Institut Universitaire de Technologie
Aix*Marseille Université

Imagerie Numérique
M4102Cin - Synthèse d'images 2

6. Collisions

DUT Informatique 2018-2019

Sébastien THON

IUT d'Aix-Marseille Université, site d'Arles
Département Informatique

Collisions

Le test de collisions entre des objets dans une scène 3D est indispensable dans une application 3D interactive (simulateur, jeu, …) :

- Est-ce que l'avion a touché le sol ?
- Est-ce que le personnage est contre le mur du couloir ?
- Est-ce que la raquette a touché la balle ?
- Est-ce que le boulet de canon a touché le bateau ?
- …

1. Principe

Dans le cadre d’animation d’objets, on est confronté au problème des collisions potentielles entre ces objets. Il faut les gérer de la manière la plus précise possible, car le gameplay d'un jeu peut reposer dessus (bowling, billard, tennis, jeu de tir, etc.)

Détection de collision

Déterminer **quand** et **où** deux objets se touchent.

Résolution de collision

Déterminer comment la collision affecte le mouvement des objets.

http://media.steampowered.com/apps/valve/2015/DirkGregorius_Contacts.pdf

2. Détection de collision

1er problème :

Pour détecter si deux objets sont entrés en collision, il faudrait vérifier si il y a intersection entre leurs volumes délimités par leurs enveloppes polygonales.

→ *Très lourd en temps de calcul, d'autant plus que le nombre de triangles est important.*

60000 triangles

5000 triangles

2ème problème :

Il y a en général dans une scène plusieurs objets 3D, mobiles ou pas (éléments de décor par exemple).
Dans une implémentation naïve, on testerait l'intersection de chaque objet avec tous les autres → complexité en $O(n^2)$.

Accélération de la détection de collision

Il est nécessaire de mettre en œuvre des stratégies d'accélération de la détection de collision.

On divise la détection de collisions en deux phases :

- **Broad phase**, ou recherche de proximité : on détermine rapidement une liste des paires d'objets pouvant s'intersecter en effectuant un simple test de voisinage. Les paires qui passent cette étape passent à l'étape suivante.

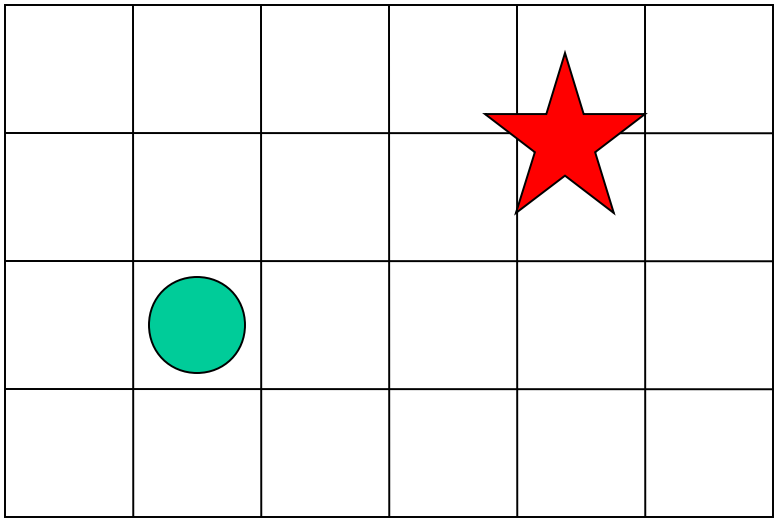
- **Narrow phase**, ou test de collision de proximité : on utilise des algorithmes plus fins pour détecter plus précisement les intersections.

2.1 Broad phase

On va lors de cette phase détecter rapidement des paires d'objets qui sont dans un même voisinage, on verra dans la seconde phase si ils s'intersectent.

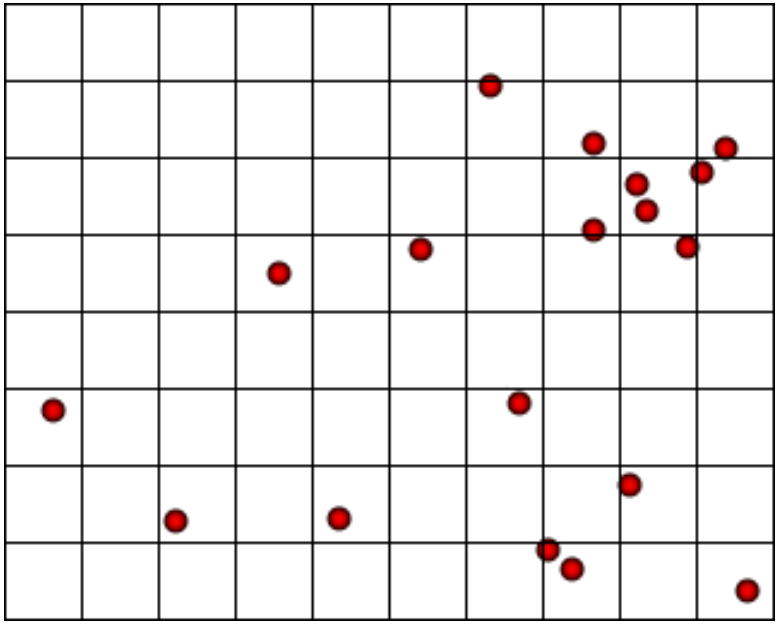
Pour cela, on divise l’espace de la scène 3D en régions.

→ Si 2 objets appartiennent à des régions différentes, alors ils ne sont pas en collision.



1) Partition uniforme de l'espace

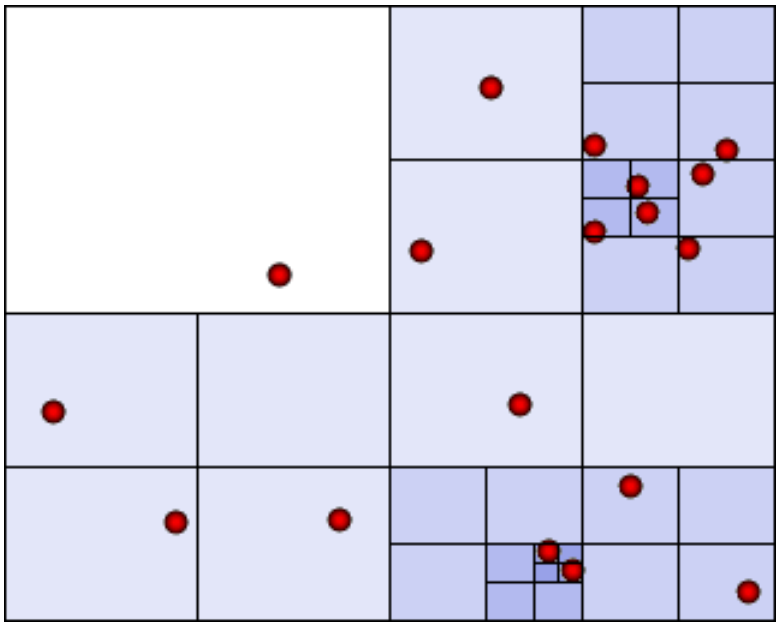
On divise l'espace de la scène 3D en régions de mêmes tailles. Lorsque des objets sont en mouvement, on met à jour la grille.

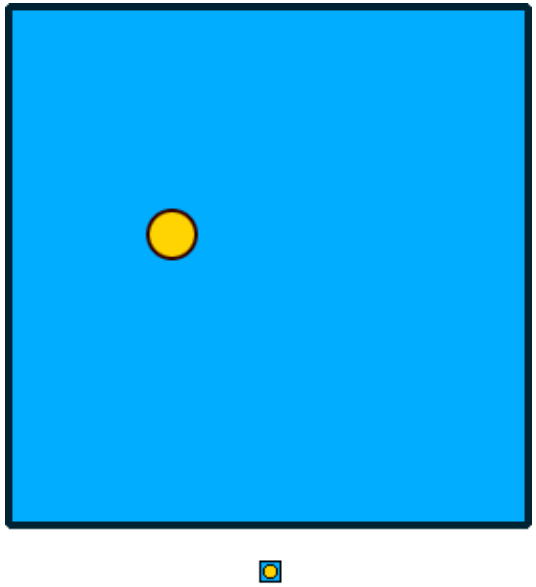


→ Problème : certaines régions peuvent contenir des milliers de polygones, et d'autres très peu voire aucun.

2) Partition adaptative de l'espace

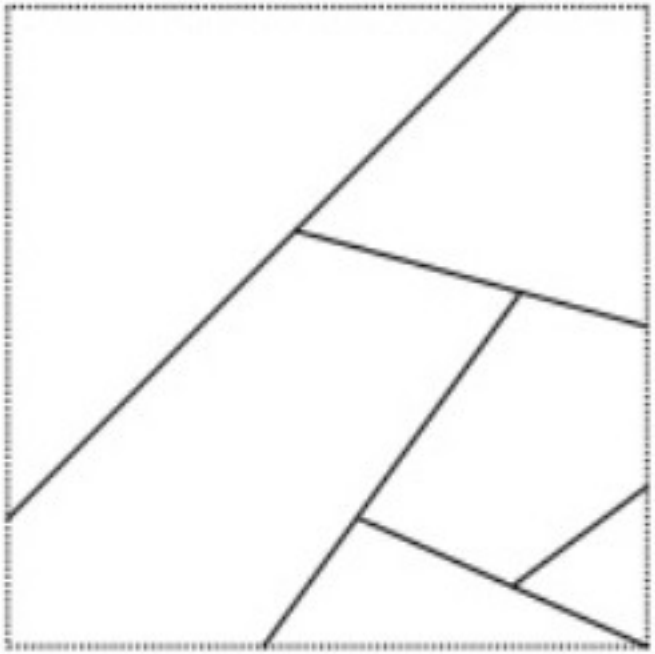
On divise l'espace de la scène 3D de manière adaptative (quadtree, octree) en régions de telles manière à ce qu'elles ne contiennent pas plus d'objets qu'un nombre donné (ou bien qu'un niveau maximal de profondeur de l'arbre est atteint).
Lorsque des objets sont en mouvement, on met à jour la grille.

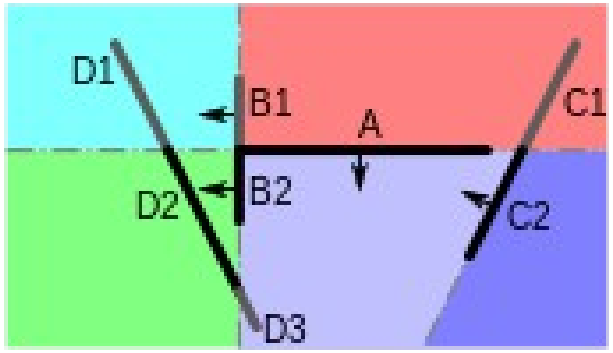


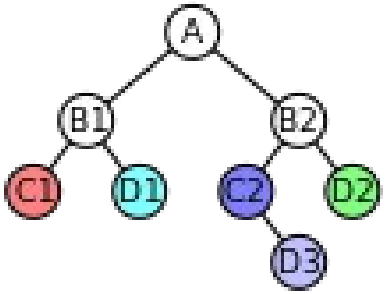


3) Arbres BSP (Binary Space Partitioning)

Arbre binaire de plans pour partitionner l'espace.

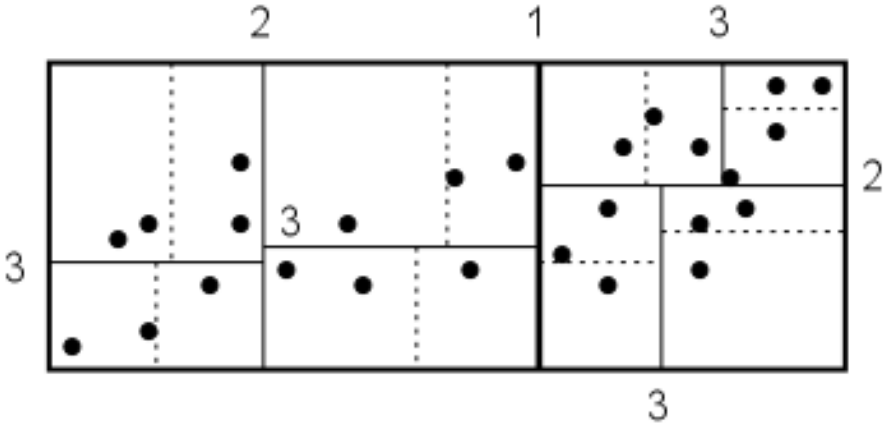






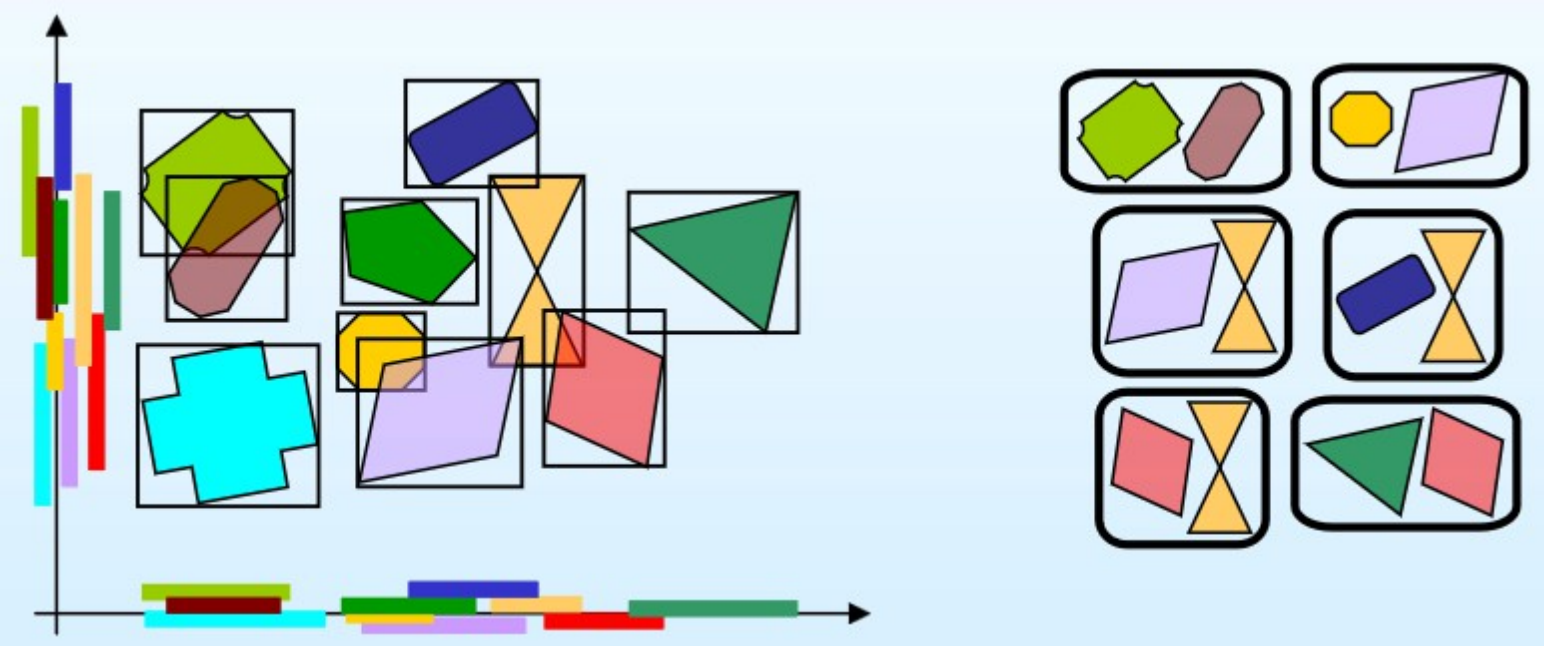
4) k-d tree (k-dimensional tree)

Subdivision hierarchique de l'espace obtenue en divisant l'espace selon un axe après l'autre.
Les k-d tree sont un cas particulier des BSP trees, avec des plans alignés par rapport aux axes au lieu d'avoir des orientations quelconques.
Les points devant un plan sont dans la branche de gauche, les points derrière sont dans la branche de droite.



5) Classement topologique (Sweep and prune)

On projette les boîte englobantes des objets sur les trois axes Ox, Oy et Oz. On obtient des intervalles de projection [debut-fin]. On retient les couples d'objets dont les projections se recouvrent sur les 3 axes.



Dans cet exemple, l'algorithme Sweep and prune retient 6 couples d'objets qui sont potentiellement en collision.

Bilan de la broad phase

Dans toutes les méthodes de subdivision de l'espace que l'on a vu, il faut mettre à jour ces structures lorsque des objets bougent.

- Le recalcul d'un quadtree, d'un octree, d'un k-d tree ou d'un BSP Tree peut être coûteux en temps de calcul. On utilisera plutôt cette subdivision pour stocker des éléments statiques de la scène (ex : le décor).

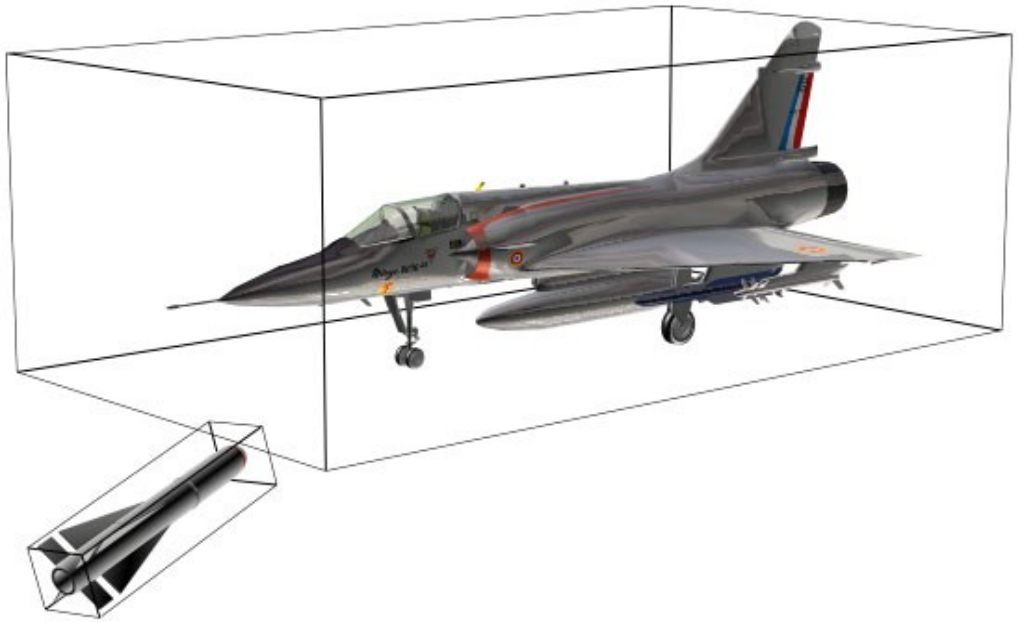
- Les grilles régulières et l'algorithme de Sweep and prune sont plus efficaces pour les objets dynamiques.

On vient d'obtenir une liste de paire d'objets potentiellement en collision, il faut ensuite effectuer des tests plus précis → **narrow phase**.

2.2 Narrow phase

Après la **broad phase**, on teste dans la **narrow phase** les intersections des objets pris 2 à 2, mais ce calcul peut être très coûteux en raison du nombre de polygones qui peut être très élevé.

Pour accélérer le test de collision entre 2 objets, on calcule l'intersection entre leurs **volumes englobants** (sphère, boîte, objet simplifié, etc.)
→ *Beaucoup plus rapide.*

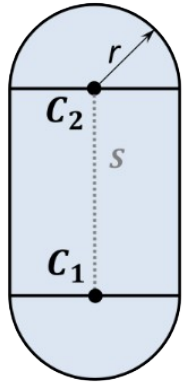


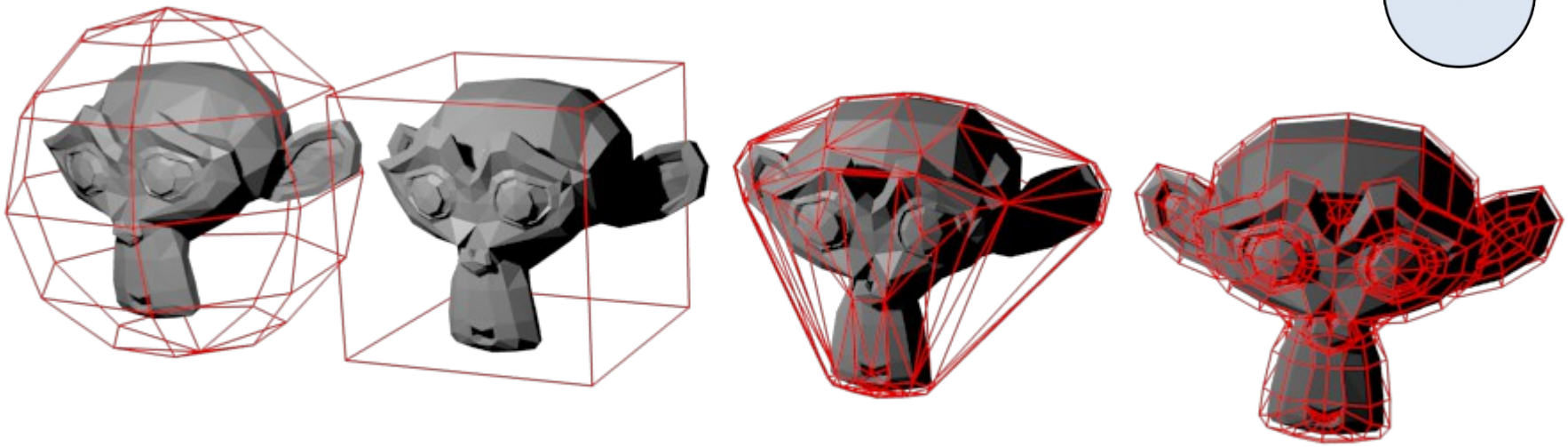
Test de collision de deux objets 3D, en testant l'intersection entre leurs boîtes englobantes

Volumes englobants (« *bounding volume* »)

Il existe de nombreux type de volumes englobants :

sphère, boite, cylindre, ellipsoïde, capsule, enveloppe convexe, enveloppe concave, ...





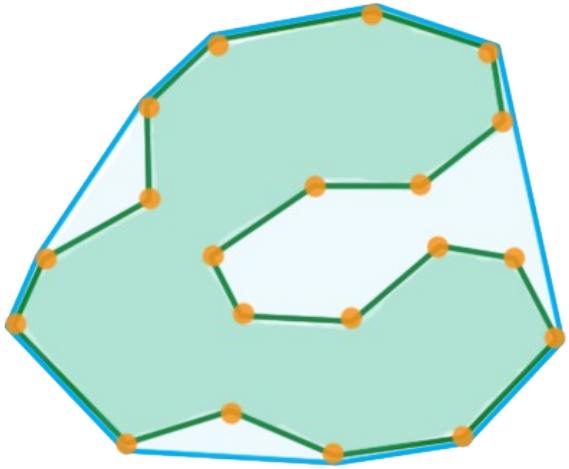
Quel volume englobant utiliser ?

Critère de forme : utiliser le volume englobant qui approche le mieux la forme de l'objet (si l'objet est globalement cubique, utiliser une boîte)

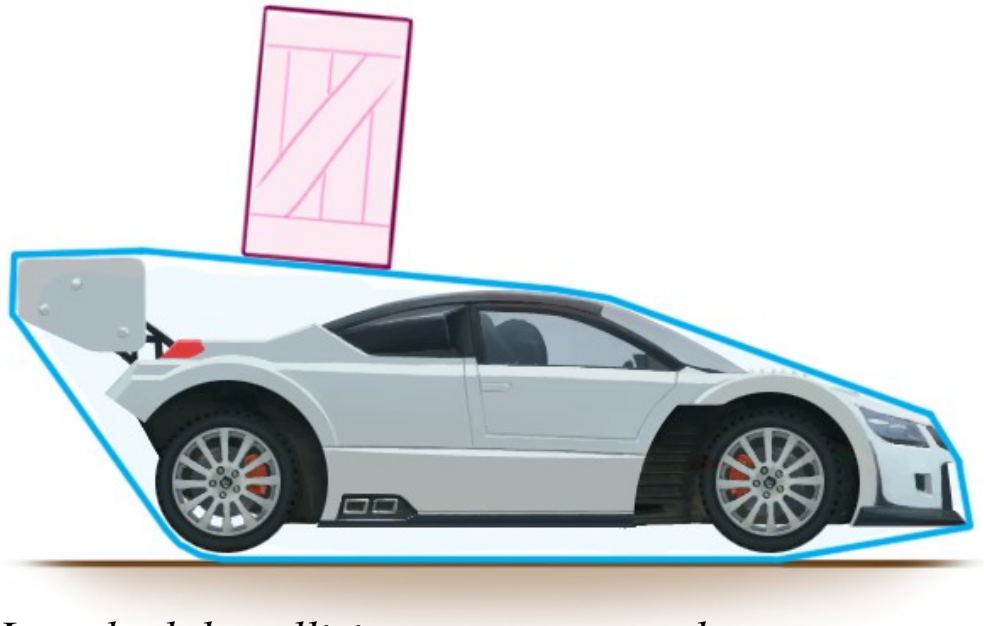
Critère de vitesse : des volumes englobant complexes nécessiteront plus de temps pour calculer leur intersection.

Pour une voiture, on utilisera une boîte englobante plutôt qu'une sphère, pour que la voiture repose bien sur le sol.

Le calcul d'intersection entre volumes concaves est très coûteux, au pire on préfèrera utiliser l'enveloppe convexe. Le calcul sera plus rapide mais la collision plus imprécise.



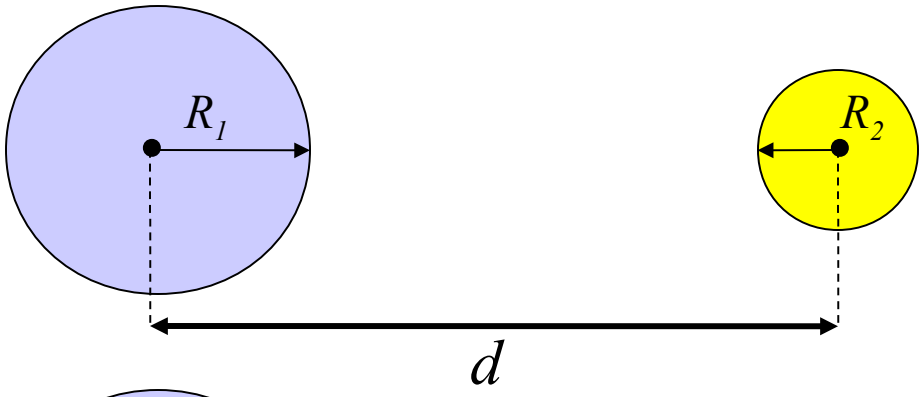
Calcul de l'enveloppe convexe, par exemple avec l'algorithme Quickhull
https://en.wikipedia.org/wiki/Quickhull



Le calcul de collision avec une enveloppe convexe est moins précis qu'avec une enveloppe concave

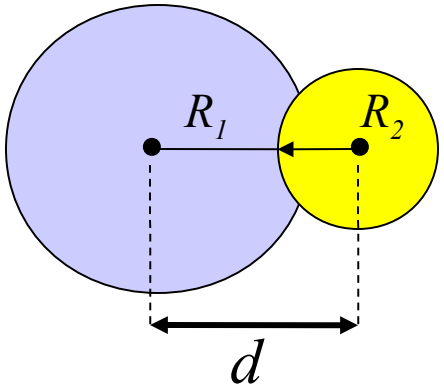
1) Intersection entre sphères englobantes

Test de collision très simple :



$d > R_1 + R_2$

→ Pas de collision

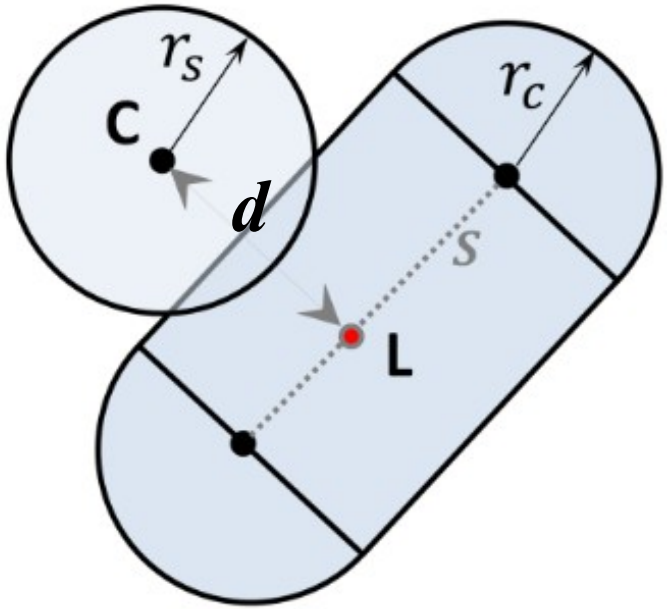


$d < R_1 + R_2$

→ Collision

Deux sphères entrent en collision si la distance qui sépare leurs centres est inférieure à la somme de leurs rayons.

2) Intersection entre sphère englobante et capsule

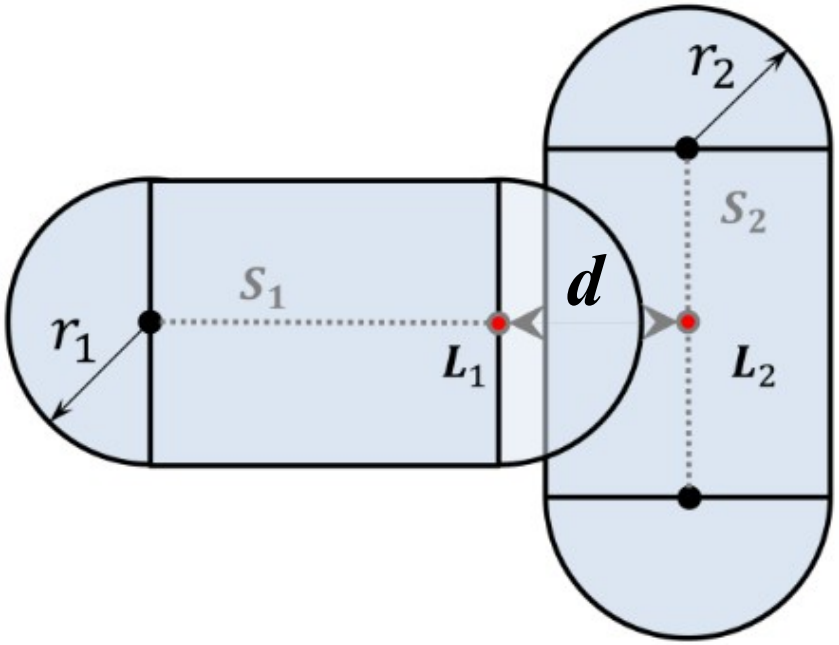


$d < r_S + r_C$

→ Collision

Il y a intersection entre une sphère et une capsule si la distance entre le centre de la sphère et le plus proche point sur le segment interne de la capsule est inférieure à la somme de leurs rayons.

3) Intersection entre deux capsules



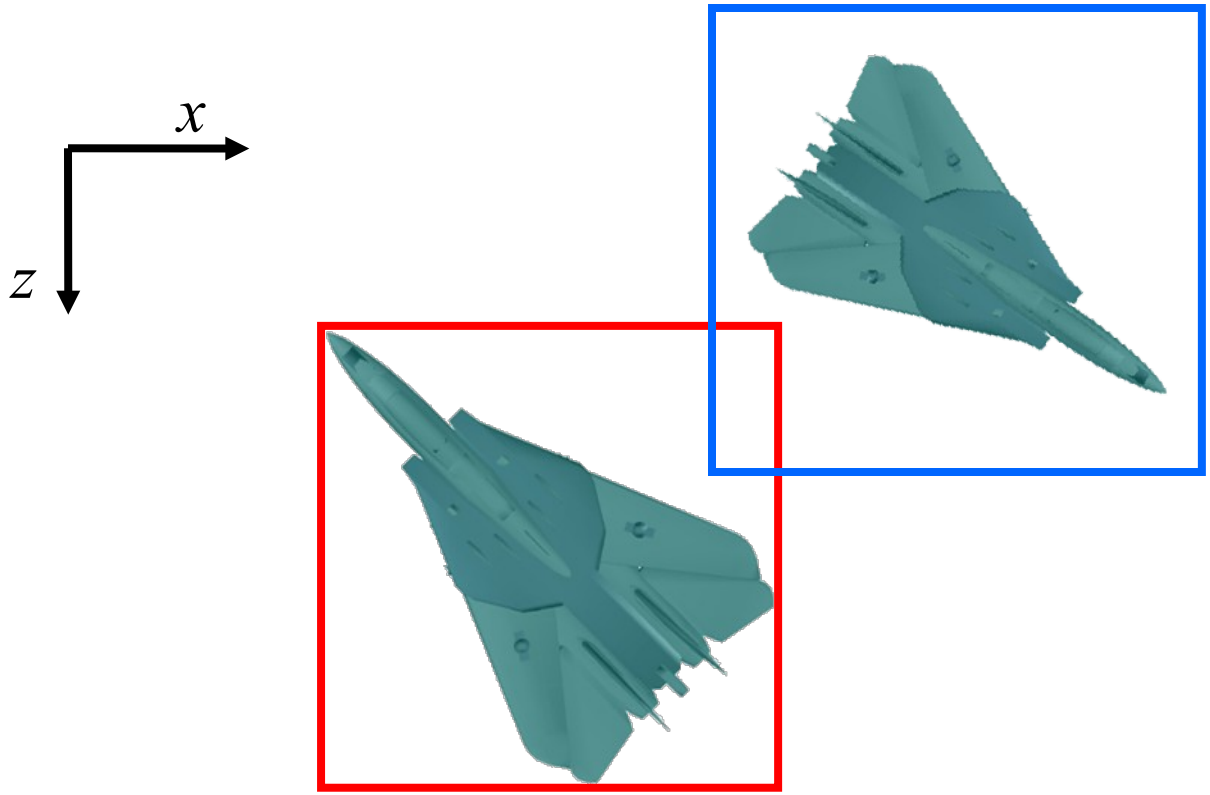
$d < r_1 + r_2$

→ Collision

Il y a intersection entre deux capsules si la distance leurs segments internes est inférieure à la somme de leurs rayons.

4) Intersection entre boîtes alignées par rapport aux axes

(**AABB** : *Axis Aligned Bounding Box*)

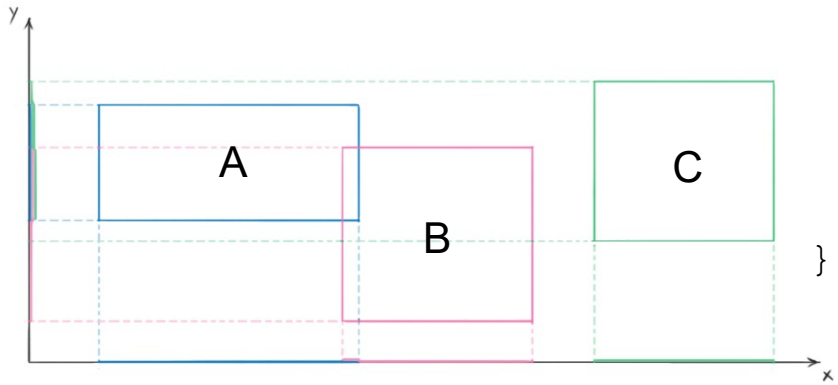


Des collisions peuvent être détectées à tort.

Pour détecter si il y a intersection entre deux boîtes AABB, on calcule la différence entre leurs min et leur max selon les deux axes, dans les deux ordres. Si l'une de ces valeurs est > 0, alors les deux boîtes n'ont pas d'intersection.

```
typedef struct {
    float x;
    float y;
} Point2D;

typedef struct {
    Point2D min;
    Point2D max;
} AABB;
```

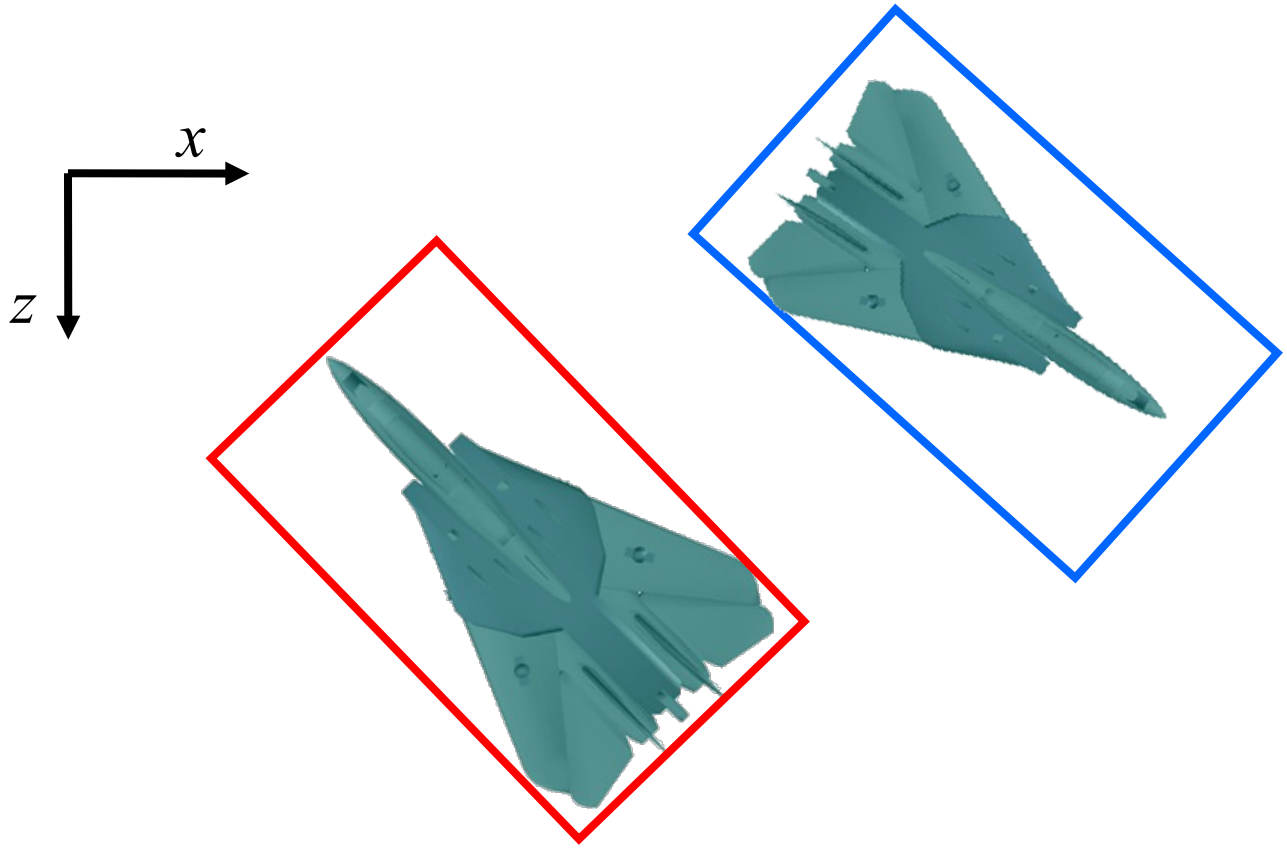
```
bool TestIntersection (AABB* a, AABB* b)
{
    float d1x = b->min.x - a->max.x;
    float d1y = b->min.y - a->max.y;
    float d2x = a->min.x - b->max.x;
    float d2y = a->min.y - b->max.y;

    if (d1x > 0.0f || d1y > 0.0f)
        return false;

    if (d2x > 0.0f || d2y > 0.0f)
        return false;

    return true;
}
```



Il n'y a pas intersection entre A et C ou B et C, mais il y a intersection entre A et B.

5) Intersection entre boîtes à l'orientation quelconque

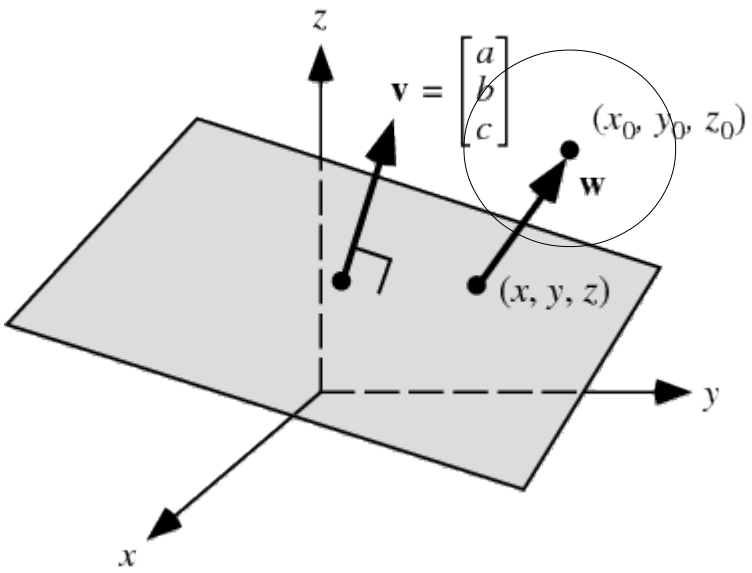
(**OBB** : *Oriented Bounding Box*)



Plus précis que AABB, mais plus coûteux à calculer.

6) Collision entre une sphère et un plan

Soit un plan d'équation : $a\,x + b\,y + c\,z + d = 0$

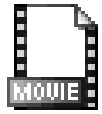


La distance *D* entre une sphère de centre (x_0, y_0, z_0) et le plan est donnée par :

$$D = \frac{a\,x_0 + b\,y_0 + c\,z_0 + d}{\sqrt{a^2 + b^2 + c^2}},$$

Si |*D*| est inférieur au rayon de la sphère, alors il y a intersection entre la sphère et le plan.

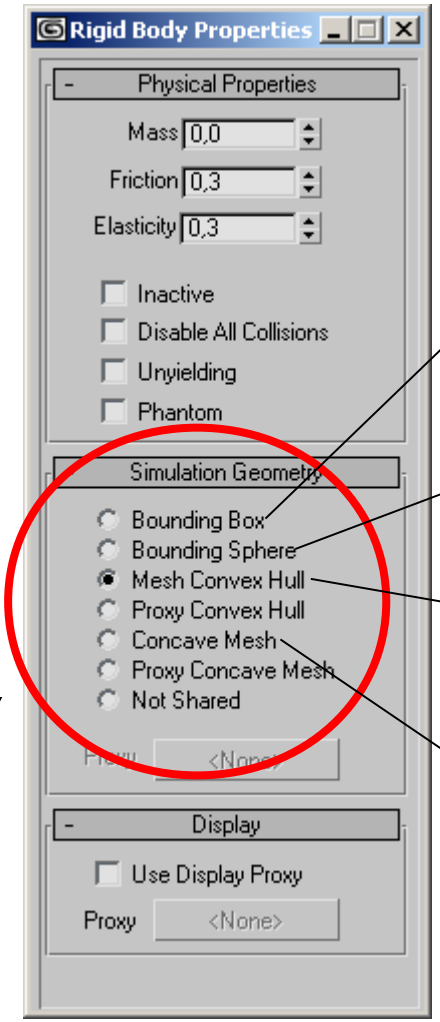
Algorithmes d'intersections entre différents types de volumes:
http://www.realtimerendering.com/intersections.html



	ray	plane	sphere	cylinder	cone	triangle	AABB	OBB	
ray	Gems p.304; SG; TGS; RTCD p.198; SoftSurfer; RTR2 p.618; RTR3 p.781	IRT p.50,88; SG; GTCG p.482; TGS; RTCD p.175; SoftSurfer (more)	IRT p.39,91; Gems p.388; Held jgt 2(4); GTweb; 3DG p.16; GTCG p.501; TGS; RTCD p.127,177; RTR2 p.568; RTR3 p.738	IRT p.91; Gems IV p.356; Held jgt 2(4); GTweb; GTCG p.507; TGS; RTCD p.194	IRT p.91; Gems V p.227; Held jgt 2(4); GTweb; GTCG p.512	Möller-Trumbore jgt 2(1); IRT p.53,102; Gems IV p.24; Held jgt 2(4); GTweb; 3DG p.17; Möller; GTCG p.485; TGS; RTCD p.153,184; Lofstedt jgt 10(2); Chirkov jgt 10(3); Lagae jgt 10(4); SoftSurfer; RTR2 p.578; RTR3 p.746; Havel TVCG June 2009	IRT p.65,104; Gems p.395; Smits; 3DG p.20; Terdiman (optimized Woo); Schroeder; GTCG p.626; TGS; RTCD p.179; Mahovsky jgt 9(1); Williams jgt 10(1); Eisemann jgt 12(4) (code); RTR2 p.572; RTR3 p.742	(IRT p.104; Gems II p.247); GTweb; Gomez; GTCG p.630; TGS; RTCD p.179; RTR2 p.572; RTR3 p.743	(Ge
plane	IRT p.50,88; SG; GTCG p.482; TGS; RTCD p.175; SoftSurfer (more)	GTweb; SG; GTCG p.529; RTCD p.207	distance of center to plane <= radius; GTweb; Gomez; GTCG p.548; TGS; RTCD p.160,219	GTweb; GTCG p.551; TGS;	GTCG p.563; RTCD p.164	Check if all vertices are on one side;(Möller jgt 2(2)); GTCG p.534; SoftSurfer	Gems IV p.74; GTCG p.634; TGS; RTCD p.161,222; RTR2 p.587; RTR3 p.755	GTweb; Gomez; GTCG p.635; TGS; RTCD p.161; RTR2 p.588; RTR3 p.757	
sphere	IRT p.39,91; Gems p.388; Held jgt 2(4); GTweb; 3DG p.16; GTCG p.501; TGS; RTCD p.127,177; RTR2 p.568; RTR3 p.738	distance of center to plane <= radius; GTweb; Gomez; GTCG p.548; TGS; RTCD p.160,219	If radii A+B >= center/center distance; Held jgt 2(4); GTweb; Gomez; GPG p.390; GTCG p.602; TGS; RTCD p.88,215,223; RTR3 p.763	If radii A+B >= center/axis distance; Held jgt 2(4); (GTCG p.602); TGS; RTCD p.114	GTweb; (GTCG p.602)	Held jgt 2(4); GTweb; Karabassi jgt 4(1); TGS; RTCD p.135,167,226	Gems p.335; Gomez; GTCG p.644; TGS; RTCD p.130,165,228; RTR2 p.599; RTR3 p.763	TGS; RTCD p.132,166	A C 3 R R
(capped) cylinder	IRT p.91; Gems IV p.356; Held jgt 2(4); GTweb; GTCG p.507; TGS; RTCD p.194	GTweb; GTCG p.551; TGS;	If radii A+B >= center/axis distance; Held jgt 2(4); (GTCG p.602); TGS; RTCD p.114	If radii A+B >= axis/axis distance; GTweb; GTCG p.602,646; TGS; RTCD p.114	(GTCG p.602)	Held jgt 2(4); TGS	TGS	TGS	(
(capped) cone	IRT p.91; Gems V p.227; Held jgt 2(4); GTweb; GTCG p.512	GTCG p.563; RTCD p.164	GTweb; (GTCG p.602)	(GTCG p.602)	(GTCG p.602)	Held jgt 2(4); GTweb; GTCG p.583			

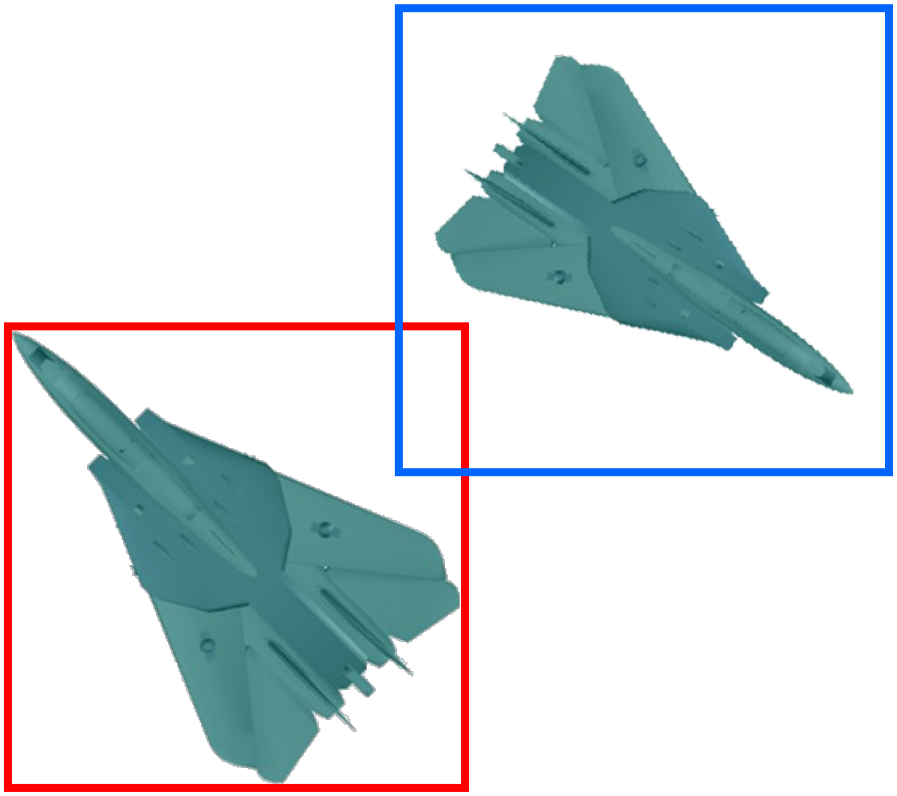
Dans le moteur physique **Reactor** utilisé par **3D Studio Max**, on peut détecter plus ou moins finement les collisions en utilisant une boîte englobante, ou une sphère englobante, ou une enveloppe convexe, ou une enveloppe concave, etc.

Enveloppe englobante de + en + précise (→ temps de calculs plus élevés)



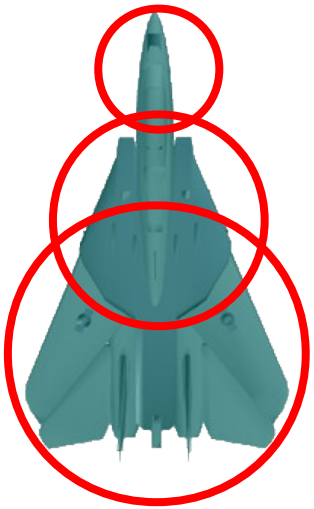
Le test d'intersection entre volumes englobants permet d'accélérer la détection de collision entre deux objets complexes.

Mais ce test est approximatif et donnera des résultats faux si les volumes englobants n'approchent pas au mieux l'objet qu'ils contiennent.

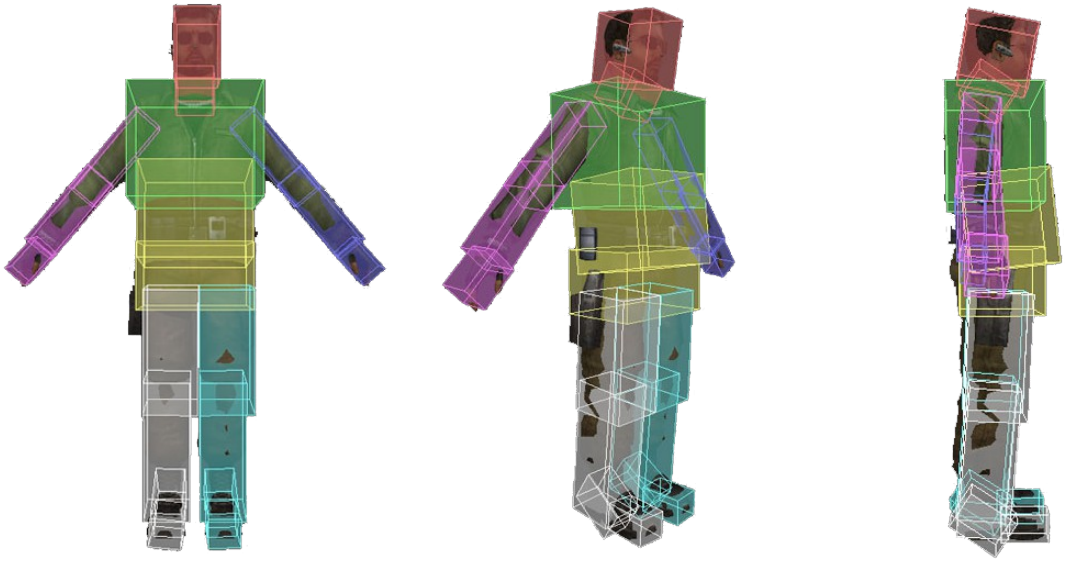


Amélioration : Liste de volumes englobants

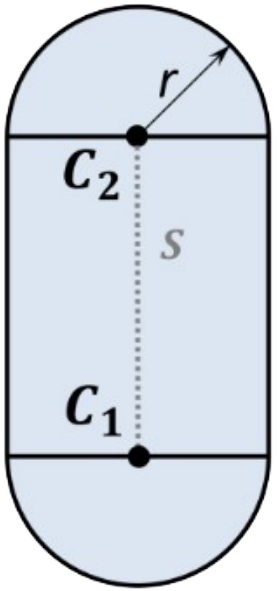
Au lieu d'utiliser un seul volume englobant qui peut être trop grossier, on en utilise plusieurs pour approcher au mieux les différentes sous-parties de l'objet.



Liste de sphères



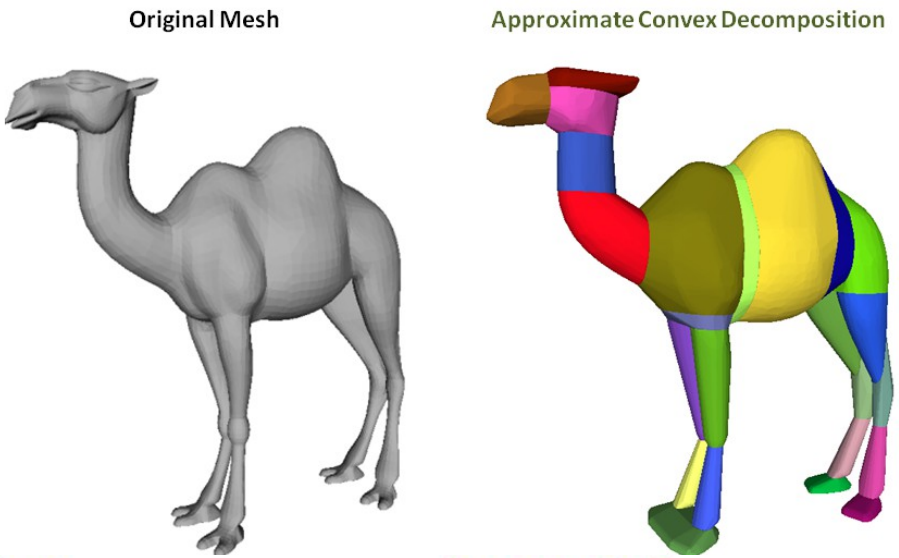
Liste de boîtes



Liste de capsules

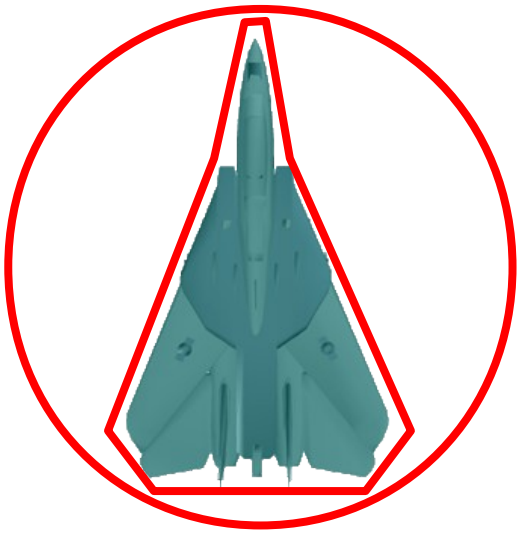
Les capsules (2 centres et un rayon) sont particulièrement bien adaptées pour englober précisément des personnages, quand on souhaite déterminer très précisément une collision (jeu de tir, animation physique avec des « ragdolls ») : on associe les centres aux pivots du squelette.

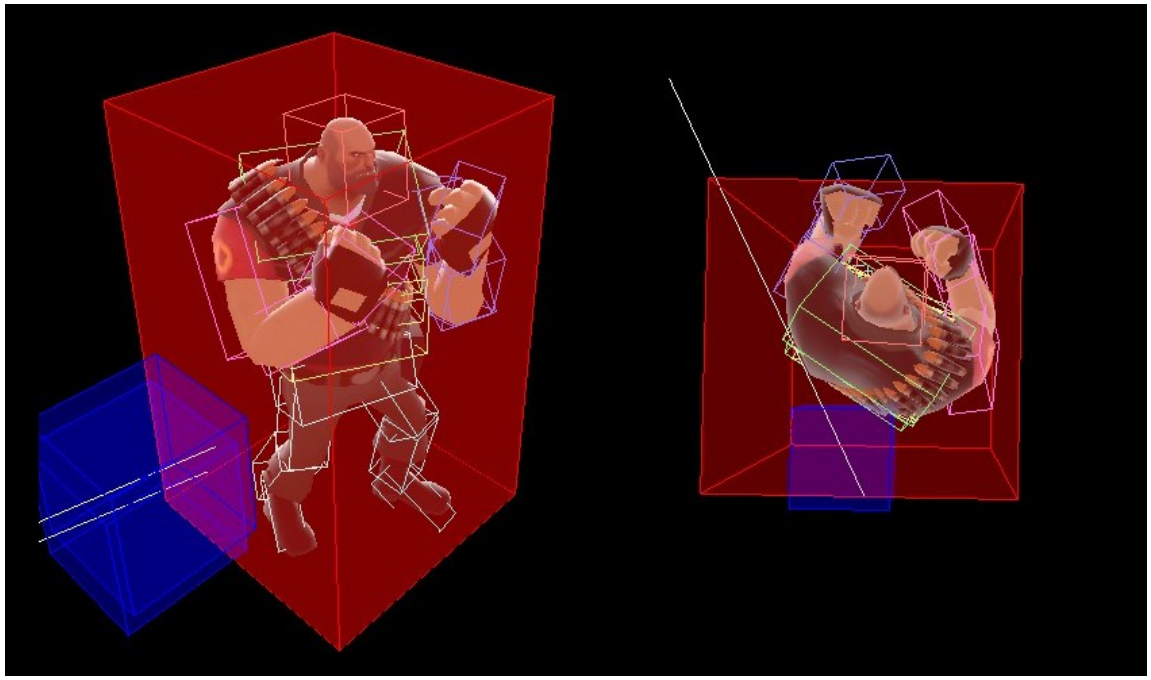
La décomposition d'un objet concave en volumes convexes peut être effectuée à la main par des graphistes, mais peut aussi être produite de manière automatique par des algorithmes tels que V-HACD.
https://code.google.com/p/v-hacd/



Amélioration : hiérarchie de volumes englobants

On peut aussi utiliser une hiérarchie de volumes en testant d'abord la collision avec un volume qui entoure tout l'objet, puis si une intersection est trouvée, on teste l'intersection avec une liste de volumes englobants plus précis comme dans la technique précédente.





3. Utilisations particulières des collisions

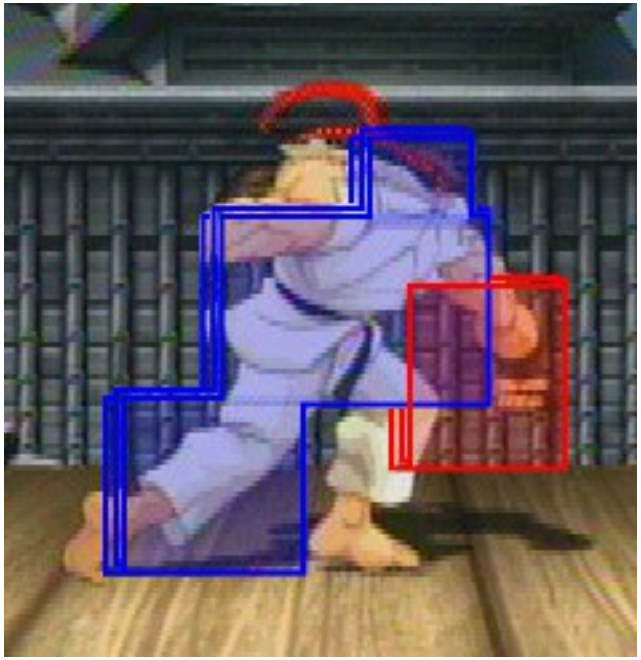
Fondamentalement, les techniques que nous avons vu permettent de savoir si deux objets s'intersectent.

Cette détection peut servir pour de nombreuses actions, pas uniquement pour gérer une collision entre objets.

3.1 Hit box, attack box

Dans les jeux vidéos, on utilise très souvent des boîtes englobantes. En particulier dans les jeux de combats il y en a de deux types :

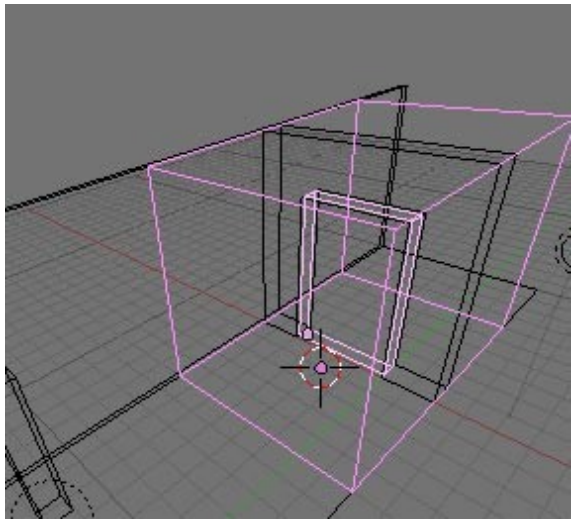
- **Hit box** (ou hurt box) : boîte englobant une partie de l'objet pouvant être frappée.
- **Attack box** : boîte englobant une partie de l'objet pouvant frapper.

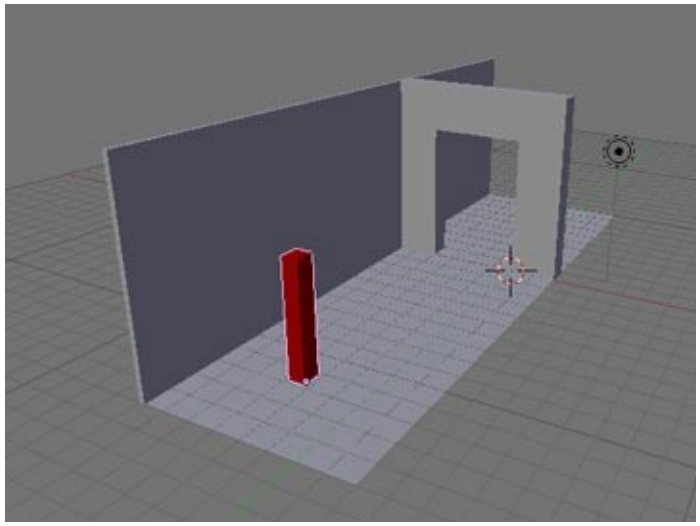


Hit boxes en bleu, attack boxes en rouge

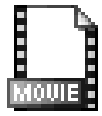
3.2 Zone de déclenchement (“Trigger zones“)

Technique très utilisée dans les jeux et en réalité virtuelle pour déclencher une action lorsque le joueur arrive (ou reste, ou sort) dans une zone définie par un volume englobant (déclenchement d'une alarme, ouverture d'une porte coulissante, apparition d'un monstre, etc.)





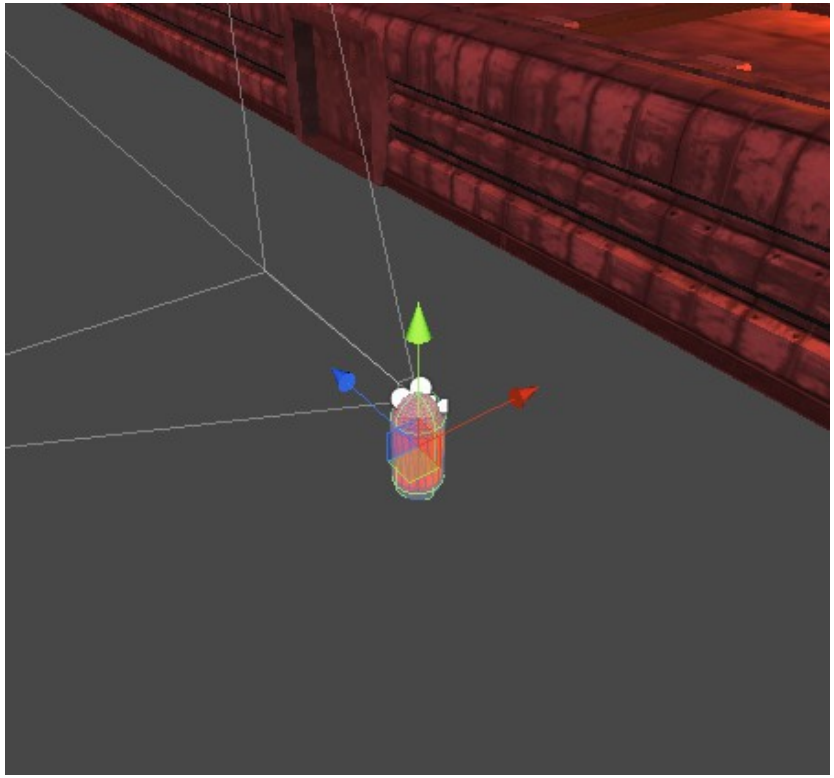
Ouverture d'une porte automatique dans le Blender Game Engine lorsque le joueur arrive dans une zone autour de celle-ci

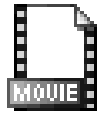


Par exemple dans Unity 3D :
https://unity3d.com/learn/tutorials/modules/beginner/physics/colliders-as-triggers

3.3 Caméra

Un volume englobant peut être associé à la position d'une caméra qui se déplace dans une scène 3D afin de détecter les collisions avec le décor et les objets et éviter que la caméra ne passe au travers de ceux-ci.





Dans cet exemple sous Unity 3D, on associe à la caméra un volume englobant de type capsule.

On reste ainsi à une certaine hauteur au-dessus du sol et on ne passe pas au travers des murs.

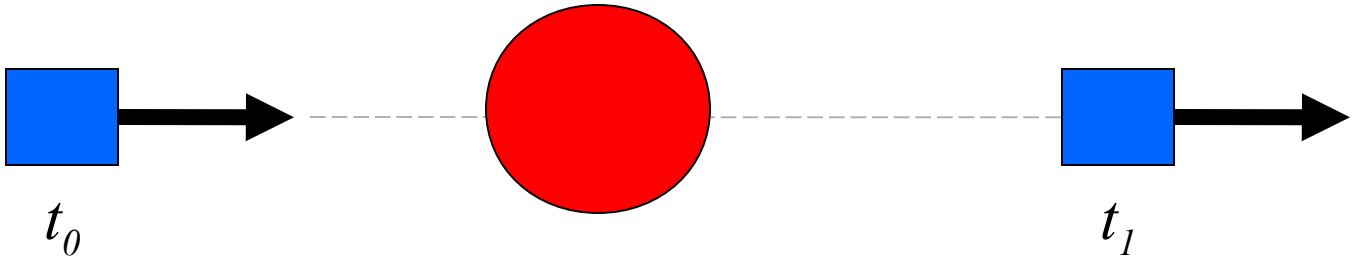
3.4 Saisie d'objets

Dans un jeu ou une simulation, on peut s'emparer d'un objet lorsqu'il y a intersection entre le volume englobant du joueur et celui de l'objet.

4. Détection de collision dynamique

Problème : Animation = échantillonnage du temps (x images/sec)

→ Si les objets se déplacent rapidement, ou si le nombre d'images par seconde est insuffisant, on risque de manquer une collision.

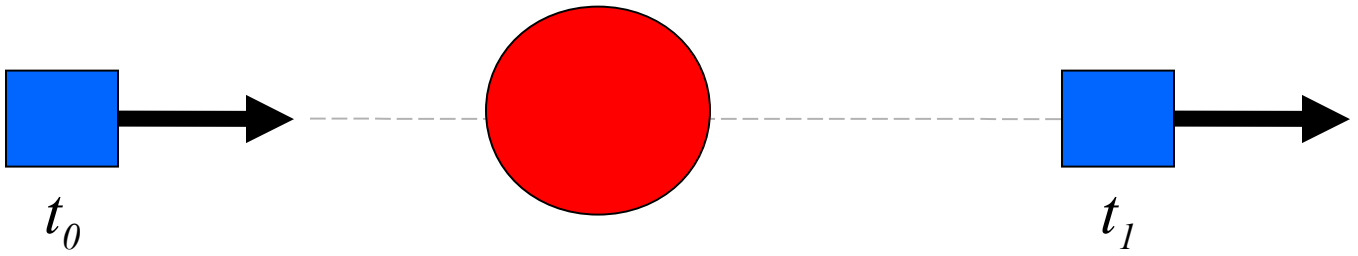


Est ce que la boîte est entrée en collision avec la sphère entre le temps t_0 et le temps t_1 ?

Plusieurs solutions, entre autres :

1) Intersection avec un rayon

Calculer l'intersection entre le rayon définit par deux positions successives d'un objet et les autres objets de la scène (raycasting).

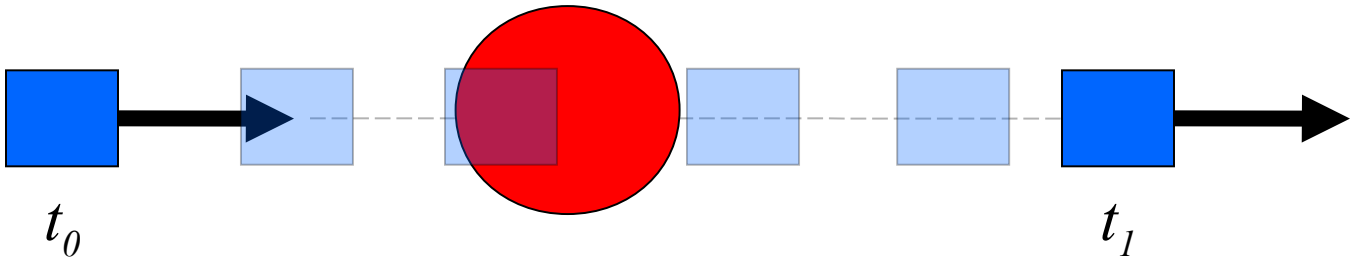


Problème : on résume un objet à une position, ne marche donc que si cet objet est très petit, par exemple une balle.

Que faire si les autres objets sont eux aussi en mouvement ?

2) Suréchantillonage

Ne pas faire le calcul de détection de collisions au même rythme que l'affichage, mais le faire plus souvent en évaluant plusieurs positions des objets entre deux affichages.



On réduit les probabilités de manquer des collisions, mais :

→ Le coût en calculs est plus élevé

→ on ne résoud pas le problème si les objets sont très petits et/ou se déplacent très vite.