INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
de BLAGNAC –TOULOUSE II
Département Informatique
1, place Georges Brassens – BP 73
31703 BLAGNAC Cedex

Snootlab
12 Boulevard d'Arcole 31000 Toulouse

Rapport du 1er Prototype

SciVisu

Interface de visualisation de données scientifique en WebGL

Projet Tutoré

Destinataires :
Nicolas GONZALEZ

Réalisé par :
Smain BARHOUMI
Maxence DELIOT
Kilian DESPORTES
Samuel JALADE

Année 2017/2018

Sommaire

1 Partie réseau (Broker MQTT)

1.1 Installation et configuration du broker mosquitto

MQTT (Message Queuing Telemetry Transport) est un protocole de messagerie qui fonctionne sur le principe de souscription / publication qui a été développé à la base pour simplifier la communication entre les machines. L'installation est possible sur quasiment toutes les plate-formes les plus communes (Windows, Linux, Mac OS X….), dans notre cas nous avons installé le broker sur le serveur nginx mis à notre disposition par le laboratoire de l'IUT de Blagnac.

Nous utilisons le broker open-source Mosquitto qui est le broker MQTT le plus utilisé.

Une fois l'installation du broker terminé (une commande), nous sommes passés à la configuration de celui-ci. Pour que l'on puisse communiquer avec un navigateur un port web socket doit être configuré. Pour cela nous avons créé un fichier broker.conf :

```
[root@ptutwebgl conf.d ]$ cat broker.conf
listener 1883

listener 1882
protocol websockets
```

On peut voir que le port 1882 est précisé comme étant un port web socket.

Pour que le broker soit tout le temps actif, nous l'avons exécuté en background :

```
CGroup: /
        ├─user.slice
        │ └─user-0.slice
        │   ├─session-1571.scope
        │   │ └─23367 mosquitto -c /etc/mosquitto/conf.d/broker.conf
```

1.2 Communication entre le site web et le broker

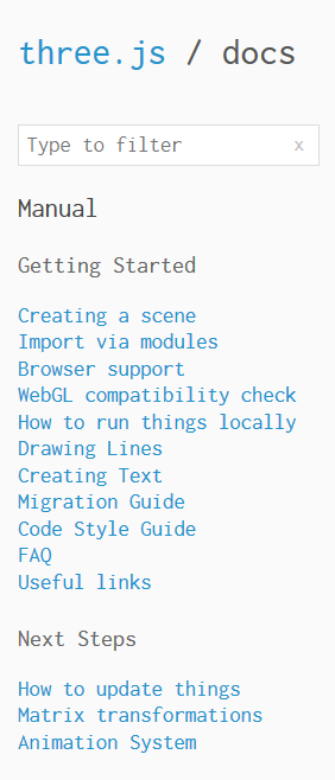
La librairie Paho MQTT nous permet de publier et de recevoir des messages par le protocole MQTT ainsi qu'établir une connexion en websocket avec le broker.

Nous avons dû prendre en main la librairie Paho MQTT pour pouvoir communiquer entre le broker et le site web. Pour communiquer, un programme publie des messages avec un nom de topic sur le serveur archibald.snootlab.info:2533(port forwarding(1882)) qui seront récupérés par le broker installé sur le serveur. Le broker regarde ensuite qui est abonné à ce topic et transmet les messages. Les messages sont récupérés grâce à un script JavaScript sur le site web utilisant la librairie Paho MQTT qui nous permet de nous abonner au topic souhaité et de choisir les actions à l'arrivée d'un message, nous pouvons donc par la suite traiter les données.

2 Partie Développement (Javascript – three.js)

2.1 Compréhension de la librairie three.js

Suivant les indications du client, nous nous sommes dirigés vers la librairie three.js pour pouvoir utiliser ses fonctions, three.js étant une surcouche plus accessible que du code webGl directement. Pour ce faire, nous avons découvert la librairie three.js en premier en parcourant sa documentation présente sur internet, ainsi que sa librairie d'exemple.



Menu de la documentation three.js reprenant toutes les fonctions disponibles.

Menu des exemples de three.js par catégorie

Nous avons également suivi plusieurs tutoriels sur internet pour avoir une structure de code de base sur laquelle nous appuyer pour ajouter d'autres options.

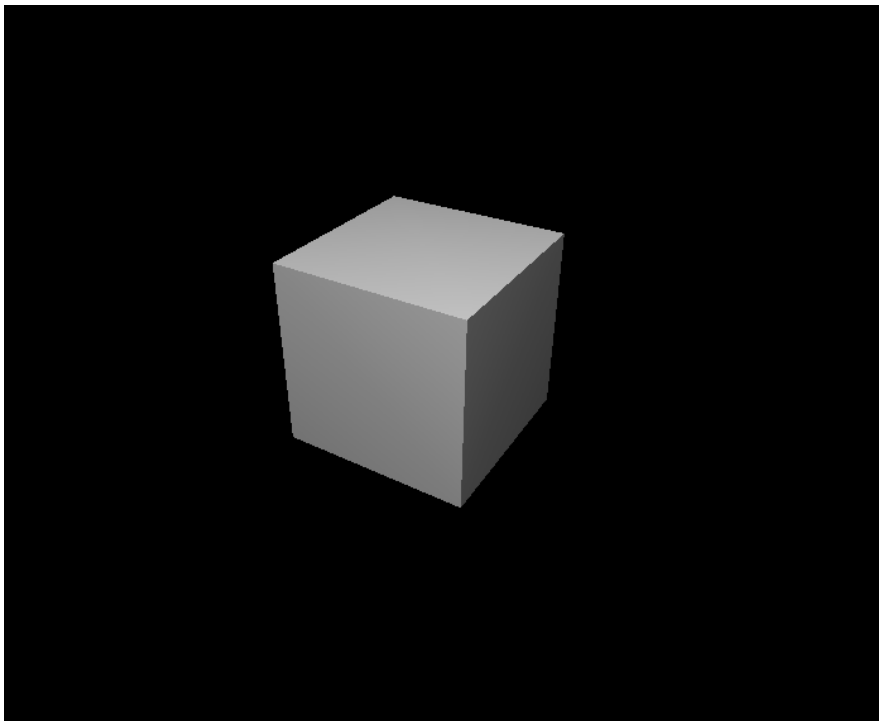
Nous avons rencontré certaines difficultés sur certains tutoriels à cause de la complexité du code pour des novices.

2.2 Implémentation d'exemple

A partir des extraits de code que nous avons récupérés sur les différents tutoriels et de la documentation fournissant les fonctions et leurs explications, nous avons pu commencer a créer nos propres codes grâce à three.js.

Les premiers codes consistaient à créer simplement des formes immobiles dans une scène et à pouvoir les visualiser, notamment un carré ou une sphère. Nous avons ensuite cherché à animer nos objets, puis à intégrer plusieurs objets dans une seule et même scène.

La plus grande difficulté que nous avons rencontrée est la modification des objets après leurs créations, notamment pour changer leurs couleurs, ou pour changer leurs textures. Nous avons finalement trouvé la solution dans la documentation après des recherches plus approfondies, étant donné que la solution se trouve sur un enchaînement de page de documentation ayant un lien entre elles (Pour changer la couleur, il faut passer par la documentation de MESH (l'objet), puis par celle de materials (matériaux de l'objet), puis celle de colors (un des attributs du matériaux) puis d'une fonction set.), enchaînement auquel nous ne sommes pas habitués, étant donné que nous ne manipulons que des documentations tel que Javadoc où toutes les méthodes sont ré-explicitées dans la page de l'objet en question.



2.3 Importation de fichier 3D

Nous nous sommes ensuite intéressés au fait d'importer des objets 3D, trouvés sur internet, sur notre scène et aux méthodes « Loader » de la documentation three.js, et nous avons dû faire des essais par nous même, ne trouvant pas d'exemple sur internet.

Nous avons ici encore rencontré des difficultés, ne connaissant pas les objets 3D *.obj* (nous nous sommes fondé sur ce format dans un premier temps), nous ne savions pas qu'un fichier associé (format *.mtl*) était nécessaire pour la représentation de notre objet 3D sur notre scène.

Une fois le problème connu, nous avons pu représenter des objets avec un *.mtl* de base créant une texture grise simple, et un autre problème fut soulevé, celui des textures complexes se fondant sur des fichiers images (*.jpg*, *.png*) associés au fichier *.mtl*, problème encore non réglé actuellement.

3 Site web (Framework Bootstrap)

3.1 Organisation du serveur web nginx

Pour commencer à configurer le serveur web fourni par le client, nous avons analysé le contenu de la machine virtuelle fournie pour savoir où étaient stockés les fichiers pour le site web.

Une fois ces fichiers trouvés, nous avons pu commencer à modifier le site web et organiser les fichiers sources .html et .js dans la machine virtuelle. Nous avons utilisé le gestionnaire de version gitHub du projet pour pouvoir organiser nos fichiers, en créant par exemple une seconde branche (en plus de la branche de base « master »), la branche « siteweb » où sont stockés nos fichiers pour le site web.

3.2 Association du broker et du javascript (three.js)

Une fois le broker mosquitto configuré et la librairie three.js prise en main, nous avons pu commencer à créer des programmes qui, sur le site web (et donc passant par la machine virtuelle), pouvaient interagir avec le broker et donc modifier le visuel de la scène webGl.

Nous avons rencontré des problèmes dans le code à cause de variables non globale, les rendant impossible a modifier après la réception d'une trame MQTT contenant un message. Une fois les problèmes de variables propres aux méthodes réglées et les fonctions de modification d'objet trouvés, nous avons pu par exemple, envoyer une trame MQTT contenant le message « c,red », qui sera lu dans une boucle if et qui servira a modifier la couleur (« c ») d'un objet en rouge (« red »).

3.3 Utilisation du framework bootstrap

Bootstrap est un framework CSS, mais pas seulement, puisqu'il embarque également des composants HTML et JavaScript. Il comporte un système de grille simple et efficace pour mettre en ordre l'aspect visuel d'une page web. Il apporte du style pour les boutons, les formulaires, la navigation…

Nous avons utilisé bootstrap pour créer le site web, le style est adapté à la charte graphique de snootlab et est entièrement responsive notamment sur smartphone.