Minot Eva
Cavani Nicolas
Moënne-Loccoz Célestin

Journal de bord

Phase 4 : Ajout d'un système élémentaire de résolution de collisions

Nous avons commencé cette dernière phase en améliorant certaines choses de la phase précédente. Nous avons ajouté des affichages de debug pour mieux visualiser les valeurs que prennent nos RigidBody (engine) et nos RenderedMesh (affichage). Surtout, nous avons corrigé un problème sur le passage de la matrice de transformation de la physique à l'affichage qui était que les mesh affichés se déformaient lorsqu'on appliquait une rotation.

1) Broad phase

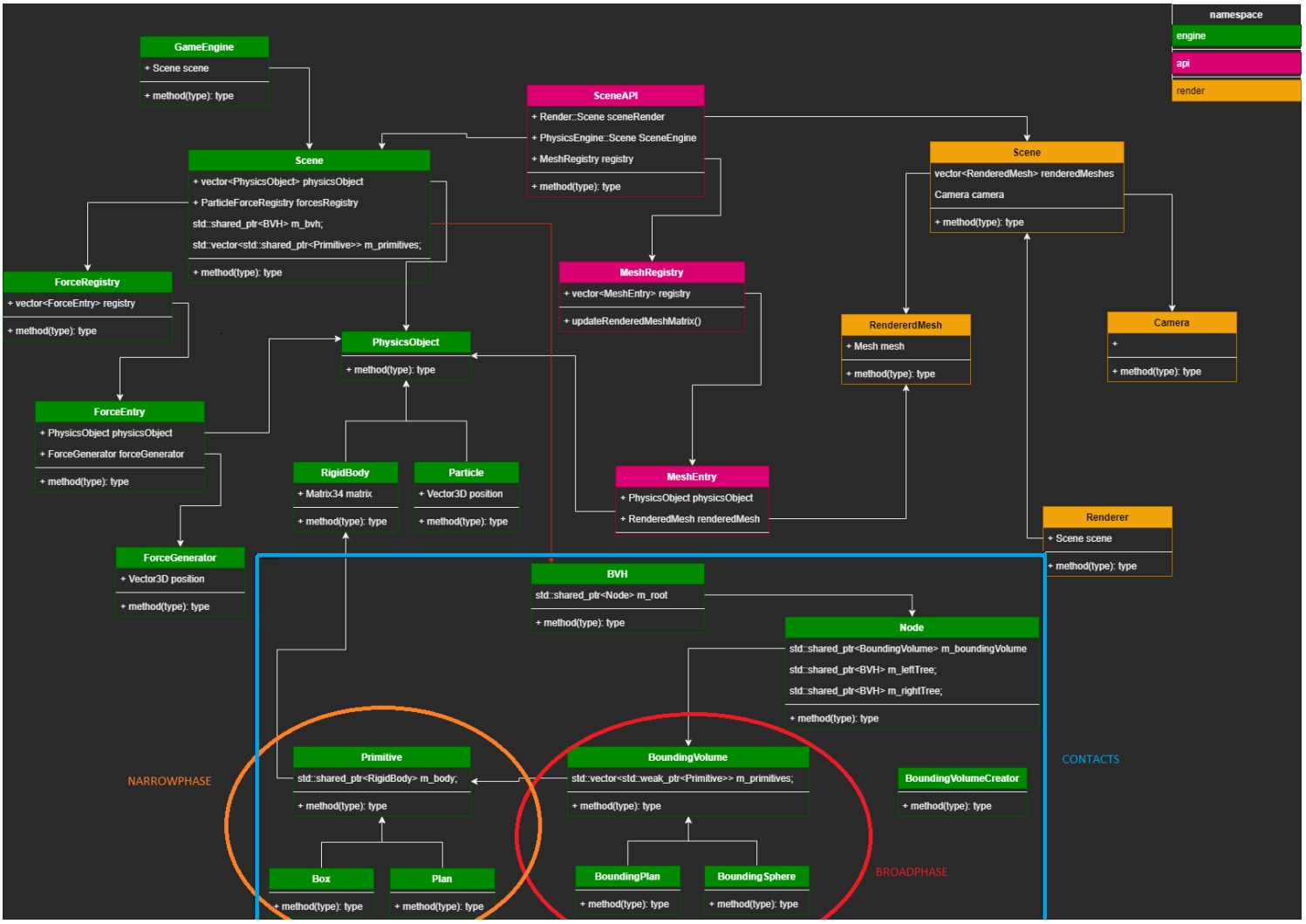
Concernant les tâches associées à la phase 4 maintenant, nous avons choisi de réaliser un BVH pour effectuer la détection de collisions.
Voici comment nous l'avons implémenté.
Une classe BVH qui contient simplement un Node ainsi que les méthodes qui permettent de le construire. Node contient un BoundingVolume des objets de ce nœud et de ceux de ces enfants, il contient aussi deux pointeurs vers les nœuds suivants de l'arbre. BoundingVolume est une classe abstraite, BoundingSphere et BoundingPlan dérive de celle ci, ces classes permettent d'englober les objets de la scène dans une sphère ou bien derrière un plan. Ils sont utiles lorsque le BVH est utilisé pour réaliser l'étape de BroadPhase lors de la phase de détection de collisions. Un BoundingVolume contient une liste de Primitive que l'on détail par la suite.
Lors de la création du BVH, la liste de toutes les Primitive qui constituent les éléments de la scène est donnée en paramètre. Il se charge ensuite de diviser cette liste en deux sous listes en regroupant ensemble les Primitives les plus proches les unes des autres. De manière récursive, ces deux sous listes sont ensuite utilisées de la même façon pour créer les nœuds suivants du BVH. Cela est répété jusqu'à ce que l'arbre ne contienne plus que des feuilles. Au cours de la détection des collisions dans la BroadPhase, et donc à chaque frame de l'engin physique, un nouveau BVH est créé.

2) Narrow phase

Les primitives correspondent à tout élément de notre moteur physique pouvant créer des contacts. Nous pouvons associer un RigidBody à une ou plusieurs primitives, n'étant pas forcément centrées sur le RigidBody. Nous pouvons également définir une primitive indépendante de tout RigidBody. Nous disposons actuellement de deux

types de primitives : Plan et Box, qui dérivent donc de la classe abstraite Primitive. Dans notre cas, nous allons associer une simple Box à chacun de nos RigidBody.
Ces primitives sont utiles lors de la NarrowPhase une fois qu'une potentielle collision a été détectée dans la BroadPhase. Pour générer les contacts réels, on dispose d'une classe nommée RigidBodyContactGenerator qui va itérer sur chacune des paires de primitives pouvant entrer en contact, appeler leurs méthodes de génération de contact, en fournissant également une référence vers notre objet de type RigidBodyCollisionData qui contiendra toutes nos collisions générées. Nous devrons alors définir, dans chacune des primitives, comment calculer les informations nécessaires pour la collision avec tous les autres types de primitives.

Voici le nouveau diagramme UML associé à la structure de notre projet :



3) Contacts

Une fois que tous nos contacts possibles ont été traités, on aura éventuellement des contacts créés dans notre objet de type RigidBodyCollisionData. Ce sera alors au tour de notre objet de type RigidBodyContactResolver d'analyser ces collisions générées pour ajouter les bonnes forces aux bons RigidBody. Cependant, dans

notre cas, nous devons simplement détecter s' il y a eu collision ou non. Nous allons simplement retourner vrai si on a au minimum un contact dans notre liste de contacts.