RAPPORT DE STAGE DE LICENCE 3 INFORMATIQUE
ISTIC, UFR Informatique & Electronique – Université de Rennes
Année universitaire 2025–2026

Conception d'un prototype de simulation par éléments finis

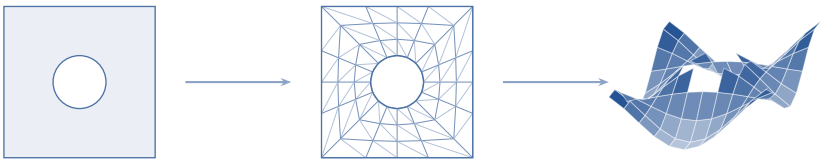
Logiciel I4Sim.jl

Organisme d'accueil
Inria Rennes – Bretagne Atlantique
Équipe-projet **I4S** (Inference for Structures)

Stagiaire : Ebnou ABDOUM

Dates du stage : 4 mai – 26 juin 2026

Encadrement : Christophe DROZ*
Pierre LAGUE†



* https://team.inria.fr/i4s/team-members/christophe-droz/

† https://team.inria.fr/i4s/team-members/pierre-lague/

Sommaire

Résumé

Remerciements. Je remercie Christophe Droz de m'avoir proposé ce sujet et de m'avoir initié à un domaine qui m'était étranger, Pierre Lague pour ses conseils sur le code et l'architecture, ainsi que l'ensemble de l'équipe I4S pour son accueil et cette opportunité. Merci enfin à l'UFR ISTIC de l'Université de Rennes, dont la formation m'a donné les bases pour aborder ce stage.

Le présent rapport rend compte du stage de fin de Licence 3 Informatique que j'ai effectué du 4 mai au 26 juin 2026 à Inria Rennes, au sein de l'équipe-projet I4S (Inference for Structures), sous la supervision de Christophe Droz et Pierre Lague. Ce stage avait pour objet la conception et le prototypage d'une interface graphique modulaire intégrant, dans un environnement unifié, les étapes d'une chaîne de simulation par éléments finis : création de géométrie, génération de maillage, attribution des matériaux, assemblage des matrices, résolution numérique (analyse modale) et visualisation des résultats.

En synthèse, le prototype développé (I4Sim.jl) couvre aujourd'hui l'ensemble de cette chaîne. Son architecture, organisée en couches et pensée générique, a été conçue pour accueillir d'autres analyses (ondulatoire, harmonique, transitoire…) sans être réécrite. Ce rapport en présente le contexte, les fondements scientifiques, les choix de conception et un bilan complet, le stage s'achevant avec la rédaction de ces lignes.

Ce sujet n'avait pas été conçu pour moi : il s'agissait d'une proposition ouverte de l'équipe, à laquelle j'ai postulé. Ce qui m'a décidé, c'est précisément ce qu'il avait d'inconnu : un langage (Julia) que je n'avais jamais pratiqué et un domaine (la dynamique des structures) qui ne faisait pas partie de mon cursus. Lors de l'entretien préalable avec M. Droz, une démonstration en temps réel des simulations menées par l'équipe m'avait intrigué ; c'est cette curiosité (découvrir le monde de la recherche, même sans y mener moi-même de travaux, et me confronter à l'inconnu) qui m'a poussé à candidater.

Sur le plan méthodologique, j'ai adopté une démarche itérative inspirée des méthodes agiles pratiquées lors du projet de groupe de l'an dernier en génie logiciel : des itérations courtes, ponctuées d'une démonstration hebdomadaire et d'une rétrospective. Cette discipline s'est révélée adaptée à un sujet exploratoire où ni les choix techniques ni les fonctionnalités finales n'étaient figés au départ.

Le rapport présente successivement le contexte d'accueil (section 1), la problématique et les tâches confiées (section 2), les notions scientifiques que j'ai dû acquérir (section 3), le travail réalisé et ses choix de conception (section 4), puis un bilan et des perspectives (section 5) ; quelques captures de l'interface figurent en annexe.

1. Contexte du stage

1.1. Inria Rennes – Bretagne Atlantique

Inria (Institut national de recherche en sciences et technologies du numérique) est un établissement public dédié à la recherche en informatique et en mathématiques appliquées. Le centre Inria Rennes – Bretagne Atlantique, sur le campus de Beaulieu, héberge une trentaine d'équipes-projets en partenariat étroit avec l'IRISA, sur des thématiques allant de la vérification formelle à la modélisation des phénomènes physiques. C'est dans ce dernier domaine que s'inscrit mon stage : l'équipe d'accueil emploie des outils numériques pour étudier le comportement dynamique de structures mécaniques, à la frontière entre informatique scientifique, calcul intensif et physique appliquée.

1.2. L'équipe-projet I4S

L'équipe I4S (Inference for Structures), commune à Inria et à l'Université Gustave Eiffel, réunit une quinzaine de personnes — chercheurs, ingénieurs, doctorants et stagiaires. Ses travaux portent sur la surveillance de l'état des structures (Structural Health Monitoring) et l'évaluation non destructive des matériaux : détecter, localiser et caractériser un endommagement à partir de mesures vibratoires, sans démonter la structure. Les applications vont de la surveillance d'ouvrages de génie civil (ponts, éoliennes…) au contrôle de pièces aéronautiques ou de matériaux composites.

Ces méthodes combinent mesures expérimentales et modèles numériques. Les travaux récents de l'équipe s'attaquent en particulier à la complexité croissante des modèles induite par les matériaux composites, les assemblages architecturés et les métamatériaux — des structures à géométrie périodique dont les propriétés vibratoires remarquables (bandes interdites, guidage d'ondes) découlent de la répétition d'une cellule unitaire [1]. Pour en réduire le coût de calcul, l'équipe développe des méthodes d'éléments finis ondulatoires fondées sur la théorie de Bloch, qui ne maillent qu'une cellule. Toutes reposent, en amont, sur la même chaîne classique : décrire une géométrie, la mailler, lui attribuer des matériaux, résoudre, visualiser. C'est précisément cette chaîne, aujourd'hui éclatée entre plusieurs logiciels, que mon stage visait à réunir dans une interface unique.

1.3. Découvrir le monde de la recherche

Au-delà de la production logicielle, ce stage a été l'occasion de découvrir le fonctionnement d'une équipe de recherche et la diversité de ses travaux : l'autonomie laissée à chacun, le rythme des points hebdomadaires, la place de la démonstration et de la critique. J'ai trouvé particulièrement stimulante la pluridisciplinarité de l'équipe, où mécaniciens, physiciens, informaticiens et mathématiciens travaillent ensemble sous un même cap ; Des journées d'échange, notamment avec la partie de l'équipe basée à Nantes, m'ont permis de découvrir cet environnement et d'en apprécier le côté humain.

2. Problématique et tâches confiées

2.1. Le domaine d'intervention

Mon stage se situe à l'intersection de trois domaines : le développement logiciel, l'informatique scientifique et la dynamique des structures côté métier. Mon travail a principalement consisté à produire du logiciel (écrire, organiser et tester un prototype fonctionnel).

L'application doit guider l'utilisateur à travers les cinq étapes d'une simulation par éléments finis, quelle que soit l'analyse finalement demandée ; sans entrer dans le détail (réservé à la section 4), chacune se résume ainsi :

1. **Géométrie** — décrire la forme à étudier ;
2. **Maillage** — la discrétiser en éléments tétraédriques ;
3. **Matériaux et FEM** — affecter les propriétés mécaniques à chaque partie, puis assembler les matrices du problème par la méthode des éléments finis (FEM, de l'anglais **Finite Element Method**) ;
4. **Analyse** — résoudre le problème ainsi posé ;
5. **Visualisation** — explorer les résultats en 3D de façon interactive.

En dynamique des structures, l'équipe mène de nombreuses analyses, dont l'analyse **modale** (fréquences et déformées propres), l'analyse **harmonique** (réponse à une excitation sinusoïdale), l'analyse **transitoire** (réponse au cours du temps) et l'analyse **ondulatoire** (propagation d'ondes dans les structures périodiques). Toutes partagent les mêmes étapes amont (géométrie, maillage, matériaux) et ne se distinguent qu'au moment de la résolution. L'enjeu était donc de concevoir une interface assez **générique** pour les accueillir toutes, en chaînant proprement les étapes et en interdisant l'accès à une étape tant que la précédente n'a pas été validée.

S'agissant d'un premier prototype, je me suis concentré, sur le conseil de M. Droz, sur l'analyse **modale** : c'est la plus simple à appréhender.

2.2. La situation initiale

Le projet ne partait pas tout à fait de zéro : les outils et les bibliothèques existent, un pour chaque étape. Mais il n'y avait, au sein de l'équipe, ni base de code commune ni procédure standardisée — chacun assemblait les briques à sa manière. Comme le rappelle le sujet, l'équipe utilise une suite de modules experts : la génération de géométries (FreeCAD), le maillage (Gmsh), la résolution numérique (Julia), l'analyse et la visualisation (ParaView [7]). À mon arrivée, ces outils étaient employés au cas par cas, combinés à la main : scripts de modélisation, Gmsh en ligne de commande (ou via son API), scripts Julia individuels pour l'assemblage et la résolution, puis ParaView pour la visualisation.

Cet enchaînement, maîtrisé par les chercheurs, est en effet rebutant pour un nouvel arrivant ou un utilisateur occasionnel : il faut connaître les conventions de chaque outil, jongler avec plusieurs formats de fichiers et ne pas se tromper dans l'ordre des opérations. Surtout, il laisse passer des **erreurs silencieuses** ... L'équipe souhaitait donc, selon les termes du sujet, « une plateforme centralisée et intuitive » qui canalise l'utilisateur, affiche à chaque étape les informations critiques et reste robuste en cas de données incomplètes.

2.3. Les tâches demandées

Le sujet fixait trois exigences principales. **Concevoir une architecture modulaire** : une interface organisée autour d'onglets correspondant aux étapes (figure 1), chaque module pouvant évoluer indépendamment. **Respecter les dépendances logiques**, avec un verrouillage actif des étapes aval tant que les pré-requis amont ne sont pas remplis. **Produire un prototype maintenable et documenté**, en s'appuyant sur des principes de génie logiciel (conception modulaire, documentation rigoureuse). S'y ajoutait une exigence de **robustesse** : l'application ne doit pas s'interrompre face à une géométrie incohérente ou à des données incomplètes, mais signaler clairement le problème.

FIGURE 1 – L'interface est organisée en onglets, un par étape (architecture modulaire). Les étapes aval restent **verrouillées** (en gris) tant que la précédente n'a pas produit son résultat.

Sur le plan technique, le sujet suggérait d'explorer `Gtk.jl` pour l'interface, tout en laissant la liberté de retenir un outil « selon sa simplicité et sa fonctionnalité » — liberté dont j'ai fait usage, comme on le verra.

Traduites en termes d'ingénierie, ces exigences se ramènent à quelques problématiques qui ont structuré le travail : garantir la cohérence du pipeline (verrouillage et invalidation des étapes aval), offrir une visualisation 3D interactive et robuste sans installation cliente, traiter des structures réalistes (multi-matériaux, périodicité) et assurer la reproductibilité des calculs. La section 4 y répond, problème par problème.

3. Notions scientifiques mobilisées

La principale difficulté du stage n'a pas été informatique mais conceptuelle : implémenter une analyse modale sans savoir, au départ, ce que représentaient les matrices K, M ou les vecteurs propres φ. C'est M. Droz qui m'a expliqué la physique nécessaire pour juger mes résultats ; je me suis aussi appuyé sur ses travaux [1] et sur la documentation de Ferrite [2]. Cette section en résume l'essentiel, qui éclaire les choix décrits plus loin.

3.1. Du solide continu au modèle éléments finis

Une structure élastique est un milieu continu, régi par des équations aux dérivées partielles que l'on ne sait pas résoudre exactement sur une géométrie quelconque. La méthode des éléments finis (FEM) discrétise le solide en petits éléments (ici des tétraèdres) reliés par leurs sommets, les nœuds ; le champ de déplacement est alors approché par interpolation à partir des déplacements nodaux, soit trois inconnues par nœud (selon x, y, z), les degrés de liberté (DDL).

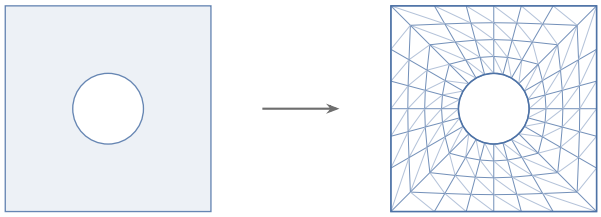


FIGURE 2 – Discrétisation d'une plaque trouée : le solide continu (à gauche) est remplacé par un maillage (à droite) dont les sommets, les nœuds, portent les degrés de liberté. Le principe est identique en trois dimensions, avec des tétraèdres.

En sommant la contribution de chaque élément, Ferrite [2] assemble deux grandes matrices creuses : la matrice de raideur K, qui traduit l'élasticité du matériau, et la matrice de masse M, qui traduit son inertie. Sur chaque tétraèdre, Ferrite calcule par quadrature de Gauss une petite contribution locale (à partir de la loi de Hooke $\sigma = \lambda\,\mathrm{tr}(\varepsilon)\,I + 2\mu\,\varepsilon$, qui relie contraintes et déformations via les coefficients de Lamé λ et μ (déduits du module d'Young E et du coefficient de Poisson ν)) puis additionne ces contributions dans les matrices globales. Les éléments employés sont des tétraèdres linéaires (P1), à quatre nœuds.

3.2. Analyse modale et déformées propres

L'analyse modale cherche les vibrations libres de la structure : les façons dont elle oscille naturellement lorsqu'on l'écarte de sa position de repos, sans effort extérieur. Ces vibrations sont les solutions (ω, φ) du problème aux valeurs propres généralisé

$$K\,\varphi = \omega^2\,M\,\varphi, \tag{1}$$

où $\omega = 2\pi f$ est la pulsation propre, f la fréquence propre (en hertz) et φ la déformée propre associée : à la fréquence f, la structure se déforme suivant la forme φ (fig. 3).

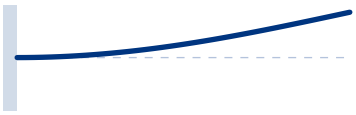


FIGURE 3 – Premier mode de flexion d'une poutre encastrée : la forme modale (trait plein) écarte la structure de sa position de repos (pointillés). C'est cette forme que l'application restitue en 3D et anime.

La fréquence se déduit de la valeur propre $\lambda = \omega^2$ par

$$f = \frac{\sqrt{\max(\lambda,\,0)}}{2\pi}\ \text{[Hz]}, \tag{2}$$

le $\max(\cdot, 0)$ absorbant les valeurs propres très légèrement négatives dues aux arrondis de l'assemblage. Un point d'unités mérite d'être signalé : les maillages étant en millimètres, les matériaux sont convertis dans un système cohérent (mm, MPa, tonne, seconde) pour que f sorte directement en hertz.

Sur une structure tridimensionnelle, les modes prennent des formes plus riches que la simple flexion : torsion, modes de membrane, modes locaux. La figure 4 montre un mode d'une plaque maintenue sur ses bords : le centre se déplace le plus, les bords restent fixes, et toute la structure oscille en phase autour de sa position de repos.

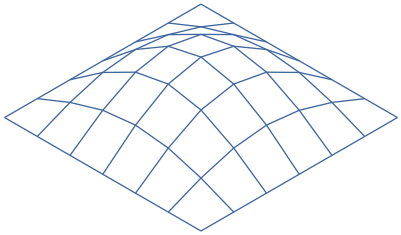


FIGURE 4 – Forme d'un mode de vibration d'une plaque carrée maintenue sur ses bords, vue sur son maillage. Le centre se déplace le plus, les bords restent fixes.

3.3. Cellule unitaire et structures périodiques

Comme l'équipe s'intéresse aux structures périodiques, j'ai ajouté la prise en charge des conditions de périodicité. L'idée physique : pour une structure faite de la répétition d'une même cellule, il suffit d'en étudier une seule, en imposant que ses faces opposées se déforment de façon identique à une translation près — la condition $u(x+d) = u(x)$, dite maître/esclave, où d est le vecteur de périodicité.

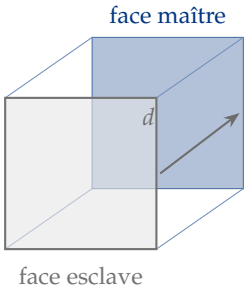


FIGURE 5 – Périodicité : on n'étudie qu'une cellule en imposant que sa face « esclave » se déforme comme sa face « maître », à la translation d près. La structure complète se déduit par répétition de la cellule.

En pratique, cette contrainte permet de n'exprimer le problème que sur les degrés de liberté réellement indépendants : on **condense** les matrices K et M sur ces seuls DDL libres avant de résoudre, puis on reconstruit la déformée sur toute la cellule. Ce mécanisme relie directement le prototype aux méthodes ondulatoires de l'équipe [1].

4. Travail réalisé

Dans cette section je le présente le travail dans l'ordre où il s'est construit : après la méthode et les choix techniques (la pile, puis l'architecture), je décris les **deux itérations** principales (un squelette de bout en bout, puis son enrichissement) avant de revenir sur la robustesse, les tests et les finitions.

4.1. Une démarche itérative

La démarche itérative m'a permis d'avancer par incréments éprouvés au fur et à mesure, et d'abandonner ce qui ne tenait pas ses promesses.

Concrètement, le stage s'est déroulé en quatre temps. Les **deux premières semaines** ont été consacrées à l'appropriation : prise en main de Julia, des outils de la chaîne et des notions de dynamique des structures, aussi au choix de la pile technique et l'architecture. Les **deux semaines suivantes** ont produit un premier livrable simple mais fonctionnel de bout en bout, du script géométrique jusqu'à l'affichage d'un mode. Les **trois semaines** d'après ont été dédiées à l'enrichissement et aux ajustements : support multi-matériaux, périodicité, moteur de visualisation, raffinements successifs. La **dernière semaine**, enfin, a servi à la finalisation et à la rédaction de la documentation complète — un guide pour l'utilisateur et une documentation technique pour les développeurs (une documentation d'étape avait d'ailleurs déjà été rédigée à l'issue de la première itération).

4.2. Le choix de la pile technique

Pistes écartées. Le sujet suggérait `Gtk.jl`, mais son installation sur macOS s'est révélée instable et, surtout, GTK n'offre aucun composant prêt à l'emploi pour la visualisation 3D interactive, pourtant centrale ici. J'ai aussi écarté **FreeCAD** comme moteur de modélisation : trop lourd (entre 1,5 et 2 Go selon le système d'exploitation) et sans interface Julia, il aurait imposé une dépendance encombrante et peu portable.

Le choix décisif. Après avoir testé plusieurs bibliothèques, j'ai retenu mes briques selon un critère directeur : la **maintenabilité** de ce sur quoi reposerait toute l'application. Le choix le plus structurant a été l'interface **web** : l'application tourne localement dans le navigateur (via **Dash**) et délègue le rendu 3D à **Plotly**, qui s'appuie sur **WebGL**, accéléré par la carte graphique. On obtient ainsi un affichage fluide de maillages de plusieurs milliers de triangles, identique d'une machine à l'autre.

La pile retenue. Les dépendances, déclarées dans `Project.toml` et installées automatiquement par Julia [6], sont les suivantes :

— **Dash** [4] **+ DashBootstrapComponents** — le framework web réactif (des **callbacks** reliant entrées et sorties) et ses composants d'interface (onglets, boutons, fenêtres modales, grille responsive).

— **PlotlyBase** (Plotly [5]) — le rendu 3D dans le navigateur via WebGL, construit côté serveur et transmis au client ; c'est lui qui assure la robustesse et la fluidité de l'affichage.

— **Gmsh** [3] — la géométrie et le maillage, avec le noyau OpenCASCADE embarqué (environ 50 Mo, API entièrement Julia).

— **Ferrite** [2] **+ FerriteGmsh** — l'assemblage par éléments finis ; FerriteGmsh lit nativement les maillages Gmsh, et Ferrite gère l'affectation des degrés de liberté, la quadrature et les

matrices creuses.

— **Arpack** — un solveur aux valeurs propres, employé dans un premier temps puis remplacé par le solveur dense de la bibliothèque standard (voir section 4.5).

— **Documenter.jl** — la génération de la documentation (guide utilisateur et documentation technique) à partir du code et de fichiers Markdown.

S'y ajoutent des modules de la bibliothèque standard de Julia : `LinearAlgebra` (le solveur dense, qui repose sur LAPACK) et `Serialization` (la persistance des résultats sur disque).

4.3. L'architecture : trois couches et un état partagé

Avant de dérouler les itérations, voici l'ossature qui relie les cinq pages. Deux regards la décrivent et se complètent : comment le **code** est organisé et comment les **données** y circulent.

Trois couches. Le code est séparé en trois couches superposées, chacune ne dépendant que de la suivante (figure 6). Tout en bas, le **backend** fait le calcul pur : décrire une géométrie, la mailler, assembler K et M, résoudre, lire et écrire les résultats. Il ignore totalement qu'une interface existe, ce qui permet de le tester seul, en quelques lignes, sans lancer de navigateur. Tout en haut, le **frontend** ne fait que présenter : il dessine les pages, capte les clics et délègue le rendu 3D à un module partagé. Entre les deux, une couche **façade** sert de couture : un module par étape, dont chaque fonction reçoit l'état, appelle le backend et y range le résultat. C'est elle, et elle seule, qui a le droit de modifier l'état — l'interface ne touche jamais directement aux matrices ni aux maillages. Cette discipline a un coût en lignes, mais elle isole les mutations en un seul endroit et rend l'ensemble bien plus simple à faire évoluer.

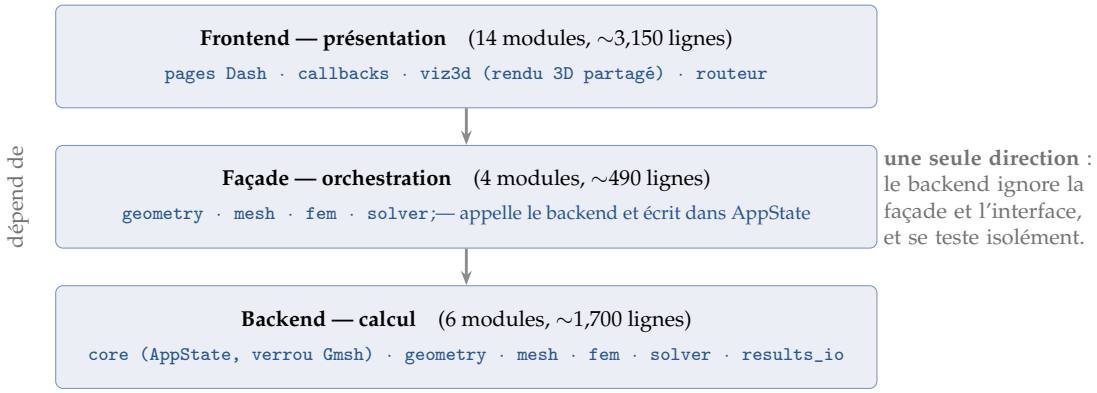


FIGURE 6 – Les trois couches du code et le sens de leur dépendance.

Un fil de données. Vue de l'autre côté, l'application est une circulation : chaque page consomme ce que la précédente a produit (figure 7). La géométrie produit un script `.geo`, le maillage un fichier `.msh`, l'assemblage les matrices K et M, le solveur les modes, que la visualisation affiche. C'est ce fil de données qui fait l'unité de l'outil.

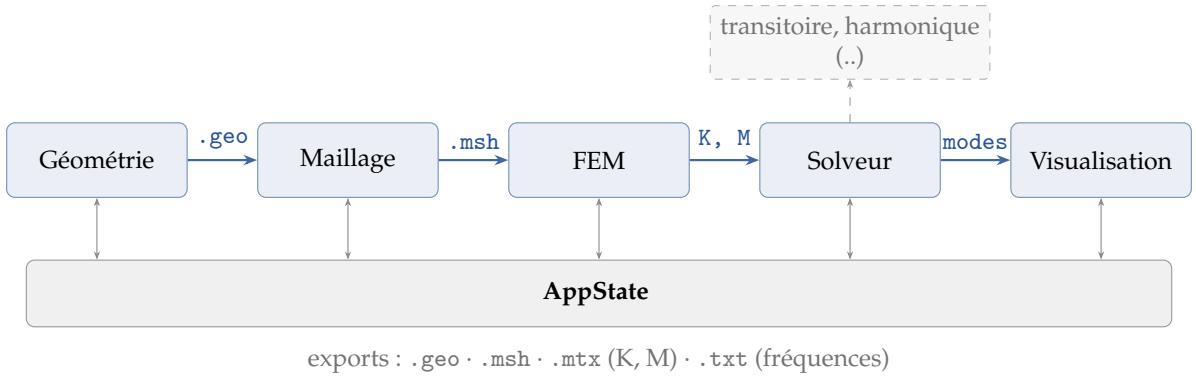


FIGURE 7 – L'orchestrateur. Chaque page consomme l'artefact produit par la précédente

Un état partagé. Plutôt que de faire transiter ces données de page en page, je les centralise dans un objet unique, `AppState`, créé au démarrage et transmis à chaque callback. Ce choix apporte trois garanties. D'abord l'**invalidation en cascade** : régénérer la géométrie efface aussitôt maillage, matrices et modes, si bien qu'on ne visualise jamais des résultats calculés sur une ancienne forme. Ensuite le **verrouillage** : tant qu'une page n'a pas produit son artefact, la barre de navigation interdit l'accès à la suivante. Enfin, la visualisation est **découplée** : les résultats sont écrits sur disque puis rechargés, de sorte qu'on peut quitter l'application et la relancer sans tout recalculer. Les trois reposent sur la même couture façade décrite plus haut.

4.4. Première itération : un squelette de bout en bout

L'objectif de la première itération était binaire : partir d'une interface vierge et obtenir, en enchaînant les cinq pages sur une géométrie simple, l'affichage d'un mode propre. La stratégie a été d'implémenter une version **minimale mais réelle** de chaque page, quitte à tout reprendre ensuite.

Géométrie. La page reposait sur un éditeur où l'on saisit (ou importe) un fichier `.geo` décrivant la forme dans la syntaxe de Gmsh. Un bouton déclenche la génération et un aperçu 3D s'affiche aussitôt ; pour rester réactif, cet aperçu ne maille que la **peau** de l'objet (maillage surfacique), calculé en quelques centaines de millisecondes là où un maillage volumique en demande plusieurs.

Maillage. Tout était délégué à Gmsh, piloté par un **unique** paramètre — la taille caractéristique des éléments. Un aperçu en fil de fer, accompagné du décompte des nœuds et des tétraèdres, permettait de juger la finesse d'un coup d'œil.

Matériaux et FEM. L'assemblage s'appuyait sur Ferrite avec un **seul** matériau appliqué à toute la pièce — trois paramètres (E, ν, ρ). Ferrite affectait les degrés de liberté, intégrait les matrices élémentaires (tétraèdres P1, coefficients de Lamé, voir section 3) et assemblait K et M creuses.

Solveur. Le problème aux valeurs propres était confié à `Arpack`, à qui l'on demandait les quelques plus petits modes ; les fréquences obtenues étaient listées dans un tableau.

Visualisation. Enfin, le mode sélectionné était affiché en 3D, le maillage **coloré** selon l'amplitude du déplacement — mais la forme elle-même n'était ni déformée ni animée à ce stade, ce qui viendra à l'itération suivante.

Pour tester sans connaître la réponse attendue, M. Droz m'a expliqué la physique sous-jacente : par exemple qu'une structure libre doit présenter six modes à fréquence quasi nulle, ses mouvements de corps rigide, ce que le solveur retrouvait bien. À l'issue de ces deux semaines, le squelette enchaînait les cinq pages de bout en bout : rudimentaire et rigide, mais fonctionnel.

4.5. Seconde itération : enrichir et fiabiliser

La seconde itération a transformé ce squelette en outil réellement utilisable. Elle a repris chacune des cinq pour la fiabiliser et la compléter, en réglant au passage les problèmes les plus instructifs.

Géométrie et maillage. Côté géométrie, l'éditeur a appris à signaler proprement un script `.geo` invalide plutôt que de planter, et il a été enrichi pour guider un utilisateur souvent peu familier de la syntaxe Gmsh : des **fragments de script prêts à insérer** (une boîte, un cylindre, un trou, une déclaration de périodicité…) et une page documentant le **contrat** attendu du fichier — en particulier les conventions de nommage des groupes physiques (domaines, faces périodiques, point fixe) que l'application sait interpréter. Côté maillage, la prise en charge a été étendue aux géométries à **plusieurs domaines**, en veillant à la conformité des éléments à l'interface entre domaines (nœuds partagés, sans recouvrement) ; le décompte affiché (nœuds, tétraèdres, nombre de domaines) donne une lecture immédiate de la taille du problème.

Matériaux et assemblage. Le support **multi-matériaux** a été ajouté : chaque sous-domaine reçoit ses propres propriétés (E, ν, ρ), et l'attribution est rendue **explicite** — tant qu'un domaine n'a pas de matériau, l'assemblage reste bloqué et la page le signale. Cette page expose aussi les **ensembles nommés** (volumes, faces, points définis dans le `.geo`) et permet d'exporter K et M au format MatrixMarket pour un usage externe.

Un solveur robuste. `Arpack` échouait à converger sur les cas à plusieurs matériaux ; après plusieurs jours de diagnostic, il a été remplacé par `eigen` de la bibliothèque standard (LAPACK) — plus lent mais inconditionnellement fiable, au prix d'une limite de taille (`MAX_SOLVER_DOF`, fixée à 5 000) assumée pour un prototype. Les matrices sont resymétrisées avant l'appel, et le cas périodique est condensé sur les seuls degrés de liberté libres ($K_r = C^\top K C$) avant résolution. La page distingue les DDL libres des DDL bloqués et indique le temps de calcul.

La périodicité. La périodicité maître/esclave a également été prise en charge. Une première tentative de détection automatique des faces appariées, trop fragile et difficile à généraliser, a été abandonnée au profit d'une simple convention de nommage : l'utilisateur déclare ses faces périodiques dans le script (selon le contrat documenté plus haut), et le solveur applique la condensation correspondante.

Un cœur visuel soigné. Tout le rendu 3D a été centralisé dans un **seul module**, gage d'une apparence cohérente d'une page à l'autre. Surtout, c'est ici que la déformée a réellement pris forme : là où la première itération se contentait de **colorer** le maillage selon l'amplitude, le maillage se **déforme** désormais suivant le mode, et l'animation fait osciller la structure autour de sa position de repos (huit palettes de couleurs, spectre des fréquences en appui). Deux écueils ont demandé du soin : empêcher la scène de "respirer" en figeant les axes, et conserver le point de vue de l'utilisateur pendant l'animation. S'y ajoutent des fonctions d'inspection (mise en évidence des ensembles nommés et des domaines), tandis que l'exécution de scripts Julia fournis par l'utilisateur a été écartée par **sécurité**.

Détails d'intégration. Relier le calcul (Julia) à l'affichage (le navigateur) a réservé quelques subtilités : Gmsh numérote ses nœuds de façon non continue alors que Plotly attend des indices à partir de zéro (d'où une table de conversion), et certaines arêtes doivent être interrompues par une valeur « vide » que le format JSON accepte. De petites choses, mais qui bloquent tant qu'on ne les a pas comprises.

Au terme de cette itération, les cinq pages forment un outil cohérent. L'**annexe** en donne un aperçu complet, page par page, sur une structure de démonstration à deux domaines.

4.6. Robustesse, tests et versionnement

Une catégorie de bugs intermittents (maillages tronqués, fichiers vides, voire arrêts brutaux) a été tracée à un problème de concurrence : Gmsh maintient un état interne global qui n'est pas conçu pour des appels simultanés, ce qui peut arriver lorsque plusieurs callbacks s'exécutent en parallèle. J'ai introduit un verrou global qui sérialise tous les accès à Gmsh, supprimant ces corruptions.

Le code a été suivi sous **Git** (dépôt GitLab de l'équipe), organisé en trois couches et une vingtaine de modules ; les messages de commit documentent les choix au fil de l'eau. Côté validation, j'ai écrit des **tests unitaires** pour chaque module du backend (géométrie, maillage, assemblage, solveur, entrées/sorties), exécutables d'une seule commande, ainsi qu'un **test d'intégration** qui rejoue toute la chaîne sur une géométrie de référence et vérifie les fréquences obtenues. Ces tests automatisés sont complétés par l'épreuve manuelle de l'interface à chaque modification.

4.7. Finitions et expérience utilisateur

Plusieurs améliorations, sans incidence sur les calculs, ont été apportées en fin de stage pour rendre l'outil agréable. L'éditeur de script a reçu une **coloration syntaxique** (réalisée côté serveur) qui distingue mots-clés, nombres et commentaires du langage `.geo` ; purement cosmétique, elle facilite néanmoins la lecture et la correction des scripts. L'interface a été structurée en **panneaux clairs**, une page par étape, avec une barre de navigation qui reflète l'avancement et verrouille les étapes non atteignables. Enfin, le rendu 3D a été homogénéisé d'une vue à l'autre (mêmes couleurs, même caméra, mêmes palettes), de sorte que l'ensemble donne une impression cohérente plutôt qu'un assemblage de pages disparates.

5. Bilan et perspectives

5.1. État du prototype

À l'issue du stage, le prototype couvre la totalité du pipeline, sur ses cinq étapes, chaînées par un verrouillage qui empêche tout saut d'étape — répondant ainsi aux trois exigences du sujet (architecture modulaire, dépendances logiques, prototype maintenable et documenté). Il dépasse même le périmètre initial (persistance des résultats, périodicité, export/import des matrices), au prix d'un choix assumé : un solveur dense borné à 5,000 degrés de liberté et des éléments linéaires, au profit d'un comportement simple et fiable.

5.2. Compétences acquises

Sur le plan technique, j'ai découvert un domaine (les éléments finis) que je n'avais jamais pratiqué, ainsi qu'un nouveau langage de programmation : Julia. Conçu spécifiquement pour le calcul scientifique, Julia s'est révélé très formateur, m'apprenant à concilier une syntaxe de haut niveau avec les exigences de haute performance numérique (compilation à la volée, manipulation optimisée de grandes matrices). Dans ce même élan, j'ai appris à manipuler des bibliothèques scientifiques aux interfaces parfois rugueuses (Gmsh, Ferrite), une bibliothèque de visualisation nouvelle (Plotly), et à diagnostiquer des problèmes numériques (singularités, erreurs d'arrondi, cohérence des unités). Côté génie logiciel, j'ai mis en pratique des principes vus en formation tels que la conception modulaire, la séparation des responsabilités, les tests unitaires, ainsi que la gestion de versions. J'ai enfin pris l'habitude de m'appuyer systématiquement sur la documentation officielle des outils plutôt que sur des recettes toutes faites.

5.3. Perspectives

Plusieurs pistes prolongent ce travail : l'intégration d'un solveur creux (par exemple `KrylovKit.jl`) est la plus immédiate. Actuellement, l'utilisation d'un solveur plein impose de stocker et de traiter tous les coefficients des matrices, y compris les zéros. La consommation mémoire croissant de façon quadratique ($O(N^2)$) et le temps de calcul de façon cubique ($O(N^3)$), cela justifie la limite de sécurité fixée à 5 000 degrés de liberté. Un solveur creux, en n'exploitant que les valeurs non nulles, permettrait de faire sauter ce verrou sur de très gros maillages. D'autres évolutions incluent : le passage à des éléments quadratiques pour une meilleure précision des fréquences ; l'ajout effectif des analyses transitoire et harmonique, déjà permises par l'ossature générique ; et un export ParaView pour les utilisateurs avancés .

5.4. Difficultés et rétrospective

Les principales difficultés ont été la barrière culturelle (traduire des attentes formulées en termes métier en spécifications informatiques), la lenteur du cycle de recompilation en Julia (que `Revise.jl` n'atténue qu'en partie) et le bug de convergence d'Arpack, qui m'a coûté plusieurs jours. Dans l'esprit des rétrospectives agiles, j'en tire trois enseignements. **Ce qui a bien fonctionné** : la démarche itérative et la séparation en couches, qui ont rendu les évolutions successives presque indolores. **Ce qui a moins bien marché** : avoir parfois cherché à anticiper des fonctionnalités futures, ce qui a alourdi le code avant qu'une rétrospective ne me conduise à le simplifier (le solveur en est l'exemple le plus net). La leçon la plus transférable reste de préférer la simplicité éprouvée à la généralité prématurée.

5.5. Conclusion personnelle

Ce stage a été ma première expérience complète de développement logiciel sous contrainte de réalité : une équipe utilisatrice, des exigences précises, une échéance courte, et la nécessité de livrer quelque chose qui fonctionne. J'en retire la conviction que les compétences acquises en formation ne prennent toute leur valeur qu'au contact d'un contexte applicatif qui les met sous tension. Découvrir, au passage, l'organisation d'une équipe de recherche et l'univers des métamatériaux a rendu cette expérience d'autant plus stimulante, et m'a confirmé mon intérêt pour l'informatique scientifique.

Bibliographie

[1] C. Droz, Modèles haute résolution pour l'analyse dynamique ultra-rapide des structures à géométrie périodique, Journées COFREND 2023, Marseille, juin 2023. https://hal.science/hal-04213224

[2] K. Carlsson, F. Ekre et al., Ferrite.jl — a finite element toolbox in Julia. https://ferrite-fem.github.io/Ferrite.jl/stable/

[3] C. Geuzaine et J.-F. Remacle, Gmsh : a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Methods Eng., 79(11), 1309–1331, 2009. https://gmsh.info/

[4] Plotly Technologies Inc., Dash (framework web réactif), documentation Julia : https://dash.plotly.com/julia

[5] Plotly Technologies Inc., Plotly for Julia : https://plotly.com/julia/

[6] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia : A fresh approach to numerical computing, SIAM Review 59(1), 2017. https://julialang.org/

[7] U. Ayachit, The ParaView Guide, Kitware, 2015. https://www.paraview.org/

Annexe — Captures de l'interface

Les captures suivantes illustrent l'interface sur une même structure de démonstration, suivie d'une page à l'autre ; cette structure est décrite en fin d'annexe.

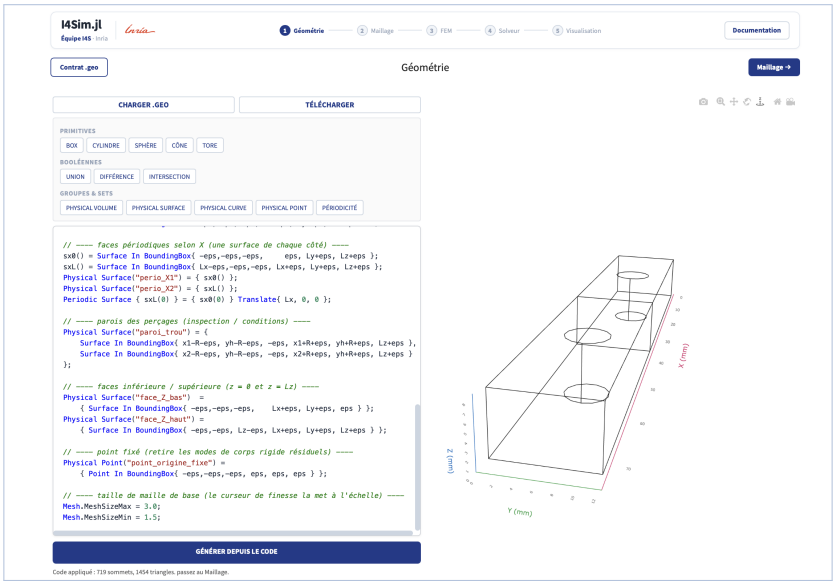


FIGURE 8 – Page **Géométrie** — éditeur de script `.geo` (primitives, snippets, coloration) et aperçu 3D des contours.

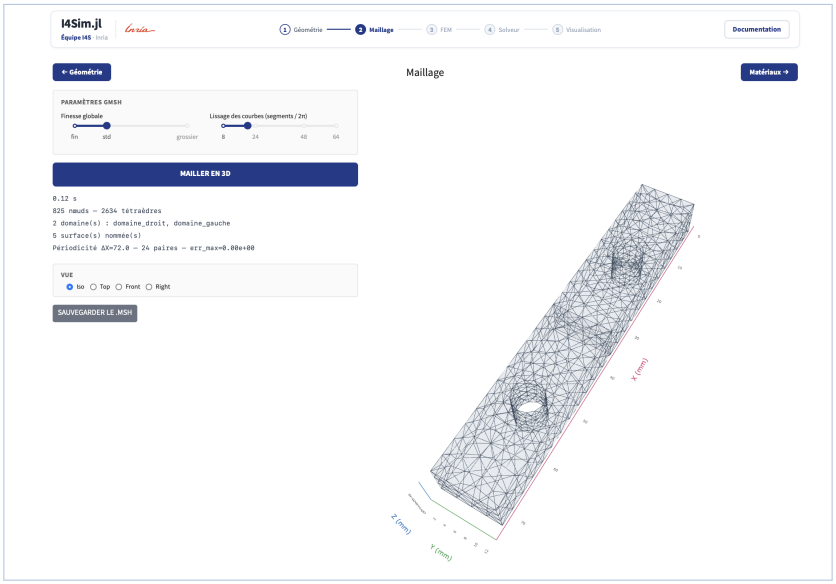


FIGURE 9 – Page **Maillage** — aperçu fil de fer du maillage tétraédrique (825 nœuds, 2 634 tétraèdres).

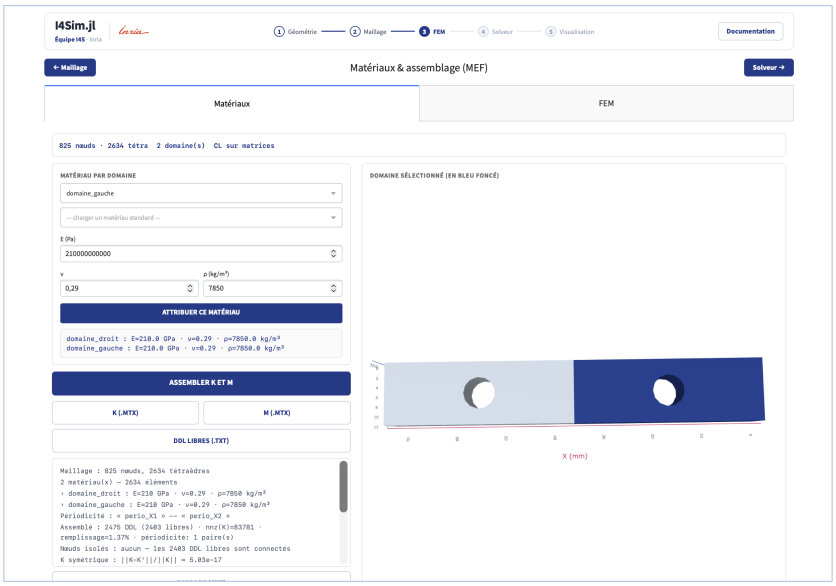


FIGURE 10 – Page **Matériaux & FEM**, onglet Matériaux — un matériau par domaine, domaine sélectionné en bleu foncé.

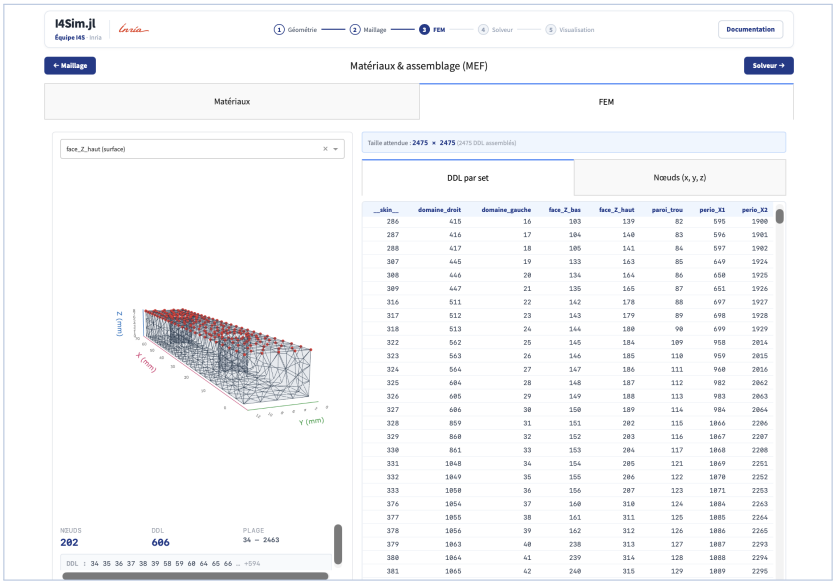


FIGURE 11 – Page **Matériaux & FEM**, onglet FEM — inspection d'un ensemble nommé (nœuds en rouge) et tables des DDL / coordonnées par set, exportables.

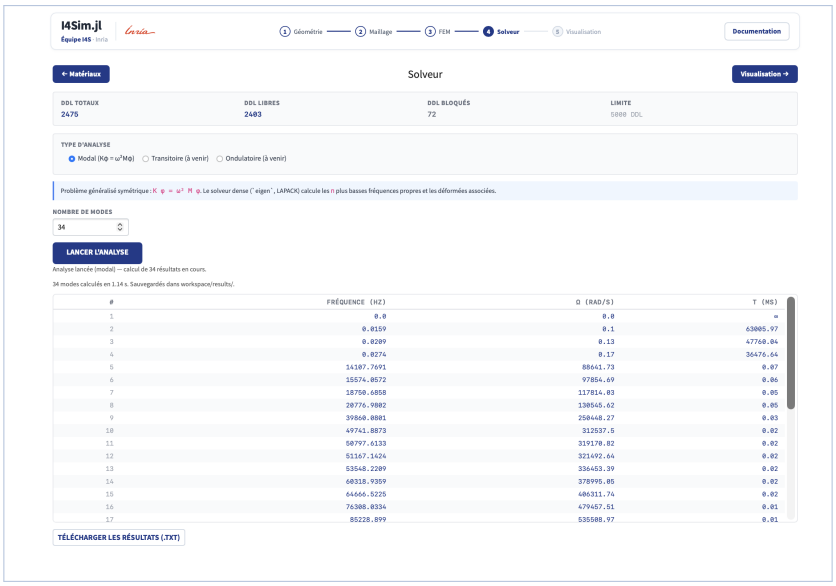


FIGURE 12 – Page **Solveur** — décompte des degrés de liberté et tableau des fréquences propres (34 modes).

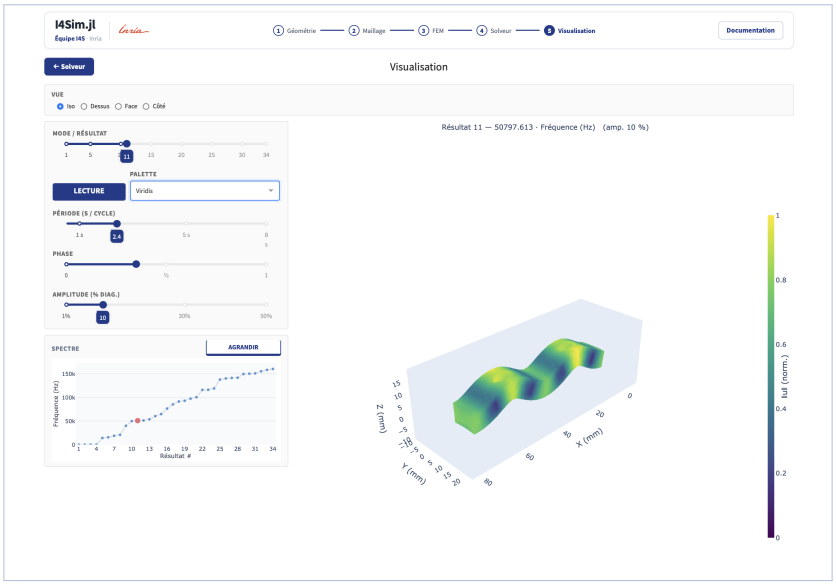


FIGURE 13 – Page **Visualisation** — déformée animée d'un mode (colorée par amplitude) et spectre.

La structure de démonstration. L'exemple est une poutre élancée ($72 \times 12 \times 8$ mm), à deux domaines accolés, percée de deux trous et périodique selon X (faces appariées, point d'origine fixé). Maillée, elle compte 825 nœuds dont le solveur extrait 34 modes en un peu plus d'une seconde : quelques modes quasi nuls, puis les modes de déformation, bien lisibles grâce à l'élancement de la pièce.