École Nationale des Ponts et Chaussées
2018 - 2019

Projet de Fin d'Études

Département Génie Civil et Construction

Mathieu Lerouge
Élève ingénieur

Tuteur professionnel : Dr. Cyril Douthe
Tuteur académique : Dr. Olivier Baverel

Rationalization and optimization of meshes for generating gridshells with nodes congruence

Generation, design and analysis of Xmesh structures

Projet réalisé au sein du Laboratoire Navier
6 / 8 avenue Blaise Pascal - Champs sur Marne
F-77455 MARNE LA VALLEE CEDEX 2

Du 4 février 2019 au 23 août 2019

Acknowledgements

Firstly, I would like to thank my tutors at the *Laboratoire Navier*, Cyril Douthe and Xavier Tellier, for the daily support they provided to me during this six-month internship. Thanks to them, I had the great opportunity to work on two challenging projects of pavilions, both materializing the scientific question developed in this study: the *IASS Xmesh pavilion* and the *CLC Xmesh pavilion*.
I would like to thank Olivier Baverel, for his supervision and his critical opinions about the structure of the IASS pavilion during meeting, as well as Laurent Hauswirth for his help on the geometric plan. I would also like to thank Nicolas Leduc, a doctoral student, as well as Sonia Zerhouni, Guillaume Lami, Thibault Lenart and Alexandre Le Pavec, students from the *Ecole d'Architecture de Paris Malaquais*, for the effort they put into the collaborative design of the IASS pavilion.
I would also like to thank Koliann Mam, Léo Démont, Romain Mesnil and Hocine Delmi for their contribution to the CLC pavilion, respectively on the design of connections, the robotic manufacturing and construction as well as the technical support.
Then, I would like to thank all my colleagues and friends from the department of *Architectured Materials and Structures (MSA)* of the laboratory. I enjoyed working within this team and sharing all the moments of conviviality we had in and outside the office.
Finally, I would like to thank Daniel Piker and Eike Schling for their availability to and the advises they gave to us about some topics of geometric optimization and construction details.

Abstract

With the development and the popularization of computer-aided industrial design software, complex shapes designs have been one of the major trends in architecture in the past thirty years. When designers are working on such *free-form* projects, one of their most important motivations is formal expressiveness. However, designs are also constrained by physical considerations, like structural performance, techniques of fabrication and construction, as well as costs. Designing free-form projects under such constraints turns out to be a difficult challenge for architects, engineers and contractors, that is often not fully anticipated resulting in large additional costs. Therefore, strategies are needed in order to rationalize the geometry of such projects while letting a certain formal freedom to the designers.

One of the most expensive part of a free-form-surface-based design is often the connections between its structural elements. In this study, we propose one strategy of *geometric optimization* for generating *Xmesh* geometries. An Xmesh is a *hexagonal mesh* approximating a smooth target surface and having node congruence. Nodes are categorized in two standard families *flat nodes* and *120-3D nodes*. We describe the theoretical and algorithmic contents of this strategy. Two projects of pavilions, the *IASS Xmesh pavilion* and the *CLC Xmesh pavilion*, will serve as applications to this strategy and their structural behaviour will be study.

Keywords: Architectural geometry, free-form surface, fabrication-aware design, rationalization, optimization

Résumé

Avec le développement et la démocratisation des logiciels de conception assistée par ordinateur, la conception de structures à géométries complexes, dites *free-form*, est devenue l'une des tendances architecturales majeures de ces trente dernières années. L'une des intentions clés des concepteurs de tels projets est l'expressivité formelle. Toutefois, de telles conceptions demeurent inéluctablemet limitées par des considérations physiques, comme ses performances structurelles, les techniques de fabrication de ses éléments, leur construction, sans oublier les divers coûts associés à chacun de ces aspects. Concevoir des projets à géométries complexes en tenant compte de telles contraintes s'avère être un véritable défi pour les architectes, les ingénieurs et les entreprises de réalisation, un défi qui est souvent mal anticipé, entrainant des coûts imprévus significatifs. Des stratégies sont donc nécessaires afin de rationaliser les géométries de tels projets tout en préservant la liberté formelle des concepteurs. Un des aspects les plus coûteux d'un project architectural *free-form* est souvent le design et la fabrication des connexions entre les éléments de structure. Dans cette étude, nous proposons une stratégie d'*optimisation géométrique* pour générer des géométries que nous nommons *Xmesh*. Un Xmesh est un maillage hexagonal, dont la forme est proche d'une surface cible lisse, présentant une certaine redondance nodale. Ses nœuds peuvent être classés en deux familles : des *nœuds plats* et les *nœuds 120-3D*. Nous décrivons le contenu théorique et algorithmique de cette stratégie. Deux projets de pavillons, le *pavillon IASS Xmesh* et le *pavillon CLC Xmesh*, serviront d'applications à cette stratégie. Leur comportement structurel sera également étudié.

Mots-clés : Géométrie constructive, surface libre, design orienté fabrication, rationalisation géométrique, optimisation géométrique

Résumé détaillé

Introduction

Avec le développement et la démocratisation des logiciels de conception assisté par ordinateur (CAO), les réalisations architecturales basées sur des formes complexes - réalisations dites *free-form* - sont devenues relativement courantes dans l'architecture contemporaine. Dans ces projets, les considérations relatives à la géométrie, la matérialité, la structure, la fabrication et la construction sont particulièrement entrelacées. De fait, le résultat final de tels projets repose sur un travail itératif et collaboratif, entre architectes, ingénieurs et entreprises de réalisation, au coeur duquel se trouve la géométrie. En effet, pour les architectes partisans du *free-form*, l'expressivité de la géométrie est l'intention première du projet ; elle est influencée par diverses considérations telles que le volume, la lumière et les fonctions programmatiques. Pour les ingénieurs, la géométrie est le squelette abstrait ou la forme approximative sur laquelle se base la structure ; elle doit être telle que les éléments de structure se comportent correctement et résistent sous chargement. Pour les entreprises de réalisation, la géométrie est cruciale car elle influence la faisabilité du projet ; la géométrie est indirectement contrainte par les techniques, le coût et le temps de la fabrication et de la construction. En considérant la géométrie comme une variable et les intérêts concurrents des différents acteurs comme un ensemble de contraintes, on peut voir le problème du choix de la géométrie du projet comme une sorte de problème d'optimisation sous contraintes multiples. La géométrie choisie doit représenter un bon compromis entre tous les critères architecturaux, techniques et constructifs.
Par conséquent, dès la conception de la géométrie d'un projet *free-form*, les designers doivent être conscients de ces diverses considérations et les intégrer d'une certaine manière dans leur processus de définition et de création de la géométrie, ce qui constitue un véritable challenge.

Au cours des dernières décennies, de nombreuses méthodes mathématiques et algorithmiques ont été mises au point pour relever ce challenge. Elles appartiennent au domaine de la *géométrie constructive*. Ces approches *rationalisent* les géométries en ce sens qu'elles visent à créer des objets géométriques présentant des caractéristiques non triviales, avantageuses sur les plans structurel et constructif, tout en garantissant une certaine liberté formelle aux concepteurs.
L'expérience a montré qu'il peut être particulièrement bénéfiques d'investir du temps afin de rationaliser les nœuds d'un projet *free-form* tel qu'un gridshell. A titre de contre-exemple, dans le cas la couverture de la grande cour du British Museum, les noeuds à l'intersection du gridshell n'ont pas été optimisés de façon à être génériques de telle sorte qu'ils ont dû être découpés individuellement à partir d'une plaque d'acier très épaisse au moyen d'une technologie plasma très couteuse [4].

Par conséquent, lors de la conception d'un gridshell, une stratégie de rationalisation géométrique que nous avons intérêt à mettre en œuvre est de chercher à générer des géométries présentant une forte redondance nodale.

Dans cette étude, nous proposons une telle stratégie qui peut être appliquée spécifiquement aux maillages hexagonaux. Dans le Chapitre 1. *Rationalisation*, nous définissons ou rappelons certaines notions géométriques à la base de notre stratégie. En particulier, nous donnons une définition du *Xmesh*, un cas particulier de maillage pensé pour intégrer des contraintes de fabrication. Dans le Chapitre 2. *Optimisation*, nous décrivons une étape clé de notre stratégie basée sur l'optimisation géométrique. Nous introduisons les outils et présentons des éléments de code que nous utilisons pour générer des maillages X. Dans le Chapitre 3. *Analyse structurelle*, nous examinons le comportement structurel de ces géométries et discutons de leurs avantages et de leurs défauts.

Ce travail est le résultat d'un stage de six mois, au sein du département *Matériaux et Structures Architecturés (MSA)* au *Laboratoire Navier*. Il s'inscrit dans le cadre du projet Impulsion *GAMES*, financé par l'*Université Paris Est*, mené par les chercheurs Cyril Douthe *(IFSTTAR)* et LaurentHauswirth *(UPEM)*. Cette étude a été essentiellement supervisée par Cyril Douthe, docteur et maître de conférences, ainsi que par Xavier Tellier, doctorant, tous deux travaillant au Laboratoire Navier, dans le cadre de l'axe de recherche portant sur les questions de rationalisation géométrique et d'optimisation structurelle, initié par Olivier Baverel il y a une dizaine d'années. Plus précisément, cette étude est étroitement liée aux recherches de Xavier Tellier [11]. D'autres projets de l'équipe de recherche traitant de ces questions se trouvent sur *thinkshell.fr*.

Tout au long de cette étude, nous ferons souvent référence à deux projets de pavillons : le *pavillon IASS Xmesh* et le *pavillon CLC Xmesh*. Les deux projets ont servi d'applications aux méthodes que nous avons mises au point. Bien que les deux projets soient basés sur un processus similaire de conception, de rationalisation, d'optimisation géométrique et d'analyse structurale, ils ont chacun leurs propres spécificités sur les plans de la conception, des matériaux, de la fabrication et de la construction. Ils illustrent donc tous les deux, mais de manières différentes, comment un Xmesh peut être réalisé, quels sont ses avantages et quels sont ses défauts.

Pavillon IASS Xmesh

Le *pavillon IASS Xmesh* est un projet réalisé dans le cadre d'un concours organisé par l'*International Association for Shell and Spatial Structures (IASS)* qui est "une association qui regroupe des ingénieurs, des architectes, des constructeurs et tous ceux qui s'intéressent aux systèmes de structures légères" [3]. Chaque année, cette association organise des symposia, ainsi que des concours selon les années. En 2019, à l'occasion de son 60ème anniversaire, elle organise un concours de pavillons auquel nous, mes tuteurs, quelques chercheurs des laboratoires Navier et LAMA ainsi que des étudiants de l'Ecole d'Architecture de Paris Malaquais et moi-même, participons. Une photo de l'ensemble de l'équipe est présentée en Figure A.1, en annexe.

Le concours de l'IASS impose diverses règles, dont les suivantes:

- *Volume*. L'ensemble du pavillon doit tenir dans un volume de 4x4x4m^3.
- *Masse*. Il doit peser moins de 192 kg.
- *Conditions de support*. Il doit s'appuyer sur le sol.
- *Transport*. Nous devons être capables de stocker tous ses composants dans 6 boîtes de 65x75x100cm^3.
- *Durée de la construction*. Il doit être construit en un jour.

La Figure I.1 montre un rendu 3D de notre proposition de pavillon. C'est un exemple concret de Xmesh. La forme du pavillon est basée sur une portion de tore. Elle présente une double courbure positive et négative. La structure est composée de poutres, de panneaux et d'assemblages sur mesure. Le matériau principal de la structure est l'aluminium.

Pavillon CLC Xmesh

Le *pavillon CLC Xmesh* est un projet réalisé dans le cadre d'un séminaire organisé chaque année par l'*Ecole Nationale des Ponts et Chaussées*. Parmi les différents séminaires de début d'année, l'un d'entre eux, intitulé Contruire Le Courbe (CLC), est consacré à la problématique de la conception et de la construction de formes courbes tridimensionnelle. Ce projet est le fruit de la collaboration de différents chercheurs et doctorants du laboratoire Navier, d'élèves ingénieurs et de moi-même. L'une des spécificités de ce projet est d'être réalisé à l'aide de robots lors de la construction. Le programme de ce projet se définit par plusieurs contraintes s'imposent, entre autres :

- *Forme*. Le pavillon doit figurer l'idée de la courbe en structure.
- *Nombre d'éléments*. Il doit être réalisé à partir de 85 poutres, au maximum, dont les longueurs doivent d'au plus 1500mm (1700mm pour certaines d'entre elles).
- *Matériau*. Il doit être en bois GL28. Les poutres ont une section de 140x60mm^3.
- *Conditions de support*. Il doit être soutenu sur trois points spécifiques d'un mur.
- *Fabrication et construction automatisée*. Il doit être fabriqué et construit grâce à l'assistance d'un robot. Cette condition apporte de nombreuses contraintes géométriques comme les angles de coupe maximaux en fonction de la longueur de la poutre fabriquée par exemple.

La Figure I.2 montre un rendu 3D de ce pavillon. Il s'agit également d'un exemple concret de Xmesh. La surface décrite par la géomtrie du pavillon présente une double courbure essentiellement positive. La structure est composée de poutres, de panneaux et d'assemblages sur mesure.

Rationalisation

La question de la *rationalisation* fait l'objet du Chapitre 1.
Par rationalisation, nous entendons le fait de contraindre la géométrie du projet à satisfaire un ou plusieurs critères spécifiques, qui ont été choisis car bénéfiques pour la fabrication, la construction et/ou les performances structurelles du projet. L'expérience a montré que cette tâche est difficile, souvent longue et peu anticipée, ce qui entraîne généralement des coûts supplémentaires élevés. Par conséquent, la rationalisation est une étape critique dans les projets de forme libre et un domaine de recherche actif. Comme le décrit R. Mesnil [6], il existe deux manières de rationaliser des conceptions.

- *Pré-rationalisation*. Lorsqu'ils adoptent une approche de pré-rationalisation (également connue sous le nom d'approche *ascendante*), les designers travaillent avec des formes connues, qui possèdent des propriétés géométriques souhaitables, à partir desquelles les designs tirent des caractéristiques physiques bénéfiques. Cependant, cette approche contraint les concepteurs à travailler avec des familles de formes spécifiques qui réduisent leur liberté formelle.
- *Post-rationalisation*. Lorsqu'ils suivent une approche de post-rationalisation (également connue sous le nom d'approche *descendante*), les concepteurs partent d'une forme donnée et essaient d'y intégrer fidèlement un design réalisable.

Dans cette étude, nous présentons une approche de post-rationalisation permettant de générer un maillage particulier que nous appelons *Xmesh* et qui satisfait de multiples critères géométriques de fabrication. Dans ce chapitre, définissez plus précisément ce qu'est un Xmesh.

Pour nous assurer que le lecteur puisse comprendre le contenu des différents chapitres, nous rappelons dans un premier un certain nombre de concepts géométriques appartenant aux mondes de la géométrie discrète et de la géométrie différentielle. Nous définissons également de nouveaux concepts, en particulier celui de maillage Xmesh.

Maillage

Un *maillage* est un ensemble de *noeuds*, d'*arrêtes* et de *facettes*. Les noeuds (ou *sommets*) sont des points de l'espace 3D. Les arrêtes sont des segments linéaires reliant des paires de noeuds. Les facettes d'un maillage sont des polygones tridimensionnels formés par des séquences fermées et irréductibles d'arrêtes. Enfin, nous assignons à chaque noeud du maillage un vecteur unitaire que nous appelons *vecteur normal*. La Figure 1.1 donne la présentation d'un maillage.
En outre, nous appelons *plan normal* d'un noeud le plan qui le contient et qui est orthogonal à son vecteur normal. Lorsqu'une arrête et les vecteurs normaux des noeuds situés à ses deux extrémités sont co-planaires, on peut assigner à l'arrête un plan naturel que l'on désigne par *plan d'arrête*. Plus généralement, on peut assigner à n'importe quel arrête un plan que l'on désigne par *plan bissecteur*. C'est le plan qui contient l'arrête et qui divise l'angle formé par la normale des noeuds aux deux extrémités de l'arrête lorsque projeté dans le plan perpendiculaire au bord. La Figure 1.3 donne une représentation des plans d'arrête et bissecteur.

Noeud sans torsion

Considérons un noeud de valence supérieur à 3. A chacune de ces n_e arrêtes adjacentes, nous pouvons définir un plan bissecteur. Dans le cas général, ces plans bissecteurs ne sont pas des plans d'arrêtes. Un *noeud sans torsion* est un noeud dont toutes les arrêtes adjacentes sont tels que les plans bissecteurs sont des plans d'arrêtes. Par conséquent, tous ces plans d'arrête sont coaxiaux et leur axe commun est le vecteur normal du noeud. Un maillage n'ayant que des noeuds sans torsion est appelé *maillage sans torsion*.

Avoir un maillage sans torsion permet de simplifier la construction. En effet, les poutres peuvent être orientées de telle sorte que leurs plans médians coïncident avec les plans d'arrêtes. Ainsi, à chaque connexion, tous les plans médians des poutres concourrantes se croisent selon un axe unique. Les connexions peuvent donc être conçues de façon standard, intégrant un axe sur lequel les poutres sont connectées.

Noeud plat

Considérons un nœud de valence n_e supérieur à 2 : le noeud est dit *plat* si ses n_e arrêtes sont toutes incluses dans son plan normal. Dit de façon équivalente, le noeud est plat si ses noeuds voisins sont tous inclus dans son plan normal. Ou encore, il est plat si lui et ses n_e noeuds voisins sont tous coplanaires et son vecteur normal est orthogonal à ce plan commun. Au contraire, si un noeud n'est pas plat, on dit qu'il s'agit d'un noeud *3D*. La Figure 1.7 montre une représentation de ces deux types de noeuds.

En pratique, une propriété utile d'un tel noeud plat est qu'il peut être recouvert d'un panneau polygonal dont le vecteur normal correspond au vecteur normal du noeud. Dans le cas du Xmesh, les noeuds plats ont trois arrêtes. Un noeud plat peut donc être recouvert d'un panneau triangulaire dont les sommets sont les noeuds voisins. D'un point de vue architectural, cela signifie qu'il existe un moyen naturel et facile de recouvrir la surface du gridshell.

Les Figures 1.8 et 1.9 montrent comment les noeuds plats sont réalisés dans le cadre des projets de pavillons IASS Xmesh et CLC Xmesh. Dans le contenu du rapport en anglais, nous explicitons quels sont les arguments relatifs à la fabrication, à la construction et au comportement mécanique qui nous on fait converger vers ces designs. Il est intéressant de noter que les technologies utilisées dans ces deux projets sont très différentes alors qu'elles matérialisent physiquement le même concept géométrique. Ceci étant dit, il demeure que ces deux instances de noeuds plats partagent le caractère commun d'être génériques, qui constitue un véritable atout sur les plans de la fabrication et de la construction. Pour finir, les capacités de translation et de rotations de ces instances de noeuds plats sont soignesement étudiées. Il est crucial de mener ces études car elles justifient la modélisation de la connectivité des noeuds plats dans les modèles de structure décrits au Chapitre 3.

Noeud 3D à 120°

Considérons maintenant un noeud 3D de valence 3. Le noeud est dit un *noeud 3D à 120°* si ses 3 arrêtes, lorsqu'elles sont projetées dans le plan normal du noeud, forment deux à deux 3 angles de 120 degrés. La Figure 1.10 représente un noeud 3D et un noeud 3D à 120°.

Comme pour le noeud plat, le noeud 3D à 120° est un avantage du point de vue de la fabrication et de la construction puisqu'il est facile de concevoir un design générique pour matérialiser ce noeud. Les Figures 1.11 et 1.12 montrent comment les noeuds 3D à 120° sont réalisés dans le cadre des projets de pavillons IASS Xmesh et CLC Xmesh. Un traitement analogue à celui mené pour les noeuds plats est réalisé dans le contenu du rapport en anglais.

Xmesh

Nous pouvons désormais définir ce qu'est un *Xmesh*. Un Xmesh est un maillege qui possède les propriétés suivantes :

- Ces facettes sont des haxagones tridimensionnels. ces noeuds ont une valence de 3.
- Ces noeuds sont sans torsion.
- En considérant le maillage comme un graphe, celui-ci est 2-colorable. Par conséquent, ces noeuds peuvent être séparés en deux familles. Les noeuds de la première familles sont des noeuds plats. Les noeuds de la deuxième famille sont des noeuds 3D à 120°.

Dans [12], X. Tellier *et al.* donnent une preuve de l'existence du Xmesh décrit dans en Section 1.2. La Figure 1.13 illustre cette preuve. Dans le contenu du rapport en anglais, nous donnons quelques brèves explications à ce sujet, mais d'autres détails sont fournis dans [12]. La preuve est développée dans un cadre asymptotique, c'est-à-dire lorsque la taille caractéristique du maillage devient de plus en plus petite, de sorte que l'on peut utiliser des notions de géométrie différentielle continue en approximation.

Optimisation

Dans le Chapitre 1, nous décrivons diverses concepts géométriques dont celui de Xmesh caractérisé par ses noeuds sans torsion, plats ou 3D à 120°. Nous mentionnons également les bénéfices que l'on peut tirer de telles géométries du point de vue de la fabrication et de la construction par le caractère générique de ces noeuds. Cependant, générer un maillage non trivial de type Xmesh n'est pas évident. Dans le chapitre 2, nous décrivons la méthode de *post-rationalisation* que nous avons implémenter pour générer des Xmeshs et que nous avons utilisé en pratique pour les projets de pavillons IASS Xmesh et CLC Xmesh.

Logiciels utilisés pour la design et l'optimisation géométriques

Pour implémenter notre méthode de génération de Xmeshs, nous avons utiliser plusieurs logiciels ou algorithmes. Nous pouvons citer *Rhinoceros3D*, *Grasshopper*, *Kangaroo2* et *Shape-Up*. Nous donnons davantage de détails dans le contenu du rapport en anglais quant aux fonctionnalités de ces logiciels, aux concepts sur lesquels ils se fondent et à l'utilisation que nous en avons. Dans le cadre de ce résumé en français, nous allons traiter que de *Kangaroo2*.

Optimisation géométrique avec Kangaroo2

Kangaroo2 est une librairie *Grasshopper* permettant de formuler et de résoudre des problèmes d' optimisation géométrique. L'auteur de ce logiciel, Daniel Piker, fournit aux utilisateurs un ensemble d'outils simples d'utilisation et unifiés autour du concept de *goal*. Un goal est un composant *Grasshopper* permettant d'appliquer des contraintes géométriques sur différents objets géométriques de *Rhinoceros3D* comme des points, des lignes, des surfaces, des maillages et autres. L'aggrégation de goals permet de définir un problème d'optimisation géométrique sous-contraintes. Un solver peut être ensuite utilisé afin de résoudre ce problème. Le solver procède par itération, en veillant à chaque étape, à trouver un meilleur compromis entre les différentes contraintes géométriques définies au moyen des goals.

Par défaut, la librairie *Kangaroo2* propose de nombreux goals dont celui de coplanarité de points. Il est aussi rendu possible aux utilisateurs de définir leur propres goals. Nous pouvons ainsi définir des goals associés aux contraintes géométriques de noeud sans torsion, de noeud plat et de noeud 3D à 120°. Des explications, illustrations et algorithmes sont fournis en Sections 2.2.2, 2.2.3 et 2.2.4 ainsi qu'en Annexes.

Application du processus de génération et d'optimisation aux projets de pavillon

Grâce à ces goals personnalisés, nous pouvons formuler un problème d'optimisation visant à obtenir un Xmesh. Nous appliquons cela aux projets de pavillons IASS Xmesh et CLC Xmesh. Nous donnons en Figures 2.15 et 2.16 les résultats de convergence de ces processus.

Ces résultats supposent l'initialisation de maillages à optimiser afin de les transformer en maillage Xmesh. La méthode d'initialisation est décrite étape par étape en Section 2.3.2.

Les résultats obtenus sont très satisfaits. Il est important de noter que les maillages obtenus sont des Xmesh à des erreurs métriques près. Ces erreurs ne sont pas un problème dans la mesure où elles sont bien inférieures aux tolérances de construction de ces deux projets.

Limites de la méthode de génération de maillages Xmesh

Cette méthode de type post-rationalisation a prouvé son efficacité dans le cadre de ces deux projets de pavillons. Néanmoins, elle présente quelques limites.

Premièrement, une première limite est liée à l'initialisation du maillage à optimiser qui s'appuie sur les *lignes de courbures* de la surface approximée par le maillage. D'une part, ces lignes de courbures doivent être relativement régulière pour pouvoir définir un maillage initial à optimiser. D'autre part, il n'est pas évident d'anticiper la forme des lignes de courbures d'une surface. En conséquence, il est courant de devoir recommencer à zéro la génération du maillage car le resultat final n'est pas satisfaisant d'un point vue architectural, structurel ou constructif.

Deuxièmement, nous avons pu observé que le processus d'optimisation pouvait rencontrer des difficultés à produire des résultats satisfaisants lorsque la courbure de la surface approximée est forte relativement à la taille des arrêtes du maillage. Plus les arrêtes du maillage sont grandes relativement au rayon de courbure de la surface, plus il est difficile de converger vers de bons résultats.

Analyse structurelle

Dans le Chapitre 3, nous nous attachons à modéliser des structures s'appuyant sur des maillages de type Xmesh. En particulier, nous expliquons comment nous avons modélisé les structures des projets de pavillons IASS Xmesh et CLC Xmesh.

Modélisation des noeuds plat et 3D à 120°

Dans le Chapitre 1, nous avons présenté les solutions technologiques utilisées afin de matérialiser les noeuds plat et 3D à 120° dans les deux projets de pavillon. Dans les Sections 3.2 et 3.3, nous expliquons comment nous modélisons ces instances de noeuds étant donnés leur forme physique.
Lorsque nécessaire nous modélisons, au moyen de logiciels d'éléments finis tel que *Karamba* ou *Abaqus*, les pièces constitutives de ces noeuds. Cela nous permet de tirer des valeurs de raideur élastique cruciales pour les modèles de structure associés aux deux projets. Cela nous permet également de vérifier la résistance de ces pièces vis-à-vis des contraintes auxquelles elles sont soumises.

Modélisation des structures

Dans la Section 3.4, nous expliquons de façon détaillée la façon dont nous avons modélisé les structures des pavillons IASS Xmesh et CLC Xmesh.
Nous utilisons le logiciel *Karamba* intégré à *Rhinoceros3D* et *Grasshopper*. Nous explicitons les conditions aux bords, les hypothèses relatifs à la connectivité des noeuds ainsi que les combinaisons de charges étudiées. Nous décrivons en particulier les hypothèses de relâchement, de blocage et de ressorts élastiques assignées aux différentes connexions. Nous nous attachons également à décrire le travail de modélisation des panneaux des noeuds plats en ensemble de bandes. Nous donnons également les diagrammes des efforts de ces structures. Il est intéressant de noter que le torseur des efforts développés dans les poutres de ces structures est complet : tous les efforts y sont présents. Ceci est une conséquence directe de la géométrie du maillage et des hypothèses faites quant à la connectivité des noeuds.
En Section 3.5, nous montrons dans quelles mesures les performances structurelles du pavillon IASS Xmesh est sensible aux raideurs élastiques de ses connections aux noeuds 3D à 120°.

Conclusion

Dans cette étude, nous avons décrit les géométries *Xmesh* et expliqué comment elles peuvent constituer de bonnes solutions géométriques pour la conception de gridshells rigides *free-form*. Nous avons développé une méthode de post-rationalisation, basée sur l'optimisation géométrique, afin de générer de telles géométries et nous avons modélisé leur comportement structurel. A travers la réalisation de deux pavillons *free-form*, qui utilisent des technologies très différentes, nous avons prouvé le potentiel et l'adaptabilité de ces géométries ainsi que nos méthodes de définition, d'optimisation, d'analyse structurale et de génération des plans de construction. Enfin, ces deux réalisations démontrent l'intérêt de méthodes intégrant en permanence des considérations architecturales, structurelles et constructives, dans le but de concevoir des projets *free-form* économiques et efficaces.
Néanmoins, nous avons également souligné quelques limites dans notre travail au cours de ce document. Sur le plan géométrique, nous avons observé que notre méthode de génération ne permet pas une véritable liberté formelle. En ce qui concerne la méthode d'initialisation basée sur des lignes de courbures, les concepteurs doivent éviter les surfaces peu régulières, car elles empêcheraient d'obtenir une grille régulière de lignes. En ce qui concerne le processus d'optimisation, les concepteurs doivent éviter les surfaces à fortes courbures par rapport à la taille typique des arrêtes du maillages.
nfin, un aspect de ce projet qui n'a pas été développé en profondeur dans ce document est la fabrication et la construction de structure Xmesh. Certains éléments concernant la fabrication et la construction ont été mentionnés dans les différents chapitres. Pourtant, il aurait été possible d'écrire un chapitre entier simplement sur la génération des données de fabrication. La raison pour laquelle nous n'avons pas développé un tel chapitre est que cette étape est sur le point d'être terminée au moment où ce document a été écrit. Cependant, les lecteurs doivent être conscients qu'il s'agit d'une partie importante de notre travail et d'un facteur d'influence majeur pour la conception de maillages Xmesh, ceux-ci ayant pour essence d'être des géométries qui tiennent compte de la fabrication et de la construction.

Contents

List of Tables

List of Figures

Appendices

Introduction

With the development and the popularization of computer-aided industrial design software, *free-form* designs, based on complex shapes, have become relatively common in contemporary architecture. Each time, the geometry, the materials, the structure, the fabrication and the construction of such projects are particularly inter-weaved. Their final design results from iterative collaborative works between architects, engineers and contractors.
In a free-form design, the essential element is its *geometry*. For the free-form architects, the geometry is the motive of the project; it is influenced by various considerations like volume, light and functions. For the engineers, the geometry is the abstract skeleton or the approximate shape on which the structure is based; it must be such that the structural elements behave properly and resist under loads. For the contractors, the geometry is indirectly fundamental as it influences the feasibility of the project; the geometry is indirectly constrained by the techniques, the cost and the time of fabrication and of construction. By considering the geometry as some variable and the concurrent interests of the various actors as some set of constraints, we can regard the choice of the project's geometry as a large optimization problem under multiple constraints. The chosen geometry shall represent a good compromise between all architectural, engineering and constructive expectations.
As a consequence, as designers of a free-form project, we shall be aware of the engineering and construction considerations that can constraint its geometry and find some ways to integrate them within the process of creation - which appears to be a complex challenge.

For the past decades, many mathematical and algorithmic methods have been developed to approach this challenge. They are part of the domain called *geometric rationalization*. These approaches *rationalize* geometries in that they aim at creating geometries with non-trivial assets while guaranteeing a certain formal freedom to the designers. Depending on the method, these assets can be beneficial to architects, engineers and/or contractors.
Experiences have shown that it is relevant to invest time and effort to rationalize the nodes of a free-form project like a grid-shell. For instance, for the cover of the great court at the British Museum, the nodes at the intersection of the grid-lines had to be individually cut from a thick steel plate by plasma technology as they were not generic enough [4]. Therefore, when designing a gridshell, one strategy of geometric rationalization we can implement is to aim at generating geometries with high node congruence.

In this study, we propose such a strategy that can be applied specifically to hexagonal meshes. In the Chapter *1. Rationalization*, we define or recall some geometric notions at the basis of our strategy. In particular, we give a definition of an *Xmesh*, a particular type of fabrication-aware geometry. In the Chapter *2. Optimization*, we describe a key step of our strategy based on *geometric optimization*. We introduce the tools and present elements of code we use to generate Xmeshes. In the Chapter *3. Structural analysis*, we look at the structural behaviour of such geometries and discuss its benefits and its flaws.

This work is the result of a six-month internship, within the department of *Architectured Materials and Structures (MSA)* of the *Laboratoire Navier*. It comes within the scope of the *Impulsion* project *GAMES*, funded by *Université Paris Est*, led by researchers Cyril Douthe *(IFSTTAR)* and Laurent Hauswirth *(UPEM)*. This study has been essentially supervised by Cyril Douthe, a doctor and engineering lecturer, as well as Xavier Tellier, a doctoral student, both working at the *Laboratoire Navier*. It is part of the research line, which focuses on questions of geometric rationalization and structural optimization, that has been initiated by Olivier Baverel about ten years ago. More specifically, this study is tightly connected to the research done by Xavier Tellier [11]. Other related projects of the research team can be found on *thinkshell.fr*.

All along this study, we will often refer to two projects of pavilions: the *IASS Xmesh pavilion* and the *CLC Xmesh pavilion*. Both projects have served as real applications of the method we built. Although both projects are based on a similar process of design, rationalization, geometric optimization and structural analysis, they each have their own design specifications, their own material, their own fabrication and construction constraints. They both illustrate, but in different ways, how an Xmesh can be realized, what are its benefits and what are its flaws.
Before starting with the first chapter, let us now give a brief presentation of both projects.

I.1 IASS Xmesh pavilion

The *IASS Xmesh pavilion* is a project motivated by a contest organized by the *International Association for Shell and Spatial Structures (IASS)* which is "an association that gathers engineers, architects, builders and all others interested in lightweight structural systems" [3]. Every year, this association organizes symposia, as well as contest depending on the year. In 2019, to celebrate its 60th anniversary, it organizes a contest of pavilions in which we, my tutors, some researchers from the laboratories Navier and LAMA as well as students from the Architecture School of Paris Malaquais and I, have taken part. A photo of the entire team is given in Figure A.1, in the appendices.

The contest imposes various rules, among them:

- *Volume*. The whole pavilion must fit in a volume of 4x4x4m^3.
- *Mass*. It must weight less than 192kg.
- *Support conditions*. It must stand on the floor.
- *Transport*. We must be able to fit all its components with six 65x75x100cm^3 boxes.
- *Construction duration*. It must be built in one day.

In the Figure I.1, we give a 3D render of our proposition of pavilion. It is an instance of an Xmesh. The shape of the pavilion is based on a portion of a torus. It presents both positive and negative double curvatures. The structure is composed of beams, panels and custom connections. The main material of the structure is aluminium.

Although, we will mainly refer to the final version of the project, one has to keep in mind that this final version results from a lot of iterations and decisions that team has taken over months. We will sometimes mention other options the team had considered to show other perspectives.

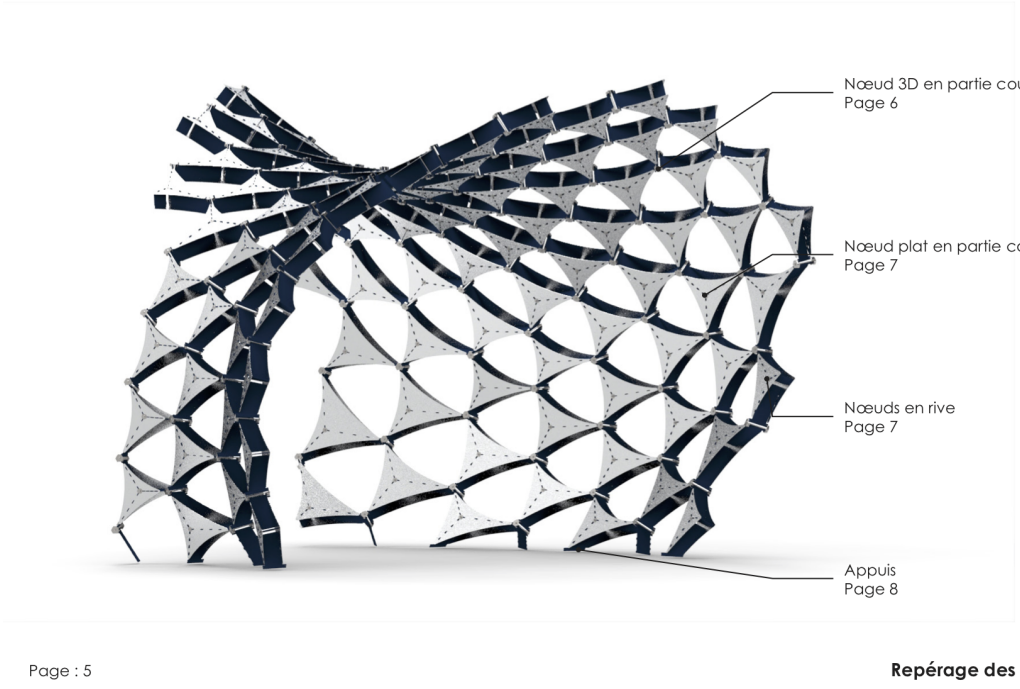


Figure I.1: 3D render of the *IASS Xmesh pavilion*. Image kindly shared by our architects colleagues G. Jami, T. Lenart, A. Le Pavec and S. Zerhouni.

I.2 CLC Xmesh pavilion

The *CLC Xmesh pavilion* is a project motivated by a seminary organized every year by the *Ecole Nationale des Ponts et Chaussées*. Among the various seminaries happening at the beginning the scholar year, one called *Contruire Le Courbe (CLC)*, literally *Building curved*, focuses on the problem of designing and building 3D complex shapes in an efficient way.
This project is the fruit of the collaboration of various researchers and PhD students from the laboratory Navier, students of Engineering degree and myself. One of the specific features of this project is to be built with the use of robots during the construction.

Several constraints are imposed, among them:

- *Shape*. The pavilion must express curvature.
- *Number and size of elements*. It must be made from a maximum number of 85 beams which must be shorter than 1500m (1700mm for few of them).
- *Material*. It must be in wood GL24. The cross sections of the beam must be 140x60mm^2.
- *Support conditions*. It must be supported on three specific points of a wall.
- *Automated fabrication and construction*. It must be manufactured and built thanks to robotic assistance. This condition brings many geometric constraints like maximal angles of cut according to the length of the manufactured beam for instance.

In the Figure I.2, we give a 3D render of this pavilion. It is also an instance of an Xmesh. The approximate surface of the pavilion has essentially positive curvatures. The structure is also composed of beams, panels and custom connections.

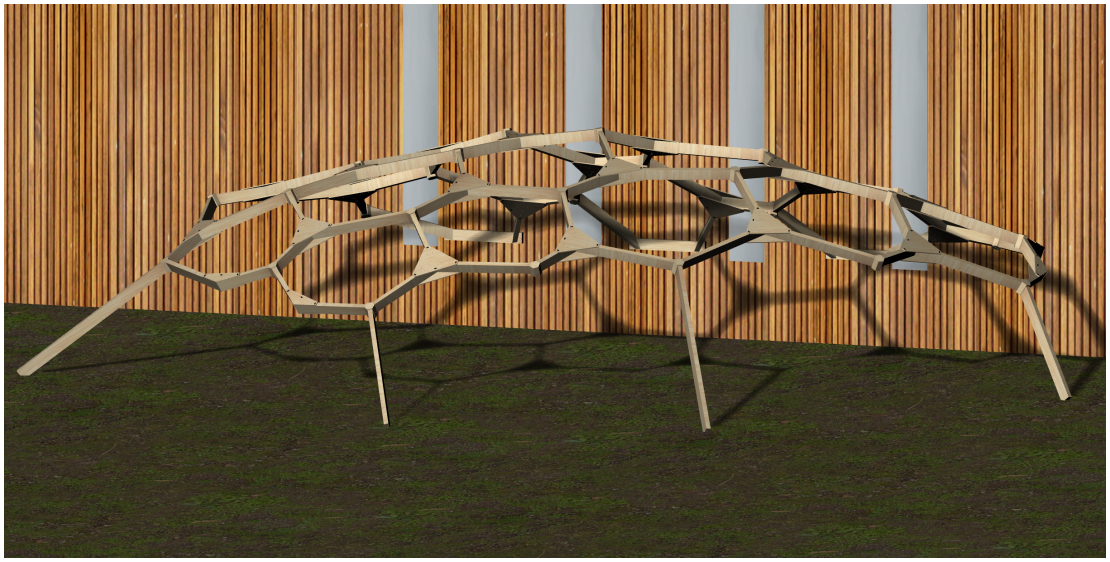


Figure I.2: 3D render of the *CLC Xmesh* pavilion

Chapter 1

Rationalization

The shape complexity of free-form architecture is often detached from manufacture and construction considerations for the purpose of formal expressiveness. A *rationalization* of the geometry comes as a necessary task in order to handle these considerations. By rationalization, we mean constraining the geometry of the design to satisfy one or several specific criteria, resulting in beneficial physical conditions regarding the fabrication, the construction and/or structural performances. Experiences have shown that this task is difficult, often long and not fully anticipated, which generally results in high additional costs. Therefore, rationalization is a critical step in free-form projects and an active domain of research.

As described by R. Mesnil [6], there are two ways for rationalizing designs.

- *Pre-rationalization*. When adopting a pre-rationalization approach (also known as *bottom-up* approach), designers works with well-known shapes, that have desirable geometric properties, from which designs get beneficial physical conditions. However, this approach constraints the designers to work with specific families of shapes reducing their formal freedom.
- *Post-rationalization*. When following a post-rationalization approach (also known as *top-down* approach), designers starts from a given shape and try to faithfully fit a feasible design on it.

In this study, we present a general *post-rationalization* approach to generate a particular mesh which we call *Xmesh* and which satisfies multiple fabrication-aware geometric criteria.

Before presenting our approach, let us recall various geometric notions about meshes and surfaces.

1.1 Recall of some geometric notions

In this section, we recall some geometric concepts as well as geometric properties that we can wish the geometry of our free-form design to satisfy in order to have beneficial physical conditions regarding its fabrication.

1.1.1 Meshes

As usually defined, a *mesh* is a set of nodes, edges and faces. *Nodes* (or *vertices*) are points of the 3D space. *Edges* are linear segments connecting pairs of nodes. *Faces* of the mesh are 3D polygons formed by closed irreducible sequences of edges. Finally, we assign to each node of the mesh a unit vector that we call its *normal vector*. The Figure 1.1 gives a representation of one mesh.

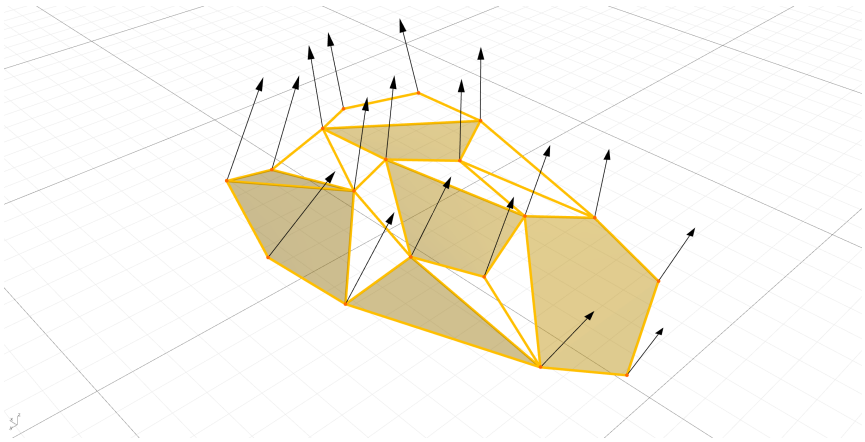


Figure 1.1: An instance of a *mesh*. The yellow segments correspond to the edges of the mesh, the orange points to the nodes, the black arrows to the normal vectors of the nodes and the surfaces to some of the faces of the mesh.

In addition, we call *normal plane* the plane that includes the node and that is orthogonal to the node's normal vector. When an edge and the normal vectors of the nodes at its ends are together co-planar, we can assign a natural plane to the edge which we designate with the locution *edge plane*. More generally, we can assign to any edge a plane which we designate with *bisecting plane*. It is the plane which contains the edge and which bisects the angle formed by the normal of the nodes at the ends of the edge when projected in the plane normal to the edge. The Figure 1.3 gives a representation of both edge and bisecting planes.

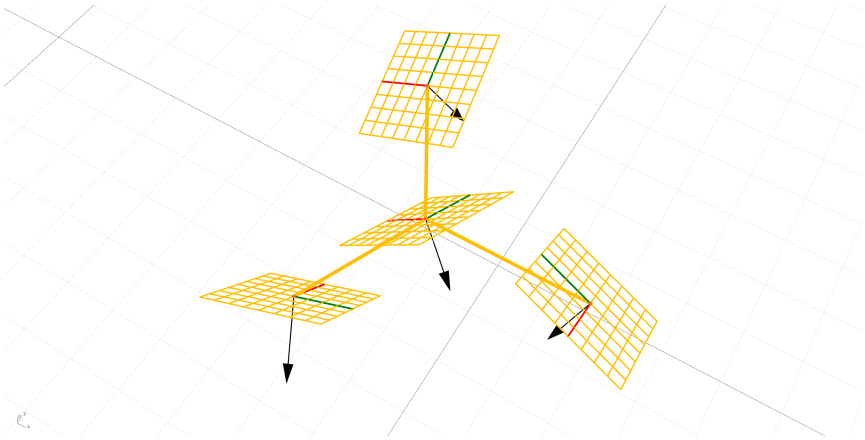
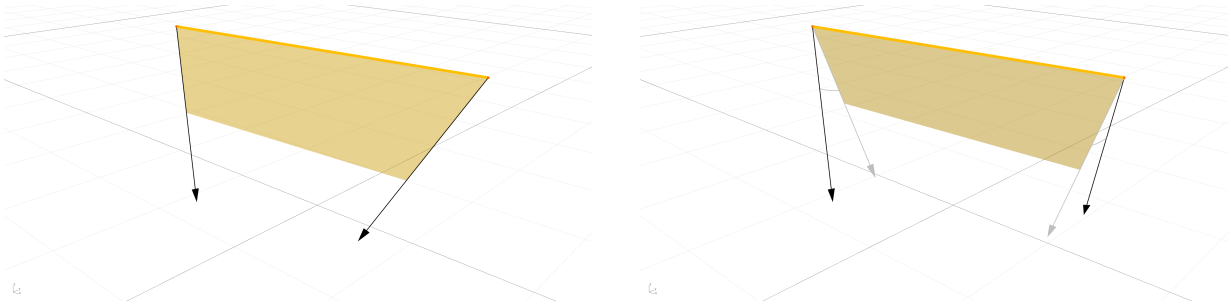


Figure 1.2: Instances of nodes' *normal planes*.



(a) An instance of an edge plane

(b) An instance of a bisecting plane

Figure 1.3: Difference between an *edge plane* and a *bisecting plane*. For the bisecting plane, the grey arrows correspond to the projection of the normal vectors of the nodes at the ends of the edge on the bisecting plane.

In the context of an architectural project, this mesh can represent the spatial geometry of the structure. The edges of the mesh correspond to the center-lines of the beams. The nodes correspond to the connections between the beams. The faces (or parts of faces) often correspond to panels. Normal vectors correspond to main axis of the connections. Bisecting planes of edges can be seen as the median planes of corresponding beams.

Various other notions can be defined around meshes. We give some important ones that we will often use in this study. For a given node, the *neighboring nodes* (or *neighbors*) are the nodes that are connected to it by an edge. The *valence* of a node is the number of edges which are connected to it; it is also the number of its neighbors.
A *hexagonal mesh* is a mesh which faces are 3D hexagons, that is to say closed sequences of six consecutive edges. One hexagonal mesh is represented in the Figure 1.4.
In this study, we focus our attention on hexagonal mesh which nodes have a valence of 3.

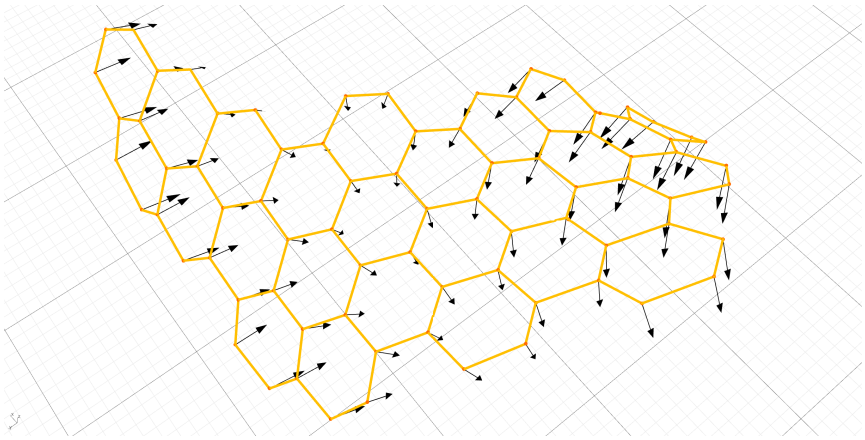
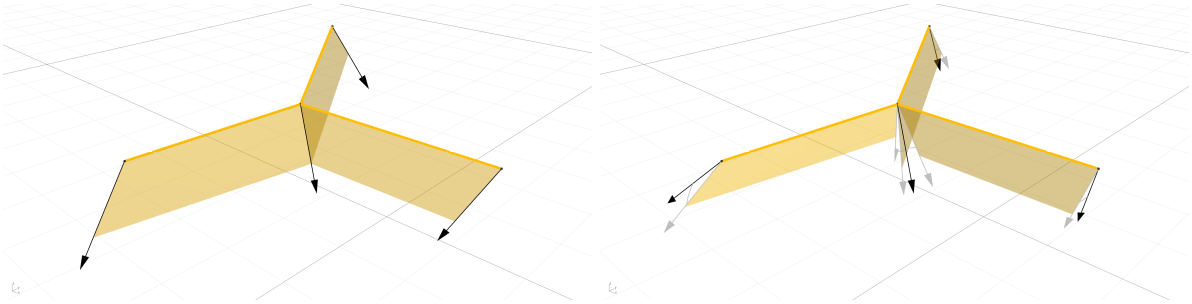


Figure 1.4: An instance of a *hexagonal mesh*. The yellow segments represent the edges of the mesh, the orange points to the nodes and the black arrows to the normal vectors of the nodes.

1.1.2 Torsion-free nodes

Let us consider a node of valence n_e higher than 3. For each of the n_e connecting edges, we can define a bisecting plane. In the general case, these bisecting planes are not edge planes. A *torsion-free* node is specifically a node where all of its connecting edges have edge planes as bisecting planes. As a consequence, all of these edge planes are co-axial and their common axis is the normal vector of the node. A mesh with torsion-free nodes is called a *torsion-free mesh*.



(a) Torsion-free node

(b) Non-torsion-free node

Figure 1.5: Difference between a *torsion-free* node and a non one. For the non-torsion-free node, the arrows in light grey represents the projection of the normal vectors of the nodes on the bisecting plane.

Regarding construction, having a torsion-free mesh helps simplifying the construction. Indeed, beams can be orientated such that their median planes coincide with the edge planes. This way, at each connection, all the median planes of the connected beams intersect themselves according to a unique axis. The connections can therefore be designed as standard axis on which beams get connected.

1.1.3 Lines of curvature

Unlike the two previous sections which were about concepts of discrete geometry, *principal curvatures* belongs to the theory of continuous differential geometry.

Regular parametrization
Let S be a surface of $\mathbb{R}^3$ and $\{\boldsymbol{s}(u,v)\}_{(u,v)\in\mathbb{R}^2}$ be a parametrization of S. $\{\boldsymbol{s}(u,v)\}_{(u,v)\in\mathbb{R}^2}$ is a *regular parametrization* if $\boldsymbol{s}_u(u_0,v_0) = \frac{\partial s}{\partial u}(u_0,v_0)$ and $\boldsymbol{s}_v(u_0,v_0) = \frac{\partial s}{\partial v}(u_0,v_0)$ are always independent.

Normal vector
Let S be a surface of $\mathbb{R}^3$, $\{\boldsymbol{s}(u,v)\}_{(u,v)\in\mathbb{R}^2}$ be a regular parametrization of S and $\boldsymbol{p} = \boldsymbol{s}(u_0,v_0)$ a point of S. A unit *normal vector* $\boldsymbol{n_p} = \boldsymbol{n}(u_0,v_0)$ of S at $\boldsymbol{p}$ is given by

$$\boldsymbol{n}(u_0,v_0) = \frac{\boldsymbol{s}_u(u_0,v_0)\times\boldsymbol{s}_v(u_0,v_0)}{\parallel \boldsymbol{s}_u(u_0,v_0)\times\boldsymbol{s}_v(u_0,v_0)\parallel} \in \mathbb{R}^2$$

where $\times$ is the cross-product.
The vector field $\{\boldsymbol{n}(u,v)\}_{(u,v)\in\mathbb{R}^2}$ is called *unit normal vector field*.

Shape operator
Let S be a surface of $\mathbb{R}^3$, $\{\boldsymbol{s}(u,v)\}_{(u,v)\in\mathbb{R}^2}$ be a regular parametrization of S, $\{\boldsymbol{n}(u,v)\}_{(u,v)\in\mathbb{R}^2}$ its unit normal vector field and $\boldsymbol{p} = \boldsymbol{s}(u_0,v_0)$ a point of S. The *shape operator* (or Weingarten map) of S at point $\boldsymbol{p}$ is a map associating a tangent vector $\boldsymbol{t_p}$ of S at $\boldsymbol{p}$ to the negative derivative of the unit normal vector field of the surface at $\boldsymbol{p}$,

$$\boldsymbol{w_p}(\boldsymbol{t_p}) = -\delta_{\boldsymbol{t_p}}\ \boldsymbol{n_p} \in \mathbb{R}^2$$

where $\delta_{\boldsymbol{t_p}}$ is the directional derivative.

Normal curvature
Let S be a surface, $\boldsymbol{p}$ be a point on S and $\boldsymbol{t_p}$ be a unit vector tangent to the surface S at $\boldsymbol{p}$. The *normal curvature* $\kappa_{\boldsymbol{p}}(\boldsymbol{t_p})$ of S at the point $\boldsymbol{p}$ in the direction $\boldsymbol{t_p}$ is given by

$$\kappa_{\boldsymbol{p}}(\boldsymbol{t_p}) = \boldsymbol{w_p}(\boldsymbol{t_p})\cdot\boldsymbol{t_p} \in \mathbb{R}$$

where $\boldsymbol{w_p}$ is the shape operator of S.

Principal curvatures
The maximum and minimum of the normal curvature $\kappa_{\boldsymbol{p},1}$ and $\kappa_{\boldsymbol{p},2}$ at a given point $\boldsymbol{p}$ on a surface are called the *principal curvatures*. The principal curvatures locally measure the maximum and minimum bending of a regular surface.

$$\kappa_{\boldsymbol{p},1} = \max_{\boldsymbol{t_p}} \kappa_{\boldsymbol{p}}(\boldsymbol{t_p}) \qquad \text{and} \qquad \kappa_{\boldsymbol{p},2} = \min_{\boldsymbol{t_p}} \kappa_{\boldsymbol{p}}(\boldsymbol{t_p})$$

Lines of curvatures

The lines of curvatures are curves on the surface whose tangents are always in the direction of principal curvature.

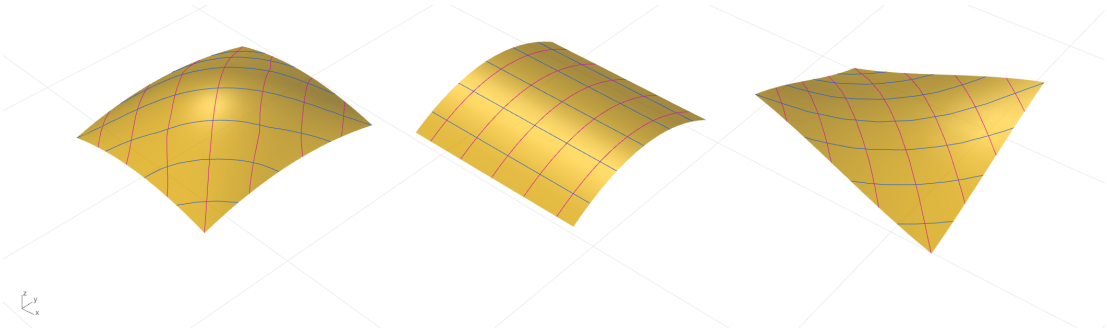


Figure 1.6: *Lines of curvatures* of three instances of surfaces. From left to right, a synclastic surface, a single-curvature surface and an anticlastic surface.

In several works like [6] or [14], lines of curvature are used as a geometric basis on which we can define or initialize a mesh. In Sections 2.1.1 and 2.3.1, we will present some tools that we use to create surface and compute its curvature lines. In Section 2.3.2, we will explain how we use the lines of curvatures of the target surface to generate an initial hexagonal mesh which then must be optimized to be turned into an Xmesh.

1.2 Geometric properties of an Xmesh

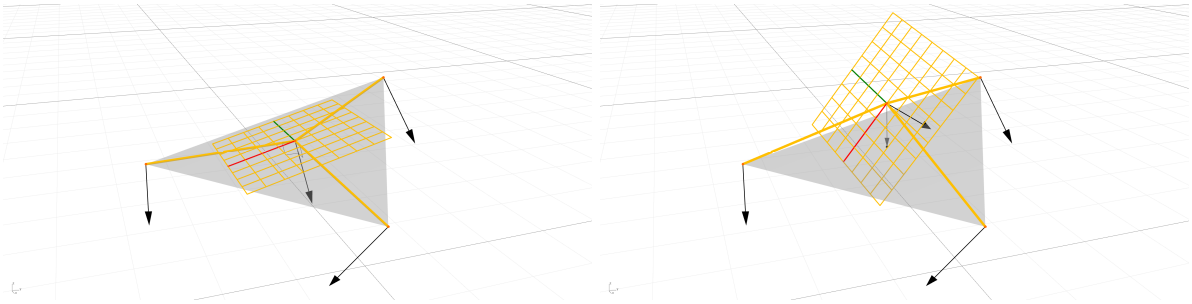
We now focus on a specific mesh that we call an *Xmesh*. An Xmesh has the following properties.

- Its facets are non-planar hexagons. Its nodes have a valence of three.
- Its nodes are torsion-free.
- Seeing the mesh as a graph, it is 2-color-able. Therefore, its nodes can be separated in two families. The nodes of the first family are flat nodes. The nodes of the second family are 120-3D nodes.

We now define the notions of *flat nodes* and *120-3D nodes*.

1.2.1 Flat nodes

Let us consider a node of valence n_e higher than 2. The node is said to be a *flat node* if its n_e edges are all included into its normal plane. Equivalently, its n_e neighboring nodes are all included into its normal plane. Or even, the n_e neighboring nodes as well as the main node are all coplanar and the normal vector of the main node is orthogonal to this common plane. If a node is not flat, it is said to be a *3D node*.
The Figure 1.7 shows a representation of these two types of nodes.



(a) An instance of a flat node (b) An instance of a 3D Node

Figure 1.7: Difference between a *flat node* and a *3D node*. In the representation of the 3D node, the grey arrow emphasizes the gap between the main node and the plane defined by its three neighbors.

In practice, an useful property of such a flat node is that it can be covered with a polygonal panel whose normal vector corresponds to the node's normal vector. In the case of the Xmesh, flat nodes have 3 edges. A flat node can be covered with a triangular panel whose vertices are its neighboring nodes. With an architectural point of view, this means that there is a natural and easy way to cover the mesh.
This local planarity property of the mesh is the benefit we use to realize the flat nodes of both IASS Xmesh and CLC Xmesh pavilions projects.

For the IASS Xmesh pavilion, we materialize each flat node with a pair of aluminium triangular panels, as the Figure 1.8 shows it. The beams, whose center-lines are the edges of the mesh, are aligned with the bisectors of this triangle and the flat node location is at the intersection of these bisectors.

For the purpose of construction efficiency, the kinematic of assembly has been thought to be reversible, to be fast and to require only a screwdriver. Note that mirrored-L-shaped stops have been distributed along the top and the bottom sides of the beams. Firstly, the three beams are sandwiched between the two triangular panels and their stops get through the rectangular holes that have been cut out from the panels. Then, the beams are slid outward along the bisectors of the triangle and stops hit one of the edges of the rectangular holes. Finally, a unique screw spans between the two panels, constricts them against the beams, and blocks the beams to slide back inward thanks to a washer.
On the fabrication plan, this flat node detail is relatively generic. It is true that the dimensions and the angles of the triangle varies depending on the length and the orientation of the edges of the mesh. However the geometric rule to generate the triangle is standard. Furthermore, the different parts of the node are either standard elements, that we can get from suppliers, or pieces that we laser-cut.
Finally, this technological solution determines the mechanical behaviour of the flat node. The lateral movements and the extension of the beams are blocked by isolated contacts with the panel at the locations of the L-shaped stops. Their vertical movements are blocked by linear contact with the panels. Finally, their contraction is blocked at their ends, thanks to a washer, attached to a screw, which prevents the beams to slide back toward the middle of the panel by hitting their first L-shaped stop. In Section 3.2, we will describe how we model this flat node in the IASS structural model, based on these kinematic considerations.

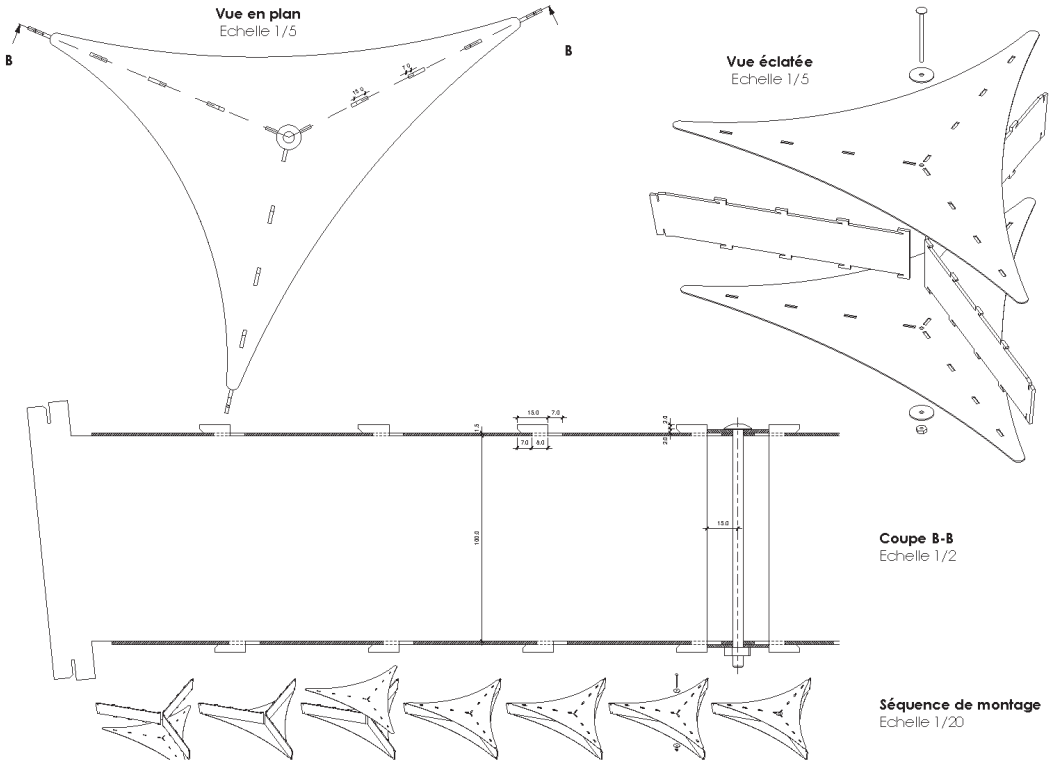


Figure 1.8: Detail of the flat node for the *IASS Xmesh pavilion* project. Image kindly shared by our architects colleagues G. Jami, T. Lenart, A. Le Pavec and S. Zerhouni.

For the CLC Xmesh pavilion, we realize this flat node with wooden triangle panels as the figure 1.9 shows it. Again, the bisectors of this triangle are aligned with the beams' center-lines, which are the edges of the mesh, and the intersection of these bisectors corresponds the flat node's location. At this location, the ends of the beams are shaped in a triangular fashion to fit together - on the condition that the beams are long enough (larger than 800mm) due to manufacturing constraints.
Here, we do not look for a reversible detail. One significant constraint is that the triangle panel shall fit into a wooden panel of 600x600mm^2 from which they are laser-cut.
On the mechanical plan, all out-of-plane movements of the beams are blocked by contact with the panels, which are sandwiched to the beams thanks to the groups of screws. All in-plane movements are essentially blocked by the groups of screws. The triangle-shaped ends of the beams - if they exist - can possibly help to block contraction of the beams, but we should not rely on these to block lateral movements as no screws are drilled here. In Section 3.2, we will describe how we model this flat node in the CLC Xmesh structural model, based on these kinematic considerations.

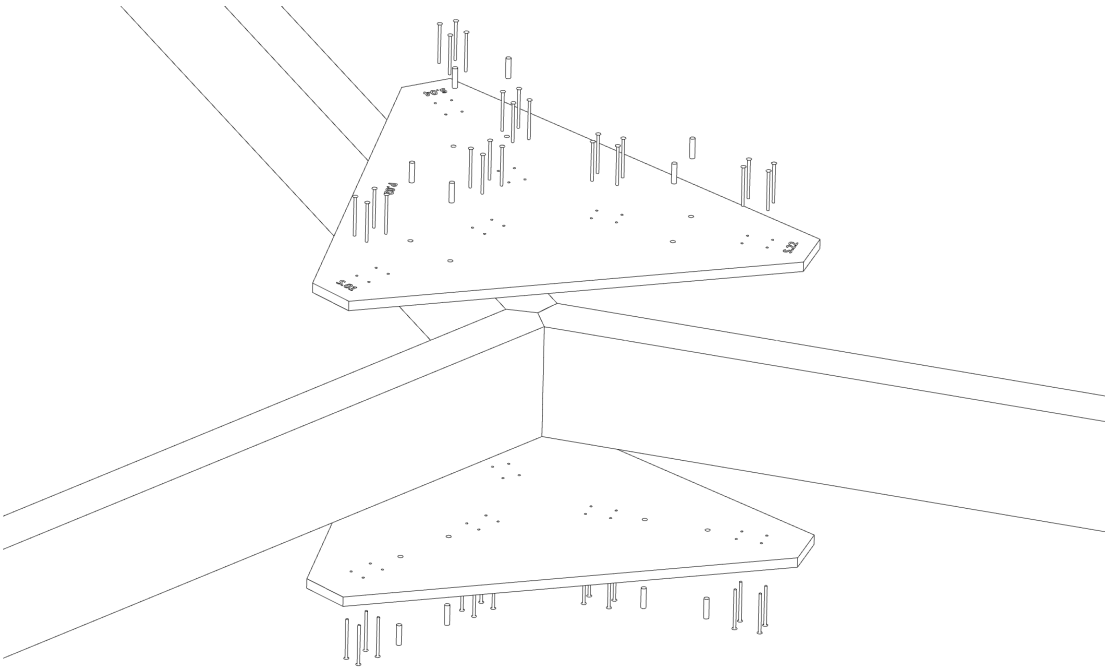
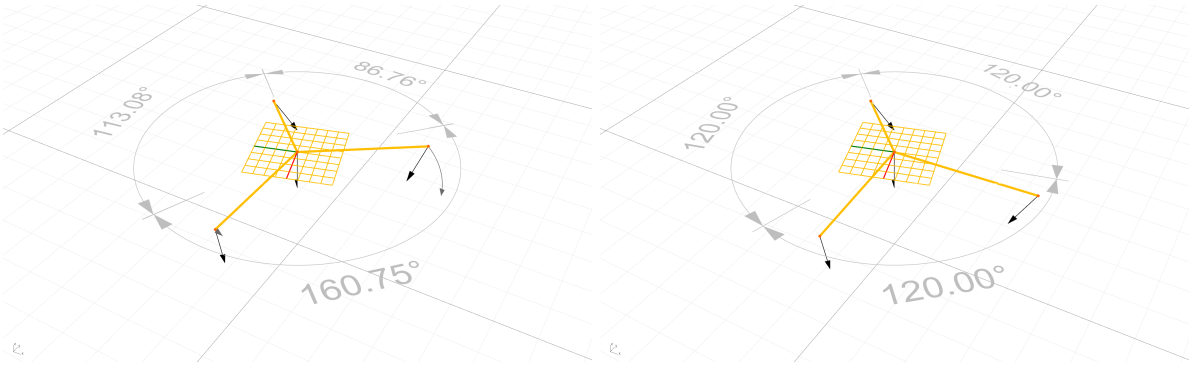


Figure 1.9: Detail of the flat node for the *CLC Xmesh pavilion* project.

1.2.2 120-3D nodes

Let us now consider a 3D node of valence 3. The node is said to be a *120-3D node* if its 3 edges, when projected into the node's normal plane, form together 3 angles of 120 degrees. The Figure 1.10 represents a 3D node and a 120-3D node.

This property is a real benefit regarding fabrication. It is easy to conceive a standard design for a 120-3D connection. Let us now show how we use this advantage for the design of the 120-3D nodes in both IASS Xmesh pavilion and CLC Xmesh pavilion projects.



(a) An instance of a 3D Node (b) An instance of a 120-3D Node

Figure 1.10: Difference between a *3D node* and a *120-3D node*.

Let us start with the IASS Xmesh pavilion.
On the fabrication plan, the 120-3D nodes are standard. They are realized thanks to two pieces of aluminium and a screw. The two pieces are folded by press-forming. Before being formed, they look like a star with three branches, each one having a 4mm-wide slit in the middle. Once stamped, they are shaped like flat hexagons with three tooth-like folds on each other edge.
Regarding the ends of the beams that correspond to 120-3D nodes, they are shaped like 133mm-high hammerheads, with 4mm-wide slits cut into them. Two reasons explain this original shape. Firstly, three beams intersecting at a given 120-3D node form variable angles with the normal axis of the node. As a consequence, as the beams have the same cross-section height, their projection along the normal axis have different lengths. Shaping the ends of the beams like hammerhead help to absorb these differences. Secondly, thanks to this local shape alteration, we can have a generic design of the 120-3D node.
Let us note that some design features are missing on the Figure 1.11 - they have been added after that the edition of this Figure. Three 4mm-wide rectangular holes are cut on top of the 120-3D nodes, in the star-shaped aluminium part. In conjunction, 4mm-wide teeth are added on each side of the hammerhead of the beam. These beams' teeth fit into the stamped pieces' holes.
About the construction, this detail has been thought to be reversible, fast and simple enough by requiring only a screwdriver - similarly to the flat node. Firstly, the two folded hexagonal pieces are connected to the hammerhead-like ends of three beams by interlocking their slits together. Then, a unique screw spans between the two pieces and constricts them against the ends of the beams. Thanks to the unique height of the hammerhead-like ends, all the screws of nodes can be the same.
Finally, this technological solution determines the mechanical behaviour of the 120-3D node. Extension of the beams, contraction, lateral movements and rotations according to their X and Y axis are blocked by contact around the imbrication of the slits of the beams and the node. Rotation according to the Z axis is particularly blocked thanks to the additional teeth-holes features described above, along with the contact around the imbrication of the slits. In Section 3.3, we will describe how we model this 120-3D node in the IASS structural model, based on these kinematic considerations.

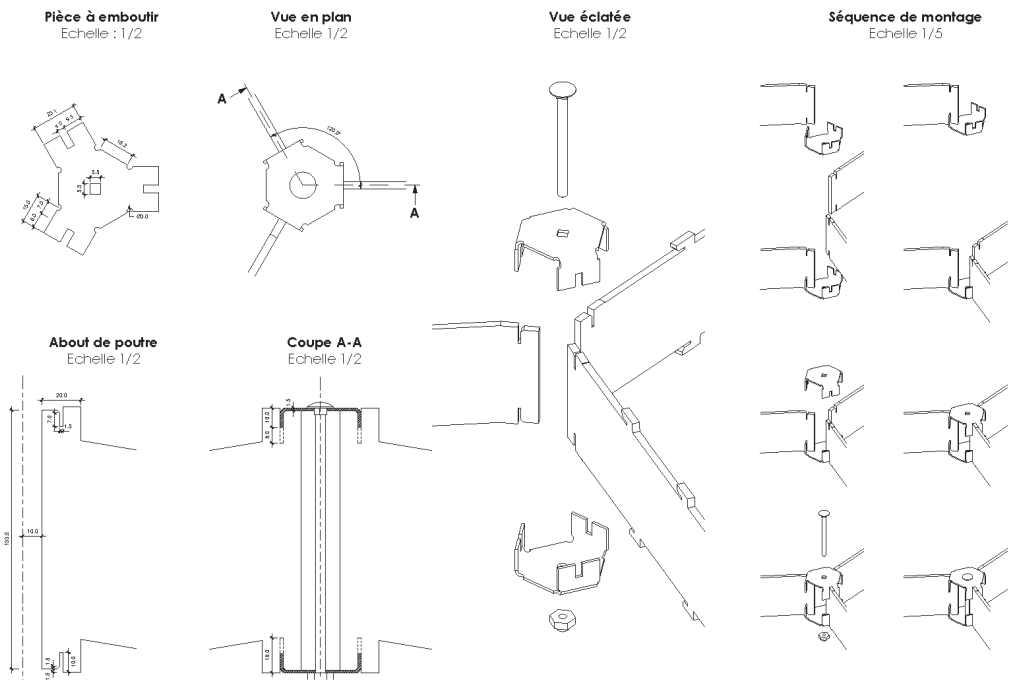


Figure 1.11: Detail of the 120-3D node for the *IASS Xmesh pavilion* project. Image kindly shared by our architects colleagues G. Jami, T. Lenart, A. Le Pavec and S. Zerhouni.

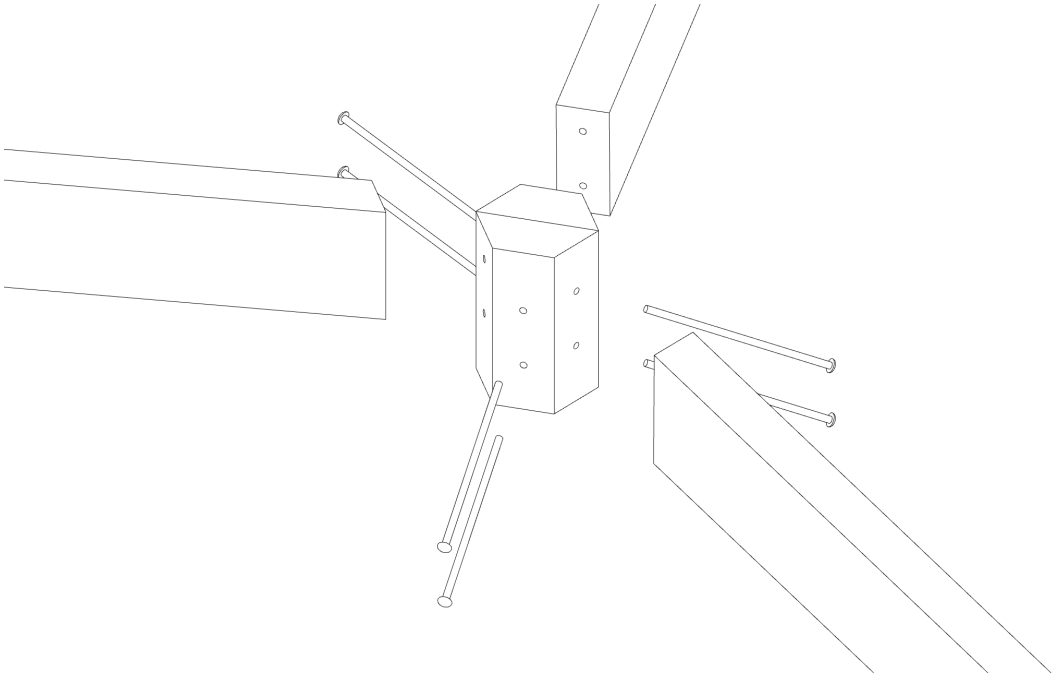


Figure 1.12: Detail of the 120-3D node for the *CLC Xmesh pavilion* project.

For the CLC Xmesh pavilion, the 120-3D nodes are materialized with wood.
They are made of two prismatic pieces of wood that are assembled together to form hexagonal prisms thanks to several 24mm-long screws. When three beams are connected to the node, six screws pass through it. When two beams are connected to the node, four screws pass through it. The two screws passing through a given beam are parallel. Their common direction is not parallel to the center-line of the beam but form a variable angle, usually about 10 to 15 degrees. The distance between the two screws is usually about 60 to 90mm. The hexagonal section of the node is standard but the height and the holes drilled through the node vary. Various aspects have been taken in consideration to come to these results :

- *Compactness of the nodes*. For visual and fabrication purposes, the 120-3D must be the shortest as possible.
- *Collision of the screws*. The spatial distribution of the four or six screws in the nodes must prevent collision.
- *Angle between the screws direction and the center-lines of the beams*. In terms of resistance, it is not efficient to screw a piece of wood along the longitudinal direction of the material. In the case of this 120-3D node, it would not be efficient to have the pair of screws parallel to the center-line of the beam as it corresponds to the longitudinal direction of the material. It is much more efficient to form angle with the longitudinal direction - the most efficient configuration being that the screws are orthogonal to the longitudinal direction. Here, we could not be orthogonal to the longitudinal direction, but we could attempt to get closer to 45 degrees.
- *Lever arm between the screws*. Again, in terms of resistance, it is more efficient to have the largest distance between the two screws of a given beam. It increases the moment capacity of the connection.
- *Minimal distance between screws and surfaces of the wooden structural element*. To prevent the wood to get pulled apart, minimal distances between the screws inside the wooden piece and its surface must be respected. These minimal distances are based on technical documentation provided by our supplier of metal hardware.

On the construction plan, these wooden nodes will be built thanks to robotic assistance. A platform has been designed and built in order to hold the hexagonal node, as shown in the Figure A.2 in the Appendices. During the construction process, a robotic arm will grab the beams and place them against the nodes. Students will then have to screws the pieces together following directions that will have been drilled beforehand.
About the mechanical behaviour, we can observe that most degree of freedom of the beams, at the connections with the 120-3D nodes, are blocked thanks to the contact between the wooden pieces and the arrangement of the pairs of screws. Extension of the beams, lateral movements and rotations according to their X and Y axis are blocked by the screws. Contraction is blocked by contact. On the rotation of the beams according to their Z axis is relatively free as the pairs of screws are aligned according to these axis and the lateral distance of contact (60mm) is relatively small. In Section 3.3, we will describe how we model this 120-3D node in the CLC structural model, based on these kinematic considerations.

1.3 Proof of the existence of an Xmesh

In [12], X. Tellier *et al.* give a proof of the existence of the Xmesh described in the Section 1.2. The Figure 1.13 illustrates this proof. We give some brief explanations about it, however further details are provided in [12]. The proof is developed in the *asymptotic* case, that is to say when the characteristic size of the mesh gets smaller and smaller, such that we can use notions of continuous differential geometry in approximation.
We consider a series of planar hexagonal meshes with decreasing faces sizes approximating S, such that one family of edges is aligned with a principal curvature direction of S. Hexagonal meshes are represented in blue in the top row of the Figure 1.13. Asymptotically, the hexagonal faces of the mesh tend to be symmetric and to get inscribed in local Dupin indicatrices of S, that are either an ellipses if S is *synclastic* or hyperbolas if S is *anticlastic*. The synclastic case is represented on the left part of the Figure while the anticlastic case is represented on the right of it.
We can now construct a new hexagonal mesh from this instance. For each hexagon of the mesh, we pick three non adjacent vertices A, C and E and we define their *Fermat point* P. We also define three segments $[AP]$, $[CP]$ and $[EP]$. We hence get a new hexagonal mesh, represented in orange. Furthermore, we can assign normal vectors to each of these vertices. Working in the *space of normals*, we can do some geometric construct to guarantee that the orange mesh satisfies all the properties of an Xmesh.

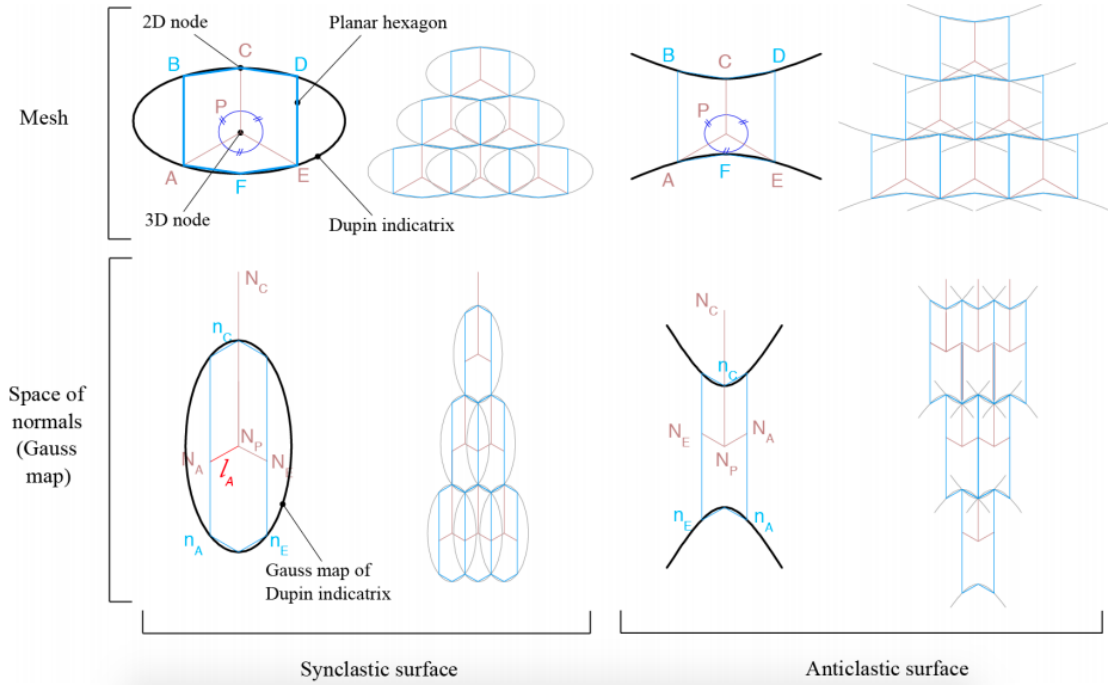


Figure 1.13: *Asymptotic proof* of the existence of an Xmesh. Image kindly shared by X. Tellier.

Chapter 2

Optimization

In the Chapter 1, we described various geometric notions about meshes and one particular mesh, the *Xmesh*. We pointed out some characteristics of meshes like torsion free-nodes, 120-3D nodes and others. We saw that these geometric features are beneficial construction-wise as we can then conceive generic connections which can help reducing design complexity and fabrication costs. However, generating a non-trivial mesh featuring such beneficial geometric properties, which we will call a *rationalized mesh*, appears to be a difficult challenge.

One way to get a rationalized mesh could be to build it by *propagation*, which is a method that falls into the group of the *pre-rationalization* methods.
In Section 1.3, we saw that geometric features can be obtained on hexagonal meshes by local asymptotic geometric construction. We could therefore think about a way to automatically generate a hexagonal mesh that would satisfy these conditions.
Let us assume that we have a rationalized mesh. If we can extend this existing mesh with additional mesh elements that satisfy the same beneficial properties, then we can extend the mesh. However, this approach rests on the assumption that the generation of new parts from existing geometric data is fully determined and can be automated. Still, finding a way to formulate the propagation as a fully determined geometric problem is not trivial, and we did not manage to define such an automatic generation. In addition, since the additional part is fully determined, the designer does not have much form-freedom.

As a consequence, we transfer our efforts onto another method which falls into the group of the *post-rationalization* methods.
Indeed, another way to get a rationalized mesh is to start from some ordinary mesh, possibly based on a target surface. We then have to bring a special effort to alter it in order to obtain the beneficial characteristics. In order to drive the transformation of the geometry towards a rationalized mesh, *geometric optimization* can be here used.
In this chapter, we explain how we can implement such a method to produce an *Xmesh* as defined in Section 1.2.

2.1 General software framework

In this section, we describe the various software that we have used to implement our post-rationalization method.

2.1.1 Rhinoceros3D and Grasshopper

In order to work with 3D geometries, we use the software *Rhinoceros3D* along with its extension *Grasshopper*.
The first is a 3D modeling software essentially based on the concept of *NURBS (Non-uniform rational B-splines)*. With its graphic user interface, it allows users to design 3D shapes like complex surfaces for architectural projects (see Figure 2.1).
The latter is a visual programming extension to *Rhinoceros3D*. It gives to users a set of *components* that are graphical interface of *Rhinoceros3D*'s API (Application programming interface). By chaining the outputs of components to the inputs of others, users can visually develop programs to generate complex parametric designs.

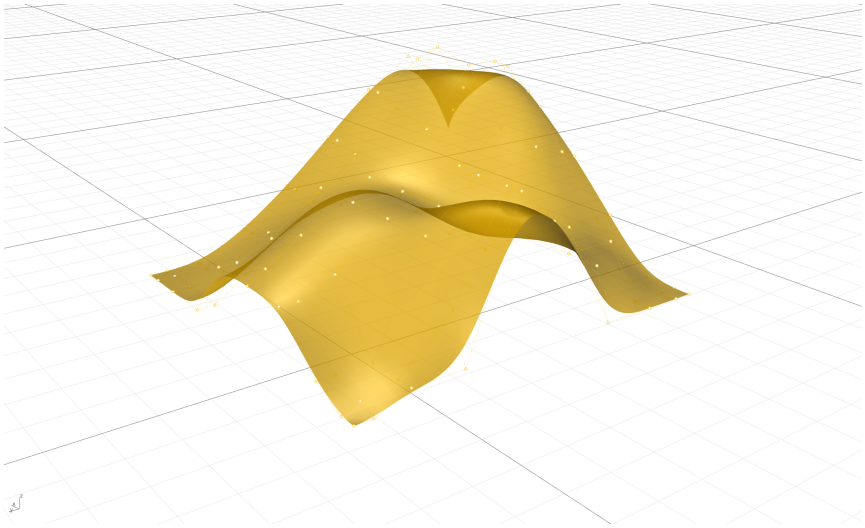


Figure 2.1: An instance of a *NURBS* surface in *Rhinoceros3D*

Many libraries can be imported into *Grasshopper* to extend the default set of components with other ones, providing functionalities like structural analysis and optimization to users.
Structural aspects will be treated in Chapter 3. There, we will describe the way we make a use of the library *Karamba* to analyze the structure of our projects.
In what follows, we are going to focus on geometric optimization, using the library *Kangaroo2*. Before getting into details with *Kangaroo2*, we give an insight of *Shape-Up*, the algorithm on which the library is based.

2.1.2 Shape-Up

Shape-Up is an algorithm developed by Sofien Bouaziz (see [1]) to optimize *geometric data* in order to satisfy *shape constrains*.
Basically, the geometric data that are processed are sets of points, which are linked to each other as parts of larger geometries. For instance, points can correspond to the vertices of a mesh, in which case they are related by their connectivity.
Shape constraints can be diverse. In the Chapter 1, we describe several local properties of meshes like torsion-free nodes or 120-3D nodes, which we could want to obtain or preserve on our geometries. Such features are examples of shape constraints as they can be formulated as geometric constraints on a set of points related to each other.

In order to achieve geometric optimization, *Shape-Up* algorithm works in two steps. They are based on respectively two key concepts: the first one on *shape projection operators* and the second one on *shape proximity function*.
Let us consider a points-based geometry and shape constraints that we would like to satisfy. The geometry can be seen as an element $\mathbf{x}$ of a set of configurations $\mathcal{X}$ (of potentially-high dimension). Shape constraints can be seen as subsets $\mathcal{C}_1, ..., \mathcal{C}_m$ of $\mathcal{X}$.
The algorithm works in two steps.

1. *Projection*. The current geometric configuration $\mathbf{x}$ is projected on each of the constraints' admissible spaces $\mathcal{C}_1, ..., \mathcal{C}_m$ to get the corresponding projected configurations $\mathbf{x'}_1, ..., \mathbf{x'}_m$. To be able to complete this step, it implies that for each constraint j, there exists a projection operator on $\mathcal{C}_j$. In [1], the authors give various examples of shape constraints and their corresponding projection operators.
2. *Least square minimization*. To quantify the distance between a given configuration and the optimal one, a shape proximity function is defined as a weighted sum of the distances between the configuration and the shape constraints' admissible spaces. Given the projected geometric configurations $\mathbf{x'}_1, ..., \mathbf{x'}_m$, a new geometric configuration $\bar{\mathbf{x}}$ is computed as a weighted combination of the projections such that the chosen coefficients minimize the shape proximity function in the sense of the least squares.

By construction, at each step this new configuration $\bar{\mathbf{x}}$ reduces (or maintains) the value of the shape proximity. However, there is no guarantees that the value of the proximity function strictly decreases at each step nor that there exists an optimal value satisfying simultaneously all the shape constraints. However, this algorithm has shown to be very powerful in practice. We will give in Section 2.4 some optimization results to illustrate these performances.

To get further explanations about *Shape-Up*, we encourage the readers to look at [1].

In the next section, we focus on the *Grasshopper* library of geometric optimization *Kangaroo2*, which is based on *Shape-Up*.

2.2 Geometric optimization with Kangaroo2

With its library *Kangaroo2*, Daniel Piker provides an easy and unified framework to define and solve geometric optimization problems within the interface of *Rhinoceros3D* and *Grasshopper*.
The library is essentially separated in two sets of *Grasshopper* components.

- *Goals*. A first set of components allows to apply geometric constraints on various *Rhinoceros3D* geometric objects like points, lines, surfaces, meshes and others. These components are called goals. To make the connection with *Shape-Up*, applying a goal to a part of the geometry that we want to process is equivalent to defining a shape constraint over a set of geometric data.
- *Solvers*. A second set of components allows to assemble all these goals into a unique optimization problem and to solve it. The solvers operate according to the two steps method of *Shape-Up* as described in Section 2.1.2.

In many cases, the set of default goals components is rich enough so that users can define their geometric optimization problem corresponding to the constrained geometry they want to obtain. However, when their geometric constraints are relatively original, they may need to define their own custom goals components. For instance, there is no default goal that constrains the nodes of a mesh to be torsion free. In such cases, users must create their own goal writing a relatively short *C#* script.

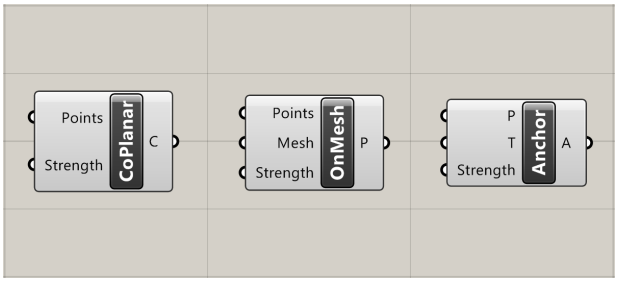


Figure 2.2: Default goal components from the library *Kangaroo2*. *CoPlanar* constrains the points *Points* to be co-planar. *OnMesh* constrains the points *Points* to be on the mesh *Mesh*. *Anchor* constrains the point P to remain where the point T is.

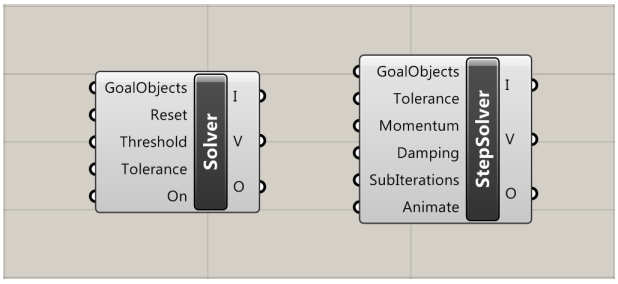


Figure 2.3: Solver components from the library *Kangaroo2*.

2.2.1 Custom goals within Kangaroo2 framework

Like many other *Grasshopper* libraries, *Kangaroo2* comes with its *DLL* (Dynamic Link Library) in which all its code is written. Using a *C#* component (see Figure 2.4) and loading the DLL of *Kangaroo2*, users can then access to the public content of the code to write their own custom goals.

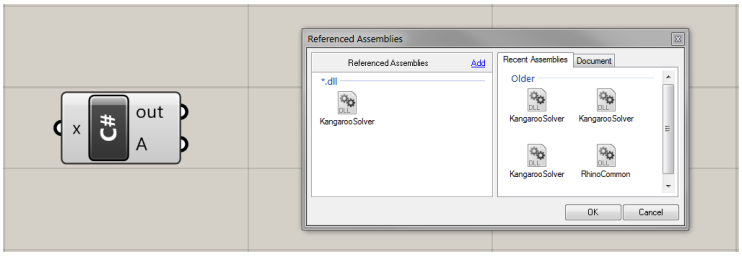


Figure 2.4: *C# Grasshopper* component loaded with the *DLL* of *Kangaroo2*.

In *Kangaroo2*, goals are based on the key-concept of *particles*. A particle is a plane of the 3D space (*Plane* in *Rhinoceros3D* like in Figure 2.5) which is equivalent to having a point of the 3D space (*Point3D*) corresponding to the plane's origin and a vector (*Vector3D*) corresponding to the plane's normal.

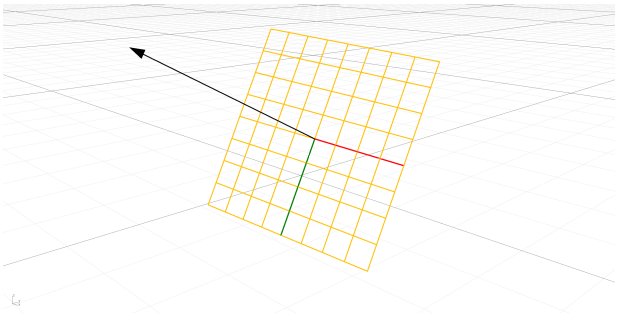
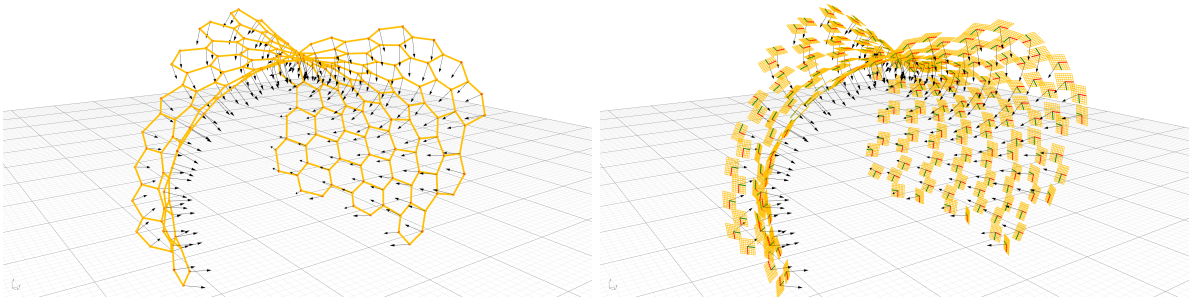
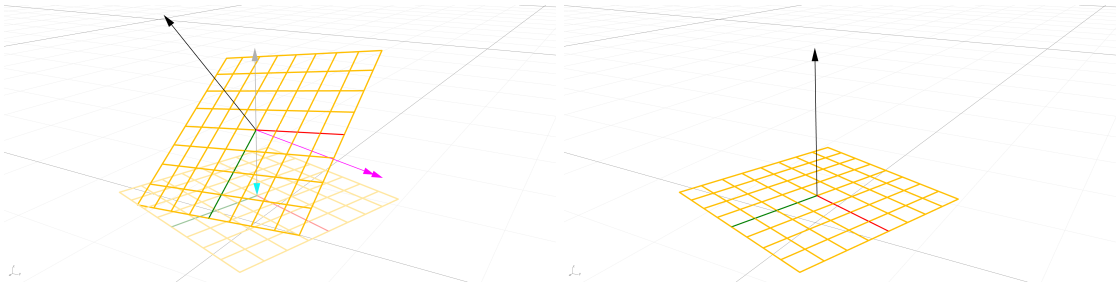


Figure 2.5: An instance of a *Plane* in *Rhinoceros3D & Grasshopper*. The tilted grid represents the local *XY* plane, the red segment the local *X* direction, the green one the local *Y* direction and the black arrow represents the local *Z* direction.



(a) An instance of a mesh (b) Its corresponding set of particles

Figure 2.6: Translation from a mesh to a set of particles.



(a) Current configuration of a particle (b) Configuration after applying transformations

Figure 2.7: Transformations to apply to a particle to satisfy a geometric constraint. The green arrow corresponds to the *translation* and the magenta double arrow corresponds to the *rotation*. Both are movements that we want to apply to the particle in order to satisfy a certain geometric constraint.

Defining our own custom goal requires two conditions.

1. *Translating to particles*. Users must be able to describe their geometry in terms of particles. Whatever the geometric objects (points, surfaces, curves, meshes...) and whatever the geometric constraint (coplanarity, area, projection on a surface, ...) that we want to apply on them, goals must be formulated via particles. For instance, when the geometric constraint that we want to model is applied on a set of points, it is quite natural to associate each point with the origin point of one particle. But, when the constraint deals with surfaces or meshes, the translation is no longer natural. Usually, when working on a surface, we can resort to its controls points and associate these points to particles' origin points. When constraining meshes, we can resort to nodes and associate the nodes with particles' planes (see Figure 2.6).
2. *Defining translation and rotation objectives*. Users must be able to formulate in terms of particles' translations and rotations how the geometry shall get transformed in order to reach the targeted geometric constraint. In other words, given a configuration of the particles, users must provide two objective *Vector3D*s for each particle: a first *Vector3D* called *Move* corresponding to the translation and a second one called *TorqueMove* corresponding to the rotation that the particle shall do such that the overall geometry satisfies the geometric constraint (see Figure 2.7).

If these two conditions are satisfied, then users can define their own custom goals. The Algorithm 2.1 gives the pseudo-code of a custom goal. In terms of *C#* programming language, a goal is a *class*. As any other class, it has a *constructor* and *methods*. We consider here a goal working with n_p particles.

1. *Constructor*. The constructor is called once to initialize the positions and the orientations of the n_p particles ($PPos$ and $InitialOrientation$ arrays), as well as their objective translations and rotations ($Move$ and $Torque$ arrays) and the respective strength coefficient of these objectives ($Weighting$ and $TorqueWeighting$). This is where the translation of the geometry in terms of particles is made.
   Let us note that if the formulation of the geometric constraints actually does not need to use any rotation, we can simply omit to initialize the arrays $Torque$ and $TorqueWeighting$.

2. *Calculate method*. This method is called at each iteration of the optimization process. This is where the objective translations and rotations of the particles ($Move$ and $Torque$ arrays) are computed and updated according to their current values.
3. *Output method*. This method is also called at each iteration of the optimization process. It enables users to output geometries at each step of the solver.

Let us point out that, most of the time, in order to implement one constraint, we can conceive several goal classes with different ways of defining and calculating the $Move$ and $Torque$ variables. For instance, let us say we want to constraint four points to lie on a common plane. We must decide what is the common plane and compute it.
A first option is to compute the best plane in the sense of the least squares considering the norm L^2. This plane can be seen as the average plane between the four points. Then, to achieve the co-planarity of the four points, we can update the four $Vector3D$s of the $Move$ array with the vectors corresponding to the differences between the points in their current states and their projections on the average plane.
A second option is to compute an average plane considering the norm L^2 factored with a relative distance to the centre of mass of the points. Then, the four $Vector3D$ of the $Move$ array are computed in the same fashion as in the first option.
Both options are actually reasonable. There is not one that is necessary better than the other. The first solution is simple and it gives results that often correspond to what we would like to get. However, if one of the four points is an outlier, then the result that we would get with second option would probably be preferred to the one we would get with the first option.

Listing 2.1: Structure of the code of a custom goal within *Kangaroo2* framework

```
public class CustomGoal : GoalObject
{
    // Class attributes
    int np; // Number of particles
    object ObjectToOutput;

    // Constructor
    public CustomGoal(Object Input1, Object Input2, ...)
    {
      // Initialization of particles
      PPos = new Point3d[np]; // Array of particles' positions
      InitialOrientation = new Plane[np]; // Array of particles' orientations
      ... // Script to initialize PPos and InitialOrientation

      // Initialization of translation and rotation goals
      Move = new Vector3d[np];
      Torque = new Vector3d[np];
      Weighting = new double[np];
      TorqueWeighting = new double[np];
    }

    // Calculate method
    public override void Calculate(List<KangarooSolver.Particle> p)
    {
      // Data from particles in current state
            // 0th particle's position and orientation
      Point3d Pos0 = p[PIndex[0]].Orientation.Origin;
      Plane Orientation0 = p[PIndex[0]].Orientation;
      ... // Script to get more particles' data

      // Calculations
      ...

      // Updates of translation and rotation goals
      Move = ...;
      Torque = ...;
    }

    // Output Method
    public override object Output(List<KangarooSolver.Particle> p)
    {
      // Output
      return ObjectToOutput;
    }
  }
```

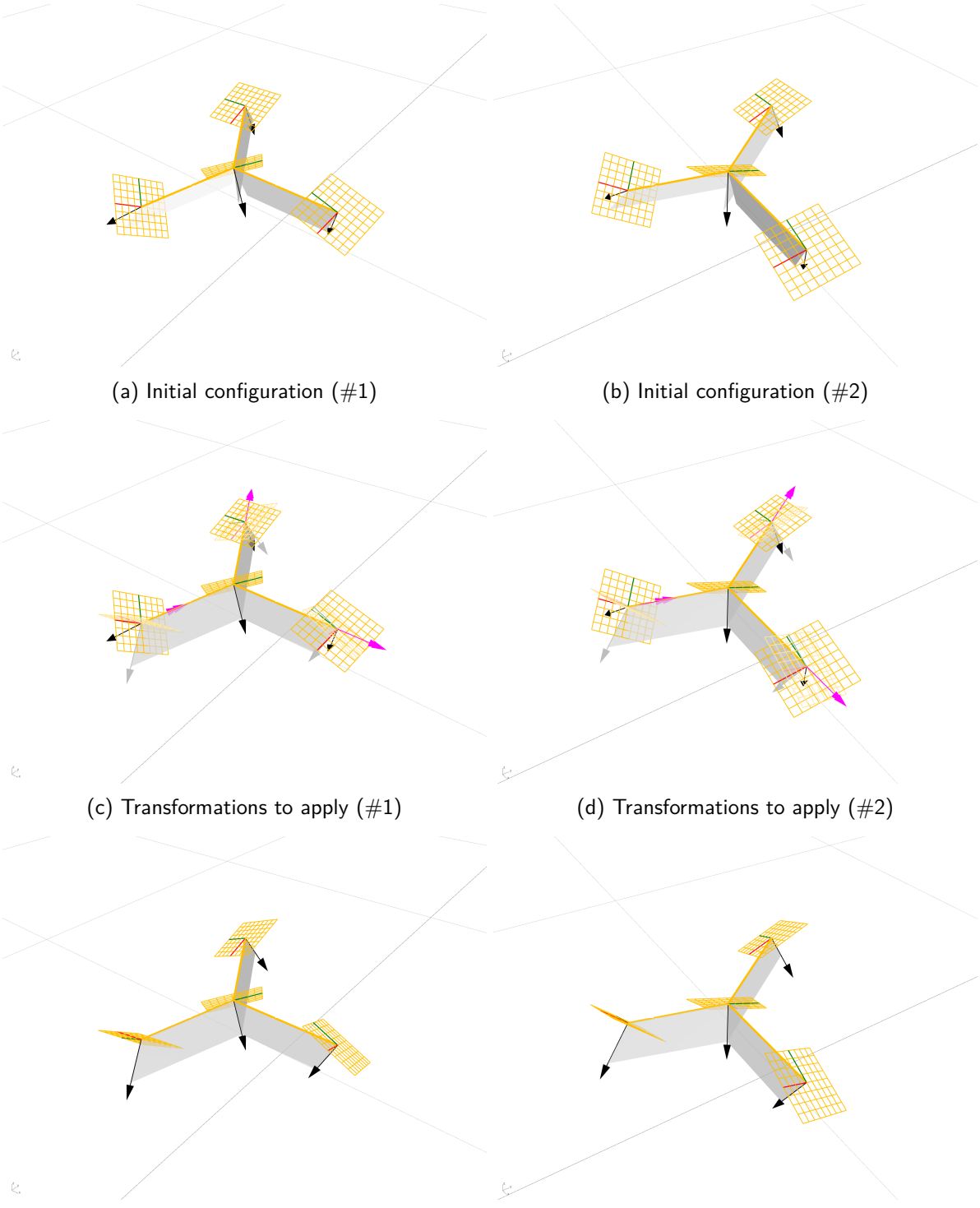
2.2.2 Torsion-free nodes custom goal

In this section, we give the definitions of the custom goal we implemented in order to constraint the nodes of a mesh to be torsion-free, as defined in Section 1.1.2.

Let us consider a mesh $\mathcal{M}$ on which we want to prescribe that a node of valence 3 shall be torsion-free. The torsion-free property is a local property of $\mathcal{M}$. It affects one chosen node, which we will call *main node*, and its three neighboring nodes, which we will call *neighbors*. The main node is the node that shall be torsion free. Each node of the neighbors is a node that shall have its normal vector contained in the plane defined by the main node's normal vector and the line connecting its origin point to the main node's origin point.

There is no default goal to prescribe this geometric condition, therefore we have to create our own custom goal. We describe now one way to program a *TorsionFree* custom goal in *Kangaroo2*.
The inputs are a *Plane*, called *MainPlane*, for initializing the particles corresponding to the main node, a list of three other *Plane*s, called *NeighborsPlanes*, for initializing the particles corresponding to the three neighbors, and a *double*, called *Strengh*, to set the relative importance of the goal.
Let us now focus on the *Calculate* method. Here is how it works. Get the current geometric configuration of the four nodes. Take one of the three neighbors, define a reference plane that is normal to line connecting the neighbor to the main node, project on it the normal vector of the main node and the one of the neighbor and get the angle formed by these projected normal vectors. This angle corresponds to the angular misalignment between the two nodes' normal vectors. If we want the main node to be torsion-free, it shall be equal to zero. In order to make it equal to zero, we can rotate each node by half of this angle, in opposite senses, around the axis defined by line connecting the two nodes. Actually, instead of projecting the normal vectors on reference planes to measure the misalignment angle and then define torques vectors, we can also compute the cross-products of the normal vectors of the main node and each one of the neighbors to get torques vectors. This way requires fewer operations and is actually how we implemented our *TorsionFree* custom goal (see Figure 2.8 and Algorithm A.1 in the Appendices).

It is easy to write a goal that prescribes a node of valence 2 to be torsion-free. The inputs are the same as above. However *NeighborsPlanes* is a list of two *Plane*s. In the *Calculate* method, we compute the direction at the intersection of the two planes defined respectively by the line going from the main to one of the two neighbors and this neighbor's normal vector. Then we compute the direction and the angle of rotation we must apply to the main node's normal vector in order to align it with this direction.



(a) Initial configuration (#1)

(b) Initial configuration (#2)

(c) Transformations to apply (#1)

(d) Transformations to apply (#2)

(e) Configuration after transformations (#1)

(f) Configuration after transformations (#2)

Figure 2.8: *Torsion-free* custom goal by rotating nodes. Left and right columns correspond to two different points of view of the same objects. On the second row, configuration in lighter colors corresponds to the targeted configuration. The magenta arrows correspond to the *rotations* we want to apply to the particles in order to satisfy the torsion-free constraint. For simplicity and legibility, We do not represent the rotation of the main node, however our goal does apply such a rotation too.

2.2.3 Flat nodes custom goal

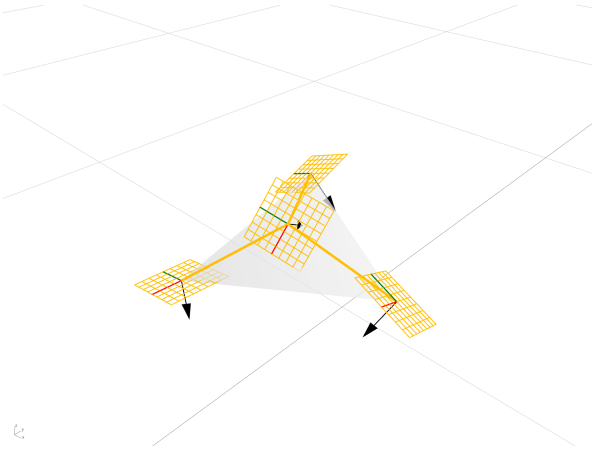
In this section, we give the definitions of the custom goals we implemented in order to constraint chosen nodes of a mesh to be flat nodes, as defined in Section 1.2.

Let us consider a mesh $\mathcal{M}$. Let $\mathcal{N}_f$ be the set of the nodes of $\mathcal{M}$ that we would like to be flat. The flat node property is a local property of $\mathcal{M}$. It affects one chosen node in $\mathcal{N}_f$, which we call *main node*, and its three neighboring nodes, which we call *neighbors*.

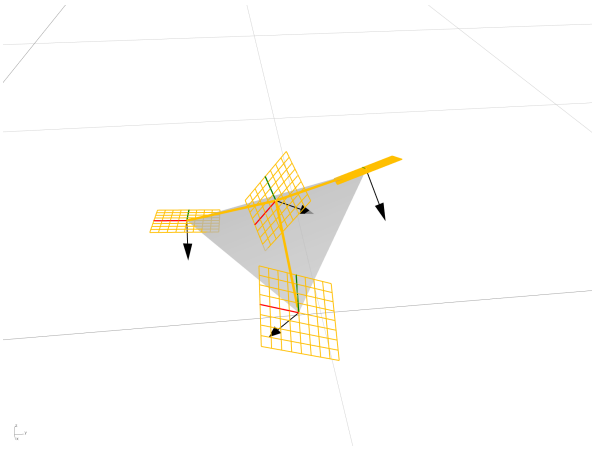
Indeed, let us take a main node from $\mathcal{N}_f$ and its three neighbors. The flat node constraint corresponds to the concurrence of two sub-constraints that we can associate respectively with two goals.

1. *Coplanarity*. The main node and its three neighbors shall be coplanar, that is to say they shall all lie on the same common plane. In order to prescribe this condition, we use the default goal component *CoPlanar* from the *Kangaroo2* library.
   Its inputs are *Points3D*, called *Points*, and a *double*, called *Strengh*. For our use, the Points that we give as inputs are the origin points of the main node and of the three neighboring nodes.
   The goal constraints the particles corresponding to the input points to move towards a computed average plane. The Strength sets the relative importance of this goal relatively to others.
2. *Orthogonality*. The main node's normal vector shall coincide with the normal vector of the common plane. There is no default goal to prescribe this condition, therefore we have to create our own custom goal. As explained before, there is not a unique way of programming a goal to achieve a constraint. Here are two different solutions.
   (a) *Rotating main node*. A first way to deal with the orthogonality sub-constraint is by rotating the normal vector of the main node around its position such that the vector gets aligned with the normal vector of the plane defined by the three neighboring points (see Figure 2.9 and Algorithm A.2 in the Appendices). In other words, the plane which contains the three neighbors position points is taken as the common plane. We prescribe the main node to have its normal vector aligned with the normal of this common plane. In this process, no particle is moved, only particles' orientations are affected.
   (b) *Moving neighbors*. A second way to deal with orthogonality is by rigidly rotating the origin points of three neighbors around the main node such that the neighboring nodes all lie on the plane of the main node (see Figure 2.10 and Algorithm A.3 in the Appendices). In other words, the common plane is here taken as the main node's plane. We prescribe the neighboring nodes to move to this plane in a solid rotation fashion. In this process, no particle has its orientation affected, particles are only moved.

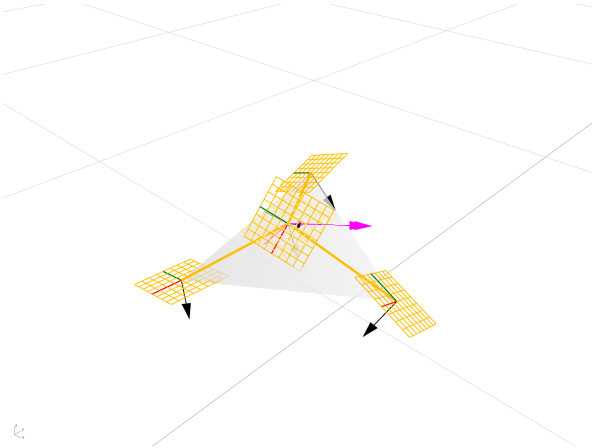
   Again, one solution is not better than the other. During the optimization process, the first solution is likely to prevail when one or several neighboring nodes are constrained not to move much. It will be easier to alter the normal of the main node rather than tilting all the neighboring nodes if some of them are blocked. However, we also benefit significantly from the second solution as it encourages neighboring nodes to move from their original positions and therefore to explore the neighboring space around the original positions. For these reasons, we use both solutions in our optimization runs, as complementary solutions.



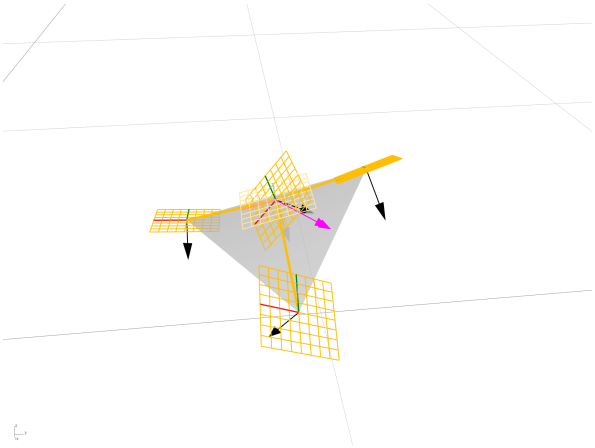
(a) Initial configuration (#1)



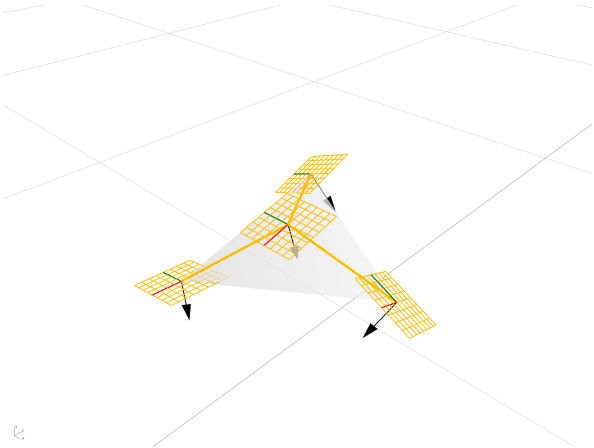
(b) Initial configuration (#2)



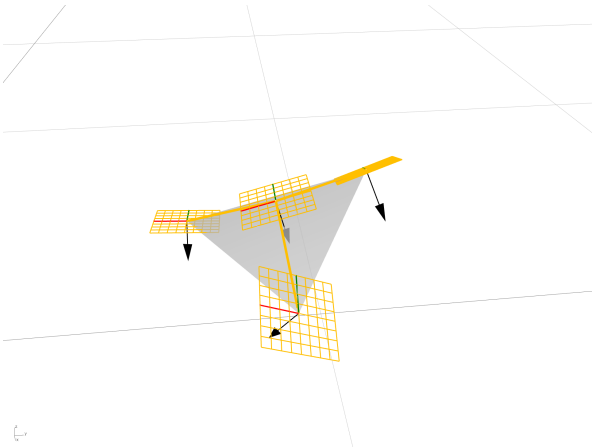
(c) Transformations to apply (#1)



(d) Transformations to apply (#2)

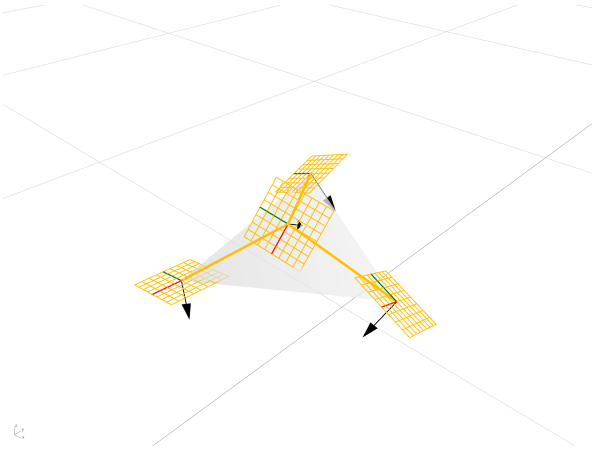


(e) Configuration after transformations (#1)

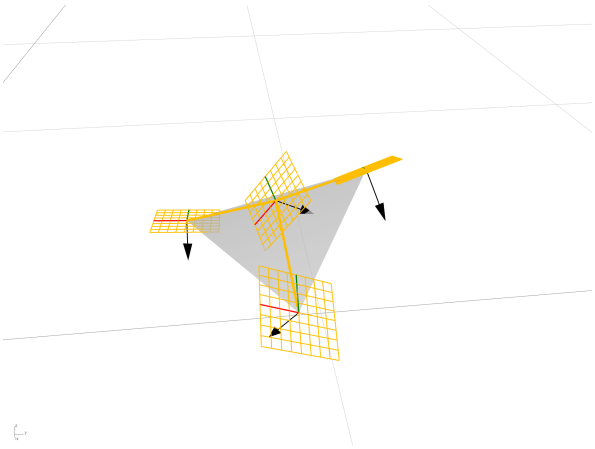


(f) Configuration after transformations (#2)

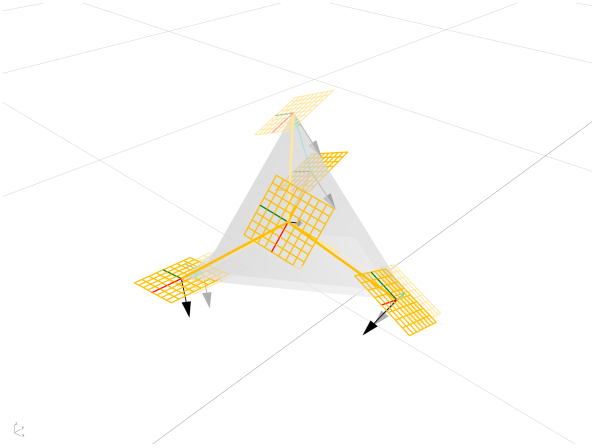
Figure 2.9: *Orthogonality* custom goal by rotating main node. Left and right columns correspond to two different points of view of the same objects. In Figures of the second row, configuration in lighter colors correspond to the targeted configuration. The magenta double arrow corresponds to the *rotation* we want to apply to the main particle in order to satisfy the orthogonality constraint.



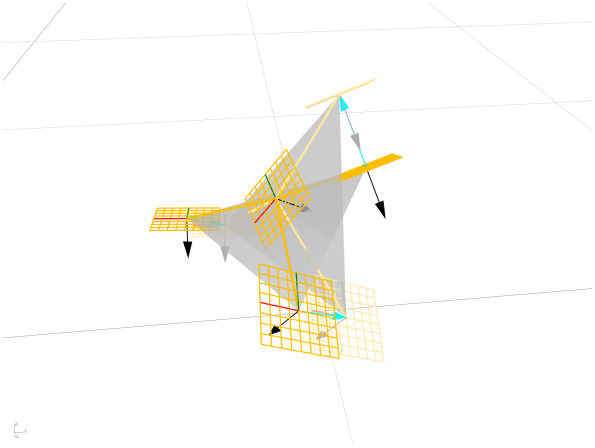
(a) Initial configuration (#1)



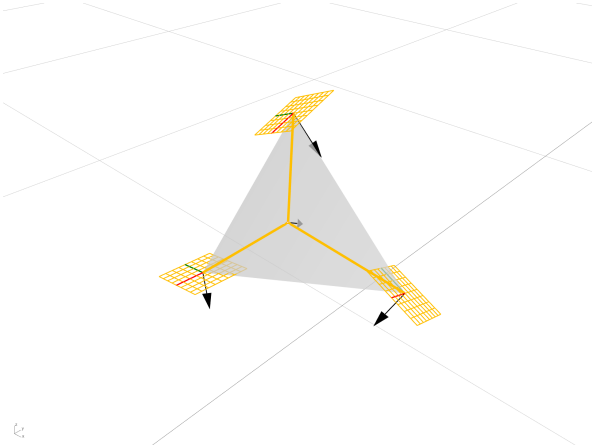
(b) Initial configuration (#2)



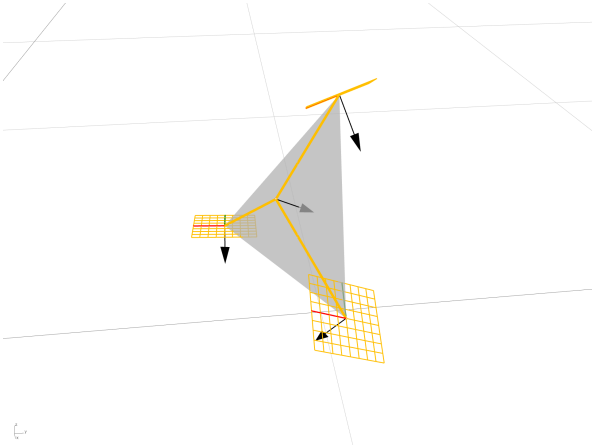
(c) Transformations to apply (#1)



(d) Transformations to apply (#2)



(e) Configuration after transformations (#1)



(f) Configuration after transformations (#2)

Figure 2.10: *Orthogonality* custom goal by moving neighboring nodes. Left and right columns correspond to two different points of view of the same objects. In Figures of the second row, configuration in lighter colors corresponds to the targeted configuration. The blue arrows correspond to the *translations* we want to apply to the neighboring particles in order to satisfy the orthogonality constraint.

2.2.4 120-3D nodes custom goal

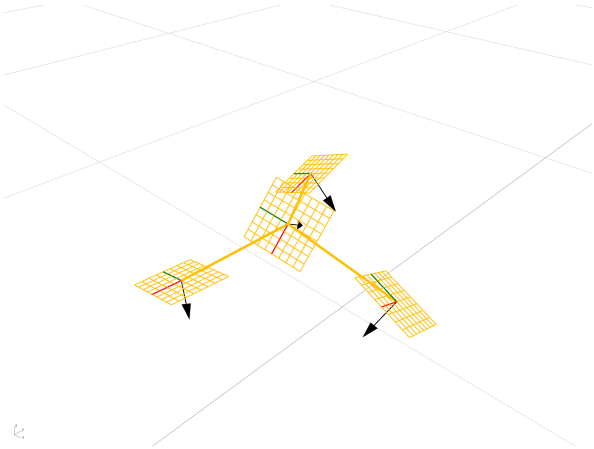
In this section, we give the definition of the custom goal we implemented in order to constraint the node of a mesh to be a 120-3D node, as defined in Section 1.2.

Let us consider a mesh $\mathcal{M}$. Let $\mathcal{N}_{120}$ be the set of the nodes of $\mathcal{M}$ that we would like to be 120-3D. Like the torsion-free node or flat node properties, the 120-3D node property is a local property of the mesh. It affects one chosen node in $\mathcal{N}_{120}$, which we call again *main node*, and its three neighboring nodes, which we call *neighbors*. The main node is the 120-3D node. Each pairs of lines connecting two neighbors to the main node shall form an angle of 120 degrees when projected in the plane orthogonal to the main node's normal vector.

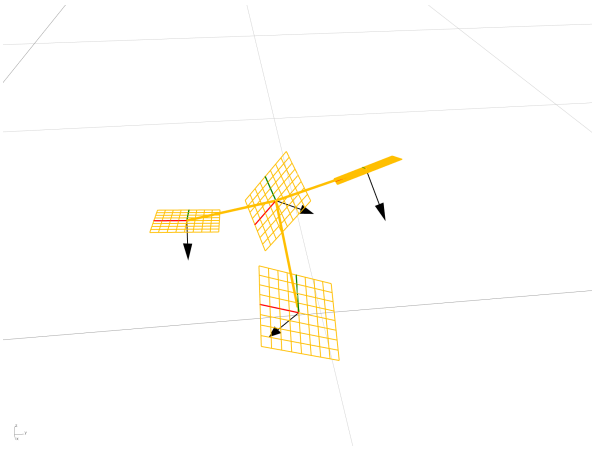
Like the two previous goals, there is no default goal to prescribe the 120-3D node geometric condition, therefore we have to create our own custom goal. Actually, we are going to present two different versions of this goal.
For both versions, the inputs are a *Plane*, called *MainPlane*, for initializing the particle corresponding to the main node, a list of three other *Plane*s, called *NeighborsPlanes*, for initializing the particles corresponding to the three neighbors, and a *double*, called *Strengh*, to set the relative importance of the goal.
Let us now focus on the *Calculate method* of both goals versions. They both start with the same first steps. Get the current configuration of the four nodes. Take the plane orthogonal to the main node's normal vector as a reference plane. Project the neighboring nodes on the reference plane. For each pair of projected neighbors measure the angle between the two lines connecting these two neighbors to the main node. If we want the main node to be a 120-3D node, this angle shall be equal to 120 degrees. In order to get the right angle value, each version of the goal has its approach.

(a) *Moving neighbors*. A first way is to rotate neighbors around the axis defined by the main node's normal vector such that each pair form 120 degrees in projection in the reference plane (see Figure 2.11 and Algorithm A.4 in the Appendices). In this method, the main node is not affected. Only the neighbors are moving.
(b) *Moving main*. A second way is to keep the neighbors at their position and to move the main node, in the reference plane, at the unique position where the three projected branches form three angles of 120 degrees(see Figure 2.12 and Algorithm A.5 in the Appendices). This unique position corresponds to the Fermat point of the triangle defined by the three projected neighbors. The Fermat point can be obtained by geometric construction. In this method, only the main node moves. The neighbors are not affected.

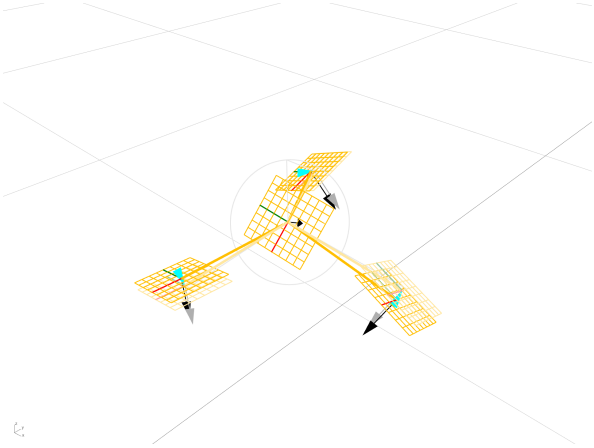
Again, one version of the goal is not better than the other, they are complementary. We use both solutions in our optimization runs.



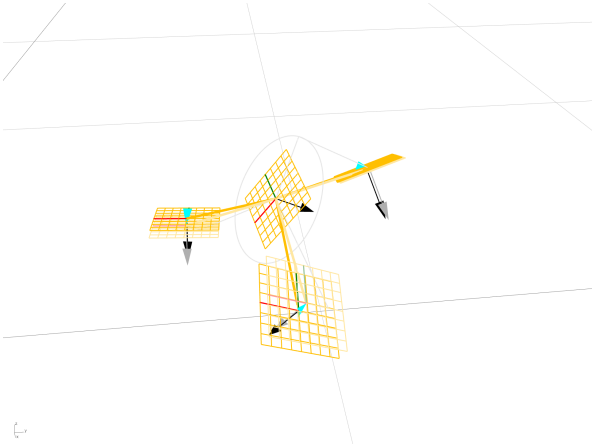
(a) Initial configuration (#1)



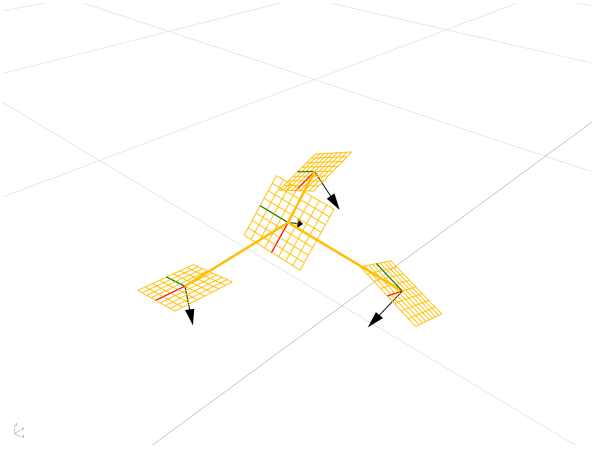
(b) Initial configuration (#2)



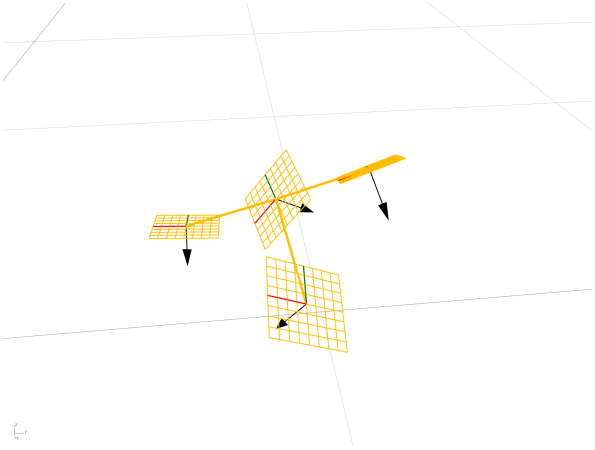
(c) Transformations to apply (#1)



(d) Transformations to apply (#2)

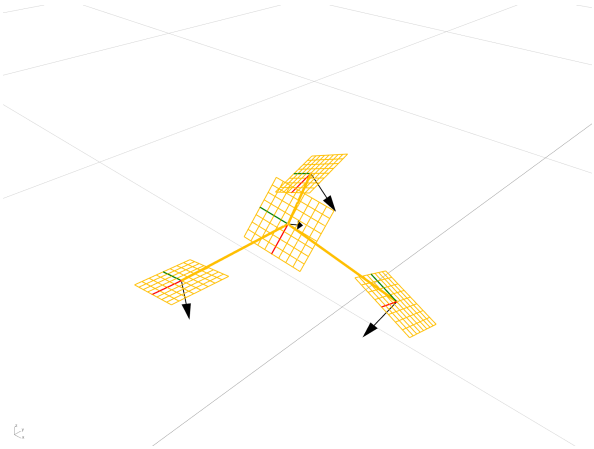


(e) Configuration after transformations (#1)

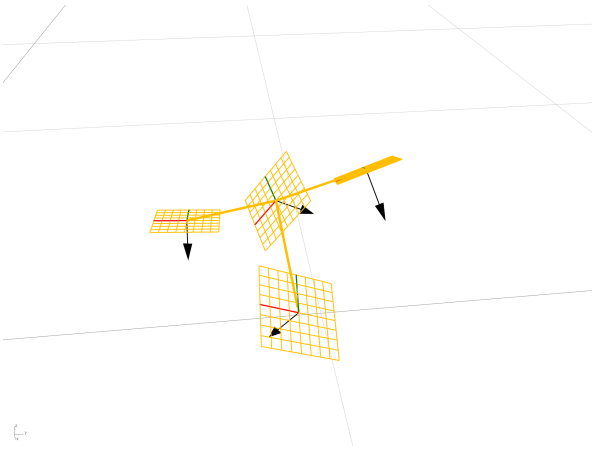


(f) Configuration after transformations (#2)

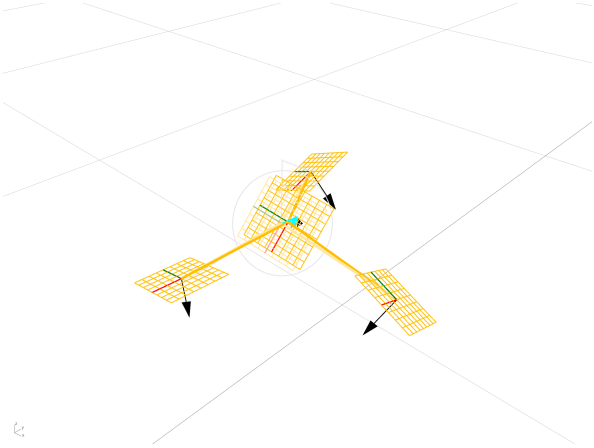
Figure 2.11: *120-3D nodes* custom goal by moving neighboring nodes. Left and right columns correspond to two different points of view of the same objects. In Figures of the second row, configuration in lighter colors corresponds to the targeted configuration. The blue arrows corresponds to the *translations* we want to apply to the neighboring particles in order to satisfy the 120-3D nodes constraint.



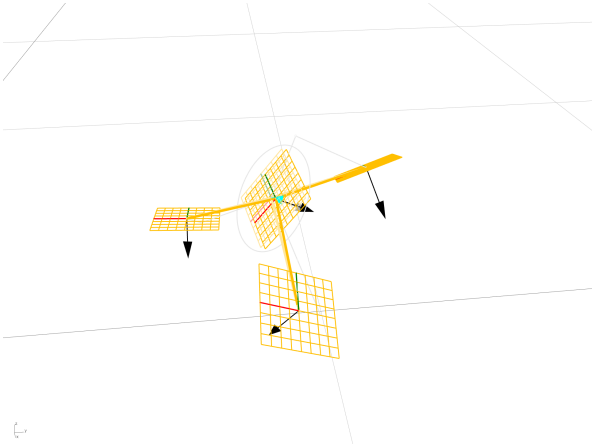
(a) Initial configuration (#1)



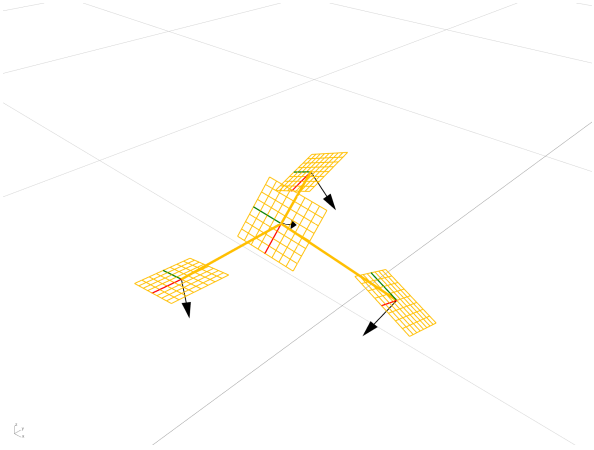
(b) Initial configuration (#2)



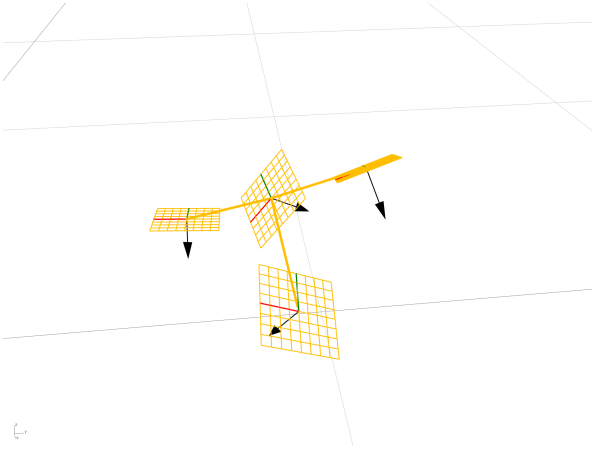
(c) Transformations to apply (#1)



(d) Transformations to apply (#2)



(e) Configuration after transformations (#1)



(f) Configuration after transformations (#2)

Figure 2.12: *120-3D nodes* custom goal by moving main nodes. Left and right columns correspond to two different points of view of the same objects. In Figures of the second row, configuration in lighter colors corresponds to the targeted configuration. The blue arrow corresponds to the *translation* we want to apply to the main particle in order to satisfy the 120-3D nodes constraint.

2.3 Initialization of the optimization

In order to run our geometric optimization process aiming at producing an Xmesh, we firstly need to initialize it with a hexagonal mesh. In this section, we describe the tool and the method we use to generate such an initial hexagonal mesh based on the lines of curvatures of a target surface. We explain how this method performs, why can it be consider as a good initialization method and what are its limits.

2.3.1 TMap

As we will see it in the next section, the method of initialization we use is based on the lines of curvatures (see Section 1.1.3) of a chosen surface. *Rhinoceros3D* does not integrate a tool for computing lines of curvatures of a surface. We have to import a specific library to the software. For our applications, we use the library *Tmap*, developed by the company *Evolute*, which offers a tool for computing such lines. Given a surface as an input, we can define the number of lines we want to get. The outputs are two sets of curves having ideally a grid-like shape.

More details about the library and the company can be found in their website [2].

2.3.2 Method based on lines of curvatures

We now present, step by step, the method we use to generate an initial hexagonal mesh. It is inspired by the method presented in [13]. Wang *et al.* describe a method based on the lines of curvatures to initialize a brick-wall-like hexagonal mesh to be then optimized to get planar. As we pointed out in Section 1.3 that the Xmesh can be based on planar-hexagons mesh, an initialization close to the brick-wall method would also be efficient for our geometry.

1. *Target surface*. The first step is to choose a target surface like the one of the Figure 2.14a. Usually, the target surface correspond to the formal expressiveness we want our mesh, and therefore our structure, to have. To make sure that the mesh that will result from the method that we are describing here will cover the surface we have in mind, we advice to extend a little the surface over its edges.
2. *Surface alterations*. We can wish the final mesh to have edges align with some of the edges of the surface. For instance, still referring to the example of the target surface of the Figure 2.14a, we can wish the final mesh to have horizontal edges on the world plane - which corresponds to the floor. As we will see in the next steps, some of the mesh edges will be aligned with the lines of curvatures of the surface by construction. Therefore, we need to make sure that the chosen edges correspond to lines of curvature of the surface. One way to guarantee this feature is to locally alter the surface around these edges such that the slope along a given edge is equal, like in the Figure 2.14b. In *Rhinoceros3D* and *Grasshopper*, we can achieve this alteration by modifying the control points of the surface around these edges. Of course, if there is no reason why we would want edges to be align with any edges of the surface, this second step can simply be skipped.

3. *Lines of curvatures*. Now that we have a satisfying target surface, we can compute its lines of curvatures. To do so, we use the library *Tmap* presented in Section 2.3.1. Computing the lines of curvature, we get an intrinsic grid of curves over the surface, as it is shown in Figure 2.14c.
4. *Definition of the 120-3D nodes*. Based on this grid of curves, we then define a mesh. We set what will become our future 120-3D nodes at some of the intersections of the curves of the grid. This selection is done according to triangular pattern, as we can see it in Figure 2.14d.
5. *Definition of the flat nodes*. We also use this triangle pattern to set what will become our future flat nodes by computing the Fermat points of these triangles, as we can see it in Figure 2.14e.
6. *Definition of the edges*. Now that we have our two families of nodes, we can define the edges of the mesh by connecting all the nodes to form a hexagonal mesh, as we show it in Figure 2.14f.

Let us insist on an important aspect: the resulting mesh is not (yet) an Xmesh but a hexagonal mesh. However, this method provides a natural way to generate a hexagonal mesh from a target surface using its lines of curvatures. In Section 1.3, we exhibited a theoretical argument justifying why it makes sense to base the definition of an Xmesh on lines of curvatures. Furthermore, this method has shown good performances. We have applied it, to both IASS Xmesh pavilion and CLC Xmesh pavilion projects, multiple times with various target surfaces, and the following optimization has converged to satisfying Xmesh solutions.

2.3.3 Limits of the initialization

As we explained it previously, there is a theoretical argument to the use of lines of curvature of the target surface to define the mesh. However, this method can easily reach its limits as the surface has to be smooth, as it should not have umbilical points, neither significant changes of curvature - like the one shown in the Figure 2.13.
Furthermore, as the orientation of the mesh is essentially guided by the movement of lines of curvatures, if we want some mesh edges to have specific orientations, we have to shape the surface to have the right lines of curvatures, which is not easy in practice. In the CLC Xmesh pavilion, we put a lot of effort to shape a target surface that would have lines of curvatures with the right orientations around the locations of the columns to ensure their mechanical efficiency.

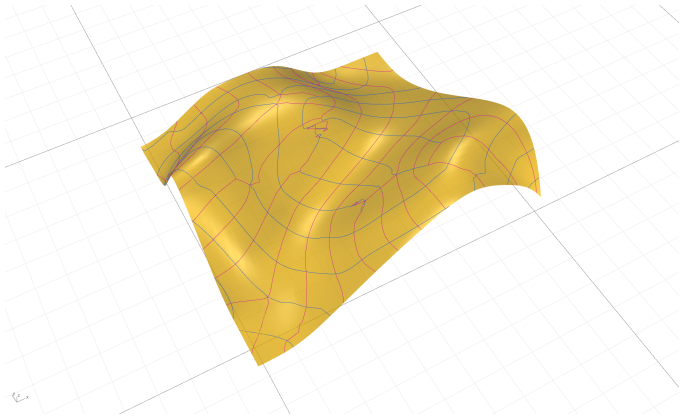
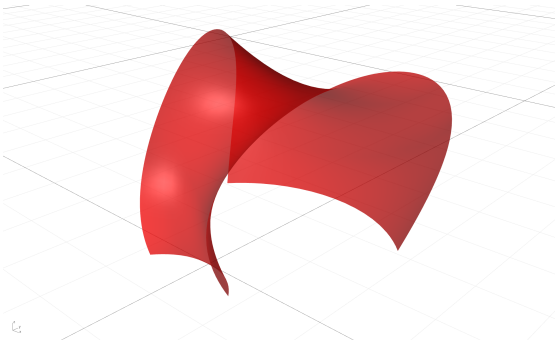
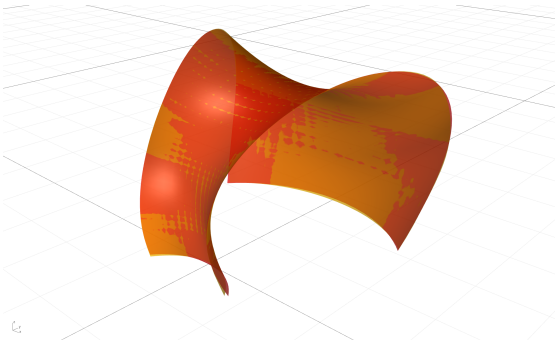


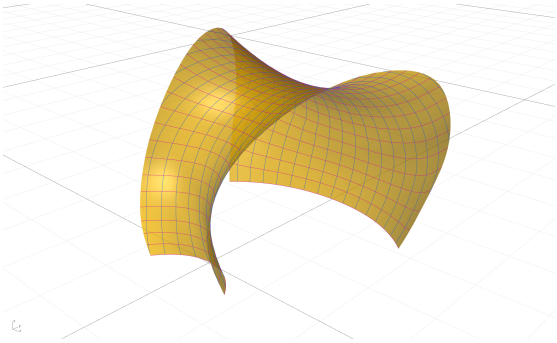
Figure 2.13: An instance of a bad initial surface.



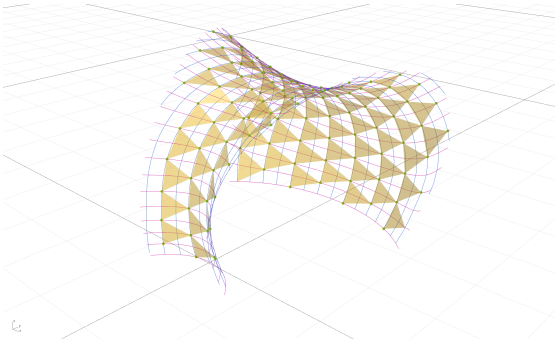
(a) Choose a target surface



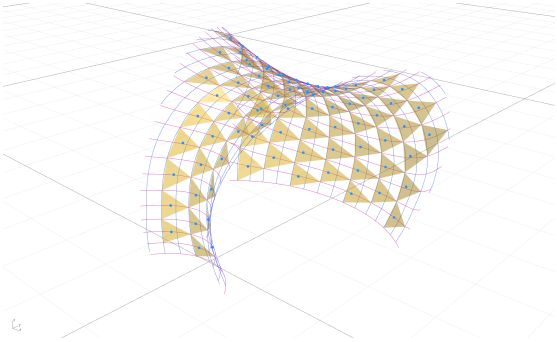
(b) Alter surface around its edges



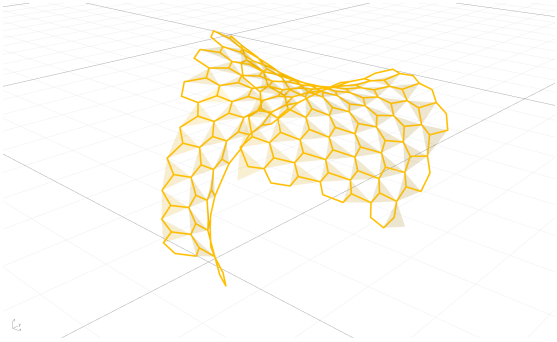
(c) Compute its lines of curvatures



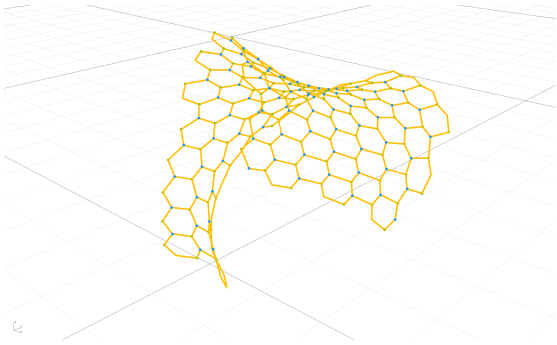
(d) Set the locations of the 120-3D nodes



(e) Set the locations of the flat nodes



(f) Set the edges of the mesh



(g) Hexagonal mesh at the end

Figure 2.14: Steps of the method of *initialization*. Application to the IASS Xmesh pavilion project.

2.4 Results of the optimization of an Xmesh

In the previous sections, we presented three types of custom goals prescribing different geometric conditions - torsion-free node, flat node (coplanarity and orthogonality) and 120-3D node - on nodes of valence 3 or 2 and we then described a method to generate a hexagonal mesh from a target surface. Now that we have an initial hexagonal mesh, we can therefore apply these custom goals to its nodes, assemble them into a *Kangaroo2* optimization problem and solve it. Step by step, the hexagonal mesh is altered to get closer to an Xmesh.

2.4.1 Weights of the goals

Before running the optimization process, we must set weights to the various involved in the process. In the Table 2.1, we give the weights we set in order to obtain the convergence results of the Section 2.4.2 for both pavilions projects.

Goals	Tors.-free	Coplan.	Ortho.		120-3D		Target srf.	Edge len.
Version			Main	Neigh.	Main	Neigh.		
Weights IASS	2.5	-0.3	2.8	-0.5	0.0	-4.0	-3.3	3.0
Weights CLC	1.8	-0.4	2.0	-0.7	-0.5	-2.0	-3.0	3.0

Table 2.1: *Weights* of the goals involved in the optimization process for both pavilions projects. From left to right, torsion-free custom goal (see Section 2.2.2), coplanarity default goal (see Section 2.2.3), orthogonality custom goal (see Section 2.2.3), 120-3D node custom goal (see Section 2.2.4), on-target-surface default goal and range-of-length default goal. Weights are given in powers of 10.

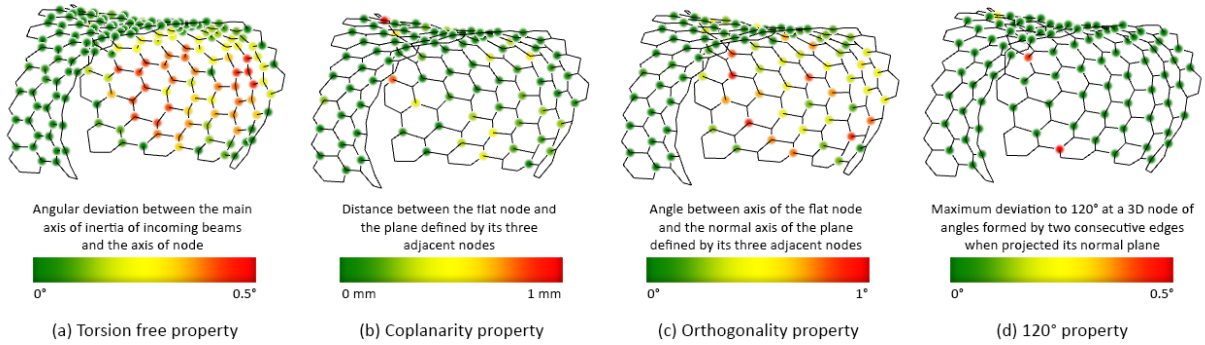
We have come to these values of weights after several runs of the optimization process, searching for the right relative differences to get fruitful results.
On one hand, we can observe that the weights are respectively about the same orders of magnitude between the two pavilion projects. The slight differences between the two projects can be due to differences of mesh geometry, edge length, number of elements or boundary conditions for instance. On the other hand, for each project, there are significant differences between the weights set to different goals. The versions of custom goals altering only the main nodes have larger weights than the ones altering all the neighboring nodes. Intuitively, it is better to alter few particles than many of them in order to fulfill the same geometric constraints. Coplanarity and on-target-surface goals do not need higher weights as they are pretty well accomplished with the initial mesh by construction. Torsion-free and orthogonality (main version) customs goals need relatively high weights as they are far from being achieved initially and all the normal vectors of the mesh nodes must be altered by propagation in order to satisfy these geometric constraints.
These weighting values are not inflexible. Slight changes of these values would likely converge to satisfying results. However, the relative difference order of magnitude seem to be important. For instance, increasing the weight of the on-target-surface default goal by a factor of 10 prevent the optimization process from converging to satisfying results regarding the other criteria.

2.4.2 Tolerances

The optimization problem that we are here processing is non-linear and non-convex. There is no theoretical guarantee that we can get an Xmesh from a hexagonal mesh. However, this optimization does enable us to get an *approximate Xmesh*. By the locution approximate Xmesh, we mean here a hexagonal mesh which geometry satisfies the geometric conditions of an Xmesh within certain small tolerances - small must be quantified relatively to the purpose of the geometry. Actually, the Xmesh, as it is defined theoretically, with geometric conditions that are exactly met, is never reached in practice. Because we assemble goals that are generally competitive, it is possible that at one step of the optimization process, one goal prescribes a particle to move towards a direction while another prescribes the same particle to move towards the opposite direction. Although these competitive scenarii can happen, thanks to the diversity of the goals and the effectiveness of the optimization solver, the overall geometric conditions get progressively satisfied down to certain tolerances that we wish to be small. Hence we only get an approximate Xmesh.
For instance, let us take a goal of coplanarity applied on one main node and three neighbors. As this goal is part of a large optimization problem to transform a hexagonal mesh into an Xmesh, it competes with other goals that also prescribes alterations. Possibly, after tens or thousands of iterations, this local coplanarity goal will be satisfied down to a tolerance of 1mm: the distance between the four nodes and their average plane will be smaller than 1mm.
At the first sight, we could think that it is unsatisfying to have an approximate Xmesh. However, we should not forget that the purpose of this computed geometry is to be applied to structures. Yet, the industrial production of structural elements as well as the construction are riddled with inevitable tolerances. For instance, beams are never perfectly straight and physically connecting two beams together leads to small metric errors. Therefore, by keeping in mind that we aim to build a structure from the computed geometry, we should be satisfied with an approximate Xmesh as long as the tolerances are *acceptable* that is to say within the tolerances of the industrial production or the ones of the construction. The unique reason why we could be satisfied only with getting an Xmesh would be for the purpose (or rather the beauty) of abstract geometry.

We apply this geometric optimization to our two main projects: the IASS Xmesh pavilion and the CLC Xmesh pavilion. In both cases, the mesh is initialized according to the method described in Section 2.3.2 and its corresponding *Grasshopper* code. Then, geometric optimization is applied using *Kangaroo2*. More specifically, goals of torsion-free nodes, flat nodes and 120-3D nodes are applied to the correct groups of nodes and assemble together into a model to give to *Kangaroo2* solver. Additional goals are also added for various purposes. Goals help preserving the proximity of the mesh vertices to the target surface, help smoothing the boundary edges and prevent the mesh to have significant and unpleasant kinks. One goal constraints edges to have a length included into a given length range for fabrication purposes. Other goals deal with boundary conditions like forcing some vertices to remain on the floor for instance. Empirically, the resulting mesh tends to satisfy all the geometrical properties within very low tolerances. These tolerances get up to ten times lower than what they are initially before any optimization.

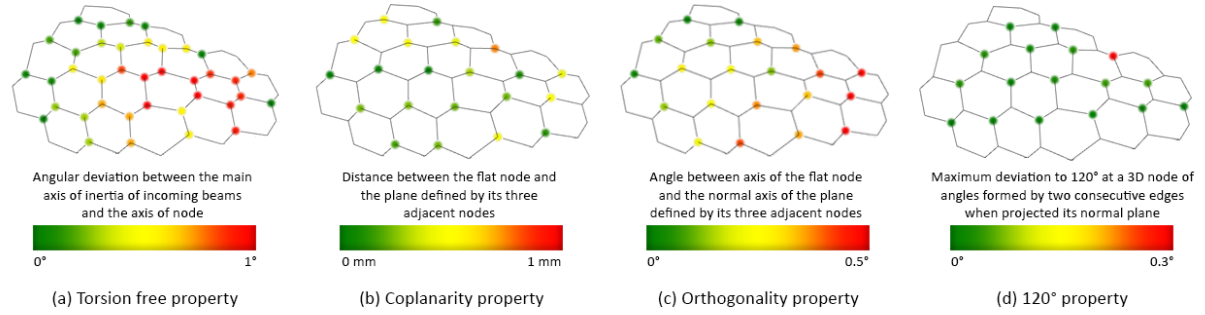
For the *IASS Xmesh pavilion* project, fabrication and construction tolerances are very small. A laser cutting machine is used to make the aluminium structural elements. Such a machine has a tolerance of 0.1mm. However, in the perspective of the construction, we have to introduce small tolerances of 0.4mm. Furthermore, if the 120-3D nodes in aluminium are laser-cut, they are then mechanically stamped which has a larger fabrication tolerance, estimated about 0.5mm / 1-2 degrees. On the whole, the tolerance for fabrication and construction can be considered as about 1mm / 1-2 degrees. As a result, when running the optimization process, we aim to get geometric tolerances smaller or equal to 1mm / 1-2 degrees, which we manage to get as it is shown in Figure 2.15.



(a) Torsion free property (b) Coplanarity property (c) Orthogonality property (d) 120° property

Figure 2.15: *Convergence* of the optimization for the IASS Xmesh pavilion. Criterium (a) corresponds to torsion-free property, criteria (b) and (c) to flat node property and criterium (d) to 120-3D property.

For the *CLC Xmesh pavilion* project, fabrication and construction tolerances are a slightly larger. The accuracy of the fabrication and the alterations of the wooden beams, nodes and panels is about 1 to 5mm. When running the optimization process, we aim to get geometric tolerances smaller or equal to 1mm / 1-2 degrees, which again we manage to get as it is shown in Figure 2.16.



(a) Torsion free property (b) Coplanarity property (c) Orthogonality property (d) 120° property

Figure 2.16: *Convergence* of the optimization for the CLC Xmesh pavilion. Columns are not represented as they were not part of the optimization process.

2.4.3 Limits of the optimization

By applying the optimization to diverse geometries, we have observed that the process would eventually reach some limits.

Firstly, we notice that the process can struggle satisfying all geometric goals within low tolerances, even if we let it run for thousands of iterations.
From these observations, we infer that the curvature of the target surface must not be too important relatively to the length of the edges of the mesh if we want our optimization process to converge to a satisfying result.
We have not got the time to quantify this relative limit of curvature. One way to study this effect could have been the following. We could have prepared a range of single curvature surfaces, like the one shown in Figure 1.6, with various curvature intensity. For each, we could have compute range of set of lines of curvatures with various density. Then, for each surface and each set of lines of curvatures, we could have apply our method of initialization and run the optimization process.

Secondly, sometimes when tolerances are reduced within acceptable values, the process can converge to a mesh that is not satisfying regarding its formal expressiveness or the orientations of its edges. The resulting mesh can present unpleasant kinks, like angular bents, which ruin the smooth feeling of free-form surface. Or, some edges, that would initially have the orientations that we wanted, could unintentionally rotate during the optimization process.
To fight these issues, we can first define additional goals into the optimization process. For instance, we can add a goal that attracts a specific vertex towards a location or an altitude. We can then change by hand the initial geometry hoping that the process will converge to a better mesh. Nevertheless, these additional attentions can quickly be tedious and even fruitless.
However, we actually have some control on the shape of the boundaries of the final mesh after its optimization. Indeed, whenever there are two nodes consecutive nodes, at the boundary of the mesh, that have a valence of 2, we can locally alter these nodes while preserving all the properties of the Xmesh. More precisely, we have two degrees of freedom on which we can play to shape the boundary as we want. This liberty is still very constrained as the geometry surrounding these two nodes is not altered. we have used this limited freedom to reshape some of the boundaries of the CLC Xmesh pavilion such that the columns could have mechanically-efficient orientations.

Chapter 3

Structural analysis

In the Chapter 1, in the Section 1.2, we defined the notion of Xmesh with all its beneficial geometric features. In particular, we explained how its nodes are seperated in two families. They are either flat nodes or 120-3D nodes. In the Chapter 2, we presented a post-rationalization method, based on geometric optimization, enabling us to obtain an Xmesh within acceptable tolerances. We applied this method to our two projects, the IASS Xmesh pavilion and the CLC Xmesh pavilion.
Having Xmesh geometries, we can now put structures on top of them. In this Chapter, we present how we can model the behaviour of the overall Xmesh structures, as well as, more specifically, some details or connections. Firstly, we introduce the tools we use for our various modelings. Then, we focus on the modeling of the flat nodes and the 120-3D nodes. Finally, we describe the global model of the Xmesh structures and analyze their results.
The two last Sections will be applied to our two practical cases, the IASS Xmesh pavilion and the CLC Xmesh pavilion.

3.1 Software framework

Before starting with any structural analysis, we briefly present the software we use in order to build our various models.
In section 2.1.1, we introduced the software *Rhinoceros3D* along with its plug-in *Grasshopper* and we explained how the latter could be extended with various libraries. *Karamba* is one of these *Grasshopper* libraries. It enables users to define a finite element model in order to analyze a structure from based on a *Rhinoceros3D* geometry.

We now describe the various inputs we use to build our models. Interested readers can find more details in [9].

- *Material*. *Karamba* provides a set of default material like steel and wood. Users can also define their own isotropic material like aluminium.
- *Section*. Users can define section of different shape and made with different materials. For instance, we can define a rectangular wooden or aluminium section.
- *Beam*. From a *Line* element, a *Vector3D* and a *Section*, we can define a *Beam* element. The line corresponds to the center-line of the beam. The vector corresponds to the main axis of inertia of the beam's section.
- *Shell*. From a *Mesh* element, we can define a *Shell* element. The surface corresponds to the mid-height of the shell.
- *Connections*. By default, two beams whose center-lines are connected to each other are fully blocked mechanically. However, users can overwrite this default connection to activate some translation or rotation degrees of freedom or to define elastic connections. In the section 3.4, we will see how these connections have a significant impact on the structure behaviour depending on the assumptions we make about their elasticity or their degrees of freedom.
- *Support*. A *Point3D* can be defined as a support. Various support condition can be applied. We mainly make the use of the simple support, that is to say a support blocking all the translation movement but allowing all the rotations.
- *Load*. Point load, line load and volume loads can all be applied over the structure. We essentially used volume loads for modeling gravity and point loads for modeling accidental load.

All these elements can then be assembled into a model using a component called *Assemble*. Users can then analyze the model using first order or second order approach, apply various combinations of loads and check that the structural elements can bear the internal forces.

We use *Karamba* for modeling the overall structure as well as some specific parts of structural elements around the nodes. When modeling shell elements from a mesh within *Grasshopper* and *Karamba*, users shall be careful with the shape of the generated mesh. Some flaws, like folds, can easily appear with the generation of the mesh, especially when the size of the element of the mesh is relatively small. These flaws can then provoke incorrect behaviours.
Other finite element analysis software, like *Abaqus* [10], offer a better control over the mesh generation. finally, we have not had to use such other software as we have managed to get rid of flaws by manually forcing the generated mesh to have particular vertices.

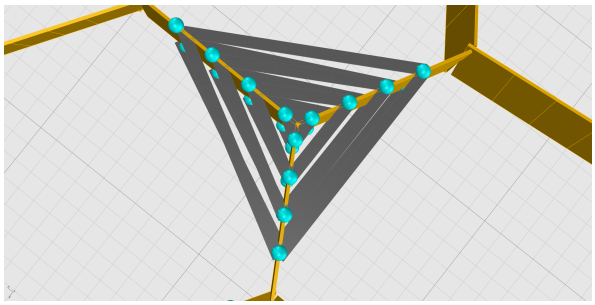
3.2 Modeling flat nodes

In Section 1.2.1 and with Figures 1.8 and 1.9, we described and represented how flat nodes are realized in both project. In both cases, connections at flat nodes are made thanks to pairs of triangular panels sandwiching the beams connected to the nodes. We now explain how we model the behaviour of these nodes and how we check their resistance.

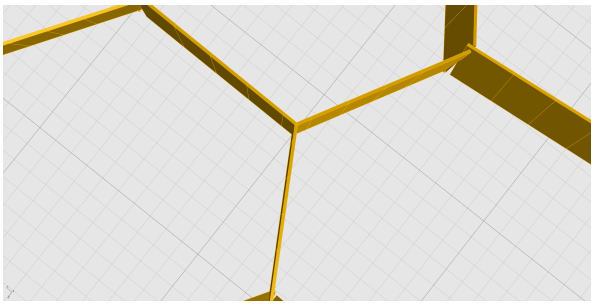
At a flat node connection, the forces and moments that transfer from one beam to another essentially have to go by the panels. Therefore they have to go through the connectors between the beams and the panels. For convenience, we call *beams-panels connectors* these elements. In the case of the IASS Xmesh, these connectors work by contact. As we can see it on Figure 1.8, about four L-shaped stops, on both top and bottom edges of each beam, slide into corresponding rectangular holes in the panels, until they hit one of their edges. Holes are accurately cut such that the stops are also laterally in contact with the panels within very low tolerances - holes are slightly larger than the stops by 0.4mm. In the case of the CLC Xmesh, these connectors are groups of screws. As we can see it on Figure 1.9, two groups of four screws are arranged on both top and bottom sides of each beam.
In both cases, the forces and moments passing through the panels have to go along lines connecting beams-panels connections. Following this intuition, we model panels as sets of *strips*. As shown in Figure 3.1, several strips connect pairs of beams, intersecting at the location of the flat node, on the top and bottom sides of the beams. In the IASS case, we model the panels with groups of 4 aluminium strips, spanning between consecutive beams of the node, as there are about 4 stops along the beams. The strips are 1.5mm high, like the real aluminium panels, and are 20mm wide. In the CLC case, we model the panels with pairs of wooden strips, as there are 2 groups of screws. The strips are 12mm high, like the wooden panels, and are 100mm wide.
Finally, to connect the strips with the intersecting beams, rigid links span between the ends of the strips and the beams' center-lines. Their directions are parallel to the main axis of the beams

In both cases, the width of the strips comes from an arbitrary choice. Indeed, on one hand, there is a significant gap in the results between having sets of one strip and sets of two or more strips, as shown on Figure 3.2a. Therefore, we have to consider set of at least two strips, so it makes sense to take as many strips as they are beams-panels connectors. On the other hand, we observe that the mechanical results of the model are not very sensitive to the width of the strips, as shown on Figure 3.2b. Thus, we are relatively free to choose the width of the strips, so we have chosen to set it such that strips do not overlap each other.

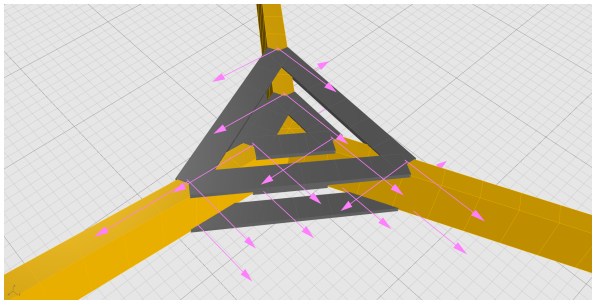
By modeling panels with multiple strips, we change relatively-wide elements for multiple narrower elements. As a consequence, these strips elements have a smaller inertia than the panels and are more likely to *buckle*, which is not representative of the panels' behaviours. Therefore, we deactivate the buckling ability of the strips. Furthermore, in the IASS case, as the panels are particularly thin (1.5mm) and the contacts with the L-shaped stops are relatively isolated, we prescribe the strips to work only in *tension* and prevent them to develop any *bending moment*. In the CLC case, as the panels are thicker (12mm) we expect them to bear some *compression* and *bending moment* so we do not apply these same restrictions.



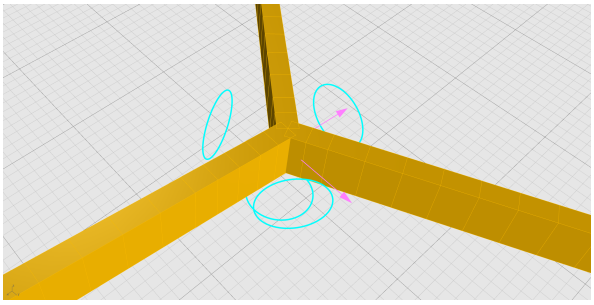
(a) [IASS] Panels modeled as 4 strips. Rotations are released at the end of the strips.



(b) [IASS] No degrees of freedom released at the intersection of beams.

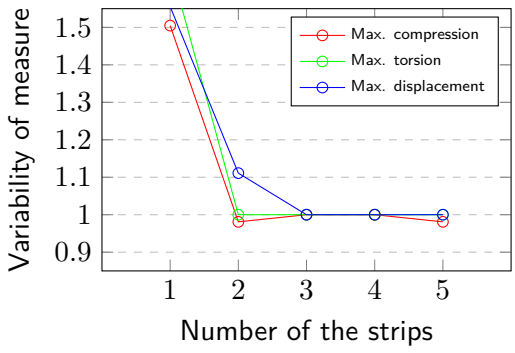


(c) [CLC] Panels modeled as 2 strips. Lateral elastic springs set at the end of the strips.

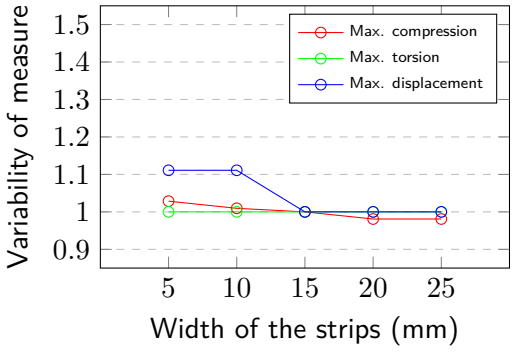


(d) [CLC] Lateral translations as well as torsional and in-plane rotations released at the intersection.

Figure 3.1: Modeling flat nodes in the *finite elements* structural models. Top row deals with the IASS Xmesh, bottom row with the CLC Xmesh. Left column shows how panels are modeled with sets of *strips* connected to the beams' center-lines thanks to rigid links. Right column shows the connectivity of the beams. Direction of pink arrows represent translations that are either released or assigned with an elastic stiffness. Normal vectors to the blue circles correspond axis of rotations that are either released or assigned with an elastic stiffness. Blue sphere signal that all moments of the structural elements are killed and all rotations at the ends are released.



(a) Sensitivity to the number of the strips



(b) Sensitivity to the width of the strips

Figure 3.2: *Sensitivity* to the parameters of the *strips*. Maximal beams forces/moments and displacements/rotations are obtained from the finite elements model of the IASS Xmesh pavilion. The configuration of reference from which the variability of the measures is computed is 4 strips of 15mm.

Based on the kinematic considerations developed in the Section 1.2.1 and this way of representing the panels with strips, we can now model the *connectivity* between the strips and the rigid links.
In the IASS Xmesh case, we block all the translations and release all the rotations at the end of the strips, as we can see it on the Figure 3.1a. In Section 1.2.1, we saw that translations according to the beams' Y and Z axis are blocked thanks the contacts between the panels and the beams at the locations of the L-shaped stops. We also saw that extension of the beams is blocked at the locations of the L-shaped stops but contraction is blocked only at the ends of the beams. However, as the strips are modeled not to work in compression, if the translation according to the beams' Z axis is blocked and if the beams get contracted by compression, strips will get compressed and will be removed from the model such that compression will be pushed back to the ends of the beams as wanted. As a consequence, we can block all translations. About the rotations, as explained above, the strips are modeled not to bear any bending moments, we therefore release all rotations. Let us point out that, even if the rotations are released at the ends of the strips, since the translations are there blocked, the sets of strips are together able to block all the rotations of the beams, as wanted.
In the CLC Xmesh case, we block vertical translation but we assign an elastic stiffness (13000kN·m^{-1}) to the lateral translations, as we can see it on the Figure 3.1c. The deformation of the screws according to their axis is significantly higher than their lateral deformations. As we can quantify the elastic stiffness related to these lateral deformations, we choose to input them into our structural model. Regarding the rotations, as explained above, we expect this time that the wooden panels will bear some bending moments. Furthermore, as shown in the Figure 1.9, the connections between the panels and the beam are realized thanks to groups of four screws. These considerations lead us to block all rotations at the end of the strips, as shown on the Figure 3.1c.

We can also the *connectivity* of the beams intersecting at the location of the flat node.
In the IASS case, as explained in the Section 1.2.1, a washer, attached to a screw, blocks the beams to slide back toward the middle of the panel by hitting their first L-shaped stop. This washer is relatively stiff compared to the beams and the panels and it is located particularly close to the axis of intersection of the beams. These observations drive us not to release any degrees of freedom at the intersection of the beams, as shown on the Figure 3.1b, especially as most of the beams' forces and moments are actually transfer through the sets of strips.
In the CLC case, we saw in the Section 1.2.1 that all in-plane movements are blocked thanks to the groups of screws and that we could not rely on the triangle-shaped ends of the beams to block these movements. For these reasons, we choose to release translations corresponding to the beams' X and Y axis and to release the rotations according to their X and Z axis, as we can see it on the Figure 3.1d. We also saw that all out-of-plane movements of the beam are blocked thanks to the contact with the panels. Therefore, we block the translation according to the beams' Z axis and the rotation according to their Y axis.

Finally, we check the resistance of the connectors.
In the IASS case, we check that the L-shaped stops could bear the shear stress caused by the normal force and the shear force developed into the aluminium strips. With conservative assumptions about the way forces in the beams transfers to the L-shape stops, we can compute the shear force apply to the foot and the head of the stops. Then, within the framework of the design of metal beam, we can compute the shear resistance of these parts. The design of the L-shaped stop is valid if the following criterion, applied to its foot and its head, is satisfied,

$$V_{Ed} \leqslant V_{Rd} = A_V \frac{f_0}{\sqrt{3}\gamma_{M_2}} \tag{3.1}$$

where V_{Ed} is the shear load applied to the metal part, V_{Rd} its shear resistance, A_V its shear area, f_0 its 0,2% proof strength and γ_{M_2} its partial factor. We give in 3.1 the results of this resistance check.

Variables		Shear area	Proof strength	Partial factor	Shear resistance	Shear load
Symbol Unit		A_V mm^2	f_0 N·mm^2	γ_{M_2}	V_{Rd} kN	V_{Ed} kN
Value	Foot	32	80	1.25	1.18	1.02
	Head	16	80	1.25	0.59	0.07

Table 3.1: Check of the *resistance* of the L-shaped stops for the IASS Xmesh pavilion.

In the CLC case, colleagues check that the screws could resist to the shear forces caused by the forces and moments transferred from the beams to the panels (*i.e.* the strips).

We could have probably modeled the panels with *shells* elements instead of strips. However, the model would have been more complex to build, to run and to analyze. Modeling panels with sets of strips is very convenient to observe and understand how forces and moments are transferred from one beam to another through the panels.

3.3 Modeling 120-3D nodes

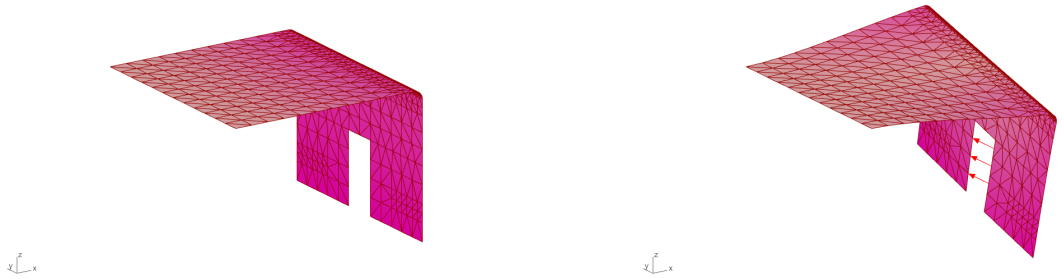
In Section 1.2.2 and with Figures 1.11 and 1.12, we described and represented how 120-3D nodes are realized in both project. The technological solutions used for the two cases are quite different. For the IASS case, the 120-3D nodes are made with pairs of two star-shaped pieces of stamped aluminium connected thanks to single screws. For the CLC case, the nodes are made with hexagonal wooden sections and multiple pairs of screws. We now explain how we model these nodes for each case.

3.3.1 Modeling 120-3D nodes for the IASS Xmesh pavilion

For the 120-3D nodes of the IASS Xmesh pavilion, we develop a *finite elements* model that isolates one of the three teeth of the stamped aluminium part.

We develop a parametric definition of the geometry of the node within *Grasshopper*. We then build a finite elements model on top of this geometry thanks to *Karamba* library. We define a *shell* element on which we apply various load cases and analyze the behaviour and the resistance of the node. As the 120-3D node connection has a central symmetry and assuming that the three beams will transfer forces and moments that are relatively close, we can apply supports blocking all translations and rotations along the edge that is opposed to the slit. The structural model of the pavilion, that will be described in Section 3.4, validates this assumption *a posteriori*. The Figure 3.3a shows a 3D render of this finite elements model.
The benefit of working within *Grasshopper* and *Karamba* framework is that we can easily change the geometry of the aluminium part and observe how its behaviour change. However, this parametric approach has its limit because it can be necessary to have manual control over the mesh on which is based the finite elements computation.

Thanks to this model, we can then quantify the *elastic behaviour* of the node. This essential information will then be used in the global model of the structure of the IASS Xmesh pavilion.
Let us consider a beam connected to a 120-3D node. Each type of force/moment transferred by the beam to the node applies a type of load on it and causes a type of displacement/rotation. By computing the ratio of the load applied over the resulting displacement/rotation, we can get an elastic stiffness of the node in conjunction with each type of forces and moments.
Transfer of normal or shear forces of the beam to the node is modeled by applying an uniform load on the mesh where the metal piece gets in contact with the beam; the corresponding displacement is computed as the average displacement of this same area of contact. For the transfer of a bending moment, we first convert it into an equivalent force couple by dividing its value by the right lever-arm. Again we apply an uniform load on the mesh where the metal piece gets in contact with the beam and we compute the average displacement of this area. Finally, we compute a rotation angle by dividing the average displacement by the right lever-arm.
Each time, we make sure that the computed value is stable when reducing the size of the mesh. For the comp The Figure 3.3b shows a 3D render of the piece under the load of a lateral force. We sum up the values of stiffness we get with this approach in the Table 3.2.



(a) One tooth of a 120-3D node.

(b) One tooth deformed under shear Y.

Figure 3.3: *Finite element models* of a *tooth-like part* of a 120-3D node in the IASS Xmesh project.

Elastic stiffness	Normal X	Shear Y	Shear Z	Torsion X	Moment Y	Moment Z
Symbol	k_{N_x}	k_{V_y}	k_{V_z}	k_{T_x}	k_{M_y}	k_{M_x}
Unit	$kN{\cdot}m^{-1}$	$kN{\cdot}m^{-1}$	$kN{\cdot}m^{-1}$	$kN{\cdot}m{\cdot}rad^{-1}$	$kN{\cdot}m{\cdot}rad^{-1}$	$kN{\cdot}m{\cdot}rad^{-1}$
Value	$21{\cdot}10^3$	$10{\cdot}10^3$	$21{\cdot}10^3$	25	25	∞

Table 3.2: *Elastic stiffness* of the 120-3D nodes in the IASS Xmesh pavilion. The directions X, Y and Z are respectively along the center-line of the beam, its weak axis and its strong axis. We assume that the rotation around the Z axis is infinitely stiff as Z moment uses the strongest inertia of the folded aluminium piece which is much larger than any other moment inertia.

Still with the same finite elements model, we can check that the nodes are able to resist to the loads of the structure. From the finite elements model of the global structure, that we will describe in Section 3.4, we can get the critical forces and moments that the nodes, and more specifically, the tooth-like pieces of the nodes have to bear. We can then apply combinations of the corresponding loads onto instances of these pieces, thanks to our finite elements model, in order to compute their stresses. We consider that the overall resistance of the 120-3D nodes is checked if the following criterion is satisfied,

$$\sigma_{VM} \leqslant f_y \quad \text{holds for more than 90\% of the nodes,} \qquad \sigma_{VM} \leqslant f_u \quad \text{otherwise} \tag{3.2}$$

where σ_{VM} is the maximal Von-Mises stress developed in the aluminium pieces, f_y is the elastic limit of the aluminium and f_u is the ultimate tensile stress of the aluminium. Ideally, we would have rather wanted all the nodes to check the first inequality of 3.2. This would have meant that the aluminium would always remain elastic while, with the current check 3.2, we allow the aluminium to become plastic; and we would rather preserve the elasticity of the nodes as it would mean that the nodes would naturally get back to their initial shape after dismantling the structure. However, as the connections between elements of the structure work essentially by contact on small surfaces and as these contacts are often located near variations of nodes' shapes, relatively-high stresses are inevitable. For these reasons, we decide to accept to overcome the value f_y as long as the proportion of nodes concerned is reasonable.

Load combination	G		G+A		G-A	
Inequality	$\sigma_{VM} \leqslant f_y$	$\sigma_{VM} \leqslant f_u$	$\sigma_{VM} \leqslant f_y$	$\sigma_{VM} \leqslant f_u$	$\sigma_{VM} \leqslant f_y$	$\sigma_{VM} \leqslant f_u$
Proportion	100%	100%	95%	100%	97%	100%

Table 3.3: *Resistance* check of the 120-3D nodes for the IASS Xmesh pavilion. The different load combinations will be described in the Section 3.4.

Finally, we check the resistance of the beams' grip at their 120-3D-node. We apply the same criterion 3.1 as for L-shaped stops. Results are given in the Table 3.4.

Variables	Shear area	Proof strength	Partial factor	Shear resistance	Shear load
Symbol	A_V	f_0	γ_{M_2}	V_{Rd}	V_{Ed}
Unit	mm^2	$N{\cdot}mm^2$		kN	kN
Value	40	80	1.25	1.48	1.02

Table 3.4: *Resistance* check of the beams' grip at their 120-3D-node end for the IASS Xmesh pavilion.

During the conception of the IASS Xmesh pavilion, we also developed a finite element model of the beam's grip, at a time where we wanted the beams to be in acrylic glass (PMMA). Unlike aluminium that is an elastic plastic material, PMMA is an elastic fragile material. In other words, when the limit of elasticity is reached, PMMA breaks. As a consequence, when designing details in PMMA, the safety factor between the load applied and the elastic limit of PMMA shall be about 10.
Hand-calculated checks similar to 3.1 showed that the safety factor was far from being satisfied. We then developed a finite element model on *Abaqus* to get more accurate results as shown on Figure 3.4. We paid attention to the shape of grip to reduce the constraints in the acrylic piece. However, despite our best efforts, we could not manage to get such a safety factor. We then turned to an aluminium solution.

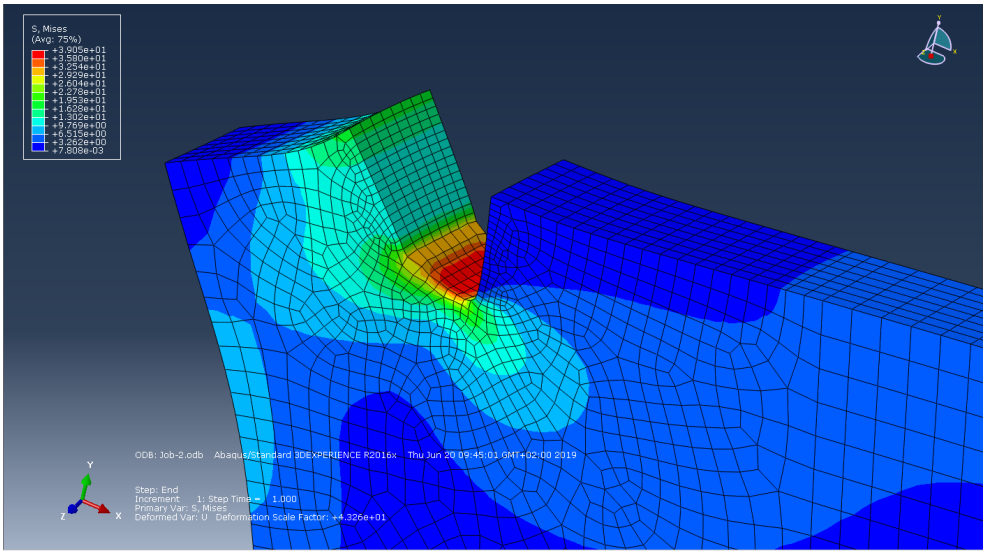


Figure 3.4: *Finite elements model* of the beams' grip for the IASS Xmesh pavilion. Beam material here is PMMA. A lateral load of 1kN is applied. Von Mises stresses are represented.

3.3.2 Modeling 120-3D nodes for the CLC Xmesh pavilion

In the CLC Xmesh pavilion, the 120-3D nodes are realized with hexagonal wooden sections and several pairs of screws.

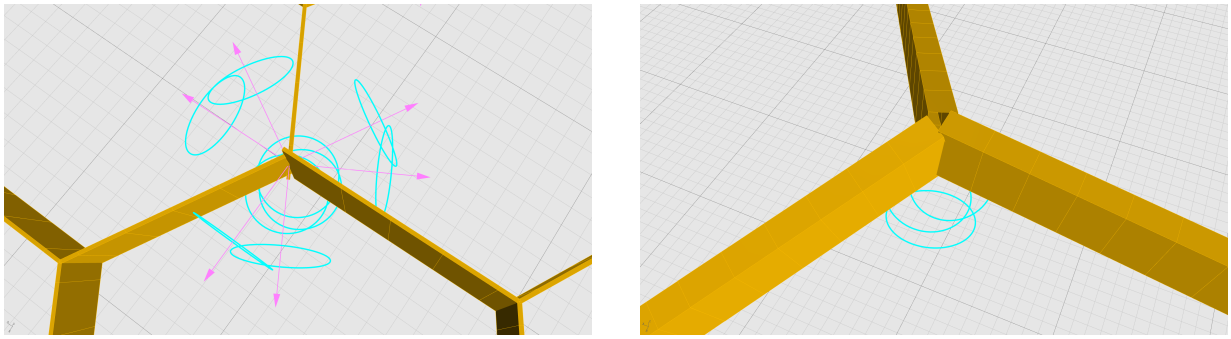
The forces and moments transfer from the beams to the nodes via the screws and possibly by contact between the wooden parts. More precisely, for each beam, tension, shear forces, bending moment of strong axis (according to the beam Y axis) or torsion (according to the beam X axis) transfer via the screws. Compression transfers by contact between the wooden parts. Bending moment about the beam's Z axis cannot be transferred as the pair of screws are aligned with the Z axis.
In other words, the resistance capacity of the nodes essentially depends on the spatial distribution of the screws inside the beams and the hexagonal nodes. In Section 1.2.2, we explained how this spatial distribution is constrained by four conditions: the pairs of screws shall not collide with each other, minimal distances between the screws and the edges of the wooden pieces shall be respected, the lever-arms between pairs of screws shall be the largest possible and the inclinations of the screws relatively to the direction of the beams' center-lines shall be the largest. Once we apply these four conditions to the geometry of the pavilion, we get the spatial distribution of the screws and we are able to quantify the resistance of the 120-3D nodes.
More precisely, the criteria, that we use to check the resistance of the 120-3D nodes, applies to their screws. As we will see it in Section 3.4.3, all types of forces and moments are present in the beams. All of them can cause normal stress and shear stress in the screws. These stresses are then combined in a Von Mises fashion. The criteria of resistance impose that the stress value obtained by this combination must be smaller than a limit of resistance. We do not go into more details about this criteria but it is important to understand that all forces and moments developed in beams indirectly take part in this criteria. It is therefore not easy to give limit values of forces and moments. That being said, we observe that critical forces and moments are generally the bending moment about the beams' Y axis (combined with compression) and the torsion.
The final geometry of the pavilion results from several iterations including some that had been conducted in order to reduce these forces and moments as they would overcome the resistance check of the 120-3D nodes.

3.3.3 Modeling 120-3D nodes in the structural model

Now that 120-3D nodes are characterized and considering the kinematic considerations developed in the Section 1.2.2, we can model them into the finite elements structural models of both pavilions.

In the IASS case, we assign elastic stiffness to rotations and translations according to the Table 3.2. Only the bending moment about the beams' Z axis is blocked as explained previously. The Figure 3.5a represents these mechanical considerations.

In the CLC case, as we mentioned it in the previous section, the bending moment about the beams' Z axis cannot be transferred at the 120-3D nodes due to the vertical alignment of the pair of screws. Therefore, we release the rotation about the beams' Z axis. As we cannot easily model the 120-3D nodes to get the elastic stiffness corresponding to the various translations and rotations, we choose to blocked all these degrees of freedom. The Figure 3.5b represents these mechanical considerations. These assumptions are reasonable as screws are arranged to be able to have large resistances against the various forces and moments (apart from the bending moment about the Z axis). On the whole, they are favorable for the deformation of structure as it is stiffer. But they are detrimental for the resistance as forces and moments are higher than what they would be if the 120-3D nodes would be modeled with some elasticity.



(a) A 120-3D in the IASS model

(b) A 120-3D node in the CLC model

Figure 3.5: Modeling 120-3D nodes in the *finite elements* structural models. *Degrees of freedom*, at the intersection of the beams, are either blocked, released or assigned with an elastic stiffness depending on the technology of the node. Pink arrows represent translations that are either released or assigned with an elastic stiffness. Blue circles represent rotations that are either released or assigned with an elastic stiffness.

During the *Construire Le Courbe* introduction week, one workshop will aim at quantifying the mechanical features of these wooden instances of 120-3D nodes. More specifically, students will measure their elastic stiffness and their resistance against various forces and moments.
Therefore, it will then be possible to input these stiffness values into the model to better represent the behaviour of the 120-3D nodes.

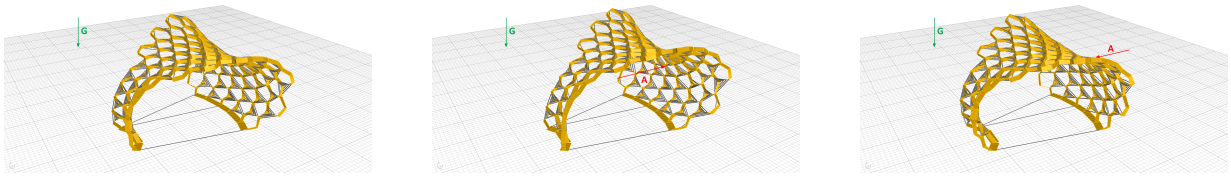
3.4 Modeling the structural behaviour of Xmesh pavilions

3.4.1 Load combinations

Both IASS Xmesh and CLC Xmesh pavilions are built in order to display the potential of Xmesh geometries in architecture, engineering and construction. They are not supposed to stand for years under particular environmental conditions. Therefore, the load cases that we must consider for the design of these pavilions are limited. We consider three combinations of loads.

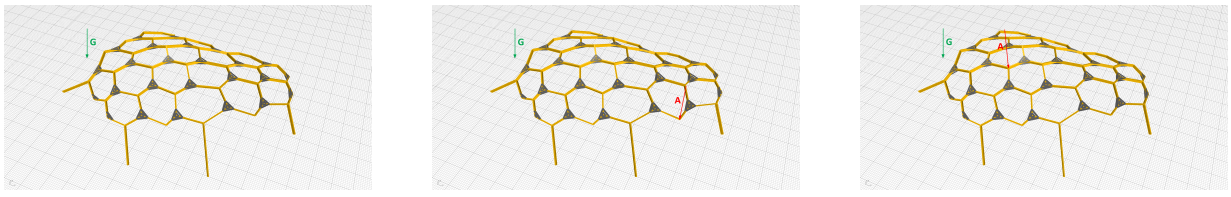
- *Combination G.* The first load combination is gravity. For the resistance of the structural elements, we must consider the ultimate load combination $1.35 \times$G. For the deformation of the structure, we must consider the service load combination $1 \times$G.
- *Combination G+A/G+A*$_1$. The second load combination is an accidental one. We consider a single force applied on a location the structure to model the load that a person could apply on the structure by mistake or to observe its behaviour. In the IASS Xmesh project, this force is a horizontal force of 20kg applied right in the middle of larger part of the structure. In the CLC Xmesh project, this force is a vertical downward force of 80kg applied on the edge of the structure. It is an ultimate load combination obtained by $1 \times$G$+1 \times$A.
- *Combination G-A/G+A*$_2$. The third load combination is also an accidental one with the same ultimate load combination. In the IASS Xmesh project, the accidental force is the opposite of the previous once. In the CLC Xmesh project, the accidental force is a vertical downward force of 80kg applied in the middle of the structure.

These load combinations are represented in the Figures 3.6 and 3.7 for both pavilions. Deformation caused by each load combination is scaled by a factor of about 10 to 20 to better observe the effects of the load on the structure.



(a) Combination G (b) Combination G+A (c) Combination G-A

Figure 3.6: *Load combinations* for the IASS pavilion. Red arrow corresponds to accidental force.



(a) Combination G (b) Combination G+A$_1$ (c) Combination G+A$_2$

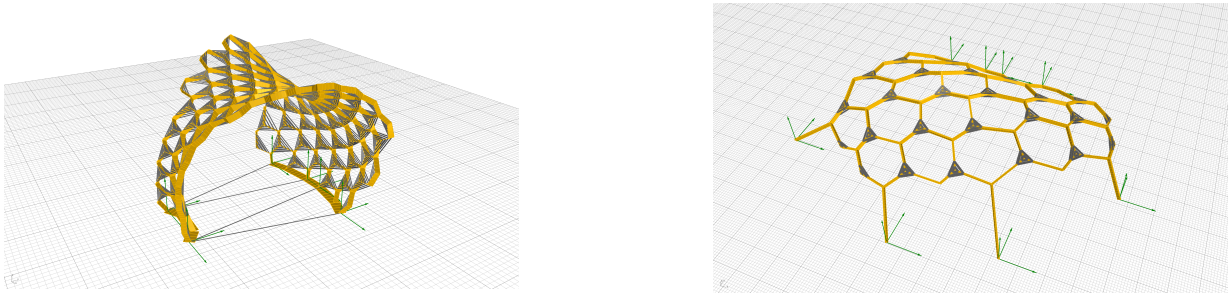
Figure 3.7: *Load combinations* for the CLC pavilion. Red arrow corresponds to accidental force.

3.4.2 Support conditions

In this section, we describe how we model the support conditions of both projects.

Using our finite elements structural model of the IASS Xmesh pavilion, we observe that it is crucial to block the translations of the structural elements at the basis of the structure - rotations can remain free. In other words, putting the pavilion down on the floor without any particular systems would not be enough to ensure the structural integrity of the pavilion, especially when considering the accidental load combinations (see Section 3.4.1). Thus we must *anchor the pavilion on the floor regarding translations*. As the exhibition is taking place in the hall of an hotel, there is nothing on the floor to hook or fix any supports and it is not allowed to drill anything in the floor. We conceive several technical solutions for anchoring the basis on the pavilion.
Firstly, we conceive two *arched floor beams* that are main supports on which the pavilion stands. They are larger and thicker than the usual beams and fit the two curved edges of the geometry of the pavilion, located on the floor. They are designed to be very stiff relatively to the usual beams of the project as well as relatively to the forces developed into them. Such beams allow to have a solid connection between all the beams which end on the floor.
Secondly, we connect these arched beams with four aluminium flat *stringers* spanning between them. Such elements prevent the two arched floor beams to move away in opposite directions. They also help to set the geometry of the pavilion by triangulating the elements on the floor.
Thirdly, we fix highly adhesive pads underneath the arched beams. These pads prevent the beams to slide horizontally as long as horizontal forces do not get too high - about 50kg laterally - which is satisfied according to our model (see Table 3.5).
Thanks to these three physical solutions, we can guarantee that the supports on the floor behave as anchors. All translations are blocked except the upward vertical one. However, according to our model, the structure does not lift under any of the load combinations.

In the case of the CLC Xmesh pavilion, supports are all simple supports that is to say supports blocking all translations but releasing all rotations. These supports are located at four locations against the wall and four locations on the floor.
On the floor, concrete footings topped with steel-post footings are arranged where the four columns reach the floor. On the wall, three wooden platforms are fixed and a continuous girder spans over these platforms establishing a simple support condition on which the structure can lean on.



(a) Support conditions in the IASS model

(b) Support conditions in the CLC model

Figure 3.8: *Support conditions*. Greens arrows represent translations that are blocked.

3.4.3 Overall analysis

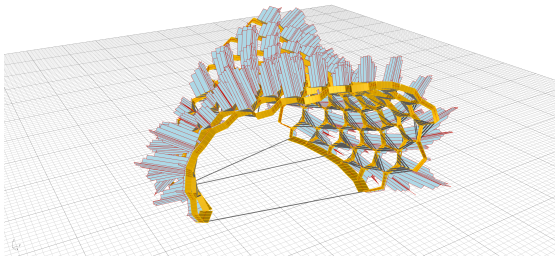
Now that we have described all the constitutive parts of the structural models of both pavilions projects, we can finally give general results about the resistance and the behaviour of these two structures.
The Figures 3.9 and 3.10 show the forces and moments diagrams, of beams and strips, for the two structures, when gravity load (load combination G) is applied to them. Along with these Figures, the Table 3.5 sums up key-values (mass, maximal beams' forces, maximal beams' moments, maximal nodes' displacements) a obtained from our analysis.

From the Figures 3.9, 3.10 and 3.11, we observe that the beams of both structures face all types of forces and moments. Firstly, the coexistence of all types of forces and moments is related to the 3D-hexagonal shape of the mesh faces; equivalently, it is related to the 3-valence of the nodes and the various inclinations of the beams' center-lines at 120-3D nodes. As a result, a normal force coming from a beam to a 120-3D node for instance is converted into normal forces as well as shear Y and Z forces in the next two beams. Similarly, bending moment Y is converted into bending moment about Y and Z as well as torsion in the next two beams. Secondly, this coexistence is due to the connectivity we have set between pairs of beams, strips and rigid links.
Usually, beams are not efficient to transfer lateral forces, bending moments about weak axis and torsion, and having such a complete distribution of all forces and moments is not ideal. Then, a solution to push away these forces and moments from the beams could be by designing a continuous structural envelope like what has been done for the pavilion presented in [7]. For the IASS Xmesh pavilion, we could have got closer to this result by improving the connections between the panels and the beams. For the CLC Xmesh pavilion, we could have so by having larger panels. However, for various reasons (like architectural intentions, contest rules or manufacturing constraints), we could not design such a continuous envelope, nor realize these improvements in both projects.

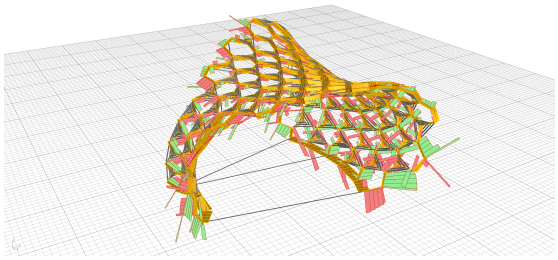
Thanks to the Table 3.5, we can easily see the differences between the two structures.
We can observe that the two structures share about the same orders of magnitude for their maximal compression and shear Z forces while their respective maximal tension, shear Y, torsion and moments Y and Z are significantly different. Then, for the IASS Xmesh itself, compression is about ten times larger than the tension as well as shear forces Y and Z; all the moments are about the same order of the magnitude. For the CLC Xmesh itself, compression, tension and shear Y are about ten times the shear Z; all the moments are about the same order of the magnitude. These differences are essentially due to four elements differently treated in the two structures:

1. The materials are different. In particular, their stiffness are very different.
2. The assumptions about the strips' behaviour and their connectivity to the rigid links are different.
3. The relative stiffness of the strips compared to the one the beams as well as the extension of the strips along the beams' top and bottom edges are significantly different.
4. The length of the elements are very different.

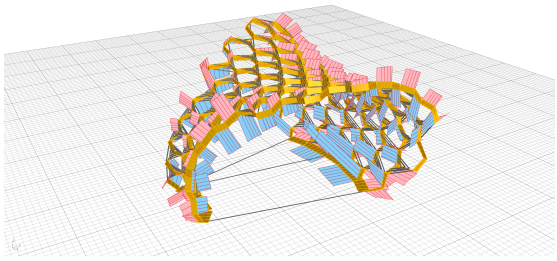
In both projects, beams and connections resistances have been checked.
For the CLC Xmesh pavilion, columns inclinations have been optimized to reduce forces and moments inside structural elements, while satisfying the constraint of maximal beam length.



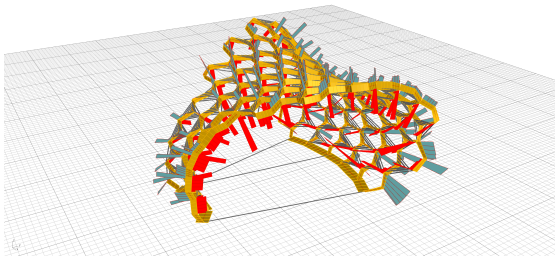
(a) [Beams] Normal force (N_x) diagram



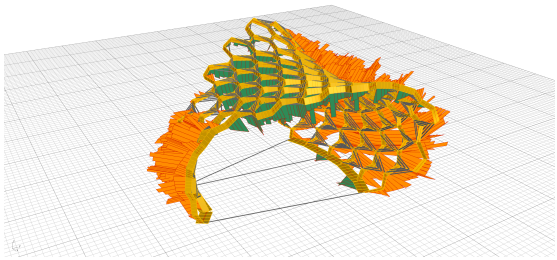
(b) [Beams] Shear force Y (V_y) diagram



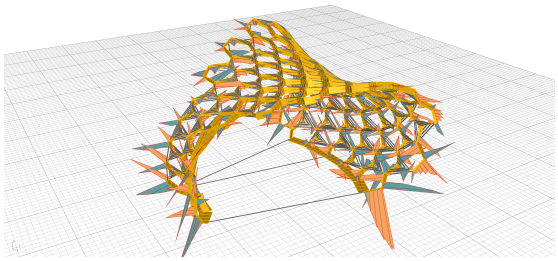
(c) [Beams] Shear force Z (V_z) diagram



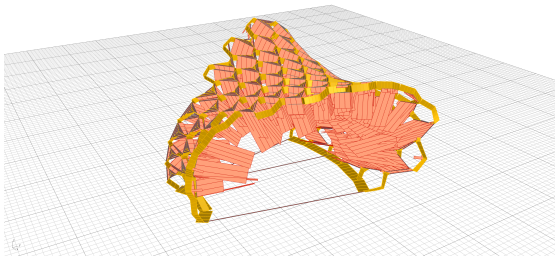
(d) [Beams] Torsion (T_x) diagram



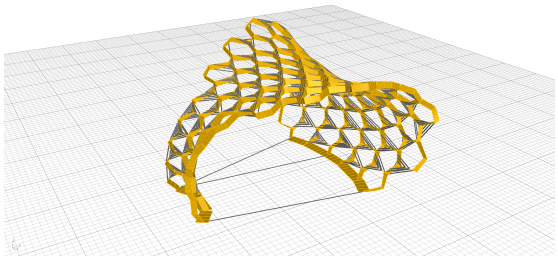
(e) [Beams] Bending moment Y (M_y) diagram



(f) [Beams] Bending moment Z (M_z) diagram

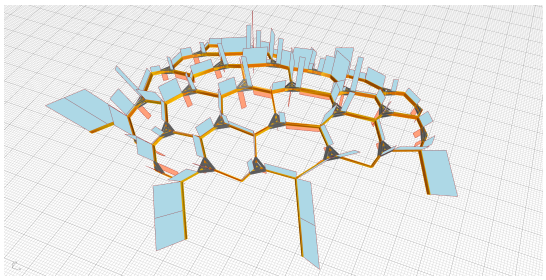


(g) [Strips] Normal force (N_x) diagram

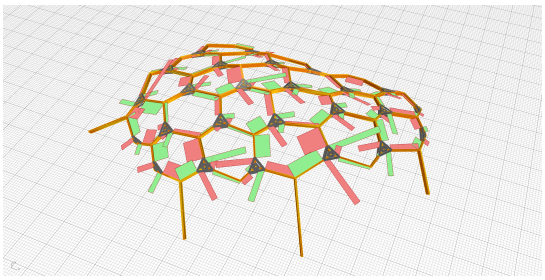


(h) [Strips] All other diagrams

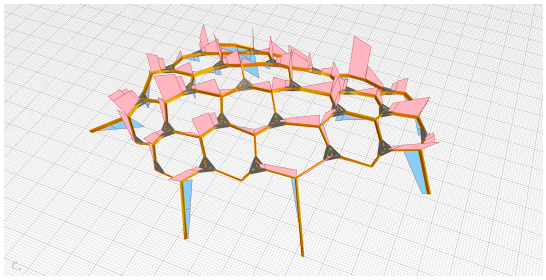
Figure 3.9: *Forces and moments diagrams*, in beams and strips, of the IASS structural model.



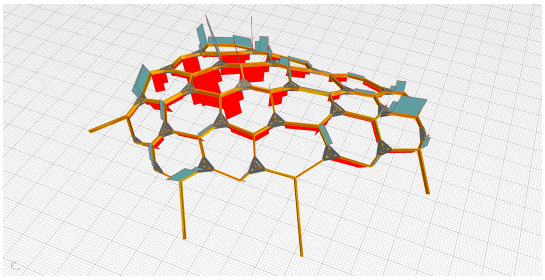
(a) [Beams] Normal force (N_x) diagram



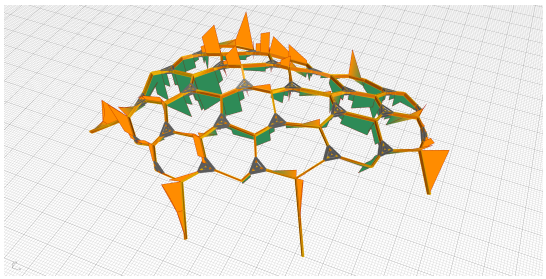
(b) [Beams] Shear force Y (V_y) diagram



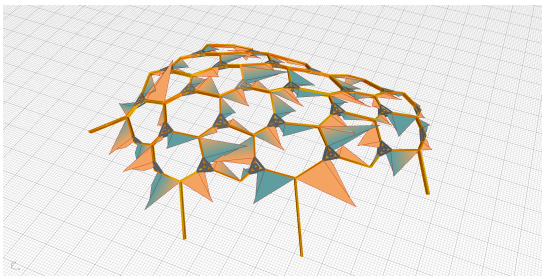
(c) [Beams] Shear force Z (V_z) diagram



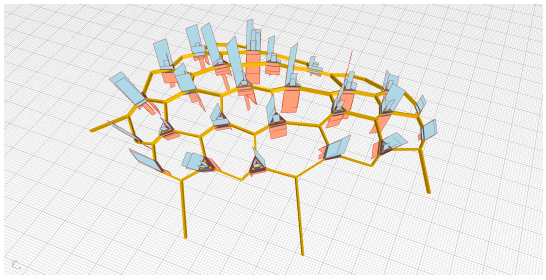
(d) [Beams] Torsion (T_x) diagram



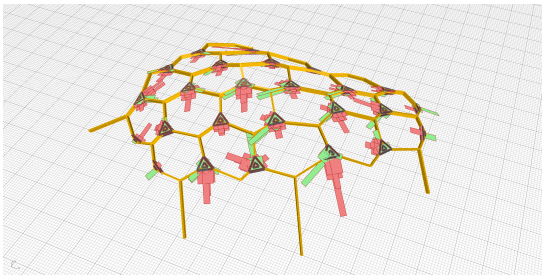
(e) [Beams] Bending moment Y (M_y) diagram



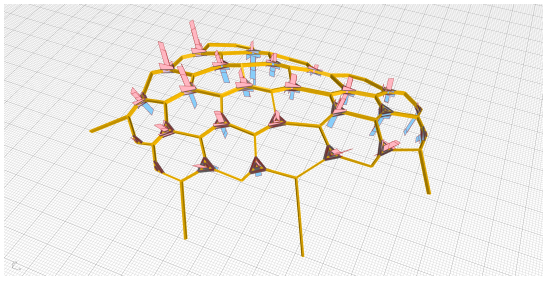
(f) [Beams] Bending moment Z (M_z) diagram



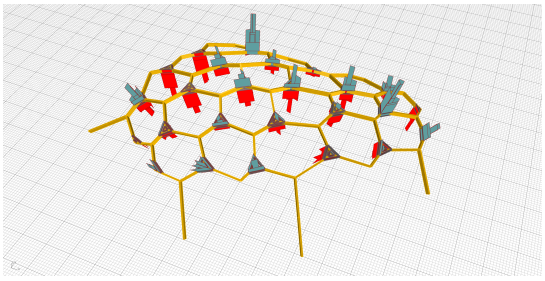
(g) [Beams] Normal force (N_x) diagram



(h) [Strips] Shear force Y (V_y) diagram

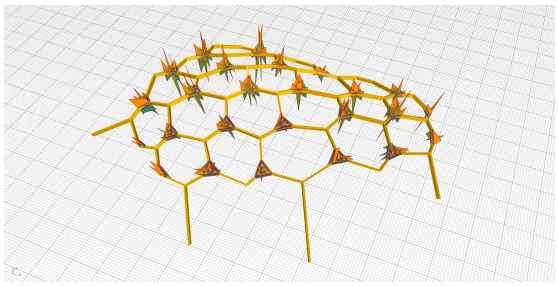


(i) [Strips] Shear force Z (V_z) diagram

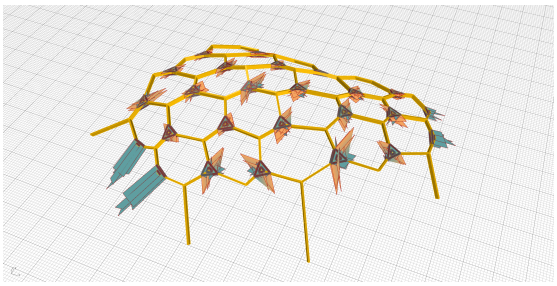


(j) [Strips] Torsion (T_x) diagram

Figure 3.10: *Forces and moments diagrams*, in beams and strips, of the CLC structural model (#1).



(a) [Strips] Bending moment Y (M_y) diagram



(b) [Strips] Bending moment Z (M_z) diagram

Figure 3.11: *Forces and moments diagrams*, in beams and strips, of the CLC structural model (#2).

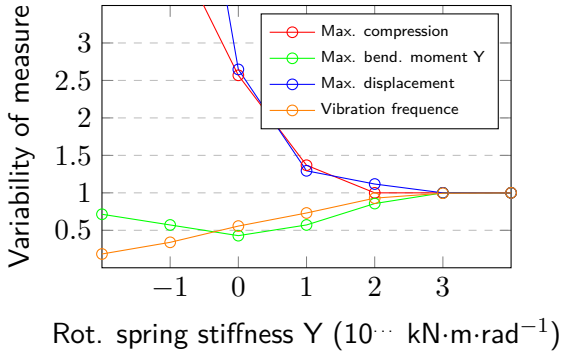
Measure	Symbol	Unit Y	IASS Xmesh pavilion			CLC Xmesh pavilion		
Number of beams			310			81		
Number of flat nodes			105			25		
Number of 120-3D nodes			105			27		
Avg. beam length	l	mm	265			1150		
Beam section	A_b	mm^2	100x4			140x60		
Strip exp. along beam edge		%	80			25		
Strip section	A_s	mm^2	1.5x15			12x70		
Beam material			Alu. 5754 O/H111			Wood GL28		
Strip material			Alu. 5754 O/H111			Wood Plywood		
Young modulus	$E/E_{0,mean}$	GPa	70			12.6		
Structure total mass	M	kg	150			500		
Load combination			G	G+A	G-A	G	$G+A_1$	$G+A_2$
Max. compression X	N_x^c	kN	1.01	1.28	0.99	1.75	2.25	1.65
Max. tension X	N_x^t	kN	0.22	0.32	0.17	1.40	1.60	1.20
Max. shear Y	V_y	kN	0.26	0.33	0.20	1.60	2.35	1.75
Max. shear Z	V_z	kN	0.18	0.21	0.08	0.30	0.45	0.30
Max. torsion X	T_x	kN·m	0.01	0.01	0.04	0.15	0.45	0.10
Max. moment Y	M_y	kN·m	0.05	0.07	0.07	0.25	0.60	0.45
Max. moment Z	M_z	kN·m	0.01	0.01	0.01	0.50	0.75	0.50
Max. stress in beam	σ	MPa	46.9	55.0	40.6	N.A.	N.A.	N.A.
Proof/bending strength	$f_0/f_{m,k}$	MPa	80	80	80	24	24.	24
Max. displacement	δ	cm	2.4	3.9	2.8	2.5	6.8	2.8
Max. displacement Z	δ_z	cm	2.0	2.1	2.1	2.5	6.5	2.8
Vibration frequence	ν	Hz	1.70			2.70		

Table 3.5: Data about the structures of both IASS Xmesh and CLC Xmesh pavilions. Beams' forces and moments are computed with ultimate load factors for combination G and service load factors for combinations G+A/G+A_1 and G-A/G+A_2. Nodes' displacements are computed with service load factors for all combinations.

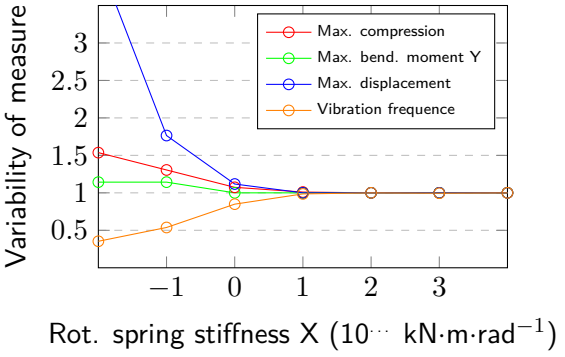
3.5 Sensitivity analysis

In this last section, we propose studying the influence of the rotational spring stiffness about beams' Y axis (in conjunction with the bending moment about beams' Y axis) of the connections at 120-3D nodes on the structural performances of the IASS Xmesh pavilion.

Throughout our various iterations of analysis of both pavilion projects, we have observed that the structural performances could vary significantly according to the values of spring stiffness that we would consider for the connections at 120-3D nodes.
More specifically, in the IASS Xmesh pavilion project, we have observed that the rotational spring stiffness about beams' Y axis (in conjunction with the bending moment about beams' Y axis) would have a significant impact on mechanical measures like maximal forces, moments or displacements.
The Figure 3.12a shows the influence of this rotational spring stiffness on various structural measures. This influence is much more significant than the one of the rotational strping stiffness about the beams' X as shown on the Figure 3.12b.



(a) Sensitivity to rotational spring stiffness Y



(b) Sensitivity to rotational spring stiffness X

Figure 3.12: *Sensitivity* of the structural performances to the *rotational spring stiffness* of the 120-3D nodes. Mechanical measures are obtained from the finite elements model of the IASS Xmesh pavilion. The reference stiffness condition from which the variability of measures is computed is such that all the degrees of freedom of the 120-3D nodes are blocked except the rotational degree studied that is set to 10^4kN·m·rad^{-1}.

It would be relevant to study the influence of the rotational spring stiffness of the connections at the 120-3D nodes on mechanical measures with a more general geometry than the one of the IASS Xmesh pavilion.
This study would reveal if this sensitivity is rather a peculiar property of the IASS Xmesh structure or if it is a general property of structures based on Xmeshes. In the latter case, this would suggest that designers that would like to use Xmesh geometries to conceive rigid gridshells shall be aware that rotational spring stiffness of 120-3D nodes about the beams' Y axis has a significant influence on the mechanical performances of the structure and that it is worth it to invest time and efforts to improve this stiffness in the connections.

Conclusion

In this study, we described *Xmesh* geometries and explained how they can stand as good geometric options for the design of free-form rigid gridshells. We developed a post-rationalization method, based on geometric optimization, to generate such geometries and we modeled their structural behaviour. Through the realizations of two free-form pavilions, which use very different technologies, we proved the potential and the adaptability of these geometries as well as our methods of mesh definition, optimization, structural analysis and generation of construction plans. Finally, both realizations proved the success of methods integrating permanently architectural, structural, manufacturing and constructive considerations, for the purpose of economic and efficient free-form designs.

Still, we also highlighted few *limits* about our work throughout this document. On the geometric plan, we observed that our generation method does not allow a complete free-of-constraints formal freedom. Regarding the initialization method that is based on lines of curvatures, designers should stay away from non-smooth surfaces or surfaces featuring umbilical points as they would prevent to get a regular grid of lines. Regarding the optimization process, designers should be careful about fast changes of curvatures and intensity of curvatures relatively to the typical size of the mesh's edges.

However, we can qualify these limits as they are not peculiar to Xmeshes. They rather are related to our generation method and our way to materialize these geometries. About the initialization, we could imagine other ways to set up a hexagonal mesh without using lines of curvatures. As explained by H. Pottmann *et al.* in [8], *Darboux cyclides* are algebraic surface which carries up to 6 families of circles. They can feature triples of circles families inscribed in their surface forming hexagonal webs. In other words, Darboux cyclides are surfaces natural featuring hexagonal meshes. By restraining the set of target surfaces to Darboux cyclides or portions of them, we could then initiate hexagonal meshes to optimize. We would still have to give a theoretical reason why to use such natural meshes. In order to expand the set of surfaces with natural hexagonal mesh, we could also imagine patching portion of Darboux cyclides similarly to what R. Mesnil does in [5]. Furthermore, we could also work on developing a pre-rationalization method rather than a post-rationalization one.

Last but not the least, one aspect of this Xmesh study that has not been deeply been developed in this document is the fabrication along with the construction. Some elements about manufacturing and constructive considerations have been mentioned throughout the various chapters. Yet, it could have been possible to write an entire chapter just about the generation of the fabrication data. The reason why we have not developed such a chapter is that this step is about to be finished at the time when this document is been written. However, readers must be aware that it has been an important part of our work as well as a main factor of influence for the designs as Xmeshes are fabrication-and-construction-aware geometries.

Bibliography

[1] Sofien Bouaziz, Mario Deus, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. Shape-up: Shaping discrete geometry with projections. *Eurographics Symposium Proceedings*, August 2012.

[2] Evolute. Evolute website. `https://www.evolute.at/`, August 2019.

[3] IASS. Iass website. `https://www.iass-structures.org/`, August 2019.

[4] Erich Handel Johann Sischka, Stephen Brown and Günther Zenkner. Die Überdachung des great court im british museum in london. *Ernst & Sohn*, March 2013.

[5] Romain Mesnil, Cyril Douthe, Olivier Baverel, and Bruno Leger. Möbius geometry and cyclidic nets: A framework for complex shape generation. *IASS Proceedings*, August 2015.

[6] Romain Mesnil, Cyril Douthe, Olivier Baverel, Bruno Leger, and Jean-François Caron. Isogonal moulding surfaces: a family of shapes for high node congruence in free-form structures. *HAL*, September 2015.

[7] Romain Mesnil, Cyril Douthe, Tristan Gobin, and Olivier Baverel. Form finding and design of a timber shell-nexorade hybrid. *AAG Proceedings*, September 2018.

[8] Helmut Pottmann, Ling Shi, and Mikhail Skopenkov. Darboux cyclides and webs from circles. *Elsevier*, January 2012.

[9] Clemens Preisinger. *Karamba, User Manual for version 1.2.2*, October 2016.

[10] Dassault System. Abaqus website. `https://www.3ds.com/`, January 2019.

[11] Xavier Tellier, Olivier Baverel, Cyril Douthe, and Laurent Hauswirth. Gridshells without kink angle between beams and cladding panels. *IASS Proceedings*, July 2018.

[12] Xavier Tellier, Sonia Zerhouni, Guillame Jami, Alexandre Le Pavec, Thibault Lenart, Mathieu Lerouge, Nicolas Leduc, Cyril Douthe, Laurent Hauswirth, and Olivier Baverel. Hybridizing vertex and face normals to design torsion free structures: application to the xmesh pavilion. *IASS Proceedings*, October 2019.

[13] Wenping Wang and Yang Liu. A note on hexagonal meshes. *Nonlinear Computational Geometry, Springer*, pages 221–233, January 2009.

[14] Wenping Wang, Yang Liu, Dongming Yan, Bin Chan, Ruotian Ling, and Feng Sun. Hexagonal meshes with planar faces. January 2008.

Appendices

Figure A.1: Photo of the *team* participating to the IASS contest. From left to right, L. Hauswirth, O. Baverel, N. Leduc, T. Lenart, A. Le Pavec, G. Lami, S. Zerhouni, M. Lerouge, C. Douthe, X. Tellier

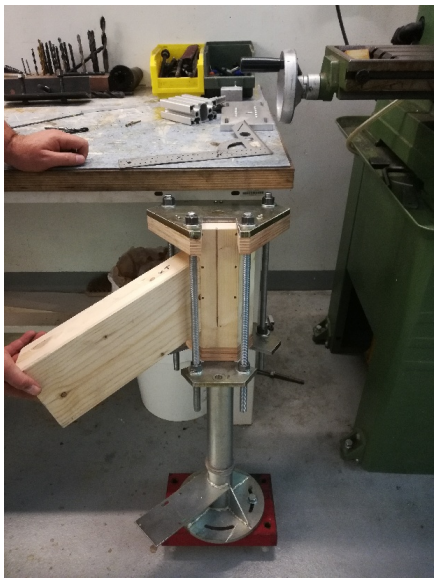


Figure A.2: Photo of the *platform* for the construction of the 120-3D node of the CLC Xmesh pavilion.

Listing A.1: Torsion-free custom goal

```
public class TorsionFreeNode : GoalObject
{
    // Constructor
    public TorsionFreeNode(Plane MainPlane, List<Plane> NeighborPlanes,
        double Strength)
    {
      // Initialization of particles
      PPos = new Point3d[4];
      PPos[0] = MainPlane.Origin;
      for (int i = 0; i < 3; i++)
        PPos[i + 1] = NeighborPlanes[i].Origin;
      InitialOrientation = new Plane[4];
      InitialOrientation[0] = MainPlane;
      for (int i = 0; i < 3; i++)
        InitialOrientation[i + 1] = NeighborPlanes[i];

      // Initialization of translation and rotation goals
      Move = new Vector3d[4];
      Torque = new Vector3d[4];
      Weighting = new double[4];
      TorqueWeighting = new double[4];
      for (int i = 0; i < 4; i++)
      {
        Torque[i] = new Vector3d(0, 0, 0);
        TorqueWeighting[i] = Strength;
      }
    }

    // Calculate method
    public override void Calculate(List<KangarooSolver.Particle> p)
    {
      // Data from particles in current state
      Point3d MainPoint = p[PIndex[0]].Orientation.Origin;
      Vector3d MainNormal = p[PIndex[0]].Orientation.Normal;
      Point3d[] NeighborPoints = new Point3d[3];
      Vector3d[] NeighborNormals = new Vector3d[3];
      for (int i = 0; i < 3; i++)
      {
        NeighborPoints[i] = p[PIndex[i + 1]].Orientation.Origin;
        NeighborNormals[i] = p[PIndex[i + 1]].Orientation.Normal;
      }

      // Declarations and calculations
      Vector3d[] MainToNeighborVectors = new Vector3d[3];
      Vector3d[] TorqueVectors = new Vector3d[3];
```

```
50          for (int i = 0; i < 3; i++)
51          {
52            MainToNeighborVectors[i] = NeighborPoints[i] - MainPoint;
53            MainToNeighborVectors[i].Unitize();
54            TorqueVectors[i] = Vector3d.CrossProduct(NeighborNormals[i],
55                MainNormal);
56            TorqueVectors[i] = Vector3d.Multiply(MainToNeighborVectors[i],
57                TorqueVectors[i]) * MainToNeighborVectors[i];
58          }
59
60          // Updates rotation goals
61          for (int i = 0; i < 3; i++)
62          {
63            Torque[0] += - TorqueVectors[i];
64            Torque[i + 1] = TorqueVectors[i];
65          }
66        }
67    }
```

Listing A.2: Orthogonality custom goal - rotating main node's normal

```
public class OrthogonalityRotatingMain : GoalObject
{
    // Constructor
    public OrthogonalityRotatingMain(Plane MainPlane,
        List<Plane> NeighborPlanes, double Strength)
    {
      // Initialization of particles
      PPos = new Point3d[4];
      PPos[0] = MainPlane.Origin;
      for (int i = 0; i < 3; i++)
        PPos[i + 1] = NeighborPlanes[i].Origin;
      InitialOrientation = new Plane[4];
      InitialOrientation[0] = MainPlane;
      for (int i = 0; i < 3; i++)
        InitialOrientation[i + 1] = NeighborPlanes[i];

      // Initialization of translation and rotation goals
      Move = new Vector3d[4];
      Torque = new Vector3d[4];
      Torque[0] = new Vector3d(0, 0, 0);
      Weighting = new double[4];
      TorqueWeighting = new double[4];
      TorqueWeighting[0] = Strength;
    }

    // Calculate method
    public override void Calculate(List<KangarooSolver.Particle> p)
    {
      // Datas from particles in current state
      Point3d MainPoint = p[PIndex[0]].Orientation.Origin;
      Vector3d MainNormal = p[PIndex[0]].Orientation.Normal;
      Point3d[] NeighborPoints = new Point3d[3];
      for (int i = 0; i < 3; i++)
        NeighborPoints[i] = p[PIndex[i + 1]].Orientation.Origin;

      // Declaration and calculations
      Plane NeighborsPlane = new Plane(NeighborPoints[0],
        NeighborPoints[1], NeighborPoints[2]);
      Vector3d TorqueDirectionVector = new Vector3d(
        Vector3d.CrossProduct(MainNormal, NeighborsPlane.Normal));
      double DifferenceAngle = Vector3d.VectorAngle(MainNormal,
        NeighborsPlane.Normal, TorqueDirectionVector);

      // Update rotation goals
      Torque[0] = TorqueDirectionVector;
     }
}
```

Listing A.3: Orthogonality custom goal - rotating neighbors' planes

```
public class OrthogonalityMovingNeighbors : GoalObject
  {
    // Constructor
    public OrthogonalityMovingNeighbors(Plane MainPlane,
        List<Plane> NeighborPlanes, double Strength)
    {
      // Initialization of particles
      PPos = new Point3d[4];
      PPos[0] = MainPlane.Origin;
      for (int i = 0; i < 3; i++)
        PPos[i + 1] = NeighborPlanes[i].Origin;
      InitialOrientation = new Plane[4];
      InitialOrientation[0] = MainPlane;
      for (int i = 0; i < 3; i++)
        InitialOrientation[i + 1] = NeighborPlanes[i];

      // Initialization of translation and rotation goals
      Move = new Vector3d[4];
      Torque = new Vector3d[4];
      Weighting = new double[4];
      TorqueWeighting = new double[4];
      for (int i = 1; i < 4; i++)
      {
        Move[i] = new Vector3d(0, 0, 0);
        Weighting[i] = Strength;
      }
    }

    // Calculate method
    public override void Calculate(List<KangarooSolver.Particle> p)
    {
      // Datas from particles in current state
      Point3d MainPoint = p[PIndex[0]].Orientation.Origin;
      Vector3d MainNormal = p[PIndex[0]].Orientation.Normal;
      Point3d[] NeighborPoints = new Point3d[3];
      for (int i = 0; i < 3; i++)
        NeighborPoints[i] = p[PIndex[i + 1]].Orientation.Origin;

      // Declarations and calculations
      Plane NeighborsPlane = new Plane(NeighborPoints[0],
        NeighborPoints[1], NeighborPoints[2]);
      Vector3d RotationDirectionVector = new Vector3d(
        Vector3d.CrossProduct(MainNormal, NeighborsPlane.Normal));
      double DifferenceAngle = Vector3d.VectorAngle(NeighborsPlane.Normal,
        MainPlane.Normal, RotationDirectionVector);
      Point3d[] NewNeighborPoints = new Point3d[3];


```

```
50        for (int i = 0; i < 3; i++)
51        {
52          NewNeighborPoints[i] = NeighborPoints[i];
53          NewNeighborPoints[i].Transform(Transform.Rotation(
54              -DifferenceAngle, RotationDirectionVector, MainPoint));
55        }
56
57        // Goal assignments
58        for (int i = 0; i < 3; i++)
59          Move[i + 1] = new Vector3d(NewNeighborPoints[i] - NeighborPoints[i]);
60      }
61    }
```

Listing A.4: 120-3D custom goal - moving neighboring nodes

```
public class OneTwentyNeighbors : GoalObject
{
    // Constructor
    public OneTwentyNeighbors(Plane MainPlane,
        List<Plane> NeighborPlanes, double Strength)
    {
      // Initialization of particles
      PPos = new Point3d[4];
      PPos[0] = MainPlane.Origin;
      for (int i = 0; i < 3; i++)
        PPos[i + 1] = NeighborPlanes[i].Origin;
      InitialOrientation = new Plane[4];
      InitialOrientation[0] = MainPlane;
      for (int i = 0; i < 3; i++)
        InitialOrientation[i + 1] = NeighborPlanes[i];

      // Initialization of translation and rotation goals
      Move = new Vector3d[4];
      Weighting = new double[4];
      for (int i = 0; i < 4; i++)
        Weighting[i] = Strength;
    }

    // Calculate method
    public override void Calculate(List<KangarooSolver.Particle> p)
    {
      // Datas from particles in current state
      Point3d MainPoint = p[PIndex[0]].Position;
      Plane MainPlane = p[PIndex[0]].Orientation;
      Point3d[] NeighborPoints = new Point3d[3];
      Vector3d[] MainToNeighborUnitVectors = new Vector3d[3];
      for (int i = 0; i < 3; i++)
      {
        NeighborPoints[i] = p[PIndex[i + 1]].Position;
        MainToNeighborUnitVectors[i] = new Vector3d(NeighborPoints[i] - MainPoint);
        MainToNeighborUnitVectors[i].Unitize();
      }

      // Declaration and calculations
      double[] MeasuredAngles = new double[3];
      double[] DifferenceAngles = new double[3];
      double[] MovingAngles = new double[3];
      for (int i = 0; i < 3; i++)
      {
        MeasuredAngles[i] = Vector3d.VectorAngle( MainToNeighborUnitVectors[i],
            MainToNeighborUnitVectors[(i + 1) % 3], MainPlane);
        DifferenceAngles[i] = (MeasuredAngles[i] - 2 * Math.PI / 3) / 2;
      }

```

```
50          for (int i = 0; i < 3; i++)
51          {
52            MovingAngles[i] = DifferenceAngles[i] - DifferenceAngles[(i + 2) % 3];
53            if (MovingAngles[i] < 0)
54              MovingAngles[i] += 2 * Math.PI;
55            NeighborPoints[i].Transform(
56                Transform.Rotation(MovingAngles[i], MainPlane.Normal, MainPoint));
57          }
58
59          // Update translation goals
60          Move[0] = new Vector3d(0, 0, 0);
61          for (int i = 0; i < 3; i++)
62          {
63            Move[i + 1] = new Vector3d(NeighborPoints[i] - p[PIndex[i + 1]].Position);
64            Move[i + 1] = 0.5 * Move[i + 1];
65            Move[0] -= Move[i + 1];
66          }
67        }
68  }
```

Listing A.5: 120-3D custom goal - moving main node

```
public class OneTwentyMain : GoalObject
{


    // Constructor
    public OneTwentyMain(Plane MainPlane,
        List<Plane> NeighborPlanes, double Strength)
    {
      // Initialization of particles
      PPos = new Point3d[4];
      PPos[0] = MainPlane.Origin;
      for (int i = 0; i < 3; i++)
        PPos[i + 1] = NeighborPlanes[i].Origin;
      InitialOrientation = new Plane[4];
      InitialOrientation[0] = MainPlane;
      for (int i = 0; i < 3; i++)
        InitialOrientation[i + 1] = NeighborPlanes[i];

      // Initialization of translation and rotation goals
      Move = new Vector3d[4];
      Weighting = new double[4];
      Weighting[0] = Strength;
    }



    // Calculate method
    public override void Calculate(List<KangarooSolver.Particle> p)
    {
      // Datas from particles in current state
      Point3d MainPoint = p[PIndex[0]].Orientation.Origin;
      Plane MainPlane = p[PIndex[0]].Orientation;
      Point3d[] NeighborPoints = new Point3d[3];
      for (int i = 0; i < 3; i++)
        NeighborPoints[i] = p[PIndex[i + 1]].Orientation.Origin;

      // Declaration and calculations
      Point3d[] ProjectedNeighborPoints = new Point3d[3];
      Point3d[] PushOutNeighborPoints = new Point3d[2];
      Line[] CrossingLines = new Line[2];
      for (int i = 0; i < 3; i++)
      {
        ProjectedNeighborPoints[i] = NeighborPoints[i];
        ProjectedNeighborPoints[i].Transform(
            Transform.PlanarProjection(MainPlane));
      }


```

```
50          for (int i = 0; i < 2; i++)
51          {
52            PushOutNeighborPoints[i] = ProjectedNeighborPoints[i + 1];
53            PushOutNeighborPoints[i].Transform(Transform.Rotation(-Math.PI / 3,
54                MainPlane.Normal, ProjectedNeighborPoints[i]));
55            CrossingLines[i] = new Line(
56                PushOutNeighborPoints[i], ProjectedNeighborPoints[(i + 2) % 3]);
57          }
58          double x, y;
59          if(!Rhino.Geometry.Intersect.Intersection.LineLine(
60            CrossingLines[0], CrossingLines[1], out x, out y))
61          {
62            Rhino.RhinoApp.WriteLine("No intersection found.");
63          }
64          Point3d NewMainPoint = CrossingLines[0].PointAt(x);
65
66          // Update translation goals
67          Move[0] = new Vector3d(NewMainPoint - MainPoint);
68        }
69  }
```