Université de Montpellier
Cursus Master en Ingénierie informatique, L3

Lucilia — Modélisation 3D d'une scène photographiée par un drone

Construction d'une application de visualisation interactive de modèles tridimensionnels

---

Rapport de projet

Étudiants :
Maëlle Beuret, Rémi Cérès

Encadrante :
M^me^ Nancy Rodriguez

Date de rédaction :
14 mai 2018

Soutenu le :
17 mai 2018

Nous tenons à remercier notre encadrante Mme Nancy Rodriguez pour son accompagnement, sa bienveillance et ses conseils précieux tout au long de ce projet qui nous ont permis de le porter à son état actuel.

Table des matières

Chapitre 1

Introduction

1.1 Contexte général du projet : modélisation 3D d'une scène photographiée par un drone

Dans le cadre du projet annuel du L3 Cursus Master en Ingénierie (CMI) informatique, nous nous sommes fixés pour objectif la reconstruction d'un modèle en trois dimensions d'une scène à partir de photographies prises par un drone, et la réalisation d'un rendu dynamique de ce modèle.

Un modèle en trois dimensions d'une scène est une représentation numérique de son espace à un moment précis. À l'inverse des photographies, qui ne sont qu'une projection des objets visibles dans une scène depuis un point de vue — sans notion de profondeur — les modèles 3D conservent des informations sur la disposition spatiale des éléments de cette scène. Ces informations peuvent ensuite être analysées et manipulées par des outils numériques.

De nombreuses situations réelles peuvent tirer parti de ces informations. Par exemple, ils peuvent se révéler cruciaux pour l'analyse *a posteriori* d'un accident de voiture, servir un intérêt plus historique en permettant de conserver une copie numérique d'un objet ou d'un bâtiment fragile du patrimoine (Remondino et Stylianidis, 2016, p. 11), comme dans la figure 1.1, ou encore permettre de découvrir et de se repérer dans une ville, comme avec le projet Google Earth [1] présenté dans la figure 1.2.

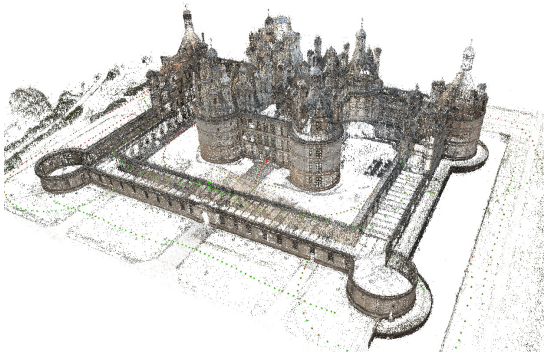


Figure 1.1 – **Nuage de points issu de la numérisation du château de Chambord** réalisée par des étudiants de l'École Nationale des Sciences Géographiques [2].

Figure 1.2 – **Rendu d'un modèle du cœur de Montpellier issu du logiciel Google Earth,** avec notamment le Carré Saint Anne, les Jardins du Peyrou et l'Arc de Triomphe.

1. Google Earth : https://www.google.com/earth
2. Détails sur le projet : http://micmac.ensg.eu/index.php/Presentation

Nous avons baptisé notre projet « Lucilia », de l'espèce d'insecte diptère *Lucilia sericata,* en référence au bruit produit par les drones lors de leur vol, qui se rapproche de celui d'une mouche.

Pour mener à bien ce projet conséquent, qui s'est déroulé sur la période d'octobre 2017 à avril 2018, nous avons constitué trois équipes de deux étudiants affectées chacune à une partie du processus de modélisation et de rendu d'une scène, illustré dans la figure 1.3.

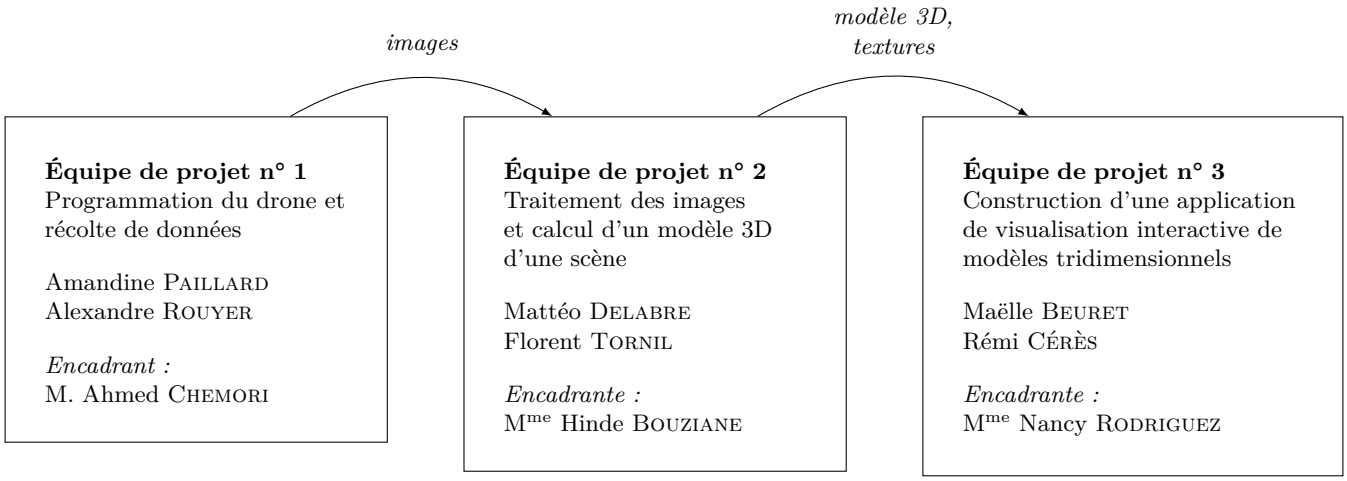


Figure 1.3 – **Positionnement des trois projets et de leur équipe dans le processus de modélisation et de rendu d'une scène.** Cette figure montre notamment les dépendances entre équipes et les données échangées de l'une à l'autre.

1.2 Objectif et cahier des charges du projet

Ce rapport se concentre sur le troisième projet intitulé «construction d'une application de visualisation interactive de modèles tridimensionnels». L'objectif est de mettre en place une application web permettant d'afficher et de présenter les modèles tridimensionnels construits par les groupes précédent. Cette application que avons nommée Lucilia Web App est l'interface entre l'ensemble du projet et l'utilisateur, elle se doit d'être la plus simple et intuitive possible.

Au début du projet, nous avons défini un cahier des charges précis afin d'établir les fonctionnalités et les contraintes que notre application doit respecter.

Application web : l'application s'exécute dans un navigateur indépendamment du système d'exploitation utilisé.

Lister les modèles disponibles : Lucilia Web App propose un menu listant les modèles disponibles, leur titre, une courte description et un aperçu du modèle.

Navigation au sein du modèle : il est possible de modifier le point de vue à partir duquel le rendu est effectué. L'utilisateur peut ainsi se déplacer au sein du modèle à l'aide de son clavier, de la souris ou d'un écran tactile.

Personnalisation de la lumière : au cours du rendu d'un modèle, les paramètres d'éclairage peuvent être modifiés en direct. L'utilisateur peut choisir différents types d'éclairages, personnaliser la couleur, l'intensité et modifier la position des sources de lumière.

Réalisation de mesures : l'utilisateur a la possibilité d'effectuer des mesures de distance, de surface et de volume directement sur le modèle.

Interface graphique simple : l'application possède une interface graphique simple et intuitive.

Dans un premier temps nous détaillerons les méthodes de travail et les outils que nous avons adoptés afin d'organiser ce projet. Ensuite, nous présenterons l'état de l'art que nous avons réalisé. Par la suite, nous décrirons la modélisation que nous avons conçue pour l'application, puis nous exposerons les différentes technologies et méthodes choisies pour son implémentation. Enfin, nous ferons un point sur l'avancement, les difficultés que nous avons rencontrées et les perspectives pour ce projet.

Chapitre 2

Organisation du projet

2.1 Méthode et organisation du travail

Le développement de Lucilia Web App a nécessité la découverte et l'apprentissage de nombreuses technologies que nous n'avions jamais utilisées auparavant. Nous avons donc choisi de travailler un maximum de temps ensemble afin que nous puissions profiter tous deux de ces nombreux apprentissages. Pour cela, nous avons adopté une technique de développement en binôme (en anglais *pair programing*). Cette technique consiste à travailler à deux sur un même poste. Un conducteur qui rédige le code et un observateur qui le relit, le vérifie et fait des suggestions au fur et à mesure qu'il est produit. Une étude de l'IEEE Computer Society (McDowell, Werner, Bullock et Fernald, 2003) montre que les étudiants utilisant cette méthode de développement produisent un travail de meilleure qualité.

Tout au long du projet, nous avons maintenu à jour un tableau de bord. Celui-ci nous permettait de faire le point sur l'avancée du projet, les difficultés que nous rencontrions ainsi que les différents schémas de modélisation. Ce tableau de bord fut également un outil de communication et d'échange important au sein de l'équipe et avec notre encadrante Mme Nancy Rodriguez.

Les réunions régulières avec notre encadrante de projet nous ont permis de bénéficier de son aide et de ses conseils sur les difficultés que nous rencontrions lors du développement. Suite à chaque réunion, nous rédigions un compte-rendu afin de faire un bilan et résumer les points importants abordés.

2.2 Répartition du travail dans le temps

Le développement de Lucilia Web App s'est déroulé en trois périodes. La première, de Septembre à mi-Octobre 2017, a permis la rédaction du sujet et le définition du cahier des charges.

La deuxième, d'Octobre 2017 à mi-Janvier 2018, a été consacrée à la recherche bibliographique. Cette recherche nous a permis de découvrir, d'étudier et de choisir les différentes méthodes et technologies à utiliser pour la réalisation de ce projet.

Enfin, la troisième période, de mi-Janvier 2018 à Avril 2018, a été consacrée à la modélisation de l'architecture de l'application, puis au développement du projet.

Cette étape s'est elle-même divisée en plusieurs sous-étapes. Nous avons commencé par une période de modélisation durant laquelle nous avons étudié différentes possibilités et réalisé plusieurs schémas, en parallèle d'une étape de développement du rendu tridimensionnel. Une fois celles-ci terminées, nous avons entamé l'implémentation de l'interface.

L'annexe A détaille cette répartition sous forme d'un diagramme de Gantt.

2.3 Outils de travail collaboratif

Tout au long du projet, nous avons utilisé différents outils afin de communiquer et pouvoir collaborer efficacement.

Tout d'abord, pour collaborer avec l'ensemble des membre du groupe, nous avons créé un dépôt git [1] commun aux trois binômes hébergé sur le site GitLab.com [2].

Nous avons également créé un groupe sur Telegram [3], une application de messagerie, afin de communiquer les informations importantes concernant le projet. Cela nous permettait notamment de discuter de nos disponibilités pour les réunions, ou bien solliciter l'aide des autres membres de projet lorsque nous devions nous entendre sur certains points entre les équipes.

Concernant la communication et la collaboration au sein de notre binôme, nous avons également utilisé Telegram pour la communication écrite, ainsi que Framatalk [4] pour la communication orale et le partage d'écran. Afin de travailler ensemble en temps réel sur les mêmes fichiers, nous avons utilisé le *plugin* Teletype [5] pour l'éditeur de texte Atom [6].

1. Git : https://git-scm.com/
2. GitLab : https://gitlab.com/ftornil/lucilia
3. Telegram : https://telegram.org/
4. Framatalk : https://framatalk.org/
5. Teletype : https://teletype.atom.io/
6. Atom : https://atom.io/

Chapitre 3

Etat de l'art

La première partie du projet fut consacrée à la réalisation d'un état de l'art. Nous avions peu de connaissances dans le domaine du rendu tridimensionnel, des interfaces graphiques et les technologies du web. Ce travail avait pour objectif de nous faire découvrir les différents concepts nécessaires pour la réalisation de Lucilia Web App.

Notre état de l'art se concentre donc sur la présentation des concepts ainsi que l'étude puis la comparaison des outils permettant de réaliser un rendu tridimensionnel et une interface graphique sur le web.

Nous avons décidé dans ce rapport de présenter seulement en détail les outils que nous avons utilisés lors du développement du projet. Notre état de l'art complet détaillant l'ensemble des outils étudiés constituait notre rapport de mi-parcours.

3.1 Méthodes et outils pour le rendu tridimensionnel sur le web

3.1.1 Présentation et définition du concept de rendu

Le rendu est un processus qui, à partir du modèle 3D d'une scène et d'un point de vue, calcule l'image 2D représentant la vue de la scène qu'aurait un observateur en ce point. Il a souvent pour finalité l'affichage sur un écran (Guillaume, 2012).

Le calcul se fait en prenant en compte les sources de lumière ainsi que différents phénomènes physiques. La figure 3.1 est un exemple de rendu du modèle de théière de l'Utah.

Quelques définitions

Modèle 3D : un modèle tridimensionnel (3D) est une représentation d'un corps physique à travers une collection de points dans un espace 3D, reliés par diverses entités géométriques telles que des triangles, des lignes, des courbes, etc..

Scène : ensemble de modèles tridimentionnels.

Caméra : point de vue à partir duquel le rendu de la scène est calculé.

1. Wikipédia : https://upload.wikimedia.org/wikipedia/commons/5/5f/Utah_teapot_simple_2.png CC-BY-SA 2.0).

Figure 3.1 – **Exemple de rendu moderne** du modèle de théière de l'Utah [1]conçu par Martin Newell en 1975.

3.1.2 Méthode de rendu : la rastérisation

Il existe plusieurs méthodes mathématiques de rendu d'un modèle 3D : les plus communes sont la rastérisation et le lancer de rayon (Van der Ploeg, 2013). Bien que nous ayons étudié les deux, nous ne présenterons ici que la rastérisation, qui est la méthode que nous utilisons pour notre projet.

La rastérisation consiste à projeter chaque objet de la scène 3D sur un écran 2D. Elle se déroule en plusieurs étapes schématisées sur la figure 3.2. Dans un premier temps les coordonnées des points de chaque objet sont projetées géométriquement dans le plan de l'écran. On obtient ainsi une image vectorielle en deux dimensions de la scène. Dans un second temps, l'image vectorielle obtenue est convertie en une image matricielle. C'est au cours de cette étape que l'on détermine les pixels qui doivent apparaître, leur intensité de lumière et leur couleur (Iris, 2017).

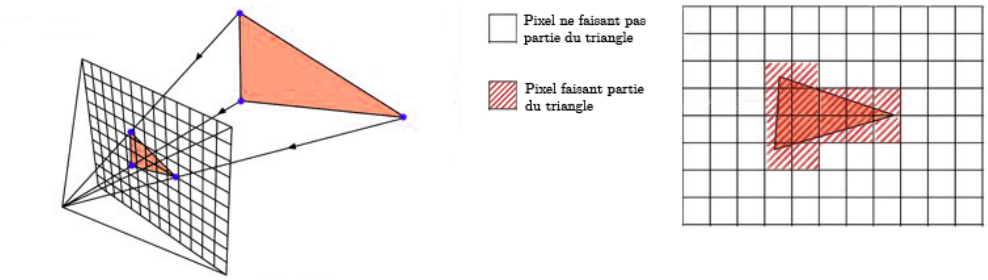


Figure 3.2 – Schéma du principe de la rastérisation [3]. La partie droite illustre la première étape de projection et la partie gauche représente la seconde étape de conversion de l'image vectorielle en une image matricielle.

Dans le cas où la projection de l'objet sort de l'écran, il n'est pas rendu. La profondeur, c'est-à-dire la distance par rapport à l'écran, est calculée afin de ne pas effectuer le rendu d'un pixel si

3. Scratchapixel : https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

un autre objet plus proche de l'écran utilise ce même pixel (Rey, 2014).

Les sommets des objets sont calculés grâce à des opérations matricielles adaptées aux fonctionnalités des matériels graphiques. La simplicité de ces calculs permet de réaliser des rendus en temps réel.

3.1.3 Outils de rendu du web

Il existe à ce jour un standard permettant de faire un rendu tridimensionnel sur le web. Il s'agit de Web Graphics Library (WebGL). Avant l'apparition de ce standard en mars 2011, (Point, 2015) il existait des solutions telles que Flash ou Java3D permettant d'y parvenir, mais tendent aujourd'hui à devenir obsolètes.

Nous avons donc décidé lors de notre état de l'art de nous concentrer sur WebGL et ses couches d'abstraction les plus utilisées.

Interface de programmation 3D pour les applications web : WebGL

WebGL est une API JavaScript *open source* développée par Khronos permettant l'affichage de graphismes 2D et 3D dans n'importe quel navigateur web compatible (MDN web docs, 2017). Elle est promue par un consortium qui regroupe à la fois des fabricants de puces (Nvidia, Intel, ARM, AMD) et des éditeurs de navigateurs (Mozilla, Opera et Chrome).

Cette API permet d'utiliser le standard OpenGL ES (*Open Graphics Library for Embedded Systems*) au sein d'une page web. OpenGL est une interface de programmation bas niveau implémentée au dessus des pilotes graphiques. Elle permet de réaliser des rendus 2D et 3D en exploitant l'accélération matérielle de la carte graphique (Bourry, 2013).

OpenGL ES est une version portable de OpenGL prévue pour fonctionner sur des appareils mobiles. WebGL a choisi d'utiliser OpenGL ES pour son large support au sein des architectures embarquées telles que les tablettes ou les téléphones.

A l'aide de WebGl, le navigateur peut utiliser les ressources de la carte graphique pour calculer le rendu. La figure 3.3 résume les différentes étapes de la communication entre le navigateur et la carte graphique.

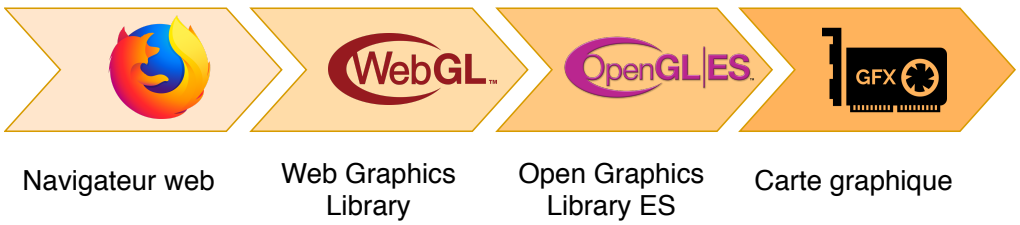


Figure 3.3 – Schéma des étapes de la communication entre le navigateur et la carte graphique (icône réalisé par Freepik [5]).

La méthode de rendu utilisée par WebGL est la rastérisation. Pour calculer le rendu, WebGL utilise le même *Shading Language* que OpenGL, GLSL ES (*OpenGL Shading Language Embedded Systems*).

4. FlatIcon : https://www.flaticon.com/

Un *Shading Language* est un langage adapté a la programmation de *shaders*. Les *shaders* sont de petits programmes écrits en langage haut niveau, qui définissent comment les pixels d'un objet tridimensionnel sont dessinés. Ils s'exécutent sur la carte graphique. Un *shader* est composé principalement de deux parties : un *vertex shader* et un *pixel shader* (également appelé *fragment shader*). Le premier transforme les coordonnées de l'objet en espace d'affichage bidimensionnel. Le second génère les couleurs finales de chaque pixel en se basant sur la texture et le matériau de l'objet, la couleur ainsi que la lumière qui y est projetée (Tony, 2012).

WebGL comporte de nombreux avantages. Tout d'abord, comme dit précédemment, il s'agit d'un standard supporté nativement par les principaux navigateurs (Chrome, Firefox, Safari, Opera)(Alexis, Ng Yik, Piotr, Jeff et Craig, 2017). L'utilisateur peut donc utiliser cette technologie directement sans devoir installer un *plugin*. De plus, WebGL est également utilisable sur n'importe quelle plateforme, allant de l'ordinateur de bureau au téléphone (Tony, 2012), (Florian, 2018).

Cette API étant basé sur l'API de OpenGL, cela lui permet d'exploiter pleinement les ressources de la carte graphique du client pour le calcul du rendu.

WebGL est écrit en JavaScript, il est donc possible d'interagir directement avec les autres éléments du document HTML. Cela permet également d'exploiter les nombreuses bibliothèques disponibles pour ce langage de programmation.

Couche d'abstraction de WebGL : Three.js

Programmer directement avec WebGL est assez complexe, cela nécessite une bonne connaissance de WebGL ainsi que du *Shading Language* (GLSL) (Dirksen, 2014). Il existe plusieurs couches d'abstraction simplifiant sont utilisation au travers d'API JavaScript haut niveau. Ces couches d'abstraction permettent de créer facilement des scènes 3D de bonne qualité sans avoir à connaître WebGL dans ses détails. Après en avoir analysé plusieurs, notament Three.js ou Babylone.js, nous avons décidé d'utiliser Three.js.

Three.js est une couche d'abstraction *open source* créée par Ricardo Cabello (*CV Ricardo Cabello* 2018) et lancée en 2010. Elle fut dans un premier temps écrite en ActionScript puis traduite en JavaScript. Elle permet de créer des scènes tridimensionnelles directement dans le navigateur en utilisant WebGL.

Three a pour objectif de simplifier et rendre plus accessible la création de graphismes 3D sur le web. Cette couche d'abstraction orientée objet réalise des abstractions des différents concepts d'un rendu. Elle définit ainsi les concepts de scène, de caméra, de lumière, etc.

Three simplifie donc la création de scènes tridimensionnelles complexes. Elle permet de réaliser plus facilement des animations et des mouvements d'objets. Cette couche d'abstraction apporte également une gestion de l'éclairage avec différents types de lumière prédéfinis.

Enfin, l'application de textures et le chargement d'objets depuis un logiciel de modélisation sont également facilités. Il est notamment possible à l'aide de Three de charger un fichier au format JSON (*Three.js docs* 2017).

Cette couche d'abstraction permet aussi de simplifier les interactions avec le rendu en apportant par exemple la possibilité de relever les coordonnées du curseur de la souris (Tony, 2012).

3.2 Interface graphique de sélection et de manipulation du rendu

3.2.1 Principe d'une interface graphique de sélection et de manipulation du rendu

Une interface graphique est un dispositif d'interaction homme-machine. L'objectif est de permettre un fonctionnement et un contrôle efficace de la machine par l'homme. La figure 3.4 est un exemple d'interface graphique.

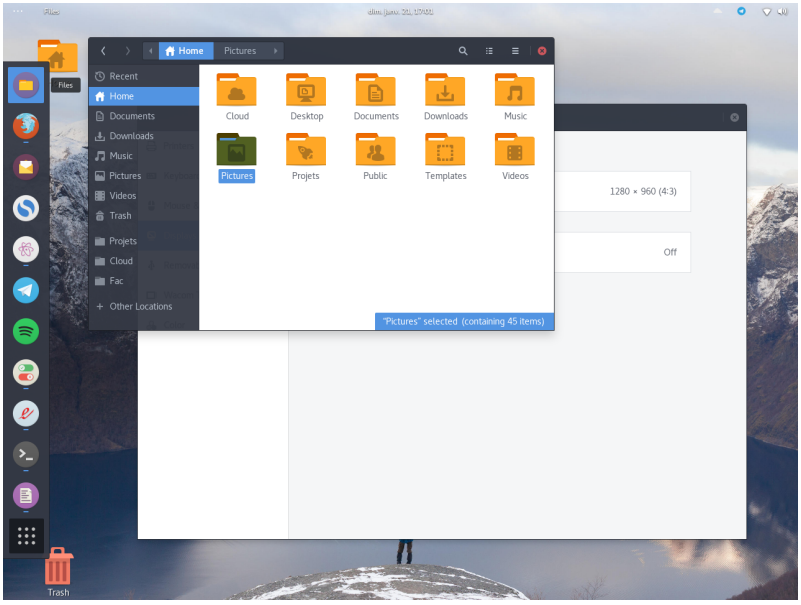


Figure 3.4 – **Interface graphique utilisateur qui suit la métaphore du bureau** Gnome 3.26 [6]

Une interface graphique est manipulée au travers d'objets ou *widgets* dessinés sous forme de pictogrammes. Ces objets peuvent par exemple être des fenêtres ou des boutons, comme illustré par la figure 3.5. L'utilisateur manipule ces objets en s'inspirant des interactions physiques qu'il aurait avec l'objet qu'il représente. Pour cela, il utilise un dispositif de pointage, généralement une souris.

3.2.2 Outil permettant de créer une interface : Vue.js

Il existe différents *frameworks* et bibliothèques JavaScript permettant de faciliter la création d'interfaces graphiques. Ces outils permettent de faciliter l'organisation du code et de séparer les données et leur affichage. Ainsi, le code obtenu est plus clair, plus compréhensible et plus simple à entretenir. Nous avons étudié Vue.js, React.js ainsi que Angular.js et finalement choisi d'utiliser Vue.js.

5. Gnome : https://www.gnome.org/
6. Wikipédia : https://upload.wikimedia.org/wikipedia/commons/d/d5/Widgets.png

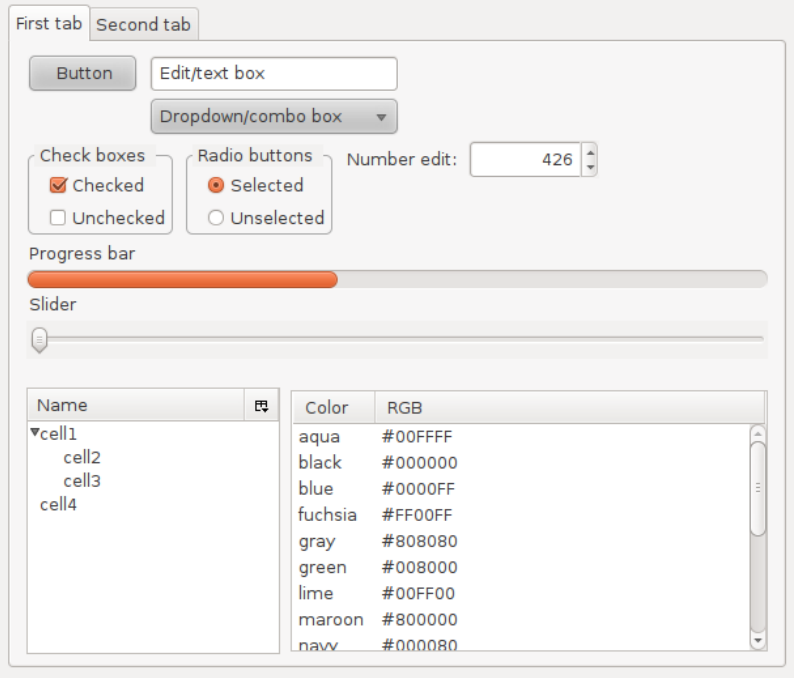


Figure 3.5 – **Exemple de widgets graphiques typiques**[6]

Vue.js est une bibliothèque JavaScript créée par Evan You[7] en 2014 distribuée sous licence MIT.

Cette bibliothèque utilise par défaut la liaison unidirectionnelle des données, c'est-à-dire que la vue est mise à jour automatiquement lorsque le modèle est modifié. Un modèle contient les données à afficher et une vue contient la présentation de l'interface graphique.

Mais Vue peut utiliser la liaison bidirectionnelle qui permet de mettre également à jour le modèle en fonction des modifications de la vue. Le but de cette bibliothèque est de combiner les avantages d'une liaison de données réactive avec une API la plus simple possible.

Vue permet d'organiser le code par composant. Un composant est un élément de l'interface graphique. Chaque composant est défini dans un fichier unique, le rendant ainsi indépendant et réutilisable lors d'un autre projet (Neuhaus, 2017). Cela permet également de modifier aisément un composant sans impacter les autres, évitant ainsi des effets de bord indésirables.

Nous avons choisi Vue principalement pour son API simple et sa documentation complète. En effet, disposant d'un temps relativement court pour notre projet, nous avons estimé plus judicieux de ne pas s'attarder longtemps sur l'apprentissage de nos outils.

Outre son utilisation simple, Vue possède également un avantage non négligeable : il est très léger, ce qui le rend particulièrement adapté pour des projets de petite taille. (Kyriakidis et Maniatis, 2016).

7. Evan You : http://evanyou.me/

Chapitre 4

Conception

Avant d'entamer l'étape d'implémentation, nous avons réalisé une étape de conception de notre application.

Dans un premier temps nous avons souhaité définir clairement le comportement de notre application. Nous avons commencé par réaliser un diagramme d'activité représenté par la figure 4.1. Ce diagramme modélise et décrit le comportement de notre application. Il met également en avant les interactions possibles de l'utilisateur avec Lucilia Web App.

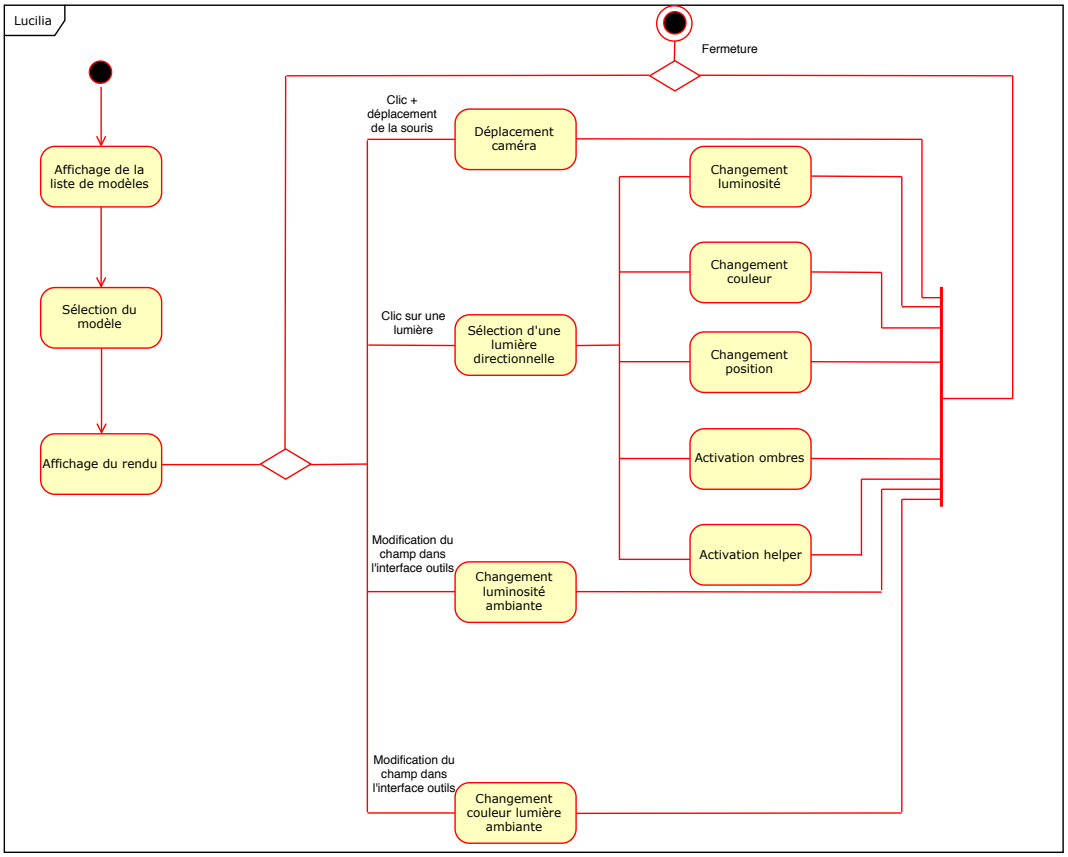


FIGURE 4.1 – Diagramme d'activité de l'application Lucilia.

Au lancement de l'application, celle-ci affiche les différents modèle disponibles. L'utilisateur sélectionne un modèle et le rendu de ce dernier s'affiche. Une fois le modèle sélectionné, l'utilisateur

peut réaliser plusieurs actions. Il peut se déplacer au sein du modèle, changer les propriétés des lumières directionnelles ou de la lumière ambiante. L'utilisateur peut réaliser les actions citées ci-dessus autant de fois qu'il le souhaite jusqu'à la fermeture de l'application ou au retour au menu de sélection du modèle.

Ce diagramme d'activité fut la première étape de notre modélisation. Maintenant que nous avons clairement défini les comportements de Lucilia Web App, nous pouvons commencer la modélisation de la partie rendu tridimensionnel ainsi que de l'interface graphique.

4.1 Définition d'une interface de programmation pour le rendu tridimensionnel

Dans le but de découvrir la bibliothèque Three, nous avons réalisé de nombreux tests. Nous avions par exemple un test d'utilisation des lumières et des ombres, ou encore un test permettant de charger un modèle.

Une fois familiarisés avec Three, nous avons été confrontés à un problème. En effet, les codes que nous avions produit était intégralement contenus dans un seul fichier, le code était peu clair et difficile à comprendre.

Bien que les tests réalisés soient parfaitement fonctionnels, ils étaient très peu dynamiques, c'est-à-dire que nous n'avions pas la possibilité de modifier les paramètres du rendu sans modifier le code en lui-même. Il n'était par exemple pas possible de rajouter une lumière directionnelle ou d'en modifier sa couleur.

Pour pallier a ce problème, nous avons réalisé le diagramme d'objets sous forme de pseudo-classes présenté par la figure 4.2.

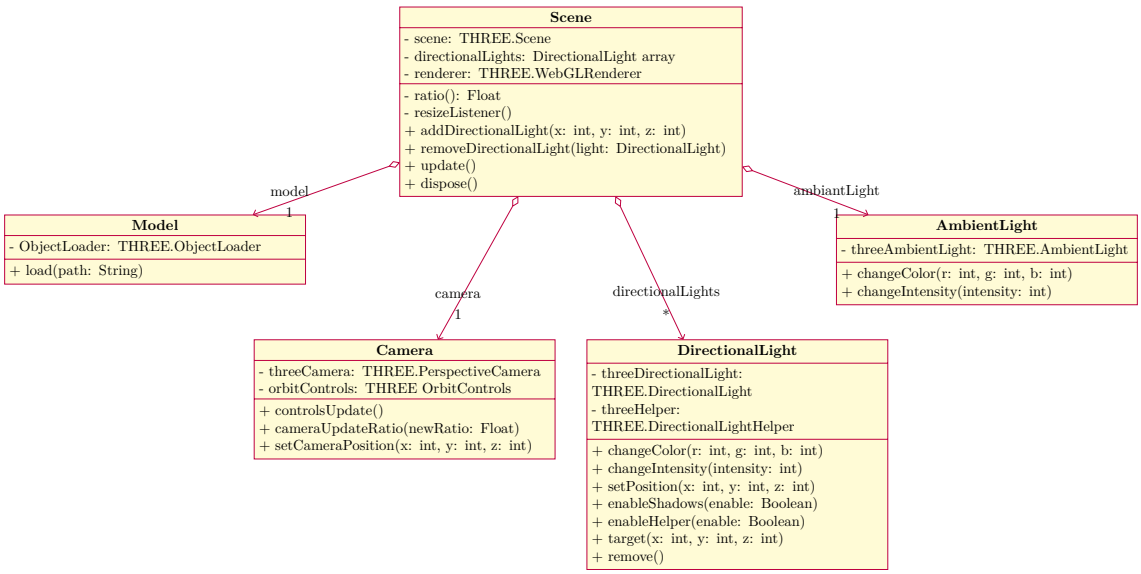


Figure 4.2 – Diagramme de classes de la partie rendu de l'application.

Nous avons créé cinq pseudo-classes :

Model est responsable de la gestion du modèle. Elle permet d'en charger et de modifier leurs propriété t-elle que la position par exemple.

Camera s'occupe de la gestion de la caméra. C'est au travers de cette classe que l'on peut modifier les propriétés de la caméra, activer le helper ou la contrôler.

DirectionalLight représente une lumière directionnelle.

AmbiantLight est en charge de la gestion de la lumière ambiante et de ses propriétés.

Scene est la classe principale. C'est elle qui est responsable de la boucle de rendu. C'est également Scene qui contient une instance de Model, de Camera, d'AmbiantLight et plusieurs de DirectionalLight.

Cette modélisation nous permet de clairement séparer les responsabilités entre les différentes classes, maitriser l'accès aux propriétés et aux objets de three.js, d'avoir un code plus clair et enfin un rendu dynamique.

Cependant, nous avons dû modifier cette modélisation par la suite car cela posait problème pour interagir avec le rendu depuis l'interface graphique.

4.2 Conception d'une interface graphique

Afin de concevoir et visualiser l'interface graphique de notre application, nous avons réalisé une maquette fonctionnelle représentée par la figure 4.3.

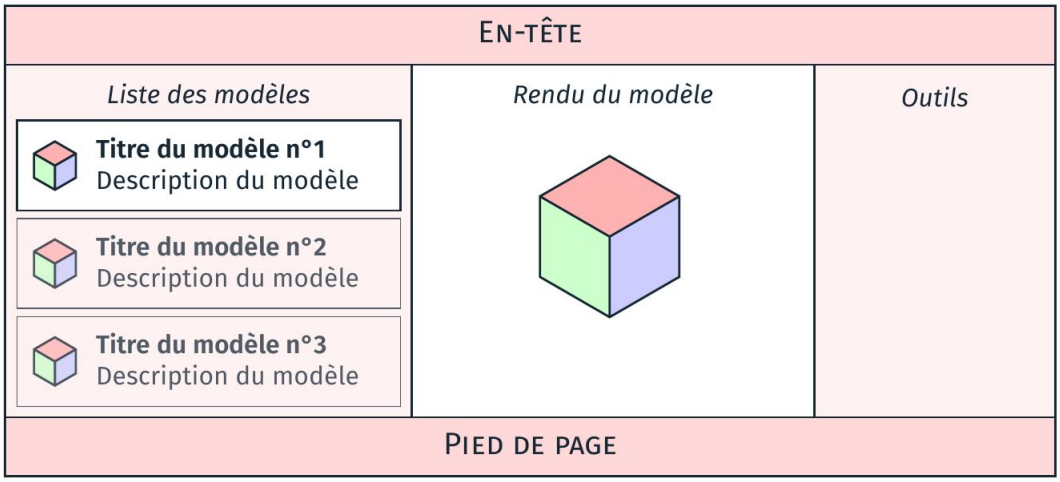


Figure 4.3 – Maquette fonctionnelle de l'application.

L'interface de notre application est constituée de trois principaux composants : l'en-tête, le pied de page et la partie centrale.

L'en-tête et le pied de page sont constitués d'un seul composant, respectivement en-tête et pied de page. Ils contiennent les informations générales de l'application tels que le nom, le logo, etc.

La partie centrale est constituée de trois composants.

Le composant liste permet de sélectionner un modèle parmi une liste de modèles disponibles sur le serveur. Chaque modèle dispose d'un nom ainsi que d'une courte description. Pour afficher

le rendu d'un modèle, il suffit de cliquer sur le nom dans la liste. Le rendu au centre affiche le rendu de la scène. Enfin, le composant outils à droite contient l'interface des outils permettant de manipuler le rendu.

L'interface des outils est composée de champs permettant de modifier des paramètres du rendu. Lors de l'ouverture d'un modèle, les paramètres généraux tels que la lumière ambiante peuvent être modifiés. L'utilisateur peut également choisir d'ajouter une lumière directionnelle à la scène en cliquant sur un bouton prévu à cet effet.

Afin de pouvoir modifier les paramètres d'une lumière directionnelle, il faut sélectionner celle-ci en cliquant dessus, et les paramètres disponibles pour cette lumière s'afficheront alors.

Dans le but de hiérarchiser et exposer clairement les inclusions et dépendances entre tous ces composants, nous les avons représentés sous forme d'arbre dans la figure 4.4.

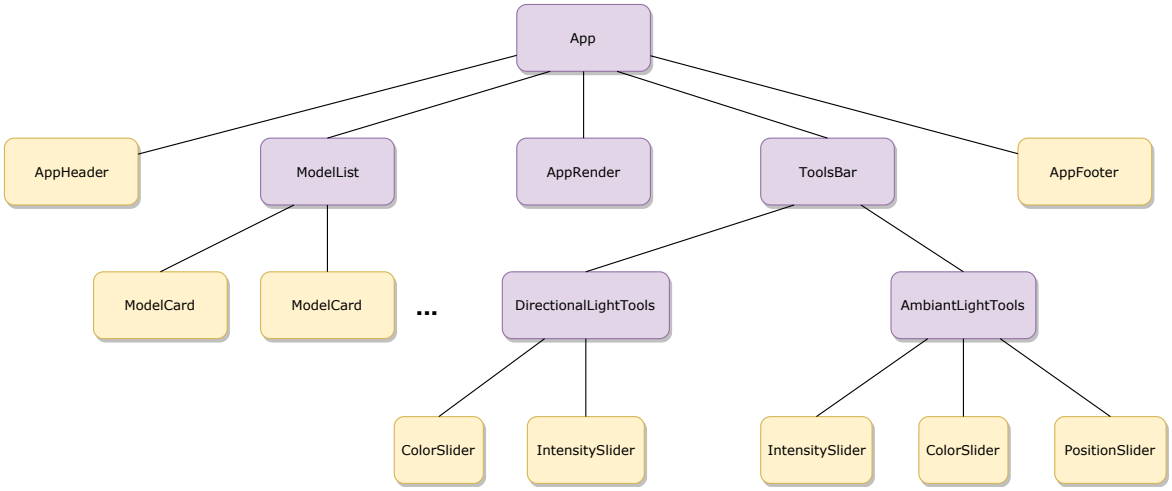


Figure 4.4 – **Arborescence des composants Vue de l'application.** En beige : les composants statiques, en violet : les composants dynamiques.

Nous avons défini que certains composants sont statiques (représentés en beige sur le schéma) et d'autres dynamiques (représentés en violet). Les composants statiques ne sont pas modifiés en fonction de l'état de l'application, alors que les composants dynamiques vont être modifiés en fonction des valeurs altérées par les composants enfants et le modèle.

Chapitre 5

Implémentation

Nous avons choisi de développer le projet Lucilia Web App en JavaScript. Ce langage permet de créer des applications web s'exécutant au sein d'un navigateur. De plus, le JavaScript est un langage multi-plateformes, il permet de porter facilement notre application sur plusieurs systèmes et également sur mobile.

Par ailleurs, le JavaScript est un langage pour lequel nous avons eu une initiation dans notre formation. Nous possédons donc des bases pour ce langage, et ce projet nous permet d'en approfondir nos connaissances. Enfin, ce langage jouit d'un communauté importante et très active, de nombreuses bibliothèques et une documentation très complète. Ces choix sont résumés dans la table 5.1.

5.1 Gestion des paquets et de dépendances

Suite à notre état de l'art, nous avons remarqué que Lucilia Web App nécessite l'utilisation de nombreux paquets tel que Three ou Vue. Nous avons donc choisi d'utiliser un gestionnaire de paquets. Notre choix s'est porté sur le gestionnaire de paquet officel de Node, npm (Node Package Manager).

Il s'agit de la plus grande base de données de paquets pour le langage JavaScript, avec approximativement trois milliards téléchargements par semaine et plus de 600 000 paquets (*Documentation officielle npm* 2018).

Npm est constitué de trois éléments :

Une base de données publique regroupant les paquets pour JavaScript ainsi que des informations les concernant.

Une interface en lignes de commande (en anglais *command line interface*, couramment abrégé CLI) qui permet au développeur d'interagir avec npm. Ce *CLI* permet notamment de rechercher, d'installer et/ou de supprimer des paquets disponibles dans la base de données.

Un site web [1] permettant d'explorer et de découvrir facilement des paquets.

Nous aurions pu choisir de ne pas utiliser de gestionnaire de paquets. Il est tout à fait possible de télécharger manuellement les bibliothèques et de les placer dans le bon répertoire. Cependant, la gestion de notre projet et de la liste de ses dépendances aurait été rapidement difficile et rendrait la collaboration et le partage de notre projet complexe.

L'utilisation de npm nous offre de nombreux avantages :

Il permet notamment une meilleure organisation du code du projet. En effet, il regroupe l'ensemble des paquets installés dans un dossier node modules, lui-même organisé en sous-répertoires

correspondant à chaque paquet. Ainsi le code propre à notre application est clairement séparé de ces dépendances.

Npm s'occupe de la gestion des dépendances, c'est-à-dire que lors de l'installation d'un paquet, npm recherche et installe automatiquement toute les dépendances nécessaires au bon fonctionnement de ce dernier. Sans l'aide d'un gestionnaire de paquets, cela représente un travail long et fastidieux, ce qui nous aurait retardés pour le développement du code.

Npm simplifie le partage et la collaboration sur le projet. L'ensemble des paquets nécessaires sont spécifiés dans le fichier *package.json* dont l'annexe B est un exemple. Ils peuvent être facilement installés à l'aide de la commande npm install exécutée à la racine du projet.

5.2 Obtenir la liste des modèles disponibles

Afin de pouvoir afficher la liste des modèles disponibles, notre application a besoin d'accéder au système de fichiers de l'utilisateur, plus particulièrement au répertoire "objects". Or, si l'application est exécutée directement dans un navigateur avec le protocole file, elle ne possède pas les autorisations nécessaires.

Pour pallier à ce problème, nous avons donc dû mettre en place un serveur. Le serveur peut accéder à son système de fichiers et peut donc récupérer la liste des modèles disponibles afin de la retourner au client.

Notre serveur a un fonctionnement simple et peut répondre à deux types de requêtes : si le client demande à avoir accès à la racine du site avec l'url "/", alors le serveur lui renvoie le fichier index.html, c'est à dire la page d'accueil de Lucilia Web App.

Si le client réalise une requête GET avec l'url "/modeles", alors le serveur lui renvoie la liste des modèles disponibles en format JSON. La figure 5.1 est un exemple de réponse du serveur sur l'url "/modules".

```
[
  "/home/user/Projects/projet_l3_cmi/projects/rendu/objects/tea.json",
  "/home/user/Projects/projet_l3_cmi/projects/rendu/objects/jupiter/jupiter.json",
  "/home/user/Projects/projet_l3_cmi/projects/rendu/objects/mario/mario.json"
]
```

Figure 5.1 – **Format de la réponse du serveur lorsque le client demande la liste des modèles disponibles.** Le serveur retourne un tableau de chemins des modèles au format JSON.

Ce serveur récupère sous format JSON la liste de modèles présents dans le dossier "objects" de

notre projet. Pour cela, nous avons implémenté l'algorithme 1.

Données : Un répertoire dir, une fonction de gestion de liste done
Résultat : La liste des fichiers du répertoire
res <— [] ;
si *le répertoire est vide* **alors**
  retourner *res* ;
fin
pour *chaque fichier dans le répertoire* **faire**
  si *f est un répertoire* **alors**
    walk(f, fonction de concaténation de listes) ;
  sinon
    ajouter f à res ;
  fin
fin

Algorithme 1 : Récupération de la liste de modèles

Nous avons choisi d'utiliser Node. En effet, Node permet de créer un serveur simplement en JavaScript, cela nous offre l'avantage d'utiliser le même langage pour l'ensemble du projet. De plus, nous utilisons déjà le gestionnaire de paquets officiel de Node, npm, pour la gestion des nos dépendances.

Pour implémenter le serveur nous avons utilisé le paquet Express. NodeJS permet de créer des serveurs, mais reste très bas niveau. Pour créer un serveur, il faut effectuer manuellement les vérifications des routes et réaliser de nombreux tests afin de sécuriser le serveur. Express est plus haut niveau, il simplifie la gestion des routes et automatise les tests basiques pour sécuriser le serveur. De plus, il fournit un ensemble de méthodes permettant de traiter facilement les requêtes HTTP.

5.3 Conception de pseudo-classe en JS

Afin d'implémenter les classes que nous avons conçues et présentées dans la section 4.1, nous avons utilisé le modèle présenté par la figure 5.3. Cette figure est un exemple simplifié d'une classe représentant un compte bancaire. Cette exemple permet de mettre en avant les principes d'implémentation utilisés. Le diagramme UML de compte bancaire est présenté dans la figure 5.2.

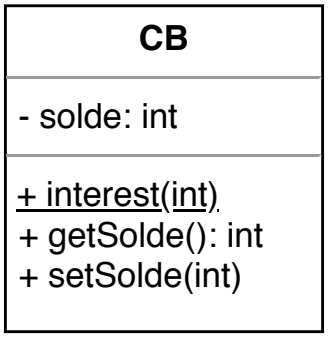


Figure 5.2 – Diagramme UML d'un compte bancaire

```
const CBPrototype = {
    interest(rate) {
        this.solde = this.solde * (1 + rate);
    }
};


const CB = (soldeInitial) => {

    let solde = soldeInitial;

    return Object.create(CBPrototype, {
        get solde() {
            return solde;
        },

        set solde(newSolde) {
            solde = Math.max(0, newSolde);
        },

    });
};

export default CB;
```

Figure 5.3 – Exemple de pseudo classe de compte bancaire en JavaScript

La classe CB représente un compte bancaire. Cette classe possède un constructeur qui permet de définir un solde initial. Elle contient également deux accesseurs. L'un permet d'obtenir le solde du compte et l'autre de le modifier. Enfin, CB possède une méthode de classe qui permet de calculer et d'ajouter les intérêts au compte.

La figure 5.4 montre un exemple d'utilisation de notre interface de programmation pour réaliser un rendu. Le résultat obtenu est illustré par l'image 5.5.

```
const scene = Scene(
  document.body,
  'objects/tea.json'
);

scene.addDirectionalLight(2, 2, 2);
scene.directionalLights[0].changeIntensity(3);
scene.directionalLights[0].changeColor(1, 0, 0);
scene.directionalLights[0].enableHelper(true);
```

Figure 5.4 – **Exemple d'utilisation de l'interface de programmation pour créer une scène.** Dans cet exemple, une scène est créée et charge le modèle tea.json. Une lumière directionnelle est ajoutée à la scène puis sa couleur et sa position sont modifiées. Enfin le Helper de la lumière est activé.

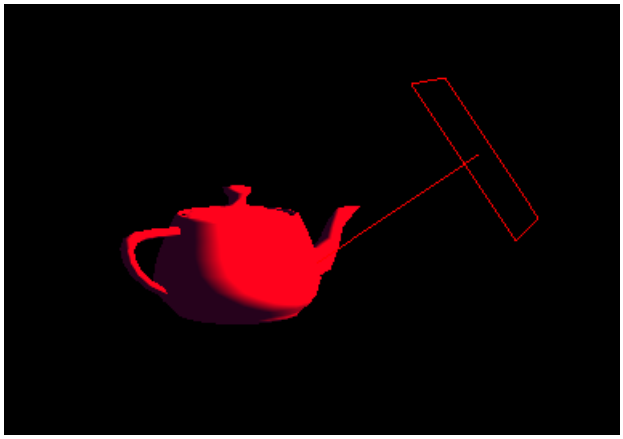


Figure 5.5 – Rendu obtenu a l'aide de notre interface de programmation.

5.4 Interactions entre Vue et Three

Le personnalisation du rendu via l'interface graphique nécessitait des interactions entre les composants des outils de Vue et le rendu effectué par Three.

Pour réaliser cette interaction, nous avons d'abord implémenté des chaînes d'événements remontant jusqu'au composants parents en commun. Nous nous sommes vite rendu compte que cette solution n'était pas valide et impliquait des chaînes d'évènements bien trop longues et complexes.

Nous avons donc cherché d'autres solutions et adopté le principe du *store*. Un *store* est un objet JavaScript partagé par l'ensemble des composants regroupant toute les variables modifiables. Il représente l'état de l'application. Nous avons mis en place un *store* dans lequel figurent les objets Three du rendu. Ainsi, ceux-ci sont accessibles pour tous les composants Vue dans lesquels nous importons le *store*.

Les composants permettant de modifier des variables dans le rendu sont associés à des écouteurs d'événements. Lorsque la valeur est modifiée, celle du *store* l'est également. Lorsque les valeurs du

store sont modifiées, une fonction permettant de modifier les valeurs dans le rendu est déclenchée et le rendu est alors mis à jour. Ce principe est illustré dans la figure 5.6.

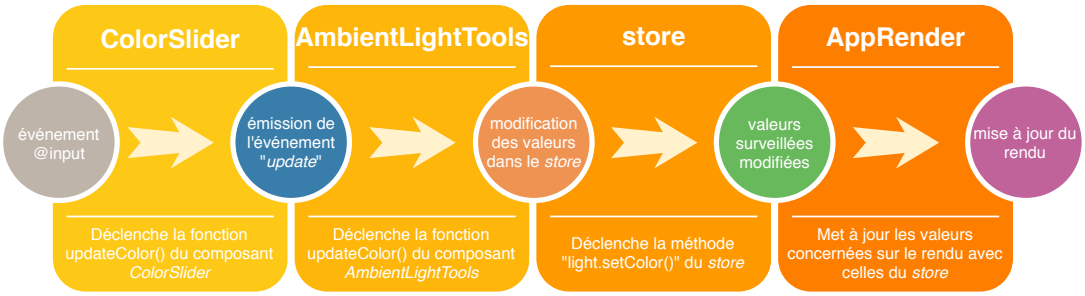


Figure 5.6 – Exemple d'interaction entre le composant ColorSlider et le *store* pour modifier la couleur de la lumière ambiante.

Cette implémentation a impliqué un changement dans notre modélisation initiale de pseudo-classes pour les objets du rendu. En effet, nous devions impérativement placer tous les objets dans le *store* ; ainsi, le système de pseudo-classes présenté précédemment n'était plus adapté.

	Fonction	**Outil**
Code	Langage	JavaScript
	Module bundler	Rollup
	Gestionnaire de paquet	NPM
	Linter	eslint
Bibliothèques	Serveur	node, express
	Rendu tridimensionnel	three.js
	Interface graphique	vue.js

Table 5.1 – **Résumé des principales technologies choisies pour l'implémentation.**

Chapitre 6

Conclusion

6.1 Avancement du projet

Au moment où nous rédigeons ce rapport, le travail sur la partie rendu est terminé. Notre application est capable de charger et d'afficher un modèle et ses texture à partir d'un fichier .json.

Nous avons également achevé le développement de l'outil caméra. Il est possible de déplacer librement la caméra grâce à une combinaison de clic et déplacement de la souris. Nous pouvons également zoomer avec la molette de la souris. Le déplacement de la caméra peut aussi être effectué à l'aide d'un clavier ou d'un écran tactile.

Nous avons aussi terminé la mise en place du serveur. Ce serveur permet d'afficher l'application et également de fournir la liste des modèle disponibles. Nous avons implémenté l'algorithme permettant de récupérer cette liste. Le résultat de l'algorithme est disponible, une fois le serveur lancé, à l'adresse http://127.0.0.1:3000/test.

Enfin, nous avons mis en place une interface graphique qui se compose d'un *header*, d'un *footer*, d'une *div* permettant d'afficher le rendu, ainsi que de deux *asides*. Le premier, destiné à afficher la liste des modèles, est encore vide. Le deuxième permet l'utilisation des outils concernant la scène de manière générale, comme la modification de la couleur de la lumière ambiante, de son intensité, ou l'ajout d'une lumière directionnelle.

Cependant, le développement de Lucilia Web App n'est pas encore terminé. Bien qu'il soit possible de récupérer la liste des modèles disponibles, nous ne l'avons pas encore intégrée au sein de l'interface graphique à cause d'une erreur de requête XMLHttp que nous allons résoudre le plus rapidement possible.

Nous n'avons pas pu mettre en place l'outil mesure. Nous avons défini avec le groupe précédent, s'occupant de la génération du modèle, une méthode afin d'obtenir l'échelle précise en plus du modèle. Cette échelle nous permettrait d'implémenter l'outil. Cependant, nous n'avons pas pu le développer car nous n'avons pas encore de modèle fourni par le groupe précédent.

Enfin, nous n'avons pas encore réussi à mettre en place le système de sélection des lumières afin de pouvoir modifier leurs paramètres individuellement. Nous souhaitons mettre cela en place rapidement. Nous souhaitons également rajouter le type de lumière SpotLight. Celle-ci se rapproche de la lumière qu'émettrait un lampadaire ou un spot.

L'image 6.1 est une capture d'écran de Lucilia Web App permettant de visualiser l'avancement du projet.

6.2 Difficultés rencontrées

La première difficulté que nous avons rencontrée concerne le choix des outils. En effet, il existe une multitude d'outils pour programmer sur le web, notamment pour réaliser des interfaces

Figure 6.1 – Capture d'écran de Lucilia Web App au moment du rendu de ce rapport.

graphique. Il nous a donc fallu étudier de nombreux outils afin de déterminer et de choisir le plus adapté à nos besoins.

Certains outils ont également été quelques peu difficiles à prendre en main car nous ne les connaissions pas. La réalisation de l'interface graphique nécessitant de combiner Vue et Three.js afin de pouvoir interagir avec le rendu a représenté une difficulté majeure. Nous avons dû tenter de nombreuses solutions et modifier à plusieurs reprises l'intégralité de notre application pour parvenir à un résultat fonctionnel.

Enfin, nous avons dû modifier notre modélisation initiale à plusieurs reprise car au fur et à mesure de l'implémentation et de notre prise en main des différents outils, nous nous rendions compte des problèmes que nous pouvions rencontrer avec la modélisation courante. Cela nous a donc pris plus de temps qu'initialement prévu.

6.3 Apports personnels

Ce projet nous a été très bénéfique. En effet, c'est la première fois que nous avons l'occasion de travailler sur un projet d'aussi grande envergure coordonnant trois équipes de deux personnes. Cela nous a permis d'apprendre à travailler en équipe et nous a demandé une grande organisation.

Ce projet nous a également permis de découvrir les principes et les concepts de tridimensionnalité et des interfaces graphiques qui nous étaient jusqu'ici totalement inconnus. Nous avons aussi pue approfondir nos connaissances concernant les technologies du web, dont les bases apprises grâce à l'unité d'enseignement «architecture et programmation du web» (HLIN510) nous ont été utiles.

Ce projet nous a permis de mettre en pratique et de consolider de nombreuses connaissances acquises tout au long de notre parcours universitaire. Nous avons notamment pu travailler nos

compétences en modélisation de projet. Cela a représenté une partie importante du travail réalisé sur celui-ci.

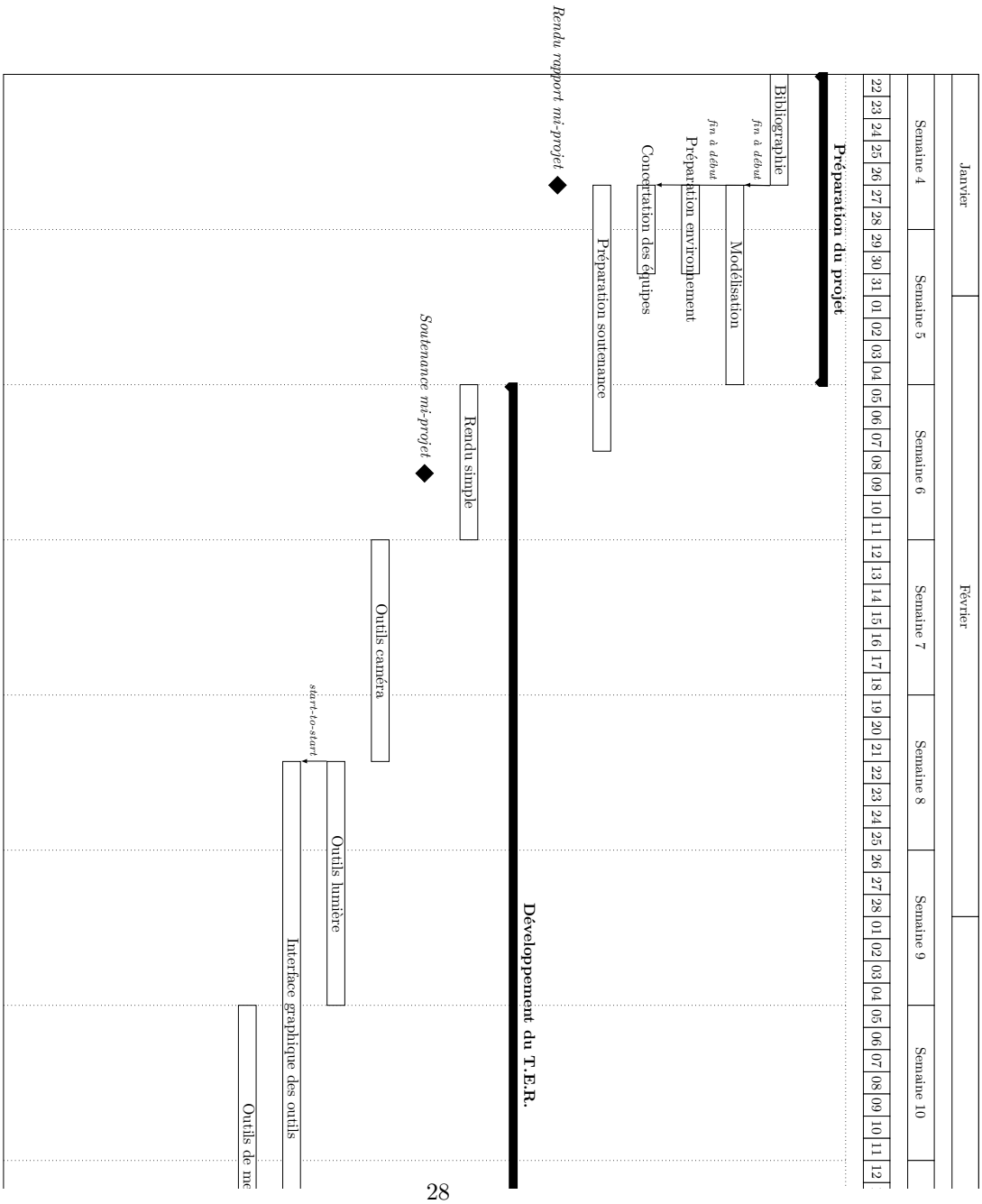
Finalement, nous avons appris à utiliser de nouveaux outils qui pourraient nous être utiles pour de futurs projets. Nous avons également pris connaissance d'autres outils qui pourraient nous servir (Angular, React, Babylon...).

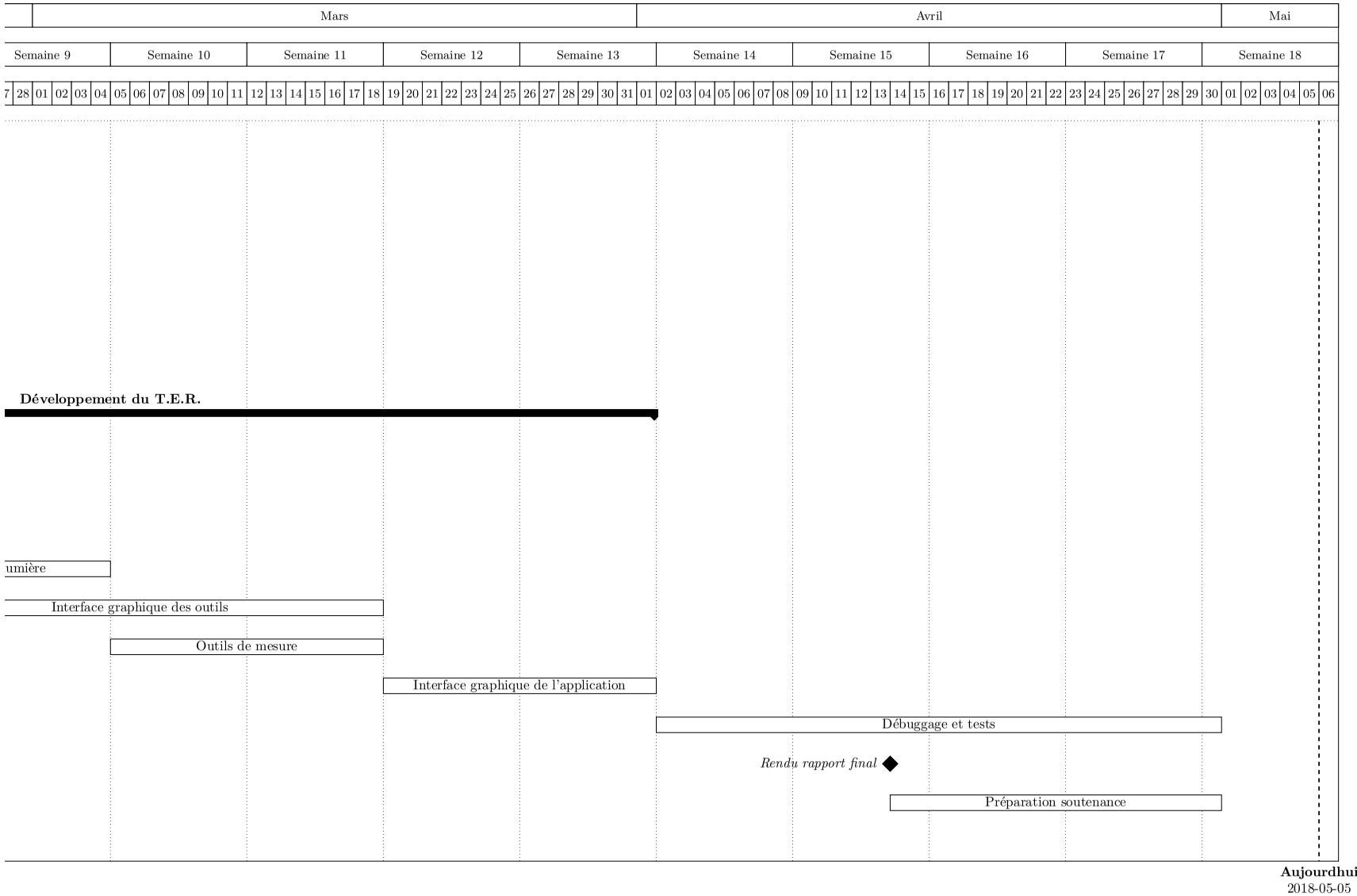
6.4 Perspectives

Lucilia n'est pas encore achevé, mais nous disposons désormais de toutes les fondations qui vont nous permettre de poursuivre son développement. Ce projet nous tient à cœur, et nous souhaitons le porter à son terme en implémentant les fonctionnalités manquantes.

Annexe A

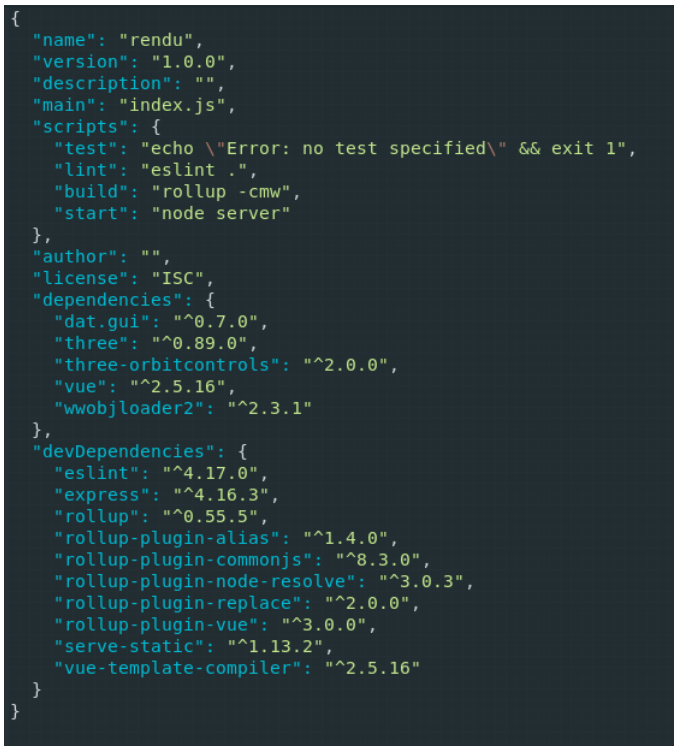
Diagramme de Gantt




Mars
Avril
Mai
Semaine 9
Semaine 10
Semaine 11
Semaine 12
Semaine 13
Semaine 14
Semaine 15
Semaine 16
Semaine 17
Semaine 18
7 28 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 01 02 03 04 05 06
Développement du T.E.R.
umière
Interface graphique des outils
Outils de mesure
Interface graphique de l'application
Débuggage et tests
Rendu rapport final
Préparation soutenance
Aujourdhui
2018-05-05

Annexe B

Configuration de npm



```
{
  "name": "rendu",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "lint": "eslint .",
    "build": "rollup -cmw",
    "start": "node server"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dat.gui": "^0.7.0",
    "three": "^0.89.0",
    "three-orbitcontrols": "^2.0.0",
    "vue": "^2.5.16",
    "wwobjloader2": "^2.3.1"
  },
  "devDependencies": {
    "eslint": "^4.17.0",
    "express": "^4.16.3",
    "rollup": "^0.55.5",
    "rollup-plugin-alias": "^1.4.0",
    "rollup-plugin-commonjs": "^8.3.0",
    "rollup-plugin-node-resolve": "^3.0.3",
    "rollup-plugin-replace": "^2.0.0",
    "rollup-plugin-vue": "^3.0.0",
    "serve-static": "^1.13.2",
    "vue-template-compiler": "^2.5.16"
  }
}
```

FIGURE B.1 – Exemple de fichier de configuration pour le gestionnaire de paquet npm.

Bibliographie

Alexis, Deveria, Phang Ng Yik, Kuczynski Piotr, Muizelaar Jeff et Barnes Craig. *Can i use WebGL*. 2017. url : https://caniuse.com/#feat=webgl (consulté le 24/12/2017).

Bourry, Xavier. *WebGL : guide de développement d'applications web 3D*. Saint-Herblain, 2013. isbn : 978-2-74607-879-6.

CV Ricardo Cabello. url : http://ricardocabello.com/ (consulté le 06/01/2018).

Dirksen, J. *Three.js Essentials*. Community experience distilled. Packt Publishing, 2014, p. 1–2. isbn : 978-1-78398-087-1. url : https://books.google.fr/books?id=3mT4AwAAQBAJ.

Documentation officielle npm. 2018. url : https://docs.npmjs.com/getting-started/what-is-npm (consulté le 05/05/2018).

Florian, Bösch. *WebGL stats*. 2018. url : http://webglstats.com/webgl (consulté le 02/01/2018).

Guillaume, Gilet. *Informatique Graphique Méthodes de Rendu*. 2012. url : http://www.unilim.fr/pages_perso/guillaume.gilet/Enseignement/Cours/MethodeRendu.pdf (consulté le 06/10/2017).

Iris, Pissens. *Théorie et historique de rendu 3D*. Rapport technique. url : https://alicelab.files.wordpress.com/2011/05/rendu3d_syllabus2011.pdf (consulté le 04/12/2017).

Kyriakidis, Alex et Costas Maniatis. *The Majesty of Vue.js*. Sous la dir. de LeanPub. 2016.

McDowell, C., L. Werner, H. E. Bullock et J. Fernald. « The impact of pair programming on student performance, perception and persistence ». In : *25th International Conference on Software Engineering, Proceedings*. International Conference on Software Engineering. IEEE ; IEEE Comp Soc, Tech Council Software Engn ; ACM ; ACM SIGSOFT ; IBM ; NORTHROP GRUMMAN Space Technol ; BMW ; NOKIA ; SUN Microsyst ; DaimlerChrysler ; Microsoft Res. 2003, p. 602–607.

MDN web docs, mozilla. *The WebGL API : 2D and 3D graphics for the web*. 2017. url : https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (consulté le 08/12/2017).

Neuhaus, Jens. *Angular vs. React vs. Vue : A 2017 comparison*. 2017. url : https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176 (consulté le 26/01/2018).

Point, Tutorials. *Web graphcis Library*. 2015, p. 11.

Remondino, Fabio et Efstratios Stylianidis. *3D Recording, Documentation and Management of Cultural Heritage*. Whittles Publishing, 2016. isbn : 978-1-84995-297-2.

Rey, Patrice. *Développement des applications web 3D avec WebGL*. Books on Demand, France, 2014.

Three.js docs. URL : https://threejs.org/docs/#api/loaders/ObjectLoader (consulté le 08/12/2017).

Tony, Parisi. *WebGL : Up and Running : Building 3D Graphics for the Web*. O'Reilly Media, 2012, p. 2–14.

Van der Ploeg, AJ. *the replacement of rasterization ?* 2013. URL : http://www.few.vu.nl/~kielmann/theses/avdploeg.pdf.