Document de conception moteur physique : Rotations,orientation,moment d'inertie et RigidBody

Groupe C

Réalisé par :

Antoine De Saint Martin DESA09050103

Victor Guiraud GUIV09029901

Axel Guza GUZA16069906

Table des matières

La conception **3**

Rotations et orientation **3**

Matrice3x3/Matrice3x4/Quaternion 3

Rigidbody 3

Forces **4**

Démonstrations **5**

Cube 5

Demo 5

Main 5

Les difficultés rencontrées **6**

Les problèmes existant : **7**

Les problèmes résolus : **7**

A prévoir : **7**

Le groupe : **8**

Source : **9**

Les méthodes : 9

Les librairies : 9

Les ressources de répertoires GIT et livre : 9

Aides sur les librairies : 9

Contexte : Dans le cadre du cours « 8INF935 – Mathématique et Physique pour le jeu vidéo » nous devons réaliser un moteur physique.

La conception

Durant la conception nous avons pris de l'inspiration d'informations du cours, de notre professeur, d'internet et de nos connaissances afin concevoir nos algorithmes. Comme durant la phase 2 nous avons essayé de nous éloigner de l'algorithme présenté dans le livre « I. Millington. Game physics engine development».

Dans chacune des parties de développement de cette phase, nous sommes parties initialement depuis les conseils d'implémentation du cours pour créer nos classes et implémenter nos fichiers .h.
Nous avons ensuite fait de notre mieux pour retranscrire les mathématiques sous forme de code C++.
Une grande partie de l'algorithme du rendu final de cette phase va diverger de l'implémentation initiale.

Rotations et orientation

Cette partie va comprendre la simulation de mouvement de rotation et d'orientation d'une particule dans un plan 3D.

Matrice3x3/Matrice3x4/Quaternion

Ces classes représentent de simples matrice, avec les différentes opérations (inverse, transposition, transformation, changement dans l'espace…) qui leur seront appliquées. De même avec les quaternions.

Rigidbody

Il en va de même pour notre corps rigide où l'on retrouve notamment une fonction d'intégration pour appliquer des modifications au corps de la particule, avec l'ajout de force et de transformation entre le monde global et le monde local.

Transform

Un transform est un élément associé à notre RigidBody, en comparaison à un moteur de jeu tel que Unity, il représente les caractéristiques de l'objet dans son environnement (taille/position/rotation) dans le monde, il ne comporte aucune information quant à la forme, la masse, le poids… ces éléments sont du ressort d'un RigidBody. Le transform est générique et pourra être utilisé pour n'importe quel objet de notre monde.

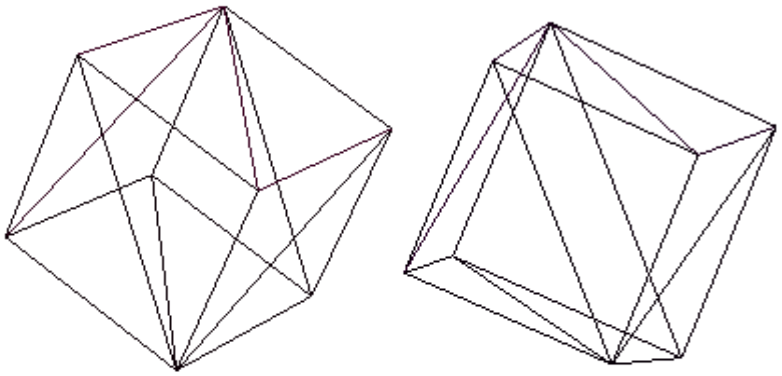
Shader/ShapeRenderer

Afin d'afficher correctement notre cube, nous avons pris l'initiative d'utiliser plusieurs possibilités offertes par OpenGL, nous avons récupéré des fichiers de shaders s'appliquant à différentes formes (ici des cubes), ces shaders offraient un dynamisme coloré sur notre cube, finalement nous n'en n'avions pas l'utilité, le rendu d'un cube noir sur fond blanc est nettement plus adapté.

Pour rendre notre cube, le script ShapeRenderer.cpp permet d'appliquer à la VAO de notre cube les paramètres suivants : position/rotation/echelle, sa méthode render() s'occupe d'effectuer tous ces changements et est donc appelée à chaque frame.

Démonstrations

Cube



Rotation du cube - Moteur Physique

Demo

Dès la compilation effectuée, le lancement de notre démonstration est possible sans besoin d'apporter des changements. Vous pouvez observer un cube se déplaçant sur écran blanc.

Vous pouvez observer plusieurs configurations, pour cela il suffit de changer la valeur de la variable APPLICATION_WANTED, sa valeur peut être mise entre 0 et 4 pour tester chacun des fonctionnements. La fonction d'initialisation est *ChooseWitchApplicationIsWanted()*, elle se situe dans "*Main_rendu3.cpp*".

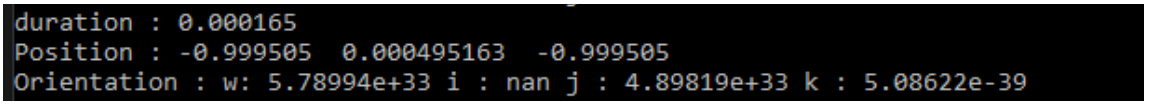
Vous pouvez changer manuellement les valeurs des vecteurs 3D passés en paramètres du constructeur de *demoCube* appelé dans le *switch case* de *ChooseWitchApplicationIsWanted()*. Comme expliqué ci-dessous dans la rubrique "Les problèmes rencontrés" nous n'avons pas réussi à utiliser IMGUI pour ce rendu, ainsi nous ne pouvons pas faire varier les paramètres lors de l'exécution, ça aurait été un atout non négligeable.

Les difficultés rencontrées

-Dans le cadre de l'implémentation du RigidBody, ce qui a été le plus difficile était de retranscrire les opérations mathématiques de la fonction d'intégration avec les méthodes que nous avions déjà mis en place.
Dans un second temps il s'agissait de comprendre comment réellement implémenter les fonctions de transformation entre le monde global et le monde local.

-Nous avons passé du temps à faire fonctionner le système [Cube / ShapeRenderer / Shader] , cette étape nous a permis cependant de mieux connaître l'ensemble du fonctionnement d'OpenGL. Une fois le cube affiché, nous pouvions debugger, ainsi les tests sur le fond attendu ["RigidBody", "Quaternion", "Matrix33/34"] sont arrivés relativement tard par rapport à la date de rendu.

-Une fois notre cube existant, affiché et avec des forces et rotations appliquées, nous avions un cube qui se déplaçait et tournait très lentement dans notre espace (des valeurs de rotation de l'ordre de 10^ + ou - 40).



Erreur de rotation - Rotation du cube - Moteur physique

Dans notre tentative de résolution de ce problème nous avons chacun passé plusieurs heures à débugger et rechercher pas à pas l'origine de ce comportement. L'erreur était causée par de simples erreurs de passage de paramètres dans certaines fonctions (par exemple nous avions un value[5] au lieu de value[7]) et le fait que nous ne transférons jamais les valeurs w i j k a notre quaternion. Une fois cela identifié nous avons donc résolu le problème.

Les problèmes existant :

-Dans certains cas, lors de la simulation, la forme du cube est altérée et une ou plusieurs faces voient leurs dimensions modifiées. Certains paramètres doivent varier légèrement, nous n'avons pas trouvé dans quel circonstance ça arrivait.

-Nous n'utilisons plus IMGUI, ce n'est pas un choix de notre part, lors des modifications de OpenGL pour ce rendu, IMGUI ne fonctionne plus, nous avons préféré ne pas passer trop de temps sur ce problème. En effet l'objectif était de montrer le déplacement d'un cube sur l'écran avec orientation et rotation, le projet en l'état montre cet objectif, IMGUI aurait été un plus.

Les problèmes résolus :

-Nous n'avions pas fait attention pour les rendus 1 et 2, la masse n'était pas utilisée dans les calculs, elle l'est à présent pour les Rigidbody.

A prévoir :

-Ajouter la masse dans les calculs gérant les particules, nous n'avons pas revu cette partie.

-Pour les générateurs de forces, lorsqu'ils seront rendus génériques pour Particle et RigidBody, il faudra mettre en place l'utilisation des accumulateurs de forces car dans nos rendus précédents, nous agissons en dur directement sur la vélocité.

-Nous n'avons pas adapté les générateurs de forces au RigidBody, à faire en vue d'amélioration.

-Nous n'avons pas adapté le système de résolution de contacts au RigidBody, à faire en vue d'amélioration.

Le groupe :

- Nous nous sommes mieux organisés pour ce rendu, tout le monde a participé et la communication s'est améliorée.

Sources

- https://github.com/ocornut/imgui
- https://www.opengl.org/
- https://moodle.uqac.ca/mod/url/view.php?id=653865

Les méthodes :

-Quaternion.h : les méthodes ToEuleur() et SetByEulerRotation() ont été récupérées sur : https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles

-demoCube.h : la méthode CubeBis() est inspirée du site : https://sciences-indus-cpge.papanicola.info/Matrice-d-inertie-des-solides

Les librairies :

-https://www.sfml-dev.org/ : SFML librairie actuellement utilisée.

-https://www.opengl.org/resources/libraries/glut/glut_downloads.php : librairie GLUT appairée à OpenGL.

-https://www.just.edu.jo/~yaser/courses/cs480/opengl/Using%20OpenGL%20and%20GLUT%20in%20Visual%20Studio.htm : utilisation de GLUT et OpenGL sur Visual Studio.

-https://www.dll-files.com/glut32.dll.html : récupération .DLL GLUT.

-https://glew.sourceforge.net/ : récupération .DLL GLEW.

Les ressources de répertoires GIT et livre :

-https://github.com/idmillington/cyclone-physics/blob/master/src/demos/main.cpp : répertoire GIT du livre associé au cours, Millington. Game physics engine development.

-https://ebookcentral.proquest.com/lib/uqac-ebooks/home.action : livre de référence du cours.

Aides sur les librairies :

-https://learnopengl.com/Getting-started/Hello-Triangle : commencement sur OpenGL.

-https://dokipen.com/modern-opengl-part-3-hello-triangle/ : commencement sur OpenGL.

https://opengl.developpez.com/tutoriels/apprendre-opengl/ : implémentation supplémentaire opengl

-https://en.cppreference.com/w/cpp/header : librairies de bases c++.

-https://www.developpez.net/forums/d1108488/applications/developpement-2d-3d-jeux/api-graphiques/opengl/dessiner-repere-xyz/ : renseignements élaboration repère avec OpenGL.

-https://www.opengl.org/resources/libraries/glut/spec3/node14.html : renseignement sur GLUT.

-https://cpp.hotexamples.com/fr/examples/-/-/GET_XBUTTON_WPARAM/cpp-get_xb-utton_wparam-function-examples.html : récupération interraction souris.

-https://learn.microsoft.com/en-us/windows/win32/learnwin32/keyboard-input : récupération interraction clavier.

-https://en.cppreference.com/w/cpp/chrono/system_clock : gestion du temps en c++ avec l'entête chrono.

-https://community.khronos.org/t/glcolor3f-not-working/69751/2 : coloration des objets avec OpenGL.