TP Three.js 6

Licence Informatique 3ème année

Année 2024-2025
durée : 6h

L'objectif de ce dernier TP est d'étudier différentes fonctionnalités supplémenataires de `Three.js` qui permettront de créer un jeu de type *Space Invaders*.

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 TP6 qui se trouvera dans le dossier où vous réalisez vos TP d'informatique graphique. Après extraction vous disposez :

- d'un dossier `js` qui contient les scripts nécessaires à ce tp ;
- d'un dossier `Modeles` qui contient quelques fichiers représentant des modèles 3D dans différents formats ;
- un fichier html pour chacun des exercices à réaliser.

Ce TP devra être rendu à la fin des 2 prochaines séances, sous la forme d'une archive nommée `TPIG6-XXX`, où `XXX` sera remplacé par votre nom de famille. Elle devra contenir le dossier TP6 et les dossiers et fichiers qui y seront contenus. Dans la mesure où l'archive à rendre risque d'être volumineuse, du fait de la présence de modèles 3D, elle sera à envoyer via un site de transferts de fichiers tel que `wetransfer` ou en fournissant le lien vers le dépôt logiciel de votre TP.

Exercice 1 - Collisions entre objets complexes

Dans un TP précédent, vous avez vu comment détecter une collision entre deux géométries simples en utilisant la fonctionnalité de tracé de rayons disponible dans `Three.js`. Bien que parfois utile, cette approche peut rapidement devenir très coûteuse lorsque la géométrie des objets devient complexe, car elle implique alors de lancer un rayon depuis chaque sommet de la géométrie. Dans cet exercice, vous allez expérimenter une autre approche, moins coûteuse, mais généralement moins précise, basée sur la notion de **boite englobante** [1].

Définition

Une boite englobante correspond à un parallélépipède rectangle, dont les faces sont toutes parallèles à l'un des plans $0x$, $0y$ et $0z$, et qui englobe de manière **minimale** [2] un objet 3D. La figure 1a montre un exemple d'une boite englobante sur un cône, tandis que les figures 1b et 1c montrent des boites similaires, mais non englobantes au sens de la définition ci-dessus (la première est trop petite et des parties de l'objet sont à l'extérieur, tandis que la seconde est trop grande).

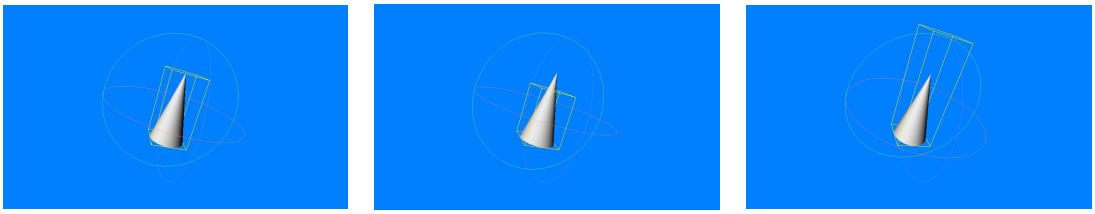
Boites englobantes dans `Three.js`

Sous `Three.js`, une telle boite est représentée par la classe `Box3`. Il y a différentes manières de construire cette boite, la plus simple étant d'utiliser la méthode `setFromObject` de cette classe, en lui passant l'objet à partir duquel on souhaite la construire :

```
const monObjet = new THREE.Mesh(...);

var boite = new THREE.Box3();
boite.setFromObject(monObjet);
```

1. `Three.js` permet également de créer des **sphères englobantes**, avec les mêmes fonctionnalités que celles qui seront vues pour les boites. Elles ne seront cependant pas détaillées dans ce TP.
2. Ceci siginifie que pour chaque face du parallélépipède, il existe au moins un point de la géométrie qui touche cette face et que aucun point de la géométrie ne peut être à l'extérieur du parallélépipède.



(a) Boite englobante correcte. (b) Boite trop petite (c) Boite trop grande.

FIGURE 1 – Exemples de boites englobantes correctes et incorrectes.

Une autre manière est d'initialiser (via le constructeur ou par accès direct) ses attributs `min` et `max`, qui représentent respectivement les coordonnées du sommet le plus « petit » et celles du plus « grand » du parallélépipède (voir figure 2).

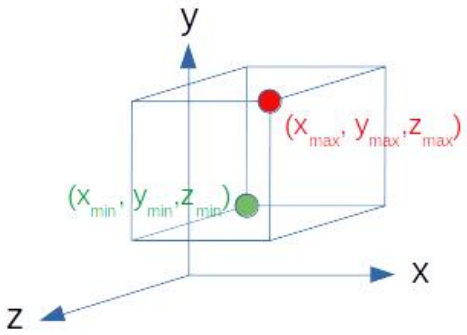
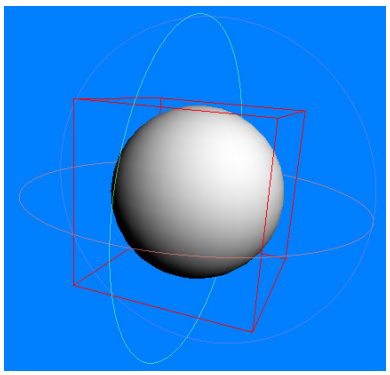


FIGURE 2 – Géométrie d'une boite englobante et position de ses points minimum $(x_{min}, y_{min}, z_{min})$ et maximum $(x_{max}, y_{max}, z_{max})$.

Un objet de type `Box3` ne peut cependant pas être affiché directement, car il ne possède ni géométrie, ni matériau. `Three.js` fournit une classe nommée `Box3Helper` qui permet de contourner ce problème. Son utilisation est alors très simple, puisqu'elle consiste à (i) créer un objet de type `Box3Helper` en précisant l'objet `Box3` associé et la couleur d'affichage souhaitée, puis (ii) à ajouter ce nouvel objet à la scène.

Exemple :

```
// création d'un objet3D
const sphere = new THREE.Mesh(
     new THREE.SphereGeometry(),
     new THREE.MeshLambertMaterial()
);
// création de la Box3 associée à l'objet3D
const boite = new THREE.Box3();
boite.setFromObject(sphere);
// création d'un "helper" rouge associé à la boite
const helper = new THREE.Box3Helper(boite, 0xff0000);
// ajout de la sphere et du helper à la scène
scene.add(sphere);
scene.add(helper);
```



REMARQUES IMPORTANTES :

- Lorsque vous calculez une boite englobante pour un objet, les coordonnées obtenues correspondent à la position de cet objet au moment du calcul. Si l'objet bouge par la suite, les coordonnées de la boite englobante ne sont pas modifiées. Il est donc nécessaire soit de recalculer celles-ci à partir de la nouvelle position de l'objet, soit de les mettre à jour en fonction du mouvement qui a été appliqué sur l'objet ;
- Un objet de type `Box3Helper` utilise les coordonnées de l'objet `Box3` fourni pour son affichage ; il n'y a donc pas de nécessité de le mettre à jour.

Intersection entre boites

L'une des fonctionnalités intéressantes des objets de type `Box3` pour ce TP correspond aux méthodes d'intersection qui leurs sont associées. Il est ainsi possible de déterminer si une boite intersecte une autre boite [3], ce qui permet d'avoir une information appoximative sur l'intersection des objets qui y sont inclus. La méthode d'intersection entre boites de type `Box3` se nomme `intersectsBox` ; elle prend comme paramètre la seconde boite avec laquelle tester l'intersection et elle retourne un booléen précisant si il y a ou non intersection entre les deux boites.

Exemple :

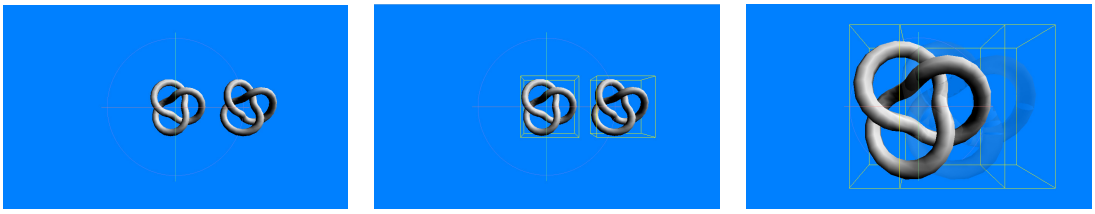
```
Box3 boite01 = new Box3(...);
Box3 boite012= new Box3(...);

if(boite1.intersectsBox(boite1))
  ...
else
  ...
```

Application

Modifiez le script `exo1.html` fourni de telle sorte que (traiter et tester chaque question l'une après l'autre) :

1. il crée deux objets de type `TorusKnotGeometry`, de rayon externe 0.5 et de rayon interne 0.1, positionnés respectivement en $x = 0.0$ pour le premier et $x = 2.0$ pour le second (voir figure 3a) ;
2. il crée une boite englobante associée à chacun de ces objets et ajoute à la scène l'objet de type `Box3Helper` correspondant (voir figure 3b) ;
3. il permet le déplacement automatique selon l'axe Ox de la seconde boite, entre les coordonnées $x = 2.0$ et $x = -2.0$ (un pas de déplacement de 0.01 permet une animation avec une vitesse correcte) ;
4. le matériau du second objet devienne semi-transparent lorsque l'on détecte une intersection entre les deux boites englobantes (voir figure 3c).



(a) Aperçu de la position des deux noeuds.

(b) Rendu des deux boites englobantes.

(c) Rendu par transparence du second noeud en cas d'intersection des boites englobantes.

FIGURE 3 – Rendus à obtenir pour l'exercice 1.

Exercice 2 - Chargement d'objets 3D

`Three.js` ne fournit que quelques objets géométriques de base, qui ne permettent pas de représenter des objets complexes tels que ceux rencontrés dans la vie réelle ou encore dans les jeux vidéos. Il est évidemment possible de modéliser ses propres objets « manuellement », mais cela est extrêmement laborieux.

De manière générale, des objets complexes sont conçus via des outils de modélisation graphique, tels Blender, 3D Studio, Maya etc. Lorsque les objets désirés sont terminés, ils sont sauvegardés dans des fichiers avec un format spécifique au logiciel utilisé ou dans des format dits **d'échange**, qui permettent de recharger les objets dans d'autres logiciels. De nombreux formats de ce type existent, tels que `obj`, `fbx`, `gltf`, etc. `Three.js` permet de gérer un certain nombre d'entre-eux et dans cet exercice, vous allez voir comment charger un objet sauvegardé dans un fichier au format `obj`.

3. D'autres routines d'intersection et d'inclusion sont disponibles, mais ne seront pas illustrées dans ce TP. Vous êtes invités à consulter la documentation en ligne pour plus de détails.

Les fichiers du format obj

Un ou plusieurs objets sauvegardés au format `obj` générent deux fichiers [4] :

— un fichier contenant la description de la géométrie des objets (coordonnées de sommets, normales, coordonnées de texture). C'est un fichier qui porte l'extension `.obj` ;

— un fichier qui contient la description des différents matériaux des objets et qui porte l'extension `.mtl`.

A ces fichiers peuvent être associées diverses images qui représentent les textures utilisées dans les matériaux.

Le dossier `Modèles` fourni avec cet énoncé contient ainsi trois modèles au format `obj`, rangés dans trois sous-dossiers différents.

Chargement d'un fichier de matériaux

Lors du chargement d'une scène au format `obj`, il est nécessaire de d'abord charger les matériaux des objets qui la composent, car ces matériaux doivent être connus au moment de la création des objets. Le chargement d'un fichier de matériaux se fait par l'intermédiaire d'un objet de la classe `MTLLoader` :

```
const mtlLoader = new THREE.MTLLoader();
```

Cet objet peut ensuite être utilisé pour charger un nombre quelconque de fichiers d'extension `.mtl`, par l'intermédiaire de sa méthode `load`, dont le prototype est :

```
load(url:string, onLoad: Function, onProgress: Function, onError:Function)
```

avec

— `url` : l'url (ou l'emplacement) où se trouve le fichier `.mtl` à charger ;

— `onLoad` : la fonction appelée pour charger le fichier. Elle prendra un tableau de matériaux comme paramètres qui sera rempli après le chargement ;

— `onProgress` : une fonction optionnelle qui permet d'afficher la progression du chargement, ce qui peut être utile si les données sont distantes et volumineuses ;

— `onError` : une fonction optionnelle qui traitera une erreur en cours de chargement.

La manière minimale d'utiliser la fonction de chargement est alors résumée ci-après :

```
mtlLoader.load('Modeles/Ufos/Ufo.mtl', function(materials){
  materials.preload();
  ...
});
```

Remarque importante : le chargement du fichier est exécuté de manière asynchrone par rapport au reste du script, afin de ne pas le bloquer si la récupération du fichier est longue. Il est donc nécessaire de tenir compte de cet aspect dans le reste du script qui utilise éventuellement ces matériaux.

Chargement d'un fichier de géométrie

Le chargement de la géométrie suit le même principe, également asynchrone, en utilisant la classe `OBJLoader`. Après avoir créé l'objet de cette classe qui permettra le chargement, il est nécessaire au minimum de lui affecter le tableau de matériaux récupéré par le chargeur de matériaux :

```
const objLoader = new THREE.OBJLoader();
objLoader.setMaterials(materials);
```

De cette manière, il sera possible d'associer le bon matériau à chaque objet chargé. Le reste du fonctionnement est similaire au chargeur de matériaux, avec l'utilisation d'une méthode `load` qui a exactement le même prototype. En pratique ; le chargement de la géométrie est souvent effectué dans la fonction `load` du chargeur de matériaux, une fois ceux-ci chargés :

```
mtlLoader.load('Modeles/Ufos/Ufo.mtl', function(materials){
  materials.preload();
  const objLoader = new THREE.OBJLoader()
  objLoader.setMaterials(materials)
  objLoader.load('Modeles/Obj/Ufo/Ufo.obj', function(objet){
    // utiliser l'objet chargé
    });
});
```

4. On ne détaillera pas ici la structure de ces fichiers.

Application

— après avoir chargé le fichiers `Ufo.obj`, placez l'objet correspondant dans la scène et affichez la structure de l'objet récupéré dans la console, afin de bien comprendre sa structuration ;
— chargez chacun des 3 objets fournis avec cet énoncé et les ajouter à votre scène côte à côte selon l'axe $0x$, en les redimensionnant pour qu'ils tiennent sur une largeur de 2 unités.

Exercice 3 - Space Invaders

A partir de ce que vous avez vu dans ce TP et dans les TPs précédents, développez un jeu de type `Space Invaders`. Vous pouvez utiliser les objets contenus dans les fichiers `obj` fournis, récupérer vos propres objets ou créer vos propres objets. Prenez cependant garde à ce que ces objets ne soient pas trop complexes, afin de ne pas vous retrouver avec des temps d'affichage trop importants, qui nuiraient à l'interactivité.

On précise également que vous pouvez utiliser n'importe quel objet 3D pour représenter les aliens, et pas forcément un modèle qui représente la version originale...