Programmation graphique 3D
M1 Informatique I3D - S1 Automne 2020–2021
Projet

Projet : Un système solaire en OpenGL

Introduction

Dans ce projet, je vous propose de modéliser un système solaire complet (ou presque). Il vous est demandé de créer une scène 3D qui contiendra l'ensemble des fonctionnalités listée dans le sujet ci-dessous. Comme pour les TP, vous réaliserez votre programme avec l'API web d'OpenGL : WebGL. Vous avez le droit (et c'est même fortement recommandé) d'utiliser la librairie easyWGL qui vous a été fournie (pensez à regarder la doc dans le fichier README). Je prêterai attention à la propreté de votre code (lisibilité, commentaires, refactoring ...).

Modalités de rendu : Ce projet est à rendre pour le 06/12 à 20h au plus tard. Vous le déposerez sur la page moodle *GraphicsProgramming*, dans le dépôt qui a été prévu pour çela. Je vous demande de **déposer juste une archive** contenant votre/vos code(s) JavaSCript, le fichier index.html qui permet de lancer le script, la librairie easyWGL et toutes les ressources nécessaires. **Vous nommerez cette archive nom_prenom.zip** (ou tar.gz comme vous voulez).

Pour démarrer rapidement, vous trouverez à coté de ce sujet les ressources suivantes :

- Un répertoire qui contient la librairie easyWGL et sa doc dans le fichier README.md. **PRENEZ CETTE VERSION !** (elle a été légèrement modifiée pour faire ce projet)
- Un répertoire images/ qui contient des textures qui vous serons utiles
- Un fichier *base_projet.js* qui contient une base du programme que vous devez écrire avec quelques indications pour vous aider dans le projet.

1 Partie I : On positionne les objets correctement dans la scène

Dans un premier temps, vous allez vous occuper d'organiser la scène de manière à avoir un système solaire complet et fonctionnel puis vous ajouterez ensuite des effets pour le rendre plus vivant (partie II).

1.1 Ma première planète

Vous pouvez commencer par modéliser une seule planète. Pour ça il vous faut une sphère !

Aide : Dans la librairie easyWGL, on vous fournit des maillages pour certaines primitives de bases (cube, tore, sphere, cylindre ...) et le "renderer" qui va avec. Avec cette solution, pas besoin de gérer les VBO, EBO, VAO ... tout est fait pour vous ! Vous pouvez ensuite accéder dans le vertex shader aux positions, aux normales et même aux coordonnées de texture. Regardez dans la doc (fichier README) à partir de la ligne 765 pour le maillage et à partir de la ligne 803 pour le renderer.

1.2 Tout un système !

Maintenant que vous avez une planète, il vous reste à créer l'ensemble du système solaire avec notre étoile (le Soleil) et les 8 planètes qui orbitent autour de lui (Mercure, Venus, la Terre, Mars, Jupiter, Saturne, Uranus et Neptune). Ici, je vous demande de positionner tous ces objets de manière correcte. Cela nécessite de respecter : la position des planètes les unes par rapport aux

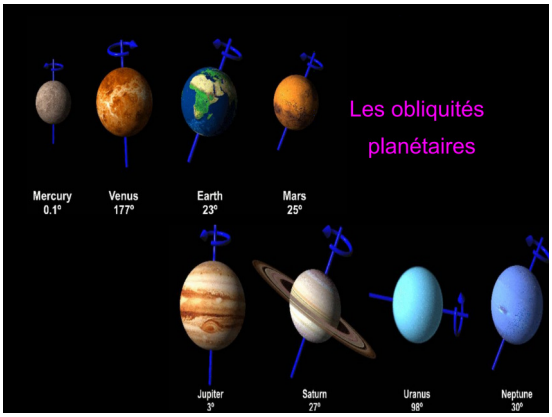
autres, dans le bon ordre, sur un même plan et selon les bonnes distances !

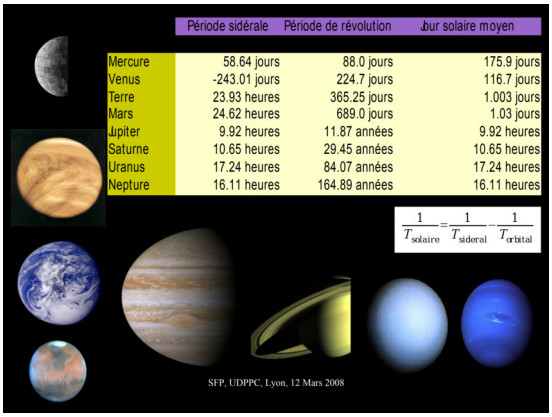
Aide : Réfléchissez à vos repères ! C'est surement une bonne idée de positionner le soleil en (0, 0, 0) dans **l'espase monde** et de faire en fonction de lui pour placer ensuite les autres objets du système solaire.

Transformations ! : vous devez calculer les matrices de transformations pour mettre tous les objets à leur bonne place dans la scène. Je vous demande également de respecter les tailles des planètes et leurs bonnes rotations. Cela signifie de respecter pour chaque planète :

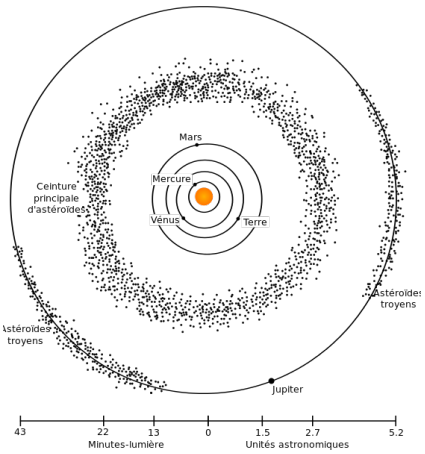
- son axe d'inclinaison
- le sens de sa rotation
- la vitesse de rotation sur elle-même (= période sidérale)
- la vitesse de rotation autour du soleil (= période de révolution). Ici, vous allez vous contenter de rotations circulaires avec le soleil comme centre (en vrai ce sont des ellipses).

Pour les distances et les tailles, vous les trouverez facilement sur internet. A vous de proposer une échelle convenable ensuite (vous pouvez tricher un peu si besoin pour que tout soit bien visible). Pour les rotations, je vous aide un peu :





1.3 Une ceinture d'astéroïdes



Je vous demande maintenant de **modéliser la ceinture d'astéroïdes** que l'on trouve dans le système solaire entre les planètes rocheuses (Mercure, Vénus, la Terre et Mars) et les planètes gazeuses (Jupiter, Saturne, Uranus et Neptune). Vous avez à disposition un fichier *rock.obj* qui contient le maillage d'un rocher, que l'on utilisera pour modéliser un astéroïde. Vous pouvez le charger avec la fonction `Mesh.loadFile` (doc ligne 792). Ici encore, vous pouvez utiliser un *renderer* pour afficher ensuite ce maillage.

Aide : Vous devez utiliser **l'instanciation** (instancing) pour afficher un grand nombre d'asteroïdes en gardant de bonnes performances : en utilisant `rend = mesh.instanced_renderer([3, VBO([50.0,50.0, 25.0,25.0], 2), 1], 0, 1, 2);` puis quand vous faites le `rend.draw()` il faut ajouter le nombre d'instance en deuxième paramètre, après le type de primitive.

Tous les astéroïdes devront être disposés sur un cercle (en choisissant un rayon r). Pour ajouter un peu d'aléatoire, vous pouvez faire légèrement varier la position, la rotation et la taille de chaque astéroïde. Vous pouvez utiliser la fonction `Math.random()` de JavaScript qui renvoie une valeur comprise entre 0 et 1 (pas besoin de définir de *seed*).

Aide : Il faudra envoyer une matrice *model* pour chaque instance, au shader qui s'occupe du rendu des astéroïdes. C'est un peu compliqué en OpenGL d'envoyer une matrice 4x4 via un VBO, vous trouverez de l'aide dans le fichier *base_projet.js*.

2 Partie II : On améliore la qualité du rendu et on rajoute des effets

2.1 Texturing

Pour ajouter des détails aux objets vous allez utiliser des textures. Il vous est donc demandé de **plaquer des textures** sur les planètes, sur les astéroïdes et sur le soleil. Vous trouverez dans les ressources associées, un répertoire images qui contient toutes les textures dont vous aurez besoin. Si elles ne vous conviennent pas, libre à vous d'en utiliser d'autres ;).

Aide : Vous utiliserez ici des textures 2D. Vous pouvez utiliser là encore la librairie easyWGL pour aller plus vite:

```
tex = Texture2d();
tex.load("/images/2k_earth_daymap.jpg",gl.RGB8);
```

Ensuite, pour plonger la scène dans un environnement, vous devez mettre en place **une skybox** en plaquant une texture de type cubeMap sur un cube (comme en TP).

Je vous propose ensuite d'ajouter des détails autour de la planète Terre en utilisant une texture pour **ajouter des nuages qui tournent autour de la Terre**. Vous trouverez une texture pour ça dans le répertoire images.

Aide : Pensez à la fonction mix() dans le shader.

2.2 Lightning

Avec un peu de lumière ce sera quand même plus beau non ? C'est bien le soleil qui éclaire les planètes ? Pour l'éclairage, je vous demande d'utiliser les BRDF que vous avez vu en TP : Ambient + Diffus + Spéculaire. Ici, la "couleur" du diffus sera la "couleur" de la texture appliquée à vos objets. Pour l'ambient vous pouvez mettre sa couleur à (0.0, 0.0, 0.0) et pour le spéculaire à (1.0, 1.0, 1.0). A vous de paramétrer le lobe spéculaire (la rugosité).

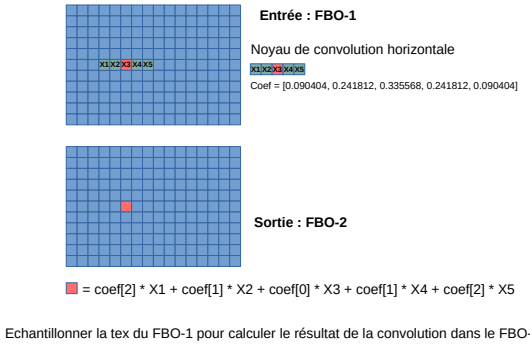
2.3 Glow

Pour simuler une atmosphère autour de la terre ou encore un halo orange autour du soleil, vous allez implémenter un effet de *glow*. Pour cela il faut utiliser **2 FBO**, chacun avec une texture de couleur. Voici comment vous devez procéder pour implémenter cet effet :

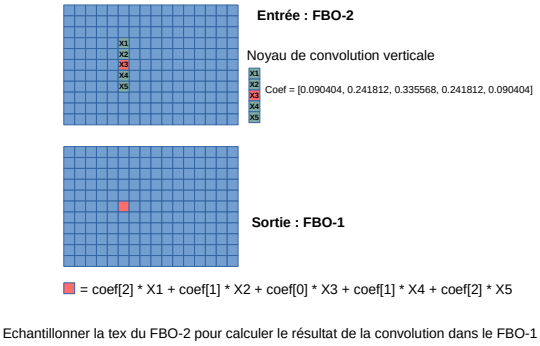
1. Faire le rendu de la sphère que l'on veut "faire briller" sans texture mais avec la couleur du glow que l'on veut. Mettre le résultat de ce rendu dans le FBO-1.
   Fragment shader : s'occupe juste d'appliquer la couleur passée en uniform.
2. Appliquer un effet de **flou gaussien** (Gaussian Blur : https:en.wikipedia.orgwikiGaussian_blur) en faisant le rendu sur un quad full-screen. On peut implémenter l'algorithme en deux passes avec un noyau de convolution 1D :
   (a) un blur horizontale sur la texture du FBO-1 en mettant le résultat dans le FBO-2
   (b) un blur verticale sur la texture du FBO-2 en mettant le résultat dans le FBO-1

   Vous pouvez utiliser un noyau de convolution de taille 5 en prenant les coeficients suivants : 0.090404 - 0.241812 - 0.335568 - 0.241812 - 0.090404.
   L'effet devient intéressant en réalisant cette étape plusieurs fois.
   Fragment shader : La couleur de sortie du fragment est calculée par l'application d'une convolution sur l'échantillonnage de la texture du FBO calculé précédement (pour une passe avec le FBO-2 on envoi la texture du FBO-1 et inversement).

3. Faire le rendu du contenu du FBO-1 sur un quad full-screen en activant le **blending** et le test de profondeur. Le blending doit juste être additif (gl.ONE, gl.ONE) pour rajouter l'effet de glow par dessus le reste de la scène.
   Fragment shader : s'occupe juste d'échantillonner la texture associé au FBO-1 qui sera envoyée par uniform (et qui doit contenir le résultat final du blur). On peut multiplier l'échantillonnage de la texture par un coefficient pour accentuer l'effet.

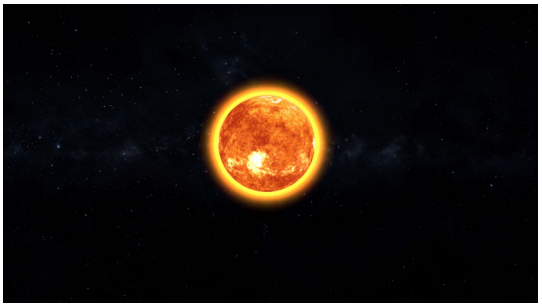


(a) 1ère passe : application du noyau de convolution horizontale



(b) 1ère passe : application du noyau de convolution verticale

(a) Le Soleil sans effet de *glow*



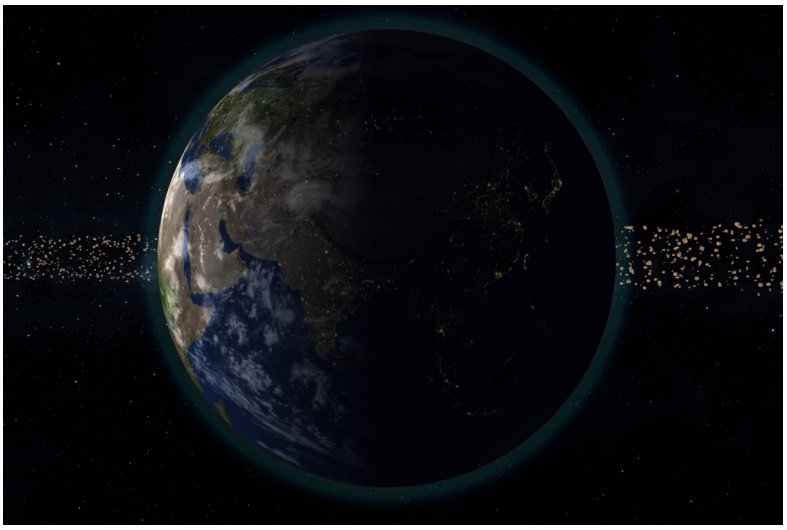
(b) Le Soleil avec effet de *glow*

Aide : Il sera peut-être nécessaire de faire varier la taille des FBO

3 Partie III : Pour aller plus loin

3.1 Jour et nuit

Pour la planète Terre, je vous propose d'ajouter la gestion de deux textures, une pour le jour et une pour la nuit. Pensez à la fonction mix() dans les shader, **selon l'intensité lumineuse** reçu par la planète. Toutes les ressources nécessaires vous sont fournis dans le répertoire images.



3.2 Et vos idées

Je vous demande de proposer quelque chose d'autre selon vos envies. Je vous laisse libre d'implémenter ce que vous voulez qui pourra améliorer votre scène 3D. Cela peut être dans les interactions, dans la qualité du rendu ...
Une (petite) partie de la note sera attribuée à vos propositions d'améliorations.