Projet Cassiopée n°84:

# Stockage et visualisation des empreintes numériques sur navigateur


Étudiants:
CHAUVEL Thomas
FROIDEFOND Benjamin
NORDIER Alexis

Encadrants:
HORAIN Patrick
FETITA Catalin

# Sommaire

# 1) Introduction

Dans le domaine de l'orthodontie, le chirurgien dentiste va très souvent être amené à réaliser des empreintes dentaires. Bien qu'originellement réalisées en plâtre avant d'être laissées dans le cabinet-même du dentiste, cette méthode présente de nombreux défauts.

C'est ainsi qu'une nouvelle révolution est en marche: celle de l'empreinte optique. En effet, alors que beaucoup d'arrachements ou de déformations de la dentition sont à craindre lors de la réalisation d'une empreinte en plâtre, l'empreinte optique permet en un temps moindre de la réaliser, avec beaucoup moins d'incertitude, et de stocker celle-ci sur l'ordinateur.

Cependant, cette manière de procéder a aussi ses propres défauts. Les empreintes réalisées ne peuvent être consultées que depuis le serveur du cabinet à moins de les stocker extérieurement. De plus, il est plus dur de les étudier intuitivement; comme sur les empreintes en plâtre, par exemple en observant les positions relatives des mâchoires supérieure et inférieure.

L'objectif de ce projet fut ainsi d'établir une base de données permettant de stocker ces empreintes numériques et de pouvoir les consulter depuis n'importe où grâce à un serveur ouvert à ceux qui reçoivent l'autorisation d'y accéder.

# 2) Comment visualiser une mâchoire sur un navigateur : *Three.js*

La question peut se poser, très légitimement, de comment afficher un modèle en 3D sur un navigateur. La tâche semble complexe, “un navigateur n’est évidemment pas fait pour ça”, “rien que cela prendra des mois”. Il se trouve en fait que des outils existent pour faciliter ce travail, le premier desquels étant *Three.js*, une librairie open source pour Javascript permettant la mise en place de complexes visualisations en trois dimensions (d’où le nom, “Three”, ou “trois” en anglais, pour trois dimensions).

Le site officiel de *Three.js*, threejs.org, offre une large variété d’exemples de visualisations 3D, certaines open source elles-mêmes. Cela nous a permis de partir d’une base solide visualisant la fameuse théière de l’Utah, référence des modèles 3D : https://threejs.org/examples/#webgl_geometry_teapot.

Une visualisation sous *Three.js* se compose de plusieurs éléments. Avant tout, un chargeur (ou *loader*) dont la fonction est de charger le modèle 3D en mémoire. Ce *loader* change en fonction notamment de l’encodage du fichier à visualiser. La théière étant encodée en .json, et nos mâchoires en .ply, il nous a donc fallu changer le *loader* avant d’espérer afficher nos mâchoires.

Le second élément important de *Three.js* est la scène. Comme son nom l'indique, la scène est l'ensemble des éléments à afficher, dans une configuration spécifiée. On ne peut voir un élément sans l'ajouter sur la scène. De même, retirer un élément de la vue de l'utilisateur revient à le retirer de la scène, sans aller jusqu'à le retirer de la mémoire. Une scène peut contenir, entre autres, les modèles des mâchoires, mais aussi un sol, ou une source de lumière (qui peut projeter des ombres sur ce sol, ou non, selon le choix du créateur de l'application). Il est également possible, si on le souhaite, de changer de scène pour afficher à volonté différents ensembles d'éléments.

Le troisième élément notable de *Three.js* est la caméra, ou en d'autres termes le point de vue de l'utilisateur. Une caméra possède une variété d'attributs qui ne nous intéressent guère dans ce projet.

Le quatrième et dernier élément notable de *Three.js* est l'afficheur, ou *renderer*, qui détermine comment l'image créée par la scène et perçue par la caméra est affichée sur l'écran de l'utilisateur. Nous ne rentrerons pas non plus dans ses spécificités.

Ainsi, pour afficher des mâchoires sur une fenêtre de navigateur, la tâche n'est pas si titanesque. En possédant une visualisation fonctionnelle, il n'est question que de faire fonctionner cette visualisation sur notre propre site, puis d'en changer le *loader* pour prendre en charge les fichiers .ply tels que nos mâchoires, et enfin d'ajouter à la scène non pas une théière mais une mâchoire haute et une mâchoire basse.

# 3) Une architecture de serveur : *Node.js* et *Express.js*

Jusque là, nous arrivons à visualiser sur notre navigateur des mâchoires en 3D. Cela est fort beau, cependant une partie du processus a été passée sous silence. *Three.js* est une librairie Javascript ; notre visualisation se fait sur une page HTML. Nous ne pouvons pas aussi simplement “demander” à notre navigateur d’afficher cette page HTML, sous peine qu’elle ne sache retrouver le code Javascript dont elle a besoin pour effectuer la visualisation. En d’autres termes, nous avons besoin d’un serveur, qui affiche la page HTML si on le lui demande et qui sache retrouver et donner à la page HTML les fichiers Javascript qu’on lui demande.

Or, jusqu’à présent, notre serveur n’est rien d’autre qu’un maigre serveur de développement lancé sous notre éditeur de code. Une des demandes de notre tuteur étant (à très juste titre) de lui permettre de visualiser lui-même ces mâchoires, il nous a fallu créer très vite un serveur plus développé. Par chance, là encore, deux noms se sont rapidement présentés : *Node.js* et *Express.js*.

*Node.js* est une librairie Javascript permettant, justement, de créer et d'opérer des serveurs web. Elle peut fonctionner seule ; cependant, elle est agréablement complémentée par *Express.js*, qui est un *framework* pour les applications web sous *Node.js*. Dit autrement, *Express.js* n’est pas utile seul mais est utile en conjonction avec *Node.js* pour en rendre l’utilisation

plus facile, sûre, et standardisée de par la prise en charge de toutes les fonctionnalités dites “bas niveau”.

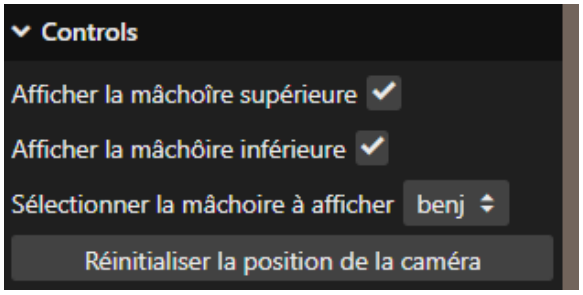
Nous avons donc créé et étendu, au fil du projet, notre serveur avec ces deux outils ; puis nous avons choisi de le rendre accessible à notre tuteur en plaçant notre serveur dans les services de MiNET, où notre tuteur peut utiliser notre application comme n’importe quel autre site internet.

Nous faisons une parenthèse ici pour ajouter un détail sur l’architecture des sites internet : en règle générale, dès qu’un site internet comprend plus de quelques pages, il est utile d’ajouter une nouvelle librairie permettant de standardiser l’affichage de ces pages au travers du site entier ; ceci afin d’éviter soit qu’un logo bouge un peu d’une page à l’autre, soit pour retirer des dizaines voire centaines de lignes à chaque page du site. Cette architecture répartit les librairies d’administration de site internet (si nous pouvons nous permettre de les appeler ainsi) en deux catégories : *frontend* (“la partie de devant”, ie ce qui est sur l’écran) et *backend* (“la partie de derrière”, ie ce qui se passe en interne pour opérer le serveur). Node.js et Express.js rentrent dans la catégorie du *backend* ; nous n’avons pas ressenti dans notre projet le besoin d’ajouter une librairie de *frontend*.

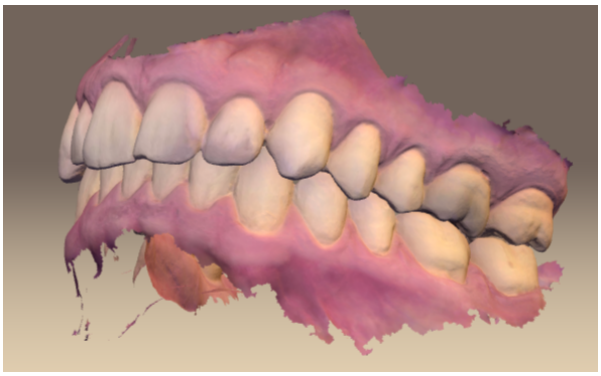
# 4) Des outils d’aide à la visualisation

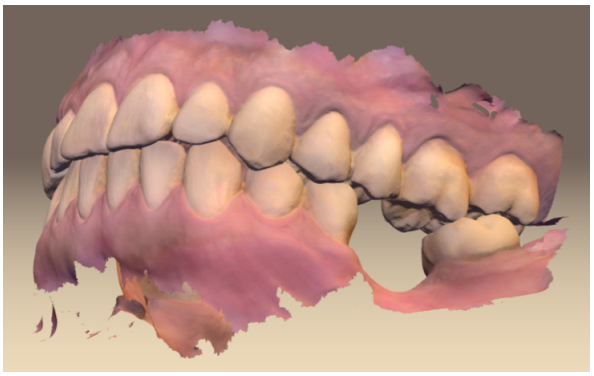
Comme le stipulait le cahier des charges, il fallait que l’on puisse observer les mâchoires de manière simple avec les outils basiques tels que le Zoom, la translation et la rotation de la caméra (réalisables à la souris). Ces options ont été implémentées à l’aide du module “OrbitControls” de la librairie Three.js.

À cela, nous avons rajouté un bouton pour pouvoir recentrer la caméra, qui est utile après une rotation suivie d’une translation qui rend la caméra difficile à manipuler. De plus, nous avons ajouté une option pour choisir si l’on voulait afficher les mâchoires ensemble ou seulement l’inférieure ou la supérieure.




Pour finir, on a implémenté un menu déroulant où l’on peut choisir à qui appartient la mâchoire que l’on affiche (celle de benj et de ludo de gauche à droite).

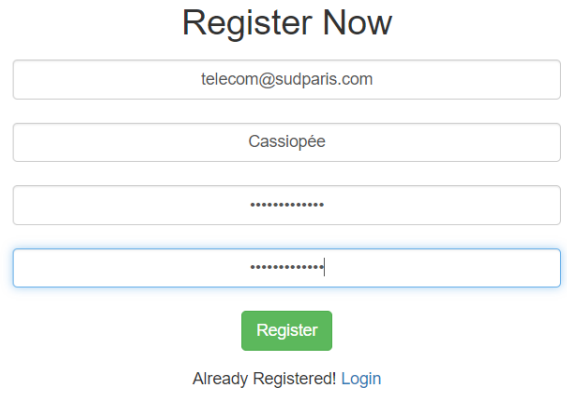


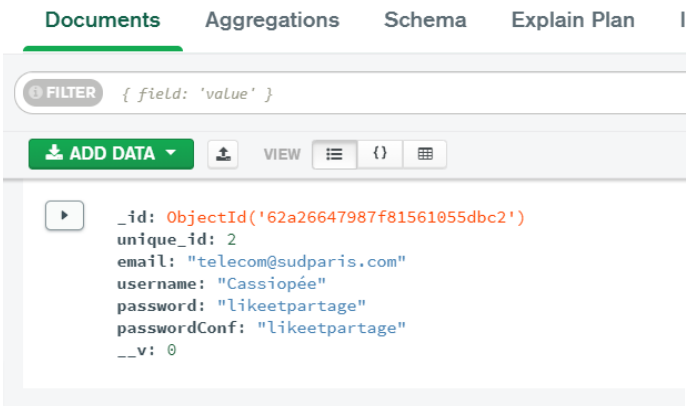


# 5) Mettre en place un login : une base de données avec MongoJS

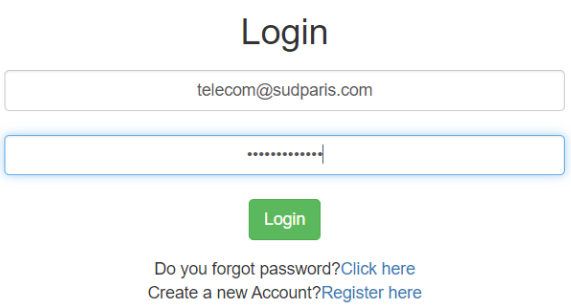
Pour notre page de login, il était nécessaire de pouvoir stocker les identifiants/mots de passe des utilisateurs, notre solution a été d'opter pour une base de données. Notre choix s'est porté sur MongoDB car elle est facile d'utilisation, et de plus, elle utilise des fichiers Json, qui sont nécessaires pour pouvoir utiliser le JWT (Json Web Tokens), qui permet, à l'aide de jetons d'accès, un échange sécurisé de données.

Une fois s'être enregistré sur la page d'inscription, nos informations sont envoyées dans la base de données MongoDB.




Ensuite, à l'aide de ses identifiants, l'on peut finalement se connecter sur la page de login et accéder à la page où sont affichées les mâchoires.



# 6) Difficultés rencontrées

Notre projet étant majoritairement porté sur du codage, il y a eu par moment des périodes où l'on avait beaucoup de mal à avancer à cause d'un manque de compréhension des outils utilisés. Nous n'avons malheureusement pas réussi à remplir toutes nos attentes et les exigences du cahier des charges

# 7) Conclusion

Nous sommes malgré tout relativement satisfaits du rendu final, bien qu'améliorable, il faudrait renforcer la sécurité du transfert de données en utilisant du JWT pour que ce projet puisse avoir une application. Également il serait appréciable d'avoir un outil pour afficher les points de contacts entre les mâchoires en surbrillance.