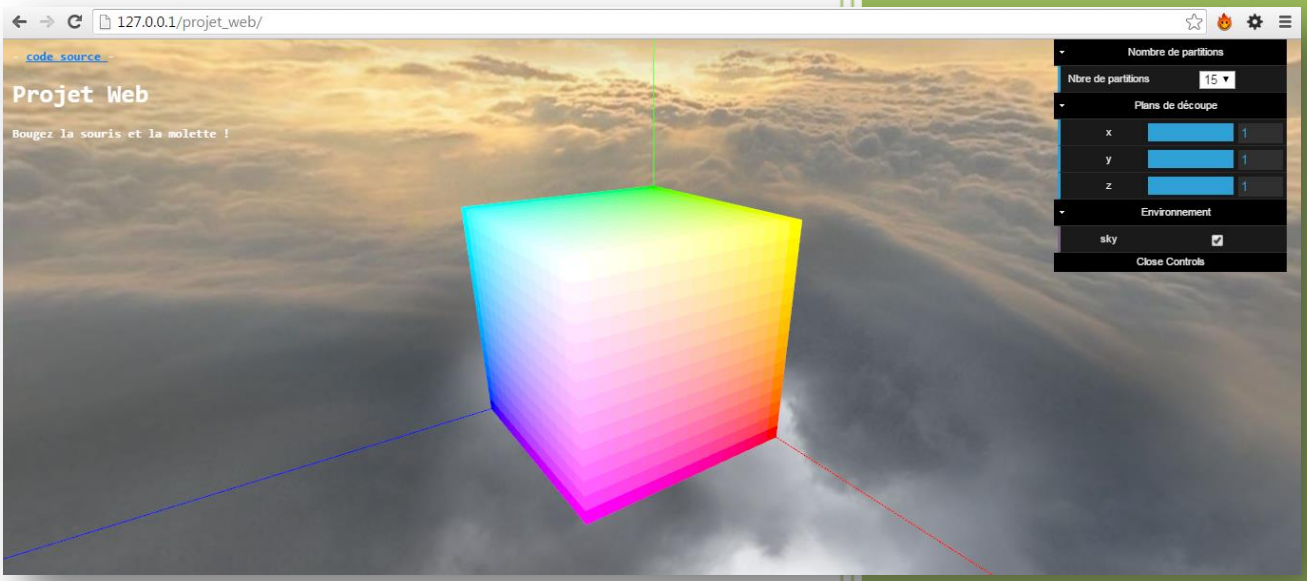
2016

Micro-projet Webmapping



Cédric MENUT

ING2

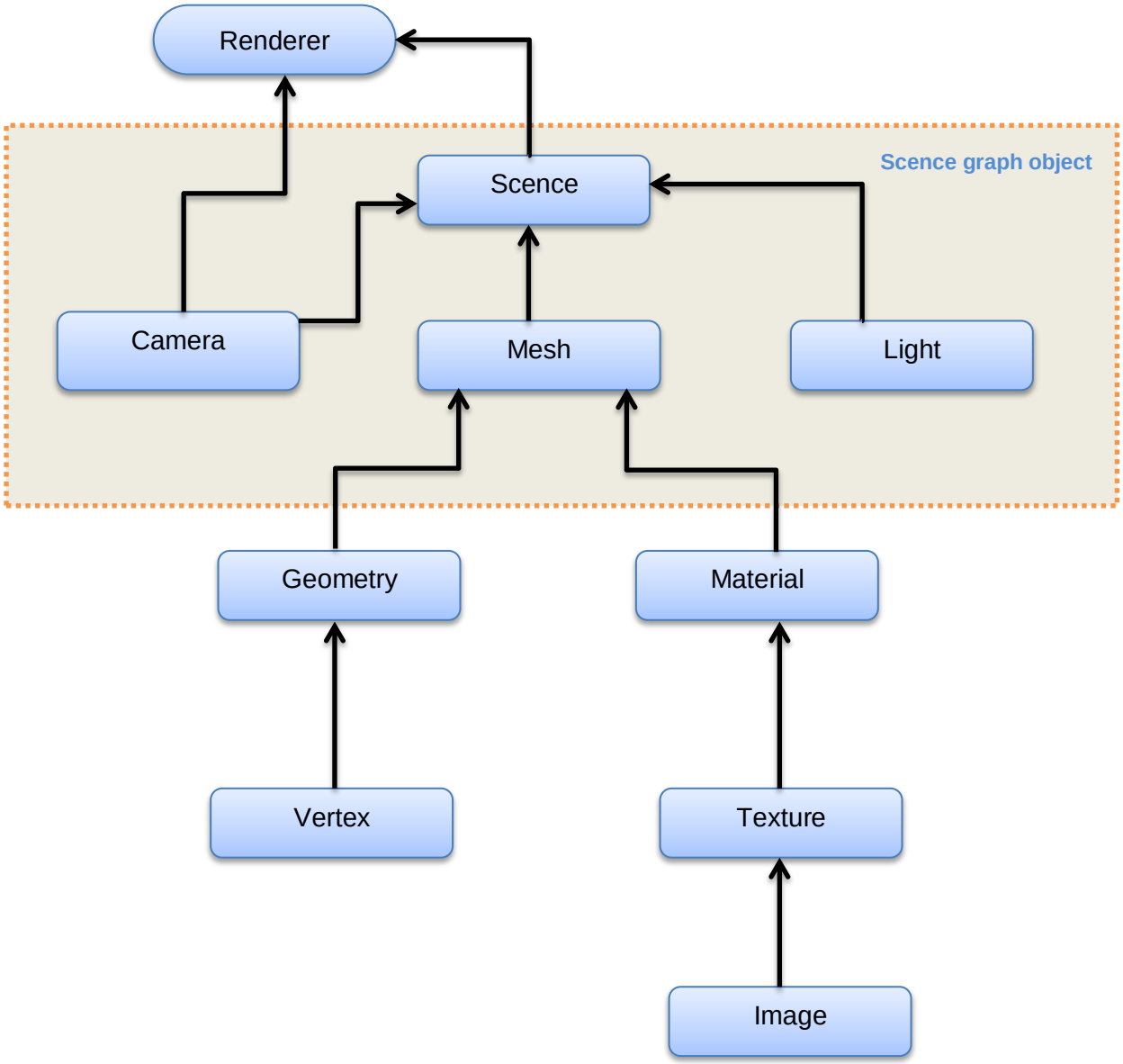
01/01/2016

Documentation programmeur

Introduction à la bibliothèque THREE.JS

L'ensemble de ce micro-projet a été réalisé en utilisant la bibliothèque JavaScript **Three.js**, accessible sur le dépôt GitHub suivant https://github.com/mrdoob/three.js/. Cette bibliothèque permet de manipuler les concepts WebGL de manière concise avec une approche très intuitive.

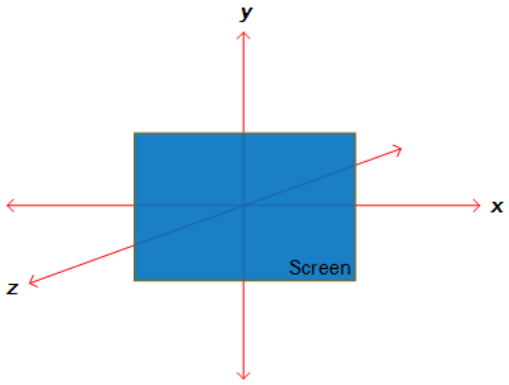
On peut résumer l'architecture d'utilisation de THREE.JS grâce au schéma suivant :



On remarque ainsi que pour travailler en WebGL nous avons besoin des éléments suivants :

1. Une scène
2. Un moteur de rendu
3. Une caméra
4. Un objet ou deux (lié à un rendu)

Pour définir la position de ces paramètres dans la scène, THREE.JS utilise un système de coordonnée dit « droitier » :



L'ensemble des sous-bibliothèques THREE.JS est appelé dans le fichier *index.html* et a été copié dans le dossier *js*.

Le fichier *cube.js* crée et appelle les fonctions nécessaires à la réalisation du micro-projet.

Script Cube.js

Premièrement, on vérifie que le navigateur supporte bien la technologie WebGL. Se référer à la documentation d'installation si besoin.

```
if (!Detector.webgl) Detector.addGetWebGLMessage();
```

Function init(){…}

Dans cette fonction on va créer l'architecture WebGL décrite sur le schéma ci-dessus.

On crée donc le moteur de rendu WebGL, la scène, et enfin les objets liés à cette scène.

Ces objets sont la lumière, la caméra et les cubes RVB.

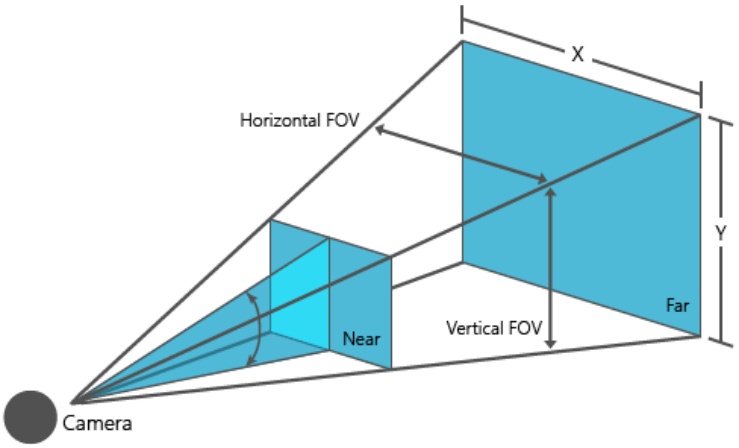
Intéressons-nous particulièrement à la création de l'objet caméra :

```
camera = new THREE.PerspectiveCamera( 60, canvasWidth / canvasHeight, 1,
10000 );
```

Les quatre paramètres de `PerspectiveCamera` sont les suivants :

- `60`
- `canvasWidth / canvasHeight`
- `1`
- `10000`

En gardant ces paramètres à l'esprit, examinons cette figure.



Source : https://msdn.microsoft.com/fr-fr/library/dn479430%28v=vs.85%29.aspx

- Le premier paramètre (`60`) définit le champ de vue vertical de la caméra en degrés (du bas jusqu'en haut de la vue). Il s'agit de l'étendue du monde observable qui est affichée à l'écran à n'importe quel moment donné. Le champ de vue horizontal est calculé d'après le champ de vue vertical.
- Le deuxième paramètre équivalent au quotient (`window.innerWidth / window.innerHeight`) définit les proportions de la caméra. Il est généralement conseillé de diviser la largeur de l'élément viewport par sa hauteur ; sinon, vous risquez d'obtenir une image qui apparaît écrasée.
- Le troisième paramètre (`1`) définit le plan du tronc de cône près de la caméra (« Near » sur la figure). Dans ce cas, le plan de tronc de cône près de la caméra coïncide pratiquement avec le plan *xy* (c'est-à-dire l'écran).
- Le dernier paramètre (`1000`) définit le plan du tronc de cône éloigné de la caméra (« Far » sur la figure). Dans ce cas, lorsqu'un objet se retrouve au-delà de ±1000 unités, il est considéré comme étant en dehors du monde Three.js visible et est coupé de la vue.

Par la suite, on crée l'objet lumière en lui affectant un type d'éclairement et le vecteur directeur de cet éclairement.

Enfin, on crée une scène secondaire qui permet à l'utilisateur de générer un nouvel environnement de type *Sky*.

Function onWindowResize (){…}

Cette fonction nous permet d'adapter la fenêtre du navigateur au rendu 3D lui-même. En effet, sans cette fonction, on observerait une perspective différente à chaque variation de taille de la fenêtre. En d'autres termes, cette fonction conserve la perspective par changement d'échelle.

Function setupGui (){…}

Cette fonction permet de créer une interface graphique pour répondre aux différentes fonctionnalités demandées. Ainsi, l'utilisateur peut faire varier le nombre de partition du cube, effectuer des coupes planes suivant les trois axes du repère et enfin de modifier l'environnement dans lequel il travail.

Cette fonction étant appelé au chargement de la page, c'est ici que l'on crée les cubes RVB.

Function render(){…}

Ici nous utilisons le moteur WebGL évoqué sur le schéma d'introduction. A chaque déplacement de la souris, on appelle cette fonction qui s'adapte aux différents paramètres sélectionnés par l'utilisateur dans l'interface graphique.

Il est à noter que les cubes sont déjà créés, et que l'on recrée à chaque déplacement de la souris uniquement l'affichage de ces cubes.

Les axes sont quant à eux supprimés et récréés afin d'être lié à l'angle inférieur du cube de côte 1 comme demandé dans les spécifications.

Function creation_cubeRVB (){…}

Dans cette fonction on distinguera la création des et leurs affichages. Pour répondre au besoin demandé, il a été créé n cube de cote 1/n ayant une couleur RVB en fonction de leurs coordonnées dans le repère local WebGL. Par définition du repère RVB, on obtient donc l'ensemble des couleurs du système colorimétrique RVB.

Function creation_axe (){…}

Ici on crée les axes X, Y, Z en prenant soin de lier l'origine de ce repère au cube de côte 1.

Function coupe_plane (){…}

Pour réaliser la fonctionnalité de coupe plane, on supprime uniquement l'affichage des cubes selon leurs coordonnées.

Perspectives

Une des solutions qui a été essayé mais qui n'a pas abouti est d'optimiser le code en affichant uniquement les cubes extérieurs. Ainsi, le contenu WebGL sera encore plus rapide et permettra une visualisions plus fluide et moins saccadée lors du mouvement générer par l'utilisateur.