CONSTRUCCIÓN DE ÁRBOLES BSP VÍA ALGORITMOS GENÉTICOS

Paul Kienholz

y

Rhadamés Carmona

y

Héctor Navarro

Universidad Central de Venezuela. Escuela de Computación
Laboratorio de Computación Gráfica. Caracas, Venezuela. 1041A
pakienholz@aol.com, rcarmona@strix.ciens.ucv.ve, hnavarro@strix.ciens.ucv.ve

Abstract

An algorithm to construct Binary Space Partition (BSP) trees based on genetic algorithms was implemented. Traditional methods for constructing BSP trees limit the search to planes obtained from the scene polygons, or parallel to the planes x=0, y=0 or z=0. In our case, to find each partition plane, the genetic algorithm is executed to find the plane's coefficients, without additional restrictions. Time measures were taken with scenes from the Quake III game, and scenes created with Lightwave, that demonstrated on practice that the results obtained using genetic algorithms are superiors than using traditional methods, with an acceptable response time.

Keywords : BSP tree, Genetic Algorithms, Scenes Navigation.

Resumen

Se implementó un algoritmo para construir árboles de Partición Binaria del Espacio (BSP) basado en algoritmos genéticos. Los métodos tradicionales de construcción de árboles BSP, limitan la búsqueda a planos obtenidos a través de los polígonos de la escena, o paralelos a los planos x=0, y=0 o z=0. En nuestro caso, para hallar cada plano de partición, se corre el algoritmo genético para encontrar los coeficientes de dicho plano, sin restricciones adicionales. Se realizaron mediciones de tiempo con escenas del juego Quake III, y otras creadas en Lightwave, que demostraron en la práctica que los resultados con algoritmos genéticos son superiores que con métodos tradicionales, con un tiempo de respuesta aceptable.

Palabras claves: Árbol BSP, Algoritmos Genéticos, Navegación de Escenas.

1 Introducción

En Computación Gráfica existe siempre la necesidad de reducir el número de polígonos a ser desplegados en una escena, para poder realizar más rápidamente su despliegue. Se han implementado diversas técnicas con el fin de descartar aquellos polígonos de la escena que no deben ser desplegados. Una de las técnicas que ha tenido mayor éxito es el uso de árboles de partición binaria del espacio (BSP por Binary Space Partition), una estructura de datos que divide recursivamente el espacio hasta dejar finalmente pocos polígonos en cada hoja del árbol. Tradicionalmente, para dividir el espacio se utilizan planos definidos por polígonos de la escena. Con esta limitación, hay casos en los que los árboles construidos no están bien balanceados. Por ejemplo, cuando se tiene una esfera representada mediante polígonos, cualquier plano que se seleccione para subdividir el espacio, únicamente podrá separar el espacio en un subespacio vacío y otro que contiene los restantes $n-1$ polígonos. En la fig. 1.1 se muestra un caso llevado a 2D. Como puede notarse, no existe ningún segmento que pueda tomarse para subdividir el objeto de forma balanceada. En este trabajo se presenta un método genético que busca planos arbitrarios para construir el BSP. De esta forma el algoritmo no está limitado por la topología de la escena.

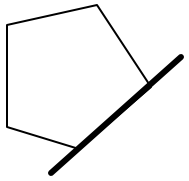

Figura 1.1: Plano de partición. Ejemplo 2D

2 Antecedentes

Los algoritmos genéticos son métodos adaptativos introducidos en 1970 por Jim Holland, que pueden usarse para resolver problemas de búsqueda y optimización. Están inspirados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los algoritmos genéticos son capaces de crear soluciones para problemas del mundo real. Los programas evolutivos son procesos estocásticos de cómputo iterativo ejecutados sobre una población de estructuras representando soluciones potenciales a un problema. En cada ciclo del proceso (generación) se evalúan las estructuras en la población actual en cuánto a su efectividad como soluciones del problema. Basado en esta evaluación se implementa una estrategia reproductiva que genera una nueva población, en la que se la da más importancia a las mejores estructuras previas. El proceso se ejecuta hasta alcanzar soluciones satisfactorias [9],[12]. Los aspectos fundamentales de todo algoritmo genético son:

- Codificación y decodificación de estructuras de información tipo cromosomas: las posibles soluciones que se representarán en los cromosomas deben codificarse de algún modo para poder representarse. Algunos ejemplos son la representación binaria, en donde la información es codificada a números binarios, y la representación real, en donde la información se representa por números reales.
- Mecanismo de selección basado en función de aptitud: la función de aptitud indica la calidad de un individuo. Se asume que si el valor de la función de aptitud para un individuo es mayor, entonces el individuo es mejor. En base al valor de la función de aptitud de cada individuo, estos son seleccionados para cruzarse, y producir la próxima generación. Los individuos con mejor función de aptitud deben tener mayor probabilidad de ser seleccionados.
- Proceso de reproducción que permita la alteración parcial de los cromosomas y la recombinación de la información en las estructuras. Deben existir mecanismos (llamados operadores genéticos) que permitan alterar la información de un cromosoma (operador de mutación), o recombinar la información almacenada en dos cromosomas (operador de cruce). Los operadores genéticos seleccionados dependen en gran medida de la forma en que fue codificada la información en los cromosomas.

Los árboles BSP fueron originalmente propuestos por Schumacker et al [16] en 1969, y refinados por Fuchs para la detección de superficies ocultas, como se describe en [7],[8]. Lo utilizaron como un método para calcular las relaciones de visibilidad entre un grupo de polígonos 3D estáticos, desde un punto de vista arbitrario. El árbol se construye previamente a la visualización de la escena. En esa implementación se utilizaron como planos de partición a planos derivados de los polígonos de la escena. Implementaciones más sencillas utilizan como planos de partición a planos arbitrarios paralelos a los planos $x=0$, $y=0$, $z=0$.

Murali et al [1], y Tobola et al [12], construyen árboles BSP para escenas compuestas por rectángulos ortogonales, y comparan su algoritmo con otros algoritmos similares. Este algoritmo es aplicable únicamente a este tipo de escenas,

que son bastante comunes en modelos arquitectónicos. Existen también herramientas que aproximan escenas 3D arbitrarias por escenas compuestas únicamente de rectángulos ortogonales, a través de los cajas delimitadoras de los objetos [13],[18].

Cassen et al [4], plantea un método evolutivo para la construcción de árboles BSP. En este método, se toman como planos separadores, los planos definidos por los polígonos de la escena. De esta forma se obtienen n planos separadores, y se pueden formar $n!$ árboles distintos. Toma como los cromosomas un subconjunto aleatorio de los posibles árboles a formar. Cada cromosoma es de tamaño n, y no es más que una permutación de los planos de partición. Como función de costo a optimizar, utilizan el número de nodos en el árbol y otra función para construir mejores árboles BSP para trazado de rayos.

Kumar et al [11], consideran la actualización del árbol en el movimiento, y agregación/eliminación de objetos en la escena. Utilizan elipsoides delimitadores entre objetos para determinar el plano de partición entre dos objetos. El algoritmo trabaja haciendo mezcla de árboles BSP's. Los planos no son totalmente arbitrarios.

Mark de Berg [6], describe un método para construir árboles BSP en dimensiones arbitrarias. Utiliza este método para ubicar puntos en el espacio. Se plantea que la subdivisión del espacio se realiza hasta llegar al nivel de objetos, y no polígonos, por lo que utiliza los cajas delimitadoras de los objetos para hacer la partición del espacio. Los planos obtenidos no pasan sobre polígonos de la escena, pero se debe tener información sobre los objetos que la conforman. El problema es que muchos tipos de escenas no proveen información sobre los objetos presentes en ella, sino que dan una lista de los polígonos.

Además de las aplicaciones ya mencionadas de árboles BSP (remoción de superficies ocultas, localización de puntos en la escena), otras aplicaciones comunes son: generación de sombras [5], operaciones de conjuntos sobre poliedros [14],[20], preprocesamiento de visibilidad para caminatas interactivas [19], trazado de rayos [17], iluminación global [23].

El BSP es muchas veces descrito como una extensión de árboles binarios ordenados para 3D. Un árbol binario ordenado es un árbol binario que cumple la condición siguiente:

$A=(r, HI, HD)$, en donde r es la raíz del árbol, y HI, HD son los subárboles izquierdo y derecho respectivamente. Además, $\forall x \in HD: r<x$ y $\forall x \in HI: r>x$.

Cuando se construye un árbol ordenado a partir de un conjunto de números reales, se utilizan siempre valores que estén dentro de la data para particionar el espacio. Es por esto, que los métodos tradicionales de árboles BSP utilizan como planos de partición a planos derivados de los polígonos de la escena. En este caso, la raíz es un plano de partición π y para determinar si un polígono P pertenece a HI o HD, se evalúa cada vértice de P en la ecuación de π. Si los vértices están detrás de π, P se coloca en HI, y si están delante, en HD. Cuando un polígono es intersectado por π, este puede duplicarse y ubicarse tanto en HI como en HD, o dividirse para formar dos nuevos polígonos, uno en HI y otro en HD. Este proceso se ejecuta recursivamente hasta obtener una condición de parada que depende de la implementación. Por ejemplo, altura del árbol o número de polígonos en cada hoja.

El árbol binario ordenado es fácil de crear ya que existe una relación de orden total entre los elementos en $\Re$. Sin embargo, cuando se extiende el problema para espacios mayores como $\Re^2$ o $\Re^3$, la construcción del árbol es difícil debido a que no existe una relación de orden total entre los puntos de estos espacios.

Naylor et al [14], analizan las características que deben tener los árboles BSP para ser considerados adecuados. Particularmente, nos interesa que el árbol sea balanceado (después de hacer una subdivisión del espacio, debe haber aproximadamente la misma cantidad de polígonos en cada nuevo subespacio creado), y que el número de polígonos intersectados por los planos de división sea mínimo.

En este trabajo se propone la construcción de un árbol BSP, utilizando planos de partición arbitrarios que permiten obtener árboles BSP más balanceados que los obtenidos por otros métodos.

3 BSP Genético

Para la construcción del árbol BSP, se implementó un algoritmo genético. La mejora principal es que, a diferencia de otros trabajos realizados [1],[4],[7],[8],[14],[17],[21],[23], no nos limitamos a elegir como planos separadores a los asociados a los polígonos de la escena, ni a planos paralelos a x=0, y=0 y z=0. Tampoco hay limitaciones acerca del tipo de objetos (i.e. convexos) que la escena pueda tener. En nuestro caso, consideramos cada plano separador como un plano arbitrario con ecuación $ax+by+cz+d=0$, en donde las incógnitas son precisamente los coeficientes del plano, haciendo que nuestro enfoque sea más general.

Para determinar el plano separador del nodo raíz del árbol BSP, ejecutamos un algoritmo genético para encontrar los coeficientes a, b, c y d, que minimiza una función de costo. Así, a partir del conjunto T de triángulos de la escena, obtenemos dos subconjuntos de triángulos: *BACK* y *FRONT*. Estos conjuntos son los triángulos dispuestos detrás y al frente del plano respectivamente. El proceso se repite para los conjuntos *BACK* y *FRONT* hasta llegar a una condición de hoja. Se puede notar que este es un algoritmo voraz, ya que en cada nodo del árbol toma una decisión sobre cuál plano utilizar para particionar el espacio, sin tomar en cuenta los efectos en futuros planos de partición [2].

Dos modelos de algoritmos genéticos fueron probados para resolver el problema: un algoritmo genético simple extendido (AGSE), y un algoritmo genético experimental (AGE). La diferencia principal entre ellos es la representación del individuo, y los operadores de cruce y mutación.

3.1 Algoritmo Genético Simple Extendido

Para este modelo se utilizará la representación binaria [9]. Los valores a, b y c están en el rango [–1,1] ya que el vector normal de los planos separadores está normalizado. El valor del coeficiente d varía según las dimensiones de la escena a la que se desea conseguir el árbol BSP. Cuando se utiliza representación binaria, y el rango de los valores es [A, B], y se utilizan n bits para su representación, el intervalo es discretizado, y se pueden representar 2^n valores distintos dentro del intervalo con un paso de $(B–A)/2^n$. En esta implementación se utilizaron 24 bits para representar cada uno de los cuatro coeficientes del plano. Para los coeficientes a, b, c, tenemos que A=–1 y B=1, por lo que el paso equivale a $1/2^{23} \approx 0.000000119$. Para el coeficiente d (distancia del plano al origen), el rango depende de la escena, pero B-A suele variar entre 100 y 1000, por lo que, en el peor de los casos el paso es de: $1000/2^{24} \approx 0.000059$.

El operador de cruce se implementó a un punto. Para la mutación, se realiza un ensayo de Bernoulli por cada bit (gen) del cromosoma, con una probabilidad dada. El éxito de este ensayo indica si el bit debe ser complementado.

3.2 Algoritmo Genético Experimental

Para este modelo, la representación utilizada es la decimal. Cada coeficiente es representado como un número punto flotante de doble precisión (64 bits). El mecanismo de cruce se basa en la interpolación lineal de las componentes del plano. Por ejemplo, si a_1 y a_2 denotan la primera componente del plano para dos individuo a cruzar, la componente a_h del hijo puede obtenerse como $a_h=a_1*t+a_2*(1–t)$. El valor de t es generado con un número aleatorio entre 0 y 1.

En cuanto a la mutación, en este caso se hace un ensayo de Bernoulli por cada dígito de las componentes del plano para decidir si cambia o no. En caso que el cambio se deba realizar, se genera un número aleatorio uniforme entre 0 y 9 para obtener el valor al cual cambiará el dígito. Un ejemplo puede ser observado en la Fig. 3.1.

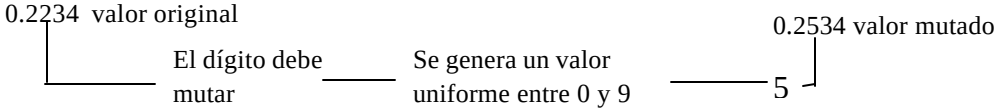

Figura 3.1: Mutación de un dígito en el modelo experimental. Por cada dígito se hace un ensayo de Bernoulli para determinar si se debe mutar o no. En este ejemplo, el ensayo de Bernoulli dio éxito para el segundo decimal. El dígito es mutado mediante un número entero aleatorio entre 0 y 9.

3.3 Función de Aptitud

La aptitud de cada individuo la definimos como:

$$f(x) = |cantf(x) - cantb(x)| + 1.5 * canti(x)$$

El valor de $|cantf(x) - cantb(x)|$ denota la diferencia en módulo de la cantidad de polígonos que se ubican delante y detrás del plano que representa el individuo x. La función $canti(x)$ denota la cantidad de polígonos que interceptan el plano. Ya que estos polígonos deben ser repetidos en el árbol BSP, se penaliza este valor multiplicándolo por un factor de 1.5 [4].

3.4 Renormalización, Selección y Sustitución

Antes de iniciar el proceso de selección de los individuos que se cruzarán, se normalizarán linealmente sus aptitudes; esto con el fin de ofrecerle mayor oportunidad a los individuos que no tienen buena adaptación, pero que aún pueden tener genes que aporten positivamente a las siguientes generaciones. La selección de los individuos para el cruce se realizó con la rueda de la ruleta [9],[12], y la sustitución, reemplazando sólo al 10% de la población (Sustitución con GAP generacional) [9],[12].

4. Implementación y Resultados

El algoritmo de BSP fue implementado en Visual C++ 6.0, en un PC con procesador Athlon de 750 Mhz, con 196 MB de RAM [10].

Nombre de la escena	Cantidad original de polígonos
Q3dm1.bsp	1942
Ctf_space.bsp	2192
Combo.lws	1433
Chessboard.lws	3783
Padshop.bsp	4090
Ctf_canyon.lws	2881
Butterflyes.lws	488

Tabla 4.1: la cantidad de polígonos para cada una de las escenas. Estas provienen del juego Quake III (*.bsp) [23] y el software de animación Lightwave [22](*.lws).

La primera prueba consistió en determinar cuál de los dos algoritmos genéticos (simple extendido y experimental) genera mejores resultados en cuanto al tiempo de convergencia y calidad de solución. Para ello se variaron los parámetros de cruce y mutación en 70 combinaciones, y para cada combinación, se realizaron 30 corridas. El tamaño de la población fue fijado en 100 individuos y la cantidad de generaciones en 1000. Los datos de las escenas tomadas para esta prueba se muestran en la tabla 4.1.

Como resultado, el algoritmo genético experimental (AGE) resultó superior en general que el algoritmo genético simple mejorado (AGSM), en cuando a la velocidad de respuesta y calidad del árbol resultante con 1000 generaciones. Pudimos notar que el AGE converge más rápidamente a la solución. Haciendo otra prueba con 2000 iteraciones, pudimos notar que ya en 300 generaciones, en general, el AGSM puede alcanzar la calidad que obtendría el AGSM en 2000 generaciones. Así concluimos que la convergencia es cerca de seis veces más rápida para el AGE. Los parámetros que en general dieron los mejores resultados en el AGE fueron de 0.85 y 0.03 para la probabilidad de cruce y mutación respectivamente.

La prueba final consistió en comparar tanto la calidad de los árboles BSP generados, como el tiempo de respuesta en 7 escenas (ver tabla 4.1) de tres métodos:

- Método 1: utilizando planos alineados.
- Método 2: utilizando como planos, los obtenidos a partir de los polígonos de la escena.

- Método 3 utilizando el algoritmo genético experimental, fijando como parámetros 100 individuos, 1000 generaciones, y la mejor combinación de parámetros de cruce y mutación encontrada para el problema (0.85 y 0.03 respectivamente).

La altura de los árboles BSP se limitó a 3, para hacer un estudio de los primeros planos de partición. Con esto se construyen 8 subespacios en los que se distribuyen los polígonos. La Fig. 4.1 muestra la cantidad de polígonos que cada método incrementó, para cada una de las escenas. Vemos que en general, el algoritmo genético tuvo un mejor resultado como era de esperarse, al considerar como planos de partición a planos arbitrarios.

Es deseable que la cantidad de polígonos en las hojas sea aproximadamente la misma. Mientras la variación sea más pequeña, el árbol resultante es más adecuado. La Fig. 4.2 muestra que la desviación estándar para la cantidad de polígonos en las hojas del algoritmo genético es inferior a los otros métodos, en todos los casos, por lo que la calidad del árbol es superior.

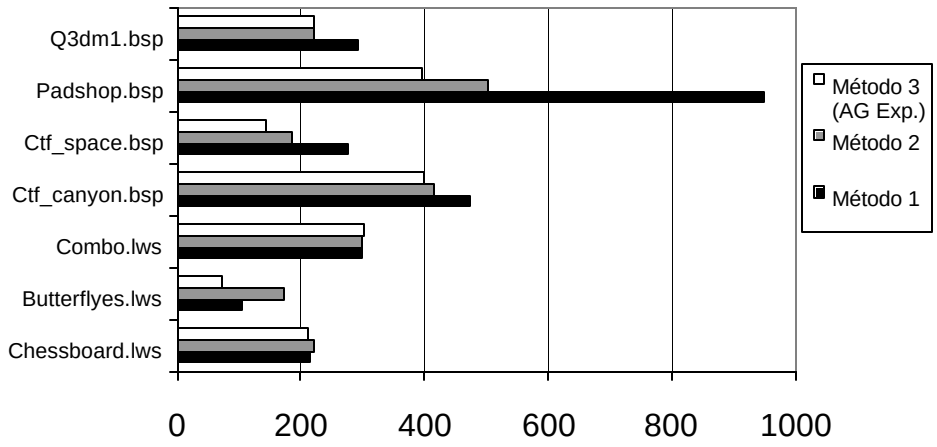

Figura 4.1: Cantidad de polígonos incrementados por cada método

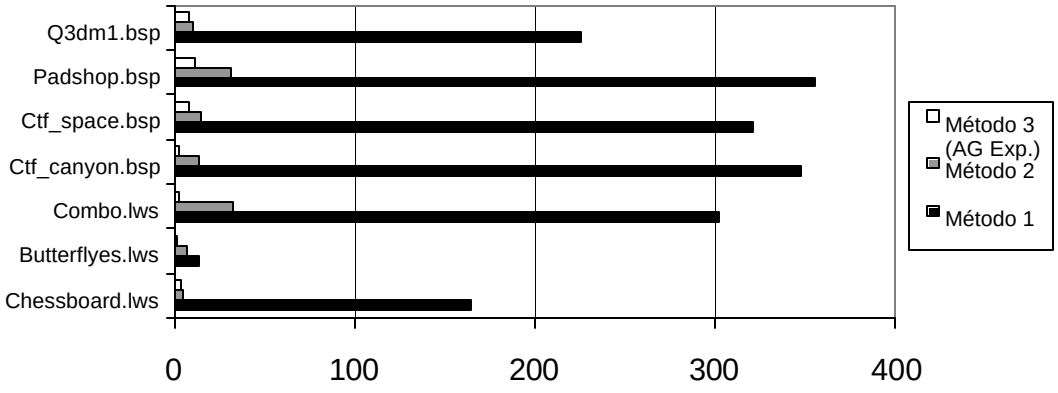

Figura 4.2: Desviación estándar de la cantidad de polígonos ubicados en las hojas

La Fig. 4.3 muestra el tiempo de construcción del árbol BSP para los tres métodos. Es claro que el algoritmo genético consume más tiempo. Pero también es cierto que este algoritmo barre efectivamente el espacio de soluciones (planos factibles de partición), mientras que los otros métodos se limitan a planos particulares de este espacio. Además, el tiempo de generación del árbol no es tan crítico, tomando en consideración que la calidad del árbol traerá beneficios en

la navegación de escenas, detección de colisiones, trazador de rayos, y demás procesos que se pueden ejecutar muchas veces a posteriori.

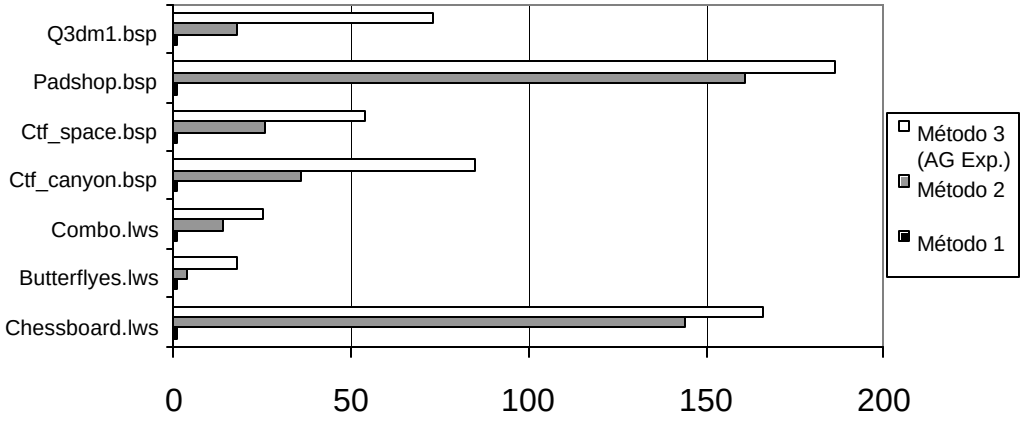

Figura 4.3: Cantidad de segundos requeridos para construir el árbol BSP

5 Conclusiones

Se propuso una solución novedosa utilizando algoritmos genéticos para la construcción de árboles BSP de una escena tridimensional basada en polígonos. Dado que el algoritmo genético no restringe el conjunto de planos posibles de partición, los resultados obtenidos son superiores en cuanto a calidad del árbol generado, respecto a los métodos tradicionales.

Aunque el algoritmo genético simple tiene un soporte teórico que asegura la convergencia en un tiempo finito, demostramos se puede realizar modificaciones al algoritmo genético original en cuanto a la representación, acelerando la convergencia a razón de 1 a 6 para este problema en particular.

Debido a la importancia que tienen los árboles BSP en la navegación de escenas, detección de colisiones, trazador de rayos, y otros problemas, recomendamos ampliamente el uso de algoritmos genéticos para la construcción de éstos, ya que la calidad de los árboles obtenidos es superior a los obtenidos con los métodos tradicionales. Esto redundará en beneficio de la aplicaciones que utilicen los árboles BSP para resolver algún problema.

6 Trabajos futuros

Si limitamos la altura del árbol a una altura *h* dada, podemos construir un cromosoma que contenga una solución para todos los nodos del árbol, es decir, los coeficientes de cada plano de partición. Bajo este enfoque, se busca el mejor árbol de altura *h* que particiona la escena. Hipotéticamente, el árbol obtenido de esta forma debe ser mejor que los árboles conseguidos en este trabajo, dado que la función de adaptación puede considerar características globales del árbol (como la cantidad de polígonos en las hojas), y no las características locales.

Otro enfoque que se puede implementar es buscar planos de partición arbitrarios, que minimicen globalmente la distancia de los vértices de los polígonos al mismo. La búsqueda de este plano se puede realizar a través de la técnica de mínimos cuadrados.

Dado que en este trabajo la altura de los árboles BSP fue fijada en tres, planteamos la posibilidad de realizar pruebas con árboles de altura mayor, e incluso, con un criterio hoja que dependa del número de polígonos y no la altura. También sugerimos probar con otros tipos de operadores genéticos para descubrir en lo posible mejores operadores para este problema particular.

Referencias

[1] Agarwal, P. y Murali, T. y Vitter J. *Practical Techniques for Constructing Binary Space Partitions for Orthogonal Rectangles*. Proceedings of 13th Annual ACM Symposium Computer Geometry. Pp 382-384. 1997.

[2] Brassard, G. y Bratley, P. *Fundamentos de Algoritmia*. Prentice Hall, Madrid 1997. ISBN: 84-89600-00-X.

[3] Campbell, A. *Modeling Global Diffuse Illumination for Image Synthesis*. PhD Thesis. Department of Computer Sciences, University of Texas, Austin. 1991.

[4] Cassen, T. y Subramanian, K. y Michalewicz, Z. *Near-Optimal Construction of Partitioning Trees by Evolutionary Techniques*. Proceedings Graphics Interface '95, pp. 263-271, 1995.

[5] Chin, N. y Feiner, S. *Near Real Time Shadow Generation Using BSP Trees*. Proceedings SIGGRAPH 89. pp 99-106. 1989.

[6] de Berg, M. *Linear Size Binary Space Partitions for Uncluttered Scenes*. Technical report UU-CS1998 -12, Dept. Comput. Sci., Utrecht Univ., 1998.

[7] Fuchs, H. *Distributing a Visible Surface Algorithm over Multiple Processors*. Proceedings of the ACM Annual Conference, Seattle, WA, October 1977, pp 499-451.

[8] Fuchs, H. y Abram, G. D. y Grant, E.D. *Near Real-Time Shaded Display of Rigid Objects*. SIGGRAPH 83, pp 65-72. 1983.

[9] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA 1989.

[10] Kienholz, P. *Técnicas para Navegar Escenas Tridimensionales en Tiempo Real*. Tesis de grado. Universidad Central de Venezuela. Escuela de Computación. Biblioteca Alonso Gamero. 2001.

[11] Kumar, A. y Kwarta, V. y Singh, B. y Kapoor S. *Using Separating Planes between Objects for Efficient Hidden Surface Removal*. Proceedings of International Conference on Visual Computing. Goa, India. 1999.

[12] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 2nd edition, Springer Verlag, Berlín 1994.

[13] Murus, M. J. *Understanding the preparation and analysis of solid models*. Techniques for Computer Graphics. Springer-Verlag. 1987.

[14] Naylor, B. *Constructiong Good Partitioning Trees*. Proceedings of Graphics Interface '93. pp 181-191. 1993.

[15] Naylor, B. y Amanatides, J. y Thibault, W. *Merging BSP trees yields polyhedral set operations*. Proceedings SIGGRAPH 90. pp 115-124. 1990.

[16] Schumacker, R. y Brand, R. y Gilliland, M. y Sharp, W. *Study for applying computer-generated images to visual simulation*. Technical report AFHRL-TR-69-14. US Air Force Human Resources Laboratory. 1969.

[17] Sung, K. y Shirley, P. *Ray Tracing with the BSP Tree*. Graphics Gems III. AP Professional. pp 271-274. 1992.

[18] Tanenbaum, P. J.. *Applications of computational geometry in army research and development*. Invited talk, second CGC Workshop on Computational Geometry, 1997.

[19] Teller, S. y Séquin, C. *Visibility preprocessing for interactive walkthroughs*. Proceedings SIGGRAPH 91. pp 61-69. 1991.

[20] Thibault, W. y Naylor, B. *Set operations on polyhedra using Binary Space Partitioning Trees*. Proceedings SIGGRAPH 87. pp 153-162. 1987.

[21] Tobola, P. y Nechvíle, K. *Lineal BSP Trees for Sets of Hyperrectangles with Low Directional Density*. WSCG 2001 Conference Proceedings. 2001.

[22] www.newtek.com

[23] www.idsoftware.com