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 : Correction de bug autour du blob. Les ressorts ont également été retravaillés.

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 additionnions 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 dû 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".

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.

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.