Moteur de jeux

Compte-rendu TP5

Mouvements et collisions

Louis Jean
Master 1 IMAGINE
Université de Montpellier
N° étudiant : 21914083

2 avril 2024

Table des matières

1 Introduction

Dans ce TP, nous nous intéressons aux mouvements d'un objet et aux collisions de ce dernier avec un terrain. Dans un premier temps, nous verrons comment donner une vitesse à l'objet, puis nous introduirons la force de gravité, avant d'enfin résoudre correctement la collision de l'objet avec le plan.

Pour ce TP, **que j'ai réalisé après le TP sur les caméras**, j'ai repris l'intégralité du code de la caméra. Il faut donc appuyer sur `T` pour cacher le curseur et prendre le contrôle de la caméra avec les touches `Z,Q,S,D` pour les translations et la souris pour les rotations. Rappuyer sur `T` pour désactiver la caméra et faire réapparaître le curseur. J'ai aussi repris le code du TP précédent (TP4) pour les classes `Plane` et `PlaneCollider`.

2 Créer un cube

Avant toute chose, j'ai du créer une classe `Cube`, qui m'a permis de générer et afficher un cube à l'écran. Une instance de `Cube` dispose d'un centre, d'une taille (de côté), d'une masse et d'une vitesse (nulle à l'initialisation).

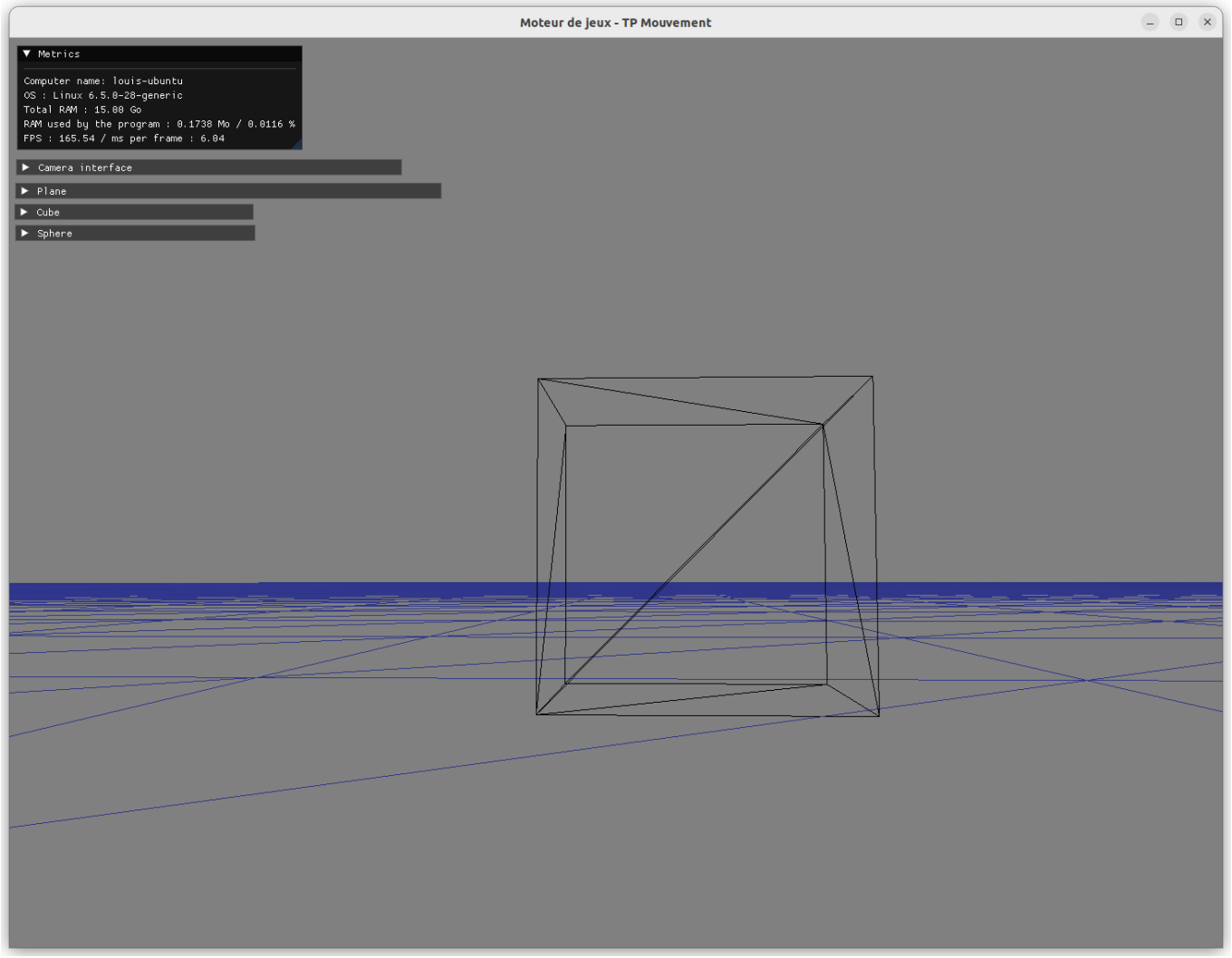

Figure 1: Cube créé et affiché

3 Donner une vitesse au cube

Pour donner une vitesse à l'objet, j'ai décidé d'écrire une fonction `launchCube`, qui lorsqu'elle est appelée, jette un cube depuis la position de la caméra, dans la direction du front de cette dernière, à une vitesse choisie par l'utilisateur (via `ImGui`).

```
void Cube::launchCube(glm::vec3 position, glm::vec3 direction, float speed) {
    this->center = position;
    this->velocity = glm::normalize(direction) * speed;
}
```

J'ai lié l'appel de cette fonction à la touche `Espace`.

Ensuite, pour effectivement voir du mouvement, il faut mettre à jour la position du cube à chaque frame. Pour ceci, j'ai créé une fonction `update`.

```
void Cube::update(float delta_time) {
    center += velocity * delta_time;
}
```

Grâce à mon graphe de scène implémenté lors du TP3, je translate le cube à chaque frame de cette manière, puisqu'un mouvement du cube se traduit par le déplacement de son centre.

```
cube_node->transform.translation = cube->getCenter();
```

Une fois ces étapes achevées, on peut appuyer sur `Espace` et voir le cube se déplacer, sans jamais retomber (si on ne l'a pas jeté en direction du sol, sinon collision) ni ralentir. C'est ici qu'entre en jeu la force de gravité.

4 Ajouter la force de gravité

Pour ajouter la force de gravité, une simple modification à la fonction `update` suffit. En effet, on applique le principe de la deuxième loi de Newton pour en déduire l'accélération de notre objet :

$$F = ma \iff mg = ma \iff g = a$$

J'ai fait le choix de prendre l'intensité de l'accélération terrestre.

```
void Cube::update(float delta_time) {
    glm::vec3 gravity = glm::vec3(0.0f,-9.81f,0.0f);
    glm::vec3 acceleration = gravity;
    velocity += acceleration * delta_time;
    center += velocity * delta_time;
}
```

Avec ceci, le cube tombe bien vers le sol.

5 Collision cube plan

Pour effectuer la collision cube plan, j'ai repris le même code que dans mon TP4, à la différence près que cette fois-ci je n'ai pas snappé la position du cube à la hauteur du plan directement, j'ai pris soin de d'abord vérifier si le cube était au dessus ou en dessous du plan.

```
void PlaneCollider::check_collision_with_cube(Cube &cube) {
    glm::vec3 cubePosition = cube.getCenter();
    float cubeBaseY = cubePosition.y - cube.getHalfSize();
    float currentHeight = plane.get_height_at_position(cubePosition.x,
   cubePosition.z);
    if (currentHeight >= cubeBaseY) {
        glm::vec3 newPosition = glm::vec3(cubePosition.x, currentHeight + cube.
   getHalfSize(), cubePosition.z);
        cube.setCenter(newPosition);
        glm::vec3 newVelocity = cube.getVelocity();
        // Annulation de la vitesse
        newVelocity = glm::vec3(0.0f,0.0f,0.0f);
        cube.setVelocity(newVelocity);
    }
}
```

De cette manière, le cube s'arrête immédiatement lorsqu'il rentre en contact avec le plan.

6 Résolution de la collision avec mise à jour de la vitesse réaliste

Niveau réalisme, l'arrêt immédiat du cube lors de la collision avec le plan, ce n'est pas terrible. C'est pourquoi j'ai amélioré cet aspect en prenant en compte des coefficients de frottement et de rebond. Pour cela, j'ai ajouté deux attributs dans ma classe `Plane` : `friction_coefficient` et `restitution_coefficient`. Le coefficient de friction représente la force de frottement avec le plan, tandis que le coefficient de restitution détermine la force de réaction du support de ce dernier. En prenant en compte ces deux nouvelles données, j'ai progressivement mis à jour la vitesse lors de la collision.

```
void PlaneCollider::check_collision_with_cube(Cube &cube) {
    glm::vec3 cubePosition = cube.getCenter();
    float cubeBaseY = cubePosition.y - cube.getHalfSize();
    float currentHeight = plane.get_height_at_position(cubePosition.x,
   cubePosition.z);
    if (currentHeight >= cubeBaseY) {
        glm::vec3 newPosition = glm::vec3(cubePosition.x, currentHeight + cube.
   getHalfSize(), cubePosition.z);
        cube.setCenter(newPosition);
        glm::vec3 newVelocity = cube.getVelocity();
        // Inversion et abaissement de la vitesse y selon le coefficient de
   restitution
        newVelocity.y = -newVelocity.y * restitution_coefficient;
        // Application de la friction sur les composantes x et z
        newVelocity.x *= (1.0 - friction_coefficient);
        newVelocity.z *= (1.0 - friction_coefficient);
        cube.setVelocity(newVelocity);
    }
}
```

Il faut voir les nouveaux coefficients comme des pourcentages. Plus le coefficient de friction est haut, plus la surface va accrocher, et donc plus elle va ralentir le cube rapidement. À l'inverse, plus le coefficient de restitution est grand, plus le cube va rebondir contre le plan.

7 Extension à la sphère

D'une manière totalement analogue, j'ai effectué le même travail pour la sphère. Il est possible d'en lancer une depuis la caméra en appuyant sur la touche `C`.

8 Rendu final de l'application

Évidemment, le format ici utilisé ne permet pas d'inclure de vidéo, donc je ne peux pas montrer le résultat d'une façon qui soit intéressante, mais je vous invite à aller essayer le programme par vous-même.

J'ai créé des fenêtres `ImGui` dédiées pour chaque objet de la scène. On peut y régler les coefficients de friction et de restitution du plan, et la vitesse de lancer du cube et de la sphère.

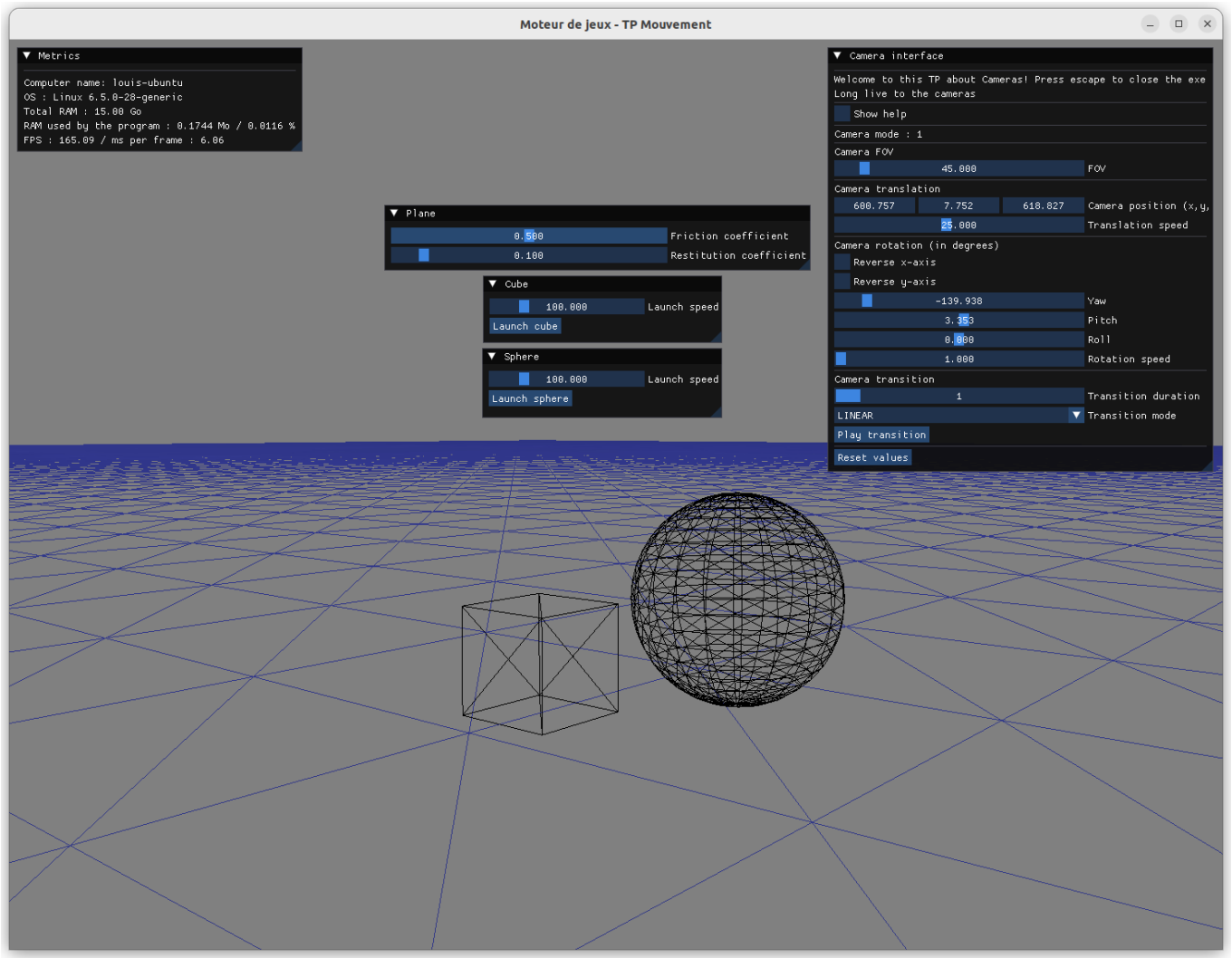

Figure 2: Rendu final de l'application

9 Conclusion

Ce TP m'a permis d'introduire un début de physique dans mon moteur de jeux. Je suis certain que cela me servira pour la suite. J'ai pris du plaisir à réaliser ce travail, en particulier après avoir réussi (j'ai dû lancer une bonne centaine de fois le cube pour apprécier le rendu).

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.