haute école
neuchâtel berne jura
arc ingénierie
saint-imier le locle delémont

ARC3D

Travail de Bachelor - 16dlm-tb-219

réalisé par
Thomas Roulin

encadrement pédagogique par
Stéphane Gobron

July 18, 2016

Résumé

Le Campus Arc 2 de la Haute-École Arc, (HE-Arc), Neuchâtel, HES-SO, est un bâtiment de grande envergure dont certaines zones ont dû être sécurisées. Des open-spaces, dont l'accès est réglementé et limité, engendrent des problèmes tant pour les visiteurs que pour les étudiants. Il est régulièrement difficile de savoir, selon le type d'utilisateur, quel chemin emprunter. Un outil facilitant les déplacements, en permettant la visualisation et le tracé du chemin pour se rendre en tout lieu et en toute salle du campus, pourrait résoudre ce problème. Ce rapport décrit le développement dudit outil, lequel facilite les déplacements à l'intérieur du bâtiment mais englobe également les itinéraires depuis la gare au campus. En résumé, le point de départ peut tout aussi être un endroit dans le bâtiment qu'un autre se trouvant dans la gare de Neuchâtel. La solution apportée se présente sous la forme d'une application web 3D accessible depuis un smartphone ou un ordinateur. La technologie utilisée est celle du WebGL, ce qui évite tous les problèmes de portabilité et d'installation.

Abstract

The Campus Arc 2 of the Haute Ecole Arc, (HE-Arc), Neuchâtel, HES-SO is a relatively large building that may be quite confusing for the newcomer. It contains multiple open spaces with regulated and restricted access that may cause problems for both visitors and students; it is difficult to know, depending on the person, which path they should take to get from one place to another within the building. A potential solution to this problem: a tool that allows the visualization of the building and the path that would get the user from a key point of the building to another. This report describes the development of said tool which also provides pathing from the train station to the campus. The solution presents itself under the form of a 3D web app accessible through a smartphone or computer. The technology used to perform this project is WebGL which avoids problems with installation process, and portability.

Table des matières

Chapitre 1

Introduction

1.1 Problématique générale

Au XXIe siècle, tout va toujours plus vite, l'être humain n'entend pas perdre de temps inutilement et, notamment, pas pour chercher son chemin. Pour se déplacer sur un site d'une certaine complexité et d'une certaine envergure, il convient d'offrir une aide au déplacement, sous la forme d'une solution rapide et facile d'accès. L'utilisation de la 3D en navigateur n'est que très peu répandue pour l'instant, mais elle pourrait tout à fait répondre à ce besoin. Proposer un tel outil, innovant et performant, aurait également l'avantage de répondre à une certaine ambition du domaine Ingénierie de l'HE-Arc de Neuchâtel, en matière de visualisation en temps réel.

1.2 Contextualisation

Dans le cadre du Campus Arc 2, le deuxième étage est considéré comme un open space. La particularité est que différents secteurs sont fermés aux visiteurs et élèves. Outre sa dimension, c'est une des causes principales de problèmes de déplacements à l'intérieur du bâtiment. Il est ainsi fondamentalement intéressant d'offrir un outil pour faciliter les trajets à l'intérieur du bâtiment et depuis la gare.

Chapitre 2

Analyse

2.1 État de l'art

Avant de démarrer le développement, il est important de se renseigner sur les travaux qui ont déjà été effectués concernant différents objectifs du projet. Le but est d'examiner si des recherches ou outils déjà créés pourraient nous aider à développer notre projet.

3D temps réel en navigateur

Dans le cadre de ce projet, il n'est pas possible de préparer à l'avance tous les différents chemins imaginables. C'est pourquoi nous devons nous orienter vers la 3D en temps réel. Le but de cette technique est de pouvoir rendre des images rapidement pour que l'œil humain ait l'impression d'une certaine fluidité. Ceci implique le rendu d'au minimum 30 images par seconde, voire 60 au mieux. Il n'est probablement pas utile de faire plus.

La technologie indiquée WebGL est maintenant supportée par la majorité des navigateurs, mobiles compris [1]. Elle offre un pont entre notre page internet et la carte graphique de l'utilisateur, c'est grâce à cela qu'il est possible d'offrir ce nombre d'images par seconde.

Cependant, le langage à utiliser est très primitif, c'est pourquoi il est intéressant d'avoir une architecture permettant de gérer une scène. Plusieurs bibliothèques WebGL offrent déjà des infrastructures ainsi que différents outils qui simplifient des tâches qui seraient redondantes et n'apporteraient rien au projet.

Le choix de la bibliothèque s'est porté sur *three.js*[2] car c'est celle qui est la plus développée et mise à jour, tout en offrant un grand nombre de fonctionnalités. Elle gère notamment le chargement de modèles et permet aussi d'effectuer, par exemple, du *Back-face culling* et du *Frustum*

culling[3].

Navigation dans un bâtiment

La navigation en intérieur alimente bon nombre de recherches, c'est un sujet sur lequel il n'existe apparemment pas de solution parfaite à l'heure actuelle. Il y a plusieurs manières d'offrir des résultats plus ou moins précis pour répondre à ce problème. La section 2.2 approfondit ce thème et décrit les différentes approches expérimentées.

Visualisation architecturale

Beaucoup de visualisations architecturales existantes se contentent de prégénérer une vidéo dans un bâtiment. De ce fait, les technologies utilisées ne sont pas les mêmes, car cela ne se fait pas en temps réel. D'autres optent pour une série de photographies parmi lesquelles il est possible de naviguer.

Il existe malgré tout plusieurs produits de visualisation en temps réel. Cependant, la plupart requièrent un logiciel installé sur l'appareil du client et, de plus, ne sont pas utilisables sur mobiles. Dans les rares produits qui utilisent WebGL, il y a par exemple Shapespark [4], qui n'en est encore qu'au stade de *pre-release* et qui semble plutôt fournir un service qui offre un produit fini pour navigateur, donc rien d'utilisable pour le développement. PlayCanvas [5] permet, de son côté, d'avoir une application en navigateur, mais l'outil est, d'une part, payant et, d'autre part, un moteur 3D qui propose bien plus de fonctionnalités que celles dont nous avons besoin.

2.2 Localisation en intérieur

Le plus grand défi de ce projet est la localisation en intérieur de l'utilisateur, par là, le degré de précision de cette localisation. De plus, il est intéressant et agréable pour l'utilisateur de disposer d'un suivi régulier de sa position durant la visualisation 3D. Cette section explique les diverses pistes empruntées ainsi que leurs résultats.

GPS

Qui dit localisation pense GPS (*Global Positioning System*), système qui pourrait, à première vue, présenter une bonne solution. Cependant, cette technologie manque de précision, principalement en intérieur. La précision

des GPS mobiles se situerait à peu près entre 4 et 50 mètres, dépendant des conditions et du smartphone utilisé. En général, l'erreur est due aux éléments entre l'utilisateur et le satellite, c'est pourquoi la localisation en intérieur ne peut pas se fier au GPS.

Triangulation Wifi

Une méthode utilisée pour la localisation en intérieur est la triangulation / trilatération, en utilisant les *Access Points* WiFi. Le concept consiste à retenir à l'avance les coordonnées de tous les AP. Chaque AP possède une adresse MAC qui les différencie les uns des autres. Ensuite, il faut scanner les AP depuis l'objet que l'ont veut localiser et, en fonction des forces des AP, il est possible de retrouver la position.

L'avantage de cette solution est l'absence d'infrastructures à mettre en place, étant donné que le Campus est déjà équipé de bon nombre d'AP. Cependant, le but de ce projet est d'offrir une application dans un navigateur Internet. Ainsi, cette contrainte empêche l'accès à toutes les informations de l'appareil concerné par son utilisation. Malgré les technologies qu'offre ce mode, il est encore impossible de récupérer les informations nécessaires à la triangulation / trilatération depuis une page web.

Triangulation Bluetooth

Une autre manière d'utiliser la triangulation est d'installer des beacons bluetooth. Il s'agit de petits émetteurs placés dans les espaces dans lesquels une localisation est nécessaire. Le fonctionnement de la triangulation est identique au WiFi.

Malheureusement, la mise en place de cette infrastructure est un désavantage à cette solution et, de plus, ces accès bluetooth ne sont pas suffisamment supportés depuis une page web. Le tableau ci-dessous 2.1, tiré du site www.caniuse.com, montre que seul Chrome peut le supporter, tout en précisant qu'il faut activer le drapeau nécessaire. Cela démontre que l'API Web Bluetooth n'est pas utilisable pour une application destinée au grand public.

Accéléromètre et Gyroscope

L'approche par accéléromètre et gyroscope est différente, car elle implique d'utiliser les capteurs internes du smartphone pour calculer une position. HTML5 fournit une API pour détecter l'orientation et les déplacements

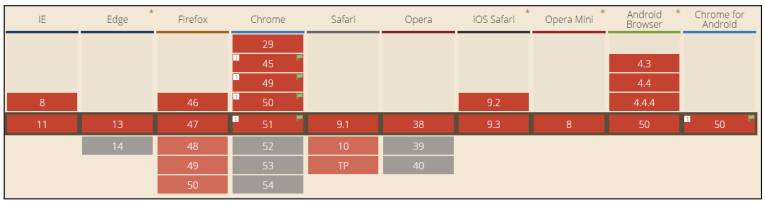

FIGURE 2.1 : Support de l'API Web Bluetooth par les différents navigateurs.

du dispositif. Autrement dit, l'accès à l'accéléromètre et au gyroscope est possible depuis un navigateur Internet.

A l'aide de l'accélération, il est théoriquement possible de calculer une position. Cependant, les valeurs de ces capteurs comportent un bruit et, pour calculer une position, il faut double-intégrer l'accélération. Cela implique qu'il faut double-intégrer le bruit et cela va créer un *drift*. En quelques secondes, l'erreur peut s'élever à une vingtaine de centimètres. En outre, le gyroscope peut avoir une valeur erronée, due à la suppression de la gravité. En d'autres termes, une erreur d'un degré sur la valeur du gyroscope représente plusieurs mètres d'erreur en quelques secondes [6].

Le but de cette approche était fondamentalement d'estimer la position de l'utilisateur, sans pour autant connaître la valeur exacte. Les démarches et réflexions prouvent qu'une approximation n'est pas envisageable avec ces capteurs.

Chapitre 3

Développement

3.1 Modèle 3D

Si l'on veut permettre à l'utilisateur de se repérer dans le bâtiment, il faut commencer par modéliser ledit bâtiment. Un projet d'étudiant a, dans le passé, permis de générer une première version d'une modélisation, à l'aide du logiciel *Blender* [7]. Cependant, après l'avoir inspectée, il s'est avéré qu'elle comportait beaucoup de polygones inutiles, voir Figure 3.1.

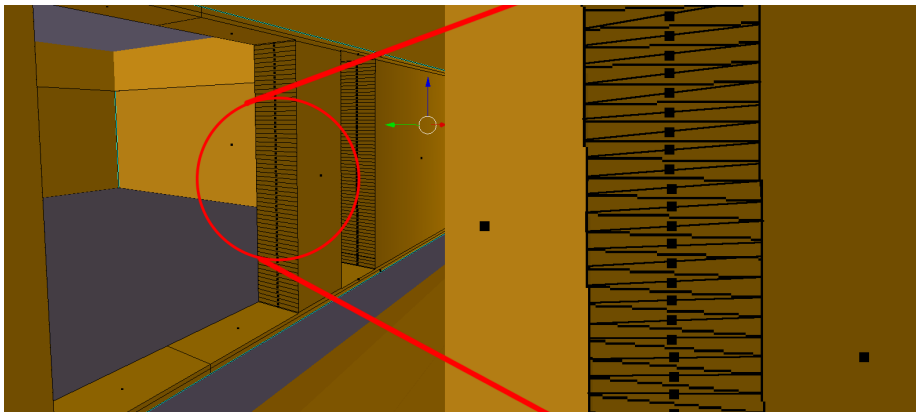

FIGURE 3.1 : Un exemple des défauts du modèle

A ce problème existent deux solutions : soit effectuer un *geometry clean up*, qui consiste à supprimer le surplus de polygones, soit modéliser à nouveau le bâtiment. Sachant que je ne possède aucune connaissance en modélisation, il a été jugé plus adéquat de modéliser une nouvelle fois l'école, ceci dans un but d'apprentissage plus logique si l'on commence par les bases.

L'environnement de modélisation choisi est *Autodesk 3ds Max* [8]. Ce logiciel professionnel offre beaucoup plus de fonctionnalités que Blender

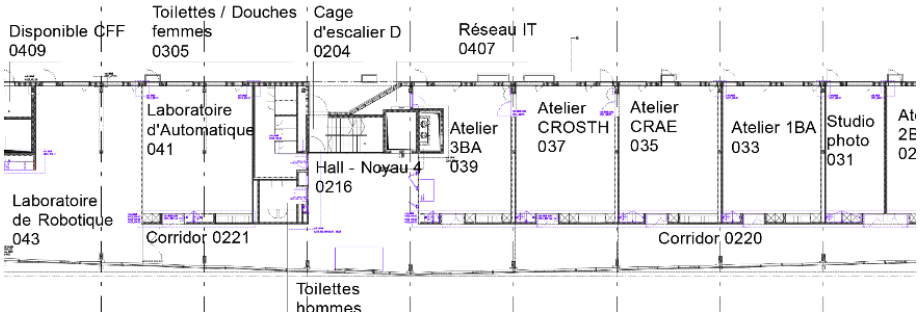

FIGURE 3.2 : Plans du bâtiment utilisés pour la modélisation.

ainsi qu'une licence étudiant gratuite, ce qui en fait une bonne solution pour la modélisation.

Modélisation géométrique

Pour modéliser correctement l'école, il est important de se munir des plans du bâtiment, car effectuer toutes les mesures sur place et à main levée est source d'erreurs et de pertes de temps. Les seuls plans qu'il a été possible de récupérer sont les plans de sols, les hauteurs ont donc été calculées par ratio. Ici, le but n'est pas la précision exacte du modèle mais de réaliser où l'on se trouve. Ce n'est donc pas un problème d'avoir un quelconque manque de précision sur les hauteurs des étages.

D'abord, la forme des murs a été reportée depuis le plan (Figure 3.2) sur *Autodesk 3ds Max* (Figure 3.3). Ensuite, la forme des murs a pu être extrudée afin d'obtenir des objets 3D, voir Figure 3.4. A partir de ce modèle, il a été possible de commencer à pouvoir travailler la véritable forme du bâtiment, c'est-à-dire de modéliser les pas de portes et fenêtres, les sols et plafonds, ainsi que les cages d'escaliers (Figure 3.5 et 3.6), pour obtenir, au final, un modèle complet du Campus Arc 2 de la Haute-Ecole Arc de Neuchâtel (Figure 3.7).

Texturisation du modèle

Maintenant que nous avons un modèle du bâtiment, nous voulons que les utilisateurs le reconnaissent. L'étape de texturisation doit servir à cela. On va appliquer des textures qui correspondent aux véritables matériaux afin de faire ressembler le modèle au maximum à la réalité.

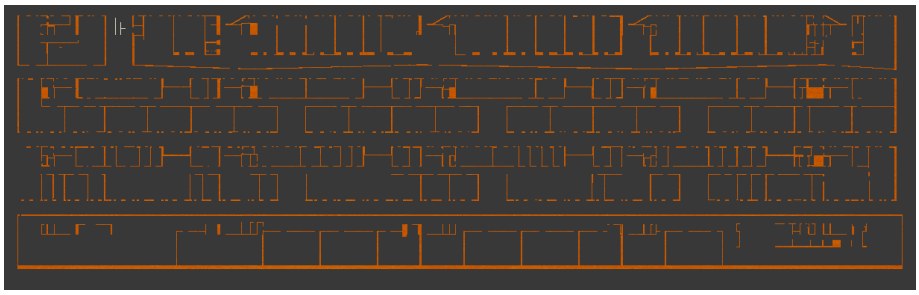

FIGURE 3.3 : Le résultat après le report du plan sur *Autodesk 3ds Max*.

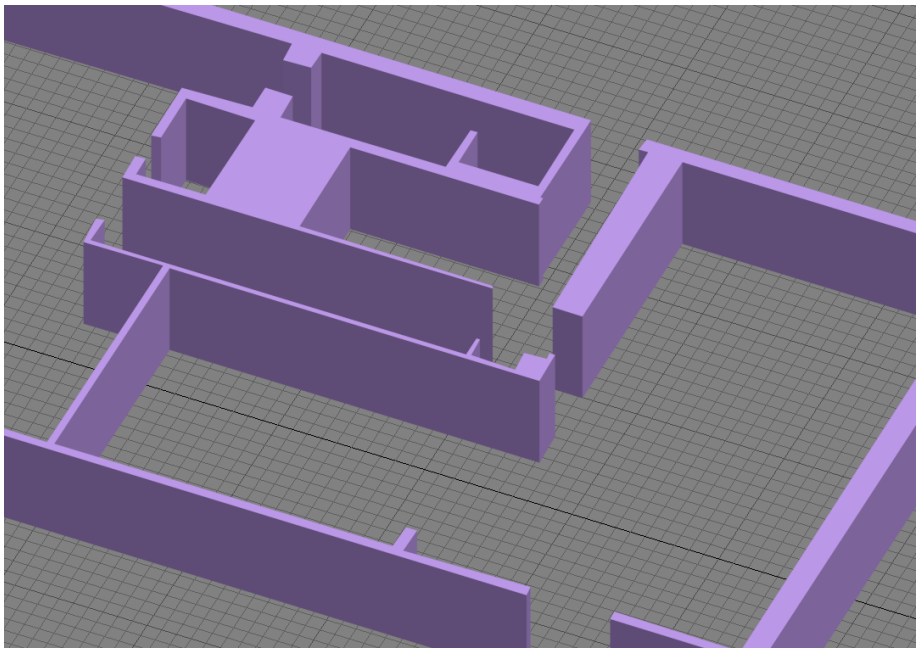

FIGURE 3.4 : Les murs du bâtiment après extrusion.

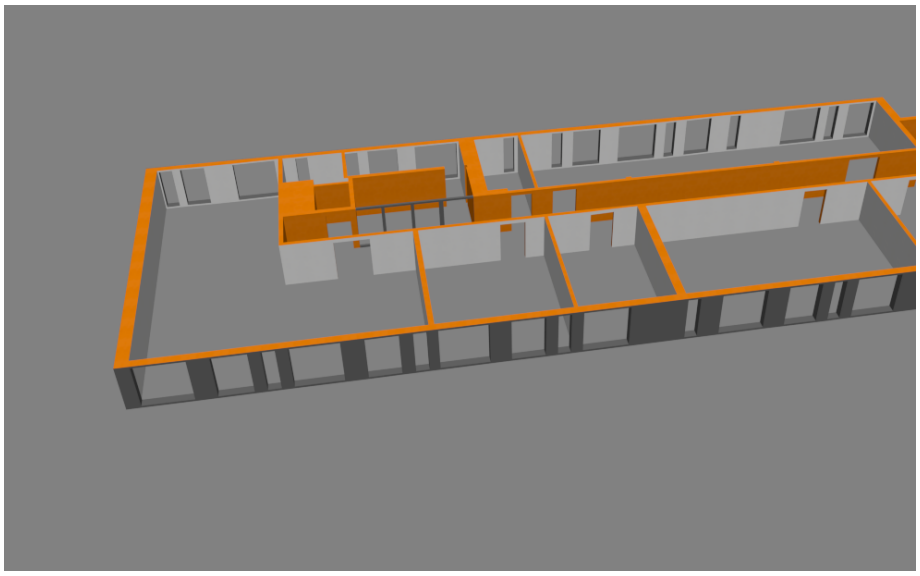

FIGURE 3.5 : Un étage du bâtiment.

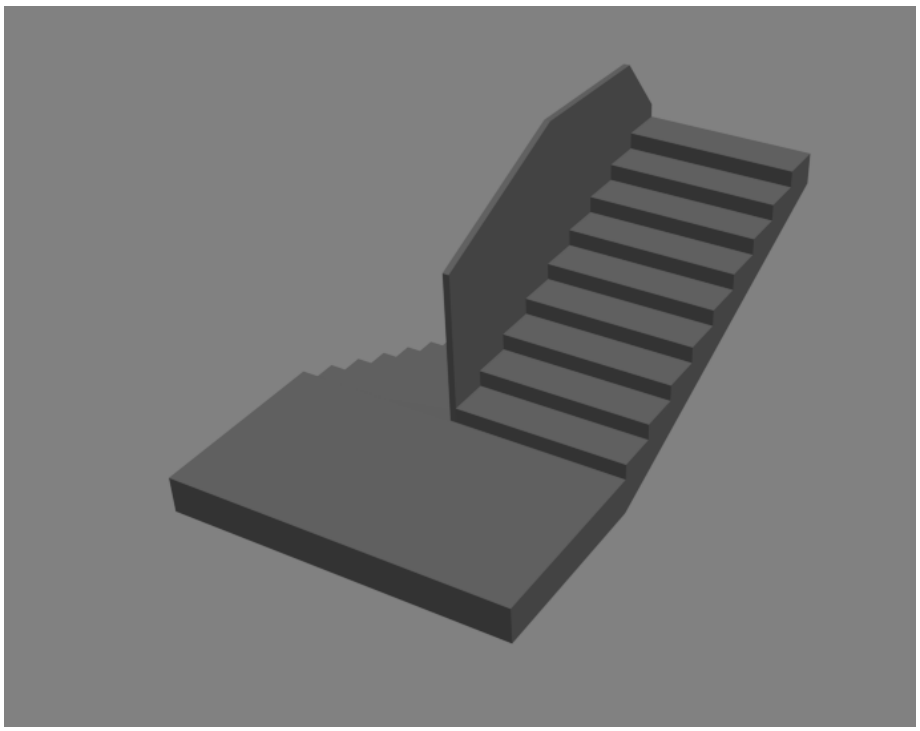

FIGURE 3.6 : Un exemple d'escalier

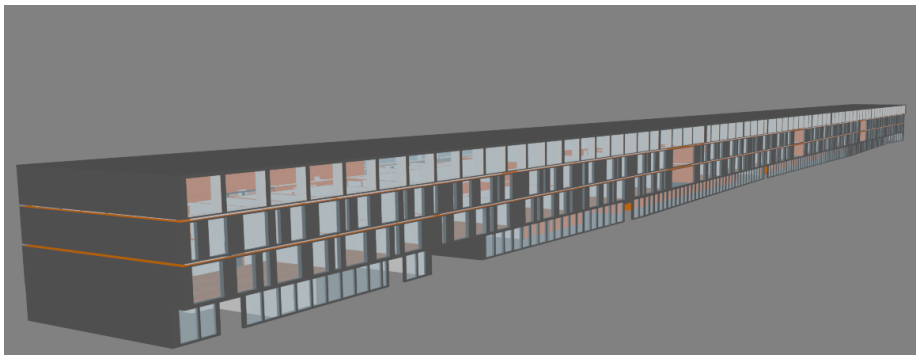

FIGURE 3.7 : Le modèle du bâtiment complet.

Exportation et importation

Afin de pouvoir utiliser notre modèle avec threejs, il faut employer un format de fichier pour pouvoir le transférer. Plusieurs formats ont étés analysés et expérimentés. Il en ressort que celui qui est le mieux supporté par threejs est le format *JSON*[9]. Il est pourtant simple d'exporter depuis *Autodesk 3ds Max* en format *OBJ*, mais threejs supporte mal le grand nombre de polygones avec ce format (voir Figure 3.8).

Autodesk 3ds Max ne permet pas, par défaut, d'exporter un modèle en format JSON. Il est néanmoins doté d'un système de plugins qu'il est possible de créer et d'ajouter au logiciel. Les développeurs de *threejs* l'ont déjà pensé et mettent à disposition un plugin permettant d'exporter notre modèle au format JSON. Cependant, la librairie étant encore en alpha, tout n'est pas toujours à jour, il a donc fallu modifier le code (écrit en Maxscript [10]) afin qu'il corresponde aux derniers standards de la librairie et que l'on puisse l'utiliser sans problème. Du côté WebGL, la librairie offre des utilitaires permettant de charger ces fichiers facilement.

3.2 Rendu graphique

Afin de rendre notre image à l'utilisateur, il existe différentes manières de calculer la couleur d'un objet en tout point. La version la plus primitive est de simplement afficher la texture que l'on a appliquée dessus, le problème est que ce n'est pas du tout agréable à regarder. C'est pour cela qu'il a été découvert plusieurs techniques afin de simuler un comportement des couleurs et des lumières afin d'obtenir un rendu plus réaliste.

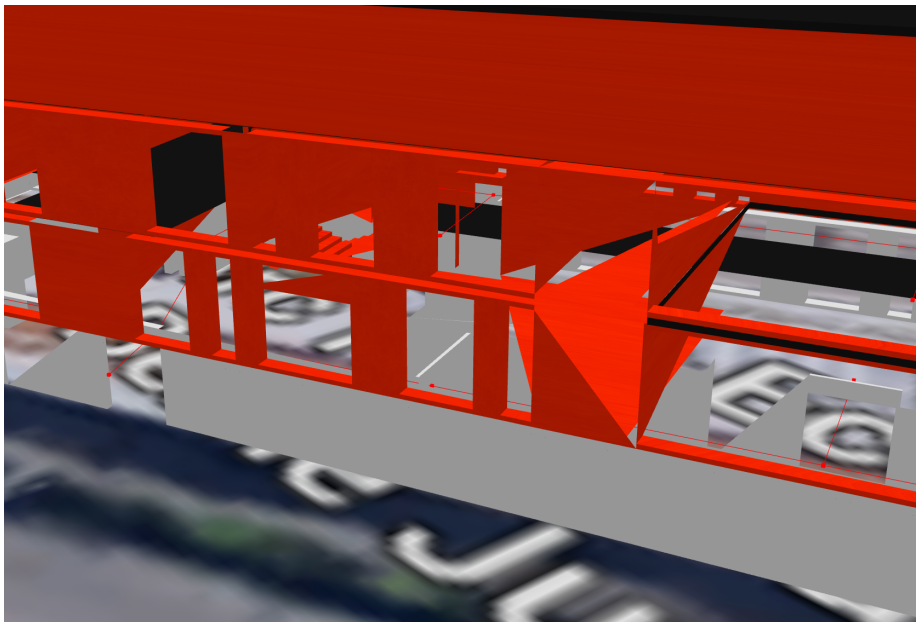

FIGURE 3.8 : threejs supporte mal les modèles composés de beaucoup de polygones en format OBJ.

Type de matériaux

Quand on parle de matériaux dans *threejs*, on va en fait définir de quelle manière se comporte le Shader [11] et sur quel modèle d'illumination on se base. Afin de gagner du temps et d'améliorer les performances, j'ai décidé d'utiliser les matériaux, donc Shader, de *threejs* car ils sont exhaustifs et déjà optimisés. Deux choix s'offrent alors à nous : soit le matériau de Lambert (Shading de Gouraud [12] et un modèle d'illumination de Lambert [13]), soit un matériau de Phong (Shading et modèle d'illumination de Phong [14] [15]).

Pour des raisons de performance sur mobile, c'est le modèle de Lambert qui a été choisi. Les rendus y sont moins photo-réalistes, mais il est bien plus important de privilégier le fait d'avoir plus de 30 images par seconde sur mobile.

Éclairage

L'éclairage de la scène a dû être limité, les modèles d'illumination demandent de faire une moyenne de toutes les sources de lumières en chaque point. Ce qui ne pose, en fait, aucun problème sur des ordinateurs de bureau, mais les smartphones ne sont pas tous capables de calculer cela, tout

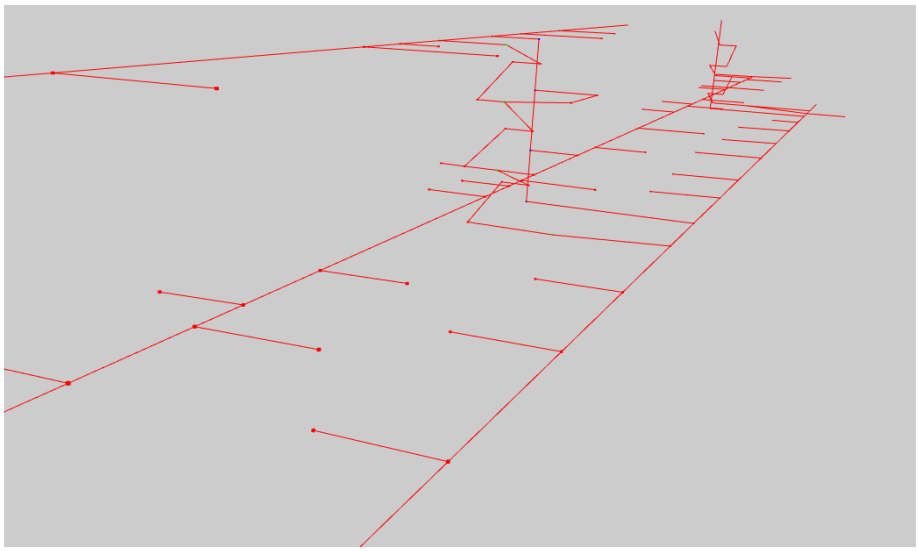

FIGURE 3.9 : Graphe du bâtiment, les points plus épais sont les nœuds.

en assurant les 30 images par seconde. Dans notre cas, on doit se contenter d'une lumière ambiante, une directionnelle et quelques sources ponctuelles.

3.3 Recherche de chemin

Cette section explique l'infrastructure et la logique qui se trouvent derrière la recherche d'un chemin. Le but est de trouver le chemin le plus court entre deux salles.

Nœuds et Graphe

Afin de pouvoir utiliser un algorithme de recherche de chemin, il faut tout d'abord mettre en place un graphe [16]. Un graphe est une structure composée d'un nombre défini de nœuds reliés entre eux. Dans notre cas, chaque nœud est un endroit prédéfini du bâtiment et il existe toujours un chemin entre deux nœuds de ce graphe, voir Figure 3.9.

Un nœud est décrit par les paramètres suivants :

- *id*, nombre, unique ;
- *nom*, texte, optionnel ;
- *position*, (x,y,z) ;
- *voisins*, tableau des identifiants voisins.

Algorithme de recherche de chemin

Pour privilégier la vitesse de calcul, l'algorithme de recherche de chemin choisi et implémenté est l'algorithme *A** [17] (*A star*). Grâce à lui, on trouve *une des meilleures* solutions en très peu de temps. En l'utilisant sur notre graphe, il est possible de trouver un des meilleurs chemins d'un nœud A à un nœud B.

Ensuite, une nouvelle fonctionnalité est venue s'ajouter au projet. Le but est de trouver les toilettes les plus proches de l'utilisateur. La différence avec la première version est que, cette fois, on désire chercher une liste de nœuds et plus un seul. Afin d'être sûr de trouver le nœud le plus proche, on ne peut plus utiliser l'algorithme *A**. C'est pourquoi, lorsqu'on se retrouve dans ce cas là, le programme bascule sur un algorithme de *Dijkstra* [18] et, avec cette recherche, il suffit de s'arrêter dès que l'on tombe sur un des nœuds que l'on recherchait.

3.4 Contrôles de la caméra

L'application a pour but d'être utilisée sur mobile et tablette mais cela n'empêche pas qu'elle soit accessible par un ordinateur. Cela implique qu'il faudra gérer de manière différente ces deux types de clients. Afin de détecter à quel type d'appareil la page doit répondre, la meilleure technique est encore d'utiliser une expression régulière sur le *User agent* [19].

Orientation mobile

Sur un ordinateur, il est possible de se déplacer à l'aide des touches fléchées. Sur un mobile, on ne peut pas utiliser le même fonctionnement. L'idée est d'utiliser le gyroscope interne du smartphone à l'aide de l'API *DeviceOrientation & DeviceMotion events*[20] supporté par tous les navigateurs mobiles[21].

Le senseur nous envoie alpha, beta et gamma qui sont respectivement les rotations sur les axes Z, X' et Y", voir Figure 3.10. On peut en déduire facilement les angles d'*Euler*, puis il est possible d'en trouver le quaternion qui exprime la rotation que l'on applique ensuite à notre caméra. Cependant, les smartphones n'envoient pas forcément des valeurs absolues, c'est-à-dire qu'il n'est pas possible de trouver le nord à partir de ces valeurs. Le magnétomètre n'est pas accessible depuis une page web, ce qui aurait pu nous aider à trouver le nord. L'interface utilisateur permet de décaler la valeur alpha du gyroscope à gauche ou à droite, afin de pouvoir calibrer à la main son orientation.

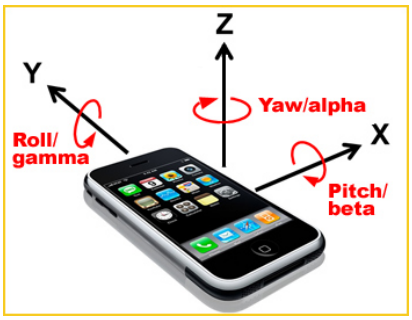

FIGURE 3.10 : Les axes ainsi que leurs rotations respectives du téléphone mobile. Source image : `http://hillcrestlabs.com`

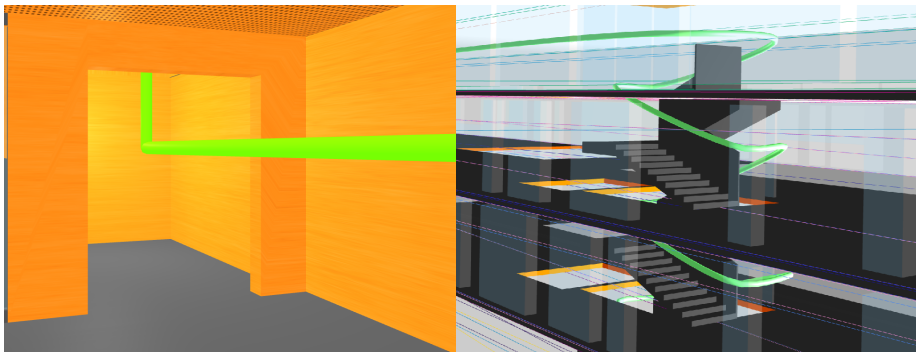

FIGURE 3.11 : La différence entre une courbe qui passe par les escaliers ou un ascenseur.

Suivi du chemin

Après que l'utilisateur ait choisi un point A et un point B, l'algorithme de recherche de chemin va définir un itinéraire à suivre. La caméra doit parcourir ce chemin, tout en gardant une certaine orientation afin que l'utilisateur puisse le comprendre. Afin que les contours soient plus naturels et moins brutaux, il est important d'interpoler les points que l'on doit suivre. Dans le cas d'Arc3D, c'est une interpolation avec méthode de Catmull-Rom qui est utilisée. Ceci pour la raison que cette méthode fonctionne avec autant de points que nécessaires et qu'il s'agit d'une méthode déjà connue de ma part.

Il reste encore un problème : si le mode *à mobilité réduite* est activé, la recherche de chemin va plutôt emprunter les ascenseurs que les escaliers. Dans ce cas-là, on va créer des courbes supplémentaires, pour chaque section du tracé, voir figure 3.11. Le programme calcule la distance parcourue de l'utilisateur et détermine sur laquelle des courbes il se trouve.

Ensuite, il reste à définir la direction de la caméra. Comme expliqué avant, il y a une courbe pour chaque section du tracé. Chacune d'elles est liée à un comportement qui est différent pour les ascenseurs (*elevator behaviour*) et pour les autres sections (*normal behaviour*). Le comportement de type *normal* va simplement regarder un point qui se trouve à une certaine distance devant l'utilisateur sur la courbe. Tandis que le comportement *elevator* est décrit comme suit : au début d'une courbe *elevator*, on stocke un vecteur unitaire dans la dernière direction de la courbe précédente et un deuxième dans la première direction de la courbe suivante. Ensuite, tout au long du trajet, on effectue une interpolation linéaire entre les deux vecteurs directeurs calculés précédemment. Cela permet d'imiter le comportement d'un humain dans un ascenseur, comportement qui l'amène généralement à pivoter sur lui-même pour être prêt à sortir du bon côté de l'ascenseur.

Live et Simulation

L'application propose encore deux modes de visualisation différents, lesquels ont été nommés *Live* et *Simulation*. Le mode *Live* devait à la base représenter le mode dans lequel l'utilisateur était géolocalisé en permanence. Étant donné que cet objectif n'est pas réalisable, c'est un autre mécanisme qui a été mis en place. La visualisation procède au déplacement seulement si l'orientation du téléphone regarde dans la direction du chemin. Tandis que le mode *Simulation* se déplace continuellement et que la caméra est fixée sur le chemin.

3.5 Interface graphique

L'utilisateur doit pouvoir interagir avec l'application, il faut donc lui offrir un interface graphique qui lui permette d'effectuer les différentes actions possibles. Cela dit, si la cible principale est un smartphone, cela implique que l'espace réservé pour l'interface est restreint. L'idée est donc de disposer de quelque chose de léger, avec les contrôles sur les bords de l'écran (Figure 3.12).

FIGURE 3.12 : Interface graphique d'Arc3D.

L'interface se compose de cinq zones bien distinctes (références couleurs sur la figure 3.13) :

- En *rouge*, le choix du trajet ou de la destination, ainsi que la possibilité d'activer ou non le mode *personnes à mobilité réduite* ;
- En *vert*, un seul bouton qui permet de passer en mode plein écran et d'en revenir ;
- En *jaune*, un curseur permettant de modifier la vitesse de déplacement ;
- En *bleu*, deux boutons pour choisir le mode *Live/Simulation* ainsi que deux boutons de mise en marche/pause/arrêt ;
- Le fond qui est notre canevas WebGL.

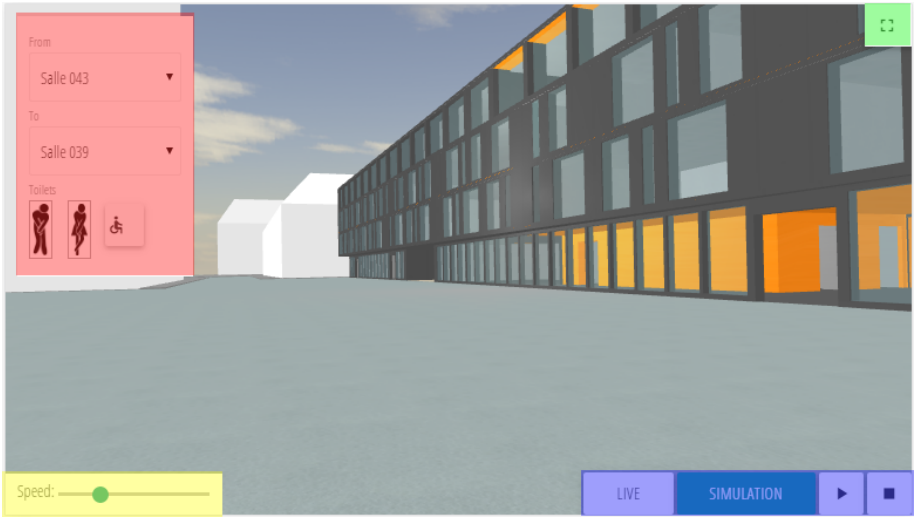

FIGURE 3.13 : Les différentes zones de l'application.

La dernière fonctionnalité de l'interface n'est pas graphique. Par dessus le canevas WebGL se trouvent deux zones cliquables : une à gauche et une à droite (Figure 3.14). Elles permettent d'effectuer la «calibration» du gyroscope à la main, c'est-à-dire d'ajuster la direction de la caméra dans la scène, en décalant respectivement à gauche ou à droite.

FIGURE 3.14 : Les deux zones cliquables qui permettent de décaler le gyroscope.

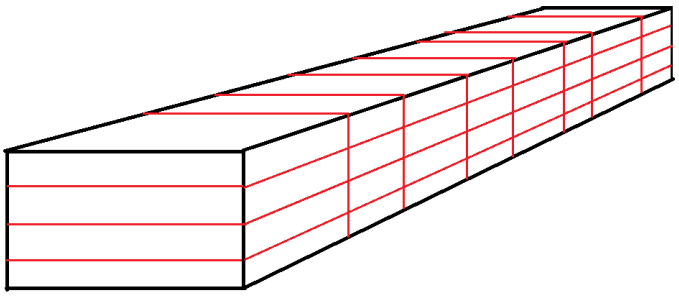

FIGURE 3.15 : Schéma d'une découpe possible du bâtiment.

3.6 Rendu en temps réel

Notre objectif est d'avoir un rendu en temps réel, c'est-à-dire une fréquence minimale de 30 images par seconde. Si tout le bâtiment devait être calculé à chaque image, il est évident que nous n'aurions pas un rendu assez rapide. Comme expliqué dans le chapitre *Analyse* de ce rapport, la librairie WebGL que l'on utilise (three.js) offre du *Backface et Frustum culling*. Grâce à ces avantages, il est déjà plus simple d'effectuer un rendu en temps réel.

Une solution imaginée pour gagner en performance était d'effectuer une découpe du bâtiment en de nombreux morceaux (Schéma en figure 3.15). Chacune de ces pièces serait sauvegardée dans deux versions différentes : une en haute qualité et l'autre en moins bonne qualité. A partir de là, l'idée était de mettre en place un système *LoD (Level of Detail)* [22], ce qui signifie qu'on charge le modèle en bonne qualité seulement si la caméra est proche, sinon on charge le modèle de qualité moins bonne. Cela dit, les tests effectués n'ont pas montré de réel gain de performance. Il a été conclu que cela était dû au *culling* déjà géré par three.js.

Textures indexées

Un problème de performance est apparu après avoir exporté la version texturée du modèle. Afin d'appliquer plusieurs textures sur un même objet avec *Autodesk 3ds max*, il faut utiliser des *Multi/Sub-Object Material*. Il s'agit, en fait, d'une liste de matériaux indexés. On l'applique à notre objet et, ensuite, on définit quel indice du matériau la face affiche. Cependant, après avoir utilisé ce genre de matériau sur le modèle, le nombre d'images par seconde sur mobile est descendu aux alentours de deux à trois. L'analyse a fait remarquer que le *renderer* est appelé environ 5000 fois et, sans les textures, il n'est appelé que 30 à 40 fois. Le problème est issu du fait

que, pour chaque fragment, il doit changer de texture. Et c'est précisément cette partie qui lui prend énormément de temps. La solution adoptée est de classer notre géométrie par textures, ainsi le *renderer* ne doit changer de textures qu'un nombre limité de fois et, par voie de conséquence, le nombre d'images par seconde est de nouveau en dessus de 30.

Matériel utilisé

L'application offre au minimum 30 images par seconde. Mais ce nombre d'images par seconde n'est pas très objectif sans les informations du matériel utilisé. Les tests effectués sur un ordinateur portable ont été réalisés avec un *HP EliteBook 8570w* :

OS Microsoft Windows 10

Chipset Mobile Intel® QM77 Express Chipset

CPU 3rd Generation Intel® CoreTM i7 Quad-Core

GPU A NVIDIA Quadro K2000M

Pour les tests sur smartphone, c'est un *Motorola Nexus 6* qui a été utilisé :

OS Android OS v6.0 (Marshmallow)

Chipset Qualcomm Snapdragon 805

CPU Quad-core 2.7 GHz Krait 450

GPU Adreno 420

Chapitre 4

Résultats

Dans ce chapitre, nous abordons les différents objectifs et leur degré d'atteinte, cela dans le but de comprendre pourquoi certains objectifs ont été atteints, d'autres partiellement atteints et un certain nombre non atteints.

4.1 Modélisation géométrique

Une ressource essentielle, dont il fallait absolument disposer, est le modèle du bâtiment. C'est une tâche qui a pris beaucoup de temps, en considérant le manque de connaissances dans le domaine. J'ai d'abord effectué la modélisation des formes de bases du bâtiment. Ensuite, j'ai reçu l'aide de Kévin Laipe (stagiaire à la Haute-École Arc) qui a continué le travail de modélisation et ensuite de texturisation, cela me permettant de me concentrer sur le développement d'Arc3D.

4.2 Temps réel

Un des points importants d'une visualisation dans le genre d'Arc3D est d'avoir une certaine fluidité. Il a fallu porter une attention particulière aux calculs envoyés à la carte graphique afin que l'œil humain ne s'aperçoive pas de coupure entre deux images rendues. Grâce à différentes astuces présentées dans le chapitre *Développement*, le produit final offre au minimum, sur le matériel utilisé, 30 images par seconde, ce qui permet d'atteindre l'objectif.

4.3 Balade de navigation

Plusieurs paramètres entrent en considération à ce sujet. D'abord, Arc3D offre une recherche de chemin optimal entre deux endroits du bâtiment ainsi que depuis la gare. Ensuite, une courbe s'affiche pour l'utilisateur (Figure 4.1). A partir de là, l'utilisateur peut choisir un mode de navigation pour démarrer.

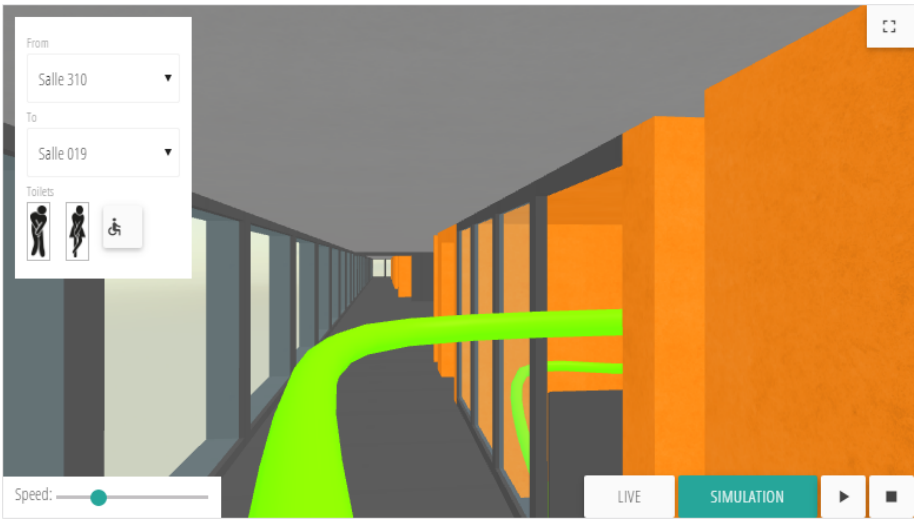

FIGURE 4.1 : La courbe qui s'affiche à l'utilisateur.

Durant son parcours réel à l'intérieur du bâtiment, suivant le chemin trouvé par le système, il est important que l'utilisateur ait vraiment l'impression de se déplacer dans le bâtiment. La caméra effectue des mouvements qui sont linéaires et qui pourraient ressembler au déplacement d'un individu. La caméra projette le parcours quelques mètres devant elle afin que l'utilisateur puisse anticiper le chemin à parcourir.

4.4 Localisation de l'utilisateur

Un des objectifs fondamentaux de ce travail est de localiser l'utilisateur tout au long de son parcours. Cependant, après analyse (voir section 2.2) des différentes technologies de le réaliser, nous nous sommes rendu compte que ce n'est malheureusement pas encore faisable. Chaque approche a présenté un ou plusieurs problèmes, résumés ci-dessous :

- Le GPS ne fonctionne pas en intérieur ;

- Les accès d'une page web sont insuffisants pour effectuer de la triangulation WiFi ;
- Les navigateurs ne supportent pas encore l'API Bluetooth pour mettre en place une triangulation à l'aide de beacons bluetooth ;
- Le niveau d'erreur de l'accéléromètre et du gyroscope est trop élevé, même pour faire une approximation de la position de l'utilisateur.

Chapitre 5

Discussion

5.1 Conclusion

Arc3D permet de trouver son chemin entre deux salles du Campus Arc 2 de la Haute-École Arc. Il s'agit d'un outil facilement accessible depuis un téléphone portable, une tablette ou encore un ordinateur. La localisation de l'utilisateur, malheureusement pas suffisamment précise, n'a cependant pas pu être mise en place pour des raisons technologiques. Le comblement de cette lacune apporterait la touche finale au projet actuel, qui aurait véritablement pu épater l'utilisateur et faciliter de façon intéressante, agréable et même ludique, l'utilisation du système.

Ce projet est une preuve de concept, certains objectifs ayant pu être atteints, d'autres partiellement atteints et certains non atteints. Cela dit, les différentes approches abordées dans le cadre de ce projet ont apporté la preuve qu'un outil de ce genre est utile pour les visiteurs d'un bâtiment dont la complexité et l'envergure sont importantes.

5.2 Perspectives

Si l'on veut pouvoir offrir une localisation en intérieur, il faut utiliser une autre plateforme. Pour l'instant, une application web ne propose pas les outils nécessaires à la conception de cette fonctionnalité dans sa totalité. Le problème est que si l'on utilise une autre plateforme, par exemple une application native, cela nécessite une installation. Il est néanmoins évident que l'on ne peut pas attendre de n'importe quel utilisateur qu'il soit enclin et capable d'installer une nouvelle application, postulant qu'il ne pourrait être de passage dans le bâtiment en question qu'une seule fois ou que cette installation pourrait lui prendre du temps.

Comme autre prise et afin de gagner en réalisme, il pourrait être intéressant de travailler sur les lumières de l'environnement. Pour cela, il faudrait énormément optimiser la manière de calcul. Il pourrait, par exemple, être plus performant de pré-calculer les lumières à l'avance.

Comme autre solution, l'utilisateur pourrait disposer d'une meilleure contextualisation dans le bâtiment si on avait placé du mobilier dans les salles, des arbres devant l'école ou encore les textures des bâtiments environnants. Si toutes ces adjonctions pouvaient être réalisées, il serait nécessaire de se focaliser à nouveau sur le rendu en temps réel. Mais être sûr qu'un téléphone mobile peut rendre plus de 30 images par seconde doit être une priorité.

Bibliography

[1] *Can I Use - WebGL*. URL: http://caniuse.com/#feat=webgl (visited on 07/12/2016).

[2] *three.js library*. URL: http://threejs.org/ (visited on 07/12/2016).

[3] *Hidden surface determination*. URL: https://en.wikipedia.org/wiki/Hidden_surface_determination (visited on 07/12/2016).

[4] *Shapespark*. URL: http://www.architectureanddesign.com.au/news/industry-news/architects-gear-up-for-real-time-visualisation (visited on 07/12/2016).

[5] *Playcanvas - Architecture*. URL: https://playcanvas.com/industries/architecture (visited on 04/12/2016).

[6] *Double integration with Sensor Fusion*. URL: https://youtu.be/C7JQ7Rpwn2k?t=23m22s (visited on 06/29/2016).

[7] *Blender website*. URL: https://www.blender.org/ (visited on 04/12/2016).

[8] *Autodesk 3ds Max website*. URL: http://www.autodesk.fr/products/3ds-max/overview (visited on 06/03/2016).

[9] *JSON - JavaScript Object Notation*. URL: https://fr.wikipedia.org/wiki/JavaScript_Object_Notation.

[10] *Maxscript Introduction*. URL: http://docs.autodesk.com/3DSMAX/14/ENU/MAXScript%20Help%202012/.

[11] *Shader*. URL: https://fr.wikipedia.org/wiki/Shader.

[12] *Gouraud Shading*. URL: https://en.wikipedia.org/wiki/Gouraud_shading.

[13] *Lambert reflectance*. URL: https://en.wikipedia.org/wiki/Lambertian_reflectance.

[14] *Phong Shading*. URL: https://en.wikipedia.org/wiki/Phong_shading.

[15] *Phong reflectance*. URL: https://en.wikipedia.org/wiki/Phong_reflection_model.

[16] *Graph (Abstract Data Type)*. URL: https://en.wikipedia.org/wiki/Graph_(abstract_data_type).

[17] *A* search algorithm*. URL: https://en.wikipedia.org/wiki/A*_search_algorithm.

[18] *Dijkstra's algorithm*. URL: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[19] *User agent*. URL: https://en.wikipedia.org/wiki/User_agent.

[20] *W3C - Orientation events*. URL: https://www.w3.org/TR/2011/WD-orientation-event-20111201/.

[21] *Can I Use - DeviceMotion*. URL: http://caniuse.com/#search=devicemotion.

[22] *Level of Detail*. URL: https://en.wikipedia.org/wiki/Level_of_detail.