Journées du Groupe de Travail en Modélisation Géométrique 2018, Aix-en-Provence

Contraintes : équations ou procédures ?

Dominique Michelucci[1], Jean-Philippe Pernot[2], Marc Daniel[3] et Sebti Foufou[4]

[1]Université de Dijon
[2]ENSAM ParisTech Aix-en-Provence
[3]LSIS Marseille
[4]American University of Abu Dabi

Résumé
Aujourd'hui, tous les modeleurs géométriques de CFAO fournissent des outils pour la modélisation géométrique avec des contraintes. Historiquement, les contraintes géométriques en 3D ont été représentées avec des équations, et les problèmes irréductibles ont été résolus avec une variante de la méthode de Newton, ou avec la méthode d'intersection des lieux. Cet article propose de remplacer les équations par des procédures. Le modeleur appelle le solveur, qui peut appeler toutes les procédures du modeleur. À leur tour, les procédures peuvent appeler le solveur pour les sous-problèmes. Cette approche rend possibles des choses impossibles auparavant et elle facilite la spécification des contraintes de CFAO sur les formes composites avec des formats hétérogènes. Le calcul de valeurs précises pour les dérivées est toujours possible. L'utilisation d'outils pour corriger et décomposer des systèmes en sous-systèmes ou pour exploiter la faible densité est toujours possible. Il est toujours possible d'utiliser des solveurs numériques de pointe et de profiter de la faible densité. Cet article présente les avantages de cette approche, ses inconvénients, les problèmes posés et sa mise en œuvre.

Cet article propose de représenter les contraintes avec des procédures, au lieu d'équations, chaque fois que cela est possible. §1 présente le sujet. §2.1 présente des méthodes existantes pour résoudre des contraintes géométriques. §2.2 présente des méthodes existantes pour le débogage, la décomposition et l'exploitation du caractère creux des systèmes d'équations. §2.3 présente les procédures existantes pouvant être appelées par le solveur. §3 présente les avantages de cette approche, ses inconvénients, les problèmes posés. §3.1 présente les avantages, 3.2 présente les inconvénients, 3.3 présente les questions posées. §4 présente une implémentation. §5 conclut.

1. Introduction

Cette section présente les grandes lignes. Aujourd'hui, tous les modeleurs géométriques fournissent des outils pour la modélisation géométrique avec des contraintes [HJA05]. §2.1 présente des méthodes standard pour résoudre les contraintes géométriques. §2.2 présente des méthodes standard pour l'étude qualitative des contraintes géométriques. La plupart du temps, les systèmes élémentaires de contraintes 3D sont traduits en systèmes d'équations non linéaires, qui sont résolus avec une méthode numérique itérative, comme une variante de la méthode de Newton.

Cet article propose de remplacer les équations par des procédures, fournies par le modeleur (ou les modeleurs). Nous utilisons le mot «procédure» pour éviter l'ambiguïté du mot «fonction», tous deux utilisés en informatique et en mathématiques. Les procédures (§2.3) reçoivent des valeurs connues (par exemple les vecteurs de valeurs numériques) ; elles calculent des fonctions mathématiques.

Le premier argument, et le plus fort, pour remplacer les équations par des procédures est que les procédures sont définitivement plus pratiques et plus puissantes que les équations. Les modeleurs géométriques sur le marché fournissent des procédures efficaces pour générer des formes, et pour interroger des formes et des ensembles de formes (appelés assemblages ou scènes). Par exemple, une procédure d'interrogation typique calcule d'une manière efficace le point d'une forme donnée S le plus proche (ou le le plus loin) d'un point

donné p. Cette procédure peut utiliser des GPU ou des structures de données d'accélération : octrees, BSP arbres, kd-arbres. Supposons maintenant que S est inconnue, car ses paramètres de génération sont inconnus. Supposons aussi qu'un point X soit dans S si $S(X,Y) = 0$ pour un système S. Pour trouver le point X de S le plus proche de p, nous pouvons résoudre le problème d'optimisation : $\min ||X-p||^2, S(X,Y) = 0$, ou nous pouvons résoudre le système Lagrangien associé et sélectionner la racine pertinente. En pratique, si un solveur de type Newton est utilisé, il devrait commencer suffisamment près de la racine attendue. Cependant, il existe une meilleure méthode. Lorsqu'un solveur itératif numérique est utilisé, il fournit des valeurs numériques (certes approximatives) pour les variables inconnues (en réalité, modifiables) X et Y à chaque étape. A l'initialisation, X^0, Y^0 sont lus sur l'esquisse. Et donc, il est bel et bien possible d'utiliser les procédures du modeleur même lorsque S est inconnu (en fait, connu mais modifiable). Il serait maladroit de ne pas utiliser ces procédures.

Un deuxième argument pour remplacer les équations par des procédures est que les équations ne sont pas disponibles dans certains cas. C'est vrai pour des formes comme les surfaces de subdivision et les fractales, qui n'ont pas d'équation mais sont le résultat de procédures. C'est vrai quand il s'agit de surfaces de forme libre souvent obtenues fastidieusement à partir de fonctions de modélisation assez sophistiquées (par exemple extrusion, lissage, mélange). Ceci est vrai lorsque l'on spécifie des contraintes sur des grandeurs mécaniques telles que la contrainte de Von Mises qui résulte d'une simulation complexe d'éléments finis. Un autre exemple d'optimisation de forme est la forme d'une aube de turbine : elle est le résultat d'un processus d'optimisation complexe qui vise au meilleur compromis entre notamment ses performances aérodynamiques et mécaniques.

Représenter des contraintes avec des procédures au lieu d'équations a de nombreux avantages (§3.1) et peu d'inconvénients (§3.2). §3.3 répond positivement à la question : est-il encore possible d'utiliser les méthodes standard existantes pour corriger, décomposer et résoudre avec cette représentation des contraintes ?

2. Méthodes standard pour les contraintes géométriques

Cette section résume les méthodes standard pour traiter les contraintes géométriques et pour l'étude qualitative des contraintes [HJA05]. Les contraintes sont représentées par des équations.

2.1. Résoudre les contraintes géométriques

Dans les problèmes géométriques euclidiens 2D classiques solubles à la règle et au compas comme le problème emblématique d'Appolonius, et les problèmes 3D dans la géométrie descriptive de Monge, les contraintes géométriques sont des incidences, des tangences, des distances ou des angles concernant des points, des droites, des cercles ou des coniques en 2D, ainsi que des plans, des sphères ou des quadriques en 3D (pas d'objets composites !). Ces problèmes sont directement traduits en systèmes d'équations polynomiales [DH00].

Les systèmes polynomiaux sont bien connus. De nombreuses méthodes de résolution sont disponibles.

Les solveurs géométriques, ou solveurs constructifs, décomposent le système de contraintes géométriques en sous-problèmes élémentaires, les résolvent et assemblent des solutions. En 2D, la décomposition peut être faite en utilisant des règles activées par le moteur d'inférence de certains systèmes experts [RSV89, VSR92], ou en utilisant des graphes, décomposés par une méthode ascendante ou descendante [JTNM06], ou en utilisant la méthode du témoin [MF09]. Il y a souvent une formule pour résoudre les sous-problèmes élémentaires (par exemple les triangles définis modulo les isométries (DMI) par trois contraintes, les quadrilatères DMI par cinq contraintes, les hexagones DMI par six contraintes). En 3D, les sous-problèmes élémentaires sont beaucoup plus nombreux et difficiles, donc d'autres méthodes de résolution sont nécessaires.

De nombreux problèmes spatiaux élémentaires peuvent être résolus avec la méthode d'intersection des lieux (LIM) : le principe est de supprimer une contrainte bien choisie, pour que l'ensemble des solutions devienne une courbe, qui est suivie ou échantillonnée jusqu'à ce que la contrainte supprimée soit satisfaite [GHY04, FS08, IMS11, ISM14, IMS15]. Les solveurs géométriques, ou solveurs constructifs, et LIM n'ont pas seulement besoin des équations, mais aussi des contraintes géométriques.

Le calcul formel [Kon92] résout exactement les systèmes polynomiaux, avec les bases de Groebner, ou la méthode de Wu-Ritt, etc. Le calcul formel fonctionne en temps au moins exponentiel. Il est utilisé dans les logiciels pédagogiques de géométrie dynamique. Il est rarement utilisé en CFAO [Kon92]. Le calcul formel repose sur des expressions symboliques d'équations polynomiales. Il peut simplifier ou résoudre des équations non polynomiales (par exemple utilisant des fonctions transcendantales) mais pas de manière garantie.

L'analyse d'intervalle [Jau01] peut isoler chaque racine régulière dans une boîte initiale donnée. Elle est

utilisée en robotique pour prouver que certains mécanismes sont corrects : ils ne peuvent pas se coincer ou se casser. L'analyse d'intervalle nécessite les équations pour contenir l'inflation des intervalles (effet enveloppant, "wrapping effect").

Les méthodes homotopiques [SWI05, DH00] trouvent toutes les racines réelles (ou complexes) des systèmes polynomiaux, et leur coût par racine est en temps polynomial. Les équations polynomiales sont nécessaires. L'homotopie est utilisée en robotique [SWI05]. L'homotopie peut aussi être utilisée comme une variante de la méthode de Newton amortie, et appartient alors à la classe suivante des méthodes itératives numériques ; dans ce cas, les équations peuvent être remplacées par des procédures.

De nombreuses méthodes itératives numériques sont fréquemment utilisées pour résoudre des systèmes d'équations non linéaires (éventuellement non polynomiales) : Newton ou Newton-Raphson, Newton amorti ou homotopie, et des optimiseurs tels que : Levenberg-Marquardt (LM) [GCG99a], la méthode de Broyden-Fletcher-Goldfarb-Shanno (BFGS) ou sa variante avec mémoire limitée (lm-BFGS), Hooke-Jeeves (HJ), la méthode du simplexe de Nelder-Mead (NMS) ou la méthode du simplexe de Torczon (TS). Pour résoudre $F(X) = 0$, les optimiseurs minimisent généralement la norme du résidu $||F(X)||_2^2$. Ces méthodes fonctionnent ausii bien avec des équations que des procédures, comme l'implantation en §4 le prouve. Un intérêt des solveurs de Newton généralisés (en utilisant la résolution des moindres carrés) et des optimiseurs est qu'ils peuvent résoudre des problèmes sur-contraints. Ceci se produit, par exemple, pour calculer le point d'intersection de trois cercles ou de trois courbes en 2D. La sur-contrainte n'est pas toujours une erreur : elle peut être utilisée pour éviter les racines parasites. Les solveurs et les optimiseurs de Newton généralisés peuvent également résoudre des problèmes sous-contraints.

Les méthodes itératives numériques nécessitent un point de départ X^0, appelé esquisse ou croquis en CFAO. L'esquisse est soit fournie par les utilisateurs (ingénieurs ou concepteurs) avec une interface graphique interactive, soit donnée par une version précédente du produit en cours de conception ou de modélisation, soit est le résultat d'un algorithme. En CFAO, à chaque étape d'une méthode itérative numérique, il est possible de visualiser l'état actuel de la figure. Elle est généralement comprise par les utilisateurs, qui peuvent vérifier que le solveur ne s'égare pas, et qui peuvent piloter le solveur. De plus, de nombreux outils interactifs permettent aux utilisateurs de détecter et de corriger les erreurs, comme les conflits entre les contraintes. De tels outils d'étude qualitative des contraintes sont essentiels.

Les contraintes strictement géométriques ne sont pas suffisantes pour la CFAO. Ainsi, à la fin des années 90, Hoffmann et al [HJA97], et Joan-Arinyo [JASR99] ont proposé une méthode hybride. L'idée est de combiner un solveur géométrique 2D et un solveur équationnel. Le solveur géométrique utilise l'approche constructive déjà mentionnée. Il est insuffisant en 3D, mais il peut être remplacé par LIM. Ainsi, au moins en principe, cette méthode hybride s'étend en 3D. Elle repose elle aussi sur des équations.

BFGS et lm-BFGS résolvent des problèmes d'optimisation non contraints. Byrd et al [BLNZ94] proposent une variante pour résoudre des problèmes d'optimisation contraints comme : $\min G(X)$ avec $L \leq F(X) \leq U$.

2.2. Outils existants pour l'étude qualitative

Il est essentiel de fournir aux utilisateurs des outils pour détecter les erreurs dans les contraintes et les corriger, pour décomposer les systèmes en sous-systèmes, et pour exploiter le caractère creux afin d'accélérer la résolution. C'est l'étude qualitative des contraintes. Nous mentionnons trois types d'outils.

Les méthodes du premier type appellent le solveur. Nous les appelons des protocoles. De nombreux protocoles ont été proposés afin de détecter des sous-systèmes contradictoires. Rappelons qu'un système trop contraint peut être contradictoire ou redondant. Voici un exemple d'un tel protocole : partir d'une figure proche de la solution attendue, et prendre en compte les contraintes (*i.e.,* résoudre) incrémentalement. Soit E_k la première contrainte non satisfaite. Alors $C = \{E_1, \dots E_k\}$ est contradictoire. Retirer ensuite de C toute contrainte E_i telle que $C - E_i$ n'est toujours pas satisfiable. Ce protocole donne le plus petit sous-système contradictoire et peut être utilisé lorsque les équations ne sont pas disponibles. Ce protocole utilise des propriétés combinatoires d'ensembles de contraintes dépendants / indépendants, qui sont formalisées par la théorie des matroïdes.

Les deux autres types d'outils n'appellent pas le solveur.

Le second type est un ensemble de méthodes combinatoires (ou structurelles) [Owe91, Ser91, Hen92, JTNM06] qui considèrent divers graphes. Par simplicité, cet article ne considère que le graphe biparti [Ser91, Hen92]. C'est un graphe biparti reliant des équations (maintenant des contraintes procédurales) et des inconnues (maintenant des variables modifiables). Les méthodes

combinatoires s'appuient sur la théorie du couplage (Matching Theory) [LP09]. Elles calculent le couplage maximal (en cardinalité) dans le graphe biparti. Elles détectent les parties structurellement sous-, sur- et bien-contraintes, et les parties structurellement sous-, sur- et bien-contraints modulo les isométries (éventuellement, modulo les similitudes). Dans les deux cas, la partie bien-contrainte se décompose en parties structurellement irréductibles. Ces outils peuvent toujours être utilisés lorsque les contraintes sont représentées par des procédures plutôt que par des équations. §4.4 explique comment construire le graphe biparti, à partir des procédures, plus préciséement de leur signature. Mais les méthodes combinatoires sont limitées : elles ne détectent que les conflits structurels. De plus une caractérisation combinatoire de la rigidité (bien-contraint modulo les isometries) est encore inconnue pour les contraintes géométriques générales et son existence est douteuse. La caractérisation combinatoire de la rigidité est le sujet de la théorie de la rigidité. Cette dernière ne considère que les distances génériques point-point, ce qui est insuffisant pour la CFAO. Le prochain type de méthodes n'a pas ces limitations.

Le troisième et dernier type d'outils est la méthode du témoin (witness) [MF09] et ses variantes [TSM$^+$11, MMS14, HKP17]. Le principe de ces méthodes est le suivant : supposons que nous voulons résoudre $F(U, X) = U - U_T = 0$, où U sont des noms de valeurs de paramètres (non modifiables), U_T est la valeur de U (T pour target, ou cible), X sont des noms de variables inconnues (modifiables). Un témoin est un couple U_W, X_W tel que U_W et X_W sont des vecteurs de valeurs numériques et tels que $F(U_W, X_W) = 0$. En outre, on suppose que le témoin est typique de (ou même très proche [HKP17] de) la cible, afin que le témoin et la cible partagent les mêmes propriétés combinatoires. Plus précisément, les rangs de chaque mineur dans le Jacobien, connu, du témoin $F'(U_W, X_W)$ et dans le Jacobien, inconnu, de la cible $F'(U_T, X_T)$ ($F'(U_T, X_T)$ est inconnu car les valeurs exactes de X_T sont inconnues) sont égaux. Sous des hypothèses bien choisies de typicalité et d'exactitude (par simplicité, il n'y a pas d'imprécision numérique), la méthode du témoin détecte toutes les dépendances entre les contraintes : elle est plus puissante que les méthodes structurelles. En pratique, les utilisateurs sont chargés de décider si le témoin est typique de la cible : par exemple, un triangle plat (respectivement un polyèdre plat) n'est pas typique d'un triangle (respectivement d'un polyèdre), et cette hypothèse ne pose pas de problème. L'hypothèse simplificatrice d'exactitude est irréaliste et plus problématique, mais Hao Hu et al [HKP17] ont récemment proposé des outils pour prendre en compte l'inévitable imprécision numérique, lors de l'utilisation de la méthode du témoin pour la modélisation de courbes et de surfaces de forme libre.

Tous ces outils peuvent toujours être utilisés lorsque les contraintes sont représentées avec des procédures plutôt qu'avec des équations. §4.4 explique la construction du graphe biparti. §4.5 explique comment la méthode du témoin peut être utilisée.

2.3. Procédures pouvant être appelées par le solveur

Dans toutes les procédures, les variables, également appelées arguments ou paramètres, sont des noms de valeurs connues. Dans les équations, les variables sont des noms d'inconnues, qui n'ont aucune valeur.

Les procédures les plus simples calculent des fonctions simples comme $\min(a, b)$, $\max(a, b)$, $|a|$. En fait, il n'y a pas de système polynomial pour caractériser $x = \min(a, b)$ pour les paramètres réels a et b. En d'autres termes, il n'y a pas de système polynomial tel que sa seule racine est $x = \min(a, b)$. En effet, il y a toujours des racines parasites (ici, $\max(a, b)$). La même chose vaut pour $x = \max(a, b)$, pour $x = |a|$, pour $x = \mathrm{sgn}(a)$ où sgn est le signe de a : $\mathrm{sgn}(0) := 0$ et $\mathrm{sgn}(a) := |a|/a$. Remarquez que toutes ces fonctions sont équivalentes : l'une suffit à définir les autres.

Un exemple 1D géométrique où ces fonctions sont nécessaires est la distance signée de $a \in \mathbb{R}$ au segment $[-1, 1]$, qui est $|a| - 1$. Un autre exemple est quand la racine la plus petite (ou la plus grande) est nécessaire, pour résoudre un problème d'optimisation qui ne peut pas être résolu par des procédures.

D'autres procédures simples et pratiques calculent les fonctions transcendantes $\exp, \log, \cos, \sin, \tan, \arctan$, etc. Elles sont nécessaires pour les hélices ou les spirales.

Le modeleur géométrique fournit de nombreuses procédures sophistiquées, pour le calcul des maillages, surfaces, NURBS, BRep, CSG, etc, pour l'ingénierie inverse, pour effectuer des simulations physiques, pour générer des commandes d'usinage, etc.

Il est pratique de distinguer les procédures de génération et les procédures d'interrogation. Les procédures de génération retournent des formes : structures de données pour les maillages, NURBS, arbres CSG, assemblages, scènes, etc. Le degré, la taille, la complexité d'une forme (par exemple, le nombre de sommets, les arêtes, les triangles d'un maillage et la topologie du maillage) peuvent varier avec les valeurs des paramètres de la procédure de génération. Ainsi, la complexité d'une forme peut changer pendant la résolution d'une itération à l'autre. Les équations n'ont pas une telle flexibilité. Les procédures d'interrogation

calculent des propriétés de la forme, comme la plus petite ou la plus grande distance entre deux formes données.

3. Avantages, inconvénients, problèmes

3.1. Avantages

Il était impossible de contraindre les surfaces de sudivision ou les fractales par des équations, car elles n'en ont pas, aussi des méthodes spécifiques étaient-elles nécessaires pour elles ; la représentation des contraintes par des procédures rend cela possible, en supposant que les procédures du modeleur s'appliquent aux surfaces de sudivision et aux fractales.

Il devient facile de gérer et de contraindre des géométries hétérogènes, comme les surfaces de subdivision, des maillages, des carreaux de NURBS restreints, des formes analytiques (quadriques, tores) [GFM$^+$16a], *e.g.,* pour imposer des contraintes de continuité G le long de surfaces contiguës de types différents comme les surfaces NURBS et de subdivision.

L'utilisation de procédures évite également de résoudre inutilement des problèmes d'optimisation : par exemple, le calcul d'une distance la plus courte ou la plus grande est effectué par une procédure d'un modeleur. Ainsi, cela évite également les problèmes d'optimisation imbriqués, difficiles à gérer.

Un autre avantage est qu'il devient possible de spécifier des contraintes sur les figures composites parce que les procédures des modeleurs gèrent de telles figures. Une figure composite est composée de plusieurs parties, avec des dimensions topologiques différentes : points, segments ou morceaux de courbes, morceaux de surfaces, morceaux de volumes. Un segment est l'objet composite le plus simple. Un maillage ou une BRep sont des objets composites. En passant, les problèmes géométriques classiques (comme le problème d'Appolonius) ne considèrent jamais les objets composites, car il n'y a plus de constructions géométriques à la règle et au compas. En CFAO, les objets composites sont essentiels : les formes, assemblages et scènes de la CFAO sont toujours des objets composites.

Nous expliquons maintenant pourquoi il est impossible, ou difficile, de spécifier des contraintes sur des figures composites avec des équations. S'il est simple de poser un système d'équations équivalent à : $A(X) = 0$ et $B(X) = 0$ (c'est juste $A(X) = B(X) = 0$), il est plus problématique de poser un système d'équations équivalent à : $A(X) = 0$ ou $B(X) = 0$, ce qui est nécessaire pour calculer la distance d'un point donné à une figure composée, ou de manière équivalente, pour calculer quel point de la partie de la figure composite est le plus proche du point donné. On peut penser à $||A(X)|| \times ||B(X)|| = 0$, mais le risque est élevé pour le solveur itératif d'être piégé par une racine parasite (un maximum local) ou par un minimum local. Il y a aussi un problème combinatoire, comme dans le problème SAT (satisfaction de contraintes booléennes) ou la programmation linéaire (quelles variables sont nulles ?). En comparaison, une procédure trouve le résultat exact et est bien plus rapide.

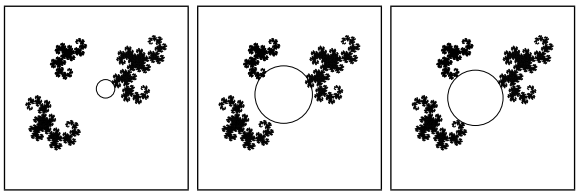

Figure 1: *Généralisation du problème d'Appolonius à trois objets arbitraires. La figure montre les trois premières itérations de BFGS. Elles suffisent pour converger visuellement vers une solution.*

Le recours à des procédures évite les inconvénients des systèmes d'équations non linéaires : il y a un nombre exponentiel de racines mais seules les racines réelles proches de l'esquisse sont pertinentes. Il n'y a pas de système polynomial caractérisant $x = \min(a, b)$ où a et b sont des paramètres réels. Idem pour $x = \max(a, b)$, pour $x = |a|$, car il y a toujours des racines parasites. Pire, certains problèmes géométriques ont des continuums de solutions parasites, dégénérées, (par exemple des solutions plates au lieu de solutions 3D), et des méthodes de déflation sont nécessaires.

La mise en œuvre de l'approche proposée ne nécessite pas le développement d'un nouveau modeleur, contrairement au projet DECO [GFM$^+$16a].

Un autre avantage est que le solveur peut utiliser un grand nombre de procédures géométriques très sophistiquées et efficaces disponibles dans le modeleur, ou dans le logiciel PDP. Quelques exemples de telles procédures sont les calculs de distances, les distances les plus éloignées, les profondeurs d'interpénétration, les boîtes englobantes, les volumes, les intersections, les opérations booléennes entre les solides, le mélange, le filetage, le maillage, la reconstruction, etc. Dans l'approche classique, la contrainte de distance ne peut concerner que deux éléments simples (points, droites ou plans), alors qu'avec des contraintes procédurales, cette contrainte peut concerner des formes composites complexes comme des assemblages. La figure 1 montre la généralisation à trois objets arbitraires du problème d'Appolonius qui consiste à trouver le cercle tangent à trois cercles donnés. Les contraintes procédurales permettent de généraliser à trois objets arbitraires, en ré-

solvant avec BFGS ou n'importe quelle autre méthode numérique itérative : $d_A(x,y) - R = d_B(x,y) - R = d_C(x,y) - R = 0$. La procédure $d_A(x,y)$ calcule la distance du point (x,y) à l'objet A, alors que x, y et R sont les inconnues. Il est possible de généraliser en utilisant les plus grandes distances D_A, D_B, D_C et en résolvant : $(d_A(x,y)-R)(D_A(x,y)-R) = (d_B(x,y)-R)(D_B(x,y)-R) = (d_C(x,y)-R)(D_C(x,y)-R) = 0$.

L'intégration d'un solveur de contraintes procédurales dans un modeleur paramétrique existant, tel que FreeCAD [Frea] et FreeSHIP [Freb] pour la CFAO, ou Blender [Ble] pour l'infographie, apporte plusieurs avantages pour un faible coût, tels que : i) améliorer les fonctionnalités du modeleur, (ii) ouvrir plus de possibilités au modeleur en termes de formulation et de résolution de contraintes, (iii) simplifier le solveur ainsi que le modeleur en termes de fonctionnalités et d'utilisation.

3.2. Inconvénients

Cette section présente les inconvénients de représenter les contraintes par des procédures. Un indéniable inconvénient est que le calcul formel ne peut plus être utilisé. Par exemple, la méthode de Buchberger considère les équations comme des règles de réécriture : $x^2 - 2 = 0$ est converti en la règle : $x^2 \to 2$. Ce n'est plus possible lorsque les équations ne sont pas disponibles (en supposant qu'elles soient polynomiales). De même, l'analyse d'intervalles [Jau01] ne peut plus être utilisée. L'arithmétique d'intervallee peut toujours être utilisée. Cependant, il est difficile de contenir l'inflation des intervalles lorsque les équations ne sont pas disponibles. Dans ce contexte, l'arithmétique par intervalles ne peut donc être utilisée que pour des intervalles initialement fins, par exemple pour le calcul des tolérances.

Pour éviter un malentendu possible, remarquez que les procédures du modeleur peuvent toujours utiliser et appeler des solveurs de subdivision [EK01, FM12], typiquement pour résoudre des sous-problèmes géométriques impliquant des courbes, surfaces ou volumes de forme libre. Fondamentalement, les procédures ont seulement besoin de connaître les valeurs numériques des coordonnées des points de contrôle, afin d'appeler les solveurs de subdivision. Les solveurs de subdivision s'appuient sur les propriétés géométriques bien connues des bases de Bernstein et des bases de splines, comme la propriété de l'enveloppe convexe et la propriété de diminution de la variation. Pour les mêmes raisons, les procédures peuvent également utiliser l'analyse par intervalles pour résoudre des sous-problèmes géométriques.

Un autre inconvénient est dû aux fonctions non analytiques (splines, NURBS, $\min, \max, |.|$, fonctions utilisant les branchements if-then-else). Ces fonctions sont non polynomiales. Elles sont essentielles en CFAO : il n'est pas possible de ne pas les prendre en compte. Les procédures les gèrent directement, contrairement aux équations. Mais l'utilisation de fonctions non analytiques a deux conséquences. Seule la seconde est vraiment un inconvénient.

Première conséquence, le problème de la dépendance d'une fonction non analytique à une variable (est-ce que $F_l(X)$ dépend de X_c ?) est indécidable. Le problème est décidable de manière probabiliste pour les fonctions analytiques, mais est coûteux en temps. Aussi, la meilleure solution est-elle que le modeleur informe le solveur des dépendances, autrement dit quelles entrées dans le jacobien ne sont pas structurellement nulles. Appelons cette information les données de parcimonie ou le graphe biparti. Il s'agit du graphe biparti classique reliant les équations (maintenant les procédures) et les inconnues (maintenant les variables modifiables). Ce graphe est utilisé pour détecter la partie structurellement sur-, sous- et bien-contrainte, et les sous-systèmes irréductibles bien contraints dans la partie bien contrainte. Les méthodes de l'algèbre linéaire creuse s'appuient également sur ce graphe [Kho12]. Aujourd'hui, l'interface de certains solveurs ou optimiseurs non linéaires ne tient pas compte des données de parcimonie. En supposant que ces données sont disponibles, il est toujours possible (§4.5) d'exploiter le caractère creux des systèmes de contraintes en CFAO, *i.e.,* le fait que les résultats des procédures ne dépendent pas de toutes les variables inconnues (ou plutôt connues mais modifiables).

Deuxième conséquence, les solveurs homotopiques qui trouvent toutes les racines des systèmes polynomiaux ne sont plus utilisables, pour deux raisons : premièrement, les équations ne sont plus disponibles, et deuxièmement, même si elles l'étaient, les fonctions calculées par les procédures sont non analytiques, donc non polynomiales, donc la théorie mathématique de l'homotopie ne s'applique plus. Par exemple, le théorème fondamental de l'algèbre (un polynôme de degré d a d des racines réelles ou complexes, ou encore $\mathcal{C}$ est un champ algébriquement clos) ne s'applique pas aux polynômes par morceaux.

3.3. Sur l'utilisation des anciennes méthodes

Une question naturelle est : peut-on continuer à utiliser les mêmes méthodes quand les contraintes sont représentées par des procédures, et non plus par des équations ? §3.3.1 considère le calcul des dérivées. §3.3.2 considère l'étude qualitative. §4 présente une implémentation qui montre que résoudre avec des pro-

cédures est aussi rapide que de résoudre avec des équations.

3.3.1. Le problème du calcul des dérivées

Un inconvénient apparent de préférer les procédures aux équations semble être que, puisque l'expression du côté gauche (LHS) des équations n'est plus disponible, il est impossible de calculer l'expression symbolique des dérivées (*i.e.,* si $f(x) = x^2$, alors $f'(x) = 2x$). Il est encore possible de calculer numériquement des différences finies, mais cela peut être trop imprécis, en particulier pour la méthode du témoin (qui calcule les rangs des mineurs des jacobiens). Il est encore possible d'utiliser la dérivation symbolique (à la compilation), mais cette dernière a des limitations, pour les instructions if-then-else, pour les boucles, pour la récursion, pour $\min, \max, |.|$, etc., et n'est pas facile à utiliser. Un résultat significatif, en §4.6 est que la dérivation automatique (à l'exécution), avec l'arithmétique des nombres duaux (ou infinitésimaux) [MF09, Fis17], calcule des valeurs précises (avec la précision de l'arithmétique à virgule flottante) des dérivées en un point donné ($f'(3) = 6$ si $f(x) = x^2$). Par exemple, si une forme dépend, d'un petit nombre $n = 6$ de paramètres $U = (u_1, \dots u_n)$, une simulation par éléments finis utilisant l'arithmétique des nombres duaux et calculant une performance $p(u_1, u_2, \dots u_n)$ calcule automatiquement, en même temps, $p(U)$ et toutes ses dérivées, *i.e.,* le vecteur gradient $\nabla p = (\partial p/\partial u_i(U))$, ce qui permet de calculer la valeur optimale U^* qui maximise la performance. Même si la procédure calculant la performance $p(U)$ génère pour une simulation physique par éléments finis un gros maillage temporaire avec $N \gg n$ sommets 3D, par exemple $N = 10^5$, seuls n nombres duaux ou infinitésimaux sont nécessaires, et pas $n + 3N$.

Ainsi, après tout, il est toujours possible de calculer les valeurs précises (mais pas les expressions symboliques) des dérivées et des vecteurs gradients lorsque des contraintes sont représentées avec des procédures.

3.3.2. Le problème de l'étude qualitative

Quand les contraintes sont représentées par des procédures, est-il encore possible d'utiliser des outils existants pour l'étude qualitative des systèmes de contraintes ? La réponse est positive : §4.4 explique comment construire le graphe biparti. §4.5 explique comment utiliser la méthode du témoin.

4. Implémentation

4.1. La nouvelle architecture

Le modeleur appelle le solveur. Le solveur peut appeler toutes les procédures, soit les procédures courantes et simples (mais indispensables) comme $\min, \max, |.|, \cos, \arctan$, etc, et aussi toutes les procédures plus sophistiquées fournies par le modeleur (ou par les modeleurs) : procédures de création et d'interrogation de formes, ou d'objets non géométriques. Symétriquement, les procédures du modeleur peuvent appeler le solveur pour les sous-problèmes.

Bien sûr, les procédures doivent être correctes et cohérentes. Par exemple, les arguments d'une procédure de génération doivent être indépendants.

Un problème technique est que les procédures renvoient souvent des points ou des vecteurs dans $\mathbb{R}^n$. Par exemple la procédure calculant un point sur une courbe de Bézier $B(P, d, t)$ (P est le vecteur des points de contrôle, d le degré, t le paramètre) renvoie le point 3D, et son trièdre de Frénet. Il est pratique de définir des procédures pour accéder à chacun des champs, à chacune des coordonnées (getX, getY, getZ [GFM$^+$16a]), mais bien sûr, il ne faut pas re-évaluer la procédure B plusieurs fois avec les mêmes arguments. Une gestion prend soin de cela [GFM$^+$16a]. Soit une mémorisation (appelée mémoïsation en programmation fonctionnelle) est utilisée, soit le solveur appelle une fonction pour permettre au modeleur de mettre à jour le modèle géométrique avant que le solveur appelle les procédures d'interrogation (telles que getX, getY, getZ) du modeleur.

Un problème technique [GFM$^+$16a] est que certains arguments des procédures ont des bornes, par exemple $t \in [0, 1]$ pour une courbe de Bézier $C(t)$. De nombreuses solutions sont possibles quand t est en dehors de son intervalle. La procédure qui calcule le point $C(t)$ peut le calculer quand t est en dehors de $[0, 1]$: en effet, il existe et est bien défini. La procédure peut également ramener la valeur de t dans $[0, 1]$ (mais sans modifier la variable t du solveur). Une autre solution est que le solveur gère les contraintes de bornes sur les variables. Alors, le modeleur doit fournir ces bornes au solveur. Soit le solveur contraint t seulement, soit il contraint tout le vecteur ΔX pour que sa coordonnée t reste dans $[0, 1]$. En combinaison avec toutes ces méthodes, il est possible d'utiliser également la contrainte procédurale : $(t - 1/2)^2 + t_s^2 - 1/4 = 0$ (ce qui implique que $t \in [0, 1]$) où t_s est une nouvelle variable, dite d'écart, associée à t. Nous n'avons pas essayé la méthode des pénalités. Byrd et al [BLNZ94] ont proposé une méthode plus sophistiquée pour gérer les contraintes de bornes, en fait pour résoudre l'optimisation contrainte avec une variante de lm-BFGS.

4.2. Interface entre modeleur et solveur

L'interface, ou le protocole de communication, entre le modeleur et le solveur doit permettre de tirer parti

du caractère creux des systèmes de contraintes procédurales. Il y a principalement deux types d'interfaces. La première interface est utilisée dans le projet DECO [GFM+16a] : le modeleur et le solveur utilisent des DAG noirs (Directed Acyclic Graphs). Cette première interface permet d'exploiter la faible densité des contraintes procédurales. Cependant, de nombreuses bibliothèques telles que GNU GSL ou Scipy, qui fournissent des solveurs ou des minimiseurs, utilisent le second type d'interface, comme suit. Pour résoudre un système $F(U, X) = 0$ avec $U = U_T$, le solveur reçoit généralement trois tableaux F, U, X : F est un tableau de pointeurs sur les procédures calculant $F(U, X)$ (les équations sont : $F(U, X) = 0$), U est le tableau de valeurs à virgule flottante pour les paramètres, X est le tableau des valeurs initiales pour les inconnues X. Le solveur peut modifier X, mais pas U. Parfois, le solveur accepte un tableau (de pointeurs sur les procédures) F' pour calculer les vecteurs gradients ∇F_i. Il peut également utiliser des différences finies pour approcher les dérivées. Le solveur reçoit également des informations techniques telles que les tailles de matrice, les valeurs de seuil pour les tests de terminaison, un nombre maximal d'itérations, etc. Le même type d'interface s'applique aux problèmes de minimisation contraints ou non contraints. Enfin, le solveur reçoit des méthodes poignées (callback), par exemple pour dessiner des images de la figure courante dans notre contexte ; ceci permet aux utilisateurs de surveiller et de piloter le processus de résolution. Clairement, cette deuxième interface n'est pas suffisante pour exploiter la faible densité des systèmes. Un autre tableau D est nécessaire pour spécifier les dépendances, c'est-à-dire le graphe biparti : $D[c]$ est la liste de tous les (indices de) variables X_k sur lesquels porte la contrainte F_c. D peut être vu comme une matrice creuse, et sa transposée T peut être utilisé à la place : $T[k]$ est la liste des (indices des) contraintes portant sur X_k. Ces deux tableaux sont clairement équivalents. Ils sont suffisants pour définir le graphe biparti et exploiter le caractère creux du système et de son Jacobien. Ils sont déjà utilisés à cette fin dans l'interface de nombreuses bibliothèques d'algèbre linéaire creuse [Saa03], par exemple pour calculer l'ordre des inconnues qui réduit le remplissage ("fill-in") de la matrice. Ces tableaux représentent le graphe biparti équations-inconnues utilisé dans la théorie du couplage (décomposition de Dulmage-Mendelsohn).

4.3. Conditions mathématiques

Comme d'habitude, les fonctions calculées par les procédures doivent être lisses (comme une fonction de distance) ou lisses presque partout (comme la fonction du point la plus proche, c'est-à-dire la projection orthogonale d'un point donné sur une forme donnée).

Cette condition mathématique peut parfois être assouplie. Par exemple Gouaty et al [GFM+16a] calcule les roues dentées contraintes : le nombre de dents doit être un nombre entier. De même pour le nombre de marches d'un escalier. En fait, les roues dentées, les escaliers, les structures en acier, les charpentes en bois ou en acier, etc. sont des formes algorithmiques : leurs méthodes de génération sont normalisées dans des documents techniques unifiés (DTU) et peuvent être directement mis en œuvre dans des procédures.

Comme d'habitude, l'esquisse doit être assez proche de la solution attendue.

Le modeleur doit fournir le graphe biparti au solveur, pour exploiter l'éparpillement du système et pour l'étude qualitative du système. Il est toujours possible d'utiliser la méthode du témoin (4.5).

4.4. Construire le graphe biparti

Nous expliquons ici comment le graphe biparti, qui liait équations et inconnues dans le cas classique, peut être construit. Considérons une procédure qui calcule une fonction $P : X \in \mathbb{R}^n \to Y = P(X) \in \mathbb{R}^m$. $Y = P(X)$ équivaut structurellement ou combinatoirement aux m équations : $e_i : Y_i - P_i(X) = 0, i = 1, \dots m$. Comme d'habitude, il y a un sommet pour chaque e_i, pour chaque $Y_i, i = 1, \dots m$, et pour chaque $X_k, k = 1, \dots n$. Une arête relie e_i à Y_i, et n arêtes lient e_i à $X_1, \dots X_n$. Ensuite, les méthodes structurelles s'appliquent. Elles détecteront, par exemple, la sur-contrainte structurelle de l'intersection de trois courbes paramétriques 2D (ou de quatre surfaces paramétriques en 3D). Leur limitation elles ne détectent que les conflits structurels) a déjà été mentionnée.

4.5. Interface pour la méthode du témoin

Pour que la méthode témoin s'applique, le modeleur (ou toute procédure appelant le solveur) doit fournir un témoin : soit une version antérieure du produit en cours de conception, soit un témoin calculé [KMF14]. Ce sont les utilisateurs qui décident si le témoin est typique de la solution attendue. De même, pour permettre à la méthode du témoin de détecter des sous-systèmes rigides, les variables doivent être étiquetées pour que la méthode du témoin puisse calculer a priori une base des mouvements rigides infinitésimaux (les nombres duaux, ou infinitésimaux, apparaissent encore) ([Fis17] pour les nombres duaux, [MF09] pour la méthode du témoin). Les balises donnent le type de coordonnées de la variable étiquetée quand elle est une coordonnée ; rappelons que les valeurs des coordonnées dépendent du repère cartésien utilisé. Une balise, *i.e.,* une coordonnée peut être :

- le x, le y ou le z d'un point. Les x, y, z du même

point doivent être liés d'une manière ou d'une autre, *e.g.,* la variable X_3 est le x du point (X_3, X_4, X_5).

- le x, le y ou le z d'un vecteur. En effet, les vecteurs et les points sont différents car ils ne se comportent pas de la même manière avec les translations. Les x, y, z du même vecteur doivent être liés d'une manière ou d'une autre.

- le a ou b ou c d'un vecteur normal. Vecteurs et vecteurs normaux sont différents : un vecteur est la différence entre deux points. Un vecteur normal est associé à une forme linéaire : $(x, y, z)^t \rightarrow (a, b, c)(x, y, z)^t$. En fait, a, b, c est la partie vectorielle de l'équation d'un plan (voir ci-dessous). Les a, b, c du même vecteur normal doivent être liés entre eux.

- le a ou le b ou le c ou le d de l'équation d'un plan : $ax + par + cz + d = 0$. Les a, b, c, d d'un même plan doivent être reliés entre eux.

- une variable géométrique indépendante du repère cartésien utilisé : rayon ou longueur, aire, volume, produit scalaire.

- enfin une variable peut être une variable non géométrique, donc indépendante du repère cartésien : énergie, force, coût, etc.

Les deux dernières balises peuvent être fusionnées. Les balises permettent à la méthode du témoin de décomposer les systèmes en sous-systèmes rigides (*i.e.,* bien-contraints modulo les isométries). Il est également possible de marquer les variables pour décomposer modulo les similitudes mais cette décomposition est plus rarement utilisée [SS06, SM06, MF09].

4.6. Nombres duaux

Cette section présente l'arithmétique des nombres duaux et montre comment ils peuvent être utilisés pour calculer les valeurs des dérivées avec la précision de l'arithmétique en virgule flottante, même lorsque les équations ne sont pas disponibles. Une telle précision est nécessaire par la méthode des témoins.

Les nombres duaux sont mieux compris avec une analogie avec des nombres complexes ($\mathcal{C}$). Pour un informaticien, écrire une bibliothèque C ++ pour une arithmétique pour les nombres duaux et pour les nombres complexes se ressemblent. Un nombre complexe z est une paire de deux nombres réels ($x \in \mathbb{R}, y \in \mathbb{R}$). La paire $(0, 1)$ s'appelle i. La partie x de z est sa partie réelle, et y sa partie imaginaire. L'addition de deux nombres complexes $z = (x, y)$ et $z' = (x', y')$ est définie par

$$(x, y) + (x', y') = (x + x', y + y')$$

Il est donc cohérent d'écrire la paire $z = (x, y)$ comme $(x, 0) + (0, y) = x(1, 0) + y(0, 1) = x1 + yi$. Le produit de z et z' est défini par

$$(x, y) \times (x', y') = (xx' - yy', xy' + yx')$$

donc $i^2 = (0, 1) \times (0, 1) = (-1, 0) = -1$. En fait, réduire i^2 à -1 donne une autre preuve de la règle de produit :

$$\begin{aligned}(x + yi) \times (x' + y'i) &= xx' + yy'i^2 + (xy' + yx')i \\ &= (xx' - yy') + (xy' + yx')i\end{aligned}$$

Il y a un isomorphisme remarquable $\phi_{\mathcal{C}}$ entre $z \in \mathcal{C}$ et la matrice réelle 2×2

$$\phi_{\mathcal{C}}(z) = \begin{pmatrix} x & -y \\ y & x \end{pmatrix}$$

En effet, $\phi_{\mathcal{C}}(z + z') = \phi_{\mathcal{C}}(z) + \phi_{\mathcal{C}}(z')$ et $\phi_{\mathcal{C}}(z \times z') = \phi_{\mathcal{C}}(z) \times \phi_{\mathcal{C}}(z')$.

Un nombre dual ressemble à un nombre complexe. C'est une paire de deux nombres réels ($x \in \mathbb{R}, y \in \mathbb{R}$). La paire $(0, 1)$ est appelée ϵ et peut être considérée comme un nombre infinitésimal. x est la partie standart de la paire (x, y) et y sa partie infinitésimale ou non standard. L'addition de deux nombres duaux (x, y) et (x', y') est définie par

$$(x, y) + (x', y') = (x + x', y + y')$$

Il est donc cohérent d'écrire la paire $z = (x, y)$ comme $(x, 0) + (0, y) = x(1, 0) + y(0, 1) = x1 + y\epsilon$. Le produit de $z = x + y\epsilon$ et $z' = x' + y'\epsilon$ est défini par

$$(x, y) \times (x', y') = (xx', xy' + yx')$$

donc $\epsilon^2 = (0, 1) \times (0, 1) = (0, 0)$. En fait, réduire ϵ^2 à 0 donne une autre preuve de la règle du produit :

$$\begin{aligned}(x + y\epsilon) \times (x' + y'\epsilon) &= xx' + (xy' + yx')\epsilon + yy'\epsilon^2 \\ &= xx' + (xy' + yx')\epsilon\end{aligned}$$

Ce coup-ci, l'isomorphisme ϕ entre le nombre dual $z = x + y\epsilon$ et la matrice réelle, de taille 2×2, $\phi(x + y\epsilon)$ est défini par :

$$\phi(z) = \begin{pmatrix} x & 0 \\ y & x \end{pmatrix}$$

En effet, $\phi(z + z') = \phi(z) + \phi(z')$, $\phi(z \times z') = \phi(z) \times \phi(z')$, donc $\phi(1/z) = \phi(z)^{-1}$ et $\phi(z^k) = \phi(z)^k$. Détaillons le produit :

$$\begin{array}{ccccc} (a + b\,\epsilon) & \times & (a' + b'\,\epsilon) & = & aa' + (ab' + ba')\,\epsilon \\ \downarrow & & \downarrow & & \\ \begin{pmatrix} a & 0 \\ b & a \end{pmatrix} & \times & \begin{pmatrix} a' & 0 \\ b' & a' \end{pmatrix} & = & \begin{pmatrix} aa' & 0 \\ ba' + ab' & aa' \end{pmatrix} \end{array}$$

Cet isomorphisme implique que :

$$\frac{1}{a + b\,\epsilon} = \frac{1}{a} - \frac{b}{a^2}\,\epsilon \text{ quand } a \neq 0$$

donc $b\epsilon$ n'a pas d'inverse (la matrice associée est non inversible). Cette règle est un cas particulier de :

$$(a + b\epsilon)^k = a^k + ka^{k-1}b\,\epsilon$$

Si P est un polynôme, alors $P(x_v + \epsilon)$ où x_v est un nombre flottant, donne $P(x_v)$ et la dérivée $P'(x_v)$:

$$\begin{aligned} P(x_v + \epsilon) &= a(x_v + \epsilon)^3 + b(x_v + \epsilon)^2 + c(x_v + \epsilon) + d \\ &= a(x_v^3 + 3x_v^2\,\epsilon) + b(x_v^2 + 2x_v\,\epsilon) + c(x_v + \epsilon) + d \\ &= (ax_v^3 + bx_v^2 + cx_v + d) + (3ax_v^2 + 2bx_v + c)\,\epsilon \\ &= P(x_v) + P'(x_v)\,\epsilon \end{aligned}$$

Ceci se généralise aux polynömes multivariés : soit il y a une seul ϵ et deux évaluations sont nécessaires :

$$\begin{aligned} Q(x_v + \epsilon, y_v) &= Q(x_v, y_v) + Q'_x(x_v, y_v)\epsilon \\ Q(x_v, y_v + \epsilon) &= Q(x_v, y_v) + Q'_y(x_v, y_v)\epsilon \end{aligned}$$

ou bien chaque variable a son propre ϵ et une seule évaluation suffit :

$$\begin{aligned} Q(x_v + \epsilon_x, y_v + \epsilon_y) \;=\; & Q(x_v, y_v) + Q'_x(x_v, y_v)\epsilon_x + \\ & Q'_y(x_v, y_v)\epsilon_y \end{aligned}$$

En fait, cette fonctionnalité (calculer $(P(x), P'(x))$ ensemble) s'étend aux fonctions non polynomiales. L'arithmétique des nombres duaux est utilisée pour calculer $f(t)$ et sa dérivée $f'(t)$ en même temps : ici f est une fonction continue et dérivable (au moins en la valeur t), et $f'(t)$ est la valeur de la dérivée de f en $t \in \mathbb{R}$. L'idée est de représenter la paire $(f(t), f'(t))$ avec un nombre dual $f(t) + f'(t)\epsilon$. C'est possible parce que l'arithmétique des nombres duaux imite les règles pour les dérivées des sommes et des produits (attention : f', g' désignent ici les dérivées de f et g) :

$$\begin{aligned} (f, f') + (g, g') &= (f + g, f' + g') \\ (f, f') \times (g, g') &= (f \times g, f \times g' + f' \times g) \end{aligned}$$

Une arithmétique pour les nombres complexes doit définir $\exp, \cos$, etc pour les nombres complexes. De même pour les nombres duaux. Cette définition est simple : pour tout f, $f(a + b\epsilon)$ est égal par définition à $f(a) + b\epsilon f'(a)$, ainsi :

$$\begin{aligned} \exp(a + b\epsilon) &= e^a + be^a\,\epsilon \\ \cos(a + b\epsilon) &= \cos(a) - b\sin(a)\,\epsilon \\ \sin(a + b\epsilon) &= \sin(a) + b\cos(a)\,\epsilon \\ \tan(a + b\epsilon) &= \tan(a) + b(1 + \tan^2(a))\,\epsilon \end{aligned}$$

La définition de la fonction $\mathrm{sgn}(a + b\epsilon)$ est facile :

$$\mathrm{sign}(a + b\epsilon) = \mathrm{sign}(a) + (1 - \mathrm{sign}(a)^2)\mathrm{sign}(b)$$

Alors la définition de la valeur absolue s'ensuit :

$$|a + b\epsilon| = \mathrm{sign}(a + b\epsilon) \times (a + b\epsilon)$$

Les définitions du min et max de deux nombres duaux peuvent être formulées de la même manière. En passant, la dérivation symbolique (au moment de la compilation) ne peut pas traiter si simplement les fonctions $|.|, \min, \max$ et le branchement conditionnel if-then-else.

r	50	100	150	200	300	400
c	1275	5050	11325	20100	45150	80200
u	3825	15150	33975	60300	135450	240600

Table 1: *r, c, u sont les nombres de rangées, de cercles, et d'inconnues.*

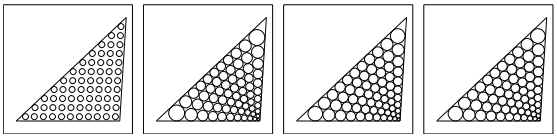

Figure 2: *Résolution avec Newton (11 rangées). Trois itérations suffisent.*

4.7. Résoudre sans équations

Cette section montre qu'il est possible de résoudre efficacement des contraintes procédurales, même en très grand nombre.

Le solveur doit résoudre sans équations les systèmes sous-jacents $E(X) = 0$, à partir d'une estimation initiale assez proche de la racine attendue. Cette proximité était déjà nécessaire avec les contraintes géométriques classiques, à savoir quand les équations sont disponibles. En pratique, les utilisateurs fournissent cette estimation initiale avec une esquisse.

Nous avons mis en œuvre et testé plusieurs méthodes de résolution : Newton, Levenberg-Marquardt [GCG99b], lm-BFGS (Broyden-Fletcher-Goldfarb-Shanno avec mémoire limitée) [BGLS06], Hooke-Jeeves, descente stochastique et l'heuristique Jaya [Rao16]. Certaines méthodes ont besoin des vecteurs

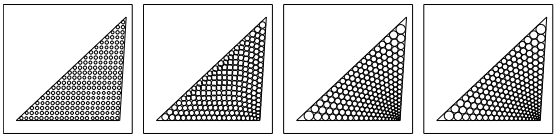

Figure 3: *Quelques étapes de résolution avec Hooke-Jeeves (20 rangées).*

Algorithme	Complexité
lm-BFGS ($m = 10$)	$O(n^{1.33})$
Newton	$O(n^{1.4})$
Levenberg-Marquardt	$O(n^{1.4})$
Hooke-Jeeves	$O(n^{1.9})$

Table 2: *n est le nombre de cercles ou d'inconnues et de contraintes (et non le nombre de rangées). Cette table donne la complexité empirique (la pente du diagramme log-log) des algorithmes pour le problème de test, en exploitant la faible densité des systèmes.*

gradients. Le calcul par différence finie centrée est suffisamment précis avec nos problèmes de test. Les méthodes ont été testées sur ce problème géométrique : soit ABC un triangle donné ; soit r un entier ; placer r rangées de cercles dans ABC, avec un cercle dans la première rangée, k dans la rangée k, de sorte que les cercles soient tangents à leurs voisins ou aux côtés de ABC. Voir les figures 2 et 3. Pour r rangées, il y a $r(r+1)/2$ cercles et trois fois plus d'inconnues : $n = 3r(r+1)/2$ car chaque cercle k a des coordonnées (x_k, y_k) inconnues pour son centre et un rayon inconnu r_k. C'est un problème bien contraint puisque le nombre de contraintes est égal au nombre d'inconnues $n = 3r(r+1)/2$. Il est également irréductible : il n'a pas de sous-système bien contraint (ni rigide). Bien que les équations (tangences cercle-cercle, ou tangences cercle-cercle) soient disponibles, elles ne sont pas accessibles par le solveur : le solveur connaît seulement un tableau de n (pointeurs sur des) procédures $E[0, \dots n-1]$, et un tableau de n inconnues V qui sont la concaténation des tuples (x_k, y_k, r_k) pour $k = 0, \dots n-1$. Les valeurs initiales pour (x_k, y_k, r_k) sont obtenues ainsi : ce problème est facile à résoudre lorsque le triangle ABC est équilatéral, ce qui donne les coordonnées barycentriques a_k, b_k, c_k (avec $a_k + b_k + c_k = 1$) pour le centre du cercle k : $(x_k, y_k) = a_k A + b_k B + c_k C$. Ceci donne une valeur initiale pour X.

Ce problème est artificiel mais très pratique pour mesurer les performances et contrôler le comportement et le résultat des algorithmes, en raison de sa nature visuelle et intuitive. La figure 2 montre les itérations de Newton pour 11 rangées, et la figure 3 quelques itérations par Hooke-Jeeves. Table 1 donne les chiffres avec plus de rangées, et table 2 les complexités empiriques des algorithmes testés. La complexité empirique pour un algorithme est la pente de la courbe (très proche d'une droite) du diagramme log-log $(\log n_i, \log T(n_i))$, où n_i est le nombre d'inconnues, et $T(n_i)$ le temps d'exécution de l'algorithme. Nous avons testé avec 50, 100, 150, 200 lignes. lm-BFGS a la meilleure complexité empirique : $O(n^{1.33})$. Cette méthode peut donc traiter de grands problèmes ("scalability"). Ensuite, Newton et Levenberg-Marquardt ont une complexité $O(n^{1.4})$. Hooke-Jeeves a une complexité $O(n^{1.9})$. Toutes ces complexités sont inférieures à $O(n^2)$, la taille d'un Jacobien dense. Pour les atteindre, il est essentiel d'exploiter la faible densité des systèmes de contraintes procédurales. Rappelons que, pour multiplier deux matrices denses, ou résoudre un système linéaire dense, la complexité de la fameuse méthode de Strassen est $O(n^{2.807})$, et la complexité de l'algorithme de Coppersmith-Winograd (la meilleure méthode connue à ce jour, mais inutilisée en pratique) est $O(n^{2.376})$. Même la méthode de Hooke-Jeeves est bien meilleure, et inférieure à $O(n^2)$. Les autres méthodes testées : gradient stochastique, et Jaya soit divergent, soit convergent avec un amortissement mais sont alors trop lentes, donc nous ne commentons pas leur complexité.

4.8. A propos des méta-heuristiques

Dans nos expériences, les heuristiques telles que Jaya et la descente stochastique n'ont pas bien fonctionné avec le problème de test. Cependant, se tourner vers des méta-heuristiques pourrait être intéressant quand il existe un grand nombre de solutions pour lesquelles les utilisateurs n'ont pas de préférence a priori, mais qu'ils peuvent rejeter une mauvaise solution même s'il leur est difficile de dire pourquoi ; ou bien s'il est trop long d'expliciter toutes les contraintes et préférences. Un problème similaire est rencontré en police scientifique : certains témoins ne savent pas décrire des visages ou des personnes, bien qu'ils puissent les reconnaître. Pour résoudre ce problème [SGM09], un logiciel (tel EvoFIT) génère un ensemble de 20 images de visages aléatoires ; le témoin choisit les visages qui ressemblent le plus au visage de la cible. Le logiciel combine des parties de visages sélectionnés, et propose une nouvelle série de 20 visages au témoin. Après quelques itérations, un portrait ressemblant est obtenu. On peut imaginer utiliser la même méthode pour choisir une solution parmi un ensemble énorme de solutions. Le modeleur génère 20 solutions, par exemple à partir de 20 graines aléatoires, améliorées avec des itérations de Newton ou certaines minimisations. Les utilisateurs rejettent les trop mauvaises solutions ; peut-être peuvent-ils dire quelle partie est bonne dans une solution et devrait être conservée. Le modeleur combine des solutions sélectionnées, comme dans les algorithmes génétiques, et améliore chaque nouvelle combinaison pour satisfaire les contraintes procédurales, avec des itérations de Newton ou un minimiseur. On peut espérer aboutir à une bonne solution après quelques étapes.

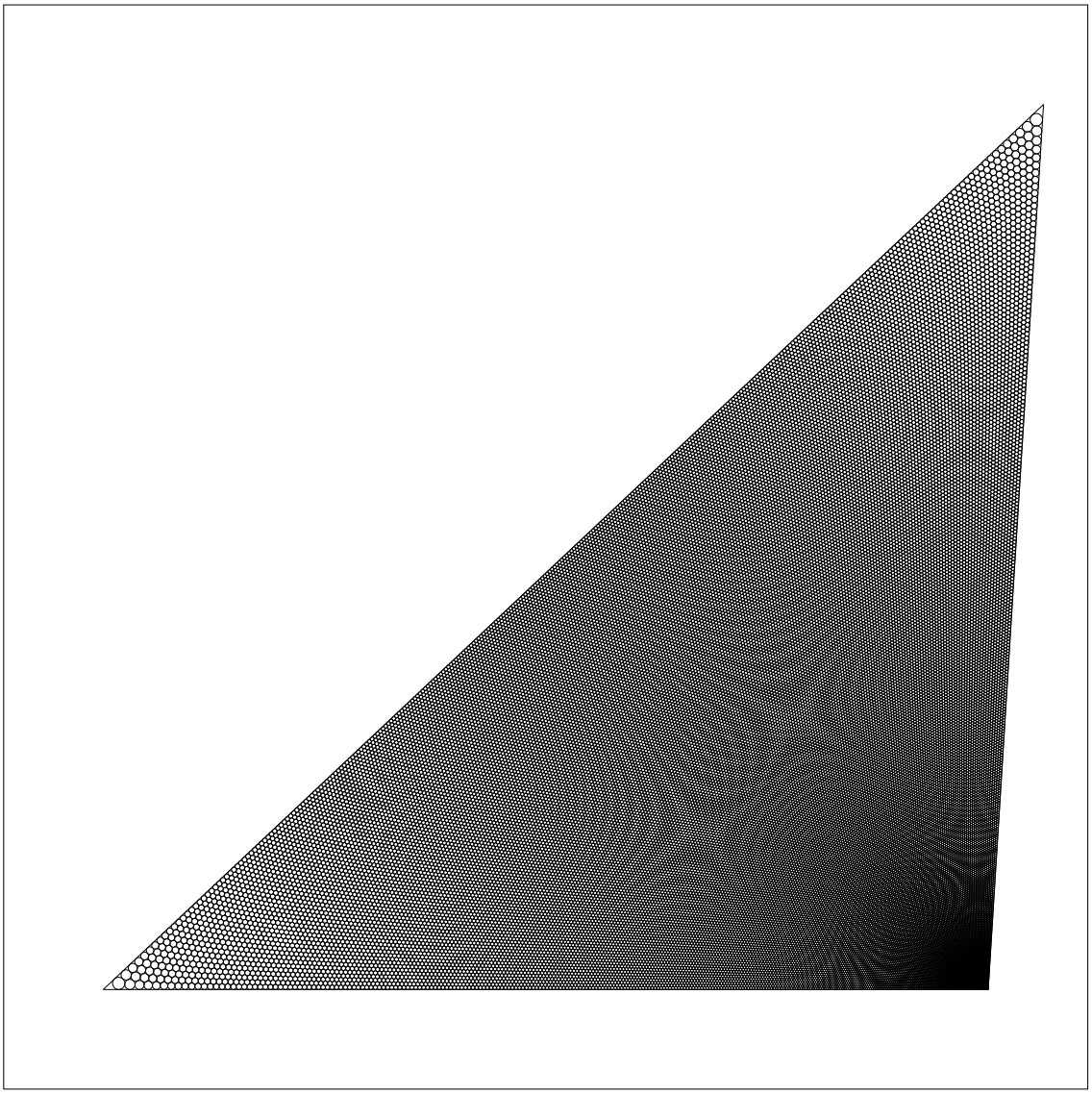

Figure 4: *Le problème de test.*

5. Conclusion

Cet article propose de représenter les contraintes avec des procédures, et il présente les avantages, les quelques inconvénients, et une implémentation qui prouve la faisabilité de notre approche, et sa "scalabilité" : les méthodes de résolution, ou d'étude qualitative, fonctionnent en temps presque linéaire et ce sans même exploiter leur parallélisme.

Par concision, nous n'avons pas pu détailler quelques travaux précurseurs [GFM$^+$16b], [DPD15], [PFGL08a], [GMMP11], [FGLP14], [PFGL08b], [LGP$^+$16].

Nous concluons avec les travaux futurs.

Une formalisation plus précise est nécessaire, pour expliciter les liens avec le PLM, les ontologies, l'analyse de forme, l'inter-opérabilité, etc.

Une extension directe de notre travail consiste à gérer une représentation hybride pour les modèles géométriques : fondamentalement un tuple (X^T, U^T, F) ou en mots, une géométrie et un ensemble de contraintes. La géométrie X^T (T like target) est une

solution de $F(U, X) = 0, U - U^T = 0$, F est l'ensemble des contraintes, représentées avec des DAG ou la source d'un programme équivalent dans un langage de script (Lua, Python, Lisp), U est le vecteur des paramètres, U_T est le vecteur des valeurs des paramètres, X est le vecteur des variables inconnues. X^T, U^T est son propre témoin. Il y a plusieurs façons de modifier et de mettre à jour le modèle.

La première façon est de modifier les valeurs de U : $U = U^*$ au lieu de $U = U^T$, et X doit être mis à jour avec quelques résolutions. Une possibilité est de suivre une courbe d'homotopie d'équation : $H(t, X) = (1 - t)F(U^T, X) + tF(U^*, X) = 0$ depuis de $t = 0, X = X^T$ jusqu'à (espérons-le) $t = 1$, $X = X^*$, ou pour réduire $||H(t, X)||$ depuis $t = 0, X = X^T$ jusqu'à (espérons-le) $t = 1$, $X = X^*$.

La deuxième voie est de spécifier de nouveaux éléments géométriques et de nouvelles contraintes : cette voie est bien connue, car il est standard d'ajouter de manière interactive et incrémentale des éléments géométriques et des contraintes à l'esquisse, de détecter et corriger les erreurs, et de résoudre.

Le troisième et dernier moyen de modifier le modèle consiste à interactivement modifier la géométrie X^T. Parfois, c'est beaucoup plus facile pour les utilisateurs de déplacer interactivement les sommets ou les faces (ou leurs plans tangents) que de changer les contraintes. La mise à jour des valeurs U ne suffit pas toujours, et les contraintes F doivent être mises à jour : c'est clairement la partie la plus difficile.

Références

[BGLS06] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization : theoretical and practical aspects*. Springer Science & Business Media, 2006.

[Ble] Blender.org. Blender for Open source 3D Creation.

[BLNZ94] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, 16 :1190–1208, 1994.

[DH00] Cassiano Durand and Christoph M. Hoffmann. A systematic framework for solving geometric constraints analytically. *Journal of Symbolic Computation*, 30(5) :493 – 519, 2000.

[DPD15] Dorian Decriteau, Jean-Philippe Pernot, and Marc Daniel. Towards declarative CAD modeler built on top of a CAD modeler. *Proceedings of CAD'15, Computer Aided Design and Applications*, pages 107–112. , jun 2015.

[EK01] Gershon Elber and Myung-Soo Kim. Geometric constraint solver using multivariate rational spline functions. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 1–10, New York, NY, USA, 2001. ACM.

[FGLP14] Bianca Falcidieno, Franca Giannini, Jean-Claude Léon, and Jean-Philippe Pernot. Processing free form objects within a product development process framework. In Jonh G. J. G. Michopoulos, Christiaan J.J. Paredis, David W. Rosen, and Judy M. Vance, editors, *Advances in Computers and Information in Engineering Research*, volume 1, pages 317–344. ASME-Press, 2014.

[Fis17] Ian Fischer. *Dual-number methods in kinematics, statics and dynamics*. Routledge, 2017.

[FM12] Sebti Foufou and Dominique Michelucci. Bernstein basis and its application in solving geometric constraint systems. *Journal of Reliable Computing*, 17 :192–208, 2012.

[Frea] FreeCAD. FreeCAD : An open-source parametric 3D CAD modeler.

[Freb] FreeSHIP. FreeSHIP : Surface Modeling.

[FS08] Arnaud Fabre and Pascal Schreck. Combining symbolic and numerical solvers to simplify indecomposable systems solving. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1838–1842, New York, NY, USA, 2008. ACM.

[GCG99a] Jian-Xin Ge, Shang-Ching Chou, and Xiao-Shan Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31(14) :867–879, 1999.

[GCG99b] Jian-Xin Ge, Shang-Ching Chou, and Xiao-Shan Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31(14) :867 – 879, 1999.

[GFM+16a] Gilles Gouaty, Lincong Fang, Dominique Michelucci, Marc Daniel, Jean-Philippe Pernot, Romain Raffin, Sandrine Lanquetin, and Marc Neveu. Variational geometric modeling with black box constraints and DAGs. *Computer-Aided Design*, 75 :1–12, 2016.

[GFM+16b] Gilles Gouaty, Lincong Fang, Dominique Michelucci, Marc Daniel, Jean-Philippe Pernot, Romain Raffin, Sandrine Lanquetin, and Marc Neveu. Variational geometric modeling with black box constraints and DAGs. *Computer-Aided Design*, 75 :1–12, 2016.

[GHY04] Xiao-Shan Gao, Christoph M. Hoffmann, and Wei-Qiang Yang. Solving spatial basic geometric constraint configurations with locus intersection. *Computer-Aided Design*, 36(2) :111 – 122, 2004. Solid Modeling and Applications.

[GMMP11] F. Giannini, E. Montani, M. Monti, and J-P. Pernot. Semantic evaluation and deformation of curves based on aesthetic criteria. *Computer-Aided Design and Applications*, 8(3) :449–464, 2011.

[Hen92] Bruce Hendrickson. Conditions for unique graph realizations. *SIAM journal on computing*, 21(1) :65–84, 1992.

[HJA97] Christoph M Hoffmann and Robert Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23(2-3) :287–299, 1997.

[HJA05] Christoph M Hoffmann and Robert Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5) :655–663, 2005.

[HKP17] Hao Hu, Mathias Kleiner, and Jean-Philippe Pernot. Over-constraints detection and resolution in geometric equation systems. *Computer-Aided Design*, 90 :84 – 94, 2017. SI :SPM2017.

[IMS11] Remi Imbach, Pascal Mathis, and Pascal Schreck. Tracking method for reparametrized geometrical constraint systems. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2011 13th International Symposium on*, pages 31–38. IEEE, 2011.

[IMS15] Rémi Imbach, Pascal Mathis, and Pascal Schreck. A robust and efficient method for solving geometrical constraint problems by homotopy. *CoRR*, abs/1503.07901, 2015.

[ISM14] Rémi Imbach, Pascal Schreck, and Pascal Mathis. Leading a continuation method by geometry for solving geometric constraints. *Computer-Aided Design*, 46 :138–147, 2014.

[JASR99] Robert Joan-Arinyo and Antoni Soto-Riera. Combining constructive and equational geometric constraint-solving techniques. *ACM Transactions on Graphics (TOG)*, 18(1) :35–55, 1999.

[Jau01] Luc Jaulin. *Applied interval analysis : with examples in parameter and state estimation, robust control and robotics*, volume 1. Springer Science & Business Media, 2001.

[JTNM06] Christophe Jermann, Gilles Trombettoni, Bertrand Neveu, and Pascal Mathis. Decomposition of geometric constraint systems : a survey. *International Journal of Computational Geometry & Applications*, 16(05n06) :379–414, 2006.

[Kho12] Mostafa Khorramizadeh. An application of the Dulmage-Mendelsohn decomposition to sparse null space bases of full row rank matrices. In *International Mathematical Forum*, volume 7, pages 2549–2554, 2012.

[KMF14] Arnaud Kubicki, Dominique Michelucci, and Sebti Foufou. Witness computation for solving geometric constraint systems. In *Science and Information Conference (SAI), 2014*, pages 759–770. IEEE, 2014.

[Kon92] K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design*, 24(3) :141–147, 1992.

[LGP+16] Z. Li, F. Giannini, J-P. Pernot, P. Véron, and B. Falcidieno. Re-using heterogeneous data for the conceptual design of shapes in virtual environments. *Virtual Reality*, pages 1–18, 2016.

[LP09] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[MF09] Dominique Michelucci and Sebti Foufou. Interrogating witnesses for geometric constraint solving. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 343–348. ACM, 2009.

[MMS14] Mireille Moinet, Guillaume Mandil, and Philippe Serre. Defining tools to address over-constrained geometric problems in computer aided design. *Computer-Aided Design*, 48 :42–52, 2014.

[Owe91] John C Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 397–407. ACM, 1991.

[PFGL08a] J-P. Pernot, B. Falcidieno, F. Giannini, and J-C. Léon. A hybrid models deformation tool for free-form shapes manipulation. In *34th Design Automation Conference (ASME DETC08-DAC49524)*, pages 647–657, New-York, USA, 2008. ASME.

[PFGL08b] Jean-Philippe Pernot, Bianca Falcidieno, Franca Giannini, and Jean-Claude Léon. Incorporating free-form features in aesthetic and engineering product design : State-of-the-art report. *Computers in Industry*, 59(6) :626–637, 2008.

[Rao16] R Rao. Jaya : A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 7(1) :19–34, 2016.

[RSV89] Dieter Roller, François Schonek, and Anne Verroust. Dimension-driven geometry in cad : a survey. In *Theory and practice of geometric modeling*, pages 509–523. Springer, 1989.

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[Ser91] David Serrano. Automatic dimensioning in design for manufacturing. In *Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, SMA '91, pages 379–386, New York, NY, USA, 1991. ACM.

[SGM09] Christopher Solomon, Stuart Gibson, and Matthew Maylin. A new computational methodology for the construction of forensic, facial composites. *Computational forensics*, pages 67–77, 2009.

[SM06] Pascal Schreck and Pascal Mathis. Geometrical constraint system decomposition : a multi-group approach. *International Journal of Computational Geometry & Applications*, 16(05n06) :431–442, 2006.

[SS06] Pascal Schreck and Étienne Schramm. Using invariance under the similarity group to solve geometric constraint systems. *Computer-Aided Design*, 38(5) :475–484, 2006.

[SWI05] Andrew J Sommese and Charles W Wampler II. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific, 2005.

[TSM+11] Simon EB Thierry, Pascal Schreck, Dominique Michelucci, Christoph Fünfzig, and Jean-David Génevaux. Extensions of the witness method to characterize under-, over-and well-constrained geometric constraint systems. *Computer-Aided Design*, 43(10) :1234–1249, 2011.

[VSR92] Anne Verroust, François Schonek, and Dieter Roller. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24(10) :531–540, 1992.