8INF935 - Mathématiques et physique pour le jeu vidéo

Phase 4 - Journal de bord

Jules RAMOS (RAMJ27020100) - Mikhael CHOURA (CHOM23060100) - Gaëlle THIBAUDAT (THIG24539900) - Béatrice GARCIA CEGARRA (GARB19510105)

Table des matières

Introduction

L'objectif de cette quatrième phase de développement était de reprendre le jeu de tir balistique de la phase 3 et de lui rajouter un système simple de résolution de collisions spécialisé dans les boîtes.Un octree ou un BSP tree devait être utilisé et visible pour subdiviser l'espace et réduire le nombre de tests de collisions à effectuer.

I. Les ajustements par rapport à la phase 3

Suivant les recommandations qui nous ont été faites, le système de gestion de la rotation des corps rigides a été modifié pour prendre comme seul centre de rotation le centre de masse du corps rigide afin de rendre l'implémentation plus réaliste.

Des boîtes non cubiques, et donc des parallélépipèdes rectangles, ont également été utilisées afin de mieux montrer visuellement le potentiel du moteur de physique.

II. L'octree

L'octree permet, de façon efficace, de calculer les potentielles collisions entre les nombreux corps rigides. Cette partie fut assez complexe à réaliser et à optimiser, mais nous avons réussi à avoir un résultat plutôt convaincant.

Concrètement, nous avons implémenté l'octree à l'aide de la structure *Node* permettant de représenter un nœud de l'octree, et également de stocker une zone de l'espace dans cette structure. Chaque zone doit disposer des solides qui sont à l'intérieur de celle-ci, nous utilisons cette fois la structure *CoverInternSphere* qui permet de stocker la sphère de collision associée à un solide, avec quelques informations supplémentaires utiles pour les opérations de l'octree. Il faut à présent compter pour chaque noeud, le nombre de solide dans sa zone, et effectuer :

- Soit un découpage de la zone, et donc un ajout de nœud à l'arbre s'il y a trop de solide. Cette tâche est réalisée par la fonction *cutNodeTree*.
- Soit une fusion des zones, s'il n'y a pas assez de solides dans le total des zones. Cette tâche est réalisée par la fonction *joinNodeTree*.

Nous avons également mis une borne minimale pour la taille de chaque zone. Cette limite est assez petite, et permettait durant la phase de test d'éviter les surcharges de découpage de zone en de nombreuses zones minuscules, ce qui rallongerait le temps de calcul inutilement.

L'octree devait aussi s'adapter au mouvement des solides. Cette partie était plus compliquée à implémenter, car il fallait modifier la structure de l'octree en essayant d'être le plus efficace possible. Nous devons donc vérifier lorsqu'un solide change de zone, c'est-à-dire lorsqu'il commence à chevaucher une autre zone, ou lorsqu'il quitte sa précédente zone. Lorsque ce cas se produit, nous faisons une mise à jour des nœuds associés au solide, puis nous regardons si nous devons faire un potentiel découpage/fusion. Cet algorithme, étant assez coûteux, ne se fait pas si le solide bouge à l'intérieur d'une même zone. De même, lorsqu'un solide arrive sur une bordure, l'algorithme ne s'exécute qu'une fois, et non pas à chaque frame où le solide reste sur la bordure.

Enfin, pour retourner les collisions potentielles, nous avons juste à prendre en compte chaque cas possible de collision dans une même zone de façon unique (sans doublon).

Pour vérifier la correction de l'octree, nous avons réalisé une fonction permettant de dessiner le découpage des zones de l'arbre de façon dynamique. Nous avons ensuite fait de nombreux tests manuels pour s'assurer du bon fonctionnement de ce module.

III. La vérification restreinte des collisions

Une fois déterminé qu'une boîte peut potentiellement entrer en collision avec une autre, la phase restreinte de détection de collisions est appliquée.

Tout d'abord, les faces d'une boîte et les sommets de l'autre doivent être récupérés grâce à leur matrice respective de rotation contenant les axes normaux de la boîte. Dans un premier temps, pour obtenir les axes qui définissent la boîte, les vecteurs colonne de la matrice de rotation sont normalisés et multipliés par la longueur de la dimension de la boîte associée (la dimension en x pour la première colonne de la matrice, celle en y pour la deuxième, etc.). Ces axes sont ensuite additionnés à la position du centre de la boîte pour obtenir soit le point au centre d'une face, dans le cas d'une translation avec un seul vecteur, soit un sommet de la boîte, dans le cas d'une translation avec deux vecteurs.

Une face, étant considérée comme un plan, est définie par un point en son centre (bien qu'il soit possible de choisir tout point quelconque à sa surface), et par l'axe normal associé. Il est ensuite possible de vérifier quels points dépassent le plan défini par une face de l'autre boîte par calcul de t, distance signée du point au plan. Étant donné que les normales des plans vont vers l'extérieur des boîtes, une distance signée négative signifie que le point a dépassé le plan. Si ce test est vrai pour les 6 faces du parallélépipède, le sommet est à l'intérieur.

Une fois ce test effectué, on stocke une liste de booléens pour, en sortie d'itération sur tous les sommets d'une boite, pouvoir récupérer quels sommets sont en collision avec l'autre boîte.Selon le nombre de sommets concernés, on peut alors déterminer si l'on est dans l'une des 3 situations à tester:

- Seul un sommet est en collision : il devient donc le point d'application de la collision
- Une arête est en collision : on prend la demi-somme des positions des deux sommets pour appliquer la collision au centre de l'arête
- Une face est en collision : en utilisant la formule de calcul de R, point le plus proche d'un point donné sur un plan, on trouve le point le plus proche du centre de la boite sur la face formée par les 4 sommets considérés, ce qui donne le centre de la face pour appliquer la collision

On peut ainsi résoudre la collision sur un point précis en coordonnées monde. Si aucune collision n'est détectée cependant, cet algorithme est appliqué en inversant les rôles (considération des faces ou sommets) des deux boîtes pour éliminer des cas limites, tels qu'une boîte bien plus grande que l'autre.

IV. La résolution des collisions

La résolution des collisions se fait en deux grandes phases.

Tout d'abord, il y a une étape de téléportation qui sépare les deux boîtes. Pour cela, une méthode similaire à celles utilisées en phase 2 est utilisée, en prenant en compte la différence de masse pour savoir comment répartir le déplacement des boîtes.

Ensuite, une force de collision, utilisant la classe RigidBodyForce de la phase 3, est ajoutée à chacune des boîtes. Afin de simuler une collision réaliste, la direction d'application de cette force sur une boîte correspond à la direction du vecteur direction de l'autre boîte. On obtient ainsi bien des boîtes qui entrent en collision, changent de trajectoire et se mettent en rotation selon la seconde boîte actrice.

Une valeur expérimentale et arbitraire de la norme de la force à appliquer pendant les collisions a été testée et sélectionnée.

V. La mise en place des démonstrations

Afin de présenter les résultats de la phase, 5 démonstrations différentes, correspondant à 5 fonctions spawnant des boîtes avec des forces initiales et des dimensions différentes, ont été implémentées dans le gameworld et sont appelées avec les touches espace, c, v, b et n. Cette solution simple permet de montrer des situations prédéterminées pour les besoins de la présentation.

VI. Les tests

Pour des raisons de complexité des méthodes, les méthodes de l'octree étant toutes liées, la détection restreinte des collisions nécessitant un gameworld complet et la résolution des collisions étant plus simple à calibrer visuellement, des tests d'intégration ont été favorisés, en testant différentes situations intégrant au fur et à mesure les différentes parties de la phase. Ces situations se sont ainsi transformées en méthodes de démonstrations citées plus haut. Des tests de régression pour la résolution des collisions, avec des tests répétés du programme complet, ont également été utilisés. Ainsi, des tests unitaires plus simples ne semblaient pas judicieux pour cette phase en particulier, tous les éléments et modules élémentaires ayant déjà été testés dans les phases précédentes, et nous n'en avons donc pas ajouté.

VII. Le déroulement de la phase

Après la répartition des tâches durant la première séance de TP le 29 novembre, les tâches à réaliser ont été séparées en 3 grandes parties: l'octree, la détection des collisions restreintes, et la résolution des collisions. L'objectif a été fixé au mercredi 6 décembre, une semaine plus tard, pour avoir une première version fonctionnelle de chacune des parties afin de réaliser des tests d'intégration et de mettre en forme le rendu.

Mikhael s'est chargé de l'octree, et de son affichage, et avait une version fonctionnelle avant la date fixée.

Jules et Béatrice ont réalisé la détection des collisions en pair coding, et un test d'intégration avec une sortie en cout a montré que l'implémentation était valide.

Gaëlle n’a pas eu de résolution de collisions fonctionnelles, et ne savait pas comment s’y prendre. N’étant pas présente au TP du 6 décembre en présentiel, un appel Teams a été mis en place afin de s’adapter à la situation.

La résolution des collisions a ainsi été implémentée sur un ordinateur en collaboration, en partage d’écran pour montrer et expliquer à Gaëlle ce qu’elle n’avait pas compris, tant dans l’implémentation des dernières phases que dans le cours.

Il est ensuite resté la création des démonstrations, que la paire Jules/Béatrice a préparé ensuite, et la préparation de la présentation qui a été réalisée par Mikhael.