Collision et dynamique des corps rigides

LOG725 Ingénierie et conception de jeux vidéo

Introduction

- Dans un jeu vidéo, les objets ne font rien sauf si on leur indique de le faire.
  - On doit donc faire un effort explicite pour s'assurer que les différents objets ne passent pas au travers l'un de l'autre.
- C'est le rôle d'un des composants centraux d'un engin: le système de détection de collisions.
- Le système de collisions est souvent intégré à l'engin de physique.
  - L'engin de physique est plutôt une simulation de la dynamique des corps rigides.

Introduction

- Un corps rigide est un objet solide non déformable.
  - Les déformation élastiques sont très complexes et rarement supportés dans les engins.
- La dynamique fait référence au processus qui détermine comment les corps rigides se déplacent et interagissent sous l'effet de forces.
- La simulation dynamique des corps rigides permet de mouvoir les objets d'une manière hautement interactive, et naturellement chaotique.
- La dynamique des corps rigides utilise beaucoup la détection des collisions.
  - La détection des collisions peut être utilisée de manière indépendante.

Possibilités d'un système de collisions/physique

- La physique ajoute souvent du réalisme qui améliore l'immersion et le plaisir, mais peut être très coûteuse.
- Quelques possibilités offertes par un tel système:
  - Simulation d'un corps sous l'influence de la gravité.
  - Système masse-ressort.
  - Environnements destructibles.
  - Volumes déclencheurs.
  - *Rag dolls*.
  - Vêtements.

Applications de la physique sur différents genres

- Simulation: *Flight Simulator, Gran Turismo.*
- Puzzles: *Portal, World of Goo*.
- Jeux sandbox: *GTA V, Spore, Minecraft*.
- Jeux avec objectifs / basés sur une histoire: Si on veut s'assurer de la linéarité du jeu, il faut faire attention avec la physique!
- En ajoutant de la physique, on sacrifie souvent du contrôle.
- Il faut donc faire attention pour ne pas que la physique entre en conflit avec les mécaniques de jeu.

Impacts de la physique sur le design

- Prédictibilité: Le chaos et la variance amenés par la physique est une source d'imprévus. Il est préférable d'animer si on veut répliquer un comportement à la perfection.
- Réglages et contrôle: On peut modifier les valeurs de la physique (comme la gravité) pour obtenir des effets intéressants. C'est plus difficile de régler la physique qu'une animation!
- Comportements émergents: Il peut arriver que la physique apporte des comportements non désirés.

Impacts de la physique sur la programmation

- Intelligence artificielle: Les chemins empruntés peuvent être imprévisibles en présence de physique.
- Comportements étranges: Les objets animés peuvent s'entrecouper un peu sans problème. Quand ils sont contrôlés par la physique, il peut y avoir des comportements étranges.
- Rendu: Le présence d'environnements destructibles peut invalider les calculs de lumière et ombre.
- Multijoueur: Les calculs de physique qui impactent le jeu doivent être simulés sur le serveur et répliqués aux clients.

Librairies de collision/physique

- Écrire un système de collision et de simulation de dynamique des corps rigides et très demandant.
- Ce système prend généralement un pourcentage significatif du code source de l'engin.
- Heureusement, plusieurs engins de collision/physique sont disponibles sur le marché.
- Très peu d'engins implémentent leur propre système de collisions/physique.

Librairies de collision/physique

- *ODE* (*Open Dynamics Engine*): Logiciel libre de collision et dynamique de corps rigides. Gratuit, et le code source est disponible.
- *Bullet*: Logiciel libre de collision/physique utilisé dans les jeux et dans les films. Supporte la détection de collision en continu.
- *Havok*: Logiciel commercial qui offrent le plus de fonctionnalités (haut prix). SDK disponible sur toutes les plateformes.

Détection de collisions

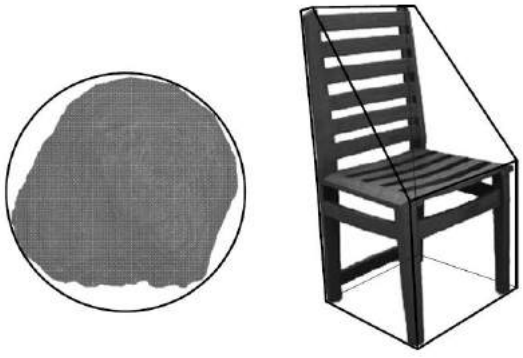
- Le but d'un système de détection de collisions est de déterminer si les objets entrent en contact.
- Chaque objet logique est représenté par une ou plusieurs formes géométriques.
  - Généralement simples, comme des sphères, des boîtes, des capsules. Peut être plus complexe.
- Le système offre aussi de l'information sur les contacts.
  - Directions, vitesses, etc.

Entités de collision

- Si on veut qu'un objet logique puisse entrer en collision avec d'autres, on doit lui donner une représentation spécifique:
  - Forme et position dans le monde.
- Cette structure est distincte de la représentation *gameplay* et de la représentation visuelle. Pourquoi?
- Pour détecter les collisions, on favorise les formes géométriquement et mathématiquement simples.
  - Une roche peut être modélisée avec une sphère pour les collisions.
  - Un corps humain peut être représenté par des capsules interconnectées.

Entités de collision

- On devrait utiliser des formes complexes seulement si la représentation simple n'est pas suffisante.



Entités de collision

- On devrait utiliser des formes complexes seulement si la représentation simple n'est pas suffisante.
- L'entité de collision comprend habituellement:
  - La forme de l'entité.
  - La transformation appliquée.
- Pourquoi ne pas utiliser la transformation de l'objet visuel?
  - La forme et la position/orientation de l'entité de collision n'est pas nécessairement la même que l'objet visuel.

Monde de collisions

- Le système de collision garde les statuts des entités de collision via un singleton: le monde de collisions.
- Le monde de collision est une représentation complète du monde conceptualisé explicitement pour les systèmes de détection de collisions.
- Avantages de stocker ces informations dans un monde à part:
  - On stocke uniquement les entités qui peuvent entrer en contact, évitant du calcul inutile.
  - Permet de stocker les données de façon plus optimale.
  - Meilleure cohésion et encapsulation dans le code.

Monde de collisions

- Si le jeu supporte la physique, le système de dynamique de corps rigides est habituellement intégré dans le monde de collisions.
  - La physique dépend beaucoup du système de détection de collisions.
- Il est fréquent que le système de physique contrôle l'opération du système de collisions.
- Chaque corps rigide est associé à une entité de collision.
- Malgré cette collaboration étroite entre la physique et la collision, les SDK découplent généralement les deux.
  - Pourquoi?

Termes mathématiques

- Forme: Pour le système de collisions, une forme est une région de l'espace ayant un intérieur et un extérieur.
  - Polyèdre ou surface courbée (sphère par exemple).
  - Même en 3D, certains objets sont représentés par un plan (ex: sol, rivière, etc).
- Intersection: L'intersection entre 2 formes est l'ensemble de tous les points (infini) qui se trouvent à l'intérieur des deux formes.

Termes mathématiques

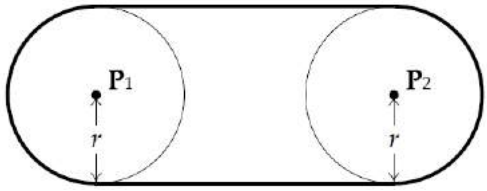
- Contact: Dans un jeu, on est plus intéressé aux informations d'un contact plutôt que l'ensemble des points de l'intersection.
  - Vecteur de séparation: Vecteur selon lequel on peut glisser les objets pour résoudre la collision.
  - Quelles entités de collision sont entrées en contact, vitesses, etc.
- Convexité: Un des concepts les plus importants dans la détection de collision est la convexité d'une forme (vs concave).
  - Forme convexe: Aucun rayon dont l'origine est à l'intérieur de la forme coupe la surface plus d'une fois.
  - Les calculs de collisions sur les formes convexes sont beaucoup plus simples (et performants) que sur les formes concaves.

Primitives de collision

- Sphère: Le volume le plus simple en 3D. La sphère est représentée par un point et un rayon.
  - Stockée dans un vecteur à 4 *floats*.
  - Format parfait pour les librairies SIMD.
- Capsule: Volume en forme de pilule, composé d'un cylindre et deux hémisphères.
  - Forme résultant d'une sphère que l'on glisse (approximation).
  - Forme plus efficace qu'un cylindre ou une boîte.

Primitives de collision

- Les capsules sont souvent utilisées pour modéliser les formes qui s'apparentent à un cylindre: membres du corps humain par exemple.
- Stockées avec deux points et un rayon.

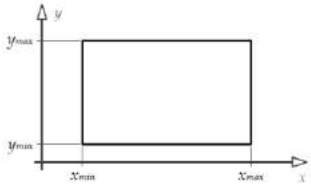


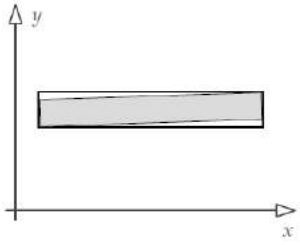
Axis-aligned bounding boxes (AABB)

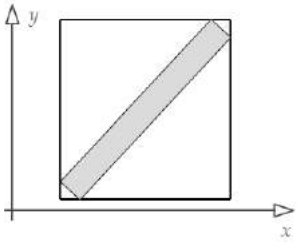
- Un AABB est un volume rectangulaire (cuboïde) dont les faces sont parallèles aux axes du système de coordonnées.
  - Une boîte alignée aux axes le sera uniquement pour ce système d'axes spécifique.
  - Défini par deux points: un contenant les coordonnées minimum, l'autre les coordonnées maximum.
- Avantage: le test de pénétration avec les autres AABB est très efficace.

Axis-aligned bounding boxes (AABB)

- Limitation: l'AABB doit être aligné avec les axes en tout temps pour rester efficace. Ceci implique que l'AABB doit être recalculé lorsque l'objet tourne.
  - Même si l'objet a une forme qui s'apparente à une boîte, l'AABB peut dégénérer selon la rotation.







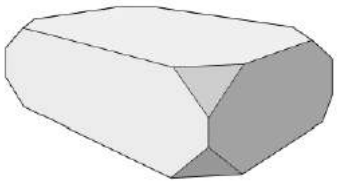
Oriented bounding boxes (*OBB*)

- Si on permet à un AABB d'effectuer des rotations, nous obtenons un OBB.
- Un OBB est représenté par trois demi-dimensions (demi largeur, demi hauteur, demi profondeur) et une transformation.
- Les OBB sont plus utilisées parce qu'ils sont meilleurs pour englober un objet à orientation arbitraire, et ils demeurent simples.

Discrete oriented polytopes (DOP)

- Un DOP est un cas plus général du AABB et du OBB.
- Un DOP est construit en prenant un nombre de plans à l'infini, et en les glissant selon leur normale jusqu'à un contact avec l'objet dont la forme est approximée.
  - AABB: 6-DOP avec les plans parallèles aux axes.
  - OBB: 6-DOP avec les normales des plans parallèles aux axes de l'objet.
- Méthode commune de construction d'un DOP: On démarre avec un OBB, puis on coupe les coins à 45 degrés.

Discrete oriented polytopes (DOP)



Volumes convexes arbitraires

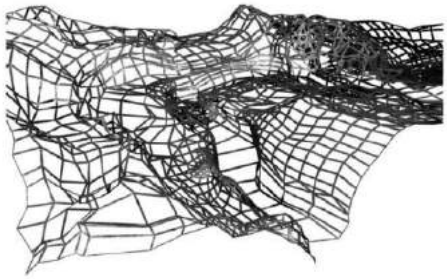
- La plupart des engins de collision permettent aux artistes 3D de construire un volume convexe arbitraire.
- Un outil hors ligne s'assure que la forme est un polyèdre convexe.
- Si la forme passe le test, elle est convertie en une collection de plans (k-DOP).
- Ces formes sont plus complexes et moins efficaces, mais des algorithmes peuvent quand même les traiter rapidement grâce à la convexité.

Soupe de polygones (*poly soup*)

- Certains systèmes supportent des formes arbitraires, non convexes : *poly soup*.
- Ils sont construits à l'aide de formes simples comme les triangles.
  - Modélisent les géométries statiques complexes, comme le terrain et les édifices.
- Très cher à calculer dans le système de collisions.
  - Les engins limitent généralement les *poly soup* aux entités qui ne prennent pas part à la simulation dynamique.

Soupe de polygones (*poly soup*)

- Un poly soup n'est pas nécessairement fermé (pas d'intérieur/extérieur): comment le système de collision sait dans quelle direction pousser pour résoudre la collision?



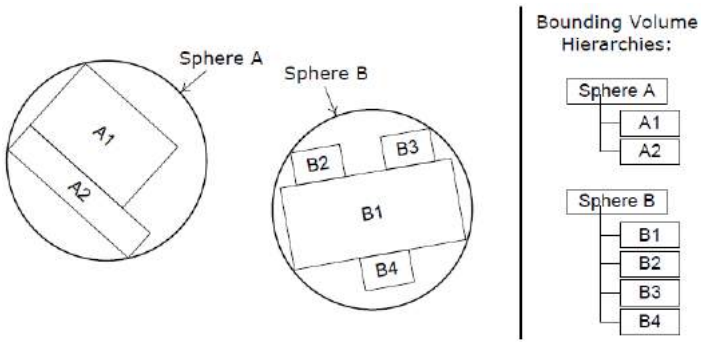
Formes composées

- Certains objets ne peuvent être approximés par une simple forme.
  - On utilise plutôt des formes composées.
- Par exemple, un chaise peut être approximée avec deux boîtes:

Formes composées

- Une forme composée est une alternative plus efficace aux *poly soups* pour modéliser une forme concave.
  - Deux ou plusieurs volumes convexes peuvent souvent être plus performants qu'un seul *poly soup*.
- Certains systèmes de collision tirent avantage du volume englobant les formes composées. Par exemple:
  - Dans Havok/Bullet, c'est une détection de collision *midphase*.

Formes composées



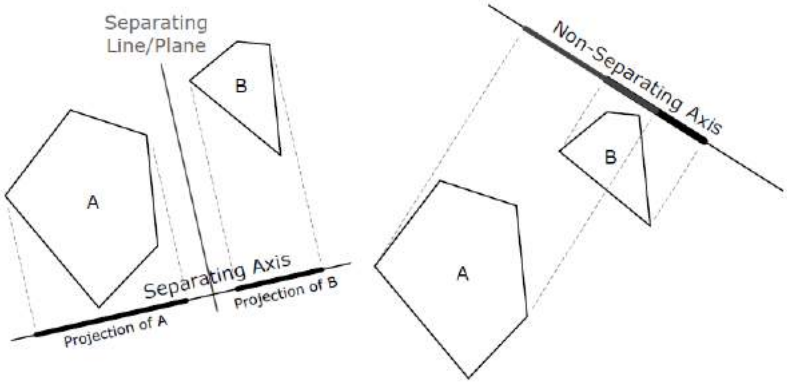
Géométrie analytique et tests de collision

- Le système de collisions utilise la géométrie analytique pour détecter les contacts.
  - On utilise pas des approximations numériques, mais bien les mathématiques exactes.
- Cas simples:
  - Point vs sphère: vecteur de séparation entre les deux. Si la norme du vecteur est plus grande que le rayon, pas de contact.
  - Sphère vs sphère: vecteur de séparation entre les deux centres. Si la norme du vecteur est plus grande que la somme des rayons, pas de contact.
- Comment détecter des collisions avec des volumes convexes arbitraires?

Séparation des convexes

- Les systèmes de détection de collisions utilisent beaucoup le théorème de l'axe de séparation.
  - S'il existe un axe selon lequel les projections des deux formes convexes ne se chevauchent pas, il n'y a pas d'intersection.
  - Si les formes sont concaves, on ne peut rien conclure.
- Plus facile à visualiser en 2D:
  - S'il existe une ligne dont la forme A est entièrement d'un côté, et la forme B entièrement de l'autre, il n'y a pas de contact.

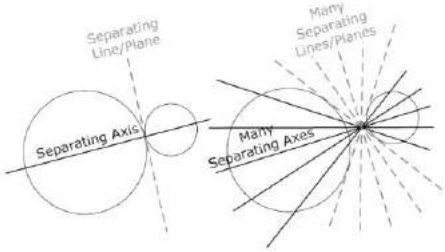
Séparation des convexes



- Comment trouver l'axe de séparation?

Séparation des convexes

- Cas simple avec les sphères:
  - Sans intersection, un axe de séparation valide est celui parallèle au vecteur entre les deux centres.

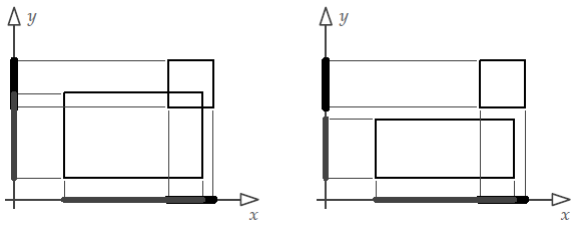


- Moins évident pour les autres formes convexes...

AABB vs AABB

- On applique le théorème de l'axe de séparation, de façon plus simple.
- Comme les faces des deux AABB sont parallèles à un système d'axes commun:
  - Si un axe de séparation existe, ce sera l'un de ces axes.
- Si les coordonnées selon tous les axes se chevauchent, il y a intersection.
  - Dans tous les autres cas: pas d'intersection.

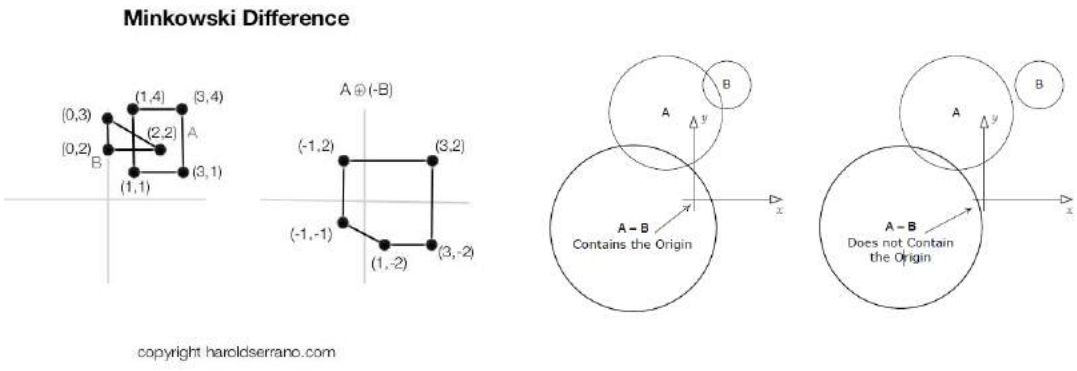
AABB vs AABB



Algorithme GJK

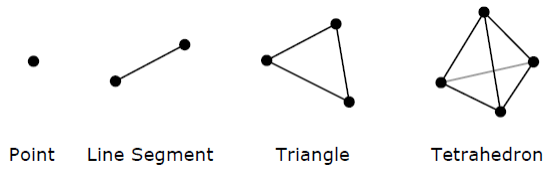
- Un algorithme très efficace existe pour détecter les intersections entre des polytopes convexes arbitraires: algorithme GJK.
- Différence de Minkowski:
  - On prend chaque point dans la forme B, et on le soustrait à chaque point dans la forme A.
- La différence de Minkowski contient l'origine, si et seulement si, les deux formes se coupent.
- Cette différence est également une forme convexe: On s'intéresse uniquement à son enveloppe convexe.
- Comment savoir si l'enveloppe convexe contient l'origine?

Algorithme GJK



Algorithme GJK

- Déterminer si une forme convexe arbitraire contient un point n'est pas performant.
- Pour déterminer si la différence de Minkowski contient l'origine, on utilise un simplex.

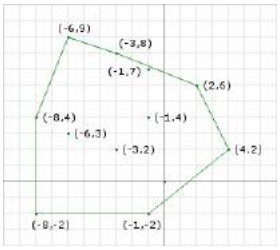


Algorithme GJK

- On débute par sélectionner un vertex arbitraire sur l'enveloppe de la différence de Minkowski : premier point du simplex.
- On trace le vecteur de direction entre ce point et l'origine.
- On sélectionne le point le plus loin dans la direction vers l'origine : deuxième point du simplex.
- À partir du vecteur perpendiculaire au simplex, on sélectionne le point le plus loin dans la direction vers l'origine : troisième point du simplex.
- Si le simplex contient l'origine: il y a intersection.
- Si on ne peut trouver un point plus près à une itération: il n'y a pas d'intersection.

Algorithme GJK – Quelques exemples

- Le but: on essaie d'encercler l'origine en s'en rapprochant.

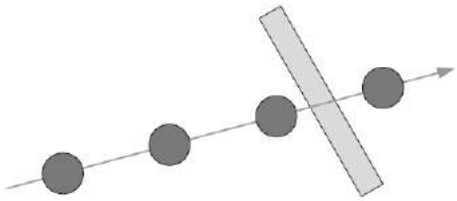


Détection de collisions entre corps en mouvement

- Les exemples précédents démontrent des intersections statiques entre les objets.
  - Plus complexe lorsque les objets sont en mouvement.
- Technique simple: à chaque *frame*, on traite chaque corps rigides de façon statique.
  - Cette technique fonctionne si les objets ne bougent pas trop rapidement entre eux.
  - Beaucoup d'engins de collision utilisent cette approche par défaut (dont Havok)!

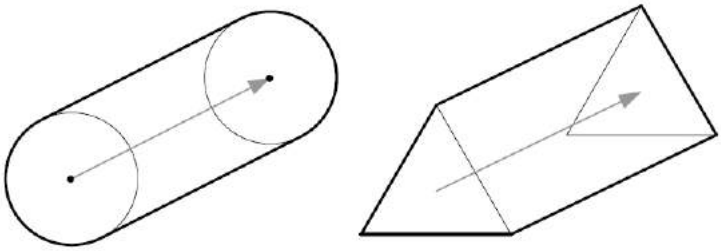
Détection de collisions entre corps en mouvement

- Cette technique échoue avec des petits objets à grande vitesse.
  - Imaginons un petit objet qui bouge si vite qu'il couvre une distance plus grande que sa propre taille dans une *frame* : *tunneling*.
  - Des exemples?



Formes balayées

- Une façon de régler le problème du *tunneling* est d'effectuer un balayage de formes.
- Une forme balayée est formée avec le mouvement d'une forme d'un point à un autre.

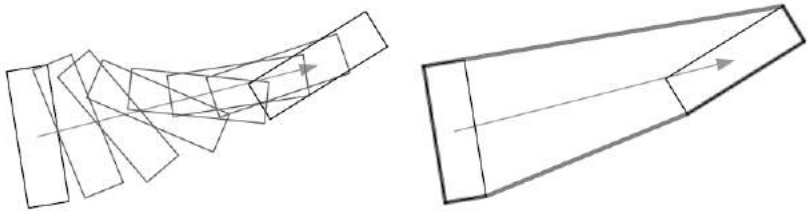


Formes balayées

- Cette approche effectue une interpolation linéaire du mouvement du objet entre deux *frames*.
- Pas toujours efficace. Pourquoi?
- Si la forme suit un tracé courbé, l'interpolation linéaire n'est pas une bonne approximation.
  - Si on crée une forme balayée selon le tracé courbé, on obtient pas nécessairement une forme convexe: beaucoup plus difficile.
- Si la forme convexe que l'on balaie est en rotation, le résultat n'est pas nécessairement convexe non plus!

Formes balayées

- On peut résoudre certains de ces problèmes en extrapolant les points extrêmes d'un mouvement pour créer une forme convexe.
- C'est malgré tout moins précis, et des algorithmes plus précis peuvent être nécessaires dans certains cas.



Détection de collision continue (*Continuous collision detection* - *CCD*)

- La CCD est une autre façon de traiter le problème du *tunneling*.
- Pour chaque forme, on garde sa position et son orientation à la dernière *frame* ainsi qu'à la *frame* courante.
- L'algorithme calcule le temps d'impact (TOI) en cherchant selon la direction du mouvement de manière itérative.

Requêtes de collision

- Une autre responsabilité du système de détection de collisions et de répondre à des questions hypothétique à propos des volumes de collision dans le monde. Par exemple:
  - Si la balle voyage à partir d'une arme vers une direction, quel objet sera rencontré en premier, s'il y a lieu?
  - Est-ce qu'un véhicule peut se déplacer d'un point A au point B sans contact?
  - Quels sont les ennemis dans un rayon prédéfini du joueur?
- À la différence des autres techniques de détection de collision, les *casts* de collision ne sont pas vraiment dans le monde.

Tracé de rayon (*ray cast*)

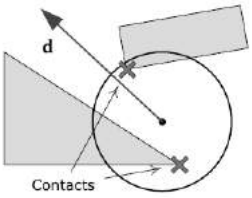
- La forme la plus simple (et de loin la plus utilisée) est le tracé de rayon.
- Un segment de ligne directionnel est tracé dans le monde, et est testé avec les entités de collision du monde.
- Les systèmes de tracé de rayon représente le segment de ligne à l'aide d'un point p_0 et le vecteur delta d:
  - $p(t) = p_0 + t*d$
- La plupart des systèmes de collision peuvent retourner le premier contact.
  - Certains peuvent retourner tous les contacts.

Tracé de forme (*shape casting*)

- Une autre requête commune: À quelle distance un objet convexe pourrait voyager dans une direction avant d'entrer en contact avec quelque chose.
- Comme avec le tracé de rayon, le tracé de forme est défini en spécifiant le point de départ p_0 et le vecteur d.
  - On spécifie en plus le type, les dimensions et l'orientation de la forme.
- On doit considérer 2 cas:
  - La forme à tracer est déjà en contact avec une autre entité.
  - La forme à tracer n'est pas en contact et est libre de bouger.

Tracé de forme

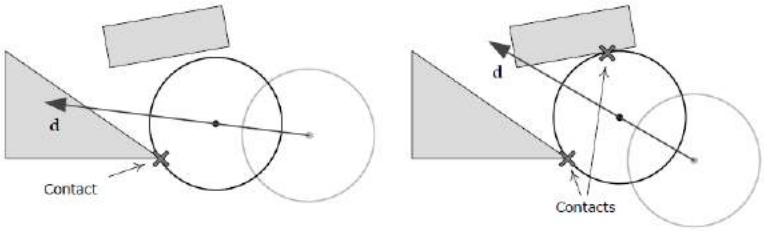
- Dans le premier cas:
  - Le système de collisions rapporte les contacts avec les entités (qui peuvent être sur la surface ou à l'intérieur de la forme tracée).



- Dans le second cas:
  - La forme peut se déplacer selon la direction avant d'entrer en contact avec une ou plusieurs entités.

Tracé de forme

-



- Quelques exemples d'application:
  - Tracé de sphère pour déterminer si la caméra entre en collision avec des objets.
  - Tracé de capsule pour s'assurer qu'un personnage animé est en contact avec le sol lors d'un déplacement sur terrain accidenté.
  - Lance roquette.

Filtres de collision

- Il est commun de vouloir activer ou désactiver les collisions entre certains types d'objets. Par exemple:
  - La plupart des objets doivent passer au travers de la surface de l'eau.
  - Les bateaux doivent pouvoir entrer en contact avec l'eau pour flotter.
- Une approche fréquente est de catégoriser les objets dans le monde et utiliser une structure pour déterminer si la catégorie peut entrer en collision.
  - Dans *Havok* par exemple, on utilise les *collision layers*.

Dynamique des corps rigides (physique)

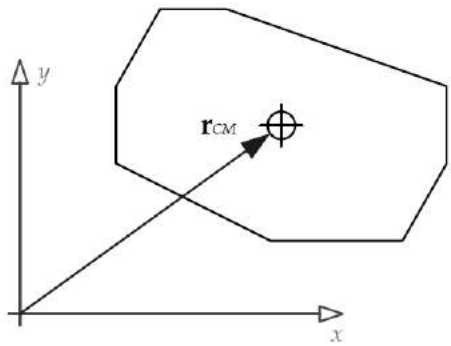
- Dans un engin de jeux, on s'intéresse à la cinématique des objets:
  - Comment ils se déplacent dans le temps.
- Jusqu'à récemment, les engins de physique utilisaient presque exclusivement la dynamique des corps rigides. Deux simplifications importantes:
  - Mécanique newtonienne: on assume que les objets obéissent aux lois de Newton. (pas de relativité et d'effet quantique)
  - Corps rigides: tous les objets simulés sont parfaitement solides.

Termes techniques

- Unités: La plupart des simulations de dynamique des corps rigides utilisent le système d'unités MKS. (mètres, kilogrammes, secondes)
- Dynamique linéaire: Description du mouvement d'un corps sans prendre en considération les effets de rotation.
- Dynamique angulaire: Description du mouvement rotationnel d'un corps.
- Centre de masse: Point d'équilibre d'un corps. Dans la dynamique linéaire, on considère que la masse est centrée au centre de masse.

Dynamique linéaire

- La position d'un corps rigide peut être représenté par un vecteur de position. Chaque force est appliqué sur ce point.



Dynamique linéaire

- La vitesse est la dérivée première de la position.
- L'accélération est la dérivée seconde de la position.
- Une force est définie comme n'importe quoi qui permet à un objet avec une masse d'accélérer ou décélérer.
  - A une magnitude et une direction (vecteur!)
  - La force nette est calculée en sommant chaque force qui agit sur un corps.
- Deuxième loi de Newton: F = ma (unité: Newton)

Dynamique linéaire

- La quantité de mouvement (*momentum*) est la masse multipliée par la vitesse.
  - Intuitivement, on peut imaginer le *momentum* comme la résistance d'un objet à arrêter.
- Question:
  - Si un train bouge à vitesse constante, aucune force n'est appliquée sur lui (système parfait sans friction/résistance de l'air). F = ma = m*0 = 0. Si la force nette est de 0, comment se fait t'il que ce soit légèrement déplaisant se faire frapper par un train?

Solution aux équations de mouvement

- Le problème central à la dynamique des corps rigides est de résoudre les équations de mouvement en sachant les forces qui agissent sur les objets.
- Solution analytique:
  - Les équations peuvent rarement être résolues de manière analytique. Par exemple: $y(t) = 1/2\ g * t^2 + v_0 * t + y_0$.
- Intégration numérique:
  - Permet de résoudre les équations différentielles *frame* par *frame*, en utilisant les résultats de la précédente en entrée pour la suivante.
  - Approximations acceptables de la physique réelle.

Dynamique angulaire en 2D

- Pour étudier le mouvement rotationnel causé par des forces appliquées ailleurs que sur le centre de masse, on utilise la dynamique angulaire.
- En 2D, la dynamique angulaire est très semblable à la dynamique linéaire.
- Pour chaque quantité linéaire, il y a un analogue en angulaire.

Orientation et vitesse angulaire

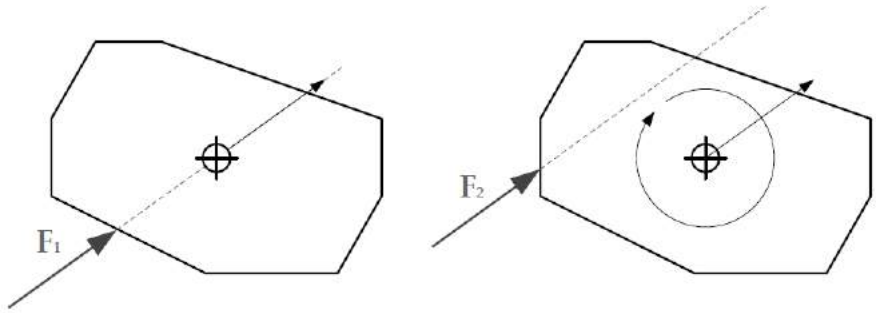
- En 2D, chaque corps rigide est une mince feuille sur un plan.
- Chaque motion linéaire survient sur le plan xy, et toute les rotations surviennent selon l'axe z.
- La vitesse angulaire mesure le taux de changement de l'angle de rotation dans le temps d'un corps rigide. (radians par seconde ou degrés par seconde)
  - Dérivée de l'angle de rotation dans le temps
- L'accélération angulaire est la dérivée seconde de l'angle de rotation dans temps. (taux de changement de la vitesse angulaire)

Moment d'inertie et moment de force

- L'équivalent rotationnel de la masse est le moment d'inertie.
- Comme la masse décrit la difficulté à modifier la vitesse linéaire, le moment d'inertie exprime la difficulté à modifier la vitesse angulaire selon un axe. (scalaire en 2D, puisqu'on tourne autour de l'axe z)
- Le moment de force est la force rotationnelle appliquée à un corps lorsqu'on applique une force ailleurs qu'au centre de masse.
  - En plus du mouvement linéaire déjà causé!

Moment d'inertie et moment de force

- Le moment de force est calculé à l'aide d'un produit vectoriel, et augmente avec la distance du centre de masse:
  - C'est pourquoi on utilise des leviers!



Réponse aux collisions

- Les termes précédents assument que les corps rigides n'entrent pas en contact avec autre chose.
- Quand ça arrive, la simulation dynamique doit effectuer des étapes pour s'assurer d'une réponse réaliste à la collision.
  - Les objets doivent jamais être dans un état d'interpénétration après que l'étape de simulation physique soit complétée!
- Dans la réalité, beaucoup de choses se passent lors d'une collision:
  - Compression des corps, changement de vitesses, perte d'énergie, etc.

Réponse aux collisions

- Dans les simulations dynamique des corps rigides en temps réel, on approxime ces détails en effectuant certaines simplifications:
  - La force de collision survient sur une période de temps infiniment petite: impulsion (les vitesses changent instantanément).
  - Pas de friction au point de contact entre les corps.
  - Les interactions subatomiques sont approximée à l'aide du coefficient de restitution ε.
    - ε = 0 : Collision parfaitement élastique. Aucune énergie n'est perdue.
    - ε = 1 : Collision parfaitement plastique. Toute l'énergie est perdue.

Réponse aux collisions linéaires (sans rotation)

- ε = coefficient de restitution
- v_1 = vecteur de vitesse du corps 1
- v_2 = vecteur de vitesse du corps 2
- m_1 = masse du corps 1
- m_2 = masse du corps 2
- n = normale aux surfaces au point de contact (normalisée)
- p = impulsion à appliquer

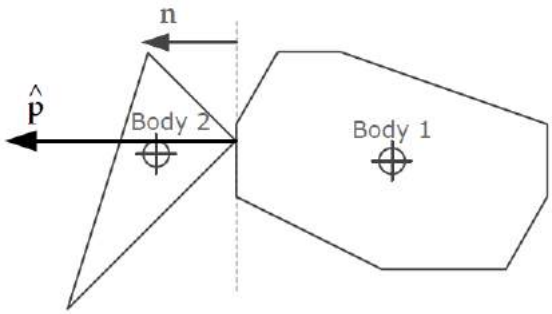
$$p = \frac{(\varepsilon + 1) * (v_2 \cdot n - v_1 \cdot n)}{\frac{1}{m_1} + \frac{1}{m_2}} * n$$

$$v_1 = v_1 + \frac{p}{m_1}$$

$$v_2 = v_2 - \frac{p}{m_2}$$

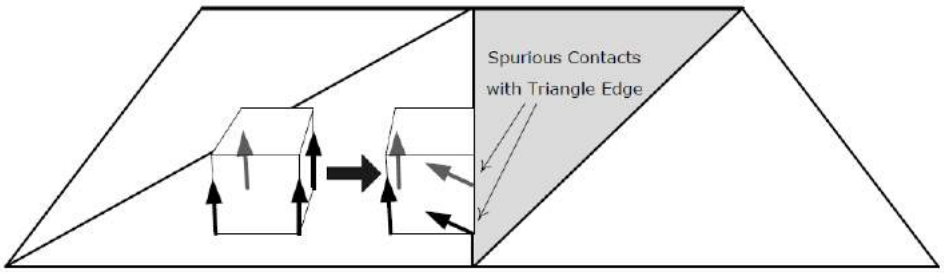
Réponse aux collisions linéaires

- Exemples! (voir collision.m)



Friction et faux contacts

- Lorsqu'on applique la friction, un problème survient lorsque la surface glisse à la frontière entre deux polygones.



Friction et faux contacts

- Ces faux contacts causent des effets indésirables et des calculs inutiles.
- Solutions possibles:
  - Analyse des points de contacts, et on abandonne ceux qui semblent faux avec des heuristiques:
    - Si on sait que l'objet glisse sur une surface, et qu'une normale de contact apparait parce qu'on est près de la frontière d'un triangle…
  - On annote le maillage avec de l'information d'adjacence des triangles: (utilisé dans *Havok* 4.5+)
    - Le système de physique peut ignorer les contacts avec les arêtes intérieures.

Repos

- Quand on retire de l'énergie d'un système simulé, les objets en mouvement doivent éventuellement s'arrêter.
- Dans une simulation, arrêter les objets en mouvement n'est pas si simple à cause de plusieurs facteurs:
  - Nombres à point flottant.
  - Approximations des calculs de physique.
- Ces facteurs font que l'objet peut « osciller » pour toujours plutôt que de s'arrêter.

Repos

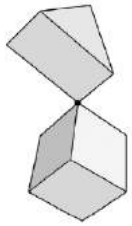
- Les engins utilisent des heuristiques pour s'assurer que les objets tombent au repos:
  - On retire de l'énergie pour s'assurer de tomber à 0.
  - On les arrête lorsque la vitesse moyenne atteint un seuil.
- La plupart des engins de physique mettent les objets au repos lorsqu'ils sont inactifs.
  - Optimisation de performance. Pourquoi calculer des mathématiques pour un objet qui n'est pas en mouvement?
- D'un manière similaire aux graphes de scènes, *Havok* et *PhysX* utilisent des *simulation islands* pour optimiser.

Contraintes

- Un corps rigide a six degrés de liberté.
  - 3 pour la translation, 3 pour la rotation.
- Les contraintes limitent le mouvement d'un objet en réduisant le nombre de degrés de libertés. Par exemple:
  - Un chandelier.
  - Une porte qui peut être ouverte, ou détruite et sortir de sa charnière.
  - Une voiture qui tire une remorque.
  - Une corde ou une chaîne.

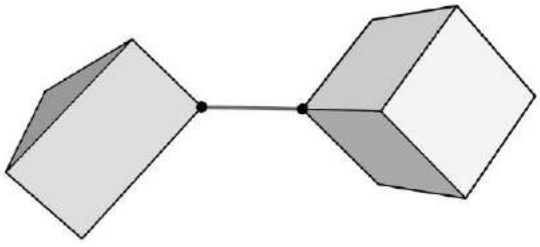
Contrainte point à point

- La plus simple des contraintes: agit comme une articulation sphéroïde.
- Les corps peuvent bouger dans la direction désirée, pourvu qu'un point sur un corps s'aligne à celui d'un autre corps.



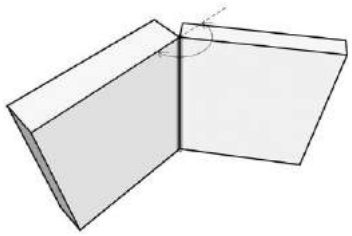
Ressort raide

- Un ressort raide (*stiff spring*) est comme la contrainte point à point, mais les deux points sont distancés.
- On peut l'imaginer comme une barre invisible entre deux points.



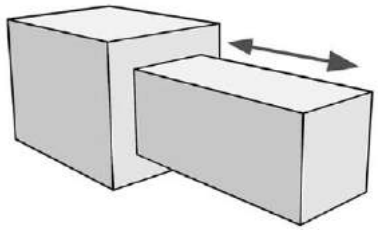
Charnières

- Une contrainte charnière limite le mouvement rotatif à un degré de liberté sur l'axe de la charnière.
  - Charnière illimitée: l'objet peut faire des rotations complètes.
  - Charnière limitée: l'objet peut tourner dans un intervalle limité d'angles. (porte)



Contrainte prismatique

- Les contraintes prismatiques agissent comme des pistons.
- Une degré de liberté pour la translation. Un ou aucun degré de liberté pour la rotation.



Rag dolls

- Les *rag dolls* sont crées en contraignant les corps rigides entre eux pour former un corps humain.
  - Capsules pour pieds, tibias, cuisses, torse, etc.
- Les corps rigides sont contraints à bouger de manière réaliste pour le corps humain: pas de membres qui plient dans le mauvais sens.
- Collaboration étroite avec le système d'animation.

Contrôle du mouvement des corps rigides

- La plupart des jeux nécessitent un contrôle sur la façon dont les objets bougent sous l'influence des forces naturelles et en réponse aux collisions:
  - Une bouche d'air qui applique une force sur les objets qui y passent.
  - Une voiture qui tire une remorque.
  - Un fusil anti gravité.
- **Gravité**: La gravité est présente dans presque tous les jeux.
  - Pas une force, mais une accélération (environ) constante qui affecte tout.
  - Pour cette raison, la direction et la valeur de la gravité sont un paramètre global dans la plupart des SDK.

Application des forces

- Un nombre arbitraire de forces peut être appliqué aux corps dans la simulation physique d'un jeu.
- Une force agit toujours sur un intervalle de temps fini.
- Les forces sont souvent dynamique: elles changent de direction et de magnitude à chaque *frame*.
- La fonction d'application d'une force dans la plupart des SDKs doit donc être appelée à chaque frame pour la durée de la force.

Application d'un couple et impulsion

- Couple:
  - Pour appliquer un effet de rotation pur à un corps, on utilise un couple.
  - Deux forces égales et opposées sur des points équidistants du centre de masse.
- Impulsion:
  - Une impulsion est une force qui agit sur un intervalle de temps infinitésimal.
  - La plus courte durée dans un jeu est le temps d'une *frame*…
  - Les SDKs offrent donc une fonction spéciale pour appliquer une impulsion (linéaire ou angulaire).

Avancement de la collision/physique dans la boucle de jeu

1. Les force et moments de force sont numériquement intégrées en avançant de Δt pour déterminer la position et orientations potentielles des corps.
2. La librairie de détection de collision détermine si des contacts ont été générés entre les objets.
3. Les collisions sont résolues en appliquant des impulsions/forces ou le solveur de contraintes.
4. Les contraintes sont satisfaites avec le solveur de contraintes.

Solveur de contraintes

- Algorithme itératif qui tente de satisfaire un grand nombre de contraintes à la fois en minimisant l'erreur entre les positions/orientations actuelles des corps et les positions/orientations idéales définies par les contraintes.
- Après l'intégration numérique, le solveur de contraintes:
  - Détermine les positions relatives des objets par rapports à leurs contraintes et calcule l'erreur.
  - S'il y a erreur, le solveur déplace les corps pour la minimiser ou l'éliminer.

Solveur de contraintes

- À chaque itération, l'intégration numérique bouge les corps hors des contraintes, et le solveur tente de les réaligner.
- Avec une approche idéale, ces itérations tendent vers une stabilisation du système.
- Comme il s'agit d'une minimisation d'erreur, il est possible d'obtenir des résultats étranges dans certains cas.

Mise à jour de la simulation physique

- À chaque frame, on met à jour la simulation physique:
  - Mise à jour des corps rigides contrôlés par le jeu: On ajuste les corps rigides dans le monde physique pour qu'ils concordent avec le jeu. (pas de physique)
  - Mise à jour des *phantoms*: Un *phantom* agit comme une entité de collision, mais sans corps rigide.
  - Mise à jours des forces, application des impulsions et ajustement des contraintes.
  - **Avancement de la collision / physique.**
  - Mise à jour des objets du jeu contrôlés par la physique.
  - Détection de collisions avec *phantoms*.
  - *Ray/shape casts*.

Exemples de collision/physique dans un jeu

- Grenades:
  - Les grenades sont parfois modélisées comme des objets physique à 100%: perte de contrôle significative!
  - On peut imposer des forces ou impulsions artificielles pour contrôler les rebonds ou limiter la distance parcourue après impact.
  - Parfois, on contrôle le mouvement de la grenade 100% manuellement: animation.
- Balles de fusil:
  - Souvent des *ray casts* instantanés. Problèmes: Pas de temps de déplacement ou d'effet de la gravité sur les longues distances.

Exemples de collision/physique dans un jeu

- Objets destructibles:
  - Les objets destructibles sont particuliers parce qu'il débutent dans un état pas endommagé, où ils apparaissent comme un objet unique cohésif.
  - Lors de la destruction, ils se brisent en plusieurs morceaux graduellement, où d'un seul coup.
  - Dans *Havok*, on divise le modèle en pièces destructibles et on assigne un corps rigide à chacune.
  - Pour des raisons de performance, on peut utiliser une version non endommagée de la géométrie, puis la remplacer lors de la destruction.

Exemples de collision/physique dans un jeu

- On peut utiliser des contraintes pour que certaines parties pendent de l'objet plutôt que se détacher complètement.
- Si l'objet est rigide et qu'on fait des trous, on doit pouvoir briser l'objet physique pour permettre de tirer des balles dans l'ouverture par exemple.
- Un autre problème complexe: Les structures qui s'effondrent lentement.
  - Lors de l'effondrement d'un pont, on doit propager l'effondrement d'un bout à l'autre lentement pour que le point semble massif.
  - Avec les fonctionnalités de base d'un SDK de physique, le pont s'effondre d'un seul coup!

Exemples de collision/physique dans un jeu

- Mouvements du personnage:
  - Le mouvement d'un humanoïde est trop complexe pour être contrôlé adéquatement avec les forces et les impulsions.
  - On utilise donc un système d'animation, et non la physique.
  - Pour déplacer le personnage, on utilise des entités de collision sphères/capsules pour les corps rigides du squelette.
  - On les utilise pour faire des *shape casts* dans la direction désirée du mouvement.
  - Permet plusieurs effets:
    - Personnage qui glisse le long d'un mur quand il y fonce.
    - Permettre au personnage de marcher au-dessus des trottoirs plutôt qu'être coincé.

Physique avancée

- Avec la dynamique des corps rigides, on peut modéliser beaucoup d'effets incroyables dans les jeux vidéo.
- Ces systèmes ont des limitations, particulièrement au delà des corps rigides.
- Beaucoup de recherche se fait dans le domaines des fluides et des corps déformables, même à l'ÉTS.

Physique avancée

- Corps déformables: Des engins commencent à supporter les corps déformables, comme DMM.
- Tissu: Le tissu est modélisé comme une feuille de points avec des masses, connectés par des ressorts rigides. Il est très difficile d'obtenir du tissu réaliste à cause des collisions avec les objets et les approximations numériques.
- Cheveux: Les cheveux peuvent être modélisés avec un grand nombre de filaments simulés, ou avec des approches simplifiées.

Physique avancée

- Simulation des fluides: Domaine de recherche très actif. Peu utilisé dans les jeux temps réel.