Application Web de Modélisation 3D et cloud computing

Benjamin Belletoile – Adrien Guillerme – Vivien Valencourt

Septembre – Décembre 2016

Encadrants : Pascal Mosbah et Julien Roland

Table des matières

Introduction

L'objectif de ce projet est de développer une solution permettant d'utiliser le logiciel ATILA déployé sur un serveur. Nous allons commencer par présenter rapidement ce logiciel, puis l'intérêt que présente le développement d'une telle solution.

Le logiciel ATILA

ATILA est un code de calcul par éléments finis développé dans le département SAMBA (Smart Acoustics Microsystems Bioengineering and Applications) de l'ISEN-Lille. L'analyse par élément finis permet de modéliser le comportement de structures soumises à des sollicitations tels que des forces, pressions, potentiels électrique ou magnétique, etc. Ce logiciel de simulation numérique est utilisé dans de multiples domaines dont l'acoustique sous-marine et l'acoustique médicale. Fort d'une histoire de près de 40 ans, ATILA est commercialisé en France et à l'international auprès de nombreux centres de recherches et grands groupes industriels.

Objectif

Actuellement, il est possible d'effectuer des calculs avec le logiciel ATILA sur son ordinateur. Il suffit d'avoir créé au préalable un modèle 3D. Le plus simple pour cela est d'utiliser le logiciel de modélisation GID car le format de fichier de ce logiciel est compatible avec celui d'ATILA.

Notre but est de déployer ATILA sur un serveur et de pouvoir l'utiliser depuis un navigateur web. Cela présente de nombreux avantages, nous allons en présenter quelques-uns. Le premier est bien sûr est de pouvoir accéder au service d'ATILA n'importe où et cela même sur un appareil limité en puissance de calcul. Cela permet aussi d'exécuter plusieurs instances d'ATILA en simultané ou de partager les résultats des calculs.

I) La structure de notre projet

Le projet est divisé en deux grandes parties : l'application Web et le serveur. L'application Web est développée en JavaScript, c'est elle qui permet à l'utilisateur de faire de la modélisation 3D dans son navigateur. Cette application devra aussi afficher les résultats des calculs du logiciel ATILA.

Le serveur est développé en NodeJS donc lui aussi en JavaScript. Nous avons fait ce choix pour permettre la compatibilité des librairies utilisées dans l'application et le serveur. Cela nous a permis de gagner du temps dans le développement. A la réception des informations du client, le serveur doit créer un maillage à partir des géométries modélisées par l'utilisateur grâce au logiciel GMSH dont nous parlerons plus tard. Pour cela il doit, dans un premier temps, convertir le fichier reçu dans un format lisible par GMSH. Une fois cela fait il faut créer un fichier lisible par ATILA qui contiendra à la fois le maillage précédemment réalisé, ainsi que les conditions aux limites appliquées sur chaque nœud du maillage. Pour finir ATILA calcul les transformations sur la géométrie et le serveur doit envoyer le résultat au client.

Nous allons maintenant détailler l'ensemble de ces étapes. Voici un diagramme de déploiement (Image 1) représentant l'architecture du projet.

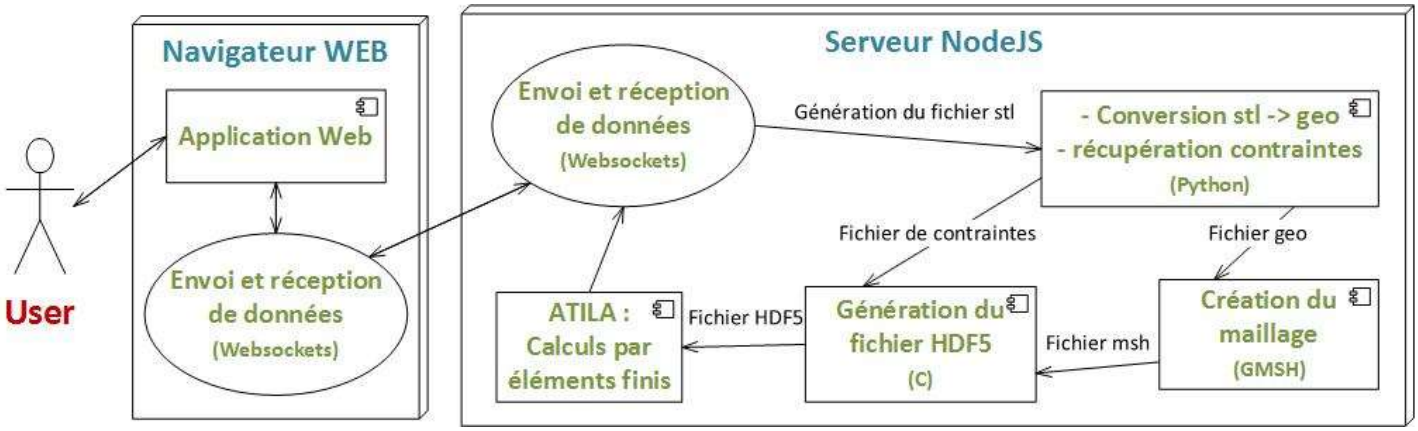

Image 1 : Diagramme de déploiement

II) Le fonctionnement

1) L'application web de modélisation 3D

Notre projet est basé sur une application Web développée en JavaScript qui se base principalement sur deux librairies : three.js et dat.GUI.js. Cette application permet à l'utilisateur de créer différentes formes (cubes, cylindres, cônes), de les déplacer, les redimensionner et d'appliquer des conditions aux limites aux différents sommets et faces.

a) Three.js

Pour commencer, il est nécessaire de parler d'OpenGL (Open Graphics Library). OpenGL est un ensemble normalisé de fonctions de calcul d'images 2D ou 3D. Cette interface de programmation est disponible sur de nombreuses plateformes où elle est utilisée pour tout type d'application.

C'est sur cette technologie qu'est basé WebGL. WebGL est une spécification d'interface de programmation de 3D dynamique pour les pages et applications HTML5. WebGL permet d'afficher, de créer et de gérer dynamiquement des éléments graphiques complexes en 3D dans la fenêtre du navigateur web d'un client. Cette technologie est actuellement intégrée dans la plupart des grands navigateurs. C'est la technologie la plus accessible pour créer des rendus en 3D dans une application web. C'est pourquoi il était naturel pour nous de nous baser sur cette technologie afin de réaliser notre application.

Three.js est une bibliothèque JavaScript qui permet de créer des scènes en 3D dans un navigateur web. Son code est open-source et son utilisation ne nécessite pas l'installation d'un plugin. Elle permet de créer des rendus en particulier en WebGL. Nous utilisons cette librairie car elle simplifie grandement la conception de rendus en 3D.

b) Dat.GUI.js

Dat.GUI est une interface pour les applications JavaScript permettant à l'utilisateur de modifier certains paramètres. Cet outil est nécessaire pour notre application car il permet d'afficher un menu en surimpression, au contraire d'un menu classique, en HTML par exemple, qui modifie le rendu et crée des problèmes d'interactions. Cette librairie est elle aussi open-source.

c) Fonctionnalités

L'application web permet plusieurs opérations simples :

- Déplacement de la caméra à l'aide de la souris : translation en maintenant le clic droit, et rotation en maintenant le clic gauche. Trois boutons présents dans le menu permettent également de placer la caméra à une position particulière.
- Création de géométries : actuellement des cubes, cylindres et cônes (Image 2).
- Déplacement des géométries : Directement à l'aide de la souris ou en utilisant le menu. (Image 2)
- Dimensionnement des géométries selon les trois axes x, y et z à l'aide du menu (Image 3).
- Sélection de sommets et de faces triangulaires (Image 4).
- Applications de conditions aux limites sur les éléments sélectionnés : actuellement des déplacements imposés selon l'un des trois axes x, y ou z. Ces conditions aux limites ne sont pas visuelles, elles seront utilisées par ATILA lors des calculs.
- Exporter les données vers le serveur afin qu'ATILA calcule les effets des conditions aux limites sur les géométries.

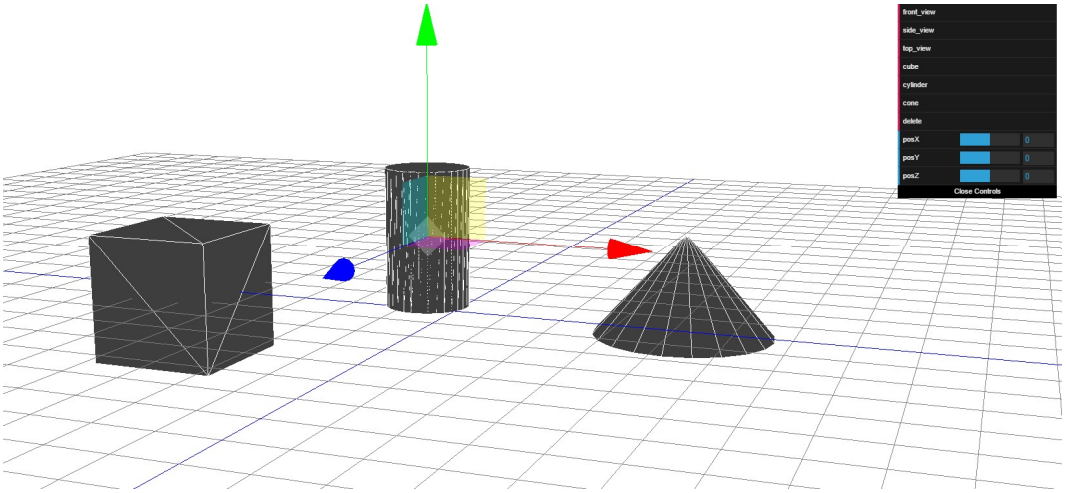

Image 2 : Création et déplacement de géométries

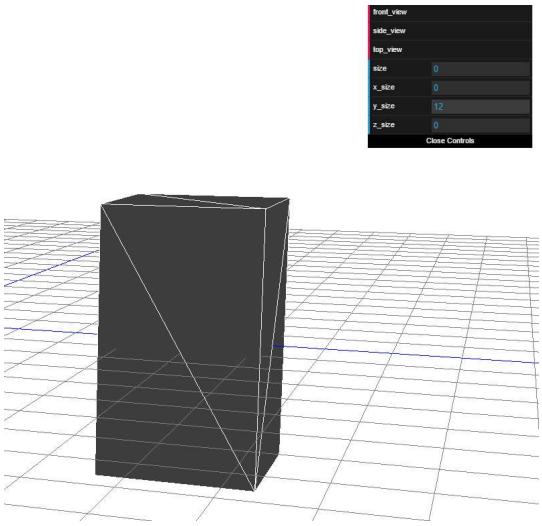

Image 3 : Dimensionnement de géométries à l'aide du menu

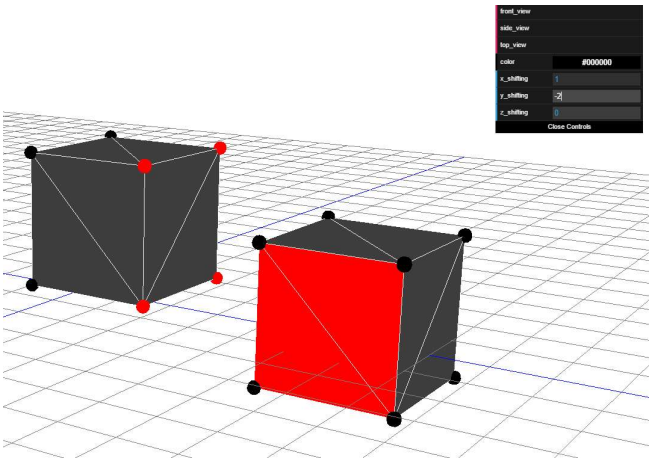

Image 4 : Sélection de faces et sommets et application de déplacements imposés à l'aide du menu

d) Utilisation

Pour utiliser cette application, il faut savoir que les fonctionnalités sont accessibles dans différents modes, qui eux-mêmes sont accessibles par l'appui d'une touche du clavier associée. Ainsi le mode par défaut, aussi accessible en appuyant sur la touche « M » est celui qui permet de créer des géométries, de les sélectionner et de les déplacer. La touche « S » permet d'accéder au mode dans lequel on va redimensionner la géométrie sélectionnée. Il existe aussi un mode permettant d'appliquer une rotation sur la géométrie sélectionnée, mais ce mode est encore en développement, il est accessible en appuyant sur la touche « R ». On appuiera sur la touche « C » pour sélectionner des éléments (faces ou sommets) et leur appliquer des conditions aux limites. Enfin on utilisera la touche « E » pour envoyer les données au serveur et permettre au logiciel ATILA d'effectuer les calculs voulus.

e) Conception

Comme nous l'avons vu précédemment : les fonctionnalités de l'application sont séparées dans différents modes, cela dans un but de clarté. Ainsi l'interface et le menu ne sont pas surchargés. Seuls les éléments nécessaires à chaque mode sont présents dans le menu. De plus cela facilite la programmation de certaines fonctionnalités. Par exemple dans le mode permettant d'appliquer des conditions aux limites, il faut pouvoir sélectionner des faces et des sommets. Pour permettre la sélection des sommets, nous affichons des sphères sur leurs positions pour permettre de simplifier la sélection. C'est plus agréable pour l'utilisateur si ces sphères sont affichées uniquement quand il souhaite sélectionner des sommets, et cela permet d'améliorer les performances car il n'est pas possible de déplacer une géométrie dans le mode d'application des conditions aux limites, et il n'est donc pas nécessaire de déplacer les sphères présentes sur chaque sommet en même temps que la géométrie.

Toujours dans ce même objectif de clarté, nous avons séparé le code de chaque mode dans différents fichiers. Chaque mode possède les trois fonctions start_mode(), stop_mode(), et on_mouse_down(). Elles permettent respectivement d'initialiser le mode (Cela permet de créer les éléments associés telles que des champs dans le menu, ou les sphères de sélection dont nous avons parlé plus tôt), De supprimer les éléments spécifiques au mode quand on le quitte (supprimer les éléments créés à l'initialisation), et définir l'action associée au clic de souris qui est différente pour chaque mode.

Pour ceux qui sont familiarisés avec la programmation orientée, nous allons maintenant décrire la classe « MyMesh » que nous avons créée. Cette classe est instanciée à la création d'une géométrie, et la propriété principale de cette classe est la géométrie. Cette géométrie est-elle-même une classe définie dans la librairie Tree.js et c'est ce qui va être affiché à l'écran. Pour faciliter la compréhension, sur chaque géométrie nous allons afficher ses arrêtes en blanc. L'ensemble de ses arrêtes est en fait une autre géométrie définie comme la première par Three.js. Elle est stockée comme seconde propriété de la classe MyMesh, cela nous permet de lui appliquer les mêmes opérations qu'à la géométrie principale (déplacement, modification de la taille, …). Ensuite on stocke dans deux tableaux les faces et les sommets auxquels on a appliqué des conditions aux limites. Ces faces et sommets sont aussi définis par des classes, ils contiennent des coordonnées de points (trois coordonnées pour un point et trois fois trois coordonnées pour une face qui est toujours définie par trois points) ainsi que des conditions aux limites : actuellement il s'agit uniquement de déplacements imposés sur chacun des trois axes x, y et z. Enfin on va stocker dans MyMesh le résultat du calcul effectué par le logiciel afin de pouvoir l'afficher à la place de la géométrie originale.

2) La communication client/serveur

Faisons un rapide récapitulatif de la situation actuelle. L'utilisateur peut donc :

- Créer une géométrie (cube, cylindre, cône)
- Placer cette géométrie où il le souhaite
- Donner la taille qu'il veut à cette géométrie
- Appliquer des conditions limites à cette géométrie

Dans la classe myMesh, on stocke :

- La géométrie
- Les arêtes de la géométrie (pour leur appliquer les mêmes transformations)
- Les sommets sur lesquels sont appliquées des conditions aux limites

On veut maintenant envoyer les données de la classe myMesh au serveur, puis renvoyer le résultat à notre application. Pour cela, on a utilisé la technologie WebSocket, très simple d'utilisation, qui permet de réaliser la communication entre le client et le serveur et donc d'envoyer et de recevoir des données. La transmission de données s'effectue de la manière suivante :

- **socket.emit('id', {'id_data1' : data1, 'id_data2' : data2});** permet d'envoyer des données. 'id' permet d'appeler la fonction ayant le même nom que 'id' au niveau du destinataire pour savoir ce qu'il faut faire lors de la réception des données. On envoie ensuite les données identifiées par des id, et qui seront récupérées sous la forme d'une table de hachage.

- **socket.on('id', function(data){});** permet de récupérer les données envoyées précédemment dans la table de hachage 'data'.

Pour en revenir à notre projet, on envoie au serveur une géométrie crée via notre application web au serveur, ainsi qu'un tableau de sommets et un tableau de faces appartenant à cette géométrie sur lesquels on a appliqué ou non des conditions limites. Le serveur récupère ces données, réalise quelques traitements dont un maillage que l'on détaillera par la suite, et renvoie ce maillage à notre application pour l'afficher dans le navigateur. Ce maillage sera ainsi stocké dans la classe myMesh afin d'éviter de devoir le recalculer si jamais la géométrie n'est pas modifiée lors du prochain traitement, car la création d'un maillage demande beaucoup de ressources.

Le but final serait d'envoyer les données nécessaires à ATILA pour réaliser une simulation, de récupérer le résultat et de l'afficher dans notre navigateur web. Pour rentrer un peu plus dans les détails, on va réaliser les étapes suivantes pour chaque géométrie que l'on a créé (on envoie les géométries au serveur une par une puis on réalise le traitement sur chacune de ces géométries de façon indépendante) :

- On envoie et on récupère via notre application le code .stl d'une géométrie qu'on a créée, on envoie ce code sous forme de texte au serveur, et on créé un fichier .stl contenant ce texte sur le serveur dans le dossier temp. Ce fichier est nommé par un numéro, qui correspond au nombre de géométries que l'on a déjà traité (si on a déjà traité une géométrie juste avant, la géométrie en cours sera nommé "1.stl").

- On doit ensuite appeler un script python et lui fournir le nom de ce fichier .stl en argument. Pour cela, on utilise un child process qui permet de lancer un traitement en parallèle sur le serveur à l'aide de la commande exec. La commande pour appeler le script python est la suivante : "'py Python/STL_to_GEO.py Temp/' + file_name + '.stl'".
- On va ensuite lancer le calcul du maillage via le logiciel GMSH (qu'on détaillera par la suite). Pour cela, on créé un nouveau child process. Cependant, comme le lancement d'un nouveau processus n'interrompt pas le déroulement du lancement principal, on est obligé de lancer ce processus via le premier child process, sinon le programme principal va écraser le calcul précédent pour réaliser le traitement des géométries suivantes. La ligne pour lancer le calcul du maillage en ligne de commandes est la suivante : "gmsh en ' + name + '.geo -3 -format stl". Le "-3" signifie qu'on réaliser le maillage sur une géométrie 3D et le "-format stl" signifie qu'on veut créer ce maillage au format stl, pour pouvoir ensuite l'afficher sur notre application.

3) GMSH

Comme ATILA est un logiciel de calcul par éléments finis, il nécessite qu'on lui donne en argument une discrétisation des géométries que l'on veut traiter. Nous allons donc créer un maillage de ces géométries.

Plus précisément nous allons créer un maillage linéaire, qui est une décomposition d'une géométrie en éléments finis, qui vont avoir des arrêtes linéaire. Dans notre cas les surfaces vont être décomposées en triangles (Image 5), et les volumes en tétraèdres (Image 6).

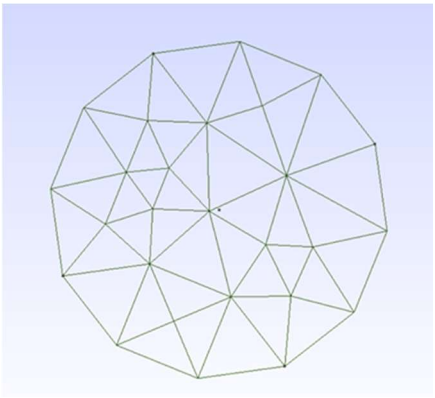

Image 5 : Maillage linéaire d'un disque

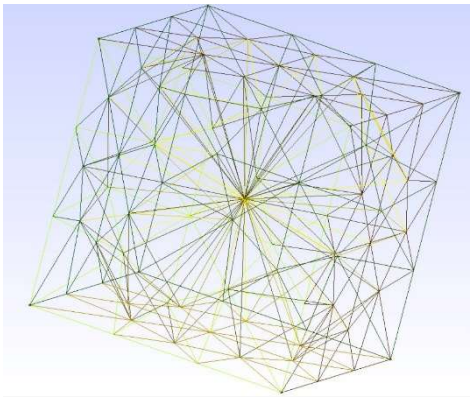

Image 6 : Maillage linéaire d'un cube

Pour faire cela nous allons utiliser le logiciel GMSH. C'est un logiciel libre de droit, développé par Christophe Geuzaine et Jean-François Remacle et qui va permettre de créer des maillages dans le format *.msh spécifiquement créé par les développeurs du logiciel dans le but de contenir des géométries discrétisées. Il est important de noter pour la suite que le format d'entrée de ce logiciel est le *.geo.

4) La conversion en fichier *.geo

La librairie three.js permet d'exporter des géométries au format *.stl uniquement. Cependant, le logiciel GMSH permet de réaliser des maillages uniquement sur des géométries au format *.geo. Il a donc fallu réaliser un script afin de réaliser cette conversion de *.stl en *.geo. Nous avons choisi de réaliser notre script en python car c'est un langage puissant, facile d'utilisation et rapide à l'exécution, notamment lors de la lecture et l'écriture de fichiers.

a) Les fichiers stl

Le format stl décrit un objet 3D uniquement par sa surface externe. Il s'agit d'une surface fermée et définie par une série de triangles. Toutes les faces sont formées à base de triangles. On définit ces triangles par leur coordonnées cartésiennes (x, y, z) et par un vecteur normal orienté vers l'extérieur de la forme afin de savoir à quelle face ce triangle appartient.

```
1  solid name
2      facet normal 1 0 0      //Vecteur normal du triangle
3          outer loop
4              vertex 0 0 1    //les 3 points du triangle
5              vertex 0 1 0
6              vertex 0 0 0
7          end loop
8      end facet
9
10     facet normal 1 0 0
11         outer loop
12             vertex 0 0 1
13             vertex 0 2 2
14             vertex 0 1 0
15         end loop
16     end facet
17 endsolid name
```

Image 7 : Exemple de fichier au format stl

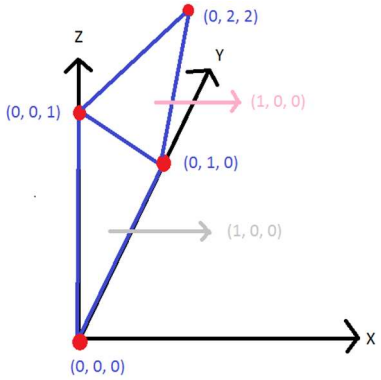

Image 8 : Schéma de géométrie telle que décrite par le format stl

Ainsi, chaque code *.stl (Image 7) décrit la géométrie formée de différents triangles (Image 8) de la manière suivante : on indique le vecteur normal du triangle puis on définit les trois sommets du triangle, et ainsi de suite jusqu'à former la géométrie souhaitée. On peut voir un exemple ci-dessus. Ce quadrilatère est formé de deux triangles, ces deux triangles formant une seule et même face plane, ils possèdent donc le même vecteur normal.

b) Les fichiers geo

Contrairement au format stl, le format geo ne décrit pas seulement une surface mais aussi un volume. De plus, l'objet 3D n'est pas défini par des triangles mais par ses faces (surfaces fermées).

Le format geo est structuré de la manière suivante en plusieurs blocs (Image 9) :

- On commence par définir tous les sommets de notre géométrie, numérotés à partir de 1 par des identifiants
- On définit ensuite toutes les arêtes numérotées à partir de 1 (on "relie deux sommets")
- On définit les faces de notre géométrie (surfaces fermées et le numéro des lignes de chaque face doivent correspondre à un cycle)
- Pour finir, on ajoute un volume à la géométrie

```
Point(1) = {0, 0, 1, 0.3};      //les sommets
Point(2) = {0, 2, 2, 0.3};
Point(3) = {0, 1, 0, 0.3};
Point(4) = {0, 0, 0, 0.3};

Line(1) = {4, 1};       //les arêtes
Line(2) = {1, 2};
Line(3) = {2, 3};
Line(4) = {3, 4};

Line Loop(4) = {1, 2, 3, 4};    //les faces
Plane Surface(5) = {4};
Surface Loop(6) = {5};

Volume(7) = {6};    //le volume
```

Image 9 : Exemple de fichier au format geo

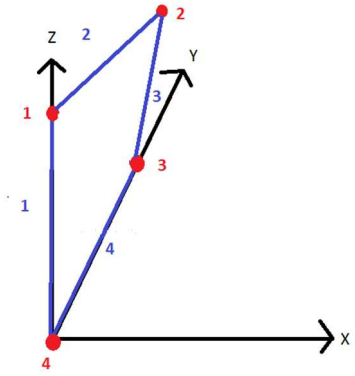

Image 10 : Schéma de géométrie telle que décrite par le format geo

On remarque que contrairement au code stl, ce quadrilatère (Image 10) n'est pas défini par des triangles mais seulement par son unique face. Il ne possède donc pas de diagonales.

c) La conversion

Pour réaliser cette conversion de stl en geo, nous avons procédé par blocs :

Le premier bloc définit les sommets la géométrie à l'aide de leurs coordonnées. Ce bloc est facile à mettre en place car les sommets sont déjà définis dans les fichiers stl.

Le deuxième bloc définit les arêtes de la géométrie par leurs deux points extrêmes. Pour ce faire, on relie tout simplement les points d'un même triangle entre eux. Le problème, c'est qu'on ne veut pas définir des triangles mais des faces. On veut donc supprimer les diagonales. Pour ce faire, on utilise le vecteur normal du triangle. Ainsi, si l'arête existe au moins deux fois sur la même face (donc est présente plusieurs fois sur des triangles différents ayant le même vecteur normal), alors on ne la crée pas sur le fichier geo.

Le troisième bloc définit les faces de la géométrie par ses arêtes limites. Pour cela, on s'aide du vecteur normal. Si deux triangles possèdent le même vecteur normal, c'est qu'ils forment une même face. On doit cependant faire attention à créer un cycle (Image 11).

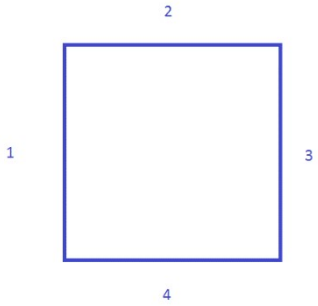

Image 11 : Une face carrée est définie par quatre arêtes dans l'ordre 1 2 3 4

Si l'on veut définir la face carrée suivante au format geo, il faut bien suivre un cycle, ici, par exemple "1,2,3,4" et non "1,3,2,4".

De plus, nous devons aussi faire attention à l'ordre dans lequel nous avons relié nos points pour former nos arêtes. Par exemple, l'arête reliant le point 1 au point 2 peut être définie dans le sens "1,2" mais également dans le sens "2,1". Il est important de garder cela en tête lors de la définition des faces, car le deuxième point de l'arête 1 doit bien correspondre au premier point de l'arête 2 par exemple. Si jamais ils ne correspondent pas, on peut ajouter un "-" devant le numéro de l'arête pour inverser l'ordre de définition des points.

5) Création d'un fichier HDF5 interprétable par ATILA

Tout d'abord, qu'est-ce qu'un fichier HDF5 ? C'est une structure de données qui peut contenir de nombreuses choses. En effet, le format HDF5 se comporte comme un dossier sous Windows, c'est-à-dire qu'il peut contenir des sous-dossiers ou des fichiers. Pour créer un fichier au format HDF5 qui puisse être interprété par ATILA, il nous faut plusieurs séries de données.

Nous devons tout d'abord récupérer tous les nœuds du maillage issu de GMSH (au format msh), puis y ajouter les conditions aux limites. En effet, dans notre application nous pouvons envoyer au serveur des données concernant un déplacement, que ce soit sur des faces ou sur des sommets de la forme.

Ensuite, pour récupérer les conditions aux limites issues de l'application, nous utilisons les web sockets pour envoyer les différents nœuds et surfaces touchés par ces conditions aux limites, puis, nous aurions souhaité les écrire dans un fichier texte.

Ce fichier texte aurait été écrit sous une forme particulière, pour pouvoir en extraire facilement les informations dans notre script python. Tous les nœuds envoyés auraient été écrit entre les mots : « Nœuds » et « FinNœuds », et les surfaces entre les mots « Surfaces » et « FinSurfaces ». De plus les différentes coordonnées de chaque nœuds auraient été séparées par des « / » et les différents nœuds séparés par des espaces (pour définir les surfaces).

a) Le format de fichier *.msh

Ensuite, avant de parler du script python, je vais définir globalement quelles sont les choses importantes à savoir dans les fichiers *.msh. Les premières lignes servent à définir le format du fichier *.msh (entre les mots « $MeshFormat » et « EndMeshFormat »). Ensuite, les nœuds du maillage sont définis entre les mots « $Nodes » et « $EndNodes ». La ligne juste après le mot « $Nodes » est un entier qui correspond au nombre de nœuds que contient le maillage. Ensuite, chaque nœud a 4 informations associées : la première est l'index du nœud (qui commence à 1), et les trois informations suivantes correspondent aux coordonnées en 3 dimensions de chaque nœud.

Enfin, les éléments du maillage sont définis entre les mots « $Elements » et « $EndElements ». Dans nos maillages (linéaires), nous n'utilisons que 4 types d'éléments :

- Les nœuds
- Les segments
- Les surfaces triangulaires
- Les volumes tétraédriques

Chacun de ces éléments a :

- Un nombre correspondant à l'index de l'élément (qui débute à 1)
- Un nombre qui correspond au type de l'élément (15 : nœud, 1 : segment, 2 : surface triangulaire et 4 : volume tétraédrique)
- Un nombre qui définit combien de valeurs seront utilisées pour définir certaines propriétés indispensables à GMSH.
- Les n valeurs suivantes sont celles qui définissent des propriétés propres à GMSH que nous n'utilisons pas (il y en a minimum 2).

- Les m dernières valeurs sont les nœuds qui définissent les éléments (une seule pour un nœud, 2 pour un segment, 3 pour une face triangulaire et 4 pour un volume tétraédrique).

b) La récupération des nœuds atteint par des conditions aux limites

Enfin, nous avons réalisé le script python qui permet de déchiffrer un fichier qui aurait été écrit sous cette forme. Ce script prend en paramètres deux fichiers : le fichier texte contenant la liste des nœuds et surfaces touchés par des conditions aux limites, ainsi que le fichier *.msh qui est le maillage de la forme exportée de l'application.

Le principe est assez simple, ce script prend une liste de nœud et une liste de surfaces en arguments, et le but est de les comparer à la liste des nœuds du fichier *.msh pour récupérer une liste de nœuds touchés par des conditions aux limites. Pour ce qui est des nœuds, il suffit simplement de les comparer un à un, mais pour les surfaces, il faut récupérer tous les nœuds qui appartiennent à cette surface, car ATILA travaille sur des nœuds.

Il a donc fallu faire un peu de géométrie. Premièrement, il fallait savoir si le point appartenait au même plan que la surface. Il faut donc, à partir des 3 points définissant la surface, déterminer l'équation du plan passant par ces trois points (à condition qu'ils ne soient pas alignés). Ensuite on teste l'équation avec les valeurs du point et si le résultat est 0, le point est bien dans le plan. Dans ce cas, on projette sur un plan, pour simplifier le résultat.

Il ne reste donc plus que deux coordonnées pour chaque point et il suffit de tester si le point appartient à la surface triangulaire, mais cette fois en 2 dimensions. Pour ce faire, nous faisons 3 tests, pour chacun des segments du triangle. Le principe est, premièrement de calculer l'équation de la droite définie par 2 des 3 points du triangle, et ensuite de tester ou est le 3e point par rapport à cette droite. On résout donc l'équation avec les coordonnées de ce point, puis avec les coordonnées du nœud que l'on teste. Si les deux sont de même signe, alors cela veut dire qu'ils sont du même « côté » de la droite. Si cette opération est vraie pour les 3 droites définies par les points du triangle, alors le nœud appartient bien à la surface triangulaire.

Une fois cette étape terminée, il faut pouvoir écrire ces données sous la bonne forme pour qu'ATILA puisse les interpréter. Pour ce faire, il existe une bibliothèque en C/C++. Cette bibliothèque s'appelle GidPost, c'est en fait ce qui est utilisé pour le post-processing de GiD. Comme le format utilisé par ATILA, est le même que celui utilisé par GiD, nous avons souhaité utiliser les mêmes outils. Cependant, cette partie n'est malheureusement pas opérationnelle et il s'agit de la seule partie manquante pour lier notre serveur et ATILA.

6) Affichage du résultat dans l'application web

Pour afficher le résultat du calcul d'ATILA dans l'application, il faut plusieurs éléments. En premier est de savoir convertir le fichier en format *.stl car c'est le seul format de rendu 3D utilisable dans un navigateur web. Il faut noter que cette conversion crée une perte d'informations car le format *.stl n'est pas un format de géométries en 3D, il permet uniquement de créer des surfaces dans un univers en trois dimensions. Cela signifie que l'on pourra afficher la surface des géométries mais pas le contenu.

Le second élément est de savoir afficher des gradients de couleur sur les géométries afin de représenter graphiquement les éventuels effets et déformations qui ont été calculés par le logiciel ATILA.

Comme vous pouvez le constater ci-dessous nous sommes effectivement capables d'afficher des gradients de couleur (Image 12). Mais la création d'un fichier lisible par ATILA étant encore en développement, nous ne pouvons bien sûr pas encore afficher le résultat du calcul. Pour le moment nous nous contentons d'afficher le maillage créé par GMSH (Image 13).

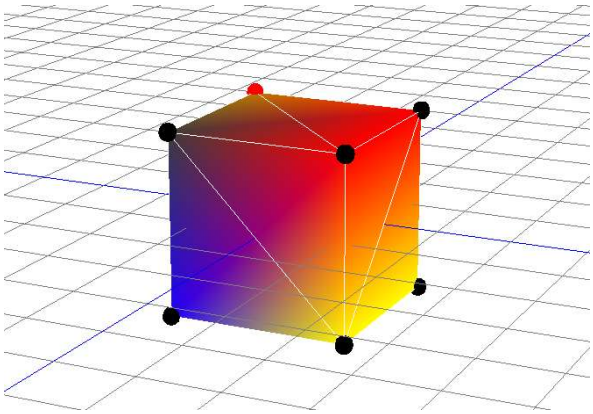

Image 12 : Gradient de couleur affiché dans l'application web

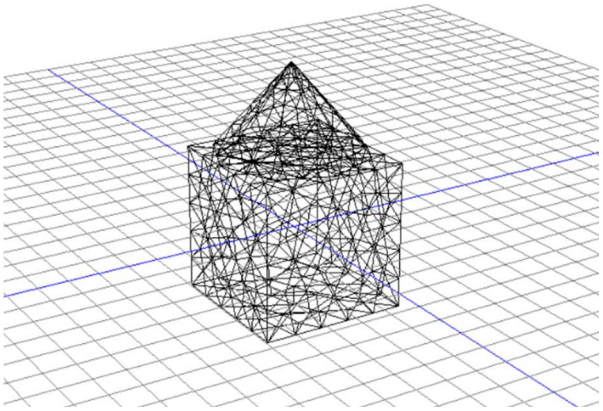

Image 13 : Maillage calculé par GMSH affiché dans l'application web

III) Etat d'avancement

1) Ce qui fonctionne

Actuellement, nous avons une bonne base au niveau de l'application. Les fonctionnalités qui vous ont été décrite précédemment sont toutes opérationnelles, à savoir :

- Créer des géométries telles que des cylindres, des cônes ou des cubes.
- Déformer ces géométries selon les 3 axes x, y, z (modification de la taille)
- Appliquer des conditions aux limites sur des faces triangulaires ou des sommets.
- Envoyer les données relatives aux géométries au serveur et afficher le maillage en retour.

Ensuite, pour ce qui est du serveur, nous avons développé toute une série de programmes qui vont permettre diverses choses à savoir :

- Générer un fichier stl contenant les informations sur les géométries exportées de notre application.
- Transformer un fichier stl en fichier geo pour qu'il soit interprétable par GMSH.
- Démarrer la création du maillage automatiquement et le convertir au format stl.
- Récupérer une liste de tous les nœuds touchés par les conditions aux limites choisies dans l'application.
- Renvoyer à l'application le maillage des géométries au format stl.

2) Ce qui ne fonctionne pas

Malgré tout cela, il reste encore quelques problèmes à corriger, ou certains outils que nous n'avons pas eu le temps de développer. Tout d'abord pour l'application :

- Le mode "Rotation" n'est pas opérationnel. Il aurait permis de faire tourner les géométries sur elles-mêmes.

Ensuite, du côté serveur :

- La partie liaison avec ATILA n'est pas terminée. Il manque la partie écriture du fichier pour ATILA.
- Le maillage des sphères pose encore quelques problèmes. Le fichier geo d'une sphère, est en fait écrit sous une forme différente des autres géométries que nous avons utilisées.

IV) Notre méthode de travail

1) Chronologie

Mercredi 28 Septembre :

- Tutoriel sur three.js,
- Création d'un cube : possibilité de le bouger et de modifier sa taille,
- Création d'un cube avec un gradient de couleur,
- Création d'un menu basique en html.

Mercredi 05 Octobre

- Ajout d'un repère 3D, ajout de flèches suivant les axes sur le cube,
- Possibilité de créer plusieurs types d'objets (un cylindre, un cône, ...),
- Possibilité de changer l'angle de la caméra à l'aide de la souris,
- Possibilité d'ajouter un gradient de couleur sur un sommet du cube.

Du Lundi 10 Octobre au Vendredi 14 Octobre

- Création des trois modes : mode création, mode conditions limites et mode changement de taille,
- Débogage de l'application web,
- Création d'un menu interactif et design avec des sliders, des textbox et des boutons (à l'aide de la librairie "Dat.GUI"),
- Fonctionnement du logiciel GMSH et comment l'utiliser en mode console,
- Exporter une géométrie au format *.stl à partir de notre application web,
- Importer un maillage au format *.stl vers notre application web pour l'afficher,
- Création d'un script python pour convertir un fichier *.stl en fichier *.geo.
  \begin{itemize}

Mercredi 19 Octobre

- Débogage de l'application web,
- Début de la documentation de l'application,
- Discussion à propos de la création du serveur,
- Répartition des tâches sur GitHub.

Mercredi 26 Octobre

- Création d'une sphère sur chaque sommet d'une géométrie afin de pouvoir les sélectionner,
- Amélioration de la modélisation des géométries,
- Possibilité de changer l'angle de la caméra en appuyant sur des boutons,
- On a commencé à suivre un tutoriel à propos de flask pour créer notre serveur, mais on s'est aperçu que créer notre serveur en nodejs serait plus adapté à notre projet.

Mercredi 09 Novembre

- Suppression d'une géométrie sur l'application,
- Possibilité de sélectionner plusieurs sommets dans le mode conditions limites,
- Tutoriels sur nodejs : envoyer un message du client vers le serveur.

Mercredi 16 Novembre

- Débogage du mode conditions limites,
- Nettoyage de l'application et révision de certains noms de fichiers,
- Tutoriels sur nodejs : envoyer un fichier du client vers le serveur en utilisant websocket.

Du Lundi 28 Novembre au Vendredi 02 Décembre

- Création du serveur nodejs,
- Exportation de la géométrie créée via notre application web au format *.stl vers notre serveur,

- Lancer notre script python nécessaire à la conversion du *.stl en *.geo sur le serveur,
- Lancer le logiciel GMSH en ligne de commandes sur le serveur
- Envoyer le maillage vers notre application et l'afficher,
- Création de la classe "myMesh" pour stocker la géométrie, son maillage et ses conditions limites.
- Récupérer le numéro de la face sélectionnée,
- Récupérer les nœuds appartenant à la face sélectionnée à l'aide d'un script python.

Du Lundi 05 Décembre au Vendredi 09 Décembre

- Débogage du script python,
- Sélection de plusieurs faces et nœuds
- Création du menu "conditions limites",
- Envoie des conditions limites au serveur,
- Installation de GiD_Post et documentation sur le format HDF5

2) Organisation du groupe

Pour s'organiser au mieux durant notre projet, nous avons utilisé l'outils GitHub. Il s'agit d'un hébergeur de projets, qui permet à des millions de développeurs de travailler efficacement ensemble sur un même projet et de le partager sur internet.

On peut ainsi héberger le dossier de l'application sur GitHub, et à chaque modification on envoie le changement. Cela permet de sauvegarder le projet sur internet afin d'éviter les pertes de données, de travailler à plusieurs sur un projet même à distance et de voir l'avancement du projet.

On a également utilisé GitHub afin de se répartir les tâches. En effet, l'outils GitHub nous permet de créer des tâches, appelées issues (Image 14). Ainsi, à chaque fois qu'on faisait une réunion de projet, on listait les différentes tâches à réaliser pour ensuite les créer sur GitHub. Lorsque l'un d'entre nous avait fini son travail en cours, il pouvait aller sur GitHub, choisir une issue parmi celles qu'on a listé, et s'y assigner pour prévenir les autres membres du groupe qu'il a décidé de travailler sur cette partie afin d'éviter que tout le monde ne travaille sur la même fonctionnalité.

Pour connaître l'état d'avancement de chaque tâche, on a créé les labels suivants : invalid (disponible), doing (en cours de développement) et ready (prête à être envoyé sur le git). On a également créé des labels afin de savoir si la tâche à effectuer faisait partie de l'application, du serveur ou de la documentation. Quand un membre du groupe avait réalisé la tâche sur laquelle il s'était assigné, il indiquait le label "ready" sur cette tâche et envoyait l'application modifié sur le git.

Il s'agit d'une manière efficace de travailler en groupe, qui se rapproche un peu de la méthode agile en gestion de projet.

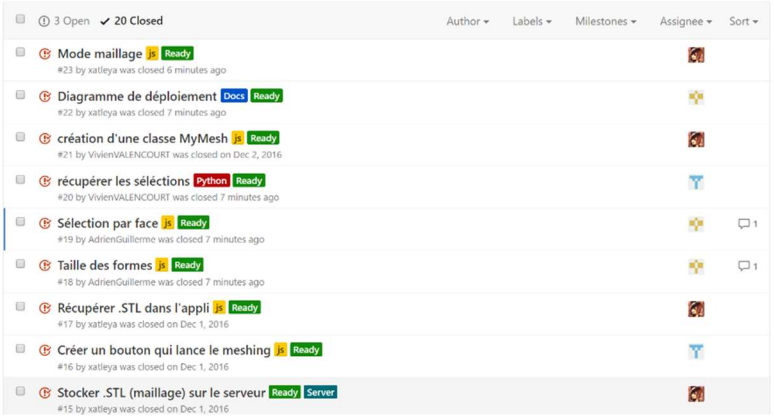

Image 14 : Illustrations des « issues » sur GitHub

3) Structure du dossier

Le dossier de l'application est structuré de la manière suivante :

- Le dossier Build : Contient les sources de la librairie Three.js
- Le dossier Javascript : Contient nos fichiers javascript :
    - createAndMoveMode : le mode création et mouvement d'un géométrie
    - main : l'initialisation de l'application et du menu
    - myMesh : la classe myMesh expliqué plus tôt
    - servClient : le mode "exportation" qui envoie des données au serveur et récupère des données du serveur
    - sizeMode : le mode changement de taille
    - vertexMode : le mode "conditions limites" qui permet de sélectionner des sommets ou des faces
- Le dossier js : Contient toutes les extensions de la librairie three.js, socket.io ainsi que dat0gui (pour le menu)
- Le dossier Temp : Contient les géométries exportées au format *.stl à partir de l'application web
- Le dossier Python : Contient nos scripts python (conversion du *.stl en *.geo et récupération des noeux sur une face)
- Le dossier node_modules : Contient des fonctionnalités supplémentaires pour node js
- Les fichiers numérotés en *.geo sont les géométries converties par notre script python
- Les fichiers numérotés en *.stl sont les maillages des géométries en *.geo
- Le fichier server.js est le fichier qu'il faut spécifier en lançant le serveur avec nodejs
- Le fichier index.html est le fichier qu'il faut lancer pour accéder à l'interface de notre application web

V) Bilan

1) Les difficultés

Ce projet s'est, dans l'ensemble, plutôt bien déroulé, mais cela ne veut pas dire qu'aucune difficulté n'a été rencontrée. En effet, les plus grandes difficultés que nous avons rencontrées sont :

La familiarisation avec chaque outil utilisé, que ce soit un programme, une librairie faisant partie d'un langage de programmation ou même les langages de programmation eux-mêmes que nous n'avions pas encore forcément manipulé.

Ensuite, nous avons aussi eu des difficultés au niveau des liaisons entre les différents programmes. En effet, dans le but d'automatiser certaines parties de notre application, il fallait permettre une communication entre les différents programmes que nous avons développés, pour qu'ils puissent s'échanger des données, mais aussi exécuter un programme au bon moment pour que le résultat puisse ensuite être utilisé.

Enfin, il y a eu un léger manque de temps pour pouvoir compléter la liaison avec ATILA et ainsi effectuer les calculs sur les géométries créées dans notre application. Il ne manque en fait qu'une partie à développer. Celle-ci est la liaison avec ATILA, et plus précisément, l'écriture du maillage dans le format HDF5, sous la même forme que les maillages utilisés par GiD (en utilisant la bibliothèque GiDpost).

2) Les succès

Comme dit ci-dessus, ce programme s'est plutôt bien déroulé et malgré le fait qu'il ne soit pas totalement abouti. Malgré la communication manquante entre le serveur et ATILA, nous avons développé tous les outils nécessaires à l'affichage du résultat dans le navigateur, à savoir l'affichage d'un maillage dans le navigateur, la possibilité de mettre un gradient de couleur sur une géométrie, mais aussi la réception de données, envoyées par le serveur, à destination du navigateur.

De plus, au niveau de l'application, nous avons développé une base solide pour pouvoir s'en servir avec ATILA.

Ensuite, notre gestion de projet fut plutôt une réussite. Nous avons appris à utiliser GitHub et quelques-unes de ses fonctionnalités pour notre travail, afin de se répartir les tâches, ainsi que d'être toujours au courant de l'avancement du projet.

Conclusion

Le bilan de ce projet est globalement très positif. En effet, malgré les difficultés rencontrées nous avons toujours réussi à avancer sur ce projet pour finalement présenter un résultat satisfaisant.

Sur le point de vue de l'apprentissage, ce projet était réellement intéressant. En effet, il était largement basé sur de la programmation, ce qui entre autres, nous plaisait, mais aussi il nous a permis d'utiliser de nombreuses technologies, que nous ne connaissions pas forcément. Par exemple, le serveur était quelque chose de totalement nouveau. De plus, les liaisons entre ces différentes technologies étaient quelque chose de totalement inconnu pour nous, ce qui enrichit d'autant plus nos connaissances.

Enfin, du point de vue humain, notre groupe a su mener à bien ce projet. Malgré les difficultés rencontrées, nous avons su garder notre motivation et continuer à améliorer ce projet jusqu'à la fin de celui-ci. De plus, le bon déroulement de ce projet peut aussi s'expliquer par la bonne entente et la rigueur au sein de notre groupe, en effet, nous avions décidé de venir à des horaires fixes pour ce projet et nous avons respecté cette règle jusqu'au bout.

Pour conclure, ce projet a été plus qu'enrichissant pour notre expérience professionnelle, nous avons vraiment appris beaucoup de choses en seulement quelques semaines, tout cela pour un résultat très satisfaisant.

Bibliographie

GiDpost. (s.d.). Récupéré sur http://www-opale.inrialpes.fr/Aerochina/gidpost/doc/gidpost.html

Gmsh 2.16. (s.d.). Récupéré sur GMSH: http://gmsh.info/doc/texinfo/gmsh.html#Non_002dinteractive-mode

Kielbasiewicz, N. (2013, octobre 23). *Introduction `a Gmsh*. Récupéré sur http://perso.ensta-paristech.fr/~kielbasi/docs/gmsh.pdf

mrdoob. (2010). *three.js*. Récupéré sur three.js: https://threejs.org/

Premiers pas avec WebSocket et Node.js (et Socket.io). (s.d.). Récupéré sur scriptol: http://www.scriptol.fr/javascript/nodejs-socket.php

S.L.Mouradian, A. A. (s.d.). *A Gmsh tutorial*. Récupéré sur https://albertsk.files.wordpress.com/2012/12/gmsh_tutorial.pdf