Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVIII, págs. 241-251 (artículo completo)
Cristian García Bauza, Pablo Lotito, Lisandro Parente, Marcelo Vénere (Eds.)
Tandil, Argentina, 3-6 Noviembre 2009

IMPLEMENTACIÓN EN GPU DEL ALGORITMO DE RADIOSIDAD DE RANGO BAJO

Eduardo Fernández[a], **Pablo Ezzatti**[a] **y Sergio Nesmachnow**[a]

[a]*Grupo Centro de Cálculo, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, http://www.fing.edu.uy/inco/grupos/cecal*
e-mail:eduardof@fing.edu.uy, pezzatti@fing.edu.uy, sergion@fing.edu.uy

Palabras Clave: radiosidad, radiosidad de rango bajo, GPU.

Resumen. Las técnicas de radiosidad se utilizan en computación gráfica para determinar la iluminación global de escenas con superficies de reflexión difusa. Estas técnicas se basan en la resolución de una integral de Fredholm de segunda especie, que al discretizarla por elementos finitos se transforma en un sistema lineal $n \times n$, con n el número de elementos (parches). El sistema tiene por expresión general $(\mathbf{I} - \mathbf{RF})B = E$. Generalmente la matriz del sistema es densa y n es mayor a 10^3, habiendo casos donde n es mayor a 10^6. Si varía la emisión de luz de los parches y no así la geometría de la escena, equivale a que varie sólo el término independiente E en el sistema lineal.
Una propuesta para abordar problemas de grandes dimensiones donde no varía la geometría de la escena es la técnica denominada Radiosidad de Rango Bajo (RRB). Esta técnica posee dos etapas, una de preprocesamiento que se realiza una vez y otra de tiempo real, donde se resuelve el sistema lineal para distintos vectores E.
Este artículo presenta dos implementaciones de la etapa de tiempo real de la técnica RRB: una implementación tradicional sobre CPU y una nueva sobre GPU (Graphics Processing Unit). En ambos casos se utilizan subrutinas de BLAS. La evaluación experimental se realiza sobre una serie de escenas que implican la resolución de sistemas lineales con valores de n entre 3.500 y 220.000. El análisis de los tiempos de ejecución y los resultados numéricos permiten concluir que la implementación en GPU alcanza niveles de eficiencia computacional de hasta 6 veces superiores a los de la implementación de CPU, sin perder calidad en los resultados. Estos resultados abren la posibilidad del desarrollo de aplicaciones gráficas interactivas que resuelvan el problema de radiosidad en tiempo real con escenas relativamente complejas.

1 INTRODUCCIÓN

La resolución de la radiosidad de una escena tridimensional es parte del problema general de sintetizar imágenes realistas por medios computacionales. Los métodos de radiosidad consideran el caso particular en que todas las superficies tienen reflexión Lambertiana, y hallan la distribución de la energía luminosa en todas las superficies de la escena (ver Figura 1). El término radiosidad es sinónimo de emitancia radiante o exitancia, potencia emitida de radiación luminosa por unidad de superficie.

La ecuación de rendering (Kajiya, 1986) es una ecuación de Fredholm de segunda especie, que tiene por caso particular a la ecuación de radiosidad (Cohen y Wallace, 1993):

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

donde $B(x)$ es la función de radiosidad y expresa un valor de radiosidad por cada punto x de la superficie, $E(x)$ indica la emisión luminosa de cada punto x de la superficie, $r(x)$ indica el coeficiente de reflectividad de la luz incidente en x, $G(x, x')$ expresa la relación geométrica entre los puntos x y x', dA' es el diferencial de área en x' y S es el conjunto de todas las superficies de la escena.

Los intentos por resolver esta ecuación se han centrado en dos métodos principales: los elementos finitos y la utilización de la traza de rayos basada en el método de Monte Carlo. En este artículo se desarrollan conceptos a partir de la primera aproximación.

Eckbert y Winget (1991), han mostrado la relación existente entre radiosidad y los elementos finitos. En esta relación, las escenas son aproximadas por mallas de polígonos llamados parches y la ecuación de radiosidad (1) es aproximada por una combinación lineal de matrices $n \times n$ y vectores $n \times 1$, con n el total de parches:

$$B = E + \mathbf{RF}B \tag{2}$$

En esta ecuación, B es un vector con los valores de radiosidad de cada parche, E es un vector con la emisión de cada parche, $\mathbf{R}$ es una matriz diagonal que indica el factor de reflectividad de cada parche y $\mathbf{F}$ es una matriz que contiene los factores de forma $\mathbf{F}_{ij}$. Un factor de forma $\mathbf{F}_{ij}$ indica el porcentaje de energía radiada por el parche i que incide en el parche j. La matriz $\mathbf{F}$ es por lo general densa, especialmente si las escenas corresponden a espacios abiertos donde cada parche ve un porcentaje elevado del resto de los parches.

Dado que la incógnita suele ser B y los demás datos están dados, la ecuación (2) se puede reformular como:

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

Como propiedades básicas del sistema lineal sobresalen:

1. n es de orden mayor a 10^3, habiendo casos con orden 10^6.
2. $\mathbf{RF}$ es una matriz densa, por lo que la memoria necesaria para su almacenamiento es de orden n^2 flotantes.
3. $\mathbf{AF} = (\mathbf{AF})^T$ siendo $\mathbf{A}$ una matriz diagonal, donde $\mathbf{A}_{ii}$ es el área del parche i.
4. $\mathbf{F}_{ii} = 0$ en la mayoría de los modelos. Se debe a que los parches suelen ser planos y la energía que cada parche emite no incide de forma directa en el mismo parche.
5. $0 \le \sum_{j=1}^{n} \mathbf{F}_{ij} \le 1$ porque la sumatoria indica el porcentaje de toda la energía emitida por el parche i que incide en el resto de los parches. Puede ser menor que 1 si parte de la energía emitida no incide en ningún parche y "se pierde en el espacio".

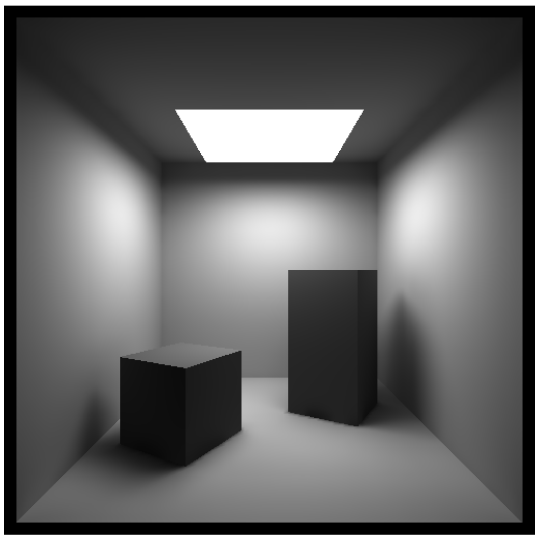

Figura 1: Imagen generada aplicando las técnicas de radiosidad

Recientemente se han propuesto algunas estrategias para el abordaje de problemas de grandes dimensiones. En particular, se destaca la técnica Radiosidad de Rango Bajo (Fernández, 2009). Esta técnica se puede descomponer en dos etapas, una de preprocesamiento y otra de cálculo en tiempo real.

En este artículo se comparan dos implementaciones de la etapa de tiempo real de la técnica Radiosidad de Rango Bajo (RRB): una tradicional sobre CPU utilizando BLAS y otra desarrollada sobre la arquitectura de GPU (Graphics Processing Unit).

El resto del artículo se estructura de la siguiente forma. La sección 2 presenta una descripción genérica de la técnica de RRB. La sección 3 explica los métodos implementados y las distintas técnicas utilizadas. Los resultados experimentales sobre los casos de prueba generados se comparan en la sección 4. Por último, se ofrecen las conclusiones y propuestas de trabajo futuro.

2 PLANTEO GENERAL DEL PROBLEMA

Una técnica muy difundida para la resolución de problemas de grandes dimensiones es la búsqueda de problemas parecidos que sean computacionalmente tratables. En particular, para el problema de radiosidad, resulta útil examinar la coherencia espacial en mallas de parches generadas a partir de las escenas modeladas. Si la malla es lo suficientemente fina, se puede establecer que en general dos parches p_1 y p_2 adyacentes y con normales parecidas ven prácticamente la misma escena (Fernández, 2009), por tanto los factores de forma $\mathbf{F}_{p_1 j}$ y $\mathbf{F}_{p_2 j}$ son prácticamente iguales para todo j, con lo que se concluye que las filas $\mathbf{F}_{p_1 *}$ y $\mathbf{F}_{p_2 *}$, son prácticamente iguales. Esta coherencia espacial es común en muchas escenas para buena parte de los pares de parches adyacentes. Del punto de vista matricial, la coherencia espacial hace que la matriz **F** tenga un rango numérico bajo y que por tanto su descomposición SVD tenga muchos valores singulares pequeños (Fernández, 2009; Hašan, 2007). El producto **RF** mantiene las relaciones lineales de las filas de **F**.

Si alcanza con combinar unas pocas filas y columnas de **RF** (k con $k<<n$) para obtener una buena representación de esta, la ecuación (3) puede transformarse en otra parecida:

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

donde $\mathbf{U}_k\mathbf{V}_k^T$ es una aproximación de rango bajo de **RF**, $\mathbf{U}_k$ y $\mathbf{V}_k$ son matrices $n\times k$ y la incógnita $\tilde{B}$ es una aproximación a *B*. Una ventaja evidente de la ecuación (4) respecto de (3) es el ahorro de memoria conseguido por necesitar sólo 2*nk* flotantes para la nueva matriz.

El error de $\tilde{B}$ se puede estimar considerando la ecuación (4) como una versión perturbada de (3) (Fernández, 2009):

$$E=(\mathbf{I}-\mathbf{U}_k\mathbf{V}_k^T)\tilde{B}=(\mathbf{I}-\mathbf{RF}+\underbrace{\mathbf{RF}-\mathbf{U}_k\mathbf{V}_k^T}_{\boldsymbol{\varepsilon}})\tilde{B}=(\mathbf{I}-\mathbf{RF}+\boldsymbol{\varepsilon})\tilde{B} \tag{5}$$

Con esta nueva expresión, la cota del error relativo de la solución es (Golub y Van Loan, 1996):

$$\frac{\left\|\tilde{B}-B\right\|}{\left\|B\right\|}\le\left\|\boldsymbol{\varepsilon}\right\|\left\|(\mathbf{I}-\mathbf{RF})^{-1}\right\|\le\frac{\left\|\boldsymbol{\varepsilon}\right\|}{1-\left\|\mathbf{RF}\right\|} \tag{6}$$

En esta cota el único término que se puede modificar es la matriz $\boldsymbol{\varepsilon}$, que depende de la aproximación $\mathbf{U}_k\mathbf{V}_k^T$.

La ecuación (4) posibilita el uso de métodos directos para el cálculo de $\tilde{B}$. Es habitual encontrar en la literatura algoritmos de radiosidad que utilizan técnicas iterativas como Gauss-Seidel (Cohen y Wallace, 1993) o métodos no estacionarios (Baranoski, 1995). Por el contrario, si se aplica la fórmula de Sherman-Morrison-Woodbury (Golub y Van Loan, 1996) a la ecuación (4), se simplifica la resolución de forma significativa. En efecto:

$$\tilde{B}=(\mathbf{I}-\mathbf{U}_k\mathbf{V}_k^T)^{-1}E=(\mathbf{I}+\mathbf{U}_k(\mathbf{I}_k-\mathbf{V}_k^T\mathbf{U}_k)^{-1}\mathbf{V}_k^T)E=E+(\mathbf{U}_k((\mathbf{I}_k-\mathbf{V}_k^T\mathbf{U}_k)^{-1}(\mathbf{V}_k^TE))) \tag{7}$$

donde $\mathbf{I}_k$ es la matriz identidad de dimensiones $k\times k$.

En (7) las expresiones se van haciendo más complejas en su formulación, de izquierda a derecha, pero más simples en cuanto a su cómputo. A la izquierda se plantea el cálculo de la inversa de una matriz $n\times n$ (con complejidad de orden n^3 y necesidad de memoria de orden n^2), y la expresión de la derecha plantea el cálculo de la inversa de una matriz $k\times k$ y varios productos matriz-vector que tienen en total un orden nk^2. Si $k<<n$, el ahorro logrado en tiempo y memoria permite resolver el problema de radiosidad en forma directa.

Cuando la geometría es fija y sólo cambian los valores de emisión de los parches, el cálculo de $\mathbf{U}_k$ y $\mathbf{V}_k$ se realiza una única vez. Por tanto en (7), la expresión de más a la derecha se puede simplificar:

$$\tilde{B}=E-\mathbf{Y}_k(\mathbf{V}_k^TE) \tag{8}$$

donde $\mathbf{Y}_k$ es una matriz $n\times k$:

$$\mathbf{Y}=-\mathbf{U}_k(\mathbf{I}_k-\mathbf{V}_k^T\mathbf{U}_k)^{-1} \tag{9}$$

En (8) se utiliza orden *nk* de memoria y la complejidad del cálculo de $\tilde{B}$, una vez hallada $\mathbf{Y}_k$, es también de orden *nk*.

2.1 Cálculo de aproximaciones de rango bajo de RF

Para el cálculo de $\mathbf{U}_k$ y $\mathbf{V}_k$ se puede utilizar cualquier matriz $\mathbf{V}\, n\times n$ ortonormal. En efecto,

$$(\mathbf{I}-\mathbf{RF}) = (\mathbf{I}-\mathbf{RF})\mathbf{V}\mathbf{V}^T = \mathbf{I}-\underbrace{\mathbf{RFV}}_{\mathbf{U}}\mathbf{V}^T = \mathbf{I}-\mathbf{U}\mathbf{V}^T = \mathbf{I}-\sum_{\substack{i=1\\ \|U_i\|\geq\|U_{i+1}\|}}^{n} U_i V_i^T \tag{10}$$

donde U_i y V_i son las columnas de **U** y **V**. Si se toman los k primeros términos de la sumatoria, se obtiene una expresión matricial igual a la utilizada en (4):

$$\mathbf{I}-\sum_{\substack{i=1\\ \|U_i\|\geq\|U_{i+1}\|}}^{k} U_i V_i^T = \mathbf{I}-\mathbf{U}_k\mathbf{V}_k^T \tag{11}$$

Como se mencionó anteriormente, la diferencia entre la matriz real y la aproximada afecta directamente al error de $\tilde{B}$. Se puede establecer genéricamente que:

$$\|\boldsymbol{\varepsilon}\| = \left\|(\mathbf{I}-\mathbf{RF})-(\mathbf{I}-\mathbf{U}_k\mathbf{V}_k^T)\right\| = \left\|\sum_{\substack{i=k+1\\ \|U_i\|\geq\|U_{i+1}\|}}^{n} U_i V_i^T\right\| \leq \sum_{\substack{i=k+1\\ \|U_i\|\geq\|U_{i+1}\|}}^{n} \|U_i\| \tag{12}$$

Para el caso particular en que **V** se obtiene por la descomposición en valores singulares, (V=$\mathcal{V}$, con **RF**=$\mathcal{U}\mathcal{D}\mathcal{V}^T$) se puede demostrar que los U_i son ortogonales y que para norma-2:

$$\|\boldsymbol{\varepsilon}\|_2 = \left\|\sum_{\substack{i=k+1\\ \|U_i\|\geq\|U_{i+1}\|}}^{n} U_i V_i^T\right\|_2 = \|U_{k+1}\|_2 = \mathcal{D}_{k+1,k+1} \tag{13}$$

En la práctica, el cálculo de la descomposición SVD es costoso. Otras bases ortonormales más convenientes por los órdenes de complejidad de sus cálculos, son las que se generan a partir de las transformadas de Fourier y Wavelet discretas (Press et al., 2007).

Aplicado al problema de radiosidad, Fernández (2009) propuso un método que genera $\mathbf{U}_k$ y $\mathbf{V}_k$ a través de un algoritmo de complejidad n^2 y orden de memoria nk. Este método consiste en establecer una jerarquía de dos niveles de parches (parches y subparches). En la escena existen k parches y cada parche se lo subdivide en d subparches de igual área, por lo que $n = dk$. Se saca provecho de la coherencia espacial al suponer que son parecidas las columnas de **RF** asociadas a los subparches que pertenecen a un mismo parche. Para cada parche se calcula la columna promedio. Existen k columnas promedio por haber k parches. La matriz $\mathbf{U}_k$ almacena todas las columnas promedio. La matriz $\mathbf{V}_k$ es una matriz dispersa (un único elemento distinto de cero por fila), generada de forma tal que el producto $\mathbf{U}_k\mathbf{V}_k^T$ es una matriz donde las columnas asociadas a los subparches que pertenecen a un mismo parche, son las columnas promedio del parche. El producto $\mathbf{U}_k\mathbf{V}_k^T$ tiene sólo k columnas diferentes.

En base a lo expuesto en esta sección, para el caso en que la geometría es fija y sólo varía la iluminación, conviene dividir el cálculo de $\tilde{B}$ en dos etapas: una primera, que denominaremos de preprocesamiento, donde se calculan $\mathbf{U}_k$, $\mathbf{V}_k$ e $\mathbf{Y}_k$, y otra que denominaremos de tiempo real, donde a partir de (8) y disponiendo del vector de emisión E se calcula $\tilde{B}$.

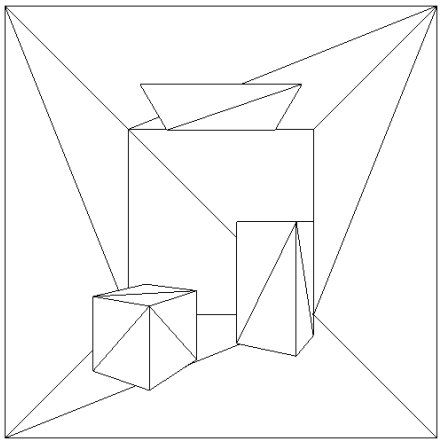

Figura 2: Cornell box original con 36 parches

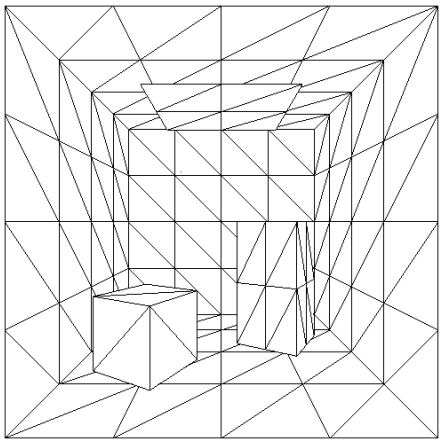

Figura 3: Escena con k = 216 parches

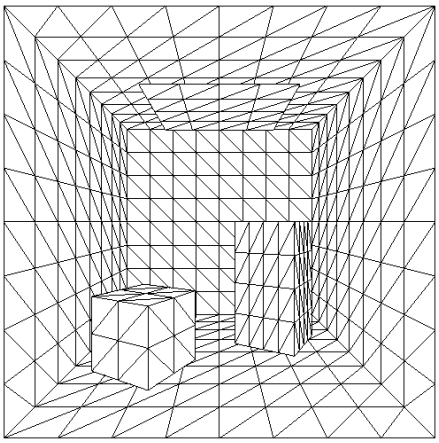

Figura 4: Escena con k = 864 parches

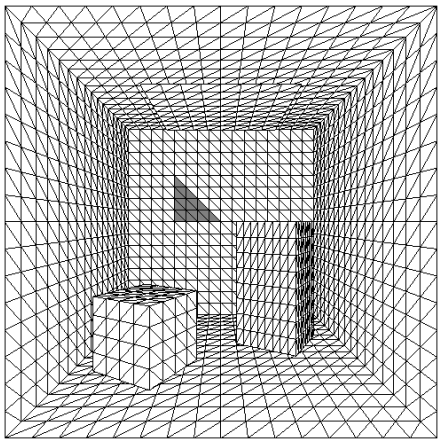

Figura 5: Escena con n = 3.456 subparches. Se resalta uno de los 216 parches de la estructura jerárquica.

3 IMPLEMENTACIÓN DE LA ETAPA DE TIEMPO REAL EN CPU Y GPU

La implementación de la etapa de tiempo real de la técnica de RRB implica el cálculo de $\tilde{B}$ a partir de la ecuación (8), asumiendo que se conocen las matrices $\mathbf{Y}_k$ y $\mathbf{V}_k$.

El orden de los cálculos está guiado por la precedencia establecida por el paréntesis contenido en (8). La primera operación es el producto $\mathbf{V}_k^T E$, cuyo resultado es un vector $k \times 1$. Ese vector es multiplicado por $\mathbf{U}_k$, dando por resultado un vector $n \times 1$ que es restado a E para hallar $\tilde{B}$. El total de las operaciones tiene complejidad de orden *nk.*

Se han realizado dos códigos que calculan $\tilde{B}$ a partir de (8). En una versión se realizan los cálculos en la CPU y en otra versión en una GPU.

Para simular el procesamiento en tiempo real, es necesario considerar que en realidad los vectores E no están disponibles a priori sino que surgen en tiempo real a partir de la interacción del usuario con las fuentes luminosas. Esto es importante, sobre todo en el caso GPU, dado que a los tiempos de cálculo hay que sumar los tiempos de envío de cada vector E y de recepción de cada vector $\tilde{B}$ entre la memoria RAM y la GPU.

La realización eficiente de productos matriciales está facilitado por la existencia de bibliotecas estándares como BLAS (Lawson, 1979). Para el cálculo de (8) alcanza con realizar

dos llamadas consecutivas a la subrutina `XGEMV` de BLAS.

La versión CPU utiliza la implementación de BLAS realizada por el proyecto ATLAS (Whaley et al., 2001), optimizada para el equipo utilizado. El código implementado consiste de la siguiente estructura:

1. Cargar las matrices $\mathbf{Y}_k$ y $\mathbf{V}_k$ y los vectores E en la memoria RAM.
2. For i = 1:CantE
3. Tomar un vector E
4. Aplicar `XGEMV` dos veces para obtener el vector $\tilde{B}$.
5. Almacenar el vector $\tilde{B}$ en la memoria RAM.
6. endFor

La versión GPU utiliza la versión de BLAS desarrollada sobre arquitectura CUDA denominada CUBLAS (NVIDIA, 2008), versión 2.0. El código implementado consiste en:

1. Cargar las matrices $\mathbf{Y}_k$ y $\mathbf{V}_k$ y los vectores E en la memoria RAM.
2. Pasar las matrices $\mathbf{Y}_k$ y $\mathbf{V}_k$ a la memoria de la GPU.
3. For i = 1:CantE
4. Pasar un vector E a la memoria de la GPU.
5. Aplicar `XGEMV` dos veces en la GPU para obtener el vector $\tilde{B}$.
6. Enviar el vector $\tilde{B}$ de la memoria de la GPU a la memoria RAM.
7. endFor

Se determina el tiempo promedio de cálculo de un $\tilde{B}$ evaluando el tiempo insumido por el *loop* divido la cantidad de iteraciones.

Tanto la versión CPU como la GPU utilizan precisión simple en los cálculos.

4 REALIZACIÓN DE PRUEBAS Y ESTUDIO DE RESULTADOS

Para comparar la eficiencia del cálculo de $\tilde{B}$ con ambas arquitecturas, se generaron varios $\mathbf{U}_k$ y $\mathbf{V}_k$ en la etapa de preprocesamiento. Se utilizaron valores de n iguales a 3.456, 13.824, 55.296 y 221.184, y valores de k iguales a 216, 864 y 3.456.

En la etapa de preprocesamiento se utilizó el algoritmo de Fernández (2009). Se parte de una Cornell Box, escena muy utilizada en computación gráfica, compuesta de 36 triángulos (ver Figura 2). Cada uno de los triángulos es dividido en partes iguales hasta que los triángulos resultantes tengan un área menor a un valor definido. Esos triángulos son los k parches en la estructura jerárquica de la escena. Luego se divide cada parche en d triángulos de igual área, generando los $n = kd$ subparches de la escena. De forma complementaria se genera una estructura auxiliar que relaciona cada subparche con su parche padre.

En la Figura 2 se muestra la escena original utilizada en los experimentos. Consta de 36 parches. Los parches de dividen obteniendo diversos valores de k y n. Las Figuras 3 y 4 representan los casos en que la cantidad de parches es k = 216 y 864. En la Figura 5 se representa el caso en que la cantidad de subparches es n = 3.456. Estos subparches se generaron a partir de los parches de la Figura 3. Se resalta uno de los parches originales.

Las seis dimensiones utilizadas para la generación de los casos de prueba cumplen con la particularidad de ser miembros de la serie $216*4^i$ con i entero entre 0 y 5. Esta particularidad posibilita encontrar varias combinaciones de n y k con igual producto nk (p. ej. nk = 55.296*216 = 13.824*864 = 3.456*3.456).

Las combinaciones testeadas son detalladas en la Tabla 1. En esta tabla, hay una secuencia de diagonales (celdas de la misma tonalidad). En cada diagonal las celdas tienen igual producto nk. El valor nk entre diagonales se va incrementando por cuatro. No se generaron los

n \ *k*	216	864	3.456
3.456	*Sí*	*Sí*	*Sí*
13.824	*Sí*	*Sí*	*Sí*
55.296	*Sí*	*Sí*	~~*No*~~
221.184	*Sí*	~~*No*~~	~~*No*~~

Tabla 1: Las combinaciones testeadas de *n* y *k*.

$\mathbf{U}_k$ y $\mathbf{V}_k$ para las doce combinaciones posibles de pares *nk*, debido a limitaciones de memoria en la tarjeta gráfica utilizada.

En el diseño de casos se buscó generar dos series de tres casos cada una con igual producto *nk*, y así tener más información para comprobar si los tiempos del cálculo de $\tilde{B}$ están relacionados con el orden teórico. De allí surge el caso de la celda de bordes punteados, donde $n = k = 3.456$. Este caso no tiene sentido práctico porque la metodología es útil cuando *k* es mucho mayor que *n*.

La Tabla 2 describe los diferentes casos de prueba generados. Por cada caso se presenta el total de parches, el total de subparches, el producto *nk* y la memoria usada en $\mathbf{U}_k$ y $\mathbf{V}_k$.

La totalidad de las pruebas se realizaron en un computador con procesador Pentium(R) Dual-Core CPU E5200 de 2.50GHz, con 2 GB de memoria RAM. El computador tiene instalada una tarjeta gráfica NVIDIA 9800 GTX+, con 512Mb de memoria.

En la Tabla 3 se presentan los tiempos obtenidos al evaluar las versiones CPU y GPU con los casos de prueba. Se despliegan los tiempos del cálculo de $\tilde{B}$ en CPU y GPU, los tiempos de transferencia de *E* y $\tilde{B}$ hacia y desde la GPU y el *speedup* de la GPU respecto a la CPU.

Observando los resultados de la Tabla 3 se pueden establecer las siguientes conclusiones preliminares:

- El *speedup* CPU/GPU varía entre 1,7 y 6 veces.
- El tiempo de cálculo de $\tilde{B}$ en CPU es proporcional de forma aproximada con el producto *nk*. Hay tiempos parecidos dentro de cada franja *nk*, y entre franjas consecutivas los tiempos cambian de escala por un factor de entre tres y cinco.
- El tiempo de cálculo de $\tilde{B}$ en GPU es más irregular. Existen dos grupos de casos: aquellos en que $k = 216$ y aquellos en que $k \neq 216$. Con la información disponible, se observa que los tiempos son proporcionales a *nk*. Por otra parte, los tiempos y el *speedup* empeoran unas tres veces cuando $k = 216$ respecto al otro grupo. Esta singularidad fue observada en los trabajos de Igual et al. (2009) y Barrachina et al. (2008), quienes plantean que las dimensiones matriciales deben ser múltiplos de 32. De todos los *n* y *k* considerados en este artículo, sólo 216 no es múltiplo de 32.
- Los tiempos de transferencia de *E* y $\tilde{B}$ son proporcionales a *n*.
- El tiempo de cálculo de $\tilde{B}$ en la GPU es entre 10 y 120 veces mayor que el tiempo de transferencia de *E* y $\tilde{B}$.
- El cálculo y transferencia de $\tilde{B}$ en la versión GPU se puede realizar a 20 Hz en todos los casos, y a más de 60 Hz si no se considera el caso 9, que tiene $k = 216$. Estos valores están dentro de los empleados por muchas aplicaciones interactivas (Ware, 2000).

Casos	**Total de Subparches** (n)	**Total de Parches** (k)	**Flotantes por matriz** (nk) en millones	**Subparches por Parche** ($d=n/k$)	**Espacio en Memoria GPU** ($\mathbf{U}_k + \mathbf{V}_k$) en Mbytes
1	3.456	216	0,7	16	5,7
2	3.456	864	3,0	4	22,8
3	13.824	216	3,0	64	22,8
4	3.456	3.456	11,9	1	91,1
5	13.824	864	11,9	16	91,1
6	55.296	216	11,9	256	91,1
7	13.824	3.456	47,8	4	364,5
8	55.296	864	47,8	64	364,5
9	221.184	216	47,8	1.024	364,5

Tabla 2: Descripción general de los casos de prueba. Los grises se corresponden con los de la Tabla 1.

Casos	**Total de Subparches** (n)	**Total de Parches** (k)	(a) **Tiempo de cálculo de $\tilde{B}$ en CPU** en ms	(b) **Tiempo de cálculo de $\tilde{B}$ en GPU** en ms	(c) **Tiempo de transf. de E y $\tilde{B}$ a GPU** en ms	(b +c) **Tiempo total de GPU** en ms	(a)/(b+c) **speedup CPU/GPU**
1	3.456	216	1,27	0,69	0,04	0,73	1,7
2	3.456	864	4,64	1,07	0,03	1,10	4,2
3	13.824	216	5,23	2,51	0,11	2,62	2,0
4	3.456	3.456	14,53	3,61	0,04	3,65	4,0
5	13.824	864	20,14	3,61	0,11	3,72	5,4
6	55.296	216	22,70	9,74	0,37	10,11	2,2
7	13.824	3.456	72,77	13,29	0,11	13,40	5,4
8	55.296	864	88,37	14,19	0,43	14,62	6,0
9	221.184	216	86,99	41,33	1,51	42,84	2,0

Tabla 3: Resultados experimentales de los casos de prueba.

Las Figuras 6, 7, 8 y 9 son algunos ejemplos de imágenes generadas a partir de los $\tilde{B}$ calculados. Se muestran algunas imágenes obtenidas para distintos valores de k y n, partiendo de la misma escena original (Figura 2). Los valores de radiosidad son interpolados con Gouraud (Gouraud, 1971), para lograr mayor sensación de realismo. Los sombreados obtenidos parecen más realistas cuando el número de parches se incrementa. La diferencia se nota más entre la Figura 7 y las Figuras 8 y 9. Es necesario realizar análisis cuantitativos y cualitativos más profundos para determinar la relación existente entre las escenas y los valores mínimos de n y k que permitan obtener precisión y sensación de realismo adecuados.

5 CONCLUSIONES Y TRABAJOS FUTUROS

El artículo presenta la implementación y evaluación de dos versiones de la etapa de tiempo real de RRB. Una versión utiliza CPU para el cálculo de $\tilde{B}$ a partir de (8) y otra versión está basada en el uso de GPU.

Los resultados numéricos de ambas versiones son similares, mientras que el desempeño computacional, evaluado en tiempo de ejecución, resulta marcadamente superior en la versión en GPU. La CPU necesita entre 1,7 y 6 veces más tiempo para realizar el cálculo de $\tilde{B}$.

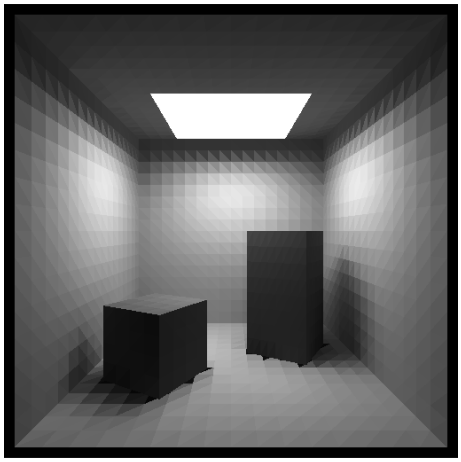

Figura 6: Resultado de radiosidad con 216 parches y 3456 subparches

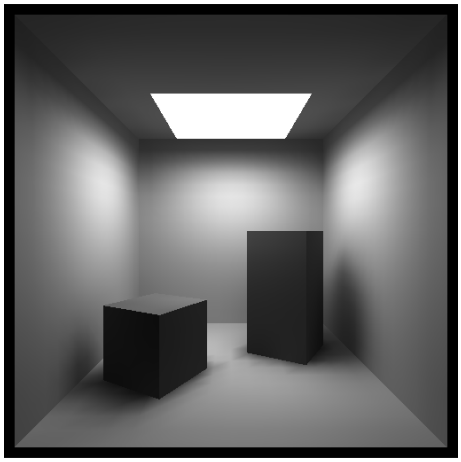

Figura 7: Iguales resultados a los de la Figura 5, interpolados con Gouraud para acentuar el realismo.

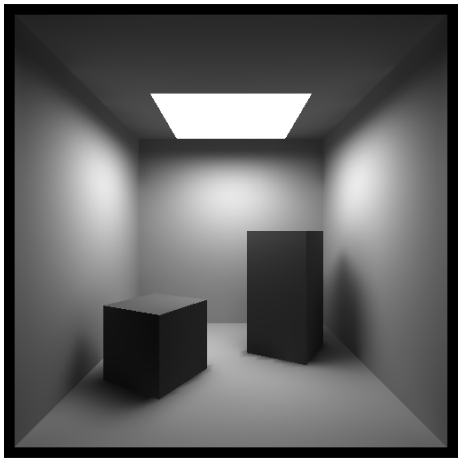

Figura 8: Resultado de radiosidad con 216 parches y 13824 subparches, interpolado con Gouraud.

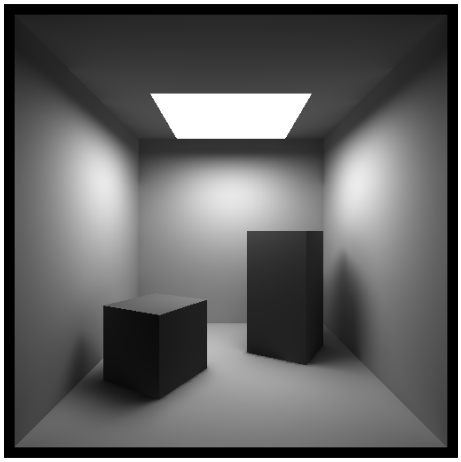

Figura 9: Resultado de radiosidad con 216 parches y 55296 subparches, interpolado con Gouraud.

La propuesta presentada señala una vía para realizar aplicaciones gráficas interactivas que resuelvan el problema de radiosidad en tiempo real. Los monitores tienen una tasa de refresco que varía entre 50 y 75 Hz, y la mayoría de los tiempos relevados muestran que es posible calcular $\tilde{B}$ a tasas iguales e incluso mayores.

Una serie de aspectos sobre la utilización de la GPU en el proceso de resolución de radiosidad admiten ser explorados con mayor detalle. En la actualidad, dentro de nuestro grupo de investigación se están investigando algunos de estos aspectos, mientras que se prevé abordar otros en el futuro cercano.

Una línea de investigación en desarrollo consiste en la utilización de estrategias de cálculo híbridas (CPU-GPU) para la resolución de la etapa de tiempo real de RRB. Por ejemplo, $\mathbf{V}_k^T E$ podría ser calculado en CPU y el resultado multiplicado por $\mathbf{Y}_k$ en GPU. Otra posibilidad consiste en la realización en forma híbrida de cada producto matriz-vector.

En este trabajo no se saca ventaja de que $\mathbf{V}_k$ puede ser dispersa. Al tener $\mathbf{V}_k$ un único elemento distinto de cero por fila, ocupa una memoria de orden *n*, y el producto $\mathbf{V}_k^T E$ tiene también complejidad de orden *n*. Con estos órdenes, los tiempos del cálculo de $\tilde{B}$ a través de (8) y la memoria necesaria se reducen a la mitad. La implementación de $\mathbf{V}_k^T E$ puede realizarse a través de Sparse BLAS (Duff, 2002), o utilizando un código ad hoc.

Otra línea de trabajo a abordar consiste en la utilización de la GPU para la resolución de la etapa de preprocesamiento en la técnica de RRB.

El comportamiento de la GPU con matrices de dimensiones no múltiplo de 32, amerita el desarrollo de técnicas que permitan generar matrices de dimensiones específicas.

Por último, resulta necesaria la realización de pruebas con escenas diversas para establecer relaciones entre la complejidad de los modelos y los valores mínimos recomendables de *n* y *k*.

REFERENCIAS

Baranoski, G., Bramley, R., y Shirley, P., Fast Radiosity Solutions For Environments With High Average Reflectance. *Proceedings of the Sixth EUROGRAPHICS Rendering Workshop*, (EGWR'95), (Rendering Techniques'95) Dublin, Irlanda. 345-356, 1995.

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

Cohen, M., Wallace, J., *Radiosity and Realistic Image Synthesis*, Academic Press, New York, 1993.

Duff, I., Heroux, M., y Pozo, R., An Overview of the Sparse Basic Linear Algebra Subprograms: The New Standard from the BLAS Technical Forum. *ACM TOMS*, 28(2): 239-267, June 2002.

Fernández, E., Low-Rank Radiosity. *Proceedings of IV Iberoamerican Symposium in Computer Graphics*, (SIACG2009), Isla Margarita, Venezuela, 2009.

Golub, G., Van Loan C., *Matrix Computations, Third Edition*. The John Hopkins University Press, 1996.

Gouraud, H., Continuous Shading of Curved Surfaces, *IEEE Trans. on Computers*, C-20(6):623-629, June 1971.

Hašan, M., Pellacini, F., y Bala, K., Matrix row-column sampling for the many-light problem. *Proceedings of ACM SIGGRAPH*, 2007.

Heckbert, P., Winget, J., Finite element methods for global illumination. *Tech. Rep. UCP/CSD 91/643*, Computer Science Division (EECS), University of California, Berkeley, July 1991.

Igual, F., Quintana-Ortí, G., y van de Geijn, R., Level-3 BLAS on a GPU: Picking the Low Hanging Fruit. *Minisymposium on High-Performance Computing and Numerical Linear Algebra. 7th International Conference of Numerical Analysis and Applied Mathematics*, (*ICNAAM 2009*). Rethymno, Crete, Greece. September 18-22, 2009.

Kajiya, J., The rendering equation. *Proceedings of ACM SIGGRAPH* (Computer Graphics), ACM Press. 143–150, 1986.

Lawson, C., Hanson, R., Kincaid, D., y Krogh, F., Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw*., 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 3rd Edition. The Art of Scientific Computing.* Cambridge University Press, 2007.

Ware, C., *Information Visualization: Perception for Design*. Morgan Kaufman, 2000.

Whaley, R., Petitet, A., y Dongarra, J., Automated Empirical Optimization of Software and the ATLAS project. *Parallel Computing*, 27(1-2):3-35, 2001.