École Polytechnique

INF443

Projet informatique graphique
La dernière planète

Louis Proffit
Paul Théron
1 juin 2021

Table des matières

Introduction et structure du projet

Nous avons cherché pour ce projet à réaliser une planète, sur laquelle sont présentes de multiples petites îles.

La structure du code de notre projet permet de comprendre son esprit.

Une classe abstraite mère, $scenevisual$, implémente les fonctions de base pour un rendu graphique :
- Une fonction d'initialisation
- Une fonction d'affichage
- Une fonction de mise à jour

Deux classe héritent de cette classe :

1. La classe planète : la planète est notre environnement principal, à partir duquel on peut atteindre les autres. Sa caractéristique principale est d'être un terrain sphérique.
2. La classe abstraite $environnement$: elle représente les autres environnements, caractérisés par un terrain relativement plat (en tout cas non sphérique). Cette classe implémente d'autres outils plus spécialisées, incompatible avec la planète :
   - Une fonction d'altitude, qui permet d'accéder à la hauteur du maillage du terrain en tout point.
   - Une skybox : toutes les scènes en ont.
   - Des constantes de dimension qui délimitent le terrain
   - Deux caméras
   - Une pré-implémentation des fonction d'affichage et de mise à jour : elles permettent de déléguer le fonctionnement des caméras et des mouvements de l'utilisateur qui sont communes à tous les environnements autres que la planète.

Cinq classes héritent de $environement$:
- La classe $forest$
- La classe $mountain$
- La classe $city$
- La classe $countryside$ (les champs, $field$ désigne "l'objet" champ)
- La classe $desert$

Chaque classe détaille son environnement et possède une implémentation et des fonctionnalités particulières, que nous allons détailler.

Dans les cinq environnements, on dispose de deux caméras.
- La caméra $cameraaroundcenter$: permet de faire pivoter et zoomer autour d'un centre, elle permet une vue aérienne, et est déjà implémentée dans la vcl.
- La caméra "$cameraminecraft$" : imite la caméra de minecraft : on déplace la vue maintenant le clic gauche, on se déplace avec les flèches ou un ZQSD pour un clavier AZERTY. La touche shift permet de "sneak", comme dans minecraft.

Dans les cinq environnements, la frontière de l'espace est délimitée par une skybox. C'est une technique très simple, sur laquelle nous n'avons pas perdu de temps. Nous n'avons pas trouvé de textures de grande qualitée et gratuites compatibles avec notre environnement, donc nous nous contentons de plus basse qualité, mais l'intérêt est ailleurs !

Dans les cinq environnements, l'espace entre le terrain et la skybox est rempli par de l'eau. Le détail de l'implémentation de l'eau est détaillé dans la scène desert.

1 Planète

La planète suit la logique du lowpoly : des graphismes plutôt "enfantins", qui permettent de s'affranchir de la contrainte de réalisme tout en proposant des scènes esthétiques. C'est une planète habitable, placée seule dans l'espace. Elle tient le rôle de "menu", puisque c'est par elle qu'on passe d'un environnement à un autre via des îles.

Le mesh de la planète est généré par tesselation, en partant d'un isocaèdre. Au contraire d'une sphère classique, cela permet de répartir les sommets du mesh presque uniformément sur la planète, sans concentration aux pôles. La surface de la planète est un océan, mis en mouvement par les déplacements individuels des sommets du mesh. Chacun d'entre eux se déplace à la fois verticalement et horizontalement selon une sinusoïde, dont la phase est un bruit de Perlin 3D. Ce mouvement est intégralement calculé dans le shader, ce qui évite les échanges CPU/GPU. Pour le mettre en place, il a fallu étendre le VBO pour lui intégrer le déphasage de chaque sommet, ainsi que la direction du déplacement horizontal. Les pulsations du mouvement horizontal et vertical sont des uniforms. L'aspect final imite bien le mouvement de l'eau, même si le fait d'ajouter de l'écume aurait sans doute amélioré la scène. La difficulté de plaquer une texture sur une sphère, sans distorsion, l'a empêché. Le mouvement de l'eau sur le rivage des îles est un des bons points de la scène.

Ces îles sont de six types, chacun désignant un des environnement. Chaque type est représenté au moins une fois. Les positions des îles sont générées aléatoirement sur la surface de la planète, en évitant les collisions. Chacune de ces îles à été crée sur Blender. Cela nous a permis de sculpter des formes assez précises, pour permettre à l'utilisateur de comprendre rapidement vers quel environnement cette île permet d'aller. Les îles de champs, de ville et de montagne sont les plus travaillées, et contiennent de nombreux détails de petite taille qu'on peut découvrir en zoomant. Chaque île est intégrée à la scène via l'import de mesh. La texture est une simple image de 8 pixels par 8, ce qui garantit un import très rapide. Le peu de couleurs disponibles n'est pas un problème pour du lowpoly. Elle est ensuite déformée procéduralement pour suivre une courbure légèrement plus importante que celle de la planète, afin que l'eau ne passe pas à travers lors de son mouvement vertical.

Le système de sélection est inspiré d'un des TD. En maintenant la touche shift, on peut choisir un environnement en cliquant sur une île. Quand la souris passe sur une île prête à être sélectionnée, elle devient rouge.

Il est possible d'ajouter à la planète un nombre quelconque d'avions et de satellites. Ceux-ci suivent une trajectoire circulaire d'axe aléatoire autour de la planète. Les deux objets ont

été crées à partir de $hierarchymeshdrawable$, et n'ont donc pas la précision graphique que peuvent avoir les îles.

Une difficulté lors de la réalisation du projet a été le changement de compilateur entre Windows et Mac, qui a posé des problèmes sur la portabilité du code.

2 Forêt

2.1 Présentation générale

Dans l'environnement forêt, on peut trouver un terrain légèrement vallonné, ainsi qu'un grand nombre de plantes et d'arbres.

2.2 Fonctionnalités

Pour cette scène, nous avons voulu reprendre l'idée globale des TD, qui implémentaient une forêt, tout en complexifiant un peu. Pour cela nous avons implémenté 2 techniques principales :

— Arbres par L-system
— Atlas de textures

Pour la génération des L-System, nous avons implémente une classe générant un arbre à parti d'une grammaire de formes. Les règles pouvant être assez simple comme vu lors des amphis, il a toutefois été complexe de trouver des arbres de forme jolie, et peu géométriques. Une perspective pour améliorer la génération des arbres serait de rajouter un aspect probabiliste dans la grammaire de forme du L-System. Les éléments en seraient beaucoup moins réguliers.

Lors de l'implémentation de l'herbe, nous voulions quelque chose d'assez irrégulier, pour simuler la broussaille présente habituellement en forêt. Une solution a été de fonctionner par atlas de texture, et un type d'herbe choisi aléatoirement lors de l'initialisation. Il a été possible d'envoyer ce nouvel argument au shader dans un nouveau vbo, grâce au type mesh float drawable, qui avait déjà été crée pour la planète, dérivé du type mesh drawable.

Une autre solution aurait été de générer les particules par instanciation, comme pour la scène de montagne que nous détailleront ensuite, mais nous avons empiriquement constaté une fois le code fini que la chute de FPS pour 25000 particules avec un vao astucieux était moindre que celle des particules.

Finalement, nous avons le rendu attendu pour la scène de foret, toutefois, la chute de FPS est assez importante notamment à cause du nombre de particules mais surtout du nombre d'arbres, qui sont individuellement longs à afficher. Une solution aurait pu être de faire d'additionner les mesh de tous les arbres, mais cela aurait nécessite une refonte des classes déjà crées en fin de projet.

Dans un dernier temps, avec de l'instanciation ainsi que des atlas de texture, nous aurions voulu implementer un feu de bois au centre de la forêt. Le code est presque fonctionnel dans la classe particule mais ne produit pas encore un résultat joli

Avec un ordinateur possédant une bonne carte graphique, il est facile et souhaitable d'augmenter énormément le nombre de particules et d'arbres.

3 Montagne

3.1 Présentation générale

La montagne est un environnement assez complet, qui met en scène un relief très marqué avec des chutes de neige et un fort brouillard. Chacune de ces techniques est optimisée permettant un affichage rapide. La scène est plus facile à visualiser en caméra aérienne, ou en caméra fpv au bord des côtes.

3.2 Fonctionnalités

Pour cette scène, nous avons voulu représenter un paysage de montagne enneigée à l'aide d'un réel travail sur les shaders, et sur opengl.

Pour cela nous avons implémenté 4 techniques principales :

— Heightmap
— Multitexturing du sol
— Brouillard
— Instanciation de particules

L'utilisation des heightmaps sera expliquée pour le désert.

L'idée dans une montagne est la présence de neige en altitude, pour cela nous avons utilisé une technique de multitexturing avec une sigmoide dans le shader associé au sol de la heightmap.

Pour plus de réalisme, nous avons voulu simuler la montagne sous une tempête de neige, pour cela, nous avons dans un premier temps implémenté un brouillard avec une importance exponentielle. Une difficulté ici a notamment été de le faire fonctionner avec tous les shaders environnants(eau et skybox), et dans l'architecture des scènes.

Finalement, nous avons crée un système simple de génération de particules par technique d'instanciation. Cette méthode conseillée par notre coordinateur a été longue à implémenter car nécessitait de sortir de tout ce qui avait été déjà implémenté dans la vcl, en créant des buffers et sous buffers opengl. Le résultat final est satisfaisant car il permet d'instancier un grand nombre de particules sans trop perdre en performances, notamment lorsque la position des particules s'actualise à chaque frame. Le modèle crée est simple, les flocons apparaissent

à des positions aléatoires sur le carte, avec une vitesse initiale puis sont soumis à la gravité. Des modèles d'accumulation entre eux ou au sol auraient pu être réalisés. L'implémentation est assez explicite et permet de générer un système de particules de manière rapide avec des lois de comportement définies.

4 Ville

4.1 Présentation générale

La ville est une ensemble des routes avec leurs croisements, ainsi que de bâtiments générés entre les routes. Une route entoure marque la frontière entre la ville et la plage, qui donne sur l'eau. La ville est générée entièrement procéduralement, et permet la génération de bâtiments de formes et tailles diverses (base carrée / rectangle / ovoïde...). Une voiture parcourt la route extérieure à vitesse fixée. Les reflets des batiments dans l'eau donnent de très bons résultats.

Nous n'avons pas implémenté de système de gestion de collisions, donc il se peut que lors de l'arrivée dans la ville, vous vous trouviez à l'intérieur d'un bâtiment et voyiez un écran gris. N'hésitez pas à avancer pour retrouver les premières routes.

4.2 Fonctionnalités

L'implémentation d'une ville entièrement par génération procédurale a été une étape importante. L'idée de la génération consiste à partir d'un rectangle, qui sera ensuite divisé soit en plusieurs parcs, plusieurs pâtes de maisons circulaires, rectangulaires, ce de façon récursive. Afin de chercher à casser les formes, les divisions, tailles et rotations des bâtiments sont aléatoires.

Il y a plusieurs types de bâtiments, des grattes ciels de différentes formes. Le choix du futur type de terrain se fait aléatoirement lors de la division récursive.

Sur cette scène, nous aurions aimé approfondir la génération des parcs, en générant de manière procédurale des barrières, bancs et chemins, avec une végétation certaine.

5 Champs

5.1 Présentation générale

L'environnement champ est similaire à la ville. On trouve quatre types de champ, délimités par des quadrilatères et un chemin de terre qui les relient entre eux. Chaque champ est cultivé avec un type de plantes. Un chemin entoure l'ensemble des champs, et crée une frontière naturelle avec l'eau. On peut trouver des tracteurs disséminés entre les champs et les chemins.

5.2 Fonctionnalités

La génération de la géométrie est procédurale et récursive. Chaque champ, quadrilatère, est divisé en deux selon une droite "en biais". Ainsi, les champs perdent rapidement leur forme rectangulaire pour s'approcher de quadrilatères quelconques. Pour éviter des champs anarchiques, la distribution des droites de division est biaisée. Le critère indiquant qu'on arrête de diviser les champs est une taille minimale pour les quatre segments du quadrilatère. Ainsi, les champs sont de taille surface globalement uniforme.

Les chemins sont générés sur les droites de division, avec une petite hauteur au dessus des champs. L'ensemble des champs et des chemins est perturbée par un bruit de Perlin de fréquence assez élevée, simulant une campagne très vallonnée.

Les plantes dans les champs sont des billboards, disposés avec un angles les uns par rapport aux autres de manière à éviter les motifs trop répétitifs et à minimiser le nombre de billboards. Globalement, la scène est assez légère à afficher (par rapport aux autres au moins).

Les tracteurs ont été générés par Blender, et sont disposés aléatoirement sur les champs.

6 Désert

6.1 Présentation générale

Le désert est un environnement constitué d'un ensemble de dunes de hauteur modulable. Il est assez simple techniquement mais permet de beaux jeux d'ombres entre la skybox, le sol, et l'eau.

6.2 Fonctionnalités

Le terrain est généré à partir de la lecture d'une heightmap trouvée sur internet. Nos fonctions de lecture lisent des niveaux de gris échantillonnés sur $[0, 255]$, donc avec une résolution assez grossière. Nous devons donc linéariser les hauteurs trouvées pour éviter que les montagnes soient des escaliers. Étant donné les faibles hauteurs des dunes, l'impact n'aurait pas été trop grand, mais cela permet quand même une amélioration.

L'aspect sable de la surface est obtenu par une simple texture qui se raccorde "presque" à elle même. Nous n'avons pas cherché à la travailler pour permettre une jonction parfaite, mais sa bonne résolution rend les défauts rares.

La hauteur des dunes est modulable en temps réel grâce à l'interface utilisateur. En choisissant une petite hauteur, on obtient une frontière moins délimitée entre l'eau et le terrain. Des oasis se forment et permettent d'admirer les reflets. Et poussant la hauteur des dunes à zéro, on voit apparaître le "profil" d'altitude de l'environnement. Ce profil est commun à tous les environnements, et permet de favoriser la transition entre l'eau et le terrain.

Finalement, il faut revenir sur l'implémentation de l'eau, assez réaliste, qui a été une étape importante dans la réalisation du projet, sur laquelle nous avons passé beaucoup de temps.

En effet, il a fallu comprendre comment rendre des objets dans différents FrameBuffers, pour la réflection et réfraction, puis les utiliser comme texture à appliquer sur le mesh d'eau.

Afin de ne pas avoir un élément statique, nous avons ensuite utilisé la technique de plaquage de déplacements et de normales, afin que la texture soit déformée au cours du temps, comme si l'eau était animée. d'un mouvement aléatoire avec des remous. Il a finalement été implementé une sorte d'effet Fresnel dans le shader lié à la surface, où le coefficient de réflection/réfraction change en fonction de l'angle sous lequel est vu la surface.

Le résultat est assez satisfaisant, mais diverses implémentations supplémentaires auraient pu être envisagées, notamment de l'écume, des vagues, avec le système de particules qui a été vu précedemment.

Conclusion

Nous avons pris énormément de plaisir, et d'heures à réaliser ce projet, nous espérons qu'il vous plaira.