Projet 3D INF443

Elsa Deville, Tom Marty

Juin 2020

1 Thème du projet

Nous avons décidé de créer **une scène sous-marine** avec des algues, des poissons, des coraux, un littoral, et un petit bateau naviguant sur une surface d'eau agitée. L'idée était de pouvoir passer successivement au dessus et en dessous du niveau de la mer.

2 Structure du code

Nous avons repris dans les grandes lignes le code fournit dans les TD précédents, en séparant dans des sous dossiers les différentes parties de notre code afin de rendre la lecture la plus claire possible. La structure des fichiers est détaillé dans le diagramme de l'image 1.

Les dossier final et test contiennent les fonctions de rendu (*setup et draw*) et elles appellent des fonctions organisées dans les dossiers terrain et water. Chaque fichier et son contenu est détaillé dans les sections suivantes. A un fichier correspond une feature dans notre scène 3D, ainsi par exemple, le fichier algue.cpp rassemble la classe et les fonctions de classe permettant de générer 1'algue, le fichier algues.cpp permet lui d'afficher un ensemble de NxN algues sur la scène et d'assurer leur coordination de mouvement et ainsi de suite.

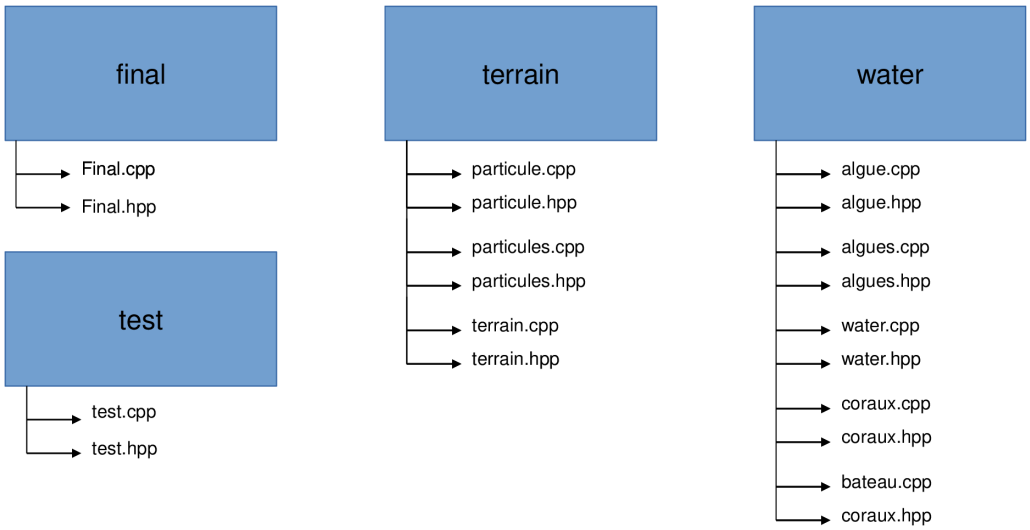

Figure 1: Structure du code

3 Composition

3.1 Le terrain

Pour générer le terrain nous avons repris la méthode explicitée lors des TD, à savoir une surface de type z = f(x,y) sur laquelle nous venons appliquer un bruit de Perlin. Voir figure 2.1.

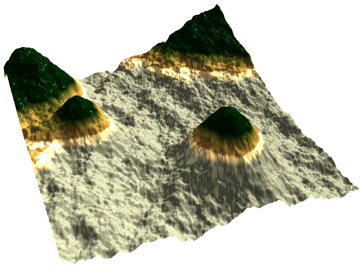

Figure 2: Terrain texturé

3.2 Les coraux

Nous avons ajouté des coraux sur le fond marin de la scène. La classe coraux contient une hiérarchie de base sur laquelle se fonde chaque corail, constituée d'un tronc en 3 parties, de 4 "branches" et de 6 "feuilles" réparties sur les 4 branches. L'appel à la fonction setup-data génère aléatoirement une liste de positions, d'axes de rotation et d'angle de rotations : pour chaque corail, chaque élément fils de la hiérarchie subit une rotation par rapport à son élément parent en fonction de cette liste. Voir figure 2.3.

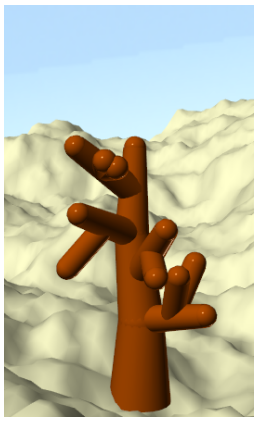

Figure 3: Corail généré aléatoirement

3.3 Les poissons

Afin d'ajouter du réalisme à notre scène nous avons implémenté des bancs de poissons reposant sur le concept des *boids* : **chaque poisson est une particule soumise à 3 règles élémentaires** vis à vis des autres particules.

- chaque particule s'éloigne d'une autre particule pour éviter une collision.
- chaque particule se dirige vers la moyenne des positions des autres particules
- chaque particule essaye d'aligner sa vitesse avec la vitesse moyenne des autres particules.

Nous nous sommes inspirés des travaux de Craig W. Reynolds qu'il a publié dans l'article : *Flocks, Herds, and Schools: A Distributed Behavioral Model* en 1987.

En plus de cela, nous avons implémenté un système d'évitement de collision des boids avec le sol et la surface de l'eau. Une particule est repoussés dans la direction symétrique à sa vitesse par rapport à la normale au sol, ce qui permet de garder un comportement naturel à l'approche d'un obstacle, du moment que la particule n'arrive pas orthogonalement à la surface. Chacune des 3 règles influe sur la variation de vitesse de chaque particule : nous

pouvons moduler leur influence respective à l'aide du gain qu'il est possible de modifier à partir d'un slider sur la scène.

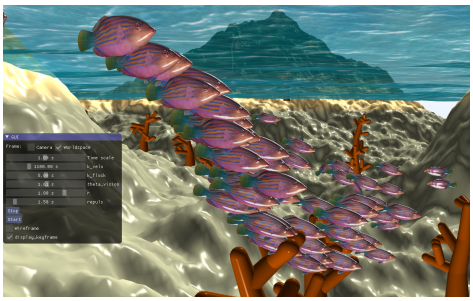

Figure 4: Banc de poisson

Afin d'améliorer le mouvement des boids nous avons implémenté un champs de vision pour chaque particule. Elles ne sont capables d'observer que les particules face à elle (on définit un angle d'ouverture θ). En ajoutant ce *manque d'information*, on espère se rapprocher d'un comportement naturel où l'individu ne peut pas traiter l'ensemble des informations provenant de toutes les directions. Enfin, nous avons également implémenté le fait que la distance entre 2 particules pondère l'influence d'une particule sur les décisions de l'autre (via une exponentielle décroissante).

En ce qui concerne le rendu graphique, nous utilisons un shader dédié couplé à un fort coefficient "specular" pour donner un effet shinny aux poissons, du aux reflets du soleil sur les écailles.

Dès lors que la simulation du mouvement des boids tournait correctement, nous avons remplacé chaque particule par un mesh de poisson importé depuis *Blender* (obtenu sur le site free 3D). Il a suffit alors de générer la base locale orthonormée (x',y',z') tel que x'= $V_{(particule}$)et y' à plat (contenu dans le plan (x,y)) de façon à orienter le mesh du poisson toujours dans la direction de déplacement. Voir figure 2.4.

3.4 La surface d'eau

La surface d'eau est une grille NxN dont la position de chaque point est mise à jour au fil du temps en se basant sur la théorie **des trochoidal waves** `https://en.wikipedia.org/wiki/Trochoidal_wave` :

Il s'agit d'une résolution à l'ordre 1 des équations de Navier-Stokes. Le résultat est une somme d'harmoniques possédant chacune une pulsation temporelle, une pulsation spatiale et un déphasage. La pulsation spatiale et la pulsation temporelle sont reliées entre elles par une relation de dispersion

$\omega_m^2 = g\,k_m\,\tanh\left(k_m\,h\right)$, de cette façon les grosses vagues se déplacent lentement. Afin de limiter le temps de calcul, on se contente d'une grille 100x100 et de 30 harmoniques.

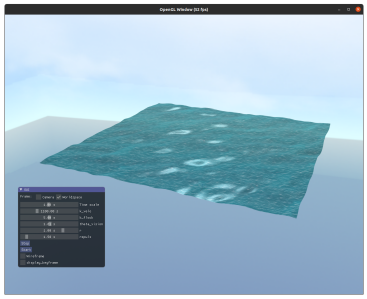

Figure 5: Simulation de la surface d'eau

On vient dans un second temps, figer une texture de mer sur la surface à plat avant de lancer la simulation. Les paramètres de chaque harmonique sont randomisés dans une plage de valeur déterminée empiriquement de sorte à obtenir un aspect le plus naturel possible. Voir figure 2.5.

3.5 Les algues

Afin de générer des algues animées, nous avons repris la méthode de résolution des équations de la surface d'eau, afin d'assurer que les mouvements des algues soient coordonnés avec ceux de l'eau. Nous avons simplement rajouté un déphasage spatial suivant la direction z de façon à ce que l'algue puisse également se plier.

Néanmoins, la structure de données utilisée pour générer un champs d'algues ne permet difficilement le calcul parallèle, ce qui influe grandement les performances de la simulation.

L'algue est un objet de type liste chaînée possédant un parent et possiblement un enfant. On peut ainsi créer une algue de la longueur que l'on souhaite, en liant de nouveaux éléments. Chaque élément de la chaîne possède également *une position* et une *position initiale*. La position est utilisé pour l'affichage tandis que la position initiale est utilisé pour le calcul des positions successives par application de la méthode des *trochoidal waves.*

Ensuite, à l'aide des positions de chaque élément de la chaîne, on affiche un billboard aléatoire parmi 4 textures sans fond d'algues récupérées sur Internet dont l'orientation est également randomisée.

Enfin, nous avons généré NxN algues sur toute la map. On obtient ainsi

un champs d'algues animées dont le mouvement est cohérent avec lui-même et avec la surface d'eau. Voir figure 2.4.

3.6 Le bateau

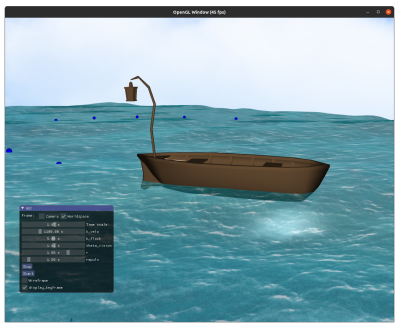

Figure 6: Simulation de la surface d'eau

Nous avons implémenté sur la surface de l'eau un bateau qui navigue en suivant une trajectoire qui peut être modifiée par l'utilisateur à l'aide de *keyframes*, comme nous l'avions implémenté pour l'oiseau dans la partie animation du TD. La trajectoire du bateau est calculée par splines cardinales. Son orientation est modifiée en alignant son avant sur la vitesse calculée par dérivée de la spline cardinale. Enfin, les oscillations du bateau sur l'eau sont simulées en alignant la normale et l'altitude du bateau sur celles de la surface de l'eau au centre du bateau. A la différence de la trajectoire de l'oiseau dans le TD sur l'animation, celle du bateau est continue, ce qui implique quelques modifications sur la méthode de calcul des splines cardinales. La variable temporelle utilisée est de type *timer.event* et pas *timer.interval*, afin d'utiliser la même variable temporelle dans toute la scène. Le bateau utilisé a été téléchargé sur internet sur le site *SketchFab*. Voir figure 2.5.

3.7 La cube map

Afin de générer un ciel pour notre scène nous avons utilisé une cube map. On créé ainsi un cube sur lequel on vient plaquer des textures de ciel sur chaque face. En choisissant une échelle très grande pour le cube et en reliant les mouvements de la caméra au mouvement de la cube map, on place constamment la cube map autour de l'utilisateur. Voir figure 2.2.

4 Optimisation/Calcul

Étant donné que plusieurs de nos features sont recalculées à chaque affichage de la scène, nous avons du faire face à des gros problèmes de performances CPU (le GPU n'étant que très peu sollicité). Nous avons commencé par

limiter le nombre d'éléments pour les features les plus coûteuses en temps de calcul : 300 poissons, 100 algues, une surface d'eau de 100x100.

Concernant les boids : Nous avons une complexité en N^2 étant donné que chaque *boid* doit observer les N autres pour calculer sa futur vitesse (et sa future position par Euler explicite). Une solution pour améliorer le temps de calcul pourrait être de mettre en place des structures accélératrices (partition de l'espace) pour passer en temps linéaire.

Concernant la surface de l'eau : Nous avons également une complexité en N^2. Une solution pour abaisser cette complexité pourrait être de baisser le nombres d'harmoniques mais cela impacte grandement le réalisme de la surface. Une autre solution pourrait être de diminuer le nombre de points du maillage (c'est à dire N) et d'interpoler des points intermédiaires.

5 Axes d'amélioration

La modélisation 3D procédurale est un domaine sans fin et la quête de réalisme pousse à toujours raffiner les modèles avec lesquels nous travaillons. Dans cette optique nous avions pensé à rajouter les éléments suivants à notre scène 3D :

- prise en compte des phénomènes de réfraction dus à l'interface air/eau
- implémentation des ombres de l'ensemble des éléments 3D
- travail sur les shaders et notamment mise en place d'un shader "underwater" et d'un shader "above water" pour dissocier la phase quand la caméra est dans l'eau de celle quand elle est au dessus.

Il serait aussi souhaitable de basculer une partie des calculs se prêtant bien au calcul parallèle (comme la simulation des boids ou de la surface) sur GPU afin de diminuer la charge du CPU et augmenter les performances générales en réfléchissant à l'intêret d'une telle méthode !

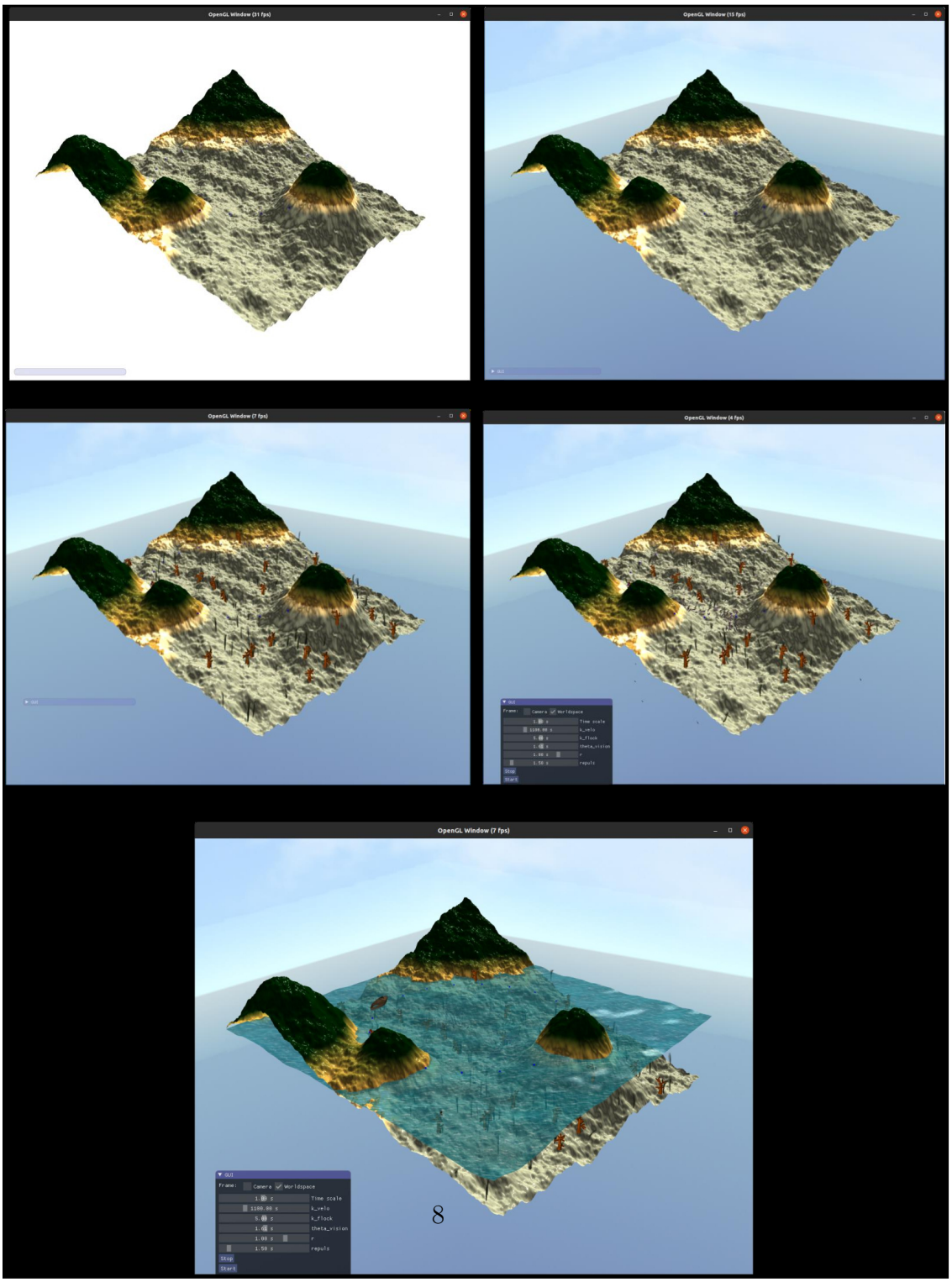

Figure 7: Décomposition du rendu OpenGL