CEIG'09, San Sebastián, Sept. 9-11 (2009)
C. Andújar and J. LLuch (Editors)

# Simplificación interactiva de vegetación usando el Hardware Gráfico

J. Gumbau , M. Chover e I. Remolar

Universitat Jaume I, Castellón, Spain

**Abstract**

*La visualización en tiempo real de vegetación es un problema actualmente en estudio debido a la enorme cantidad de primitivas que tienen que dibujarse. Todas ellas son necesarias para describir la geometría de las plantas. Sin embargo, no todas y cada una de estas primitivas contribuyen en la misma forma a la imagen final de la planta: algunas no son visibles en función de la situación de la cámara. Este trabajo, centra su estudio en este hecho con el fin de reducir la cantidad de geometría necesaria para representar un árbol y presenta una representación multirresolución variable dependiente de la vista. El presente método se basa en un sistema de simplificación basado en el punto de vista que se realiza de forma eficiente en la Unidad de Proceso Gráfico (GPU). La apariencia final de los árboles se preserva en todo momento al alterar el tamaño de las hojas restantes, evitando así dar el aspecto de poda. Nuestro enfoque aprovecha la arquitectura en paralelo de la GPU a la hora de determinar en tiempo real las hojas visibles del árbol. Los datos a visualizar no necesitan ser transmitidos a través del bus, debido a que son generados en la misma GPU. Esto hace que se reduzca de forma notable el tiempo de extracción y visualización del follaje.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Informática Gráfica]: Picture/Image Generation—

## 1. Introducción

La visualización realista de plantas siempre ha sido un inconveniente a la hora de visualizar escenas en tiempo real. Este hecho es debido a la gran cantidad de geometría que se necesita para visualizar cada planta. Los trabajos aparecidos hasta la fecha que tratan este problema, pueden ser clasificados en dos grupos diferentes: los métodos basados en imágenes y los basados en geometría. Los primeros, los basados en imágenes, convierten parte o toda la geometría en texturas o impostores orientados a la cámara para reducir el tiempo de visualización. En cambio, los basados en geometría aplican diferentes métodos de simplificación para disminuir el número de primitivas que forman la planta, dependiendo de la distancia a la cámara o de la complejidad del objeto desde un determinado punto de vista.

Los métodos basados en imágenes usan menos geometría y proporcionan unos buenos resultados a una distancia media o lejana. En cambio, el hecho de cambiar la geometría por imágenes hace que se produzcan problemas de paralaje a corta distancia. Los métodos basados en geometría, por otro lado, ofrecen una apariencia más realista en distancias cortas debido a que pueden mostrar gran cantidad de detalle. No obstante, el problema de este tipo de métodos es que cuanto más detalle se represente en una planta, más elevado será el número de primitivas a visualizar. Este hecho constituye el cuello de botella de estos métodos. Los modelos multirresolución o algoritmos de nivel de detalle (LoD) ayudan a adaptar el número de primitivas necesario según unos determinados criterios, y así hacer posible la visualización en tiempo real de estas representaciones. Estos se pueden clasificar en modelos multirresolución discretos y continuos.

Los modelos multirresolución continuos aplican un proceso de simplificación a los objetos geométricos, reduciendo de forma gradual la complejidad poligonal de las mallas. Esta simplificación se realiza generalmente como un preproceso. Al cambiar el nivel de detalle de los objetos de



DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/217-226

forma suave, evitan que el observador pueda apreciar el cambio en el número de primitivas. No obstante, los modelos multirresolución continuos no funcionan bien a la hora de representar el follaje de los árboles [DCSD02]: los troncos están modelados habitualmente mediante mallas continuas, pero las hojas se representan mediante un conjunto de polígonos aislados. Esta parte del árbol, compuesta por geometría dispersa, no presenta buenos resultados cuando se le aplica un método de simplificación estándar basado en contracción de aristas. No obstante, algunos modelos multirresolución especialmente desarrollados para esta parte del árbol han presentado su propio método de simplificación adaptado a la naturaleza del follaje, como los basados en colapso de hojas [RCB*] o los basados en la poda aleatoria de hojas [CH05].

A la hora de definir un modelo multirresolución, es necesario determinar una estructura de datos para almacenar la información y los algoritmos para acceder a estas estructuras, orientados a extraer la información necesaria de la forma más eficiente. Estos algoritmos seleccionan el nivel de detalle más adecuado en tiempo real, siguiendo un determinado criterio, como la distancia a la cámara o la importancia del objeto en la escena. Estos datos se envían a la unidad de procesamiento gráfico, GPU, para poder visualizar la geometría del nivel de detalle actual, cada vez que éste cambie. El tráfico de datos entre CPU y GPU puede, en estos casos, constituir un cuello de botella importante y producir, por tanto, un tiempo de visualización elevado.

Este trabajo propone un modelo multirresolución para el follaje basado en geometría, completamente implementado y almacenado en la GPU. De esta forma se obtiene un gran detalle al realizar tomas extremadamente cercanas del objeto. El beneficio directo de este modelo es la notable reducción en los tiempos de extracción del nivel de detalle requerido y su posterior visualización, debido a que la GPU puede realizar este trabajo y acelerarlo utilizando su estructura de procesadores en paralelo. Los datos extraídos no serán en en este caso transmitidos por el bus, debido a que también se generan en este hardware gráfico.

Para realizar el proceso de simplificación, el modelo presentado utiliza el concepto de poda para eliminar geometría innecesaria, según el trabajo presentado por Cook and Halstead [CH05]. En primer lugar, el follaje se divide en una nube 3D de celdas que contienen las hojas de la copa del árbol, asignadas según un criterio espacial. Posteriormente, el algoritmo de simplificación asigna un número aleatorio a cada una de las hojas de las celdas, que inducirá un orden de simplificación de las hojas. A continuación, se calcula la visibilidad de cada una de las celdas tridimensionales desde un conjunto de cámaras situadas alrededor del árbol. Esta visibilidad se almacena y, en tiempo real, los algoritmos de extracción de datos la consultan para extraer el nivel de detalle adecuado según la vista de cámara activa en ese momento. Esto implica que disminuir el nivel de detalle del follaje conlleve la eliminación de las partes menos importantes, según un determinado punto de vista. Además, como el objetivo es aprovechar el poder computacional de la GPU, el modelo multirresolución presentado posee algoritmos de extracción de datos paralelizables. De esta forma, la simplificación dependiente de la vista se realiza de forma eficiente y en tiempo real en la GPU.

La implementación se ha realizado en CUDA, una API que aprovecha los beneficios de la arquitectura de multiprocesadores de la GPU. CUDA hace posible manejar manualmente la memoria de este hardware gráfico, permitiendo asignar o liberar memoria, así como leer y escribir cualquier dirección de memoria, acciones no disponibles en la *pipeline* gráfica tradicional. Nuestro modelo aprovecha estas ventajas, lo que le permite acceder a los datos y escribirlos en otra dirección de memoria de forma eficiente.

## 2. Trabajo previo

Existe en la literatura, un número muy amplio de trabajos sobre la visualización de plantas en tiempo real. Como ya se ha dicho en la sección anterior, la investigación realizada puede clasificarse en dos grupos principalmente: trabajos que usan imágenes para representar estos objetos o trabajos que usan únicamente geometría, adaptando el número de polígonos usados para representar plantas a los requerimientos del hardware gráfico.

### 2.1. Visualización basada en imágenes

Este es uno de los métodos más habituales a la hora de representar árboles debido a su simplicidad. El uso de impostores son el ejemplo más popular de la visualización basada en imágenes. Este método sustituye la geometría de un objeto por una imagen de éste texturada sobre un polígono, que está totalmente inmerso en la escena. No obstante, aunque el método tiene muchas ventajas debida a la notable reducción de geometría, presenta algunos inconvenientes, como por ejemplo, la pérdida de realismo cuando la cámara está cerca del objeto.

Uno de los primeros trabajos presentados que usan esta técnica es el de Max [Max96]. El autor añade información de profundidad a las imágenes precalculadas, para así realizar un montaje adecuado de éstas en la escena. Posteriormente, en [MDK] aprovecha el hardware a la hora de representarlos. En este contexto, Shade et al. [SSHS98] y Chang et al. [CBL98] presentan las imágenes con profundidad por niveles, conocidos como LDI. Estos métodos visualizan objetos a partir de representaciones pre-calculadas basadas en pixeles, a las que se les añade información de profundidad desde diferentes puntos de vista. Esta información permite generar diferentes vistas de la escena a partir de las LDI almacenadas.

Otros autores trabajan con texturas volumétricas. Meyer

et al. [MN] convierten objetos complejos en texturas volumétricas con diferentes resoluciones antes de aplicarles la iluminación a la escena. Posteriormente, los mismos autores en 2001 [MNP01] y Reche et al. [RMD04] en 2004 obtienen imágenes 2D a partir de texturas volumétricas y las combinan dependiendo de la posición de la cámara. No obstante, estos métodos tienen el inconveniente típico: una baja calidad en las distancias cortas.

García et al. [GSSK05] resuelven el problema del paralaje producido por el uso de imágenes, usando impostores para representar grupos de hojas, que se combinan a la hora de representar el follaje. Esto aumenta la calidad y el detalle conseguido en las tomas cercanas de cámara. Este método fue mejorado en la siguiente versión, [GP08]. En los últimos años, han aparecido algunos trabajos que usan una técnica parecida, la nube de *billboards*, es decir, polígonos texturados orientados siempre al observador, como los presentados por Décoret et al. [DDSD03], Fuhrmann et al. [FMU05], Dylan et al. [LEST06] y Mantler et al. [MJW07]. También Speedtree [Spe09], una de las aplicaciones comerciales más populares actualmente en el campo de la visualización de árboles, implementan esta técnica.

Finalmente, uno de los más recientes trabajos es el presentado por Linz et al. [LRDM06]. Los autores estiman la opacidad en un volumen para generar y visualizar texturas dependientes de la vista, asociadas a las celdas en las que ha descompuesto el volumen inicial.

Debido a que el problema causado por la pérdida de realismo debido a la cercanía del observador al objeto es bastante difícil de solucionar, han aparecido algunos trabajos que combinan imágenes con geometría, como el presentado por Remolar et al. [RRCR04]. Los autores dividen la escena en zonas dependiendo de la distancia a la cámara. Los objetos en la zona lejana se presentan únicamente mediante impostores orientados al observador. A medida que se va acercando éste al objeto, se combinan ambas representaciones, hasta que la cercanía es tanta que el objeto se representa únicamente por geometría.

### 2.2. Visualización basada en geometría

La principal ventaja de este método es que no se produce la pérdida de realismo a medida que el observador se acerca al árbol, pero el elevado número de polígonos que forman el objeto hace necesario el uso de ciertas técnicas que posibiliten la visualización en tiempo real.

Remolar et al. [RCB*] disminuyen el número de hojas que forman la copa del árbol, aumentando el tamaño de las que permanecen. De esta forma, evitan que el árbol cambie de aspecto al tener menos hojas, evitando así el aspecto de poda. Otros trabajos publicados, cambian la primitiva de dibujado, utilizando puntos o líneas en lugar de triángulos. Es el caso de Weber and Penn [WP95] o Stamminger et al. [SD01]. Posteriormente, siguiendo esta línea de investigación, aparecen los trabajos de Deussen et al. [DCSD02] y Gilet et al. [GMN05] que permiten adaptar de forma interactiva y en tiempo real el número de puntos o líneas dependiendo de la importancia del objeto en la imagen final de la escena.

En los últimos años, han aparecido algunos trabajos basados en modelos multirresolución. Algunos basados únicamente en imágenes, como los presentados por Meyer et al. [MNP01] y Lluch et al. [LCV], y otros basados en geometría, como el presentado por Remolar et al. [RCRB03]. Basándose en la simplificación presentada en [RCB*], los autores hacen posible la reducción del número de polígonos que forman el follaje en tiempo real según un determinado criterio. Finalmente, en 2006 Rebollo et al. [RRCR06] [RRCG06] mejoraron este trabajo adaptando el modelo multirresolución al hardware gráfico.

## 3. Simplifación del follaje

La simplificación aplicada en nuestro modelo ha sido la llamada por los autores *Stochastic pruning*. Es una técnica de simplificación presentada por Cook y Halstead [CH05] para acelerar la visualización de objetos formados por geometría dispersa, como es el caso de los árboles y las plantas. Este tipo de objetos no producen buenos resultados al ser simplificada su apariencia mediante métodos generales de simplificación de superficies continuas. Debido a este hecho, *Stochastic pruning* se basa en la observación de que cuanto más lejos esté un objeto de la cámara, más primitivas se dibujan en un mismo píxel y, por tanto, se desperdicia tiempo de visualización en cálculos superfluos.

El objetivo de este método es descartar una gran cantidad de geometría innecesaria a la hora de visualizar el objeto. Los autores demuestran que el conjunto de geometría que permanece puede determinarse de forma aleatoria, dependiendo únicamente de la distancia a la cámara. El método se basa en asignar a cada elemento un número aleatorio, usando un muestreo por niveles preferiblemente, e interpretando esta numeración como el orden de simplificación. Ellos presentan la fórmula mostrada en la Ecuación 1 para determinar el número de elementos que permanecen, por ejemplo hojas o pequeñas ramas en el follaje. En la fórmula, identifican $z$ con la distancia a la cámara, $u$ como la cantidad de elementos que permanecen. El parámetro $h$ lo define el usuario. Este determina la agresividad de la simplificación en función de la distancia a la cámara.

$$u = z^{-log_h 2} \tag{1}$$

No obstante, simplemente eliminando geometría de los objetos produciría un cambio drástico en la apariencia de los árboles simplificados. Para evitar esto, *Stochastic pruning* mantiene también el área y el contraste en la imagen del

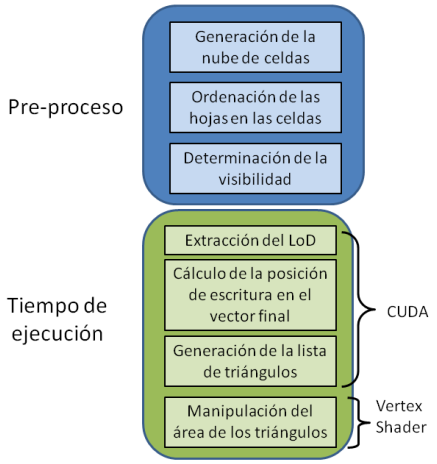


**Figura 1:** *Esquema del método presentado.*

objeto. Este objetivo lo consigue re-dimensionando las hojas que permanecen en el árbol y aplicándoles un color determinado, de forma que la imagen final del árbol simplificado difiera lo menos posible del original.

El modelo multirresolución presentado en este artículo se basa en esta simplificación. De esta forma, es posible descartar la geometría no visible desde un determinado punto de vista, y adaptarlo al hardware gráfico.

## 4. Descripción general del modelo

El presente trabajo propone un modelo multirresolución variable dependiente de la vista, adaptado a geometría dispersa, como el follaje de los árboles y totalmente implementado en GPU. Los pasos a seguir para construir el modelo se resumen en la Figura 1.

En primer lugar, en un pre-proceso, se distribuye una nube de celdas tridimensionales sobre la copa del árbol. Cada una de estas celdas acoge a un número determinado de hojas, de forma que todas las que forman el follaje estén asignadas a una celda. Posteriormente, le aplicamos el método de simplificación *Stochastic pruning* que, en este paso previo, únicamente le asignará a cada hoja de la celda un número aleatorio que constituirá el orden de simplificación de cada una de ellas. Las hojas de cada celda se ordenan en memoria según este número asignado. Finalmente, se sitúa alrededor de la copa un conjunto de cámaras, tal y como se muestra en la Figura 2. Desde cada una de las cámaras se valora la visibilidad de cada una de las celdas que forman el follaje. De esta forma, por cada celda se almacena un valor que determina la visibilidad de las hojas que contiene desde un punto de vista predeterminado. Esta acción se repite para todas las cámaras colocadas en el pre-proceso.

Mientras que estos pasos se realizan tanto en CPU como en GPU, el proceso de extracción en tiempo real de los datos geométricos que forman el nivel de detalle requerido, se realiza en su totalidad en GPU. Este proceso implica generar los triángulos necesarios para representar las hojas de cada celda en una determinada resolución, teniendo en cuenta la situación de la cámara en tiempo real. Cada una de las celdas distribuidas en el follaje es capaz de manejar la cantidad de detalle de la parte de la copa que contiene. Dada una determinada dirección de la vista, el número de las hojas visibles en cada celda se calcula interpolando los valores de visibilidad almacenados de las tres cámaras más cercanas a la real. Este proceso genera un valor de visibilidad que indica el número de hojas a visualizar finalmente.

Una vez obtenida por cada celda los índices resultantes que componen el nivel de detalle actual, éstos deben escribirse en una única lista para su posterior visualización. Cada una de las celdas deberá calcular la posición en esa lista de visualización donde debe empezar a escribir sus índices. Este proceso se aclara gráficamente en la Figura 5.

El proceso también modifica el tamaño de la geometría resultante para mantener la apariencia del objeto, según [CH05]. Esto se realiza en el *vertex shader* de forma eficiente.

Al haber distribuido el follaje en celdas independientes, es posible paralelizar los distintos cálculos de visibilidad de cada una de ellas y la posterior extracción de la geometría que forma el nivel de detalle. Esto aprovecha la estructura hardware de la GPU. Para sacar un mayor rendimiento, el modelo se ha implementado en CUDA. Esta API permite gestionar de forma eficiente todos los recursos de la GPU sin limitaciones de bus, y ofrece una interfaz para poder compartir datos y recursos con otras API's de visualización, como OpenGL o Direct3D.

Las siguientes secciones describen el proceso seguido con más detalle.

## 5. Pre-proceso

### 5.1. Generación de la nube de celdas

El primer paso en el pre-proceso es crear la nube de celdas tridimensionales sobre la copa del árbol. Es importarte generar las celdas teniendo en cuenta la distribución de las hojas y la forma de esta parte del árbol. El objetivo es maximizar la cantidad de hojas incluidas en cada celda y minimizar el número de celdas repartidas sobre el follaje, restringiendo el tamaño aproximado de cada una de ellas. Con esta finalidad, se ha utilizado el método presentado por Gottschalk et al. [GLM] que genera una nube de cajas envolventes orientadas, ajustándose lo más posible a la forma de la copa. Este método cumple con el objetivo de orientar las cajas para hacer que con el mínimo número de cajas todas las hojas estén incluidas en ellas.

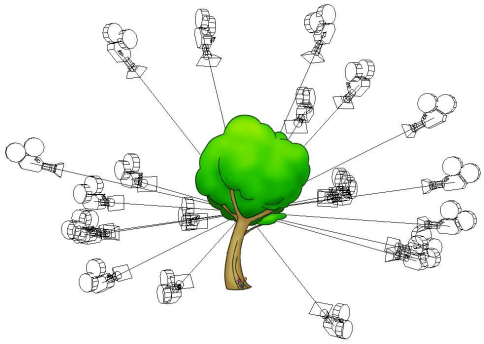

**Figura 2:** *Ejemplo de posicionamiento de cámaras en el pre-proceso.*

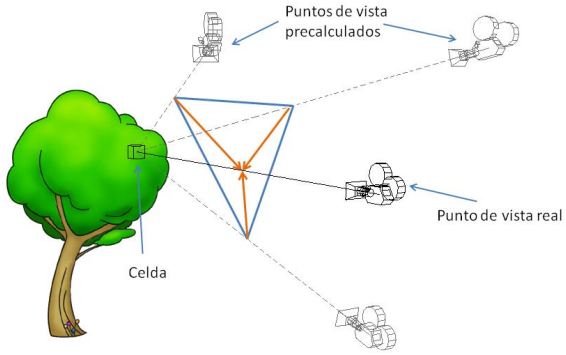


**Figura 3:** *El factor de visibilidad de un punto de vista se interpola, en tiempo real, a partir de la visibilidad almacenada de los tres puntos de vista más cercanos.*

### 5.2. Ordenación de las hojas en las celdas

Una vez las hojas están incluidas en celdas, éstas deben ordenarse según un orden determinado. El método *Stochastic pruning* es el encargado de asignar un número aleatorio a cada una de las hojas, que condicionará posteriormente su orden de simplificación. Una vez cada hoja tiene su número asignado, éstas se ordenan en una lista, almacenada en GPU.

### 5.3. Determinación de visibilidad

Una vez realizado los dos procesos anteriores, el siguiente paso es asignar a cada celda un índice de visibilidad de las hojas que contiene desde un conjunto de cámaras situadas alrededor de la copa. Estas cámaras se localizan fuera del follaje, apuntando al centro de su volumen envolvente. De esta forma, pueden obtener la figura completa de esta parte del árbol, tal y como se muestra en la Figura 2.

El factor de visibilidad es una función del tipo $vis(celda, pvista)$, la cual asocia a cada par celda/cámara un valor decimal en el rango $[0,1]$, que representa el número de hojas de la celda que son visibles desde el punto de vista analizado. La visibilidad de la celda se mide teniendo en cuenta el número de pixeles de las hojas que no están ocultas por el resto del follaje. El coste de esta parte del pre-proceso es $O(n_{celdas} * n_{pvista})$, siendo $n_{celdas}$ el número de celdas que forman la nube y $n_{pvista}$ el número de puntos de vista localizados alrededor del árbol. Nuestra implementación utiliza técnicas de oclusión por hardware, realizadas en la GPU, para obtener el número de pixeles que pasan el buffer de profundidad.

El siguiente pseudo-código clarifica el proceso seguido para calcular los valores de visibilidad.

Básicamente, para cada punto de vista el contenido de cada celda se visualiza cuatro veces. La primera y segunda pasada se usan para obtener el número de pixeles visibles desde el actual punto de vista, sin tener en cuenta la oclusión desde otras celdas. La primera se utiliza para inicializar el buffer de profundidad y la segunda para obtener el número de pixeles visibles ($totalPixels$). Las otras dos visualizaciones se utilizan para calcular el número de pixeles visibles desde el actual punto de vista: la tercera visualiza todas las celdas para actualizar el buffer de profundidad, y la última, visualiza de nuevo la celda actual para contar el número de pixeles que pasan el test de profundidad y, por tanto, equivale al número de pixeles que son visibles desde el punto de vista que estamos analizando, ($visPixels$). Finalmente, el factor de visibilidad para cada celda se calcula siguiendo la Ecuación 2.

**Algorithm 1** Cálculo de la visibilidad de las celdas en la GPU

$celdas \leftarrow ListadeCeldas$
$hojas \leftarrow ListadeHojas$
$pvista \leftarrow ListadePuntosDeVista$
**Ensure:** $visCelPvista$
**for all** $c \in celdas$ **do**
  **for all** $p \in pvista$ **do**
    $borraMemoria(color, profundidad)$
    $visualizaHojasCelda(c, p)$
    $resetQuery()$
    $visualizaHojasCelda(c, p)$
    $totalPixels \leftarrow queryVisualizado()$
    $visualizaTodasHojas(p)$
    $resetQuery()$
    $visualizaHojasCelda(c, p)$
    $visPixels \leftarrow queryVisualizado()$
    $visCelPvista[c, p] \leftarrow \frac{visPixels}{totalPixels}$
  **end for**
**end for**

$$vis(celda, pvista) = \frac{visPixels}{totalPixels} \in [0,1] \qquad (2)$$

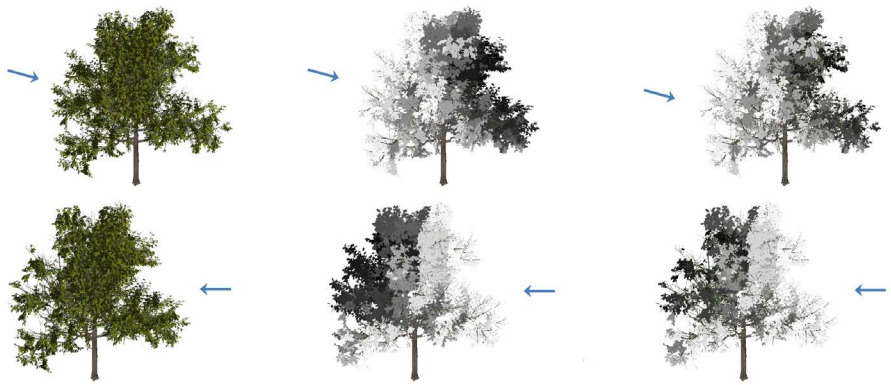

**Figura 4:** *Representación de la visibilidad de las hojas según la escala de grises: cuanto más blanco es el color de las hojas, más visibles son éstas. La flecha representa la dirección de la vista. Las columnas de la izquierda y derecha muestra el árbol con la simplificación aplicada desde el punto de vista indicado por la flecha. La columna central muestra la geometría al completo sin simplificar.*

La Figura 4 muestra los resultados del proceso de visualización. Cada línea representa un caso diferente. La visibilidad se muestra como una escala de grises: cuanto más oscuras son las hojas, menos visibles serán desde ese punto de vista. En la Figura, la dirección del punto de vista se muestra con una flecha.

## 6. Proceso de Extracción y Visualización

Una vez realizado el pre-proceso, esta sección describe los pasos realizados en tiempo real para obtener el nivel de detalle adecuado, según los parámetros existentes. Este proceso, se ha orientado a aprovechar al máximo el hardware gráfico de la GPU. Por esta razón, se distinguen tres fases bien diferenciadas. En primer lugar, se determinará el nivel de detalle a visualizar. A continuación, se genera la lista definitiva de los triángulos que finalmente se visualizarán, y finalmente se modifica el tamaño de los triángulos, en el caso de que sea necesario, para preservar la apariencia final del árbol.

### 6.1. Determinación del LoD

Con el fin de evitar visualizar toda la geometría que forma el follaje, este proceso calcula el nivel de detalle adecuado de cada celda, $LoD_{factor}$, en función del punto de vista y de su visibilidad. El nivel de detalle, por tanto, es una función del tipo mostrado en la Ecuación 3, donde el resultado depende tanto de la visibilidad de las celdas desde ese punto de vista, $vis(celda, pvista)$, como de la distancia del objeto a la cámara, $dist$.

$$LoD_{factor}(vis(celda, pvista), dist) \in [0,1] \qquad (3)$$

El factor de visibilidad real de cada celda, $vis(celda, pvista)$, se calcula teniendo en cuenta los factores de visibilidad calculados en el pre-proceso para las cámaras establecidas. En tiempo real, la estimación de la visibilidad de una celda se obtiene combinando linealmente la visibilidad almacenada de los tres puntos de vista más cercanos al actual. La Figura 3 muestra este proceso.

El hecho de influir en el nivel de detalle requerido el punto de vista actual, condiciona a que el modelo multirresolución creado sea variable, es decir, no tendrá la misma resolución en todas las zonas del follaje. No obstante, el modelo creado también podría variar la resolución de todo el objeto de forma uniforme. Para esto, simplemente sería necesario fijar $vis(celda, pvista)$ a 1, para cada una de las celdas. En ese caso, el nivel de detalle pasaría a depender únicamente de la distancia a la cámara.

Una vez fijada la visibilidad de la celda, se debe determinar el número de hojas que van a ser simplificadas. En este último paso se tiene en cuenta la distancia de la cámara a la celda. Con esta finalidad, se fijan dos distancias, una cercana, $dcerca$ a la cámara y otra lejana, $dlejos$. Cuando la distancia del árbol a la cámara, $d$, sea inferior a la fijada como cerca, $d \leq dcerca$, se mostrará el nivel máximo de detalle en cada celda. Cuando la distancia sea mayor a la fijada como lejos, $d \geq dlejos$, el detalle será mínimo. Entre las dos distancias se aplica una reducción lineal.

Finalmente, se obtiene para cada celda un factor de nivel de detalle, $LoD_{factor}$, que ha sido calculado en paralelo en la GPU. Este factor se almacena para cada celda en una zona de la memoria de video. El coste de almacenaje es $O(n_{celdas})$, siendo $n_{celdas}$ en número total de celdas.

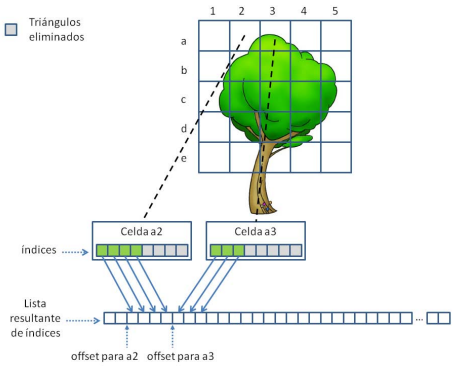


**Figura 5:** *Ejemplo de particionamento del follaje en celdas, generación de índices y cálculo de la posición de escritura en la lista final de triángulos.*

### 6.2. Generación de la lista de triángulos

El objetivo de este proceso es generar una lista de triángulos que aproximan la geometría contenida en cada celda, dependiendo del factor de nivel de detalle calculado previamente. Este valor $LoD_{factor}$ determina el número de hojas que serán visualizadas de cada celda para el actual punto de vista. Debido a que las hojas han sido ordenadas en un preproceso según el número estocástico que se le había asignado, generar las que forman un determinado nivel de detalle se reduce simplemente a copiar las primeras $n_{hojasCelda} \cdot LoD_{factor}$ hojas de cada celda, siendo $n_{hojasCelda}$ la cantidad de hojas contenidas originalmente en la celda.

El proceso de extracción implica generar una lista de todas las hojas que representan el nivel de detalle de la copa para un determinado punto de vista. No obstante, debido a que cada celda genera simultáneamente y en paralelo los índices de sus hojas visibles, es necesario que cada una conozca la posición de la lista de hojas donde debe escribir sus datos. Este proceso conlleva consultar el factor de nivel de detalle calculado previamente para cada celda. Con esta finalidad, los algoritmos de extracción toman como entrada la lista de $LoD_{factor}$ almacenados para las celdas y, según el número de hojas que va a escribir cada celda en la lista final, calcula la posición de comienzo de escritura de la siguiente celda, acumulando las hojas ya escritas. La Figura 5 clarifica este proceso.

Una vez calculada las posiciones de escritura en la lista final, el algoritmo está listo para empezar a generar índices. A medida que cada celda va escribiendo sus datos en esta lista final, se incrementa una variable global, almacenada en memoria, que va acumulando la cantidad de índices generados. Esto es necesario para que finalmente la GPU conozca el número total de hojas a visualizar.

### 6.3. Preservación del área visual proyectada

En tiempo de ejecución, los algoritmos diseñados aplican algunas estrategias con el fin de reducir el cambio visual producido al eliminar las hojas menos relevantes de cada celda. La estrategia usada es modificar el área visualizada de la geometría que prevalece, de forma que la apariencia visual sea lo más parecida posible al objeto original.

Como se explica en el trabajo de Cook y Halstead, el área total de las hojas que forman el follaje puede expresarse mediante la Ecuación 4, donde $a$ es el área media de cada hoja individual y $n$ es la cantidad de hojas visualizadas.

$$area_{total} = na \tag{4}$$

No obstante, cuando se le aplica simplificación, el área total pasa a ser *nau* (Ecuación 1). Debido a que la simplificación disminuye esa área total, ésta debe ser compensada para mantener la concordancia entre niveles de detalle. En ese caso, los elementos que no han sido simplificados, se escalan por el factor $s$, resultando

$$area_{total} = naus, s = 1/u \tag{5}$$

En la práctica, este proceso se realiza en el *vertex shader* y tiene un coste de visualización casi despreciable.

A diferencia de [CH05], en el presente trabajo no se realiza un análisis por cada píxel de las hojas que prevalecen. Debido a que el método presentado en [CH05] no fue pensado para ejecutarse en tiempo real, éste usa una representación más compleja de la geometría que forman las hojas. En su caso, es necesario adaptar el color de las hojas que forman el nivel de detalle. Debido a que en nuestro método la hoja se representa mediante una imagen texturada sobre un polígono, la textura aplicada a esos polígonos una vez se han escalado actúa como mecanismo de preservación del color por sí mismo. La Figura 6 y 7 muestran cómo el contraste de las imágenes se mantiene en los diversos niveles de detalle obtenidos.

## 7. Resultados

Esta sección chequea el modelo multirresolución presentado en este artículo con algunos tests prácticos. Con esta finalidad, se ha configurado un entorno experimental. Todos los test han sido ejecutados en una máquina Athlon64 3500+ con 3GB de RAM y una tarjeta gráfica GeForce 8800GT. Los modelos utilizados en los test se muestran en la Tabla 1.

A la hora de mostrar los resultados de nuestro método, se ha optado por dos opciones: mostrar resultados visuales y de tiempos de extracción.

Una ventaja de los métodos basados en geometría es la

**Figura 6:** *Imágenes comparativas de la calidad de la simplificación con el árbol Quercus Cerris. La fila superior muestra un árbol a diferentes distancias de la cámara, con el máximo detalle. La fila inferior muestra el mismo árbol a las mismas distancias, pero con las simplificaciones debidas a la dependencia de la vista y la distancia a la cámara.*

| Árbol | Triángulos |
|---|---|
| Olea europaea | 32,214 |
| Fraxinus ornus | 57,290 |
| Quercus cerris | 82,310 |
| Cedrus atlantica | 262,084 |

**Tabla 1:** *Complejidad de los árboles utilizados en los experimentos.*

gran calidad de imagen obtenida en vistas cercanas. Nuestro método, debido a ser variable y dependiente de la vista, ofrece esta ventaja al mismo tiempo que reduce la cantidad de polígonos visualizados.

Por otro lado, la Figura 6 y la Figura 7 muestran una comparativa de la calidad de la simplificación de nuestro método. El nivel de detalle del follaje depende de dos criterios: el factor de visibilidad dependiente de la vista y una función lineal de la distancia a la cámara del objeto. Estos dos factores se combinan para producir el actual $LoD_{factor}$, que determina la cantidad de hojas a ser eliminadas de cada celda del árbol. Estas figuras comparan el modelo original y el simplificado en las mismas condiciones, es decir, con la misma distancia a la cámara y la misma orientación.

En cuanto a los tiempos de extracción, la Tabla 2 muestra el tiempo de extracción de varios niveles de detalle para cuatro tipos diferentes de árboles, al reducir su complejidad al 66%, 33% y 15%. La tabla muestra los tiempos en milisegundos. Cada medida implica el tiempo de asignar memoria para los *vertex buffers* almacenados en hardware a CUDA, la ejecución en el núcleo de CUDA, el cuál extrae el nivel de detalle, y el tiempo de liberar esa memoria más tarde. Además, cada medida muestra, en la tabla, un número entre paréntesis que indica el tiempo de extracción de los índices para el nuevo nivel de detalle sin tener en cuenta el tiempo consumido por CUDA para asignar y liberar la memoria del hardware.

| Árbol | 66% | 33% | 15% |
|---|---|---|---|
| Olea europaea | 1.48(0.54) | 1.41(0.50) | 1.51(0.47) |
| Fraxinus ornus | 1.96(0.80) | 1.71(0.61) | 1.54(0.62) |
| Quercus cerris | 1.63(0.83) | 1.42(0.65) | 1.12(0.53) |
| Cedrus atlantica | 2.86(2.23) | 2.29(1.53) | 1.75(1.12) |

**Tabla 2:** *Tiempos de extracción del nivel de detalle en milisegundos para diferentes árboles. Los números entre paréntesis se refieren al tiempo de ejecución que CUDA tarda en generar el nivel de detalle deseado. Los números fuera del paréntesis se refieren al tiempo total incluyendo el tiempo de mapeado y consulta de los buffers.*

Con el fin de demostrar los beneficios de utilizar la GPU durante el proceso de extracción del nivel de detalle en lugar de la CPU, mostramos el gráfico de la Figura 8. Esta figura compara los tiempos de extracción de reducir un árbol a un 30% de su complejidad geométrica mediante dos métodos, uno basado en CPU, como el trabajo de Rebollo et al. [RRCR06], y nuestro método, basado en GPU. En la Figura se ve claramente que, debido a que nuestro modelo

**Figura 7:** *La fila superior muestra el árbol Cedrus Atlántica a diferentes distancias, visualizado con el máximo detalle. La fila inferior muestra el mismo árbol a la misma distancia que el superior, al que se le ha aplicado la reducción dependiente de la vista y la distancia con las siguientes reducciones : 70%, 50%, 35%, 20% y 15% respectivamente.*

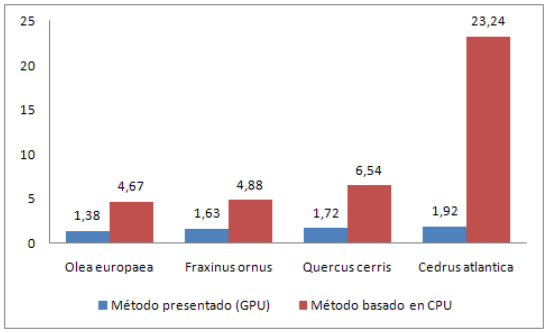


**Figura 8:** *Comparación de los tiempos de extracción del nivel de detalle con un modelo multirresolución geométrico que realiza el proceso de extracción en CPU [RRCR06].*

utiliza aceleración hardware y procesamiento en paralelo, los tiempos de extracción obtenidos con nuestro método mejoran ampliamente los obtenidos con el método basado en CPU.

Es importante destacar que en la Tabla 2 (los números entre paréntesis) la diferencia entre el tiempo necesario para ejecutar los procesos de CUDA en la GPU (generar el nivel de detalle requerido) y el tiempo total incluyendo el mapeado y liberación de la memoria en hardware para ser utilizado por CUDA.

## 8. Conclusiones

Este artículo presenta un modelo multirresolución variable para la copa de los árboles, dependiente de la vista y cuyos algoritmos de extracción y visualización se ejecutan totalmente en la GPU. El modelo utiliza para reducir el número de primitivas que forman este tipo de objetos, el trabajo sobre simplificación de primitivas dispersas *Stochastic pruning*. Mediante esta técnica se seleccionan en tiempo real los triángulos que forman el nivel de detalle adecuado, según la posición de la cámara y la distancia a ésta. Mediante esta técnica de simplificación es posible descartar una gran cantidad de geometría mientras se mantiene la apariencia de la figura del árbol.

El modelo multirresolución creado utiliza la GPU tanto para almacenar los datos como para gestionar la extracción y visualización de los datos geométricos que forman el nivel de detalle adecuado. Con esta finalidad, la implementación final del modelo se ha realizado usando CUDA. Esta API permite el acceso a cualquier dirección de la memoria de video de la GPU, por lo que no es necesario almacenar los datos en CPU, como se hacía de forma habitual. Esto permite evitar el tráfico de datos entre estas dos partes del ordenador, típico de los modelos multirresolución tradicionales. También nos permite liberar la CPU de la tarea de extracción del nivel de detalle, trasladando esta tarea a la GPU. Este hardware gráfico ofrece la ventaja de poder realizar las operaciones necesarias para extraer la geometría, en paralelo, sacando provecho a los multiprocesadores que la GPU posee. Además, otra ventaja de realizar los procesos en este hardware, es la escalabilidad de estos procesos, te-

niendo en cuenta que la velocidad a la cual la GPU ejecuta estos procesos es muy elevada, comparándola con la que trabaja la CPU.

Una línea de investigación en la cual estamos trabajando es reducir los saltos visuales debidos a la simplificación de hojas, realizada en un cambio de nivel de detalle. Este efecto es todavía visible, aunque ocurra en las partes menos visibles del árbol debido a la dependencia de la vista de nuestro modelo multirresolución.

Como trabajo futuro, proponemos usar la nueva API standard OpenCL en lugar de CUDA para una mejor gestión de los detalles. Esto aumentaría el rendimiento porque, en la práctica, compartir un recurso entre CUDA y las API gráficas (OpenGL o Direct3D) implica gestionar una copia interna en memoria de video. La gestión de la memoria de video es actualmente muy eficiente debido al gran ancho de banda, el cual permite crear rápidamente copias internas. No obstante, esta tarea genera un disminución en el rendimiento de los procesos desarrolladas en este hardware gráfico. El siguiente estándar en computación general en la GPU, OpenCL, mejorará esta tarea y ofrecerá una gestión óptima de los recursos compartidos en la GPU, sin la necesidad de realizar copias internas. Esto aumentará, a su vez, el rendimiento del método presentado en este artículo.

## Agradecimientos

Este trabajo ha sido subvencionado por el MCT (beca TSI-2004-02940 y proyectos TIN2007-68066-C04-02 y TIN2007-68066-C04-01) y por Bancaja (P1 1B2007-56).

## References

[CBL98] CHANG C., BISHOP G., LASTRA A.: *LDI Tree: A Hierarchical Representation for Image-based Rendering*. Tech. rep., 1998.

[CH05] COOK R. L., HALSTEAD J.: Stochastic pruning. *Proceedings of Eurographics Workshop on Natural Phenomena* (2005).

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *VIS '02: Proceedings of the conference on Visualization '02* (2002), pp. 219–226.

[DDSD03] DÉCORET X., DURAND F., SILLION F., DORSEY J.: Billboard clouds for extreme model simplification. In *Proceedings of the ACM Siggraph* (2003), ACM Press.

[FMU05] FUHRMANNAND A., MANTLER S., UMLAUF E.: Extreme model simplification for forest rendering.

[GLM] GOTTSCHALK S., LIN M. C., MANOCHA D.: Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics*, Annual Conference Series, 171–180.

[GMN05] GILET G., MEYER A., NEYRET F.: Point-based rendering of trees. In *Eurographics Workshop on Natural Phenomena* (2005), E. Galin P. P., (Ed.).

[GP08] GARCÍA I., PATOW G.: Igt: inverse geometric textures. *ACM Transaction Graphics 27*, 5 (2008), 1–9.

[GSSK05] GARCÍA I., SBERT M., SZIRMAY-KALOS L.: Leaf cluster impostors for tree rendering with parallax. In *Proc. Eurographics 2005 (Short Presentations)* (2005), Eurographics.

[LCV] LLUCH J., CAMAHORT E., VIVO R.: Procedural multiresolution for plant and tree rendering. In *AFRIGRAPH '03:*.

[LEST06] LACEWELL J. D., EDWARDS D., SHIRLEY P., THOMPSON W. B.: Stochastic billboard clouds for interactive foliage rendering. *Journal of graphics tools 11*, 1 (2006), 1–12.

[LRDM06] LINZ C., RECHE A., DRETTAKIS G., MAGNOR M.: Effective multi-resolution rendering and texture compression for captured volumetric trees. In *Proceedings of the Eurographics Workshop on Natural Phenomena* (2006), Eurographics, (Ed.).

[Max96] MAX N.: Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Rendering Techniques '96, Proceedings of the Eurographics Workshop* (1996), pp. 165–174.

[MDK] MAX N., DEUSSEN O., KEATING B.: Hierarchical image-based rendering using texture mapping hardware. In *Rendering Techniques '99, Proceedings of the Eurographics Workshop*.

[MJW07] MANTLER S., JESCHKE S., WIMMER M.: Displacement mapped billboard clouds. In *Proceedings of Symposium on Interactive 3D Graphics and Games* (2007).

[MN] MEYER A., NEYRET F.: Interactive volumetric textures. In *Eurographics Rendering Workshop 1998*.

[MNP01] MEYER A., NEYRET F., POULIN P.: Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering* (2001), Springer-Verlag.

[RCB*] REMOLAR I., CHOVER M., BELMONTE O., RIBELLES J., REBOLLO C.: Geometric simplification of foliage.

[RCRB03] REMOLAR I., CHOVER M., RIBELLES J., BELMONTE O.: View-dependent multiresolution model for foliage. *Journal of WSCG'03 11*, 2 (2003), 370–378.

[RMD04] RECHE A., MARTIN I., DRETTAKIS G.: Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings) 23*, 3 (2004), 720–727.

[RRCG06] REBOLLO C., REMOLAR I., CHOVER M., GUMBAU J.: Hardware-oriented visualisation of trees. In *Lecture Notes in Computer Science vol. 4263/2006* (2006), pp. 374–383.

[RRCR04] REMOLAR I., REBOLLO C., CHOVER M., RIBELLES J.: Real time tree rendering. In *Lecture Notes in Computational Science 3039* (2004), pp. 173–180.

[RRCR06] REBOLLO C., REMOLAR I., CHOVER M., RIPOLLÉS O.: An efficient continuous level of detail model for foliage. In *Proc. of 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2006)* (2006), UNION agency, pp. 335–342.

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), pp. 151–162.

[Spe09] Speedtree, interactive data visualization inc. `http://www.idvinc.com/speedtree/`, 2009.

[SSHS98] SHADE J., S.GORTLER, HE L., SZELISKI R.: Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), ACM Press, pp. 231–242.

[WP95] WEBER J., PENN J.: Creation and rendering of realistic trees. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), Cook R., (Ed.), ACM Press, pp. 119–128.