Université du Québec
École de technologie supérieure

Département de Génie Logiciel
École de technologie supérieure
Montréal

Développement d'un moteur physique permettant la simulation du comportement dynamique des corps rigides

Jérôme Schmaltz
SCHJ23117909

Rapport final
PHE-RPT

Projet synthèse du baccalauréat en génie logiciel
à l'École de Technologie Supérieure (ETS).
Hiver 2006

DÉVELOPPEMENT D'UN MOTEUR PHYSIQUE PERMETTANT LA SIMULATION DU COMPORTEMENT DYNAMIQUE DES CORPS RIGIDES

RÉSUMÉ

Les moteurs physiques sont des composantes de plus en plus présentes aussi bien dans l'industrie, intégrés dans les simulateurs, que dans les jeux, intensifiant le niveau de réalisme. Leurs complexités en font des éléments difficilement intégrables et utilisables tandis que leurs conceptions basées sur la performance se traduisent par leur manque de portabilité. L'étude présente un moteur physique développé afin de pallier à ces manquements en exposant parallèlement divers cas de simulations réalisées avec celui-ci. Une analyse comparative entre modèles analytiques et données empiriques pour chaque cas de simulation témoignera de l'exactitude du moteur développé.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

1. INTRODUCTION

1.1 Problématique

De nos jours, les investissements se font de plus en plus importants dans le milieu de la simulation. Dans les domaines d'ingénierie, il n'est plus chose rare que de voir croître des compagnies spécialisées dans cette discipline. Actuellement, la simulation n'est plus entièrement réservée à l'industrie. Effectivement, depuis quelques années déjà, elle fait partie intégrante de l'industrie des jeux vidéo.

On pourrait définir la simulation comme étant la résultante de la reproduction d'un comportement réel, approximé numériquement et régit selon des lois physiques. La simulation s'accomplit par ce que l'on nomme couramment un engin ou moteur physique. Il est en fait le cœur de tout le processus d'animation en temps réel.

Acteur prenant de l'importance dans l'industrie, le moteur physique permet d'assurer la validité et de vérifier chaque design et conception d'ingénierie. Pour ce faire, chaque conceptualisation est numérisée pour ensuite être intégrée à un simulateur et testée. D'un tout autre ordre d'idées, le moteur physique peut aussi bien servir au divertissement. Incorporé à de nombreux titres de la scène des jeux vidéo, il intensifie le niveau de réalisme infligé à l'expérience de l'utilisateur en le plongeant dans un monde virtuel assiégé par une représentation en trois (3) dimensions de chaque scène caractérisée par le comportement convainquant de tout objet l'entourant.

Selon Eberly [4], l'avènement d'un tel type de composante est sans conteste dû à la prise en charge des calculs graphiques par les processeurs intégrés à même les cartes vidéo. Anciennement, ces mêmes calculs étaient desservis par le processeur central. Matériellement, la quantité d'opérations découlant des rendus de scènes monopolisait le temps du processeur central et ne permettait donc pas de s'en servir efficacement afin de simuler la réalité. Avec la naissance des processeurs graphiques et en constatant leur constante évolution, il est presque chose banale de nos jours que de simuler des environnements physiques complexes à l'aide des processeurs centraux.

Présentement, on retrouve différentes implémentations et architectures de moteurs physiques. Cependant, bon nombre d'utilisateurs font face à des problèmes de facilité d'utilisation et de portabilité avec ces moteurs. Effectivement, pour la plupart de ces engins, un temps considérable doit être prévu pour la maîtrise des fonctionnalités qu'ils offrent. Par surcroît, puisque la notion de performance est prédominante, leur utilisation se limite souvent à une seule plateforme. Dans de tels cas, que peut faire un utilisateur peu expérimenté dans le domaine qui veut simuler un cas simple de physique mécanique afin d'étudier le comportement de corps rigides?

Ce projet synthèse s'est penché sur l'analyse l'élaboration partielle d'un engin physique simulant le comportement dynamique de corps rigides. La conception de ce moteur à tenter de résoudre le problème de facilité d'utilisation relié à l'implémentation de cas de simulation simple avec les moteurs physiques existants par la proposition d'une architecture portable et extensible.

1.2 Portée du projet

La portée du projet est cernée en un premier temps par une implémentation partielle des fonctionnalités possibles d'un moteur physique. Plus précisément par la conception d'une architecture ne permettant uniquement que la simulation numérique de comportements physiques apparentés aux particules ou corps solides, en excluant les corps déformables, en deux (2) dimensions. Deuxièmement, aucune considération n'a été apportée en ce qui concerne l'optimisation du code. Rappelons à ce fait que l'objectif premier est une étude portée sur les moteurs physiques et leurs architectures.

1.3 Objectifs

L'élaboration et la conception d'un moteur physique ne sont pas tâches triviales. Considérant ce fait, l'établissement des objectifs primaires (ainsi que les livrables associés) a été borné à la simulation physique proprement dite, non au développement d'un pipeline graphique complet qui considérerait autant le positionnement, l'orientation que l'affichage d'un objet et d'effets de lumières. Ici, il est bien important de noter que *l'engin physique est en fait une composante de l'architecture de l'engin graphique*.

Le développement de ce projet a été graduellement orienté par tous les facteurs de risque préalablement définis. Tel que suggéré par M. Eric Paquette, un développement itératif a été employé pour découvrir ou résoudre toutes formes de risques possibles [1] et a permis un meilleur suivi du progrès global [1].

Tels que mentionnés précédemment, les objectifs élémentaires de ce projet ont reposé en premier temps sur la simulation numérique du comportement physique de corps rigides. Donc, l'établissement d'une architecture de base permettant de simuler l'évolution d'un corps dans un environnement sans contraintes avec les dynamiques de Newton puis dans un environnement avec contraintes (forces contraignantes) à l'aide des mêmes équations. Finalement, il y a eu analyse d'un dernier cas illustrant la situation où deux corps rigides entrent en contact. Pour ce faire, il y aura eu modification de l'architecture de base afin d'assurer la gestion de collisions des corps rigides (détection et réponse aux collisions). Parallèlement à tout cela, un autre objectif considérable a été de représenter la simulation de ces cas sous forme graphique. Ici, il y a eu intégration de composantes supplémentaires (par conséquence un changement a l'architecture) afin de lier les fonctionnalités de représentation graphique à celles des simulations physiques. De la sorte, une attention particulière a été portée à l'abstraction des interfaces du module de rendus graphiques afin de se détacher de dépendances quelconques à un système particulier.

1.4 Méthodologie employée

La section courante décrit la méthodologie, techniques et outils employés lors du développement du projet. Une approche itérative a donc été proposée en soumettant des prototypes fonctionnels pour chacun des cas de simulation afin de s'assurer de cerner les risques les plus importants. Grâce à l'incrémentation du niveau de difficulté de chaque itération et en considérant le fait qu'il y a toujours eu une compatibilité entre eux, la livraison du dernier artefact a permis de finaliser l'architecture finale.

En règle générale, les documents livrés ont nécessité l'utilisation d'outils tels que Microsoft™ Word ou LaTeX, ce dernier étant préféré pour la génération de documentation scientifique. Il est à noter que les sources de renseignements utilisées pour rédiger la documentation proviennent d'articles scientifiques, thèses et livres spécialisés.

L'architecture a été développée selon le modèle itératif ADD [5] permettant de construire autour d'attributs architecturaux.

Pour chaque cas de simulation livré, la méthodologie suivante a été empruntée, à savoir :

1. L'élaboration du cas
1.1 Description du cas
1.2 Modélisation mathématique et physique du cas
1.3 Génération de points de contrôle
2. Implémentation du cas
2.1 Transformation du modèle en langage de programmation
2.2 Intégration du modèle programmé avec le moteur physique
2.3 Simulation du cas (génération des valeurs numériques du cas)
2.4 Recueil des valeurs numériques simulées
3. Vérification des données
3.1 Comparaison des valeurs théoriques générées en 1.3 avec celles obtenues empiriquement en 2.3.
4. Ajustements et correctifs

Remarques :

1.3 : Pour la généralisation de points de contrôle, l'isolement de situations ont été pré déterminées en conditions du temps, de la vitesse, de l'accélération ou bien lors de collisions entre objets.

2.3 : Afin de calculer et d'approximer numériquement les simulations de cas concrets physiques, l'utilisation de l'intégrateur de Runge-Kutta de 4^{e} ordre a été choisie [3,7]. La simulation des cas concrets ont été créés à partir de l'élaboration de situations physiques basées sur des dynamiques Newtoniennes [3,7].

3 : Les données recueillies ont été vérifiées à partir des points de contrôle établi en 1.3 et ceux en 2.3. Il est à noter qu'une légère variance (acceptables) entre les données a été constatée dû au fait que l'intégration numérique provoque de légères erreurs de troncation.

4 : Toute erreur a été corrigée dans le code et dans les documents étant rattachés au cas.

2. EXIGENCES SPÉCIFIQUES

2.1 Fonctionnalités

La présente section identifie de façon plus détaillée la liste des fonctionnalités soutenue par le moteur physique. Les principales exigences sont d'ordre fonctionnel et non fonctionnel et basés sur les objectifs [1].

2.1.1 Résolution numérique d'équations différentielles

La résolution d'équations différentielles s'est fait par le biais d'une technique numérique permettant d'approximer, avec le plus de précision possible et en un temps raisonnable, une fonction mathématique quelconque dérivée d'une modélisation physique. Il est à noter que les détails techniques entourant le choix de la méthode numérique sont abordés plus loin.

2.1.2 Représentation graphique de la simulation numérique

La simulation numérique d'un cas quelconque a été représentée à l'écran par le biais d'une librairie graphique. Le vecteur d'état (position, vitesse, etc.) de chaque corps rigide ainsi que sa description spatiale (coordonnées des sommets, couleurs) ont été transmis à l'appareil graphique [1] pour l'affichage.

2.1.3 Simulation du mouvement sans contrainte

Le moteur physique a permis le calcul des attributs physiques[1] d'un corps rigide de type polygone convexe dans un environnement où aucune force physique externe n'a influencé sa trajectoire. Pour ce faire, l'approche mathématique et physique a été basée sur la dynamique newtonienne [3].

2.1.4 Simulation du mouvement avec forces contraignantes

Le moteur physique a permis le calcul des attributs physiques[1] d'un corps rigide de type polygone convexe dans un environnement où le corps est soumis à des forces externes telles que la gravité, la friction ou force dissipatrice quelconque[2]. Pour ce faire, l'approche mathématique et physique a été basée sur la dynamique newtonienne [3].

2.1.5 Gestion de collisions

La gestion de collision s'est exécutée premièrement par la détection d'une collision puis du calcul d'une réponse à cette collision entre deux corps rigides de type polygone convexe. L'influence des attributs

[1] Par attribut physique on identifie notamment la masse, centre de gravité, accélération et vitesse de l'objet.
[2] Par forces dissipatrices on entend forces de friction par exemple.

physiques de chacun des corps rigides a été considérée afin de simuler avec le plus de précision possible, le monde réel. La réponse à cette collision a été exprimée en termes de position, vitesse, accélération, et ce, angulairement et linéairement [3, 7, 11].

2.2 Performance

Une scène complexe composée de plusieurs milliers de polygones peut être une tâche ardue et coûteuse en temps pour un engin graphique. L'intégration à celui-ci d'un moteur physique, afin d'influencer le comportement des primitives composant la scène graphique, devrait se faire avec le moins d'impact possible en terme de temps.

Il n'est pas trivial de quantifier avec exactitude le temps maximal alloué pour chacun des calculs de simulation accordé. Cependant, on a pu juger des performances du moteur physique par le biais du nombre d'images par secondes (FPS, *frames per second*) requis pour afficher une scène. À cet effet, nous avions considéré qu'un débit de 30 FPS sur une carte graphique ATI Radeon 7500™ AGP 4x doté de 64 Mo DDR à été de mise.

2.3 Exactitude

Bien qu'avec certaines approximations numériques permettant la conservation d'une performance respectable, l'exactitude des calculs issus des cas de simulation a tenu compte de marges d'erreur acceptables[3] afin de représenter avec le plus de fidélité possible la réalité.

2.4 Extensibilité

Une des forces principales du moteur physique a été la possibilité d'étendre ses fonctionnalités afin de permettre la simulation de cas additionnels. Cette extensibilité a été traduite dans l'architecture même du moteur afin de faciliter l'intégration de composantes additionnelles. La quantification de cet attribut est en fait nominal puisque nous n'avons pas disposé pas de données issues d'un projet semblable pour en faire une comparaison.

2.5 Portabilité

La portabilité des engins physiques connus est assez restreinte puisque leur implémentation est orientée exclusivement vers une plateforme spécifique. Une des exigences importantes dont on a tenu compte est l'écriture d'une architecture portable qui a pu faire fonctionner le moteur physique tant sur des environnements Microsoft Windows™ que Solaris™.

2.6 Facilité d'utilisation

La facilité d'utilisation a été un requis important puisqu'il est en relation directe avec la problématique qui a été définie [1]. Malheureusement, cette exigence n'a pu être quantifiée et vérifiée puisqu'aucune étude sur des sujets n'a été produite.

[3] Ici le terme « acceptable » est arbitraire supposant implicitement la quantification d'une marge donnée lors de la présentation d'un cas de simulation.

2.7 Contraintes de design

Plusieurs contraintes ont été fixées dans cette section afin de faciliter le développement du moteur physique ainsi que la réalisation des cas de simulation. La liste ci-dessous n'est pas exhaustive du fait que dépendamment des cas, il y a eu ajout de certaines contraintes (pour la plupart physique).

- Le développement des cas de simulation par le biais de Java.
- La représentation graphique des cas de simulation a été produite avec la librairie JOGL, une implémentation d'OpenGL™ pour Java™.
- La modélisation des cas de simulation a été produite en une (1) ou deux (2) dimensions. L'utilisation de la troisième dimension aurait complexifié énormément certains cas de simulation, notamment le troisième.
- La simulation des cas s'est effectuée exclusivement avec des polygones convexes puisque les algorithmes et théorèmes utilisés ont été dépendant de cette condition.

3. REPRÉSENTATION ARCHITECTUALE

Afin de conceptualiser l'architecture de base de l'engin physique, l'utilisation la méthode itérative ADD [5] a été adoptée. L'accomplissement de cette méthode a été légèrement modifié afin de présenter uniquement les étapes suivantes :

1. Choisir le module à décomposer
2. Détailler le module selon :
    a. Choix d'attributs architecturaux
    b. Choix de patrons architecturaux qui épaulent les attributs architecturaux
    c. Définir les interfaces des modules

Pour chacune des itérations proposées dans ADD, les choix conceptuels se sont effectués à travers les attributs architecturaux :

- Portabilité
- Extensibilité
- Facilité d'utilisation

Il est à noter, que considérant la problématique définie [1], il a été impératif de considérer en premier lieu ces attributs architecturaux. Le choix de tels attributs a influencé aussi le fait de retrancher l'attribut de *performance* de la liste puisque l'emphase s'est établie sur la portabilité de l'architecture.

À titre de rappel [12], voici les tactiques architecturales sélectionnées [5] afin d'épauler les attributs architecturaux préalablement définis :

Portabilité : Un des patrons architecturaux des plus utilisés afin de permettre une portabilité maximale à l'architecture développée est sans contredit la *machine virtuelle*. Effectivement, la *machine virtuelle* permet un niveau d'abstraction supplémentaire entre les modules des couches inférieures d'une application et le système d'exploitation.

Extensibilité : La modélisation interne de l'architecture est établie avec un *modèle en couches*. Chacune de ces couches permet un niveau d'abstraction particulier permettant une facilité de maintenance accrue ainsi qu'une extensibilité plus efficace.

Facilité d'utilisation : Selon Bass [5], une des tactiques utilisées pour faciliter l'utilisation de l'architecture est la séparation de l'interface graphique (GUI) du reste. Essentiellement ce qui est prôné par l'approche proposée est l'implémentation d'un système de *communication implicite* par la conception d'un système de *modèle-vue-contrôleur*.

À partir des nombreuses itérations de la méthode ADD, une architecture de base a été créée. Son évolution, reposant sur l'ajout de fonctionnalités multiples (à partir des cas de simulation créés) a permis de finaliser sa conceptualisation. Les prochaines sous-sections subséquentes présentent l'architecture détaillée, sous forme de diagrammes de classes, résultant du moteur physique développé au sein de ce projet.

Prenons la figure 3.1 qui représente une vue d'ensemble des paquetages de l'architecture finale.

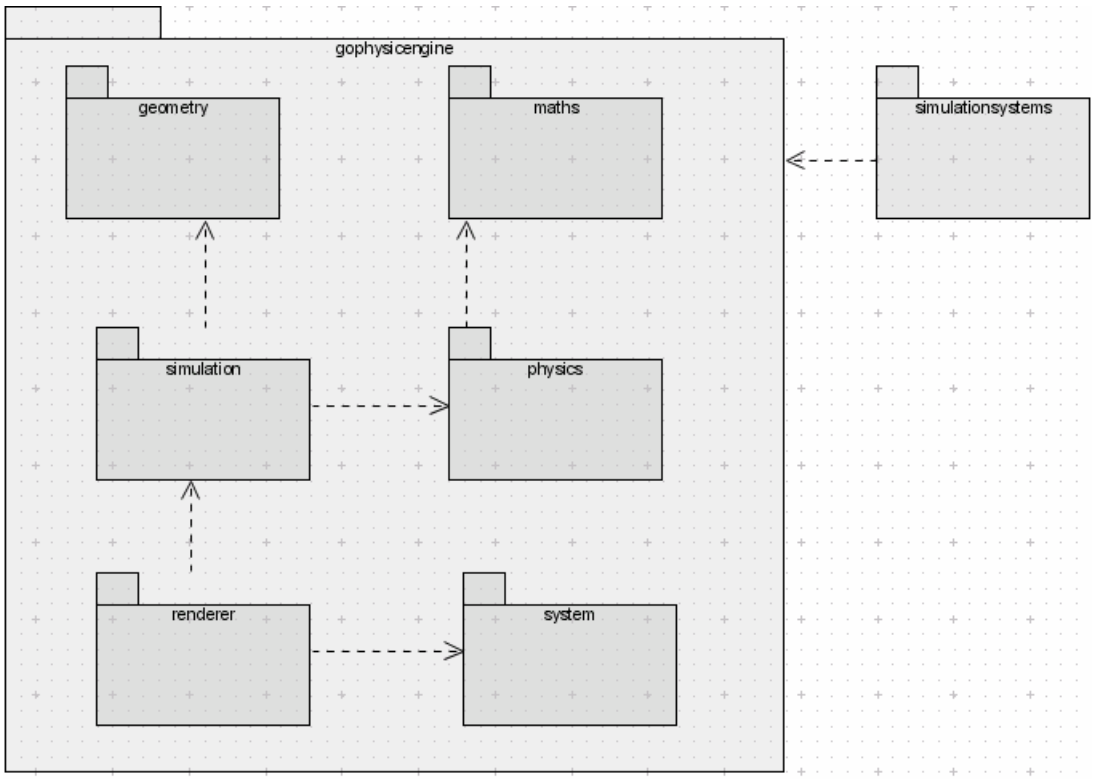


Figure 3.1 Architecture (Vue d'ensemble)

Deux paquetages majeurs scindent le moteur physique. Le paquetage *GoPhysicEngine* et *SimulationSystems*. *GoPhysicEngine* contient exclusivement tous les sous paquetages propres à la simulation :

Geometry :	Utilisé pour modéliser un corps rigide quelconque en polygone convexe.
Simulation :	S'occupe de la gestion des nœuds composant une scène. Par nœuds on identifie tous les corps rigides présents dans une scène (un environnement quelconque) modélisés sous forme de polygones convexes, visibles par l'utilisateur.
Physics :	Contiens les classes définissant les comportements génériques des corps rigides.
Renderer :	Séries d'interfaces et classes abstraites généralisant l'emploi du rendu de scène à l'écran.
System :	Configuration de base du moteur physique.

Le deuxième paquetage *SimulationSystems*, dicte le comportement à utiliser lors de l'intégration de nouveaux cas de simulation au moteur physique. De plus amples informations suivront.

Détaillons maintenant chacun des sous paquetages précédemment identifiés afin de connaître leurs rôles et responsabilités au sein du système, pour cela observons la figure 3.2.

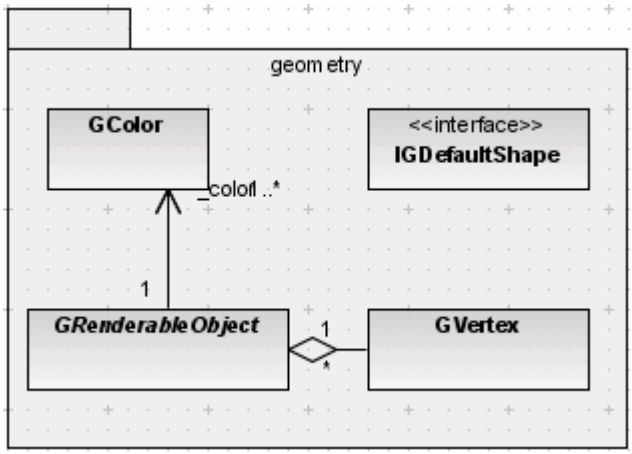


Figure 3.2 Diagramme de classes du paquetage *Geometry*.

Tel que mentionné, le paquetage *Geometry* est utilisé pour modeler à l'écran les divers corps rigides visibles de la simulation. Par modeler, on désigne l'action de représenter graphiquement un corps visible sous la forme d'un polygone convexe. Le corps rigide à envoyer au système de rendu est basé sur la classe abstraite *GRenderableObject*. Cette classe permet de conserver les coordonnées des sommets du polygone ainsi qu'une couleur permettant de le représenter à l'écran. Chaque coordonnée est symbolisée par la classe *GVertex* tandis que la couleur est représentée par *GColor*. Le fait de définir les classes *GVertex* et *GColor* a permis de conserver un niveau d'abstraction suffisant pour ne pas dépendre d'un système de rendu particulier.

La figure suivante, proposent les trois (3) classes contenues dans le paquetage *Maths*.

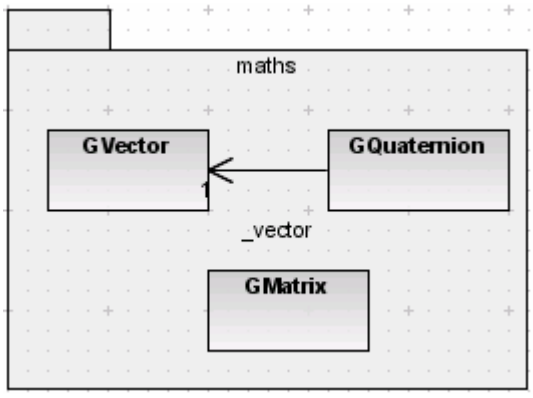


Figure 3.3 Diagramme de classes du paquetage *Maths*.

À ce fait, le paquetage Maths contient les principales classes de base permettant de manipuler les entités mathématiques d'algèbre vectorielle. *GVector* représente un vecteur, *GQuaternion*, un quaternion (pas

utilisé concrètement dans le projet, puisque nos modèles reposent sur les deux (2) dimensions) et finalement *GMatrix* qui aide aux opérations matricielles.

Voici le paquetage *Physics*, représenté par la figure 3.4.

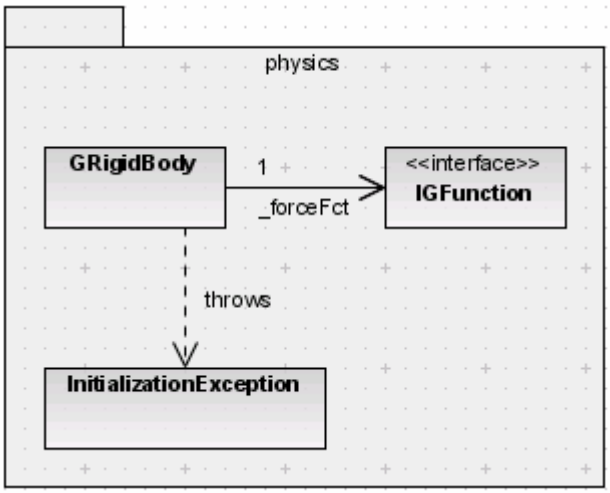


Figure 3.4 Diagramme de classes du paquetage *Physics*.

Le paquetage *Physics* est au cœur même du moteur physique puisqu'il permet la représentation des états physiques des corps rigides par la classe *GRigidBody*. De plus, cette même classe possède la capacité d'approximer numériquement le comportement du corps rigide pour chaque pas de temps de la simulation. Finalement, l'interface *IGFunction* décrit le comportement qu'une classe l'implémentant devrait définir pour représenter une force ou torsion à appliquer à un corps rigide.

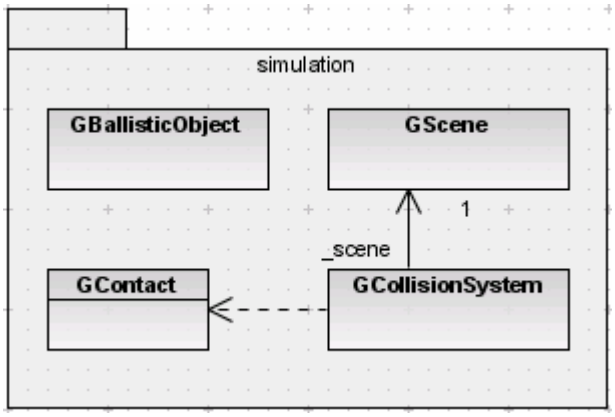


Figure 3.5 Diagramme de classes du paquetage *Simulation*.

La figure 3.5 illustre les diverses classes contenues dans le paquetage *Simulation*. À se faire, la classe *GScene* contient tous les nœuds d'une scène à représenter graphiquement. La classe *GCollisionSystem*, a comme responsabilité la détection de collision entre nœuds d'une scène. Lorsqu'une collision survient, la

description de celle-ci (nœuds concernés, type de collision, point de contact) est créée afin de calculer la réponse à celle-ci, sous forme d'impulsion, à appliquer aux corps en contact.

Finalement, le paquetage *Renderer*, représenté par la figure 3.6, expose les niveaux d'abstractions créés afin d'interface divers systèmes de rendus. Comme on peut le voir actuellement, le moteur physique supporte le rendu par OpenGL™ ainsi que par AWT, standard de Java™.

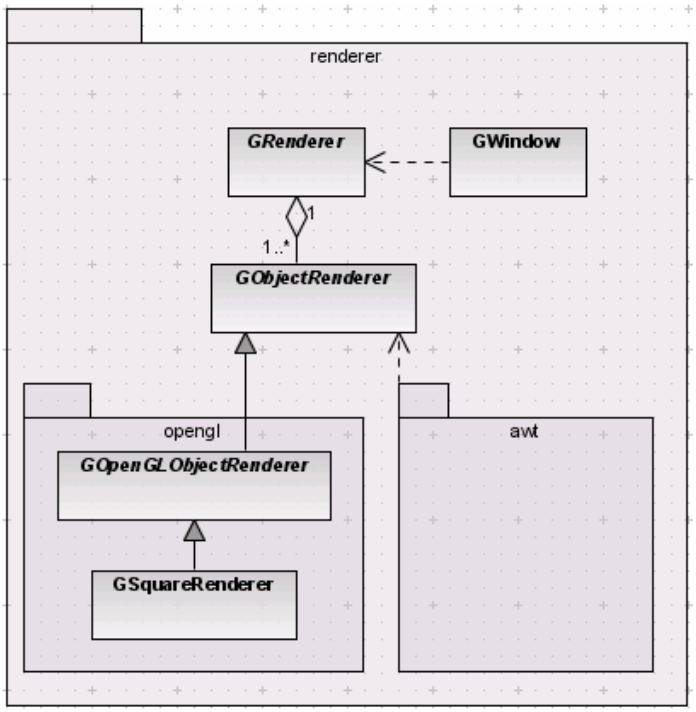


Figure 3.6 Diagramme de classes du paquetage *Renderer*.

La classe *GRenderer* est une classe qui définit le comportement à suivre afin de produire une scène à l'écran. La scène est rendue dans une fenêtre, implémentée par *GWindow*. Le *GRenderer* contient plusieurs objets de type *GObjectRenderer*, classe qui permet de spécifier la méthode par laquelle un objet de type GRenderableObject sera rendu à l'écran selon le type de *GRenderer*. Comme on peut le voir, le sous-paquetage OpenGL, contient une série de classe permettant de spécifier les méthodes de la norme OpenGL (apportées par JOGL) afin de rendre à l'écran les objets sous OpenGL™. Ainsi, la classe *GSquareRenderer* (permet le rendu d'un carré) est une spécialisation de *GOpenObjectRenderer* qui généralise le rendu d'objets sous OpenGL™.

Observons maintenant une vue d'ensemble de l'architecture afin de représenter les liens globaux entre classes et expliquer le comportement du moteur physique lors d'une simulation d'un cas.

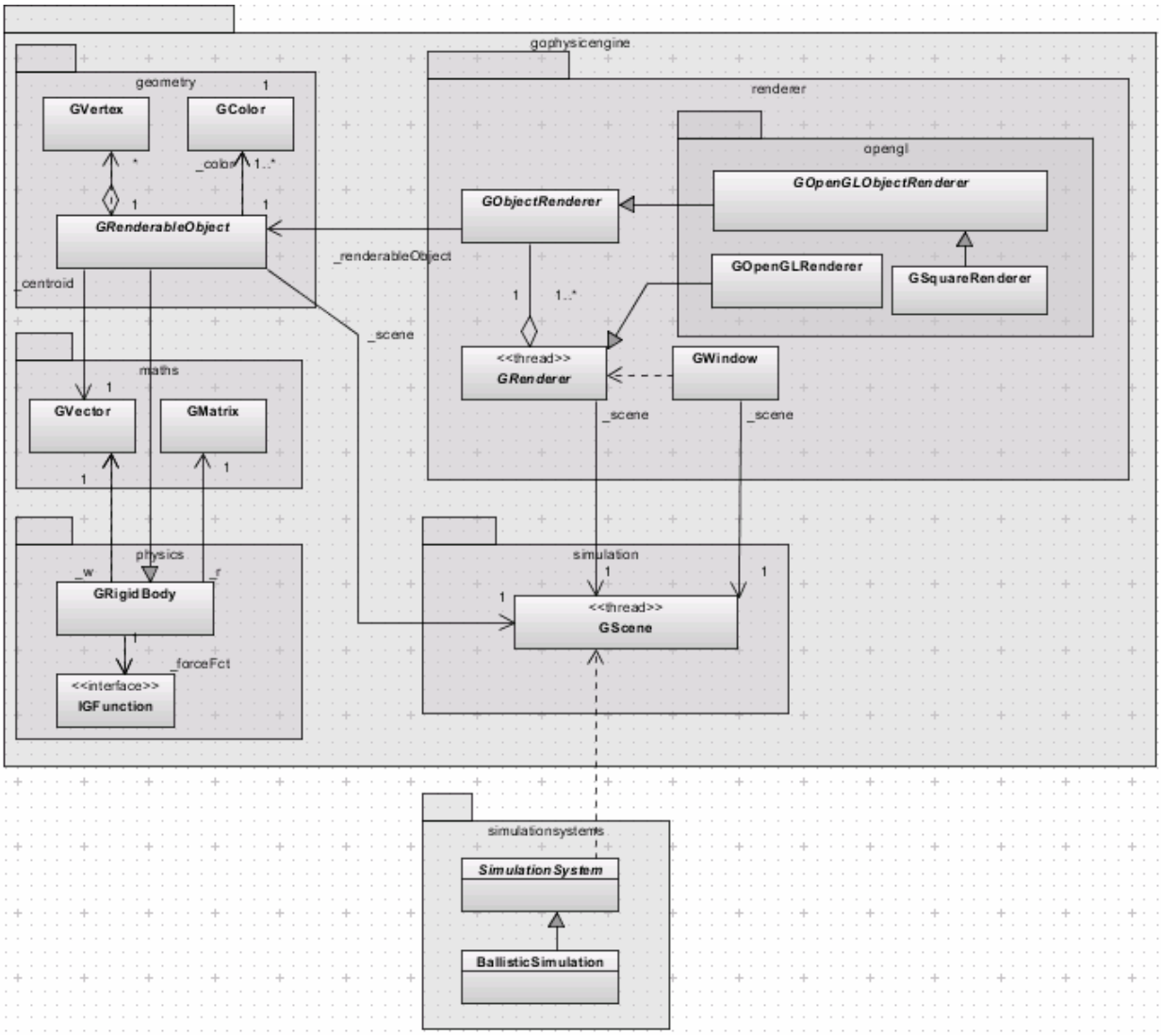


Figure 3.7 Diagramme de classes d'ensemble.

Chaque système de simulation (*SimulationSystem*) possède son nombre de corps rigides (*GRigidBody*) et la description de leur comportement (par l'entremise de *IGFunction*). Si un objet de la simulation doit être rendu à l'écran, il est automatiquement défini comme étant un *GRenderableObject*, qui contient de l'information sur son apparence (couleur : *GColor* et apparence : tableau de *GVertex*). Par la suite, chaque objet qui doit être rendu est à une méthode utilisée pour le rendu (classes héritant de *GObjectRenderer* : manière de dessiner l'objet compte tenu du système de rendu définit). Chaque objet à dessiner à l'écran est contenu dans un graphe de scène, *GScene*. Cette classe est un fil d'exécution où à chaque itération, ou pas d'intégration, elle demande aux objets de mettre à jour leurs états physiques (position, vitesse, orientation, etc.). En même temps, le système de rendu graphique *GRenderer*, opéré par un fil d'exécution à l'interne (non défini sur le diagramme), met à jour l'affichage en considérant les états physiques et l'allure graphique de tous les nœuds de la scène sur le canevas d'affichage inclut dans la classe *GWindow*.

4. PRÉALABLES

Cette section propose une courte discussion sur les éléments physiques et mathématiques abordés dans le projet. Elle consiste premièrement à familiariser préalablement le lecteur à acquérir certaines notions de bases concernant les mathématiques (algèbre linéaire) et la physique, afin de lui permettre de mieux saisir les différentes explications entourant les choix architecturaux du moteur physique et les discussions concernant les cas de simulation. D'autre part, elle agit à titre de référence pour le lecteur ayant déjà une base avec les concepts présentés.

4.1 Mathématique

4.1.1 Polygone convexe

« Un polygone (du grec ancien *polus,* nombreux, et *gónia,* angles) est une figure géométrique fermée, formée d'une suite de segments, chacun d'entre eux partageant une extrémité avec le suivant, délimitant ainsi un contour polygonal fermé. » [13]

Due à certaines limitations provenant d'algorithmes employés lors des cas de simulation, l'utilisation de polygones convexes a été de mise. À titre indicatif, la convexité d'un polygone peut être vérifiée si et seulement si toutes les diagonales, c'est-à-dire tout segment de droite qui joint deux sommets non consécutifs, sont entièrement à l'intérieur de l'espace délimité par le polygone.

La figure 4.1 nous présente deux polygones dont le premier est convexe et le second concave. L'exemple s'appuie sur la définition présentée.

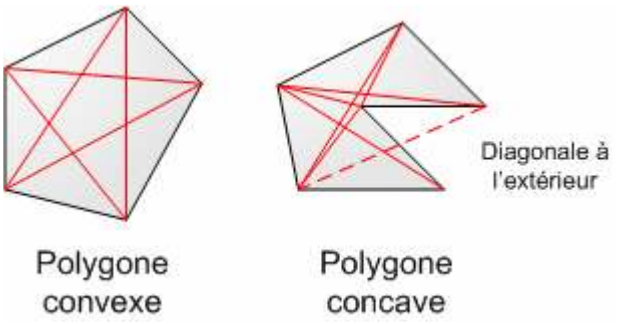


Figure 4.1 Convexité et concavité de polygones.

Ainsi, comme le présente le deuxième polygone, la concavité est obtenue par la création d'une diagonale tracée à l'extérieur du périmètre définit par le polygone.

4.1.2 Vecteur

Un vecteur désigne une quantité (telles une vitesse ou une force) qui à la fois possède une grandeur, direction et un sens. Sa représentation spatiale se traduit par un segment orienté ou une flèche. La longueur de cette flèche représente la grandeur du vecteur, la droite qui la porte, indique la direction et la flèche, le sens du vecteur.

La majorité des cas de simulation sont représentés sous forme vectorielle puisque plus naturellement attribuables au concept d'espace. Le vecteur $\vec{x}$ représentera ainsi la position de l'objet dans l'espace, $\vec{v}$ sa vitesse linéaire et $\vec{a}$ son accélération linéaire.

Il est souhaitable que le lecteur prenne le temps de bien saisir les opérations vectorielles de base (addition, soustraction, projection, produit vectoriel et scalaire) afin de comprendre la totalité des explications et preuves démontrées dans les sections subséquentes.

4.1.3 Matrice de rotation

Surnommée aussi rotation vectorielle, la matrice de rotation est définie comme étant un vecteur $\vec{v}' = [x'; y']$ étant le résultat d'une rotation de θ autour d'un axe. Elle est exprimée comme étant le produit entre le vecteur original $\vec{v}$ par la matrice de transformation [13]. À titre indicatif, nous allons nous contenter uniquement de définir la matrice de rotation dans le plan puisque le moteur physique n'opère qu'en deux (2) dimensions.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Grâce à la matrice de rotation, il sera dès lors possible de situer un objet dans un espace en deux (2) dimensions.

4.2 Physique

4.2.1 Corps rigide

Afin de définir correctement ce qu'est un corps rigide, la notion de particule doit être couverte préalablement. Selon Eberly [3], une particule est en faite une très petite quantité de matière dont on a la présomption qu'elle occupe un point unique dans l'espace. La particule est une représentation simpliste des corps rigides puisqu'elle ne possède pas, à l'instar de ceux-ci, de moment angulaire. Un corps rigide est une combinaison d'une large quantité de particules occupant des positions fixes relatives à elles même. La région qui le compose est occupée par sa masse.

Un requis essentiel qui sera présenté dans notre système stipule que des corps rigides ne s'interpénètrent jamais.

4.2.2 Notions de position, vitesse et accélération vectorielles

On peut définir la position du centre de masse d'un corps rigide, ou d'une particule, par un vecteur $\vec{x}(t)$ variant avec le temps *t*. On peut calculer la vitesse linéaire instantanée par la dérivée $\frac{d\vec{x}(t)}{dt} = \vec{v}(t)$ et son accélération linéaire par une double dérivée de la position ou une simple dérivée de la vitesse, $\frac{d^2\vec{x}(t)}{dt^2} = \frac{d\vec{v}(t)}{dt} = \vec{a}(t)$.

4.2.3 Moment linéaire

Par définition le moment linéaire, ou quantité de mouvement, est défini comme étant la masse du corps rigide multiplié par sa vitesse linéaire. Il prend la forme suivante

$$p(t) = m\frac{d\vec{x}(t)}{dt}$$

4.2.4 Moment angulaire

Le moment angulaire est analogue au moment linéaire dans le cas des rotations. Effectivement, l'analogie se développe à partir du fait que la masse *m* définie dans l'équation précédente possède son équivalent appelé inertie *I* dans le cas d'une rotation. Afin de définir le moment angulaire, la notion de moment de force τ doit être exposée.

$$\vec{\tau} = \vec{R} \times \vec{F}$$

Qui représente le produit croisé entre le vecteur *R*, déterminant le point à partir du centroïde d'un polygone convexe où la force *F* sera appliquée. Le moment de force détermine dans quel sens le corps rigide évoluera.

Le moment angulaire *L* quant à lui est fonction de

$$\vec{L} - I\vec{\omega}$$

puis sa dérivée est directement liée au moment de force

$$\frac{d\vec{L}}{dt} = I\frac{d\vec{\omega}}{dt} = \vec{\tau} = \vec{R} \times \vec{F}$$

4.2.5 Inertie

L'inertie est présentée comme étant la résistance attribuée à un objet pour occasionner une rotation d'un corps autour d'un axe. Intuitivement, cela se produit lorsque l'objet possède une masse importante et que la distance par rapport à l'axe est élevée. Des résultats empiriques ont démontré que l'inertie était en relation avec mr^2, où *m* est la masse de l'objet et *r* sa distance par rapport à l'axe.

Les cas de simulation présentés comportent des calculs d'inertie. Ceux-ci sont liés avec la forme de l'objet (rectangulaire). On peut définir l'inertie des objets simulés par l'équation suivante.

$$I = \frac{m(b^2 + h^2)}{12}$$

Où *b* et *h* sont respectivement la base et la hauteur de l'objet (rectangulaire).

4.3 Méthodes d'intégration numérique

La base d'un moteur physique ne repose pas uniquement sur la modélisation physique proprement dite, mais aussi de la résolution numérique des systèmes d'équations différentielles en découlant. Cette section se veut être un survol des motivations et conceptions des méthodes numériques en y exposant aussi les critères de stabilité. À se faire, deux méthodes d'intégration (Euler et Runge-Kutta) seront brièvement expliquées.

Le choix d'une méthode numérique sur une autre dépend essentiellement de l'élaboration d'un compromis entre stabilité, acuité et vitesse. La théorie exposée ci-dessous est largement inspirée des travaux d'Eberly [3]. Ainsi, une méthode possédant une acuité et stabilité exemplaire sera malheureusement limitée par sa vitesse d'exécution. Cela est dû au fait que l'approximation faite d'une fonction différentielle arbitraire sera plus longue à calculer.

Afin d'expliquer le principe de fonctionnement des méthodes numériques, posons l'équation différentielle du premier ordre suivante, accompagnée de conditions initiales.

$$x' = f(t,x) \qquad t \geq t_0, x(t_0) = x_0$$

L'équation différentielle précédente décrit le comportement d'un système arbitraire qui devra être résolu par un intégrateur numérique. Les méthodes numériques reposent presque toutes sur le développement d'un polynôme approximant la fonction différentielle sur une zone bornée. Ce polynôme est développé avec le théorème de Taylor duquel engendre le polynôme de Taylor d'allure suivante.

$$x(t_1) = \sum_{k=0}^{n} \frac{x^{(k)}(t_0)}{k!}(t_1 - t_0)^k + \frac{x^{(n+1)}(t)}{(n+1)!}(t_1 - t_0)^{n+1}$$

Ce théorème stipule qu'on peut approximer la valeur de *x'(t)* par un polynôme sur une zone bornée de la fonction *x(t)*.

Cependant, comme toute méthode numérique, un facteur clé à considérer est les erreurs d'arrondissement produites par l'approximation de fonctions différentielles. Les erreurs d'approximation *E* provenant du polynôme de Taylor sont de la forme

$$E = kh^n$$

Où *k* est une constante,
h le pas d'intégration et
n le degré du polynôme de Taylor employé.

Le degré du polynôme de Taylor est en étroite relation avec le temps de calcul nécessaire pour l'approximation; de la même manière qu'il l'est pour l'acuité du résultat.

La méthode d'Euler, qui est aussi la plus simple des méthodes numériques, doit sa popularité à sa facilité d'utilisation. Malheureusement, elle est dotée d'erreur d'arrondissement important qui la rend instable dans certains cas puisqu'elle est seulement du premier ordre. Donc le degré *n = 1*.

Ainsi, pour un pas d'intégration $h = 0.1$ et $n = 1$, l'erreur d'approximation $E = 0.1k$. Le fait de choisir un degré plus élevé rendra, comme nous allons le voir, pour une même constante k, automatiquement l'approximation plus précise. Si $h = 0.1$ et $n = 4$, alors $E = 0.0001k$.

Le fait de décrémenter le pas d'intégration h, pour un polynôme de degré $n=1$ produit un résultat analogue, mais nécessite beaucoup plus d'itération pour approximer la fonction jusqu'au temps final ce qui entraîne un temps de calcul plus élevé.

Les méthodes de Taylor d'ordres élevés connaissent moins d'imprécision puisque leurs erreurs d'arrondissement se trouvent largement diminuées. Ainsi, la méthode de Runge-Kutta présentement utilisée dans ce projet, utilise des polynômes de Taylor de degrés plus élevés ce qui rend le processus un peu moins rapide, mais produit des résultats plus précis. Comparons maintenant la stabilité d'une méthode numérique (en lien avec son E, son erreur d'approximation) avec la stabilité *physique* d'un cas de simulation.

La stabilité d'une méthode numérique est en lien direct avec la stabilité *physique* d'un système d'équations différentielles. On dira que la solution analytique produite $\varphi(t)$ est stable si toutes les solutions numériques $\psi(t)$ produisent des valeurs proches en tout temps t. Cela engendre un énoncé concernant la dépendance continuelle d'une solution en relation avec les conditions initiales proposées. Ainsi si $x'(t)=f(t,x)$ est une équation différentielle du système et que $x(t_0)=x_0$ est sa condition initiale, la solution sera notée par x(t;x0). Si on compare les solutions $x(t;x_0+\delta)$ et $x(t;x_0)$, où δ est un vecteur non nul de magnitude très petite, on veut s'assurer que $\left| x(t_0; x_0+\delta)-x(t_0;x_0) \right| = \left| (x_0+\delta)-x_0 \right| = \left| \delta \right|$, qui possède une valeur très petite. La question à élucider est à savoir si la différence restera constante dans le temps. Ce concept est appelé, par Eberly [3], la *stabilité physique* d'un système.

La stabilité physique dépend aussi de la stabilité numérique des méthodes. Pour cela, l'intégrateur est jugé sur trois concepts soit, la *consistance*, la *convergence* et la *stabilité*. Pour une méthode à pas simple, telle que Euler ou Runge-Kutta, la *consistance* est définit étant

$$\lim_{h \to 0} \left[\max_{1 \le i \le n} \left| \tau_i \right| \right] = 0$$

Où τ_i représente l'erreur d'approximation locale à la $i^{ème}$ itération. Intuitivement on dit donc que pour de petits pas, à un temps t quelconque, l'erreur d'approximation locale devrait être très petite.

La deuxième définition qualifie de convergente une méthode numérique si

$$\lim_{h \to 0} \left[\max_{1 \le i \le n} \left| x(t_i) - y_i \right| \right] = 0$$

Cela stipule que pour de petits pas d'intégration, l'erreur maximale en tout temps t entre l'approximation y_i et la vraie solution $x(t_i)$ devrait être très petite.

Finalement, une méthode numérique est dite *stable*, si de petits changements aux conditions initiales des équations différentielles produisent de petits changements auprès des approximations. Formellement, si x_0 et x_1 sont deux valeurs de conditions initiales différentes et que y_i est l'approximation de $x(t_1,x_0)$ et que z_i est l'approximation de $x(t_i,x_1)$ alors pour tout $\varepsilon > 0$ et $\delta > 0$, suffisamment petits, $|y_i - z_1| < \varepsilon$ pour n'importe quel $|x_1 - x_0| < \delta$.

Même si les toutes les conditions précédemment énoncées sont respectées par Euler et Runge-Kutta (critères de Lipschitz), il n'en reste pas moins que toute la difficulté réside en le choix d'un pas d'intégration *h* produisant un résultat *stable*. Plusieurs études [3, 14], dont une autre démontrée ci-dessous, ont démontrées que le choix d'un pas quelconque sans analyse préalable produisait un résultat physique instable pour les méthodes d'intégration d'Euler (implicite et explicite) appuyant le choix de la méthode de Runge-Kutta, même en étant moins rapide, pour le moteur physique.

Hoffman [14] a démontré qu'en résolvant l'équation différentielle $\frac{dy}{dt} = -1000(y - (t+2)) + 1$ avec comme condition initiale y(0) = 1, le pas d'intégration h pouvait produire l'instabilité, tel que démontré par les deux figures suivantes[4].

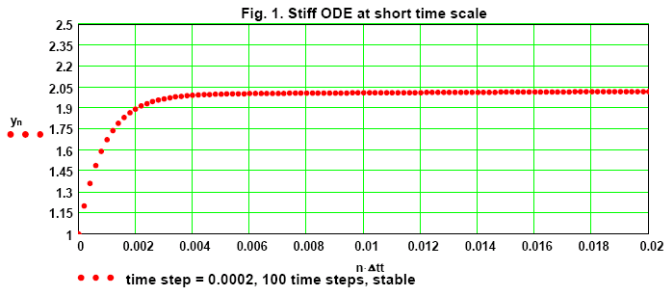


Figure 4.2 Stabilité d'un système avec un Euler

Figure 4.3 Instabilité de l'intégrateur d'Euler

Les prochaines sections démontreront justement le résultat numérique de cas de simulation formulé à partir de systèmes d'équations différentielles simples. La stabilité des résultats, en constante comparaison avec les solutions analytiques produites, montre l'efficacité de Runge-Kutta.

[4] Source : Hoffman [14], http://www2.latech.edu/~dmg/stiff_ODE_solution.PDF

5. CAS DE SIMULATION I : SIMULATION DU MOUVEMENT D'UN CORPS RIGIDE DANS UN ENVIRONNEMENT SANS CONTRAINTE

5.1 Description du cas

Le premier cas de simulation présenté est le plus trivial rencontré dans le projet : un système masse-ressort simulant un mouvement simple oscillatoire harmonique (*MHS pour mouvement harmonique simple*). L'implémentation de ce cas fût simpliste dû à l'élémentarité du modèle théorique décrivant le comportement du système. Ce premier cas de simulation a été contraint à plusieurs facteurs dans le but unique de tester adéquatement l'intégrateur numérique de Runge-Kutta ainsi que tout le système graphique permettant d'afficher l'état du MHS. À titre indicatif, précisons que le système présenté dans cette section n'expose pas l'interaction de corps rigides, mais plutôt de particules. Effectivement, la masse qui est sujette aux forces exercées par le ressort n'occasionne aucun mouvement angulaire. Rappelons que la réalisation d'un tel cas a eu comme but exclusif de tester adéquatement les éléments calculatoires du moteur physique.

Pour la simulation du système masse-ressort, les contraintes suivantes ont été appliquées :

- Mouvement en une (1) dimension
- Aucune force contraignante telle que la friction, la gravité
- Aucune gestion de collisions

Les contraintes et l'implémentation du mouvement de la masse par rapport au ressort ont été établies selon la deuxième loi de Newton,

$$F = m\frac{dv}{dt} \qquad (5.1)$$

La section suivante présentera une description plus détaillée du comportement de la masse et du ressort selon les lois de Newton.

5.2 Présentation du modèle théorique

Le système masse-ressort est qualifié de mouvement oscillatoire simple puisque le mouvement de la masse est périodique et est en relation constante avec la position d'équilibre du système; position où aucune force nette n'interagit et où la vitesse de la masse est maximale. Voici une première figure faisant état de la logique ainsi que du modèle mathématique qui émerge du système.

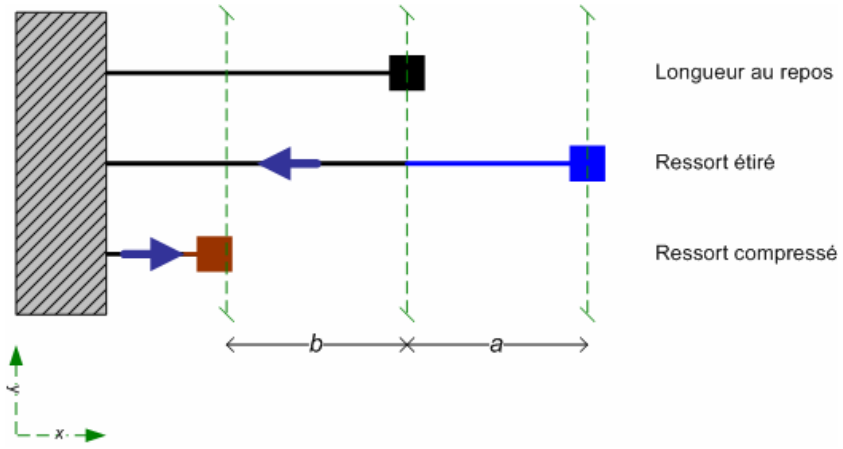


Figure 5.1 Les trois (3) états typiques d'un système masse ressort.

Selon la figure 5.1, trois cas de bases existent dans ce système. Premièrement, le cas du ressort au repos, où aucune force n'est exercée sur la masse. Ce cas se produit lorsque la position de la masse concorde avec celle de la longueur du ressort au repos. Dans cette position, la *vitesse est maximale* puisqu'elle n'est sujette à aucune force. Le deuxième cas surgit lorsque la longueur au repos du ressort est dépassée par la masse, c'est-à-dire lorsque le ressort est *étiré*. L'étirement du ressort, créé une force de restauration sur la masse, engendrée par le ressort, qui essaie de ramener celle-ci vers l'état d'équilibre caractérisé par la concordance de la position de la masse à la longueur du ressort (premier cas). On peut noter par *a*, l'étirement produit par la masse sur le ressort. Finalement, le dernier cas survient lorsque la masse en mouvement dépasse le point d'équilibre et comprime le ressort. La compression du ressort est obtenue lorsque la position de la masse est inférieure à la longueur du ressort au repos ce qui en résulte par une force de répulsion qui pousse la masse vers le point d'équilibre. La force de répulsion est proportionnelle à la compression notée par *b* sur la figure.

La force résultante du système masse-ressort peut s'exprimer à partir de la deuxième loi de Newton à travers les trois (3) cas présentés de façons suivantes :

$$F = m\frac{d^2\psi}{dt^2} = -k\psi \qquad (5.2)$$

Où F est la force résultante agissant sur la masse (N)
m est la masse en (kg)
ψ est l'élongation ou la compression du ressort (m)
k est la constante de rappel du ressort (N/m)

Ainsi, ψ prend la valeur de

$$\psi = x - l_0$$

Où x est la position courante de la masse
l_0 est la longueur du ressort au repos

Voici un bref résumé des trois cas décrits précédemment en considérant la modélisation de la force du ressort.

Cas	**Élongation**	**Force résultante**
Ressort au repos	$x = l_0$ $\psi = x - l_0 = 0$	$F = -k\psi,\ F = -k(0) = 0$ Aucune force n'est appliquée à ce moment sur le ressort.
Ressort étiré	$x > l_0$ $\psi : x - l_0 > 0$ $\psi > 0$	$F = -k\psi, F < 0$ La force résultante est négative (force de restauration). Elle cherche à attirer la masse vers le point d'équilibre.
Ressort compressé	$x < l_0$ $\psi : x - l_0 < 0$ $\psi < 0$	$F = -k\psi,\ F > 0$ La force résultante est positive et cherche à repousser la masse vers la position d'équilibre.

Tableau 5.1 Représentation mathématique des trois (3) cas d'un système masse ressort.

À titre de rappel, exposons une seconde fois l'équation différentielle décrivant le comportement du système :

$$F = m\frac{d^2\psi}{dt^2} = -k\psi \qquad (5.3)$$

La solution de cette *équation différentielle* est de la forme $x(t) = A\cos(\omega t + \phi)$

Où A représente l'amplitude du mouvement autour du point d'équilibre
ω est considéré comme la fréquence du système
φ est le déphasage du système

Rappelons que

$\omega = \frac{2\pi}{T} = \sqrt{\frac{k}{m}}$ où T est la période, k la constante de rappel et m la masse.

5.3 Analyse des données empiriques et théoriques

La présentation des résultats portera sur les conditions initiales définies ci-dessous qui ont été au préalable implémentées dans le moteur physique. Voici la liste des conditions initiales :

À $t = 0$, $v_0 = 0$ (v_0 étant la vitesse initiale de la masse)

La longueur au repos du ressort est de $l_0 = 50$ m
La position de départ de la masse se trouve à $x_0 = 100$ m
La force du ressort $k = 50$ N/m
La masse m du poids lié au ressort est de 20 kg

Selon les équations présentées, trouvons les valeurs d'*amplitude*, de *fréquence* et de *déphasage* afin de construire une équation permettant de calculer les valeurs théoriques pour un tel système.

$x(t) = A\cos(\omega t + \varphi)$ $x(t=0) = A\cos(\omega(0) + \varphi)$ $x(t=0) = A\cos(\varphi)$ Sachant que le ressort est étiré à t_0 $l_0 - x_0 = A\cos(\varphi)$ $50 - 100 = A\cos(\varphi)$ $-50 = A\cos(\varphi)$ Dérivons l'équation de position pour créer une 2e équation	$\frac{d}{dt} A\cos(\omega t + \varphi)$ $v(t) = -\omega A\sin(\omega t + \varphi)$ $v(t=0) = -\omega A\sin(\omega(0) + \varphi)$ $0 = -\omega A\sin(\varphi)$ Sachant que $\tan(\varphi) = \frac{\sin(\varphi)}{\cos(\varphi)}$ $\frac{0}{-50} = \frac{-\omega A\sin(\varphi)}{A\cos(\varphi)}$	$\tan(\varphi) = \frac{0}{(-50 \times -\omega)}$ $\arctan(\frac{0}{50\omega}) = \varphi = 0$ Remplaçons φ dans l'équation $-50 = A\cos(\varphi)$ On trouve que $A = -50$, on rejette cette solution pour $\varphi = 0$ et on lui ajoute π pour que $A > 0$.

L'équation résultante décrivant un système masse ressort avec les conditions présentées est donc

$$x(t) = 50\cos(\frac{\sqrt{10}}{2}t + \pi) \qquad (5.4)$$

Les mêmes conditions ont été implémentées dans le moteur physique afin de produire numériquement un comportement analogue. Voici les résultats empiriques et théoriques obtenus pour $t = [0, 57.2]$ s.

La méthode d'intégration numérique employée pour ce cas de simulation a été Runge-Kutta d'ordre 4. Le pas d'intégration a été fixé à 0.2 s.

La figure 5.2 ci-dessous représente la position de la masse selon les conditions fixées précédemment en fonction du temps.

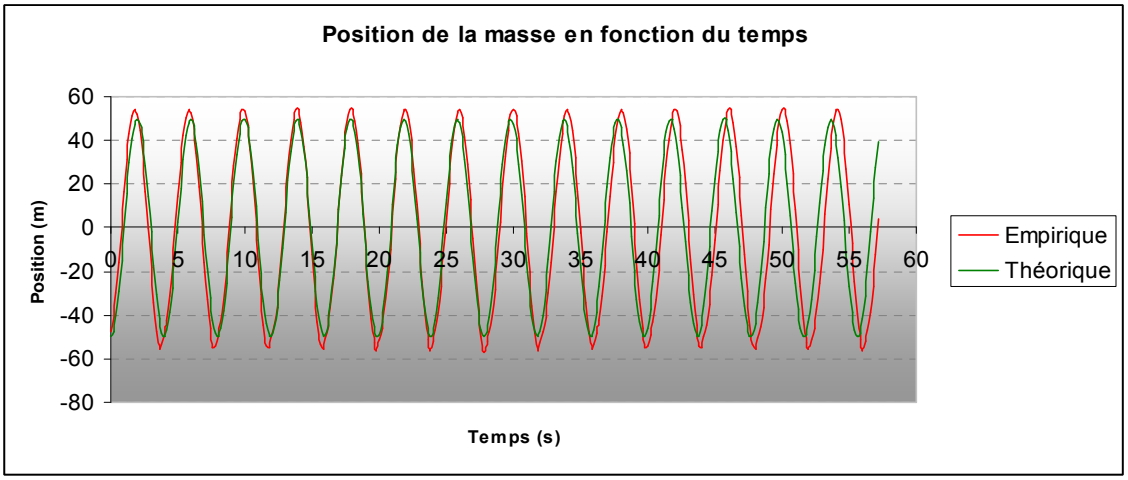


Figure 5.2 Graphique représentation la comparaison entre le modèle théorique et empirique par intégration selon Runge-Kutta.

Comme on peut le voir, l'intégrateur numérique de Runge-Kutta donne de bons résultats même si on assiste à un léger déphasage lorsque $t > 40$ s. Ce déphase est occasionner par l'instabilité de la méthode numérique, sujet couvert dans la section 4.3, par le choix du pas d'intégration. La figure 5.3 illustre la représentation de la simulation telle qu'exécutée à partir de l'implémentation en Java™ du système masse-ressort.

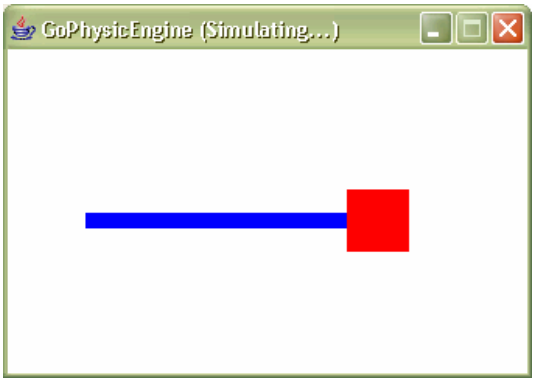


Figure 5.3 Interface utilisateur de la simulation du système masse-ressort.

La simulation s'est très bien comportée sur un système doté d'un AMD Athlon™ Thunderbird™ 1.4Ghz en offrant un débit facilement atteint de cinquante (50) trames par secondes. De plus, la demande au niveau du temps de calcul du processeur[5] a été très basse puisqu'elle a avoisinée les deux (2) pour cent en moyenne.

Note : La demande au niveau de temps de calcul comprend ici l'affichage ainsi que le calcul des états physique du système de simulation.

[5] Simulation exécutée sous Microsoft Windows™ XP Pro (Service Pack 2)

La figure 5.4 confirme cette statistique.

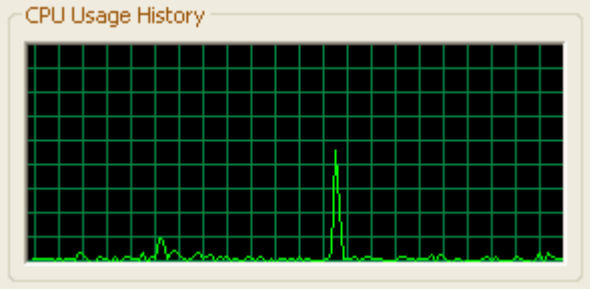


Figure 5.4 Utilisation du processeur en pourcentage durant la simulation du système masse-ressort.

5.4 Discussion

Malgré la faible instabilité de la méthode numérique Runge-Kutta, il a été démontré[6] que cette méthode était préférée à celle d'Euler qui, pour un système semblable, était rapidement divergente et instable. Même si l'implémentation d'un tel système est triviale, elle a néanmoins permis de prouver l'efficacité du système physique et la stabilité de la méthode numérique pour résoudre les équations différentielles. De plus, son intégration au système graphique de Java™ a donné l'occasion d'ajouter un module graphique permettant d'afficher les objets en deux (2) dimensions.

[6] Eberly nous démontre dans son livre *Game Physics*, l'instabilité de la méthode d'Euler pour un système représentant le mouvement du pendule de Foucault (p.494).

6. CAS DE SIMULATION II : SIMULATION DU MOUVEMENT D'UN CORPS RIGIDE AVEC FORCES CONTRAIGNANTES

6.1 Description du cas

Le deuxième cas présenté est beaucoup plus complexe que le précédent, dû en premier lieu à l'intégration de mouvement en deux (2) dimensions, en considérant des forces frictionnelles et finalement en prenant en compte le mouvement angulaire d'un corps rigide. Pour ce faire, la réalisation d'un cas simulant la trajectoire d'un projectile a été conçue. La conception de ce cas à donner l'occasion de parfaire le système d'affichage graphique en incorporant la fonctionnalité de rotation permettant d'émuler le mouvement angulaire du projectile. De plus, l'intégration de forces dissipatives liées aussi bien au mouvement linéaire qu'angulaire a nécessité un effort soutenu afin de représenter le plus fidèlement la réalité.

Le cas se présente comme suit :

1. Positionnement d'un objet en position (x_0, y_0)
2. Au temps $t_0=0$, on applique une force sur l'objet le projetant dans l'espace.
3. Au même moment ($t_0=0$) et ce, pendant un court instant ($\Delta t << 1$), on applique une force impulsive sur l'objet permettant de stimuler un mouvement angulaire.
4. Au cours de la durée de la simulation, et ce, jusqu'au temps t_f, soit le temps de la collision de l'objet avec le sol, on calcule la trajectoire linéaire et le mouvement angulaire du corps en considérant les forces dissipatrices.

En résumé, voici les divers éléments implémentés dans cette simulation :

- Mouvement en deux (2) dimensions
- Force frictionnelle de l'air (s'appliquant au mouvement linéaire et angulaire)
- Force gravitationnelle
- Implémentation du moment angulaire
- Force impulsive stimulant le moment angulaire du corps rigide
- Afin de simplifier la simulation, les contraintes suivantes ont été appliquées :
- Force gravitationnelle constante, n'est pas altérée par la hauteur

Afin de calculer la trajectoire du projectile, la simulation s'est basée sur la deuxième loi de Newton.

$$\vec{F} = m\frac{d\vec{v}(t)}{dt}$$

La section suivante présentera en détail l'application d'une telle loi en considérant les forces dissipatives.

6.2 Présentation du modèle théorique

La trajectoire d'un projectile est influencée principalement par une vitesse initiale (v_0). Les composantes de cette vitesse sont établies à partir de l'angle d'azimut α qu'elle forme avec l'axe des abscisses. Ces composantes influenceront directement la portée du projectile ainsi que sa hauteur maximale.

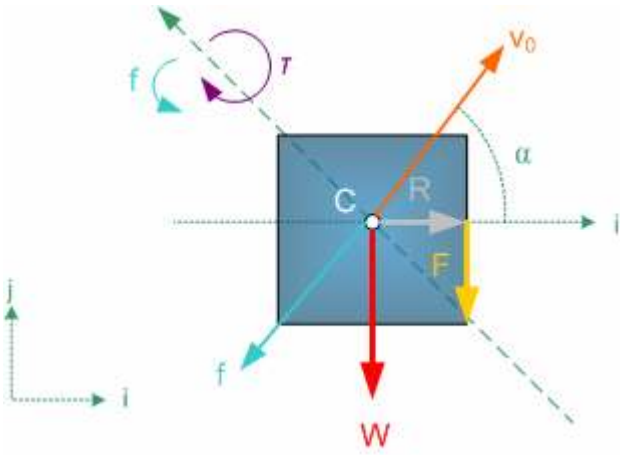


Figure 6.1 Représentation des forces sur le projectile.

Sur la figure 6.1, observons les principales forces exercées sur le corps rigide. Principalement, deux (2) forces majeures sont présentes. La première étant le poids de l'objet combiné avec la gravité ($\vec{w}$) et la deuxième étant le moment de force τ (appelé moment de force ou torque en anglais) qui produit un mouvement angulaire au corps rigide. Regardons maintenant à quel moment interviennent les forces lors du mouvement de trajectoire et du mouvement angulaire.

Mouvement linéaire

Le mouvement d'un projectile dans un espace en deux (2) dimensions est d'une part uniforme en direction de l'axe des abscisses et uniformément accéléré en direction de l'axe des ordonnées. Ce qui veut dire, qu'en y application la deuxième loi de Newton, $\sum \vec{F} = m\frac{d\vec{v}(t)}{dt}$, on se retrouve avec

$$\vec{F}_x = m\frac{d\vec{v}(t)}{dt} \qquad (6.1)$$

la vitesse étant constante,

$$\vec{F}_x = 0 \qquad (6.2)$$

En prenant maintenant l'axe des ordonnées et en ne considérant que le poids et la gravité,

$$\vec{F}_y = -mg \qquad (6.3)$$

À titre indicatif, rappelons que m est le poids de l'objet et g la constante de gravité.

La force de friction se présente comme étant de force contraire au mouvement (tel qu'illustré à la figure 6.1). La friction f varie selon le carré de la vitesse. Dans notre cas, définissons *D* comme étant le coefficient de friction. Ce coefficient (*D*) varie selon la densité de l'air (ρ), de l'aire de l'objet qui entre en contact (*A*) et finalement un coefficient de contact (*C*) qui varie de 0.2 à 1.0 selon la forme de l'objet. Ainsi,

$$D = \frac{\rho C A}{2} \tag{6.4}$$

Modélisons le fait que la friction *f* varie au carré de la vitesse.

$$f = Dv^2 \tag{6.5}$$

Où

$$v^2 = v_x^2 + v_y^2 \tag{6.6}$$

La direction de la force de friction *f* est toujours de sens contraire à la vitesse.

$$f_x = -Dvv_x \tag{6.7}$$

$$f_y = -Dvv_y \tag{6.8}$$

Si on reprend les équations 6.2 et 6.3 afin d'y intégrer la gravité et les forces de friction, on se retrouve avec

$$\sum \vec{F}_x = -Dvv_x \tag{6.9}$$

$$\sum \vec{F}_y = -mg - Dvv_y \tag{6.10}$$

Puisque l'intégrateur numérique utilisé dans le moteur physique calcule les forces engendrées en fonction du moment linéaire $\vec{P}$, définissons ce qu'est le moment linéaire et lions les équations 6.9 et 6.10 à celui-ci.

$$\vec{P} = m\vec{v} \tag{6.11}$$

Le moment linéaire est défini comme étant une tendance qu'un corps rigide possède à rester positionné en ligne droite en absence de forces externes. En prenant en considération la deuxième loi de Newton, la dérivée du moment linéaire peut s'exprimer de la façon suivante.

$$\frac{d\vec{P}}{dt} = m\frac{d\vec{v}}{dt} \tag{6.12}$$

L'équation 6.12 peut ainsi prendre la forme suivante qui permet le lien direct avec les équations 6.9 et 6.10.

$$\frac{d\vec{P}}{dt} = m\frac{d\vec{v}}{dt} = \vec{F} \qquad (6.13)$$

Séparons le moment linéaire en fonction des axes.

$$\frac{d\vec{P}_x}{dt} = -D|\vec{v}|v_x \qquad (6.14)$$

$$\frac{d\vec{P}_y}{dt} = -D|\vec{v}|v_y \qquad (6.15)$$

Mouvement angulaire

En ce qui concerne le mouvement angulaire, nous devons considérer deux (2) forces majeures. Premièrement le moment de force (ou torque en anglais) puis la friction qui contre le mouvement angulaire. La friction s'applique exactement de la même façon qu'avec le mouvement linéaire, c'est-à-dire qu'elle varie au carré de la vitesse.

Le moment de force τ est défini de manière suivante :

$$\vec{\tau} = \vec{R} \times \vec{F} \qquad (6.16)$$

Qui représente le produit croisé entre le vecteur *R*, déterminant le point à partir du centroïde où la force *F* sera appliquée. Le moment de force détermine dans quel sens le corps rigide évoluera.

Afin d'incorporer la friction au moment de force, nous avons besoin de lier la vitesse angulaire *w* à l'équation 6.16. Pour ce faire, mentionnons que le moment angulaire *L* est fonction de

$$\vec{L} - I\vec{\omega} \qquad (6.17)$$

puis que sa dérivée est directement liée au moment de force

$$\frac{d\vec{L}}{dt} = I\frac{d\vec{\omega}}{dt} = \vec{\tau} = \vec{R} \times \vec{F} \qquad (6.18)$$

Où I représente l'inertie du corps rigide (tenseur d'inertie sous forme de matrice dans un cas en trois (3) dimensions ou scalaire en deux (2) dimensions). L'inertie intervient dans notre cas puisqu'elle témoigne de la tendance du corps à continuer de tourner sur lui-même. Pour les cas de simulations, le choix s'est arrêté sur une forme rectangulaire. Le moment d'inertie s'exprime donc de la manière suivante.

$$I = \frac{m(b^2 + h^2)}{12} \qquad (6.19)$$

Où *b* est considéré comme étant la longueur de la base du rectangle et *h* sa hauteur.

Concrètement, dans la conception du moteur physique, la dérivée s'exprime à travers un intervalle de temps très petit. L'équation 6.18 peut ainsi prendre la forme suivante.

$$\frac{\Delta \vec{L}}{dt} = I \frac{\Delta \vec{\omega}}{dt} = \tau \qquad (6.20)$$

En rappelant que la friction de l'air varie au carré avec la vitesse, intégrons-la à l'équation précédente.

$$\frac{\Delta \vec{L}}{dt} = I \frac{\Delta \vec{\omega}}{dt} - D\left|\vec{\omega}\right|^2 = \tau \qquad (6.21)$$

Analysons maintenant les données issues du moteur physique afin d'y faire une comparaison avec le modèle théorique.

6.3 Analyse des données empiriques et théoriques

Analysons premièrement le comportement du projectile dans un milieu sans friction. Grâce aux équations 6.2 et 6.3, il est possible de représenter la trajectoire à travers le temps.

Voici les données initiales que nous avons prises pour exécuter ce test.

Données initiales	
Masse (m)	0.3 kg
Vitesse initiale (v_0)	60 m/s
Angle d'attaque (α)	45°
Constante de gravité (g)	9,80655 m/s^2

Tableau 6.1 Données initiales pour la trajectoire sans friction.

La figure suivante démontre les trajectoires empirique et théorique exécutées entre les temps [0, 8.7] s.

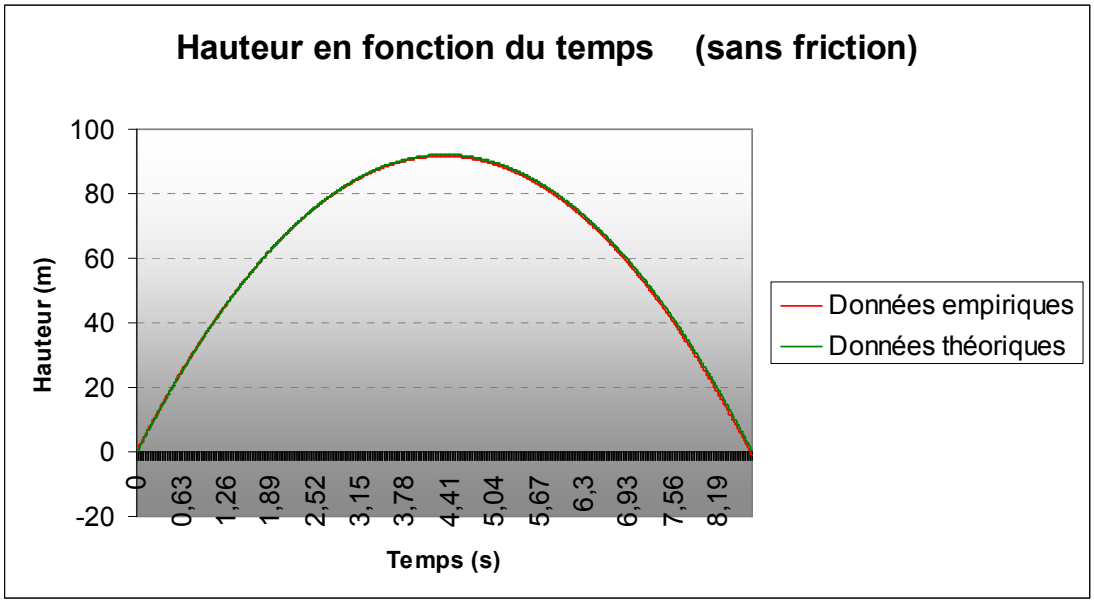


Figure 6.2 Modèle théorique et modèle empirique, sans friction.

Comme on peut le voir, les résultats se rapprochent énormément entre le modèle théorique et les données empiriques produites par le moteur physique. En moyenne, les écarts entre les données théoriques et empiriques varient de 1,004695%, ce qui est négligeable puisque l'approximation se fait à moins d'un mètre près.

La figure suivante propose le mouvement angulaire donné au corps solide. Préalablement, calculons l'inertie de notre objet.

$$I = \frac{m(b^2 + h^2)}{12}$$

$$I = \frac{0.3(0.1^2 + 0.1^2)}{12} = 0.0005$$

Maintenant, afin de comparer les valeurs théoriques et empiriques de la vitesse angulaire ω, rappelons que la relation avec le moment de force τ.

$$\frac{d\vec{L}}{dt} = I\frac{d\vec{\omega}}{dt} = \vec{\tau} = \vec{R} \times \vec{F}$$

Puisque *dt* est un intervalle de temps très court,

$$\frac{\Delta\vec{L}}{dt} = I\frac{\Delta\vec{\omega}}{dt} = \tau = \vec{R} \times \vec{F}$$

Le vecteur $\vec{R} = \left[0.05\vec{i};0.05\vec{j};0\vec{k}\right]$, qui représente le point d'application de la force sur notre corps rigide et $\vec{F} = \left[0.3\vec{i};0.4\vec{j};0\vec{k}\right]$ représentant la force exercée à ce point. Le produit croisé est orienté ans le sens des $\vec{k}$ positifs. Dans notre cas, le moment de force $\tau = 0.0005\,\vec{k}$. Finalement, le Δt=0.01s dans la simulation.

En isolant $\Delta\omega$, on trouve

$$\Delta\omega = \frac{\vec{\tau}}{I}\Delta t = \frac{0.005}{0.0005}(0.01) = 0.1^o$$

La variation de la vitesse est donc de 0.1^o par tranche de Δt. L'application du moment de force a été appliquée pendant 0.7s, telle une force d'impulsion. Ce qui fait que notre vitesse angulaire devrait croître jusqu'à 0.7^o. Observons ce qui s'est passé pour la même simulation.

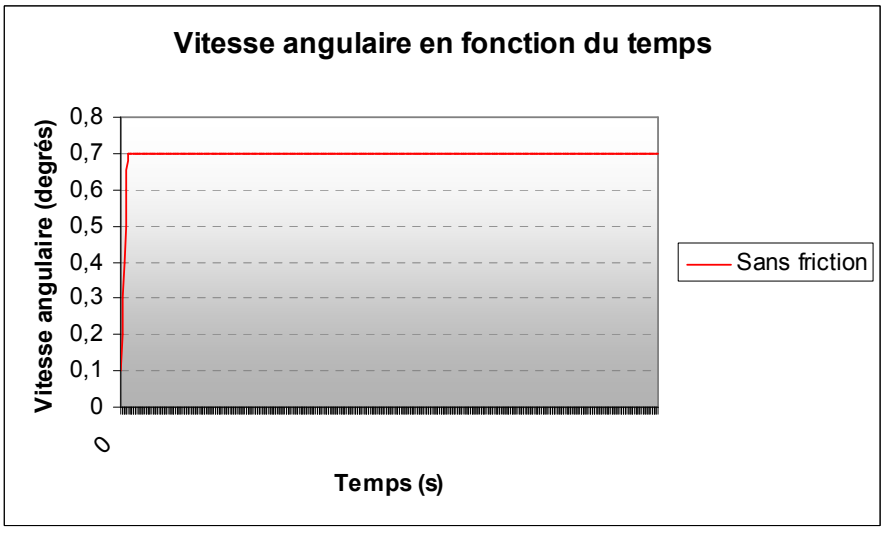


Figure 6.3 Vitesse angulaire, sans friction.

Comme prédit, la vitesse angulaire augmente linéairement jusqu'à atteindre le 0.7^o.

Comme deuxième partie d'analyse, considérons un modèle avec friction. Rappelons que la friction dépend directement de la vitesse. Pour ce faire, nous allons considérer les divers coefficients intervenant lors du calcul de la force de friction. Nous allons reprendre les mêmes informations de base du cas précédent, mais en y ajoutant les éléments suivants.

Données initiales	
Masse (m)	0.3 kg
Vitesse initiale (v_0)	60 m/s
Angle d'attaque (α)	45^o
Constante de gravité (g)	9,80655 m/s^2
Coefficient de contact (C)	0.5
Dimensions de l'objet	0.1m x 0.1m
Aire en contact (A)	0.01 m^2
Densité de l'air (ρ)	1.2 kg/m^3

Tableau 6.2 Données initiales pour la trajectoire sans friction.

Commençons par calculer le coefficient de friction (D),

$$D = \frac{\rho C A}{2} = \frac{1.2 \times 0.5 \times 0.01}{2} = 0.003$$

Rappelons que ce coefficient intervient directement dans la force de friction en fonction de la vitesse au carré.

$$\frac{d\vec{P}_x}{dt} = -D|\vec{v}|v_x \qquad \frac{d\vec{P}_y}{dt} = -D|\vec{v}|v_y \qquad \frac{\Delta\vec{L}}{dt} = I\frac{\Delta\vec{\omega}}{dt} - D|\vec{\omega}|^2 = \tau$$

La figure 6.4 montre le comparatif entre les données empiriques avec et sans friction.

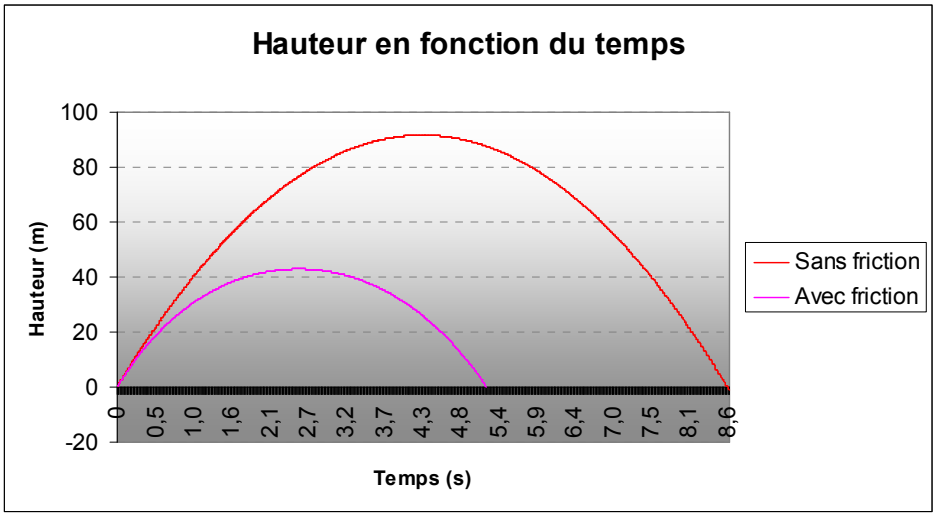


Figure 6.4 Données empiriques avec et sans friction.

L'impact de la friction sur le modèle démontre bien de l'importance de l'aérodynamisme du projectile. Ainsi avec la forme rectangulaire possédant une surface de contact énorme, la portée de la trajectoire est directement affectée.

Voici ce qui en est de la vitesse angulaire.

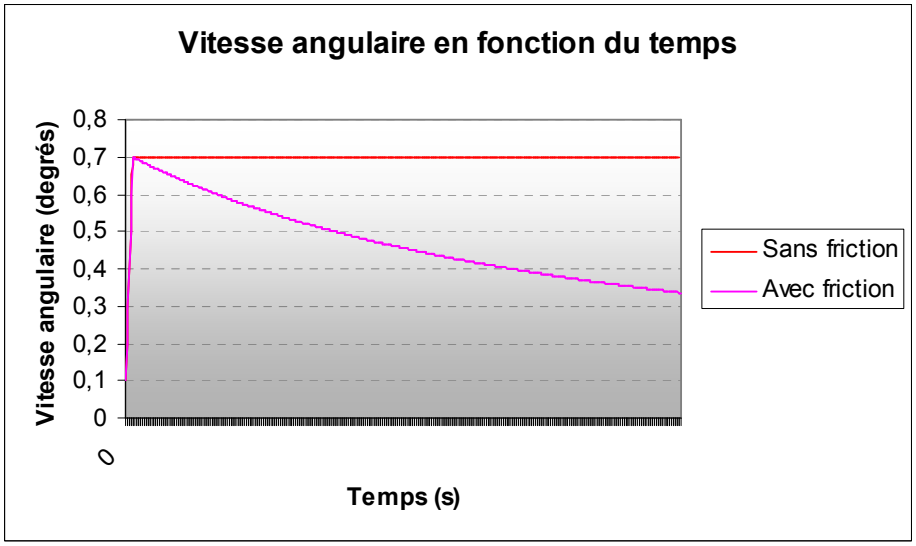


Figure 6.5 Données empiriques avec et sans friction pour la vitesse angulaire.

On peut voir que la vitesse angulaire avec friction décroît rapidement à mesure lorsque la vitesse angulaire est grande puis a tendance à se stabiliser lorsque la vitesse angulaire est moins considérable. Le fait s'explique simplement puisque la friction est fonction de la vitesse angulaire au carré. Une valeur faible au carré devient encore plus petite et occasionne moins d'impact sur le comportement général de la vitesse angulaire.

Figure 6.6 Interface utilisateur de la simulation du projectile (avec friction).

La figure 6.6 nous montre l'interface utilisateur produite pour simuler la trajectoire d'un projectile. On peut y retrouver la position courante du projectile (x, y) ainsi que la vitesse angulaire.

La prochaine figure nous montre l'utilisation du processeur pendant la simulation de la trajectoire du projectile. La simulation comprend l'affichage à l'écran.

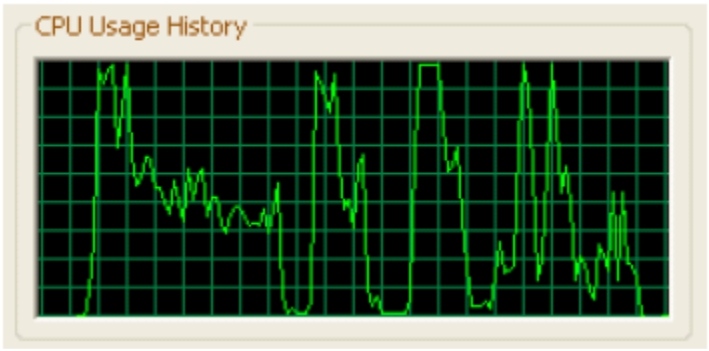


Figure 6.7 Utilisation en pourcentage du CPU pendant la simulation.

Comme on peut le voir, le processeur est beaucoup en demande ce qui témoigne de la quantité de calcul considérable à résoudre à chaque tranche de temps. On peut voir chaque pique comme étant les initialisations du système, après cela, il n'en tient qu'à des calculs de base comme le positionnement ou les calculs de vitesses.

6.4 Discussion

Il est fort à parier que la modélisation d'un pareil cas en trois (3) dimensions serait très gourmande pour le processeur. Comme en témoigne encore la figure 6.7, il serait intéressant de se pencher sur l'optimisation du code afin de le rendre plus rapide. Étant quand même concerné par le problème de lenteur de Java™ sur un langage non interprété et natif, il reste quand même assez de jeux côté ressource pour rendre le cas plus complexe au niveau calcul.

7. CAS DE SIMULATION III : DÉTECTION DE COLLISION DANS UN ENVIRONNEMENT AVEC CONTRAINTES.

7.1 Description du cas

Le cas de simulation précédent n'était composé que de corps rigides soumis à aucune contrainte puisque les interactions n'étaient pas possibles entre eux. L'absence de ces contraintes a donc favorisé l'utilisation des dynamiques newtoniennes qui seront conservées dans ce cas de simulation au détriment des dynamiques Lagrangiennes, étant pourtant un choix naturel concernant les corps en mouvements en présence de contraintes [3]. Dans les applications plus réalistes, une considération particulière est apportée à l'interaction des objets composant une scène. À ce fait, il faut pouvoir déterminer, par une approximation découlant d'une tolérance préalablement fixée (*voir section suivante*), si deux objets se sont interpénétrés et la manière dont ils devront réagir après l'impact. Ces deux éléments forment ce que l'on appelle un système de détection de collision. La présente section illustrera les diverses approches empruntées afin de déterminer les réactions physiques engendrées par des contacts entre deux corps physiques ainsi que les méthodes employées pour déterminer si deux polygones convexes entrent en contact et le point de collision entre ceux-ci.

En fonction des divers états des corps rigides (leur élasticité, leur vitesse, etc.) plusieurs comportements sont possibles à la suite d'un contact: la collision, le roulement, le glissement et le repos. Afin de simuler dynamiquement ces comportements, plusieurs approches ont été amenées par Mirtich, Canny [17] et Baraff [2]. La première consiste à appliquer une série d'impulsions sur les corps de manière à les contraindre à ne pas s'interpénétrer. La deuxième, suggérer par Baraff [15], consiste à renforcer des contraintes de non-pénétration.

Pour un état de collision se produisant entre deux corps rigides, la vitesse linéaire et angulaire des objets est abruptement modifiée. L'intégrateur de Runge-Kutta du 4^{e} ordre, au sein du moteur physique, ne peut malheureusement pas résoudre les équations différentielles décrivant les états physiques discontinus des objets en collision. À ce fait, il repose sur une prémisse de continuité et que de le contraindre à résoudre un état discontinu quelconque, engendrerait une instabilité des résultats [18]. Un moyen de contourner ce problème est en fait de stopper l'intégrateur numérique, de déterminer l'état présent de chaque corps rigide en contact, de calculer par la suite l'état après la collision et de créer une *force impulsive* qui sera appliquée aux corps après le redémarrage de l'intégrateur.

Selon Baraff [13], l'état de repos se calcule de manière triviale puisque nul n'est le besoin de stopper l'intégrateur numérique. L'état de repos conduit à la détermination d'une *force de contact* de direction inverse à la gravité et de magnitude égale à la force de son poids. Ainsi, la force de contact exercée sur l'objet en repos l'empêche de traverser celui sur lequel il gît. Mirtich et Canny [17] amènent un concept analogue, mais se traduit plutôt par une série de forces d'impulsion maintenant l'objet en place. Même si démarche préconisée semble plus simple et effective [17], il n'en est pas moins qu'elle produit malencontreusement des effets de tremblements qui doivent être contrôlés par hystérésis [19].

Afin de pouvoir calculer les forces résultantes d'une rencontre entre deux corps rigides, il faut déterminer préalablement le point de contact. Pour cela, une heuristique dynamique, basée sur la géométrie computationnelle, est préférée à des méthodes déterministes et semi-déterministes, même si elle semble

moins effective, selon Kim et Rossignac [20]. Il n'en reste pas moins que l'approche dynamique est plus facile à implémenter et qu'elle suffit aux exigences du présent cas de simulation.

La prochaine section explique d'une façon théorique les diverses approches préconisées dans le but de construire un système de collision.

7.2 Présentation du modèle théorique

Cette section tentera de décrire théoriquement les principales étapes composant un système de collision. Pour ce faire, les étapes chronologiques engendrées dans un système de collision seront décrites en exposant le comportement d'un système face à l'intégrateur numérique et en apportant, par la même occasion, différentes notions physiques liées au contact de corps rigides.

Dans un moteur physique, la présence d'un système de collision à un impact de même ampleur sur le réalisme qu'il offre à la simulation qu'au temps de calcul nécessaire à son fonctionnement. Il n'en va se dire que l'implémentation de son mode de fonctionnement est critique à la performance globale du système. À chaque itération de l'intégrateur numérique, le système de collision doit savoir quels objets de la scène entrent en contact afin d'ultérieurement, déterminer les forces impulsives ou de contact à soumettre aux corps en contact dans le but de modifier leurs dynamiques.

Pour ce faire, Baraff [15] et Zachmann [22] proposent d'utiliser une série de tests de complexités incrémentales permettant, en premier lieu, de déterminer s'il existe des nœuds sujets à entrer en contact. Les nœuds candidats sont déterminés par leurs proximités respectives. Une méthode brute consisterait comparer la distance de chacun des nœuds entre eux, $O(n^2)$ comparaisons seraient ainsi produites à chaque itération [22] . Puisque le processus se veut être le moins demandant possible, réduire ce nombre de comparaison ne pourrait être que bénéfique. Un algorithme tiré de la géométrie computationnelle, créé par Fortune [16], pallie à cette problématique en réduisant le nombre d'opérations à O(n log n).

Pour illustrer l'algorithme de *sweep* de Fortune, considérons la figure suivante arborant différents polygones dans une scène graphique.

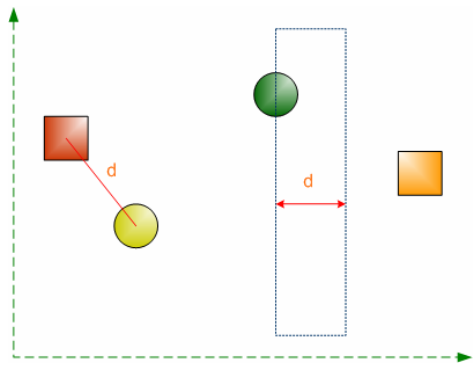


Figure 7.1 Algorithme de *sweep* par Fortune.

L'algorithme déplace de gauche à droite une zone de largeur d issue de la distance entre les deux premiers centroïdes trouvés. Il déplace cette zone vers la droite en faisant coïncider la bordure gauche de la zone avec le centroïde du prochain polygone. Si dans l'aire de la zone se trouve un centroïde appartenant à un autre

polygone, la zone prendra une dimension reflétant la nouvelle distance minimale. Chaque fois que l'on attribue une nouvelle distance, on s'assure de mémoriser les polygones concernés.

Malheureusement, l'algorithme de Fortune [16] est contraint à déterminer la distance minimale entre polygones de mêmes dimensions. Une approche plus concise, mais plus complexe, réside en une minimisation de la distance entre deux points provenant chacun de deux polygones. Cette méthode se résout à l'aide de méthodes LCP (*linear complementary problem*) présentées dans Eberly [3] mais ne fera pas l'objet d'étude dans le cadre de cette simulation.

Une fois les candidats établis, la deuxième phase consiste à déterminer s'il y a interpénétration entre les couples ou pas. À ce sujet, une panoplie de techniques et d'algorithmes différents ont été thèmes de recherche, tous offrant un degré de précision spécifique inversement proportionnel à leur rapidité d'exécution [22]. Le cas de simulation a été conceptualisé en fonction du théorème de séparation d'axes (SAT) en conjonction avec l'approximation des polygones de la scène en rectangles (OBB, *Oriented bouncing box*) puisque simple et suffisant pour la complexité de la simulation.

Le théorème de séparation d'axes stipule que pour deux polygones convexes, la projection de ceux-ci sur un axe déterminé nous indique s'il y a juxtaposition. Pour un polygone convexe quelconque, les axes de séparation possibles sont issus des normales de chaque côté (*figure 7.2*).

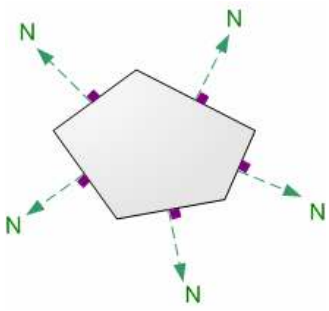


Figure 7.2 Axes de séparations possibles pour un polygone convexe quelconque.

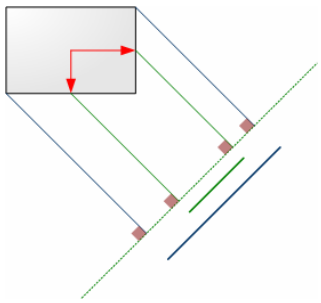


Figure 7.3 Projection des sommets et points milieux sur un axe de séparation.

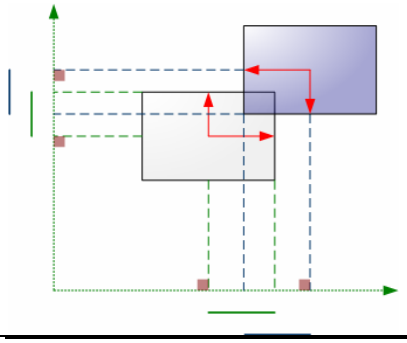


Figure 7.4 Superposition des projections entre deux polygones convexes.

Le test d'interpénétration s'effectue en projetant les sommets ou points milieux de chaque polygone sur l'axe de séparation et est concluant lorsqu'il y a juxtaposition sur tous les axes de séparation. Ce test est très rapide puisqu'il est interrompu dès qu'un axe ne présente pas de juxtaposition. La figure 7.3 présente la projection des sommets sur un axe de séparation quelconque. La ligne bleue épaisse représente la projection des points extrêmes sur l'axe tandis que la ligne verte épaisse au bas de l'axe représente la projection des points milieux du polygone, souvent utilisés à cause de sa simplicité. On peut voir dans la figure 7.4 un exemple typique de cas de superposition puisque tous les axes de séparation démontrent un chevauchement des projections des points milieux.

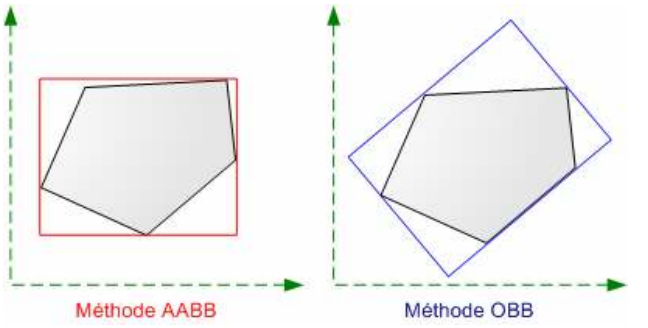


Figure 7.5 Exemple de la méthode AABB et OBB.

L'utilisation de ce théorème est encore plus efficace lorsque jumelé à OBB puisque la définition des axes de séparation est basée sur une forme rectangulaire. Cela peut être d'une aide précieuse lorsque les candidats sujets à une collision potentielle sont complexes. La méthode d'OBB consiste à former une boîte englobant une forme complexe afin de faciliter l'application du théorème des axes de séparation en ne projetant que les points milieux de la boîte et non les sommets que l'on devra déterminé comme extrêmes. Toute la difficulté réside en la création de la boîte, puisqu'à l'instar de la méthode AABB (*axis-aligned bouding box*), l'approche OBB délimite les zones par rapport à l'orientation des polygones et non des axes, occupant du coup une aire plus petite [3] (*voir figure 7.5*).

L'application d'une telle approche est chose triviale. À ce fait, considérons la figure 7.6.

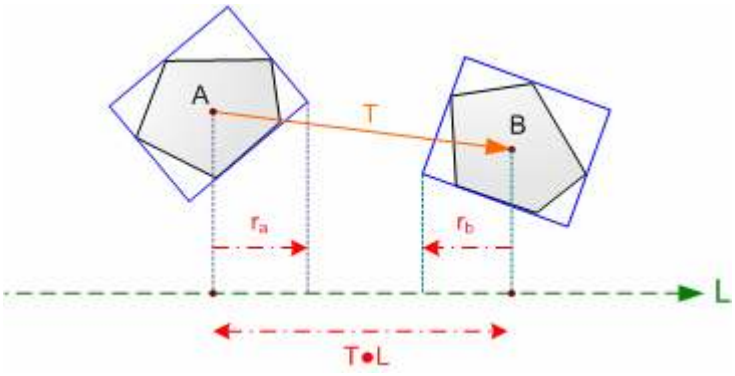


Figure 7.6 Détermination d'interpénétration entre deux OBB.

Le but est donc de projeter le point milieu et le sommet extrême de chaque polygone convexe sur un axe et de comparer la distance qui sépare les deux boîtes.

À ce fait, considérons

A La boîte englobant le polygone de gauche.
B La boîte englobant le polygone de droite.
$\vec{L}$ Un axe de projection quelconque.
$\vec{r}_a$ Le vecteur projeté sur l'axe $\vec{L}$ issu du point milieu du polygone A convexe ainsi que d'un sommet extrême.

$\vec{r}_b$ Le vecteur projeté sur l'axe $\vec{L}$ issu du point milieu du polygone B convexe ainsi que d'un sommet extrême.

$\vec{T}$ Le vecteur reliant les centres des polygones.

Il n'y a pas interpénétration des polygones lorsque $|\vec{r}_a| + |\vec{r}_b| < |\vec{T} \bullet \vec{L}|$.

Si quelques couples de nœuds de la scène que ce soit sont reconnus pour s'être interpénétrés, on interrompt l'intégrateur numérique afin de déterminer le temps approximatif de la collision par une méthode de bissection exposée par Press, Flannery, Teukolsky et Vetterling [21]. La détermination du temps approximatif est une conséquence de la démarche dynamique, puisqu'avec un modèle déterministe le temps de la collision pourrait être déterminé quadratiquement. La méthode binaire de bissection engendre des comparaisons entre les états d'interpénétrations des nœuds de la scène afin d'approximer le mieux possible (en prenant en compte une certaine tolérance, ε) le temps de collision exact, t_c. La figure 7.7 donne un exemple d'une tolérance arbitraire défini pour un cas quelconque. Il est à noter que, plus la tolérance est petite, plus la précision sera grande, mais du coup impliquera plus de temps de calcul.

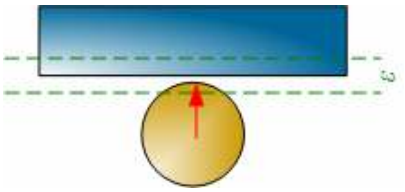


Figure 7.7 Exemple d'une tolérance arbitraire.

Afin d'illustrer la méthode de bissection, considérons

t temps courant de la simulation

Δt le pas utilisé dans l'intégrateur numérique

Le principe est simple, si à t aucune interpénétration n'est occasionnée et qu'à $t + \Delta t$ c'est effectivement le cas, il s'agit de demander à l'intégrateur de recalculer les états des corps à $t + \frac{\Delta t}{2}$. S'il y a collision, on recalcule à $t + \frac{\Delta t}{4}$, s'il n'y en a pas à $t + \frac{3}{4}\Delta t$ et ainsi de suite en faisant varier le pas de l'intégrateur de moitié jusqu'au moment où la distance (positive ou négative) séparant les deux nœuds soit inférieure ou égale à ε. ε étant la tolérance (distance) maximale permise pour considérer qu'il y a contact entre polygones.

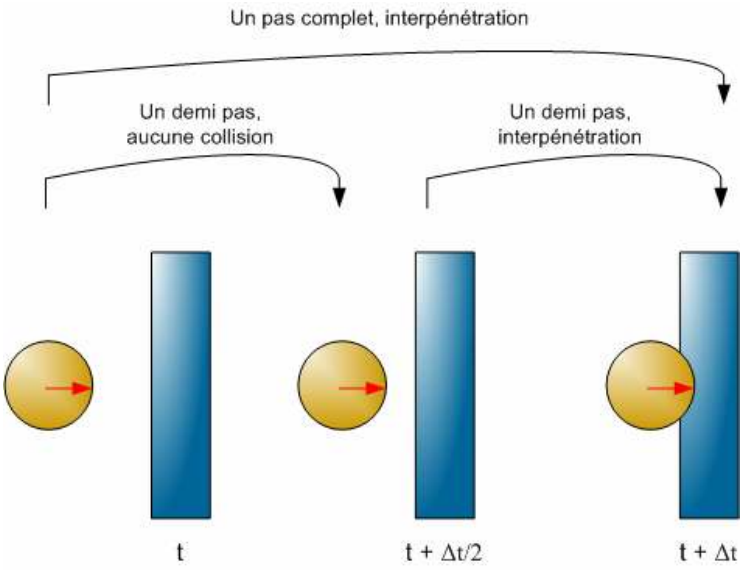


Figure 7.8 État de l'objet en fonction des pas de temps.

Avant de calculer les forces de contact ou d'impulsion à administrer aux corps en contact, la détermination du point de collision peut se faire par simple géométrie dans le cas de simulation présent. Or, ce n'est pas toujours le cas pour des formes plus complexes dans des dimensions supérieures à deux (2). Dans ces cas, la détection exacte du point de collision se fait par l'entremise de structures d'arbres BSP (binary space partitionning) OBB (oriented bouding box) afin de décomposer l'objet complexe en objet plus simple dont chacune des partitions compose un nœud de l'arbre [22]. À noter que cette même méthode est employée pour pallier à la concavité de certains polygones en le décomposant en objets convexes.

Avec les coordonnées du point de collision, il est maintenant possible d'appliquer des forces impulsives afin de modifier les moments linéaires et angulaires des corps en contact[7]. Pour ce faire, supposons le cas où un sommet du corps A entre en contact avec le contour du corps B au point P tel qu'illustré par la figure 7.9.

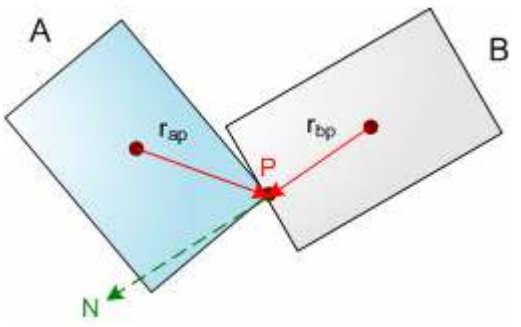


Figure 7.9 Deux corps rigides (polygones convexes) entrant en contact.

[7] Nous nous limiterons ici à des corps de densité constante.

Afin de déterminer les forces impulsives conséquentes, on se doit de définir les variables suivantes qui caractérisent les états des deux corps en contact avant et après la collision.

A	Le polygone convexe A.
B	Le polygone convexe B.
m_a	Masse du corps représenté par le polygone A
m_b	Masse du corps représenté par le polygone B
P	Point de rencontre des deux polygones A et B
$\vec{r}_{ap}$	Le vecteur liant le centre de masse du corps A au point P
$\vec{r}_{bp}$	Le vecteur liant le centre de masse du corps B au point P
ω_{a1} , ω_{b1}	Vitesse angulaire de pré collision des corps A et B
ω_{a2} , ω_{b2}	Vitesse angulaire de post collision des corps A et B
$\vec{v}_{a1}$, $\vec{v}_{b1}$	Vitesse linéaire de pré collision des corps A et B
$\vec{v}_{a2}$, $\vec{v}_{b2}$	Vitesse linéaire de post collision des corps A et B
$\vec{v}_{ap1}$, $\vec{v}_{bp1}$	Vitesse de pré collision du point d'impact sur les corps A et B
$\vec{N}$	La normale au contour du corps B
ε	Coefficient de restitution

On se doit préalablement de déterminer la vitesse linéaire du point P sur les corps A et B.

$$\vec{v}_{ap1} = \vec{v}_{a1} + \omega_{a1} \times \vec{r}_{ap} \quad (7.1)$$

$$\vec{v}_{bp1} = \vec{v}_{b1} + \omega_{b1} \times \vec{r}_{bp} \quad (7.2)$$

D'une façon analogue, on peut retrouver les vitesses linéaires finales du point P.

$$\vec{v}_{ap2} = \vec{v}_{a2} + \omega_{a2} \times \vec{r}_{ap} \quad (7.3)$$

$$\vec{v}_{bp2} = \vec{v}_{b2} + \omega_{b2} \times \vec{r}_{bp} \quad (7.4)$$

On peut maintenant définir une vitesse linéaire relative qui exprime les vitesses des points de contact sur les corps A et B.

$$\vec{v}_{ab1} = \vec{v}_{ap1} - \vec{v}_{bp1} \quad (7.5)$$

$$\vec{v}_{ab2} = \vec{v}_{ap2} - \vec{v}_{bp2} \quad (7.6)$$

En utilisant les formules 7.1 et 7.2, on peut réécrire les formules précédentes de la manière suivante.

$$\vec{v}_{ab1} = \vec{v}_{a1} + \omega_{a1} \times \vec{r}_{ap} - \vec{v}_{b1} - \omega_{b1} \times \vec{r}_{bp} \qquad (7.7)$$

$$\vec{v}_{ab2} = \vec{v}_{a2} + \omega_{a2} \times \vec{r}_{ap} - \vec{v}_{b2} - \omega_{b2} \times \vec{r}_{bp} \qquad (7.8)$$

Considérons maintenant le vecteur normal $\vec{N}$ au contour de B qui subit l'impact. Déduisons maintenant la vitesse en direction de la normale $\vec{v}_{n1}$, par le produit scalaire.

$$\vec{v}_{n1} = \vec{v}_{ab1} \bullet \vec{N} \qquad (7.9)$$

Si nous incluons maintenant le coefficient de restitution, ε dans la formule 11.9, le lien entre la vitesse linéaire relative initiale et finale, peut s'exprimer ainsi.

$$\vec{v}_{ab2} \bullet \vec{N} = -\varepsilon \vec{v}_{ab1} \bullet \vec{N} \qquad (7.10)$$

Le coefficient de restitution ε varie entre [0,1]. Lorsqu'il vaut 1, la réflexion est parfaite et aucune énergie cinétique n'est perdue. Cependant, dans le cas où il vaut 0, la perte d'énergie est maximale et entraîne les corps à rester en contact au point P. Afin de renforcer la contrainte de non-pénétration telle qu'avancée par Baraff [15], une force impulsive doit être développée en direction de la normale lorsqu'une collision se produit ou parallèle à la normale lorsque les corps restent en contact. À ce moment, une force de contact, en fonction du poids du corps, sera appliquée.

Les formules 7.11 et 7.12 représentent les équations décrivant la vitesse linéaire finale pour chaque corps après l'application d'une force impulsive.

$$\vec{v}_{a2} = \vec{v}_{a1} + j\frac{\vec{N}}{m_a} \qquad (7.11)$$

$$\vec{v}_{b2} = \vec{v}_{b1} - j\frac{\vec{N}}{m_b} \qquad (7.12)$$

Où j est un scalaire représentant l'impulsion en direction de la normale $\vec{N}$ qui modifie le moment linéaire.

Puisque le moment linéaire est en fonction du poids et qu'on recherche une vitesse, on s'assure de diviser le terme suivant le coefficient j par le poids de chacun des corps.

Les équations 7.11 et 7.12 se différencient aussi par le fait que l'impulsion j est de sens contraire pour les deux corps, les repoussant ainsi l'un de l'autre.

Une forme analogue est employée pour la détermination des vitesses angulaires.

$$\omega_{a2} = \omega_{a1} + \frac{(r_{ap} \times j\vec{N})}{I_a} \qquad (7.13)$$

$$\omega_{b2} = \omega_{b1} + \frac{(r_{bp} \times j\vec{N})}{I_b} \qquad (7.14)$$

Où *I* représente la matrice d'inertie (ou tenseur) lorsqu'appliqué à la troisième (3) dimension ou le scalaire représentant l'inertie en deux (2) dimensions pour chacun des corps.

Reprenons maintenant l'équation 7.10 afin de déterminer le paramètre d'impulsion *j*[8].

$$\vec{v}_{ab2} \bullet \vec{N} = -\varepsilon \vec{v}_{ab1} \bullet \vec{N}$$

Faisons l'expansion de l'équation afin de développer le terme de gauche.

$$(\vec{v}_{ap2} - \vec{v}_{bp2}) \bullet \vec{N} = -\varepsilon \vec{v}_{ab1} \bullet \vec{N}$$
$$(\vec{v}_{a2} + \omega_{a2} \times \vec{r}_{ap} - \vec{v}_{b2} - \omega_{a2} \times \vec{r}_{bp}) \bullet \vec{N} = -\varepsilon \vec{v}_{ab1} \bullet \vec{N}$$

En utilisant maintenant les équations 7.13 et 7.14.

$$((\vec{v}_{a1} + j\frac{\vec{N}}{m_a}) + (\omega_{a1} + \frac{(\vec{r}_{ap} \times j\vec{N})}{I_a}) - (\vec{v}_{b1} + j\frac{\vec{N}}{m_b}) - (\omega_{b1} - \frac{(\vec{r}_{bp} \times j\vec{N})}{I_b})) \bullet \vec{N} = -\varepsilon \vec{v}_{ab1} \bullet \vec{N}$$

On peut reconnaître que le terme $\vec{v}_{ab1} \bullet \vec{N}$ apparaît dans la partie gauche, on l'envoie donc à droite pour produire l'équation suivante.

$$(j\frac{\vec{N}}{m_a} + (\frac{\vec{r}_{ap}}{j\vec{N}}) \times \frac{\vec{r}_{ap}}{I_a} + j\frac{\vec{N}}{m_b} + (\frac{\vec{r}_{bp}}{j\vec{N}}) \times \frac{\vec{r}_{bp}}{I_b} = -(1+\varepsilon)\vec{v}_{ab1} \bullet \vec{N}$$

En assumant que $\vec{N} \bullet \vec{N} = 1$ et en utilisant la règle du triple produit scalaire,

$$(\vec{A} \times \vec{B}) \bullet \vec{C} = (\vec{B} \times \vec{C}) \bullet \vec{A}$$
$$(\vec{A} \times \vec{B}) \times \vec{A} \bullet \vec{B} = (\vec{A} \times \vec{B}) \bullet (\vec{A} \times \vec{B}) = (\vec{A} \times \vec{B})^2$$

[8] La preuve est développée par Erik Neumann, http://www.myphysicslab.com/collision.html.

On peut simplifier l'équation et isoler le paramètre d'impulsion *j*.

$$j = \frac{-(1+\varepsilon)\vec{v}_{ab1} \bullet \vec{N}}{\frac{1}{m_a} + \frac{1}{m_b} + \frac{(\vec{r}_{ap} \times \vec{N})^2}{I_a} + \frac{(\vec{r}_{bp} \times \vec{N})^2}{I_b}} \qquad (7.15)$$

Grâce à l'équation 7.15, on peut maintenant calculer les vitesses linéaires et angulaires finales en utilisant les équations 7.11, 7.12, 7.13 et 7.14.

À titre indicatif, si on veut calculer l'impulsion avec un corps au poids infini, tel que la rencontre entre une balle et un mur par exemple, on devra présumer que $m_b \to \infty$ et par conséquent que $I_b \to \infty$ pour que la forme résultante du paramètre d'impulsion généré à partir de l'équation 7.15 nous donne

$$j = \frac{-(1+\varepsilon)\vec{v}_{ab1} \bullet \vec{N}}{\frac{1}{m_a} + \frac{(\vec{r}_{ap} \times \vec{N})^2}{I_a}} \qquad (7.16)$$

7.3 Discussion

Exceptionnellement, cette simulation n'a pu être implémentée dans le moteur physique. Les causes de cette décision sont en faite liées à une contrainte de temps empêchant une adaptation de l'architecture courante aux nouvelles normes requises pour l'intégration du système de collision. Il n'en reste pas moins que le sujet est très complexe puisque de nombreux auteurs proposent des approches complètement différentes. Parmi ceux-ci, la forte majorité nous propose des modèles dynamiques non déterministes qui utilisent une granularité de plus en plus fine afin de limiter les temps de calcul.

La complexité peut s'accroître très rapidement lorsqu'il s'agit de gérer la collision de polyèdres puisque ceux-ci se meuvent dans un espace à trois (3) dimensions. En plus de s'assurer de leur convexité, dans le cas échéant une décomposition sera de mise, et afin d'éviter un surplus de calculs inutiles, il est fortement conseillé de borner virtuellement les arêtes des objets en les stockant dans des structures de type arbres BSP afin d'accélérer les algorithmes. Malheureusement, la formation des bornes (boîtes entourant les parties de notre modèle) n'est pas tâche triviale.

Comme on le devine aisément, l'intégration d'un tel système occasionne une charge de calcul additionnelle ayant un impact considérable sur le moteur physique. Ainsi, il est très important de choisir correctement les algorithmes à utiliser afin de garder uniquement un petit nombre de candidats à traiter. Si tel n'est pas le cas, une baisse des performances est à prévoir puisque les algorithmes subséquents à la tâche initiale sont lourds.

8. OUVERTURE

Le développement d'un moteur physique ne fut pas une tâche triviale du fait que la modélisation de systèmes d'équations différentielles de certains cas de simulation (particulièrement le cas II avec intégration de forces dissipatives) fut complexe et difficilement testable. La complexité de ces tests et de la vérification des données a donné du fil à retordre puisqu'une modélisation analytique a dû être préalablement établie afin de contre-vérifier les approximations de la méthode numérique Runge-Kutta. Les erreurs détectées par cette contre-vérification ont été corrigées à même les modèles exposés.

Malgré la contrainte de temps empêchant l'implémentation finale du troisième cas de simulation, l'idée générale et les modèles théoriques établis sur les études mentionnées à la section 7 concernant la détection de collision permettront une intégration simpliste à l'architecture proposée dans ce document et n'entraîneront que très peu de changements.

Cette même architecture à répondu aux objectifs fixés [1] (démontré par l'implémentation des cas de simulation) et les attributs architecturaux ayant été à la base même de la conceptualisation ont su cerner la problématique avancée.

8.1 Limitations du système

Les contraintes préalablement définies [1] du système ont borné le moteur physique à la simulation de la dynamique de corps rigides uniquement. Nulle question n'était ici d'approximer le mouvement de corps déformables.

Le moteur physique, bien que supportant une approche plausible en trois dimensions requérant une quantité faible de changements, s'est concentré uniquement sur une modélisation en deux (2) dimensions des simulations simplifiant énormément les modélisations et allégeant les calculs.

La modélisation du système de collision (section 7) a été conçue de manière à prendre en compte uniquement des polygones convexes. L'utilisation de modèles convexes complexes ou polygones concaves aurait nécessité une décomposition afin de réduire les formes à des polygones convexes simples.

8.2 Recommandations

À titre de recommandations, il serait intéressant de porter le design à un langage tel le C++, tout en gardant la portabilité du système, afin d'incrémenter les performances et permettre une représentation en temps réel. Cela serait idéal dans le cas d'une interopérabilité future avec un engin graphique, dont la plupart sont écrits dans ce langage.

En ce qui concerne l'intégrateur numérique, une étude plus poussée devrait être faite afin de mieux cerner les conditions de Liptischitz (décrits à la section 4.3) caractérisant la stabilité de la méthode. Cette analyse pourrait assurer en toutes conditions, la validité des approximations à partir de la modélisation analytique. Ainsi, lorsqu'une erreur surviendrait, les efforts seraient concentrés uniquement sur la validation des

systèmes d'équations différentielles excluant toutes recherches sur les approximations numériques accélérant du coup la charge de travail.

8.3 Extensions possibles

Le passage de la deuxième à la troisième dimension serait une tâche ardue, mais permettrait une simulation de cas beaucoup plus complexe.

L'écriture d'un module supplémentaire afin de traiter des fichiers comportant des scripts de simulation permettrait de définir, avec n'importe quel éditeur de texte, les conditions initiales des objets à inclure dans une simulation. L'utilisateur pourrait ainsi ne plus avoir à compiler chaque fois le moteur physique ainsi que le cas de simulation.

9. BIBLIOGRAPHIE

[1] Jérôme Schmaltz, Proposition de projet (PHE-PJP[9]), ETS, Montréal, 2006.

[2] Jérôme Schmaltz, Glossaire (PHE-GLO[10]), ETS, Montréal, 2006.

[3] David H. Eberly. Game Physics, Morgan Kaufmann Publishers, San Francisco, 2004.

[4] David H. Eberly. 3D Game Engine Architecture, Morgan Kaufmann Publishers, San Francisco, 2005.

[5] Len Bass, Paul Clements, Rick Kazman. Software Architecture in Practice, Second edition, Addison-Wesley Publishers, San Francisco, 2003.

[6] Object Management Group, Inc. Unified Modeling Language, http://www.uml.org/, 2006.

[7] Jovan Popovic, Steven M. Seitz, Michael Erdmann. Motion sketching for control of rigid-body simulations, ACM Transactions on Graphics (TOG), Volume 22 Issue 4, ACM Press, Octobre 2003.

[8] Zoltan Konyha, Kresimir Matkovic, Helwig Hauser, Interactive 3D Visualization Of Rigid Body Systems, Proceedings of the 14th IEEE Visualization 2003 (VIS'03) VIS '03), IEEE Computer Society, Octobre 2003.

[9] Rick Kazman, Mark Klein, Paul Clemenst, ATAM : Method for architecture evaluation, http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf, Technical Report, SEI, Février 2006.

[10] Ferdinand P. Beer, E. Russell Johnston Jr. Vector Mechanics for Engineers: Dynamics, Third SI Metric Edition, McGraw-Hill Ryerson Limited, San Francisco, 1999.

[11] Johan Gästrin. Physically Based Character Simulation – Rag Doll Behaviour in Computer Games, Royal Institute of Technology, Stockholm, 2004.

[12] Jérôme Schmaltz, Requirements, constraints and Architecture (PHE-RCA[11]), ETS, Montréal, 2006.

[13] Wikipedia, www.wikipedia.com, Avril, 2006.

[14] J. Hoffman, Numerical methods for Engineers and Scientists, http://www2.latech.edu/~dmg/stiff_ODE_solution.PDF, Avril, 2006.

[15] D. Baraff. An introduction to Physically Based modeling: Rigid Body simulation II – Nonpenetration constraints, Carnegie Mellon University, 1997.

[9, 12, 13] Version disponible sur CD-ROM

[16] S. Harris, J. Ross. Beginning Algorithms. Worx Press. 2006.

[17] B. Mirtich, J. Canny, Impulse-based simulation of rigid bodies. ACM Press, New York, 1995.

[18] P. Viot. Méthode d'analyse numérique. http://cel.ccsd.cnrs.fr/cours/cel-19/numeri.pdf. Paris. 2003.

[19] B. V. Mirtich. Impulse-based dynamic simulation of Rigid Body Systems. University of California. 1996.

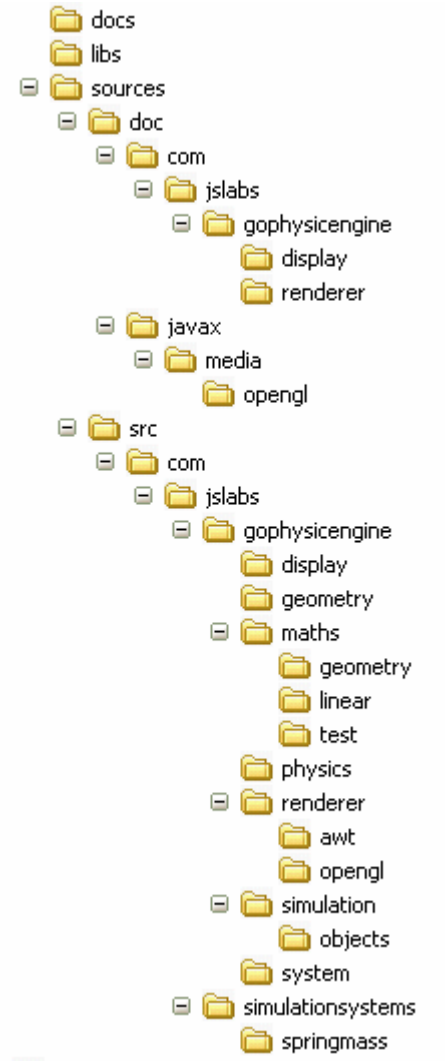
[20] B. Kim, J. Rossignac. Collision prediction. Georgia Institute of Technology. Atlanta. ASME. 2003.

[21] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T.Vetterling. Numerical Recipes. Cambridge University Press, 1986.

[22] J. Dequidt. L. Grisoni. P. Meseure. C. Chaillou. Traitements de collisions entre objets rigides convexes. Université de Lille 1.

10. ANNEXE A : CONTENU DU CD-ROM

Voici l'arborescence des répertoires du CD-ROM livré avec le document.



Répertoire	Contenu
docs	Tous les documents PDF livrés durant le projet, y compris le rapport final.
libs	Librairies nécessaires à la compilation du projet.
sources	Les sources du projet, comprenant par défaut le cas de simulation 2.