Journal de Développement (1)

Mathématiques et Physiques pour le Jeu Vidéo

Difficultés rencontrées

Avec l'emploi de la bibliothèque OpenFrameworks, nous avons eu un peu de mal à mettre en place notre projet avec l'outil collaboratif git. En effet, certains fichiers ne devaient pas être partagés pour que le dépôt fonctionne. De plus, la création du dossier local lié au dépôt devait être bien placé et correctement relié à la bibliothèque OpenFrameworks.

Notre projet jusqu'à présent n'est pas très optimisé, en effet pour les différents opérateurs que l'on a programmé, nous créons de nouveaux pointeurs temporaires pour effectuer les opérations plutôt que de travailler directement via les pointeurs et d'accéder au vecteur courant par ce biais. c'est par exemple le cas avec la méthode *operator==()* qui copie le vecteur en paramètre pour en faire un pointeur local.

Astuces de Programmation

Nous avons fait un fichier de test à part pour regrouper les tests sur les diverses fonctionnalités que nous avons implémenter.

Nous avons mis en place un système de branches sur Git pour chacune des fonctionnalités qui nous permet donc de les développer sans venir perturber le travail des autres avec des fichiers qui ne sont pas encore fonctionnels. Cela nous permet également de faciliter la répartition des tâches ; en effet, une fois chaque branche créée, il est facile de s'en attribuer une et de travailler dessus.

Choix de Conception

Nous avons choisi de mettre l'axe z sur la profondeur plutôt que la hauteur. Cela permet de passer en deux dimensions facilement avec l'axe X et l'axe Y déjà bien placé. Nous avons donc dû adapter la gravité et la position du sol et les définir par rapport à l'axe Y plutôt que l'axe Z comme cela se fait habituellement en physique.

Nous avons choisi en accord avec les conseils proposés sur moodle de prendre une longueur de frame variable plutôt que de fixer une durée de frame à l'avance. Pour ce faire, nous récupérons le frame rate actuel à l'aide de la fonction *ofGetFrameRate()*. Pour obtenir la longueur d'une fame il nous suffit donc de prendre l'inverse de ce frame rate ce qui nous permet d'avoir une plus grande adaptation à la vitesse de calcul de chaque machine.

Le nombre de rectangles d'intégration par frame n'est pas fixé à l'avance mais est par défaut de 50. Cette valeur par défaut sera sûrement adaptée au fur et à mesure si avec les différents ajouts il s'avère nécessaire de la diminuer pour garder une bonne fluidité.

On peut également imaginer des améliorations qui consistent à avoir un découpage plus fin lorsqu'il y a de grosses variations et de diminuer ce nombre de pas sur les passages où le mouvement est davantage linéaire.

Journal de Développement (2)

Mathématiques et Physiques pour le Jeu Vidéo

Journal de Bord

Semaine 6 :

Mercredi : Après le retour de l'évaluation, changement de la méthode d'intégration pour passer de celle des rectangles à la méthode d'Euler et début de l'implémentation des générateurs pour les forces avec notamment les interfaces.
Samedi : Répartition des tâches entre les différent·es membres du groupe.

Semaine 7 :

Mercredi : Implémentation du système de détection de collision et des forces de gravitation et de friction. Mise en place d'un système de level pour tester les différents "mini-jeux" demandés.
Jeudi : Mise en place de la sélection des levels.
Vendredi : Implémentation des différentes forces de ressorts et du système de résolution des collision avec notamment le cable, la tige et la création d'un CollisionHandler.
Dimanche : Réorganisation des fichiers, création des blobs, gestion de la force élastique et ajout des impulsions.

Semaine 8 :

Lundi : Amélioration du CollisionHandler.
Mardi : Nous avons fait un énorme travail sur l'ensemble de la journée pour améliorer l'ensemble des fonctionnalités qui étaient pour certaines encore peu fonctionnelles. C'est notamment le cas du blob, des câbles et des Collisions pour lesquels les opérations float*Vector3D produisait un nombre important d'erreurs. Nous avons de plus amélioré notre gestion des collisions et ajouté les contacts au repos. Le traitement des forces de friction, nous ont permis de les intégrer rapidement au contact au repos.
Mercredi : Ajout des dernières fonctionnalités utiles comme l'affichage du nombre de fps et de la position du noyau du blob ainsi que du nombre de particules le composant. Nous avons également ajouté des câbles pour lier les particules du blob et garder une plus grosse cohérence de celui-ci même dans des cas extrêmes. Nous avons en outre ajouté des commandes pour permettre au blob de se déplacer dans le mini-niveau créé et de faire apparaître des particules dans le niveau (touche p). Enfin, nous avons ajouté des fonctionnalités bonus qui permettent au blob de se séparer en morceau puis de se recomposer au contact avec une particule. Pour finir, nous avons nettoyé le projet de tous les artéfacts de développement et ajouté des commentaires, des fichiers .h et corriger des problèmes des Vector3D.

Difficultés rencontrées

Les forces de ressort ont été les forces les plus difficiles à implémenter. En effet, ceux-ci avaient des comportements imprévisibles dû à des erreurs de calcul sur les flottants. Nous faisions la somme de plusieurs vecteurs avec la même composante nulle pour chacun d'eux, cependant la somme sur cette composante n'était pas nulle. Cette erreur qui aurait pu sembler négligeable, s'est avérée avoir un impact important après quelques frames et nous avons donc changé notre calcul de somme.

Des cas limite ont également dû être traités au cas où ils venaient à advenir au cours de la simulation. C'est par exemple la situation ou la particule et la fixation du ressort se confondent, en effet dans cette situation, le vecteur entre les deux est nul, nous avons choisi que dans ce cas, aucune force ne serait appliquée. Un autre exemple de situation limite est le cas de l'application de forces sur un objet de masse infini, dans ce cas nous avons considérés que l'objet ne subissait pas de force (en effet comme il est de masse infini ces forces n'auront aucun effet d'après la deuxième loi de Newton). Enfin, on peut également relever le cas où les fps mesurés sont nuls, nous avons considéré que la particule restait immobile pendant la durée de la frame. Ces erreurs sont particulièrement difficiles à détecter car surviennent à l'improviste au fur et à mesure des tests.

Du fait de la rapidité de la mise en place de certaines parties du TP pour des raisons d'efficacité, nous avons commis certaines erreurs de développements. Par exemple, nous n'avons pas créé de .h pour chaque force et avons immédiatement créé les fichiers .cpp. Dans le cas de classe Vector3D nous avons négligé l'implémentation de certains de calculs qui se révèlent à posteriori approximatifs du fait de la manipulation de float en cascade. Pour ce faire nous aurions dû, entre autres, utiliser les références des Vector3D plutôt que de les cloner.
Sur les Vector3D, en dehors des approximations précédemments mentionnés nous avons également constaté des erreurs de calculs significatives dans le cas de figure suivant : Vector3D*float*float. En faisant le calcul sous la forme de Vector3D*(float*float), aucune erreur ne se produit. Nous nous contentons d'utiliser cette méthode pour la phase 2 mais l'erreur sera corrigée pour la phase 3.

```
( 0.000000, -1.000000, -0.000000 ) * -370.265 * 0.5 = ( 1.042107, 0.000000, 0.000000 )
( -0.000066, -1.000000, -0.000000 ) * -370.537 * 0.5 = ( 1.042107, 0.000000, 0.000000 )
( -0.000199, -1.000000, -0.000000 ) * -370.808 * 0.5 = ( 1.042107, 0.000000, 0.000000 )
( -0.000399, -1.000000, -0.000000 ) * -371.077 * 0.5 = ( 1.042107, 0.000000, 0.000000 )
( -0.000666, -1.000000, -0.000000 ) * -371.343 * 0.5 = ( 1.042107, 0.000000, 0.000000 )
```

Le calibrage des paramètres nous a demandé de faire quelques tests pour estimer les meilleurs paramètres sur la masse des particules et la mise à l'échelle à l'écran afin que tout puisse être aisément visualisable. De même, nous n'avions pas forcément d'idées d'ordres de grandeur pour les paramètres des forces comme la constante d'élasticité d'un ressort et nous avons donc pu observer des comportements "extrêmes".
Le calibrage des paramètres des ressorts entre les particules du blob ont été particulièrement laborieux mais permet d'avoir un résultat cohérent et améliore grandement le rendu visuel.

Astuces de Programmation

La grande ressemblance entre les différentes forces appliquées nous a grandement facilité le travail. En effet, par de simples copier-coller des diverses classes nous avons rapidement pu implémenter les diverses classes. Celà a notamment été pertinent avec les diverses forces appliquées par les ressorts et les élastiques qui se ressemblent beaucoup.

Pour simuler la longueur fixe des ressorts du blob au lieu de créer une nouvelle force nous avons utilisé des ressorts parfaits et ajouté un cable entre le noyau du blob et les particules avec une taille fixe définie en fonction de la longueur du ressort:

Choix de Conception

Nous avons suivi les recommandation du cours pour l'implémentation des différentes parties de notre simulateur Physique. Nous listons les forces en présence et les associons aux particules auxquelles elles s'appliquent dans un registre d'enregistrement. Nous parcourons ensuite le registre pour calculer les forces appliquées et les ajouter à l'accumulateur de la particule concernée. Cet accumulateur correspond donc à la résultante des forces qui est utilisé lors de la deuxième loi de Newton (cf. phase 1). L'update effectué, il faut alors vider le registre et faire l'intégration de l'accélération de chaque particule pour finalement passer à la frame suivante.

Les câbles ont été traités comme il a été recommandé : nous avons créé des collisions virtuelles quand le câble se tendait. Pour les tiges nous avons de la même façon suivi les recommandations en considérant que l'interaction entre deux particules liées par un câble était un contact au repos.

Nous avons décidé de traiter le blob comme une particule centrale ("core") que l'on peut déplacer, le reste du blob se déplace alors simplement du fait de la liaison que l'on crée entre ce noyau et les autres particules reliées. Cette liaison se fait à l'aide de ressorts entre deux particules pour garder cet aspect "souple" et en déformation du blob.
En ce qui concerne les mouvements du blob, ceux-ci sont résolus via l'application d'une force sur son noyau ce qui entraîne donc une inertie importante et une impression de "molesse" dans ses déplacements malgré leur intensité.

Nous avons utilisé une double boucle de détection de collisions pour permettre la fusion d'une particule avec le blob et le fonctionnement correct de la collision. Cette seconde boucle est en fait une détection effectué en amont de la boucle de détection et résolution des collisions, elle fait en sorte de parcourir les collisions entre les particules du blob et l'ensemble des particules du monde et en cas de détection d'une collision entre une particule du blob et une particule en dehors du blob on ajoute la particule extérieure à la liste des particules du blob et on la relie au noyau de ce dernier.

Journal de Développement (3)

Mathématiques et Physiques pour le Jeu Vidéo

Journal de Bord

Semaine 9 :

Mercredi : Nous avons mis à jour notre projet avec les retours que nous avons eu au cours de la présentation et avons restructuré notre dépôt git. Nous avons aussi commencé l'implémentation des Matrix3 ainsi que des Quaternions. Nous avons écrit en parallèle les tests pour ces classes afin d'améliorer notre implémentation en temps réel.

Semaine 10 :

Mardi : Amélioration des tests sur les Quaternions pour notamment tenir compte des imprécisions dûes aux opérations sur des flottants et correction des calculs effectués avec le constructeur d'Euler pour améliorer les performances.
Mercredi : Nous avons effectué la correction des tests sur les Matrix3 et la correction de Matrix3 au vu du résultat des tests. Nous avons également implémenté Matrix4 et avons corrigé certaines erreurs sur les Quaternions.

Semaine 11 :

Mercredi : Changement des getters et setters pour les Matrix car le système précédent causait des erreurs. Nous avons également commencé l'implémentation des RigidBodies. La scène de visualisation a été améliorée avec l'ajout d'une skybox, de light et le déplacement de la caméra qui permet de mieux visualiser ce qu'il se passe. Nous avons aussi ajouté un système pour effectuer des tirs de RigidBodies. Nous avons donc implémenté deux deux addons : skybox et FirstPersonController.

Semaine 12 :

Lundi : Nous avons ajouté la somme et la normalisation au Quaternion et nous avons également amélioré les RigidBodies.
Mardi : Nous avons amélioré le modèle du RigidBody pour qu'il puisse prendre une forme quelconque. Nous avons également mis à jour les registres et les forces pour qu'elles puissent s'appliquer sur les RigidBodies en plus des particules. Une fois ceci fait, nous avons pu créer des RigidBodies de formes variées comme une guitare, une voiture, une bouteille, une chaise ou encore une table.
Mercredi : Nous avons pour finir implémenter la dynamique rotationnelle pour pouvoir faire tourner les RigidBody. Nous avons également mis à jour le ofApp pour donner la possibilité de choisir l'objet à lancer, ainsi que les forces qui lui seront appliquées. Il est également possible d'ajouter une force supplémentaire comme un spring ou un elastic. Enfin, nous avons permis de mettre la simulation en Pause, de la relancer et d'activer ou désactiver

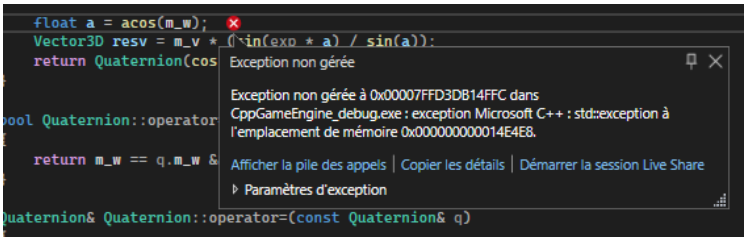
l’impulsion initiale. Pour faciliter la prise en main de tous ces éléments nous avons ajouté un GUI donnant les commandes à l’utilisateur.ice. Enfin nous avons ajouter un marqueur montrant la position d’où a lieu l’impulsion initiale

Difficultés rencontrées

Les approximations sur les flottants nous ont posé de nombreux problèmes. Une grande partie de notre temps a été utilisée afin de déboguer des erreurs de flottants, survenant sans comprendre exactement pourquoi.
Les floats nous ont également posé problème pour le calcul des arccos() car après plusieurs opérations, les approximations font que la norme des rotations diffère de 1 (demandant alors de normaliser les Quaternions après chaques opérations).

Les getters et setters sur les Matrix à l'aide de crochet causait de nombreux problèmes qui ont pu être observés via les tests. Nous avons donc changé notre gestion des accesseurs en créant une fonction "at" qui permet de réaliser efficacement l'opération.

Afin de tester la classe Quaternion et les fonctions qui y sont définis nous avons dû effectuer certains calculs pour vérifier que les résultats obtenus par notre implémentation sont corrects. Celà à poser quelques difficultés, en effet, pour la fonction SLERP nous avons utilisé au départ un site qui donnait des résultats erronés. Nous avons donc dû croiser plusieurs résultats pour en trouver des "valides" et donc nous assurer de la validité de notre calcul.



Toujours pour les Quaternions, nous avons laissé une exception non gérée sur l'exponentiation. Celle-ci n'empêche pas la compilation, mais nécessite de faire "Continuer" pour poursuivre l'exécution.

Nous avons voulu créer une méthode static permettant de créer une Matrix3 à partir d'un Quaternion. Cependant, nous avons eu des erreurs de compilation non annoncées par VisualStudio nous poussant donc à investiguer pour nous rendre compte que l'erreur provenait de dépendances circulaires.

Le passage des coordonnées locales vers les coordonnées dans le monde nous ont posé des difficultés pour les ressorts et les élastiques.

Astuces de Programmation

Pour les Quaternions nous avons écrit une fonction fixFloat() qui permet de corriger les approximations dues à la manipulation de float. De même la fonction EqualsWithTolerance() permet de tester l'égalité sur les Quaternions de manière souple pour tenir compte des approximations.

Pour gagner du temps, nous nous sommes appuyés sur Vector3D pour implémenter les Quaternions notamment les produits vectoriels.

Choix de Conception

Nous avons créé un nouveau dossier pour regrouper les structures de données dont les matrices, les quaternions et les vector3D.
Pour le calcul du déterminant d'une matrice 4x4 nous avons utilisé le calcul du déterminant de matrice 3x3 pour avoir les cofacteurs de la matrice 4x4 et ainsi avoir le déterminant souhaité.

Pour la création des RigidBody, nous avons décidé de prendre une liste de OfPrimitives afin de créer le mesh global qui lui est associé. Ensuite nous avons défini ses primitives en tant qu'enfant d'une sphère, celle-ci correspondant au centre de masse.

Journal de Développement (4)

Mathématiques et Physiques pour le Jeu Vidéo

Journal de Bord

Semaine 13 :

Mercredi : Choix des implémentations.

Semaine 14 :

Mercredi : Nous avons réalisé l'implémentation de base de l'Octree. Nous avons aussi implémenté les sphères et commencé l'implémentation des box. Nous les avons définis comme héritant de la classe Collider. Enfin, pour faciliter la lecture des tests, nous les avons séparés en plusieurs classes.
Vendredi : Nous avons travaillé sur l'affichage des octrees et la gestion de leurs feuilles ainsi que sur la gestion du centre d'inertie du RigidBody.

Semaine 15 :

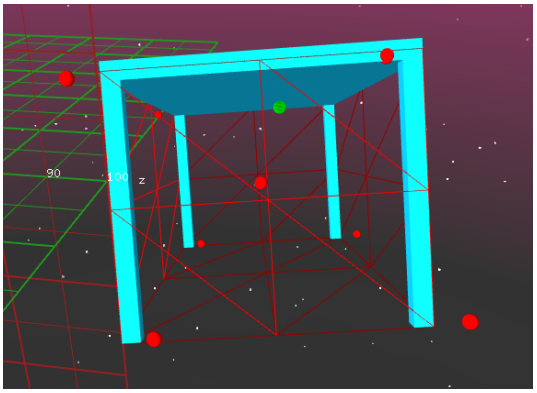
Lundi : Nous avons commencé à travailler sur la gestion des collisions entre deux boîtes englobantes, la plupart des cas ont été traités sauf les contacts entre les arêtes des boîtes englobantes.
Mardi : Nous avons finalisé l'implémentation des collisions entre deux boîtes englobantes.
Mercredi : Nous avons travaillé sur les colliders pour donner à des objets une boîte englobante et une sphère en passant par la création d'une nouvelle classe : GameObject. Nous avons mis en place le CollisionHandler pour les RigidBody à partir de tout ce qui a été réalisé précédemment. Nous avons également dû ré-implémenter la classe plane car sa gestion des collisions n'était plus à jour. Nous avons en outre continuer à travailler sur l'affichage des différents éléments qui interviennent dans la détection de collision avec notamment l'affichage des box colliders, des sphères englobantes et des sommets.
De même, nous avons implémenté les plans et modélisé notamment le sol.

Difficultés rencontrées

Nous avons rencontré des difficultés à lier les RigidBody et les Colliders associés (boîte et sphère) car nous avions des problèmes de Cross References. Nous avons donc résolu ce problème en créant une classe GameObject qui contient le rigidbody et les deux colliders qui lui sont associés et gérer tout à partir de ce GameObject.

Les plans sont fonctionnels mais la résolution des collisions est mal gérée avec ces derniers.

Nous avons obtenu de légères différences entre la boîte de collision souhaitée et la boite de collision réelle liée à des erreurs d'approximation de float comme on peut le voir sur l'image ci-dessous.



Astuces de Programmation

Le problèmes des doubles inclusions avec les classes nous posé quelques difficultés pour la gestion des collisions. En effet, pour gérer les collisions boite/plan et plan boite nous avions besoin d'inclure les classes l'une dans l'autre. Pour palier ce problème nous avons donc dû gérer uniquement ce cas de collision à partir des boites.

Choix de Conception

Nous avons défini une classe Collider de laquelle hérite les classes Sphere, Plane et Box. Cela permet de définir des propriétés communes à ces trois structures et de pouvoir les manipuler plus facilement (translation, centre, etc.).

Nous avons choisi d'utiliser des octrees plutôt que des BSPtree car celà nous semblait plus aisé à implémenter.

Pour la gestion des collisions entre RigidBodies nous avons mis en place la classe RBCollisionHandler. Cette classe nous permet de regrouper toutes les structures précédemment programmées et d'ainsi résoudre les collisions.
Pour celà, à chaque frame, nous commençons par la création d'un nœud racine de l'Octree puis nous lançons la division de celui-ci. Ensuite nous récupérons toutes les feuilles et à

l'intérieur de celui-ci et effectuons alors la gestion des collisions des GameObjects que contient la feuille. Pour clan, nous effectuons d’abord une détection large avec les sphères englobantes des GameObjects puis avec leur boites englobantes. Si la collision est validée, nous calculons alors les impulsions créées.