Introducción a la detección de colisiones entre objetos convexos en tiempo real mediante el algoritmo de distancia de Gilbert-Johnson-Keerthi

Emmanuel Rojas Fredini, Franco Victorio, y Néstor Calvo

Facultad de Ingeniería y Ciencias Hídricas, Universidad Nacional del Litoral, Santa Fe, Argentina
{erojasfredini,victorio.franco,nestor.calvo}@gmail.com
http://fich.unl.edu.ar/

Resumen La detección de colisiones es un problema de suma importancia en la programación de videojuegos y otros ámbitos. En su forma más sencilla, se trata de determinar si hay o no una intersección entre dos o más objetos. En aplicaciones de tiempo real, al problema de evaluar muchas colisiones potenciales se suma el hecho de necesitar hacerlo una vez por frame. El algoritmo de GJK proporciona un método eficiente para tratar con esta situación, ya que utiliza los resultados de cada frame para hacer la evaluación más rápidamente en el frame siguiente. En el presente trabajo se introduce el método y una implementación en C++, y se discuten posibles usos y mejoras para ésta.

Keywords: Detección de Colisiones, Resta de Minkowski, Símplices, Puntos de Soporte, Coherencia Temporal, GJK

1. Introducción

El problema de la detección de colisiones consiste en determinar si dos o más objetos se están intersectando. Esto surge especialmente en la programación de videojuegos[1][4], pero también se aplica en otros ámbitos[2] como simulaciones físicas de ingeniería, modelado y animación 3D, sistemas CAD/CAM, robótica, entre otros. En su forma más sencilla, el problema es determinar si hay o no una intersección, pero también puede necesitarse saber cuándo y dónde se produce, siendo estas dos últimas de complejidad creciente. La realización de estos cálculos se suele llamar determinación de colisión.

En particular en los videojuegos, la detección de colisiones contribuye a generar la ilusión de "realidad" en el mundo virtual; por ejemplo, previniendo que los personajes caigan a través de los pisos o atraviesen las paredes y permitiendo realizar consultas espaciales como puede ser la línea de visión de los enemigos. Un requisito fundamental en esta aplicación es que la detección se realice en tiempo real, característica que no es siempre necesaria por ejemplo en las simulaciones de ingeniería o en las animaciones. En general es necesario que un

gran número de figuras de colisión sean procesadas por el sistema de detección de colisiones a una tasa de 30 o 60 veces por segundo ya que la mayoría de los videojuegos, sean plataforma PC, consolas de sobremesa o móviles, poseen una tasa de frames per second (fps) en este rango. Esto hace que la detección de colisiones en el contexto de los videojuegos sea un proceso costoso computacionalmente hablando, siendo necesario dedicarle a ésta un porcentaje importante del tiempo de procesamiento del que se dispone. De esta manera, un algoritmo de detección de colisiones pobremente diseñado puede resultar en un cuello de botella en la performance del videojuego.

En el presente trabajo se presenta el método de GJK para obtener la distancia entre dos objetos convexos, el cual puede ser utilizado para resolver el problema de la detección de colisiones. Primero se introduce el método, explicando en detalle el algoritmo para calcular la distancia y examinando sus principales características. Luego se comenta la implementación en C++ que se realizó. Finalmente, se exponen las conclusiones y se discuten los trabajos futuros que pueden realizarse.

2. Métodologia

A continuación se explican brevemente algunos conceptos necesarios para comprender el método y luego se describe en detalle el algoritmo. En el texto se considerará equivalente vector y punto pues se considera dado un sistema de referencia ortonormal arbitrario con lo cual entenderemos por "punto" al vector posición del mismo.

2.1. Conceptos previos

- Convexidad: se dice que un polígono $\mathcal{P}$ es convexo si todos los segmentos de línea entre dos puntos cualesquiera de $\mathcal{P}$ están completamente dentro de $\mathcal{P}$. En la Fig. 1 pueden verse ejemplos de esto.

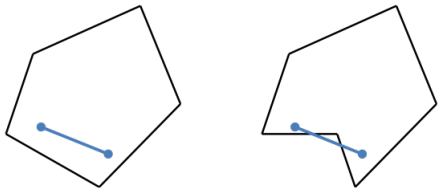

Figura 1. A la izquierda se muestra un polígono convexo. A la derecha se muestra un polígono cóncavo.

- Combinación afín: Es la combinación lineal de N puntos de forma tal que los coeficientes sumen uno; si los coeficientes además están entre cero y uno se

denomina combinación convexa. Un conjunto de puntos tiene independencia afín si ningún punto puede ser expresado como combinación afín del resto.

$$CA: \qquad P = \textstyle\sum_{i=1}^{N} P_i\alpha_i \quad | \quad \sum_{i=1}^{N} \alpha_i = 1$$

$$CC: \qquad P = \textstyle\sum_{i=1}^{N} P_i\alpha_i \quad | \quad \sum_{i=1}^{N} \alpha_i = 1 \quad \wedge \quad \alpha_i \geq 0 \quad \forall i \in [1, N]$$

- Casco convexo (*Convex hull*): Es el conjunto de combinaciones convexas que pueden resultar de un conjunto fijo de puntos; equivalentemente es la intersección de todos los semi-espacios que contienen al conjunto.
- Símplice (*simplex*): un d-símplice es el casco convexo de $d + 1$ puntos con *independencia afín* en un espacio de d o más dimensiones. Un símplice es un d-símplice para un d dado. Por ejemplo, el 0-símplice es un punto, el 1-símplice es un segmento de línea, el 2-símplice es un triángulo y el 3-símplice es un tetraedro.
- Punto de soporte: dado un objeto, un punto (no necesariamente único) que está máximamente alejado en una dirección $\bar{D}$ dada se llama punto de soporte en dicha dirección. Es decir, un punto $\bar{P}$ que maximiza $\bar{P} \cdot \bar{D}$.
- Resta de Minkowski: Sean $\mathcal{A}$ y $\mathcal{B}$ dos conjuntos de puntos y sean A y B los vectores posición correspondientes a pares de puntos en $\mathcal{A}$ y $\mathcal{B}$. La resta de Minkowski se define como $\mathcal{A} \ominus \mathcal{B} = \{A - B : A \in \mathcal{A}, B \in \mathcal{B}\}$. Esta operación se ilustra en la Fig. 2.
- Punto de soporte de la resta de Minkowski: dados dos conjuntos de puntos $\mathcal{A}$ y $\mathcal{B}$, un punto de soporte V de su diferencia de Minkowski para una dirección D está definido como:

$$V = A - B \quad | \quad (A - B) \cdot D \geq (A^* - B^*) \cdot D \qquad A \in \mathcal{A},\ B \in \mathcal{B},\ \forall\ A^* \in \mathcal{A},\ \forall\ B^* \in \mathcal{B}$$

Los puntos A y B son entonces aquéllos que maximizan $(A - B) \cdot D$. Este máximo se puede desarrollar de la siguiente manera:

$$\begin{aligned} \text{máx}\,((A - B) \cdot D) &= \text{máx}\,((A \cdot D + B \cdot (-D)) \\ &= \text{máx}\,(A \cdot D) + \text{máx}\,(B \cdot (-D)) \end{aligned}$$

En este último resultado se observa que el A que maximiza el primer término corresponde al punto de soporte de $\mathcal{A}$ en la dirección D y el B que maximiza el segundo término corresponde al punto de soporte de $\mathcal{B}$ en la dirección $-D$.

2.2. Descripción del método

El algoritmo de Gilbert-Johnson-Keerthi es un método eficiente para determinar si un par de objetos convexos se intersectan. Es un algoritmo iterativo y

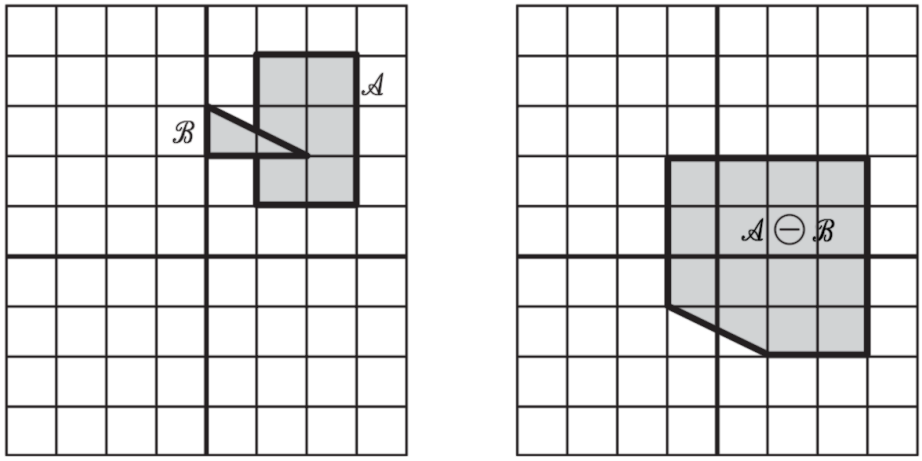

Figura 2. En la sección izquierda se muestran los polígonos originales A y B. En la sección derecha se muestra el resultado de la resta de Minkowski. Imagen tomada de [1].

basado en símplices que obtiene la distancia euclídea y los puntos más cercanos entre los objetos. Los únicos requisitos para los objetos son la convexidad y la posibilidad de obtener puntos de soporte para cualquier dirección dada. Es decir que los objetos pueden ser poliedros (o politopos) o cualquier objeto descripto analíticamente.

El algoritmo de GJK no utiliza los convexos directamente; en su lugar, trabaja con la diferencia de Minkowski entre ellos, ya que la distancia de ésta al origen es igual a la distancia entre los objetos. Esto se debe a que la distancia al origen de un punto cualquiera, $A - B$, de la diferencia de Minkowski es precisamente la norma de $A - B$ (la distancia entre A y B); por lo tanto, el punto más cercano al origen es aquél formado por el par de puntos más cercanos de los objetos y define la distancia entre los mismos. Si la diferencia de Minkowski contiene al origen, entonces hay intersección.

El algoritmo no realiza el cálculo de la diferencia de Minkowski de forma explícita, ya que este cálculo no es trivial y requiere de un procesamiento costoso. En cambio, se obtienen puntos de soporte de la diferencia de Minkowski de forma determinística. Con éstos se forman símplices de hasta $d + 1$ puntos, donde d es la dimensión en la que se está trabajando. Los puntos se eligen en cada iteración de forma tal que cada nuevo símplice esté más cerca del origen, hasta contenerlo o no poder acanzarlo.

En problemas simples de colisión entre objetos (videojuegos) la dimensión del espacio es dos o tres. En cambio en problemas de robotica como motion planning puede llegar a ser muy grande. Si los objetos no fueran convexos se pueden utilizar algoritmos de subdivisión en convexos o cualquier técnica adecuada.

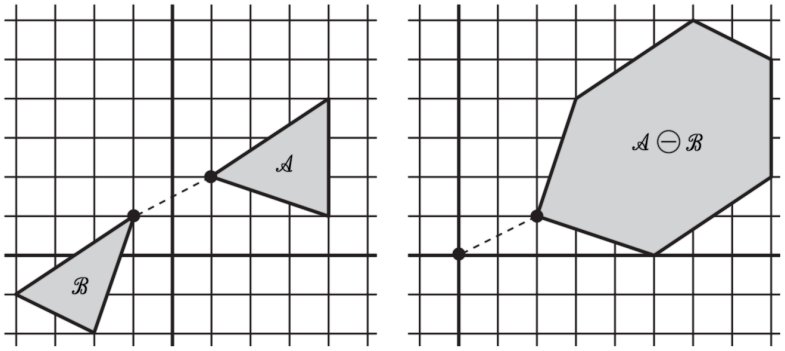

Figura 3. A la izquierda se ven dos convexos, A y B, separados por una distancia. A la derecha se puede ver la diferencia de Minkowski entre los convexos, donde la menor distancia al origen es igual a la distancia entre los polígonos. Imagen tomada de [1].

2.3. Algoritmo

Sea $\mathcal{C}$ la diferencia de Minkowski entre los dos objetos que se están evaluando. En cada iteración, el algoritmo de GJK mantiene un conjunto de hasta $d + 1$ puntos de soporte de $\mathcal{C}$, cuyo casco convexo forma un símplice y donde d es la dimensión en la que se está trabajando. Estos puntos de soporte se obtienen como se explicó en la sección 2.1. Si este símplice contiene al origen, entonces los objetos se intersectan; en caso contrario, se realiza otra iteración en la cual el símplice se actualiza, obteniéndose uno nuevo del cual se garantiza que estará más cerca del origen que el anterior. El algoritmo finaliza cuando el símplice contiene al origen o cuando no puede obtenerse un símplice más cercano al origen. En este último caso, el punto más cercano al origen se encuentra a una distancia igual a la distancia entre los objetos. A continuación se explica paso a paso el funcionamiento del algoritmo:

1. Se inicializa con un símplice $\mathcal{Q}$ formado por uno o más puntos de soporte de la diferencia de Minkowski.
2. Se calcula el punto P de $\mathcal{Q}$ más cercano al origen.
3. Si P es el origen entonces los convexos están intersectando y el algoritmo termina informando de la intersección.
4. Se eliminan de $\mathcal{Q}$ los puntos que no se requieren para definir P como combinación convexa.
5. Se calcula un nuevo punto de soporte V de la diferencia de Minkowski en la dirección $-P$.
6. Si V no está más cerca del origen que P, termina el algoritmo ya que los convexos no se intersectan. Se informa de la distancia entre $\mathcal{A}$ y $\mathcal{B}$, que está dada por la norma de P.
7. Se agrega V a $\mathcal{Q}$ y se vuelve al paso 2.

En la Fig. 4 se puede ver un ejemplo en 2D de la ejecución del algoritmo antes detallado. En ésta se puede observar cómo se busca el punto más cercano

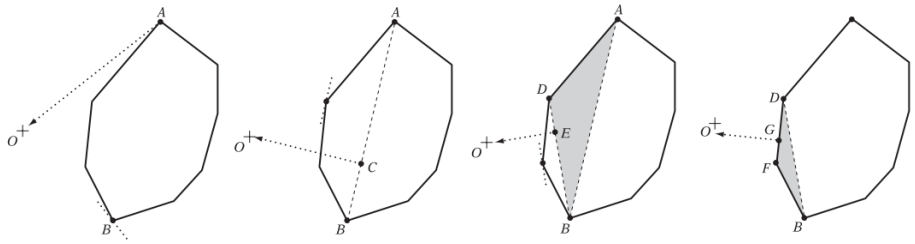

Figura 4. Ejemplo de ejecución del algoritmo. Imagen tomada de [1].

al origen O en sólo unas pocas iteraciones. El algoritmo comienza eligiendo arbitrariamente el punto A como el símplice inicial, i.e. $\mathcal{Q} = \{A\}$. Al estar $\mathcal{Q}$ compuesto sólo por el punto A, éste es el que tiene la norma mínima. Luego se busca un nuevo punto en la dirección $-A$, obteniéndose el punto B. El nuevo punto es agregado al símplice quedando éste: $\mathcal{Q} = \{A, B\}$. Ahora el punto con norma mínima en el símplice es el punto C, que se encuentra entre los puntos A y B. Como el nuevo punto C solo puede ser formado por la combinación convexa de los puntos A y B, $\mathcal{Q}$ no es reducido. A continuación se busca el punto más alejado en la dirección $-C$, dando como resultado el punto D, y se agrega al símplice quedando éste: $\mathcal{Q} = \{A, B, D\}$. Una vez más se busca el punto de menor norma en el símplice, obteniéndose así el punto E. El símplice $\mathcal{Q}$ se reduce eliminando el punto A ya que E se puede formar como combinación convexa de los puntos D y B. Se obtiene el punto más alejado en la dirección $-E$, consiguiéndose el punto F, el que se agrega al símplice quedando éste: $\mathcal{Q} = \{B, D, F\}$. Se calcula el punto de norma mínima G en el símplice actual. El símplice $\mathcal{Q}$ se simplifica eliminándose el punto B ya que no es imprescindible para obtener G y se busca el punto más alejado en la dirección $-G$. Esta vez el punto obtenido no es más cercano al origen que G, por lo que el algoritmo finaliza, devolviendo que las figuras no colisionan y que su distancia mínima de separación es $|G|$.

2.4. Características

Una característica muy importante cuando estamos tratando simulaciones físicas es que en general, de iteración a iteración el cambio de posición de los objetos no es tan grande. A esta característica se la suele llamar coherencia temporal y puede ser explotada por el algoritmo para una convergencia al resultado mucho más rápida. Como se detalló en la sección anterior, el algoritmo de GJK es un algoritmo iterativo y, como tal, un aspecto fundamental es cómo iniciar la primera iteración. Si se está trabajando con una aplicación en tiempo real, en cada frame se debe correr el algoritmo y, por lo tanto, inicializarse. Una ventaja de GJK es la posibilidad de utilizar el símplice de la última iteración del frame anterior para inicializar el algoritmo en el nuevo frame. Esto en general logra que, luego del primer frame, la detección de colisión se realice en tiempo prácticamente constante.

Si entre una iteración y la siguiente el cambio de posición de los objetos puede ser significativamente grande, se dan dos problemas. Por un lado, se pierde la ventaja de la coherencia temporal antes mencionada. Por el otro, puede darse el fenómeno de *tunneling*: esto sucede cuando existe una colisión que sa da entre dos instantes de tiempo sucesivos, pero no es detectada en ninguno de ellos (por ejemplo, una bala atravesando una pared). Para evitarlo se debe utilizar alguna técnica de detección continua de colisiones.

Se puede analizar la cota superior del costo computacional del algoritmo para la primera iteración analizando la cantidad de triángulos en que se puede dividir la diferencia de Minkowski para problemas de dos dimensiones. Si los dos polígonos convexos de entrada tienen n y m vértices respectivamente, la diferencia de Minkowsky es un convexo de $n+m$ vértices y por lo tanto se puede subdividir en $n+m-2$ triángulos[5]. Dado que para las siguientes iteraciones se aprovecha la coherencia temporal, el costo es variable en función de la granularidad de la discretización temporal. La complejidad computacional en casos prácticos es constante esperado[1].

3. Implementación

Para el presente trabajo se implementó el método de GJK en C++, utilizando la librería gráfica SFML para la visualización. El programa muestra dos ventanas: en una, dos polígonos de ejemplo pueden moverse, mientras en la otra se muestra la resta de Minkowski correspondiente. Ésta se muestra por motivos didácticos ya que, como se mencionó anteriormente, no es calculada por el método. Además, cuenta con la posibilidad de mostrar paso a paso cómo se busca el punto de menor norma en el espacio de Minkowski. En las Fig. 5 y 6 se pueden ver capturas de la aplicación.

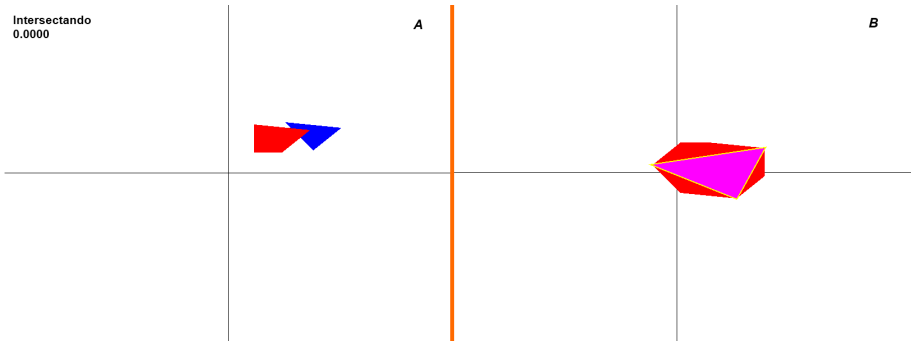

Figura 5. En el recuadro A hay dos polígonos colisionando en el programa; en la esquina superior izquierda informa que están intersectando y abajo la distancia a la que se encuentran los polígonos (que es cero por estar intersectando). En el recuadro B se puede ver la resta de Minkowski de éstos, y en violeta el símplice final que encuentra que encierra al origen.

La implementación se realizó en dos dimensiones, pero la extensión a tres dimensiones es directa. En cuanto al costo computacional, el aumento es debido a que es necesario buscar el punto más cercano al origen en un símplice de a lo sumo una dimensión más. Este aumento de dimensión en el símplice conlleva que la cantidad de *features*, como segmentos, puntos, caras o volumen, aumente. En dos dimensiones el número total de *features* es 7 y en tres dimensiones es 15.

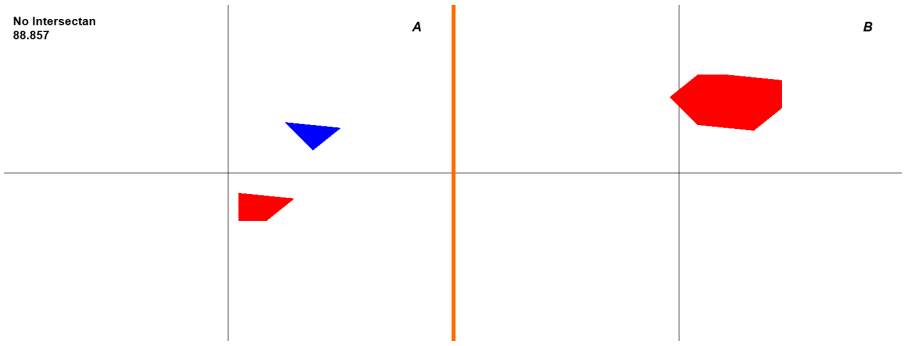

Figura 6. En el recuadro A hay dos polígonos separados en el programa; en la esquina superior izquierda informa que no hay intersección y abajo la distancia a la que se encuentran los polígonos. En el recuadro B se puede ver la resta de Minkowski de éstos, que no contiene al origen ya que los polígonos no se intersectan.

4. Conclusión

El problema de detección de colisiones es complejo y su solución es fundamental para, por ejemplo, el desarrollo de motores de física para videojuegos. El método de GJK es una de muchas posibles alternativas para enfrentar este problema, y cuenta con ventajas notables. En aplicaciones de tiempo real, la característica de coherencia temporal permite evaluar si hay colisiones en tiempo prácticamente constante cuando los objetos se desplazan poco entre frame y frame. El único requisito para que el método sea aplicable es que los objetos sean convexos y que se puedan obtener puntos de soporte para cualquier dirección. Si los objetos son cóncavos, el algoritmo obtiene la distancia entre sus cascos convexos. Por todo esto y por la sencillez de su implementación, sumada a las distintas posibilidades de optimización, el método de GJK es una elección a considerar a la hora de solucionar la detección de colisiones.

5. Trabajos futuros

A continuación se listan algunas posibles aplicaciones y mejoras a la implementación del método:

- Mejorar la búsqueda de puntos de soporte, que actualmente se hacen de forma naïve (utilizando hill climbing, por ejemplo).
- Obtener el par (o uno de los pares) de puntos más cercanos cuando no hay intersección y obtener la información de intersección cuando la halla.
- Una extensión para PyGame (librería en Python para desarrollo de videojuegos) que implemente el método en C++, mediante un wrapper. Actualmente existe una extensión, pero implementada directamente en Python, lo cual es presumiblemente más lento que una extensión en C++.
- Realizar optimizaciones al algoritmo mediante paralelización SIMD utilizando instrucciones vectoriales. En particular utilizando el nuevo set de instrucciones Advanced Vector eXtensions(AVX) que se incorporó en las últimas líneas procesadores. Para sacar provecho se debería reformular el algoritmo y las estructuras de datos utilizadas.

Referencias

1. Christer Ericson. *Real-time Collision Detection.* Morgan Kaufmann, January 5, 2005.
2. Elmer G. Gilbert, Daniel W. Johnson, S . Sathiya Keerthi. *A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space.* IEEE JOURNAL OF ROBOTICS AND AUTOMATION, VOL. 4, NO. 2, APRIL 1988.
3. Chong Jin Ong, Elmer G. Gilbert. *Fast Versions of the Gilbert-Johnson-Keerthi Distance Algorithm: Additional Results and Comparisons.* IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 17, NO. 4, AUGUST 200.
4. Ashok Kumar, Jim Etheredge, Aaron Boudreaux. *Algorithmic and Architectural Gaming Design: Implementation and Development.* IGI Global, May 31, 2012.
5. Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. *Computational Geometry: Algorithms and Applications Third Edition.* Springer, 2008.