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

Tests de inclusión punto en objeto basados en árboles de intervalos

F. Martínez[1] y A.J. Rueda[1] y F.R. Feito[1]

[1]Departamento de Informática, Universidad de Jaén

Abstract

En este trabajo presentamos varios tests de inclusión punto en objeto, para objetos de dos y tres dimensiones, que utilizan árboles de intervalos como estructura de datos básica. Los tests destacan por su sencillez, escaso consumo de memoria y, como se demuestra empíricamente, rapidez.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

1. Introducción

El test de inclusión punto en objeto es una operación geométrica básica en Informática Gráfica, Sistemas de Información Geográfica y otras áreas. En general, este tipo de tests se pueden clasificar en dos categorías. En la primera, los algoritmos trabajan con la representación de la frontera del objeto, sin realizar ningún procesamiento previo ni emplear ninguna estructura de datos adicional. Un ejemplo de estos métodos es el test de la suma de cruces de Shimrat [Shi62], corregido por Hacker [Hac62]. En este test se contabiliza el número de veces que un rayo lanzado desde el punto de consulta atraviesa los lados o caras del objeto, si este número es impar el punto está en el objeto, en otro caso está fuera. En la segunda categoría se realiza un procesamiento previo del objeto—normalmente una descomposición—, y se construye una estructura de datos alternativa a la representación mediante la frontera del objeto. Esta estructura implica unas necesidades de memoria adicionales, que se compensa con una aceleración en la ejecución del test. Normalmente, estos tests aplican la suma de cruces, pero gracias a la estructura obtenida pueden descartar a muchos lados o caras que no pueden intersecan con el rayo lanzado desde el punto de consulta.

Un árbol de intervalos—*interval tree*, véase [dBvKOS00]—es una conocida estructura de datos geométrica que permite almacenar una serie de n intervalos. Su construcción requiere un tiempo $O(n \log n)$, y sus necesidades de memoria son $O(n)$. La estructura permite consultar los k intervalos que contienen a un valor dado en tiempo $O(\log n + k)$. En este artículo describiremos cómo se puede utilizar los árboles de intervalos para desarrollar varios tests de inclusión que destacan por su sencillez, bajo consumo de memoria, y rapidez. Los tests se basan en calcular el intervalo angular ocupado por los lados o caras del objeto con respecto a un rayo que parte de un punto arbitrario—a dicho punto lo denominaremos *origen*, y lo denotaremos como O. Estos intervalos se guardan en el árbol de intervalos. Dado un punto de consulta q, se calcula el ángulo del segmento $\overline{Oq}$ con respecto al rayo que parte del origen, y se consulta al árbol para obtener los lados o caras que abarcan al ángulo. Solamente algunos de estos lados o caras pueden intersectar con una línea que pase por el origen y el punto de consulta. El resto de lados o caras pueden ser, por tanto, descartados para un test de cruces.

Lo que resta de artículo se estructura de la siguiente manera. La Sección 2 describe un test de inclusión punto en polígono basado en el árbol de intervalos, y lo compara con algunos de los tests existentes para este tipo de objetos. En la Sección 3 se realiza una extensión del test para que trabaje con poliedros. En la Sección 4 se describe un test de inclusión punto en poliedro que se apoya en el uso de una descomposición espacial—además de emplear árboles de intervalos. La Sección 5 describe otros conocidos tests de punto en poliedro con los que poder comparar nuestras propuestas. En la Sección 6 se presentan distintos resultados experimentales sobre los tests punto en poliedro. Final-

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/057-064

mente, la última sección aporta algunas conclusiones sobre el trabajo desarrollado.

2. Test de inclusión punto en polígono

En el caso de que se trabaje en dos dimensiones, se desea desarrollar un test de inclusión punto en polígono. En primer lugar, vamos a definir el intervalo angular ocupado por una arista del polígono respecto a un rayo que parte del origen, es paralelo al eje X, y avanza en sentido positivo.

Definición 2.1 Dada una arista a del polígono, formada por los vértices v_1 y v_2, y un rayo r que parte del punto origen, O, es paralelo al eje X, y avanza en sentido positivo, sean α y β los ángulos formados por los segmentos $\overline{Ov_1}$ y $\overline{Ov_2}$ con respecto a r en sentido antihorario. Se define el intervalo angular ocupado por a con respecto a r como:

$$Intervalo(a,r) = \begin{cases} [min(\alpha,\beta), máx(\alpha,\beta)] & \text{si } |\alpha-\beta| < \pi \\ [máx(\alpha,\beta), min(\alpha,\beta)] & \text{si } |\alpha-\beta| > \pi \end{cases}$$

La Figura 1 ilustra el intervalo angular ocupado por dos aristas. El conjunto de intervalos se almacena en un árbol de intervalos. La Figura 2 ilustra el funcionamiento del test de inclusión. Dado un punto de consulta q, se computa el ángulo, α, del segmento $\overline{Oq}$ con respecto al rayo r. A continuación, se consulta el árbol de intervalos para obtener el conjunto S de las aristas—en este caso $S = \{C,D,E\}$—cuyo intervalo angular contiene al ángulo. Supongamos, que realizamos un test de cruces con un rayo que parte de q, discurre por la línea determinada por el vector $\overrightarrow{Oq}$, y tiene el mismo sentido que dicho vector—véase el rayo x de la Figura 2. Entonces, las aristas que puede cruzar dicho rayo deben encontrarse entre los elementos de S. Si cada arista almacena la ecuación general de la recta determinada por la arista, $ax+by+c=0$, el rayo cruzará la arista si $(aq_x+bq_y+c)(aO_x+bO_y+c) \geq 0$, es decir, si el punto de consulta y el origen se encuentran en el mismo lado con respecto a la línea representada por la ecuación.

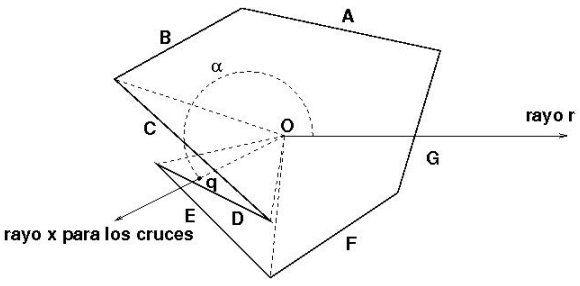

Figura 2: *Test de inclusión punto en polígono.*

El procesamiento previo de este test consiste en construir los intervalos angulares e insertarlos en el árbol de intervalos; para un polígono de n lados ésto se realiza en tiempo $O(n\log n)$. Los requerimientos de memoria son $O(n)$. En cuanto a la eficiencia del test, requiere $O(\log n+k)$ para consultar los k intervalos que contienen al punto de consulta, y $O(k)$ para seleccionar de entre estos lados cuales se cruzan con el rayo, lo que hace un total de $O(\log n+k)$.

Tabla 1: *Tiempos de ejecución (en milésimas de segundo).*

Algoritmo	Número de lados			
	10	50	2000	10000
Cruzes	0.29	1.12	32.8	153.0
Rejilla (400 celdas)	0.08	0.11	0.2	0.6
Trapezoides (20 franjas)	0.11	0.21	1.4	5.8
Árbol de intervalos (centroide)	0.80	1.34	2.1	4.4
Árbol de intervalos (esquina)	0.80	1.51	2.8	6.5

Tabla 2: *Tiempos de procesamiento previo (en milésimas de segundo).*

Algoritmo	Número de lados			
	10	50	2000	10000
Rejilla (400 celdas)	0.05	0.15	1.2	5.9
Trapezoides (20 franjas)	0.03	0.05	0.7	3.8
Árbol de intervalos	0.02	0.06	1.3	6.2

En cuanto a aspectos de implementación, señalar que las aristas atravesadas por el rayo utilizado para calcular los ángulos, como la arista G de la Figura 2 o la B de la Figura 1, precisan dos intervalos. Por ejemplo, si G ocupa el intervalo $[\theta,\alpha]$, entonces se verifica que $\theta > \alpha$, y, por tanto, se precisarán de los intervalos $[0,\alpha]$ y $[\theta,2\pi]$. También hay que señalar que el centroide es una buena elección para el origen, pues, en general, las consultas sobre el árbol de intervalos darán como resultado menos aristas de las que darían si el origen fuera exterior al rectángulo mínimo que envuelve al polígono.

En la Tabla 1 se muestra el resultado de ejecutar 1000 tests de inclusión consecutivos sobre un conjunto de polígonos cóncavos generados aleatoriamente, con tamaños comprendidos entre los 10 y los 10000 vértices. Los tests se han realizado con puntos aleatorios situados en el rectángulo mínimo que envuelve al polígono. Aparte del test de cruces, en la comparativa se han incluido dos de los test más eficientes que se conocen: el test de trapezoides y el test de la rejilla, utilizándose la conocida implementación de Haines de dichos tests [Hai94]. Como puede observarse, el test basado en el árbol de intervalos obtiene buenos resultados, especialmente en polígonos con un gran número de aristas. Se han tomado tiempos de dos ejecuciones del test basado en el árbol de intervalos; en la primera el origen se sitúa en el centroide del polígono, y en la segunda en una esquina del rectángulo mínimo que envuelve al polígono.

La Tabla 2 muestra los tiempos de procesamiento previo. Los distintos algoritmos obtienen tiempos similares, siendo todos muy eficientes en este aspecto.

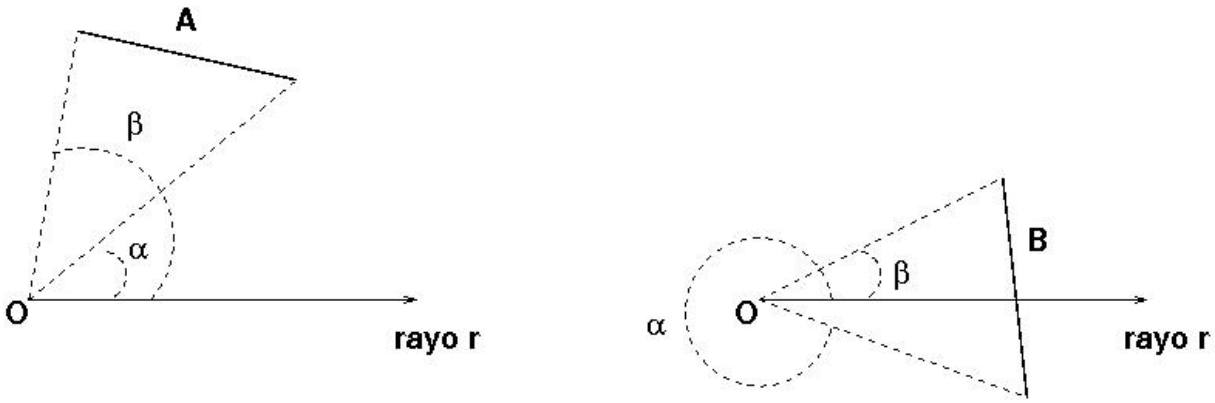

Figura 1: *Intervalo angular ($[\alpha,\beta]$) ocupado por las aristas A y B con respecto al rayo r dado el origen O.*

3. Test de inclusión punto en poliedro

En esta sección se describe cómo puede extenderse de manera sencilla el test descrito en la sección previa para que trabaje con poliedros en 3D.

Para extender el test a 3D hay que proyectar ortogonalmente las caras del poliedro en un plano; para cada cara proyectada se calcula el intervalo angular que ocupa con respecto a un rayo, situado en el plano de proyección, y que parte del origen proyectado. A continuación se define el intervalo angular ocupado por una cara triangular proyectada, para caras con más lados la definición es similar, pero algo más compleja.

Definición 3.1 Sea c una cara triangular, y P un plano de proyección. Sea c' la proyección de c en P, formada por los vértices v_1, v_2 y v_3, sea O el origen proyectado en P, sea r un rayo contenido en P y que parte de O, y sean α, β y θ los ángulos formados por los segmentos $\overline{Ov_1}$, $\overline{Ov_2}$ y $\overline{Ov_3}$ con respecto a r en sentido antihorario. Sea x el mínimo de estos tres ángulos, y z el máximo. Si c' contiene a O, entonces el intervalo angular ocupado por c' con respecto a r es $[0,2\pi]$, en otro caso es el siguiente:

$$Intervalo(c,r) = \begin{cases} [min(x,z), máx(x,z)] & \text{si } |x-z| < \pi \\ [máx(x,z), min(x,z)] & \text{si } |x-z| > \pi \end{cases}$$

La Figura 3 ilustra el intervalo angular ocupado por distintas caras triangulares proyectadas.

Los distintos intervalos angulares asociados a las caras proyectadas del poliedro se almacenan en un árbol de intervalos. Para realizar un test de inclusión sobre un punto de consulta, q, se proyecta el segmento $\overline{Oq}$—donde O es el origen—sobre el plano de proyección, y se computa su ángulo, α, con respecto al rayo utilizado para calcular los intervalos. A continuación, se consulta el árbol de intervalos para obtener el conjunto S de las caras que, proyectadas, ocupan un intervalo angular que contiene a α. Se puede realizar un test de cruces con un rayo que parte de q, discurre por la línea determinada por el vector $\overrightarrow{Oq}$, y tiene el mismo sentido que dicho vector; las caras del poliedro que pueden cruzar dicho rayo serán un subconjunto de S. La Figura 4 muestra el conjunto S de caras devueltas por el árbol de intervalos para una consulta concreta. En este caso el origen es el centroide del poliedro y el plano de proyección es $Z = 0$—el eje Z atraviesa la calavera desde la parte frontal a la trasera.

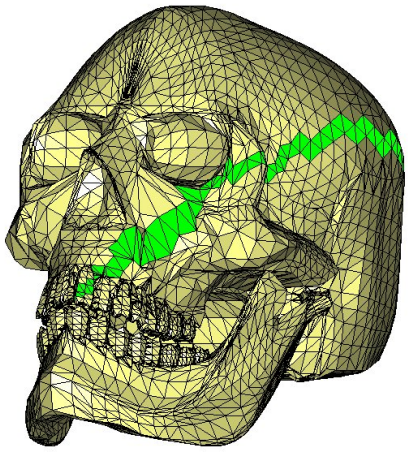

Figura 4: *Conjunto de caras devueltas (en verde) por una consulta al árbol de intervalos.*

El procesamiento previo de este test se realiza en tiempo $O(n \log n)$, donde n es el número de caras del poliedro. Los requerimientos de memoria son $O(n)$, y la eficiencia del test es $O(\log n + k)$, donde k es el número de caras cuya proyección abarca al punto de consulta proyectado. Observando la Figura 4 se puede deducir que, en general, k será bastante inferior a n, y que la mayoría de las caras asociadas a los k intervalos no intersectarán con el rayo empleado en la cuenta de cruces.

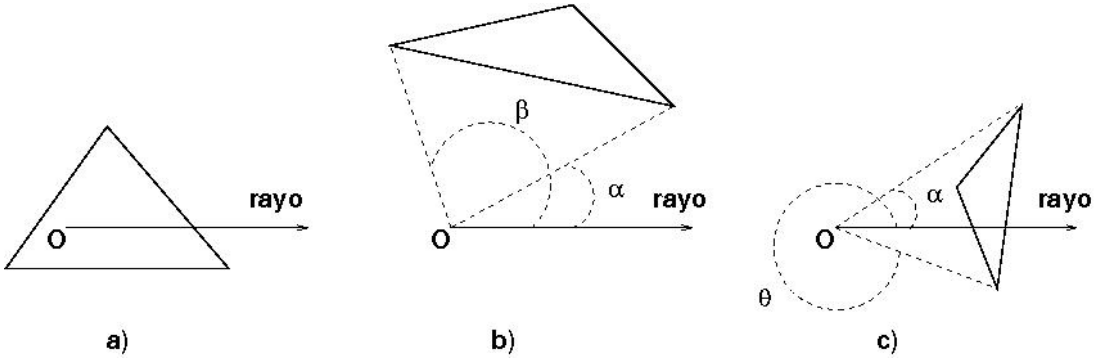

Figura 3: *Intervalos angulares ocupados por tres caras triangulares: a)* $[0,2\pi]$*, b)* $[\alpha,\beta]$*, c)* $[\theta,\alpha]$*. Caras y origen aparecen proyectados.*

4. Árboles de intervalos y descomposición espacial

En esta sección se presentará un test similar al de la sección previa, pero que se apoya en el uso de una descomposición espacial. Esto posibilita que se trabajen con árboles de intervalos de menor tamaño, y que devuelven, en general, un menor número de intervalos en sus consultas, lo que conlleva una mejora en la eficiencia del test.

La descomposicón espacial que se utilizará viene conformada por n semiplanos determinados por una línea r que atraviesa el punto origen, estos semiplanos dividen el espacio en n regiones, véase la Figura 5. Cada cara del poliedro se asociará con las regiones de la subdivisión del espacio que intersecta. Para obtener una descomposición adecuada a los objetivos del test, y que se adapte a la forma del poliedro, se realizará un barrido del espacio rotacional con un semiplano delimitado por la línea r, y que tiene a r como eje de rotación. Los eventos del barrido son los vértices de las caras del poliedro. Durante el barrido se eligen los semiplanos que constituyen la partición del espacio. Supongamos que S_x es el último semiplano seleccionado para la partición del espacio, y que actualmente el plano de barrido es S_y, ver Figura 6. Al realizar el barrido es fácil llevar la cuenta de las caras que caen en una de las siguientes categorías:

1. Situada entre los semiplanos S_x y S_y—cara b.
2. Que cortan únicamente a S_x—cara a.
3. Que cortan únicamente a S_y—cara c.
4. Que cortan a S_x y S_y—cara d.

Este registro proporciona una gran cantidad de información para determinar si S_y es una buena opción como semiplano de la partición. En concreto, se seleccionará un semiplano siempre que $n_1 > \frac{n_2+n_3+n_4}{2}$, donde n_x representa el número de caras que pertenecen a la categoría x. Las Figuras 7 y 8 muestran dos particiones del espacio empleando esta estrategia para orígenes situados en distintos lugares.

El conjunto de caras que intersectan una región de la partición del espacio se almacenan en un árbol de intervalos, de

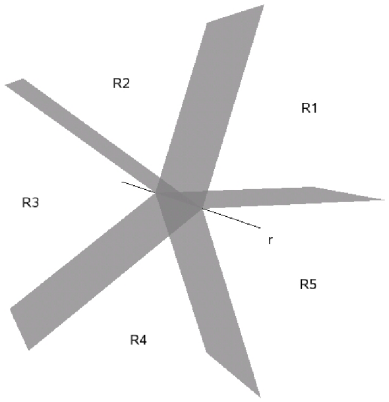

Figura 5: *División del espacio en 5 regiones.*

acuerdo con el intervalo angular ocupado por sus proyecciones en un plano que contiene a la recta origen de los semiplanos, ver Figura 9. Dado un punto de consulta q, una búsqueda binaria permite calcular la región de la división del espacio en que cae. A continuación se proyecta q sobre el plano de proyección para obtener el ángulo del segmento $\overline{Oq}$ con respecto a r, véase la Figura 9.b). Este ángulo se utiliza para consultar en el árbol de intervalos asociado a la región—como en el anterior test—las caras del poliedro que pueden intersectar con un rayo que parte del punto de consulta.

La Figura 10 muestra, en verde, el conjunto de caras devueltos por el árbol de intervalos asociado a la región que contiene un punto de consulta—el mismo que se empleó en la Figura 4. En este caso la recta origen de los semiplanos atraviesa el centroide del poliedro y se dirige desde la parte frontal a la trasera de la calavera—eje Z—, el plano de proyección para los árboles de intervalos asociados a las regiones es $X = O_x$, donde O es el origen. En la Figura 10 los

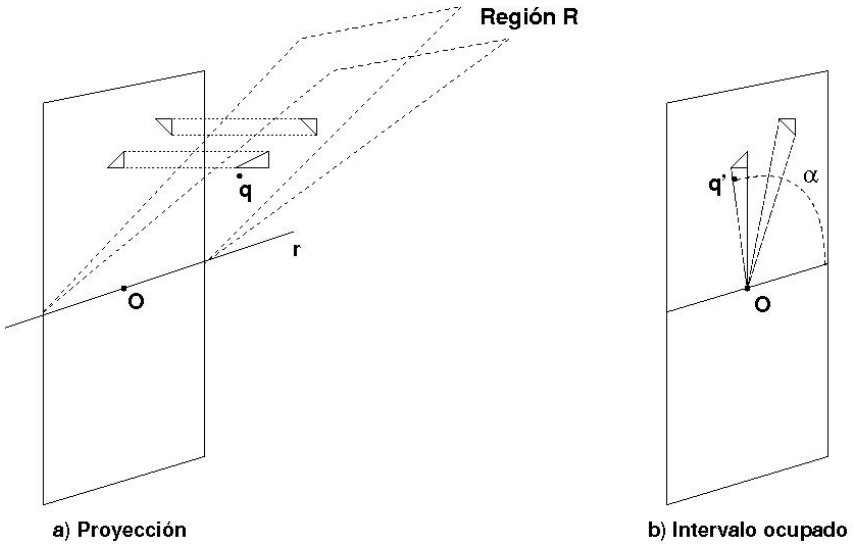

Figura 9: *Intervalo angular ocupado por caras triangulares.*

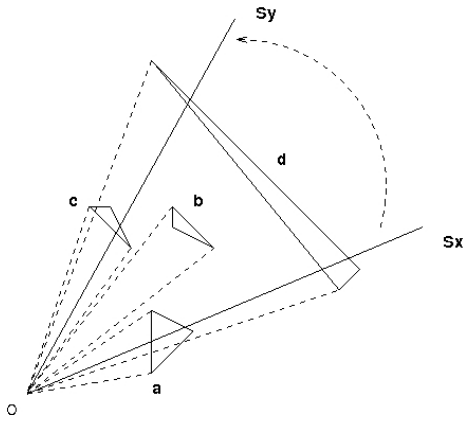

Figura 6: *Barrido del espacio rotacional proyectado sobre un plano perpendicular a r.*

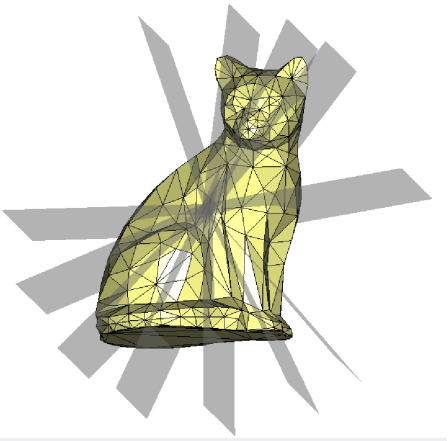

Figura 7: *División del espacio para un origen situado en el centroide.*

triángulos atravesados por los semiplanos de la división del espacio aparecen en rojo. Como puede observarse, el número de caras devueltas en las consultas de los árboles de intervalos dependerá del tamaño de las regiones de la división del espacio. A menor tamaño, menor número de caras devueltas. Esto sugiere disminuir el tamaño de las regiones, pero entonces se incrementa el tamaño de la representación, como se indica en la siguiente subsección. Por lo tanto, habrá que adoptar una solución de compromiso; en este sentido el uso de la condición $n_1 > \frac{n_2+n_3+n_4}{2}$ ha proporcionado buenos resultados.

4.1. Aspectos de implementación

De cara a implementar este test es preciso almacenar los semiplanos utilizados en la división del espacio—ordenados según el ángulo de barrido, para que pueda aplicarse la búsqueda binaria. Por cada región de la subdivisión del espacio hay que almacenar un árbol de intervalos para las caras que intersectan dicha región. Cierto número de caras intersectarán más de una región, por lo que incrementarán el tamaño de la representación.

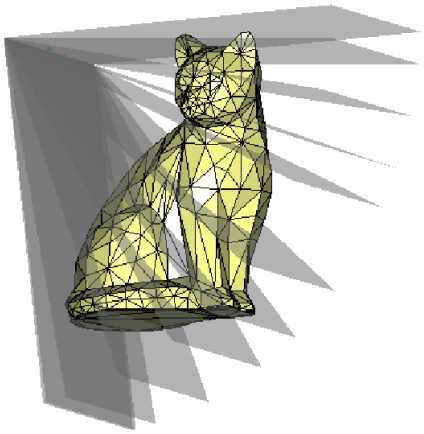

Figura 8: *División del espacio para un origen situado fuera de la caja envolvente mínima.*

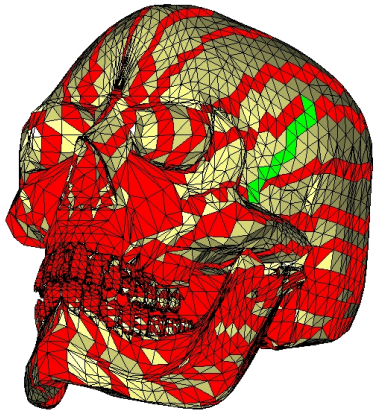

Figura 10: *Conjunto de caras devueltas (en verde) por una consulta al árbol de intervalos asociado a una región.*

5. Otros tests punto en poliedro

En esta sección se describen dos tests de inclusión punto en poliedro basados en descomposiones del espacio. En [OSF05] puede consultarse una descripción detallada de los tests, junto con una comparativa de estos tests y de otros más. En las siguientes subsecciones presentamos versiones optimizadas de los dos tests explicados en dicho artículo.

5.1. Método basado en un octree

De cara a utilizar un *octree* para un test de inclusión, cada nodo hoja debe almacenar su estado de inclusión: totalmente dentro, totalmente fuera, o parcialmente dentro. En el último caso el nodo hoja debe almacenar una lista de la caras del poliedro que intersectan el octante asociado al nodo. Dado un punto de consulta situado en la caja envolvente mínima del poliedro—si está fuera el punto no estará en el poliedro—, se utiliza la estructura arbórea del *octree* para encontrar el nodo hoja cuyo octante asociado contiene al punto. Si el estado de inclusión del nodo es totalmente dentro o totalmente fuera, entonces este será también el resultado del test. En otro caso se crea un segmento desde el punto de consulta a una de las esquinas del octante—cuyo estado de inclusión debe haberse computado cuando se construyó el *octree*. A continuación se calculan el número de intersecciones entre el segmento y la lista de caras del nodo—sólo estamos interesados en intersecciones dentro del octante. Una vez conocidos el número de intersecciones y el estado de inclusión de la esquina resulta trivial determinar el estado de inclusión del punto aplicando el teorema de la curva de Jordan.

La utilización del estado de inclusión de la esquina para octantes parcialmente incluidos en el poliedro está inspirado en el test de inclusión punto en polígono que utiliza una rejilla [Hai94]. El método basado en un *octree* implementado en [OSF05] calcula el número de intersecciones de un rayo, que parte del punto de consulta, con todas las caras que intersectan a los nodos hoja atravesados por el rayo. Obviamente, nuestra propuesta es mucho más eficiente.

Si el poliedro viene definido por caras triangulares se sugiere utilizar en la implementación el test de intersección rayo-triángulo descrito aquí [AMHH08].

5.2. Método basado en un árbol BSP

Resulta muy sencillo construir un árbol BSP alineado a polígono para un test de inclusión punto en poliedro [OSF05]. El espacio se divide recursivamente con los planos determinados por las caras del poliedro. Cuando un plano de división corta una cara del poliedro en dos fragmentos, cada fragmento se añade a su subespacio correspondiente. Si un subespacio sólo contiene una cara—o un fragmento de cara—del poliedro, entonces el plano determinado por la cara se utiliza como un nodo hoja, su subespacio negativo representa un subconjunto convexo del poliedro, y el subespacio positivo una región exterior al poliedro. Por lo tanto, el test de inclusión se reduce a una clasificación espacial recursiva del punto de consulta en cada nodo. Cuando se llega a un nodo hoja se utiliza su plano asociado para determinar el estado de inclusión, si el punto cae en el subespacio positivo entonces es externo al poliedro, en otro caso es interno.

A continuación, se listan las principales diferencias entre la implementación de este test que hemos realizado y la explicada en [OSF05]:

- De cara a construir el árbol BSP hemos utilizado un algoritmo recursivo, en lugar de emplear un algoritmo interactivo basado en una pila.
- Cuando un plano corta una cara de un poliedro, la cara se divide en dos fragmentos, cada fragmento se añade

a su subespacio correspondiente. En [OSF05] la cara se añade a ambos subespacios—es decir, no se divide en dos fragmentos. Nuestro enfoque reduce el tamaño del árbol y acelera el test de inclusión.

- De cara a obtener un árbol más equilibrado hemos seguido el consejo dado en [dBvKOS00], poniendo las caras del poliedro en orden aleatorio antes de comenzar la construcción del árbol. Esto reduce significativamente el tamaño del árbol, y acelera el test de inclusión.

6. Experimentación

En esta sección se muestran los resultados de algunas comparativas realizadas entre los distintos tests de inclusión punto en poliedro. En los experimentos se han utilizado los modelos de las Figuras 3, 7, 11, y 12. Las pruebas se realizaron en un ordenador con procesador Intel Core 2 a 2,40 GHz y una memoria RAM de 2 GB. En los tests basados en árboles de intervalos se ha elegido el centroide del poliedro como punto origen.

Figura 11: *Modelo del Buda feliz.*

Figura 12: *Modelo de dragón.*

La Tabla 3 muestra el tiempo empleado por los distintos algoritmos para calcular 1000 tests de inclusión punto en poliedro con puntos generados aleatoriamente dentro de la caja envolvente mínima del poliedro. La segunda columna indica el número de caras del modelo. El test de la suma de cruces es el más lento con diferencia. El que obtiene mejores resultados con todos los modelos es el test basado en el árbol BSP. El test basado en el *octree* también obtiene muy buenos resultados. En la implementación de este método se ha utilizado un umbral de tamaño en la creación del *octree*. Si la lista de caras que intersectan el octante asociado a un nodo es inferior al umbral, entonces el nodo no se subdivide más, convirtiéndose en un nodo hoja. En la tabla aparece entre paréntesis el umbral empleado. Como puede observarse, a menor umbral se obtienen mejores resultados. En general, el tiempo de ejecución de este método varía poco con el tamaño del modelo. Recuérdese que en este test se procesa un punto de consulta localizando la región de la subdivisión del espacio que contiene al punto—utilizando la estructura de árbol—, y comprobando el número de intersecciones con las caras que intersectan dicha región—el número máximo de caras viene determinado por el umbral comentado. Las dos últimas columnas de la tabla muestran los resultados de los tests basados en árboles de intervalos, la penúltima el test más sencillo, explicado en la Sección 3, y la última el que se apoya en la utilización de una descomposición espacial. Ambos tests obtiene buenos resultados, aunque conforme crece el tamaño del modelo el apoyo en la descomposición espacial aporta mayores beneficios.

La Tabla 4 muestra los tiempos de procesamiento requeridos por los diferentes enfoques. Aquí destaca positivamente el método basado únicamente en árboles de intervalos—recuérdese que su orden es $O(n \log n)$. El enfoque basado en el *octree* tiene un procesamiento previo muy costoso, que aumenta cuando se disminuye el umbral de tamaño empleado en su construcción.

7. Conclusiones y trabajo futuro

En este trabajo se han presentado varios algoritmos punto en objeto basados en el uso de árboles de intervalos como estructura de datos que aporta cierta información espacial, y permite descartar algunos lados o caras en un test de suma de cruces.

El test punto en poliedro proporciona una gran mejora con respecto al test de cruces, con un costo de procesamiento previo—$O(n \log n)$—y memoria—$O(n)$— moderado. La combinación del uso de árboles de intervalos y de una descomposición espacial proporciona unos tiempos de ejecución aún mejores. Sin embargo, se precisan más recursos de memoria y procesamiento previo.

Uno de los trabajos futuros consiste en desarrollar alguna estrategia o heurística para determinar planos de proyección adecuados para un modelo dado—en los tests punto en poliedro.

Tabla 3: *Tiempos de ejecución para 1000 tests de inclusión (en segundos)*

Modelo	Caras	Cruces	BSP	Octree (1000)	Octree (200)	Árboles	Árboles + descomp.
Gato	760	0.036	0.00029	0.054	0.0051	0.0025	0.0015
Calavera	21792	0.62	0.00077	0.017	0.0041	0.012	0.0035
Buda	99983	3.51	0.00083	0.022	0.0042	0.064	0.0069
Dragón	871414	28.98	0.00085	0.014	0.0024	0.13	0.0087

Tabla 4: *Tiempos de procesamiento previo (en segundos)*

Modelo	BSP	Octree (1000)	Octree (200)	Árboles	Árboles + descomp.
Gato	0.017	< 0.001	0.001	0.0008	0.001
Calavera	0.51	0.14	0.45	0.02	0.085
Buda	1.62	1.56	9.42	0.1	0.89
Dragón	7.62	130.4	790.3	1.1	16.8

Agradecimientos

Este trabajo ha sido parcialmente subvencionado por el Ministerio de Ciencia y Tecnología, así como por la Unión Europea, a través del proyecto de investigación TIN2007-67474-C03-03, y por la Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía y la Unión Europea, a través de los proyectos de investigación P06-TIC-01043 y P07-TIC-02773.

Quisiéramos agradecer también a los revisores por sus comentarios.

Referencias

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[dBvKOS00] DE BERG M., VAN KREVELD M., OVERMARS M., SCHWARZKOPF O.: *Computational Geometry Algorithms and Applications*, segunda ed. 2000.

[Hac62] HACKER R.: Certification of algorithm 112: position of point relative to polygon. *Communications of the ACM 5* (1962), 606.

[Hai94] HAINES E.: Point in polygon strategies. In *Graphics Gems IV*. Academic Press, 1994, pp. 24–46.

[OSF05] OGAYAR C. J., SEGURA R. J., FEITO F. R.: Point in solid strategies. *Computers & Graphics 29*, 4 (2005), 616–624.

[Shi62] SHIMRAT M.: Algorithm 112: position of point relative to polygon. *Communications of the ACM 5* (1962), 434.