Moteur de jeux

TP5

Introduction

Dans ce TP, la physique du rebondissement d'un cube sur un plan a été implémentée.

Compilation et exécution

La compilation du programme sur Linux est effectuée à partir d'un simple `make clean` puis `make -j`. Le programme peut ensuite être exécuté avec `./main`.

Commandes

Les commandes utilisées dans ce TP sont les suivantes :

- **Z** et **S** : Déplacements avant/arrière
- **Q** et **D** : Déplacements gauche/droite
- **E** et **A** : Déplacement haut/bas
- **Espace** : Lancement d'un cube
- **TAB** : Changement du modèle physique

TP5

Ajout d'objets dynamiques

L'ajout d'objets dynamiques dans notre scène se fait d'abord par l'implémentation de la classe `RigidBody`, contenant notamment la position, la rotation et les vélocités linéaires et angulaires du corps solide, mais également un `TransformTree*` (nœud de notre graphe de scène) qui permet d'appliquer aux objets rendus les transformations du corps solide qui leur est associé. Ici, la touche **Espace** permet de créer dans la direction de la caméra un cube avec une vélocité de $8\,m.s^{-1}$, en supposant ici que les unités de distance en mètres (m), et avec une rotation relative à la caméra.

Cette classe possède une fonction `update` prenant en paramètre une intervalle de temps Δt, qui met permet de mettre à jour à chaque itération de notre boucle de jeu les différents corps solides présents et les transformations du graphe de scène. En l'occurrence, sa position $\boldsymbol{p}$, sa rotation $\boldsymbol{\theta}$ et leurs vélocités linéaire $\boldsymbol{v}$ et angulaire $\boldsymbol{\omega}$ sont mises à jour d'après les équations du mouvement. On note également $\boldsymbol{a}$ et $\boldsymbol{\alpha}$ l'accélération linéaire et angulaire respectivement, sous forme d'angles d'Euler.

$$\boldsymbol{v}(t+\Delta t)=\boldsymbol{v}(t)+\boldsymbol{a}(t)\Delta t$$
$$\boldsymbol{p}(t+\Delta t)=\boldsymbol{p}(t)+\boldsymbol{v}(t)\Delta t$$
$$\boldsymbol{\omega}(t+\Delta t)=\boldsymbol{\omega}(t)+\boldsymbol{\alpha}(t)\Delta t$$
$$\boldsymbol{\theta}(t+\Delta t)=\boldsymbol{\theta}(t)+\boldsymbol{\omega}(t)\Delta t$$

Ici, nous estimons que $\boldsymbol{\alpha}(t)=\mathbf{0}$, donc seule la force gravitationnelle agit sur l'objet, d'où $\boldsymbol{a}(t)=\boldsymbol{g}=(0,-9{,}81,0)$. Nous l'implémentons assez facilement dans la fonction `update` :

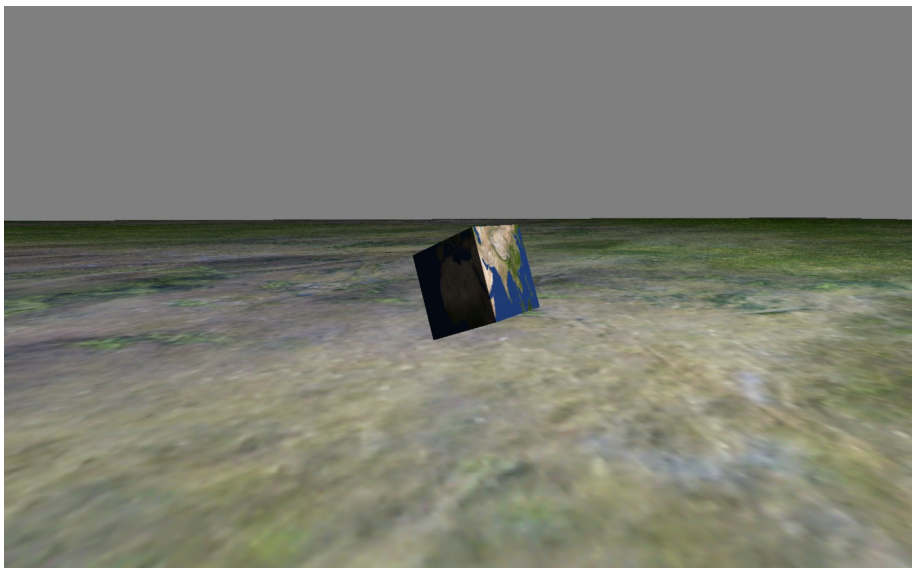
```
void RigidBody::update(float delta) {
    velocity += delta * glm::vec3{0.0f, -9.81f, 0.0f};
    position += delta * velocity;
    rotation = glm::quat{delta * angularVelocity} * rotation;
    transformTree->transform.setTranslation(position);
    transformTree->transform.setRotation(rotation);

    // ...
}
```

Détection de la collision boîte-plan

Le plan est ici un plan horizontal passant par $y=0$. Tandis la carte d'altitude sert de repère pour ce plan, il s'étend en réalité à l'infini. La collision entre notre cube et ce plan-ci est gérée dans la fonction `update` par une simple implémentation d'une collision entre une boîte orientée et un plan aux axes alignés. Elle appelle une fonction `getCollidingPoint` qui calcule en fonction de l'orientation de la boîte, les coordonnées de l'ensemble des points de la boîte, et vérifie si la composante verticale y_p de ces derniers est inférieure à y_{plan}, auquel cas, une collision existe.

Cette fonction permet de déterminer le coefficient de pénétration, c'est à dire $\delta=\max_p\left\{\left|y_p-y_{plan}\right|\right\}$, à appliquer afin de traiter le cas où la boîte pénètre le plan, ainsi que le point de collision $\boldsymbol{P}_c$ qui est une interpolation de tous les points pénétrants.



Réponse simple de la collision

On souhaite faire rebondir la boîte sur notre plan statique. La méthode la plus simple est de simplement déplacer l'objet du coefficient de pénétration δ dans la direction de la

normale (donc vers le haut) puis de déterminer la nouvelle vélocité. Il s'agit ainsi de faire refléter la vitesse relative de ces deux objets par rapport à la normale. Dans notre cas, il s'agit tout simplement d'inverser la composante v_y.

Il est possible d'y intégrer un coefficient de restitution e et de friction f au long de la tangente, donc sur les axes x et z, qui dépendent notamment des matériaux en collision. Cela permet donc de perdre de l'énergie cinétique pendant la collision, ce qui s'apparente aux modèles réels.

$$p_y = p_y + \delta$$
$$\boldsymbol{v} = (fv_x, -ev_y, fv_z)$$

Suite à cela, les cubes lancés rebondissent donc sur le plan.

Réponse complexe de la collision

Un modèle de résolution plus complexe a aussi été implémenté ; en revanche, celui-ci sort de la portée de ce TP, mais il reste disponible en appuyant sur la touche **TAB**.

Il se base sur un modèle basé sur l'impulsion ainsi que sur le modèle de friction de Coulomb simplifié, où deux impulsions sont appliquées à la fois relativement à la normale et à la tangente de la collision. Soit $\boldsymbol{r} = \boldsymbol{P_c} - \boldsymbol{p}$ la différence entre le point de collision et le centre de masse, $\boldsymbol{I} = \frac{c^2}{6}\boldsymbol{I_3}$ le coefficient d'inertie du cube de coté c et $\boldsymbol{v_r} = \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{r}$ la vélocité relative des deux objets, l'impulsion résultant sur la normale est la suivante, avec un objet de masse 1 :

$$j_n = \frac{-(1+e)\boldsymbol{v_r} \cdot \boldsymbol{n}}{1 + \boldsymbol{n} \cdot (\boldsymbol{I}^{-1}(\boldsymbol{r} \times \boldsymbol{n}) \times \boldsymbol{r})}$$

Et sur la tangente :

$$j_t = \frac{f\,\boldsymbol{v_r} \cdot \boldsymbol{t}}{\boldsymbol{r} \cdot (\boldsymbol{I}^{-1}(\boldsymbol{r} \times \boldsymbol{t}) \times \boldsymbol{t})}$$

Avec $-j_n \leq j_t \leq j_n$, sinon j_t est contraint (clamp) entre ces valeurs-ci.

L'impulsion est donc appliquée de cette manière sur les vélocités, selon une direction $\boldsymbol{d}$ (normale ou tangente) :

$$\boldsymbol{v} = \boldsymbol{v} + j_d \boldsymbol{d}$$
$$\boldsymbol{\omega} = \boldsymbol{\omega} + \boldsymbol{I}^{-1}(r \times (j_d \boldsymbol{d}))$$

Note : certains rebonds peuvent apparaître pour des objets au repos du à des imprécisions numériques dans le modèle.