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

ARC3D

Travail de Bachelor - 16dlm-tb-219

Thomas Roulin

encadrement pédagogique par
Stéphane Gobron

July 17, 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 déplacements depuis la gare au campus.

La solution apportée se présente sous la forme d'une page Internet accessible depuis un smartphone ou un ordinateur. La technologie utilisée est celle du WebGL, ce qui évite tous les problèmes de librairies externes ou de plugins.

Abstract

Karim halp me pls.

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 de voir si des recherches ou outils déjà créés pourraient nous aider à développer notre projet.

3D temps réel en navigateur

Dans le cadre ce projet il n'est pas possible de pré-rendre les différents chemins imaginables. C'est pourquoi nous devons nous orienter vers la 3D en temps réel. Le but de cette dernière est de pouvoir rendre des images assez vite pour que l'œil humain ait l'impression que ce soit fluide. Ceci implique de rendre au minimum 30 images par seconde, voir 60 au mieux, plus n'est pas utile.

La technologie propice WebGL est maintenant supportée par la majorité des navigateurs, mobile 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 a utiliser est très primitif, c'est pourquoi il est intéressant d'avoir une architecture permettant de gérer notre 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'y a pas de solution parfaite. Il y a plusieurs manières d'offrir des résultats plus ou moins précis pour répondre à ce problème. La section 2.2 parle plus en profondeur de ce thème et décrit les différentes approches expérimentées.

Visualisation architecturale

Beaucoup de visualisations architecturale existantes se contentent de prégénérer une vidéo dans un bâtiment. De ce fait, les technologies utilisées ne sont pas les mêmes, ce n'est pas du 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 de ces derniers demandent un logiciel installé sur le client et de plus ne sont pas utilisable sur mobile. Dans les rares qui utilisent WebGL il y a par exemple Shapespark [4], qui n'est encore qu'au stade de *pre-release* et a plutôt l'air d'un service qui offre un produit fini pour navigateur, donc rien d'utilisable pour le développement. PlayCanvas [5] permet aussi d'avoir une application en navigateur, cependant l'outil est un moteur 3D qui propose bien plus de fonctionnalités que nous avons besoin et surtout est payant.

2.2 Localisation en intérieur

Le plus grand défi de ce projet est la localisation intérieure 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 intérieure 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.

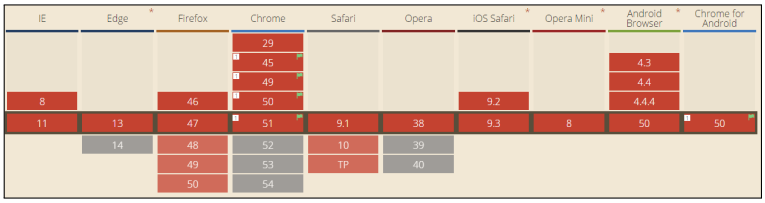

FIGURE 2.1 : Support de l'API Web Bluetooth par les différents navigateurs.

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 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 en premier lieu obtenir un modèle de ce dernier. Un projet d'étudiant a dans le passé permis de générer une première version de ce dernier à l'aide du logiciel *Blender* [7]. Cependant après l'avoir inspecté, il s'est avéré qu'il comportait beaucoup de polygones inutiles, voir Figure 3.1.

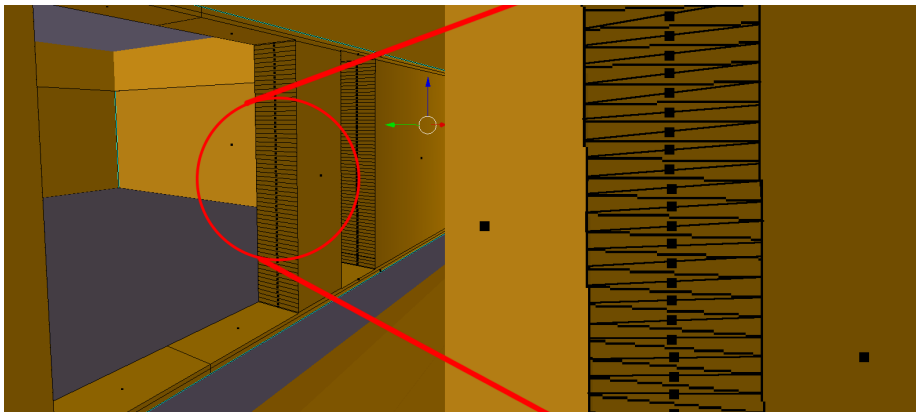

FIGURE 3.1 : Un exemple des défauts du modèle

A ce problème existe deux solutions : soit effectuer un *geometry clean up* qui consiste à supprimer le surplus de polygone, soit modéliser à nouveau le bâtiment. Sachant que je ne possède aucune connaissance en modélisation il a été jugé plus sage 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 et

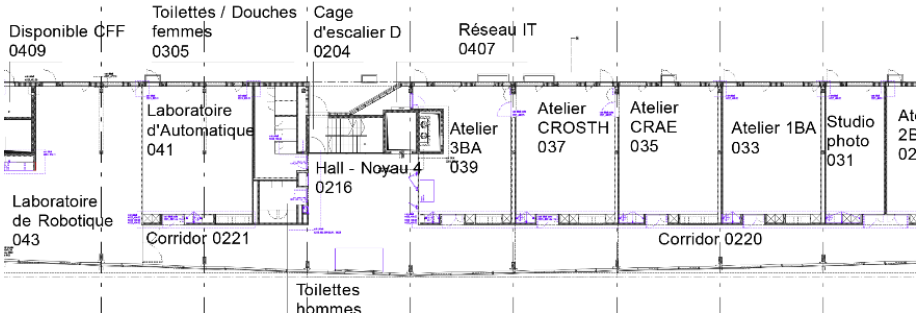

FIGURE 3.2 : Plans du bâtiment utilisés pour la modélisation.

une licence étudiant gratuite, ce qui fait de cela 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 à la main est source d'erreur et de perte 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 se 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.

Premièrement, 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 extrudé afin d'obtenir des objets 3D, voir Figure 3.4. A partir de ce modèle il a commencé à être possible de pouvoir travailler la véritable forme du bâtiment. C'est à dire 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 sert à cela, on va appliquer des textures qui correspondent aux véritables matériaux afin de faire ressembler le modèle au maximum à son origine.

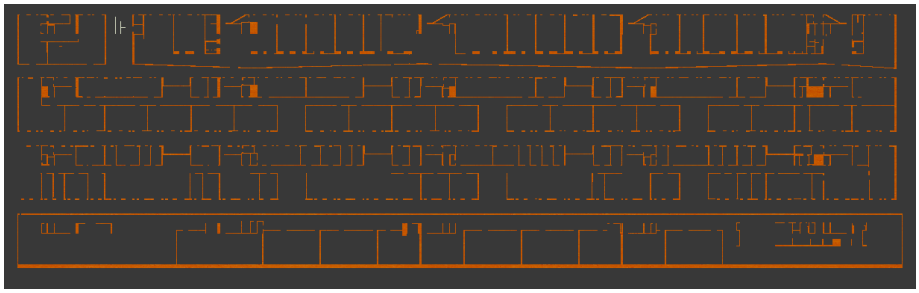

FIGURE 3.3 : Le résultat après le report du plan sur *Autodesk 3ds Max*.

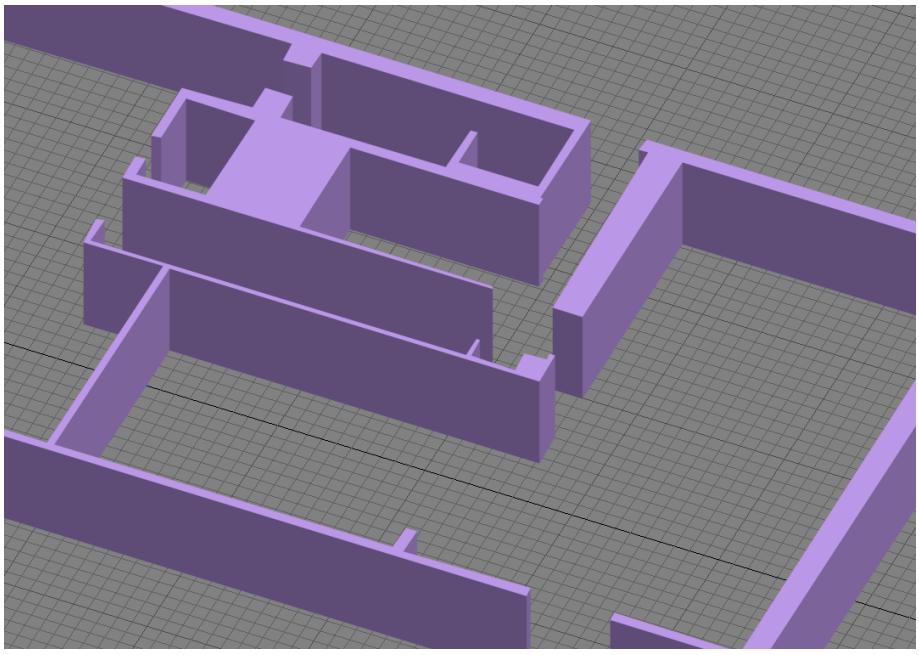

FIGURE 3.4 : Les murs du bâtiment après extrusion.

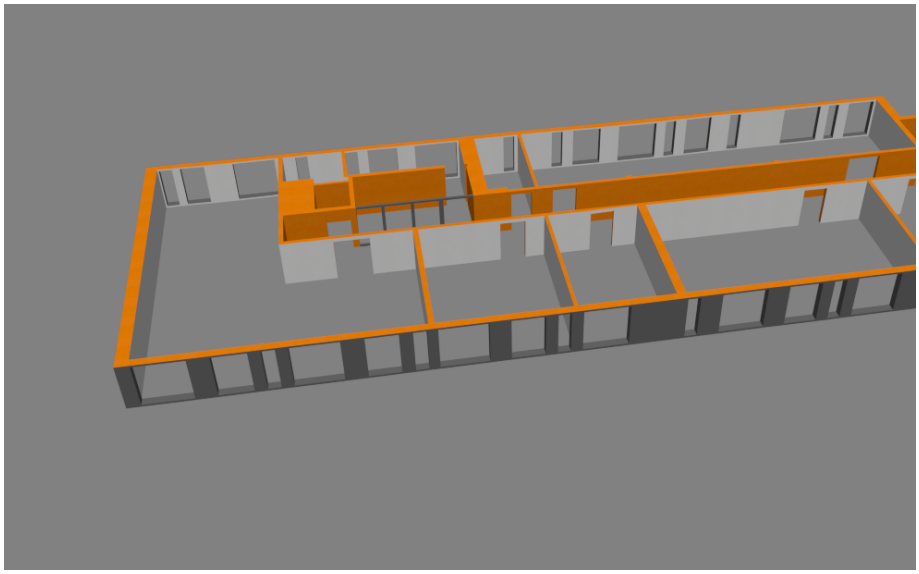

FIGURE 3.5 : Un étage du bâtiment.

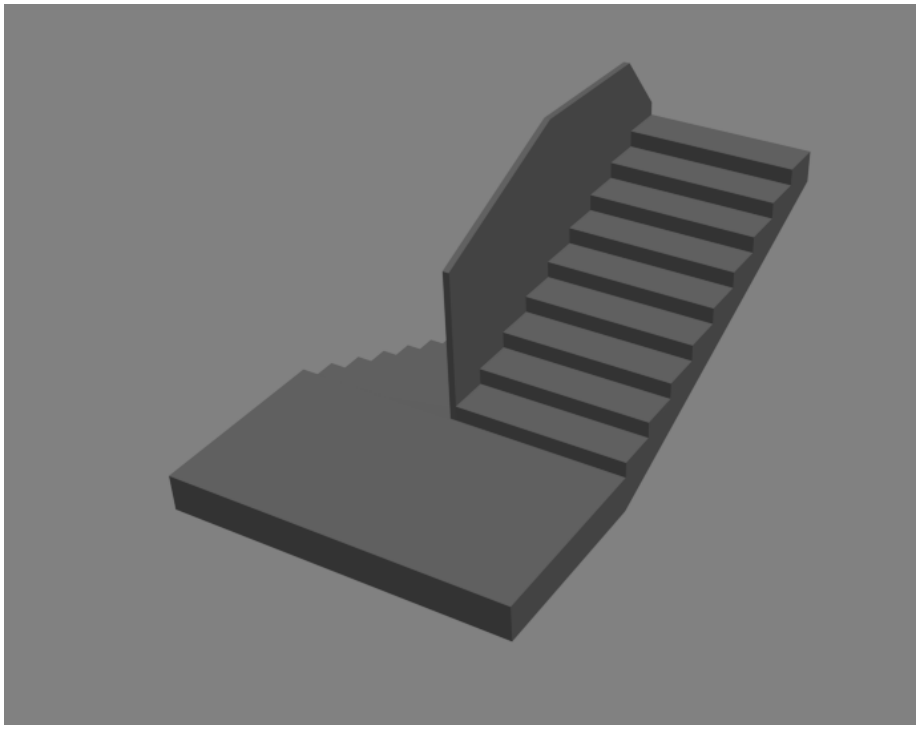

FIGURE 3.6 : Un escalier du bâtiment.

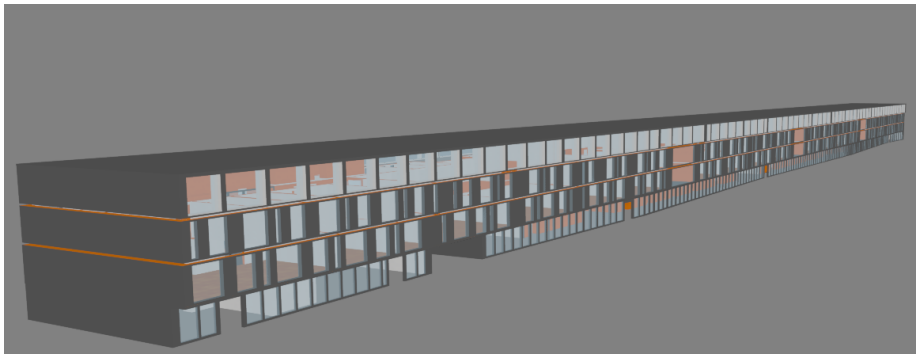

FIGURE 3.7 : Le modèle du bâtiment complet.

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, 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*, cependant threejs supporte mal le grand nombre de polygone avec ce format (voir Figure 3.8).

Autodesk 3ds Max ne permet pas de base 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 développeur de *threejs* ont déjà pensés à ça 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 appliqué dessus, le problème est que c'est loin d'être 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.

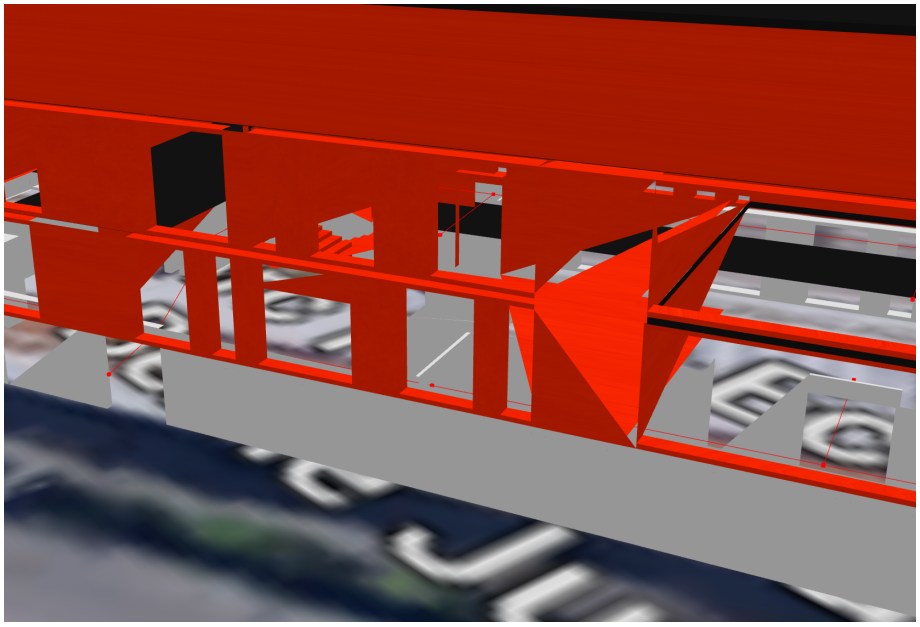

FIGURE 3.8 : threejs supporte mal les modèles composés de beaucoup de polygones en format OBJ.

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 des performances j'ai décidé d'utiliser les matériaux, donc Shaders, de *threejs* car ces derniers sont exhaustifs et déjà optimisés. Deux choix s'offrent alors à nous : soit le matériaux de Lambert (Shading de Gouraud [12] et un modèle d'illumination de Lambert [13]), soit un matériaux 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 sont moins photo-réalistes, mais il est bien plus important d'avoir plus de 30 images par seconde sur mobile.

Éclairage

L'éclairage de la scène est limité, les modèles d'illuminations demandent de faire une moyenne de toutes les sources de lumières en chaque point. Ce qui ne pose aucun problème sur des ordinateurs de bureau, mais les smartphones ne sont pas tous capable de calculer cela tout en assurant les 30 images par seconde. Dans notre cas, on doit se contenter d'une lumière

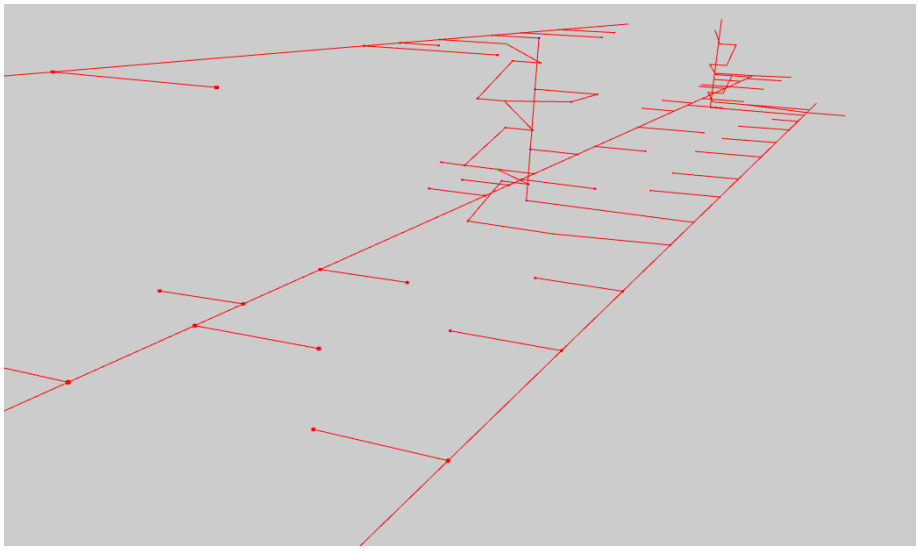

FIGURE 3.9 : Graphe du bâtiment, les points plus épais sont les nœuds.

ambiante, une directionnelle et quelques sources ponctuelles.

3.3 Recherche de chemin

Cette section explique l'infrastructure et la logique qui se trouve derrière la recherche de 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 fini de nœuds reliés entre eux. Dans notre cas, chaque nœud est un endroit prédéfini du bâtiment, 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 ce dernier, on trouve *une des meilleures* solutions en très peu de temps. En utilisant ce dernier 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 si 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 utilisé 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 mobile[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.

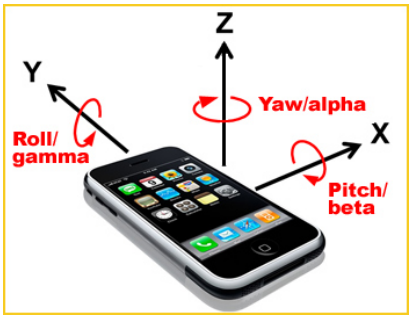

FIGURE 3.10 : Les axes ainsi que leurs rotations respectives d'un téléphone mobile. Source image : `http://hillcrestlabs.com`

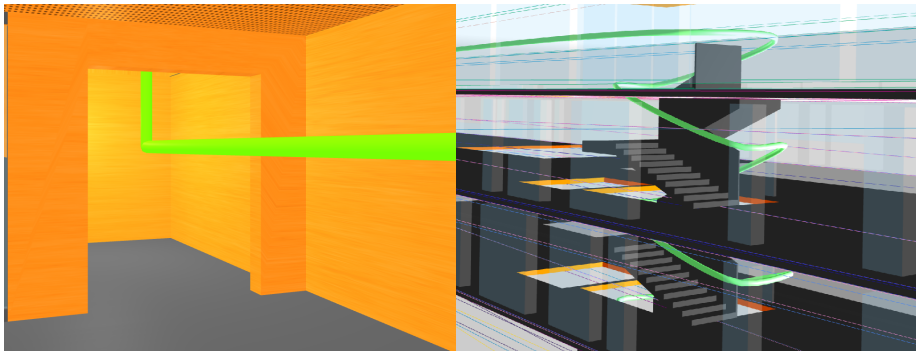

FIGURE 3.11 : La différence entre une courbe qui passe par les escaliers ou un ascenseur.

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 comprenne. Afin que les contours soit 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 utilisé. Ceci pour la raison que cette méthode fonctionne avec autant de points que nécessaire et que c'est une méthode déjà connue de ma part.

Il réside 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 est 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 de ces dernières est lié à un comportement qui est différent pour les ascenseurs (*elevator behaviour*) que pour les autres sections (*normal behaviour*). Le comportement de type *normal* va simplement regarder un point qui se trouve à une certaine distance 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.

Ce qui permet d'imiter le comportement d'un humain dans un ascenseur. Ce dernier a généralement tendance 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. Ces derniers ont été nommés *Live* et *Simulation*. Le mode *Live* devait à la base représenter le mode où l'utilisateur était géolocalisé en permanence. Étant donné que cet objectif n'était 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 le chemin. Tandis que le mode *Simulation* se déplace sans arrêt et la caméra est fixé 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. Cependant si la cible principale est un smartphone cela veut dire que l'espace réservé pour l'interface est restreint. L'idée a été d'avoir 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.

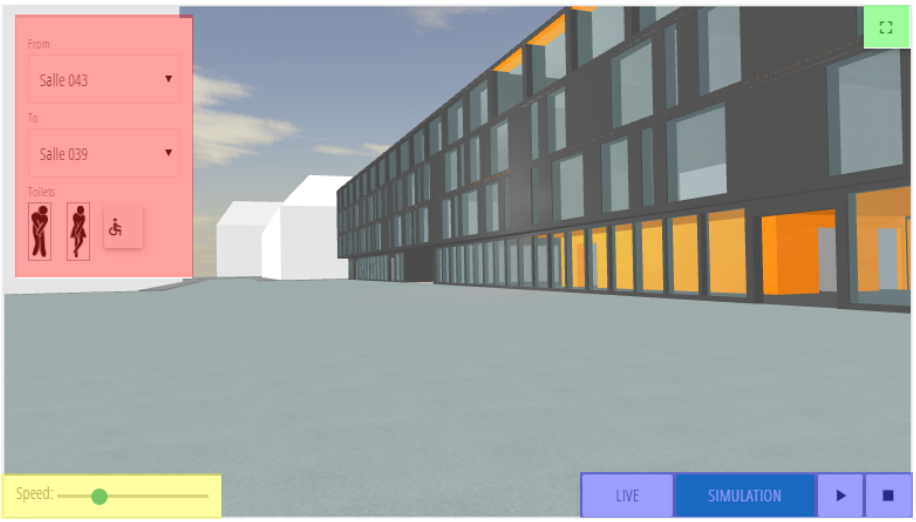

FIGURE 3.13 : Les différentes zones de l'application.

La dernière fonctionnalité de l'interface n'est pas graphique. Par dessus le canevas WebGL se trouve deux zones cliquables : une à gauche et une à droite (Figure 3.14). Ces dernières 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 la gyroscope.

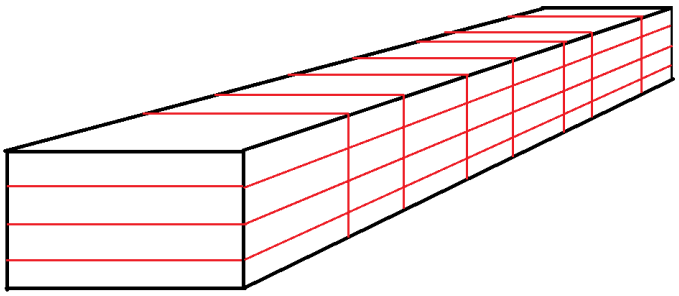

FIGURE 3.15 : Schéma d'une découpe du bâtiment possible.

3.6 Rendu temps réel

Notre objectif est d'avoir un rendu en temps réel, c'est à dire une fréquence d'image au minimum de 30 par secondes. Si tout le bâtiment devrait être calculé à chaque image il est clair que nous 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 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 de ce dernier, sinon on charge le modèle en qualité plus basse. Cependant les tests effectuées n'ont pas montrés un réel gain de performance. Il a été conclu que ceci était du au *culling* géré par la three.js.

Textures indexées

Un problème de performance s'est montré 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*. C'est en fait 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'image par seconde sur mobile est descendu aux alentours de deux ou trois. L'analyse a fait remarquer que le *renderer* est appelé environs 5000 fois et sans les textures

il est appelé 30 à 40 fois. Le problème vient en fait que pour chaque fragment il doit changer de texture et c'est cette partie qui lui prend énormément de temps. La solution adopté est de classer notre géométrie par textures, comme cela le *renderer* ne doit changer qu'un nombre limité de fois de textures et le nombre d'image par seconde est de nouveau en dessus de 30.

Matériel utilisé

L'application offre au minimum 30 images par seconde. Mais le nombre d'images par seconde n'est pas très objectif sans les informations du matériel utilisé. Les tests effectués sur un ordinateur portable se sont fait 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

Ici nous discuterons des différents objectifs et de leur réussite ou non. Ceci dans le but de comprendre pourquoi certains objectifs ont abouti contrairement à d'autres.

4.1 Modélisation géométrique

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 faire attention aux calculs envoyés à la carte graphique afin que l'œil humain ne voit 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 secondes, ce qui remplit l'objectif.

4.3 Balade de navigation

Plusieurs mécanique rentre en compte sur ce point, premièrement Arc3D offre une recherche de chemin optimal entre deux endroits du bâtiment ainsi que depuis la gare. Durant la balade qui parcourt le chemin trouvé il est important que l'utilisateur ait l'impression de se déplacer dans le bâtiment. La caméra effectue des mouvements qui sont linéaires et pourraient ressembler au déplacement d'un humain. Le caméra regarde quelques mètres devant elle afin que l'utilisateur puisse anticiper le chemin.

4.4 Réalisation des objectifs

Chapitre 5

Discussion

5.1 Conclusion

5.2 Perspectives

Objectifs pas réussis : Comment les réussir.
Améliorations possibles.
Pas de perspectives faciles.

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.