Tractament de col·lisions en temps real

Carles Ros Martínez

is04732@salleurl.edu

(16 de maig del 2001)

Taula de contingut

1. Introducció

Molts sistemes d'animació interactius necessiten treballar amb una gran quantitat d'entitats virtuals que es mouen i interactuen entre elles. En aquestes animacions no es pot predir amb anterioritat quin serà el comportament de l'usuari i les entitats; per tant, cal anar creant l'animació en temps real. Això vol dir que la imatge mostrada s'ha d'anar redibuixant al menys 10 frames per segons (fps) per ser acceptada per l'usuari com a interactiva, i fins a 50 fps per ser considerada amb un rendiment de temps real vertader. Així, només hi haurà disponibles, en el millor dels casos, de 20 milisegons per dur a terme l'actualització de totes les entitats de la simulació i representar l'escena.

En aquest article es presenta un esquema general per afrontar el problema del tractament de col·lisions en temps real. A més, degut a la seva estreta relació amb el tema del tractament de col·lisions, es parlarà de dos mètodes per simular la força de gravetat i es tractarà breument el problema de la pèrdua de precisió de decimals que sorgeix en els ordinadors. Al final de l'article també s'explica detalladament un exemple d'un cas real de tractament de col·lisions implementat en el motor d'un joc.

1.1. Components d'un sistema de tractament de col·lisions

Un sistema de tractament de col·lisions consisteix de dos components. El primer component és un algoritme de detecció de col·lisions (també anomenat **algoritme de detecció**), el qual cerca objectes les superfícies dels quals hagin començat a penetrar. Si l'algoritme de detecció troba algun objecte en aquest estat, llavors el sistema de tractament de col·lisions crida al segon component, un algoritme de reacció a les col·lisions (també anomenat **algoritme de reacció**). Aquest algoritme s'encarrega de corregir el comportament dels objectes de manera que deixin de penetrar. La correcció pot implicar l'aplicació de certes lleis de la dinàmica, com ara fer que un objecti reboti.

Aquest article es centrarà sobretot en la detecció de col·lisions, explicant només una aproximació al lliscament i el rebot pel que fa a la reacció a les col·lisions, en ser aquestes les reaccions més usuals. Per un millor comportament en la reacció a les col·lisions caldria entrar en més profunditat en el món de la dinàmica.

2. Detecció de col·lisions

Tot algoritme de detecció pot experimentar certs problemes. Per entendre'ls i poder-los afrontar es parteix d'un algoritme bàsic, mostrant els errors que presenta i corregint-los.

2.1. Un algoritme de detecció bàsic

La figura 2.1 presenta un algoritme de detecció bàsic i una aplicació que el crida.

```
 1  Aplicació()
 2  {
 3      per ( t = t0 fins a t1 en increments de trep ) {
 4          obté dades dels dispositius d'entrada
 5          actualitza el comportament dels objectes per t
 6          fes {
 7              infoCol = detecta( t, objectes )
 8              si ( infoCol conté col·lisions )
 9                  resposta a les col·lisions
10          } mentre ( infoCol contingui col·lisions )
11          representa cada objecte de objectes per t
12      }
13  }
14
15  /* La variable tant persisteix d'una crida a l'altre. */
16  /* Inicialment tant = t0. */
17  detecta( tact, objectes )
18  {
19      per ( t = tant fina a tact en increments de tdet ) {
20          mou objectes a la seva posició per t
21          per ( cada objecte Oi de objectes ) {
22              per ( cada objecte Oj de {objectes – Oi} )
23                  si ( Oi i Oj intersecciones )
24                      afegir Oi i Oj a infoCol
25          }
26          si ( infoCol conté col·lisions ) {
27              tant = t
28              retorna ( infoCol )
29          }
30      }
31      tant = tact
32      retorna ( infoCol )
33  }
```

Figura 2.1: Un algoritme de detecció bàsic

L'aplicació representa un frame de l'animació cada t_{rep}. Per representar-lo, primer obté dades de l'usuari i després actualitza el comportament dels objectes (línies 4 i 5).

Tot seguit es crida l'algoritme de detecció per saber si alguns dels objectes estan en col·lisió (línia 7). Si és així, es crida l'algoritme de reacció per corregir el comportament (línia 9). Com que el nou canvi de comportament pot comportar noves col·lisions cal tornar a passar per tot el procés del tractament de col·lisions (línia 10). Finalment, l'aplicació representa tots els objectes (línia 11) i avança al següent frame.

L'algoritme de detecció cerca col·lisions en l'interval entre t_{ant} i t_{act}; és a dir, des de l'ultima vegada que ha estat cridat fins al temps actual de l'aplicació. El procés de detecció es dur a terme cada t_{det}; per tant, s'assumeix que t_{rep} és un múltiple de t_{det}. A cada pas t l'algoritme de detecció troba la posició per cada objecte segons t (línia 20). Llavors l'algoritme comprova per totes les parelles d'objectes (línies 21 i 22) si es produeix alguna intersecció (línia 23) i l'afegeix a la llista de parelles en col·lisió (línia 24), la qual es retorna després de comprovar totes les col·lisions i guardar-se el temps de l'última detecció (línies 26 a 29). En cas de no produir-se cap col·lisió el temps de l'última detecció serà el temps actual de l'aplicació i es retorna buida la llista de parelles en col·lisió (línies 31 i 32).

2.2. Problemes de l'algoritme de detecció bàsic

Apareixen tres problemes en l'algoritme de detecció bàsic:

1) Problema de comprovar les col·lisions cada cert instant de temps a intervals regulars (línia 19). És el que s'anomena ***fixed-timestep weakness***.
2) Problema de comprovar les interseccions entre totes les parelles d'objectes de l'escena (línies 21 i 22). S'anomena ***all-pairs weakness***.
3) Problema de determinar si les superfícies de dos objectes interseccionen (línia 23). Rep el nom de ***pair-processing weakness***.

2.3. Fixed-timestep weakness

El fet de comprovar si dos objectes interssseccionen cada t_{det} instant de temps pot comportar que certes col·lisions no es detectin. La figura 2.2 n'és un clar exemple.

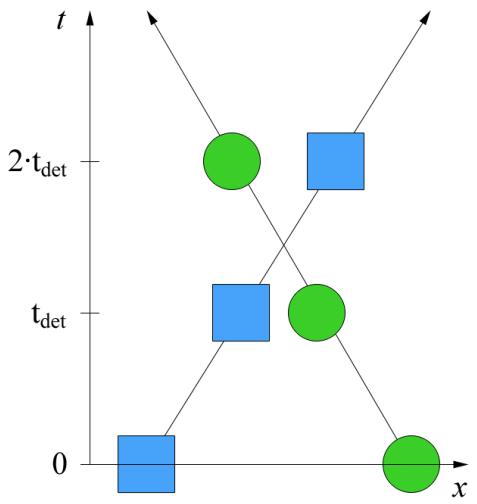


Figura 2.2: L'algoritme de detecció bàsic es salta aquesta col·lisió

En l'exemple els dos objectes es mouen en sentit contrari en l'eix x i mostra la seva posició en cada increment de temps t_{det}. Com que només es comprova si els dos objectes intersecciones cada t_{det} instant de temps, no es detecta la col·lisió i, per tant, durant la simulació els dos objectes es travessaran. Escurçant la llargada de t_{det} es redueix la possibilitat de que apareguin aquests errors, ja que es duran a terme més comprovacions d'intersecció durant el mateix interval de temps; però, d'altra banda, l'algoritme també tindrà un pitjor rendiment. Una manera d'evitar aquest problema és fer que t_{det} s'adapti a cada situació (***adaptive timestep***): haurà de ser més gran quan menor sigui la possibilitat de col·lisió i més petit quan més possibilitat de col·lisió hi hagi.

2.3.1. *Bounding* de moviment

Tot i tenir un increment de temps adaptiu, per més petit que sigui t_{det}, sempre hi haurà un interval entre dos instants de temps marcats per t_{det} en el que no es comprovarà les col·lisions, possibilitant l'aparició d'aquest tipus d'error. El que interessa és poder comprovar les col·lisions durant tot l'interval de temps, en comptes de només a certs instant de temps. Per poder fer això es té en compte l'evolució espacial d'un objecte durant l'interval de temps i es defineix un embolcall (*bounding*) que engloba tot l'espai pel que passa. A aquest embolcall se li dóna el nom de ***bounding* de moviment**. Ara, a cada procés de detecció de col·lisions es generen els boundings de moviment i la prova d'intersecció es fa entre aquests. Si es produeix una intersecció se sap que en algun instant de l'interval de temps s'ha produït una col·lisió. A la figura 2.3 es pot veure com, amb els boundings de moviment aplicats al cas de la figura 2.2, aquests intersecciones i, per tant, es detecta la col·lisió.

Una altra avantatge ve del fet de que la precisió en la detecció de col·lisions ara serà igual de bona independentment de la fragmentació de l'interval de temps, per tant, es pot reduir la detecció de col·lisions a una sola iteració, de manera que els boundings de moviment ja englobin tot l'interval de temps entre t_{ant} i t_{act}.

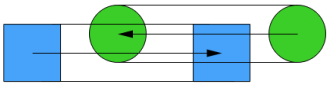


Figura 2.3: Els boundings de moviment solucionen la *fixed-timestep weakness*.

Degut a la freqüent generació i comprovació d'intersecció de boundings de moviment, cal que sigui ràpida tant la generació com la comprovació d'intersecció dels boundings. Això porta a que cal escollir un tipus de bounding senzill, el qual amb quasi tota seguretat tindrà l'inconvenient de que no representarà exactament l'espai pel que ha passat l'objecte. Per una correcta detecció de col·lisions, el bounding de moviment ha de ser prou gran com per incloure tot l'espai pel que passa l'objecte; per tant, hi haurà una regió del bounding que no correspondrà a l'objecte. Si la intersecció entre dos boundings es produeix en aquesta regió, es detectarà una falsa col·lisió. Per solucionar això després de detectar una intersecció entre dos boundings cal comprovar si els dos objectes realment col·lisionaran en algun moment de l'interval de temps. En un sistema sense boundings de moviment això seria una operació costosa i inviable per una simulació a temps real, però els boundings de moviment descarten ràpidament moltes

col·lisions, reduint el número d'aquestes comprovacions. La figura 2.4 mostra aquest problema utilitzant un AABB com a tipus de bounding de moviment (els tipus de bounding de moviment s'explicaran més detalladament quan es doni una solució a la *pair-processing weakness*).

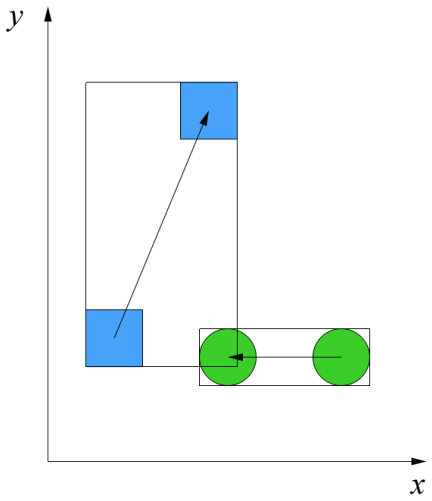


Figura 2.4: Es detecta una falsa col·lisió degut a la mala aproximació del bounding de moviment.

Aquesta comprovació també evita un altre problema que pot sorgir del fet d'utilitzar boundings de moviment. Els boundings de moviment només guarden la informació espacial de l'objecte i no la temporal. Per tant, tot i que el bounding aproximi perfectament l'espai pel que passa un objecte, no retén la posició on estava a diferents instant de temps, cosa que fa que es pugui detectar una col·lisió en l'espai en què realment els objectes no haguessin de col·lisionar temporalment. Però gràcies a la comprovació que soluciona el cas anterior també queda solucionat aquest problema. A la figura 2.5 s'utilitzen boundings que aproximen perfectament l'espai pel que passa l'objecte, i tot i així es pot veure com apareix el problema esmentat.

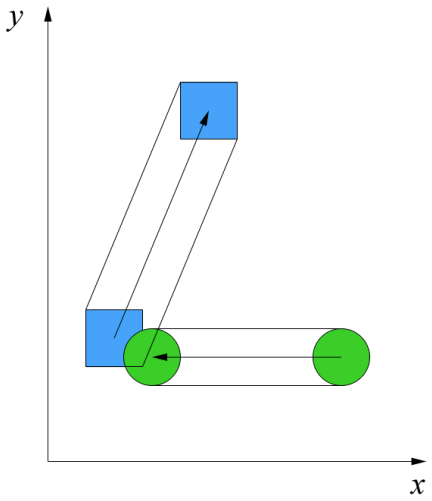


Figura 2.5: Es detecta una falsa col·lisió degut a que el bounding de moviment no guarda la informació temporal.

En cas de voler guardar la informació temporal junt amb l'espacial en el mateix bounding de moviment es podria optar per utilitzar boundings espacio-temporals, els quals serien com els boundings de moviment però tenint el temps com a quarta dimensió. Aquest bounding evita el problema anterior, però també augmentarà el cost en la comprovació d'interseccions entre boundings. A més, el fet de no poder representar-los gràficament, a diferència dels boundings de 3 dimensions, en dificulta la comprensió i implementació. En favor de la claredat de comprensió, en aquest article només es tractarà amb boundings de 3 dimensions.

Hi ha un altre cas en què els boundings de moviment poden dur a detectar una falsa col·lisió. Quan un objecte col·lisiona amb un altre aquest pot canviar el seu comportament, podent fer que la seva trajectòria variï i, per tant, es produeixin noves col·lisions pel que resta d'interval de temps (o no es produeixin les que s'havien detectat posteriorment). Per tant, de totes les col·lisions que es detecten durant l'interval de temps s'ha de tractar primer la més immediata temporalment, descartant totes les col·lisions posteriors en què estaven implicats els objectes que han col·lisionat, i, un cop tractada la col·lisió, cal tornar a detectar les col·lisions per aquests objectes pel que queda de l'interval de temps a partir de l'instant de la col·lisió. Aquest procés s'ha d'anar repetint fins que no quedin col·lisions per tractar. En la figura 2.6 la primera iteració del procés de detecció de col·lisions només detecta la col·lisió entre el quadrat i el cercle. Un cop tractada la col·lisió, tant el quadrat com el cercle han variat la seva trajectòria i, degut a aquest canvi, el quadrat col·lisiona amb el triangle. Aquesta col·lisió no es detectarà a menys que es torni a passar el quadrat pel procés de detecció de col·lisions a partir de l'instant de la seva col·lisió amb el cercle.

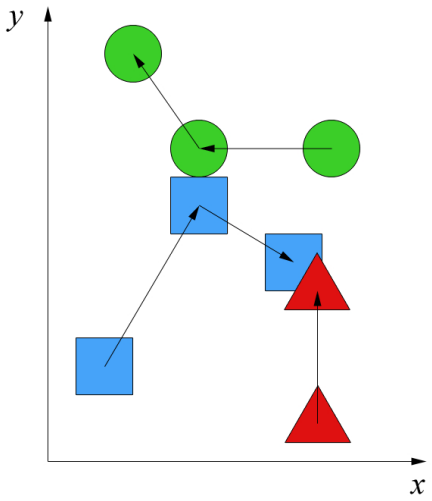


Figura 2.6: Es recalculen les col·lisions pels objectes que col·lisionen.

2.4. All-pairs weakness

La utilització de boundings de moviment augmenta el rendiment del procés de detecció de col·lisions, ja que els boundings descarten ràpidament moltes de les parelles d'objectes que no col·lisionen. Això comporta que per escenes amb pocs objectes ja sigui viable una simulació a temps real. Però, a mesura que l'escena va creixent en nombre d'objectes, la gran quantitat de comprovacions d'intersecció que es du a terme farà que el sistema deixi de funcionar en temps real.

Una característica comuna en la gran majoria d'escenes és que els objectes de què la composen solen estar distribuïts en un espai molt gran en relació a la mida d'aquests. Això fa que es comprovin interseccions entre boundings de moviment d'objectes que estan molt allunyats l'un de l'altre, sent normalment molt poques les que genereran una intersecció. Aquí l'interessant serà tenir una estructura que organitzi els objectes de manera que es pugui descartar qualsevol possibilitat de col·lisió entre molts dels objectes sense haver de fer la prova d'intersecció. Una bona solució es basa en dividir l'espai en regions més petites, de manera que els objectes que estan en dos regions diferents ja es pot afirmar que no col·lisionaran.

Normalment aquesta estructura és compartida amb el sistema de visualització del motor de l'aplicació, cosa que farà que no depengui només del sistema de detecció de col·lisions el fet de triar l'estructura més adequada. Tenir una estructura separada de la utilitzada pel sistema de visualització pot agilità la detecció de col·lisions, però abans de decantar-se per aquesta solució cal avaluar el cost afegit que suposarà per l'aplicació el fet de tenir i mantenir una altra estructura, tant per l'impacte en el rendiment com per la memòria extra que necessitarà.

Aquest article està orientat a un model general del tractament de col·lisions i una estructura d'aquest tipus pot ser totalment diferent d'una aplicació a una altra segons les necessitats de casdascuna; per tant, no s'entra en detall en cap tipus d'estructura per organitzar els objectes. Tot i així cal mencionar que algunes de les més utilitzades són els arbres *BSP*, les basades en el *portal engine* i els *octrees*.

2.5. Pair-processing weakness

Degut a la complexitat en la forma que un objecte pot tenir, definir un algoritme que comprovi si les superfícies de dos objectes interaccionen en un cert moment de la simulació implicarà que l'algoritme hagi de contemplar una gran multitud de casos en què aquestes superfícies poden interseccionar, fent que l'algoritme sigui molt poc eficient. Si els objectes tinguessin formes menys complexes, llavors es reduiria automàticament el nombre de casos en què les superfícies d'aquests podrien interseccionar, podent així definir un algoritme més ràpid. Per tant, per tal d'accelerar aquesta fase, s'aproximarà els objectes a una forma més senzilla, de manera que l'algoritme que comprovi les interseccions ho faci entre aquestes formes i no entre els objectes en si. Cal tenir en compte que aquesta aproximació implica que els objectes utilitzats pel sistema de visualització i el de col·lisions seran diferents; per tant, per evitar que l'usuari es senti desconcertat per aquesta diferència entre el que veu i el que realment col·lisiona, convindrà arribar a un compromís entre l'objecte en si i la seva aproximació, que, com sempre, dependrà de les necessitats de cada aplicació. A la figura 2.7 es pot veure com es detecta una col·lisió abans de que l'objecte visualitzat realment hagi col·lisionat.

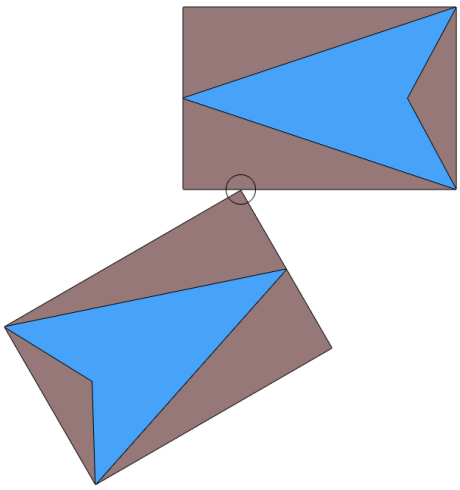


Figura 2.7: L'objecte del món de les col·lisions xoca, cosa que no succeeix amb el del món visual.

Igual que amb la solució donada per la *fixed-timestep weakness* aquí també es proposa la utilització d'un embolcall per tal d'aproximar l'objecte. Com ja s'ha dit anteriorment, la gràcia d'utilitzar un bounding senzill és que comprovar interseccions entre ells és molt eficient. També cal dir que normalment serà una bon criteri que el bounding inclogui totalment l'objecte que aproxima, doncs d'una altra manera l'usuari podria veure com certes parts dels objectes interseccionarien sense detectar-se una col·lisió. En aquest article es pren aquest criteri.

Cada tipus de bounding té els seus avantatges i inconvenients; però, per regla general, com millor s'aproxima a l'objecte un bounding més lent serà comprovar les interseccions entre ells. Com a orientació a l'apartat 3 se'n detallen alguns dels més utilitzats, presentant els avantatges i inconvenients de cadascun.

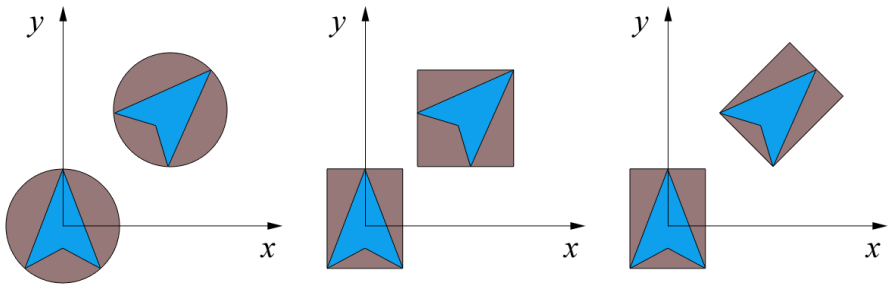
2.6. Sopa de polígons

En algunes simulacions a més d'objectes també caldrà tractar amb grups de polígons individuals. Un joc és un clar exemple d'aquest cas, a on hi ha objectes que es mouen per un món estàtic format de polígons. Això implica que també caldrà detectar col·lisions entre els objectes i els polígons. La seqüència per determinar una col·lisió entre un objecte i un polígon segueix els mateixos passos que la utilitzada entre dos objectes, amb la diferència que el polígon sempre és estàtic. A més, normalment interessarà considerar el polígon en si en comptes d'aproximar-lo amb un bounding, de manera que l'error en la col·lisió degut a aproximar l'objecte no augmenti més.

3. Comparativa dels tipus de boundings més comuns

Degut a la gran importància d'escollir els tipus de boundings més adequats per utilitzar com a bounding de moviment i per aproximar els objectes, en aquesta secció es comparen els més comuns, indicant com es comporta cadascun davant les diferents operacions que es realitzen duran la detecció de col·lisions. Abans, però, es presenten els diferents tipus de boundings que s'utilitzaran en la comparativa, els quals també es mostren a les figures 3.1a, 3.1b i 3.1c envoltant un objecte:

- **Esfera (*bounding sphere*)**: esfera normal.
- **Caixa alineada al món (*axis aligned bounding box – AABB*)**: com el nom indica, és una caixa que sempre manté la mateixa orientació que els eixos del món, de manera que cada cara és paral·lela a algun dels plans definits pels eixos del món i perpendicular als altres dos.
- **Caixa orientada (*oriented bounding box – OBB*)**: a diferència de la caixa anterior, aquesta pot tenir una orientació diferent a la dels eixos del món.



D'esquerra a dreta, figures 3.1a, 3.1b i 3.1c: Esfera, caixa alineada al món i caixa orientada per aproximar l'objecte, en la seva posició inicial i amb l'objecte transformat.

Cada tipus de bounding serà bo en algunes propietats i operacions i dolent en altres, de manera que el que pot ser adequat per una aplicació no ho sigui en una altra.

En aquesta comparativa es separa el bounding de moviment de l'utilitzat per aproximar l'objecte, doncs les propietats desitjades que han de complir ambdós són diferents. Tot i així es té en compte la relació que existeix entre ambdós.

3.1. Comparativa dels tipus de boundings de moviment

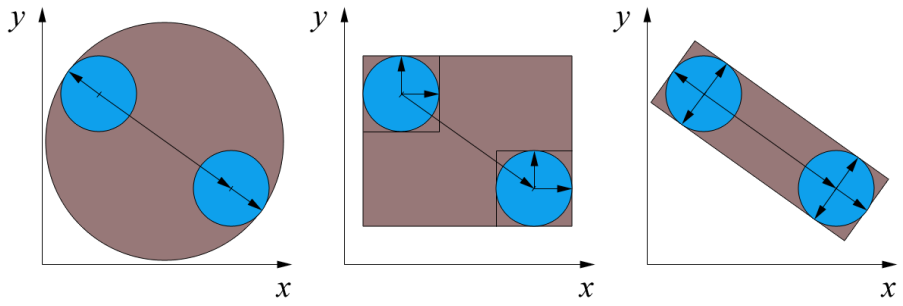
Un bounding de moviment interessa que sigui el màxim d'òptim en les següents característiques:

- Rapidesa de creació
- Màxima aproximació a l'espai pel que passa l'objecte
- Eficiència en l'algoritme d'intersecció

3.1.1. Creació

La rapidesa en la creació del bounding de moviment no vindrà només determinada pel tipus de bounding escollit per aquest, sinó també pel tipus de bounding escollit per aproximar l'objecte, doncs segons la complexitat del bounding que aproximi l'objecte serà més o menys fàcil determinar l'espai pel que ha passat i, en conseqüència, això afectarà a la velocitat de creació del bounding de moviment. Típicament, però, crear una esfera o una caixa alineada al món és més ràpid que crear una caixa orientada.

Per crear una esfera cal trobar els dos punts més allunyats de tot l'espai pel que passa l'objecte, els quals faran de diàmetre de l'esfera. Les figures 3.2a, 3.2b i 3.2c presenten un exemple on s'utilitzen esferes per aproximar l'objecte i un tipus de bounding de moviment diferent a cadascuna, i, concretament a la figura 3.2a, el bounding de moviment és una esfera. A l'exemple es pot veure que per determinar el diàmetre de l'esfera només ha calgut sumar al vector de velocitat el radi de l'esfera en ambdós extrems.



D'esquerra a dreta, figures 3.2a, 3.2b i 3.2c: Esfera, caixa alineada al món i caixa orientada com a boundings de moviment.

Crear una caixa alineada al món representa crear primer les caixes alineades al món de la posició inicial i final de l'objecte i llavors trobar la caixa alineada al món pel bounding de moviment és una operació quasi gratuïta. La figura 3.2b mostra com es crea una caixa alineada al món pel bounding de moviment, creant primer les caixes per la posició inicial i final de l'objecte, que en aquest cas surten de sumar al centre de l'objecte el radi de l'esfera en cadascun dels eixos del món.

La caixa orientada per crear com a bounding de moviment ha de tenir la mateixa orientació que el vector de velocitat i les seves dimensions vindran determinades pels punts més llunyans de l'objecte respecte el vector de velocitat. A la figura 3.2c es dóna un exemple d'una caixa orientada com a bounding de moviment, per la qual s'han determinat les seves dimensions sumant al centre de l'esfera el seu radi, dos cops en sentit perpendicular al vector de velocitat i un altre com a extensió d'ambdós extrems del vector de velocitat.

3.1.2. Aproximació a l'espai pel que passa l'objecte

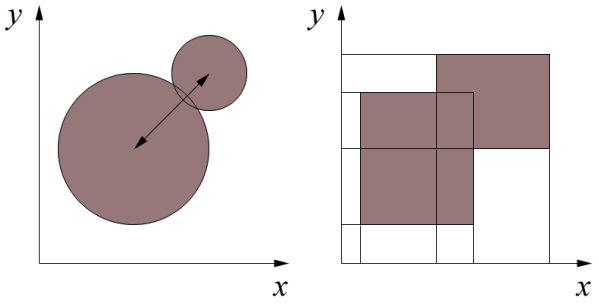
Donant un cop d'ull a les figures 3.2a, 3.2b i 3.2c es veu clar que el tipus de bounding que millor aproxima l'espai pel que passa l'objecte és la caixa orientada, la qual només conté espai no ocupat per l'objecte en els dos extrems. Aquest espai, a més, es manté independentment de la llargada del vector de velocitat, a diferència dels altres dos tipus de boundings, que com més gran sigui el vector de velocitat més espai no ocupat per l'objecte contindran, augmentant la possibilitat de detectar col·lisions falses que després faran augmentar el cost global del sistema de detecció de col·lisions. El fet

de que aquest empobriment en l'aproximació és molt exagerat en el cas de l'esfera fa que se solgui descartar com a tipus de bounding de moviment, doncs l'avantatge que representaria en els altres dos aspectes a considerar tenen un rendiment semblant als de la caixa alineada al món, la qual dóna una millor aproximació a l'espai pel que passa l'objecte. De fet la caixa alineada al món aproxima l'espai pel que passa l'objecte d'una manera molt semblant a la caixa orientada quan el vector de velocitat és molt petit o molt proper en direcció a algun dels eixos del món.

Com a regla general, els tipus de boundings que millor aproximen l'espai pel que passa l'objecte solen tenir un major cost en els altres dos aspectes. Per tant, a l'hora de triar un tipus de bounding caldrà avaluar si el fet de descartar més col·lisions falses en aquesta etapa compensa el cost afegit en les altres etapes.

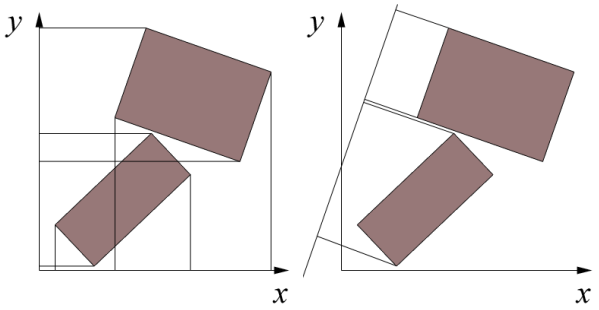
3.1.3. Operació d'intersecció

Aquí tornen a ser els més eficients l'esfera i la caixa alineada al món. En el cas de dues esferes, determinar si es produeix una intersecció entre elles es redueix a comprovar si la distància que les separa és menor que la suma dels seus radis. Pel que fa a dues caixes alineades al món, es poden projectar les dues sobre cadascun dels tres plans definits pels eixos del món i, aprofitant la propietat de que totes les cares d'una d'aquestes caixes són perpendiculars a algun dels eixos del món, determinar si les dues caixes interseccionen es redueix a operacions de comparació d'una sola dimensió. Comprovar si dues caixes orientades interseccionen ja és una operació més complexa, doncs hi ha múltiples casos on dues d'aquestes caixes poden interseccionar. Les figures 3.3a, 3.3b i 3.3c mostren un exemple de cadascun dels tres casos esmentats, mostrant a la figura 3.3c com el mètode de projecció utilitzat per les caixes alineades al món no és suficient en el cas de les orientades.



D'esquerra a dreta, figures 3.3a i 3.3b: Mètodes per determinar interseccions entre esferes i entre caixes alineades al món.

Per determinar si dues caixes orientades interseccionen un mètode a seguir seria projectar les dues caixes sobre cadascun dels plans definits per les cares de les dues caixes, cosa que dóna un màxim de 6 plans, i, si en tots els plans les projeccions interseccionen, llavors les dues caixes també estan en intersecció.



Esquerra, figura 3.3c: El mètode utilitzat per determinar interseccions entre dues caixes alineades al món no és suficient per les orientades.
Dreta, figura 3.3d: En un dels plans de projecció no es produeix intersecció entre les projeccions de les dues caixes.

3.2. Comparativa dels tipus de boundings per aproximar l'objecte

A l'hora de triar el tipus de bounding més apropiat per aproximar l'objecte caldrà que aquest sigui el màxim d'òptim en les següents característiques:

- Rapidesa d'actualització
- Màxima aproximació a la forma de l'objecte
- Eficiència en l'algoritme d'intersecció

3.2.1. Actualització

Degut a la gran quantitat d'objectes que pot contenir una escena, el bounding d'un objecte normalment es crea només un cop, amb l'objecte que aproxima centrat a l'origen de coordenades. Llavors només si l'objecte experimenta una transformació cal actualitzar el seu bounding.

Pel que fa a transformacions de translació, els tres tipus de bounding solucionen l'actualització aplicant-se la mateixa translació. És en el cas de les transformacions de rotació on apareixen grans diferències de rendiment. En aquest aspecte destaquen les esferes, doncs són rotacionalment invariables i, en conseqüència, la seva actualització a la rotació és gratuïta. Pel que fa a les caixes orientades, la seva operació d'actualització és força ràpida, doncs només caldrà que se'ls hi apliqui la mateixa rotació que experimenti l'objecte. És en les caixes alineades al món on l'operació d'actualització deguda a una rotació ja no és tant trivial. Quan l'objecte rota hi ha una gran probabilitat de que part de l'objecte quedi fora del seu bounding, fent que sigui necessaria una nova reconstrucció del bounding per tal de que contingui tot l'objecte. Això representa haver de recórrer tots els vèrtex de l'objecte, amb el cost de rendiment que suposa. Les figures 3.1a, 3.1b i 3.1c mostren un objecte amb el seu bounding. A l'objecte se li ha aplicat una transformació de translació i una altra de rotació.

Una alternativa és utilitzar caixes alineades al món prou grans com per contenir sempre l'objecte independentment de la seva orientació, de manera que sigui rotacionalment invariable; però això té l'inconvenient de que ara el bounding aproximarà pitjor al seu objecte, podent encarir el cost de les altres etapes del sistema de detecció de col·lisions, a més d'augmentar el marge d'error amb el món visual.

3.2.2. Aproximació a la forma de l'objecte

Dir que un tipus de bounding aproximarà a un objecte millor que un altre depèn exclusivament de la forma dels objectes que conté una escena. Tot i així, com a regla general els objecte solen tenir una forma més aviat rectangular, i, per tant, les caixes acostumen a aproximar millor l'objecte que les esferes. Concretament aquest és el punt fort de les caixes orientades, i sol ser aquesta la principal raó per la que una aplicació que necessiti simular més fiablement el món real les pot triar, tot i el cost que suposen en les demés etapes.

Fins i tot les caixes orientades poden no ser suficient pels requeriment de certes aplicacions, en les que caldria utilitzar tipus de boundings que aproximessin millor els objectes. Però aquests boundings suposarien un cost tant gran que no permetrien una execució en temps real. Una solució és utilitzar arbres de boundings, de manera que a cada nivell s'aproximi més fiablement l'objecte. Aquí cal dir que les caixes orientades, degut a la seva propietat d'orientar-se, un arbre construït amb elles convergirà cap a la forma de l'objecte més ràpidament que els altres dos tipus de boundings.

Una altra variant és tenir un bounding més senzill contenint el bounding que realment aproxima l'objecte. Per exemple, es podria tenir una caixa alineada al món per descartar ràpidament moltes col·lisions, i, dins d'aquesta caixa, una d'orientada per tal de simular més fiablement les col·lisions.

3.2.3. Operació d'intersecció

Un cop determinada una intersecció entre dos boundings de moviment, encara cal esbrinar si realment els objectes col·lisionen i a quin punt i moment ho fan. Amb esferes i caixes alineades això és pot solucionar fàcilment, però no així amb les caixes orientades, on el cost d'aquesta operació pot ser molt elevat, sobretot degut a la complexitat que afegeix el fet d'haver de tenir en compte el temps. Aquest cost pot fer que sigui indispensable utilitzar els boundings espacio-temporals esmentats anteriorment, el quals ajudaran a determinar el moment en què es produeix la col·lisió, eliminant així la component del temps de l'operació de detectar el punt de col·lisió entre les dues caixes orientades.

4. Reacció a les col·lisions

Un cop determinat el punt de col·lisió entre dos objectes, cal variar el seu comportament segons aquest punt de col·lisió i les lleis de la dinàmica que s'hagin implementat en la simulació. En aquest article, però, no s'entra en el món de la dinàmica, i només s'explica una aproximació de dues de les reaccions a les col·lisions més comuns: el lliscament i el rebot.

4.1. Lliscament

La solució proposada és molt simple. Primer de tot cal trobar el pla de lliscament, el qual és el mateix que el pla de col·lisió. Aquest passarà pel punt de col·lisió i la seva orientació dependrà de la posició i velocitat en el moment de la col·lisió i també de la forma dels dos boundings que col·lisionen. Un cop determinat el pla de lliscament es projecta el tros restant del vector de velocitat en el moment de la col·lisió sobre aquest pla i s'escurça aplicant-hi un factor de fricció. El vector resultant indicarà cap a on i quant cal desplaçar l'objecte per simular una reacció de lliscament. La figura 4.1 presenta dues caixes orientades en col·lisió, per les quals s'ha calculat el vector de lliscament seguint el mètode explicat.

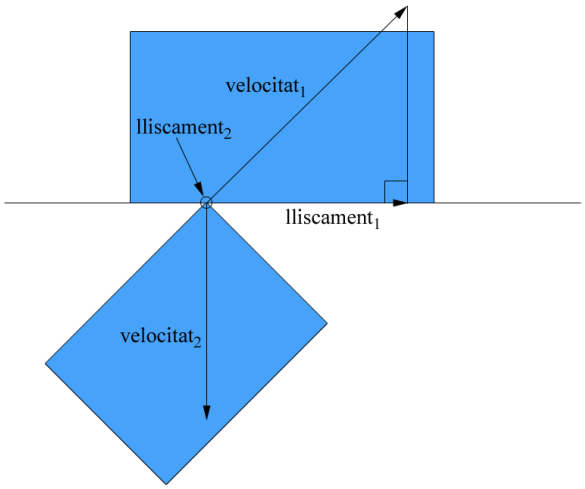


Figura 4.1: Excedents dels vectors de velocitat projectats sobre el pla de col·lisió entre dues caixes orientades.

Aquesta reacció permetrà a un objecte pujar certs obstacles i lliscar per les parets, que d'altra manera farien que l'objecte s'aturés.

4.2. Rebot

Pel que fa al rebot, la solució proposada és tant simple com la proposada pel lliscament, doncs la reacció per rebot segueix un mètode molt semblant. De fet, les úniques diferències són que en comptes de projectar el tros restant del vector de

velocitat sobre el pla de col·lisió, ara cal reflectir aquest vector, i que en comptes d'escurçar-lo amb un factor de fricció, s'escurça amb un factor d'esmorteïment. La figura 4.2 en mostra un exemple.

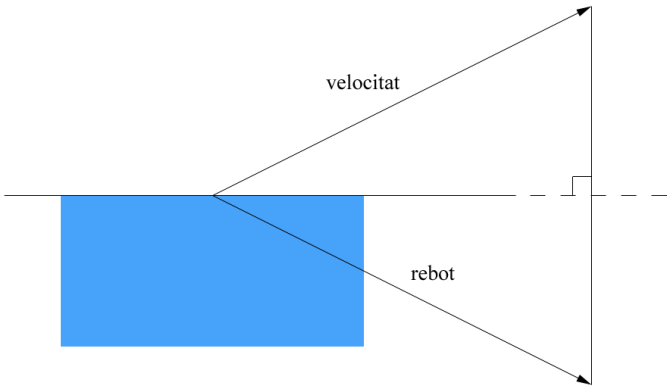
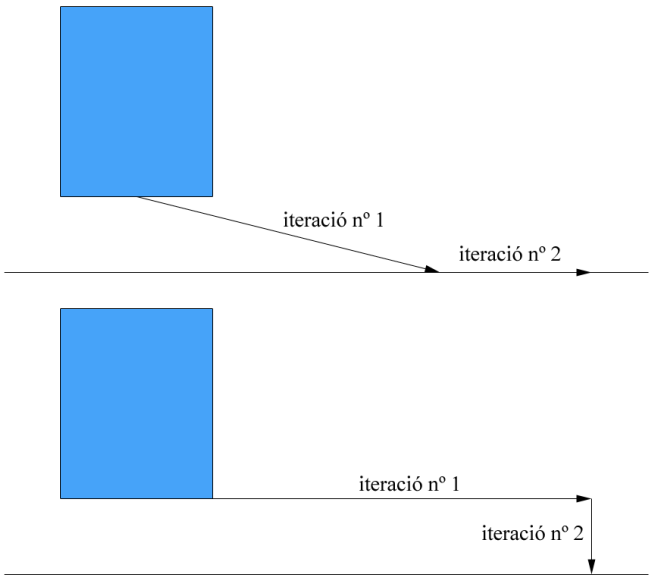


Figura 4.2: Excedent del vector de velocitat reflectit sobre el pla de col·lisió entre una caixa alineada al món i un polígon.

5. Gravetat

Aplicar la força de la gravetat forma part del sistema de dinàmica de l'aplicació; però per simulacions senzilles, es pot simular la gravetat aplicant-la directament al vector de velocitat de l'objecte. Així, al vector de velocitat que es passa junt amb l'objecte al sistema de tractament de col·lisions ara se li suma la velocitat proporcionada per la gravetat. Però ara pot ser que un objecte que abans pugés certs obstacles gràcies a la reacció per lliscament, ja no sigui capaç de pujar-los degut a haver afegit al vector de velocitat una component en sentit cap a baix. Això potser no sigui el desitjat. En aquest cas hi ha la possibilitat de passar l'objecte dues vegades pel sistema de tractament de col·lisions: una amb el vector de velocitat original i una segona amb només el vector de gravetat. Això farà que la gravetat s'apliqui després de que l'objecte hagi lliscat degut al seu vector de velocitat. Podria semblar que això representa un doble cost en el tractament de col·lisions, però fixant-nos bé en com són en general les escenes es pot veure que normalment representarà el mateix cost. Les figures 5.1a i 5.1b mostren un exemple d'aplicar la gravetat junt o separada del vector de velocitat. En ambdues figures es pot veure com ha calgut passar dues vegades pel sistema de tractament de col·lisions. En la figura 5.1a s'ha detectat una col·lisió que comporta haver de tornar a passar pel tractament de col·lisions per l'interval posterior a la col·lisió; mentre que en la figura 5.1b no es detecta cap col·lisió pel vector de velocitat i si per la segona passada amb el vector de la gravetat, que retorna un vector de lliscament nul i, per tant, es pot donar per finalitzat el tractament de col·lisions.



De dalt a baix, figures 5.1a i 5.1b: Els dos mètodes per simular la gravetat passen dos cops pel sistema de tractament de col·lisons.

Amb una correcta combinació de la gravetat, la reacció de lliscament i la fricció es pot aconseguir que un objecte romangui quiet en superfícies poc inclinades i rellisqui en d'altres amb pendents més pronunciades.

6. Error de precisió

Els ordinadors treballen amb números finits, cosa que vol dir que els números decimals tenen un màxim de precisió. En molts casos la precisió proporcionada per aquests serà suficient; però sempre hi haurà la possibilitat de que certes operacions no donin el resultat correcte degut a haver realitzat l'operació amb números molt pròxims al límit de la precisió suportada pels ordinadors. A la figura 6.1 es mostra un codi aparentment inofensiu. La comprovació de que a i b siguin més grans de 0 implica que c no serà mai 0 i, per tant, mai hi haurà divisió per 0. Això, però, no és del tot cert en un ordinador. Suposem que a = 0.012 i b = 0.025 i que el màxim de precisió d'un ordinador és 0.001. Això dóna per a * b un resultat de 0.0003; però com que l'ordinador només pot guardar fins a 0.001 el valor es trunca per donar 0.000. En conseqüència c = 0 i les següents operacions treballaran amb un valor erroni, que en aquest cas generarà una divisió per zero.

```
1  si ( a > 0 i b > 0 ) {
2       c = a · b
3       d = 1 / c
4  }
```

Figura 6.1: Codi que no funciona correctament amb valors molt petits

Evidentment l'exemple anterior és molt exagerat i els ordinadors disposen d'una major precisió; però el cas sempre es pot donar per bona que sigui la precisió. Una manera de solucionar això és troba en modificar les comparacions de la línia 1, de manera que no es comprovi si és major que 0, sinó si és major que un valor èpsilon molt proper al zero, de manera que l'operació de multiplicació no pugui donar mai un 0 degut a una falta de precisió.

7. Exemple

L'exemple descrit en aquesta secció és el sistema de tractament de col·lisions que utilitza una aplicació real. L'aplicació en si és una versió beta d'un motor 3D d'un joc (concretament el motor **OpenDoor**), el qual de moment només consta d'un món estàtic de polígons i un personatge que explora aquest món.

El sistema de visualització del motor 3D es basa en el *portal rendering*, el qual, a grans trets, organitza la geometria del món en zones o habitacions; per tant, això farà que només sigui necessari buscar interseccions entre el personatge i els polígons de les habitacions per les que passi, que en la majoria dels casos serà només una. Pel que fa al bounding per aproximar l'objecte, es va tenir en compte que el joc seria d'acció, cosa que fa que siguin més importants una detecció ràpida de col·lisions que no pas una detecció molt precisa. Això descarta les caixes orientades, i, degut a la seva propietat de ser rotacionalment invariables, es va decidir per les esferes en front de les caixes alineades al món. El bounding de moviment triat va ser la caixa alineada al món, doncs el joc es va pensar perquè normalment no hi hagués una gran acumulació de personatges i objectes en un mateix lloc, de manera que amb una caixa alineada al món normalment es descartarà la mateixa quantitat de possibles parelles d'objectes en col·lisió que una caixa orientada. Resumint, en la versió actual del motor (i també d'aquest exemple), es té:

- El personatge s'aproxima amb una esfera.
- S'utilitza una caixa alineada al món com a bounding de moviment.
- La geometria s'organitza en habitacions.

7.1. Algoritme de tractament de col·lisions

Anem ja a aplicar la teoria explicada als apartats anteriors. A la figura 7.1 s'ensenya l'algoritme que segueix el sistema de tractament de col·lisions del motor 3D i com el joc el crida.

```
 1  Joc()
 2  {
 3      per ( t = t0 fins a ∞ en increments de trep ) {
 4          obté dades dels dispositius d'entrada
 5          actualitza el comportament del personatge per t
 6          velocitat = velocitat del personatge per t
 7          tracta( personatge, velocitat )
 8          tracta( personatge, gravetat )
 9          representa el món per t
10      }
11  }
12
13  tracta( objecte, velocitat )
14  {
15      fes {
16          infoCol = detecta( objecte, velocitat )
17          actualitza la posició de objecte segons infoCol
```

```
18          si ( infoCol conté col·lisions )
19              velocitat = respon( infoCol, objecte, velocitat )
20      } mentre ( infoCol contingui col·lisions )
21  }
```

Figura 7.1: Algoritme de tractament de col·lisions

Actualment l'algoritme tracta les col·lisions entre un objecte (el personatge en aquest cas) i els polígons del món. A més de l'objecte se li passa com a paràmetre addicional la velocitat que tindrà aquest durant el seu tractament, de manera que es pugui simular la gravetat amb el mètode de dues passades (línies 6 a 8). El primer que fa l'algoritme és cridar al primer component del tractament de col·lisions, l'algoritme de detecció, guardant a *infoCol* la informació útil que generi (línia 16). Tot seguit, actualitza la posició de l'objecte bellugant-lo la distància lliure de col·lisions que indica *infoCol* (línia 17). Si s'ha produït alguna col·lisió es crida al segon component del tractament de col·lisions, l'algoritme de reacció, del qual s'obté el nou vector de velocitat que s'aplicarà a la següent iteració del tractament de col·lisions (línies 18 i 19). El procés de tractament de col·lisions s'anirà repetint mentre es produeixi alguna col·lisió (línia 20).

7.2. Algoritme de detecció

La figura 7.2 presenta l'algoritme de detecció.

```
1  detecta( objecte, velocitat )
2  {
3      boundObj = bounding de objecte
4      boundMov = genera_bound_mov( boundObj, velocitat )
5      polsPot = obté_pols_potencials( boundObj, boundMov )
6      cols = troba_col·lisions( boundObj, velocitat, polsPot )
7      infoCol = primera_col·lisió( cols )
8      retorna ( infoCol )
9  }
```

Figura 7.2: Algoritme de detecció

El primer que es fa en l'algoritme és obtenir l'esfera amb què s'aproxima l'objecte (línia 3). Amb aquesta esfera, i no amb l'objecte en si, és amb la que es realitzaran totes les operacions, fent-se evident així la diferència entre els móns del sistema de col·lisions i del de visualització. El primer pas de la detecció de col·lisions és generar el bounding de moviment a partir de l'esfera i la velocitat associada a l'objecte (línia 4). Amb el bounding de moviment s'obtenen els polígons amb què potencialment pot col·lisionar l'esfera (línia 5). D'aquests polígons es troben aquells amb què l'esfera realment col·lisiona (línia 6). D'aquestes col·lisions es mira quina és la primera que es produirà (línia 7). Finalment es retorna la informació d'aquesta col·lisió (línia 8).

7.2.1. Generació del bounding de moviment

La figura 7.3 mostra l'algoritme que genera el bounding de moviment.

```
1  genera_bound_mov( esfera, velocitat )
2  {
3      origen = posició de esfera
4      destí = origen + velocitat
5      radi = radi de esfera
6      caixa.x_mín = mínim( origen.x, destí.x ) – radi
7      caixa.x_màx = màxim( origen.x, destí.x ) + radi
8      caixa.y_mín = mínim( origen.y, destí.y ) – radi
9      caixa.y_màx = màxim( origen.y, destí.y ) + radi
10     caixa.z_mín = mínim( origen.z, destí.z ) – radi
11     caixa.z_màx = màxim( origen.z, destí.z ) + radi
12 }
```

Figura 7.3: Algoritme de generació del bounding de moviment

Per construir una caixa alineada al món cal conèixer els sis plans que la formen. Per determinar aquests plans, cal conèixer el punts d'origen i destí i el radi de l'esfera (línies 3 a 5). Després es comparen les components x, y i z dels dos punts, i els plans surten de restar a les components de valor més petit el radi, i de sumar-lo a les de valor major (línies 6 a 11). La figura 7.4 mostra gràficament la construcció d'un bounding de moviment. Es pot veure com per trobar els plans que formen la caixa alineada al món només cal sumar als punts d'origen i destí de l'esfera el valor del radi. Aquesta suma es realitza a la component x, y o z que correspongui per cada pla, i el valor sumat serà de signe positiu o negatiu segons convingui.

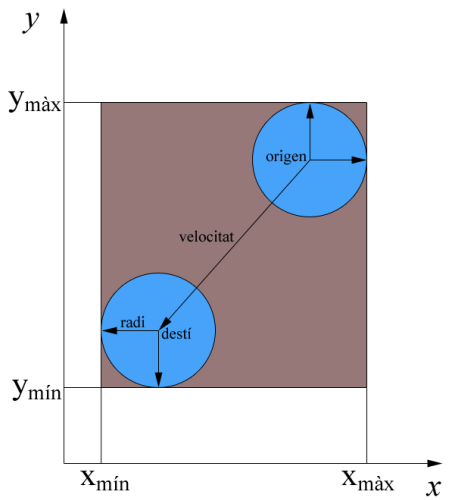


Figura 7.4: Construcció d'un bounding de moviment

7.2.2. Obtenció dels polígons potencials

La figura 7.5 presenta l'algoritme encarregat d'obtenir els polígons potencials. Aquests són tots aquells polígons que intersseccionen amb el bounding de moviment; és

a dir, aquells pels que farà falta utilitzar un algoritme d'intersecció més acurat per esbrinar si realment l'esfera hi col·lisiona.

```
1  obté_pols_potencials( esfera, boundMov )
2  {
3      habsOcupades = habitacions que conté el món i que
       intersseccionen amb boundMov
4      per ( cada habitació Hi de habsOcupades ) {
5          per ( cada polígon Pj dels polígons que conté Hi ) {
6              boundPol = bounding de Pj
7              si ( boundMov i boundPol intersseccionen )
8                  si ( boundMov i Pj interserccionen ) )
9                      afegeix Pj a polsPot
10         }
11     }
12     retorna ( polsPot )
13 }
```

Figura 7.5: Algoritme d'obtenció de polígons potencials

L'algoritme aprofita l'organització en habitacions que estableix el motor en ser basat aquest en el *portal rendering*. Així, el primer que es fa és determinar quines habitacions intersseccionen amb el bounding de moviment (línia 3), de manera que ja es descarten tots els polígons de les altres com a polígons amb què l'esfera pot col·lisionar. El *portal rendering* connecta dues habitacions amb un portal, de manera que perquè un objecte canviï d'habitació ha de travessar forçosament un portal. Per tant, per determinar les habitacions que intersseccionen amb el bounding de moviment només caldrà comprovar quins portals intersseccionen amb aquest. És més, aquesta comprovació només s'haurà de fer amb els portals de l'habitació on es trobi inicialment l'esfera, i, si es troba alguna intersecció, realitzar aquesta operació recursivament amb l'habitació on porti aquell portal. La figura 7.6 mostra un exemple amb l'esfera travessant un portal i, per tant, canviant d'habitació. Es pot veure com això implica que el portal intersecciona amb el bounding de moviment.

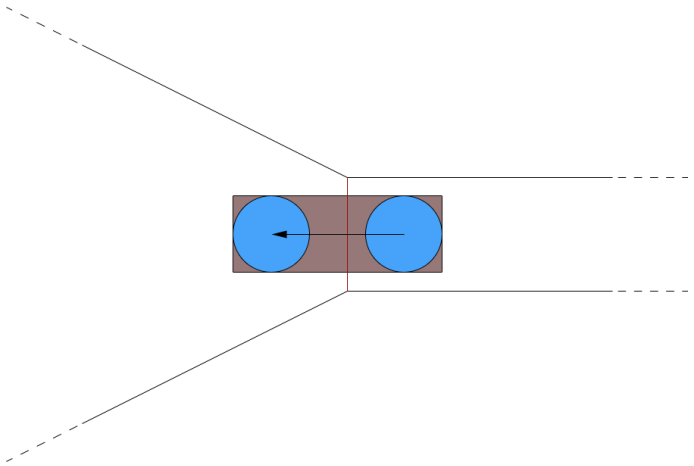


Figura 7.6: Un portal intersecciona amb el bounding de moviment

Comprovar una intersecció entre dues caixes alineades al món és una operació molt ràpida. Tenint en compte això, per accelerar encara més el procés d'obtenció de polígons potencials cada polígon té la seva pròpia caixa alineada al món com a bounding de polígon, de manera que s'obté aquest bounding (línia 6) i es comprova primer la intersecció entre el bounding de moviment i el del polígon (línia 7). A la figura 7.7 es pot veure com un dels boundigs dels tres polígons no intersecciona amb el bounding de moviment i, per tant, ja es pot descartar.

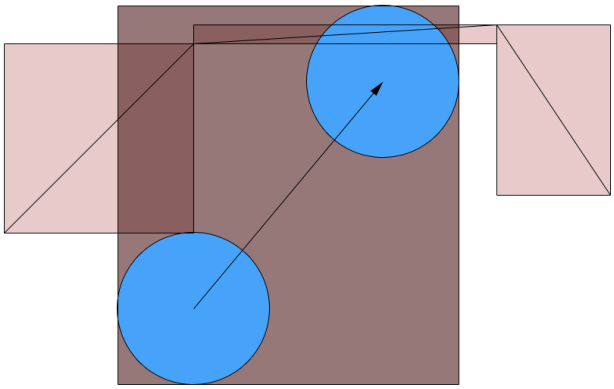


Figura 7.7: Bounding de moviment i polígons amb els seus boundings

Només en el cas que es produeixi una intersecció entre el bounding d'un polígon i el bounding de moviment, es comprova una intersecció entre el polígon i el bounding de moviment (línia 8). Per realitzar aquesta operació s'utilitza un algoritme de clipping de polígons. Si finalment el polígon intersecciona amb el bounding de moviment llavors s'afegeix als polígons potencials (línia 9). Aquest procés es realitza amb tots els polígons de les habitacions ocupades pel bounding de moviment (línies 4 i 5) i, en acabat, es retornen els polígons potencials (línia 12).

7.2.3. Càlcul de les col·lisions

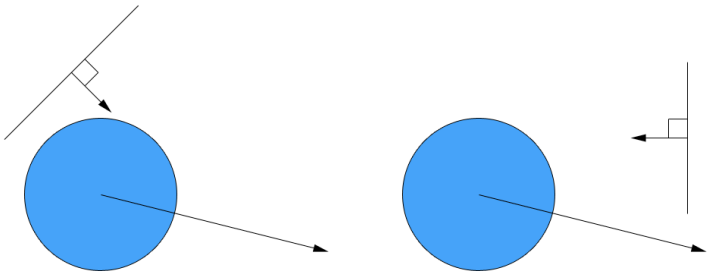
L'algoritme de la figura 7.8 troba totes les col·lisions que es produeixen entre l'esfera d'un objecte i els polígons potencials trobats a l'etapa anterior.

```
1  troba_col·lisions( esfera, velocitat, polígons )
2  {
3      origen = posició de l'esfera
4      vel_u = velocitat normalitzada
5      per ( cada polígon P_i de polígons ) {
6          norm = normal de P_i
7          si ( norm * vel_u < 0 ) {
8              ptCol_e = punt d'intersecció del raig definit per
                   origen i -norm amb esfera
9              ptCol_p = punt d'intersecció del raig definit per
```

```
                    ptCol_e i vel_u amb el pla que defineix P_i
10                  si ( ptCol_p no està dins P_i ) {
11                      ptCol_p = punt de P_i més proper a ptCol_p
12                      ptCol_e = primer punt d'intersecció del raig
                        definit per ptCol_p i -vel_u amb l'esfera
13                      si ( no existeix ptCol_e )
14                          continuar amb la següent iteració
15                  }
16                  distCol = mòdul de ptCol_p – ptCol_e
17                  si ( distCol <= mòdul de velocitat )
18                      afegeix distCol i P_i a col·lisions
19              }
20      }
21      retorna ( col·lisions )
22  }
```

Figura 7.8: Algoritme que troba les col·lisions entre l'esfera de l'objecte i els polígons potencials

L'algoritme realitza l'operació de detecció de col·lisió per tots els polígons potencials (línia 5). Aquesta operació comença comprovant si el producte escalar de la normal i el vector de velocitat dóna una valor positiu o negatiu (línia 7). En cas de ser negatiu voldrà dir que l'esfera s'està allunyant del polígon i, per tant, es pot descartar. Les figures 7.9a i 7.9b mostren els dos possibles casos.



D'esquerra a dreta figura 7.9a i 7.9b: Un polígon del que l'esfera s'allunya i un altre al que s'apropa

Un cop trobat un polígon al que l'esfera s'apropa, es calcula el punt de col·lisió de l'esfera en la seva posició d'origen (línia 8). Movent el pla del polígon fins a tocar l'esfera es pot veure com la distància des del punt de col·lisió fins al centre de l'esfera tindrà la direcció de la normal del polígon i magnitud del valor del radi. Això queda clar gràficament a la figura 7.10.

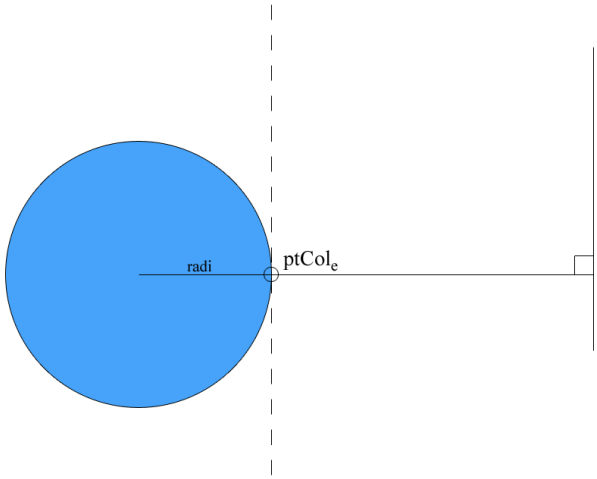


Figura 7.10: Punt de col·lisió de l'esfera

El següent pas és calcular el punt de col·lisió del polígon. Per fer això primer es troba el punt de col·lisió amb el pla del polígon, el qual correspon al punt d'intersecció amb el pla d'un raig del mateix sentit que el vector de velocitat i traçat des del punt de col·lisió de l'esfera (línia 9). Seguint l'exemple de la figura 7.10, a la figura 7.11 es mostra l'operació d'intersecció del raig esmentat amb el pla del polígon.

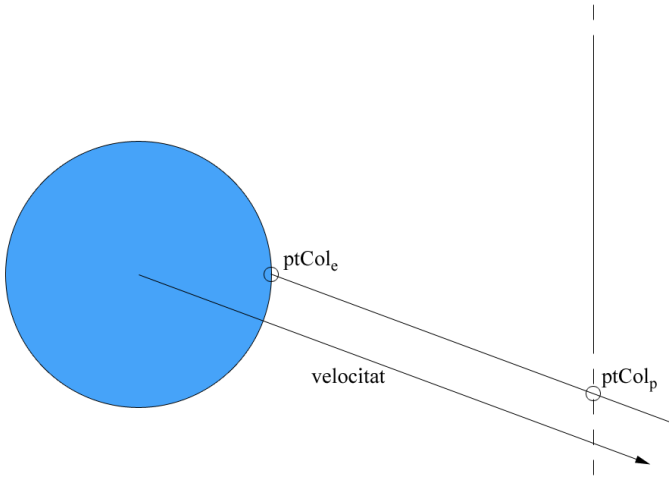


Figura 7.11: Punt de col·lisió amb el pla del polígon

A continuació es comprova si el punt trobat anteriorment ja es troba dins el polígon (línia 10). Si no és així, el punt de col·lisió serà aquell del polígon que més a prop estigui d'aquest punt. Per tant, cal trobar aquest nou punt de col·lisió (línia 11). Gràficament aquest punt és molt evident, tal i com mostra la figura 7.12.

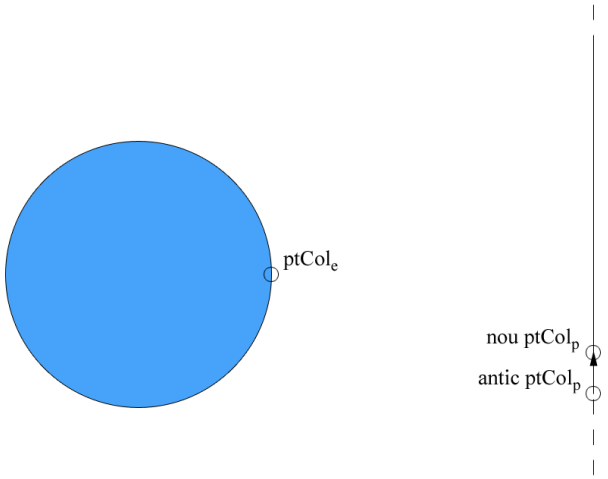
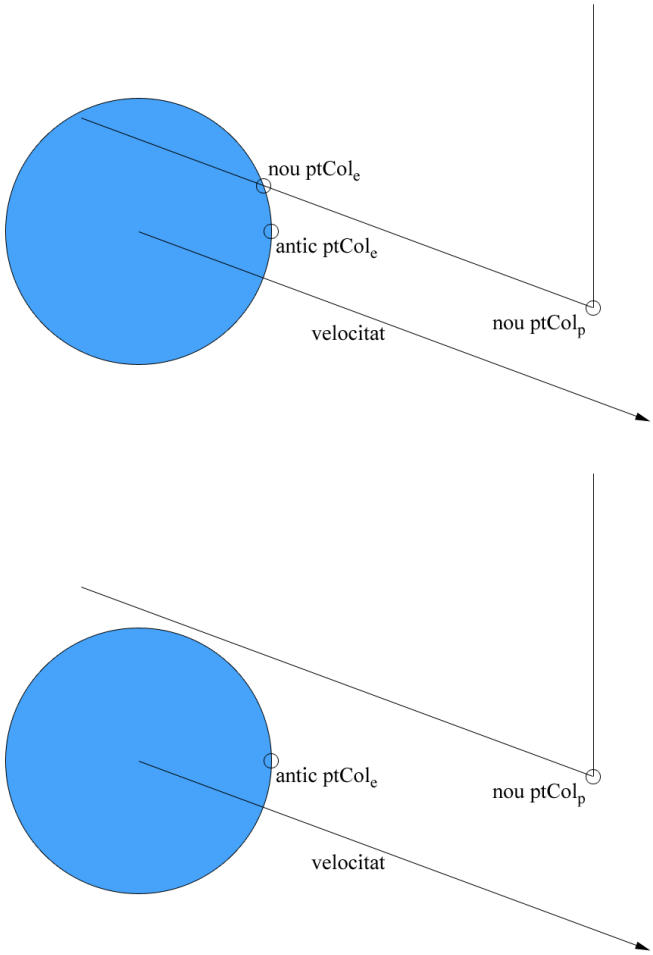


Figura 7.12: El nou punt de col·lisió és aquell del polígon més proper a l'antic punt de col·lisió

Si ha calgut trobat un nou punt de col·lisió pel polígon això ha invalidat el punt de col·lisió de l'esfera i, per tant, també s'haurà de trobar nou punt de col·lisió per l'esfera. Aquest punt correspondrà al primer punt d'intersecció amb l'esfera del raig amb el mateix sentit que el vector de velocitat invertit i traçat des del nou punt de col·lisió del polígon (línia 12). També es pot donar el cas que aquest raig no intersecconi l'esfera, cosa que voldrà dir que no es produeix una col·lisió entre l'esfera i el polígon i, per tant, es prosseguirà amb el següent polígon (línies 13 i 14). Les figures 7.13a i 7.13b mostren els dos casos possibles.



De dalt a baix, figura 7.13a i 7.13b: Un cas on es troba el nou punt de col·lisió i un altre on no es produeix col·lisió

Tot i haver trobat el punt de col·lisió, si l'esfera no s'ha de desplaçar prou distància com per arribar-hi llavors aquesta no es produirà. Per això es calcula la distància entre el punt de col·lisió de l'esfera i el del polígon (línia 16) i, només si aquesta distància és inferior o igual a la distància que vol recórrer l'esfera, la qual ve definida pel mòdul del vector de velocitat, (línia 17) s'afegeix el polígon i la distància de col·lisió a la llista de col·lisions (línia 18).

Un cop finalitzat aquest procés per tots els polígons es retorna la llista de col·lisions.

7.2.4. Elecció de la col·lisió més immediata

De totes les col·lisions trobades a l'apartat anterior, només serà vàlida la primera que es produeixi, doncs aquesta farà que per la resta de l'interval de la detecció de col·lisions s'utilitzi un altre vector de velocitat. La figura 7.14 presenta l'algoritme que troba el punt de col·lisió que es produirà primer d'entre tots els que conté la llista de col·lisions.

```
1  primera_col·lisió( col·lisions )
2  {
3      si ( col·lisions no conté cap col·lisió )
4          retorna ( ∅ )
5      dist_mín = ∞
6      per ( cada col·lisió C_i de col·lisions ) {
7          dist = distància de col·lisió de C_i
8          si ( dist < dist_mín ) {
9              dist_mín = dist
10             pol = polígon de C_i
11         }
12     }
13     afegeix dist_mín i pol a infoCol
14     retorna ( infoCol )
15 }
```

Figura 7.14: Algoritme que troba la primera col·lisió en produir-se

Primer de tot, es comprova que realment la llista de col·lisions contingui alguna col·lisió (línia 3). Si en conté es recorren totes les col·lisions (línia 6) i per cadascuna es compara si la seva distància de col·lisió és inferior a la més curta trobada fins al moment (línia 7). Si és així, es fixa la nova distància com la més curta i se'n guarda el polígon com a informació útil de col·lisió (línies 9 i 10). Un cop acabat, s'afegeix a la informació de col·lisió el polígon que hagi esdevingut el primer amb què l'esfera xocarà junt amb la distància de col·lisió (línia 13) i es retorna aquesta informació (línia 14).

7.3. Algoritme de reacció

Quan s'ha detectat una col·lisió pel personatge interessa que aquest no es quedi aturat incondicionalment, sinó que es vol que llisqui pel polígon amb què ha xocat. A més, també és desitjable que aquest lliscament sigui més o menys accentuat en funció de l'angle de col·lisió amb el polígon i la fricció que se li hagi associat per simular la textura física d'aquest. Al final, aquesta reacció permetrà que el personatge pugui continuar avançant en arrambar-se a una paret i també pugui pujar automàticament per pendents petites i obstacles baixos, com ara escales.

Tot el que cal per obtenir aquesta reacció és calcular el vector de lliscament i tornar a executar el procés de tractament de col·lisions, ara utilitzant el vector de lliscament com a vector de velocitat. La figura 7.15 mostra l'algoritme de reacció, el qual actualment tot el que fa és calcular un vector de lliscament i retornar-lo.

```
1  respon( infoCol, objecte, velocitat )
2  {
3      esfera = bounding de objecte
4      pol = polígon que conté infoCol
5      velExcedent = tros del vector velocitat després del punt de
       col·lisió
6      plaCol = pla tangent a esfera en el punt de col·lisió
7      lliscament = projecció de velExcedent sobre plaCol
```

```
 8      lliscament = lliscament escurçat pel factor de fricció de pol
 9      retorna ( lliscament )
10  }
```

Figura 7.15: Algoritme de reacció

El vector de lliscament es calcularà a partir de la part del vector de velocitat que quedi després del punt de col·lisió; és a dir, aquella part que no s'ha tingut en compte a l'hora d'actualitzar la posició de l'objecte en haver detectat una col·lisió (línia 5). Aquest tros de vector es projecta sobre el pla de col·lisió, el qual és aquell pla tangent a l'esfera en el punt de col·lisió, per obtenir un vector de lliscament sense fricció (línies 6 i 7). Per aplica la fricció només cal escurçar el vector de lliscament per un factor definit pel polígon amb què s'ha col·lisionat (línia 8). Finalment, es retorna aquest vector de lliscament (línia 9). La figura 7.16 mostra la projecció sobre el pla de col·lisió del tros excedent del vector de velocitat, i també com s'escurça pel factor de fricció.

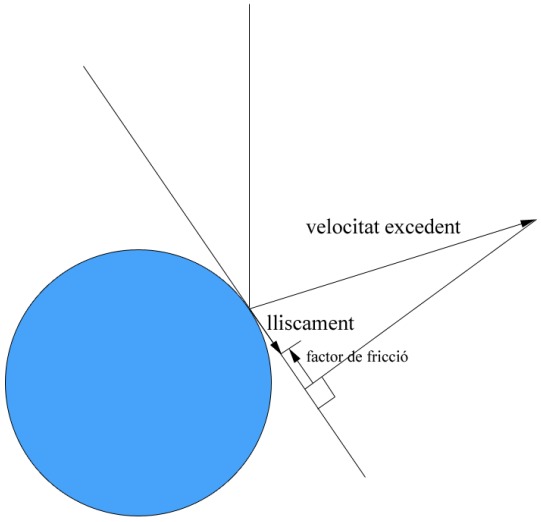


Figura 7.16: Vector de lliscament a partir de l'excedent de velocitat

8. Bibliografia

Aquest article està molt influenciat pels dos articles següents:

- *Collision Detection for Interactive Graphics Applications* (Philip M. Hubbard – 1995), on s'expliquen els tres problemes fonamentals que ha de resoldre tot algoritme de detecció. L'article presenta una solució utilitzant arbres d'esferes amb boundings espacio-temporals.
  Aquest article i altres del mateix autor es poden trobar a la web de l'autor: http://www.cs.wustl.edu/~pmh.
- *Generic Collision Detection for Games Using Ellipsoids* (Paul Nettle – 2000), on s'expliquen les operacions que cal realitzar pel tractament de col·lisions entre una el·lipse i un món de polígons.
  Aquest article es pot trobar a la secció de publicacions de la web http://www.fluidstudios.com.

Articles sobre boundings d'objecte:

- *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection* (S. Gottschalk, M. C. Lin i D. Manocha – 1996), que es pot trobar a http://www.cs.unc.edu/~dm/collision.html.
- *The Quickhull Algorithm for Convex Hulls* (C. Bradford Barber, David P. Dobkin, Hannu Huhdanpaa – 1996), es pot trobar a http://www.cs.princeton.edu/~dpd/Papers.html.

Articles sobre dinàmica:

- Brian Mirtich té a la seva web diversos articles sobre l'impuls: http://www.cs.berkeley.edu/~mirtich/
- http://www.cfxweb.net/civax, on es pot trobar demos i codi sobre la simulació de roba.
- http://www.cs.unc.edu/~hirota/290/papers.html, conté links a articles sobre la dinàmica aplicada a objectes rígids i no rígids.

Web amb codi font:

- http://www.magic-software.com, l'apartat de codi gratuït conté codi de múltiples tests d'intersecció.

Webs relacionades amb el desenvolupament de jocs:

- http://www.gamasutra.com
- http://www.gamedev.net
- http://www.flipcode.com
- http://www.stratos-ad.com
- http://www.game-developer.com

Web amb articles sobre el motor **OpenDoor**:

- http://www.macedoniamagazine.com/