INICIACIÓN A LA INVESTIGACIÓN
REVISTA ELECTRÓNICA
UNIVERSIDAD DE JAÉN
Ini Inv, 1: a8 (2006)

Algoritmos de Inclusión de Puntos en Mallas de Triángulos

Carlos J. Ogayar, Rafael J. Segura, Francisco R. Feito

Programa de Doctorado "Métodos y Técnicas Avanzadas de Desarrollo de Software". Departamento de Lenguajes y Sistemas Informáticos. Universidad de Granada. C./Periodista Daniel Saucedo Aranda s/n 18071 - Granada.
cogayar@ujaen.es, rsegura@ujaen.es, ffeito@ujaen.es

Resumen

Las mallas de triángulos se han convertido en el formato estándar de representación de objetos 3D en los campos de modelado y visualización. Esta forma de descripción de la información espacial para sólidos requiere la implementación de algoritmos robustos y eficientes para su procesamiento. En este trabajo se presenta un estudio de soluciones existentes para resolver el problema de clasificación de un punto en un sólido B-Rep definido mediante una malla de triángulos.

INTRODUCCIÓN

En la actualidad la mayoría de los objetos 3D se aproximan mediante mallas de triángulos. La razón de esto es la simplicidad de la estructura, y el hecho de que sirve para realizar una representación aproximada de cualquier sólido. En muchas áreas de la Informática Gráfica se utilizan mallas, como en modelado y en visualización. Por esta razón, es conveniente el desarrollo de algoritmos eficientes de tratamiento de mallas para distintas situaciones. Una de ellas es el test de inclusión de un punto en un sólido, que constituye una herramienta fundamental para otros procesos de mayor nivel, como CSG, detección de colisiones, mutirresolución, clasificación de elementos, conversión de representaciones, etc. Cuanto más eficiente y preciso sea el test de inclusión, mejores serán las soluciones que se implementen a mayor nivel.

Algoritmo de inclusión basado en el Teorema de la Curva de Jordan

Los métodos más utilizados para la inclusión de puntos en cualquier entidad geométrica están basados en el Teorema de la Curva de Jordan (O'Rourque J., Computational Geometry in C, Cambridge University, 1994). Esta técnica es extremadamente popular debido a su simplicidad y a su extenso campo de aplicación. Funciona lanzando un rayo desde un punto en una dirección aleatoria, con lo que se producirá un número de intersecciones con la frontera; si este número es impar, el punto queda dentro, y si es par, el punto queda fuera.

Los algoritmos de inclusión basados en el Teorema de la Curva de Jordan no necesitan estructuras de datos precalculadas. Esta forma está considerada como la más rápida para el test de inclusión sin preprocesamiento (Möller et al., Journal on Graphics Tools, 1997, 2:21-28), aunque aquí demostramos que el método de Feito-Torres es mejor con mallas de triángulos. El principal problema del método de Jordan es que el proceso debe ser repetido, cambiando la dirección del rayo, cuando se detecta una intersección con un vértice o una arista, así como en el caso

en que el punto está contenido en una cara o el rayo seleccionado es coplanar a uno de los triángulos. La elección del algoritmo de intersección rayo-triángulo influirá decisivamente en la eficiencia y precisión del algoritmo; las elecciones más acertadas son (Badouel D., An Efficient Ray-polygon Intersection, En: Graphic Gems. Academic Press, 1990, pp. 390-393) ó (Möller et al., Journal on Graphics Tools, 1997, 2:21-28).

Una optimización conveniente para acelerar el algoritmo de intersección rayo-triángulo utilizado en el método de Jordan consiste en la utilización de una estructura de segmentación espacial, como el BSP o el octree, que permiten descartar fácilmente un gran número de intersecciones. Para la navegación del octree los métodos de (Gargantini et al., Computer Graphics Forum, 1993, 12) y (Revelles et al., Journal of WSCG, 200, 8) son los más aconsejables.

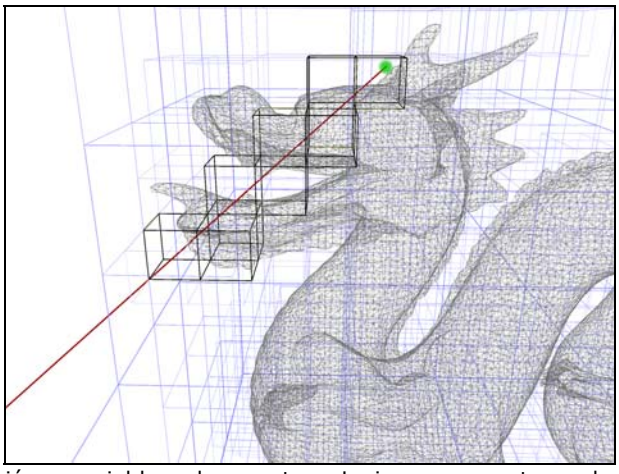

Ilustración 1. Optimización espacial basada en octree. La imagen muestra nodos hoja intersectados por un vector (rojo) desde un punto interior (verde). Los voxels marcados son el resultado de ejecutar Gargantini sobre el octree.

Algoritmo de inclusión basado en espacio de voxels

El algoritmo del grid ó espacio de voxels consiste en dividir el espacio utilizando una segmentación uniforme, mediante la cual se definen celdas contenidas total o parcialmente dentro del sólido. La inclusión de cada celda debe ser calculada con algún algoritmo adicional, como el basado en el teorema de la curva de Jordan. Lo que suele hacerse es probar la inclusión del centro de cada voxel, aunque pueden utilizarse métodos más complejos, como el muestreo de varios puntos contenidos en el voxel para establecer el estado de inclusión mayoritario (inclusión total con imprecisión), o bien determinar un estado de inclusión parcial. También pueden utilizarse métodos estocásticos y probabilísticos para determinar la inclusión usando varias muestras. Los principales problemas del grid son la dependencia respecto de un algoritmo adicional de inclusión para calcular el estado de los voxels, así como el espacio 3D que se desperdicia como consecuencia de una segmentación uniforme. Como ventaja, hay que destacar la rapidez del algoritmo de recorrido de la estructura.

Algoritmo de inclusión basado en Octree

En esta ocasión, la estructura se calcula teniendo en cuenta la distribución del sólido, de forma que se generan octantes de forma recursiva en aquellas partes donde se concentran los polígonos. Esto produce un gran ahorro de memoria, y

sobre todo, elimina el número de tests de inclusión para construir la estructura, ya que habrá muchos menos octantes que voxels en el grid equivalente. Como con el espacio de voxels, aparecen problemas de aliasing y el rendimiento vuelve a estar en función del algoritmo seleccionado para la inclusión inicial de puntos pertenecientes a cada octante. En esta ocasión, el recorrido por la estructura no es tan eficiente como en el grid, aunque existen métodos paramétricos como los descritos que reducen esta diferencia drásticamente.

La navegación por el octree es trivial y por tanto, una vez construido el octree que representa al sólido, el test de inclusión de un punto en un subespacio es tan sencillo como profundizar en el árbol hasta llegar a un nodo raíz válido, o ser descartado en el proceso, lo que reduce la complejidad de la clasificación a O(log n).

Algoritmo de inclusión basado en BSP

La partición binaria del espacio (BSP) es una estructura recursiva que divide el espacio clasificando los elementos contenidos en el mismo. Al clasificar un punto como exterior ó interior utilizando esta estructura, basta con recorrer el árbol comprobando en cada nodo la posición relativa del punto respecto del plano asociado a dicho nodo, hasta llegar a un nodo hoja. Los nodos que representan cada subespacio en un nivel determinado del árbol binario están etiquetados de igual forma que el signo del subespacio al que representan; positivo y negativo. Cuando se llega a un nodo hoja, utilizando el plano asociado, se comprueba la posición del punto; si queda en el subespacio positivo el punto se considera fuera del sólido, si queda en el subespacio negativo, se considera dentro. En el caso de estar situado sobre el plano de corte, es cuestión de criterio considerar su posición. Para la inclusión de puntos en mallas de triángulos, se utilizan BSP alineados con los polígonos. Los nodos hoja en la estructura del BSP contienen subconjuntos convexos del sólido. El test de inclusión de puntos es muy eficiente cuando se utiliza un BSP. El problema es la gran cantidad de memoria consumida.

Algoritmo de inclusión de Feito-Torres

El algoritmo de inclusión de Feito-Torres (Feito et al., Computer & Graphics, 1997, 21:23-30) es un método que comprueba la inclusión de un punto en un poliedro genérico sin resolver ningún sistema de ecuaciones. Dado un sólido definido mediante una malla triangular y un punto para calcular su inclusión en aquél, el algoritmo de Feito-Torres se basa en la descomposición del sólido en símplices, que para el caso 3D resulta en un conjunto de tetraedros. Cada tetraedro está formado por los vértices de cada triángulo de la malla que representa al sólido y un punto adicional, común a todos los tetraedros del conjunto. Para simplificar los cálculos, y sin perder generalidad, se utiliza el origen de coordenadas como el vértice adicional para cada tetraedro. A este tipo de tetraedro se le llama tetraedro original.

Considerando el conjunto de tetraedros originales asociados al sólido, se selecciona el subconjunto de tetraedros en los cuales el punto a testar está incluido (los tetraedros pueden solaparse). A continuación, se calcula el signo de cada uno de ellos, y se realiza la sumatoria. Este resultado numérico determinará si el punto está dentro (>0) o fuera (≤ 0). La utilización de tetraedros originales viene motivada por la simplificación del cálculo de sus signos, que se realiza mediante determinantes.

Tanto el algoritmo de Jordan como los basados en segmentación espacial sufren una penalización extra asociada a la complejidad de la malla de triángulos, ya sea por el aumento de casos especiales en el primero, como de utilización de memoria en los últimos. El algoritmo de Feito-Torres no requiere de preprocesamiento y es totalmente independiente de la topología del sólido, si bien hay ciertas optimizaciones opcionales (relacionadas con el preprocesamiento) que aceleran drásticamente los cálculos. Además, es válido para modelos con agujeros y multirresolución. Se puede optimizar todo el proceso mediante un sencillo preprocesamiento que precalcula la mayoría de los datos necesarios durante la ejecución del algoritmo. Estas tareas opcionales son:

– Centrar el sólido en el origen de coordenadas y calcular los octantes, respecto del origen de coordenadas, que intersectan cada cara. Cuando se realiza el test de inclusión se comprueba si cada octante y el punto a incluir comparten octantes, permitiendo descartes inmediatos.

– Precalcular las cajas envolventes de cada tetraedro original (formado por los vértices de cada triángulo y el origen de coordenadas).

– Precalcular el signo de cada tetraedro original.

Nótese, que cada optimización es independiente de las demás, lo que permite seleccionar al usuario las más convenientes en función de la memoria disponible.

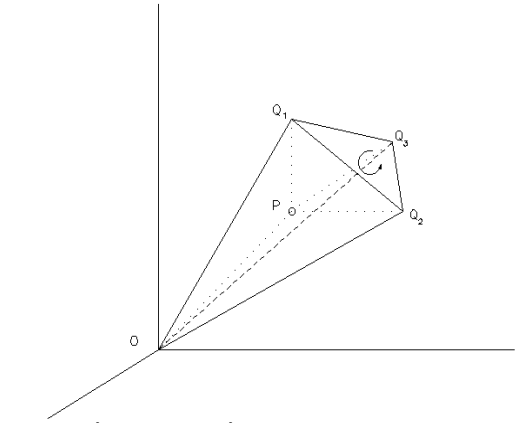

Ilustración 2. Inclusión de un punto en un tetraedro.

	Grid	Octree	Polygon Aligned BSP	Jordan	Feito-Torres
Representación del sólido	Aproximada	Aproximada	Exacta	Exacta	Exacta
Complejidad de inclusión	O(k)	O(log n)	O(log n)	O(n)	O(n)
Eficiencia	Muy alta	Alta	Alta	Normal	Normal
Estructura de segmentación espacial asociada	Muy costosa	Costosa	Costosa/Muy costosa	Ninguna	Ninguna
Adaptación a inclusión	Baja	Baja	Alta	Alta	Alta

Tabla 1. Características de los métodos estudiados y su conveniencia para la inclusión de puntos en mallas de triángulos.

Implementación de los algoritmos de inclusión

Los algoritmos que se han probado son Jordan, Jordan optimizado con octree, Feito-Torres, Feito-Torres optimizado y una solución basada en BSP. No se presentan resultados utilizando algoritmos basados en clasificadores espaciales (espacio de voxels y octree), ya que los resultados no son exactos, y ante todo importan los métodos robustos y precisos. Las implementaciones se han optimizado razonablemente, ya que es interesante contar con la mejor de las versiones posibles.

BSP. La implementación del BSP es bastante convencional. Para ir subdividiendo el espacio se toman los planos donde están contenidas las caras del sólido. Si un plano divisor corta algún polígono, el plano que contiene al mismo es considerado en los dos subespacios resultantes; esto produce un árbol de gran tamaño con modelos muy complejos, especialmente los que cuentan con muchas concavidades. La versión del algoritmo de creación del BSP es iterativa y mucho más eficiente en tiempo y en el uso de la memoria que la clásica recursiva.

Jordan. Para aplicar el teorema de la curva de Jordan se ha implementado como algoritmo de intersección rayo-triángulo el de Möller-Trumbore, modificado para detectar con un error determinado cuándo se produce una intersección con un vértice o una arista, y cuándo el rayo es coplanar al triángulo a intersectar. Cada vez que se produce un caso especial, se repite el proceso utilizando un vector aleatorio distinto. Este es, sin duda, el punto débil del algoritmo, que obliga a utilizar una gran precisión decimal.

			Feito-Torres (opt)		**Jordan+octree6**		**BSP**	
	Vértices	*Triángulos*	*tiempo*	*memoria*	*tiempo*	*memoria*	*tiempo*	*memoria*
Celtic Cross	1849	2366	<0.001s	61Kb	0.079s	135Kb	0.001s	408Kb
Hydrant	15822	5786	<0.001s	150Kb	0.188s	270Kb	1.282s	4.6Mb
Battery	4763	9522	<0.001s	285Kb	0.281s	401Kb	1.500s	720Kb
Mobile Phone	13025	25946	0.015s	674Kb	0.578s	281Kb	54.514s	158Mb
Golf Ball	23370	46205	0.031s	1.2Mb	1.188s	861Kb	72.484s	2.7Mb
V8 Engine	76214	152553	0.132s	3.9Mb	4.469s	2.1Mb	514.141s	576Mb
Venus Sculpture	139217	277512	0.235s	7.2Mb	6.687s	1.4Mb	442.109s	328Mb
Seat	282535	564960	0.502s	14.7Mb	15.219s	3.5Mb	mem_error	mem_error
Chinesse Dragon	437645	871414	0.782s	22.5Mb	27.766s	4.2Mb	mem_error	mem_error

Tabla 2. Datos sobre cada modelo 3D utilizado y su coste en preprocesamiento. La unidad de tiempo es el segundo.

Para mejorar el rendimiento se ha utilizado en la versión mejorada un octree como optimizador espacial. Partiendo del mínimo cubo envolvente, se generan octantes que se subdividen como máximo hasta una profundidad determinada, de forma que para cada nodo hoja hay asociada una lista de polígonos que lo intersectan. Esta es la variante del octree que propone Glassner para ser utilizada

como optimizador en el ray-casting. Para recorrer el octree (traversing), se ha utilizado el algoritmo de Gargantini.

Feito-Torres. La implementación del algoritmo de Feito-Torres utiliza tan sólo una función para comprobar la inclusión de un punto en un tetraedro original, y la función principal para comprobar la suma de los volúmenes de los tetraedros originales que incluyen el punto de prueba. En cuanto a la optimización, ésta consiste en una estructura donde se almacenan al código de octante, la caja envolvente y el signo de cada tetraedro original; toda esta información será invariable ante el cambio de punto a incluir.

Estudio comparativo

Las comparativas se realizaron con 9 modelos que varían su complejidad poligonal desde los ~2500 triángulos hasta los ~875000. Algunos de ellos presentan una gran cantidad de concavidades y agujeros, ideales para poner a prueba los métodos dependientes de la forma.

En la Tabla 2 se indican los datos de los modelos utilizados. La máquina de prueba fue un Pentium 4 a 2GHz con 512Mb de RAM trabajando con Windows XP.

Jordan. El algoritmo de Jordan se muestra menos eficiente cuando hay una gran densidad de vértices y polígonos, y muy especialmente cuando se presentan concavidades, como en el caso del modelo V8-Engine. La versión optimizada con octree (de profundidad 6 en las pruebas realizadas) mejora enormemente los tiempos, si bien necesita un tiempo de preprocesamiento moderado y cierta cantidad de memoria. En este caso, los tiempos son sensibles al reparto de los polígonos entre los octantes del octree, que a veces puede no ser tan eficiente.

	Feito	Jordan	Feito(op)	Jordan(op)	BSP
Celtic Cross	0.875s	0.969s	0.063s	0.021s	0.001s
Hydrant	2.188s	2.407s	0.172s	0.031s	0.002s
Battery	3.578s	3.907s	0.375s	0.029s	0.007s
Mobile Phone	9.672s	10.953s	1.359s	0.172s	0.041s
Golf Ball	17.907s	19.841s	2.406s	0.062s	0.367s
V8 Engine	55.152s	639.841s	7.469s	0.359s	0.167s
Venus Sculpture	119.372s	129.712s	13.910s	1.031s	0.008s
Seat	237.971s	264.324s	31.409s	1.219s	mem_error
Chinesse Dragon	336.879s	3759.402s	45.942s	1.781s	mem_error

Tabla 3. Tiempos de CPU para la inclusión de 1000 puntos. La unidad de tiempo es el segundo.

BSP. El BSP es, sin duda, el algoritmo más rápido de clasificación espacial. Su inconveniente es el coste extremo en preprocesamiento cuando se trata con modelos complejos, tanto en tiempo de cálculo como en memoria. El problema principal aparece cuando el plano asociado a un triángulo de la malla corta a otros polígonos, en cuyo caso hay que añadir sus respectivos planos a los dos subespacios resultantes. Esto ocasiona un crecimiento desmesurado del árbol BSP, lo que suele ocurrir cuando se presentan concavidades y agujeros en el modelo. BSP es muy dependiente de la forma del sólido.

Feito-Torres. Los tiempos de los tests son mejores comparados con los óptimos de Jordan; en torno al -10%. En el caso de mallas complicadas, Jordan sufre sus casos especiales y repite varias veces los cálculos, hasta llegar a resultados realmente pobres; Feito-Torres sin embargo, es independiente de la forma del sólido, y mantiene una relación lineal directa con el número de polígonos, sin importar la cantidad de convexidades y agujeros que se presenten. La versión optimizada de Feito-Torres proporciona una reducción sustancial de los tiempos (un ~85% menos) y proporciona un gasto de memoria moderado, si bien el tiempo de preprocesamiento es ínfimo.

En la Tabla 3 se muestran los tiempos de ejecución obtenidos para cada uno de los métodos estudiados. Las pruebas se realizaron generando 1000 puntos aleatorios incluidos en la caja envolvente del sólido. En la Tabla 4 se muestra una valoración cualitativa de cada uno de los métodos en función de ciertos criterios como la eficiencia, el coste del preprocesamiento, la dependencia de la forma del sólido en el rendimiento, la escalabilidad y la dificultad de implementación. Por escalabilidad se entiende la relación directa entre el aumento de la complejidad del sólido y el tiempo de cálculo del algoritmo. En la tabla se menciona la escalabilidad del peor y mejor caso; esto está pensado para algoritmos como el basado en el teorema de la curva de Jordan o el basado en el BSP, que dependerán de varios factores para su desempeño, como por ejemplo la forma del sólido.

	Feito	**Jordan**	**Feito(op)**	**Jordan(op)**	**BSP**
Eficiencia	buena (6)	moderada (5)	bastante buena (7)	muy buena (8)	excelente (9)
Tiempo de preprocesamiento	nada (9)	nada (9)	muy bajo (8)	moderado (5)[1]	muy alto (2)[2]
Memoria de preprocesamiento	nada (9)	nada (9)	moderado (5)	moderado (6)[1]	muy alto (2)[2]
Dependencia de la forma del sólido	ninguna (9)	moderada (5)	ninguna (9)	moderada (5)	extrema (0)
Escalabilidad (mejor caso)	excelente (9)	bastante buena (7)	excelente (9)	muy buena (8)	excelente (9)
Escalabilidad (peor caso)	excelente (9)	pobre (2)[3]	excelente (9)	pobre (3)[3]	muy pobre (1)[4]
Dificultad de implementación	muy baja (9)	baja (7)	baja (8)	normal (6)	normal (5)

Tabla 4. Características de cada método de inclusión implementado.
Los valores de calidad varían de 0 (peor) a 9 (mejor) 1) Depende de la profundidad del octree 2) Depende de la forma del sólido 3) Debido a casos especiales en las intersecciones 4) Debido a concavidades y agujeros

Como conclusión cabe destacar la conveniencia de cada algoritmo para distintas circunstancias. El mejor algoritmo sin preprocesamiento es Feito-Torres. Las soluciones intermedias en utilización de recursos son Feito-Torres optimizado y Jordan con octree (de profundidad media). Si la memoria lo permite y se realizan muchos tests de inclusión, de forma que se compensa el preprocesamiento, BSP es con mucho la mejor opción.

Ilustración 3. Modelos 3D utilizados para las pruebas.