Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

Tipo de artículo: Artículo original
Temática: Programación paralela y distribuida
Recibido: 10/04/2016 | Aceptado: 15/05/2016

A-PIT: Estructura de Subdivisión Espacial Aceleración de Raytracing en GPU

A-PIT: Spatial Subdivision Structure for Raytracing Acceleration on GPU

Jean Le'Clerc Arrastia [1*], Alejandro Piad Morffis [1], Ludwig Leonard Méndez [1]

[1] Facultad de Matemática y Computación, Universidad de La Habana. CP.: 10400

[*] Autor para correspondencia: leclerc@matcom.uh.cu

Resumen

El amplio número de aplicaciones que tiene hoy en día la síntesis digital de imágenes realistas ha estimulado el desarrollo de los medios de cómputo y técnicas algorítmicas empleadas con este fin. El algoritmo *raytracing* ha sido ampliamente usado en este sentido, fundamentalmente cuando se requiere representar correctamente determinados fenómenos físicos que produce la interacción de la luz con la materia. Su simplicidad y elegancia, permiten simular efectos ópticos como la reflexión, refracción y sombras. La detección del primer objeto intersectado por un rayo, proceso que constituye la base de dicho algoritmo, es computacionalmente costosa y puede consumir fácilmente el 95% del tiempo de renderizado. Producto de esto se han diseñado diversas estructuras de datos que reducen la cantidad de operaciones realizadas en esta tarea. En este trabajo se propone una nueva estructura de datos, denominada *A-PIT*, diseñada para acelerar las consultas de intersección de rayos en la implementación del algoritmo *raytracing* en GPU. Se muestra que la utilización de esta estructura de datos en una implementación acumulativa del algoritmo raytracing permite renderizar interactivamente escenas de complejidad media.

Palabras clave: raytracing, gpu, a-buffer, interval tree, subdivisión

Abstract

The digital image synthesis has a broad range of applications, which have stimulated research in hardware and algorithms to this end. The raytracing algorithm has been widely used for this purpose, especially to simulate certain physical phenomena that arise in the interaction between light and matter. Its simplicity and elegance enable the simulation of optical effects such as reflection, refraction and shadows. Detecting the first object that intersects a ray, which is the basis of this algorithm, is a computationally complex

operation that can easily consume 95% of rendering time. Hence, several data structures have been designed to reduce the complexity of this operation. This paper proposes a new data structure, the A-PIT, designed to accelerate the ray intersection operation in a GPU implementation of raytracing. We show that using this data structure in an accumulative version of raytracing enables the interactive rendering of scenes of medium complexity.

Keywords: *raytracing, gpu, a-buffer, interval tree, subdivision*

Introducción

Desde los inicios de la disciplina gráficos por computadoras, la adecuada modelación de la inter-acción de la luz con la materia ha sido un tema de investigación activo, y aún se considera un problema abierto. La inherente complejidad de los modelos físicos que explican la distribución de la luz en la naturaleza, representa un desafío para el cómputo eficiente de las magnitudes involucradas en el cálculo de la iluminación. La correcta simulación de estos procesos constituye el objetivo principal de la síntesis digital de imágenes realistas; teniendo una amplia gama de aplicaciones que abarcan desde el diseño asistido por computadora (*computer aided design*, CAD) y la visualización de procesos científicos, hasta la industria del entretenimiento.

Este problema se ha enfocado tradicionalmente en la búsqueda de aproximaciones numéricas a la ecuación de iluminación (Kajiya, 1986), la cual describe estos efectos con precisión. La técnica *raytracing* (Whitted, 1980) representa uno de los algoritmos más populares y sencillos empleados con este fin, permitiendo la simulación de efectos como la reflexión y refracción de *Fresnel* (Torrance & Sparrow, 1967), las sombras, y la iluminación directa. Fue formulado originalmente de manera recursiva, y se basa en muestrear caminos de luz que llegan al observador. Una de las principales desventajas que presenta es la cantidad de operaciones que es necesario realizar para determinar qué parte de la escena es intersectada por un rayo de luz. Este elevado costo computacional, al igual que en otros similares, ha dificultado su aplicación en escenarios interactivos. Para aliviar este problema, se han desarrollado diversas estructuras de datos buscando acelerar el proceso de intersección, es decir, reducir el tiempo que esta toma.

El conjunto de instrucciones disponibles en los procesadores gráficos (GPU), limita en gran medida su expresividad. El hecho de solo contemplar instrucciones de salto hacia adelante en el flujo de ejecución del programa, y que su modelo de memoria hasta hace poco tiempo era fundamentalmente de lectura exclusiva y una escritura por píxel, lo

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

constatan. Aunque se han incluido nuevas funcionalidades en estos dispositivos no han sido suficientes para la implementación directa de técnicas como *raytracing*.

Una implementación de *raytracing* en GPU requiere de estructuras de aceleración que permitan realizar consultas de intersección de rayos de forma eficiente y fácil de implementar. La estructura de datos *A-Buffer* (Carpenter, 1984) puede ser implementada en GPU, gracias a las nuevas potencialidades de hardware para construir una representación global de la escena en la memoria de video. Ha sido empleada con éxito en la generación de efectos multi-fragmento tales como la transparencia, con independencia del orden, y el *anti-aliasing*. Aplicando ciertas transformaciones, esta estructura puede ser utilizada para realizar consultas de intersección de un rayo con la geometría de la escena para rayos no coherentes con el observador. La presente investigación surge tras los resultados obtenidos en la generación de efectos de iluminación en espacio de vista, y la motivación de formular una estructura de aceleración basada en el *A-Buffer* de la cual pueda auxiliarse una implementación del algoritmo *raytracing* en GPU.

Metodología computacional

Iluminación Global

Para generar imágenes digitales con un alto nivel de realismo es preciso modelar la iluminación presente en la escena, basándose en las leyes físicas que rigen el transporte de la luz, así como su interacción con ciertos materiales.

La definición formal del problema de la iluminación global fue planteada por *Kajiya* (Kajiya, 1986) con la llamada ecuación de iluminación (*rendering equation*, RE). Esta ofrece una descripción generalizada del comportamiento de la iluminación en cualquier punto del espacio, asumiendo la ausencia de un medio de transporte (es decir, los cuerpos observados se encuentran en el vacío).

La ecuación de iluminación es:

$$L_0(x,w) = L_e(x,w) + \int_\Omega f_r(x,w',w) L_i(x,w')(w' \cdot n) dw' \quad (1)$$

Donde:

$L_0(x,w)$: es la iluminación observada en el punto x de una superficie en la dirección w.

$L_e(x,w)$: es la iluminación emitida desde el punto x de una superficie en la dirección w. Este término no incluye la iluminación reflejada.

$L_i(x,w)$: es la iluminación incidente en el punto x de una superficie desde la dirección w.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

Ω: es el hemisferio superior orientado alrededor de la normal n de la superficie en el punto x.

$f_r(x, w', w)$: es la fracción de luz incidente desde la dirección w' en el punto x que se refleja por la dirección w. Esto se conoce como función de distribución de reflexión bi-direccional (*bi-directional reflectance distribution function*, BRDF).

Intuitivamente, la ecuación de iluminación asevera que la emisión luminosa en el punto x de una superficie en la dirección w es igual a la iluminación emitida por la superficie, sumada a toda la iluminación incidente que es reflejada en la dirección w.

Raytracing

Raytracing es un elegante y simple algoritmo de iluminación global, que facilita la representación de sombras y superficies especulares. Su formulación se remonta a 1980, cuando *Whitted* (Whitted, 1980) lo introdujo como extensión a la técnica *raycasting*. El algoritmo consiste en un muestreo puntual de cantidades infinitesimales de luz que llegan al observador, y se fundamenta en que la luz puede ser muestreada en sentido inverso, o sea, desde el observador hacia la fuente de luz. La entrada para el algoritmo consiste en una descripción de la geometría de la escena (objetos, materiales y fuentes de luz), la posición del observador, y un plano de imagen que representa la pantalla. El objetivo es calcular el color final de cada píxel en la pantalla, analizando uno o más rayos que parten del observador y atraviesan la pantalla en el píxel correspondiente. Los rayos que parten del observador son denominados rayos primarios. Para calcular la iluminación de un rayo primario con dirección w, se computa la intersección más cercana al observador de dicho rayo con la geometría de la escena. En el punto de intersección x se calcula la iluminación directa recibida de todas las fuentes de luz, esto es si no existe un objeto que impida su visibilidad. Para materiales perfectamente especulares se genera un rayo en la dirección de reflexión w_r (2).

$$w_r = 2(w \cdot n)n - w \tag{2}$$

Para materiales translúcidos se genera además un rayo en la dirección de refracción w_t (3), y se divide la potencia asignada a la iluminación del rayo incidente, según las ecuaciones de *Fresnel* (Torrance & Sparrow, 1967).

$$w_t = -\frac{n_1}{n_2}(w - (w \cdot n)n) - n\sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 (1 - (w \cdot n)^2)} \tag{3}$$

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

donde n_1 y n_2 son los índices de refracción de los medios separados por una superficie en el punto x. La iluminación de estos nuevos rayos en las direcciones w_r y w_t, si existen, se calculan análogamente.

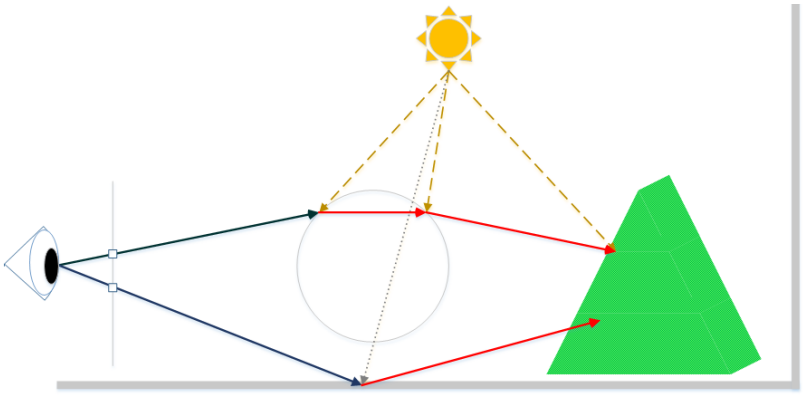

Figura 1. Algoritmo *raytracing*. Los rayos azules representan rayos primarios; mientras que los rayos amarillos y grises denotan si la iluminación directa alcanza el punto o no, respectivamente. Los rayos rojos representan rayos reflejados o refractados, según sea el caso

Las ecuaciones de *Fresnel* permiten modelar de forma físicamente correcta la interacción de la luz con los materiales translúcidos. No obstante, su cómputo resulta costoso. En implementaciones concretas, se usa la aproximación de *Schlick* (Schlick, 1994) que produce resultados plausibles con un costo computacional mucho menor.

La definición clásica de *raytracing* solo calcula la iluminación indirecta debido a rebotes perfectamente especulares (reflexión y refracción de *Fresnel*), convirtiéndola así en una aproximación muy simple a la ecuación de iluminación. No permite, por ejemplo, el cálculo de iluminación indirecta en superficies difusas. Un método más completo para aproximar la ecuación de iluminación, conocido como *pathtracing* (Kajiya, 1984), fue presentado por *Kajiya* en el mismo artículo en que propuso dicha ecuación.

Estructuras de Aceleración

El elevado costo computacional del algoritmo *raytracing* representa un desafío para su aplicación interactiva. Para determinar la primera intersección en el trayecto de un rayo con la geometría de la escena, es necesario hacer una considerable cantidad de pruebas de intersección. Además, cada intersección real puede generar a su vez dos rayos más, provocando un crecimiento exponencial de la complejidad del algoritmo, en función de la cantidad de rebotes

que se desee simular. *Whitted* (Whitted, 1980) reportó que aproximadamente el 95% del tiempo de renderizado[1] se concentra en el cálculo de pruebas de intersección entre rayos y objetos de la escena. Muchos algoritmos de iluminación global organizan la geometría de la escena, generalmente definida por triángulos, de tal forma que puedan descartar una gran parte de ella en la búsqueda de la primera intersección en la trayectoria de un rayo. Al modelo que representa este agrupamiento de los triángulos se le nombra estructura de aceleración.

Las estructuras de aceleración se utilizan para reducir el número de pruebas de intersección, y por tanto acelerar el proceso de renderizado. Una estructura de este tipo debe ofrecer un equilibrio entre su costo de construcción y reducción de pruebas de intersección, para que pueda ser considerada eficiente. Al mismo tiempo, es deseable que sea simple de representar computacionalmente. Estas se pueden clasificar en dos categorías: esquemas de subdivisión espacial (*spatial subdivision schemes*) y jerarquía de volúmenes delimitantes (*bounding volume hierarchies*, BVH) (Rubin & Whitted, 1980). La estructura de aceleración *kd-tree* se clasifica como un esquema de subdivisión espacial. Esta ha sido utilizada eficientemente en CPU para realizar consultas de intersección y estimaciones de densidad (Jensen, 2001). La construcción en tiempo real del *kd-tree* en GPU (Zhou, Hou, Wang, & Guo, 2008) puede ser empleada para acelerar una implementación de *raytracing* (Horn, Sugerman, Houston, & Hanrahan, 2007; Wald & Havran, 2006; Shevtsov, Soupikov, & Kapustin, 2007). Las jerarquías de volúmenes delimitantes, aunque no se ha descrito como una estructura rápida en CPU, su simplicidad ha permitido la obtención de buenos resultados en la aceleración de las consultas de intersección en GPU (Lauterbach, Garland, Sengupta, Luebke, & Manocha, 2009; Lauterbach, Yoon, Tuft, & Manocha, 2006; Gunther, Popov, Seidel, & Slusallek, 2007).

A-Buffer y Pseudo-Interval Tree

El *A-Buffer* es una estructura de datos, propuesta por *Carpenter* (Carpenter, 1984), que asocia a cada píxel de la pantalla una lista de los fragmentos de la geometría de la escena que son visibles a través de dicho píxel. Es decir, cada fragmento representa la intersección de una superficie de la geometría de la escena con un *frustum*[2] infinito que parte de un píxel del plano de imagen en dirección opuesta al observador, como se muestra en la Figura 2. Estos fragmentos son ordenados por su profundidad, y la longitud de la lista es llamada complejidad de profundidad. Dicha estructura ha sido empleada satisfactoriamente para la generación de sombras y la correcta simulación de

[1] Anglicismo que proviene del inglés *render* y significa literalmente representar; más en su forma verbal, entre otras opciones, se emplea como “representar o ilustrar artísticamente”, o “procesar (una imagen) para hacerla parecer sólida o tridimensional” en el contexto computacional

[2] Porción de una figura geométrica, en este caso pirámide, comprendida entre dos planos.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

transparencia sin previa ordenación (*Order-Independent Alpha Blending*) (Bavoil, Callahan, Lefohn, Comba, & Silva, 2007; Bavoil & Myers, 2008; Bavoil & Enderton, 2011). Los dos enfoques, de su implementación en GPU, recogidos en la literatura consisten en una lista enlazada (Garanzha & Loop, 2010; Barta & Kovács, 2011), y un arreglo de tamaño prefijado (Jang & Han, 2008).

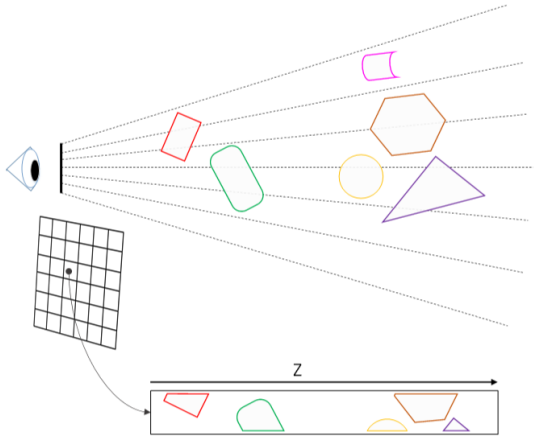

Figura 2. La estructura A-Buffer almacena las intersecciones con la geometría de la escena según el observador

La ventaja principal de la primera alternativa radica en que el espacio ocupado por cada lista es proporcional a la complejidad de profundidad del píxel al que está asociada. En cambio, representa una desventaja el hecho de realizar una búsqueda lineal para determinar el fragmento más próximo a un punto del espacio. Por otra parte, el empleo de un arreglo de tamaño fijo para almacenar los fragmentos, permite ordenarlos en tiempo $O(n \log n)$, donde n es la cantidad de elementos a considerar. Esto favorece la rápida detección del fragmento más cercano a un punto del espacio, realizando una búsqueda binaria en la profundidad. A su vez, esta tiene la desventaja de que todo píxel que tenga más fragmentos asociados que la cantidad máxima permitida (preestablecida) en los arreglos perderá información, y se desperdicia espacio en los píxeles donde la cantidad de fragmentos no cubren esta cantidad. Estas implementaciones carecen, fundamentalmente, de un adecuado balance entre el espacio ocupado y su eficiente utilización en las consultas. *Vasilakis* (Vasilakis & Fudos, 2012) explota la naturaleza esparcida de los píxeles vacíos en una imagen para construir un *A-Buffer*, que evita las limitaciones presentes en las propuestas basadas en listas enlazadas y arreglos de tamaño fijo. Esta extensión recibe el nombre de *S-Buffer*.

Si en cada una de las caras de un cubo centrado en el observador se construye un *A-Buffer*, se obtiene una subdivisión del espacio. Esta representación de la escena, con resolución dependiente del observador, posibilita el diseño de una función de intersección de rayos. Para determinar la primera intersección con la geometría de la escena en el trayecto

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

de un rayo se pueden visitar, en orden ascendente según su distancia al origen del rayo, los *frustums* atravesados por dicho rayo. En cada uno de estos *frustums* se verifica si alguno de sus fragmentos es intersectado. En caso de que la sección del *frustum* analizado se considere que está suficientemente lejos del origen del rayo, como para que la luz incidente desde este punto sea despreciable, se detiene el proceso de búsqueda haciendo finita su ejecución. De esta forma se reduce la cantidad de pruebas de intersección, considerando solamente los triángulos que yacen en los *frustums* de la trayectoria del rayo. Sin embargo, si un *frustum* visitado contiene muchos fragmentos distantes del rayo, entonces se estará realizando un número considerable de operaciones innecesarias. Para mejorar esto se propone filtrar tales fragmentos mediante una condición necesaria, pero simple, que cumplan los fragmentos intersectados por un rayo, y luego verificar la intersección real. Cada fragmento en un *frustum* está delimitado por un intervalo de profundidad en espacio de vista, y de igual forma sucede con la sección de un rayo que atraviesa dicho *frustum*. Si un rayo intersecta un fragmento en un *frustum*, entonces necesariamente se deben solapar sus intervalos de profundidad en este.

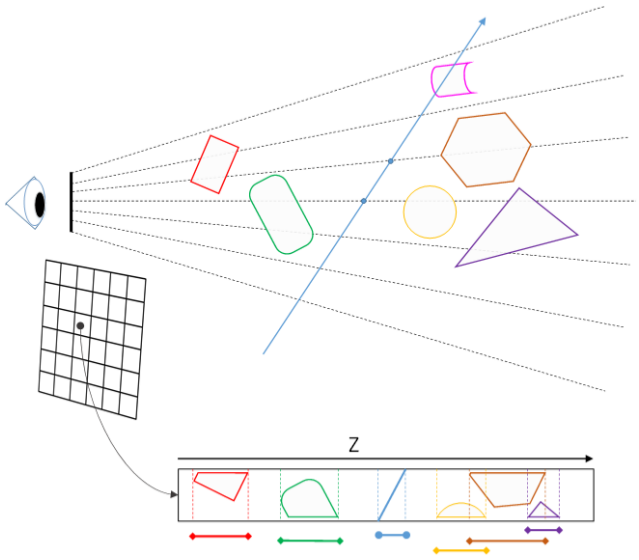

Figura 3. Modelación en segmentos de los fragmentos almacenados en un píxel

Esta observación permite plantear como condición de filtrado buscar en cada *frustum* los fragmentos en que su intervalo se solape con el intervalo asociado al rayo.

El *interval tree* (McCreight, 1980; Edelsbrunner, 1980) es una estructura de datos diseñada para reportar todos los intervalos que se solapan con un intervalo dado, con complejidad temporal y espacial óptima (Preparata & Shamos, 1985). Esta estructura genera una partición del conjunto de intervalos a consultar, empleando un árbol binario de

búsqueda. Cada nodo almacena todos los intervalos que contienen su discriminante y no contienen el discriminante de ningún ancestro a él. Esto garantiza que los subconjuntos almacenados en los nodos son disjuntos dos a dos. Por tanto, para que la unión de estos subconjuntos resulte en el conjunto de intervalos a consultar, la elección de los discriminantes debe garantizar que todo intervalo contenga al menos un discriminante de este árbol. Estas invariantes son adoptadas por el *interval tree*. No obstante, su potencial radica en que los números asignados a los nodos del árbol son seleccionados cuidadosamente para que su altura sea logarítmica respecto a la cantidad de intervalos guardados. Asimismo, cada subconjunto se almacena adecuadamente, en dos listas ordenadas, para agilizar las consultas. Aunque resulta idónea la aplicación de esta estructura de datos al problema, su construcción inherentemente recursiva (Preparata & Shamos, 1985) dificulta su implementación en GPU[3]. No obstante, se propone una alternativa que facilita dicha implementación y mantiene las mismas invariantes estructurales, aún y cuando no garantiza que su altura sea logarítmica respecto a la cantidad de intervalos que almacena. Esta será denominada *pseudo-interval tree* (PIT). En cada nodo del *pseudo-interval tree* se almacena el discriminante y una sola lista de intervalos, sin ningún criterio de ordenación. La gran diferencia respecto a su original radica en que su algoritmo de construcción es online, es decir, un proceso incremental. Este inicialmente parte de un árbol nulo, y procesa intervalo por intervalo insertándolos en la estructura construida hasta el paso anterior. Para cada nodo analizado se determina si el intervalo se almacena en él o se le cede a alguno de sus hijos, según su discriminante. Si en este proceso se alcanza un nodo nulo, entonces se sustituye por uno nuevo, tomando como discriminante el punto medio entre los extremos del intervalo a insertar, y se agrega como único elemento a su lista.

Construcción en GPU

La estructura a construir en GPU es una integración entre el *A-Buffer* y *el pseudo-interval tree* (A-PIT). La misma está compuesta por 4 arreglos: una lista de triángulos, una lista de nodos, una lista de fragmentos y una matriz de entradas.

La lista de triángulos almacena una descomposición en triángulos de la escena para permitir el cálculo de las intersecciones de forma exacta. La lista de nodos almacena los nodos del árbol que particiona los fragmentos visibles a partir del píxel correspondiente. En cada nodo se almacena su discriminante, las correspondientes referencias a sus nodos hijos y al padre, y el índice del primer fragmento que almacena en la lista de fragmentos. El conjunto de intervalos (fragmentos) que incluye un nodo está representado por una lista enlazada. La lista de fragmentos almacena

[3] Las características actuales de los procesadores gráficos no admiten ciclos potencialmente infinitos, lo cual impide expresar la recursión en total plenitud.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

los elementos de estas listas enlazadas. Para cada fragmento se almacena su profundidad mínima y máxima en espacio de vista, el índice del triángulo correspondiente en la lista de triángulos, y el índice de su sucesor en la lista enlazada a la cual corresponde. La matriz de entradas es una textura bi-dimensional que permite obtener para un píxel el índice del nodo raíz del árbol, en la lista de nodos, que particiona el conjunto de fragmentos contenidos en el *frustum* inducido por dicho píxel.

En la Figura 4 se muestra el *pseudo-interval tree* construido para los fragmentos encontrados en el píxel representados en la Figura 2.

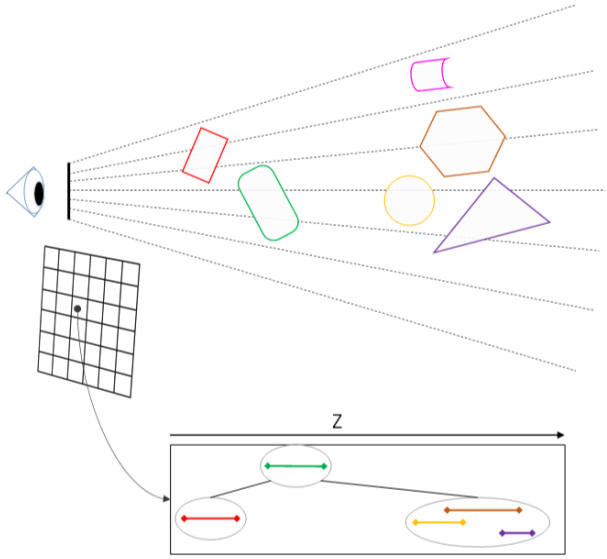

Figura 4. Pseudo-interval tree construido para los fragmentos encontrados en un píxel

Consulta al A-PIT

Con la estructura de datos construida es posible identificar rápidamente los fragmentos que se encuentran en un *frustum* a cierta profundidad en espacio de vista. Precisamente se construye un *pseudo-interval tree* por cada píxel para cumplir este objetivo. La búsqueda en el *pseudo-interval tree* es idéntica a la del *interval tree* (Preparata & Shamos, 1985), solo que, al no incluir ningún criterio de ordenación en las listas asociadas a los nodos, es necesario recorrerlas completamente. Este algoritmo consiste esencialmente, en determinar qué subconjuntos (nodos) de fragmentos (intervalos) deben ser examinados en busca de potenciales intersecciones con el intervalo de entrada.

Aunque la formulación de la búsqueda es de naturaleza recursiva, se puede implementar fácilmente en GPU realizando un recorrido en preorden por los nodos a visitar, de forma iterativa haciendo uso de las referencias al padre en cada nodo. Dado que la cantidad de instrucciones a ejecutar en un *shader* solo puede depender de datos inmutables

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

durante su ejecución[4], el número de nodos en el árbol resulta una cota bien ajustada como número máximo de iteraciones para este recorrido.

Formulación Acumulativa de Raytracing

El algoritmo *raytracing* se define en términos de una función de intersección de rayos y las ecuaciones de *Fresnel*. La formulación original produce una explosión recursiva para representar todas las posibles combinaciones de reflexiones y refracciones a través de múltiples rebotes con superficies especulares. Esto no puede ser representado directamente en un *shader* por las inherentes limitantes que poseen los procesadores gráficos.

Para hallar una formulación no recursiva, es necesario realizar algunas simplificaciones. Una posible solución es procesar todas las combinaciones de rayos en los primeros rebotes, y en los restantes determinar si el rayo se refleja o refracta de acuerdo al factor de reflexión. De esta forma se puede computar la iluminación obtenida en un punto, evaluando el camino de cada rayo por independiente, con recursión de cola, y combinando sus contribuciones. Una vez conocido que en cada rebote se va a seguir un solo camino, se puede expresar la iluminación obtenida en el punto x después de k rebotes mediante la siguiente recurrencia:

$$L_k(x) = (1 - R(x)) \cdot I(x) + R(x) \cdot L_{k-1}(x_r) \quad (4)$$

donde L_k es la iluminación calculada para k rebotes, $R(x)$ es el factor de reflexión (refracción)[5] en el punto x, $I(x)$ es la iluminación que incide directamente de las fuentes de luz (iluminación local) y x_r es el punto obtenido de la intersección del rayo reflejado (refractado) con la escena. Desarrollando la ecuación (4) se obtiene:

$$L_k(x) = \sum_{i=0}^{k} \left[\left(1 - R(x_k)\right) \cdot I(x_k) \right] \cdot \left[\prod_{j=0}^{i-1} R(x_j) \right] \quad (5)$$

donde x_k es el punto obtenido tras la intersección de k rebotes, según el camino tomado. La ecuación (5) puede evaluarse mediante la acumulación de dos funciones: el producto de los valores $R(x_j)$ y la suma de los valores de iluminación en cada paso. Las ecuaciones para el k-ésimo paso son:

$$R_k = R_{k-1} \cdot R(x_k) \quad (6)$$

$$L_k = L_{k-1} \cdot \left(1 - R(x_k)\right) \cdot I(x_k) \quad (7)$$

[4] Esto incluye tanto *buffers* de solo lectura como variables que se pueda inferir su valor en tiempo de compilación.

[5] Fracción de la luz incidente que es reflejada (refractada) por la superficie.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

El algoritmo obtenido se auxilia de texturas para acumular los resultados parciales a medida que los rayos se distribuyen por la escena. También se debe tener en cuenta que en cada iteración se deben calcular los parámetros necesarios para el siguiente rebote (origen y dirección del rayo, etc.). Nótese que estas recurrencias definen un único camino del rayo, por lo que será necesario aplicar esta fórmula (con otra combinación de rebotes) tantas veces como muestras se quiera tomar.

Resultados

A continuación, se muestran los resultados obtenidos con la técnica propuesta en este artículo.

Todos los algoritmos fueron implementados íntegramente en el lenguaje de *shader* HLSL (Feinstein, 2013) de DirectX 11. Asimismo, se utilizó la biblioteca SlimDX (Ardito, Buono, Costabile, Lanzilotti, & Simeone, 2009) en conjunto con el lenguaje C#, para manejar las escenas y el acoplamiento de los *shader* implementados. La configuración de hardware empleada en los experimentos consiste en un procesador Intel Core i7-2600 a 3.4GHz con 4GB de memoria y un procesador de video NVidia GeForce GTX 690.

Para validar la efectividad del uso de la estructura de datos presentada se implementó una variante del algoritmo *raytracing* acumulativo auxiliándose del *A-PIT* como estructura de aceleración y otra sin emplearlo, o sea, utilizando solamente el *A-Buffer*.

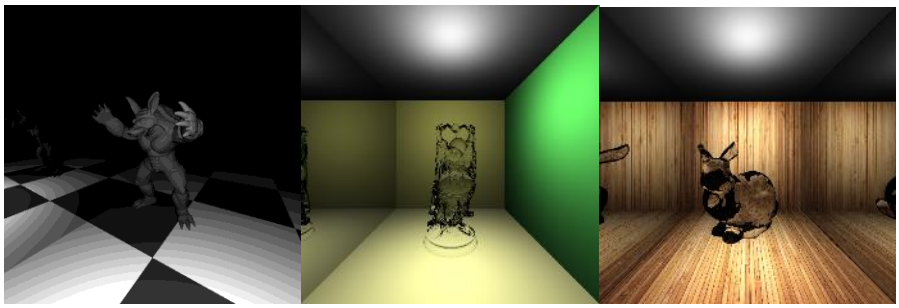

Figura 5: Visualización con *raytracing* de las escenas empleadas en la experimentación

La Figura 5 muestra una visualización del conjunto de escenas seleccionadas para las pruebas realizadas. Todas son representadas con una resolución de 512 x 512 y su complejidad varía en cuanto a su geometría (cantidad de triángulos), cantidad de objetos y si están delimitadas o no. La primera escena (*Stanford Bunny*) muestra una malla de 4968 triángulos con reflexión y refracción de *Fresnel*, delimitada por dos planos con reflexión especular perfecta a los lados y tres planos opacos con texturas en las posiciones restantes. La segunda escena (*Armadillo*) muestra una malla de 69188 triángulos opaca con un plano a su derecha con reflexión especular perfecta y otro debajo opaco con una

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 71-87

textura; esta no se encuentra delimitada. La tercera escena (*Happy Buddha*) muestra una malla de 105 triángulos con reflexión y refracción de *Fresnel*, delimitada por un plano con reflexión especular perfecta a su derecha y 4 planos opacos de diferentes colores en las posiciones restantes. Todas poseen una única fuente de luz puntual.

Tabla 1. Comparación entre el tiempo promedio (en milisegundos) que demora el renderizado de un cuadro utilizando el *A-Buffer* y el *A-PIT*

Resolución del *A-Buffer*	**Técnica**	**Escena**		
		Stanford Bunny (4K)	*Armadillo* (69K)	*Happy Buddha* (100K)
64 x 64	*A-Buffer*	42	67	178
	A-PIT	20	39	122
	Mejora	**53%**	**42%**	**31%**
128 x 128	*A-Buffer*	40	53	152
	A-PIT	19	26	81
	Mejora	**52%**	**51%**	**47%**
256 x 256	*A-Buffer*	50	60	142
	A-PIT	23	30	63
	Mejora	**54%**	**50%**	**56%**

La Tabla 1 expone la ganancia promedio de velocidad en cada una de las escenas de pruebas. En esta se puede observar que aumentar la resolución del *A-Buffer* subyacente, no necesariamente implica un incremento en la velocidad de las consultas de intersección. No obstante, es favorable hacerlo cuando la concentración de triángulos en el *frustum* asociado a un píxel es demasiado alta. Asimismo, revela el buen comportamiento del *A-PIT* en escenas donde se procesa un gran número de rayos no coherentes con el observador.

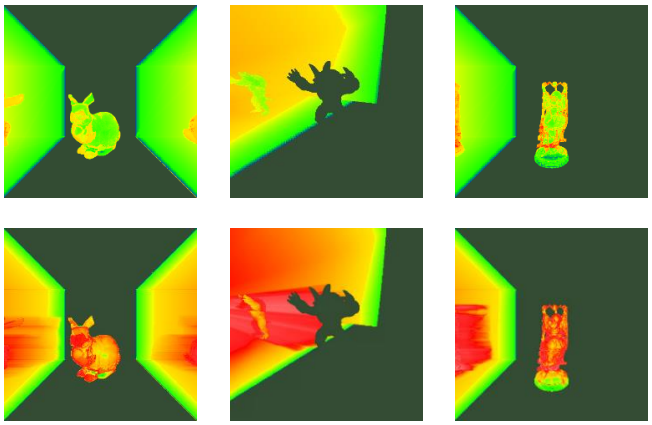

Figura 6. Comparación de complejidad (cantidad de operaciones). La fila de superior muestra la complejidad del *A-PIT* y la inferior la del *A-Buffer*

La Figura 6 expone una escala de colores, donde azul es menor y rojo es mayor, la complejidad por píxel. Aquí la complejidad de un píxel expresa la cantidad de *frustums* visitados sumado a la cantidad de fragmentos analizados para realizar las consultas de intersección en dicho píxel, es decir, una medida proporcional a la cantidad de operaciones realizadas para calcular el color de un píxel.

Intuitivamente en escenas delimitadas por varias superficies especulares se procesan una cantidad considerable de rayos no coherentes con el observador que colisionan con alguna superficie. Es decir, en escenas como la primera, no es muy grande el volumen de espacio vacío que necesita atravesar un rayo para encontrar una intersección. Esto significa que en estos casos lo más importante es optimizar la forma de encontrar la primera intersección en aquel *frustum* que la contenga, lo cual evidencia el buen rendimiento del *A-PIT* en esta escena. Por la propiedad de no ser delimitada, la segunda escena enfatiza las características del *A-PIT*. Dicha estructura es incapaz de reconocer rápidamente cuando un rayo se pierde en el infinito, teniendo que recorrer todos los píxeles hasta encontrarse suficientemente lejos de su origen para descartar el rayo. A su vez cuando un rayo no coherente con el observador está próximo a una superficie, reconoce rápidamente si hay o no posibles intersecciones. Esto se muestra por la alta complejidad que tienen los píxeles que generan rayos que no encuentran intersecciones, después de algunos rebotes, en contraste con los que todos los rebotes simulados intersectan a alguna superficie. En la tercera escena se refleja la importancia de particionar los fragmentos por cada *frustum*. Aquí se puede observar claramente la eficiencia de esta estructura de datos al disminuir considerablemente la cantidad de operaciones realizadas en cada píxel. Esto es una consecuencia del enfoque teórico adoptado para concebir dicha estructura.

Conclusiones

En este artículo se presenta una nueva estructura de datos denominada *A-PIT*, capaz de acelerar las consultas de intersección de rayos para algoritmos de iluminación global y que puede ser implementada completamente en los dispositivos de aceleración gráfica actuales.

Esta estructura surge como una combinación de las ideas del esquema de subdivisión espacial de la geometría de la escena, en listas de fragmentos, realizado por el *A-Buffer* y el método de partición de un conjunto de intervalos del *interval tree*. La misma se concibió con el objetivo de minimizar la cantidad de pruebas de intersección para rayos no coherentes con el observador, especialmente para aquellos rayos que

colisionarán con la geometría de la escena. Además, se esbozó cómo utilizarla en una implementación concreta de la variante acumulativa de *raytracing* presentada.

El conjunto de experimentos realizados mostró tanto las ventajas como deficiencias del *A-PIT*, mostrando que puede ser utilizada para visualizar interactivamente escenas de complejidad media con una calidad visual aceptable.

En cada una de las escenas se reflejó como el número de operaciones en los *frustums* con alta densidad de fragmentos se reduce considerablemente. Las escenas no delimitadas representan las de mayor dificultad para esta estructura. Cuando la geometría de la escena se encuentra bien distante del camino de un rayo, no puede evitar procesar todos los *frustums* en su trayecto, aunque no haya intersecciones en ellos.

En la literatura científica se han propuesto estructuras de datos que se basan en el mismo principio de subdivisión del *A-Buffer* y abordan el problema de atravesar rápidamente grandes volúmenes de espacios vacíos sin optimizar la búsqueda en las zonas donde hay muchos fragmentos cercanos a la trayectoria del rayo. Es decir, enfrentan el problema de la aceleración de las consultas de intersección con un enfoque totalmente ortogonal al del *A-PIT*. Se propone como trabajo futuro estudiar la factibilidad de la integración del *A-PIT* con estas estructuras.

Referencias

AALUND, F. P.; FRISVAD J. R.; BÆRENTZEN J. A. Interactive Global Illumination Effects Using Deterministically Directed Layered Depth Maps. En: Proceedings of the 26th Eurographics Symposium on Rendering-Experimental Ideas and Implementations. Eurographics Association, 2015.

ARDITO, C.; BUONO P.; COSTABILE M. F.; LANZILOTTI R.; SIMEONE A. L. Comparing low cost input devices for interacting with 3D Virtual Environments, En: 2nd Conference on Human System Interactions 2009. HSI'09, 2009, p. 292-297.

BARTA, P.; KOVÁCS B. Order Independent Transparency with PerPixel Linked Lists. Budapest University of Technology and Economics, 2011.

BAVOIL, L.; CALLAHAN S. P.; LEFOHN A.; COMBA J.; SILVA C. T. Multi-fragment effects on the GPU using the k-buffer, En: Proceedings of the 2007 symposium on Interactive 3D graphics and games, 2007, p. 97-104.

BAVOIL, L.; ENDERTON E. Constant-memory order-independent transparency techniques. NVIDIA Corp, 2011.

BAVOIL, L.; MYERS K. Order independent transparency with dual depth peeling, NVIDIA OpenGL SDK, 2008.

CARPENTER, L. The A-buffer, an antialiased hidden surface method, ACM Siggraph Computer Graphics, 1984, 18 (3): p. 103-108.

EDELSBRUNNER, H. Dynamic data structures for orthogonal intersection queries, 1980.

FEINSTEIN, D. HLSL Development Cookbook, Birmingham, Packt Publishing, 2013.

GARANZHA, K.; LOOP C. Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing, Computer Graphics Forum, 2010, 29 (2): p. 289-298.

GUNTHER, J.; POPOV S.; SEIDEL H-P.; SLUSALLEK P. Realtime ray tracing on GPU with BVH-based packet traversal. En: IEEE Symposium on Interactive Ray Tracing 2007 RT'07, 2007, p. 113-118.

HORN, D. R.; J. SUGERMAN; M. HOUSTON; HANRAHAN P. Interactive k-d tree GPU raytracing, Symposium on Interactive 3D Graphics: Proceedings of the 2007 symposium on Interactive 3D graphics and games, 2007, 30: p. 167-174.

JANG, H.; HAN J. Fast collision detection using the A-Buffer, The Visual Computer, 2008, 24 (7): p. 659-667.

JENSEN, H. W. Realistic image synthesis using photon mapping. Massachusets, AK Peters, 2001. 195 p.

KAJIYA, J. T. The Rendering Equation, ACM Siggraph Computer Graphics, 1986, 20: p. 143-150.

KNOWLES, P.; LEACH G.; ZAMBETTA F. Fast sorting for exact OIT of complex scenes. The Visual Computer, 2014, 30(6-8): p. 603-613.

LAUTERBACH, C.; GARLAND M.; SENGUPTA S.; LUEBKE D.; MANOCHA D. Fast BVH construction on GPUs, Computer Graphics Forum, 2009, 28 (2): p. 375-384.

LAUTERBACH, C.; YOON S-E.; TUFT D.; MANOCHA D. RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs. En: IEEE Symposium on Interactive Ray Tracing 2006, 2006, p. 39-46.

LEFEBVRE, S.; HORNUS S.; LASRAM A. Per-pixel lists for single pass a-buffer. CRC Press, 2014.

LIPOWSKI, J. D-buffer: irregular image data storage made practical. OptoElectronics Review, 2013, 21(1): p. 103-125.

MCCREIGHT, E. M. Efficient algorithms for enumerating intersecting intervals and rectangles, 1980.

PREPARATA, F. P.; Shamos M. I. Computational geometry: an introduction, New York, Springer-Verlag, 1985, 411 p.

RUBIN, S. M.; WHITTED, T. A 3-dimensional representation for fast rendering of complex scenes, ACM SIGGRAPH Computer Graphics, 1980, 14 (3), p. 110-116.

SCHLICK, C. An Inexpensive BRDF Model for Physically-based Rendering, Computer Graphics Forum, 1994, 13 (3): p. 233-246.

SHEVTSOV, M.; SOUPIKOV A.; KAPUSTIN A. Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes, Computer Graphics Forum, 2007, 26 (3): p. 305-404.

TORRANCE, K. E.; SPARROW, E. M. Theory for off-specular reflection from roughened surfaces, JOSA, 1967, 57 (9): p. 1105-1112.

VASILAKIS, A.; FUDOS I. k+-buffer: Fragment Synchronized k-buffer. En: Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2014, p. 143–150.

VASILAKIS, A.; FUDOS I. S-buffer: Sparsity-aware Multi-fragment Rendering, En: Eurographics (short papers). Citeseer, 2012, p. 101-104.

WALD, I.; HAVRAN V. On building fast kd-trees for ray tracing, and on doing that in O(N log N). En: IEEE Symposium on Interactive Ray Tracing 2006, 2006, p. 61-69.

WHITTED, T. An improved illumination model for shaded display, Communications of the ACM, 1980, 23 (6): p. 343-349.

ZHOU, K.; HOU Q.; WANG R.; GUO B. Real-time KD-tree construction on graphics hardware, ACM Trans. Graph. SIGGRAPH Asia, 2008, 27 (5): p. 121-126.