Thomas BARTHELEMY
Antoine LAROYE
Option RV – 2018/2019

11/02/2019

CONAV
-
WebGL

Introduction

Le TP fait par Alexandre Anchyse et Pierre-Louis Antonsanti donne quelques bases de Javascript pour celles et ceux (comme nous) qui n'ont jamais pratiqué ce langage. Les quelques éléments détaillés sont assez clairs sauf pour l'explication sur la portée des variables (§ **this et _this**), où notamment les exemples de codes sont trop chargés pour être clairs dans le cadre des explications qui les accompagnent.

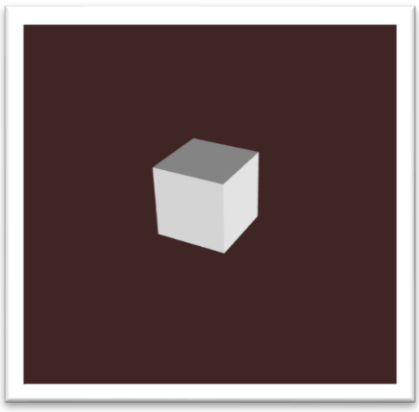
Pendant tout le TP, le plus compliqué est de debugger… Quand il y a une erreur, aucune information ne permet de déterminer où elle se trouve ! La scène n'est tout simplement pas rendue (écran blanc)… Peut-être devrait-il être précisé en début de sujet que F12 permet d'afficher la console de la page web, où on peut voir les erreurs de nos scripts…

Partie I

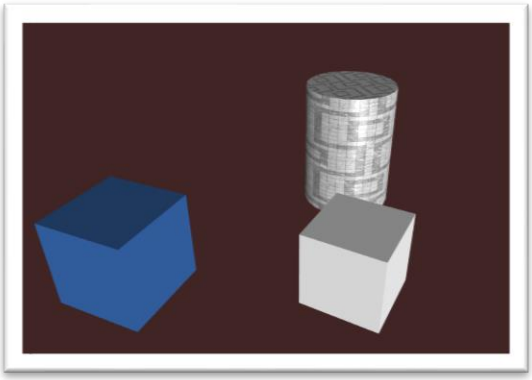
I.1 Scène : mesh, materials, lumières et skybox

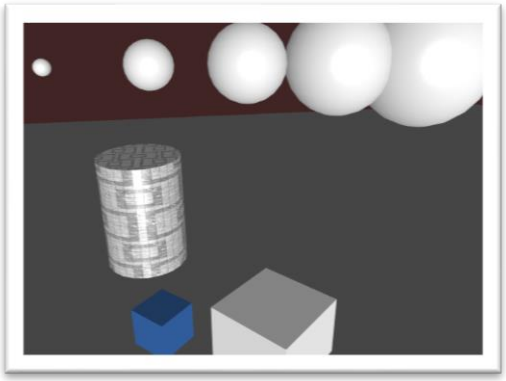
La doc de BabylonJS permet de trouver assez simplement quelques infos importantes, comme choisir la couleur de fond ainsi qu'activer les collisions. Cela nous permet d'arriver à la suite du TP, où l'on nous demande de créer une scène avec un certain nombre d'éléments… Mais cette étape est difficile à aborder au début car la scène fournie dans le TP fonctionne mais n'est pas claire… La caméra est placée dans une box sans texture (pas une skybox), et une boîte est aussi placée dans le décor mais n'est pas visible !

Il aura fallu fouiller et modifier plusieurs lignes de codes avant d'avoir quelque chose de simple et visible : un cube blanc placé devant la caméra (peut-être faudrait-il commencer par là).



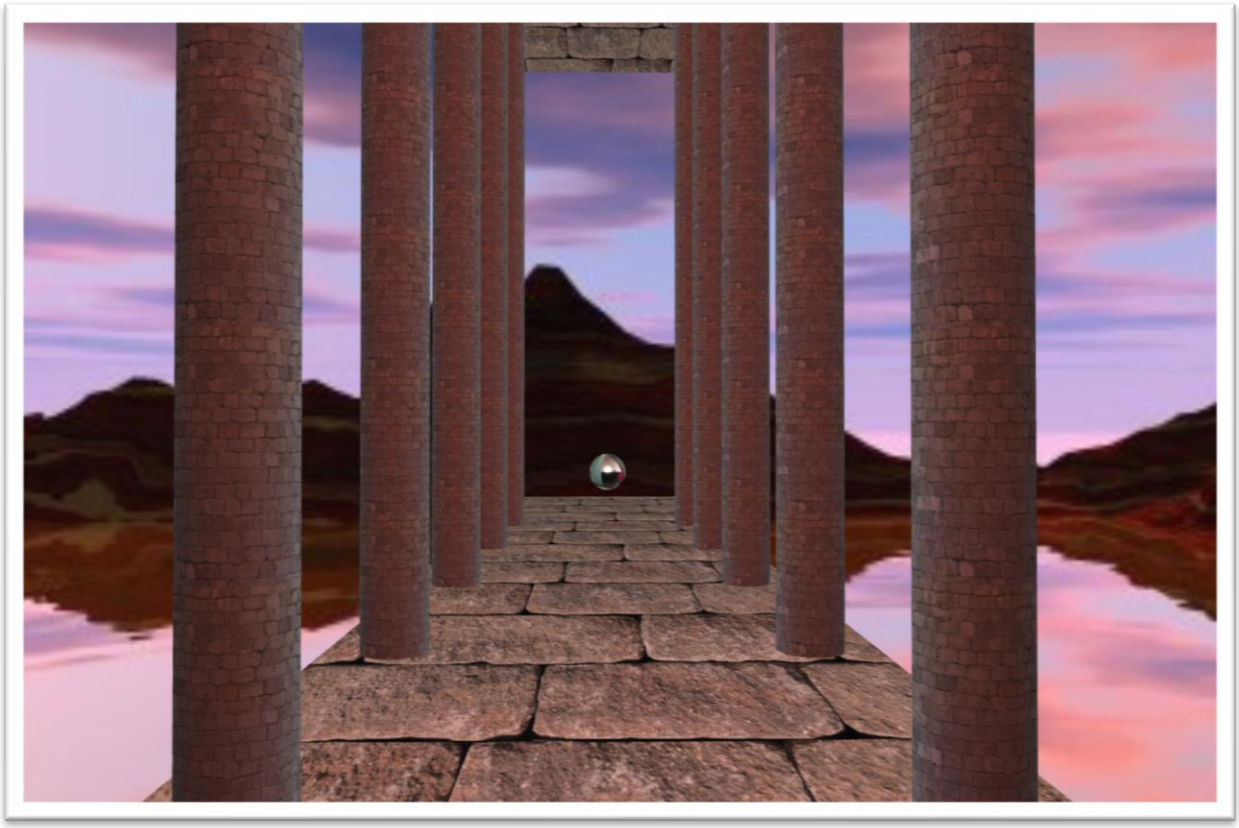
Ensuite, nous avons testé un peu les formes de bases et comment modifier leur position/apparence…





Finalement, on a construit une scène avec ces éléments :

- Meshes texturés et positionnés
- Skybox
- Lumières : hémisphérique (blanche) + directionnelle (rouge-orangée)
- Sphère avec texture vidéo (moi au bureau qui glisse sur ma chaise)

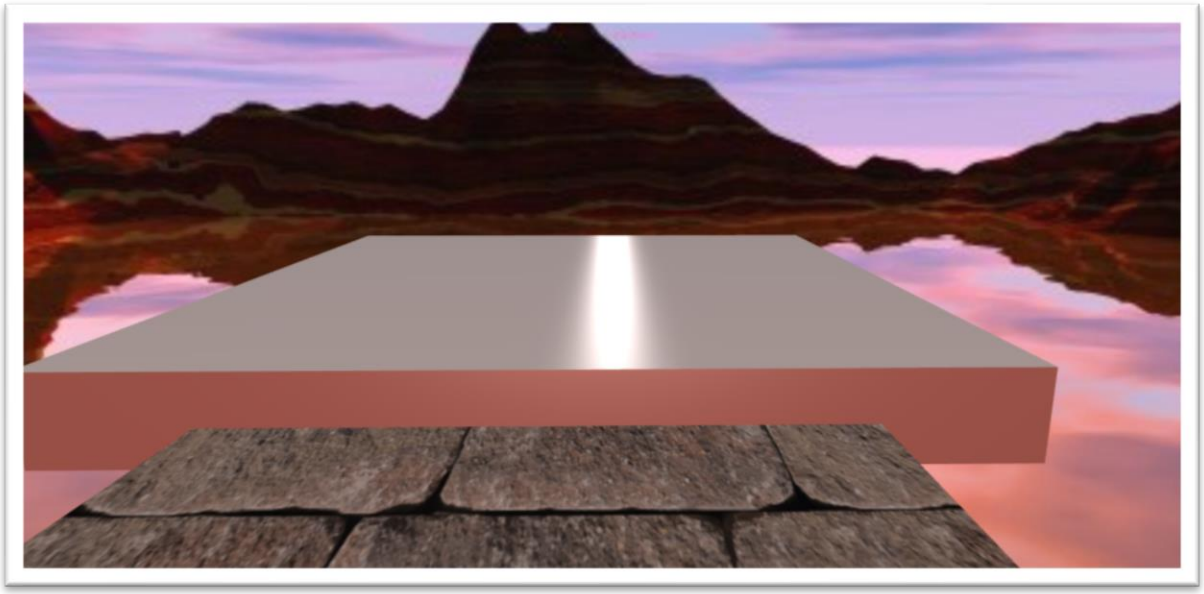


On a récupéré les 6 images de la skybox sur un site qui donne les images suffixées par [_bk, _ft, _lt, _rt, _up, _dn] au lieu des [_px, _nx, _py, _ny, _pz, _nz] requis par BabylonJS, et il faut faire la ''conversion'' à la main (c'est long et pas pratique). Peut-être que plus de détails sur les skybox permettrait d'aborder ce sujet plus facilement. Le lien donné dans le pdf du TP : http://3delyvisions.co/skf1.htm conduit sur un site de rideaux de douche, c'est normal ?

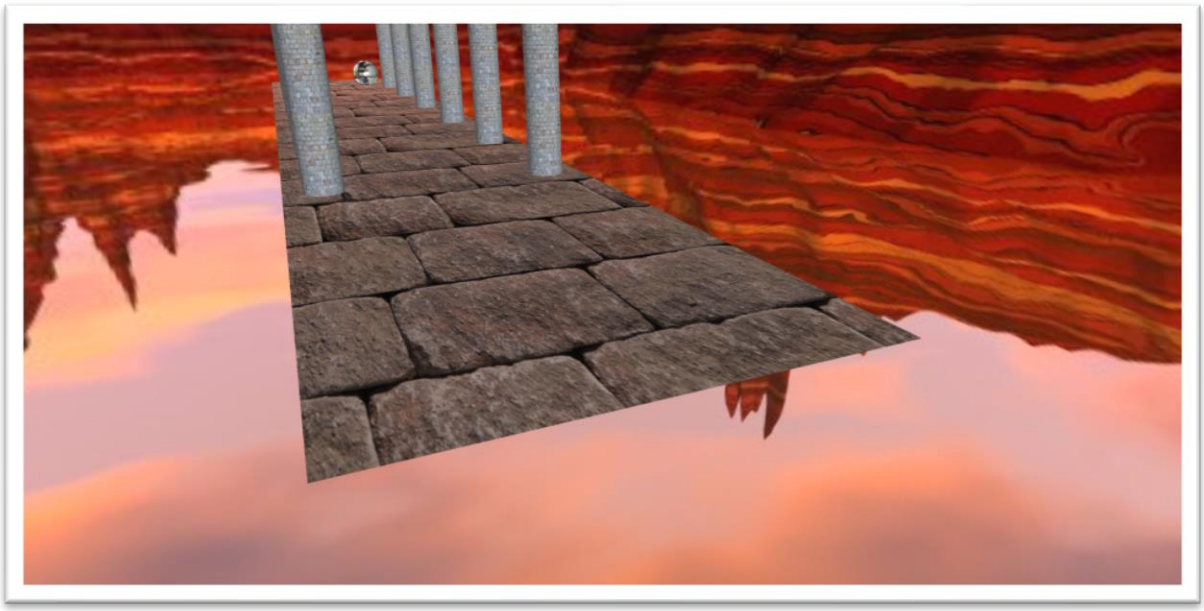
I.2 Animations et collisions

La partie animation est assez peu détaillée dans le TP. La doc de BabylonJS permet de comprendre comment animer nos meshes, mais intégrer ces lignes de code dans les scripts fournis pour le TP est parfois laborieux lors de cette étape… Finalement, on a réussi à animer la vidéosphère (la sphère avec la texture vidéo) en translation, ainsi qu'une plateforme qui va de bas en haut (ascenseur).

Une difficulté importante lors de cette étape a été la collision avec l'ascenseur… En effet, la collision ne se fait que lorsque la caméra est en mouvement. Donc si l'ascenseur est en bas, on peut aller dessus ; puis il va monter, et le joueur reste au même endroit : il flotte dans le vide ! Et si le joueur bouge et que l'ascenseur n'est plus en bas, alors il tombe (évidemment).



Quand l'ascenseur est en bas, on peut monter dessus



Et quand il remonte, le joueur reste dans le vide

En revanche si le joueur reste en mouvement sur l'ascenseur, alors il monte avec ! (mais pas très bien quand même). On a pas réussi à debugger cela juste avec des colliders…

Partie II

II.1 Déplacements

Dans le sujet, on nous fait faire tout pleins de trucs compliqués pour gérer les mouvements de la caméra (définition des axes de mouvements, qui sont initialisés à faux, puis passés à vrai lors de l'appui d'une touche, calcul de l'angle de la souris pour avoir la bonne direction etc…).

Dans un tuto (http://www.demonixis.net/premier-pas-en-3d-avec-webgl-et-babylon-js/), toutes ces étapes sont réalisées en quelques lignes :

```
this.camera.keysUp = [90]; // Touche Z
this.camera.keysDown = [83]; // Touche S
this.camera.keysLeft = [81]; // Touche Q
this.camera.keysRight = [68]; // Touche D;
```

En mettant ce code dans la fonction *_checkMove* du prototype de la classe Player, on a un déplacement rapide et fonctionnel de la caméra. Pas besoin de calculer des angles entre le *forward* de la caméra et l'axe sur lequel on souhaite se déplacer. Néanmoins, on a quand même essayé de faire le déplacement comme le sujet du TP proposait, mais sans succès… On a pensé que le problème de collision venait du fait qu'on n'utilise pas (avec ces 4 lignes de code ci-dessus) *moveWithCollision(vector3)* mais on n'a pas pu le tester du coup.

II.2 Saut

La partie la plus ~~galèèèèèère~~ challenging du TP. On a essayé de suivre les indications données dans le TP pour faire un saut, mais c'était impossible. On a donc créé un *EventListener* qui déclenche une animation de saut quand on appuie sur espace. Malheureusement cette technique est limitée, car si on spamme la barre espace pendant le saut, le joueur resaute et resaute… Si on ne touche à rien, il reste en l'air, et si on bouge, il tombe (s'il y a un sol en-dessous ça va, mais sinon il tombe dans les méandres du temps et de l'espace du framebuffer).

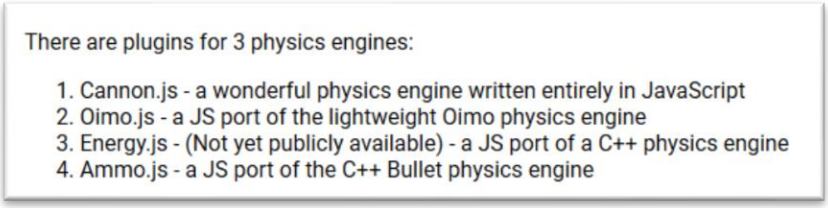
Conclusion

Ce TP permet de découvrir le framework Babylon JS et de s'amuser avec. Nous avons pu voir, comme avec OpenGL, OpenSceneGraph et Unity, comment créer des meshes simples, appliquer des textures, créer des lumières etc… Notre scène reprend tous ces éléments : on y trouve des primitives texturées, animées, une skybox, des sons, on peut se déplacer et sauter… En faisant ce TP, on a parcouru un certain nombre de tutoriels et de forums afin de réussir à surmonter les obstacles qu'on rencontrait. En particulier, on a eu affaire à 2 principales difficultés :

- La collision avec des meshes en mouvement
- Le saut

Et sur Internet, les utilisateurs de Babylon JS recommandaient souvent d'utiliser un moteur physique. Cela permet de créer un *impostor* pour les meshes, et donc de mieux gérer les collisions avec des objets animés, ainsi que d'appliquer des forces à des objets. Cela permettrait donc de résoudre assez simplement ces 2 difficultés majeures rencontrées lors du TP. En effet, il suffirait alors de définir le vecteur gravité et de donner une impulsion verticale au joueur (la valeur de la force détermine la hauteur du saut). La chute se fait ensuite toute seule grâce au moteur physique et la gravité de la scène. Pour les collisions, l'utilisation d'un *impostor* permettrait de gérer dynamiquement les collisions (et cela empêcherait des blocages avec les collisions d'objets en mouvement comme notre ascenseur).

Apparemment, Babylon JS possède des plugins pour des moteurs physiques :


There are plugins for 3 physics engines:

1. Cannon.js - a wonderful physics engine written entirely in JavaScript
2. Oimo.js - a JS port of the lightweight Oimo physics engine
3. Energy.js - (Not yet publicly available) - a JS port of a C++ physics engine
4. Ammo.js - a JS port of the C++ Bullet physics engine

Capture de la doc de Babylon JS (https://doc.babylonjs.com/how_to/using_the_physics_engine)

Et il serait intéressant de voir l'utilisation de l'un de ces plugins dans le cadre du TP. Ainsi, on apprendrait à ajouter des *impostors* (à mettre sur la caméra et les objets animés pour les collisions), et créer des forces pour faire quelques interactions (force verticale lors du saut, chute/lancer d'objets…). Cela permettrait de faire une scène proche de ce qu'on sait déjà faire sur Unity !

Au passage, on a essayé d'intégrer un moteur physique, mais on n'a pas réussi à trouver comment le lier/activer dans notre scène (et pourtant on a essayé longtemps, en recoupant des infos de moults forums). S'il était possible d'avoir un tuto qui permet de faire :

1. Des formes simples (meshes + textures)
2. Des lumières
3. Des déplacements (moins compliqués que la méthode fournie)
4. Des collisions
5. Intégrer un moteur physique pour gérer des actions

Avec pour chaque partie quelques explications sur ce qu'on est en train de coder, et quelques questions pour confirmer l'étape, cela permettrait de découvrir (parce que le webGL, on ne connait vraiment pas du tout) les fondamentaux de ce langage sans trop galérer.