OPEN GL BOIDS

FLYING WHALES

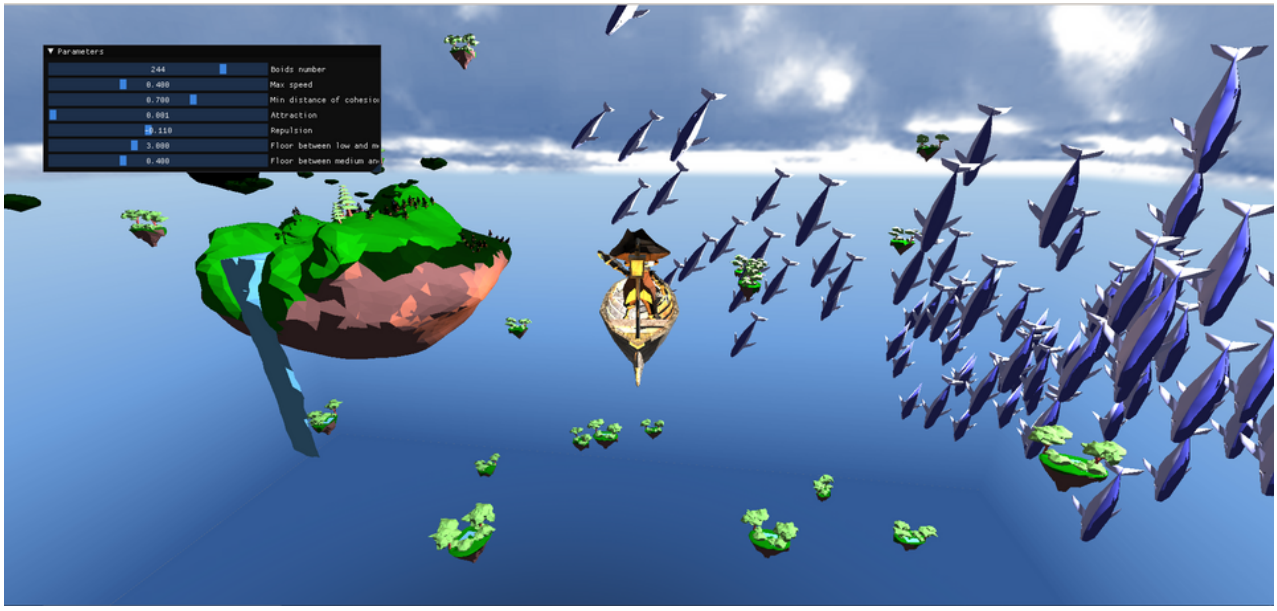
Charline Le Pape
Juliette Jeannin

https://github.com/Chabraka/progObjet

Description

Dans *Flying Whales* vous incarnez un pirate voguant dans les airs parmi des îles et des baleines volantes. Vous pouvez observer ce qui vous entoure en tournant la caméra avec la souris et en zoomant/dézoomant avec le scroll. Vous pouvez jouer avec le comportement des baleines : les faire se séparer, se rapprocher, être en petit ou grand groupes... Ou jouer sur les paramètres de niveau de détails.

Lien git du projet : https://github.com/Chabraka/progObjet



Boids

Tout d'abord nous avons créé les boids. Ils sont régis par les comportements de cohésion, de séparation et d'alignement à l'aide de paramètres qui sont le facteur d'attraction, le facteur de répulsion et la distance minimale ou seuil à partir du quel on considère le mouvement des baleines à proximité. par les comportements de cohésion, de séparation et d'alignement à l'aide de paramètres qui sont le facteur d'attraction, le facteur de répulsion et la distance minimale ou seuil à partir du quel on considère le mouvement des baleines à proximité.

Ainsi nous avons une classe *Boid* pour décrire les comportements d'un boid et une classe *Boids* qui les contient tous et permet de les comparer pour adapter leur déplacement mais également de tous les dessiner et de charger leur texture en une seule fois.

De plus, nous avons ajouté une rotation sur les baleines, ce qui fait qu'elles sont orientées selon la direction de leur mouvement. Nous leur avons également ajouté des collisions mais vous les trouverez commentées dans notre code. En effet nous trouvions que le comportement des boids était trop altéré par celles-ci alors nous avons préféré les retirer. Elles faisaient que les baleines changeaient brusquement de direction et parfois restaient bloqués en "paquets".

Concernant les paramètres implémentés, nous avons utilisé une structure Parameters qui est un singleton avec des attributs publiques afin que ces derniers puissent être manipulés par le joueur via l'interface ImGui. Ayant des fonctions avec parfois 3 ou 4 arguments, cela nous as aussi permis de ne pas en rajouter d'autres étant donné que Parameters était appelable à n'importe quel endroit du code.

La plupart des méthodes utilisées sur les boids (dessin, calcul de la position...) sont itératives : on utilise des boucles for sur le vecteur de *Boid*. Une amélioration envisageable est celle d'utiliser des iterator pour des fonctions peu complexes.

Arpenteur & Camera

Plus adaptée pour les jeux à la troisième personne, nous avons choisi de faire une Trackball camera. Celle-ci peut être tournée à l'aide de la souris et on peut zoomer et dézoomer avec la molette. Tout cela est géré par des événements p6 qui est la librairie que nous avons utilisée à travers le template de Jules Fouchy.

Nous avons d'abord eu quelques difficultés à l'implémenter car en TP celle-ci tournait autour d'un objet sphérique qui restait au centre du repère mais dans notre cas l'objet bouge donc on a mis un certain temps à trouver dans quel ordre faire les transformations pour qu'elle tourne autour du personnage et non autour de l'origine du repère de la scène mais nous y sommes parvenues à force d'essais.

L'arpenteur de déplace en avant avec la touche z ou la flèche du haut, en arrière avec s ou la flèche du bas, vers le haut avec la barre espace et vers le bas avec la touche Maj. Les flèches latérales ainsi que "q" et "d" permettent au joueur de changer son orientation avec une rotation selon l'axe x. Comme la caméra connait la position du joueur, quand celui-ci tourne elle tourne en même temps pour s'adapter au champs de vision du pirate.

L'arpenteur possède également une vitesse et une accélération qui font qu'il a des déplacements fluides : accélération au début d'un mouvement et décélération à la fin. On a donc un arpenteur animé avec une caméra qui suit son point de vue.

L'arpenteur possède une détection des collisions : il est repoussé et ralenti par les baleines et toutes le îles flottantes. Chaque objet possède un radius correspondant à la taille de sa hit box. Les collision fonctionnent comme cela mais font que le personnage "saute" un peu. L'une de nous les a codés sur sa virtual box, avec un temps de rafraîchissement plus long, les collisions à vitesse de rafraîchissement normal semblaient beaucoup moins bien fonctionner : le ralentissement et le déplacement étant un peu trop forts. Pour le calcul de collisions on a l'algorithme suivant :

calcul des collisions avec *MainIsland*
pour i allant de 0 à nombre *Boids*
    calcul des collisions avec *Boid*
 pour j allant de 0 à nombre *Islands*
    calcul des collisions avec *Island*

Obstacles

Les obstacles sont des îles : une grande île, *MainIsland*, et des plus petites. Comme pour les boids on a une classe pour l'entité *Island* et une classe *Islands* qui regroupe toutes les Island dans un vecteur, charge les textures en une fois et les dessine (itératif) à des positions aléatoires dans la box.

On a alors la MainIsland au centre de la box et des petites îles tout autour placées aléatoirement, auxquelles on a ajouté une rotation pour qu'elles ne semblent pas toutes identiques.

Skybox

La skybox est exclusivement faite en OpenGL (ce n'est pas un modèle comme le reste des éléments), d'où les classes *Skybox* et *SkybowOpenGL*. La texture est une cubemap de ciel et est appliquée en utilisant la structure Vertex3DUV.

La skybox n'interagit pas avec la lumière du personnage, et en tant que ciel on peut s'attendre à avoir un affichage uniforme sur toute sa surface. Pour obtenir ce résultat la skybox a son propre shader, qui n'applique que sa texture, sans prendre en compte les lumières.

Lumières

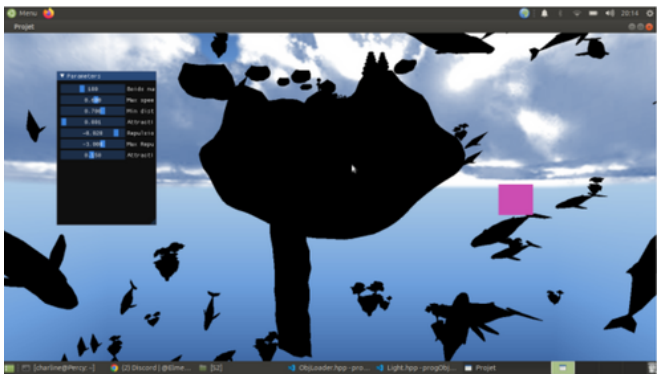
En parlant de lumières, 2 ont été implémentées : une ponctuelle sur l'arpenteur et une ponctuelle/omnidirectionnelle assez forte en haut de la box qui éclaire la scène du dessus pour faire une lumière d'ambiance. Elles suivent le modèle de Blinn-Phong.

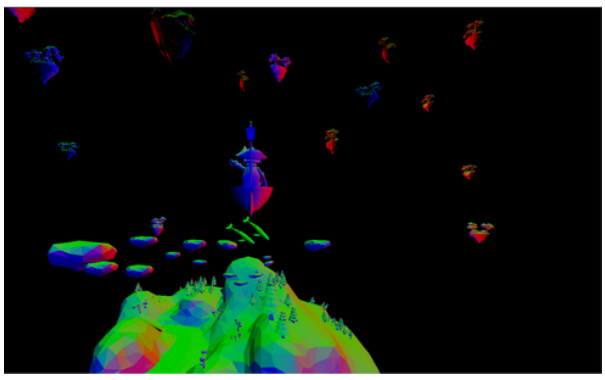
Avec les mouvements de l'arpenteur décrits ci-dessus, il est compliqué de trouver la position exacte pour que la lumière reste au niveau du lampion, on a donc la lumière qui d=change l'éclairage au niveau de l'arpenteur quand il tourne.

Par rapport à l'éclairage global, servant à simuler le Soleil, plusieurs tests ont été réalisés pour que la partie ombre ne soit pas trop foncée et qu'on puisse voire la texture du dessous même si elle est peu éclairée.

Au niveau du code, deux classes ont été utilisées : *LightProperties*, contenant les propriétés des objets, et *Light* pour la position et l'intensité de la lumière. On pas eu le temps de récupérer les données des fichiers mtl, donc on a rentré des valeurs à l'œil pour éclairer les obj. Il y a 2 types de données à transmettre : les propriétés lumineuses du matériaux (à récupérer dans les champs de la classe) et les propriétés de la lumière (position, couleur) à donner depuis le main.

Nous avons rencontré pas mal de problèmes pour ajouter la lumière car beaucoup de fonctions dépendent des unes des autres étant encapsulées, il faut ajouter les arguments dans plusieurs fonctions à la fois.





Par exemple il y a eu un moment ou nous n'avions pas de couleur (image de gauche), et un autre ou on d'est retrouvé avec les normales (image de droite).

ImGui

Nous avons utilisé ImGui à l'aide de la librairie p6. pour créer une fenêtre avec des paramètres de simulation modifiables à n'importe quel moment. Ainsi on peut changer le comportement des boids et le niveau de détails des îles et des boids, mais nous allons détailler cela juste après.

Modèles 3D

Nous avons téléchargé des modèles 3D gratuits, parfois avec leur texture, puis nous les avons convertis en .obj avec Blender et créé différents niveaux de détails. La création de parser a été compliquée car il a fallu décomposer le fichier et stocker les vertex pour chercher les préfixes et v, vt, vn. Sur ce point là on regrette un peu de ne pas avoir utilisé une bibliothèque ce qui aurait pu nous faire gagner du temps.

Pour le rendu des modèles 3D nous avons utilisé deux classes : *Objrenderer* pour un objet et *MultiRenderer*, qui possède 3 renderer d'objet et prend en compte la distance de l'objet à la caméra. On a donc 3 modèles 3D différents pour les petites îles ainsi que pour les baleines, chacun avec un niveau de détails différent.

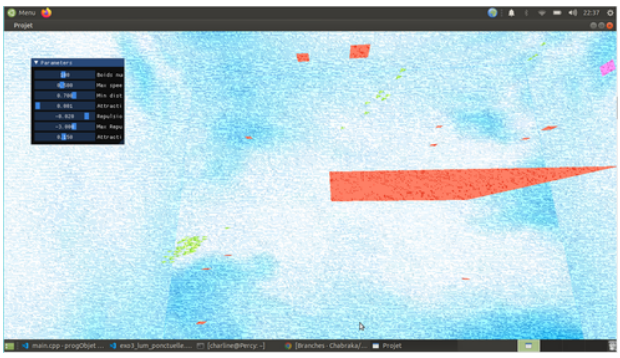
LODs

Les niveaux de détails utilisés sont les suivants : low, medium et high. Les îles et les baleines peuvent être affichés dans chacun des trois LOD. Il convient de définir les seuils de bascule de l'un à l'autre. Dans *Parameters* nous avons défini deux valeurs, une pour le seuil entre low et medium et un autre entre medium et high. A travers l'interface ImGui l'utilisateur peut changer ces valeurs pour avoir le niveau de détails qu'il souhaite. Sinon par défaut, différents niveaux changent selon la distance entre l'objet et la caméra.

Difficultés

Une première difficulté a été le passage de la 2D à la 3D, notamment avec l'ajout d'un déplacement suivant l'axe z.

Ensuite nous n'avions pas le même affichage ou rendu visuel du projet donc l'une voyait des erreurs quand l'autre n'en voyait pas, ce qui était parfois un peu compliqué pour se comprendre mais nous avons plus échangé à ce sujet pour corriger ce problème. Par exemple l'une de nous voyait que les couleurs étaient mixées (demi-transparent) et l'autre non.



Comme déjà évoqué nous avons aussi rencontré des problèmes avec les collisions des boids et la lumière. Ceux ci ont été réglés en faisant des tests et en persévérant mais ça a pris du temps.

Autre point, les TP étant assez denses, à nous deux nous avions tout fait mais chacune de nous n'as pas eu le temps de tout faire avant la fin des cours de synthèse d'image, il a donc fallu se familiariser avec la caméra et les lumières dans le cadre du projet pour l'une d'entre nous.

Nous avons mis pas mal de temps à trouver des solutions pour des difficultés rencontrées, ce qui fait que nous n'avons pas eu le temps de faire nos propres modélisations ou d'utiliser un algorithme d'ombrage à notre grand regret.

Nous avions un peu d'appréhension avec certain des critères d'évaluations que nous n'avions pas vu en cours, notamment l'ombrage ou les LODs mais finalement nous nous en sommes bien sorties avec ce dernier.

Une dernière difficulté a été de garder un code propre tout au long du projet. Au début c'était plutôt correct mais à la fin il a fallu faire un gros nettoyage.

Améliorations

Avant d'implémenter notre arpenteur, nous avions codé un *Tracker* qui entrainait/attirait les boids avec lui selon le facteur d'attraction qu'on lui donnait. On avait trouvé cela intéressant mais nous l'avons enlevé du projet final. Cela fonctionnait bien en 2D pour rajouter de l'aléatoire dans le mouvement des boids mais en 3D l'espace était bien plus grand donc on avait moins l'effet "Ping Pong de la 2D.

On avait aussi pensé a faire un cycle jour/nuit ou faire que le pirate pêche les baleines mais nous avons privilégié les critères demandés (et nous nous sommes dit que ce n'était pas très éthique !).

Dans les paramètres modifiables nous avons surtout mis ce qui était relatif au boids mais nous aurions pu ajouter des paramètres liés aux îles comme leur nombre par exemple.

Récapitulatif

Boids avec 3 comportements enfermés dans un cube (baleines)
GUI avec Dear ImGui
Modélisation non-originale
Trois niveaux de détails (low, medium, high)
Cube texturé qui englobe la simulation (skybox)
Elements de décor texturés (îles)
Camera à la troisième personne (trackball)
Arpenteur animé
Deux lumières ponctuelles
Pas d’algorithme d'ombrage