TP Three.js 3

Licence Informatique 3ème année

Année 2023-2024
durée : 3h

L'objectif de ce troisième TP est d'expérimenter la génération des ombres en `Three.js`, ainsi que l'utilisation de contrôleurs permettant de faciliter l'animation des caméras.
Dans un premier temps, récupérez l'archive jointe à cet énoncé et extrayez les dossiers et fichiers qui s'y trouvent dans un sous-dossier nommé TP3 qui se trouvera dans le dossier où vous réalisez vos TP d'informatique graphique. Après extraction vous disposez :

- d'un dossier `js` qui contient divers scripts qui seront utilisés durant le tp, ainsi que le fichier `mesObjets.js`. Ce dernier contient la fonction `cubeColore(c)` qui permet de créer un cube de côté `c`, centré à l'origine, dont les 6 faces sont de couleurs différentes (correction du dernier exercice du TP précédent) ;
- un fichier `tp3_threejs.html` qui permet l'affichage du cube défini dans le fichier `mesObjets.js`.

Ce TP devra être rendu à la fin de la séance, sous la forme d'une archive nommée `TPIG3-XXX`, où `XXX` sera remplacé par votre nom de famille. Elle devra contenir le dossier TP3 et les dossiers et fichiers qui y seront contenus et sera à envoyer via un site tel que `wetransfer`.

Pour tous les TPs Three.js, vous pourrez consulter la documentation officielle en ligne, à l'adresse :
`https://threejs.org/docs/index.html`

Exercice 1

Dans ce premier exercice, on vous demande de remplacer le cube coloré par 3 objets au sein de votre scène (voir figure 1) ; ces 3 objets posséderont un matériau de type « Lambert » et leur géométrie sera :

- une sphère jaune de rayon 1, placée en $x = 2$;
- un disque rouge de rayon 1, placé en $x = -2$ (voir objet `Circle` dans la documentation) ;
- un « noeud torique » bleu de taille 0.5 et de rayon 0.2 (voir objet `TorusKnot` dans la documentation).

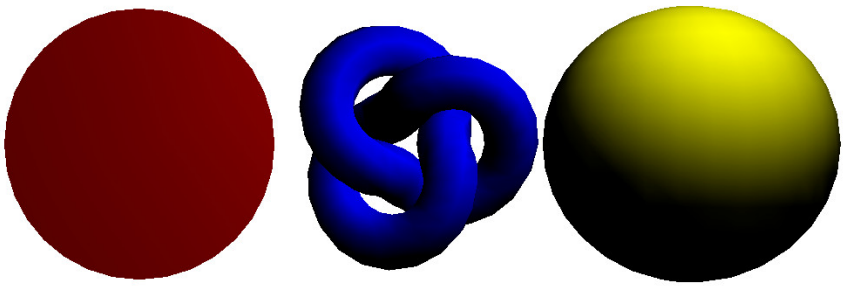


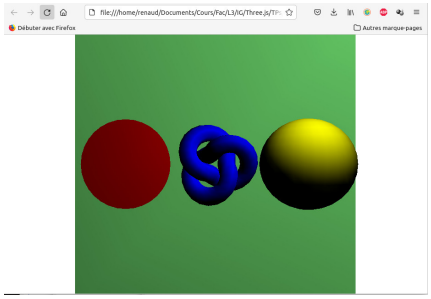
FIGURE 1 – Aperçu de la scène à obtenir.

Lorsque ces objets auront été correctement placés, vous développerez une fonction nommée « update() » dont l'objectif sera de mettre à jour la position / l'orientation de chaque objet à chaque *frame* afin de pouvoir les animer. Les animations à réaliser vous sont données ci-après et il est conseillé de les réaliser les unes après les autres en les testant à chaque étape :

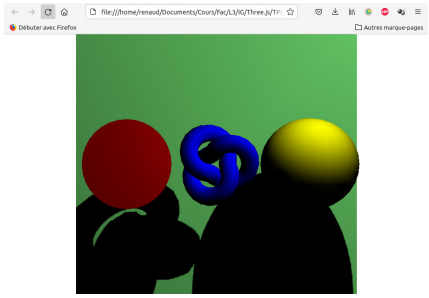
- le disque devra tourner autour de son axe Oy à une vitesse de 0.01 radians par image ;
- le noeud torique devra tourner autour de son axe Ox à une vitesse de 0.007 radians par image ;
- la sphère devra être animée d'un mouvement montant et descendant alternativement ; elle montera à une hauteur maximum de 2 unités, puis redescendra à une hauteur minimale de -2 unités, et ainsi de suite.

Exercice 2

Dans ce second exercice, on va s'intéresser à la génération d'ombres par les différents objets, avec l'objectif de pouvoir rendre les images obtenues plus réalistes. Afin de pouvoir percevoir ces ombres, vous allez dans un premier temps ajouter un objet derrière les 3 objets actuellement présents. Cet objet sera un « plan » vert (voir l'objet `PlaneGeometry` dans la documentation), de taille 10×10 et placé à $z = -3$. Vous devez obtenir une image similaire à celle apparaissant en figure 2a.



(a) Vue de la scène après ajout du plan arrière vert.



(b) Vue de la scène après activation des ombres.

FIGURE 2 – Vue de la scène sans et avec ombre.

Ajouter les ombres nécessite plusieurs étapes :

1. activer l'ombrage au niveau du renderer :

   ```
   renderer.shadowMap.enabled = true;
   ```

   L'algorithme utilisé pour calculer les ombres est celui de la « *Shadow Map* », qui est désactivé par défaut, car les ombres représentent un surcoût de calcul important ;
2. préciser, pour chaque source, si elle peut générer une ombre ou pas. Ce réglage n'est évidemment pas naturel, mais il permet d'éviter des calculs supplémentaires quand on décide qu'une source ne doit pas générer d'ombre. Chaque source possède ainsi un atrribut `castShadow`, qui est par défaut à la valeur `false`. Pour activer la génération d'ombres à partir d'une source, il faut passer la valeur de cet attribut à `true` ;
3. préciser si un objet peut générer (attribut `castShadow`) et/ou recevoir (attribut `receiveShadow`) une ombre. À nouveau, il s'agit d'économiser du temps de calcul si on ne souhaite pas qu'un objet puisse générer une ombre ou qu'il ne peut pas recevoir d'ombre. Par défaut ces deux attributs sont à `false` pour tous les objets.

Dans cet exercice, il est donc nécessaire (au minimum) d'activer la réception d'ombres sur le plan et la génération d'ombres par les 3 autres objets, pour obtenir un résultat similaire à celui de la figure 2b.

Exercice 3

Complétez votre application de telle sorte que l'appui sur la touche `espace` permette d'activer/désactiver l'animation des objets. De même, l'appui sur les touches `q` et `s` doit permettre de déplacer la source horizontalement (respectivement à gauche et à droite), tandis que l'appui sur les touches `z` et `w` doit permettre un déplacement vertical de cette même source (respectivement vers le haut et vers le bas).

Pour mieux percevoir la position de la source, vous modifierez votre code de manière à créer un objet composé de :

1. la source de lumière ponctuelle présente dans le script fourni ;
2. une sphère rouge, d'un diamètre de 0.1 unité, centrée sur la position de la source ponctuelle. On précise que la source ponctuelle se trouvant **à l'intérieur de la sphère**, celle-ci ne pourra pas être éclairée et apparaîtra noire si vous utilisez un matériau de type Lambert ; il est donc préférable d'utiliser pour cette sphère un matériau de type basique.

C'est cet objet composite qui bougera en réponse aux touches de déplacement de la source.

Exercice 4

Dans l'état actuel de votre application, la caméra est fixe et seuls les objets ont été associés à une animation. Déplacer la caméra peut impliquer des calculs relativement complexes, selon le mouvement

souhaité : déplacement autour de a scène ou d'un objet, déplacement à la première personne etc. La bibliothèque de fonctions `Three.js` fournit un ensemble de contrôleurs, permettant de gérer directement ces mouvements sans avoir à développer de code supplémentaire.

Le contrôleur « OrbitControls »

Ce contrôleur permet de déplacer la caméra autour du repère global de la scène, en effectuant des rotations autour de l'origine de ce repère et/ou des zooms avant ou arrières par rapport à cette même origine.

L'ajout de ce contrôleur à votre script est très simple et s'effectue en deux étapes :

1. créer un contrôleur de type `OrbitControls` en lui passant comme paramètre la caméra qui a été créée (afin que le contrôleur puisse modifier ses attributs) et le « DOM » associé à votre renderer :
   ```
   const controls = new THREE.OrbitControls(camera, renderer.domElement);
   ```
2. ajouter un appel à la fonction `update` du contrôleur dans la fonction d'animation, afin que ses attributs puissent être mis à jour à chaque frame .

Modifiez votre script en conséquence et, après avoir étudié la documentation en ligne sur les interacteurs disponibles, manipulez la caméra associée à votre scène par son intermédiaire.

Remarque importante : le dossier `js` qui vous a été fourni avec ce tp contient le script de ce contrôleur dans le fichier `OrbitControls.js`. Vous penserez à l'inclure dans votre script principal.

Le contrôleur « ArcballControls »

Très similaire au précédent, ce second contrôleur permet de déplacer la caméra autour d'un point de l'espace fixé par l'utilisateur ; par défaut, ce point est initialisé avec les coordonnées de l'origine du repère global, mais il peut être déplacé en double-cliquant sur le point choisi sur la surface d'un objet. Un curseur 3D, composé de 3 cercles colorés placés dans les plans du repère global, vous permet de voir en permamence où se trouve le point de l'espace autour duquel votre caméra peut tourner ou zoomer.

Il s'utilise de manière similaire au premier contrôleur, en précisant que :

— le nom de la classe à utiliser est cette fois `ArcballControls` ;
— son constructeur prend la scène comme troisième paramètre ;
— le fichier script corrrespondant est inclus dans le dossier `js` sous le nom `ArcballControls.js`

Application : Mofidiez votre code pour tester ce nouveau contrôleur à la place du précédent.

Sélection du contrôleur

Modifiez votre code de telle sorte que votre application utilise par défaut un contrôleur de type `OrbitControls`, puis que l'appui sur les touches `a` et `o` permette de changer le contrôleur utilisé respectivement par un contrôleur `ArcballControls` et `OrbitControls`. On précise que pour pouvoir changer le contrôleur utilisé, il faut d'abord supprimer le contrôleur actif par l'intermédiaire de sa méthode `dispose()` (voir documentation)