# Documentation Projet Moteur de Jeux
# Destruction d'environnement

BELDJILALI Maxime, CARO Thomas

Mai 2024

## 1 Introduction

Dans ce projet nous utilisons les librairies ReactPhysics3D et Real Time Dynamic Destruction ainsi que des parties de code de TP précédents.

## 2 Organisation

### 2.1 Dossier camera

Le dossier camera comporte ce qui est nécessaire pour la caméra. C'est concrètement les fichiers du TP6 sur la caméra.

### 2.2 Dossier common

Ce dossier dispose de la classe Mesh qui contient les informations de sommets,normales et de triangles d'un maillage ainsi que ses buffers.

Il y a la classe permettant de charger un objet au format obj. On a aussi une classe permettant de générer une texture à partir du chargement d'une image, que l'on fait à l'aide de la librairie stbi incluse dans common. Nous avons aussi la classe Quaternion pour les rotations de caméra ainsi qu'une classe pour charger les shaders.

Pour l'envoi de balle lors du clic gauche et donc la génération de maillage de balles nous avons la classe control.cpp.

Pour finir nous avons jc-voronoi qui nous permet de créer des diagrammes de Voronoi utilisés dans la création de fracture dans FractureGenerator.

### 2.3 Dossier entity

Nous avons la classe Entity qui relie le maillage aux données physiques associées issues de ReactPhysics3D. Chaque entité a :

- Un graphe de scène donc accès au parent et aux enfants
- Une transform(classe Transform)
- Un objet Mouvement permettant de déplacer l'entité(classe Mouvement issue de transform.cpp)
- Un maillage(classe Mesh)
- Un RigidBody(ReactPhysics3D)

Nous avons aussi les DestructibleEntity qui sont celles vouées à être détruites et se briser selon un diagramme de Voronoi. Pour faire cela nous avons aussi la classe FractureGenerator qui permet de briser véritablement un objet et de créer les maillages issus de la destruction de l'objet, qui auront chacun une durée de vie.

Très importante, la classe Scene qui comprendra la caméra et toutes les entités, ainsi que la fonction de détection de contact entre 2 entités, et donc l'appel au FractureGenerator.

Nous avons aussi une classe Heightmap qui permet de générer un terrain à partir d'une heightmap.

### 2.4 Dossier physics

Ici il y a notre fichier transform.cpp, qui comporte non seulement la Transform mais aussi la classe Mouvement disposant de la position, la vitesse et l'accélération.

### 2.5 Dossier shaders

Ce dossier contient nos shaders, dans lesquels nous avons fait un Phong simple avec l'intensité ambiante, diffuse et spéculaire, pour améliorer le rendu.

## 3 Fonctionnalités

### 3.1 Camera

La caméra dispose d'une partie des fonctionnalités du TP Caméra et nous avons donc une interface pour la manipuler, ainsi que les touches dédiées :

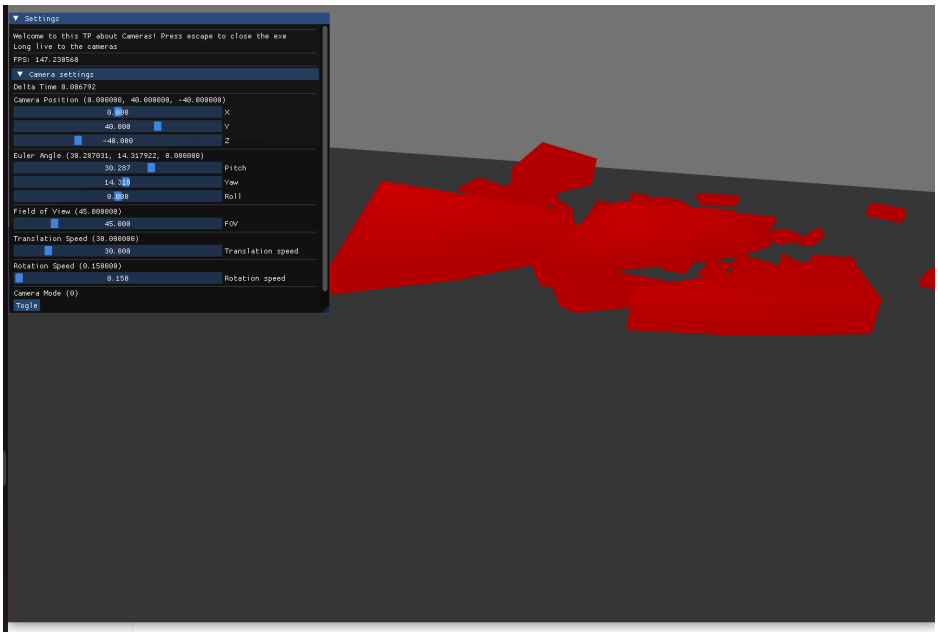


FIGURE 1 – Interface Caméra

Globalement le déplacement se fait avec les zqsd pour avant/arrière et droite/gauche ainsi que ae pour haut/bas, et la rotation de la caméra avec la souris.

### 3.2 Texture

La classe texture.cpp permet de générer des textures.

### 3.3 Génération de terrain par heightmap

Nous avons la possibilités en utilisant les heightmap du dossier assets de générer le terrain que l'on souhaite :

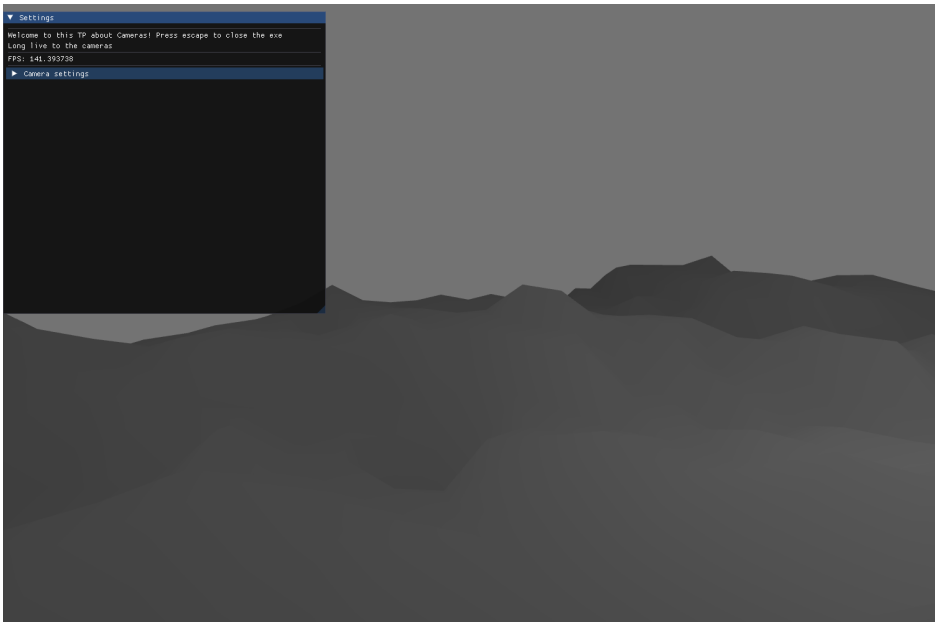


FIGURE 2 – Exemple de heightmap

Exemple de code pour le faire :

```
// Height Map
Rectangle rec;
rec.bottomLeft = glm::vec3(a: -75.0f, b: 0.0, c: -75.0f);
rec.right = glm::vec3(a: 150.0f, b: 0.0f, c: 0.0f);
rec.up = glm::vec3(a: 0.0f, b: 0.0f, c: 150.0f);
m_heightMap = HeightMap(map: rec, hRes: 30, vRes: 30, filename: "../assets/map/default-heightmap-1024x1024.png");
m_heightMap.maxHeight(maxH: 40.f);
m_heightMap.build(hRes: 30, vRes: 30);
m_heightMap.currentMesh().hasTexture(has_texture: false);
m_heightMap.currentMesh().color(color: glm::vec3(a: 0.30f, b: 0.30f, c: 0.30f));
m_heightMap.currentMesh().texture(path: "../assets/map/rock.png");
```

FIGURE 3 – Code de Heightmap

## 3.4 Envoi de balles par le joueur

L'envoi de balle par le joueur est controlé par le fichier control.cpp.

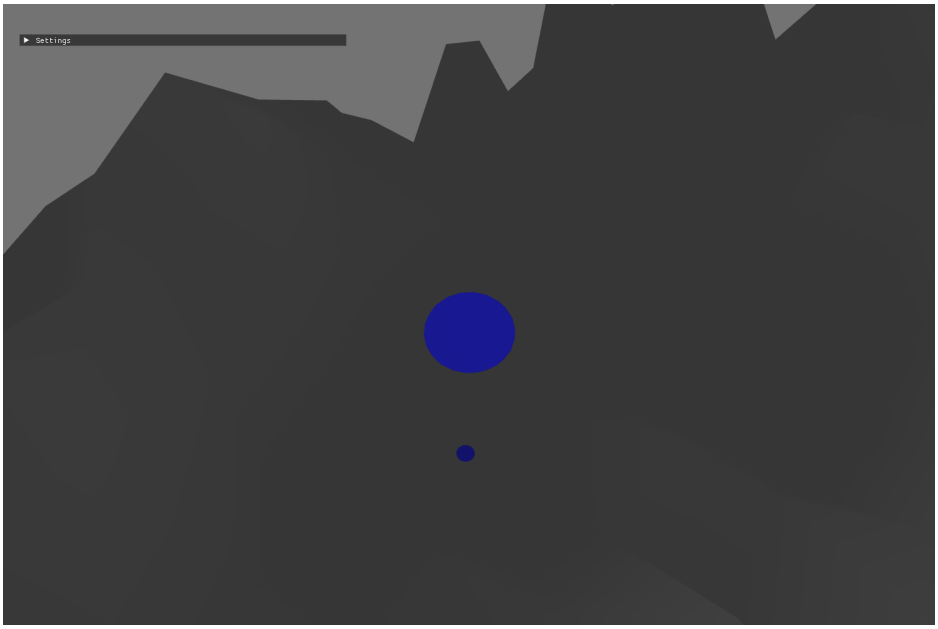


FIGURE 4 – Envoi de balle

L'envoi se fait avec le clic gauche de la souris à l'aide d'une fonction callback. Les propriétés de la balle peuvent être changées dans cette partie de control.cpp :

```
if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS){
    Entity sphere = Entity("../assets/entities/sphere.off");
    sphere.currentMesh().recomputeNormalsAndTexCoords();
    sphere.currentMesh().hasTexture(false);
    sphere.currentMesh().color(glm::vec3(0.6f, 0.1f, 0.1f));
    sphere.movement().position = camera->getPosition() + front;
    sphere.move();
    sphere.shouldRender(true);
    using namespace reactphysics3d;

    PhysicsCommon * physicalCommon = scene->physicsCommon();
    PhysicsWorld * world = scene->world();

    scene->entities().push_back(sphere);
    scene->entities().back().loadEntity(world);

    float radius = 0.5f;
    float strength = 5000.f;
    glm::vec3 weightedFront = strength * front;
    SphereShape* sphereShape = physicalCommon->createSphereShape(radius);
    Collider* entityCollider = scene->entities().back().physicalEntity()->addCollider(sphereShape, Transform::identity());
    scene->entities().back().physicalEntity()->applyLocalForceAtCenterOfMass(Vector3(weightedFront.x, weightedFront.y, weightedFront.z));
}
```

FIGURE 5 – Code de balle

## 3.5 Destruction d'objet par diagramme de Voronoi

Comme vu sur l'image pour la partie Camera, nous pouvons détruire des objets suivant un diagramme de Voronoi. On peut générer la destruction à partir du point de collision avec la balle :

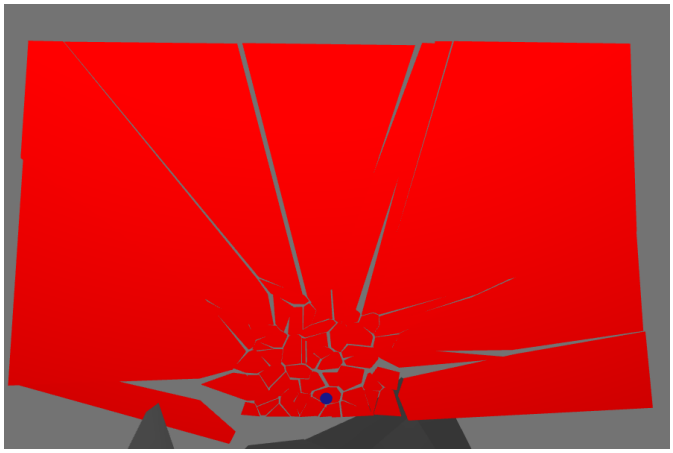

FIGURE 6 – Collision avec balle et destruction

La destruction se fait dans FractureGenerator.cpp qui génère des points autour du point de contact, créer un diagramme de Voronoi à partir de ça puis génère les nouvelles entités à partir des cellules du diagramme en les séparant.

Les points pris autour du point de contact sont générés au hasard avec RandomizePoints. On peut choisir le pattern de destruction, ici comme on envoie des balles on utilise le circlePattern et on peut controler le rayon autour duquel les points seront choisis. ApplyPhysicsFracture applique la physique aux éléments fragmentés, calculant la force que l'on appliquera sur eux au moment de la fracture. Pour finir MeshFromSite est utilisée dans Fracture, qui est la fonction qui calcule les fractures selon le point de contact et les points pris au hasard par RandomizePoints, afin de créer les entités fracturées.