TP Three.js 1

Licence Informatique 3ème année

Année 2023-2024
durée : 3h

L'objectif de ce premier TP est d'expérimenter quelques éléments de base de Three.js, qui couvrent la modélisation, les matériaux et l'animation. Pour réaliser ce TP, récupérez le fichier `tp1.html` qui est fourni avec cet énoncé et installez le dans un dossier de votre compte utilisateur nommé `TP1` (il est conseillé de créer ce dossier dans un dossier dédié aux TP d'informatique graphique). Vous serez amenés à le modifier au fur et à mesure des exercices.

Ce TP devra être rendu à la fin de la séance, sous la forme d'une archive nommée `TPIG1-XXX`, où `XXX` sera remplacé par votre nom de famille. Elle devra contenir le dossier TP1 et les dossiers et fichiers qui y seront contenus et sera à envoyer via un site tel que `wetransfer`[1] vers l'adresse suivante : `christophe.renaud@univ-littoral.fr`.

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

Exercice 1

Dans cet exercice, vous allez être amenés à compléter le script de base qui vous est fourni pour afficher un premier objet. Ce script prend la forme suivante :

```
<script>

  // Création de la caméra

  // création d'un objet Box

  // création de la scène

  // création du renderer

  // création de la fonction d'affichage/animationx1
  function animer() {
// à compléter
  };

  // lancer l'affichage/animation
  animer();

</script>
```

Il énumère toutes les étapes à compléter pour pouvoir afficher une première image. Afin d'en arriver à cet affichage, suivez les différentes étapes ci-dessous, en prenant garde à tester chacune d'entre-elles dans votre navigateur, afin de corriger les éventuelles erreurs au fur et à mesure :

1. créez une caméra perspective, dont l'ouverture sera de 60° et le ratio de taille d'un pixel correspondra au rapport entre la largeur et la hauteur de votre fenêtre d'affichage (en javascript, vous pourrez utiliser par exemple `window.innerWidth / window.innerHeight`) ;
2. créez un objet `box` en créant d'abord sa géométrie (attributs par défaut), puis son matériau (materiau basique avec valeurs par défaut), enfin le mesh qui regroupera la géométrie et le matériau ;

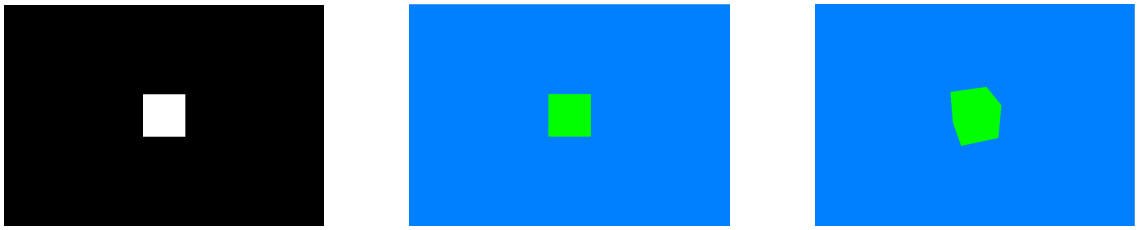
1. Le dossier contenant des scripts `javascript`, il ne pourra pas être envoyé directment par mail, la passerelle antivirus de l'ULCO emêchant l'envoi de ce type de fichiers.

3. créez la scène et y ajouter votre mesh ;
4. créez le renderer, les dimensions de la zone d'affichage devant être les mêmes que celles de la fenêtre du navigateur ;
5. complétez enfin la fonction `animer`.

A ce stage, Ô déception, rien n'apparaît ... Ceci est normal, car par défaut la caméra est positionnée aux coordonnées $(0, 0, 0)$, tandis que le centre de la boite se trouve également à cet endroit. Vous vous trouvez donc à l'intérieur de la boite et il est donc nécessaire, soit de déplacer celle-ci, soit de déplacer la caméra, ce qui va être la solution adoptée dans cet exercice.

Chaque caméra dispose d'un attribut `position`, qui est un vecteur en 3 dimensions permettant de spécifier l'endroit où elle se trouve. Par défaut, ce vecteur est initialisé avec les valeurs $(0, 0, 0)$. Pour déplacer la caméra, il suffit donc de changer les coordonnées de cet attribut en accédant à ses propres attributs nommés `(x, y, z)`.

Application : Déplacez la caméra à la position `(0, 0, 5)`. On rappelle qu'elle est orientée par défaut selon l'axe $-Oz$, que le cube est, par défaut, de côté 1.0 et centré à l'origine. Vous devez donc voir apparaître sa face avant au milieu de votre fenêtre (voir figure 1a).



(a) Rendu après l'exercice 1. (b) Rendu après l'exercice 2. (c) Rendu après l'exercice 3.

FIGURE 1 – Evolution des rendus à obtenir pour les 3 premiers exercices.

Exercice 2

Par défaut, le materiau `basic` est de couleur blanche. Il est possible de préciser sa couleur au moment de sa création, en utilisant son attribut `color`. On rappelle qu'une couleur est représentée par un triplet `(rouge, vert, bleu)`, chaque valeur pouvant être représentée par un nombre réel compris entre 0.0 et 1.0 ou un entier compris entre 0 et 255 (afin de pouvoir être stocké sur un octet). La classe `Color` permet de créer des couleurs de différentes manières (voir pour plus de détails `https://threejs.org/docs/api/en/math/Color`). Affecter une couleur à un matériau `basic` au moment de sa création peut également prendre plusieurs formes, quelques unes étant fournies ci-dessous pour la couleur bleu :

```
const material = new THREE.MeshBasicMaterial( {color: 0x0000ff} );
const material = new THREE.MeshBasicMaterial( {color: 'rgb(0, 0, 255)'} );

const couleur = new THREEE.Color( 0.0, 0.0, 1.0 );
const material = new THREE.MeshBasicMaterial( {color: couleur} );
```

Application : (voir figure 1b)

1. changez la couleur du cube, de manière à ce qu'il apparaisse vert ;
2. changez la couleur du fond d'écran, de manière à ce qu'il apparaisse avec une forte composante bleue et une composante moyenne verte. On précise que la scène dispose d'un attribut `background` qui permet de représenter la couleur de fond. Celui-ci peut être modifié **après** création de la scène.

Exercice 3

Il est possible d'appliquer des transformations géométriques aux objets, afin de les déplacer ou encore changer leur orientation. Dans ce dernier cas, on peut leur appliquer des rotations autour des trois axes du repère, par l'intermédiaire des méthodes `rotateX`, `rotateY` et `rotateZ`. Ces méthodes prennent en paramètre un angle en radian et effectuent la rotation de l'objet appelant selon l'angle demandé autour de l'axe correspondant au nom de la méthode, dans le sens indiqué sur la figure 2.

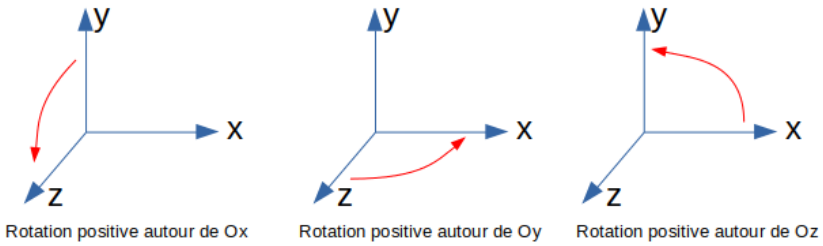


FIGURE 2 – Sens des trois rotations autour des axes du repère pour un angle positif. Le sens est inversé pour un angle négatif.

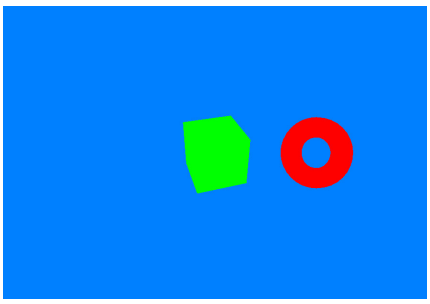
Application :

- modifiez l'orientation du cube, de telle manière qu'il subisse une rotation de 0.6 radians autour de l'axe Ox, puis de 0.3 radians autour de l'axe Oy (voir figure 1c) ;
- modifiez votre fonction `animer()` de telle sorte qu'à chaque appel de cette fonction, le cube subisse une rotation supplémentaire de 0.01 radians autour de l'axe Oy (les rotations se cumulant, ceci aura pour effet de donner une illusion d'animation à votre cube).

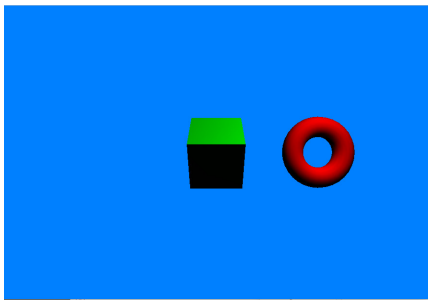
Exercice 4

Ajoutez un tore de couleur rouge à votre scène. Ce tore aura un diamètre de 0.5 unités et un diamètre du tube de 0.2 unités. Vous définirez les valeurs visuellement correctes pour le découpage de la circonférence du tube et celle du tore. L'objet devra être positionné sur la droite de votre cube, son centre étant positionné aux coordonnées $(2.0, 0.0, 0.0)$ (voir figure 3a). Vous modifierez pour ce faire l'attribut position de l'objet.

Comme pour le cube, vous modifierez ensuite votre fonction `animer()` de telle sorte que le tore tourne autour de l'axe OX.



(a) Rendu après l'exercice 4.



(b) Rendu après l'exercice 5.

FIGURE 3 – Evolution des rendus à obtenir pour les exercices 4 et 5.

Exercice 5

Le matériau `basic` utilisé jusqu'à présent ne permet pas de rendre compte des effets d'un éclairage, puisqu'il considère une couleur constante et uniforme sur toute la surface de l'objet auquel il est rattaché. Pour pouvoir prendre en compte ces effets d'éclairage, vous allez modifier le matériau utilisé pour les deux objets par un matériau lambertien, via la classe `MeshLambertMaterial`. Vous conserverez la couleur qui a été mise précédemment, sachant qu'elle sera interprétée cette fois comme un pourcentage de réflexion de la lumière incidente sur l'objet. Ainsi, une valeur $(128, 0, 255)$ sera interprétée comme 50% de réflexion dans le rouge, 0% dans le vert et 100% dans le bleu. Si on note $K_d(\lambda)$ la valeur de ce pourcentage pour chaque longueur d'onde λ (ici uniquement celles du rouge, du vert et du bleu), alors la valeur de l'intensité réflechie par un point de l'objet sera calculée par la formule suivante :

$$I_P(\lambda) = K_d(\lambda) \times I_s(\lambda) \times cos(\theta)$$

où $I_s(\lambda)$ représente l'intensité de la source pour la longueur d'onde λ et θ l'angle d'incidence de la lumière reçue.

Application :

- remplacez les matériaux utilisés précédemment par un matériau lambertien et visualisez le résultat : les objets doivent apparaître ... noirs ; ceci est normal, puisque votre scène ne comporte aucune lumière et, par conséquent, le terme $I_s(\lambda)$ de l'équation précédente est nul ;
- ajoutez une source **ponctuelle** à votre scène, dont les caractéristiques seront les suivantes : source de couleur blanche, d'intensité fixée à 1.0, et placée en $(2, 2, 2)$. Après cet ajout, vous devez obtenir le résultat visible sur la figure 3b.

Exercice 6

Three.js offre un certain nombre d'objets prédéfinis, mais ceux-ci sont restreints à des formes mathématiques classiques ; lorsque l'on souhaite modéliser des objets plus complexes et plus réalistes, il est nécessaire de passer par des `BufferGeometry` dont le rôle est de stocker l'ensemble des **triangles** qui composent la surface d'un objet. La version la plus simple d'un tel objet correspond à un tableau de nombres réels, chacun d'entre-eux représentant une coordonnée d'un sommet d'un triangle (voir figure 4).

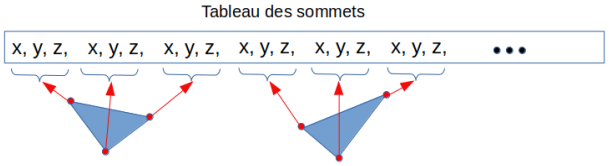


FIGURE 4 – Principe du tableau de sommets stockés dans un `BufferGeometry` : chaque sommet est représenté par ses coordonnées `(x, y, z)`.

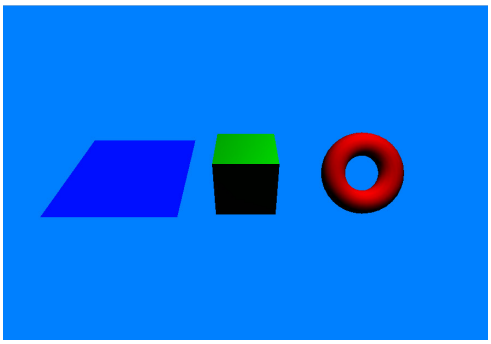
Exemple : Le code de la fonction fournie ci-dessous permet de construire un objet composé d'un seul triangle, auquel sera associé un matériau basic. Un mesh est construit à partir de ces deux informations et retourné par la fonction.

```
function triangle(){

    // créer un BufferGeometry
    const geometry = new THREE.BufferGeometry();

    // créer le tableau contenant les coordonnées des sommets des triangles
    const vertices = new Float32Array( [
        -1.0, 0.0, -1.0, // coordonnées du 1er sommet
        -1.0, 0.0, 1.0,  // coordonnées du 2nd sommet
         1.0, 0.0, 1.0    // coordonnées du 3me sommet
    ] );

    // associer les sommets au BufferGeometry (3 coordonnées par sommet -----------|)
    geometry.setAttribute( 'position', new THREE.Float32BufferAttribute( vertices, 3 ) );

    // créer un matériau basique bleu
    const diffus = new THREE.MeshBasicMaterial( {color: 0x0000ff } );

    // créer le mesh et le retourner
    const mesh = new THREE.Mesh( geometry, diffus );
    return mesh;
}
```

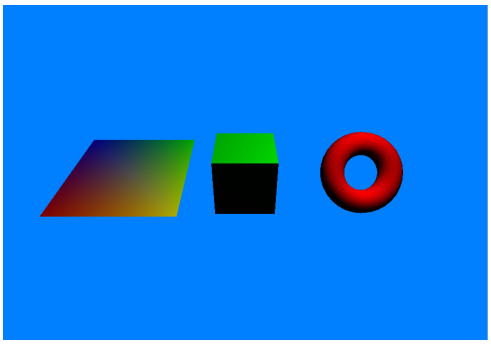
Application

- dans le dossier où vous développez votre TP, créez un sous-dossier nommé `js` qui contiendra les sources javascript que vous allez développer ;
- ajoutez y un fichier nommé `mesObjets.js` dans lequel vous ajouterez une fonction nommée `carreBasic()`. Cette fonction devra construire un carré de côté 2 unités, placé dans le plan OXZ et centré à l'origine. Il devra être composé de deux triangles et de couleur bleu (vous pourrez vous inspirer grandement de la fonction `triangle()` donnée en exemple). Vous prendrez garde à ce que les sommets des deux triangles soient donnés dans un ordre permettant d'obtenir une normale dirigée vers le haut ;

— modifiez votre script principal de manière à (i) inclure le fichier `js/mesObjets.js` et (ii) ajouter le carré dans la scène, à gauche du cube vert qui s'y trouve (voir figure 5a). Comme pour le tore, vous le ferez tourner autour de l'axe Ox.
— vous notez qu'après un demi-tour, le carré disparaît, puis réapparaît à nouveau. Ceci est lié au fait que l'on ne voit que la face externe de l'objet lorsqu'il est tourné vers nous. Pour pouvoir voir aussi la face interne, vous devez spécifier la valeur `THREE.DoubleSide` à l'attribut `side` du matériau. Ajoutez cette valeur après la couleur actuellement présente et testez à nouveau votre script principal.



(a) Rendu après l'exercice 6.



(b) Rendu après l'exercice 7.

FIGURE 5 – Evolution des rendus à obtenir pour les exercices 6 et 7.

Exercice 7

Afin de pouvoir tenir compte des effets lumineux sur la surface du carré, il faut changer le matériau, comme pour les deux précédents objets. Pour un `BufferGeometry`, il faut cependant rajouter des informations afin que les calculs d'éclairage puissent se faire. Il faut ainsi ajouter, pour chaque sommet de triangle, une normale en ce sommet (donc un vecteur en 3 dimensions) et une couleur ou coefficient de réflexion K_d (donc 3 composantes (R,V,B)). Le mécanisme est le même que pour les sommets, à savoir créer un tableau de normales et un tableau de couleurs, puis les ajouter au `BufferGeometry`. En reprenant l'exemple du triangle, l'architecture générale du code à écrire est donc la suivante :

```
function triangle(){

    // créer un BufferGeometry
    const geometry = new THREE.BufferGeometry();

    // créer le tableau contenant les coordonnées des sommets des triangles
    const vertices = new Float32Array( [ ... ] );

    // créer le tableau contenant les normales des sommets des triangles
    const normales  = new Float32Array( [ ... ] );

    // créer le tableau contenant les couleurs des sommets des triangles
    const colors = new Float32Array( [ ... ] );

    // associer les sommets au BufferGeometry (3 coordonnées par sommet -----------|)
    geometry.setAttribute( 'position', new THREE.Float32BufferAttribute( vertices, 3 ) );
    // associer les normaless au BufferGeometry (3 coordonnées par sommet ------|)
    geometry.setAttribute( 'normal', new THREE.Float32BufferAttribute( normals, 3 ) );
    // associer les couleurs au BufferGeometry (3 couleurs par sommet --------|)
    geometry.setAttribute( 'color', new THREE.Float32BufferAttribute( colors, 3 ) );

    // créer un matériau basique bleu
    const lambert = new THREE.MeshLambertMaterial( {
        color: 0xffffff, // reflectance diffuse du matériau
        side: THREE.DoubleSide,// tenir compte des deux faces
        vertexColors: true // il y a des couleurs en chaque sommet
    });
```

```
    // créer le mesh et le retourner
    ...
}
```

Notez ici l'apparition d'un attribut supplémentaire dans le matériau (`vertexColors`) qui permet de préciser qu'un tableau de couleurs est présent dans la géométrie. Dans le cas contraire, c'est la couleur du matériau qui sera utilisée.

Application : Ajoutez une fonction `carreLambert()` à votre fichier `mesObjets.js`, dans laquelle le même carré que précédemment sera créé, mais avec des normales orientés vers le haut, une couleur différente en chaque sommet et un matériau lambertien. Cette fonction sera appelée à la place de celle de la question précédente dans votre script principal et le rendu devra être similaire à celui apparaissant en figure 5b, avec une interpolation entre les couleurs présentes aux sommets des triangles (les couleurs du carré peuvent varier en fonction de vos choix).

Exercice 8

En vous inspirant de ce qui a été vu dans les deux exercices précédents, ajoutez une fonction `cubeColore()` à votre fichier `mesObjets.js`. Cette fonction créera un cube de côté 2 unités, dont chaque face aura une couleur différente et un matériau lambertien. Elle viendra remplacer le carré coloré au niveau de l'affichage.