Contenidos

INTRODUCCIÓN 3
Sobre motores de Física 4
Que simulamos? 4
Detección de Colisión Discreta (posteriori) o Continua (priori) 4
Resolución de Contactos 5
Reacción a Colisión: Impulsos o Fuerzas? 5
Solución de Penetración 6
Introducción al proceso 7
Etapas detección de colisiones 8
Broad phase 12
Narrow phase 12
Figuras de colisión 13
¿Qué son? 13
Tipos comunes 13
Particionamiento espacial 15
Grids 15
QuadTrees 16
Generación de contactos 18
Test de Colisión 18
SAT: Separating Axis Test 18
Limitaciones del algoritmo SAT: 24
Ejemplos de test de colisión usando SAT: 24
Dos AABBs: 24
Dos OOBBs: 25
Un OOBB y un círculo: 29
Generación de contactos: 32
Resolución de colisión 34
Cantidad de movimiento en choques 34
Choques o Colisiones: 35
Respuesta de cuerpos rígidos 39
Impulso lineal 40
Impulso angular 40
Impulso y velocidad 41
Velocidad resultante 41
Interpenetración 41
Colisiones en Box2D 42

Contactos por iteración....42
Listeners....42
Joints....43

INTRODUCCIÓN

En esta unidad nos centraremos en los problemas de colisión en tiempo real, su simulación e implementación eficiente. Primero arrancaremos dando un panorama en una clasificación de motores físicos en base a ciertas características que nos darán un panorama y nos servirán en las sesiones posteriores para describir a que categoría pertenecería la implementación de la simulación que realizaremos.

Sobre motores de Física

Veremos algunas características de implementación de motores físicos que nos servirán para clasificarlos. Además cuando debamos elegir un motor para usar en nuestros proyectos características como el método de resolución de contactos o si la detección de colisión es discreta o continua nos dará una idea de la estabilidad, artifacts y costo computacional de la simulación a realizar. Tengamos en cuenta que cuando hablamos de motores físicos nos referimos a motores de tiempo real y no motores científicos o de alta precisión.

Que simulamos?

Al pretender realizar una simulación física lo primero a decidir es que vamos a simular. En general en un motor físico lo que es plausible simular son:

- Partículas
- Cuerpos Rígidos

La diferencia fundamental es que el enfoque de partículas no requiere de la matemática de rotación, que si requiere la de cuerpos rígidos. Los motores que simulan partículas son comúnmente llamados motor de agregación de masa. La idea es utilizar varias partículas unidas mediante resortes, barras, cables y demás restricciones para formar cuerpos más complejos. Luego estas estructuras si bien las partículas en particular no poseen rotación, las estructuras formadas por varias partículas si poseen rotación que sale naturalmente de la relación entre las partículas. Entonces por ejemplo para realizar una caja en 2D con un motor de agregación de masa posiblemente usemos 4 partículas y varias restricciones entre estas. La debilidad de este tipo de motores es que es casi imposible crear objetos verdaderamente rígidos, incluso las cajas que formemos tendrán cierta flexibilidad. Además las simulaciones de ciertos cuerpos por estar formados por tantas restricciones y partículas pueden llegar a ser computacionalmente más caro que el enfoque de cuerpos rígidos. Por otro lado la simulación de particulas es muy usado ya que son la base de los efectos de partículas como son el humo, fuego, lluvia y demás efectos que se logran usando un gran número de partículas.

Por otro lado los cuerpos rígidos per se poseen la física de rotación, esta es una de las razones por la cual los motores de cuerpos rígidos son más complejos de realizar. Básicamente todos los motores de física para video juegos, como Box2D, simulan cuerpos rígidos.

Detección de Colisión Discreta (posteriori) o Continua (priori)

En general los motores pueden utilizar detección de colisiones continua para ciertos objetos y para el resto usar detección discreta, como es el caso de Box2D donde los objetos definidos como “bullet” utilizan detección continua. Pero existen otros motores que se decantan por un solo método de detección de colisiones, todos los objetos detectan de forma continua o discreta.

En el caso de detección discreta la simulación avanza el tiempo de refresco de la simulación, y luego se fija en esa posición si los objetos están en colisión. Este método se llama a posteriori porque no se calcula el instante verdadero de colisión sino que verifica si se produjo la colisión.

Por otro lado la detección continua calcula el momento de colisión exacto de los objetos, como se calcula el momento de colisión podemos prever la colisión y resolverla antes de que los objetos se penetren, es decir que

podemos detectar la colisión un refresco antes de que estos se sobrelapan. Se lo llama a priori porque se detecta la colisión antes de que suceda.

Si bien la detección de colisión continua es preferible es considerablemente más costosa y no siempre necesaria. Además es más difícil de implementar ya que el problema de detección de colisiones continua tiene una dimensión más, que es el tiempo, es decir si teníamos un problema de detección de colisiones en 2D la detección continua se transforma en uno en 3D porque debemos ver si colisionan en las2 dimensiones físicas y además atreves de tiempo. También es cierto que para estimar el movimiento atreves del tiempo el detector de colisiones debe "saber" sobre las leyes de movimiento del cuerpo y sus fuerzas que lo afectan como la fricción o damping.

Resolución de Contactos

Una de las partes más difíciles, si no la más difícil, de realizar una simulación física es la resolución a las relaciones entre objetos. En estas relaciones estamos hablando de contactos, ya sean en reposo o colisión, y otras restricciones entre objetos como joints. Los enfoques que se usan para la resolución de contactos son:

- **Método iterativo secuencial:** Consiste en resolver las colisiones y demás constraints una por una, intentando resolver una a la vez, de aquí el nombre secuencial. El orden en que se resuelven es fundamental. Es iterativo porque se suele resolver varias veces cada contacto, esto se debe a que la resolución de un contacto puede afectar otros, imaginémonos un caso de unas 20 cajas de caen formando una pila, cuando al caja del fondo colisione contra el piso esta se resolverá colocándose sobre el suelo en contacto y seguramente con una velocidad(o fuerza) de rebote, este reposicionamiento puede hacer que colisione o penetre a la caja de arriba, luego resolvemos esta colisión, pero a su vez esa caja empieza a colisionar y penetrar la de arriba, bueno este es solo un caso de cierta complejidad podría ser peor, ya que podría ciclar entre 2 colisiones cuya solución "rompe" otro contacto y ciclar entre estos.

- **Método basado en el Jacobiano:** Este método conlleva un desarrollo matemático más complejo que el enfoque anterior. La idea es resolver todos los contactos al mismo tiempo. Para resolver es necesario implementar un método de resolución de LCP (Linear Complementary Problem) o Problema Complementario Lineal que básicamente es un problema similar a los sistemas lineales que vimos pero con ciertas complicaciones. Otra forma consiste en invertir la matriz del Jacobiano, que no siempre es posible. Este método es el que utiliza Box2D.

Si bien el segundo método es en general más correcto, aunque en general también más costoso computacionalmente es posible demostrar que son equivalentes bajo ciertas condiciones.

Reacción a Colisión: Impulsos o Fuerzas?

Otra característica es como se realiza la reacción a las colisiones. La forma más correcta es aplicar fuerzas, es decir si una pelota cae al suelo, al colisionar el suelo le aplica una fuera normal a este que hace que la pelota rebote. Pero esa fuerza normal es aplicada en un lapso pequeñísimo de tiempo y su fuerza es muy grande, así que otra forma de hacer reaccionar a la pelota seria aplicarle un impulso que como vimos es una variación en la cantidad de movimiento en un lapso de tiempo(o dicho de otra forma la integral de la fuerza en un lapso de tiempo), entonces para aplicar ese impulso lo que haremos es aplicar una velocidad al cuerpo, ya que consideramos que el impulso se aplica en un lapso infinitesimal de tiempo. Por otro lado veamos un caso donde un libro descansa en contacto sobre una mesa, en ese caso la fuerza normal a la mesa anula la gravedad y el libro se mantiene sin ninguna velocidad, en un caso así de contacto en reposo al resolver aplicando un impulso puede producir un comportamiento antinatural, aunque para resolver casos como este existen métodos para solucionarlo. Por otro lado aplicar fuerzas puede ser complejo ya que como se dijo esta fuerza se debería aplicar sobre un lapso muy pequeño de tiempo. Entonces resumiendo lo que podemos hacer para hacer reaccionar 2 cuerpos cuando estos chocan es:

- Aplicar Fuerzas

- Aplicar Impulsos

También es posible aplicar una combinación de estos, es decir por ejemplo usar fuerzas para los contactos en reposo e impulsos para las colisiones. No obstante en general es más sencillo y por eso más usando en los motores de física usados en video juegos los impulsos de manera única para resolver las reacciones a colisiones, motores de este tipo se los conoce como "basados en impulsos" (impulse-based).

Solución de Penetración

Antes que nada se tiene que saber si el sistema de detección de colisiones permite que se puede producir una penetración, ya que usando detección de colisión continua es posible evitar esto. Pero en general la penetración existe y es un problema que hay que solucionar ya que es físicamente imposible. Para resolverla podemos tomar distintas medidas que pasmos a mencionar:

- **Modificar la posición:** Si tenemos una penetración de 2 objetos la solucionamos moviendo los objetos, ya sea teniendo en cuenta sus masas para que los objetos más "pesados" se muevan menos que los más "livianos" u bajo otro esquema, en la dirección normal a la colisión.

- **Aplicar impulsos:** Para resolver la penetración le aplicamos una velocidad a los cuerpos que se están penetrando en direcciones opuestas, teniendo en cuenta la normal de colisión, para que la penetración se resuelve luego de la integración.

- **Aplicar fuerzas:** Este método se llama de "penalty", para resolver la penetración le aplicamos una fuerza a los cuerpos que se están penetrando en direcciones opuestas, teniendo en cuenta la normal de colisionó, para que la penetración se resuelve luego de la integración.

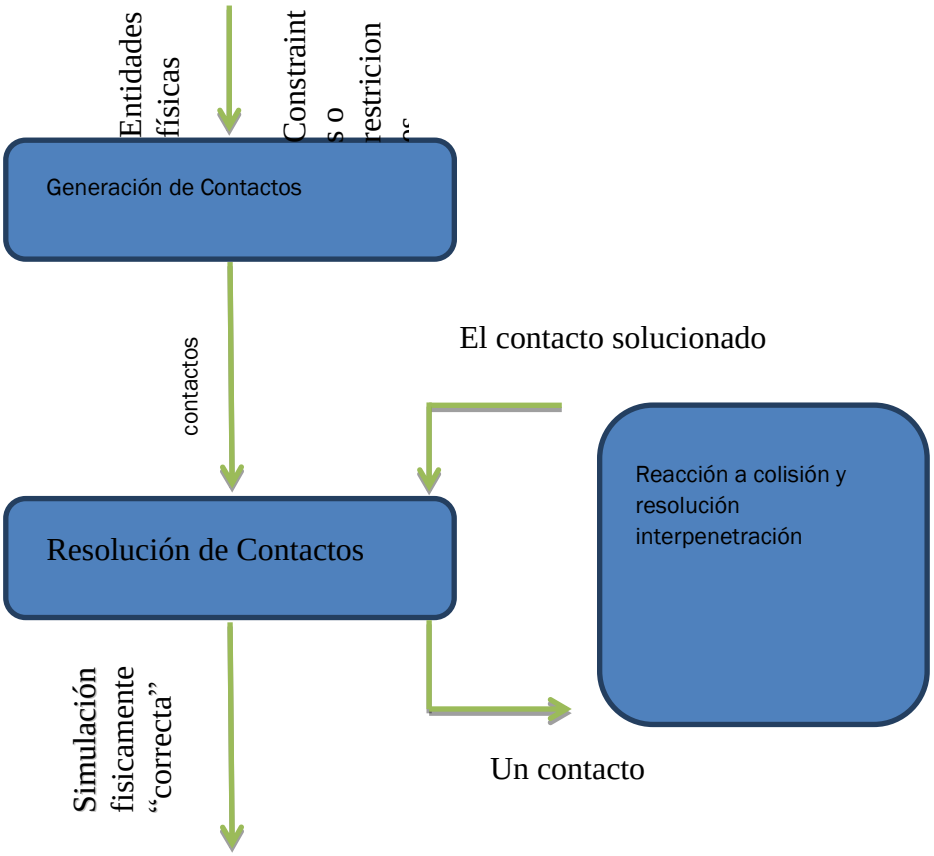
Notemos que lo que varía en los distintos métodos es que se modifica, y que consiste en modificar la posición u una derivada de esta, ya sea la velocidad o la aceleración. En general modificar directamente la posición es bastante usado.

Introducción al proceso

Nos dedicaremos a 3 temas fundamentales para la implementación de un motor físico. Estas 3 etapas son:

- Generación de contactos
- Reacción a colisión y resolución interpenetración
- Resolución de contactos

Estas 3 unidades funcionales son todo lo que se necesita para implementar y comprender un motor físico, tan complejo como Box2D o tan simple como el motor de agregación de partículas que se vio. Expresadas las 3 unidades en un diagrama de flujo:



Pasaremos a dar una explicación breve de cuál es la función de cada uno de los módulos:

- **Generación de contactos:** La función de este modulo es recibir el conjunto de todos los objetos físicos de la simulación (en general estos estarán en una estructura de datos como veremos más adelante) y en función de si están colisionando generara una estructura que llamaremos contacto, el cual contiene información de la colisión suficiente para resolverla. Para ser más general un contacto quiere decir que se violo una restricción de un par de objetos, la colisión es un tipo, pero también existen otras como restricciones de distancia entre objetos, juntas de rotación, juntas prismáticas, etc. Luego de generar los contactos de todos los objetos de la escena pasa estos al siguiente modulo de Resolución de Contactos.

- **Resolución de Contactos:** Este modulo depende (usa) el modulo de Reacción a colisión y resolución interpenetración. La función es en base a la lista de contactos que posee decide en qué orden resolverlas usando cierto esquema con varios criterios, y para la resolución de cada contacto utiliza el modulo del cual depende. La salida de este modulo es la simulación en ese paso de tiempo, las comillas en correcta es debido a que en un dispositivo discreto y secuencial como una computadora solo podremos hacer una

aproximación, así que en general no sería extraño que en casos complejos nuestra simulación pueda permitir que cuerpos rígidos se interpenetren, o no cumplan con alguna restricción que les impusimos.

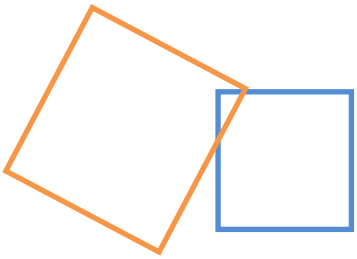
- **Reacción a colisión y resolución interpenetración:** Este modulo es que efectivamente resuelve cada contacto. Es decir en función de la información de contacto que posee resuelve las colisiones e interpenetraciones que puedan darse. De nuevo como se dijo en el modulo de Generación de contacto, un contacto puede representar otro tipo de restricción además de las de colisión e interpenetración, así que el nombre de Reacción a colisión y resolución interpenetración no es el más feliz, más correcto sería Resolución de Contacto (notar que está en singular).

Etapas detección de colisiones

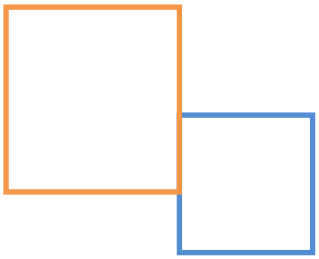
La detección de colisiones forma parte del modulo de Generación de Contactos. Antes que nada es conveniente definir que es una colisión para nuestra simulación. Podemos dar 2 definiciones:

1. Cuando dos objetos, sean A y B, están interpenetrando o tocando sus lados:

Interpenetrando



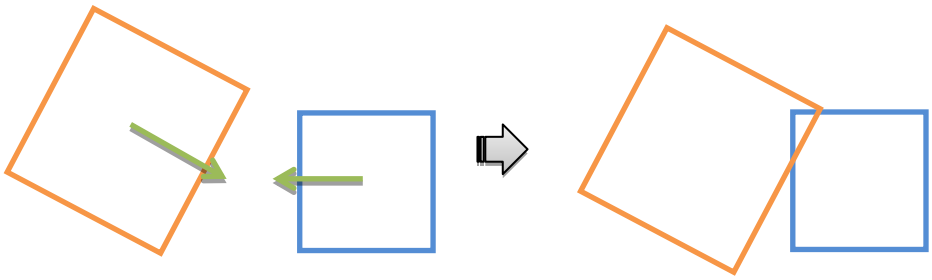
Tocando sus lados



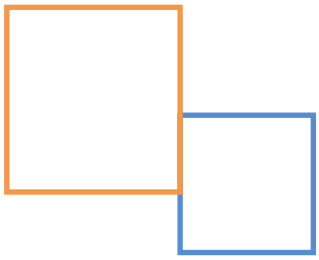
2. Cuando dos objetos, sean A y B, están tocan sus lados en este instante de tiempo o en la próxima iteración de la simulación estarán interpenetrando:

t_0 Según la definición 2 hay colisión

t_1



Tocando sus lados



Vemos que la segunda definición es mejor ya que podemos anticiparnos a una interpenetración, y evitar las interpenetraciones siempre es lo más conveniente ya que nos evita tener que resolverlas, y al resolverlas poder producir más interpenetraciones. El problema es que no es trivial poder predecir la interpenetraciones antes que sucedan, para ello deberíamos utilizar detección de colisiones continua (incluso esto no necesariamente significa que jamás vallamos a tener interpenetraciones). En nuestro caso veremos detección de colisiones discreta, así que consideraremos una colisión como se define en el punto 1.

Para probar si dos objetos están colisionando debemos hacer una prueba entre cada par de objetos para, es decir si nuestra simulación tiene N cuerpos rígidos, luego para probar la colisión entre cada par de objetos debemos hacer:

$$C(N,2) = \frac{N!}{(N-2)!\,2!}$$

$$C(N,2) = \frac{(N-1)N}{2}$$

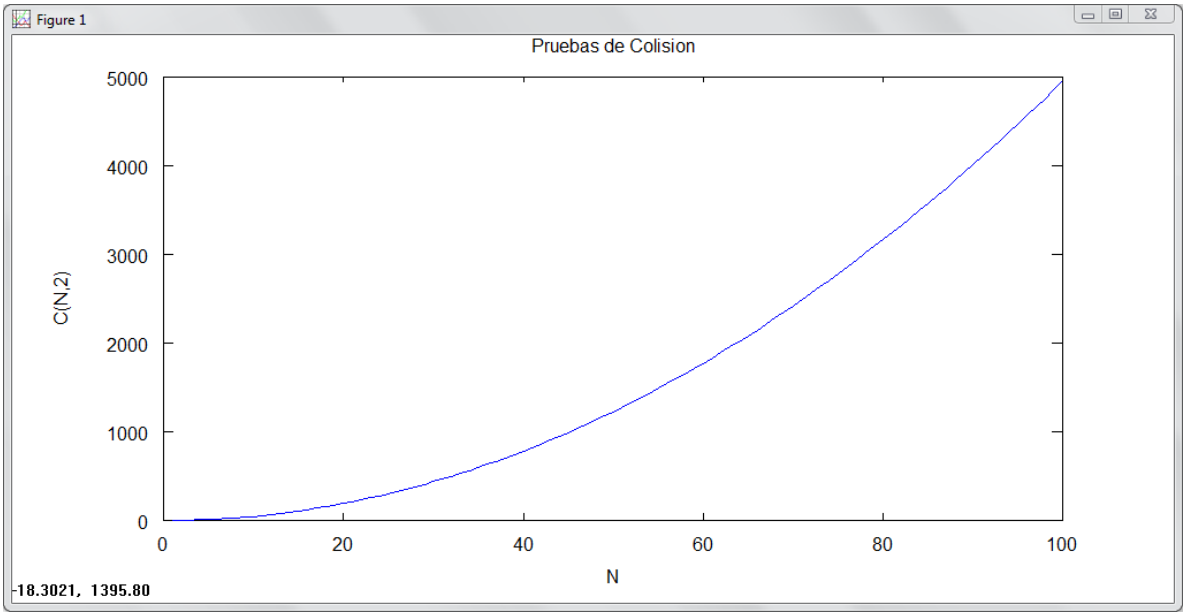
Es decir que deberemos hacer $\frac{(N-1)N}{2}$ pruebas de colisión para comprobar todos los posibles pares que estén colisionando. Para ejemplificar veamos si $N = 25$ luego deberemos hacer:

$$C(25,2) = \frac{600}{2}$$

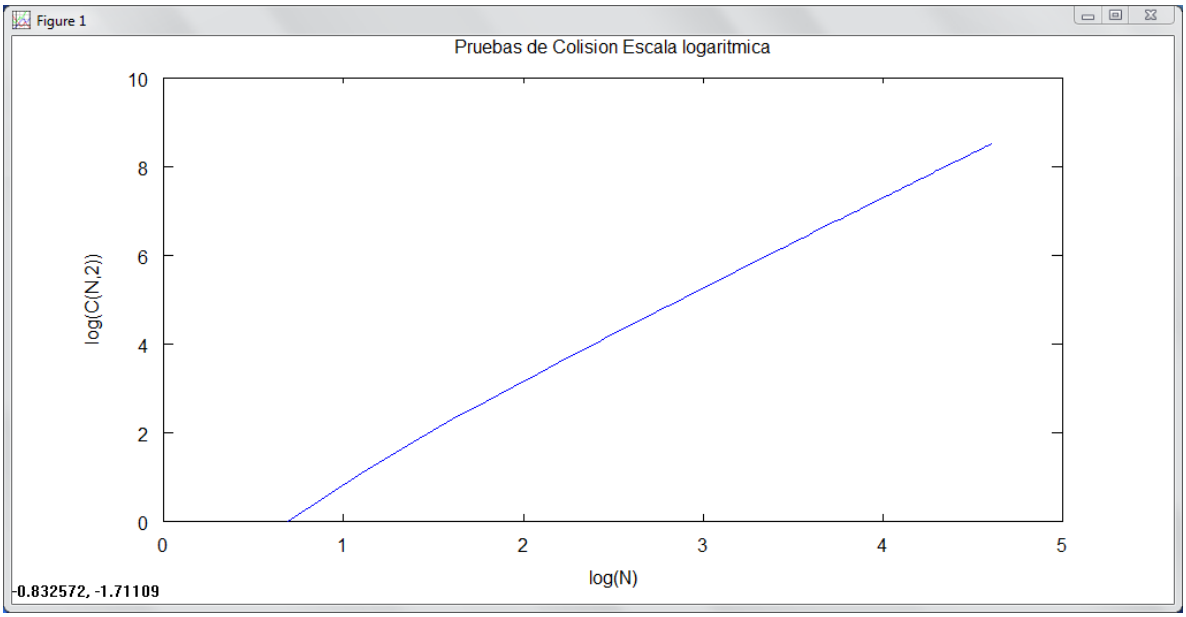
$$C(25,2) = 300$$

Es decir que con una escena de solo 25 objetos, si estamos corriendo a solo 30 frames por segundo entonces tendremos menos de 34 milisegundos de los cuales solo podremos usar una pequeña fracción a detección de colisiones, en ese par de segundos que nos quedaran deberemos hacer 300 pruebas de colisiones. Además una escena promedio no sería muy extraño que tuviésemos unos 10.000 objetos en escena, con un grupo tan grande

de objetos si los tuviésemos en una lista dependiendo de la potencia del procesador y la arquitectura de memoria tal vez ni siquiera podríamos recorrer la lista, y mucho menos realizar $C(10.000,2)$ test de colisión. Veamos como el número crece rápidamente al aumentar el número de elementos en la escena. Para ver la velocidad de crecimiento de la cantidad de pruebas veamos la grafica de $C(N,2)$ para los 100 primeros valores:



Por las propiedades de los logaritmos si sacamos el logaritmo de el dominio y la imagen, luego las funciones de potencias dan rectas y la pendiente de la recta es el exponente de la potencia. Luego el grafico en escala logarítmica es:



Es aproximadamente una recta, y si calculamos su pendiente es aproximadamente 2. Es decir que la prueba de colisión es aproximadamente $O(N\text{^}2)$.

Para mejorar esto podríamos organizar todos los N cuerpos de la escena en G grupos de tal forma que todo cuerpo pertenezca a 1 y solo 1 grupo y un grupo puede tener un rango de 0 a N cuerpos, donde sabemos que solo puede haber colisiones entre objetos de un mismo grupo. Es decir sea $a_i^{G_j}$ el cuerpo numero i perteneciente al conjunto numero j, luego estamos seguros que solo puede haber colision entre dos cuerpos $a_i^{G_j}$ y $a_k^{G_j}$ con $i \neq k$(ya que el objeto siempre colisionará con sí mismo), es decir entre 2 cuerpos pertenecientes al mismo grupo, pero jamás colisionaran dos cuerpos $a_i^{G_j}$ y $a_k^{G_l}$ con $j \neq l$(osea de grupos diferentes). Además queda claro que mientras más igualmente repartidos estén los cuerpos en los grupos mejor será ya que en cada grupo debemos hacer $C(N, 2)$ pruebas y es preferible hacer pruebas entre más conjuntos pequeños que una sola prueba con 1 solo conjunto. Dicho esto último queda claro que hacer un agrupamiento donde pongamos todos los cuerpos en un solo conjunto es exactamente lo mismo que no hacer ningún agrupamiento.

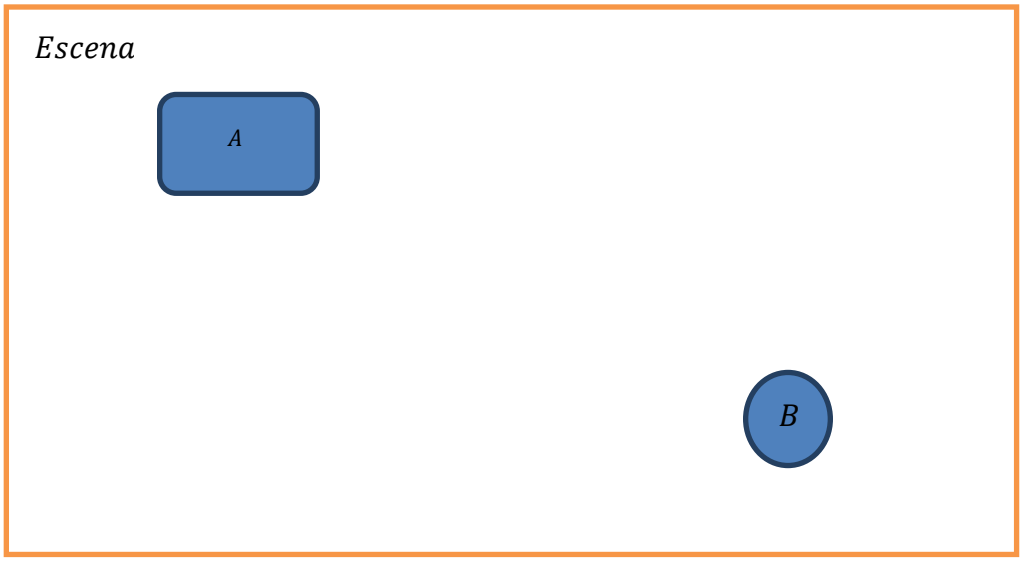
Dicho de esta forma parece la solución perfecta, ahora el problema es cómo podemos crear de los N cuerpos de la escena el conjunto de grupos. Recapitulemos las características que deseamos para nuestro agrupamiento:

- Un numero G de grupos lo más grande posible.
- Que los cuerpos estén repartidos entre los grupos de forma que el numero de cuerpos en un grupo G_i sea lo más chico posible.

Podemos agrupar los cuerpos en base a características similares que estos posean. El agrupamiento puede ser bajo 2 criterios:

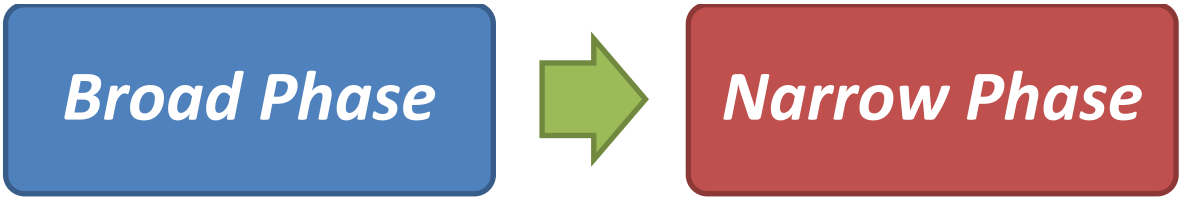
- Agrupar de forma lógica por el tipo de cuerpo o alguna propiedad del cuerpo.
- Agrupar por la posición espacial del cuerpo.

Podemos descartar el primer criterio ya que lo que queremos es un conjunto donde los cuerpos en su interior sean los únicos que puedan colisionar entre sí, y que un conjunto de cuerpos sean “Cuerpos circulares”, ”Cuerpos con masa 1”, etc no cumplen con esa característica. Por otro lado imaginemos que probamos colisión entre 2 objetos A y B que están ubicados espacialmente de la siguiente forma:



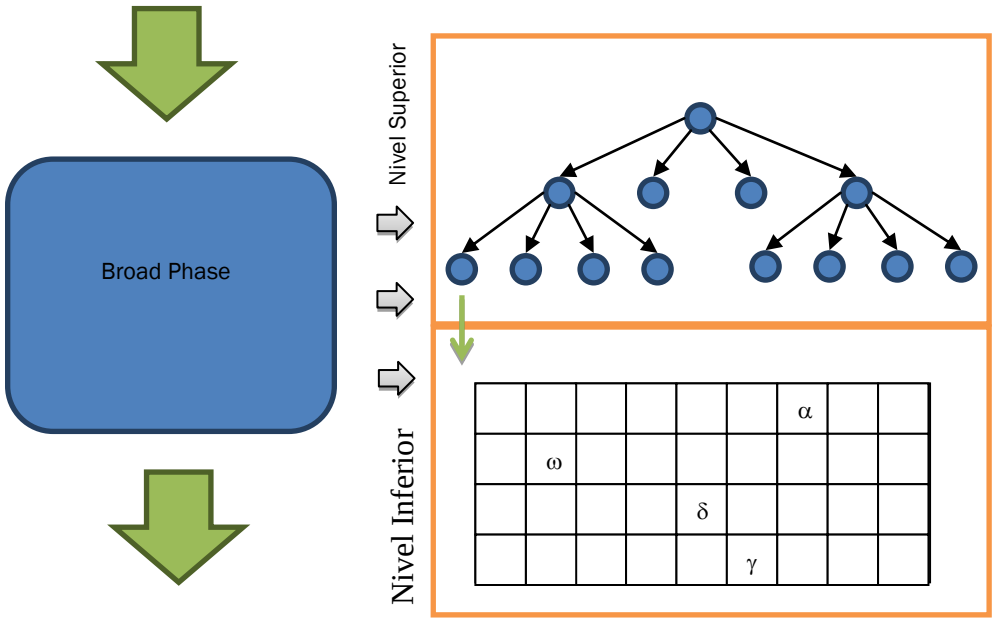
Podemos observar que de antemano podríamos ver que los 2 objetos están en esquinas opuestas en la escena y están muy lejos entre sí como para colisionar. Es decir que si podríamos deducir u organizar los objetos de tal forma que pudiésemos saber cuándo 2 objetos están muy lejos entre sí como para siquiera probar su colisión podríamos reducir drásticamente la cantidad de pruebas de colisión que debemos hacer. Vemos entonces que 2 cuerpos que

están cercanos físicamente son los que tienen posibilidad de colisionar entre sí. Por esto deducimos que lo que requerimos es un Ordenamiento Espacial de los cuerpos en escena para nuestro problema. Este es un orden en el que guardamos los objetos en escena atreves del cual podemos deducir alguna información espacial sobre este. En MAVI vimos conceptualmente algo sobre ordenamiento espacial, en esta segunda parte profundizaremos un poco más y veremos 2 tipos: 1) Grillas 2) Quad Tree.



Broad phase

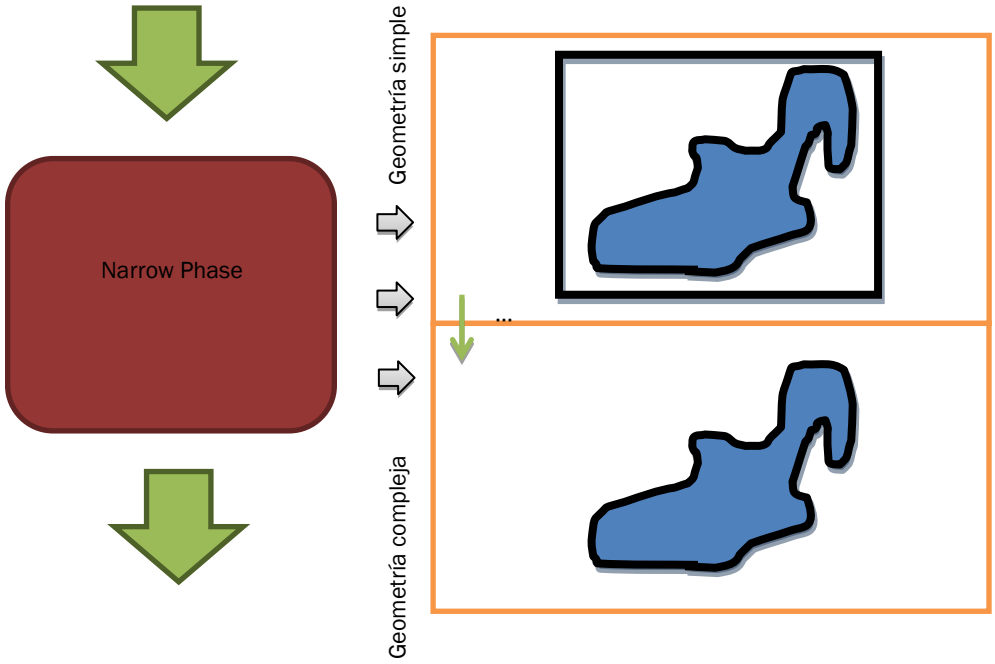
Este es el primer paso por el cual eliminamos o descartamos grandes grupos de cuerpos que sabemos que no colisionaran entre sí. Para crear los grupos usamos Ordenamientos Espaciales que vimos en la introducción que son los que nos convienen para nuestro problema. En este paso si bien muchas veces solo se utiliza un solo ordenamiento o nivel de ordenamiento esto no necesariamente debe ser así, por ejemplo podríamos tener que a un nivel mayor utilizamos un cierto ordenamiento espacial, por ejemplo Quad Tree, y a un nivel inferior utilizamos otro ordenamiento como una Grilla u otra estructura que no veamos en el curso. Por ejemplo:



Narrow phase

Este paso tiene lugar una vez pasado por el Broad phase. La idea ya no es eliminar o descartar grandes grupos de cuerpos, sino que entre dos cuerpos probar si colisionan usando una geometría que permita un cálculo más rápido de detección de colisión, aunque la geometría no sea demasiado representativa del cuerpo. Es decir por ejemplo si tuviésemos un modelo de un personaje tridimensional en un fps y una bala que puede estar por colisionar con el modelo, antes de probar la colisión con la geometría real del modelos(o una buena aproximación de esta) probaríamos una colisión con por ejemplo una caja que contiene el modelo, de esta forma si la bala no está

colisionando con la caja contenedora del modelo luego tampoco lo estará con el modelo y solo perdimos una pequeña cantidad de tiempo detectando colisión con la caja contenedora y no con la compleja geometría real del modelo, por otro lado si la bala colisiona con la caja contenedora luego puede pasar que la bala colisione o no con el modelo real ya que como la caja era considerablemente más grande que el modelo luego la bala podría pasar lejos aún del modelo, entonces si se da la colisión con la caja luego probaríamos la colisión contra la geometría verdadera del modelo. Esta es la idea de narrow phase o paso fino, se usan distintas geometrías simples que "envuelven" a un cuerpo y que su cálculo de colisión son rápidos y cuando se debe probar colisión se lo hace desde la geometría más simple a la más compleja, de esa forma la mayoría de los casos no se dan colisiones y se ahorra el cálculo caro computacionalmente hablando de detección de colisión de geometrías complejas. La jerarquía de geometrías puede ser tan simple como tener solo la geometría definitiva y una más simple o más compleja como tener varias geometrías simples y al final la compleja. Un ejemplo:



Figuras de colisión

¿Qué son?

Las figuras de colisión son geometrías simples y son hasta cierto punto representativas de una geometría más compleja y sirven para poder realizar operaciones inherentemente costosas computacionalmente primero sobre estas y solo realizarlas sobre la geometría más compleja si es absolutamente necesario. La idea es usar figuras como círculos, rectángulos, elipses, capsulas y demás geometrías relativamente simples para representar cuerpos más complejos como puede ser una nave espacial por ejemplo. En el narrow phase se comentó que se las usaba para ese fin.

Tipos comunes

Algunas figuras de colisión muy usadas son:

1. AABB:

   Son cajas rectangulares que poseen la misma orientación que los ejes canónicos $\boldsymbol{x}$ e $\boldsymbol{y}$, es decir que tienen dos lados paralelos al eje $\boldsymbol{y}$, y dos lados paralelos al eje $\boldsymbol{x}$. En el Figura 1 podemos ver un AABB.

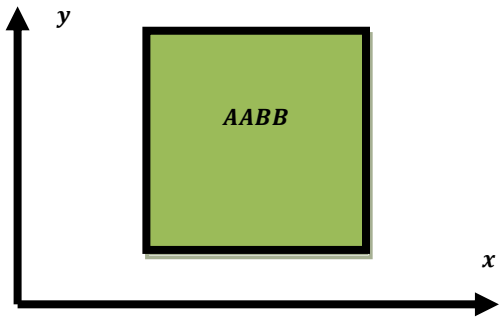
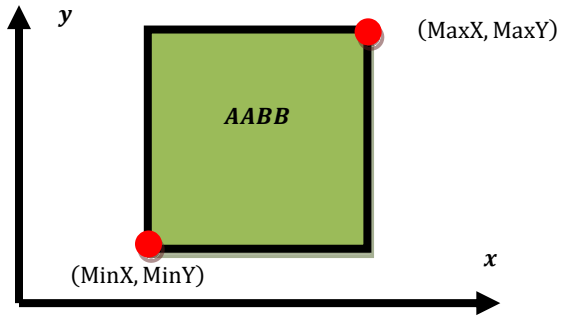


FIGURA 1: EJEMPLO DE AABB

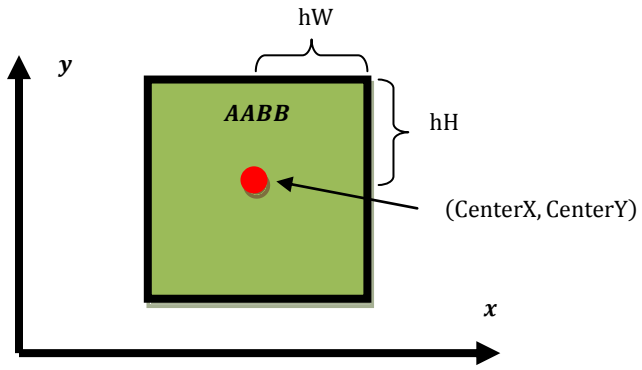
Existen distintas representaciones, todas igualmente validad para un AABB, aunque a veces alguna es más conveniente dependiendo del algoritmo que se use y de su implementación. Algunos ejemplos de implementación:

```
class AABB
{
        public:
            float MinX;//Coord X de punto inferior
            float MinY;//Coord Y de punto inferior
            float MaxX;//Coord X de punto superior
            float MaxY;//Coord Y de punto superior
    };
```




```
class AABB
{
```

```
        public:
                float CenterX;//Coord X del centro
                float CenterY;//Coord Y del centro
                float hW;//Medio ancho
                float hH;//Media altura
};
```



Notemos que todas las representaciones propuestas tienen la misma cantidad de datos, solo cambia que datos son.

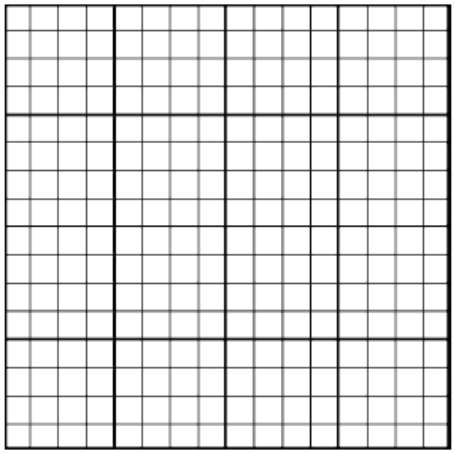
2. OOBB: Este es un bounding box que no se encuentra alineado con los ejes coordenados, sino que rota con el objeto. Son más difíciles de mantener y costosos.

3. Bounding Circle: Otra figura de colisión muy sencilla la cual consiste en rodear el objeto con un círculo. En este caso sólo necesitamos el centro y su radio.

Particionamiento espacial

Grids

La grilla o grid es una estructura muy simple y eficaz cuando los objetos se distribuyen uniformemente en el espacio del mundo. La idea de la grilla consiste en dividir el espacio en una serie de celdas iguales como muestra la siguiente figura:



De esta forma la pertenencia de cada objeto en la grilla queda unívocamente determinada por la posición espacial del objeto. Por ello es muy rápido encontrar objetos cercanos a otros y construir la estructura. En cada celda de la grilla pueden existir varios objetos al mismo tiempo, en el límite si todos los objetos estuviesen en la misma celda entonces tendríamos el peor caso: buscar todos contra todos. Pero normalmente si se elige adecuadamente el tamaño de la grilla esto no ocurre.

Ante la presencia de objetos que ocupen múltiples celdas entonces se puede realizar dos alternativas:

- Agregar el objeto en todas las celdas que ocupa.
- Agregarlo a la que se encuentra sobre su centro, pero al chequear colisiones deberemos recorrer también ahora las celdas vecinas.

La grilla es una estructura muy utilizada especialmente en 2D por su simplicidad y buenos resultados.

QuadTrees

Como vimos las grillas son simples y eficaces, el problema con estas es que al dividir el espacio de forma regular luego es normal que suceda que para las áreas altamente poblada de cuerpos el tamaño de la grilla sea muy grande y para áreas escasamente pobladas el tamaño de la grilla será muy pequeño. Además también tenemos que administrar los cuerpos inmensos que se esparcen por varias grillas.

La solución a estos problemas es particionar el espacio de forma adaptativa, es decir que las regiones de división sean más pequeñas en zonas donde hay muchos cuerpos y más grandes en zonas donde hay pocos objetos. El ordenamiento espacial adaptativo más usado en 2D es el Quad Tree.

La idea del ordenamiento es crear un árbol donde cada nodo representa un área y una lista con 0 a N cuerpos. El nodo raíz del árbol posee toda el área de la escena, luego este posee 4 hijos (de ahí el nombre **Quad** Tree), donde cada uno de esos hijos representa un área rectangular, las áreas de los 4 hijos no se solapan entre si y la unión del área de los 4 hijos forma el área del nodo padre, a su vez cada uno de estos hijos se subdivide en 4 hijos con las mismas propiedades de los 4 hijos primeros, y así sucesivamente hasta un final. Veamos en Figura 2 esquemático de un Quad Tree en acción:

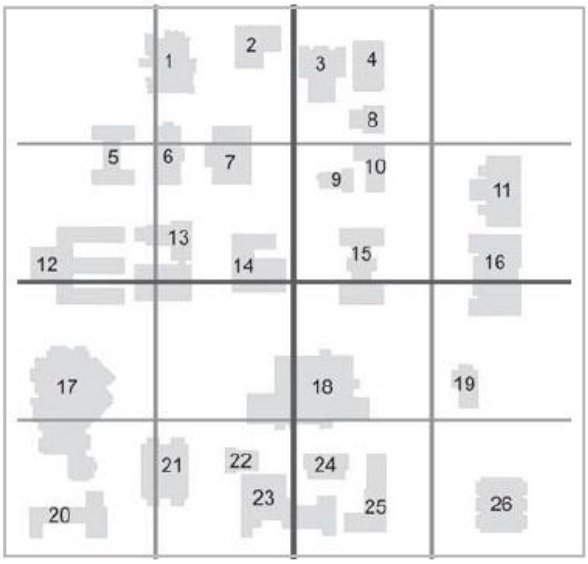


FIGURA 2: QUAD TREE

Luego la pregunta será como usamos esta estructura para ordenar los cuerpos? La idea es que cuando un cuerpo está totalmente contenido en el área de un nodo hijo luego pasamos a ese nodo hijo. Repetimos para los próximos nodos hijos mientras el cuerpo este totalmente contenido en el área del nodo hasta que lleguemos a un nodo hoja o el cuerpo no esté completamente contenido en el área de un nodo. Luego si llegamos a un nodo hoja lo que hacemos es agregar el cuerpo a ese nodo hoja, por otro lado si se da que el cuerpo a un nivel no está totalmente contenido en el área de uno de los nodos hijos agregamos el cuerpo a el nodo de ese nivel. A continuación vemos en Figura 3 como quedaría el quad tree del ejemplo anterior:

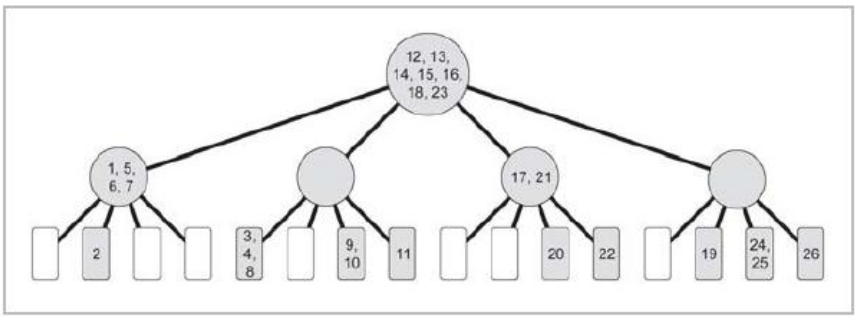


FIGURA 3: ÁRBOL DE UN QUAD TREE

Las implementaciones de estos ordenamientos pueden variar de implementación a implementación, ya que algunas características de la formación y mantenimiento del árbol no son estrictas. Por ejemplo en el grafico anterior supusimos(para simplificar el ejemplo) que en nuestro Quad Tree las areas de los nodos hijos era la misma para los 4, si bien esta es una implementación totalmente aceptable podríamos mejorar el algoritmo para que intente darle áreas a los 4 hijos de tal forma que en cada hijo quede una cantidad lo más aproximada posible a $\frac{1}{4}$ de los cuerpos que corresponden a este hijo, esto haría que el árbol este balanceado lo cual es una característica deseable en un árbol. O podríamos hacer que solo halla elementos en las hojas del quad tree, a costa de que por ejemplo un elemento pueda estar más de una vez en el árbol. En la Figura 4 podemos ver un nodo de un quad tree con sus hijos.

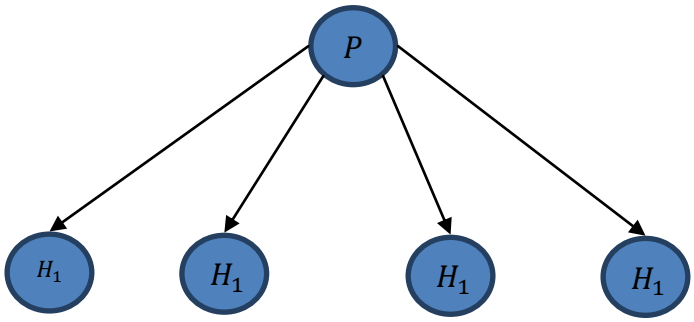


FIGURA 4: NODO DE QUAD TREE

Supongamos que P es el nodo padre y H_i con i entre 1 y 4 son los hijos. Un Quad Tree siempre debe cumplir que:

- La intersección de las áreas de los nodos H_i dan 0.
- La unión de las áreas de los nodos H_i dan igual al área del nodo P.

Estrategias de corte de la generación de hijos:

- El numero de cuerpos en el nodo hijo es la suficientemente pequeño, es decir por ejemplo hay menos o igual a k cuerpos.
- El área de los próximos nodos hijos será más pequeña que un área que especificamos.
- Expandir el Quad Tree hasta un cierto nivel del árbol.

Generación de contactos

Test de Colisión

Para resolver problemas físicos y problemas lógicos en videojuegos una de las operaciones más importantes e incluso muchas veces las más caras computacionalmente y más complejas son las de colisión. La prueba más básica que se puede realizar dentro de estas es una prueba de colisión. Esta prueba es solo una operación booleana que dado un par de objetos nos dice si estos están colisionando. Existen muchos algoritmos para resolver estas pruebas e colisión, los más comunes son SAT, GJK y V-Clip. Nosotros nos enfocaremos en usar SAT para realizar las pruebas.

SAT: Separating Axis Test

Las siglas de este método se pueden traducir como prueba de eje de separación, solo es aplicable para figuras de colisión convexas (entre las que incluimos en particular polígonos, líneas y círculos). Analicemos como podríamos identificar que dos objetos no están colisionando, como se ve en la Figura 5, una forma que con la que podemos ver si dos cuerpos están separados seria encontrando una línea (un plano) entre estos dos, tal que en un lado de esta línea este uno de los cuerpos y del otro lado quede el otro cuerpo. Entonces si podemos encontrar esa línea que los separe de esa forma podemos estar seguros que los cuerpos no están colisionando. Ahora pensemos en el caso contrario donde los cuerpos están colisionando, como se ve en la Figura 6, en este caso es evidente que no existe una línea que separe los dos cuerpos.

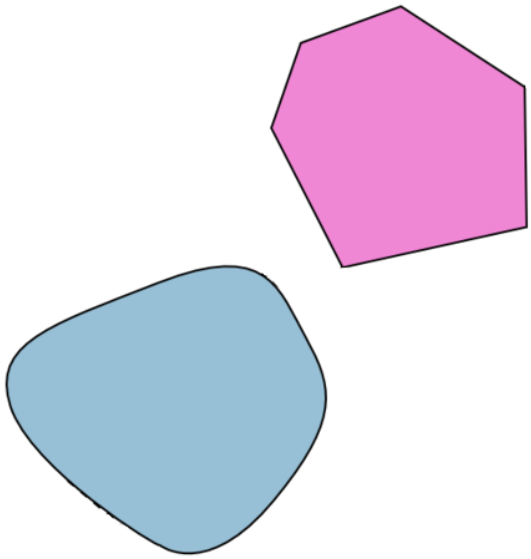


FIGURA 5: DOS CUERPOS CONVEXOS SEPARADOS

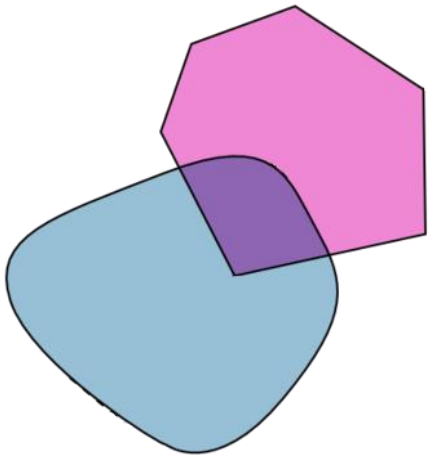


FIGURA 6: DOS CUERPOS CONVEXOS COLISIONANDO

Hasta ahora suena agradable, solo debemos ver si existe una línea que separe un par de cuerpos convexos, si esta existe luego los cuerpos están separados, si no existe estos están colisionando. Pero como vemos si esa línea existe? bueno primero lo primero, antes de ver cómo resolver el problema enunciemos un teorema sobre el que nos basaremos y básicamente dice lo que acabamos de ver, este teorema se llama teorema del eje de separación(Separating Axis Theorem) o teorema del hiperplano de separación[1], cuyo esquema se encuentra en la Figura 7, y dice como sigue:

[1] Para ver una demostración de este teorema ver en "Collision Detection in Interactive 3D Environments" de Gino van den Bergen

Teorema del eje de separación:

Si dos cuerpos convexos están separados luego existirá una línea de separación entre ellos y si existe dicha línea de separación necesariamente estos estarán separados. El eje ortogonal a esa línea de separación lo llamaremos eje de separación.

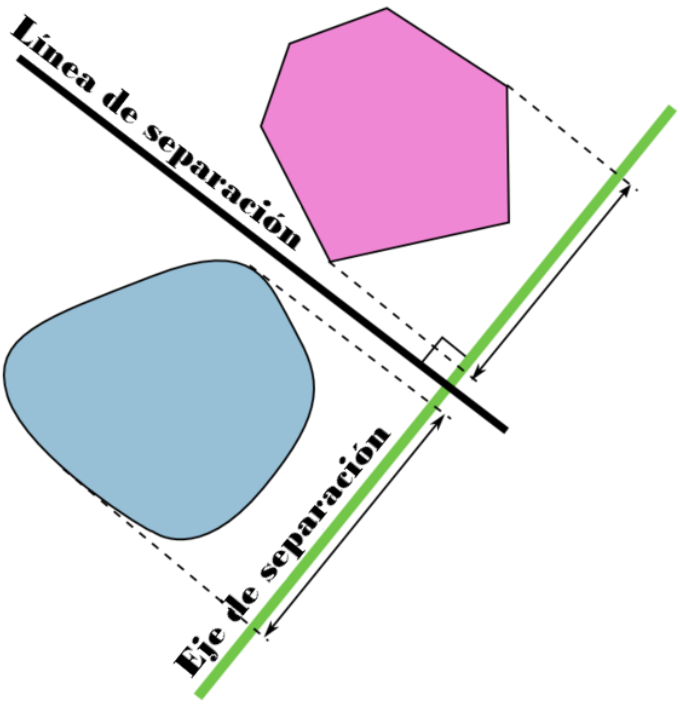


FIGURA 7: TEOREMA DEL EJE DE SEPARACIÓN

Ahora detengámonos un momento a pensar porque pedimos que los dos polígonos sean convexos? acaso dos cuerpos separados, donde al menos uno es cóncavo, se cumplirá que siempre podemos separarlos usando una línea? bueno la respuesta a eso es no siempre, y un ejemplo que demuestra que el teorema del eje de separación no se cumple para cuerpos cóncavos está en la figura 8.

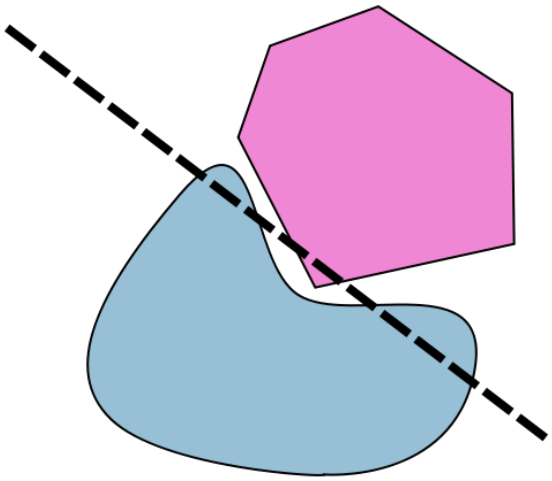


FIGURA 8: CUERPOS CÓNCAVO Y CONVEXO

Esta idea que graficamos es toda la base del método SAT.

Ahora volvamos a la pregunta anterior como sabemos cuál es esa línea de separación. Reformulemos la pregunta, porque no preguntamos cuál es el eje de separación, ya que si tenemos el eje de separación entonces podemos estar seguros que existe una línea de separación. Entonces el objetivo es encontrar el eje de separación, si es que este existe, si no existe los objetos estarán colisionando. Si analizamos los casos de colisión de figuras convexas se puede deducir que no cualquier eje puede ser un eje de separación, entonces separaremos un conjunto de ejes que llamaremos **ejes posibles de separación**, estos serán aquellos ejes que pueden llegar a separar los dos objetos convexos.

Ahora que ejes posibles de separación podemos tener? la respuesta es que para **polígonos convexos en 2D**(en 3D y si no es un polígono deberemos hacer otro análisis) el eje de separación debe ser una de las normales a las caras de los polígonos. Para ver eso de forma intuitiva imaginemos que dos polígonos están solo en contacto y no tienen interpenetración, entonces el contacto solo puede ser de 3 formas: 1) **cara con cara**, 2) **vértice con cara** y 3) **vértice con vértice**, como se ve en la Figura 9. Ahí podemos ver que la línea de separación, en color azul, siempre corresponde con el eje de separación que es la normal a una cara del polígono que está en color gris. En fin concluimos por observación que para polígonos si existe eje de separación debe ser una normal de uno de los polígonos.

FIGURA 9: TIPOS POSIBLES DE COLISIÓN

Ahora como comprobamos si un eje posible de separación es efectivamente un eje de separación? para responder a esta pregunta lo que en general se hace es proyectar cada uno de los vértices de los polígonos en el eje posible de separación y sacar el máximo y mínimo valor de los vértices de cada uno de los polígonos, luego mediante una desigualdad, que pregunta si las proyecciones se superponen, sabremos si estos están en colisión o no. Podemos ver gráficamente la desigualdad en la figura 6. Entonces el pseudocódigo del SAT en 2D para casos generales es:

1. Por cada eje posible de separación S hacemos:

a) Calcular $min_A = min\left(Dot\left(V_i^A, S\right)\right)$ y $max_A = max\left(Dot\left(V_i^A, S\right)\right)$
   - Donde $min\left(Dot\left(V_i^A, S\right)\right)$ calcula el producto punto mínimo entre todos los vértices V_i^A, donde es el vértice i correspondiente al cuerpo A, y S
   - Donde $max\left(Dot\left(V_i^A, S\right)\right)$ calcula el producto punto máximo entre todos los vértices V_i^A, donde es el vértice i correspondiente al cuerpo A, y S

b) Calcular $min_B = min\left(Dot\left(V_i^B, S\right)\right)$ y $max_B = max\left(Dot\left(V_i^B, S\right)\right)$
   - Ídem función $min(\dots)$ y $\max(\dots)$

c) Si $min_A > max_B$ || $max_A < min_B$ entonces **los polígonos no colisionan(termina)**

2. Los polígonos colisionan(termina)

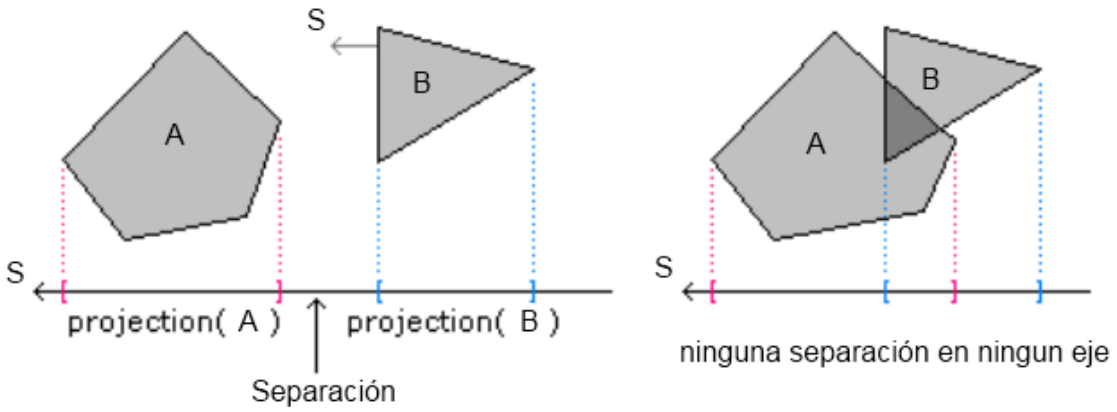


FIGURA 10: ESQUEMA DEL FUNCIONAMIENTO DE SAT EN 2D

Más adelante veremos que en muchos casos particulares de figuras de colisión podremos evitar algunos de los cálculos del caso general. Además notemos que lo que hace entonces el algoritmo SAT es darnos una salida temprana (llamadas EarlyOut), es decir si encontramos al menos un eje posible de separación donde los cuerpos no colisionen, es decir que es un eje de separación verdadero, entonces los cuerpos no colisionan. Dicho de otra forma para que los cuerpos colisionen se debe dar que en todos los ejes posibles de separación los polígonos se interpenetren. Entonces en los casos más comunes, donde los cuerpos no colisionan, no hará falta chechear todos los ejes posibles de separación sino que el algoritmo terminara antes, esta es una de las razones que este algoritmo sea eficiente como prueba. En la Figura 11 podemos ver una iteración posible de SAT en 2D.

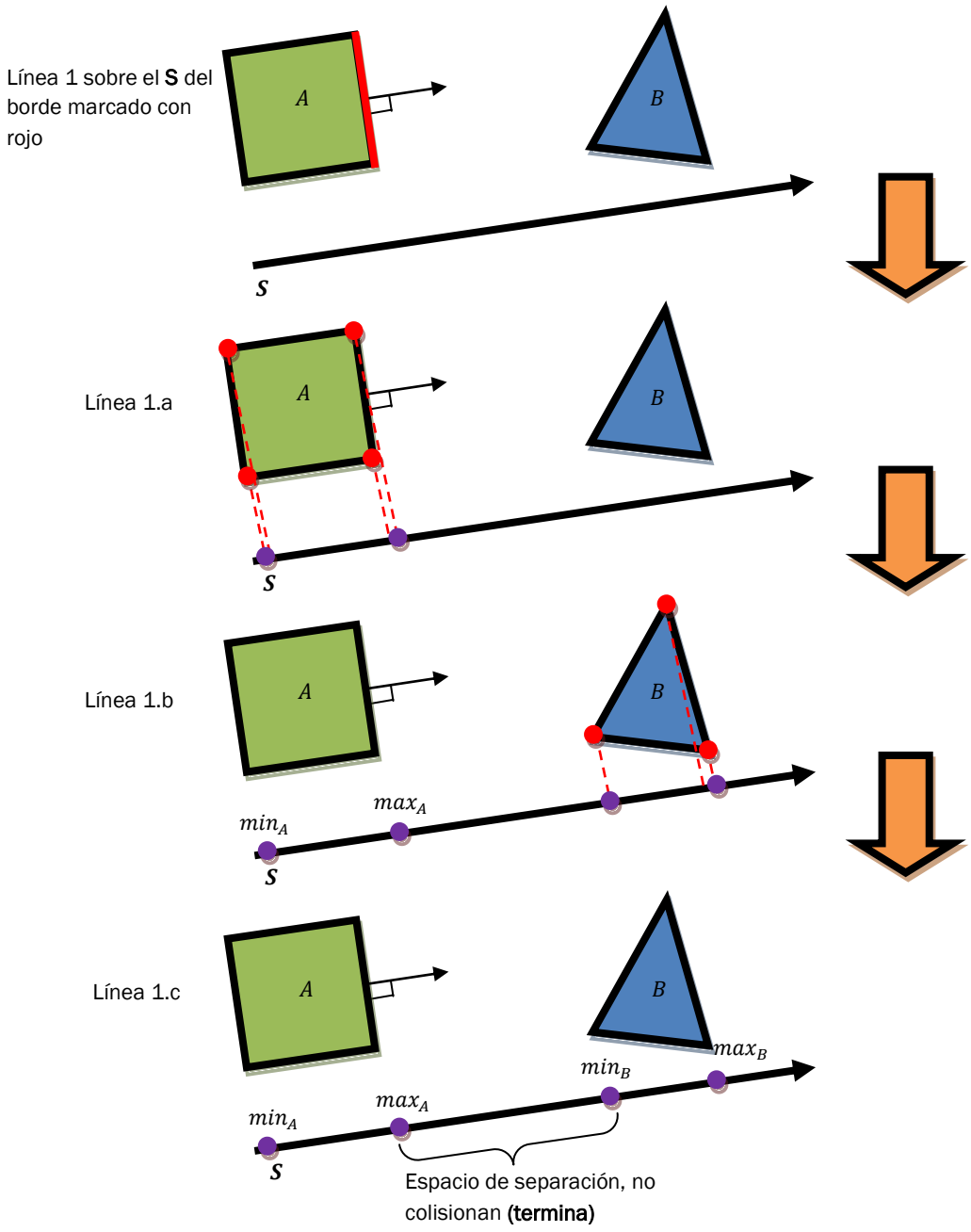


FIGURA 11: EJEMPLO DE ITERACIÓN DE SAT 2D

Limitaciones del algoritmo SAT:

Las limitaciones y problemas que tiene este método son las siguientes:

1. No sirve si alguno de los cuerpos es cóncavo.

2. Si los polígonos tienen muchas caras entonces el algoritmo se vuelve lento en comparación con otros métodos (como GJK), ya que tendremos en general $F_A + F_B$ ejes posibles de separación, donde F_X es el número de caras del polígono X.

3. Para cuerpos que sean convexos pero que no sean polígonos, como por ejemplo si es un círculo, tendremos que analizarlos de forma especial ya que los ejes posibles de separación dependerá de la forma de los cuerpos.

Ejemplos de test de colisión usando SAT:

Dos AABBs:

Recordemos que AABB son siglas de Axis Align BoundingBox, es decir que son 2 cajas que están alineadas con los ejes $\boldsymbol{x}$ e $\boldsymbol{y}$. Es por esto que las dos cajas no tienen orientación, o mejor dicho tienen la misma orientación. Ejemplos de objetos que se pueden representar con esta figura de colisión, es por ejemplo el jugador en un plataformer donde en general el jugador siempre está de pie y no se inclina, por otro lado en un caso, de por ejemplo, una pelota de rugby que reacciona de forma física y por lo tanto rota no nos serviría. Este es posiblemente el caso más simple que podemos considerar ya que solo hay dos ejes posibles de separación, que son los mismo ejes $\boldsymbol{x}$ e $\boldsymbol{y}$. Esto es así ya que como dijimos antes los ejes posibles de separación son las normales a las caras de los polígonos y al estar las AABB alineadas con los ejes, las normales son los mismos ejes $\boldsymbol{x}$ e $\boldsymbol{y}$, lo que significa que para sacar la proyecion de los vértices sobre el eje no hace falta realizar productos internos sino que ya la posición de los vértices en ese eje es igual a la proyección sobre ese eje. Además las dos cajas tienen la misma orientación y por lo tanto las mismas normales es decir que los ejes posibles de separación correspondientes a las normales de las caras de los dos cuerpos son las mismas. El pseudocódigo para realizar la prueba es como sigue:

```
bool TestAABBvsAABB(const BoundingBox& A, const BoundingBox& B)

{

 //en eje X estan separados?

                        if( A.m_MaxX < B.m_MinX || B.m_MaxX < A.m_MinX )

                                return false;


//en eje y estan separados?

                        if( A.m_MaxY < B.m_MinY || B.m_MaxY < A.m_MinY )

                                return false;


                        return true;//entonces estan colisionando

}
```

Vemos que ese es el método que ya conocíamos para probar colisiones entre AABB. Además pongamos atención en lo que dijimos antes, lo que hacemos es probar si en todos los ejes posibles de separación interpenetran, si lo

hacen están colisionando, pero si en al menos uno de los ejes no interpenetra luego no están colisionando. Un ejemplo donde los cuerpos están separados, en la Figura 12 vemos que en el eje y los cuerpos colisionan (en la región violeta), pero en el eje x no lo hacen y por eso el algoritmo concluye que no están colisionando.

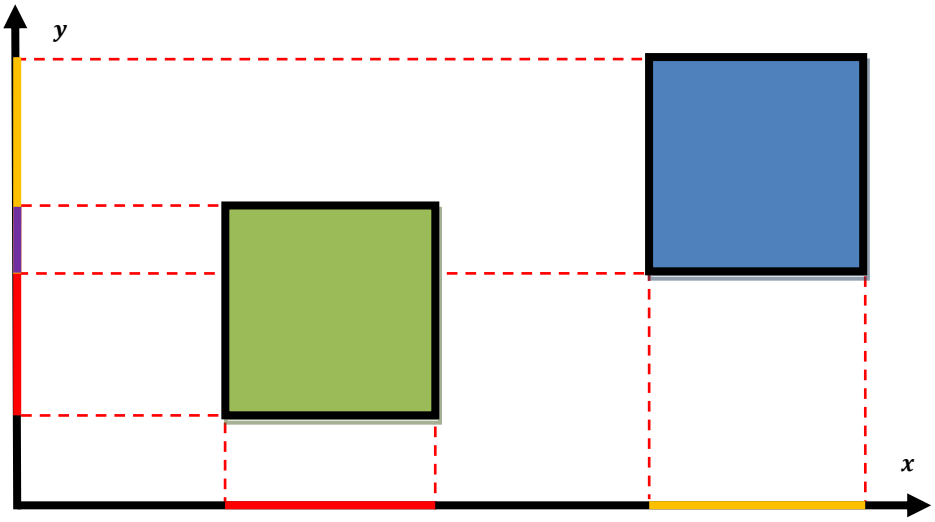


FIGURA 12: AABBS SEPARADOS

Otro ejemplo, donde los cuerpos colisionan, está en la figura 6. Vemos que los polígonos

Otro ejemplo, donde los cuerpos colisionan, está en la Figura 13. Vemos que los polígonos colisionan en los dos ejes posibles de separación, por lo tanto están colisionando.

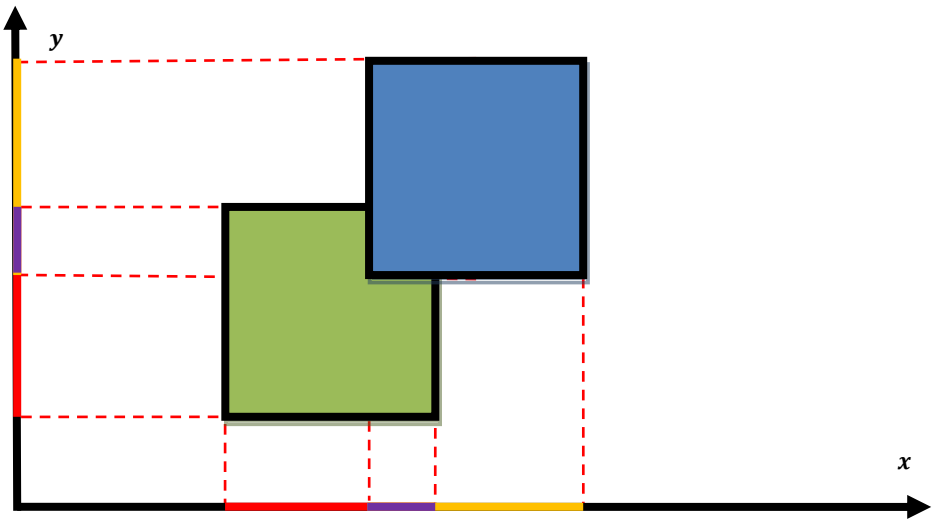


FIGURA 13: AABBS COLISIONANDO

Dos OOBBs:

Recordemos que OOBB son siglas de Object Oriented BoundingBox, es decir que son 2 cajas con orientaciones particulares cada una. Este es un caso un poco más complejo que el anterior, como las dos cajas pueden tener

orientaciones diferentes entonces las normales a sus caras en general serán distintas entre sí y además serán distintas a los ejes $\boldsymbol{x}$ e $\boldsymbol{y}$. Pero tiene la particularidad de que al ser una caja los lados opuestos poseen normales opuestas, es decir que con probar con cualquiera es suficiente ya que la dirección del eje de separación no nos dice nada, lo que hace que solo necesitemos probar dos normales por polígono. Usaremos una función que llamaremos $\boldsymbol{Perp}$, está en una función que dado un vector, que será un lado de un polígono, nos devolverá otro vector normal al de entrada, apuntando hacia afuera de ese lado, la implementación será:

$$Perp(x, y) = (y, -x)$$

```
Vector2<float> Perp(const Vector2<float> &edge)
{
	return Vector2<float>(edge.y,-edge.x);
}
```

El código del test será:

```
bool TestOOBBvsOOBB(const OOBB& A, const OOBB& B)
{
	Vector2<float> Normal[4];
Normal[0] = Perp(A.V2-A.V1);
Normal[1] = Perp(A.V3-A.V2);
Normal[2] = Perp(B.V2-B.V1);
Normal[3] = Perp(B.V3-B.V2);

float Amin = FLT_MAX,  Amax = FLT_MIN;
float Bmin = FLT_MAX,  Bmax = FLT_MIN;
float aux;
//Por cada eje normal
for(int i=0; i < 4 ;++i)
{
	//Calculo de A
	aux = Dot(A.V1 , Normal[i] );
	Amin = min( aux, Amin );
	Amax = max( aux, Amax );
	aux = Dot(A.V2 , Normal[i] );
```

```
            Amin = min( aux, Amin );
            Amax = max( aux, Amax );

            aux = Dot(A.V3 , Normal[i] );
            Amin = min( aux, Amin );
            Amax = max( aux, Amax );
            aux = Dot(A.V4 , Normal[i] );
            Amin = min( aux, Amin );
            Amax = max( aux, Amax );

            //Calculo de B
            aux = Dot(B.V1 , Normal[i] );
            Bmin = min( aux, Bmin );
            Bmax = max( aux, Bmax );
            aux = Dot(B.V2 , Normal[i] );
            Bmin = min( aux, Bmin );
            Bmax = max( aux, Bmax );

            aux = Dot(B.V3 , Normal[i] );
            Bmin = min( aux, Bmin );
            Bmax = max( aux, Bmax );
            aux = Dot(B.V4 , Normal[i] );
            Bmin = min( aux, Bmin );
            Bmax = max( aux, Bmax );

//en este eje posible de separacion, estan separados?
            if( Amin  >  Bmax || Amax < Bmin )
                        return false;
}

return true;//entonces estan colisionando
```

}

En la Figura 14 vemos dos polígonos que están separados, ahí solo se graficaron los ejes correspondientes a las normales del objeto $\boldsymbol{A}$. Vemos que el eje posible de separación $\boldsymbol{n_1}$ luego de la prueba no es un eje de separación verdadero, pero por otro lado el eje $\boldsymbol{n_2}$ si lo es y en su análisis el algoritmo sale concluyendo que no están colisionando.

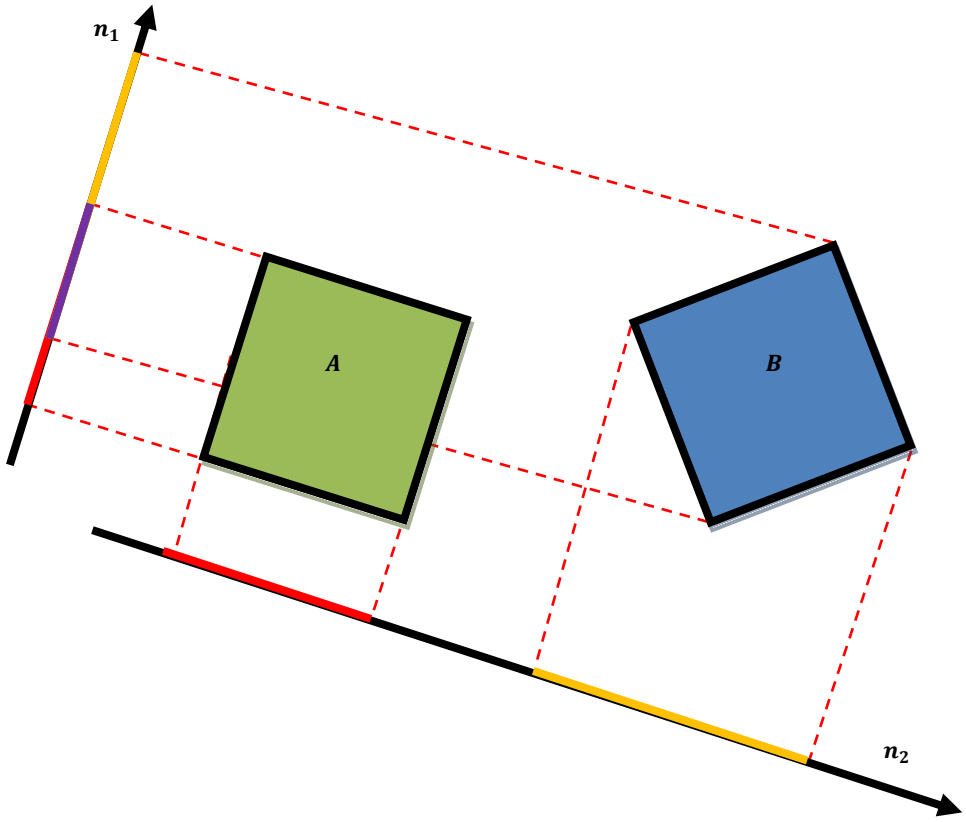


FIGURA 14: OOBBS SEPARADOS

Por otro lado en la Figura 15 vemos dos polígonos que están colisionando, de nuevo solo se graficaron los ejes posible de separación correspondientes al objeto $\boldsymbol{A}$. Podemos apreciar que la penetración se da en los dos ejes posibles de separación, y si analizamos los ejes correspondientes al polígono $\boldsymbol{B}$ veremos qué sucede lo mismo, por esto es que el algoritmo concluye que los polígonos están colisionando.

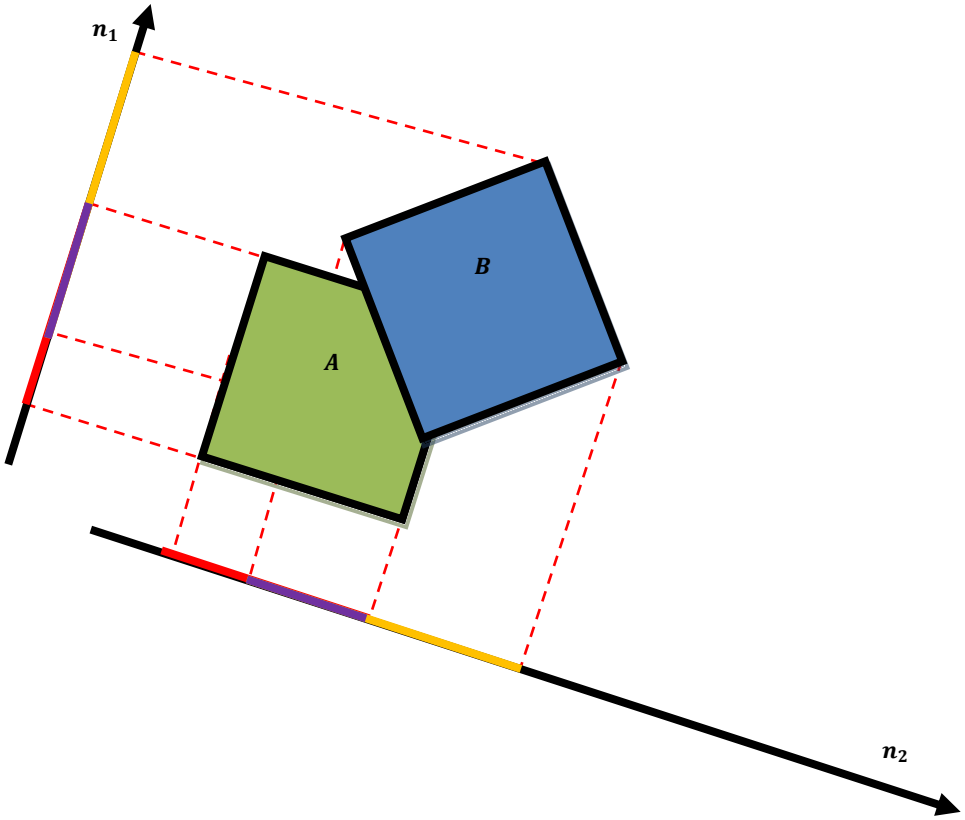


FIGURA 15: OOBBS COLISIONANDO

Un OOBB y un círculo:

Este es un caso particular ya que antes dijimos que para test de polígonos convexos con usar las normales a las caras de los dos polígonos como ejes posibles de separación era suficiente, pero si bien un circulo es un cuerpo convexo y por lo tanto el teorema del eje de separación se sigue aplicando un circulo tiene infinitas normales, como se ve en la Figura 16. A pesar de esto analizando podemos ver que con probar el eje que pasa por el centro del círculo y por cada uno de los vértices del OOBB, además de las normales del OOBB son todos los ejes posibles de colisión. Otra particularidad que tenemos que tener en cuenta es que en cualquier eje posible de colisión el círculo se proyecta con su diámetro, es decir proyectamos su centro y le sumamos y restamos el diámetro para sacar el máximo y mínimo respectivamente. La idea se grafica en la Figura 17.

El código es:

```
bool TestCirclevsOOBB(const Circle& A, const OOBB& B)

{

float Amin = std::numeric_limits<float>::max();

float Amax = std::numeric_limits<float>::min();

float Bmin = std::numeric_limits<float>::max();

float Bmax = std::numeric_limits<float>::min();
```

```
                float aux;


        //Por cada eje normal del OOBB
                for(int i=0; i < B.m_Normals.size() ; ++i)
                {
                        for(int j=0; j < B.m_Vertexs.size() ; ++j)
{
                                aux = abs(Dot(B.m_Vertexs[j], B.m_Normals[i]));
                                Bmax = max(aux,Bmax);
                                Bmin = min(aux,Bmin);
}

                        aux = abs(Dot(A.m_Center, B.m_Normals[i]));
                        Amax = aux+A.m_Radius;
                        Amin = aux-A.m_Radius;

//en este eje posible de separacion, estan separados?
                if( Amin  >  Bmax || Amax < Bmin )
                        return false;
                }

                sf::Vector2<float> circlePolygonVertexAxis;
//Por cada eje entre el centro circulo y vertice OOBB
                for(int i=0; i < B.m_Vertices.size() ; ++i)
                {
                        //Eje Posible de Separacion
                        EPS = A.m_Center - B.m_Vertices[i];

for(int j=0; j < B.m_ Vertexs.size() ; ++j)
{
                                aux = abs(Dot(B.m_Vertexs[j], EPS));
```

```
                                        Bmax = max(aux,Bmax);
                                        Bmin = min(aux,Bmin);
}


                                aux = abs(Dot(A.m_Center, EPS);
                                Amax = aux+A.m_Radius;
                                Amin = aux-A.m_Radius;

//en este eje posible de separacion, estan separados?
                        if( Amin  >  Bmax || Amax < Bmin )
                                return false;
                        }

                        return true;//entonces estan colisionando
}
```

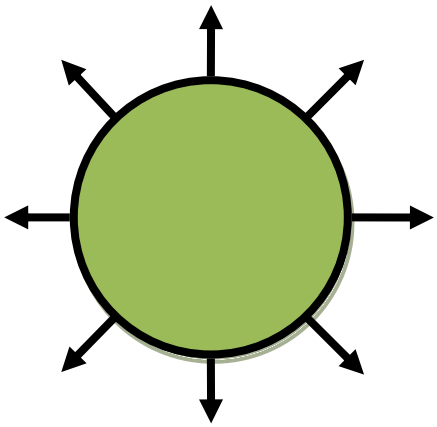


FIGURA 16: INFINITAS NORMALES DE UN CIRCULO

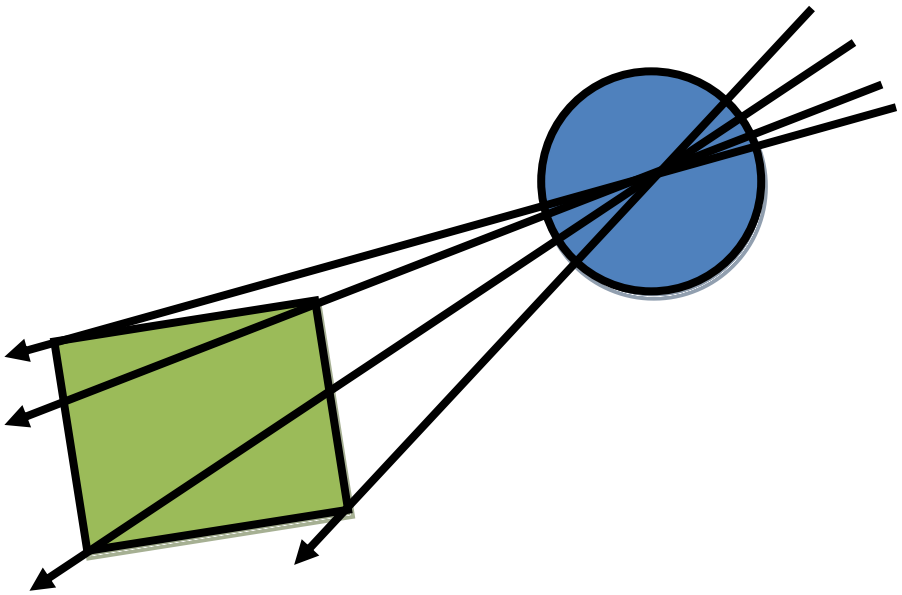
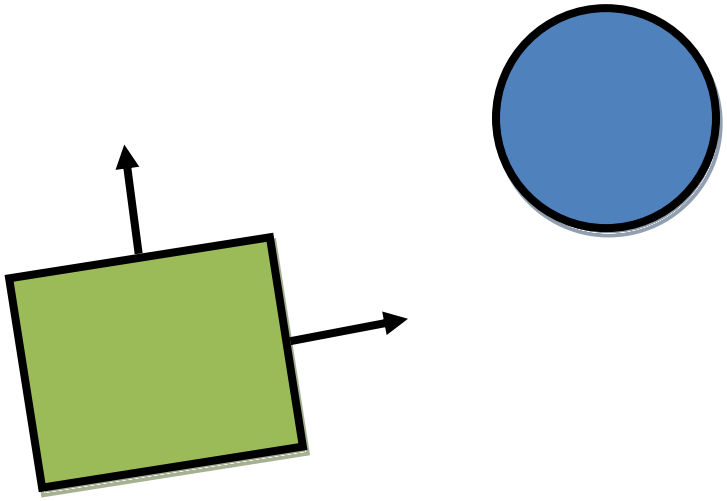


FIGURA 17: EJES POSIBLES DE SEPARACIÓN DEL CASO



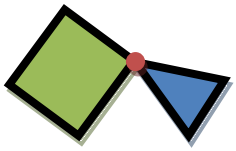
La implementación que se dio arriba en realidad no es la mejor forma de realizar la prueba, ya que en verdad el único eje posible de colisión entre el centro del circulo y los vértices es el que pasa entre el centro del circulo y el vértice del OOBB más cercano al círculo, así que una mejora simple para esto es en vez de probar con todos los ejes de todos los vértices solo probar con el más cercano al círculo. Existen otras mejoras a este algoritmo que no hablaremos aquí[2].

Generación de contactos:
Una colisión entre dos polígonos en 2D puede ser entre de 3 tipos:

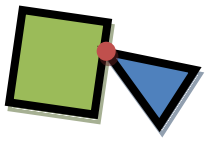
1. Vértice con vértice.
2. Cara con vértice o vice-versa.

[2] Esta mejora esta explicada en "A Simple Method for Box-Sphere Intersection Testing" de James Arvo.

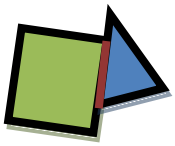
3. Cara con cara.



Vértice - Vértice



Cara - Vértice



Cara - Cara

Una vez que sabemos que un par de cuerpos colisionan luego deberemos obtener un par de datos de esa colisión, o contacto como nos gusta llamarla, que serán necesarias para luego resolver la colisión. En general los datos necesarios y suficientes para resolver la colisión son:

- **Punto de colisión:** Es el punto donde se da la colisión. En realidad si es una colisión entre dos vértices o entre una cara y un vértice será un punto, pero si es una colisión entre dos caras luego será un segmento determinado por 2 puntos. No obstante la selección del punto de colisión no es tan crítica, así que no afectara de sobremanera la simulación.

- **Normal de colisión:** Es la dirección en la que se aplicara un impulso por efecto de la colisión. En general la normal de colisión se calcula apuntando desde el primer cuerpo de la colisión hacia el segundo, es decir se toma la normal de colisión primer cuerpo (la del otro cuerpo será opuesta a esta).

- **Penetración:** Es un escalar que nos dice que tan interpenetrados están los cuerpos en el eje de la normal de colisión. Este dato es crucial para resolver interpenetraciones. Es decir si dos cuerpos están enterrados hasta cierta medida (penetración), luego para separar estos cuerpos entre si los moveremos la magnitud de penetración en la dirección de la normal de colisión.

Si decidimos solo almacenar un solo punto de colisión (por más que la colisión sea del tipo cara-cara), podríamos almacenar estas informaciones de contacto en una clase contacto como se ve en la Figura 18: Datos de contacto. El código seria de la forma:

```
class Contact

{

        Vector2<float> PuntoContacto;


        Vector2<float> Normal;


        float Penetracion;

};
```

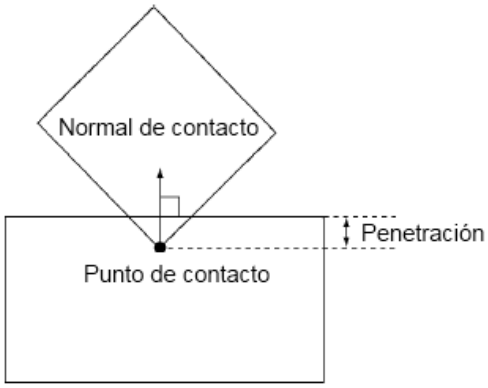


FIGURA 18: DATOS DE CONTACTO

En general el cálculo de los datos de contacto puede llegar a ser costoso. Por eso que muchas veces se utiliza una prueba como EarlyOut, verificando primero que la colisión se dé, luego estando seguro que hay colisión. Entonces en general se suele usar un código con la forma:

```
if (hayContacto())

{

        econtrarContacto();

}
```

Resolución de colisión

Hasta ahora hemos estudiado cómo detectar una colisión y qué datos son importantes a la hora de generar los contactos. El próximo paso es estudiar cómo reaccionar físicamente ante una colisión entre dos objetos. Para ello primero estudiemos físicamente qué principios físicos nos ayudan a resolver la colisión. Para ello veamos el tema de choques de la física clásica. Primero recordemos algunos conceptos que hemos estado viendo anteriormente y los veremos aplicados al caso de objetos puntuales.

Cantidad de movimiento en choques

Cuando dos objetos entran en contacto por un breve intervalo de tiempo y ejercen grande fuerzas uno contra el otro, da como resultado un cambio brusco de sus “movimientos”. Ejemplo de ello es cuando se patea una pelota, cuando dos vehículos colisionan, cuando se golpean las manos al aplaudir, todos estos ejemplos y por supuesto, muchos otros, se explican con la teoría de las “colisiones o choques”.

Las colisiones se analizan en función de una magnitud física asociada al movimiento y se denomina: “cantidad de movimiento”

Cuando una fuerza ejerce su acción sobre un cuerpo, de acuerdo a la segunda ley de Newton vimos que: $\bar{F} = m\bar{a}$

Si la Fuerza es constante, también lo es la aceleración, por tal motivo la ecuación anterior puede escribirse como:

$$\bar{F} = \frac{\Delta v}{\Delta t} m$$

$$\bar{F} = \frac{v_2 - v_1}{t_2 - t_1} m$$

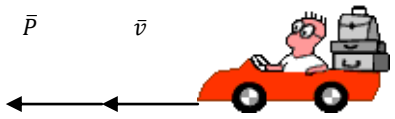
En donde v_1 y v_2y son las velocidades para el el intervalo de tiempo t 1 y t2.

A la ecuación anterior la podemos escribir de la siguiente manera:

$$\bar{F} = \frac{v_2 m - v_1 m}{t_2 - t_1}$$

De la última expresión podemos analizar que ésta ecuación se aplica para fuerzas que no son constantes. La fuerza ejercida sobre un cuerpo es igual a: La variación temporal de la cantidad $m.v$ y *que se denomina Cantidad de movimiento, la cual es una magnitud vectorial* y algunas bibliografías la representan como $\bar{P}$ cuyas unidades entonces serán: [Kg.] . [m/s].

La dirección y sentido de la cantidad de movimiento *P* coincide con la dirección y sentido de la velocidad



Ya definido Cantidad de movimiento *P* la última expresión se puede escribir como:

$$F = \frac{\Delta P}{\Delta t}$$

Por cuestiones de simplificación dejamos de colocar la rayita arriba de P, para solo tomar el módulo del vector (pero recordar que es una magnitud vectorial).

De la expresión anterior hacemos el siguiente análisis:

- La dirección y sentido de la cantidad de movimiento *P*, *coincide* con la dirección y sentido de la fuerza *F* ejercida.
- Que cuando dos cuerpos interactúan, se modifica la cantidad de movimiento de ambos.
- La intensidad de la variación de movimiento es *F. Δt,* por lo que depende de la intensidad de la fuerza que se ha ejercido en dicho intervalo de tiempo.
- La cantidad de movimiento *P*, de un cuerpo depende de su *velocidad y de su masa,* portal motivo para una masa determinada, la cantidad de movimiento será mayor cuanto mayor sea la velocidad

Ejemplo: sea un móvil que se mueve a 170 Km/h, habrá que aplicar una fuerza mayor para frenarlo que cuando el mismo móvil se mueve a 80 Km/h.

Para ambos casos la cantidad de movimiento **final** será nula ya que la velocidad final en ambos casos será cero, pero en el tiempo que el móvil se mueve a mayor velocidad la cantidad de movimiento será mayor en dicho tiempo

Choques o Colisiones:

Cuando se golpea una pelotita de tenis con la raqueta, se produce un choque entre dos cuerpos. Lo mismo ocurre al patear una pelota o bien podemos tomar ejemplos como el de los meteoritos al chocar con el planeta tierra, el fenómeno de Fusión entre partículas de uranio o los ejemplos más comunes y vistos lamentablemente por todos cuando colisionan dos vehículos. En todas las situaciones se analizan los cambios en el sistema a partir de la ***conservación de la cantidad de movimiento.***

¿Cómo se produce un choque y que pasa con cada cuerpo en ese instante?

Se considera que se produce un choque entre dos cuerpos cuando éstos interactúan en un lapso **muy corto** de tiempo.

Cuando dos cuerpos chocan, la cantidad de movimiento del **sistema** (ejemplo pelota y raqueta) se conserva, es decir:

$$\bar{P} = m_1 v_1 + m_2 v_2 = constante$$
$$\bar{P} = \bar{P}_1 + \bar{P}_2$$

Suma de cantidades de movimiento del sistema formado por el cuerpo 1 y el 2, que puede ser perfectamente en la dirección x e y.

Un ejemplo para entender como es éste fenómeno físico es el análisis de las bolas de pool, cuando golpea una contra otra. En dicho juego, se debe golpear la bola blanca, la cual sale y debe chocar con otra bola de color que se encuentra en reposo. Para facilitar el análisis, supongamos que la bola blanca solo tiene velocidad inicial en la componente en x, entonces la cantidad de movimiento para dicha bola blanca es:

$$\bar{P}_0 = m_b v_0 \qquad con\ \ v_0 = velocidad\ inicial$$

La cantidad de movimiento para la otra bola de color es cero, ya que está en reposo.

Luego de la colisión, la bola de color deja su estado de reposo, entonces:

$$\bar{P}_f = m_b v_2 + m_c v_1$$

¿Cómo se explica esto?

Cuando dos cuerpos chocan entre sí pueden ocurrir diferentes situaciones:

1- **Que los cuerpos interactúen de tal manera que varía la energía cinética del sistema:** en éstos casos solo se mantiene constante la cantidad de movimiento del sistema (formado por los dos cuerpos)

2- Que los cuerpos interactúen, choquen y que luego permanezcan juntos después de la colisión, también acá variará la energía cinética del sistema.

En estos casos se dice que los cuerpos quedan ***pegados*** por lo que a estos tipos de choques se los denomina: ***Choques Plásticos o Inelásticos***

Ejemplo de ello se da cuando un vehículo embiste a otro y luego del impacto, ambos continúan moviéndose juntos. Uno arrastrado por el otro. En éste caso ocurre que la velocidad, luego del impacto, es la misma para los dos vehículos, pero la energía cinética del sistema disminuye debido a las deformaciones

3- Que los cuerpos interactúen de tal manera que la energía cinética del sistema permanezca constante.

Sea el choque de bolas de pool, se puede afirmar que tanto la energía cinética como la cantidad de movimiento del sistema se mantienen constante, a éste tipo de choque se los llama ***Choques elásticos***

$$\bar{P} = constante = m_1 v_1 + m_2 v_2 = m_1 v_1' + m_2 v_2'$$
$$E_c = \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 v_1'^2 + \frac{1}{2} m_2 v_2'^2$$

De las ecuaciones anteriores se identifica que:

v_1 y v_2 son las velocidades antes del choque y v_1' y v_2' son las velocidades luego del choqu.

Entonces podemos decir que:

En un ***choque elástico*** *se conserva la Energía cinética* Total, esto quiere decir que la energía cinética *total* de todos los objetos del sistema después del choque, es igual a su energía cinética *total* antes del choque. Esto se traduce como que los objetos al chocar no se deforman.

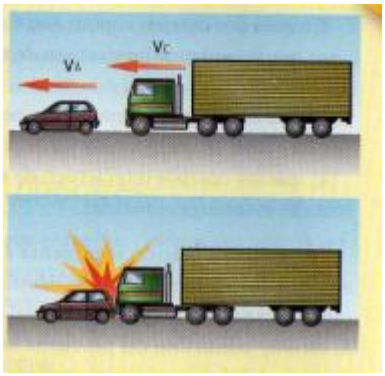
$$E_{cf} = E_{ci}$$

En el ***choque Inelástico***, *no se conserva la energía cinética total*, eso explica el hecho que, los objetos al chocar no recuperan su forma original o también podría generar calor o sonido por la fricción, esto hace que se pierda algo de energía cinética

$$E_{cf} < E_{ci}$$

Ejemplo de aplicación 1:

Un camión de 5000 Kg viajaba de una localidad otra por una ruta recta a una velocidad de 80 Km/h. La visibilidad era muy buena y no había neblina, sin embargo por causas que aún no se conocen, el camión ha **chocado** a un automóvil de 1000 kg que se desplazaba adelante del él a 60 km/h, Calcular:



a) ¿Cuál era la velocidad de los vehículos inmediatamente después del impacto?
b) ¿Qué transformaciones energéticas se desarrollaron durante el choque?
c) ¿Cuál era la variación de energía mecánica del sistema debido al choque?

Solución:

a) En el choque se conserva la cantidad de movimiento del sistema, o sea

$$\bar{P}_i = \bar{P}_f$$

En el instante antes del choque:

$$\bar{P}_i = m_a v_a + m_c v_c$$

Luego del choque $\bar{P}_i = (m_a + m_c)v$

Usando ambas:

$$m_a v_a + m_c v_c = (m_a + m_c)v$$

$$v = \frac{m_a v_a + m_c v_c}{m_a + m_c} v = \frac{1000\ Kg\ .60\frac{Km}{h} + 5000\ Kg\ .80\frac{Km}{h}}{6000\ Kg} v \cong 68\frac{Km}{h}$$

b) La energía cinética del sistema, formado por el auto y el camión disminuyó, parte se disipó como calor y otra parte ocupó en trabajo de deformación, ambos vehículos quedaron averiados y abollados luego del choque.

c)

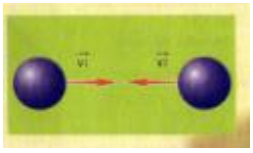
$$E_{ci} = \frac{1}{2} m_a v_a^2 + \frac{1}{2} m_c v_c^2$$
$$E_{ci} = \frac{1}{2} 1000\ Kg. \left(60\frac{Km}{h}\right)^2 + \frac{1}{2} 5000\ Kg. \left(80\frac{Km}{h}\right)^2$$
$$E_{ci} = 17800\ J$$

por otro lado

$$E_{cf} = \frac{1}{2}(m_a + m_c)v^2$$
$$E_{cf} = \frac{1}{2}(1000\ Kg + 5000\ Kg). \left(68\frac{Km}{h}\right)^2$$
$$E_{cf} = 13872\ J$$
$$\Delta E_c = E_{cf} - E_{ci} = -3928\ J$$

Ejemplo de Aplicación 2:

Dos bolas de masa 0,1 Kg y 0,2 Kg, se desplazan en la misma dirección y **chocan y rebotan.** Calcular la velocidad de cada uno después de chocar elásticamente si al principio se mueven con sentido contrario y velocidades de 10 m/s y 5,0 m/s.



Solución:

En un choque elástico se conserva la cantidad de movimiento y la energía cinética.

De acuerdo con el sistema de referencia los datos son:

$v_1 = 10\frac{m}{s}$ $\quad$ $m_1 = 0{,}1\ Kg$

$v_2 = -5{,}0\frac{m}{s}$ $\quad$ $m_2 = 0{,}2\ Kg$

$$\bar{P}_i = m_1 v_1 + m_2 v_2$$
$$\bar{P}_i = 0.1\ Kg.10\frac{m}{s} + 0.2\ Kg\ .\left(-5\frac{m}{s}\right)$$

$$\bar{P}_f = m_1 v_1' + m_2 v_2'$$
$$\bar{P}_f = 0.1\ Kg\ v_1' + 0.2\ Kg\ v_2'$$

$$E_{ci} = \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2$$

v_1 y v_2 son las velocidades antes del choque

$$E_{ci} = \frac{1}{2}\ 0.1\ Kg\ 100\frac{m^2}{s^2} + \frac{1}{2} 0.2\ Kg\ 25\frac{m^2}{s^2}$$
$$E_{cf} = \frac{1}{2} 0.1\ Kg\ {v_1'}^2 + \frac{1}{2} 0.2\ Kg\ {v_2'}^2$$

v_1' y v_2' son las velocidades después del choque

Recordemos que:

$\bar{P}_i = \bar{P}_f$ y $E_{cf} = E_{ci}$

Por medio de igualación y sustitución en las ecuaciones matemáticas de arriba, se obtiene que:

$$V_1' \cong -10\frac{m}{s} \qquad v_2' \cong 5\frac{m}{s}$$

Ahora veamos que sucede si aplicamos estas ideas a los cuerpos rígidos.

Respuesta de cuerpos rígidos

Como se dijo anteriormente para simular cómo reaccionan dos cuerpos al colisionar podemos usar fuerzas o impulsos. Como nuestros cuerpos son rígidos y la respuesta la aplicamos en un solo paso de tiempo vamos a trabajar con impulsos que nos permiten cambiar “instantáneamente” la velocidad de los cuerpos de manera de que los mismos no se penetren.

Recordemos la definición de velocidad de aproximación:

$$v_c = v_a \cdot (p_b - p_a) + v_b \cdot (p_a - p_b)$$

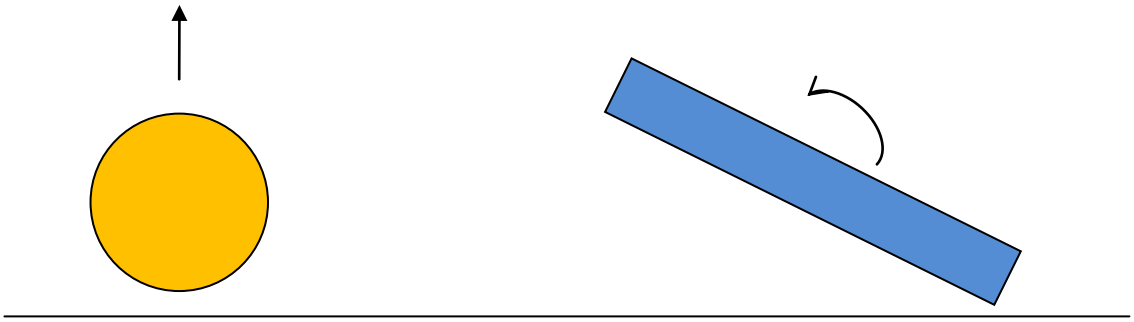
Que es la velocidad a la cual dos cuerpos v_a y v_b con posiciones p_a y p_b se aproximan.

Imaginemos sin perder generalidad que tenemos un cuerpo A que rebota con otro B que es fijo en el espacio. Supongamos que el punto de contacto en A es a_c y en B b_c. Dependiendo del material de los cuerpos A rebotará o se quedará pegado a B. La velocidad con que el punto a_c se aleja del punto b_c luego del contacto está dada por:

$$v_c' = -cv_c$$

Dónde v_c' es la velocidad de aproximación luego del contacto y c es el coeficiente de restitución que determinará cómo rebotarán los objetos. Este coeficiente depende del material de los objetos involucrados en la colisión. Si $c = 1$ entonces tendremos una colisión elástica, dónde el punto se alejará a la misma velocidad a la que se acercaba conservando la energía. Si $c = 0$ luego ambos objetos quedarán pegados ya que velocidad de aproximación luego de la colisión será 0. Con este parámetro podemos configurar en la mayoría de los motores físicos cómo serán las colisiones entre objetos. Este coeficiente además es una medida de la disipación de energía que se producirá en el choque.

Como estamos trabajando con cuerpos rígidos al chocar el cuerpo A recibirá un impulso lineal (como si fuera un objeto puntual) y un impuso angular que hará que el cuerpo rote debido al choque. Si imaginamos una pelota rebotando en el suelo, como la fuerza de gravedad pasa por el centro de masa de la misma, entonces la pelota al chocar contra el piso sólo recibirá un impulso lineal, mientras que si arrojamos por ejemplo una regla como se muestran en la figura, la misma recibirá un impulso lineal pequeño y un gran impulso angular que hará que el cuerpo rote:



Impulso lineal

La componente lineal es muy sencilla, veamos el cambio que produce en la velocidad lineal un impulso unitario:

$$\Delta v_l = \frac{1}{m}$$

Esto quiere decir que el cambio de la velocidad es igual al a inversa de la masa del objeto A.

Impulso angular

En este caso la derivación es más compleja por lo que omitiremos la derivación. Sea p la posición del punto de contacto relativa al origen del cuerpo A. Luego el impulso angular u generado por un impulso unitario es:

$$u = p \times n$$

Dónde n es la normal en el punto de contacto a_c. Por otro lado el cambio de velocidad angular debido al impulso angular u es

$$\Delta\omega = I^{-1}u$$

Dónde I es el tensor de inercia del cuerpo.

Finalmente con estas dos ecuaciones podemos obtener el cambio de velocidad lineal del punto debido al impulso angular u

$$\Delta v_a = \Delta\omega \times p$$

Impulso y velocidad

Veamos en esta sección cómo calcular el impulso necesario para obtener un Δv deseado:

$$g = \frac{\Delta v}{r}$$

Dónde g es el impulso buscado y r es el cambio de velocidad por impulso unitario visto en las dos secciones anteriores:

$$r = \Delta v_l + \Delta v_a$$

Con estas ecuaciones ya podemos calcular cuál es el impulso que debemos aplicar al cuerpo en el punto de contacto para que se produzca un cambio de velocidad Δv.

Velocidad resultante

Ahora que ya sabemos cómo calcular el impulso necesario para crear el cambio de velocidad del cuerpo, es necesario que determinemos cuál será la velocidad final del punto de contacto luego de la colisión.

La velocidad con que los puntos de colisión se alejarán luego de impacto está dado por la relación vista anteriormente:

$$v_c' = -cv_c \Rightarrow \Delta v_c = -v_c - cv_c = -(1+c)v_s$$

Pasando en limpio:

$$\Delta v_c = -(1+c)v_s$$

Este es el cambio de velocidad que debe resultar de aplicar el impulso. Con este cambio de velocidad podemos ir a la sección anterior y calcular cuál es el impulso necesario para producirlo. Luego aplicamos el impulso obtenido y tendremos nuestro objeto A rebotando de B.

Si el objeto B no fuera estático entonces deberíamos repetir los cálculos para B teniendo en cuenta que la normal será la inversa de la normal usada en A.

Interpenetración

Como la detección de colisiones se suele realizar a posteriori es normal que los objetos se interpenetren generando un comportamiento no realista. Debido a esto no alcanza con aplicar el impulso resultante de la colisión sino que además es necesario detectar y separar los objetos que se enciman. Para esta tarea existen diferentes técnicas en las cuales no entraremos en detalle. En la bibliografía pueden investigar y extenderse sobre este tema.

La forma más sencilla de resolver el problema se denomina “proyección lineal” y consiste en separar los objetos moviéndolos en la dirección de la normal hasta que no se encimen más. La cantidad que se mueven es proporcional a la inversa de la masa. Es el método más sencillo de implementar. Otras técnicas populares más avanzadas son resoluciones basadas en la velocidad y proyecciones no lineales.

Colisiones en Box2D

Box2D pone a disposición del usuario la clase “contact” que posee información de un contacto. Para acceder los contactos que se han producido se puede hacer de dos formas: Iterando o mediante Listeners.

Contactos por Iteración

Box2D permite acceder a todos los contactos que se produjeron en el mundo durante el paso de tiempo de la siguiente manera:

```
for (b2Contact* c = myWorld->GetContactList(); c; c = c->GetNext())

{

//Hacer algo con c

}
```

Si solamente necesitamos los contactos de un cuerpo determinado, entonces podemos hacer:

```
for (b2ContactEdge* ce = myBody->GetContactList(); ce; ce = ce->next)

{

b2Contact* c = ce->contact;

//Hacer algo con c

}
```

De esta forma podemos recuperar estructuras b2Contact. Estas estructuras poseen información sobre el contacto producido. Entre los métodos más destacables podemos mecionar:

- GetManifold(): Devuelve un objeto b2Manifold que posee información sobre el tipo de contacto, el punto en el cual sucedió y la normal de contacto.
- GetFixtureA()/GetFixtureB(): Devuelven los fixtures que colisionaron. Y mediante estos se puede acceder a los objetos b2Body que los contienen.
- SetEnabled(true/false): Permite ignorar un contacto en el paso de tiempo actual

Listeners

Otra alternativa que tenemos para recuperar información de las colisiones es trabajando con los ContactListeners. Estas son clases que reciben notificaciones antes eventos de colisión. Para utilizarlos debemos crear una clase que herede de b2ContactListener e implementar los métodos según las necesidades:

```
class MyContactListener : public b2ContactListener

{

public:
```

```
void BeginContact(b2Contact* contact)

{ // evento que se llama cuando comienza el contacto }

void EndContact(b2Contact* contact)

{ // evento que se llama cuando termina el contacto }

void PreSolve(b2Contact* contact, const b2Manifold* oldManifold)

{ // evento que se llama luego de la detección de colisiones pero antes      //de la resolución de las mismas }

void PostSolve(b2Contact* contact, const b2ContactImpulse* impulse)

{ // evento que se llama luego de la resolución de colision y da //información del impulso resultante de la misma  }

};
```

Luego esta clase se debe registrar con el mundo de Box2D usando el método

```
phyWorld= new b2World(b2Vec2(0.0f,9.8f) ,true);

MyContactListener *l= new MyContactListener();

phyWorld->SetContactListener(l);
```

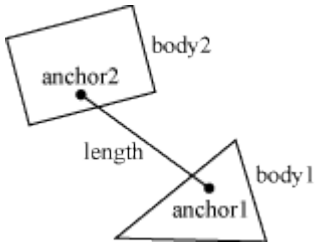
Con esta clase se puede determinar cuándo dos objetos chocan o cuando alguno cae sobre otro, etc. Es importante destacar que desde estos métodos no es posible modificar el estado de Box2D, ya que generaría estados inconsistentes durante el paso de tiempo.

Joints

Por último vamos a aprender cómo trabajar con restricciones en Box2D. Las joints o restricciones son condiciones que se le imponen a un cuerpo o par de ellos que deben respetar durante la simulación. Estas condiciones pueden involucrar por ejemplo que dos cuerpos deban mantenerse siempre a la misma distancia o que un cuerpo no pueda moverse a lo largo de un eje, etc. Los resortes por ejemplo son un tipo de joint en Box2D.

En todos los casos debemos llenar una estructura describiendo el tipo de Joint y luego pedirlo al mundo de Box2D que lo cree. A continuación veremos los casos más comunes de Joints:

Distance Joint



Este joint hace que dos objetos se mantengan a la misma distancia. Debemos llenar una estructura b2DistanceJointDef:

```
b2DistanceJointDef jointDef;

jointDef.Initialize(myBody1, myBody2, worldAnchorOnBody1, worldAnchorOnBody2);

jointDef.collideConnected = true;
```

Dónde myBody1 y myBody2 son de tipo b2Body y worldAnchoronBody1/2 indican los puntos de cada objetos que se unen.

Este tipo de Joints pueden comportarse como un resorte usando las siguientes propiedades:

```
jointDef.frequencyHz = 4.0f;  //frecuencia de oscilación

jointDef.dampingRatio = 0.5f; //entre 0 y 1 haces las veces de K del resorte
```

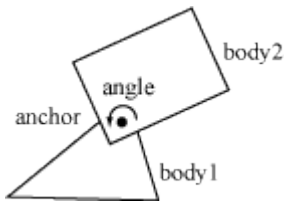
Revolute Joint

Estos joints unen dos cuerpos en un punto de anclaje y les permite rotar alrededor del mismo.

```
b2RevoluteJointDef jointDef;

jointDef.Initialize(myBody1, myBody2, myBody1->GetWorldCenter());
```

Los dos primeros parámetros son los cuerpos que intervienen y el tercero es un punto en coordenadas globales que funcionará de anclaje o “anchor” en inglés como se muestar en la siguiente figura:



Además estos joints permiten establecer un ángulo máximo y mínimo de rotación relativa y una velocidad de rotación. Esta velocidad de rotación hace que los cuerpos roten alrededor del punto de anclaje simulando un motor por ejemplo.

En el siguiente ejemplo se configuran los ángulos máximos y mínimos y el torque máximo que puede aplicar el joint. También se setea la velocidad del motor a 0:

```
jointDef.lowerTranslation = -5.0f; //limite inferior en radianes

jointDef.upperTranslation = 2.5f; //limite superior en radianes

jointDef.enableLimit = true; //activamos los limites

jointDef.maxMotorTorque = 1.0f;  //el torque maximo que puede aplicar

jointDef.motorSpeed = 0.0f; //la velocidad deseada

jointDef.enableMotor = true; //si deseamos que se comporte como un motor
```

Existen muchos casos de joints además de los dos vistos, pero no los desarrollaremos en detalle en el apunte. Los interesados pueden referirse a la documentación de Box2D.

BIBLIOGRAFÍA

Ian Millington. Game Physics Engine Development. Morgan Kauffman Publishers, San Francisco 2007.

Gettys W.; Keller, F.; Skove, M. *Física Clásica y Moderna*. McGraw-Hill Inc., Madrid, 1991.

Sears, F.; Zemansky, M.; Young, H.; Freedman, R. *Física Universitaria*. Vol. 1, Addison Wesley Longman, 1998.

Resnic, Halliday. *Física para estudiantes de Ciencias e Ingeniería*. Parte I, México, Compañía Editorial Continental SA, 1967.

Alonso, Finn. *Física: Vol. I: Mecánica*. Fondo Educativo Interamericano, 1970.

Botto, J.; González, N.; Muñoz, J. *Fís Física*. Buenos Aires, Tinta Fresca, 2006.

Gaisman, M.; Waldegg Casanova, G.; Adúriz-Bravo, A.; Díaz, F.; Lerner, A.; Rossi, D. *Física. Movimiento, interacciones y transformaciones de la energía*. Buenos Aires, Santillana, 2007.