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

Bq-Rtree: una estructura de datos para la visualización de entornos urbanos en tiempo real

J.L Pina[1], E. Cerezo[1] and F. Seron[1]

[1]Grupo de Informática Gráfica Avanzada (GIGA)
[1]Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza (España)
[1]Instituto de Investigación e Ingeniería de Aragón (I3A)

Resumen

El view-culling o podado de la pirámide de visión, es una técnica de aceleración habitualmente usada para conseguir movimientos de cámara en tiempo real en escenarios virtuales complejos. El rendimiento de esta técnica depende en gran manera de la estructura de datos sobre la cual se realice el podado. En este artículo se propone una estructura de datos, la BqR-Tree, la cual es una R-Tree modificada que mejora el rendimiento habitual de las R-Tree en la fase de view-culling. Para construir la estructura en tiempo de preprocesado, la ciudad se divide por medio de una partición espacial en quadtree y la manzana urbana se adopta como la unidad básica. La finalidad de esta estructura es acelerar la visualización de escenarios complejos urbanos que contengan, no sólo elementos estáticos, sino también dinámicos. La utilidad de esta estructura ha sido probada con datos poco estructurados, lo que la hace apropiada para casi cualquier conjunto de datos urbanos, y ha sido contrastada con otras estructuras de datos obteniendo resultados que suponen un promedio del 30 % de mejora en la velocidad de visualización.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.6]: Graphics data structures and data types—

1. Introducción

Los paisajes urbanos se cuentan entre los más voluminosos y complejos que se pueden visualizar y son particularmente interesantes para una gran variedad de aplicaciones: movimientos de multitudes, evacuaciones en situaciones de emergencia, paseos virtuales, planificación urbana, seguimiento del tráfico, estudio y evaluación del impacto ambiental del ruido, etc.

No solamente el tamaño y complejidad de los entornos urbanos actuales crece continuamente, sino también la necesidad de poblarlos de elementos móviles (gente, automóviles,...). De hecho, la visualización de entornos urbanos altamente poblados implica la resolución de dos problemas: la visualización interactiva de grandes entornos estáticos, y la visualización de multitudes animadas o de otros elementos móviles. Ambas tareas son caras en términos de tiempo de cómputo y sólo ahora se está empezando a poder ejecutarlas con éxito.

Diversas técnicas han sido desarrolladas (ver sección de Trabajos relacionados) para acelerar la visualización de escenas complejas. Este artículo presta su atención a una de ellas, el podado de la pirámide de visión (view-culling) y, en particular, al uso de una estructura de datos apropiada para acelerarlo. Como es bien sabido, un grafo de escena es un grafo acíclico que describe las entidades y dependencias de todos los elementos gráficos de la escena. El podado (culling) del grafo de escena elimina todos los objetos que se encuentran fuera de la pirámide de visión (view frustum), por medio de los volúmenes envolventes de los nodos. Un podado rápido es particularmente importante si se trata de grandes grafos de escena. La organización de la compleja geometría de la escena en una adecuada estructura de datos puede, en gran manera, mejorar el podado, dando como resultado un menor tráfico de triángulos entre la CPU y la tarjeta gráfica. La ventaja de crear una estructura de datos es que el trabajo se realiza en tiempo de preprocesado, produciendo su carga posterior una incidencia mínima en la visualización de la escena en tiempo real. Todo esto sin perjuicio de otras técnicas que se pueden aplicar con posterioridad para mejorar aún más la

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/125-133

velocidad de visualización de cada imagen.
El objetivo de este trabajo es encontrar una estructura de datos especialmente adecuada para la visualización de entornos urbanos complejos que contengan gran cantidad de entidades estáticas y dinámicas. Las estructuras de datos habitualmente se diseñan pensando únicamente en elementos estáticos, pero hoy en día, la consideración adicional de elementos móviles se hace imprescindible. Como se explicará más adelante, la consideración de la manzana urbana como la unidad básica urbana es una forma natural de integrar los elementos estáticos y móviles en la estructura de datos y permite realizar paseos y vuelos por la ciudad sin usar caminos predeterminados. Por ello, en este artículo se propone la estructura de datos que se denomina BqR-Tree, la cual es una mejora de la muy conocida R-Tree. Para su construcción se descompone la ciudad en un quadtree, siendo la unidad básica la manzana. Como se detallará más adelante, las ventajas de dicha estructura son:

- La estructura de datos propuesta es susceptible de ser aplicada a datos urbanos con baja o incluso ninguna estructuración. Por lo tanto, puede ser aplicada a datos adquiridos por diversas fuentes: los tradicionales sistemas de información geográfica (GIS 2D), mediciones sobre el terreno, etc.
- La identificación de los edificios no es necesaria. La unidad básica considerada es la manzana, la cual puede ser identificada en casi cualquier circunstancia, y es adecuada para integrar, buscar y realizar un rápido podado de los elementos móviles en el grafo de escena.
- Las pruebas realizadas sobre distintos modelos urbanos y con distinto número de elementos móviles muestran que la estructura propuesta permite una visualización más rápida que el resto de las estructuras de datos usadas hasta la fecha.

2. Trabajos relacionados

Entre la bibliografía se pueden encontrar una gran variedad de soluciones relativas a la visualización de grandes modelos de datos en tiempo real. Los métodos se pueden clasificar en cinco grandes grupos de técnicas: LOD o nivel de detalle, impostores, podados por oclusión (occlusion culling), podado de la pirámide de visión (view frustum culling) y preprocesado. Algunas de ellas se suelen combinar para producir mejores resultados. En cualquier caso, la mayoría de estas técnicas se han desarrollado para grandes modelos poligonales estáticos, así, su aplicación a miles de entidades dinámicas complejas, como actores, no es un asunto trivial.
LOD es una de las técnicas más frecuentemente usadas: se han probado LODs continuos [DB05], pero reducen la velocidad de visualización; también se han probado modelos jerárquicos [FS93] en los cuales se elige la adecuada resolución de acuerdo con el nodo del árbol que está siendo mostrado. En cualquier caso, es muy difícil evitar molestos saltos entre una imagen y otra de distinta resolución.
La técnica de impostores [ACP08] se basa en el uso de imágenes de los objetos en lugar del propio objeto 3D. El correcto uso de esta técnica permite visualizaciones realistas con un bajo coste gráfico, aunque realmente, el problema de mostrar modelos con ciento de miles de polígonos no se resuelve sino que se evita. Una pequeña mejora de esta técnica es la de reutilizar imágenes, usada en entornos estáticos [SLS*96] o implementada para individuos animados [BT04], mediante el almacenamiento de imágenes precalculadas: la imagen de un objeto se almacena en tiempo real después de mostrarla en pantalla, y mientras la imagen que se debe mostrar del objeto no cambie significativamente ésta es la que se continúa mostrando en pantalla. Dobbyn et al [DB05] presentan un sistema híbrido: mientras el objeto está lejano se usa un LOD de imágenes (impostores), pero si el objeto se encuentra más cerca de un umbral dado el LOD usado es la representación geométrica del objeto.
La técnica del podado por oclusión (occlusion culling), usando solamente recursos de software [TS91], aprovechándose del manejo de GPU [BWPP01] o en conjunción con LOD [ASVNB00], se basa en la eliminación de los objetos que no son visibles debido a que otros los ocultan de la cámara. La identificación de los objetos ocultos desde el punto de vista de la cámara supone su mayor reto. Además, cada vez que la cámara o un objeto se mueve es necesario realizar una nueva identificación. A pesar de ésto, es una de las técnicas más usadas, ya que, en un entorno plagado de elementos gráficos cercanos entre sí, algunos objetos -los más cercanos a la cámara- ocultarán a los más distantes. Aunque el algoritmo requiere la identificación de objetos ocultos y ésto consume tiempo de cómputo, el tiempo ahorrado es considerablemente superior [COCSD03]. Esta técnica es habitualmente usada en entornos urbanos [GF05] pero no es adecuada para vuelos urbanos en tiempo real [TS91]. En su forma original tampoco es adecuada para el tratamiento de elementos móviles.
El podado de la pirámide de visión (view culling) es una técnica de aceleración muy estudiada para entornos estáticos [HV07], pero raramente implementada en escenas dinámicas. Seron et al. [SFA02] han propuesto dos nuevos tipos de nodos para la integración de elementos móviles en un grafo de escena. Nirnimesh et al. [NN06] han propuesto el uso de múltiples pirámides de visión, y Wald et al. [IWS07] han implementado esta técnica en el trazado de rayos.
Preprocesado es el nombre dado a todas aquellas técnicas relativas a la estructuración de datos que facilitan la fase de visualización. Han demostrado ser muy útiles para la visualización sin consumir tiempo de cómputo en el momento de la visualización. Una forma de preprocesado es la conocida como Fuera-de-Memoria (Out-of-Core), la cual ha sido usada de manera intensiva en otras área. Una adecuada distribución de los datos en disco permite una paginación eficiente [DRJ*99]. Así, toda la información relacionada entre sí se almacena en la misma página de disco; posteriormente, cuando se necesita, se recupera [ESC00]. Respecto al uso de

estructuras de datos, del estudio comparativo de estructuras de datos publicado por Gaede et al [GG98], se concluye que aquellas estructuras de datos que pertenecen a las familias de K-D-Tree y de R-Tree proporcionan el mejor rendimiento. Las estructuras que pertenecen a la primera categoría utilizan métodos de partición del espacio que consisten en dividir el espacio a lo largo de hiperplanos predefinidos sin considerar su distribución. La regiones resultantes son disjuntas entre sí y su unión completa el espacio total. Los quadtree pertenecen a esta categoría. Las estructuras que pertenecen a la segunda categoría usan métodos de partición que dividen el espacio de datos en contenedores de VEM (Volumen Envolvente Mínimo) lo que puede conllevar el posible solapamiento de algunas regiones [FJ03]. En este segundo grupo de las R-Tree, las más usadas en entornos urbanos, las mejores variantes han sido analizadas por Katayama et al [KS97]. De acuerdo con sus pruebas de rendimiento, la VamSplit R-Tree [WJ96] es la estructura de datos con un mejor rendimiento en cuanto a la velocidad de visualización. La VamSplit R-Tree es una optimización de R-Tree y es más un método de construcción que una estructura de indexación por sí misma. La VamSplit R-Tree es una estructura estática (una inserción o una operación de borrado conlleva una reorganización de todo el árbol) y su algoritmo de construcción del árbol está basado en el de las K-D-Tree.
En este artículo se propone el uso de una estructura de indexación, la BqR-Tree, que se crea en tiempo de preprocesado con propósitos de aceleración y que está basada en la R-Tree. Para comprobar su rendimiento, la estructura VamSplit R-Tree se ha usado como referencia ya que como se ha indicado anteriormente, es la que hasta el momento ha obtenido mejores resultados.

3. La estructura BqR-Tree

La estructura Bq-Rtree es una R-Tree mejorada cuyo método de partición espacial es una descomposición en quadtree. Dicho método de descomposición es infrecuente en entornos urbanos, pero no único [BJLS06] [Man03]. Comparada con la descomposición en K-D-Tree que se usa en la VAMSplit R-tree, la descomposición en quadtree es más apropiada para la distribución espacial de las manzanas y requiere menos tiempo de podado ya que es menos profunda.
La estructura incorpora, además, una segunda mejora. Los VEM de los nodos (cuadrantes) son esferas. Según Katayama [KS97] ésto produce una mejora en la velocidad de acceso a los nodos y en el espacio necesario para su almacenamiento. No obstante, el uso de esferas produce demasiado solapamiento, razón por la cual, como se verá, para las hojas (geometría) los VEM son la intersección de esferas con cubos en lugar de esferas como en los nodos.

3.1. Construcción del grafo de escena a partir de las manzanas

La estructura BqR-Tree propuesta se basa en la descomposición de una ciudad en manzanas: la manzana se considera la unidad mínima e indivisible de la ciudad, y es la base de la citada estructura. El término manzana se usa para denominar al grupo de elementos urbanos completamente rodeados de calles, es decir, el significado habitual de manzana urbana.
Para la construcción de la BqR-tree, es necesario:

- Identificar todas las manzanas en la escena y calcular sus centros geométricos y sus VEM.
- Asignar cada elemento gráfico de la ciudad a una manzana.

En la Figura 1 se muestra una parte de la ciudad tomada como ejemplo (Zaragoza, España) con las manzanas y sus centros ya identificados. En los ficheros iniciales no estaban identificados ni los edificios ni las manzanas. Por tanto, ha sido necesario llevar a cabo el filtrado, estructuración y elevación 3D. La identificación de las manzanas se produce, en este caso, a partir de las calles, las cuales vienen ya identificadas de inicio.

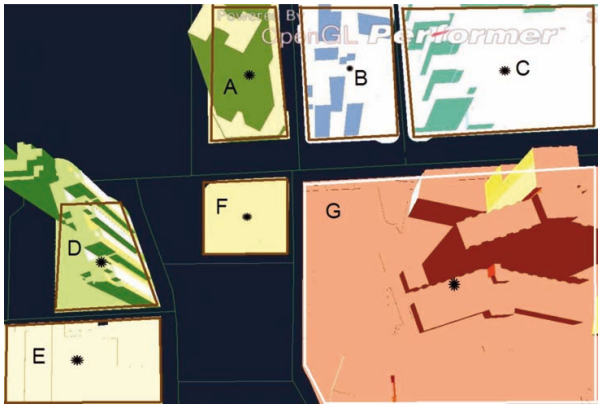

Figura 1: *Identificación de las manzanas y sus centros en la ciudad ejemplo*

El árbol de la estructura de datos BqR-Tree tree se forma a partir de la división recursiva de la ciudad en cuadrantes. Un cuadrante se considera indivisible si contiene únicamente una manzana. No se permite la división de los elementos básicos (manzanas), ya que ésto conllevaría una alta dispersión de trozos de manzana a lo largo del árbol, lo cual redundaría en una pérdida de rendimiento. Por ello, si una división atraviesa una manzana, toda la manzana se asigna a un único cuadrante, el que contiene el centro geométrico de la manzana. Al término del proceso, cada cuadrante final contiene una manzana o está vacío. En la Figura 2 se muestra una decomposición en quadtree realizada sobre el ejemplo de la Figura 1.

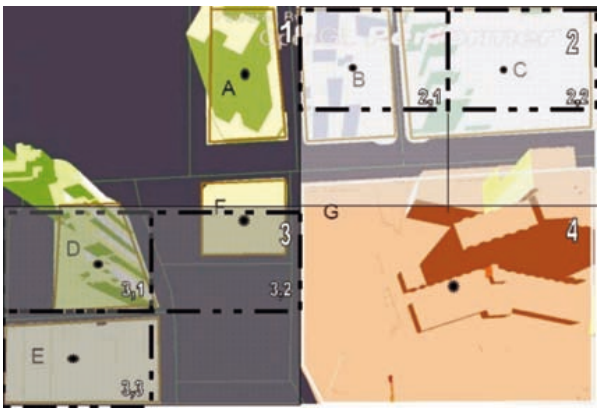

Figura 2: *Descomposición en quadtree del ejemplo de la Figura 1*

El segundo paso es la construcción, propiamente dicha, del árbol. Para ésto, cada cuadrante se asigna a un nodo o sub-nodo y cada manzana se asigna a una hoja. La Figura 3 muestra una representación gráfica de una estructura BqR-Tree correspondiente al ejemplo de la Figura 1. Las hojas se representan por cuadrados y los nodos por elipses. Los cuadrantes vacíos no se asignan a ningún nodo, lo que califica al quadtree como adaptativo. Para identificar a todos los elementos del árbol, nodos y hojas, se usa una clave numérica. Es necesario almacenar la clave de cada elemento del árbol debido a que no se almacenan los nodos vacíos. Cada hoja del árbol almacena toda la geometría de la manzana además del VEM de dicha manzana (intersección de esfera y cubo). Cada nodo almacena un puntero a sus descendientes y a su VEM (esfera), el cual está compuesto por la unión de los VEMs de sus descendientes en forma bottom-up.

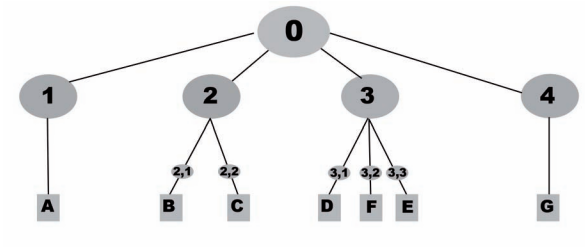

Figura 3: *Representación del árbol visto del ejemplo correspondiente a la Figura 1*

Las propiedades de la estructura de datos resultante son las siguientes:

- La estructura es un árbol de VEM (esferas) de los nodos (los cuadrantes son disjuntos pero no lo son sus VEMs asociados) y finaliza con los VEM de cada manzana (intersección de esfera y cubo) y su geometría.
- Aunque es muy similar a las estructuras de datos SR-Tree y VamSplit R-Tree, la mayor diferencia consiste en el uso de una descomposición en quadtree sin división de las manzanas, y al uso de la intersección de esferas y cubos como VEMs para las hojas.
- La complejidad de la búsqueda de un elemento es proporcional a la profundidad del árbol, siendo O(n) en el peor de los caso, donde n es la profundidad máxima del árbol, la cual tiende a ser pequeña. La unión, intersección y su complemento comparten al misma complejidad, todo lo cual lleva a un muy eficiente podado del árbol.

3.2. Gestión de los elementos móviles en el grafo de escena

Cada elemento móvil (EM) siempre está asociado con una calle y dirección. Cada calle está compuesta de dos direcciones (izquierda o derecha) y cada dirección (semi-calle) se asocia con una manzana. Por tanto, cada EM se asocia de forma no ambigua a una y solamente una manzana, y la geometría de cada elemento móvil se almacena en su hoja correspondiente. Así, la posición y velocidad de cada EM es fácil y rápidamente conocida en cada momento. Cuando los EMs se mueven, es necesario inspeccionar sus conexiones. Únicamente los EMs cuyo movimiento les lleva a otra manzana deben ser comprobados. Si la nueva calle pertenece a una manzana distinta, entonces los EMs deben ser desconectados de su vieja manzana y conectados a la nueva. Aunque en realidad, no todos los EMs deben ser re-conectados, únicamente aquellos que están dentro de la pirámide de visión (view-frustum) o acaban de abandonarla. Para el resto de los EMs, es suficiente con anotar su cambio y realizarlo cuando entren en la pirámide de visión.

4. Resultados

Los datos usados en las pruebas corresponden a la ciudad de Zaragoza y han sido amablemente cedidos por el ayuntamiento de la ciudad. El fichero inicial estaba en formato de Microstation y hubo de ser convertido a un fichero de texto que contenía 2.013.900 puntos. La estructura BqR-Tree construida se compone de 145 nodos y 96 hojas. Además se han crearon ficheros auxiliares para las 96 manzanas con un tamaño total de 300 Mb. Todas las pruebas se han realizado con un ordenador provisto de un procesador DualP4 Xeon Pentium 4 a 2.8 GHz y con una memoria de 2 Gb. La tarjeta gráfica es una GeForce Fx 6800 Ultra con una memoria de 256 Mb y el sistema operativo Windows XP.
La pruebas se han realizado usando el grafo de escena OpenGL Performer debido a que es una herramienta cuyo uso está muy extendido, aunque por supuesto, cualquier otro grafo de escena se podría haber usado. Para la carga de la estructura, se sumistró al grafo de escena un fichero con toda la geometría y los elementos (nodos y hojas) de la estructura BqR-Tree. Aunque Performer está equipado con herramientas que permiten acelerar la visualización, únicamente se ha

implementado la técnica de podado de la pirámide de visión (view frustum culling), la cual se deriva directamente del podado de la estructura de datos, con objeto de comprobar que el incremento de velocidad de visualización es achacable única y exclusivamente a la estructura de datos. En Performer, como en todos los grafos de escena, el podado de la pirámide de visión se realiza en sentido de arriba abajo del árbol. Se analiza si el volumen envolvente (bounding-volume) del primer nodo, el raíz, está dentro de la pirámide de visión, si el volumen envolvente del nodo está completamente dentro todo el nodo se acepta, si está completamente fuera todo el nodo se rechaza y si está parcialmente dentro la comprobación continúa de manera recursiva con sus descendientes. En cuanto al resto de técnicas de aceleración disponibles en Performer, si se hubiese usado la habitual elimiación de caras posteriores (backface culling) o LOD, éstas hubiesen afectado por igual a todas las estructuras de datos disminuyendo el número de triángulos a tratar por las estructuras de datos. Respecto a la técnica de podado por oclusión, ésta se aplicaría después del podado de la pirámide de visión y, por tanto, también afectaría por igual a todas las estrcuturas de datos. Por tanto, el efecto de todas ellas sería el del mismo aumento en la velocidad de visualización de todas las estructuras de datos usadas.

4.1. Pruebas con los elementos estáticos

La pruebas se han realizado sobre el modelo 3D de ciudad, ya mencionado, el cual se compone de 1.688.023 triángulos. Se ha implementado un programa con los elementos necesarios para permitir movimientos de cámara libres o guiados. Se ha diseñado un recorrido por la ciudad conteniendo todos los movimientos de cámara habituales en vuelos y paseos virtuales. El recorrido (ver Figura 4) comienza en las afueras de la ciudad a nivel del suelo, y continua dando un paseo hasta el corazón de la ciudad. Una vez allí, la cámara asciende verticalmente hasta un punto por encima de los tejados y se gira hacia el suelo. Finalmente, se realiza un vuelo diagonal desde esta posición hasta el suelo.

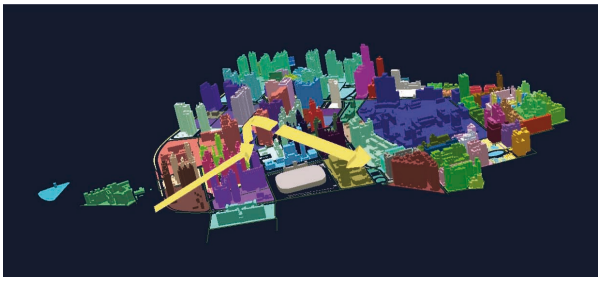

Figura 4: *Recorrido de la cámara*

Para determinar el grado de mejora obtenido con el uso de la BqR-Tree se decidió compararla con la estructura de datos que arrojaba mejores resultados según los resultados de otros autores. Como ya se ha visto en la sección de Trabajos relacionados, la estructura de datos VAMSplit R-Tree es la que ha obtenido mejor velocidad en las pruebas de visualización. Además, se usó como referencia un modelo sin estructura. La evolución del tiempo (en segundos) necesario para visualizar cada una de las tres estructuras de datos mencionadas previamente se puede observar imagen a imagen en la Figura 5:

- La línea amarilla (superior) representa al modelo sin estructura (NO).
- La línea roja (media) representa al modelo con la estructura VamSplit R-Tree (VS), y la línea azul (inferior) representa al modelo con la BqR-Tree (BQ).

Se han incluido en la Figura 5 algunas imágenes de la vista de la cámara en los momentos clave.
Como ya se ha expresado anteriormente, la única técnica de aceleración implementada es la del podado de la pirámide de visión (view frustum culling), la cual se deriva directamente del podado de las estructuras. La línea amarilla (superior) es constante, y por tanto, el tiempo de visualización para cada imagen es siempre el mismo. Ésto era esperado, ya que no hay aceleración por podado de la pirámide de visión si no hay estructura de datos. Como ya se suponía, es más lenta que cualquiera de las otras dos estructuras de datos.

Respecto a las otras dos líneas, la primera consideración es que la línea azul (inferior), la que representa a la BqR-Tree, es la línea con mayor velocidad de visualización para cada imagen, y es notablemente inferior a la línea roja (medio), la que representa a la VamSplit R-Tree. De hecho, la línea roja no es mejor en ningún momento del recorrido; como mucho llega a ser igual a la línea azul en un reducido número de imágenes. La línea correspondiente a la BqR-Tree line también es notablemente más regular que la que representa a la VamSplit R-Tree, la cual muestra dos grandes depresiones y múltiples oscilaciones. Si se analizan los resultados la razón resulta clara.
Al comienzo del recorrido, la cámara ve desde lejos gran parte de la ciudad y, por tanto, no hay posibilidad de podado de la pirámide de visión; ésto explica la inicial igualdad entre las dos líneas. No obstante, conforme la cámara se adentra en la ciudad, el podado de la pirámide de visión cobra mayor importancia y, por tanto, la desviación entre ambas líneas aumenta. El ascenso de la cámara tiene lugar entre las imágenes 48 y 140; éste es el intervalo en el que las dos estructuras más se aproximan. Posteriormente, el vuelo en diagonal las vuelve a separar. En las últimas 20 imágenes ambas líneas se aproximan de nuevo, momento en el que la cámara mira hacia el suelo y las posibilidades de podado de la pirámide de visión son similares para ambas estructuras.
La explicación para estas diferencias entre ambas estructuras se comprende mejor observando la Figura 6; en ella se muestra el número de triángulos visibles en cada imagen. La línea amarilla (superior) es constante de nuevo, como se esperaba: la ausencia de podado fuerza al sistema a enviar todos los triángulos de la ciudad a la tarjeta gráfica para cada

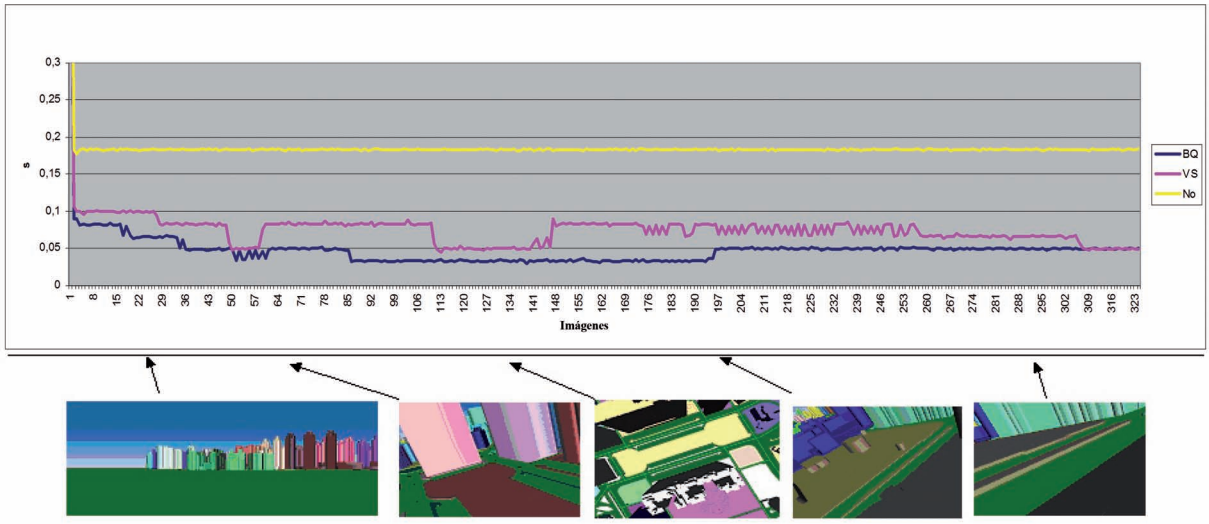

Figura 5: *Evolución del tiempo de visualización para cada una de las estructuras al realizar el vuelo de la Figura 4*

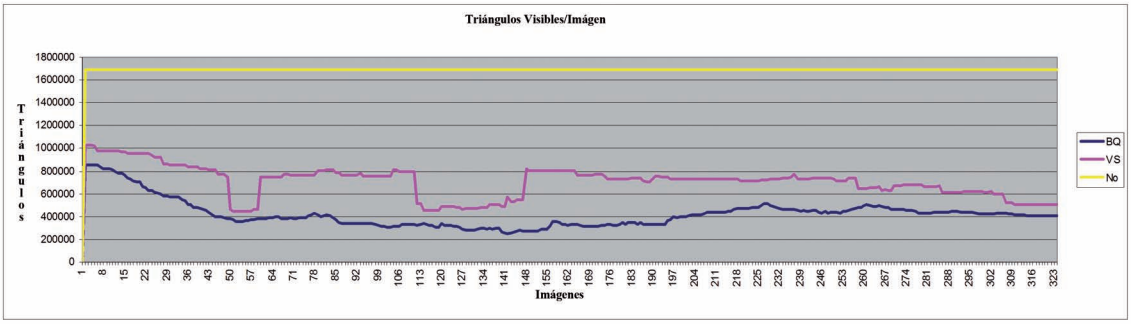

Figura 6: *Evolución del número de triángulos para cada una de las estructurasal realizar el vuelo de la Figura 4*

imagen. El comportamiento de las otras dos líneas es similar al de la Figura 5, las mismas oscilaciones se repiten de nuevo. Por tanto, se puede concluir que la nueva estructura de datos BqR-Tree, permite un podado más eficiente.

El resultado promedio para cada estructura se muestra en la Tabla 1. El cálculo del ratio entre el tiempo promedio de visualización de ambas estructuras, arroja una marcada mejora a favor de la BqR-Tree; la división BqR-Tree/VamSplit R-Tree da como resultado 0,638 y la división BqR-Tree/No estructura 0,261. Por tanto, se obtiene un incremento en la velocidad de visualización de casi un 40 % en relación con la estructura de datos VamSplit R-Tree y los resultados son un 75 % mejor que no usar ninguna estructura.

Tabla 1: *Resultados promedio para cada estructura*

	segundos/imagen	triángulos/imagen
No estructura	0,1850	1.688.023
VamSplit	0,0758	704.873
BqR-Tree	0,0483	420.894

El siguiente paso ha sido averiguar cómo el tamaño y la distribución de la ciudad afecta a los resultados. Para ésto,

Tabla 2: *Tiempo promedio de visualización (segundos/imagen) para cada modelo de ciudad con las estructuras BqR-Tree (Bq) y VamSplit R-Tree (Vs)*

	C1	C2	C3	C4	C5	C6
Bq	0,021	0,048	0,053	0,055	0,057	0,064
Vs	0,031	0,076	0,079	0,084	0,079	0,094

se crearon seis modelos diferentes de ciudad (Figura 7). El primero de ellos llamado C1, es el que tiene menor número de triángulos pero el mismo volumen que el modelo inicial comentado, C2; ésto permite comprobar la influencia de la densidad de triángulos en el rendimiento de cada estructura. El tercer modelo, C3 es el resultado de la unión de los dos modelos anteriores, C1 y C2 en la dirección del eje X. El cuarto modelo, C4 es como C3 pero con C1 añadido a C2 en la dirección del eje Y. El quinto modelo, C5 es como C3 y C4 pero ahora C1 ha sido añadido en la dirección de la diagonal XY. Finalmente, el sexto modelo de ciudad, C6, es el modelo C2 al que se le han añadido tres C1, en las direcciones X, Y y XY.
Es muy interesante observar la evolución del tiempo de visualización para cada estructura en relación con el número de triángulos. En la Tabla 2 se muestran los valores promedio. Como se podría esperar, un mayor número de triángulos conlleva mas tiempo de visualización por imágen, pero el incremento en el tiempo de visualización no es lineal respecto al número de triángulos. Ésto se pone de manifiesto en la Figura 8 donde se muestran los resultados obtenidos con la estructura de datos BqR-Tree (Bq). La línea amarilla (superior) representa el número de triángulos de cada ciudad dividido por los de C1. La línea azul (inferior) muestra el valor promedio del tiempo de visualización de BqR-Tree en relación al valor de C1. Como se puede observar:

- El modelo C2 tiene 4,6 veces más triángulos que C1, pero la velocidad de visualización es únicamente 2,3 veces mayor que la de C1 (justo la mitad).
- El modelo C6 tiene 6,5 veces más triángulos que C1, pero el tiempo de visualización es únicamente 3 veces mayor que el de C1 (menos de la mitad).

Así, el incremento del tiempo de visualización es mucho menor que el incremento en el número de triángulos; la separación entre las dos líneas incluso aumenta cuando el número de triángulos lo hace.

4.2. Pruebas con los elementos móviles

Para estas pruebas se han usado coches como elementos móviles. El modelo de coche usado se compone de 387 triángulos. La geometría de los coches se almacena en un nodo independiente, al cual no se ha aplicado ni LOD ni la eliminación de caras posteriores. Las pruebas se han implementado en el modelo de ciudad C2 previamente mencionado, y los coches se han situado inicialmente de manera aleatoria

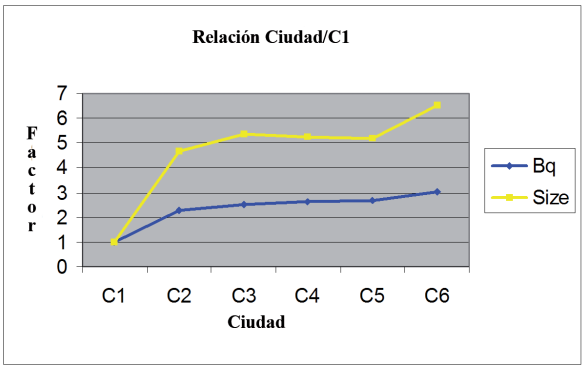

Figura 8: *Número de triángulos y velocidad de visualización en relación a C1 para los diferentes modelos de ciudad (usando la BqR-Tree)*

en las calles de la ciudad. Para cada imagen cada coche se desplaza una cantidad fija.
Se han realizado pruebas con coches conectados a la estructura de datos BqR-tree y a la previamente mencionada VamSplit-Rtree.

En la Figura 9 se muestran los resultados para ambas estructuras de datos con 100 y 1.000 coches moviéndose por la ciudad tal y como se puede apreciar, las diferencias entre ambas estructuras se mantienen después de la incorporación de los elementos móviles.

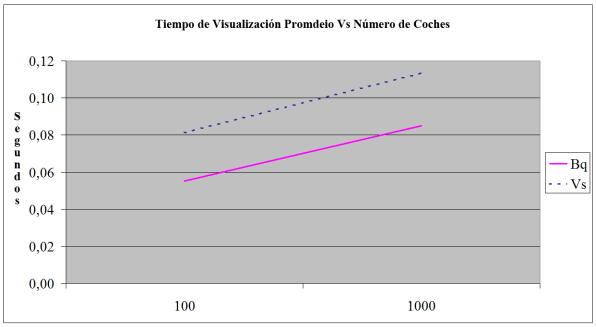

Figura 9: *Evolución del tiempo promedio de visualización (segundos por imagen) respecto al número de coches de la ciudad para ambas estructuras (Vs:VamSplit-Rtree, Bq:BqR-tree).*

Para mejorar la velocidad de visualización es necesario añadir otras técnicas de aceleración. La solución mas habitual es implementar LOD. Con el fin de comprobar la influencia de la inclusión de otra técnica de aceleración en la estructura de datos BqR-Tree se hicieron pruebas con coches compuestos de varios niveles de detalle (LOD). Así, las mismas pruebas se aplicaron a coches con LOD con un rango de entre 18 y 1.701 triángulos (Figura 10). En la Tabla 3 están

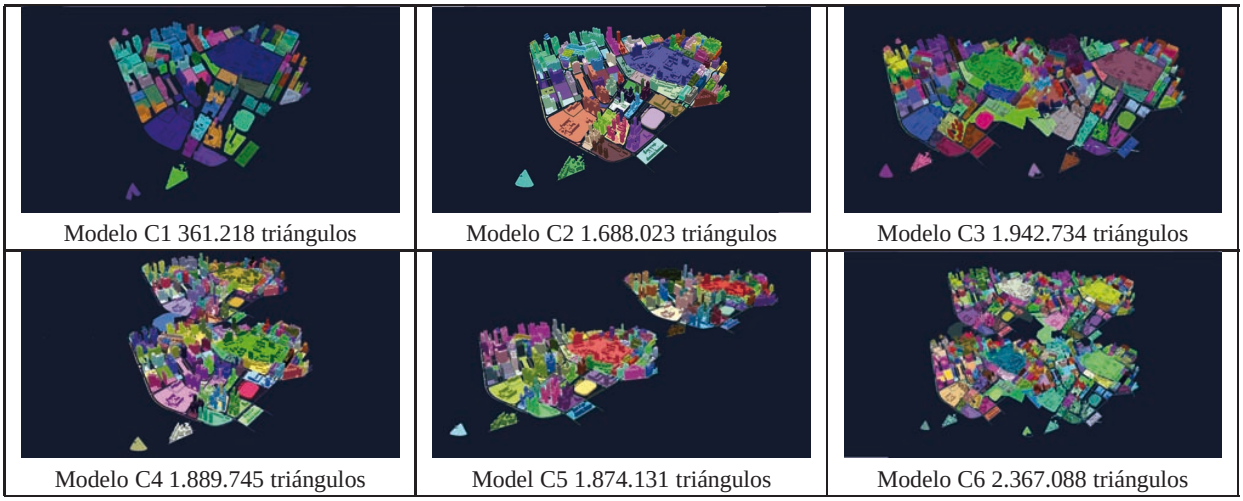

Figura 7: *Modelos de ciudad generados para las pruebas*

Tabla 3: *Evolución del tiempo promedio de visualzación (segundos/imagen) para cada número de coches con LOD*

	Segundos/Imagen 1.000 coches LOD	Segundos/Imagen 10.000 coches LOD
BqR-Tree	0,0515	0.0612
VamSplit	0,0800	0.0858

los tiempos promedio para 1.000 y 10.000 coches con LOD para cada estructura de datos. Las diferencias entre ambas estructuras se mantienen aunque, evidentemente, los tiempos absolutos disminuyen.
Por otro lado, después de conseguir una estructura de datos que agilice el podado de la pirámide de visión, se prevé que una de las mayores causas de retardo o cuello de botella, la constituya la comunicación entre la CPU y la GPU ya que se envía en cada frame toda la geometría potencialmente visible. Para conocer el alcance de este cuello de botella, se han realizado pruebas con el modelo de coche de 387 triángulos ya mencionado sobre el modelo de ciudad inicial, enviando este modelo de coche (no así la ciudad) previamente a la GPU para su almacenamiento y reutilización. Los resultados muestran una mejora en el tiempo de visualización del orden del 15 % para cualquier estructura usada. Es evidente que si se hace uso de las técnicas de almacenamiento de geometría para los elementos móviles y estáticos en la GPU, se podrían conseguir hasta un 30 % de mejora adicional en el tiempo medio de visualización sea cual fuere la estructura de datos usada.

5. Conclusiones y trabajo futuro

Se ha presentado una nueva estructura de datos, la BqR-Tree, especialmente diseñada para visualización de escenarios urbanos poblados de elementos móviles. Sus principales características son:

Figura 10: *Modelos de coche con niveles de detalle comprendidos correspondientes a 1.701, 511, 262 y 50 triángulos, de izquierda a derecha y de arriba a abajo*

- La estructura se define considerando la manzana como la unidad básica y lógica. La ventaja de la manzana respecto a la tradicional unidad, el edificio, es que se identifica fácilmente sin importar el formato de los datos fuente, y permite la inclusión de elementos móviles de una manera natural.
- La utilidad de la estructura de datos se ha comprobado con datos con muy poca estructuración, lo que hace su apliación apropiada a casi todos los datos urbanos.
- Los resultados de las pruebas muestran que cuando se usa la estructura de datos BqR-Tree para realizar paseos y vuelos virtuales urbanos (tanto con como sin elementos móviles poblando la ciudad) el tiempo necesario para la visualización mejora un 30 % de promedio en com-

paración con las estructuras de datos que han conseguido mejores resultados hasta la fecha.

Respecto a los siguientes pasos, la idea es incluir semántica en la BqRTree, es decir, tener en cuenta la información semántica a la hora de realizar el podado de la pirámide de visión.

6. Agradecimientoss

Este trabajo ha sido parcialmente financiado por la Dirección General de Investigación de España, con número de contrato TIN 2008-0574 y por el Gobierno de Aragón por medio de los acuerdos IAF200810574 y CyT2008/0486.

Bibliografía

[ACP08] ANDUJAR C. D. J., P. B.: Relief impostor selection for large scale urban rendering. *IEEE Virtual Reality Workshop on Virtual Citiscapes: Key Research Issues in Modeling Large-Scale Immersive Urban Environments* (2008).

[ASVNB00] ANDUJAR C., SAONA-VAZQUEZ C., NAVAZO I., BRUNET P.: Integrating occlusion culling and levels of details through hardly-visible sets. *Computer Graphics Forum 3* (2000), 19.

[BJLS06] BUM-JONG LEE J.-S. P., SUNG M. Y.: View-dependent simplification of complex urban scenes using weighted quadtrees. *Advances in Artificial Reality and Tele-Existence, Springer Berlin 4282/2006* (2006), 1243–1252.

[BT04] B.ULICNY P. D. H. C., THALMANN D.: Crowdbrush: Interactive authoring of real-time crowd scenes. *Proc. ACM SIGGRAPH Symposium on ComputerAnimation* (2004).

[BWPP01] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum Proceedings of EUROGRAPHICS 2004* (2001), 615–624.

[COCSD03] COHEN-OR D., CHRYSANTHOU Y., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. *IEEE Transaction on Visualization and Computer Graphics 9 3* (2003), 412–431.

[DB05] DOLLNER J., BUCHHOLZ H.: Continuous level-of-detail modeling of buildings in 3d city models. *Proceedings of the 13th annual ACM international workshop on Geographic information systems* (2005), 173–181.

[DRJ*99] DAVIS D., RIBARSKY W., JIANG T. Y., FAUST N., HO S.: Real-time visualization of scalably large collections of heterogeneous objects. *IEEE Visualization 99* (1999), 437–440.

[ESC00] EL-SANA J., CHIANG Y.-J.: External memory view-dependent simplification. *Computer Graphics Forum 19*, 3 (2000).

[FJ03] FONSECA M., JORGE J.: Indexing high-dimensional data for content-based retrieval in large databases. *Technical report, INESC-ID* (2003).

[FS93] FUNKHOUSER T. A., SEQUIN C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics 27*, Annual Conference Series (1993), 247–254.

[GF05] GRUNDHÖFER A. B. B. S. R., FRÖHLICH B.: Level of detail based occlusion culling for dynamic scenes. pp. 37–45.

[GG98] GAEDE V., GUNTHER O.: Multidimensional access methods. acm comput. surv. *Computer Graphics 30 2*, Annual Conference Series (1998), 170–231.

[HV07] H.T. VO S.P. CALLAHAN P. L. V. P. C. S.: Streaming simplification of tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics 13-1* (2007), 145–155.

[IWS07] I. WALD W. R. MARK J. G. S. B. T. I. W. H. S. P., SHIRLEY P.: State of the art in ray tracing animated scenes. *STAR Proceedings of Eurographics* (2007), 89–116.

[KS97] KATAYAMA N., SHINICHI S.: The SR-Tree: an index structure for high-dimensional nearest neighbor queries. *Proceedings of ACM SIGMOD* (1997), 369–380.

[Man03] MANOCHA D.: Real-time motion planning for agent-based crowd simulation. *In Proc. IEEE Visualization (2003). 4* (2003).

[NN06] NIRNIMESH P. H., NARAYANAN P.: Culling an object hierarchy to a frustum hierarchy. *Springer Berlin 4338/2006, Computer Vision, Graphics and Image Processing* (2006), 252–263.

[SFA02] SERON. F RODRIGUEZ. R C. E., A P.: Adding support for high-level skeletal animation. *IEEE Transactions On Visualization And Computer Graphics 8, 4* (2002), 360–372.

[SLS*96] SHADE J., LISCHINSKI D., SALESIN D. H., DEROSE T., SNYDER J.: Hierarchical image caching for accelerated walkthroughs of complex environments. *SIGGRAPH 96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), 75–82.

[TS91] TELLER S. J., SEQUIN C. H.: Visibility preprocessing for interactive walkthroughs. *Computer Graphics 25*, 4 (1991), 61–68.

[WJ96] WHITE D., JAIN R.: Similarity indexing: Algorithms and performance. *Proc SPIE Vol .2670, San Diego, USA* (1996), 62–73.