

# INF433 Projet Rapport

Yiming Qin Kunhao ZHENG

June 1, 2020

## 1 Description Générale

Dans ce projet, on décrit une scène sur la mer. C'est un endroit où cachent beaucoup de récif, et où un combat vient de se passer. Le combat ont tué beaucoup de poissons et détruit des bateaux, donc on voit un naufrage et des planches qui floatent sur la mer. Encore, il y a un combattant qui tournoie ce zone marine, et jète des missiles sans arrêts. Dans le ciel, il y aussi a un oiseau bizarre et giant, qui chasse le combattant.

Du coté des codes, on essaie d'utiliser toutes les techniques introduits dans le cours, notamment la simulation, l'animation et texture etc. On trouve en plus un moyen pour représenter la vague en faire varier le bruit de Perlin au cours du temps.

En fait, cet oiseau vient de science-fiction *Hyperion*, qui est un humain-killer et donc le symbol de la guerre. On espère que ce scène peut faire des gens comprendre que la guerre est cruelle et laide.

## 2 Détails d'implémentation

### 2.1 La vague

Pour créer l'effet de la vague. Nous partons du terrain initial qui utilise le bruit de Perlin.

Le bruit de Perlin est formé à partir d'une fonction pseudo-aléatoire  $b(p)$  qui a tout point  $p$  renvoit une valeur scalaire déterministe.  $b$  est une fonction lisse à valeur entre  $[0, 1]$  et non périodique.

Le bruit de Perlin  $P$  est obtenu en sommant plusieurs instance de  $bb$  avec des fréquences augmentant, et amplitude diminuant.

$$P(p) = h \sum_{k=0}^{octaves} \alpha^k b(2^k sp)$$

D'où  $\alpha$  est la persistance,  $s$  est le paramètre de scaling,  $h$  est le paramètre de hauteur.

Jusqu'à maintenant le terrain reste encore statique. Pour simuler une vague. Nous avons fait varier les paramètre  $s$  et  $h$  en fonction du temps. Précisément,  $s$  est fait varier de façon croissante de 0.1 à 10 tandis que  $h$  de façon décroissante de 1 à 0.5. En plus, on a deux différentes échelles temporelles pour les deux variables. De cette manière on arrive à créer un effet de vague.

On fait une petite remarque que, pour que les variations de vague sont fluides, on ajoute des variables booléans, qui nous permet à la fin du processus de répéter ses variations dans le sens revers, et recommencent le processus dans le sens direct.

Les fonctions concernées sont :

```
float evaluate_perlin_terrain_z(float u, float v, const gui_scene_structure&gui_scene)
vec3 evaluate_perlin_terrain(float u, float v, const gui_scene_structure&gui_scene)
void scene_model :: set_gui()
```

## 2.2 La Créature Inconnue

La créature inconnue prend sa concepte de la science-fiction *Hyperion*, qui est un humain-killer et donc le symbol de la guerre.

On utilise l'animation hiérarchique pour créer et animer une créature inconnue, qui porte deux ailes bizarres et deux jambes longues et minces. L'animation hiérarchique se fait principalement sur les deux parties du corps. Pour simuler la marche, on ajoute la temporisation dans la rotation des différentes parties des jambes.

Il suit aussi une animation descriptive dans l'espace que nous allons discuter plus tard.

3 fonctions lui sont concernés :

```
hierarchy_mesh_drawable create_creature()
void set_creature_rotation(float t_creature)
void set_data_creature_animation(std :: map < std :: string, GLuint > & shaders)
```

## 2.3 Le Combattant

Le combattant qui vole dans l'animation est construit aussi par la construction hiérarchique. Néanmoins, cet objet ne fait pas une animation hiérarchique. En revanche, il est conçu pour l'animation descriptive.

Nous avons fixé quelques points dans l'espace et nous avons fait l'interpolation par spline cardinale. Remarquons que si nous utilisons cette façon d'interpolation, le premier point et le dernier point ne sont pas passés par l'objet.

La spline cardinale interpolant les positions  $p_i$  aux instants  $t_i$  peut s'exprimer sous la forme suivante :

$$\begin{aligned} \forall i \in [1, N - 2], \forall t \in [t_i, t_{i+1}] \\ p(t) &= (2s^3 - 3s^2 + 1)p_i + (s^3 - 2s^2 + s)d_i + (-2s^3 + 3s^2)p_{i+1} + (s^3 - s^2)d_{i+1} \\ s &= \frac{t - t_i}{t_{i+1} - t_i} \\ d_i &= 2K \left( \frac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}} \right) \\ d_{i+1} &= 2K \left( \frac{p_{i+2} - p_i}{t_{i+2} - t_i} \right) \end{aligned}$$

Nous avons choisi  $K = 1$  ici. Après avoir calculé le  $p(t)$  à chaque l'instant, nous pouvons faire une translation de l'objet pour le faire suivre l'interpolation.

Pourtant, cela ne suffit évidemment pas. Parce que la simple translation ne peut pas simuler l'orientation de déplacement de l'objet. Pour améliorer, il faut tenir compte à la dérivé de  $p(t)$  :

$$\begin{aligned} p'(t) &= \frac{dp(s)}{ds} \frac{ds}{dt} \\ &= \frac{1}{t_{i+1} - t_i} ((6s^2 - 6s)p_i + (3s^2 - 4s + 1)d_i + (-6s^2 + 6s)p_{i+1} + (3s^2 - 2s)d_{i+1}) \end{aligned}$$

Ce qui reste est de faire une rotation du vecteur orienté par l'objet au vecteur  $p'(t)$ .

3 fonctions lui sont concernés :

```
hierarchy_mesh_drawable create_plane()
const vcl :: vec3 set_plane_rotation(float t_creature)
void set_data_plane_animation(std :: map < std :: string, GLuint > & shaders)
```

## 2.4 Skybox

Une skybox est un cube entourant la scène sur laquelle une texture d'environnement est plaquée. La skybox permet de donner l'illusion que la scène est plongée dans un environnement plus large. La texture utilisée doit correspondre au cas d'un cube déplié, et les couleurs au niveau des arêtes doivent être cohérentes sur l'ensemble des faces.

Une fois le cube créé, celui-ci est placé de manière à englober la scène, et centré en permanence autour de la position de la caméra. La taille est précisée à l'aide d'un paramètre  $b$  passé dans la fonction `create_sky(float b)`.

## 2.5 Missle

Le missile est une combinaison d'un cylindre et d'une cone, à laquelle on applique la simulation, pour qu'il sorte du combattant et tombe vers la mer avec la gravité.

Pour initialisation, on crée le missile ensemble avec le combattant à fin d'avoir la même vitesse et direction. Une fois le missle soit créé, on s'en serve les formules suivantes pour renouveler sa vitesse et sa position. Ici  $F$  correspond à la gravité.

$$v = v + F \frac{1}{m} \times dt$$
$$p = p + v \times dt$$

Le missile s'agit des fonctions suivantes:

```
void scene_model :: set_missle_animation(const vec3& p_der)
const vec3 scene_model :: set_plane_rotation(float t_creature)
```

## 2.6 Autres objets

Les autres objets qui apparaissent dans notre model sont principalement statiques, y inclut les poissons mortes, un naufrage, les planches et des roches.

- Les roches sont créé par la même façon que la vague, mais statique, c'est-à-dire, sans faire varier les paramètres dans le bruit de Perlin.
- Les poissons mortes est créées grâce au billboard qui consiste à utiliser des images d'objets sur fond transparent en tant que texture pour représenter des objets d'apparence complexe sur une géométrie simple. On modifie aussi des poissons pour qu'elles tournent avec le caméra. En plus, pour avoir un effet de floater sur la mer, les poissons ont un *timer* pour changer leur position ensemble avec la vague en temps.
- Les planches sont des parallélépipèdes rectangulaires qui flotent dans la mer, dont l'effet est réalisé comme les poissons mortes.
- Le naufrage est simplement créé en ajoutant des points et des connections dans son mesh correspondant.

Dans la but d'ajouter la vraisemblance, les objets dessus ont tous leur propre texture.  
Les fonctions correspondantes sont:

```
mesh create_island(const gui_scene_structure& gui_scene)
mesh create_fish(float length, float width)
mesh create_box(float height, float width, float length)
mesh create_boat(float length, float width, float height)
```

### 3 Démonstration du scène

Nous donnerons ensuite deux captures de l'écran de notre animation : l'un est dans la configuration de shader "mesh" et l'autre dans "wireframe", dans lesquelles vous pouvez retrouver tous les objets présentés ci-dessus.

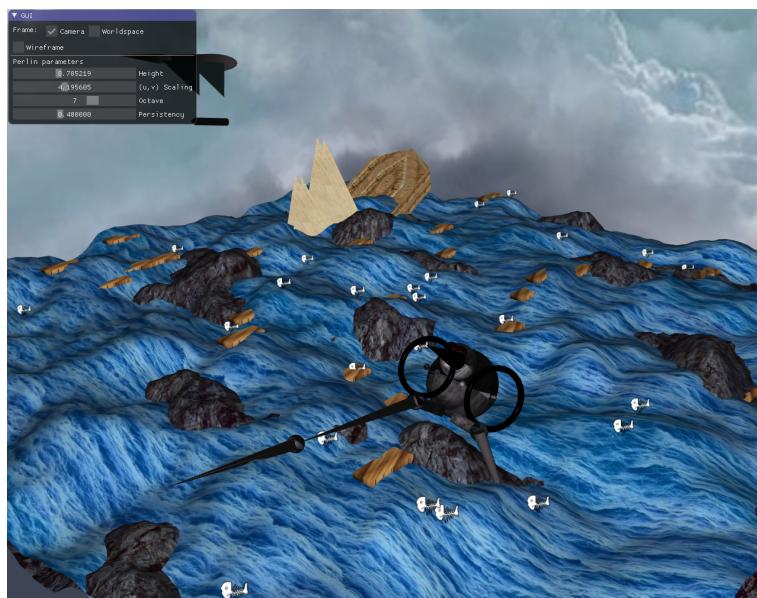


Figure 1: Démonstration du scène

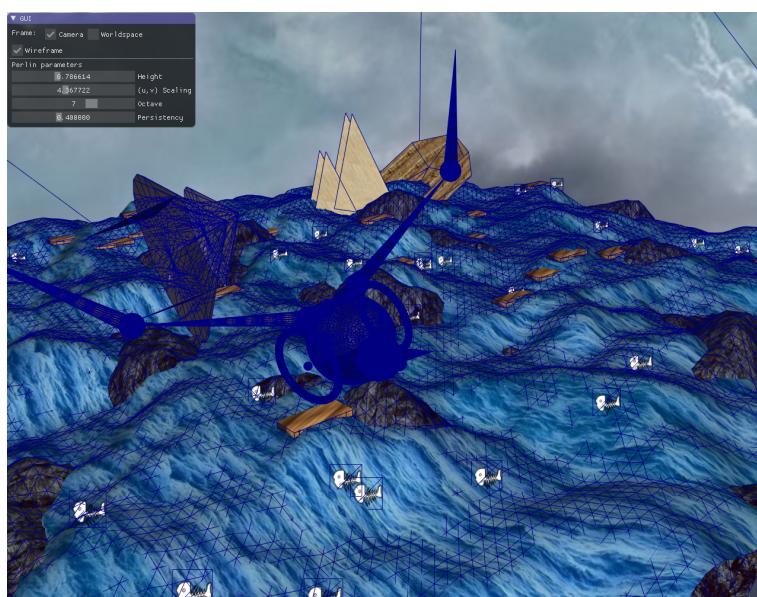


Figure 2: Démonstration du scène avec wireframe