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

Un sistema de caché para la visualización interactiva de campos de luz de alta resolución

Miguel Escrivá, Joan Blasco, Francisco Abad, Emilio Camahort, Roberto Vivó

Instituto Universitario de Automática e Informática Industrial
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia, Spain
{mescriva, jblascoi, fjabad, camahort, rvivo}@dsic.upv.es

Abstract

Los modelos de visualización basados en campos de luz almacenan valores de radiancia muestreados a lo largo de un conjunto de lineas orientadas en un espacio 4-dimensional. Esta elevada dimensionalidad junto al hecho de que estos modelos se suelen basar en un muestreo denso del campo de luz hace imposible la carga de un modelo completo directamente en memoria principal o memoria de la GPU para su visualización. Así pues, dicha información debe ser almacenada en memoria secundaria y se hace necesario un mecanismo adecuado para transferir esa gran cantidad de información desde la memoria secundaria hasta la memoria de la GPU a fin de visualizar estos modelos.

En este trabajo presentamos un estudio teórico para la construcción de una caché orientada a la carga de la información de radiancia para campos de luz, junto a los resultados preliminares de su implementación. Esta caché se basa en una fragmentación de las imágenes que forman el conjunto de datos original para, utilizando un esquema de carga bajo demanda, conseguir maximizar el uso de los recursos disponibles y mejorar la velocidad de visualización de múltiples campos de luz. Esta aproximación permite además su combinación con técnicas de multirresolución que se suman para conseguir una mayor aceleración de la visualización.

Categories and Subject Descriptors (according to ACM CCS):
Computer Graphics [I.3.3]: Picture/Image Generation–Display algorithms—
Computer Graphics [I.3.5]: Computational Geometry and Object Modeling–Curve, surface, solid, and object representations—
Computer Graphics [I.3.6]: Methodology and Techniques–Graphics data structures and data types—

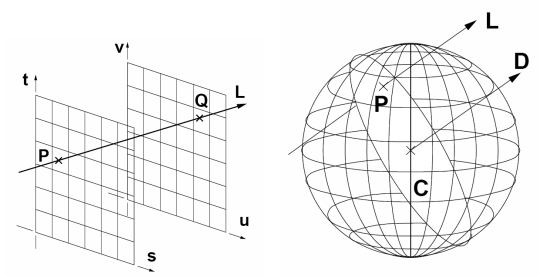
1. Introducción

Los campos de luz son una técnica de representación de gráficos por computador que pertenece al conjunto de los métodos de modelado y visualización basados en imagen [LH96,GGSC96,CLF98,PPS97]. Los campos de luz están basados en la definición de la función plenóptica [AB91].

Como tal se caracteriza por ser capaz de visualizar objetos tridimensionales utilizando para ello la información de radiancia capturada desde una escena, ya sea real o sintética. Esta forma de visualización es alternativa a la tradicional en la que se usa principalmente información geométrica. Así, el aspecto tridimensional de la visualización se consigue a partir de la utilización de un conjunto de imágenes correspondientes a diferentes puntos de vista, que son, por regla general, capturadas previamente a la visualización.

Debido precisamente a que se basan en la información de radiancia almacenada en imágenes, las representaciones basadas en campos de luz hacen que la complejidad de la representación resulte independiente de la complejidad de la escena representada. Dicha ventaja se hace manifiesta a la hora de visualizar escenas con un alto grado de complejidad, ya sea geométrica o del modelo de iluminación. La figura 1 muestra dos parametrizaciones típicas del campo de luz. Dichas parametrizaciones establecen básicamente la forma de

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/075-084



(a) Parametrización planar. (b) Parametrización esférica.

Figura 1: *Dos parametrizaciones de las líneas que forman el soporte del campo de luz.*

capturar la radiancia de una escena. En la sección 2 se presentarán los trabajos donde se puede encontrar una descripción detallada de dichas parametrizaciones.

Este hecho, junto a la facilidad con la que se puede realizar la captura de los datos necesarios para la visualización, ha suscitado un interés notable por este tipo de técnicas. Dependiendo de los requisitos de la aplicación, la captura de los modelos presenta un tipo de problema distinto. Por ejemplo, cuando el campo de luz se genera a partir de modelos sintéticos, la captura puede tener costes temporales altos dependiendo de la complejidad de visualizar cada muestra. Por otro lado, la captura de campos de luz de objetos reales requiere normalmente la obtención precisa de diferentes imágenes desde diferentes puntos de vista, que varían según la parametrización elegida.

Por otro lado, la complejidad de un campo de luz viene determinada por el muestreo realizado durante su captura. Así, dependiendo del modelo de visualización elegido y del muestreo realizado en la adquisición, podemos hablar de representaciones con distintos grados de resolución, tanto direccional como espacial, que en definitiva determinará la complejidad del modelo.

En nuestro caso nos centramos en representaciones uniformes con un muestreo denso del campo de luz. Dichas representaciones pueden contener miles de imágenes con una resolución espacial alta, adquiriendo el conjunto de datos del modelo de campo de luz un tamaño considerable. Esto nos lleva a buscar métodos eficientes para el manejo y la carga de las imágenes, ya que la cantidad de datos que forman el campo de luz puede exceder con mucho las capacidades de la memoria principal y, todavía más, las del hardware gráfico. Así pues se hace necesario el uso de sistemas avanzados de caché que resuelvan este problema.

En este trabajo presentaremos los primeros resultados de la implementación de un sistema de caché preparado para el manejo de la información utilizada por un sistema de visualización de campos de luz.

En la siguiente sección hacemos un breve repaso de la investigación sobre los campos de luz, recordando algunas de sus implementaciones más destacadas y resumiendo avances en la materia. Así mismo realizamos una revisión de algoritmos de caché útiles para establecer las bases de nuestro propio sistema para campos de luz.

Tras esto pasaremos a detallar el problema de la gran cantidad de información que almacenan los campos de luz, para a continuación analizar los requisitos y presentar un estudio para la construcción de la caché. Acto seguido, en la sección 5, explicaremos cómo se ha de modificar el algoritmo de dibujado para poder trabajar con la caché y finalmente mostraremos los primeros resultados obtenidos.

2. Antecedentes

Como es sabido, en los sistemas informáticos es habitual la aplicación de técnicas de gestión de la memoria basadas en cachés. Esto es debido a las limitaciones en el espacio disponible en las memorias más rápidas frente a la capacidad de almacenamiento de la memoria secundaria, más lenta.

En el presente trabajo nos centraremos en la aplicación de estas técnicas sobre campos de luz esféricos. Estos utilizan una parametrización distinta a otros modelos como los campos de luz planares propuestos inicialmente por Levoy y Hanrahan [LH96] o Gortler y otros [GGSC96]. En dichos trabajos se utilizan imágenes del objeto a representar tomadas desde varios puntos situados sobre un plano y se repite este proceso seis veces para formar un cubo que envuelva el objeto. En la parametrización esférica propuesta por Camahort y otros [CLF98] se utiliza una disposición esférica de los puntos de vista alrededor del objeto que evita los artificios creados en las intersecciones entre planos. En esta parametrización se suele utilizar un mecanismo de multirresolución en el que los conjuntos de imágenes usados poseen tamaños desde 20 hasta 20480 imágenes.

Varios autores han propuesto otras parametrizaciones alternativas como por ejemplo los campos de luz de superficie [WAA*00, CBCG02], que usan como superficie de muestreo una que aproxime el objeto con precisión, muy similar a la técnica llamada *view-dependent texture mapping* [DTM96]; o técnicas basadas en modelos geométricos aproximados [BBM*01], cierto tipo de información de profundidad [SVSG01] o una medida del enfoque de los puntos del objeto [TN05].

La necesidad de un sistema de caché para acelerar la carga de imágenes se presenta de forma natural, tanto en el caso de la visualización de campos de luz como en el ámbito de los gráficos por computador en general. Así, los algoritmos de caché eficientes se han convertido en una herramienta fundamental para incrementar la rapidez de acceso a la información en cualquier tipo de sistema. Sin embargo, la conveniencia de usar un algoritmo u otro depende en última instancia de los patrones de acceso a la memoria.

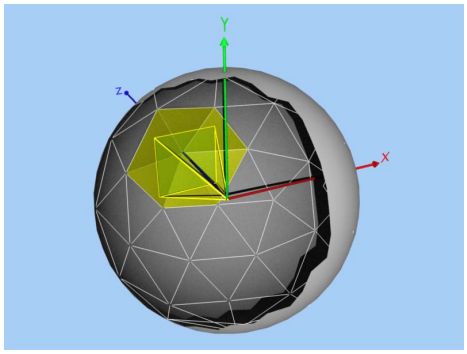
En la actualidad existen diferentes algoritmos para mantener memorias caché que alcanzan resultados más o menos satisfactorios en función de estos patrones [LChK*96, MM04, QJP*07]. Estos resultados se evalúan habitualmente en virtud de los aciertos que logre la política de reemplazo utilizada y el impacto adicional que implican los cálculos realizados sobre el tiempo de carga normal. A este respecto, se han utilizado ampliamente las políticas de tipo LRU (*Least Recently Used*) ya que han sido las que permitían unos mejores resultados con un coste muy bajo de procesamiento. Las políticas LRU se caracterizan por adaptarse rápidamente a cambios en los patrones de referenciación de los bloques de memoria, ya que se basan en una cantidad de información reducida (la última vez que el bloque fue accedido). Esto sin embargo provoca que los accesos que realizan barridos sufran grandes penalizaciones.

Las políticas LFU(*Least Frequently Used*) mejoran los resultados de las anteriores, pero padecen el problema de que cuando un bloque deja la caché se pierde la información histórica sobre éste.

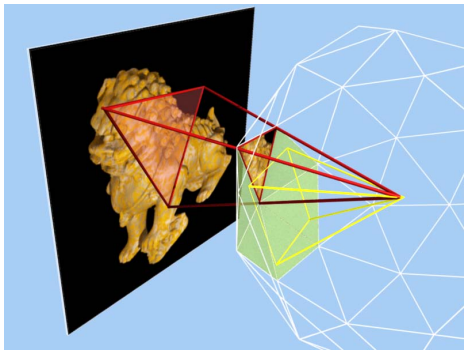
En la actualidad la atención parece centrada en los métodos basados en distintas combinaciones de políticas de los dos tipos anteriores [LChK*96, MM04]. Un ejemplo de estos es el algoritmo ARC (*Adaptive Replacement Cache*) de Megiddo y Dharmendra [MM04]. Dicho algoritmo es similar a los algoritmos LRU en tanto a que es de implementación sencilla y no necesita mucho procesamiento adicional, sin embargo es resistente a las penalizaciones por accesos de barrido.

El estudio de técnicas adecuadas para la gestión de la memoria se tuvo en cuenta ya durante los primeros desarrollos sobre campos de luz [LH96, GGSC96]. Sin embargo, los objetivos y las condiciones planteadas en aquellos trabajos diferían de las actuales. Aparte del cambio en la capacidad y las arquitecturas de memoria principal y hardware gráfico, el planteamiento de representar múltiples campos de luz de manera simultánea supone un cambio significativo en la estrategia a seguir para la carga y reemplazo de las imágenes cargadas. Estos factores hacen que surja la necesidad de experimentar con nuevas aproximaciones para la gestión de la información del campo de luz que permitan incrementar el número de campos de luz visualizados a velocidades interactivas.

El incremento del nivel de detalle deseado en aplicaciones gráficas siempre va en aumento, y han aparecido técnicas relacionadas con la virtualización de texturas como los atlas de texturas [NVI04], los clipmaps [TMJ98] u otros sistemas de gestión alternativos [LDN04], que a partir de la fragmentación de texturas grandes permiten aumentar el nivel de detalle sin incrementar en exceso el coste asociado a ello.



(a) El volumen de la vista se sitúa dentro de la esfera teselada seleccionando los triángulos visibles (en amarillo).



(b) Las coordenadas de textura de cada triángulo se obtienen proyectando sus vértices sobre el plano donde está la imagen del campo de luz (en rojo).

Figura 2: *Algoritmo de visualización de un campo esférico.*

3. Modelado y visualización de campos de luz

El sistema utilizado para modelar y visualizar campos de luz ha sido presentado ya anteriormente [BEA*08, EBA*08]. Este sistema abstrae las características comunes a varios tipos de campo de luz. En este caso, hemos basado el desarrollo del presente trabajo sobre la parametrización esférica, ya que esta ofrece una representación que no depende de los parámetros de la cámara.

3.1. Modelo de representación esférico

En esta parametrización los rayos vienen dados por su dirección $D = (\theta, \phi)$ y por su intersección $P = (u, v)$ con un plano ortogonal a D (ver figura 1(b)). Para generar un campo de luz a partir de un modelo geométrico se muestrea la radiancia de las líneas que atraviesan la esfera unidad, dentro de la que se sitúa debidamente escalado el modelo, obteniéndose así las imágenes del campo de luz [CLF98].

Estas líneas se determinan como una discretización del conjunto de las direcciones (θ, ϕ) del espacio 3D. A partir

de puntos sobre la esfera unidad, se triangula ésta tomando como muestras direccionales los centros de los triángulos que la forman. Así, cada muestra aproxima el haz de direcciones que, partiendo del centro de la esfera, atraviesa la superficie del triángulo.

Para muestrear las líneas del soporte del campo de luz se discretizan todas las líneas que son paralelas a la muestra direccional e intersectan con la esfera. En la figura 1(b) estas líneas son las que atraviesan el círculo C. En la práctica, las radiancias del campo de luz se obtienen mediante el cálculo de una proyección ortográfica del objeto a lo largo de cada muestra direccional. Cada imagen así obtenida se asocia al haz de direcciones que aproxima la muestra y se guarda como una textura asociada al triángulo que la representa.

Respecto al algoritmo de visualización utilizado, se utiliza actualmente una adaptación del algoritmo del Lumigraph [GGSC96]. Este algoritmo está ilustrado en la figura 2. Inicialmente, se coloca el volumen de la vista dentro de la esfera teselada y se calculan los haces o *pencils* que intersectan con él (si es necesario se escalan la esfera o el volumen quedando ambos centrados en el origen). Los haces intersectados corresponden a los triángulos de la esfera visibles en esa vista (ver figura 2(a)). A continuación, cada triángulo se dibuja texturado con la imagen asociada a dicho triángulo en el campo de luz. Las coordenadas de textura se calculan proyectando sus vértices sobre el plano soporte de la textura (en rojo en la figura 2(b)). La figura 3 muestra un campo de luz visualizado de este modo con los triángulos dibujados en alámbrico sobre la imagen.

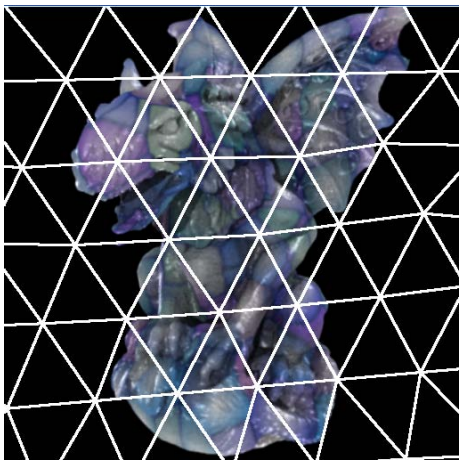


Figura 3: *Visualización de un campo de luz. Superpuesto se pueden apreciar los* pencils *necesarios para visualizarlo con 1280 direcciones y una anchura de campo de* 45^o.

4. Fragmentación de las imágenes del campo de luz

Consideremos, por ejemplo, un campo de luz con una resolución de imagen de 512×512 y 20480 direcciones. Necesitaríamos unos 40GB de capacidad para almacenar este campo de luz si no utilizáramos compresión. Utilizando un algoritmo de compresión sin pérdida podríamos obtener un *ratio* de compresión de 10:1 (asumiendo que la mayoría de las imágenes contienen una considerable cantidad de píxeles de fondo). Con un algoritmo de compresión con pérdida podríamos comprimir mucho más las imágenes, a costa de perder calidad en el resultado, llegando a un *ratio* de 40:1 dependiendo de la cantidad de píxeles de fondo.

Nivel	% Área
0	53,61 %
1	17,74 %
2	5,28 %
3	1,41 %
4	0,37 %
5	0,10 %

Tabla 1: *Porcentaje aproximado de las imágenes que se utiliza al dibujar un* pencil *del campo de luz.*

A fin de gestionar la memoria requerida para obtener una visualización de conjuntos de datos de este tamaño, se hace necesaria una solución de cache y/o compresión eficientes. En [EBA*07] se describe un algoritmo de cache predictivo que mantiene en memoria las imágenes que se van a utilizar para visualizar el campo de luz en los siguientes frames.

Sin embargo, cuando este algoritmo carga una imagen del campo de luz, ésta se mantiene completa en memoria, a pesar de que en la práctica sólo se requiere una porción reducida de dicha imagen para la visualización. En la tabla 1 puede verse el porcentaje aproximado de cada imagen que se utiliza para dibujar un *pencil* en función del nivel de detalle en el que se visualizará el campo de luz. Este porcentaje varía también dependiendo de la distancia a la que esté situada la cámara. Los valores de la tabla 1 se han calculado para una distancia de la cámara al campo de luz que permita visualizar el campo de luz completamente dejando el mínimo espacio sobrante en la pantalla, como ocurre en la figura 3.

Como puede apreciarse en la tabla 1, el porcentaje que se utiliza de cada una de las imágenes es muy pequeño. Nuestra intención es visualizar campos de luz de grandes dimensiones, en los que se acrecentaría notablemente la cantidad de información superflua cargada en memoria. Según los resultados mostrados en la tabla 1 podemos ver que con un porcentaje de utilización de entre un 1.41 % y un 5.28 % para cada imagen, el sistema de caché de imágenes completas no resuelve de manera eficiente el problema de carga durante la visualización de campos de luz.

La solución que proponemos es fraccionar las imágenes que componen el campo de luz en varias subimágenes (*tiles*) de tamaño más pequeño y adecuado para visualizar el campo de luz. Por otra parte, se plantea una arquitectura distinta del sistema de caché que permitirá mejorar el aprovechamiento

nCL/M	256	512	1024	2048	4096
Bpp = 4					
1	0,3125	1,25	5	20	80
2	0,625	2,5	10	40	160
3	0,9375	3,75	15	60	240
4	1,25	5	20	80	320
Bpp = 6					
1	0,46875	1,875	7,5	30	120
2	0,9375	3,75	15	60	240
3	1,40625	5,625	22,5	90	360
4	1,875	7,5	30	120	480
Bpp = 8					
1	0,625	2,5	10	40	160
2	1,25	5	20	80	320
3	1,875	7,5	30	120	480
4	2,5	10	40	160	640

Tabla 2: *Espacio necesario (en Gigabytes) para almacenar nCL campos de luz de una resolución direccional de 1280 (nivel de resolución 3), en función de los Bpp y de la resolución espacial.*

de las capacidades multihilo de las nuevas generaciones de procesador.

4.1. Análisis de la información requerida

La cantidad de memoria necesaria para almacenar un campo de luz viene definida por

$$size = D \times M^2 \times Bpp \tag{1}$$

En esta relación D representa el número de direcciones en el que se ha discretizado el espacio de direcciones del campo de luz y viene dado por $D = 20 \times 4^L$, siendo L el nivel de detalle. Así mismo M se corresponde con la resolución espacial (número de píxeles por imagen) y Bpp con el número de bytes por píxel de cada una de las imágenes.

Nuestra intención es visualizar una escena que contenga varios campos de luz, tal y como se describe en [EBA*08]. La cantidad de memoria necesaria para almacenar y visualizar varios campos de luz viene definida por

$$size = \sum_{i=1}^{nCL} D_i \times {M_i}^2 \times Bpp_i \tag{2}$$

Si suponemos que todos los campos de luz tienen las mismas características (resolución direccional, espacial y bytes por píxel), tenemos que son necesarios $nCL \times D \times M^2 \times Bpp$ bytes para almacenar estos campos de luz. La tabla 2 muestra el espacio necesario para almacenar un campo de luz con una resolución direccional de 1280 direcciones, usando 4, 6 y 8 bytes por píxel.

Nivel	Núm. direcciones	45^o	60^o
0	20	1	2
1	80	4	7
2	320	16	27
3	1280	62	107
4	5120	245	427
5	20480	980	1707

Tabla 3: *En esta tabla puede verse el número de* pencils *necesarios para visualizar el campo de luz en función del nivel de detalle en el que se está visualizando para unas anchuras de campo de* 45^o *y* 60^o.

Sin embargo, no es necesaria la visualización de toda esta información de manera simultánea. Si tomamos en cuenta el ángulo sólido visible podemos concluir que el número de *pencils* visibles en un momento dado vendrá dado por:

$$P(fov_y) = D\frac{fov_y}{2\pi}\sin\frac{fov_y}{2} \tag{3}$$

donde fov_y es el *field of view*.

4.2. Estudio del tamaño óptimo de los tiles

Siguiendo con el razonamiento anterior, puede apreciarse que el número de *pencils* depende de la resolución direccional en la que se visualiza el campo de luz y la anchura de campo utilizada. A continuación, en la tabla 3 se muestra el número de *pencils* dependiendo de estos valores.

El número de píxeles por *pencil* (ppp) se puede estimar dividiendo la cantidad total de píxeles en pantalla por el número de *pencils* a dibujar. Nuestro software siempre toma en cuenta un porcentaje de *pencils* a dibujar mayor que el calculado por $P(fov_y)$, en concreto un 30 % más, ya que esta es la cifra que asegura el dibujado correcto de todos los *pencils* visibles. Así, el número de píxeles por *pencil* se puede calcular a partir de la resolución del dispositivo de salida ($R_{out_x} \times R_{out_y}$) :

$$ppp = \frac{R_{out_x} \times R_{out_y}}{1{,}3 \times P(fov_y)} \tag{4}$$

A partir de este dato, podemos realizar un cálculo que nos indique el tamaño de imagen óptimo (Res_{img}), así como la resolución ideal de las texturas (Res_{tex}) para almacenar esa imagen. Teniendo en cuenta que utilizamos *pencils* triangulares, la resolución de las imágenes cuadradas se obtiene como :

$$Res_{img} = \sqrt{2 \times ppp} \tag{5}$$

Tomando en cuenta que el hardware gráfico mejora su rendimiento con resoluciones potencia de 2 :

$$Res_{tex} = 2^{\lceil \log_2 Res_{img} \rceil} \tag{6}$$

Por otra parte, el número de bytes por frame (bpf) que es necesario procesar será:

$$Bpf = P(fov_y) \times {Res_{Tex}}^2 \times Bpp \tag{7}$$

Con toda esta información, es fácil comprobar que la velocidad de transferencia necesaria para el paso de la información de memoria secundaria a GPU a fin de visualizar el campo de luz viene determinada por los bytes necesarios por cada frame (Bpf) multiplicado por la velocidad a la que mostremos la información (fps – frames por segundo):

$$vel = Bpf \times fps = \tag{8}$$

$$= D\frac{fov_y}{2\pi}\sin\frac{fov_y}{2} \times 2^{\left\lceil \log_2 \sqrt{2 \times \frac{R_{Out_x} \times R_{Out_y}}{1,3 \times D \frac{fov_y}{2\pi} \sin\frac{fov_y}{2}}} \right\rceil^2} \times Bpp \times fps$$

En los cálculos siguientes, tomaremos la visualización de un campo de luz en un nivel de detalle 3 con una anchura de campo de 45^o o 60^o como referencia, dependiendo del tipo de pantalla que se desee utilizar para la visualización.

En la tabla 4 se muestran varios datos. Entre ellos encontramos el número de píxeles a dibujar en pantalla, así como el número de píxeles (por término medio) que ocupa cada *pencil*. También se muestran el tamaño de imagen necesario por *pencil* (tomando en cuenta la forma triangular de éstos y la forma cuadrada de las imágenes utilizadas, es necesario leer el doble de píxeles de los que se demanda en el dibujado) y el tamaño de textura adecuado para ese tamaño de imagen. Todos estos datos aparecen calculados en función de distintas resoluciones de pantalla.

Por último, en la tabla 5 puede apreciarse la cantidad de información que debemos procesar en cada frame, expresada en MB. Ésta dependerá del número de *pencils* y del tamaño de la textura correspondiente, así como de la velocidad (en MB/seg) a la que resulta necesario transferir la información para conseguir una visualización a 25, 45 y 60 frames por segundo. Por ejemplo, para una resolución de salida de $1600x1200$ tendriamos una media de unos 23704 píxeles por pencil, lo que se ajustaria a un tamaño de imágen de 217^2 y necesitaríamos una textura de 256^2 para cargar esta imagen en el hardware gráfico. Cabe destacar que estos cálculos se han realizado bajo los supuestos de la utilización de imágenes RGB con 1 byte por componente así como mapas de profundidad con una resolución de 2 bytes por píxel, para un total 5 bytes por píxel.

Tam. Textura	MB	25 FPS	45 FPS	60 FPS
45^o				
128^2	6,33	158,20	284,76	379,68
256^2	25,31	632,81	1139,06	1518,75
60^o				
64^2	2,73	68,35	123,04	164,06
128^2	10,93	273,43	492,18	656,25
256^2	43,75	1093,75	1968,75	2625,00

Tabla 5: *Cantidad de información a transferir a la GPU por frame y velocidad de transferencia necesaria para conseguir alcanzar los FPS deseados (en MB/s).*

Según los resultados extraídos de este estudio (ver tabla 4, hemos decidido fragmentar las imágenes que dan soporte al campo de luz usando un tamaño de *tile* de 256^2.

5. Implementación de la caché

Nuestra máquina de referencia está equipada con un Intel Core 2 Quad a 2.4Ghz con 2GB de memoria RAM y una tarjeta gráfica NVIDIA GeForce 8800 GT con 512 MB de memoria.

Hemos realizado un estudio del tiempo necesario para leer los datos del disco duro y transferirlos a la GPU basándonos en la información del hardware disponible y el sistema operativo. Para transferir los datos a la GPU de la manera más rapida posible hemos seguido los consejos que NVIDIA propone en su documento [NVI05]. Durante las pruebas realizadas hemos podido concluir que el tiempo de lectura de un *tile* comprimido usando PNG desde disco es de 13 ms, y el tiempo de descompresión del *tile* es de aproximadamente 0,005 ms, mientras que el coste temporal de trasferir uno de estos *tiles* de memoria principal a GPU se sitúa en unos 0,3 ms.

A la luz de esos resultados es fácil comprobar que resulta imposible leer desde la memoria secundaria la totalidad de la información necesaria para visualizar cada frame manteniendo un frame-rate aceptable. Como solución a este problema se presentó en [EBA*07] una caché para la visualización de campos de luz que se encargaba de la carga y mantenimiento en memoria principal de las imágenes del campo de luz y se aseguraba así mismo de cargar solamente aquellas que no se encontraban ya en la caché.

Como ya se ha visto, es ineficiente en términos de espacio usar un modelo de caché de imágenes completas cuando sólo se utiliza un porcentaje muy bajo de cada una. Por esta razón hemos reimplementado dicha caché para realizar cargas de *tiles* del campo de luz en vez de imágenes completas.

La nueva caché se basa en el algoritmo ARC [MM04] y posee una arquitectura de tres niveles. El primer nivel es el encargado de la transferencia de los datos demandados desde

Resolución	núm. píxel/ *pencil*	Tam. imagen	Tam. textura
Anchura de campo de 45^o			
500x500	3087	78^2	128^2
1000x1000	12346	157^2	256^2
800x600	5926	108^2	256^2
1024x768	9710	139^2	256^2
1280x1024	16182	179^2	256^2
1600x1200	23704	217^2	256^2
Anchura de campo de 60^o			
500x500	1786	59^2	64^2
1000x1000	7143	119^2	128^2
800x600	3429	82^2	128^2
1024x768	5618	106^2	128^2
1280x1024	9363	136^2	256^2
1600x1200	13715	165^2	256^2

Tabla 4: *Esta tabla muestra el número de píxeles por* pencil *y tamaño ideal de imagen que deberían tener los tiles para poder contener ese* pencil*, así como su tamaño de textura.*

memoria secundaria a memoria principal. Los *tiles* del campo de luz utilizan una compresión sin pérdida, en concreto el formato PNG, que permite una descompresión relativamente rápida. Esto permite mantener los datos del campo de luz comprimidos en memoria principal, lo que favorece una mejor utilización del espacio en memoria alojando una mayor cantidad de información ya que el tiempo de descompresión es despreciable.

Por otro lado, los *tiles* utilizados más frecuentemente se almacenan en un segundo nivel de caché, donde permanecen descomprimidos en formato BGRA preparados para ser enviados a la tarjeta gráfica de manera directa sin el coste adicional de la descompresión. Finalmente, un tercer nivel de caché se encuentra en la GPU (ver figura 4).

Nuestro anterior modelo de caché hacía ya uso de técnicas multihilo, separando la visualización y la carga de las imágenes en hilos de ejecución independientes, cosa que permitía que no se bloqueara la visualización de los campos de luz a causa de cargas síncronas. En el nuevo sistema implementado se ha decidido potenciar el uso de estas técnicas aprovechando así mejor las capacidades de los nuevos procesadores multinúcleo (ver figura 4). El nuevo sistema utiliza un hilo por cada fuente de datos disponible, lo que permite estar leyendo datos de varios discos de manera simultánea. Estos hilos sólo se encargan de la lectura de los datos en crudo desde la memoria secundaria a la caché de *tiles* comprimidos. Existe además otro grupo de hilos encargado de la descompresión de estos *tiles* y de su actualización en la caché de *tiles* descomprimidos cuando son necesarios. En la figura 4 puede verse cómo estos hilos trasladan la información del campo de luz desde la memoria secundaria a la memoria de la tarjeta gráfica. Los hilos marcados con la nomenclatura $hiloL_x$ en la figura son los encargados de leer la información

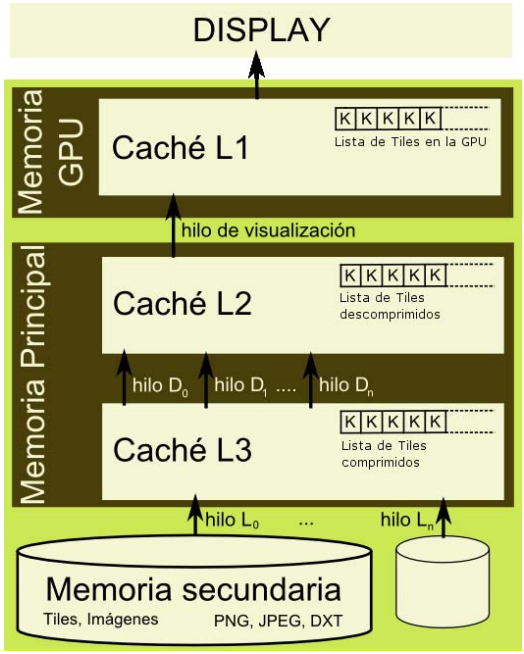


Figura 4: *Arquitectura de la cache de tres niveles. Las flechas muestran cómo los diferentes hilos de ejecución mueven la información de un nivel al siguiente.*

de los discos, los marcados como $hiloD_x$ se encargan de la descompresión de los *tiles*, y por último, el hilo de visualización es el encargado de subir esta información a la memoria de la GPU.

5.1. Modificación del algoritmo de visualización

Tal y como se ha comentado en la sección 3, el algoritmo de visualización utilizado es una adaptación del algoritmo del Lumigraph. En éste se coloca el volumen de la vista dentro de la esfera teselada y se calculan los haces que intersectan con él, así como las coordenadas de textura obtenidas proyectando los vértices sobre el plano soporte de la textura y finalmente dibujando el triángulo.

Este algoritmo supone que toda la imagen está cargada en una única textura. Sin embargo con las modificaciones antes propuestas, este supuesto cambia ya que las imágenes que componen el campo de luz se encuentran fragmentadas y es posible que el *pencil* a dibujar no caiga completamente dentro de una sección de imagen, de manera que necesitemos cargar varias imágenes para dibujar el *pencil*. En la figura 5 puede verse una imagen del campo de luz de 8192^2 píxeles compuesta por 32^2 *tiles* de 256^2 cada uno. Superpuesto se muestra un *pencil* del campo de luz y marcados en azul se puede ver los *tiles* necesarios para visualizarlo. Para dibujar este *pencil* en concreto sólo es necesario tener cargados en la GPU 30 de los 1024 *tiles* que componen esta imagen del campo de luz, lo que representa aproximadamente un 3 % de los tiles que componen la imagen del campo de luz.

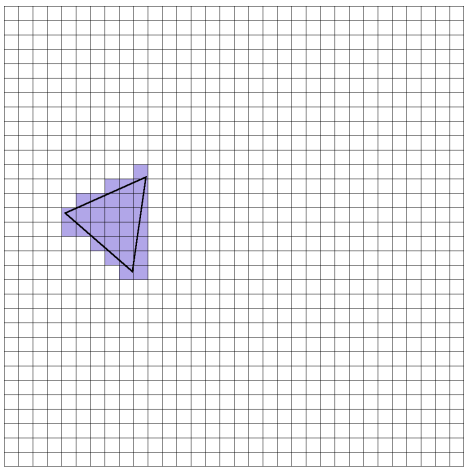


Figura 5: *Imagen del campo de luz de* 8192^2 *compuesta de* 32^2 *tiles de* 256^2 *cada uno. Superpuesto hay un* pencil *del campo de luz y marcados en azul los tiles necesarios para visualizarlo.*

Para poder visualizar correctamente los campos de luz, tendremos que modificar levemente el algoritmo de visualización. Una vez calculadas las coordenadas de textura, debemos averiguar qué *tiles* son necesarios para dibujar el *pencil* (en la figura 5 son los que están marcados en azul) y asegurarnos de que estén cargados. Después tendremos que dibujar el *pencil*, usando para ello una de las dos soluciones siguientes.

La primera es calcular las intersecciones del *pencil* con los *tiles* y dibujar cada uno de los triángulos que se formen con sus nuevas coordenadas de textura. Esta opción, aunque no es muy costosa computacionalmente, obliga a hacer algunos cálculos en la CPU. Nosotros hemos preferido liberar a la CPU de estos cálculos y utilizar el algoritmo propuesto por Lefebvre [LDN04] que se puede implementar fácilmente utilizando shaders.

Consiste en cargar en un *pool* los *tiles* necesarios para visualizar el *pencil* y construir una textura de indirección con referencia a los tiles que necesitamos (ver figura 6). Estas texturas de indirección tienen un tamaño pequeño y son fácilmente manejables. En el caso de imágenes del campo de luz de 8192^2 y *tiles* de 256^2, las texturas de indirección deberían ser de 32^2. Mediante un *fragment shader* es fácil calcular el color que le corresponde a cada fragmento del *pencil* usando la textura de indirección para acceder al *pool* de *tiles*.

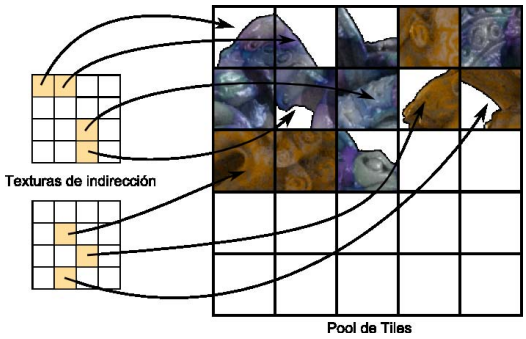


Figura 6: *Arquitectura del pool de tiles almacenados en la GPU y las texturas de indirección.*

El sistema anterior también se encargaba de seguir el movimiento de la cámara y averiguar qué *pencils* serían visibles en los siguientes frames y precargar las imágenes del campo de luz correspondientes a esos *pencils*. Debido a que ahora tenemos las imágenes que componen el campo de luz divididas en *tiles* más pequeños, no basta sólo con estimar qué pencils serán visibles, sino que debemos averiguar cuáles de los *tiles* que componen la imagen correspondiente al pencil son los que se necesita precargar. Esto se hace aplicando el mismo algoritmo que se utiliza para la visualización del campo de luz.

5.2. Resolución de los mapas de profundidad

También hemos estudiado la forma de reducir la cantidad de información que necesitamos para visualizar un campo de luz con corrección de profundidad. Al necesitar menos información para visualizar el campo de luz, será necesario cargar una cantidad menor y podremos mantener en la caché una mayor cantidad de información que nos puede servir para visualizar un mayor número de campos de luz o precachear nueva información que probablemente usemos en los siguientres frames.

En la caché almacenamos dos tipos diferentes de información, información de radiancia en imágenes de color e información de profundidad en mapas de profudidad. Estas imágenes las podemos comprimir con algoritmos conocidos ya sean con pérdida o sin pérdida (JPEG, PNG, DXTC, ...). Pero también almacenamos información de profundidad. Esta información de profundidad no podemos comprimirla usando un algoritmo de compresión con pérdida, aunque sí que podemos reducir la resolución de estos mapas.

Durante la fase de captura, se capturan y almacenan los mapas de profundidad a una resolución de 32 bits. Hemos realizado varios tests para comprobar el efecto que se produce en las imágenes resultantes al reducir el número de bits de los mapas de profundidad. Al comparar las imágenes obtenidas usando 8 bits de profundidad con las obtenidas unsado 32 bits, no se aprecian diferencias a simple vista.

En la figura 7 puede verse el modelo de campo de luz del león chino intersectando con un plano geométrico. La imagen muestra en color rojo las diferencias existentes entre usar mapas de profundidad de 32 bits y mapas de profundidad de 8 bits de resolución. Puede apreciarse que las diferencias ocurren justo en la intersección del plano con el león. Aun así, estas diferencias son mínimas, representando menos de un 1 % de la imagen resultante, y no son perceptibles a simple vista.

Figura 7: *Modelo de campo de luz del león chino intersectando con un plano geométrico. En rojo se muestran las diferencias entre usar mapas de profundidad de 8 o 32 bits de resolución.*

6. Resultados

En este artículo hemos usado varios modelos geométricos del proyecto AIM@Shape. Para capturar el campo de luz de estos modelos se ha utilizado una granja de hasta 100 ordenadores pertenecientes a los laboratorios docentes de nuestro departamento utilizando el trazador de rayos Yafray sobre Blender. Para generar las imágenes de los campos de luz se han utilizado *Ambient Occlusion*, iluminación especular de Cook-Torrance y sobremuestreo de 8 muestras por píxel.

Figura 8: *Visualización simultánea de múltiples modelos del campo de luz.*

En la figura 8 pueden apreciarse una escena visualizada usando nuestro software.

Las imágenes resultantes son visualmente idénticas a las obtenidas con el software anterior (ver figura 8), pero con un considerable ahorro en cuanto a la memoria necesaria para visualizar los campos de luz. Según como puede verse en la sección 4 el ahorro de memoria necesaria para visualizar los campos de luz puede llegar a ser considerable, lo que nos permite componer escenas con un mayor número de objetos o mantener en caché otros *tiles* que previsiblemente se visualizarán en los siguientes frames.

7. Conclusiones y trabajo futuro

En este artículo hemos presentado un sistema de caché orientado a la carga de la información de radiancia para campos de luz. Este sistema se basa en una fragmentación de las imágenes que forman el conjunto de datos original. Se utiliza un esquema de carga bajo demanda y un algoritmo de predicción para la carga de las imágenes que probablemente sean utilizadas en los próximos frames. Se han paralelizado lo máximo posible la lectura de disco de las imágenes, la descompresion y fragmentación de éstas y la carga en la GPU para aprovechar los nuevos procesadores multinúcleo y maximizar el uso de los recursos disponibles.

En vista de los resultados expuestos en la sección 5.2 se pretende estudiar cómo afectaría al resultado visual el reducir la resolución de las imágenes de profundidad. El uso de imágenes de 8 bits para los mapas de profundidad afecta muy poco al resultado final, y es posible que una reducción espacial de estas imágenes no afecte en gran medida al resultado final.

Agradecimientos

Nuestro trabajo ha sido financiado en parte por becas y ayudas del Vicerrectorado de Investigación de la Universidad Politécnica de Valencia y por el proyecto TIN2005-08863-C03-01 del Ministerio de Ciencia e Innovación.

El modelo geométrico del león (Chinese Dragon) es cortesía del INRIA. Los modelos de la gárgola (Gargoyle) y el guerrero (Grog) son cortesía del VCG-ISTI. Los tres se obtuvieron del repositorio del proyecto AIM@SHAPE.

References

[AB91] ADELSON E. H., BERGEN J. R.: The plenoptic function and the elements of early vision. *Computational Models of Visual Processing* (1991), 3–20.

[BBM*01] BUEHLER C., BOSSE M., MCMILLAN L., GORTLER S., COHEN M.: Unstructured lumigraph rendering. In *SIGGRAPH '01* (2001), ACM, pp. 425–432.

[BEA*08] BLASCO J., ESCRIVÁ M., ABAD F., QUIRÓS R., CAMAHORT E., VIVÓ R.: A generalized light-field API and management system. In *Proceedings of WSCG'2008* (Plzen – Bory, Czech Republic, 2008).

[CBCG02] CHEN W.-C., BOUGUET J.-Y., CHU M. H., GRZESZCZUK R.: Light field mapping: Efficient representation and hardware rendering of surface light fields. In *SIGGRAPH 2002 Conference Proceedings* (2002), Hughes J., (Ed.), Annual Conference Series, ACM Press/ACM SIGGRAPH, pp. 447–456.

[CLF98] CAMAHORT E., LERIOS A., FUSSELL D.: Uniformly sampled light fields. In *Proc. Eurographics Rendering Workshop '98)* (1998), pp. 117–130.

[DTM96] DEBEVEC P. E., TAYLOR C. J., MALIK J.: Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 11–20.

[EBA*07] ESCRIVÁ M., BLASCO J., ABAD F., CAMAHORT E., VIVÓ R.: Real-time foveal light fields for spatial imaging devices. In *Proceedings of* 4^{th} *European Conference on Visual Media Production* (London, UK, 2007).

[EBA*08] ESCRIVÁ M., BLASCO J., ABAD F., CAMAHORT E., VIVÓ R.: Visualización autoestereoscópica de múltiples campos de luz. In *Actas del 18 Congreso Español de Informática Gráfica, CEIG 2008* (2008), Eurographics Association, pp. 11–20.

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *Proc. SIGGRAPH '96* (1996), pp. 43–54.

[LChK*96] LEE D., CHOI J., HUN KIM J., NOH S. H., MIN S. L., CHO Y., KIM C. S.: *LRFU (Least Recently/Frequently Used) Replacement Policy: A Spectrum of Block Replacement Policies.* Tech. rep., In IEEE Transactions on Computers, 1996.

[LDN04] LEFEBVRE S., DARBON J., NEYRET F.: *Unified Texture Management for Arbitrary Meshes*. Tech. Rep. RR5210-, INRIA, may 2004.

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *Proc. SIGGRAPH '96* (1996), pp. 31–42.

[MM04] MEGIDDO N., MODHA D. S.: Outperforming lru with an adaptive replacement cache algorithm. *Computer 37*, 4 (2004), 58–65.

[NVI04] NVIDIA: *Improve Batching Using Texture Atlases*. Tech. rep., 2004.

[NVI05] NVIDIA: *Fast texture downloads and readbacks using pixel buffer objects in opengl*. Tech. rep., 2005.

[PPS97] PETER-PIKE SLOAN MICHAEL F. COHEN S. J. G.: Time critical lumigraph rendering. In *Proc. 1997 Symposium on Interactive 3D Graphics* (1997).

[QJP*07] QURESHI M. K., JALEEL A., PATT Y.Ñ., STEELY S. C., EMER J.: Adaptive insertion policies for high performance caching. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture* (New York, NY, USA, 2007), ACM, pp. 381–391.

[SVSG01] SCHIRMACHER H., VOGELGSANG C., SEIDEL H., GREINER G.: Efficient free form light field rendering. In *Proceedings of Vision, Modeling,and Visualization 2001* (2001).

[TMJ98] TANNER C. C., MIGDAL C. J., JONES M. T.: The clipmap: a virtual mipmap. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), ACM, pp. 151–158.

[TN05] TAKAHASHI K., NAEMURA T.: Unstructured light field rendering using on-the-fly focus measurement. *ICME 2005. IEEE International Conference on Multimedia and Expo* (July 2005), 205–208.

[WAA*00] WOOD D.Ñ., AZUMA D. I., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D. H., STUETZLE W.: Surface light fields for 3d photography. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 287–296.