos casos probados de productos matriz-vector.

Algoritmo 3 RRB dispersa en GPU.

1 Cargar la matriz $\mathbf{Y}_k$ y los vectores I y E en memoria RAM.
2 Transferir la matriz $\mathbf{Y}_k$ y el vector I a la memoria de la GPU.
3 **Para cada** (j=1:$CantE$) **hacer**
4 Tomar un vector E_j y enviarlo a la GPU.
5 Calcular en GPU X_i=$\left(\mathbf{V}_k^T E_i\right)_i, \forall i$ con $\sum_{s=I_i}^{I_{i+1}-1} E_j(s)$
6 Aplicar XGEMV en GPU para realizar $\widetilde{B}_j = E_j - \mathbf{Y}_k^T X$.
7 Enviar el vector $\widetilde{B}_j$ de la memoria de la GPU a la memoria RAM.
8 **Fin.**

Las dos únicas diferencias de estos algoritmos en comparación al implementado por Fernández et al. (2009) corresponden a las líneas donde se calcula X_i utilizando la Ecuación 8 y la operación de tranferencia de datos.

5. ANÁLISIS EXPERIMENTAL Y DISCUSIÓN DE RESULTADOS

Los resultados teóricos presentados (almacenamiento de $\mathbf{V}_k$ en memoria de $O(k)$ y cálculo del producto $\mathbf{V}_k^T E$ con complejidad $O(n)$), motivaron la realización de un estudio comparativo entre los valores de tiempo de cómputo de $\widetilde{B}$ y el espacio de memoria utilizados con la nueva metodología, y aquellos presentados previamente por Fernández et al. (2009). Los resultados presentados en el trabajo anterior muestran que los algoritmos propuestos son útiles cuando la matriz $\mathbf{V}_k$ es densa, pero son ineficientes cuando $\mathbf{V}_k$ es dispersa y cumple con las características expresadas en la sección previa. El estudio comparativo realizado en este trabajo consiste en el cálculo de la Ecuación 6 para las mismas escenas, tamaños de matrices y arquitecturas de hardware y software empleados en el trabajo anterior, pero utilizando esta vez la Ecuación 8 para el cálculo del producto $\mathbf{V}_k^T E$.

5.1. Casos de prueba

Se utilizaron los casos de prueba presentados en el trabajo previo (Fernández et al., 2009), diseñados a partir de una *Cornell box* a la que se le aplican diversas discretizaciones (con $n = 3456$, 13824, 55296 y 221184 elementos). Utilizando el algoritmo 2MF o el 2MCE se construyen matrices $\mathbf{U}_k$ y $\mathbf{V}_k$ de dimensiones $n \times k$, con los valores de n mencionados y los valores de $k = 216, 864$ y 3456. No se realizan pruebas con matrices de dimensiones 55296×3456, 221184×864 ni 221184×3456 ya que no fueron consideradas en el artículo original, dado que sus tamaños impedían almacenarlas en la memoria de la GPU utilizada.

5.2. Plataforma de desarrollo y ejecución

El hardware utilizado consiste en un computador con procesador Pentium Dual-Core E5200 a 2.50GHz, con 2 GB de memoria RAM. El computador tiene instalada una tarjeta gráfica NVIDIA 9800 GTX+, con 512MB de memoria.

Para la implementación de los algoritmos en CPU se utilizó la versión de BLAS realizada por el proyecto ATLAS (Whaley et al., 2001), optimizada para el equipo utilizado. En GPU se utilizaron las distribuciones 2.0 de CUDA y CUBLAS.

Tanto la plataforma computacional como las versiones de BLAS y CUBLAS fueron las mismas utilizadas en el trabajo previo por Fernández et al. (2009).

5.3. Evaluación experimental

La comparación de la memoria utilizada para el cálculo de $\widetilde{B}$ cuando $\mathbf{V}_k$ se trata como una matriz densa y cuando se maneja como dispersa, permite concluir que en el caso disperso se utiliza aproximadamente la mitad de la memoria en comparación con el denso. La memoria es ocupada principalmente por las matrices $\mathbf{U}_k$ y $\mathbf{V}_k$, que en ambos casos son matrices de dimensiones $n \times k$, con $n \gg k$. Cuando $\mathbf{V}_k$ es dispersa, es sustituida por un vector de largo $k+1$, por lo que ocupa unas n veces menos memoria que en el caso denso.

En la Tabla 1 se muestran las relaciones de *speedup* alcanzados al calcular el producto $\mathbf{V}_k^T E$ entre las versiones implementadas en CPU y en GPU, cuando la matriz $\mathbf{V}_k$ es densa y dispersa (cuatro implementaciones en total). La Tabla 1 compara el tiempo de cómputo del producto matriz-vector con la función `XGEMV` (en BLAS y CUBLAS), con el tiempo de cómputo del producto utilizando la Ecuación 8 en CPU y GPU, calculando el *speedup* relativo entre las diversas implementaciones. En la primer columna se muestran los resultados del trabajo previo (Fernández et al., 2009), realizado sobre idéntica plataforma experimental, y en las siguientes columnas se presentan las nuevas relaciones de *speedup*. Los mejores valores de *speedup* para cada comparación se destacan con negrita.

Los resultados experimentales muestran la mejora en eficiencia de las nuevas implementaciones de los algoritmos utilizando memoria dispersa respecto a los resultados previos. Se destacan los resultados de *speedup* entre ambas implementaciones para CPU (densa/dispersa), con un valor máximo de **335,5**. y los resultados de *speedup* entre ambas versiones implementadas en GPU, con un valor máximo de **37,47**. Ambos máximos se obtuvieron para el caso con dimensión $\mathbf{13824 \times 3456}$, como consecuencia del volumen de los cálculos en la sumatoria y la estrategia de paralelismo seguida en `XGEMV`.

Dimensión	DensaCPU / DensaGPU	DensaCPU / DispersaCPU	DensaGPU / DispersaGPU	DispersaCPU / DispersaGPU	DensaCPU / DispersaGPU
3456 × 216	1,18	15,67	8,00	0,60	9,40
3456 × 864	4,56	68,33	9,00	0,60	41,00
13824 × 216	1,23	21,67	11,36	0,64	13,93
3456 × 3456	2,75	75,75	18,33	**0,67**	50,50
13824 × 864	4,48	92,11	13,21	0,64	59,21
55296 × 216	1,00	17,42	11,20	0,64	11,20
13824 × 3456	**5,97**	**335,50**	**37,47**	**0,67**	**223,67**
55296 × 864	4,70	94,03	13,09	0,65	61,55
221184 × 216	1,26	24,01	10,99	0,58	13,84

Tabla 1: Speedup al calcular $\mathbf{V}_k^T E$ utilizando la ecuación RRB con geometría fija.

En la Tabla 2 se presentan los valores de speedup del cálculo completo de $\widetilde{B}$. Se han considerado los tiempos de ejecución reportados en el trabajo previo (Fernández et al., 2009) para el cálculo en CPU y GPU con la matriz $\mathbf{V}_k$ densa y se evalúa el *speedup* obtenido al aplicar los algoritmos 2 y 3 para el cálculo de $\widetilde{B}$ en CPU y GPU con la matriz $\mathbf{V}_k$ dispersa para los casos presentados en la Tabla 1. Los mejores valores de *speedup* para cada comparación se destacan con negrita.

Dimensión	DensaCPU / DensaGPU	DensaCPU / DispersaCPU	DensaGPU / DispersaGPU	DispersaCPU / DispersaGPU	DensaCPU / DispersaGPU
3456 × 216	1,91	1,95	4,50	4,40	8,60
3456 × 864	4,59	1,69	2,38	6,48	10,93
13824 × 216	2,02	1,91	**5,21**	5,50	10,53
3456 × 3456	3,99	1,43	2,05	5,70	8,17
13824 × 864	5,37	1,88	3,41	9,73	18,32
55296 × 216	2,10	1,90	5,04	5,58	10,61
13824 × 3456	5,48	1,74	3,59	**11,34**	19,67
55296 × 864	**5,90**	1,95	3,62	10,95	**21,34**
221184 × 216	2,07	**2,05**	5,04	5,08	10,43

Tabla 2: Speedup del cálculo de $\widetilde{B}$ utilizando la ecuación RRB con geometría fija.

En la segunda columna de la Tabla 2 se observa que la versión en CPU utilizando una matriz $\mathbf{V}_k$ dispersa es en promedio casi dos veces más rápida que la versión con una matriz $\mathbf{V}_k$ densa. Este resultado se debe a que el tiempo de cómputo en la versión densa es empleado principalmente en la realización de dos productos matriz-vector (utilizando la función `XGEMV`) que demoran aproximadamente igual tiempo, y a que en la versión con la matriz dispersa uno de los productos es realizado con un código en promedio 90 veces más rápido. En la tercer columna de la Tabla 2 se observa que el mejor valor de *speedup* es de 5,21 y el peor valor 3,4. Los mejores resultados se obtuvieron cuando $k = 216$, que al no ser múltiplo de 32 implica un uso ineficiente de los recursos de cómputo en las GPU (Barrachina et al., 2008), por lo cual podrían obtenerse valores de *speedup* aún superiores para otros valores de k.

Un *speedup* mayor a 2 en el cálculo de $\widetilde{B}$ en GPU parece ser erróneo si se considera que se repite la misma situación que en la CPU, donde dos productos matriz-vector prácticamente iguales se sustituyen por un único producto y otra operación mucho más eficiente. La explicación consiste en que a diferencia de lo que ocurre en el cálculo de $\widetilde{B}$ en CPU cuando la matriz $\mathbf{V}_k$ es densa, las dos ejecuciones de la función `XGEMV` no requieren el mismo tiempo de cómputo en GPU. La segunda ejecución de la función `XGEMV` (cuando se realiza el cálculo de $\mathbf{V}_k^T E$) es unas 3,5 veces más rápida que la primera ejecución, llegando a ser unas 6 veces más rápida cuando $k = 216$, como consecuencia de la forma rectangular de las matrices (el mayor desempeño se alcanza cuando se tienen más filas, ya que `XGEMV` paraleliza el procesamiento de cada fila). Por este motivo, la sustitución de la primer ejecución de `XGEMV` por un código casi 15 veces más rápido en promedio produce un *speedup* final mayor a 2.

En la cuarta columna de la Tabla 1 se observa que la implementación de la Ecuación 8 en GPU es más lenta que en CPU. Este resultado sugiere como línea promisoria desarrollar un método híbrido para tomar ventaja de esta característica, donde $\mathbf{V}_k^T E$ se calcule en la CPU y la operación `XGEMV` restante se calcule en GPU. Con esta innovación se realizaría cada operación utilizando el hardware de mayor rendimiento. Otras ventajas del método híbrido consisten en que se disminuirían los tiempos dedicados a las comunicaciones (solo se transmitirían k números de punto flotante de la memoria RAM a la GPU) y que se podrían utilizar en forma concurrente ambos dispositivos (esto es, cuando la GPU está calculando el segundo producto matriz-vector para el cálculo de la radiosidad $\widetilde{B}$, la CPU ya puede estar calculando el primer producto del siguiente $\widetilde{B}$). Este tipo de estrategias híbridas y concurrentes han mostrado importantes aceleraciones en cómputos algebraicos (Ezzatti et al., 2010).

En la Tabla 3 se observan los resultados de los cuatro algoritmos (los dos desarrollados en el trabajo previo para matrices densas y los dos presentados en este artículo para matrices dispersas) evaluados de acuerdo a las imágenes por segundo (fps por su sigla en inglés) que son capaces de procesar.

Dimensión	**CPU**		**GPU**	
	Densa	**Dispersa**	**Densa**	**Dispersa**
3456×216	**814**	**1591**	**1556**	**7000**
3456×864	221	372	1014	2414
13824×216	196	374	395	2059
3456×3456	70	100	278	569
13824×864	51	96	273	933
55296×216	47	90	100	504
13824×3456	14	24	76	275
55296×864	12	23	70	252
221184×216	11	23	23	116
Promedio	159	299	421	1569

Tabla 3: Imágenes por segundo procesadas por las cuatro implementaciones desarrolladas, para los diferentes casos

Los resultados de la Tabla 3 son teóricos, en el sentido que expresan la cantidad de veces por segundo que se podrían generar imágenes de las escenas presentadas, si el tiempo de cómputo únicamente residiera en el cálculo de la radiosidad $\widetilde{B}$. En realidad, la generación y despliegue de cientos de miles de triángulos en la memoria de pantalla puede aumentar significativamente el período de latencia. Las relaciones de *speedup* de los diversos algoritmos son idénticas a las presentadas en la Tabla 2. En CPU la cantidad mínima de imágenes por segundo se duplica respecto al trabajo previo, alcanzando los **23** fps, y en GPU se quintuplica, pasando de 23 fps a **116** fps. Los valores promedio se duplican en CPU y se cuadruplican en GPU. Por último resulta llamativo el pico en GPU, alcanzándose **7000** fps para el caso disperso en que las matrices tienen dimensiones 3456×216, más de cuatro veces superior al caso de las mismas dimensiones trabajando con matrices densas.

6. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha estudiado la utilización de matrices dispersas en la técnica de radiosidad de rango bajo y se han presentado nuevas versiones de algoritmos implementados en CPU y en GPU que resuelven eficientemente el problema de radiosidad, tomando en cuenta que la matriz $\mathbf{V}_k$ es una matriz dispersa de características particulares.

Tomando como base el trabajo previo (Fernández et al., 2009), se desarrolló una metodología de cálculo adaptativo de la malla gruesa, mediante la incorporación del concepto de coherencia espacial. Se definió la función $P : e \rightarrow p$, que relaciona el espacio de índices de los elementos con el espacio de índices de los parches, con la que es posible construir una matriz $\mathbf{V}_k$ dispersa. Como consecuencia de la permutación de las filas de $\mathbf{V}_k$, es posible transformarla en otra matriz cuya estructura puede almacenarse eficientemente en un vector I de dimensión $k+1$. Esta nueva estrategia permite simplificar el cálculo de de $\mathbf{V}_k^T E$, reduciendo el uso de memoria y el orden de complejidad del cómputo de $O(nk)$ a $O(n)$.

Se realizaron estudios comparativos entre los tiempos de ejecución de los algoritmos implementados en CPU y GPU, y con los tiempos de ejecución reportados en el trabajo previo (Fernández et al., 2009). Los resultados experimentales permitieron comprobar la utilidad de la nueva técnica que utiliza matrices dispersas, alcanzando factores de aceleración (*speedup*) en GPU de **4,1** en promedio, con un máximo de **5,2**), mientras que la implementación en CPU alcanzó valores promedio de **1,9** y un máximo de **2,1**. La evaluación de las imágenes por segundo que es capaz de procesar cada algoritmo, los resultados mostraron que en GPU se alcanza un mínimo de **116** cálculos de radiosidad por segundo (para una escena con 221.184 parches) y un máximo de **7.000** cálculos de radiosidad por segundo (para una escena con 3.456 parches), siendo el valor promedio de **2.069** imágenes por segundo. Las aceleraciones de los métodos en CPU al utilizar una matriz $\mathbf{V}_k$ dispersa fueron menores a los conseguidos en GPU, pero se alcanzaron mínimos y máximos similares a los que se obtienen en GPU cuando la matriz $\mathbf{V}_k$ es densa. Además, los resultados experimentales permitieron verificar que el método implementado en GPU para el cálculo de $\widetilde{B}$ es eficiente, a pesar de que el cómputo de de $V_k^T E$ utilizando la Ecuación 8 requiere tiempos superiores al cómputo en CPU.

Las principales líneas de trabajo futuro se orientan a mejorar la eficiencia de los métodos propuestos, con el objetivo de lograr procesar imágenes realistas en el menor tiempo posible, e introducir el procesamiento de escenas con movimientos. Entre las líneas a desarrollar de forma inmediata se destacan la implementación de un algoritmo que está basado en un esquema híbrido concurrente, capaz de sacar provecho de la velocidad de cálculo del producto $\mathbf{V}_k^T E$ en CPU y la realización de experiencias con hardware gráfico más potente, para poder calcular en tiempo real la radiosidad de una escena con más de un millón de parches. A mediano plazo se propone abordar el diseño e implementación de algoritmos que, siguiendo el enfoque propuesto en este trabajo, realicen el cálculo de la radiosidad de la escena en tiempo real, en contextos donde se permita la realización de modificaciones relativamente pequeñas a la geometría de la escena.

REFERENCIAS

Barrachina S., Castillo M., Igual F., Mayo R., y Quintana-Orti E. *Evaluation and Tuning of Level 3 CUBLAS for Graphics Processors*, capítulo Workshop on Parallel and Distributed Scientific and Engineering Computing. PDSEC, Miami (EE.UU.), 2008.

Barrett R., Berry M., Chan T.F., Demmel J., Donato J., Dongarra J., Eijkhout V., Pozo R., Romine C., y der Vorst H.V. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

Björck A. y Dalhquist G. *Numerical Mathematics in Scientific Computing*. SIAM, 1999.

Cohen M., Wallace J., y Hanrahan P. *Radiosity and realistic image synthesis*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

Dutre P., Bala K., Bekaert P., y Shirley P. *Advanced Global Illumination*. A. K. Peters, Ltd, Natick, MA, USA, 2006. ISBN 1568813074.

Ezzatti P., Quintana-Ortí E., y Remón A. Improving the performance of matrix inversion with a tesla gpu. In *HPC 2010, High-Performance Computing Symposium*, páginas 3211–3219. 2010. ISSN 978-3-642-14121-8.

Fernández E. Low-rank radiosity. In *Proceedings IV Iberoamerican Symposium in Computer Graphics*, páginas 55–62. 2009.

Fernández E. *Resolución del problema de radiosidad usando matrices de rango bajo*. Tesis de Maestría, Universidad de la República, Montevideo, Uruguay, 2010.

Fernández E., Ezzatti P., y Nesmachnow S. Implementación en GPU del algoritmo de radiosidad de rango bajo. In *Proceedings XVIII Congreso sobre Métodos Numéricos y sus Aplicaciones*, páginas 241–251. 2009.

Fernández E. y Nesmachnow S. Conceptos sobre radiosidad de rango bajo. In *Proceedings ECIMAG 2010*. 2010.

Golub G. y Loan C.V. *Matrix Computations*. The Johns Hopkins University Press, 1996.

Goreinov S. y Tyrtyshnikov E. The maximal-volume concept in approximation by low-rank matrices. *Contemporary Mathematics*, 280(1):47–51, 2001.

Goreinov S., Tyrtyshnikov E., y Zamarashkin N. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261:1–21, 1997.

Keller A. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, páginas 49–56. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

Lawson C.L., Hanson R.J., Kincaid D.R., y Krogh F.T. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

NVIDIA. *CUDA CUBLAS Library*. NVIDIA Corporation, Santa Clara, California, 2008.

Press W., Teukolsky S., Vetterling W., y Flannery B. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2007.

Sloan P., Kautz J., y Snyder J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, páginas 527–536. ACM, New York, NY, USA, 2002.

Sutherland I., Sproull R., y Schumacker R. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys*, 6:1–55, 1974.

Wang R., Wang R., Zhou K., Pan M., y Bao H. An efficient GPU-based approach for interactive global illumination. In *ACM SIGGRAPH 2009 papers*, páginas 1–8. ACM, New York, NY, USA, 2009.

Whaley R., Petitet A., y Dongarra J. Automated empirical optimization of software and the atlas project. *Parallel Computing*, 27(1-2):3–35, 2001.