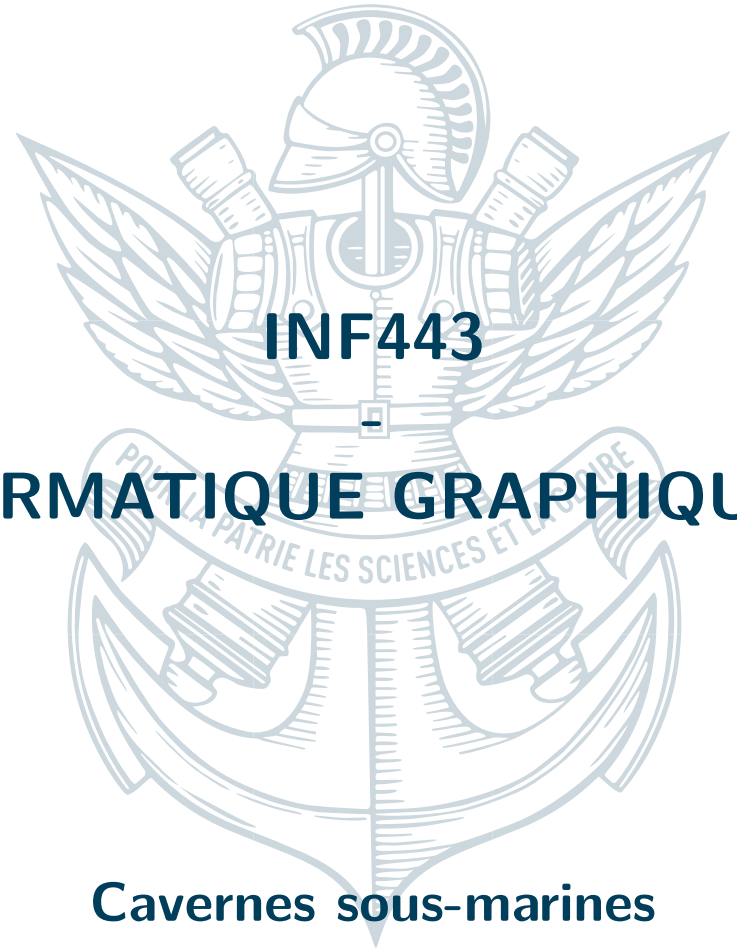

INF443
-
INFORMATIQUE GRAPHIQUE 3D

Cavernes sous-marines

31 mai 2019

Alexis de La Fournière, Valentin Villecroze

1
PRÉSENTATION DU PROJET

Le but de ce projet était de réaliser une scène 3D permettant l'exploration de cavernes sous-marines générées procéduralement.

Figure 1 – Inspiration

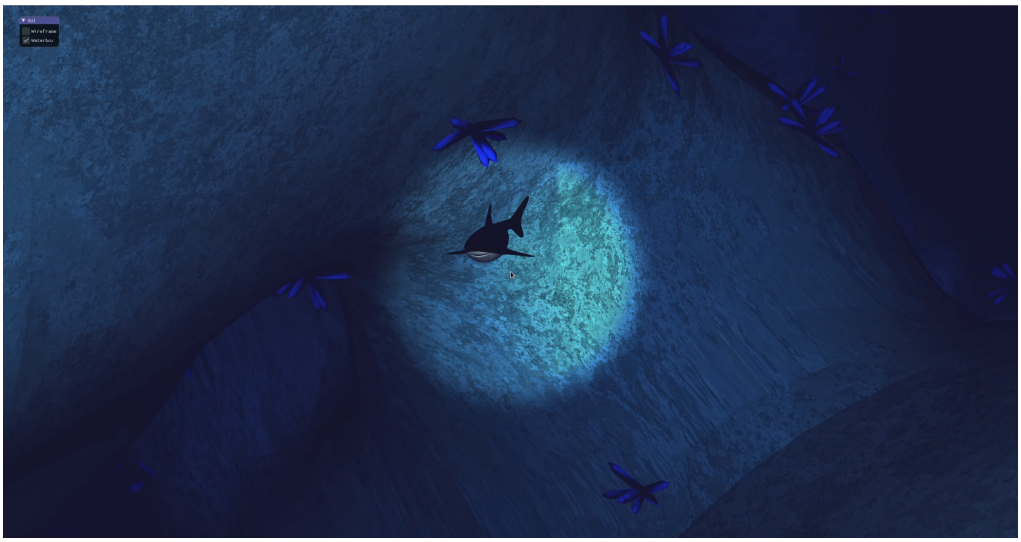

Figure 2 – Notre scène

2
DESCRIPTION DE L'IMPLÉMENTATION

2.1 Requin et murène

2.1.1 • Modélisation

Les modèles du requin et de la murène ont été fait sur blender puis exportés au format .obj et chargés en utilisant les fonctions proposées par vcl. En ce qui concerne les techniques de modélisation, nous avons notamment utilisé la technique de subdivision avec lissage (algorithme de Catmull-Clark) pour obtenir une belle forme arrondie à partir d'un maillage grossier. Nous avons ensuite rajouté les différents détails sur le maillage subdivisé.

2.1.2 • Texture

La phase de texturing s'est avérée particulièrement complexe, puisque les formes non standards nous rendaient difficile l'attribution de coordonnées. Finalement, nous avons décidé d'utiliser les textures en tant que bruit perturbant légèrement la couleur originale. Malgré tout, la difficulté de plaquer la texture fait qu'à certains endroits le bruit est étiré, ce qui donne un mauvais rendu. Pour pallier à ce problème, nous avons envisagé d'utiliser une texture 3D de bruit de Perlin directement générée dans le code. Malheureusement, nous n'avons pas eu le temps de comprendre comment utiliser les fonctions opengl pour enregistrer convenablement une texture 3D.

Pour le requin blanc, la couleur blanche ou bleue est atttribuée directement dans le shader en fonction de la position du point dans le repère de l'objet. Cette fonction est globalement une fonction affine, modulée par un bruit tiré de la texture. Le choix de faire ce calcul dans le fragment shader, et non d'attribuer une couleur aux points dans la mise en mémoire est dû au fait que dans ce cas il y aurait eu une zone d'interpolation, ce qui ne correspond pas à la réalité.

2.1.3 • Animation

L'animation des modèles a nécessité le codage d'un nouveau vertex shader ("ondulant"). En effet la nage d'un poisson nécessite la déformation continue de la géométrie qu'on ne saurait subdiviser en primitives. Pour effectuer ces ondulations nous nous sommes appuyés sur le fait que ces déformations ont principalement lieu le long d'un seul axe (nous avons pris z arbitrairement). L'idée que nous avons implémentée est de dire que le shader reçoit sous la forme d'un spline cardinale la nouvelle forme prise par l'axe z du mesh, reconstruit la géométrie autour en se plaçant dans la base de Fresnel. Côté `C++`, nous avons créé une classe héritant

de `mesh_drawable` (`fish`) qui, sachant une fonction, donne les paramètres au shader pour que l'axe z prenne la forme de cette fonction. Pour imiter la nage d'un poisson, une fonction de type xe^{x-l} donne un bon rendu.

2.2 Caverne

2.2.1 • Marching cubes

Pour générer des cavernes sous-marines procédurales dans lesquelles évoluer, nous avons décidé d'utiliser une surface implicite définie à l'aide d'un bruit de Perlin 3D.

Pour cela, nous avons implémenté l'algorithme de Marching Cubes, en s'inspirant du code proposé par Paul Bourke sur son site *http ://paulbourke.net/geometry/polygonise/*, en l'adaptant à une utilisation au sein de la bibliothèque vcl.

2.2.2 • Division en chunks

Pour éviter un temps de chargement trop long lors du calcul du mesh représentant la caverne, nous avons décidé de diviser l'espace en plusieurs cubes de taille réduite (des *chunks*), et de calculer les surfaces uniquement dans les cubes proches de la position de la caméra (la distance d'affichage est réglable).

2.2.3 • Cristaux

Pour briser la monotonie de la caverne, des cristaux ont été ajouté de manière aléatoire dans celle-ci (un cristal par chunk environ). La surface de chaque chunk est donc un `mesh_drawable_hierarchy` où on place le cristal sur une face aléatoire de la surface de la caverne, en l'orientant dans la direction de la normale à cette face.

2.2.4 • Texture

Plaquer une texture 2D sur la surface de la caverne s'est révélée complexe, car les orientations des triangles la composant étant aléatoires, certaines directions se sont retrouvées bien plus étirées que d'autres. Comme pour le requin, le temps a manqué pour comprendre l'utilisation des textures 3D dans OpenGL.

Il a également fallu modifier la fonction `draw` des `mesh_drawable_hierarchy` pour permettre l'utilisation de différentes textures pour les différentes éléments de la hiérarchie.

2.2.5 • Déplacement du requin

Nous avons également implémenté un déplacement du requin en fonction du gradient de la fonction définissant la surface implicite.

2.3 Vision sous-marine

2.3.1 • Le shader « underwater »

Pour créer des rendus imitant le milieu sous-marin, nous avons eu besoin de coder un nouveau fragment shader. L'objectif de ce shader était double : gérer plusieurs lumières et de différents types (principale, cristaux lunineux et spot) et rendre compte des effets d'absorption de la lumière par l'eau.

Pour cela nous avons défini une structure regroupant plusieurs attributs : position, couleur, intensité, rayon (pour donner une impression de zone lumineuse), et rayon interne et externe du cône pour les spots. Ensuite, nous calculons pour chacune de ces lumière les éclairage diffus et spéculaires que nous multiplions par un facteur d'atténuation qui est une exponnentielle de la distance parcourue par la lumière. Finalement la couleur finale est une interpolation linéaire entre la couleur de l'objet et une couleur de fond, pondérée par l'éclairage.

2.3.2 • Les lumières

Nous avons envisagé plusieurs types de source lumineuse dans notre scène :

— lumière principale, qui suit la caméra.

— lumière diffuse émise par des "cristaux" fixes dans la scène.

— spot lumineux éclairant un cône.

Pour modéliser ces différentes formes de lumière et notamment la manière dont elles sont absorbées, nous utilisons deux paramètres : le rayon et la force. L'absorption vaut alors :

$$\exp\left(\vec{\text{abs}} \times \frac{d-r}{s}\right)$$

avec

— $\vec{\text{abs}}$ un vecteur général donnant la propension du milieu à absorber dans les différentes couleurs.

— d la distance parcourue par la lumière, donc la somme de la distance objet-lumière et objet-caméra.

— r le rayon de la source.

— s la force de la source.

Ce modèle s'appuie sur la loi de Beer-Bambert qui dit que l'absorbance (log de l'intensité lumineuse) dépend du produit de la longueur parcourue par le facteur d'absorption. Le paramètre de rayon permet de créer des zones lumineuses diffuses donnant l'illusion que la source n'est pas ponctuelle. Couplé à un paramètre de force faible, cela nous permet de créer des îlots de lumière émanant de cristaux Enfin précisons que puisque le shader est limité à dix lumières annexes, nous caluculons à chaque image quels sont les dix lumières les plus proches, pour les passer en paramètre au shader.

2.3.3 • Gestion de la caméra

Pour créer une sensation d'immersion, nous avons changer le contrôle de la caméra pour utiliser un contrôle première personne de type jeu de tir. Pour cela nous réglons le paramètre de scale (distance au centre de rotation) à 0, puis nous fixons le curseur au centre de l'écran. Enfin nous effectuons les rotations et déplacements adaptés lorque l'utilisateur bouge la souris ou appuie sur les touches directionnelles ou ZQSD/WASD. Finalement, la touche échap permet de sortir de ce mode (f pour y entrer de nouveau) ce qui permet notamment de débloquer le curseur.

Nous avons également ajouté une gestion des collisions avec la caverne en utilisant la fonction définissant la surface implicite.

3
LISTES DES FICHIERS

Architecture

- `src/`
  - `project/`
    - `cavern` : permet la génération et l'affichage de la carverne.
    - `cristal` : crée un mesh représentant un cristal.
    - `fish` : permet l'affichage du requin et de la murène.
    - `light` : permet la création et l'affichage des lumières.
    - `loader` : permet le chargement d'un fichier .obj en objet `fish`.
    - `perlin3DTexture` : permet la génération d'une texture 3D suivant un bruit de Perlin (inachevé).
    - `scene` : contrôle l'affichage de la scène.
    - `waterbox` : permet l'affichage d'une texture de fond.
  - `vcl/external_lib/`
    - `marching_cubes` : réalise l'algorithme Marching Cubes sur une fonction quelconque.
  - `vcl/core/scene/`
    - `camera` : modifications pour utiliser le mode "fps".
    - `camera_control_glfw` : modifications pour contrôler la caméra avec les touches directionnelles.
- `shaders/project/`
  - `ondulant` : permet l'ondulation du requin et de la murène.
  - `requin` : affiche les couleurs du requin.
  - `volumetric` : permet de texturer en fonction de la position d'un mesh.
  - `underwater` : donne l'aspect sous-marin.