N° d'ordre: 5276

Université Louis Pasteur - Strasbourg 1
Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection
UMR 7005 CNRS-ULP

THÈSE

présentée pour obtenir le titre de

Docteur en SCIENCES
Spécialité INFORMATIQUE

par

Arnaud FABRE

Contraintes géométriques en dimension 3

soutenue le **30 Novembre 2006**,
devant la commission d'examen composée de :

M. Pascal SCHRECK, Directeur de Thèse,
Professeur de l'Université Louis Pasteur à Strasbourg
M. Jerzy KORCZAK Rapporteur Interne,
Professeur de l'Université Louis Pasteur à Strasbourg
Mme Véronique GAILDRAT Rapporteur Externe,
Maître de Conférences, habilitée à diriger des recherches,
de l'Université Paul Sabatier à Toulouse
M. Dominique MICHELUCCI Rapporteur Externe,
Professeur de l'Université de Bourgogne à Dijon
Mr. Mahmoud MELKEMI Examinateur,
Professeur de l'Université de Haute Alsace à Mulhouse
M. Pascal MATHIS Invité,
Maître de conférence de l'Université Louis Pasteur à Strasbourg

Table des matières

Table des figures

Introduction

L'informatique graphique ne fait pas défaut à la règle commune qui veut que les utilisateurs demandent des outils toujours plus puissants pour s'adapter à leurs desiderata et pour faciliter le processus de modélisation d'objets numériques. Dans le cadre de la modélisation géométrique, l'utilisation de contraintes *doit permettre de décrire, puis d'engendrer automatiquement un objet à partir d'une spécification de ses propriétés topologiques et dimensionnelles. On parle alors de modélisation par contraintes et de résolution de systèmes de contraintes géométriques (SCG).*

La tradition « formaliste » de l'équipe IGG du LSIIT de Strasbourg a orienté l'étude de ce problème suivant une approche formelle de la géométrie et des constructions géométriques avec, notamment, une première expérience dans le domaine de l'enseignement assisté par ordinateur, puis son application dans le cadre du dessin et de la conception assistés par ordinateur. Cette approche conduit à une résolution exacte des systèmes de contraintes géométriques et les avantages qu'on peut en retirer sont, entre autres, la prise en compte des cas particuliers et la possibilité de donner des explications sur la manière de trouver la ou les solutions, ou les raisons pour lesquelles aucune solution n'a été trouvée.

Cependant, cette approche est difficilement applicable lorsqu'on envisage des contraintes géométriques en dimension 3 car on ne connaît pas de méthodes de résolution exactes en dehors d'une classe de problèmes très restreinte. Ce problème est rendu compliqué d'une part par l'explosion combinatoire du nombre de cas à traiter et par l'irréductibilité de certains problèmes, et d'autre part, pour des raisons d'ergonomie et d'interfaçage entre l'utilisateur et le logiciel.

Le travail présenté ici concerne l'application de méthodes géométriques formelles aux contraintes 3D et l'expérimentation de nouvelles méthodes d'interaction. La description d'univers géométriques est au cœur de notre approche aussi bien pour l'interfaçage que pour la résolution de contraintes.

Sommaire

Domaine

Depuis les travaux de Sutherland [Sut63], la prise en compte de contraintes géométriques aussi bien sur le plan de l'interface, que sur le plan de la résolution, reste un sujet d'actualité en modélisation géométrique.

Selon les communautés qui se sont intéressées à ce problème, les moyens d'y répondre ont été différents, mais tous concordent sur la difficulté de caractériser les problèmes de contraintes géométriques, *i.e.* savoir au préalable s'il existe une solution, et sur la difficulté de les résoudre.

Ces problèmes se rencontrent dans plusieurs domaines qui ont recours aux contraintes :

- la Conception Assistée par Ordinateur (CAO) au niveau de la conception des maquettes numériques virtuelles (des pièces mécaniques, des bâtiments, des systèmes électriques), *etc.* ;
- l'Enseignement Assisté par ordinateur (EAO) dans la preuve de théorèmes de géométrie, pour résoudre des problèmes de constructions géométriques, *etc.* ;
- la modélisation déclarative pour la création d'un environnement virtuel ;
- en robotique, pour trouver des configurations d'un robot ou d'un mécanisme (plate-forme de Stewart) ;
- dans la reconstruction de scènes acquises à partir de dessins à main levée ou de photos ;
- en biochimie, dans le cadre l'étude des molécules ;
- *etc.*

Trouver une solution à un système de contraintes est un problème étudié depuis les années 1970. Le problème de la satisfaction de contraintes (CSP : Constraint Satisfaction Problem) a donné lieu à ce qu'on appelle la programmation par contraintes qui rassemble un ensemble de techniques de résolutions très générales.

Cette communauté a formalisé la définition d'un système de contraintes par trois ensembles : les contraintes, les variables sur lesquelles portent ces contraintes, et les domaines d'expression de ces variables.

L'application directe de ces méthodes de résolution est malheureusement peu satisfaisante dans le cadre de la géométrie. En effet, le domaine d'expression des variables est, en pratique, celui des nombres réels, et l'existence de cas dégénérés et de théorèmes, impliquant des dépendances très fortes entre objets, nécessite d'adapter les méthodes issues de la programmation par contraintes [Doh95].

C'est en partie pour cette raison qu'une autre communauté s'est formée dans le domaine de la modélisation géométrique pour trouver des méthodes mieux adaptées à la résolution de contraintes géométriques. Dans ce contexte, les systèmes de contraintes sont généralement exprimés sous forme d'une esquisse côtée (*cf.* figure 1) traduite en un système d'équations non-linéaires, mais peuvent aussi être décrits par des fonctions d'optimisation dont la sémantique est cognitive, ou esthétique comme en modélisation déclarative [KGC97].

Ces deux communautés (CSP et modélisation géométrique) se sont intéressées au développement de *solveurs* géométriques qui donnent une ou plusieurs solutions (cf.

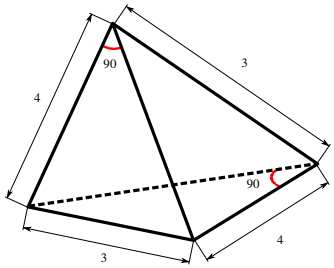


FIG. 1 – Esquisse cotée d'un tétrèdre avec la spécification de 4 contraintes de distances et deux contraintes d'angles

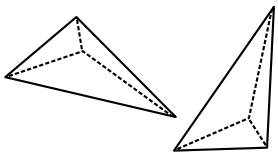


FIG. 2 – Deux solutions au problème de contraintes de la figure 1

figure 2) en positionnant et orientant automatiquement des entités géométriques.

Si l'on regroupe l'ensemble des qualités attendues d'un solveur, on obtient le cahier des charges d'un solveur idéal dont la description n'a pas beaucoup changé depuis [RSV88]. Celui-ci énonce les sept propriétés suivantes :

1. détection des systèmes non-consistant, (aucune solution) ;
2. résolution indépendamment de l'ordre de déclaration des contraintes ;
3. correction : toutes les solutions trouvées sont bien des solutions ;
4. complétude : on trouve toutes les solutions, ou toutes les solutions réelles ;
5. interactivité : la résolution est suffisamment rapide pour permettre une interaction ;
6. extensibilité à de nouveaux types de contraintes et d'entités ;
7. indépendance par rapport à la dimension (2D ou 3D).

La conclusion était déjà qu'aucun solveur ne cumulait toutes ces bonnes propriétés et qu'il faudrait probablement combiner toutes les techniques existantes afin d'obtenir un compromis convenable.

La grande majorité des travaux a porté sur les problèmes de géométrie plane, déjà très difficiles à résoudre par les méthodes classiques des CSP. Et depuis quelques années, beaucoup de travaux se focalisent sur le moyen de dominer la complexité en taille des systèmes de contraintes géométriques. L'application de la stratégie « diviser pour régner » a permis de résoudre de nouveaux problèmes et/ou d'accélérer la phase de résolution en proposant une planification de la résolution des sous-systèmes dont les solutions locales sont assemblées pour fournir la solution globale du problème.

Lorsqu'un problème accepte un nombre fini de solutions, on dit qu'il est bien-contraint. Dans le cas où il n'existe pas de solutions dans l'espace complexe, il est sur-contraint, et, à l'inverse, quand il y en a une infinité, il est sous-contraint.

Aujourd'hui, la plupart des solveurs considèrent communément la *constriction* modulo *les déplacements*, *i.e.* deux solutions sont équivalentes s'il existe un déplacement permettant de passer de l'une à l'autre.

Ainsi, pour passer d'un problème bien-contraint *modulo* les déplacements à un problème bien-contraint, il suffit de fixer un repère pour les déplacements. En 2D, par exemple, on peut fixer un point et une direction.

Mais compter *a priori* le nombre de solutions est particulièrement difficile en géométrie où les incohérences et les redondances sont parfois cachées et dévoilées par l'application de théorèmes de géométrie. Un exemple simple de dépendance est fourni par la relation de Chasles pour les angles d'un polygone.

Problématique

En 2D, un important travail a été effectué sur le sujet. Notamment, des caractérisations structurelles de la rigidité générique permettent de prévoir la constriction d'un système de contraintes.

Combinées aux méthodes de décomposition combinatoires et de résolution d'équations numériques, ces techniques permettent de résoudre beaucoup de problèmes de contraintes 2D couramment rencontrés en CAO.

Cependant, même dans le cadre général 2D, on ne connaît pas de caractérisation structurelle exacte de la géométrie, en raison des nombreuses dépendances géométriques à considérer, comme par exemple l'inégalité triangulaire (la somme des valeurs de deux côtés d'un triangle doit être supérieure à la valeur du troisième côté).

En 3D, les problèmes rencontrés sont beaucoup plus difficiles à résoudre en raison de l'explosion du nombre de cas à traiter. De plus, les extensions des caractéristiques combinatoires qui donnaient satisfaction en 2D, n'ont pas le même impact en 3D. Enfin, on rencontre plus facilement des problèmes irréductibles en 3D. La tendance actuelle est aux généralisations et aux extensions des méthodes 2D à la dimension supérieure et à la recherche de méthodes plus puissantes. Mais ces méthodes rencontrent des difficultés qu'il reste à surmonter pour répondre à cette question : quelles méthodes de résolutions utiliser pour des problèmes de contraintes 3D ?

Approche

D'un point de vue global, la modélisation par contraintes géométriques concerne la modélisation d'un univers géométrique, l'interaction Homme-Machine et la résolution. Ce point de vue, qui est celui défendu par la modélisation déclarative, implique de considérer l'utilisateur au centre de la modélisation par contraintes géométriques.

Cette démarche met en évidence les différentes influences de l'interaction dans la résolution. Et cette influence globale s'exprime directement par l'élément clé partagé

à la fois par l'utilisateur, l'interface et la résolution : *l'univers géométrique*.

Ainsi, notre démarche est de s'attaquer à la résolution de contraintes géométriques 3D de manière globale. D'abord, il nous faut décrire la manière dont sont exprimés les systèmes de contraintes géométriques grâce à l'univers géométrique considéré. Ensuite, nous devons décrire la définition de tels systèmes du point de vue de l'utilisateur et donc la manière de modéliser des objets en spécifiant des contraintes. Enfin, résoudre les systèmes obtenus en respectant au maximum les propriétés définies plus haut.

Formaliser

Une formalisation des notions d'univers géométriques et de systèmes de contraintes géométriques doit ainsi être effectuée dans un premier temps. Nous proposons de distinguer les techniques de résolution existantes en prenant appui sur leur univers géométrique dans le but de dégager des univers géométriques 3D particuliers pour guider la résolution.

Modéliser

Afin d'être à même de proposer une technique de résolution adaptée à un problème de modélisation par contraintes, nous pensons qu'il est important d'étudier la manière de modéliser.

En 3D, la plupart des logiciels de modélisation proposent à l'utilisateur un mode d'interaction basé sur des vues projectives, la sélection dans ces vues, et les déplacements suivant les repères locaux aux objets. Mais, à notre connaissance, aucun ne propose des outils permettant de poser et manipuler des contraintes géométriques dans un contexte tridimensionnel.

L'interaction traditionnelle ne nous semble pas assez intuitive et reste limitée par l'utilisation de périphériques 2D. Ainsi, nous proposons d'étudier interaction et méthode de résolution de manière conjointe en utilisant la Réalité Virtuelle (RV) pour visualiser et interagir.

Plus précisément, la représentation sur un écran 2D et la manipulation par clavier/souris est très lourde dans le cadre 3D. L'utilisation des technologies issues de la réalité virtuelle est une piste toute indiquée pour résoudre ce problème de l'interaction 3D, mais ceci pose en retour beaucoup d'autres problèmes :

- une visualisation stéréoscopique n'est pas suffisante pour permettre d'interagir de manière intuitive ;
- la manipulation à 6 degrés de libertés n'est pas forcément adéquate pour tous les objets et notamment les objets contraints ;
- la navigation reste un problème essentiel qui n'est pas résolu automatiquement par le passage à la RV.

Nous pensons résoudre ces problèmes par la création d'outils spécifiques au domaine des contraintes géométriques et l'adaptation de techniques issues de la RV. Nous expérimentons dans ce mémoire la possibilité d'utiliser ces techniques, notamment

les gestes, pour spécifier des contraintes géométriques en 3D.

Il suffit de considérer un domaine restreint pour se convaincre que l'interaction joue un rôle décisif sur la manière de résoudre les contraintes géométriques. Ainsi, nous proposons d'étudier le cadre simple des *systèmes isothétiques* pour illustrer cette relation. Un solveur en temps réel pour les contraintes isothétique est présenté pour la conception en design architectural.

Résoudre

Les méthodes laissées un peu de coté en raison de leur relative lenteur ou du nombre de solutions proposées, sont de nouveau sollicitées pour augmenter la puissance de résolution des solveurs 3D.

Parmi celles-ci, les méthodes à base de connaissances géométriques permettent une approche constructive de la résolution et une décomposition en fonction de la capacité des solveurs à résoudre effectivement les sous-systèmes.

Pascal Schreck et Pascal Mathis ont développé à Strasbourg de tels solveurs symboliques de contraintes géométriques en 2D [DMS98, Sch02]. Ce type de solveur utilise des règles de chaînage avant et produit un plan de construction paramétré par des valeurs d'angles et de distances. Lorsque ces valeurs sont modifiées, il suffit de réévaluer le plan de construction pour mettre à jour la solution. Certaines constructions élémentaires, comme par exemple l'intersection entre une droite et un cercle, fournissent plusieurs solutions. Lorsque l'on fait varier les valeurs des paramètres, la continuité des solutions n'est pas assurée et Caroline Essert-Villard [EV01] a proposé de recourir à l'homotopie symbolique et au gel de branche pour résoudre ce problème.

Cumulant les avantages de l'exactitude, et de la possibilité de donner des explications géométriques sur le processus de résolution, ces méthodes nous semblent à même de surmonter la plupart des problèmes inhérents au passage à la dimension 3. Malheureusement, dans l'espace, les méthodes géométriques formelles sont difficilement applicables et sont souvent peu ou pas efficaces.

Ainsi, nous proposons de nous intéresser aux problèmes irréductibles pour une méthode de décomposition donnée. La combinaison de notre approche constructive et de résolution numérique peut permettre de résoudre ces problèmes en gardant au maximum les bonnes propriétés de la résolution formelle. La méthode de reparamétrisation que nous proposons étend la portée de ces solveurs symboliques aux contraintes géométriques 3D.

Mettre en oeuvre

Historiquement, tous les travaux de notre équipe concernant la résolution de contraintes géométriques ont été influencés par le domaine des constructions géométriques dans l'enseignement de la géométrie (EIAO pour Enseignement Intelligent / Interactif Assisté par Ordinateur). Les points de vue de la CAO et de l'EIAO sont pourtant différents :

- Les problèmes de géométrie en EIAO sont en général petits et astucieux et ont une vocation pédagogique. La plupart du temps, on peut les résoudre à l'aide

des outils du géomètre (règle et compas) et toutes les solutions possibles sont recherchées.
- Du point de vue de la Conception Assistée par Ordinateur, les problèmes sont souvent gros mais généralement réductibles à des problèmes plus simples. On cherche en général des solutions proches de celle attendue par le concepteur.

Cette différence nous a aiguillée vers une approche au niveau méta où l'univers géométrique, c'est-à-dire le contexte dans lequel se déroule la résolution, est une donnée fournie au logiciel. Ainsi, suivant les objets et les contraintes considérés, des solveurs spécifiques et optimisés peuvent être développés.

Nos prototypes d'interfaces se basent sur une librairie de Réalité Virtuelle développée au LSIIT [Ste06]. Ils permettent de valider expérimentalement les idées évoquées, au sein d'une architecture prenant en compte l'univers géométrique d'une part au niveau de l'interaction et, d'autre part, au niveau de la résolution grâce à un *noyau pour la résolution de contraintes*.

À ce noyau, se sont greffés un format de fichiers nommé *Geometric Constraint Mark-up Language* (GCML) utilisant la technologie XML [WSMF06] pour décrire les univers géométriques mais aussi les systèmes de contraintes qui s'y rapportent, ainsi qu'un outil permettant de *générer et optimiser des solveurs* au sein d'une application de résolution de contraintes géométriques.

Plan du mémoire

Le premier chapitre propose la définition formelle de la notion d'univers géométrique et expose une classification des principales méthodes de résolution par leur univers géométrique. Le second chapitre présente des expérimentations de techniques d'interaction 3D issues de la réalité virtuelle pour la modélisation par contraintes. Le troisième chapitre décrit une méthode hybride de résolution de contraintes 3D, nommée reparamétrisation, et les expérimentations menées dans l'univers des polyèdres. Le quatrième chapitre détaille la mise en œuvre de notre approche paramétrée par l'univers géométrique et définit un format de fichier pour décrire les univers et les systèmes de contraintes.

Chapitre 1

Formalisation

La résolution de systèmes de contraintes géométriques est une fonctionnalité offerte par la plupart des logiciels de Conception Assistée par Ordinateur (CAO). Elle permet à l'utilisateur de spécifier des contraintes dimensionnelles, comme en dessin technique, entre des objets géométriques ou un assemblage d'objets.

D'une manière générale, trouver rapidement toutes les solutions d'un système de contraintes est un problème difficile : s'il existe en 2D des méthodes relativement performantes et capables de résoudre la plupart des problèmes couramment rencontrés en dessin technique, cela n'est pas le cas en dimension 3 où, à l'heure actuelle, les utilisateurs se contentent d'extensions imparfaites de méthodes 2D.

Nous proposons d'aborder le problème en mettant en évidence les différents cadres dans lesquels s'appliquent les méthodes de résolution existantes : les univers géométriques.

Notre méthode repose sur l'expression de l'univers géométrique dont nous commençons par proposer une définition formelle ainsi que des notions qui en dépendent : système de contraintes géométriques et solutions. Puis, nous présentons les principales méthodes existantes en mettant en avant l'univers géométrique qu'elles sous-tendent et en essayant d'en établir une classification en suivant ce point de vue.

Sommaire

1.1 Définitions

La description de systèmes de contraintes peut donner lieu à des ambiguïtés. Pour les éviter, nous proposons de spécifier l'univers dans lequel s'exprime le problème.

Dans un premier temps, des exemples d'ambiguïtés sont présentés. Puis, les définitions formelles nécessaires sont données.

Contexte et ambiguïtés

Que ce soit en Enseignement Assisté par Ordinateur (EAO) ou en CAO, les énoncés sont exprimés dans un contexte qui n'est pas toujours clairement décrit.

Soit un triangle ABC.
Les distances respectives entre
A et B, B et C, et A et C
valent 4, 3 et 5.

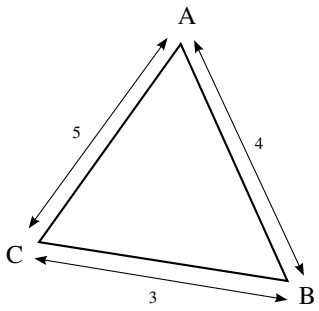


FIG. 1.1 – Un système de contraintes géométriques : énoncé et esquisse.

Dans l'esquisse cotée de la figure 1.1, on considère un triangle et des contraintes de distances entre sommets. Dans cet énoncé, la description présuppose un *contexte* géométrique, *i.e.* ce qu'est un point, ce qu'est un segment, et qu'il faut relier 3 points par des segments pour faire un triangle.

La description de ce contexte devient nécessaire lorsque les termes et les annotations utilisés deviennent moins courants que ces simples objets bien définis en école primaire.

Soit un triangle ABC tel que
AB=4 cm, BC=3 cm et ABC=90°.
Construire le triangle ABC.

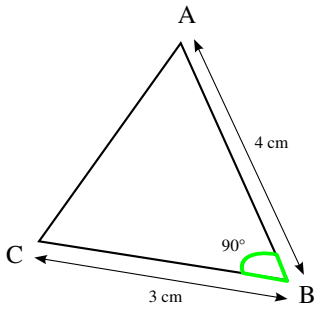


FIG. 1.2 – Énoncé plus précis mais ambigu.

De plus, des ambiguïtés peuvent apparaître s'il existe plusieurs définitions possibles pour un même objet. Par exemple, dans la figure 1.2, on considère un angle à la place d'une distance. Selon que l'angle est orienté ou non, il est possible d'obtenir un ensemble plus ou moins grand de solutions : avec ou sans les symétriques par rapport à n'importe quelle droite du plan. Ainsi, certaines des solutions à l'énoncé

1.1 sont représentées dans la figure 1.3, mais suivant la définition d'un angle, elles ne sont pas toutes des solutions de l'énoncé 1.2.

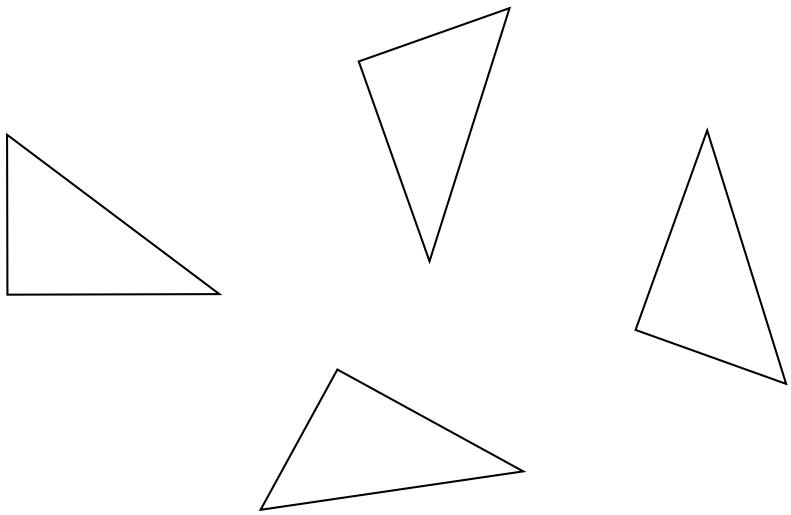


FIG. 1.3 – Quelques solutions aux problèmes 1 et 2.

La description d'une solution peut être simplement donnée par les coordonnées des sommets du triangle dans un repère canonique comme c'est le cas en CAO : $A(0,0), B(4,0), C(4,3)$. Ainsi, la connaissance de la manière de produire cette figure solution est perdue. C'est pour cela qu'en EAO, il est demandé d'écrire cette manière de faire en utilisant des connaissances communes préalablement acquises.

Par exemple, pour la figure 1.2, une solution particulière, en supposant défini un repère canonique xOy, peut être décrite de manière textuelle par *un plan de construction* :

```
Dessiner un point A aux coordonnées (0,0).
Tracer une droite d passant par A dans la direction de l'axe Ox.
Placer un point B à une distance de 4 unités de A sur d.
Tracer une droite d' passant par B et faisant un angle de 90° avec d.
Placer un point C à une distance de 3 unités de B sur d'.
Surligner les segments AB, BC, et AC.
Effacer les droites d et d'.
```

FIG. 1.4 – Plan de construction correspondant à la figure 1.2

Cette description parait très précise mais elle reste pourtant ambigüe quant au choix des orientations de droites et d'angles. On peut ainsi la transformer en programme de construction en ajoutant des conditionnelles et des boucles [Sch93] et ainsi exprimer syntaxiquement l'ensemble des solutions.

Abstraction des valeurs numériques

En faisant abstraction des valeurs numériques, on passe à une description permettant de généraliser la solution. Ainsi, l'ensemble de toutes les solutions à un problème de contraintes géométriques est représentée de manière générique. En supprimant la dépendance du repère canonique et en remplaçant les valeurs des distances par des symboles, on obtient :

```
Dessiner un point libre A.
Tracer une droite d passant par A.
Placer un point B à une distance de i unités de A sur d.
Tracer une droite d' passant par B et
 faisant un angle de j unités avec d.
Placer un point C à une distance de k unités de B sur d'.
Surligner les segments AB, BC, et AC.
Effacer les droites d et d'.
```

Les exemples précédents mettent en avant l'importance de bien exprimer l'univers géométrique d'un système de contraintes et de sa ou ses solutions. Ainsi, un même symbole employé dans deux contextes différents peut mener à une confusion et donc à une erreur :

- s'il manque un élément de syntaxe permettant de préciser le contexte, on dira que c'est une erreur syntaxique, comme dans l'exemple 1.1,
- si l'existence des différentes significations possibles n'est pas précisée, on dira que c'est une erreur sémantique, comme dans l'exemple 1.2.

Considérer ces deux niveaux permet de lever ces ambiguïtés et ainsi définir plus précisément ce qu'est un problème de contraintes géométriques dans un univers géométrique donné.

▷ **Exemple 1.1.1**
Lorsqu'on écrit $x^2+1=0$, on utilise les symboles x, 2, $+$, 1, $=$ et 0. En mathématique, des sens sont associés à chacun de ces symboles :

- $+$ dénote une opération entre deux éléments,
- 0 représente souvent l'élément neutre dans une structure algébrique,
- $=$ est utilisé pour indiquer une relation de congruence,
- x^2 dénote l'opération de mise au carré, qui se traduit par $x \times x$, où $\times$ est la multiplication.

Lorsqu'on parle d'équations pour désigner les objets mathématiques de ce type, on admet l'existence d'un univers mathématique connu par tous. ◁

Les sections suivantes présentent les définitions des notions d'univers géométriques, de systèmes de contraintes géométriques et de solutions. Ces définitions s'appuient sur le formalisme des spécifications algébriques [Wir90, GWM+00].

1.1.1 Syntaxe d'un univers géométrique

Commençons par définir la notion classique de *signature hétérogène* d'un univers géométrique :

Définition 1.1.1 Signature hétérogène
Une signature hétérogène est un triplet $\Sigma = \langle S, F, P \rangle$ où :
- S est un ensemble fini de symboles appelés *sortes*,
- F est un ensemble fini de *symboles fonctionnels* muni de deux fonctions de typage :
  - l'arité , $\mathtt{ar} : F \to S^*$, qui décrit le type des arguments des fonctions ;
  - et la co-arité, $\mathtt{coar} : F \to S$, qui décrit le type du résultat de la fonction.
- P est un ensemble fini de *symboles prédicatifs* muni également d'une fonction arité, $\mathtt{ar} : P \to S^*$, qui décrit le type des arguments du prédicat.

Par convention, on note :
- pour les symboles fonctionnels : $f : s_1 \dots s_n \to s$ avec $f \in F$, $\mathtt{ar}(f) = s_1 \dots s_n$, et $\mathtt{coar}(f) = s$,
- pour les symboles prédicatifs : $p : s_1 \dots s_n$ avec $p \in P$, $\mathtt{ar}(p) = s_1 \dots s_n$.

L'arité d'un symbole peut être le mot vide, noté ε, et c'est ce qu'on appelle une constante : $a :\to s$ avec $\mathtt{ar}(a) = \varepsilon$ et $\mathtt{coar}(a) = s$.

▷ **Exemple 1.1.2**
Prenons la signature suivante :

Signature Elem
sortes :
elem
symboles fonctionnels :
$c : \to$ elem
s : elem $\to$ elem
$\_a\_$: elem elem $\to$ elem
symboles prédicatifs :
$\_e\_$: elem elem

Ainsi, c est une constante, l'arité de a et de e est $\mathtt{ar}(a) = \mathtt{ar}(e) =$ elem elem et la co-arité de s est $\mathtt{coar}(s) =$ elem. ◁

Cette notion de signature permet de définir les symboles de la syntaxe d'un univers géométrique. Elle permet aussi de fournir un typage pour des variables exprimées dans cet univers, que l'on note $v : s$ avec v une variable et $s \in S$ pour un univers $\Sigma = \langle S, F, P \rangle$ donné. Les variables typées par des sortes de l'univers Σ sont dites Σ-sortées.

▷ **Exemple 1.1.3**
Dans l'exemple 1.1.1, x, 1 et 2 sont des variables dans la signature de l'exemple 1.1.2. ◁

L'application des symboles fonctionnels ou prédicatifs de la signature à des variables typées, permet de produire des *termes*.

▷ **Exemple 1.1.4**
Dans la signature de l'exemple 1.1.2, soient x, y : elem.
Un exemple de terme construit sur les variables x et y et la constante c est :

$$s(x)as(s(c))ey$$

◁

On peut voir la signature comme un *patron* pour la fabrication de termes. Si seuls des symboles fonctionnels sont utilisés, on dira que l'on a affaire à un terme fonctionnel, par exemple $x^2 + 1$. On appelle terme prédicatif les termes comportant un symbole prédicatif. Ainsi, $x^2 + 1 = 0$ est un terme prédicatif. On notera Tf_Σ et Tp_Σ l'ensemble de tous les termes respectivement fonctionnels et prédicatifs.
Lorsque l'ensemble des variables est vide, on parle de *terme clos*. Par exemple, dans la signature de l'exemple 1.1.2, $s(c)$ est un terme clos.

Un univers géométrique est l'expression d'une théorie logique du premier ordre dont un modèle privilégié est de nature géométrique. Pour constituer des formules, on considère une syntaxe permettant de combiner et de quantifier les termes :

- les connecteurs logiques habituels : $\Rightarrow$ pour l'implication, $\wedge$ pour l'intersection, $\vee$ pour l'union, $\neg$ pour la négation, dans cet ordre de priorité,
- les quantificateurs : $\exists$ (il existe) et $\forall$ (quelque soit) portant sur des variables typées.

La partie quantification est préfixée à l'aide du symbole |. Les symboles (et) sont utilisés pour préciser la portée des opérateurs logiques.

▷ **Exemple 1.1.5**
$\forall\ (a : \text{elem})\ \exists\ (b : \text{elem})\ |\ \neg\ (a = 0) \wedge (s(a) = b) \vee (s(b) = a)$ est une formule logique dans la signature précédente. ◁

Nous avons maintenant tous les ingrédients nécessaires pour la définition de la syntaxe d'un univers géométrique :

Définition 1.1.2 Syntaxe d'un univers géométrique
La syntaxe d'un univers géométrique est composée de :
- une signature Σ,
- un ensemble de formules dans Σ, nommées **axiomes**.

▷ **Exemple 1.1.6**
Dans [Hil71], Hilbert propose 21 axiomes associés à la signature suivante :

Signature Hilbert
sortes :
point
droite plan
angle
symboles prédicatifs :

$\bowtie$: point point point
_ in_1 _ : point droite
_ in_2 _ : point plan
_ in_3 _ : droite plan
_ $\cong$ _ : angle angle

La signification de cette signature est décrite dans l'exemple 1.1.8. Nous ne présentons que deux de ces axiomes : le premier et le onzième.

Le premier axiome traduit l'existence d'une droite passant par 2 points :

$$\forall \text{ (a,b : point) } \exists \text{ (c : droite) } | \text{ (a } in_1 \text{ c)} \wedge \text{(b } in_1 \text{ c)}$$

Plus élaboré, le onzième axiome exprime que sur 3 points pris au hasard sur une droite, l'un d'entre eux se situe entre les deux autres :

$$\forall \text{ (d : droite) } \exists \text{ (a,b,c : point) } |$$
$$(((\text{a } in_1 \text{ d}) \wedge (\text{b } in_1 \text{ d}) \wedge (\text{c } in_1 \text{ d})) \Rightarrow (\bowtie(\text{a,b,c}) \vee \bowtie(\text{b,a,c}) \vee \bowtie(\text{a,c,b})))$$

$\triangleleft$

1.1.2 Sémantique d'un univers géométrique

Pour donner une signification à un univers géométrique, il faut donner un sens à chaque élément de sa signature. C'est ce qui est fait en définissant une Σ-algèbre :

Définition 1.1.3 **Σ-algèbre**
Soit Σ une signature hétérogène.
Une Σ-algèbre E consiste à associer :
- un ensemble E_s à chaque sorte $s \in \Sigma$,
- une fonction $\tilde{f} : E_{s_1} \times \ldots \times E_{s_n} \mapsto E_s$ par symbole fonctionnel $f : s_1 \ldots s_n \rightarrow s$ défini dans Σ,
- un sous-ensemble $\tilde{p} : E_{s_1} \times \ldots \times E_{s_n}$ par prédicat $p : s_1 \ldots s_n$ défini dans Σ.

Lorsque $(o_1, \ldots, o_n) \in \tilde{p}$, on dit que $\tilde{p}(o_1, \ldots, o_n)$ est *vraie* ou *vérifiée*.

Le passage d'une signature à une Σ-algèbre s'appelle une *interprétation de la signature*. Plusieurs interprétations différentes peuvent être associées à une même signature.

$\triangleright$ **Exemple 1.1.7**
La signature de l'exemple 1.1.2 peut s'interpréter comme une spécification des entiers naturels $\mathbb{N}$. En effet, c est une constante représentant l'élément neutre, les symboles s et a peuvent être associés aux opérations classiques de successeur et d'addition et e à l'égalité.

Mais, une interprétation par n'importe quel groupe, comme par exemple $\mathbb{Z}/3\mathbb{Z}$, convient. De la même manière, si on considère que le symbole a a pour signification la soustraction et que le symbole s dénote le prédécesseur, on obtient les entiers négatifs $\mathbb{N}^-$.

Dans la syntaxe communément admise en mathématiques, le symbole 1 de l'exemple 1.1.1 fait partie des symboles fonctionnels et correspond à $s(c)$ dans la signature de l'exemple 1.1.2. ◁

Définition 1.1.4 Valuation
Soit Σ une signature et X un ensemble de variables Σ-sortées. Une valuation de variables est une fonction $v : X \mapsto E$ respectant les types : si $x : s$ alors $v(x) \in E_s$.

Pour interpréter les formules, les symboles de prédicats correspondent à des relations booléennes dans les Σ-algèbres, les connecteurs logiques sont interprétés suivant les tables de vérité bien connues et les variables libres (non quantifiées) sont interprétées comme si elles étaient quantifiées universellement.

▷ **Exemple 1.1.8**
Dans l'axiomatique d'Hilbert (*cf.* exemple 1.1.6), les sortes dénotent classiquement les objets points, droites, plans et angles dans $\mathbb{R}^3$, les symboles fonctionnels sont tous des constantes permettant de construire les objets précédents, le prédicat $\bowtie$ dénote la contrainte qu'un point est au milieu de deux autres points, in_1,in_2 et in_3 dénotent l'incidence d'objets à d'autres objets et enfin $\cong$ la relation d'égalité entre 2 angles non orientés. ◁

1.1.3 Système de contraintes et solutions

Dans cette section, nous présentons la manière dont s'exprime la syntaxe d'un problème de contraintes dans un univers donné :

Définition 1.1.5 Système de contraintes géométriques
Étant donnée une signature Σ, un système de contraintes est un triplet $S = \langle C, \chi, A \rangle$ où :
- C est une conjonction finie de termes prédicatifs,
- χ est un ensemble Σ-sorté fini de symboles appelés inconnues,
- A est un ensemble Σ-sorté fini de symboles appelés paramètres,

avec $\chi \cap A = \emptyset$.

Ainsi, un système de contraintes peut être syntaxiquement défini et la sémantique des inconnues de ce système n'est pas donnée explicitement dans la Σ-algèbre associée.

Définition 1.1.6 Solution d'un système
Soit un système $S = \langle C, \chi, A \rangle$.
Pour une valuation de A, on appelle solution d'un système de contraintes géométriques une valuation $v : \chi \mapsto E$ telle que l'interprétation de C soit vérifiée.

L'ensemble des solutions d'un système S est noté $F(S)$. On peut abstraire la notion de solution d'un système en la paramétrant par les variables de A.

Définition 1.1.7 Solution paramétrée
Une solution paramétrée à un système de contraintes $S = \langle C, \chi, A \rangle$ est un ensemble de valuations $v_a : \chi \mapsto E$ avec $a = \{a_1, ..., a_n\}$ donné dans A.

Une *solution formelle* est une formulation paramétrique de $F(S)$. Elle est souvent représentée par un programme de construction en EAO, *i.e.* une suite d'opérations avec des conditionnelles.

Définition 1.1.8 Constriction
Dans une Σ-algèbre donnée, on dit qu'un système de contraintes géométriques S est :
– bien-contraint si $F(S)$ est fini ;
– sous-contraint si $F(S)$ est infini ;
– sur-contraint si $F(S)$ est vide.

Une solution paramétrée peut être bien, sur et sous contrainte selon la valuation des paramètres choisie. Par exemple, un triangle contraint par la longueur de ses trois côtés est sur-contraint si les longueurs ne respectent pas l'inégalité triangulaire.

1.2 Univers géométriques et solveurs

Un solveur est une fonction de P_U dans V, avec P_U l'ensemble des systèmes de contraintes $S = \langle C, \chi, A \rangle$ exprimables dans l'univers U et V un ensemble de valuations $v : \chi \mapsto E$.

Il est *correct* s'il n'engendre pas de fausses solutions, *i.e.* si $V \subset F(S)$; il est dit *complet* par rapport à un univers U, s'il est capable de calculer toutes les solutions pour chaque système exprimé dans U, *i.e.* si $F(S) \subset V$.

Un solveur *paramétrique* est un solveur capable de produire des solutions formelles. Les résultats de tels solveurs sont des fonctions que l'on peut interpréter pour déterminer les solutions numériques en fonction des valeurs numériques des paramètres. Les solveurs symboliques peuvent manipuler les paramètres pendant le processus de résolution et ils construisent ces fonctions sous forme de programmes ou de plans de construction (comme par exemple dans la figure 1.4), ou de bases standard (ou de Gröbner). C'est pourquoi on appelle ces solveurs des *solveurs constructifs*.

Dans cette section, nous décrivons les univers géométriques dans lesquels les solveurs s'appliquent, à partir de la présentation des techniques générales de résolution.

Les solveurs sont habituellement classés en considérant 3 critères :
– les structures de données sous-jacentes ;
– le type de décomposition utilisé ;
– la nature de la solution produite.

Structures de données

Des travaux de Sutherland [Sut63] jusqu'à [BP07] et en passant par [RSV88], une taxonomie plus ou moins respectée s'est construite. Ainsi, les méthodes de résolutions sont généralement classées suivant les structures de données employées :
– les méthodes à base d'équations, [Wu84, Cho88, Kon92] ;
– les méthodes à base de connaissances, [Ald88, VSR92, DMS98] ;

– les méthodes à base de (hyper-)graphes, [Owe91, AAM93, BFH+95].

Cette taxonomie adopte un point de vue où les problèmes de contraintes sont respectivement décrits :

– soit par des équations, souvent polynômiales ;
– soit par des prédicats de la logique du premier ordre ;
– soit par des valuations de sommets et d'arêtes en théorie des graphes.

La résolution de systèmes de polynômes fait appel soit à des techniques très puissantes de calcul algébrique mais dont la complexité est au moins exponentielle - ce qui explique pourquoi elles sont peu utilisées en CAO -, soit à des techniques itératives numériques efficaces mais dont la convergence numérique n'est pas maîtrisée.

La représentation par des prédicats permet l'utilisation de techniques d'inférence sur des connaissances géométriques à l'aide d'un système expert ou de règles de réécriture. En restreignant la classe des problèmes traités, la complexité de ces solveurs est polynomiale dans le nombre d'inconnues et le temps de résolution est proportionnel au nombre de règles considérées. Dans le cadre de la géométrie en général, ces règles sont très nombreuses et une des solutions retenues pour améliorer l'efficacité de ces solveurs est de restreindre ce nombre de règles à un univers géométrique moins général.

Les techniques d'analyse combinatoire issues de la théorie des graphes sont plus abstraites car elles s'intéressent à la structure des systèmes de contraintes. Malheureusement, les caractérisations structurelle proposée sont incorrectes dans le cadre général de la géométrie. Elles sont pourtant très efficaces pour des conditions de généricité des systèmes et résolvent ainsi un grand nombre des problèmes 2D issus de la CAO industrielle.

Le type de décomposition

Pour pouvoir dominer la complexité en taille d'un problème de contraintes géométriques, la majorité des solveurs actuels mettent en application le paradigme : « diviser pour régner ».

Dans ces solveurs, la résolution est ainsi constituée de deux phases : la phase de planification, consacrée à la recherche d'un plan de décomposition-assemblage de sous-systèmes et une phase de résolution proprement dite où les sous-systèmes irréductibles sont résolus.

En pratique, la résolution par décomposition contient généralement les étapes suivantes :

– décomposer un système en parties,
– résoudre les parties,
– assembler les solutions des parties.

La résolution est appliquée localement sur chacun des sous-systèmes, et les solutions partielles sont assemblées pour obtenir la solution globale.

Les travaux sur la décomposition mettent en avant cette première phase d'analyse de systèmes de contraintes et font tous référence à une certaine granularité des problèmes en utilisant différents termes pour désigner les sous-systèmes : *macros-figures* pour Cugini *et al.* [CDG85], CD-*sets* pour Sunde [Sun86], *clusters* pour Hoffmann *et al.* [HC02b], ...

Ces solveurs sont généralement regroupés suivant le type de décomposition :
- les décompositions équationnelles ou par blocs,
- les décompositions géométriques ou décompositions utilisant l'invariance par les déplacements,
- les décompositions structurelles, ou décompositions suivant des heuristiques combinatoires.

Mais la finesse d'abstraction n'implique pas forcément la finesse de décomposition. Même si les décompositions équationnelles peuvent « casser des objets en plusieurs morceaux », elles ne donnent pas forcément la décomposition la plus fine, car on obtient des composantes fortement connexes, qui pourraient être cassées par des méthodes tirant parti de connaissances géométriques et en particulier de l'invariance par déplacement (*cf.* 1.2).

Les méthodes utilisant une heuristique pour décomposer sont plus efficaces mais font perdre la sémantique géométrique à la base de cette possibilité de décomposer. Par exemple, ces méthodes ne prennent pas en compte la présence de cas dégénérés, ou de singularités induites par des théorèmes ne faisant intervenir que des incidences, comme le théorème de Pappus vu en fin de chapitre (exemple 1.20).

Globalement, ces décompositions peuvent être scindées en deux types : les décompositions ascendantes et les décompositions descendantes. Ces termes s'expliquent lorsqu'on considère le parcours d'un arbre représentant la décomposition où la racine est le SCG global et les autres nœuds les sous-systèmes déterminés récursivement.

Cette distinction ne met pas assez en avant l'importance relative de la planification par rapport à la résolution. Nous utilisons plutôt le terme de *décomposition a priori* lorsqu'on présume que la résolution des sous-systèmes est faite par des solveurs complets, et le terme de *décomposition constructive*, lorsque la résolution de sous-systèmes permet de déduire des connaissances pour résoudre un autre sous-système.

Les décompositions *a priori* procèdent essentiellement par l'application d'un critère combinatoire de décomposition. À la base, cette caractérisation combinatoire est issue d'un cadre particulier appelé *théorie de la rigidité* [GSS97] qui cherche à caractériser la rigidité des graphes représentant un système de contraintes de distance entre points. Intuitivement, un assemblage d'objets est rigide s'il est indéformable : *i.e.* ses objets n'admettent aucun mouvement relativement les uns aux autres. La notion de rigidité a été principalement étudiée par la communauté de topologie structurale qui a présenté différentes formes de rigidité. Pour une présentation plus détaillée des différentes formes de rigidité, le lecteur pourra consulter [Jer02].

Les critères combinatoires sont utilisés soit pour l'analyse de la connectivité du

graphe pour détecter les repères en commun et des motifs d'assemblage, où la notion de paire d'articulation joue un rôle essentiel [Owe91], soit pour la recherche de composantes structurellement bien-contraintes en considérant la densité des graphes [HLS98, Jer02, Sit03].

Dans cette recherche de composantes structurellement bien-contraintes, on peut aussi citer la décomposition équationnelle [LM94a] qui travaille sur le graphe biparti inconnues-équations (voir 1.3.3). Un couplage parfait permet de dégager les variables structurellement bien, sur et sous contraintes et les composantes connexes du graphe donnent une planification de la résolution.

La classe des méthodes de décompositions constructives utilise la capacité du solveur à résoudre des sous-systèmes pour découvrir ces parties à assembler. Parmi les méthodes de décomposition constructives, on peut distinguer :

- les méthodes à base de pattern [TN97, Sun87],
- les méthodes à base de propagation de degrés de libertés [HV95, BFH+95],
- les méthodes à base de systèmes de connaissances [But79, Ald88, Sch93].

Ces méthodes de décomposition utilisent plus ou moins explicitement une des caractéristiques des systèmes de contraintes géométriques : l'invariance par déplacement [Mat97].

En effet, les systèmes de contraintes géométriques sont souvent invariants par le groupe euclidien $\mathbb{I}$ des isométries directes, on dit $\mathbb{I}$-invariant. Alors, les définitions de bonne, sur et sous constriction deviennent :

- le système S est bien-contraint *modulo* $\mathbb{I}$ s'il existe un nombre fini de solutions à une isométrie près, ou en d'autres mots, l'ensemble $F(S)/\mathbb{I}$ des orbites de $F(S)$ sous l'action du groupe $\mathbb{I}$, est fini,
- S est sous-contraint *modulo* $\mathbb{I}$ si $F(S)/\mathbb{I}$ est infini,
- S est sur-contraint *modulo* I si $F(S)/\mathbb{I} = F(S) = \emptyset$.

Un système bien-contraint *modulo* $\mathbb{I}$ peut être bien-contraint en ajoutant quelques contraintes capables de fixer un repère, comme par exemple un point et une direction en 2D ou un point et 2 directions en 3D.

La décomposition de systèmes $\mathbb{I}$-invariants nécessite l'ajout de contraintes pour rendre bien-contraints les sous-systèmes. Dans notre terminologie, nous appelons de telles contraintes le bord du système.

▷ **Exemple 1.2.1**

La figure 1.5(a) présente un hexagone $ABCDEF$ pour lequel la longueur de chacun des 6 côtés ainsi que la valeur de 3 angles sont imposées. L'utilisation de règles de construction géométriques ne permet pas de construire facilement un hexagone répondant à ces contraintes. En revanche, l'emploi de règles classiques par un mécanisme de décomposition-recombinaison conduit à un plan de construction assez élémentaire. Dans un premier temps, le triangle AEF est construit. La distance AE peut alors être calculée sur AEF. Ce triangle est retiré et remplacé par la distance AE (figure 1.5(b)). Cette nouvelle contrainte constitue le bord de AEF, qui est composé des éléments communs au reste de la figure : les points A et E, et les contraintes sur ces éléments qui sont invariantes par déplacement.

Une procédure similaire permet d'agir de même avec le triangle ABC. Le sys-

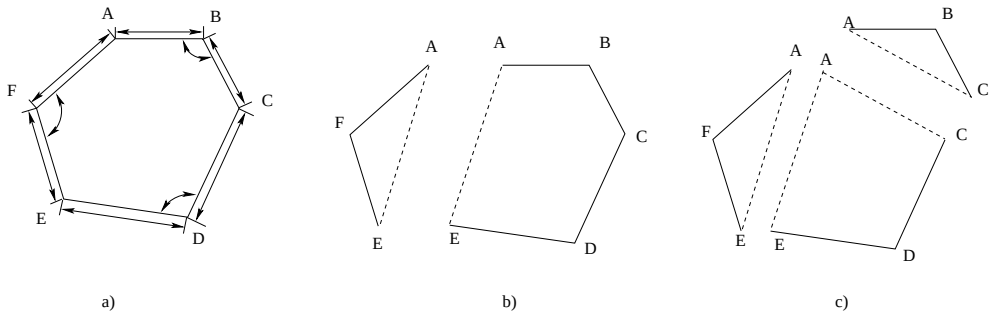


FIG. 1.5 – Hexagone à 6 distances et 3 angles

tème de contraintes restant est un quadrilatère $EDCA$ (figure 1.5(c)) qu'une simple construction géométrique permet de résoudre. Les 3 sous-systèmes ainsi formés (deux triangles et un quadrilatère) peuvent ensuite être assemblés par déplacement le long des points qu'ils partagent deux à deux. ◁

On peut pourtant comparer certaines méthodes de décomposition *a priori* et constructives suivant l'univers d'application de ces méthodes. Ainsi, les méthodes de Sunde [Sun86], Owen [Owe91], et Fudos-Hoffmann [FH97] sont des méthodes de décomposition qui considèrent des objets à 2 degrés de libertés et des contraintes dont le degré de restriction vaut 1.

La résolution est toujours très simple : triviale chez Sunde [Sun86], propagation de contraintes chez Hoffmann *et al.* et Ait Aoudia [AAM93], et résolution de triangles chez Owen [Owe91]. L'assemblage est trivial chez Owen [Owe91], alors qu'il est sophistiqué chez Sunde [Sun86], et intermédiaire chez Hoffmann *et al.*.

Même si dans des univers 2D particuliers, les décompositions *a priori* résolvent beaucoup de problèmes de contraintes couramment rencontrés en CAO, dans le cadre général, elles restent incorrectes car on ne connaît pas de caractérisation structurelle combinatoire exacte de la rigidité. De plus, les dépendances d'un ensemble de contraintes sont difficilement détectables.

Quant aux décompositions équationnelles, même si elles permettent de décomposer très finement en travaillant au niveau des coordonnées, elles ne tirent pas partie de la géométrie qui indique comment décomposer des composantes fortement connexes par la mise en évidence de repères locaux [DMS98]. Les décompositions constructives sont plus lentes et plus complexes dans le cadre général.

La nature de la solution

La résolution de systèmes de contraintes géométriques a aussi été étudiée dans le cadre de l'Enseignement Assisté par Ordinateur (EAO) lorsque les chercheurs se sont intéressés aux exercices de constructions à la règle et au compas dans l'enseignement secondaire. Cette approche a conduit au développement de la géométrie dynamique [Bau90] et de tuteurs pour la géométrie [ADT92]. Ces techniques et ces savoir-faire issus des constructions géométriques ont aussi été appliqués à la CAO [Sch93].

Même si les domaines de la CAO et de l'EAO poursuivent fondamentalement le même but, *i.e.* trouver des solutions à un système de contraintes géométriques, les formulations et les exigences sont différentes comme peut le rappeler le tableau 1.6.

	CAO	EAO
Entrée du problème	une *esquisse cotée*	un *énoncé* du problème de construction
Type de résolution	peu importe	résolution *formelle*
Sortie du problème	une (ou plusieurs) figure(s) répondant aux cotations	une manière de construire le résultat (un *plan de construction*)
Nombre de solutions	une ou plusieurs solutions proches de l'esquisse	Ensemble de toutes les solutions
Nature de la solution	solution avec une certaine *tolérance*	*Solutions exactes*

FIG. 1.6 – Tableau présentant les caractéristiques des méthodes d'EAO et de CAO

Le critère essentiel de divergence issu de cette rencontre EAO/CAO est celui de la nature de la solution, *i.e.* l'obtention d'une solution générique exacte dans un cas, et d'une solution numérique approchée avec une certaine tolérance dans l'autre.

Ainsi, on peut distinguer le caractère de la méthode : *formel* ou *numérique* suivant la nature de la solution donnée.

D'une manière générale, l'approche formelle ou symbolique, qui regroupe les méthodes à base de calcul algébrique et les méthodes à bases de connaissances géométriques, donne l'ensemble des solutions exactes, comme par exemple la résolution par la méthode de Lebesgue étudiée dans [Sch01]. L'approche numérique, au contraire, fournit généralement une seule solution approchée (voir plus loin 1.3.3).

Les méthodes formelles sont en pratique plus lentes que les méthodes numériques car elles imposent la manipulation de termes.

Les méthodes formelles géométriques issues de l'EAO peuvent être adaptées au cadre de la CAO. Tout d'abord, esquisse cotée et énoncé du problème sont deux formes de représentations d'un système de contraintes géométriques. Il n'y a donc pas d'incompatibilité entre ces deux descriptions à condition qu'elles vérifient toutes deux la syntaxe de l'univers géométrique considéré. Les cotations étant l'expression de propriétés géométriques, les méthodes issues des constructions géométriques peuvent être directement utilisées en CAO car le solveur agit comme une boîte noire.

Ensuite, l'évaluation numérique d'une représentation formelle de la solution permet de visualiser interactivement les modifications de placement des objets en EAO, ou des cotations en CAO.

Pour finir, la solution la plus « proche » de l'esquisse cotée fournie par l'utilisateur peut être exhibée à l'aide de la technique du gel de branche exposée dans [EV01].

Ainsi, les qualités des méthodes formelles géométriques issues de l'EAO peuvent être mises à profit dans un solveur de CAO :

- savoir où et pourquoi on échoue, pour pouvoir corriger l'erreur ;
- avoir une connaissance du nombre de solutions ;
- disposer d'une structuration de l'espace des solutions lorsqu'il y a un grand nombre de solutions et mettre à jour les solutions ;
- pouvoir modifier les valeurs numériques des paramètres sans avoir à faire appel au solveur ;
- contrôler la précision de la solution numérique (travailler en double précision, en arithmétique des intervalles, ...) ;
- montrer qu'on peut construire une figure avec des instruments donnés, ce qui peut être utile, par exemple, dans le cadre de l'usinage.

Néanmoins, les techniques à base de connaissances géométriques sont intrinsèquement limitées par l'axiomatique qu'elles adoptent.

De plus, on rencontre des problèmes irréductibles que seule une méthode globale par approximations numériques successives permet de résoudre. La combinaison de ces deux types de résolution au sein d'une décomposition favorisant la résolution formelle, permet en pratique de résoudre une classe de problèmes plus étendue.

La lenteur des techniques formelles géométriques est un problème récurrent pour leur application dans des logiciels de CAO, mais, des méthodes formelles restreintes à une axiomatique particulière peuvent être instanciées en méthodes numériques pour gagner en efficacité comme par exemple la méthode de [Sun87], grâce à de la compilation de systèmes experts en structures de données efficaces [WSMF06].

Types d'univers

Lorsque l'on considère l'univers géométrique dans lequel s'applique une méthode, les trois critères présentés apparaissent naturellement.

Le critère des structures de données conduit à considérer 3 univers possibles :

- celui de la géométrie euclidienne, telle qu'elle est exprimée depuis Euclide ;
- celui de l'analyse, tel que Descartes l'a introduite ;
- et celui des graphes : l'univers combinatoire, tel qu'Euler l'a présenté.

Le critère de la décomposition nous en apprend plus sur les liens qui unissent ces univers :

- l'univers géométrique euclidien est l'univers de référence dans lequel le problème est exprimé mais les figures solutions sont exprimées dans l'univers analytique pour l'affichage sur la machine ;
- représenter les systèmes d'équations par des graphes permet de faire de la décomposition d'un point de vue combinatoire ;
- représenter la géométrie par des graphes permet d'abstraire les objets géométriques et les contraintes, et de faire de la décomposition combinatoire.

Et enfin, la solution peut être représentée symboliquement dans ces 3 univers mais une solution numérique n'est exprimée que dans l'univers analytique.

Ces trois univers sont présentés dans les trois sections suivantes :

1.2.1 Univers euclidien

Pour décrire les univers de manière formelle, nous utilisons une syntaxe descriptive basée sur la définition d'univers géométrique donnée au début du chapitre (*cf.* 1.1.2) : **sortes**, **symboles fonctionnels**, **symboles prédicatifs** et **axiomes**.

L'univers euclidien est un univers classique étudié à l'école primaire et secondaire. On en distingue deux, celui du plan et celui de l'espace, autrement dit de dimension 2 et 3. Nous ne présentons ici qu'une partie de ces deux univers, celle qui est généralement utilisée dans les méthodes à base de connaissance.

Univers euclidien 2D

Univers Euclide2D
sortes :
point
longueur
angle
droite
rayon
cercle
symboles fonctionnels :
$droite_{pp}$: point point → droite
$cercle_{pr}$: point rayon → cercle
$point_c$: cercle → point
$rayon_c$: cercle → rayon
$point_{ppll}$: point point longueur longueur → point
$inter_{dd}$: droite droite → point
$inter_{cc}$: cercle cercle → point
$inter_{dc}$: droite cercle → point
symboles prédicatifs :
$dist_{pp}$: point point longueur
$dist_{cc}$: cercle cercle longueur
$egal_{pp}$: point point
$egal_{ll}$: longueur longueur
$angle_{4pp}$: point point point point angle
$angle_{dd}$: droite droite angle
app_{pd} : point droite
app_{pc} : point cercle
$parall$: droite droite
$perp$: droite droite
$tang_{cc}$: cercle cercle
axiomes :
$\forall\ (p_1, p_2 : \text{point}) \wedge (\text{k} : \text{longueur})\ dist_{pp}(p_1, p_2, k) = dist_{pp}(p_2, p_1, k)$
$\forall\ (p_1, p_2 : \text{point})\ dist_{pp}(p_1, p_2, 0) \Leftrightarrow p_1 = p_2$
$\forall\ (p_1, p_2, p_3 : \text{point})\ dist_{pp}(p_1, p_2, k_1) \wedge dist_{pp}(p_2, p_3, k_2) \wedge dist_{pp}(p_1, p_3, k_3)$
$\quad \subset k_3 \leq k_1 + k_2$
...

Classiquement, la sorte *point* dénote les points et ainsi de suite pour les objets géométriques classiques. Les sortes *longueur*, *rayon* et *angle* dénotent des mesures spécialisées pour les contraintes de distances, les cercles et les contraintes d'angles.

Parmi les symboles fonctionnels, les constantes expriment la création d'un nouvel objet géométrique correspondant à la sorte de co-arité. Les noms sont préfixés par un c dans ce cas.

Le symbole $droite_{pp}$ dénote une fonction qui à partir de deux points distincts donne une droite passant par ces deux points. Le nom représente ainsi la co-arité et l'arité avec en indice les initiales des sortes utilisées.

Le symbole fonctionnel $point_{ppll}$ est un symbole atomique, c'est-à-dire qu'il factorise une construction connue pour éviter la présence d'objets intermédiaires. Il dénote la construction d'un point intersection de deux cercles dont les centres et les rayons sont respectivement les points et les longueurs de l'arité.

Les symboles $inter_{cc}$ et $inter_{dc}$ représentent respectivement l'intersection de deux cercles et l'intersection d'une droite et d'un cercle. Suivant que certains points soient égaux ou alignés, ou que certaines distances prennent la valeur 0, il y aura entre 0 et 2 solutions possibles pour ces symboles fonctionnels. Ce sont des *multi-fonctions*. La prise en compte des multi-fonctions peut se faire au niveau syntaxique, en utilisant des conditionnelles et une fonction différente pour chaque solution possible [Sch93]. Mais, pour ne pas surcharger les définitions formelles, nous avons choisi dans cette thèse de les considérer au niveau sémantique (voir chapitre 5).

Parmi les symboles prédicatifs, $dist_{pp}$ spécifie que la distance entre deux objets de sorte *point* est une valeur de sorte *longueur*.

$egal_{pp}$ et $egal_{ll}$ sont deux symboles spécifiant l'égalité respectivement entre points et entre longueurs. Nous utilisons dans la suite le symbole prédicatif $=$ comme sucre syntaxique pour désigner ce type de relation.

$angle_{4p}$ indique la mesure d'un angle entre 2 bi-points, et $angle_{dd}$ entre deux droites.

app_{pd} et app_{pc} spécifient l'appartenance d'un point à une droite ou à un cercle.

Enfin, *parall* et *perp* sont utilisés pour spécifier un parallèlisme ou une perpendicularité entre droites et $tang_{cc}$ pour la tangence de deux cercles.

On peut remarquer les axiomes indiquant que deux points sont égaux lorsque la distance les séparant est nulle, et l'inégalité triangulaire entre 3 points. Ces deux axiomes définissent les dégénérescences possibles lorsque l'on considère les points.

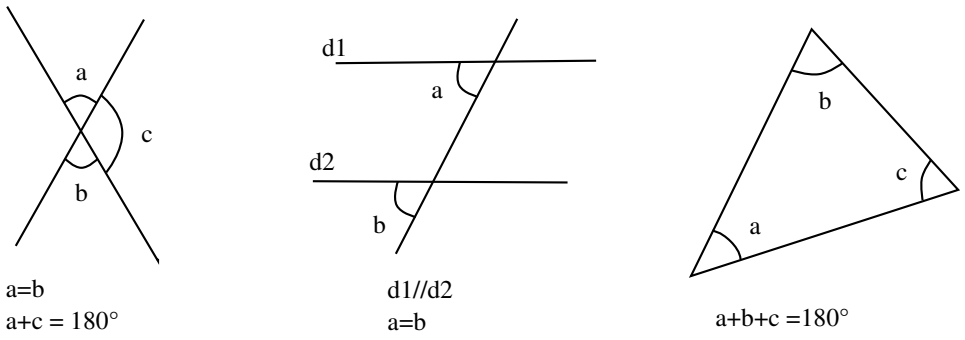


FIG. 1.7 – Relation entre les angles, angles alternes-internes, relation de Chasles

Les axiomes ne sont pas tous cités dans cette description, il serait trop long de le faire en extension. Les premiers axiomes décrits sont ceux caractérisant ce qu'est une distance. Mais il pourrait y en avoir bien d'autres comme par exemple pour les angles : ceux indiquant l'égalité des angles lorsque deux droites se coupent, ceux sur l'égalité des angles alternes-internes, ou la relation de Chasles dans un triangle, *etc.* (*cf.* figure 1.7). Ce sont ces axiomes qui permettent de traiter les cas particuliers de la géométrie, en tout cas ces singularités et les dégénérescences induites par l'alignement, le parallélisme, la coplanarité, *etc.*.

Univers euclidien 3D

En dimension 3, les types *sphere* et *plan* sont ajoutés aux sortes 2D. Cela augmente la complexité de l'univers en terme de symboles fonctionnels mais aussi en terme d'axiomatique (non décrite ici).

Univers Euclide3D
sortes :
point
longueur
angle
droite
rayon
sphere
plan
cercle
symboles fonctionnels :
$droite_{pp}$: point point → droite
$point_{3p3l}$: point point point longueur longueur longueur → point
$cercle_{pplr}$: point plan rayon → cercle
$cercle_{2p}$: point point → cercle
$point_c$: cercle → point
$rayon_c$: cercle → rayon
$plan_c$: cercle → plan
$sphere_{pr}$: point rayon → sphere
$centreSphere$: sphere → point
$rayonSphere$: sphere → rayon

$plan_{3p}$: point point point → plan
$plan_{pd}$: point droite → plan
$plan_{dpl}$: droite plan → plan
$plan_{dd}$: droite droite → plan
$plan_{rad}$: sphere sphere → plan
$inter_{dpl}$: droite plan → point
$inter_{2pl}$: plan plan → droite
$inter_{pls_1}$: plan sphere → point
$inter_{pls_2}$: plan sphere → cercle
$inter_{dd}$: droite droite → point
$inter_{ds}$: droite sphere → point
$inter_{ss1}$: sphere sphere → point
$inter_{ss2}$: sphere sphere → cercle
$inter_{3s}$: sphere sphere sphere → point
$inter_{dc}$: droite cercle → point
$inter_{cc}$: cercle cercle → point
$inter_{cs}$: cercle sphere → point
$inter_{sc}$: sphere cercle → point
$inter_{cpl}$: cercle plan → point
$inter_{plc}$: plan cercle → point

symboles prédicatifs :

$dist_{pp}$: point point longueur
$dist_{ppl}$: point plan longueur
$dist_{pd}$: point droite longueur
$dist_{ps}$: point sphere longueur
$dist_{2pl}$: plan plan longueur
$dist_{dpl}$: droite plan longueur
$dist_{pls}$: plan sphere longueur
$dist_{dd}$: droite droite longueur
$dist_{ds}$: droite sphere longueur
$dist_{cc}$: cercle cercle longueur
$dist_{ss}$: sphere sphere longueur
$angle_{4p}$: point point point point angle
$angle_{2d}$: droite droite angle
$angle_{2pl}$: plan plan angle
$angle_{dpl}$: droite plan angle
$parall_{2d}$: droite droite
$perp_{2d}$: droite droite
$parall_{2pl}$: plan plan
$perp_{2pl}$: plan plan
$parall_{dpl}$: droite plan
$perp_{dpl}$: droite plan
app_{pd} : point droite
app_{ps} : point sphere
app_{ppl} : point plan
app_{dpl} : droite plan

$tang_{pls}$: plan sphere
$tang_{ds}$: droite sphere
$tang_{ss}$: sphere sphere
axiomes :
...

Dans cet univers euclidien 3D, le cercle correspond également à la sorte *cercle* mais sa construction requiert la donnée d'un point, d'un rayon et d'un plan pour le symbole $cercle_{pplr}$. Toutefois, il peut aussi être construit par $cercle_{2p}$ à partir de deux points p_1 et p_2, $p1$ étant le centre, $\overrightarrow{p_1p_2}$ définissant la normale et $\|p_2 - p_1\|$ le rayon.
On peut remarquer que $plan_{dd}$, qui dénote la construction d'un plan à partir de deux droites, n'est défini que si les deux droites sont coplanaires.
$inter_{pls_1}$ et $inter_{pls_2}$ sont caractéristiques de l'augmentation de la complexité lorsqu'on passe en dimension 3. On ne peut plus vraiment parler de multi-fonction lorsqu'une construction peut donner des objets de types différents. Nous scindons cette construction en deux multi-fonctions avec des signatures différentes.

D'un point de vue sémantique, les clés sont dans la section précédente 1.2.1, décrivant l'univers euclidien 2D, et nous ne détaillons pas plus cet univers. Notons toutefois que le passage à la dimension supérieure entraîne une complexification du point de vue du nombre d'objets et du nombre de symboles mais aussi et surtout du point de vue des multi-fonctions et de l'expression de fonctions à co-arité variable.

1.2.2 Univers analytique

L'univers analytique permet de s'affranchir de la notion de dimension qui régit l'univers euclidien. L'élément de base de l'univers analytique, commun à tous les objets et aux contraintes de l'univers euclidien, est la *coordonnée*.

Univers Analytique
sortes :
elem
symboles fonctionnels :
$0 : \rightarrow$ elem
succ : elem $\rightarrow$ elem
_ + _ : elem elem $\rightarrow$ elem
_ − _ : elem elem $\rightarrow$ elem
_ × _ : elem elem $\rightarrow$ elem
_/_ : elem elem $\rightarrow$ elem
_^_ : elem elem $\rightarrow$ elem
symboles prédicatif :
_ = _ : elem elem
_ < _ : elem elem
_ > _ : elem elem
_ ≤ _ : elem elem
_ ≥ _ : elem elem

axiomes :
$\forall(x, y : elem)\ x + y = y + x$
$\forall(x, y : elem)\ x \times y = y \times x$
$\forall(x : elem)\ x + 0 = 0\ \forall(x : elem)\ x \times 1 = 1$
$\forall(x : elem)\ \exists(y : elem) \mid x + y = 0$
$\forall(x : elem) \mid x \neq 0 \Rightarrow \exists(y : elem) \mid x \times y = 1$
$\forall(x, y, z : elem)\ x + (y + z) = (x + y) + z$
$\forall(x, y, z : elem)\ x \times (y \times z) = (x \times y) \times z$
$\forall(x, y, z : elem)\ x \times (y + z) = x \times y + x \times z = (y + z) \times x$

La sémantique généralement associée à cet univers est celle des nombres réels $\mathbb{R}$ ou complexes $\mathbb{C}$. Mais, elle peut tout aussi bien être celle des nombres rationnels $\mathbb{Q}$ ou d'extensions de nombres rationnels par des radicaux $\mathbb{Q}[\sqrt{2}]$, *etc.*

Cet univers représente donc un corps $\mathbb{K}$ muni des 5 opérations classiques : l'addition, la soustraction, la multiplication, la division et la puissance. Les axiomes identifient classiquement la commutativité, l'existence d'un élément neutre et d'un inverse, ainsi que l'associativité de l'addition et de la multiplication. On peut aussi noter la distributivité de la multiplication pour l'addition.

Les symboles prédicatifs permettent de comparer ces nombres entre eux et notamment le symbole $=$ qui permet de définir des systèmes d'équations à n variables. La plupart du temps, les équations considérées sont algébriques, *i.e.* on considère l'anneau des polynômes à n variables sur les réels : $\mathbb{R}[X_1, \ldots, X_n]$.

La sorte *point* de l'univers euclidien 2D est représentée dans l'univers analytique par un couple de coordonnées.
Les contraintes géométriques sont quant à elles associées à des systèmes d'équations. Ces systèmes peuvent s'exprimer en utilisant uniquement les opérations précédentes.

Les énoncés sont généralement donnés sous forme de termes dans l'univers euclidien. Pour utiliser des solveurs algébriques, les énoncés doivent être traduits dans l'univers analytique. Cela est fait au moyen d'une fonction de morphisme d'univers.

En 2D, cette fonction associe :
- *longueur* et $elem$
- *point* et $(elem, elem)$
- *cercle* et $(elem, elem, elem)$
- ...
- $dist_{pp}$: *point point longueur* et une équation $(x_1 - x_2)^2 + (y_1 - y_2)^2 = k^2$ avec $x_1, x_2, y_1, y_2, k : elem$,
- ...

En 3D, cette fonction associe :
- *longueur* et $elem$
- *point* et $(elem, elem, elem)$
- *sphere* et $(elem, elem, elem)$

– ...
– $dist_{pp}$: *point point longueur* et une équation $(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2 = k^2$ avec $x_1, x_2, y_1, y_2, z_1, z_2, k : elem$,
– ...

Il faut noter que la notion de dimension n'a pas complètement disparue, elle est notamment présente avec le nombre de sortes *elem* que l'on associe à la sorte *point* de n'importe quel univers géométrique.

1.2.3 Univers combinatoire

Dans les méthodes combinatoires, la géométrie est abstraite en un graphe où les degrés de liberté et de restriction sont des pondérations des sommets et des arêtes.

Cette tentative de compilation de la géométrie euclidienne a permis de caractériser grossièrement de manière combinatoire la notion de constriction d'un système de contraintes à l'aide de la notion de rigidité. En fait, cette caractérisation ne fonctionne pas dans le cadre général. Elle est néanmoins utilisée comme une heuristique pour détecter la constriction structurelle générique dans beaucoup de méthodes de résolution.

Univers Combinatoire
sortes :
graphe
sommet
arête
scalaire
symboles fonctionnels :
ddl : sommet → scalaire
ddr : arête → scalaire
setDdl : sommet scalaire → sommet
setDdr : arête scalaire → arête
addS : graphe sommet → graphe
addA : graphe arête → graphe
adj : graphe sommet arête → graphe

La sorte *sommet* dénote les sommets d'un (hyper-)graphe, *arête* dénote ses (hyper-)arêtes et la relation *adj* est la spécification de l'adjacence d'un sommet et d'une arête : c'est ce qu'on appelle un (hyper-)*graphe de contraintes*. Les symboles fonctionnels *ddl* et *ddr* sont des pondérations des sommets et des arêtes que l'on nomme degrés de liberté et degrés de restriction. Ces pondérations représentent intuitivement les mouvements et les blocages possibles de ces objets, ou plus grossièrement le nombre de coordonnées des entités géométriques et le nombre d'équations à inconnues réelles des contraintes.

Ici encore, un morphisme permet de passer de l'univers euclidien à l'univers combinatoire dans lequel les degrés de liberté associés aux objets et les degrés de restriction associés aux contraintes sont les seules informations conservées. C'est pour cette raison que les caractérisations combinatoires ne sont vérifiées que dans un cadre générique restreint comme nous allons le voir dans la section suivante.

1.3 Classification

Nous proposons une classification des principales méthodes de résolution selon l'univers géométrique sur lequel elle repose. Notre classement s'appuie en premier lieu sur la dimension de l'univers et en second lieu sur la complexité de l'univers, *i.e.* le nombre d'objets et de contraintes considérés.

1.3.1 Univers à 2 dimensions

La plupart des travaux en résolution de contraintes sont situés dans des univers 2D restreints pour permettre soit l'utilisation d'une propriété limitée à un sous-groupe d'objets ou de contraintes, soit pour favoriser l'efficacité de l'algorithme proposé.

Univers des points et des distances

L'univers géométrique le plus simple est celui qui concerne les points et les contraintes de distances :

Univers PointsDistances2D
sortes :
point
longueur
symboles fonctionnels :
$point_{ppll}$: point point longueur longueur $\rightarrow$ point
$fixe$: point longeur $\rightarrow$ point
symboles prédicatifs :
$dist_{pp}$: point point longueur
$egal_{ll}$: longueur longueur
axiomes :
$\forall\ (p_1, p_2 : \text{point}) \wedge (\text{k} : \text{longueur})\ dist_{pp}(p_1, p_2, k) = dist_{pp}(p_2, p_1, k)$
$\forall\ (p_1, p_2 : \text{point})\ dist_{pp}(p_1, p_2, 0) \Leftrightarrow p_1 = p_2$
$\forall\ (p_1, p_2, p_3 : \text{point})\ dist_{pp}(p_1, p_2, k_1) \wedge dist_{pp}(p_2, p_3, k_2) \wedge dist_{pp}(p_1, p_3, k_3)$
$\quad \Rightarrow k_3 \leq k_1 + k_2$

Pour cette signature, il existe pourtant une caractérisation combinatoire de la rigidité structurelle en 2D :

Theorème 1.3.1 Théorème de Laman Un système de contraintes géométriques 2D constitué de points et de distances est structurellement rigide si et seulement s'il vérifie $2n - 3 = c$ et pour chaque sous-système $2n - 3 \geq c$ avec n le nombre de points et c le nombre de contraintes de distances génériques, en particulier non nulles. □

Un SCG structurellement rigide est bien-contraint *modulo* un déplacement. Il suffit de fixer un repère 2D qui enlève 3 degrés de liberté.

La condition de Laman en théorie de la rigidité signifie intuitivement qu'il y a exactement le nombre de contraintes nécessaires et qu'aucun sous-système n'est sur-contraint.

Dans les univers 2D et 3D suivants, l'extension de cette condition aux objets et contraintes de degrés supérieurs est nécessaire dans le cadre générique mais n'est pas suffisante pour caractériser la rigidité.

Dans ce cadre très simple, un algorithme de propagation des degrés de liberté peut être mis en œuvre comme dans l'exemple suivant :

▷ **Exemple 1.3.1**

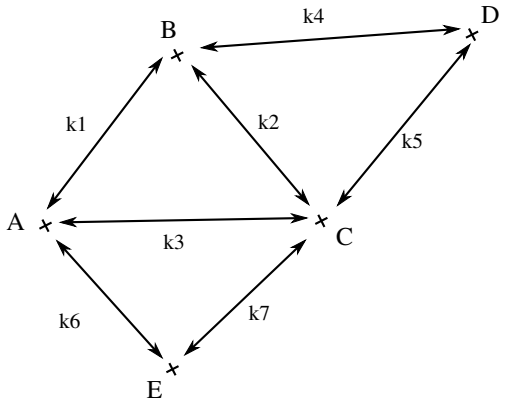


FIG. 1.8 – Une figure dans l'univers PointsDistances2D

Le SCG associé à la figure 1.8 est :

$$S = \langle \begin{cases} dist_{pp}(A, B, k1) \\ dist_{pp}(B, C, k2) \\ dist_{pp}(A, C, k3) \\ dist_{pp}(B, D, k4) \\ dist_{pp}(C, D, k5) \\ dist_{pp}(A, E, k6) \\ dist_{pp}(C, E, k7) \end{cases} , \{A, B, C, D, E\}, \{k_1, k_2, k_3, k_4, k_5, k_6, k_7\} \rangle$$

Il est bien-contraint *modulo* les déplacements. Le plan de construction équivalant à fixer un repère dans notre univers est :

```
A=cPoint()
B=fixe(A,k1)
```

La règle géométrique à appliquer pour résoudre le SCG est :

$$\text{Si } dist_{pp}(p_1, p_2, k_1) \wedge dist_{pp}(p_1, p_3, k_2) \text{ alors } p1 = point_{ppll}(p_1, p_3, k_1, k_2)$$

Après trois applications, le plan de construction final est donc :

```
A=cPoint()
B=fixe(A,k1)
C=point_ppll(B,A,k2,k3)
```

```
D=point_ppll(B,C,k4,k5)
E=point_ppll(A,C,k6,k7)
```

◁

Cette méthode est adaptée pour les SCG bien-contraints les plus simples. Mais elle ne permet pas de résoudre des systèmes dits indécomposables comme celui de la figure 1.9. Pire encore, elle ne permet pas de résoudre tous les systèmes vérifiant la condition de Laman.

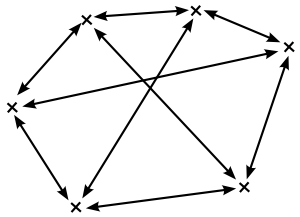


FIG. 1.9 – Hexagone bien-contraint irréductible

Univers bi-1D

Fitzgerald *et al.* [Fit81] proposent de ne considérer que les distances horizontales et verticales entre droites en 2D et les distances entre elles.

Univers Bi1D
sortes :
hor
ver
longueur
symboles fonctionnels :
$propage_{hor}$: hor longueur $\rightarrow$ hor
$propage_{ver}$: ver longueur $\rightarrow$ ver
symboles prédicatifs :
$dist_{hor}$: hor hor longueur
$dist_{ver}$: ver ver longueur
axiomes :
$\forall\ (d_1, d_2 : \text{hor}) \wedge (\text{k} : \text{longueur})\ dist_{hor}(d_1, d_2, k) = dist_{hor}(d_2, d_1, k)$
$\forall\ (d_1, d_2 : \text{ver}) \wedge (\text{k} : \text{longueur})\ dist_{ver}(d_1, d_2, k) = dist_{ver}(d_2, d_1, k)$
$\forall\ (d_1, d_2, d_3 : \text{hor}) \wedge (k_1, k2, k3 : \text{longueur})\ dist_{hor}(d_1, d_2, k_1) \wedge dist_{hor}(d_2, d_3, k2) \wedge$
$dist_{hor}(d_1, d_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (d_1, d_2, d_3 : \text{ver}) \wedge (k_1, k2, k3 : \text{longueur})\ dist_{ver}(d_1, d_2, k_1) \wedge dist_{ver}(d_2, d_3, k2) \wedge$
$dist_{ver}(d_1, d_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$

Les sortes *hor* et *ver* représentent ainsi les droites horizontales et verticales et $dist_{hor}$ et $dist_{ver}$ dénotent respectivement les distances entre droites horizontales et entre droites verticales. Les deux derniers axiomes sont l'expression de la relation de Chasles pour les distances entre chaque type de droite.

La propagation des distances s'effectue dans un univers à 1 dimension pour les droites horizontales ($propage_{hor}$) et verticales ($propage_{ver}$). Cet univers très restreint permet de produire un solveur simple et efficace (voir chapitre 2) pouvant être utilisé

pour la spécification de lignes « porteuses » en conception architecturale. Ce solveur ne contient que deux règles :
Si $dist_{hor}(d_1, d_2, k)$ alors $d_1 = propage_{hor}(d_2, k)$
Si $dist_{ver}(d_1, d_2, k)$ alors $d_1 = propage_{ver}(d_2, k)$

Dans cet univers, un repère est constitué d'une droite horizontale et d'une droite verticale. Et la condition pour qu'un tel système soit bien contraint est que la formule $n - 1 = c$ soit respectée pour chaque type de droite avec n le nombre de droites et c le nombre de contraintes. Malheureusement, cette méthode n'est pas extensible aux univers plus complexes.

Univers des bi-points

On peut étendre la signature de l'univers *PointsDistances2D* pour prendre en compte la contrainte d'angle entre bi-points :

Signature BiPoints
étend PointsDistances2D
sortes :
angle
symboles prédicatifs :
angle : point point point point angle

Méthode de Sunde L'univers des solveurs commerciaux de Sunde : PICME [Sun86] et les CD-sets et CA-sets [Sun87], est celui des bi-points. La résolution est basée sur des règles d'assemblage simples gérées par un système expert dans son premier solveur et sur des structures de données *ad-hoc* dans le second.

Cette mise en avant de l'assemblage est caractéristique des méthodes procédant par décomposition constructive.

Ainsi, dans le deuxième solveur, deux structures de données sont distinguées :
- les CD-sets (a. Constrained distance sets), regroupant des points dont les coordonnées sont déterminées dans un repère lié aux contraintes de distances,
- les CA-sets (a. Constrained angle sets), regroupant les bi-points reliés par des contraintes d'angle.

Ces deux structures représentent la rigidité des points et contraintes de distances pour les CD-sets, et la rigidité des directions pour les CA-sets.

Les CD- et CA-sets sont construits et assemblés de manière incrémentale au fur et à mesure de la pose des contraintes sur l'esquisse. Cette construction se fait en suivant des règles générales (*cf.* axiomes ci-dessous) permettant de créer des composantes rigides de plus en plus grandes, exprimées par des coordonnées dans un repère local. À notre connaissance, c'est l'une des premières méthodes à tirer systématiquement parti de l'invariance par déplacement de certaines parties de la figure pour résoudre un problème de contraintes.

Cette invariance est capturée par les règles qui assemblent des CD-sets s'il existe un repère pour les déplacements en commun et pour les CA-sets s'il existe un repère

pour les similitudes, le groupe de transformations conservant les rapports de distance.

Univers Sunde
Étend BiPoints
sortes :
CD
CA
symboles fonctionnels :
$addC$: point point CD $\rightarrow$ CD
$addA$: point point CA $\rightarrow$ CA
$joinC$: CD CD $\rightarrow$ CD
$joinA$: CA CA $\rightarrow$ CA
symboles prédicatifs :
app_{CD} : point CD
app_{CA} : point CA
axiomes :
`Règles de formation des CD et CA sets :`
$\forall(p_1, p_2 : point)(k : longueur)\ dist_{pp}(p_1, p_2, k) \supset addC(p_1, p_2, cCD)$
$\forall(p_1, p_2, p_3, p_4 : point)(a : angle)\ angle(p_1, p_2, p_3, p_4, a) \supset$
$addA(p_1, p_2, p_3, p4, cCA)$
`Règles de jointures des CD et CA sets :`
$\forall(p_1, p_2, p_3, p_4, p_5 : point)(a : angle)(cd_1, cd_2 : CD)\ app_{CD}(p_1, cd_1) \wedge$
$app_{CD}(p_1, cd_2) \wedge app_{CD}(p_2, cd_1) \wedge app_{CD}(p_3, cd_1) \wedge app_{CD}(p_4, cd_2) \wedge$
$app_{CD}(p_5, cd_2) \wedge angle(p_2, p_3, p_4, p_5, a) \supset joinC(cd_1, cd_2)$
$\forall(p_1, p_2, p_3, p_4 : point)(a : angle)(ca_1, ca_2 : CA)\ app_{CA}(p_1, ca_1) \wedge$
$app_{CA}(p_2, ca_1) \wedge app_{CA}(p_3, ca_2) \wedge app_{CA}(p_4, ca_2) \wedge angle(p_1, p_2, p_3, p_4, a) \supset$
$joinA(ca_1, ca_2)$

Les valeurs numériques sont calculées au fur et à mesure et les règles sont codées « en dur », mais c'est véritablement une méthode basée sur des connaissances géométriques comme le montre l'implantation sous forme de système expert par Verroust [Ver90].

Par ailleurs, cette méthode permet de tester la sur-rigidité durant la saisie des contraintes : si un utilisateur tente d'ajouter une contrainte dans un CD-set, le solveur le détecte et peut le signaler à l'utilisateur indiquant ainsi le CD-set à corriger.

La méthode de base de Sunde ne permet pas de résoudre tous les problèmes 2D constitués de *motifs* de triangles et de quadrilatères. Il faut pour cela avoir recours à un ensemble de règles plus important comme l'ont montré Verroust *et al.* [VSR92]. Dans ce travail, 4 règles permettent d'assembler toutes les configurations de triangles contraints en angles et en distances et de la même manière 6 règles permettent d'assembler toutes les configurations de quadrilatères.

Par exemple, la règle principale des triangles indique que lorsqu'un point et un angle sont partagés par deux CD-sets, alors on peut les regrouper au sein d'un même CD-set (*cf.* le troisième axiome).

La relation de Chasles est prise en compte par les règles œuvrant sur les CA-sets.

Malheureusement, une implantation sous forme de système expert [VSR92] ou de règles de réécriture [ARMP02], fait perdre l'efficacité en temps de la méthode initiale, due à son aspect numérique et à ses règles codées en dur.

Lorsqu'il ne reste plus qu'un seul CD-set, c'est-à-dire une composante rigide pour tout le SCG, le problème est résolu. Si ce n'est pas le cas, cela signifie, soit que le solveur n'est pas assez puissant pour décomposer le problème, soit que celui-ci est sous-contraint.

Une étude exhaustive des configurations 2D avec 2, 3, 4 points (*cf.* figure 1.10) montre que les règles précédentes permettent de résoudre tous les cas bien contraints (laissé à la sagacité du lecteur). L'hexagone de la figure 1.9 est vraisemblablement le problème le plus simple non soluble par la méthode de Sunde.

Univers des points et des droites 2D et extension aux cercles 2D

On considère souvent en CAO des univers géométriques contenant des points et des droites. La signature d'un tel univers est basée sur la signature BiPoints, à laquelle s'ajoute la sorte supplémentaire *droite*.

Considérer la sorte *droite* permet d'enrichir le cadre, et aussi de simplifier parfois l'écriture de contraintes, notamment celle de contraintes d'angle. Avec cette sorte, de nouveaux types de contraintes sont ajoutés comme l'appartenance d'un point à une droite, ou le parallélisme de deux droites :

Signature PointsDroites2D
étend PointsDistances2D
sortes :
angle
droite
symboles fonctionnels :
$droite_{pp}$: point point → droite
$inter_{dd}$: droite droite → point
symboles prédicatifs :
app_{pd} : point droite
$parall$: droite droite
$angle$: droite droite angle

Nous avons déjà mentionné le fait que le théorème de Laman ne s'applique pas dans des univers contenant autre chose que des points et des distances génériques en dimension 2. L'introduction de droites et d'angles permet ainsi de trouver de nombreux contre-exemples à une extension du théorème de Laman dans cet univers.

Examinons ainsi le cas illustré à la figure 1.11 : avec 5 points, 4 droites, 4 distances, 10 incidences points-droites et un parallélisme, le critère est vérifié. Pourtant, cette figure n'est pas rigide mais localement sur-contrainte puisqu'on peut utiliser le théorème de Thalès $\frac{AM}{AB} = \frac{MN}{BC}$ pour trouver la contrainte de distance AM déjà spécifiée.

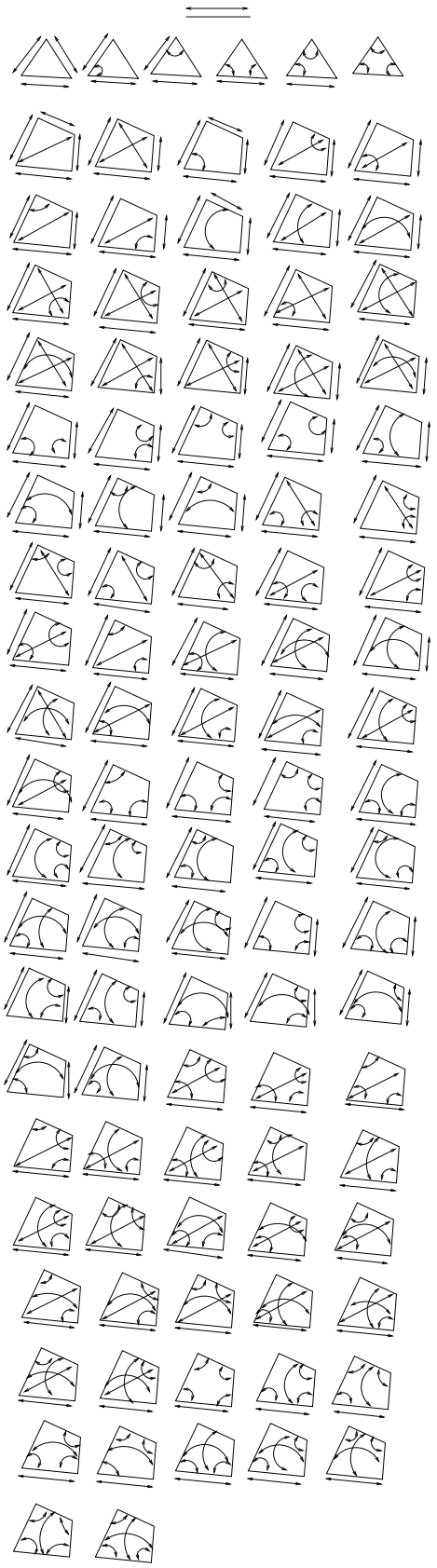


FIG. 1.10 – Les configurations de base en 2D pour les triangles et les quadrilatères dans l'univers des bi-points

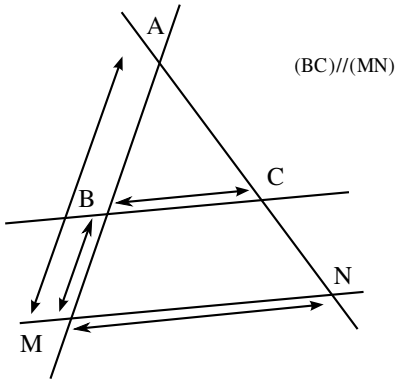


FIG. 1.11 – Exemple non-rigide mais vérifiant la caractérisation de Laman

Dans cet univers, on peut appliquer une méthode qui comme la méthode de Sunde s'intéresse à la présence de repères : la méthode d'Owen, dont l'applicatin permet de décomposer *a priori* suivant des repères pour les déplacements, matérialisés par des paires d'articulation.

Méthode d'Owen La méthode d'Owen [Owe91] procède de manière descendante pour décomposer un système de contraintes géométriques. L'heuristique de décomposition est basée sur la recherche de paires d'articulations dans un graphe de contraintes.

Univers Owen
Étend Combinatoire
sortes :
paire
arêtev
symboles fonctionnels :
$cherchePaire$: graphe → paire
$coupe_1$: graphe paire → graphe
$coupe_2$: graphe paire → graphe
$ajouteVirtuelle$: graphe arêtev → graphe

Une paire d'articulations est une paire de sommets du graphe de contraintes telle que le retrait de ses 2 sommets et des arêtes incidentes déconnecte le graphe en deux sous-graphes.

Le découpage au niveau des paires d'articulations implique la duplication des sommets. Il en résulte qu'un des sous-graphes n'est pas rigide. Le solveur doit alors ajouter un *lien virtuel* dans chaque sous-graphe non rigide après scission. Ce lien virtuel est généralement une distance entre les 2 sommets d'articulation (*cf.* figure 1.12).

Une décomposition récursive est appliquée et on obtient des composantes irréductibles qui sont, soit des configurations triangulaires, soit des composantes tri-connexes plus complexes.

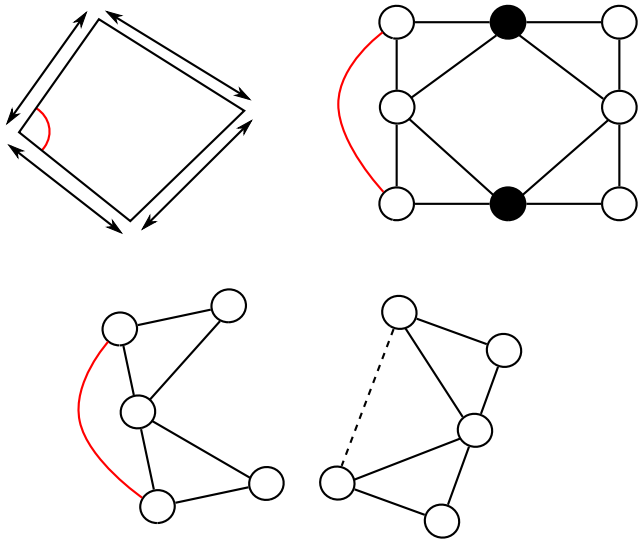


FIG. 1.12 – Exemple de paire d'articulation (en noir) et de lien virtuel (en pointillé)

La résolution des sous-graphes se fait alors dans un ordre qui dépend de l'adjonction des arêtes virtuelles : les composantes tri-connexes n'en contenant pas sont d'abord résolues, puis les valeurs des liens virtuels des sous-graphes adjacents sont déduites de ces configurations, ce qui permet de les résoudre à leur tour, et ainsi de suite. L'assemblage se fait dans l'ordre inverse.

Arinyo *et al.* [ARMP02] ont montré que les SCG résolubles par la méthode d'Owen sont des SCG dont on peut trouver récursivement 3 sous-graphes partageant deux à deux un objet : des composantes tri-connexes.

Remarque
Contrairement à la méthode de Sunde, la méthode d'Owen est limitée aux cas bi-connectés. Même si la méthode de Sunde est dépendante de la représentation sous forme de CA- et de CD-sets, la règle du parrallèlogramme permet de résoudre des problèmes représentés par un graphe tri-connecté.

La méthode d'Owen a pourtant été implantée avec succès dans le solveur commercial *D-Cubed* [DC]. Elle est ainsi intensivement utilisée en CAO.

Univers de graphes L2L1

On se rend compte si on étudie les fonctions de passage de l'univers *Points-Distances2D* vers l'univers combinatoire que les objets de degré de liberté égal à 2 partagent des contraintes de degré 1. Ceci permet d'avoir recours à des algorithmes à base de graphes comme par exemple celui d'Owen.

On remarque aussi qu'il est possible d'ajouter la sorte *cercle*. Ces objets sont courament rencontrés en CAO, mais ils doivent être considérés avec leur rayon fixé

pour entrer dans ce cadre.

Univers Cercles2D
étend signature PointsDistances2D
sortes :
cercle
symboles fonctionnels :
$inter_{dc}$: droite cercle → point
$point_{ppll}$: point point longueur longueur → point
$droite_{da}$: droite angle → droite
$droitePar_{dp}$: droite point → droite
$droitePerp_{dp}$: droite point → droite
symboles prédicatifs :
app_{pc} : point droite
$tang_{dc}$: droite cercle

L'univers *Cercles2D* peut donc être relié avec l'univers des graphes *L2L1* présenté ci-dessous :

Univers L2L1
étend Univers Combinatoire
axiomes :
$\forall\ (s : sommet)\ ddl(s) = 2$
$\forall\ (a$:arête$)\ ddr(a) = 1$

Méthode de Fudos et Hoffmann La méthode décrite par Fudos et Hoffmann [BFH+95, FH97] est aussi une méthode de décomposition constructive travaillant sur un graphe de contraintes binaires.

C'est une méthode ascendante, comme celle de Sunde, qui a la particularité de traiter les SCG bien, sur- et sous-contraints en ajoutant un test de cohérence des redondances à la résolution et en permettant de rajouter des contraintes à la volée dans une phase descendante pour rendre le système bien-rigide.

Univers FH
étend Univers L2L1
sortes :
grappe
symboles fonctionnels :
$cCluster$: sommet sommet → grappe
$propage$: sommet grappe → grappe
$assemble$: grappe grappe → grappe
ddr_{sc} : sommet grappe → scalaire
axiomes :
`Formation de grappes :`
$\forall(s_1, s_2 : sommet)(a$:arête$)\ \exists(c : grappe)\ adj(s_1, a) \wedge adj(s_2, a) \supset c = cCluster(s_1, s_2)$
`Propagation des degrés :`
$\forall\ (s : sommet)(c : grappe)\ ddl(s) = ddr_{sc}(s, c)\ c = propage(s, c)$

`Règles d'assemblage :`
$\forall\ (s_1, s_2, s_3 : sommet)(c_1, c_2 : grappe)$ (a :arête) $app(s_1, c_1) \wedge app(s_1, c_2) \wedge app(s_2, c_1) \wedge app(s_3, c_2) \wedge adj(s_2, a) \wedge adj(s_3, a) \supset c_3 = assemble(c_1, c_2)$
$\forall\ (s_1, s_2, s_3 : sommet)(c_1, c_2, c_3, c_4 : grappe)\ app(s_1, c_1) \wedge app(s_1, c_2) \wedge app(s_2, c_1) \wedge app(s_2, c_3) \wedge app(s_3, c_3) \wedge app(s_3, c_2)$
$\supset c_4 = assemble(assemble(c_1, c_2), c_3)$
$\forall\ (s_1, s_2, s_3 : sommet)(c_1, c_2, c_3, c_4 : grappe)\ app(s_1, c_1) \wedge app(s_1, c_2) \wedge app(s_2, c_1) \wedge app(s_2, c_3) \wedge app(s_3, c_3) \wedge app(s_3, c_2)$
$\supset c_4 = assemble(assemble(c_1, c_2), c_3)$

La méthode procède par assemblage en agrégats structurellement rigides : des *grappes* ou « clusters ». La formation initiale des grappes se fait par une phase de propagation des degrés de liberté suivant la règle : $\sum ddl = \sum ddr$. Cette règle correspond dans l'univers de dimension 2 à une triangularisation (au sens des systèmes d'équations) de blocs de taille 2×2 (deux inconnues, deux équations).

Bien sûr, ces agrégats initiaux sont formés à partir de repères fixés qui correspondent à deux objets géométriques.

Trois règles d'assemblage sont considérées :

1. un sommet et une arête sont partagés entre deux grappes (*cf.* figure 1.13) ;
2. trois grappes partagent deux à deux un sommet (*cf.* figure 1.14) ;
3. deux grappes sont reliés par trois arêtes (*cf.* figure 1.15).

Cette méthode met en avant l'idée de rigidification progressive d'une figure qui n'est pas sans rappeler la méthode de Sunde avec pourtant quelques différences :

- pas de dépendance vis-à-vis des structures de données ;
- les règles considérées sont plus générales ;
- il y a une claire séparation entre résolution locale et assemblage.

Cette méthode souffre des mêmes problèmes que celle d'Owen et en particulier elle est incomplète. De plus, Schreck [Sch02] a montré que tout problème énonçable dans l'univers de Sunde et résoluble par la méthode de Fudos et Hoffmann l'est également par la méthode de Sunde.

Mais, comme pour celle d'Owen, cette méthode englobe un univers plus large où il existe des SCG non exprimables chez Sunde.

Autres méthodes D'autres méthodes s'appliquent dans cet univers, comme celle d'Ait-Aoudia *et al.* [AAM93] qui applique l'algorithme du couplage maximum au graphe de contraintes pour obtenir des composantes fortement connexes (voir section 1.3.3), ou de Trombettoni [Tro97] qui y applique une rétro-propagation des degrés de liberté en partant des feuilles du graphe qui vérifient $\sum ddl = \sum ddr$, et en itérant sur le graphe privé des éléments ainsi déterminés.

Univers euclidien 2D

Pour s'affranchir des restrictions de l'univers L2L1, des extensions des méthodes précédentes ont été proposées. Elles généralisent les notions d'articulations, en considérant des triplets d'articulations pour Owen par exemple, ou le comptage des degrés dans un graphe, comme Hoffmann *et al.* le proposent.

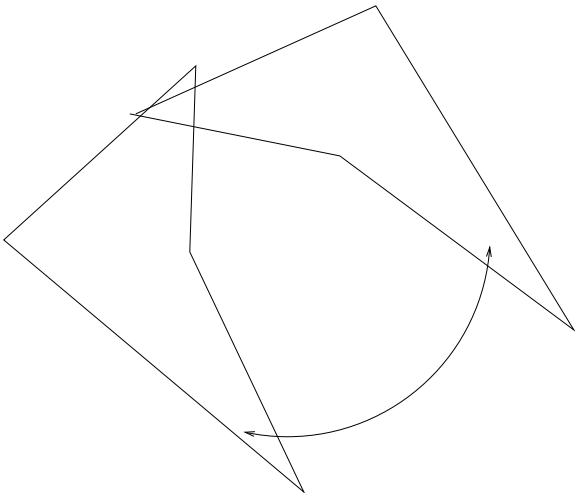


FIG. 1.13 – Règle d'assemblage 1 de BFH

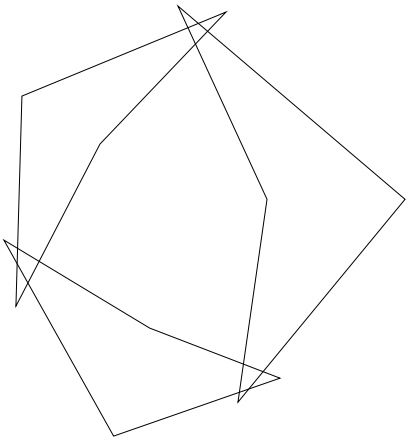


FIG. 1.14 – Règle d'assemblage 2 de BFH

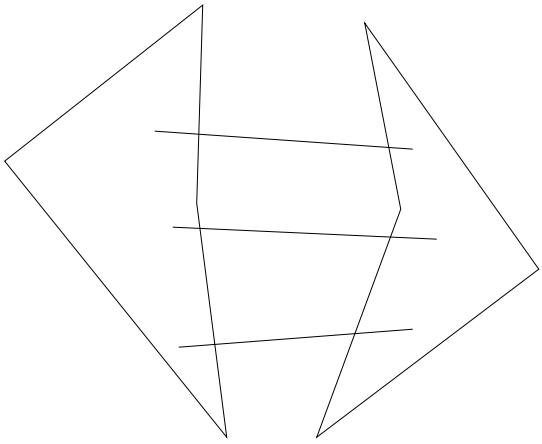


FIG. 1.15 – Règle d'assemblage 3 de BFH

D'autres voies ont aussi été explorées avec l'extension de la notion de couplage parfait par Sitharam *et al.* ou avec l'exploration fine des propriétés géométriques à l'aide de systèmes à base de connaissances.

De plus, des méthodes mixtes comme celle de Jermann *et al.* ont été proposées.

Méthode d'Hoffmann, Lomonosov et Sitharam Les extensions de la méthode précédente sont dues en particulier à Hoffmann *et al.* [HLS01b, HLS01a, HC02b, HC02a]. Elles permettent de considérer des objets avec n'importe quel degré de liberté, et introduisent la sorte cercle de rayon non-fixé.

Jermann *et al.* [CBG02, JNT03] proposent un autre critère de rigidité pour l'heuristique, de nouvelles manières de décomposer et assembler, et en utilisant du calcul par intervalles.

Ces extensions sont basées sur un cadre commun, où sont distinguées 2 phases :
- la phase de décomposition,
- et la phase d'assemblage.

Il faut noter que la phase d'assemblage n'a jamais été décrite en détail dans les travaux d'Hoffmann, mais qu'elle a été étudiée par Jermann *et al.*[Jer02].

La décomposition se partage en 3 phases répétées récursivement :
- l'opération de fusion qui consiste à constituer une grappe bien-rigide de taille minimale en utilisant un algorithme de flot,
- l'opération d'extension, qui consiste à faire grossir une grappe en incorporant les voisins suivant l'heuristique de rigidité choisie,
- l'opération de condensation, qui réalise la réduction de la grappe en 1 sommet (Condensation Algorithm - CA)

Il est intéressant de noter que deux opérateurs de condensation ont été proposés pour améliorer la correction de CA :
- Frontier Algorithm (FA) qui permet de réduire les contraintes internes à la grappe en gardant le bord,
- Modified FA, qui applique la réduction si la grappe interne définit au moins un repère local.

Méthodes à base de connaissances Les méthodes à base de connaissances [Ald88, VSR92, DMS98] se basent sur une représentation par des prédicats et des techniques d'inférence pour résoudre des SCG. Dans ces méthodes, le nombre de règles a tendance à croître exponentiellement avec le nombre de types de contraintes. Aussi, ces méthodes intègrent généralement des stratégies qui leur permettent d'éviter une explosion combinatoire du nombre de cas à envisager.

Les méthodes constructives se basent sur les connaissances et les savoir-faire des constructions géométriques issues de l'EAO [But79, Sch93].

La méthode des lieux géométriques consiste à transformer des contraintes géométriques en contraintes d'incidences sur un lieu définissant progressivement un objet cherché.

Aldefeld en CAO [Ald88] présente ce qui constitue les deux phases principales de sa méthode :

- la production d'un plan de construction, généralement à l'aide d'un système expert de complexité exponentielle,
- l'évaluation numérique, très rapide.

Cette approche a été exploitée et étendue avec :

- le choix d'une solution à partir de critères du type « point à gauche ou à droite d'une ligne »,
- la prise en compte des contraintes globales comme par exemple l'aire, avec fixation d'une inconnue puis relaxation et correction par échantillonnage,
- l'invariance par déplacements.

Brüderlin [Brü86, Brü88] utilise des règles de réécriture pour démontrer la convergence d'un ensemble de règles de constructions géométriques.

Un plan de construction obtenu par une méthode géométrique formelle équivaut à une décomposition fine suivant les objets du problème à résoudre. Mathis *et al.* [Mat97] proposent d'unifier les phases d'assemblage des méthodes de décomposition par la mise en correspondance des repères locaux par des déplacements. De plus, leur méthode basée sur la collaboration de solveurs au sein d'une architecture multi-agents permet de combiner solveurs constructifs convergents ou non et solveurs numériques.

L'efficacité de cette approche de résolution par décomposition constructive est mise en évidence au sein d'un modeleur 3D, par l'animation de mécanisme en modifiant à la volée des paramètres du plan de construction.

D'une manière générale, les méthodes constructives sont limitées par le nombre de règles considérées. Mais, elles ont l'avantage d'être correctes et complètes par rapport à cet ensemble de règles, ce qui n'est pas le cas des solveurs plus généraux à base d'heuristiques combinatoires.

1.3.2 Univers à 3 dimensions

Trouver une méthode de résolution de contraintes géométriques dans l'espace reste un enjeu majeur pour la communauté s'intéressant aux contraintes.

La complexité de cet univers est partiellement due au nombre de cas différents à considérer qui conduit dans notre formalisme à introduire des « multi »-fonctions à co-arité différentes. De plus, on ne connait pas de caractérisation simple de la rigidité comme en 2D. Enfin, il existe un grand nombre de systèmes 3D minimaux irréductibles. Par exemple, on peut citer une grande partie des polyèdres convexes décrits par des distances et des coplanarités. En effet, ils sont composés de faces la plupart du temps sous-contraintes, et d'après le théorème de Cauchy [Cau13], ils sont génériquement rigides.

Ces systèmes complexes sont habituellement directement résolus par des méthodes numériques avec leurs inconvénients bien connus comme l'instabilité numérique, et l'incapacité de trouver efficacement toutes les solutions.

Pourtant, quelques tentatives ont eu lieu pour étendre les méthodes 2D à la dimension supérieure et notamment pour considérer un univers plus simple composé d'objets de mêmes degrés de liberté, mais aussi en considérant la connexité des

graphes dans l'univers mettant en jeu points, droites et plans.

Dans ce même univers, une méthode hybride mélangeant des informations des 3 univers géométriques (euclidien, combinatoire, analytique) a été proposée par Gao *et al.* [GHY04]. Elle a été appliquée sur des exemples de tailles relativement petites et semble souffrir de désavantages très handicapants. Nous verrons au chapitre 4 comment notre travail a permis d'améliorer cette méthode.

Univers des mécanismes

Le premier univers que l'on peut considérer en 3D est celui des mécanismes qui est un univers combinatoire où il n'y a qu'un seul objet de base : un repère local à chaque pièce mécanique. Ainsi, tous les éléments ont six degrés de liberté, trois en rotation et trois en translation.

Univers Mécanismes
sortes :
marqueur
déplacement
degrés
symboles fonctionnels :
$cMarqueur$: $\rightarrow$ marqueur
ddl : marqueur $\rightarrow$ degrés
$action$: marqueur degrés $\rightarrow$ déplacement
$applique$: marqueur déplacement $\rightarrow$ marqueur

Kramer [Kra90, Kra91, Kra92] s'intéresse aux mécanismes 3D constitués de pièces mécaniques liées par des articulations. À chaque pièce, la méthode présentée associe un marqueur, qui est un repère local. Les contraintes liant ces marqueurs sont l'expression des articulations sous la forme de degrés de restriction représentant :

- les translations,
- les rotations,
- les dimensions.

Une analyse combinatoire de ces degrés permet de planifier un ordre de résolution pour les mécanismes rigides. La résolution effective s'effectue par l'application d'un déplacement entre marqueurs. Suivant les types de degrés liant les marqueurs, un tableau d'actions indique les déplacements à appliquer. Il y a un grand nombre d'actions possibles suivant les degrés mis en jeu et Kramer ne les décrit pas tous. C'est pour cela que nous n'en décrivons aucun dans l'axiomatique de l'univers.

Lorsqu'aucune action ne peut être validée, une analyse des lieux géométriques des marqueurs restés libres est appliquée grâce à un autre tableau indiquant comment calculer ces intersections.

Bien sûr, cette méthode souffre des mêmes inconvénients que les méthodes *a priori* mais elle permet de résoudre tous les mécanismes fermés rigides d'au plus 3 pièces mécaniques.

Univers points, droites, plans

k-connexité Dans [GZ03], les auteurs présentent une généralisation de la méthode d'Owen [Owe91] et de la reformulation d'arbre de décomposition d'Arinyo [ARMP02]. Ils proposent donc de considérer des arbres binaires de décomposition dans l'univers géométrique des points et des droites, 2D ou 3D.

L'univers géométrique ne change pratiquement pas par rapport à celui d'Owen : la sorte *paire* dénote maintenant un n-uplet d'articulations et les algorithmes de recherche et d'ajout d'arêtes virtuelles sont modifiés pour prendre en compte cette généralisation.

L'heuristique utilisée par Gao *et al.* est une extension directe du théorème de Laman :

> **Définition 1.3.2** Pour n objets et m contraintes dans un univers de dimension d, l'heuristique combinatoire de rigidité vaut :
>
> $$\sum_i^n ddl_i - \frac{d(d+1)}{2} = \sum_j^m ddr_j$$

En 3D, cette caractérisation rencontre les contre-exemples classiques des double-bananes 1.16,1.17, 1.18.

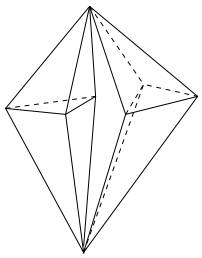


FIG. 1.16 – La double-banane

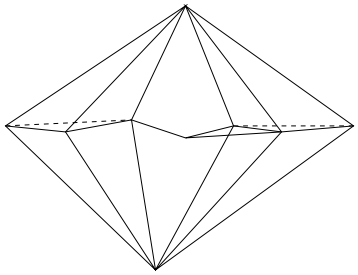


FIG. 1.17 – La double-banane à arête saillante

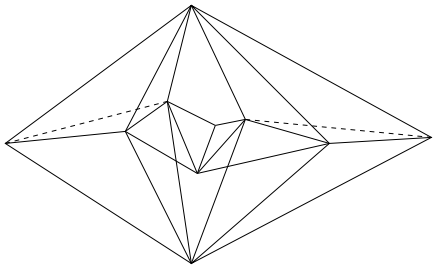


FIG. 1.18 – La double-banane accordéon.

Un algorithme, dont la complexité est en $O(v^{\frac{1}{2}}e^2)$, est proposé pour identifier qu'un graphe est k-connexe. Malheureusement, il est difficile d'établir avec certitude la k-connexité en dehors d'un cadre de généricité supposé car les pondérations de certaines arêtes peuvent être différentes suivant les valeurs des paramètres.

D'après eux, la borne de la k-connexité en 2D est $k = 3$ et en 3D, avec des points et des plans, $k = 5$. Si les droites sont considérées en plus, alors $k = 7$.

Les n-uplets d'articulation sont détectables avec l'algorithme de Hopcroft et Tarjan [HT73] pour les paires et la tri-connectivité en $O(V + E)$, et l'algorithme de Kanevsky et Ramachandran [KR87] avec les triplets et la 4-connectivité.

La décomposition se fait par duplication des n-uplets dans les 2 parties. La condition d'arrêt est d'avoir 3 primitives dans les sous-graphes ou des sous-systèmes sans n-uplet d'articulations.

Suivant la valeur de la fonction de déficit d'un graphe, introduite par Arinyo *et al.* [ARMP02], un découpage en 2 parties bien contraintes peut généralement être appliqué. Lorsqu'il n'existe qu'une partie bien-contrainte, l'ajout du bord de la figure à la partie sous-contrainte permet de la rendre bien-contrainte. Si aucune partie n'est bien-contrainte, une autre décomposition doit être appliquée. Dans le cas où aucune décomposition de ce type n'est possible un solveur numérique ou la méthode LIMd [GHY04] présentée au paragraphe suivant, doit être appliqué.

Les algorithmes présentés sont de complexités polynomiales de l'ordre de $O(V^2)$ en 2D, et $O(V^7)$ en 3D. Mais ils montrent surtout les limites d'une telle approche, en étudiant tous les cas de feuilles à 3 primitives et en montrant que certains cas ne sont pas rigides alors qu'ils sont structurellement rigides.

Méthode d'intersection des lieux géométriques L'univers géométrique de cette méthode est bien difficile à spécifier. Il conjugue les 3 univers géométriques et nécessite de nombreuses transformations et explications. Nous ne nous lançons pas dans sa description pour ne pas embrouiller le lecteur.

La méthode de Gao *et al.* [GHY04] consiste à générer automatiquement tous les problèmes contenant 6 contraintes avec des points, des droites et des plans.

Pour la catégorie des systèmes minimaux non décomposables, il propose de modifier le système de contraintes S en un système décomposable S'_k, soit en *oubliant* une inconnue, soit en substituant une contrainte par une autre, et en ajoutant un paramètre k dans les deux cas. Les deux systèmes ne sont pas équivalents mais

toutes les solutions de S peuvent être retrouvées en échantillonnant le paramètre k, et en calculant toutes les solutions de S'_k pour chaque valeur de k et en choisissant la meilleure solution approximée de S. Cette méthode est une combinaison de méthodes numériques et de méthodes constructives. Elle permet de trouver toutes les solutions et semble robuste. Mais elle s'applique à de petits systèmes et n'effectue que des modifications limitées du système initial.

L'idée est assez simple : considérons un système S qui est décomposable à la condition d'oublier une inconnue x_1. Convertir x_1 en paramètre permet de résoudre le problème mais celui-ci devient alors sur-contraint. Il faut donc retirer une contrainte pour avoir un problème bien-contraint.

Leur méthode consiste à produire une séquence de construction grâce à un algorithme à heuristique combinatoire présenté dans la figure 1.19.

Lorsque le critère combinatoire n'est pas satisfait, il suffit alors de rechercher des contraintes à enlever pour le rendre applicable. La recherche du nombre de contraintes minimum à enlever est faite de manière exhaustive.

Le même nombre de contraintes est ajouté à l'aide de la séquence de construction par ce qu'ils appellent des « paramètres guidant la résolution ».

Finalement, l'échantillonnage des paramètres des contraintes ajoutées permet de déformer la figure. Une étude de l'intersection des lieux géométriques permet de trouver les solutions du système initial.

Les problèmes que l'on peut souligner sont les suivants : premièrement, l'utilisation d'une heuristique combinatoire conduit à ce que leur séquence de constructions est incorrecte dans le cadre général (*cf.* contre exemple suivant 1.3.2). Deuxièmement, la recherche exhaustive du nombre de contraintes à enlever est très coûteuse : $O(n^d)$ pour d contraintes. En comparaison, l'algorithme de Newton est en $0(n^3)$. Troisièmement, l'échantillonnage doit se faire en choisissant le bon intervalle $[a, b]$ et un pas s satisfaisant : la complexité est $O(L\frac{b-a}{s})^d$.

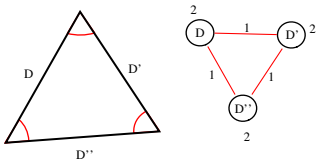
LIM0 Algorithm
Input : a constraint problem.
Output : a construction sequence CS with initial value $\emptyset$

1. Let o be a primitive in the problem satisfying $d = DEG(o) \leq DOF(o)$.
2. Let $o_1, \ldots, o_d$ be the primitives having constraints with o.
   Then add a construction $o = \text{INTER}(o_1, \ldots, o_d)$ to CS.
   Terminate if no such o exists.
3. Remove o and the constraints involving o from the problem and go to Step 1.

FIG. 1.19 – Algorithme tiré directement de [GHY04]

▷ **Exemple 1.3.2**
L'utilisation du critère $d = \text{DEG}(o) \leq \text{DOF}(o)$ dans l'algorithme de Gao et *al.* est

valide dans un cadre générique mais pas dans le cadre général. Prenons le contre-exemple bien connu, où la dépendance algébrique est introduite par la relation de Chasles dans un triangle : la somme des angles valant 180°, si on connaît deux angles, on peut déduire la valeur du troisième. Ainsi, un triangle défini par 3 angles remplit la condition de Laman sans être rigide.



Le degré de liberté d'une droite est 2, et le degré de restriction d'une contrainte d'angle est 1. On obtient alors la séquence de construction suivante :

```
D = INTER(D',D'')
```

◁

1.3.3 Univers à n dimensions

La représentation par des équations permet de s'affranchir de la dimension. Les méthodes de résolution équationnelle sont, soit des techniques exactes très puissantes de calcul algébrique mais dont la complexité est exponentielle (dans le meilleur des cas) - ce qui explique pourquoi elles sont peu utilisées en CAO -, soit des techniques itératives numériques efficaces mais dont la convergence numérique n'est pas garantie.

La décomposition de systèmes d'équations en sous-systèmes permet de réduire la complexité du calcul algébrique mais aussi les phénomènes d'approximations numériques. Ainsi, chaque sous-système est structurellement bien-contraint, *i.e.* il a le même nombre d'équations que de variables et aucune sous-partie n'a plus d'équations que de variables.

Les trois sections suivantes exposent ces méthodes analytiques.

Méthodes de calcul algébrique

Les méthodes de calcul algébrique sont des méthodes formelles qui manipulent des polynômes. La résolution exacte de tels systèmes fait appel à la théorie de l'élimination dans les anneaux de polynômes [Kal91]. Pour ces techniques, les calculs ne conduisent pas à une forme explicite des solutions, mais donnent une représentation du système d'équations initial dans une base différente à partir de laquelle il est possible d'extraire plus facilement les solutions.

De manière générale, les méthodes algébriques formelles traitant le cas général sont très coûteuses (exponentielles) mais permettent des calculs arithmétiques exacts. Cet avantage est souvent mis en avant par rapport aux méthodes numériques approchées dans les domaines où la précision est critique comme la robotique médicale.

Nous présentons ici deux méthodes : la méthode basée sur le calcul des bases de Gröbner qui possède les mêmes solutions que le système initial et celle de Ritt-Wu permettant de calculer un système triangulaire.

Bases de Gröbner L'algorithme de Buchberger [Buc85] permet de générer une base particulière d'un idéal, appelée base de Gröbner. La spécificité de cette base est de posséder les mêmes solutions que le système initial.

Intuitivement, le calcul des bases de Gröbner est fondé sur la réduction de polynômes à l'aide de techniques d'élimination polynomiale, et sur le calcul de résolvantes de paires de polynômes, nommées conséquences.

Cet algorithme applique successivement le calcul des conséquences et leur réduction *modulo* l'ensemble des polynômes. Si l'élimination des variables se fait dans l'ordre lexicographique, la base obtenue est quasiment triangulaire.

L'algorithme de Buchberger est coûteux en temps et en mémoire : doublement exponentiel dans le nombre de variables, dans le pire des cas.

Cette technique de résolution de systèmes d'équations a été mise en œuvre entre autres [EY88, Kon92] par L.W. Ericson et C.K. Yap pour un éditeur géométrique, LINETOOL. Dans ce système, ils utilisent des structures de données particulières permettant un calcul plus efficace. Toutefois, il ne s'agit là que d'heuristiques et la complexité de l'algorithme reste exponentielle. Cette méthode est très coûteuse mais l'accent est mis sur la précision et non l'interactivité de l'outil.

> **Remarque**
> Il est important de noter que lorsque le problème est sous-contraint (il y a une infinité de solutions) la base de Gröbner rendue est une représentation simplifiée du système initial représentant l'ensemble des solutions. Lorsque l'ajout de contraintes se fait de manière interactive, les bases de Gröbner successives représentent incrémentalement et de manière formelle les solutions du problème que l'on est en train de décrire.

Ritt-Wu Même si l'approche de W.T. Wu [Wu84], détaillée et implantée par S.C Chou [Cho88] et réutilisée par Wang [Wan02], se place dans une optique différente de la nôtre : la démonstration automatique de théorèmes de la géométrie élémentaire, elle peut s'appliquer à la résolution de contraintes géométriques.

L'algorithme mis au point par Ritt [Rit50] et redécouvert par Wu cherche une décomposition de S en ensembles caractéristiques irréductibles de polynômes, chaque ensemble contenant les polynômes générateurs d'un idéal premier dont les zéros sont des racines de S. Cette méthode procède en quatre étapes :

Pour commencer, l'énoncé est traduit en un ensemble d'équations :

$$\begin{cases} h_1(u_1, ..., u_n, x_1, ..., x_r) = 0 \\ \quad ... \\ h_r(u_1, ..., u_n, x_1, ..., x_r) = 0 \end{cases}$$

où $u_1, ..., u_n$ sont des variables indépendantes ou paramètres, $x_1, ..., x_r$ sont des inconnues qui dépendent des premières et $h_1, ..., h_r$ des polynômes de $\mathbb{R}[u_1, ..., u_n, x_1, ..., x_r]$. Les variables $x_1, ..., x_r$ peuvent représenter des coordonnées de points, des longueurs, des valeurs d'angles etc. Le théorème à démontrer doit pouvoir être exprimé par une équation du type :

$$g(x_1, ..., x_r) = 0$$

Dans la deuxième étape, le système est mis sous forme triangulaire par un algorithme mettant en jeu la pseudo-division des polynômes h_i. On obtient alors un système de la forme :

$$\begin{cases} f_1(u_1, ..., u_n, x_1) = 0 \\ ... \\ f_r(u_1, ..., u_n, x_1, ..., x_r) \end{cases}$$

où les f_i sont des polynômes.

La troisième étape consiste à (pseudo-)diviser successivement g par $f_r, ..f_1$ où chaque f_i est considéré comme un polynôme en x_i. Le résultat de ces divisions s'exprime par une équation de la forme :

$$I_1^{s_1} ... I_r^{s_r} g = q_1 f_1 + ... + q_r f_r + R$$

où R est le reste de la division de g par les f_i, les s_i sont des entiers positifs ou nuls, les q_i sont des polynômes et les I_i les cœfficients de plus haut degré des f_i par rapport à x_i. Si R est nul et si $I_i \neq 0$ pour tout $i = 1, ..., r$, alors la propriété exprimée par g est vérifiée.

La quatrième phase est une discussion où les conditions de non dégénérescence $I_i \neq 0$ sont interprétées géométriquement et analysées.

Cette méthode a été mise en œuvre en liaison avec la méthode de Lebesgue pour résoudre algébriquement des problèmes de constructions à la règle et au compas [Che92].

Chou [Cho88] utilise cette méthode pour prouver de nombreux théorèmes classiques de la géométrie comme ceux de Pappus (*cf.* figure 1.20), de Simson, de Desargues, *etc.*. Pourtant sa complexité exponentielle et la discussion de la quatrième phase en limite l'utilisation dans le domaine de la CAO.

Méthodes numériques

Les méthodes numériques sont des méthodes itératives qui à partir d'une approximation initiale des valeurs des inconnues permettent de calculer de meilleures approximations. Les équations peuvent être transcendantes, *i.e.* non-algébriques.

Ces méthodes sont très rapides mais aussi sujettes aux instabilités numériques inhérentes à la représentation des nombres réels par des flottants.

Ces problèmes de stabilité numérique ont donné naissance à l'arithmétique des intervalles, permettant d'effectuer ces calculs sur une représentation des nombres réels par des intervalles dont la finesse peut être modifiée.

Nous présentons ici les principales techniques numériques utilisées pour la résolution de contraintes géométriques, à savoir, l'itération de Newton-Raphson et l'homotopie, et une introduction succinte à l'arithmétique des intervalles.

Newton-Raphson Aux alentours de 1669, Isaac Newton propose un algorithme pour résoudre une équation polynomiale $f(x)$ de degré 1 par perturbation. À partir d'une valeur initiale g proche de la solution, il suffit de remplacer l'inconnue par $g \pm \varepsilon$ et d'évaluer $f(g \pm \varepsilon)$. On applique alors une itération en utilisant une valeur de ε très petite : $x_{n+1} = x_n \pm \varepsilon$ de manière à se rapprocher de la racine de l'équation. Pour obtenir une solution avec une précision de plus en plus importante, il suffit de diminuer ε lorsqu'on est à ε près de la racine. Graphiquement, cela revient à échantilloner la courbe sur des intervalles $[-\varepsilon, +\varepsilon]$ très petits pour pouvoir se rapprocher de la solution.

En 1690, Joseph Raphson propose d'utiliser la pente de la tangente à la courbe en g comme ε. L'itération de Newton-Raphson devient alors : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

Malheureusement, cette suite dépend fortement de l'approximation initiale et peut ne pas converger vers la solution la plus proche topologiquement ou même ne pas converger du tout si $f'(x)$ s'annule à une étape de l'itération. Cette instabilité se retrouve lorsque l'on considère un système à k équations (multi-varié) $f(X^k)$. À la place de $f'(x)$, on considère le Jacobien J qui est une matrice des dérivées de chaque polynôme en chaque inconnue :

$$X^k_{n+1} = X^k_n - J^{-1}(X^k_n) f(X^k_n)$$

Dans [LLG81], Light *et al.* proposent d'utiliser la décomposition LU pour accélérer le calcul de la Jacobiennne, et fournit un algorithme de résolution locale en cas de changement de quelques paramètres seulement. Cette méthode très rapide a été implantée avec succès dans certains logiciels [Nel85] et notamment des logiciels commerciaux de CAO.

Cette méthode a été très étudiée [HB78, Lig85, LG82] car sa convergence est quadratique et l'esquisse fournit une approximation initiale satisfaisante. En revanche, sa convergence est difficile à contrôler puisque ses bassins d'attraction ont des formes fractales [PR86]. Ainsi, il n'y aucune garantie de trouver la solution recherchée malgré une approximation initiale très proche de la solution.

Homotopie L'homotopie est une méthode qui déforme continuement un système d'équations simple S_2 et une de ses solutions $s_2 \in F(S_2)$ afin d'obtenir une solution s_1 du système initial S_1.

Une fonction homotopique permettant de passer d'une solution s_2 de S_2 à une solution s_1 de S_1 est par exemple :

$$H(t, X) = (1 - t)S_2(X) + tS_1(X)$$

avec $(0, s_2)$ comme solution initiale. Le suivi du chemin homotopique se fait par prédiction-correction. La prédiction se fait généralement en gardant la solution X_i au temps t_{i+1} et la correction est effectuée à l'aide de la méthode de Newton-Raphson

décrite précédemment pour trouver X_{i+1}, ou une variante.

Trouver le système S_2 adéquat avec le même nombre de solutions que S_1 est un problème clé qui peut s'avérer délicat suivant les contraintes considérées. La version classique, où S_2 est un système canonique, permet de retrouver toutes les solutions de S_1 mais est très lente et sujette à des instabilités numériques.

Michelucci *et al.* [LM94b, LM95, FMJ05] ont exposé une variante adaptée aux systèmes de contraintes géométriques en partant de l'esquisse X_0 pour trouver le système S_2 : $S_2(X) = S_1(X) - S_1(X_0)$.

Ainsi, l'homotopie est une méthode plus lente mais aussi plus robuste que la méthode précédente. En effet, ses bassins d'attraction sont semi-algébriques (quand S_1 et S_2 sont algébriques), *i.e.* avec des frontières lisses. On peut de plus affiner le chemin suivi en modifiant le pas t suivant le comportement de la méthode de correction. Finalement, il est possible de trouver toutes les solutions algébriques en suivant autant de chemins homotopiques qu'il y a de solutions *modulo* l'imprécision de l'arithmétique réelle et le nombre de solutions possibles.

Durand [Dur98, DH99, DH00] montre qu'une méthode homotopique généralement très lente en 3D à cause du nombre de chemins à suivre, peut être quelque peu accélérée en appliquant manuellement des simplifications d'équations.

Arithmétique des intervalles L'arithmétique des intervalles fut introduite par Moore [R.E96] pour permettre une représentation en précision finie des nombres réels et éviter la propagation d'erreurs lors de calculs numériques.

On connaît ainsi plus ou moins finement un nombre donné car il est compris entre deux bornes. La réduction de cet intervalle permet d'obtenir une approximation de la valeur réelle que l'on désire représenter à une précision désirée.

Les opérations classiques sur les intervalles sont :

- $[a,b] + [a',b'] = [\lfloor a + a' \rfloor, \lceil b + b' \rceil]$
- $[a,b] - [a',b'] = [\lfloor a - a' \rfloor, \lceil b - b' \rceil]$
- $[a,b] \times [a',b'] = [\lfloor min(aa', ab', a'b, bb') \rfloor, \lceil max(aa', ab', a'b, bb') \rceil]$
- $1/[a,b] = [\lfloor 1/b \rfloor, \lceil 1/a \rceil]$ si $0 \notin [a,b]$

Cette arithmétique permet de raisonner sur des boîtes englobantes. Mais, l'écueil essentiel de cette arithmétique est la croissance des intervalles au fur et à mesure des calculs qui conduit à une gestion parfois lourde de ces boîtes. Ainsi, des opérateurs de filtrage [Del00], permettent de tester la consistance des intervalles pour garantir la présence d'une racine.

En reprenant les méthodes du type Newton-Raphson, et en utilisant cette arithmétique, on obtient une liste de boîtes contenant les solutions régulières et singulières, ou pas de solution du tout si la méthode n'arrive pas à conclure. On peut aussi calculer des encadrements sans utiliser d'intervalles comme avec les bases de Bernstein [GS01, MP05].

En conclusion, les méthodes utilisant l'arithmétique des intervalles sont globalement plus lentes que les autres méthodes numériques. Mais, les garanties qu'elles apportent en font un moyen privilégié pour résoudre de petits sous-systèmes après

l'application d'une décomposition [JTNR00, Jer02]. Dans ce tour d'horizon, nous ne décrivont pas plus ces méthodes car il ne nous parait pas pertinent d'y avoir recours dans une première approche de la résolution de systèmes de contraintes 3D.

Décompositions équationnelles

La décomposition de systèmes en sous-systèmes peut être réalisée à partir d'une analyse combinatoire du système d'équations. Dans ce cadre, deux manières sont présentées ici, la première représente la structure des équations par un graphe biparti inconnues-équations pour proposer une décomposition, et la deuxième propose un test numérique probabiliste pour s'assurer de la rigidité d'un système d'équations.

Graphes bipartis Un graphe biparti est un graphe constitué de deux types de nœuds, et dont deux nœuds d'un même type ne peuvent être reliés par une arête. Il permet notamment de représenter les équations à plusieurs inconnues où inconnues et équations sont les deux types de nœuds. Une arête relie ainsi les équations aux inconnues qu'elles utilisent.

Cette représentation des systèmes d'équations permet d'étudier leur structure. L'idée est de déterminer l'existence d'un couplage maximum entre les deux types de nœuds. Le critère à maximiser est le nombre de composantes structurellement bien-contraintes, c'est-à-dire les composantes où le nombre de variables est égal aux nombre d'équations.

Lorsque tous les sommets du graphe apparaissent dans le couplage, on dit qu'il est parfait. La recherche d'un couplage parfait a une complexité polynomiale.

Theorème 1.3.3 König-Hall
Un système d'équations indépendantes algébriquement est structurellement bien contraint si et seulement si son graphe biparti associé admet un couplage parfait. □

À partir d'un couplage parfait, il est possible de trouver des composantes irréductibles bien-contraintes, en appliquant un algorithme de complexité polynomiale de recherche de composantes fortement connexes [CLR90]. Pour cela, il faut orienter les arêtes du couplage dans les 2 sens, et les arêtes restantes, depuis les équations vers les inconnues.

Dans le cas où le système n'est pas bien-contraint, l'algorithme de Dulmage-Mendelshon [DM58] permet alors de faire apparaître les parties bien/sur/sous contraintes.

Ces deux types de décompositions ont été appliqués dans le domaine de la modélisation par contraintes par Ait-Aoudia *et al.* [AAM93] et par Bliek *et al.* [BNT98]. Latham et Middleditch [LM96] ont proposé d'étudier les graphes bipartis avec des objets et des contraintes. Ce graphe n'est somme toute qu'un graphe biparti variables-équations condensé ou le couplage maximum est pondéré par un critère combinatoire.

L'indépendance des équations est la condition *sine qua none* de validité de ce type de méthodes mais la vérification de cette indépendance qui peut être faite en appliquant l'algorithme de Buchberger précédent, est très coûteuse. La méthode suivante permet de contourner cette difficulté en utilisant un critère probabiliste pour la détection de parties structurellement bien-contraintes.

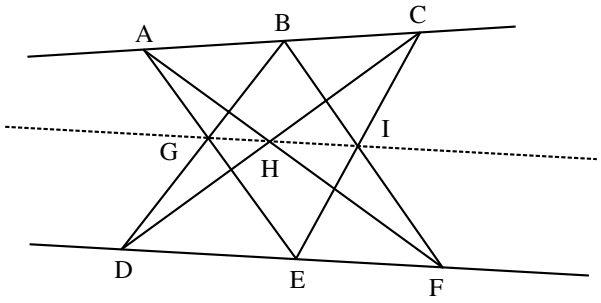


FIG. 1.20 – Théorème de Pappus : Soit A,B,C, et D,E,F trois points alignés quelconques. Les points G,H et I intersections des segments (AE) et (BD), (AF) et (CD) et (BF) et (CE) sont alignés.

Décomposition numérique probabiliste Lamure et Michelucci [LM98] calculent le rang de la matrice jacobienne du système d'équations en quelques points choisis aléatoirement. Si ce rang est déficient en ces quelques points, il y a une forte probabilité que les équations soient dépendantes. Cette méthode, nommée méthode numérique probabiliste (NPM) est issue du domaine de la théorie de la rigidité pour décider de la rigidité des graphes en toute dimension.

Elle a été étendue récemment [FMJ05, MF06] à toutes les contraintes géométriques. Elle permet ainsi de détecter des dépendances dues aux théorèmes projectifs d'incidence comme celui de la figure 1.20, plus subtiles que celles détectées par les techniques combinatoires. À partir de cette détection rapide de la rigidité qui revient à calculer le rang de la jacobienne grâce à la méthode de Gauss, un algorithme glouton est proposé pour décomposer le système : l'oubli d'une contrainte permet de trouver des sous-systèmes rigides maximaux.

La difficulté est qu'il faut trouver un témoin, *i.e.* une configuration qui vérifie les contraintes projectives (alignement, coplanarité, incidence) du système à résoudre. En pratique, trouver un témoin en CAO est souvent trivial, mais en théorie cela peut être arbitrairement difficile. Pour ce faire, ils proposent de s'appuyer sur le prouveur de Jurzak [MJ04]. Combinée avec de l'arithmétique exacte et des calculs dans un corps fini pour éliminer les erreurs d'arrondis, cette méthode semble être utilisable d'un point de vue pratique. Ils proposent d'étudier, dans le futur, le calcul en virgule flottante, en ayant recours aux calcul du SVD (vecteurs propres) et à d'autres méthodes de gestion de l'erreur.

Conclusion du chapitre

Pour clore ce tour d'horizon des méthodes de résolution de contraintes géométriques, effectué à partir de la formalisation de la notion d'univers géométriques, trois remarques d'ordre général nous permettent d'introduire les réalisations présentées aux chapitres suivants.

Premièrement, le problème de l'interface homme-machine est peu abordée lorsque l'on considère les méthodes de résolution. Il nous semble pourtant que la résolution et l'interaction sont fortement liées. De plus, on peut remarquer qu'il y a eu peu d'expérimentations dans des univers géométriques restreints en 3D comme cela a été le cas en 2D avec par exemple l'univers des bi-points, etc.

Nous proposons donc d'étudier conjointement l'interaction et les univers géométriques pour dégager de nouvelles méthodes efficaces dans des domaines restreints : cela donne lieu dans le chapitre suivant à l'utilisation de la réalité virtuelle pour la modélisation par contraintes 3D et à la proposition d'un solveur efficace dans le cadre restreint des univers isothétiques.

Deuxièmement, la décomposition est une stratégie primordiale pour la résolution de contraintes géométriques que ce soit en 2D ou en 3D. Mais en 3D, les méthodes à base de décomposition produisent le plus souvent des systèmes irréductibles difficiles à résoudre en raison de leur taille. La résolution numérique de ces systèmes n'est plus satisfaisante et nous proposons de combiner efficacement résolution formelle et numérique pour résoudre ces systèmes. Ainsi, le chapitre 3 présente la méthode de la reparamétrisation qui permet notemment d'étendre à la troisième dimension l'utilisation des solveurs formels à base de connaissances géométriques développés à Strasbourg.

Troisièmement, les solveurs formels en 3D souffrent encore plus qu'en 2D d'une relative lenteur. Si la reparamétrisation permet de bénéficier de leur puissance, il faut tout de même permettre leur utilisation d'une manière efficace en proposant des optimisations. Pour effectuer ces optimisations, nous proposons l'utilisation d'une architecture logicielle adaptée à la résolution de contraintes en général et qui permet d'automatiser la gestion des univers géométriques et des méthodes de résolutions qui s'y rapportent. Le chapitre 4 présente cette architecture ainsi que la méta-compilation d'univers géométrique.

Chapitre 2

Modélisation

La modélisation par contraintes est proposée dans la plupart des logiciels commerciaux sous la forme d'un « plugin » utilisé en complément d'une modélisation classique. Ces logiciels proposent à la fois un procédé pour saisir les contraintes et une méthode de résolution prenant en compte le type de contraintes posées.

Malheureusement, en 3D, les outils disponibles pour poser des contraintes géométriques ne sont pas adaptés. En effet, l'interaction est limitée par l'utilisation d'une interface 2D : clavier, souris et écran, alors que les objets et les manipulations possibles sont de nature 3D. Nous proposons donc de recourir à certaines technologies issues de la réalité virtuelle pour construire une interaction directe en 3D pour la modélisation par contraintes. Ainsi, trois expérimentations sont mise en place au moyen de gants de données couplés à une visualisation stéréoscopique :

- *les constructions géométriques dynamiques ;*
- *la résolution de problèmes isothétiques ;*
- *la pose de contraintes de distances, d'angles et de coplanarités.*

Néanmoins, il ne suffit pas de changer de périphériques pour obtenir une pose de contraintes en 3D. Il faut également considérer la manière d'interagir avec ces outils. C'est pourquoi nous avons expérimenté l'utilisation de gestes et d'outils virtuels spécifiques à la modélisation géométrique dans chacun des trois cadres envisagés. Ceci a donné lieu à trois prototypes, chacun appartenant à l'un des domaines que nous présentons dans ce chapitre.

Sommaire

2.1 Concepts

Le but de ce chapitre n'est pas de proposer une nouvelle interaction et de la comparer à celles existantes, mais de préciser la dépendance entre l'interaction homme-machine et la modélisation par contraintes en 3D. Afin de pouvoir expérimenter les technologies offertes par la réalité virtuelle, nous introduisons rapidement dans cette section les notions relatives à l'interaction, les technologies générales de la Réalité Virtuelle (RV), et les périphériques et métaphores d'interaction que nous avons utilisés pour mettre au point notre interaction 3D directe.

2.1.1 Vocabulaire

L'interaction est l'étude des réactions réciproques de deux systèmes communicants : à savoir, pour nous, un utilisateur et un logiciel de modélisation *via* une interface Homme-Machine.

La communication entre deux systèmes se fait à l'aide de périphériques physiques et virtuels qui servent de support à des *métaphores d'interaction*.

Les périphériques physiques sont dits d'entrée lorsqu'ils servent à faire passer une interaction de l'utilisateur vers l'interface et périphériques de sortie dans le sens inverse. Les périphériques virtuels permettent de signaler la réaction du logiciel à une action faite par l'utilisateur *via* son interface. Par exemple, le curseur suit les mouvements de la souris, ou bien un son signale un événement particulier du système.

Les périphériques tangibles dont nous disposons aujourd'hui tels que le clavier, l'écran, la souris, les imprimantes ou les scanners, *etc.* sont la conséquence historique du développement de l'informatique de bureau. Leur introduction a été accompagnée de l'amélioration des métaphores d'interaction implantées dans les systèmes d'exploitation. Ainsi, un saut qualitatif a été effectué en passant du traitement textuel de l'information à son traitement graphique avec l'introduction de l'interface graphique et de la métaphore d'interaction de type WIMP (*Windows Icons Menus Pointing devices*) pour Fenêtres, Icônes, Menus et Souris. L'utilisation de la souris notamment a permis d'exercer une *interaction directe* sur des objets conceptuels d'interface nommés *widgets* (*Window Gadget*), mais aussi sur des objets géométriques dans le cadre de la modélisation géométrique.

Dans le cas d'une interaction directe, on peut associer aux périphériques une certaine dimension de fonctionnement : nous appellerons par la suite *interaction 2D*, une interaction effectuée à l'aide de la souris sur un écran classique, et *interaction 3D*, une interaction utilisant des périphériques physiques et virtuels à plus de 2 degrés de liberté.

Bien que l'interaction directe classique avec le couple souris/écran soit la norme dans les « environnements de bureau », elle n'en reste pas moins limitée à un surface planaire. L'interface et les périphériques 2D limitent l'interaction avec des objets en trois dimensions. Avec l'émergence de la RV, les possibilités se sont élargies et

on peut considérer des périphériques d'entrées ou de sorties offrant plus de libertés pour l'utilisateur. Cependant, l'augmentation des possibilités induites par ces périphériques ou par les métaphores d'interaction qui les accompagnent entraînent un accroissement de la charge cognitive concernant l'interaction que les recherches actuelles tentent de gommer. Il n'existe pas de métaphores ou de périphériques « ultimes », ni même de normes pour l'instant, et la question de l'interaction 3D est une problématique qui est encore à l'étude.

2.1.2 Technologies de la Réalité Virtuelle

La réalité virtuelle désigne un ensemble de technologies et de systèmes dont le point commun fondamental est de procurer à l'utilisateur de ces outils une sensation d'*immersion* dans un univers créé par ordinateur. Cet univers peut être une simulation de la réalité ou au contraire un environnement totalement imaginaire. À notre sens, la RV est un dénominateur commun à l'utilisation d'artefacts matériels et logiciels permettant de produire une interaction à l'aide des 5 sens et d'agir sur les sensations et les sentiments d'un être humain. La combinaison de ces technologies permet la mise au point d'interactions *multimodales*.

Il n'existe pas de RV unique, mais bien une multitude de RV, définies en fonction de leur cadre d'utilisation et des buts recherchés. Par exemple, dans le traitement des phobies, la RV permet d'explorer à l'aide de tous ses sens un monde dépouillé des dangers physiques du réel. La RV peut aussi s'appliquer au partage et à la manipulation d'informations entre plusieurs personnes [KTY96].

Par ailleurs, le caractère ludique de la RV s'est traduit par des investissements importants de la part de l'industrie des jeux vidéos. Les premiers produits grand public bénéficiant de ces technologies commencent à être diffusés : la console Wii de Nintendo qui est munie d'une manette à capteur de mouvements [Nin06], ou le jeu Hitman [Eid] qui permet l'utilisation de gants de données (*cf.* figure 2.1).

FIG. 2.1 – Gant de donnée P5 d'Essential Reality.

Nous ne donnons ici qu'un aperçu des nombreuses technologies proposées en les décomposant suivant les périphériques matériels et les métaphores d'interaction utilisées en modélisation 3D. Le lecteur intéressé trouvera plus de détails dans le mémoire de thèse de Ludovic Sternberger [Ste06] ou encore dans le *Traité de la réalité virtuelle* [Fuc03].

Interaction matérielle

Entrées Il existe beaucoup de dispositifs physiques permettant d'agir sur une scène 3D. La première grande famille de périphériques d'entrée regroupe les équivalents de la *souris* à plusieurs degrés de liberté. Par exemple, le **wand** (*cf.* figure 2.2) est un dispositif de pointage muni de capteurs permettant de repérer sa position dans l'espace suivant ses 6 degrés de liberté. Il permet ainsi de manipuler des objets 3D virtuels en leur appliquant des translations dans toutes les directions et des rotations dans l'espace. Un ensemble variable de boutons et de molettes accompagne généralement ces périphériques.

FIG. 2.2 – Un wand permettant l'interaction à 6 degrés de liberté

Une autre famille de périphériques, les périphériques d'acquisition de mouvements, utilisent la position et l'orientation d'une partie de notre corps pour agir sur la scène virtuelle. Des capteurs sont positionnés sur l'utilisateur et retournent des informations à l'ordinateur qui calcule les informations nécessaires pour l'interaction. On distingue la capture à partir d'images des autres procédés d'acquisition comme ceux utilisant les ultrasons ou les gants de données (cf. figure 2.3).

FIG. 2.3 – Gant de données

Sorties La plupart des dispositifs matériels proposent une sensation visuelle tridimensionsionnelle à l'aide de **la vision stéréoscopique** : deux images différentes

de la même scène sont affichées suivant le point de vue de chaque œil. Par exemple, avec une paire de lunettes à obturateurs (*cf.* figure 2.4), chaque œil de l'utilisateur reçoit uniquement l'image qui le concerne et le cerveau fusionne les 2 vues pour reconstituer la scène en 3D.

FIG. 2.4 – Paire de lunettes pour la stéréoscopie active

Les matériels à retour d'effort, ou haptiques constituent à la fois des périphériques d'entrées et de sorties. Ils permettent de retourner à l'utilisateur une force physique à l'aide de moteurs agissant sur ses membres, en fonction des actions effectuées dans l'univers virtuel. Le phantom 2.5 est un matériel typique du retour d'effort et permet de simuler par exemple des opérations chirurgicales.

FIG. 2.5 – Retour haptique

Interaction logicielle

La taxonomie que nous présentons se base sur celle présentée par Bowmann [Bow99] en considérant que l'interaction peut revêtir 3 formes :

- la navigation ;
- la sélection ;
- la manipulation.

Ces formes d'interaction se font au travers des périphériques d'interaction matériels présentés ci-dessus. On dit que l'interaction est directe lorsque l'on dispose de détecteurs de position et d'orientation, et indirecte si on en est dépourvu. Dans ce cas là, on distingue les niveaux d'indirection physiques des niveaux virtuels qui se caractérisent par un retour sensoriel plus ou moins marqué.

Naviguer La navigation dans un univers virtuel consiste à se déplacer, à trouver son chemin à l'aide de repères visuels, sonores, ou tactiles. Habituellement, la navigation se contente d'être régie par le contrôle du déplacement du point de vue de l'utilisateur. En fonction de la taille du dispositif et du type d'univers, plusieurs

types de navigation sont possibles comme celui du paradigme du monde en miniature [SCP95] qui permet de manipuler le point de vue dans une reproduction de la scène plus petite.

Sélectionner La sélection d'un objet de la scène se décompose en une tâche de désignation de cet objet et une tâche de validation. La sélection permet de déclencher des commandes ou de choisir un objet sur lequel on va appliquer une action, comme une manipulation par exemple.

On peut caractériser les techniques de sélection par la distance séparant l'utilisateur des objets qu'il désire sélectionner. Ainsi, la sélection locale s'effectue lorsque l'objet est à portée d'interaction, par exemple avec une boîte englobante. La sélection distante s'applique lorsque l'objet est hors de portée. Par exemple, la technique du rayon laser et ses nombreux dérivés [Min95, ZBM94, FHZ96] permettent de sélectionner les objets à distance en les désignant avec un faisceau laser virtuel. Ceci peut se faire aussi grâce à la technique du Gogo [PBWI96] où un bras virtuel s'allonge jusqu'à ce que l'avatar de la main puisse sélectionner l'objet ou par l'intermédiaire du monde en miniature ou au moyen de poupées vaudoues [PSP99].

Manipuler La manipulation des objets consiste essentiellement en l'application de transformations géométriques telles que les déplacements et la mise à l'échelle. Les manipulations s'effectuent traditionnellement par rapport à un repère local à l'utilisateur, mais elles peuvent aussi être effectuées par rapport au repère local de l'objet considéré. Par exemple, la métaphore du Homer (Hand centered Object Manipulation Extending Raycasting) [BH97] consiste en une manipulation locale à l'objet combinée à une sélection distante avec la métaphore du laser. Les manipulations peuvent aussi s'effectuer au travers d'intermédiaires tels que le monde en miniature.

2.2 Outils

D'un point de vue logiciel, des bibliothèques de bas niveau sont fournies par les constructeurs de matériel. Mais, à un plus haut niveau, il n'existe pas d'équivalent satisfaisant aux bibliothèques d'interface 2D comme *Qt*, *Gtk* ou les frameworks *Windows* ou *MacOs*, puisque d'une manière générale il n'existe pas de paradigme d'interaction établi.

Nous avons donc participé au développement d'une bibliothèque de haut niveau permettant de définir ses propres outils d'interaction 3D. De plus, cette bibliothèque propose les équivalents 3D des widgets 2D ce qui facilite le passage d'un environnement de bureau à un environnement de réalité virtuelle.

2.2.1 Bibliothèque pour l'interface et l'interaction 3D

La bibliothèque *VRLib* a été développée dans le but d'explorer les techniques d'interaction 3D en réalité virtuelle. Son élaboration est le fruit d'un travail en groupe de quatre ans. Les détails de ce projet font l'objet d'une partie du mémoire de thèse de Ludovic Sternberger [Ste06] qui en a été le coordinateur. De plus, on

trouve dans cette thèse une comparaison détaillée des bibliothèques existantes et de leur fonctions respectives.

La philosophie opérationnelle de cette bibliothèque se rapproche de celle de la bibliothèque d'interface 2D *Qt* [Qt] en proposant une couche objet de haut niveau permettant une programmation rapide avec peu de lignes de code. La *VRLib* met à disposition une interface objet en C++ qui contient une couche d'abstraction du matériel, la gestion de scènes géométriques et des outils d'interaction.

Elle est indépendante du matériel et nécessite le recours à une librairie proposant une couche de bas niveau de type VRJuggler [vrj] pour que la liaison avec le matériel soit effective. Elle supporte l'abstraction des contextes graphiques OpenGL pour un rendu sur plusieurs machines.

La bibliothèque contient des outils offrant la possibilité de maintenir un graphe de scène composé d'objets géométriques. Les communications sont gérées, soit par un mécanisme de files de messages pour la communication système-objets, soit par un mécanisme de « slots et signaux » pour la communication entre objets d'interface [Qt]. Le mécanisme des slots et signaux consiste à établir une connection entre un évènement (le signal) et l'action à effectuer (le slot) entre des instances de classes différentes en programmation orientée objet. Cela permet une gestion propre de la programmation évènementielle.

Un ensemble de widgets a été développé, tels que des boutons, des panneaux, des fenêtres, des ascenseurs, afin de permettre le développement de la partie graphique de l'interface.

Cette bibliothèque permet de développer non seulement une interface graphique en Réalité Virtuelle mais propose aussi des outils pour définir l'interaction avec les objets de l'application à un haut niveau d'abstraction. L'interaction est basée sur des outils d'interaction, éléments programmables, qui permettent d'adapter aux objets manipulés le type d'interaction désirée. Toutes les informations concernant l'interaction sont regroupées au sein de ces outils.

Notre participation à la conception de cette bibliothèque s'est traduite par la mise en place d'une interaction spécifique aux objets géométriques contraints en 3D. Cette interaction repose sur le développement d'une interaction gestuelle et sur la gestion de la modélisation par contraintes. Nous présentons dans le reste de cette section quels ont été les choix effectués dans ce cadre.

2.2.2 Interaction gestuelle

D'une manière générale, les gestes constituent un moyen intuitif de communication entre êtres humains. Dans le cadre de la réalité virtuelle, l'interaction gestuelle est une technique classique qui nécessite la capture du mouvement et la reconnaissance des gestes. L'interaction bi-manuelle utilise la coordination des deux mains pour réaliser des actions. Classiquement, on distingue la main dominante, qui effectue généralement l'action, de la main non-dominante, qui sert de référentiel à l'autre

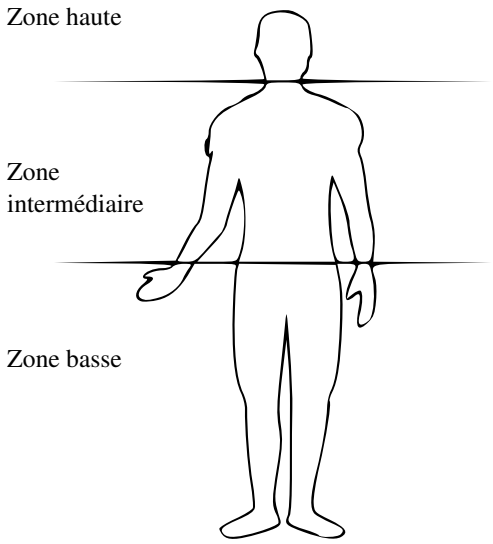


FIG. 2.6 – Zones de reconnaissance de gestes

main.

Les mains adoptent des *postures* que le système doit reconnaître et interpréter. La reconnaissance de postures est complétée par l'analyse des déplacements dans l'espace et dans le temps.

Bien sûr, il est très difficile de distinguer les gestes ayant un sens pour l'interface, des gestes « parasites » qui font parti du flot naturel de l'activité des mains et des bras. Nous avons donc choisi d'utiliser des zones d'activité et des gestes prédéfinis pour l'interaction.

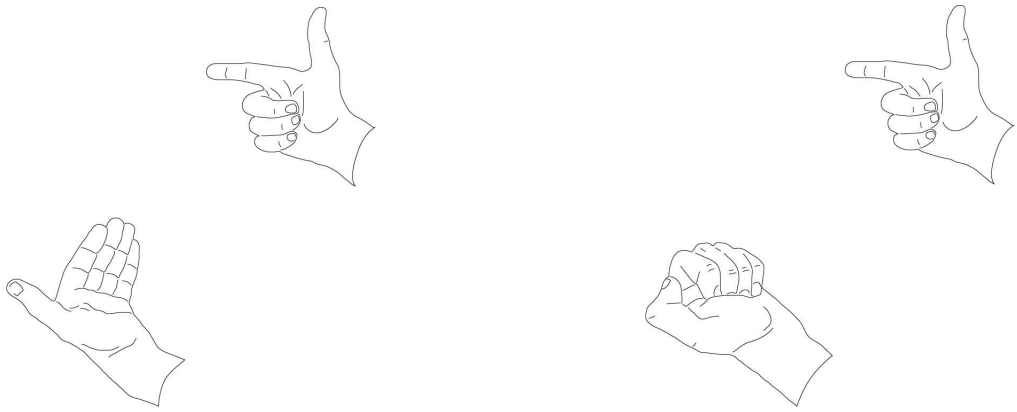
Principe des zones Dans une boîte qui englobe l'utilisateur, aucune interprétation des gestes n'est effectuée. Cette zone « morte », dépendante de la corpulence de l'utilisateur, est calculée à partir de la position des capteurs.

D'une manière générale, nous avons défini trois zones représentées à la figure 2.6 :
- la zone basse, située en dessous de la ceinture ;
- la zone intermédiaire, située entre la ceinture et le cou ;
- la zone haute, située au dessus du cou.

Ces trois zones sont appliquées de manière arbitraire à 50% et 10% de la hauteur du capteur sur la tête. Bien sûr, une phase de calibrage par rapport à la morphologie de l'utilisateur est toujours possible en modifiant les fichiers de configuration de la bibliothèque.

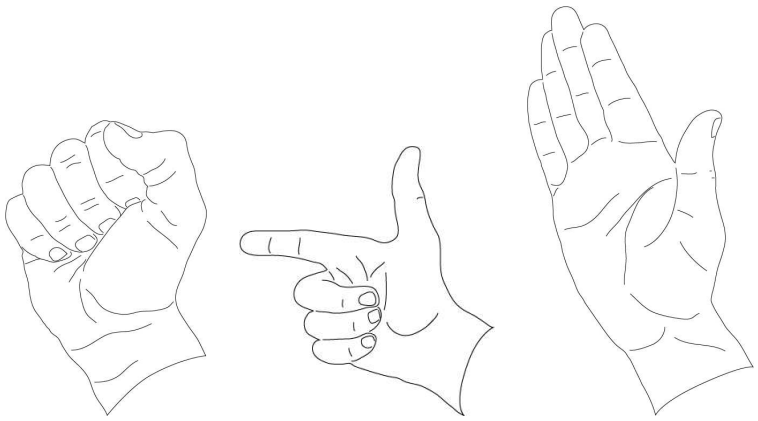
Par défaut, ces zones sont réservées à certaines activités. La zone basse est considérée comme une zone morte, la zone intermédiaire comme une zone de reconnaissance de gestes pour l'interaction directe sur les objets géométriques, et la zone haute est réservée au contrôle d'application, comme par exemple, l'appel à des menus, ou le changement de modes d'interaction.

Dictionnaire de gestes La sélection d'objets se fait grâce à un geste de désignation présenté à la figure 2.7. La main dominante désigne un objet avec le pouce et



(a) Désignation d'un segment (b) Sélection d'un segment

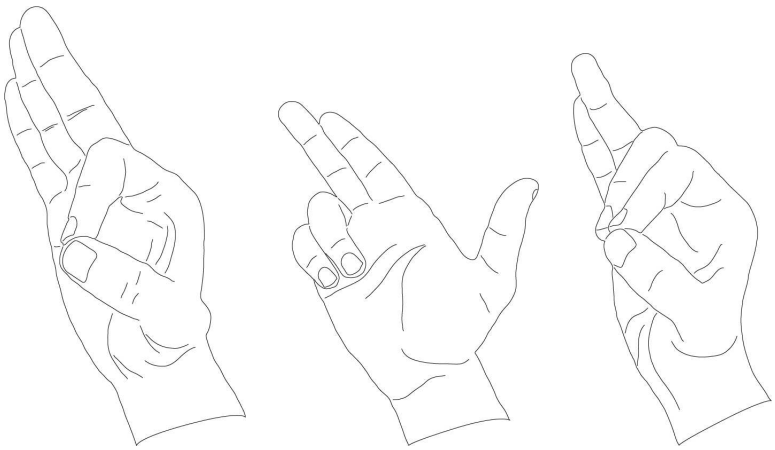
FIG. 2.7 – Sélection d'objets



(a) Point (b) Segment (c) Plan

l'index relevé et la main non dominante agit comme un déclencheur pour la sélection.

Pour améliorer la précision de cette technique de sélection, des gestes spécifiques ont été affectés aux sortes de l'univers géométrique considéré. Ils constituent ainsi un point de vue sémantique associé à l'univers géométrique euclidien 3D et qui est propre à la RV. Ils sont regroupés dans un dictionnaire de gestes qui sert de paramètre à l'application. Nous distinguons six gestes pour les principales sortes géométriques définies dans l'univers géométrique euclidien 3D (*cf.* figure 2.8). Le premier geste est utilisé pour ajouter ou sélectionner un point (fig.2.8(a)). La posture associée à un segment est un index pointé en avant (fig.2.8(b)). La droite est représentée par l'index et le majeur relevés (fig.2.8(e)). La main ouverte correspond à un plan (fig.2.8(c)). La *pince de crabe* (le pouce et l'index formant un cercle) représente naturellement un cercle (fig.2.8(d)). Et si le majeur est aussi plié, le geste désigne une sphère (fig.2.8(f)).



(d) Cercle (e) Droite (f) Sphère

FIG. 2.8 – Dictionnaire de gestes pour la RV

Ce dictionnaire a été construit de manière à avoir des gestes suffisament éloignés les uns des autres pour une bonne reconnaissance mais aussi pour conserver un lien « cognitif » avec les objets qu'ils représentent. Dans les langages de signes, il existe des gestes spécifiques pour les objets géométriques, mais nous n'avons pas tenté de les utiliser car ils nécessitent pour leur reconnaissance un traitement plus important : ce ne sont pas seulement des postures qui doivent être reconnues mais des gestes dynamiques.

Reconnaissance de postures Les gants de données fournissent un ensemble de valeurs correspondant à la flexion des phalanges de chaque doigt. Une phase préalable de calibration des gants doit être effectuée pour déterminer un domaine de validité pour ces flexions. Les valeurs de ces flexions ne sont pas très précises et dans la pratique l'utilisation d'une reconnaissance basée directement sur ces valeurs n'est pas suffisante pour distinguer plus de deux postures classiques : main ouverte et main fermée.

Nous avons donc utilisé un réseau de neurones par main pour la reconnaissance de postures, comme cela a déjà été fait dans [OH99, SW00, SSK01]. Le réseau de neurones est composé de 100 neurones dans une couche cachée. La phase d'apprentissage s'effectue sur un ensemble de postures où les valeurs sont idéales, jusqu'à obtenir une erreur inférieure à 0.5%. Une posture est considérée comme reconnue si le réseau de neurones donne une valeur supérieure à 80% pour une posture.

Pour les gestes décrits dans cette section, les résultats obtenus sur une dizaine d'utilisateurs permettent d'obtenir une reconnaissance correcte 86.77% du temps.

2.2.3 Gestion des contraintes

La gestion des contraintes d'incidence objet-droite, objet-plan, et objet-sphère est intégrée par défaut dans le noyau de réalité virtuelle. Ces contraintes permettent

de fixer un objet dans l'espace, ou de le manipuler dans un sous-espace comme une droite ou un plan.

Ainsi, nous considérons 3 sortes de manipulations d'objets sous-contraints : les manipulations axiales, planaires et rotoïdes.

Classiquement, la manipulation d'un objet contraint revient à une entité spécialisée : le *gestionnaire de contraintes*. Dans le cadre des contraintes d'incidences, ce gestionnaire est chargé de traduire les manipulations effectuées par l'utilisateur en des manipulations vérifiant les contraintes de l'objet.

Lorsqu'une contrainte de manipulation est associée à un objet, celui-ci devient contraint et le gestionnaire se charge de calculer le déplacement valide à partir du déplacement réel soumis par l'utilisateur. Ces contraintes de manipulation sont en fait des fonctions de projection des déplacements sur des objets de l'univers considéré.

Au niveau de l'interface, la représentation de ces contraintes nécessite un affichage particulier. Nous proposons d'utiliser des cubes pour montrer l'incidence d'un objet à un plan, des cônes pour l'incidence à une droite et des sphères pour l'incidence à une sphère. Manipuler des objets sous-contraints est directement possible à l'aide de ces *poignées*.

S'agissant des autres contraintes et notamment les contraintes dimensionnelles, il faut recourir à un solveur externe. Le gestionnaire de contraintes a été conçu pour être extensible et servir d'interface pour intégrer des solveurs de contraintes. Le mécanisme est inchangé dans la bibliothèque *VRLib*. En effet, le gestionnaire reçoit des demandes d'introduction de nouvelles contraintes ou de déplacements d'objets contraints. Il va donc les traiter de façon à déterminer la nouvelle position de l'objet, tout en respectant la ou les contraintes qui lui sont associées et en tenant compte des répercussions éventuelles sur les autres objets de la figure.

La résolution peut être soit incrémentale, soit déclenchée à un instant donné.

2.3 Interaction en réalité virtuelle

Pour nos expérimentations, nous utilisons un environnement de réalité virtuelle, nommé *Workbench* (*cf.* figure 2.9) de chez Barco, constitué de deux écrans formant un dièdre ouvert et de quatre caméras infrarouges permettant de suivre les mouvements de la tête et des mains à l'aide de capteurs fixés sur des lunettes de stéréoscopie active et sur des gants de données. Les gestes de l'utilisateur sont reconnus après un calibrage et une phase d'apprentissage à l'aide de la bibliothèque *VRLib*.

Nous présentons dans cette section les expérimentations réalisées dans cet environnement. La première application concerne le domaine des constructions géométriques 3D développé en réalité virtuelle. Ce fut pour nous un premier pas vers la modélisation géométrique en CAO et en particulier pour l'esquisse d'objets 3D. À

FIG. 2.9 – Workbench

cette occasion, nous avons conçu et implanté un ensemble d'outils pour améliorer la précision de l'interaction.

Nous exposons ensuite une expérience menée dans un univers de résolution simple nommé univers isothétique. Nous étudions les contraintes et les gestes mais aussi la résolution qui a été mise en place dans cet univers.

Enfin, une modélisation par pose de contraintes sur une esquisse est décrite dans la dernière partie de cette section.

2.3.1 Constructions géométriques dynamiques

Lorsqu'un énoncé géométrique est donné, il n'est pas facile d'en déduire une figure lisible. Le problème devient encore plus ardu lorsque cette figure doit permettre d'observer des propriétés géométriques particulières, et il est pratiquement insoluble sans l'aide de l'ordinateur si cet énoncé considère l'espace à trois dimensions. De nombreux logiciels commerciaux 2D [Mec94, Jac95, Kor99, Cha99] ont été développés dans le domaine de l'EIAO pour répondre à cette question. Ils définissent tous une architecture dans laquelle les outils de construction, accessibles au moyen d'un jeu de menus, sont imposés et traduisent plus ou moins l'utilisation d'outils de tracés classiques comme la règle et le compas. Cette manière d'imposer un cadre géométrique a donné lieu au qualificatif de *micro-monde de géométrie* qui a été popularisé par le logiciel Cabri-Géomètre [Bau91, LL92], un des plus fameux logiciels de constructions géométriques en 2D.

Ainsi, l'utilisateur peut poser des points et des droites à l'aide de fonctions de construction, mais il peut aussi modifier dynamiquement la figure par manipulation directe de ces objets de telle sorte que les constructions restent vérifiées. Ceci se traduit par la gestion d'un graphe de dépendance au sein de l'application, qui permet de rejouer la construction pour des valeurs d'objets initiaux différents. Le premier logiciel ayant mis en œuvre cette idée est Cabri-géomètre qui a consacré le terme de *Géométrie Dynamique*.

FIG. 2.10 – Problème d'Apollonius

Le passage à la dimension trois a été peu étudié. Quelques logiciels comme Cabri 3D [Qas97], GeospacW [Geo98], ou Calques 3D [Lab98, Lab99] permettent de faire des constructions 3D avec les périphériques et les métaphores d'interaction 2D traditionnels mais leur utilisation dans l'enseignement de la géométrie 3D n'a pas été concluant. En pratique, la visualisation et la manipulation de la géométrie 3D avec des outils 2D ne sont pas assez intuitives et ne permettent pas une interaction efficace. L'utilisation d'un environnement de réalité virtuelle améliore les constructions géométriques car l'utilisateur est capable d'attraper ces entités 3D et d'interagir avec elles directement en 3D.

Nous avons développé un prototype de constructions géométriques, appelé *Coyote-Géomètre* [FMS04], dans notre environnement de réalité virtuelle. Notre approche est basée sur l'utilisation de gants de données pour reconnaître des postures. C'est la principale différence avec les études précédentes comme Construct3D de Hannes Kaufmann [Kau02]. Appliquée à un autre domaine, notre approche est comparable à celle de Zeleznik *et al.* [ZHH96] qui proposent une interface basée sur les gestes pour approcher la modélisation de polyèdres 3D.

Cette interface nous a permis d'explorer les constructions géométriques en 3D en favorisant notamment l'analyse de problèmes de construction en 3D [FCMS03] comme par exemple celui de la construction d'un plan tangent à 3 sphères données (*cf.* figure 2.11) et le problème d'Apollonius 3D (*cf.* figure 2.10) qui consiste en la construction d'une sphère tangente à quatre autres sphères données. Un des buts

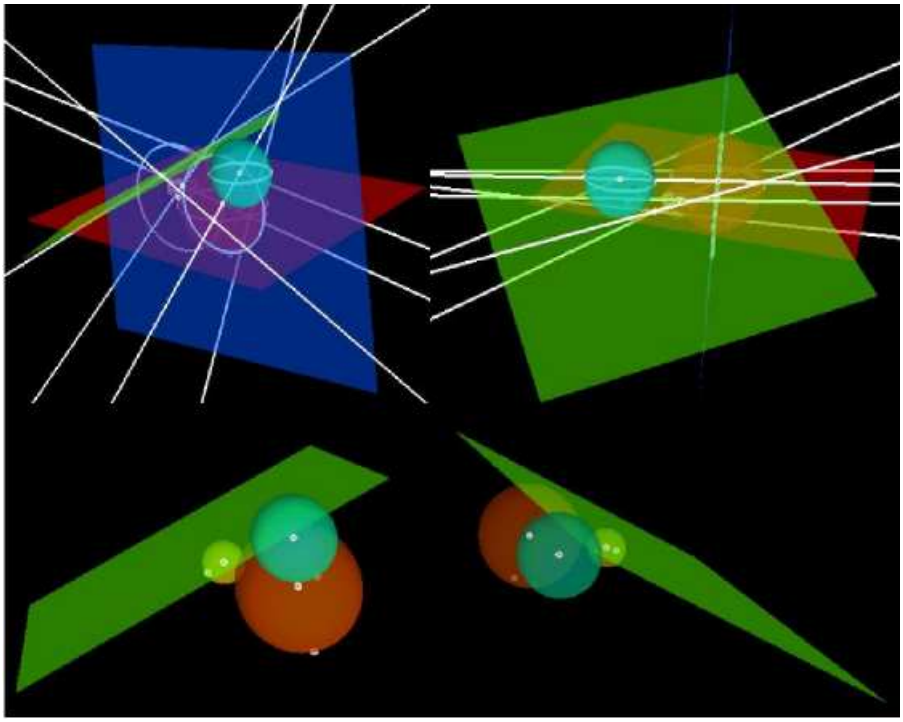


FIG. 2.11 – Plan tangent à 3 sphères

poursuivis par la conception de *Coyote-géomètre* consiste à fournir une meilleure perception et compréhension de la géométrie dans l'espace. Pour arriver à ce but, l'interaction 3D que nous fournissons se veut peu intrusive et a été pensée pour accompagner l'utilisateur à chaque étape de sa construction. Ainsi, l'interaction gestuelle permet de simplifier les mécanismes de sélection et de construction en étant associée à des widgets adaptés à la modélisation [FSSB06].

Exemple de construction géométrique 3D

Plaçons-nous dans l'univers de la géométrie euclidienne 3D tel que décrit dans le chapitre précédent (à la section 1.2.1), et considérons l'énoncé suivant :

Énoncé2.3.1 L'intersection d'un cube et d'un plan est un polygone pouvant avoir 3, 4, ou 6 cotés. □

Premièrement, comme notre univers ne contient pas de primitives cube, nous allons devoir en construire un à partir des symboles fonctionnels disponibles.

Un plan de construction textuel d'un cube en fil de fer pourrait être le suivant (il y a généralement plusieurs manières de réaliser une même figure) :

1. Soit p_1 un point
2. Soit d_1 une droite passant par p_1
3. Soit d_2 une perpendiculaire à d_1 passant par p_1
4. Construire π le plan passant par d_1 et d_2
5. Soit p_2 un point de d_1
6. Construire σ la sphère de centre $p1$ passant par p_2
7. Soit p_3 une des intersections entre d_2 et σ

8. Soit d_3 la perpendiculaire au plan π passant par p_1
9. Soit p_4 une des intersections de d_3 et σ
10. Construire la parallèle d_4 à d_1 passant par p_3
11. Construire la parallèle d_5 à d_2 passant par p_2
12. Soit p_5 l'intersection de d_4 et d_5
13. Construire la parallèle d_6 à d_1 passant par p_4
14. Construire la parallèle d_7 à d_3 passant par p_2
15. Soit p_6 l'intersection de d_6 et d_7
16. Construire la parallèle d_8 à d_4 passant par p_3
17. Construire la parallèle d_9 à d_2 passant par p_2
18. Soit p_7 l'intersection de d_8 et d_9
19. Soit d_{10} la perpendiculaire de d_6 passant par p_6
20. Soit d_{11} la perpendiculaire de d_4 passant par p_5
21. Soit p_8 l'intersection de d_{10} et d_{11}

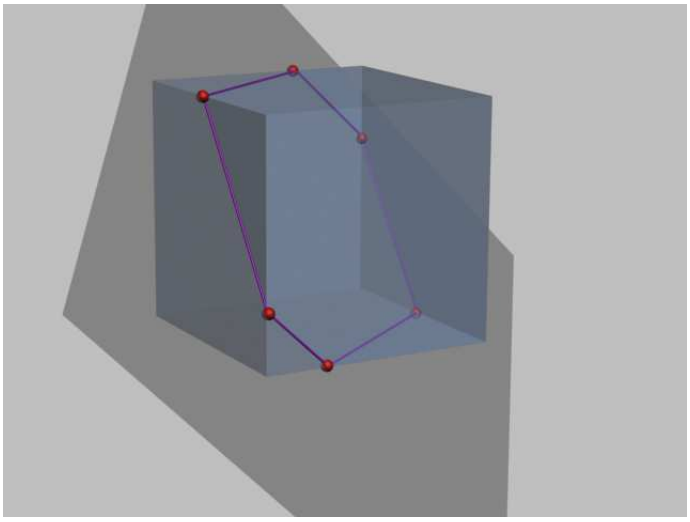


FIG. 2.12 – Intersection d'un cube et d'un plan

Ensuite, un plan libre est créé et les intersections entre le plan et le cube doivent être déterminées. Dans cet exemple, certains objets sont libres comme le point $p1$, d'autres sont partiellement définis comme $p2$, et d'autres enfin sont totalement définis comme $p5$.

Au départ d'une construction, l'environnement est vide. En désignant une place vide dans la scène avec la main dominante et en fermant la main non-dominante, l'utilisateur effectue une action de création d'un nouvel objet libre. Le type de cet objet est défini par la forme de la main dominante (*cf.* figure 2.8). Il est alors possible de désigner les objets afin de les manipuler ou de les utiliser pour de nouvelles constructions.

Passer d'une étape à l'autre est automatique. Mais le passage à un mode de navigation doit être explicite.

D'un point de vue interaction 3D, il est donc nécessaire de considérer ces quatre étapes :

1. la sélection des objets ;
2. la construction de nouveaux objets ;
3. la manipulation des objets libres et des objets contraints ;
4. la navigation dans cette scène.

Pour chacun de ces points, nous exposons le fonctionnement général du système qui permet d'explorer les configurations décrites par l'énoncé précédent (*cf.* figure 2.12).

Sélection

Comme nous l'avons décrit à la section 2.1.2, la sélection se décompose en une phase de désignation et une phase de validation. Dans la première phase, l'utilisateur désigne un objet avec la main non dominante ouverte. Ensuite, dans la seconde phase, il valide la sélection en refermant sa main non-dominante.

Cette phase de sélection est facilitée par la mise en place d'un système de calques agissant comme un filtre pour la sélection. Un calque est un élément de décomposition de l'espace en termes d'objets. Ainsi, on associe un calque à un ensemble d'objets sur lesquels on peut appliquer des modifications comme par exemple les rendre visibles ou invisibles mais aussi effectuer un déplacement global.

Des calques prédéfinis permettent de regrouper les objets géométriques d'une même sorte pour permettre un mécanisme de filtrage particulier : par défaut, il existe un calque par sorte. Ainsi, la sélection devient non-ambigüe et les objets « parasites » sont éliminés de la phase de désignation. Dans ce schéma de sélection, il y a une correspondance entre une sorte d'objet, une posture et un calque particulier. Cette correspondance constitue une sémantique d'interaction pour les sortes de l'univers géométrique considéré. Visuellement, un calque est représenté par un cube en fil de fer.

De plus, dans leur usage classique, les calques permettent de décomposer le processus de construction, en cachant des parties de constructions intermédiaires pour une meilleure compréhension de la construction. L'affichage progressif des calques en les rendant visibles permet une explication de la construction aux étudiants étape par étape.

Enfin, la sélection multiple est possible directement en sélectionnant un calque afin d'éviter de sélectionner chaque objet. Finalement, les calques dépendent d'un calque parent global représentant la scène globale et permettant un équivalent du raccourci clavier *ctrl-a* dans un environnement WIMP. La sélection multiple peut tirer profit de l'interaction bi-manuelle. Un volume peut être sélectionné en désignant, par une boîte englobante, chaque objet en faisant partie.

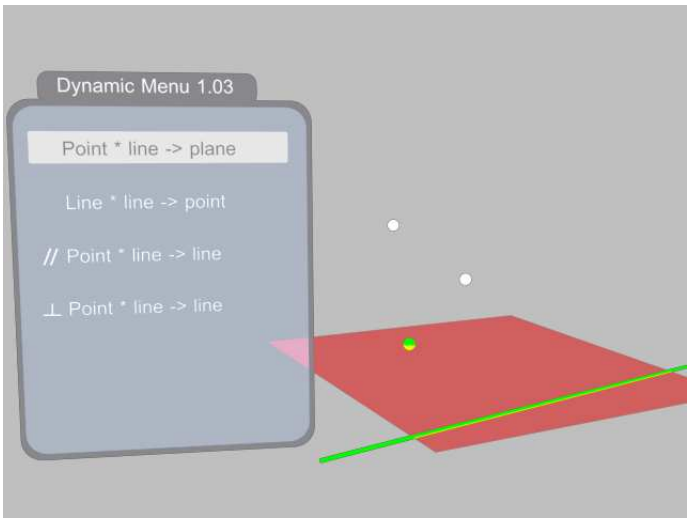


FIG. 2.13 – Menu contextuel

FIG. 2.14 – Boite à outils pour un micro-monde de géométrie

Construction

Dans notre prototype, les deux étapes nécessaires à la construction d'un nouvel objet : sélectionner les objets intervenant dans la construction et sélectionner la fonction de construction, peuvent se dérouler dans n'importe quel ordre. En effet, lorsqu'une construction est définie, le système attend la sélection des objets désirés, et inversement, si des objets sont sélectionnés, les constructions disponibles sont restreintes à l'arité de ces objets.

Classiquement, un menu des constructions possibles permet de sélectionner la construction à effectuer. Pour faciliter le choix de la construction, les constructions correspondant aux objets déjà sélectionnés sont proposées à l'aide d'un menu contextuel. La main non-dominante peut y effectuer la sélection pendant que la main dominante se concentre sur la sélection d'objets (*cf.* figure 2.13). L'utilisateur peut ainsi travailler de manière efficace sans perdre du temps à rechercher une construction dans de grands menus hiérarchiques. Ainsi, si une construction a une arité différente de toutes les autres, elle est sélectionnée automatiquement sans que l'utilisateur intervienne. Cette manière de faire diminue l'imposante intrusion des boîtes à outils classiques (*cf.* figure 2.14) dans le champ de vision de l'utilisateur.

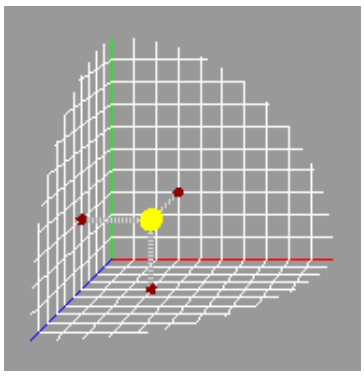


FIG. 2.15 – Grille magnétique régulière

Manipulation

L'utilisation de périphériques à 6 degrés de liberté permet de manipuler librement des objets dans l'espace. Mais les matériels utilisés ne permettent pas d'effectuer des manipulations précises et un outil d'interface spécifique doit permettre le placement exact d'objets.

De plus, dans le cadre des constructions géométriques, il y a plusieurs catégories d'objets : les objets libres, les objets partiellement définis et les objets fixés dans l'espace. La manipulation des objets sous-contraints mais non libres n'est pas en adéquation avec une manipulation à 6 degrés de liberté. Une interaction spécifique doit donc être mise en œuvre.

Pour résoudre ces deux problèmes, nous proposons d'avoir recours à deux outils virtuels : une grille magnétique régulière et un repère local augmenté.

La grille magnétique régulière présentée dans la figure 2.15, permet de placer les objets géométriques avec précision. Chaque objet est positionné sur un nœud de la grille et ne peut être déplacé que sur un autre nœud, à la manière d'aimants attirant l'objet.

La finesse de la grille peut être modifiée selon la distance entre la main non-dominante et le centre de la grille. Ainsi, cet outil permet de résoudre les problèmes de précisions lors du placement des objets.

Pour le problème de manipulation des objets partiellement définis, nous proposons de rassembler les poignées décrites précédemment, pour manipuler les objets sous-contraints, sur un repère local associé à l'objet. Ce repère local augmenté possède ainsi des poignées pour la mise à l'échelle, les rotations et les translations (*cf.* figure 2.16).

Ce repère n'est matérialisé qu'au moment où un objet est sélectionné. Ainsi, on peut agir indirectement sur l'objet grâce à ce repère. Les cônes sont utilisés pour représenter les degrés de translation, les sphères sont associées aux degrés de rotations et les cubes sont utilisés pour l'opération de mise à l'échelle.

Cela permet par exemple de déplacer finement le plan dans l'exemple d'intersection plan/cube pour visualiser comment varie le polygone d'intersection.

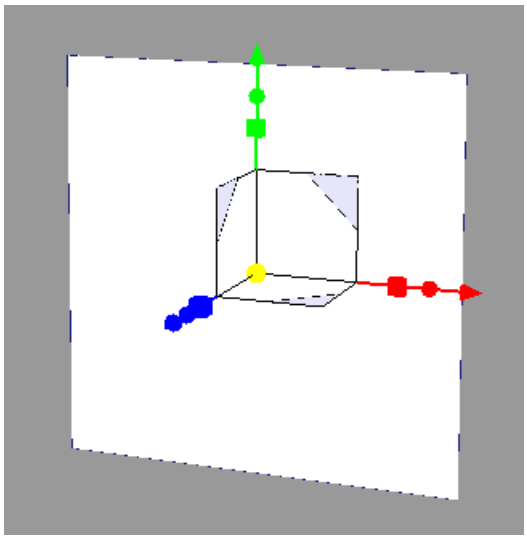


FIG. 2.16 – Plan et son repère local augmenté

Navigation

Dans le cadre des constructions géométriques, la navigation est un élément essentiel de l'interaction, puisqu'elle permet de se placer dans des positions permettant de visualiser les invariants. La recherche d'une position adéquate nécessite une navigation précise. De plus, dans une situation d'apprentissage, une navigation pédagogique conseillée par l'enseignant doit pouvoir être mise en place.

Le repère augmenté décrit précédemment peut être utilisé à des fins de navigation en considérant le repère canonique de la scène. Ainsi, en reprenant le même principe que pour le repère local augmenté, la navigation est effectuée en manipulant un widget de manière efficace et précise.

Pour la mise en place d'une navigation programmée, le dessin d'un chemin dans la scène et le choix de l'orientation d'une caméra le long de ce chemin doivent être possibles. Nous proposons d'utiliser une technique basée sur les mouvements des mains et la métaphore du rayon déformable [SB04]. L'utilisateur dessine simplement un chemin 3D avec un rayon de forme libre qui est une planification de la navigation.

La figure 2.17 présente la manipulation à deux mains du rayon déformable. Lorsque les mains sont proches l'une de l'autre, rien ne se passe. Si la main dominante s'éloigne de la main non-dominante ouverte, le rayon se forme en suivant le vecteur de direction défini par les deux mains. Si la main non-dominante est fermée, le rayon revient sur ses pas.

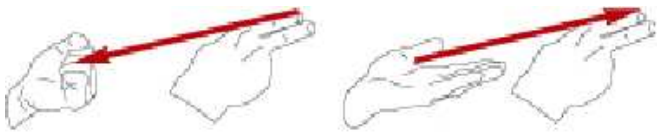


FIG. 2.17 – Manipulation du rayon déformable.

Dans l'exemple du cube coupé par un plan, l'enregistrement de ce chemin permet de rejouer la navigation pour montrer les 6 côtés de l'hexagone solution.

2.3.2 Univers isothétiques

Nous avons vu, au chapitre précédent, que des problèmes décrits par des droites 2D horizontales et verticales et des contraintes de distance peuvent être résolus dans un univers 1D pour chacun des types de droites (*cf.* univers *Bi1D* 1.3.1). Cela revient à considérer deux types de contraintes de distance différentes : les distances horizontales et les distances verticales.

On appelle *univers isothétique*, un univers où les axes du repère canonique servent à définir de tels objets et le jeu de contraintes associé.

Des solveurs 1D permettent de résoudre indépendamment les sous-systèmes obtenus en considérant chaque type de contraintes isothétiques. L'assemblage des solutions se fait simplement en faisant coïncider les repères isothétiques qui forment le repère canonique.

Pour plus de simplicité, des sortes homogènes sont utilisées dans les contraintes isothétiques. Ainsi, en 3D, on peut distinguer 3 univers isothétiques intégrant chacun des contraintes de distance définies suivant les axes Ox, Oy et Oz :

- l'univers des plans isothétiques, constitué de plans frontaux, de profils ou horizontaux ;
- l'univers des droites isothétiques, constitué de droites horizontales, verticales et de profondeur ;
- l'univers des points isothétiques, constitué de points.

Nous allons décrire ces trois univers isothétiques, et montrer que la résolution se ramène à une résolution dans l'univers des plans isothétiques. Nous présentons ensuite une interaction possible en considérant les plans. Des interfaces similaires peuvent être déduites pour les univers constitués de points ou de droites placés isothétiquement.

Présentation des univers

Plans isothétiques L'univers des plans isothétiques est composé de 3 sortes principales $front$, $profil$ et hor qui correspondent respectivement aux :

- plans de front $(x0y)$ d'équation $z = a$;
- plans de profils (yOz) d'équation $x = a$;

– plans horizontaux (xOz) d'équation $y = a$.

a étant un nombre réel qui constitue la sémantique de la sorte *scalaire*.

Cet univers est composé de 3 types de distance :

– les distances frontales entre plans de profils, *i.e.* parallèles à l'axe Ox ;
– les distances verticales entre plans horizontaux, *i.e.* parallèles à l'axe Oy ;
– les distances de biais entre plans de front, *i.e.* parallèles à l'axe Oz.

Les axiomes indiquent la commutativité des contraintes, la relation de Chasles et les règles de propagation à utiliser pour la résolution. Les symboles fonctionnels permettent de résoudre, par simple propagation dans chacun des espaces à 1 dimension ainsi générés, un système de contraintes exprimé dans cet univers.

Univers PlansIsothetiques
sortes :
front
profil
hor
scalaire
symboles fonctionnels :
$propage_{front}$: front scalaire $\rightarrow$ front
$propage_{profil}$: profil scalaire $\rightarrow$ profil
$propage_{hor}$: hor scalaire $\rightarrow$ hor
symboles prédicatifs :
$dist_{biais}$: front front scalaire
$dist_{front}$: profil profil scalaire
$dist_{vert}$: hor hor scalaire
axiomes :
`Commutativité :`
$\forall\ (f_1, f_2 : \text{front}) \wedge (\text{k} : \text{scalaire})\ dist_{biais}(f_1, f_2, k) = dist_{biais}(f_2, f_1, k)$
$\forall\ (p_1, p_2 : \text{profil}) \wedge (\text{k} : \text{scalaire})\ dist_{front}(p_1, p_2, k) = dist_{front}(p_2, p_1, k)$
$\forall\ (h_1, h_2 : \text{hor}) \wedge (\text{k} : \text{scalaire})\ dist_{vert}(h_1, h_2, k) = dist_{vert}(h_2, h_1, k)$
`Relation de Chasles :`
$\forall\ (f_1, f_2, f_3 : \text{front}) \wedge (k_1, k2, k3 : \text{scalaire})\ dist_{biais}(f_1, f_2, k_1) \wedge dist_{biais}(f_2, f_3, k2) \wedge dist_{biais}(f_1, f_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (p_1, p_2, p_3 : \text{profil}) \wedge (k_1, k2, k3 : \text{scalaire})\ dist_{front}(p_1, p_2, k_1) \wedge dist_{front}(p_2, p_3, k2) \wedge dist_{front}(p_1, p_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (h_1, h_2, h_3 : \text{hor}) \wedge (k_1, k2, k3 : \text{scalaire})\ dist_{vert}(h_1, h_2, k_1) \wedge dist_{vert}(h_2, h_3, k2) \wedge dist_{vert}(h_1, h_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
`Propagation :`
$\forall\ (f_1, f_2 : \text{front}) \wedge (k : \text{scalaire})\ dist_{biais}(f_1, f_2, k) \Rightarrow f_1 = propage_{front}(f_2, k)$
$\forall\ (p_1, p_2 : \text{profil}) \wedge (k : \text{scalaire})\ dist_{front}(p_1, p_2, k) \Rightarrow p_1 = propage_{profil}(p_2, k)$
$\forall\ (h_1, h_2 : \text{hor}) \wedge (k : \text{scalaire})\ dist_{vert}(h_1, h_2, k) \Rightarrow h_1 = propage_{hor}(h_2, k)$

Droites isothétiques Pour les droites isothétiques, l'univers associé est celui composé des 3 sortes suivantes :

- *hor*, pour les droites horizontales, d'équation $\begin{cases} y = a \\ z = b \end{cases}$;
- *vert*, pour les droites verticales, d'équation $\begin{cases} x = a \\ z = b \end{cases}$;
- *prof*, pour les droites en profondeur, d'équation $\begin{cases} y = a \\ x = b \end{cases}$.

avec a, b, des nombres réels de la sorte *scalaire*.

Une droite d'une sorte donnée est totalement définie par la donnée de deux types de contraintes :

- une droite horizontale doit être contrainte par une distance verticale et une distance de biais.
- une droite verticale doit être contrainte par une distance de front et une distance de biais.
- une droite de profondeur doit être contrainte par une distance verticale et une distance de front.

Il faut donc considérer 6 types différents de contraintes de distance en fonction des droites définies dans l'arité. Il faut bien noter que les contraintes entre des droites qui ne sont pas de mêmes sortes ne sont pas autorisées dans l'univers isothétique que nous considérons. Nous pourrions les prendre en compte, mais dans ce cas, on ne peut plus effectuer la résolution en sous-espaces de dimension 1.

Dans l'univers *DroitesIsothetiques*, la propagation est effectuée si deux distances pour une même sorte sont réunies. Étant donné qu'une droite 3D est l'intersection de deux plans, la résolution peut se ramener à une résolution dans l'univers des plans isothétiques. Ainsi, pour un type de droite donnée, une droite est totalement définie lorsque l'assemblage de deux solutions 1D est réalisé.

Univers DroitesIsothetiques
sortes :
hor
vert
prof
scalaire
symboles fonctionnels :
$propage_{hor}$: hor hor scalaire scalaire → hor
$propage_{vert}$: vert vert scalaire scalaire → vert
$propage_{prof}$: prof prof scalaire scalaire → prof
symboles prédicatifs :
$dist_{v_1}$: hor hor scalaire
$dist_{b_1}$: hor hor scalaire
$dist_{f_1}$: vert vert scalaire
$dist_{b_2}$: vert vert scalaire
$dist_{f_2}$: prof prof scalaire

$dist_{v_2}$: prof prof scalaire
axiomes :
`Commutativité :`
$\forall\ (h_1, h_2 : \text{hor}) \wedge (\text{k} : \text{scalaire})\ dist_{v_1}(h_1, h_2, k) = dist_{v_1}(h_2, h_1, k)$
$\forall\ (h_1, h_2 : \text{hor}) \wedge (\text{k} : \text{scalaire})\ dist_{b_1}(h_1, h_2, k) = dist_{b_1}(h_2, h_1, k)$
$\forall\ (v_1, v_2 : \text{vert}) \wedge (\text{k} : \text{scalaire})\ dist_{f_1}(v_1, v_2, k) = dist_{f_1}(v_2, v_1, k)$
$\forall\ (v_1, v_2 : \text{vert}) \wedge (\text{k} : \text{scalaire})\ dist_{b_2}(v_1, v_2, k) = dist_{b_2}(v_2, v_1, k)$
$\forall\ (p_1, p_2 : \text{prof}) \wedge (\text{k} : \text{scalaire})\ dist_{f_2}(p_1, p_2, k) = dist_{f_2}(p_2, p_1, k)$
$\forall\ (p_1, p_2 : \text{prof}) \wedge (\text{k} : \text{scalaire})\ dist_{v_2}(p_1, p_2, k) = dist_{v_2}(p_2, p_1, k)$
`Relation de Chasles :`
$\forall\ (h_1, h_2, h_3 : \text{hor}) \wedge (k_1, k_2, k_3 : \text{scalaire})\ dist_{v_1}(h_1, h_2, k_1) \wedge dist_{v_1}(h_2, h_3, k_2) \wedge$
$dist_{v_1}(h_1, h_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (h_1, h_2, h_3 : \text{hor}) \wedge (k_1, k_2, k_3 : \text{scalaire})\ dist_{b_1}(h_1, h_2, k_1) \wedge dist_{b_1}(h_2, h_3, k_2) \wedge$
$dist_{v_1}(h_1, h_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (v_1, v_2, v_3 : \text{vert}) \wedge (k_1, k_2, k_3 : \text{scalaire})\ dist_{f_1}(v_1, v_2, k_1) \wedge dist_{f_1}(v_2, v_3, k_2) \wedge$
$dist_{f_1}(v_1, v_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (v_1, v_2, v_3 : \text{vert}) \wedge (k_1, k_2, k_3 : \text{scalaire})\ dist_{b_2}(v_1, v_2, k_1) \wedge dist_{b_2}(v_2, v_3, k_2) \wedge$
$dist_{b_2}(v_1, v_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (p_1, p_2, p_3 : \text{prof}) \wedge (k_1, k_2, k_3 : \text{scalaire})\ dist_{f_2}(p_1, p_2, k_1) \wedge dist_{f_2}(p_2, p_3, k_2) \wedge$
$dist_{f_2}(p_1, p_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (p_1, p_2, p_3 : \text{prof}) \wedge (k_1, k_2, k_3 : \text{scalaire})\ dist_{v_2}(p_1, p_2, k_1) \wedge dist_{v_2}(p_2, p_3, k_2) \wedge$
$dist_{v_2}(p_1, p_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
`Propagation :`
$\forall\ (h_1, h_2, h_3 : \text{hor}) \wedge (k_1, k_2 : \text{scalaire})\ dist_{v_1}(h_1, h_2, k_1) \wedge dist_{b_1}(h_1, h_3, k_2)$
$\Rightarrow h_1 = propage_{hor}(h_2, h_3, k_1, k_2)$
$\forall\ (v_1, v_2, v_3 : \text{vert}) \wedge (k_1, k_2 : \text{scalaire})\ dist_{f_1}(v_1, v_2, k_1) \wedge dist_{b_2}(v_1, v_3, k_2)$
$\Rightarrow v_1 = propage_{vert}(v_2, v_3, k_1, k_2)$
$\forall\ (p_1, p_2, p3 : \text{prof}) \wedge (k_1, k_2 : \text{scalaire})\ dist_{f_2}(p_1, p_2, k_1) \wedge dist_{v_2}(p_1, p_3, k_2)$
$\Rightarrow p_1 = propage_{prof}(p_2, p_3, k_1, k_2)$

Points isothétiques L'univers des points isothétiques est composé de points et de distances isothétiques.

Un point étant l'intersection de trois plans, la résolution peut s'appliquer indépendamment suivant les trois coordonnées d'un point.

Univers PointsIsothétiques
sortes :
point
longueur
symboles fonctionnels :
propage : point longueur point longueur point longueur → point
symboles prédicatifs :
$dist_{hor}$: point point longueur
$dist_{vert}$: point point longueur
$dist_{biais}$: point point longueur
axiomes :
`Commutativité :`

$\forall\ (p_1, p_2 : \text{point}) \wedge (\text{k} : \text{longueur})\ dist_{hor}(p_1, p_2, k) = dist_{hor}(p_2, p_1, k)$
$\forall\ (p_1, p_2 : \text{point}) \wedge (\text{k} : \text{longueur})\ dist_{vert}(p_1, p_2, k) = dist_{vert}(p_2, p_1, k)$
$\forall\ (p_1, p_2 : \text{point}) \wedge (\text{k} : \text{longueur})\ dist_{biais}(p_1, p_2, k) = dist_{biais}(p_2, p_1, k)$
`Relation de Chasles :`
$\forall\ (p_1, p_2, p_3 : \text{point}) \wedge (k_1, k2, k3 : \text{longueur})\ dist_{hor}(p_1, p_2, k_1) \wedge dist_{hor}(p_2, p_3, k2)$
$\wedge dist_{hor}(p_1, p_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (p_1, p_2, p_3 : \text{point}) \wedge (k_1, k2, k3 : \text{longueur})\ dist_{vert}(p_1, p_2, k_1) \wedge dist_{vert}(p_2, p_3, k2)$
$\wedge dist_{vert}(p_1, p_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 = k_1 + k_3$
$\forall\ (p_1, p_2, p_3 : \text{point}) \wedge (k_1, k2, k3 : \text{longueur})\ dist_{biais}(p_1, p_2, k_1) \wedge$
$dist_{biais}(p_2, p_3, k2) \wedge dist_{biais}(p_1, p_3, k_3) \Rightarrow k_3 = k_1 + k_2 \vee k_1 = k_2 + k_3 \vee k_2 =$
$k_1 + k_3$
`Propagation :`
$\forall\ (p_1, p_2, p_3, p_4 : \text{points}) \wedge (k_1, k_2, k_3 : \text{longueur})\ dist_{hor}(p_1, p_2, k_1) \wedge$
$dist_{vert}(p_1, p_3, k_2) \wedge dist_{biais}(p_1, p_4, k_3) \Rightarrow p_1 = propage(p_2, k_1, p_3, k_2, p_4, k_3)$

En fonction de l'univers géométrique choisi, la même résolution de contraintes peut être appliquée mais les objets modélisés sont différents : les directrices ou les plans directeurs en dessin architectural, ou des objets en fil de fer (*cf.* figure 2.18).

Résolution de contraintes isothétiques

Comme nous l'avons dit précédemment, la résolution associée à ce type d'univers est très simple. On se retrouve à décomposer la résolution de problèmes 3D en résolutions 1D puisque les contraintes posées sont groupées dans une dimension unaire : les distances isothétiques. Il suffit de dissocier les distances horizontales, verticales, et de biais. Nous montrons comment cela s'effectue de manière pratique pour les contraintes de distance entre plans isothétiques puisque les univers *DroitesIsothetiques* et *PointsIsothetiques* peuvent s'exprimer dans l'univers *PlansIsothetiques*.

Nous pouvons décrire l'algorithme de résolution sur 3 graphes. Chacun contient des nœuds correspondant à des plans d'un type donné. Des arêtes valuées par la distance entre deux plans indiquent l'existence d'une contrainte.

Les nœuds contiennent une valeur qui est la position du plan sur l'axe considéré. Par exemple, dans le graphe des contraintes verticales, ces valeurs sont des ordonnées sur l'axe Oy car les plans considérés sont du type $y = a$.

De plus, l'adjonction d'une contrainte entre deux plans doit vérifier la relation de Chasles. Ceci est facilement détecté puisque cela équivaut à la création d'un cycle.

Finalement, la résolution consiste en une simple propagation des valeurs à l'aide des règles définies dans l'axiomatique de l'univers *PlansIsothetiques*. La disposition des plans les uns par rapport aux autres dans l'esquisse permet de déterminer si l'on choisit une addition ou une soustraction des valeurs des nœuds et des arêtes dans le cas de distance non signées.

L'intersection de plans de sortes différentes permet de construire des segments et des points qui forment des figures parallélépipédiques (*cf.* figure 2.18).

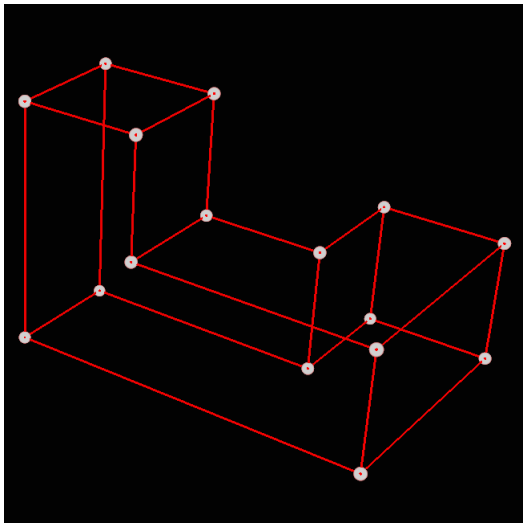


FIG. 2.18 – Objet 3D isothétique

Interaction isothétique

L'interaction que nous avons mise en place dans le cadre des univers isothétiques est combinée à une résolution incrémentale des contraintes posées. Nous ne décrivons que l'interaction concernant l'univers composé de plans isothétiques ; l'adaptation aux autres univers isothétiques se révèle aisée.

Les plans sont représentés par des quadrilatères transparents dont la manipulation des coins permet le redimensionnement. Les contraintes de distance sont représentées par une flèche à double sens (figure 2.19) dont les extrémités sont manipulables pour modifier la valeur des contraintes.

FIG. 2.19 – Contrainte de distance

Sélection La sélection des objets de l'interface s'effectue à l'aide d'une seule main, celle si pouvant être la main dominante ou la main non-dominante. La désignation du plan se fait par le geste de la main ouverte (figure 2.8(c)) et celle des contraintes par le geste où seuls le pouce et l'auriculaire sont levés (figure 2.20). La sélection est validée lorsque la main est fermée.

L'orientation des mains permet de faire la différence entre les trois types de plans ou les trois types de contraintes.

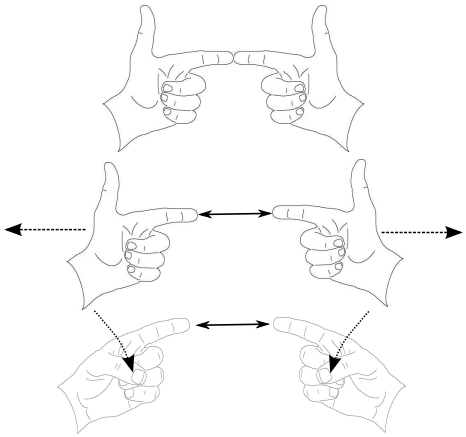


FIG. 2.20 – Geste pour la pose d'une contrainte isothétique

Création La création des objets nécessite une gestuelle à deux mains. Un plan est ajouté à la scène à l'endroit où l'utilisateur tape dans ses mains ouvertes. Inversement, l'éloignement progressif des mains jointes par les index jusqu'à sélectionner les deux plans concernés en rabattant le pouce permet de saisir une contrainte isothétique (*cf.* figure 2.20). De la même manière que pour la sélection, l'orientation de la main détermine le type de plan ou de contrainte à créer.

Manipulation Seules les translations isothétiques applicables sur des objets sont possibles après leur sélection. Les coins des plans, et les extrémités des contraintes sont des zones dédiées à la modification des propriétés internes des objets. Ces zones réagissent à une interaction bi-manuelle qui par éloignement ou rapprochement des mains permet de modifier soit la taille des plans soit la valeur de la contrainte assimilée à la longueur de sa représentation.

Navigation La navigation dans ce prototype est effectuée à l'aide d'un repère canonique augmenté dont le principe de fonctionnement est identique au repère local augmenté présenté plus haut.

Les univers isothétiques permettent de se rendre compte de la dépendance existante entre résolution et interaction. En effet, les choix effectués pour composer l'interface et la manière d'interagir permettent d'utiliser des méthodes de résolution adéquates. Nous allons considérer dans la suite un univers plus général où la définition d'une interaction basée sur les gestes permet de poser intuitivement des contraintes sur une esquisse.

2.3.3 Pose de contraintes sur une esquisse

Nous avons développé le prototype *CadVR* permettant de poser des contraintes de distance, d'angle et de coplanarité en 3D dans un univers contenant les sortes *points*, *droites* et *plans*.

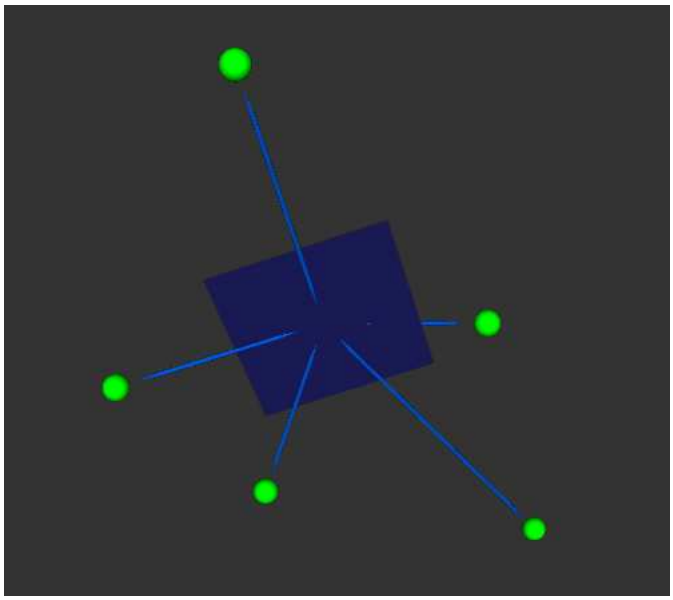


FIG. 2.21 – Contrainte de coplanarité pour 5 points

La représentation des contraintes de distance est identique à l'interface isothétique décrite ci-dessus. Les coplanarités sont représentées par un quadrilatère transparent désignant le plan formé par les points contraints, et par des segments reliant les points concernés à ce plan (*cf.* figure 2.21). Les angles sont représentés par des arcs de cercle entre les objets concernés. Pour des raisons d'occlusions générées par le quadrilatère, sa taille doit être réduite. Son orientation quant à elle est définie à partir des trois premiers points sélectionnés.

L'utilisation d'une interaction gestuelle 3D couplée aux outils techniques virtuels décrits précédemment permet de construire des problèmes de contraintes de manière précise, en satisfaisant au niveau de l'interface les contraintes d'incidence grâce à une approche de géométrie dynamique.

La sélection des objets se fait grâce au dictionnaire de gestes présenté dans la figure 2.8.

Création La pose de contraintes nécessite généralement le recours à un menu hiérarchique comme dans les interfaces classiques.

Afin de se dispenser de ces menus devenus trop encombrants, nous proposons ici d'esquisser à l'aide de gestes les contraintes sans avoir à sélectionner d'objets géométriques.

La pose de contraintes se fait en dessinant dans l'espace comme avec un crayon, à l'aide du geste de sélection par défaut (*cf.* figure 2.22). Ainsi, le dessin effectué par la main permet de distinguer le type des contraintes. Les objets les plus proches de la courbe sont les objets concernés par la contrainte.

Un mouvement rectiligne du doigt spécifie une distance entre les objets les plus proches des extrémités. Un trait courbe spécifie un angle entre les objets les plus proches. Un trait définissant une courbe fermée indique une coplanarité entre les objets les plus proches de la forme dessinée.

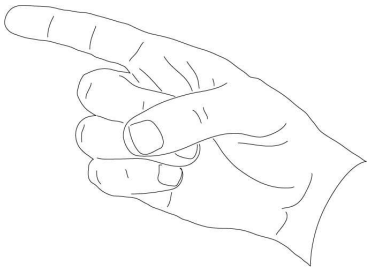


FIG. 2.22 – Posture pour le dessin de contraintes

Valeurs numériques L'utilisation d'un clavier physique est difficile avec des gants de données. Entrer les valeurs numériques associées aux contraintes est donc problématique. De la même manière que dans l'univers isothétique, ce problème peut se régler par la manipulation bi-manuelle des extrémités des représentations des contraintes : éloigner les mains augmente la valeur et les rapprocher la diminue. Par défaut, la valeur des contraintes est celle mesurée au moment d'esquisser la contrainte. L'entrée de valeurs numériques pourrait se faire au moyen d'un clavier numérique virtuel ou par un widget dédié, mais nous n'en avons pas mesuré la commodité.

Conclusion du chapitre

Dans ce chapitre, nous avons montré qu'il est possible d'appliquer les technologies issues du domaine de la réalité virtuelle à la modélisation par contraintes géométriques en 3D. En effet, une interaction réellement en 3D a été appliquée aux contraintes 3D. Nous avons proposé une interaction gestuelle spécifique et une intégration des contraintes dans le modeleur à l'aide d'un gestionnaire de contraintes.

Trois prototypes ont été présentés. Le premier se nomme Coyote-géomètre et concerne la géométrie dynamique en 3D pour l'enseignement assisté par ordinateur. Le second est un modeleur par contraintes basé sur les contraintes isothétiques orienté pour la conception architecturale. Enfin, le troisième prototype propose la pose de contraintes géométriques de distance, d'angle et de coplanarité pour l'expression de systèmes de contraintes 3D.

Pour le moment, il n'existe pas d'interaction 3D normalisée. La prochaine étape pour poursuivre ce travail est donc la réalisation de tests pour valider les interfaces proposées dans ce chapitre. Le but final consiste à concevoir une interface adaptée à la modélisation par contraintes 3D. Pour ce faire, nous comptons proposer une interface multi-modale avec l'adjontion d'autres types d'interactions, comme par exemple l'interaction vocale.

L'étude de l'interface et de l'interaction pour des systèmes de contraintes 3D nous a permis de débuter la réalisation d'un solveur de contraintes isothétiques qui permet de simplifier la résolution dans un cadre bien particulier. Mais pour des systèmes qui s'expriment dans un univers plus complexe, il est nécessaire de recourir à une méthode de résolution plus puissante.

Chapitre 3

Résolution

Les méthodes qui semblent actuellement les plus prometteuses pour résoudre des problèmes de contraintes géométriques en 3D permettent de combiner finement les approches numériques et formelles. Ainsi, la méthode par intersection de lieux géométriques développée par Gao et al. *essaie de s'affranchir dans une certaine mesure des problèmes posés par l'irréductibilité combinatoire de petits systèmes de contraintes. Tout en suivant l'idée d'assemblage de grappes, ils énumèrent et résolvent presque tous les problèmes comprenant moins de six contraintes dans l'univers contenant points, droites et plans en 3D avec les contraintes de distance et d'angle. Cette méthode, initiée en même temps que nos propres travaux, est un cas particulier de ce que nous avons appelé la* reparamétrisation, *méthode hybride permettant de résoudre des problèmes 3D irréductibles. Nous avons nommé ces problèmes des systèmes* k-quasi-décomposables *par rapport à une méthode de décomposition donnée et nous proposons une mise en œuvre de la reparamétrisation mieux adaptée aux problèmes de taille plus importante. L'étude de la méthode de Gao* et al. *et la formalisation de la reparamétrisation sont exposées dans les deux premières sections, et une mise en œuvre est développée dans la troisième. Les résultats des expériences menées sur la classe des polyèdres sont proposés dans la quatrième et dernière section.*

Sommaire

3.1 Approche par intersection de lieux géométriques

Nous avons vu dans le chapitre 2, comment les outils de tracé classiques, tels que la règle et le compas, ont donné lieu, en informatique géométrique, à des outils de construction définissant des *micros-mondes de géométrie* et comment, par la suite, ceux-ci ont évolué pour aboutir au concept de *géométrie dynamique*.

Dans les logiciels de géométrie dynamique, lorsque l'utilisateur fait varier les positions des objets libres, la construction est rejouée sur les nouvelles valeurs des paramètres permettant ainsi, lorsque la variation des objets libres se fait de manière quasi continue, d'animer des figures et de mettre en évidence certains invariants. Si la figure obtenue ne satisfait pas l'utilisateur, il essaie de bouger certains éléments pour se retrouver dans la « bonne » configuration.

Ces paramètres (distances, angles, *etc.*) qui n'apparaissent pas explicitement dans l'énoncé peuvent être utilisés pour transformer le système de contraintes initial en un système plus facilement résoluble. L'enjeu de cette méthode est de déterminer ces nouvelles contraintes. Cette idée de déformation d'une figure pour satisfaire de nouvelles propriétés peut être mise à contribution dans le cadre de la résolution de contraintes.

En effet, il existe des systèmes minimaux irréductibles en 3D empêchant une méthode de construction géométrique de s'appliquer. Il faut se replacer dans un cas favorable connu, dans lequel on peut trouver un plan de construction de la figure avec des paramètres, comme par exemple des points, à faire varier. Pour cela, il suffit de fixer un certain nombre d'inconnues *a priori* (c'est-à-dire, de les considérer comme des paramètres). Une fois la construction effectuée, il reste un certain nombre de contraintes à satisfaire, et il ne reste plus qu'à les « rattraper », en faisant varier les inconnues temporairement fixées pour avoir la bonne solution.

Cette méthode utilisée pour permettre une résolution par intersection de lieux géométriques par Gao *et al.* [GHY02, GHY04], a été initiée en même temps que nos propres travaux, et se rapproche de la phase d'intersection dans la méthode de Kramer (*cf.* chapitre 1) pour des chaînes articulées de mécanismes.

La démarche de Gao et *al.* est un peu différente de la nôtre. En effet, ils ont commencé par considérer variables et systèmes d'équations pour résoudre de manière hybride des problèmes irréductibles en 3D.

Il nous semble important de bien comprendre les méthodes proposées par Gao *et al.* puisque nous allons en reprendre certains ingrédients. Nous présentons donc les deux variantes de leur méthode de manière plus formelle que cela n'est fait dans les deux papiers cités.

3.1.1 Première méthode de Gao *et al.*

Rappelons que nous appelons système de contraintes dans un univers géométrique donné un triplet $\langle C, \chi, A\rangle$ où C est un ensemble de termes prédicatifs construits dans l'univers, dont les variables appartiennent à $\chi \cup A$. χ est l'ensemble des inconnues et A l'ensemble des paramètres ; on impose, bien sûr que $\chi \cap A = \emptyset$. Les solutions

sont cherchées dans un ensemble construit sur la Σ-algèbre associée à l'univers géométrique.

Dans un premier temps, Gao *et al.* considèrent un système d'équations sans paramètres : les contraintes sont des équations à inconnues réelles. Soit

$$S = \langle \left\{ \begin{array}{l} F_1(x_1, \ldots, x_n) = 0 \\ \ldots \\ F_n(x_1, \ldots, x_n) = 0 \end{array} \right. , \{x_1, \ldots, x_n\}, \emptyset \rangle$$

un tel système qu'on suppose bien contraint et irréductible au sens de Dulmage-Mendelson (*cf.* théorème 1.3.3).

Fixer une inconnue, ou, plus précisément, donner à celle-ci le statut de paramètre, produit un système sur-contraint. En tout état de cause, on doit avoir une partie bien contrainte qui se décompose en composantes irréductibles mettant en jeu $n-1$ équations et les $n-1$ inconnues restantes. Ainsi, on met une équation de côté, et on peut espérer que le système résultant soit décomposable : au besoin, on peut essayer chacune des inconnues et des équations possibles et voir avec quelle configuration on a le meilleur résultat.

Supposons, sans perte de généralité, que x_n et F_n soient respectivement l'inconnue devenue paramètre et l'équation « mise de coté ». Le système obtenu $S^{(1)}$ est donc de la forme :

$$S^{(1)} = \langle \left\{ \begin{array}{l} F_1(x_1, \ldots, x_{n-1}, u) = 0 \\ \ldots \\ F_{n-1}(x_1, \ldots, x_{n-1}, u) = 0 \end{array} \right. , \{x_1, \ldots, x_{n-1}\}, \{u\} \rangle$$

Une solution à un tel système est une *fonction* de $\mathbb{R}$ dans $\mathbb{R}^{n-1}$ qui à toute valeur de u associe un $n-1$-uplet de valeurs pour $(x_1, \ldots x_{n-1})$ de sorte que $S^{(1)}$ soit satisfait.

Soit $f(u)$ une telle fonction : tout n-uplet $(f(u), u)$ est « presque » une solution de S puisque, par construction, il satisfait les $n-1$ premières équations. On peut donc faire varier u de manière à satisfaire l'équation en u : $F_n(f(u), u) = 0$, pour obtenir une solution de S.

La question de la correction et la complétude de cette méthode n'est pas abordée dans les articles mentionnés plus haut. Nous les examinons avant de nous pencher sur la manière *effective* de résoudre les questions posées par cette méthode.

Correction

Soit le système :

$$S^{(2)} = \langle \begin{cases} F_1(x_1, \ldots, x_{n-1}, u) = 0 \\ \ldots \\ F_{n-1}(x_1, \ldots, x_{n-1}, u) = 0 \\ F_n(x_1, \ldots, x_{n-1}, u) = 0 \\ x_n = u \end{cases} , \{x_1, \ldots, x_n\}, \{u\} \rangle$$

$S^{(2)}$ est évidemment décomposable en :
- $S^{(3)} = S^{(1)} \cup \langle \{F_n(x_1, \ldots, x_{n-1}, u), \{x_1, \ldots, x_{n-1}\}, \{u\} \rangle$;
- et $S^{(4)} = \langle \{x_n = u\}, \{x_n\}, \{u\} \rangle$.

Soit $f(u)$ une solution de $S^{(1)}$. Si $f(u)$ est une solution de $S^{(3)}$, alors $(f(u), u)$ est une solution de $S^{(2)}$ car il est trivial que u soit une solution de $S^{(4)}$. Il est clair que $(f(u), u)$ est une solution de S puisque S et $S^{(2)}$ sont les mêmes systèmes à une réécriture près.

Complétude

La question de la complétude (est-ce que toute solution du système initial peut être trouvée par cette méthode ?) est un peu moins évidente et passe par des compléments de définition.

L'ensemble des solutions d'un système d'équations ou de contraintes paramétrées est un ensemble fonctionnel peu utilisable directement.

On dit qu'un système de contraintes $\langle C, \chi, A \rangle$ est bien contraint s'il existe un nombre fini de fonctions des paramètres qui permettent de trouver toutes les solutions en χ pour les valeurs du paramètre u données.

Mais, évidemment, il arrive souvent qu'il y ait un nombre infini de telles fonctions. Nous dirons qu'un ensemble de fonctions solutions d'un tel système est complet si et seulement s'il permet de trouver toutes les solutions pour toute valeur donnée de u.

Un système de contraintes est donc bien contraint si et seulement s'il admet un ensemble fini et complet de solutions paramétrées.

Il est clair que le fait que S soit bien contraint n'implique pas que $S^{(1)}$ soit bien contraint : il suffit de considérer le contre-exemple suivant :

▷ **Exemple 3.1.1**

$$S = \langle \{x_2 = 0, x_1 = 0\}, \{x_1, x_2\}, \emptyset \rangle$$

qui donne

$$S^{(1)} = \langle \{u = 0\}, \{x_1\}, \{u\} \rangle$$

est sur-contraint suivant la définition donnée plus haut. ◁

Cependant, on a la propriété suivante :

Proposition 3.1.1 Avec les notations précédentes, si S est un système bien contraint et si on considère un changement « inconnue vers paramètre » tel que $S^{(1)}$ est bien contraint, alors la méthode de Gao *et al.* est complète. □

En effet, soit $s = (v_1, \ldots, v_{n-1}, v_n)$ une solution de S, s est forcément une solution du système

$$S_1 = \langle \begin{cases} F_1(x_1, \ldots, x_n) = 0 \\ \ldots \\ F_{n-1}(x_1, \ldots, x_n) = 0 \end{cases} , \{x_1, \ldots, x_n\}, \emptyset \rangle$$

.

Donc, si $\{f_1, \ldots f_k\}$ est un ensemble complet de solutions de $S^{(1)}$, il existe une fonction, disons f_1, telle que $f_1(v_n) = (v_1, \ldots, v_{n-1})$. Finalement, on a :

$F_n(f_1(v_n), v_n) = 0$ puisque $s = (v_1, \ldots, v_{n-1}, v_n)$ est une solution de S. Donc, il existe une solution de $S^{(1)}$ et une solution en u de $F_n(f(u), u) = 0$ permettant de retrouver la solution cherchée de S.

Solutions formelles

Ainsi, pour trouver toutes les solutions de S, il faut obtenir :
- toutes les solutions de $S^{(1)}$, c'est-à-dire un ensemble de fonctions paramétrées par u capable de produire pour toute valeur de u l'ensemble des solutions de $S^{(1)}$,
- toutes les solutions des équations $F_n(f(u), u) = 0$ où f est une solution de $S^{(1)}$ et u est l'inconnue.

À notre sens, la manière la plus pratique de faire consiste à résoudre *formellement* le système $S^{(1)}$.

En effet, les fonctions $f(u)$ peuvent être exprimées formellement sous forme de plans de construction interprétables pour des valeurs de u données, ce qui permet de résoudre plus facilement les équations du type $F_n(f(u), u) = 0$.

Mais, lorsqu'on envisage une résolution formelle de $S^{(1)}$, on pose des contraintes supplémentaires très fortes sur le choix de la variable à considérer comme un paramètre puisque le système $S^{(1)}$ doit être résoluble dans l'univers considéré, c'est-à-dire qu'on doit pouvoir au minimum *exprimer* les solutions formelles dans le langage de l'univers géométrique et qu'on doit disposer d'un solveur formel capable de les trouver. Nous verrons plus bas que ce point, qui est un peu laissé de coté dans l'article original puisque Gao et *al.* résolvent numériquement, revêt une importance capitale et s'intègre dans notre cadre géométrique paramétré.

Avec une telle méthode, résoudre les équations $F_n(f(u), u) = 0$ peut se faire rapidement avec des méthodes numériques efficaces comme la dichotomie, Newton-Raphson (on peut alors perdre la complétude) ou même l'échantillonnage. Mais, on peut éventuellement imaginer un autre mécanisme où le système $S^{(1)}$ est résolu à la demande pour une valeur de u fixée.

3.1.2 Deuxième méthode de Gao *et al.*

La méthode que nous venons de décrire est, finalement, assez éloignée du contexte géométrique de la CAO. En effet, d'une part la transformation d'un système de contraintes en système d'équations introduit le biais de la traduction qui peut être très important : les inconnues qui en résultent ne sont peut-être pas toujours les plus pertinentes, et, d'autre part, cette méthode n'intègre pas du tout l'invariance par déplacement qui est, comme nous l'avons vu, une des composantes fondamentales des méthodes de décomposition géométrique.

En posant :

$$S' = \langle \begin{cases} F_1(x_1, \ldots, x_n) = 0 \\ \ldots \\ F_{n-1}(x_1, \ldots, x_n) = 0 \\ x_n = u \end{cases} , \{x_1, \ldots, x_n\}, \{u\} \rangle$$

on s'aperçoit que la méthode décrite plus haut est un cas particulier de la méthode consistant dans un système d'équations à remplacer une équation de S, ici $F_n(x_1, \ldots, x_n) = 0$, par une autre, ici $x_n = u$, pour donner le système S', avec la différence notable que ceci peut se réaliser avec des contraintes géométriques et qu'on peut ainsi conserver l'invariance par déplacement et les méthodes de décompositions afférentes.

C'est ce qu'ont fait Gao *et al.* dans leur deuxième méthode qui, étant donné un système de contraintes sans paramètres $S = \langle C, \chi, \emptyset \rangle$, ou plus précisément

$$S = \langle \begin{cases} c_1(o_1, \ldots, o_n) \\ \ldots \\ c_m(o_1, \ldots, o_n) \end{cases} , \{o_1, \ldots, o_n\}, \emptyset \rangle$$

consiste à remplacer une contrainte, disons c_m, par une autre contrainte avec un paramètre u pour donner le système $S' = \langle C \cup \{d\} \setminus \{c_m\}, X, \{u\} \rangle$, ou de manière plus lisible,

$$S' = \langle \begin{cases} c_1(o_1, \ldots, o_n) \\ \ldots \\ c_{m-1}(o_1, \ldots, o_n) \\ d(o_1, \ldots, o_n, u) \end{cases} , \{o_1, \ldots, o_n\}, \{u\} \rangle$$

Comme précédemment, ce système est résolu formellement pour produire des solutions de la forme $f : u \mapsto (o_1, \ldots, o_n)$ qui servent à produire le système à une seule contrainte en l'inconnue u, $U = \langle \{c_m(f(u))\}, \{u\}, \emptyset \rangle$ qui est lui-même résolu par échantillonnage (dans les cas considérés par Gao *et al.*, le paramètre u est réel, c_m se traduit par une équation et l'échantillonnage correspond à un tracé de courbe).

Si $f(u)$ est une solution de S' et si u_0 est une solution de U, alors les auteurs posent que $f(u_0)$ est une solution de S, autrement dit, que leur méthode est correcte.

Une démonstration du même style que celle proposée plus haut montre effectivement que cette manière de faire est correcte. En revanche, la question de la

complétude, qui est implicitement admise par Gao *et al.*, se pose.

Complétude Pour les mêmes raisons que dans la démonstration de complétude de la section précédente, on doit tout d'abord supposer que le système S' est bien contraint (ce qui est une hypothèse très forte pour les systèmes paramétrés, mais qui est en général vérifiée). Si on considère ensuite une solution quelconque $(v_1, \dots v_n)$ de S, alors c'est évidemment une solution du système sans paramètre :

$$S_1 = \langle \begin{cases} c_1(o_1, \dots, o_n) \\ \dots \\ c_{m-1}(o_1, \dots, o_n) \end{cases} , \{o_1, \dots, o_n\}, \emptyset \rangle$$

En revanche, rien ne garantit qu'il existe une valeur u_0 telle que $d(v_1 \dots, v_n, u_0)$ soit vérifiée, même si cela est vérifié dans tous les exemples présentés dans leur article.

C'est pourquoi, nous posons la définition :

> **Définition 3.1.2** Une contrainte paramétrée $d(x_1, \dots, x_n, u)$ est non occlusive, si pour toute valeur $v_1, \dots, v_n$ des inconnues, il existe une valeur u_0 telle que $d(v_1, \dots, v_n, u_0)$ soit vérifiée.

Par exemple, la contrainte $x_n = u$ utilisée dans la première méthode décrite plus haut, est non occlusive.

Il découle de cette définition, en suivant la preuve de la section précédente, que si :

- S est bien contraint ;
- S' est bien contraint ;
- et d est une contrainte paramétrée non occlusive ;

alors la méthode décrite ici est complète, *i.e.* on obtient toutes les solutions de S en considérant toutes les solutions de S' et toutes les solutions de U.

3.1.3 Problèmes avec k $\geq$ 1

La première et la deuxième méthode diffèrent par le type de contrainte à rajouter ($x_n = u$ dans la première et n'importe quelle contrainte) mais surtout par le nombre de contraintes qu'il est possible de remplacer (1 dans la première et un nombre quelconque dans la seconde). Mais sur ce dernier point, la mise en œuvre sur des exemples n'a été appliquée que pour une contrainte.

La génération automatique des systèmes acceptant au plus 6 contraintes dans l'univers des points, droites et plans 3D, leur a permis de valider expérimentalement cette méthode.

La méthode utilisée pour choisir la contrainte à remplacer et la nouvelle contrainte à considérer résulte d'une stratégie de recherche exhaustive qui permet d'assurer qu'on est dans de bonnes conditions pour appliquer la méthode. En revanche, ces stratégies de « force brute » s'accommodent mal de la généralisation consistant à remplacer k contraintes.

En effet, pour un problème où il faudrait remplacer k contraintes :

- la recherche exhaustive du meilleur ensemble de contraintes à remplacer a une complexité en $O(n^k)$;
- leur manière de produire la fonction $f(u_1, \ldots, u_k)$ n'est pas correcte dans le cas général (*cf.* contre-exemple à la section 1.3.2) ;
- l'échantillonnage implique de choisir correctement l'intervalle $[a, b]$ et le pas s pour une bonne approximation des solutions et la complexité est en $O(L\frac{b-a}{s})^k$.

3.2 Reparamétrisation

La seconde méthode de Gao *et al.* peut être généralisée à une technique de résolution de systèmes 3D irréductibles que nous avons nommée la *reparamétrisation*.

Cette section propose de mettre en évidence les points clés de cette méthode et de définir un schéma de résolution. Nous donnons notamment la définition des systèmes pouvant être reparamétrés par rapport à une méthode de décomposition donnée. Nous avons nommé ces problèmes des problèmes k-quasi-décomposables lorsque le remplacement de k contraintes est nécessaire.

3.2.1 k-transmutation

En plus des opérations usuelles sur les systèmes de contraintes, nous introduisons l'opération de transmutation qui transforme un système de contraintes en un autre possédant les mêmes inconnues mais des paramètres différents. Plus précisément, nous posons :

> **Définition 3.2.1** Une *k-transmutation* est une opération qui transforme un système $S = \langle C, \chi, A\rangle$ en un autre système $S' = \langle C', \chi, A'\rangle$ en remplaçant k contraintes de C par k autres contraintes portant sur les mêmes inconnues pour donner C'.

Cette opération est une composition d'autres opérations classiques plus élémentaires définies par le choix des contraintes à enlever et à ajouter. En général, un système obtenu par k-transmutation d'un autre ne lui est pas équivalent.

Nous allons considérer que si les contraintes retirées contiennent des paramètres propres, ceux-ci sont retirés et si les contraintes ajoutées contiennent des paramètres, ceux-ci sont ajoutés au système.

▷ **Exemple 3.2.1**

$$S = \begin{cases} C = \{dist_{pp}(A, B, k_1), dist_{pp}(A, C, k_2), dist_{pp}(B, C, k_3)\} \\ \chi = \{A, B, C : point\} \\ A = \{k1, k2, k3 : length\} \end{cases}$$

produit par une 1-transmutation le système :

$$S' = \begin{cases} C = \{dist_{pp}(A, B, k_1), dist_{pp}(A, C, k_2), angle(A, B, C, a_1)\} \\ \chi = \{A, B, C : point\} \\ A = \{k1, k2 : length, a_1 : angle\} \end{cases}$$

◁

Un changement de contraintes n'a vraiment d'intérêt que si l'ensemble des contraintes ajoutées est disjoint de l'ensemble des contraintes retirées. Cela nous conduit à la définition :

Définition 3.2.2 Une k_1-transmutation entre deux systèmes S et S' est dite minimale s'il n'existe pas de k_2-transmutation entre S et S' tel que $k_1 > k_2$.

Par définirion, la *distance syntaxique* entre deux systèmes S et S' admettant une k-transmutation minimale est égale à k.

Autrement dit, si C est l'ensemble des contraintes de S et C' l'ensemble des contraintes de S', S' se déduisant de S par une k-transmutation, la distance syntaxique entre S et S' vaut :

$$k = |C'| - |C' \cap C| = |C \smallsetminus C'|$$

3.2.2 k-quasi-décomposabilité

Nous avons vu que même en ne considérant que les méthodes combinatoires, il existe plusieurs méthodes pour décomposer un système de contraintes. On peut juger de leur efficacité selon le critère du domaine d'application (c'est entre autres, celui que nous avons utilisé au chapitre 2), mais aussi, pour un système de contraintes donné, selon leur capacité à obtenir des sous-systèmes de petite taille.

Définition 3.2.3 Soit D une méthode de décomposition et S un système de contraintes, la D-granularité de S est la taille du plus grand sous-système obtenu par l'application de la méthode D sur S.

L'efficacité d'un solveur généraliste se mesure généralement par la taille du plus grand système irréductible à résoudre. Donc, en supposant que l'opération d'assemblage qui suit la décomposition est peu coûteuse en temps, la D-granularité mesure, en quelque sorte, le gain d'efficacité par rapport à un système irréductible.

Inversement, dans la méthode de la reparamétrisation, le coût d'une transmutation pour rendre un système décomposable se mesure dans la taille du système qu'il faudra résoudre numériquement.

Définition 3.2.4 Un système irréductible pour une méthode D est k-quasi-décomposable si et seulement s'il existe une k-transmutation le transformant en un système décomposable par D.

▷ **Exemple 3.2.2**

L'octaèdre générique S présenté dans la figure 3.1 est composé de 6 points reliés par 12 contraintes de distances différentes, représentées par des arêtes. Ce système

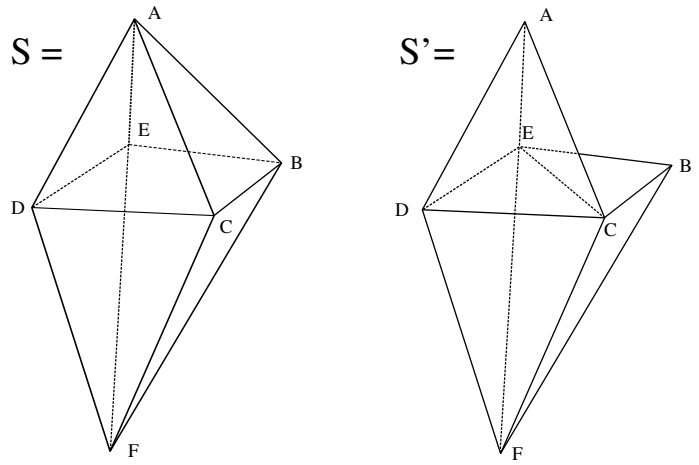


FIG. 3.1 – Octaèdre 1-quasi-décomposable : système initial (à gauche) et système 1-transmuté (à droite)

s'écrit formellement :

$$S = \langle \begin{cases} dist_{pp}(A, B, k_1) \\ dist_{pp}(B, C, k_2) \\ dist_{pp}(C, D, k_3) \\ dist_{pp}(D, E, k_4) \\ dist_{pp}(A, E, k_5) \\ dist_{pp}(A, C, k_6) \\ dist_{pp}(A, D, k_7) \\ dist_{pp}(B, E, k_8) \\ dist_{pp}(F, B, k_9) \\ dist_{pp}(F, C, k_{10}) \\ dist_{pp}(F, D, k_{11}) \\ dist_{pp}(F, E, k_{12}) \end{cases} , \{A, B, C, D, E, F\}, \{k_1, \ldots, k_{12}\} \rangle$$

Une méthode de décomposition D_{3a} qui permet de découper suivant des triplets d'articulations ne peut pas décomposer ce système. Ce système est donc irréductible pour D_{3a}.

Remarque
Une autre méthode de décomposition peut proposer de décomposer selon les faces triangulaires, mais pour assembler ces triangles, il faudrait avoir une règle prenant en compte les huit triangles d'un seul coup.

L'application d'une 1-transmutation particulière transforme S en S' par suppression de la contrainte de distance entre A et B et l'ajout de la contrainte de distance

entre C et E :

$$S' = \langle \begin{cases} dist_{pp}(C, E, k_{13}) \\ dist_{pp}(B, C, k_2) \\ dist_{pp}(C, D, k_3) \\ dist_{pp}(D, E, k_4) \\ dist_{pp}(A, E, k_5) \\ dist_{pp}(A, C, k_6) \\ dist_{pp}(A, D, k_7) \\ dist_{pp}(B, E, k_8) \\ dist_{pp}(F, B, k_9) \\ dist_{pp}(F, C, k_{10}) \\ dist_{pp}(F, D, k_{11}) \\ dist_{pp}(F, E, k_{12}) \end{cases} , \{A, B, C, D, E, F\}, \{k_2, \ldots, k_{13}\} \rangle$$

Le système S' est décomposable par D, donc S est 1-quasi-décomposable pour la méthode D_{3a}.
La même 1-transmutation peut être appliquée si la méthode considérée applique une simple propagation des degrés de liberté. ◁

3.2.3 Schéma général

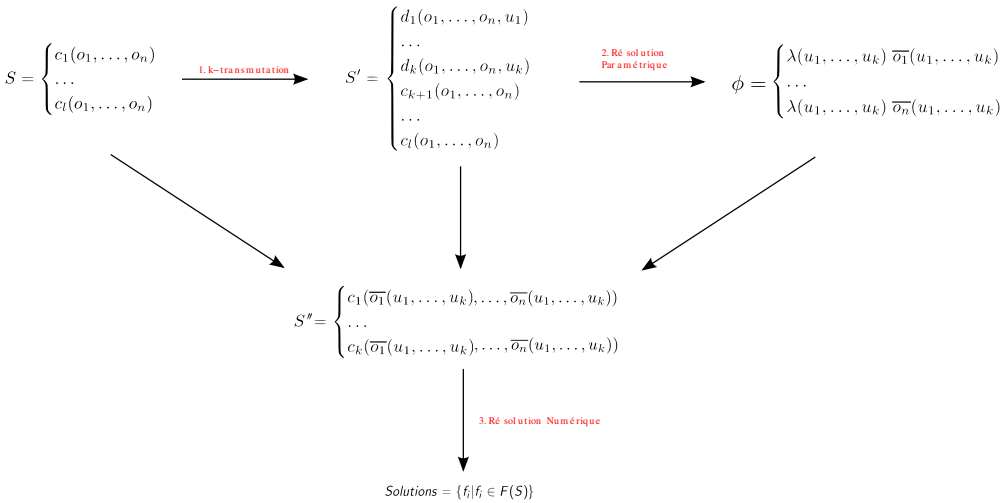


FIG. 3.2 – Schéma de la reparamétrisation

Nous appelons reparamétrisation la méthode consistant à résoudre les systèmes k-quasi décomposables suivant une méthode de décomposition.

Cette méthode comprend quatre étapes :

- trouver une k-transmutation de S produisant un système de contraintes S' soluble par une méthode paramétrique D ;
- résoudre S' à l'aide de D ;

- remplacer les inconnues par leur expression formelle en tant que solutions de S' dans les équations enlevées de S pour donner le système S'' ;
- résoudre numériquement le système S'' pour retrouver les solutions de S.

La démonstration donnée plus haut pour la correction et la complétude se généralise ici : si le système S' est bien contraint et que les contraintes paramétrées ajoutées sont non occlusives alors la méthode est correcte et complète à condition que le solveur formel et le solveur numérique soient eux aussi corrects et complets.

3.3 Mise en œuvre

Dans cette section, une mise en œuvre spécifique [FS06] de chaque étape de la reparamétrisation présentée dans la figure 3.2 est proposée.

Tout d'abord, conformément à notre approche formelle, nous avons choisi d'utiliser un algorithme à base de connaissances pour la phase de résolution paramétrique du système S'.

Pour choisir la k-transmutation à appliquer, nous proposons un algorithme incrémental qui minimise ce que nous avons appelé *la distance syntaxique locale* à chaque situation de blocage du solveur constructif considéré. La distance syntaxique locale est une distance syntaxique calculée entre un *motif* d'une règle de construction et le *voisinage* de résolution de S à un instant donné. Les définitions de ces termes sont présentées dans la sous-section suivante.

Ainsi, le solveur constructif et le choix des contraintes à remplacer sont intimement liés. Le solveur constructif utilise les connaissances tirées de l'axiomatique de l'univers géométrique considéré pour calculer cette distance locale, ce qui permet de choisir les contraintes à remplacer à la volée.

Cet algorithme local constitue une heuristique dont deux variantes peuvent être appliquées suivant le type de solveur choisi : soit par chaînage avant, soit par chaînage arrière.

Enfin, nous proposons de recourir à la méthode numérique itérative de Newton-Raphson pour rattraper les contraintes « oubliées » à la place de l'échantillonnage effectué par Gao *et al.*. Elle ne permet pas de récupérer toutes les solutions mais a l'avantage d'être automatique et rapide. Les différents problèmes dûs à cette méthode sont traités et expliqués.

Nous discutons bien sûr de la correction et de la complétude de chaque élément et de l'optimalité des heuristiques présentées.

3.3.1 Choix d'une k-transmutation et résolution paramétrique

Avant de présenter l'algorithme de minimisation de la distance syntaxique locale, nous devons poser quelques définitions.

Définitions

Dans notre cadre de logique du premier ordre (ou logique des prédicats), le système à base de connaissance que nous utilisons applique classiquement la règle d'inférence du *modus ponens* (*cf.* chapitre 2). Un moteur d'inférence teste la possibilité d'appliquer une règle constructive dans un base de faits. Si la règle est applicable, elle est appliquée et ajoutée à la base de faits.

Pour simplifier, nous considérons un solveur constructif qui produit à chaque étape de résolution un nouveau terme fonctionnel

$$o_{i+1} = f_\alpha(o_1, \ldots, o_i)$$

où l'objet o_{i+1} est construit à partir des objets $o_1, \ldots, o_i$ définis antérieurement. L'ensemble de ces termes constitue un plan de construction P :

$$P = \begin{cases} o_1 = f_1() \\ \ldots \\ o_n = f_n(o_1, \ldots, o_{n-1}) \end{cases}$$

Pour éviter d'alourdir les notations par une permutation d'indices, on pose que l'objet o_i a été construit avant l'objet o_j si $i < j$. De plus, on estime qu'un solveur constructif modifie incrémentalement un système de contraintes géométriques en transformant chaque inconnue en paramètre à chaque fois qu'une règle peut s'appliquer, et donc qu'un nouveau terme est ajouté au plan de construction. À l'étape i, un tel système noté $S_i = \langle C_i, \chi_i, A_i \rangle$ est :

$$S_i = \langle \begin{cases} c_1(o_1, \ldots, o_n) \\ \ldots \\ c_n(o_1, \ldots, o_n) \end{cases} , \{o_i, \ldots o_n\}, \{o_1, \ldots, o_{i-1}\} \rangle$$

et est associé au plan :

$$P = \begin{cases} o_1 = f_1() \\ \ldots \\ o_{i-1} = f_{i-1}(o_1, \ldots, o_{i-2}) \end{cases}$$

Définition 3.3.1 Nous appelons *voisinage* N_i de S_i le système de contraintes engendré par l'ensemble des contraintes :

$$\{c(x, a_1, \ldots, a_p) \in C_i \mid x \in \chi_i \; et \; (a_1, \ldots, a_p) \in A_i\}$$

c'est-à-dire les contraintes où une seule variable appartient encore à χ_i.

Ce voisinage définit en fait un ensemble d'inconnues à la frontière de ce qui a déjà été résolu. Il permet de réduire le nombre de candidats potentiels à la prochaine étape d'application de règles.

Le voisinage à l'étape i est un système de contraintes engendré par l contraintes :

$$N_i = \langle \begin{cases} c_1(o_1, \ldots, o_i) \\ \ldots \\ c_l(o_1, \ldots, o_i) \end{cases} , \{o_i, \ldots o_n\}, \{o_1, \ldots, o_{i-1}\} \rangle$$

▷ **Exemple 3.3.1**
Dans l'exemple 3.2.2 de l'octaèdre, à l'étape 2 ($2^{\text{ème}}$ image en partant de la gauche dans la figure 3.4 donnée à la fin de cette sous-section), le système comprend les trois points D, E, F et son voisinage se compose donc des contraintes de distance faisant intervenir D, E ou F. Le voisinage N_1 de ce système est :

$$N_1 = \langle \begin{cases} dist_{pp}(C, D, k_3) \\ dist_{pp}(A, E, k_5) \\ dist_{pp}(A, D, k_7) \\ dist_{pp}(B, E, k_8) \\ dist_{pp}(F, B, k_9) \\ dist_{pp}(F, C, k_{10}) \end{cases} , \{A, B, C\}, \{D, E, F, k_3, k_5, k_7, k_8, k_9, k_{10}\} \rangle$$

A ce stade, les valeurs de $D, E, F, k_3, k_5, k_7, k_8, k_9, k_{10}$ sont connues alors que les valeurs de A, B, C sont encore inconnues. ◁

Dans ce voisinage, la recherche d'inconnues et de contraintes pouvant être unifiées aux prémisses d'une règle va permettre ou non son application. Pour cela, le solveur s'appuie sur ce que nous appelons les motifs :

> **Définition 3.3.2** Un motif est un ensemble de m contraintes sur des variables non instanciées
>
> $$M^\alpha = (c_1(X_1, \ldots, X_{p_1}), \ldots, c_m(X_1, \ldots, X_{p_m}))$$
>
> X est une variable au sens ou elle représente l'ensemble de toutes les instances d'une sorte donnée. On dit que m est la taille du motif.

Un motif M^α_j est constitué des prémisses de la $j$ème règle valide dans un univers U donné.

▷ **Exemple 3.3.2**
Une règle permettant de construire un point en 3D à partir de trois contraintes de distances peut s'écrire :

$$si \begin{cases} dist_{pp}(X_1, X_2, X_3) \\ dist_{pp}(X_1, X_4, X_5) \\ dist_{pp}(X_1, X_6, X_7) \end{cases} \quad alors\ X_1 = point_{3p3l}(X_2, X_4, X_6, X_3, X_5, X_7)$$

avec X_1 inconnue et $X_2, \ldots, X_7$ connus.

Le motif associé est donc :

$$M^\alpha = (dist_{pp}(X_1, X_2, X_3), dist_{pp}(X_1, X_4, X_5), dist_{pp}(X_1, X_6, X_7))$$

◁

Dans le cadre de la reparamétrisation, les motifs qui ne sont pas complètement instanciés nous intéressent tout particulièrement. Nous définissons donc l'unification à partir de la notion de substitution de la manière suivante :

Définition 3.3.3 Une substitution σ pour un motif

$$M_j^\alpha = (c_1(X_1, \ldots, X_{p_1}), \ldots, c_m(X_1, \ldots, X_{p_m}))$$

est une instanciation des variables de M_j^α par des objets appartenant à un système $S = \langle C, \chi, A \rangle$ telle que $\sigma(c_1), \ldots, \sigma(c_s) \in C$.

Si $m = s$, cette substitution est un unificateur. Et si cet unificateur est minimal, alors la règle j est applicable. Sinon, $|m - s|$ est la *distance syntaxique locale* entre N_i et M_j. On écrit $dsl(N_i, M_j) = |m - s|$.

▷ **Exemple 3.3.3**
Pour le motif et le voisinage choisis dans les deux exemples précédents, une substitution est :

$$\begin{cases} X_1 = C \\ X_2 = D \\ X_3 = k_3 \\ X_6 = F \\ X_7 = k_{10} \end{cases}$$

Dans ce cas, $s = 2$ et $dsl(N_1, M_j) = 1$. ◁

Algorithme de choix à la volée

L'algorithme du choix de la k-transmutation à appliquer que nous avons développé repose sur la définition de la k-quasi-décomposabilité. En effet, on suppose l'existence d'une k-transmutation entre deux systèmes S et S' ce qui peut se traduire par une distance syntaxique entre ces deux systèmes : $k = |C'| - |C' \cap C|$.

L'algorithme de choix à la volée se contente d'essayer de minimiser la distance syntaxique globale entre S et S' en minimisant la distance syntaxique locale à chaque fois que le solveur associé ne peut plus continuer à résoudre. Ainsi, si aucune règle ne peut s'appliquer, le reparamétreur va choisir un motif M_j minimisant la distance syntaxique locale dans le voisinage N_i. A partir de ce motif, le reparamétreur choisit suivant une heuristique les contraintes à ajouter et/ou à supprimer.

L'algorithme peut être décrit par :

```
Tant que le solveur constructif échoue :
{
    calculer N_i
    calculer dsl(N_i,M_j) pour chaque motif M_j
    choisir le motif M minimisant dsl(N_i,M_j)
    appliquer l'heuristique sur M
}
```

Nous avons distingué deux heuristiques suivant le cas où :
- $s < m$, *i.e.* il manque $m - s$ de contraintes pour pouvoir appliquer le motif M_j ;
- $s > m$, *i.e.* il y a un excès de $s - m$ contraintes.

Le choix de Gao *et al.* d'effectuer un algorithme de rétro-propagation des degrés de liberté conduit à se retrouver dans une phase où il y a presque toujours un excès de contraintes. Lorsqu'un excès est rencontré, des contraintes sont supprimées. En revanche, aucune mention n'est faite de la manière d'ajouter les contraintes qui remplacent celles supprimées.

En effet, suivant la manière d'inférer pour résoudre le problème, on tombe préférentiellement sur deux heuristiques différentes :
- le chaînage arrière où l'on rencontre des excès de contraintes,
- le chaînage avant où l'on rencontre un manque de contraintes.

Heuristiques

Nous proposons pour chacun de ces deux cas une heuristique pour ajouter ou enlever des contraintes.

Chaînage arrière L'heuristique par chaînage arrière conduit à un excès de contraintes. Le choix des contraintes à supprimer revient à sélectionner les contraintes permettant l'unification d'une règle et à supprimer les contraintes restantes.

Dans le cas, où il n'existe pas de règle applicable, il faut ajouter une règle pour former un repère.

```
S'il existe un motif :
{
    Appliquer le motif et ajouter le terme
        dans le plan de construction
    Supprimer les contraintes en excès
    Déplacer l'objet inconnu vers les paramètres
}
sinon
{
    Ajouter des contraintes pour former un repère
}
```

La formation d'un repère est aisée dans le cas où il y a peu d'objets restant à contraindre. Pourtant, ce n'est pas le cas la plupart du temps. Il faut donc fournir des règles de formation de repères assez complexes (*cf.* annexes).

▷ **Exemple 3.3.4**
Considérons l'exemple le plus simple pour illustrer ces heuristiques : l'octaèdre générique. L'application de l'heuristique suivante peut se faire en considérant l'application de la règle de construction considérée dans les exemples précédents et d'une règle de formation de repère :

Si 3 points sont reliés par 3 contraintes de distance, alors on peut construire la figure *modulo* les déplacements :

$$si \begin{cases} dist_{pp}(X_1, X_2, X_3) \\ dist_{pp}(X_2, X_4, X_5) \\ dist_{pp}(X_1, X_4, X_6) \end{cases} \quad alors \begin{cases} X_1 = mkPoint(0,0) \\ X_2 = mkPoint(X_3, 0) \\ X_4 = inter_{2spl}(X_1, X_5, X_2, X_6, mkPlan(0,0,1,0)) \end{cases}$$

avec X_3, X_5 et X_6 connus et X_1, X_2 et X_4 inconnus.

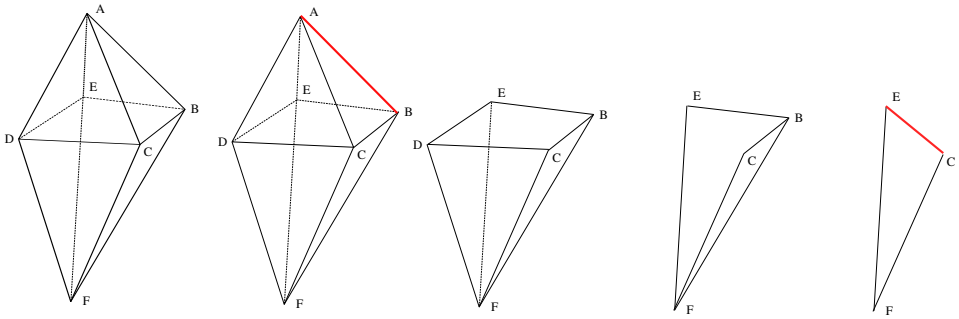


FIG. 3.3 – Reparamétrisation de l'octaèdre par chaînage arrière

La contrainte $dist_{pp}(A, B, k_1)$ est supprimée lorsqu'on considère la règle de construction avec A comme inconnue X_1. Le point A est alors construit à partir des points E, C et D.

La résolution se poursuit normalement avec la construction du point D à partir de E, C et F, et du point B à partir des points E, C et F.

À la fin de la résolution, on tombe sur deux contraintes de distances reliant 3 points et il suffit d'ajouter la contrainte $dist_{pp}(C, E, k_{13})$ pour obtenir un repère 3D pour les déplacements au sens décrit dans l'annexe B. ◁

Chaînage avant L'heuristique par chaînage avant conduit à se retrouver avec un manque de contraintes. Elle s'applique bien sûr après l'ajout d'un repère si le système est bien contraint *modulo* les déplacements. Le choix des contraintes à ajouter se fait en complétant le motif sélectionné. Il comporte des variables non instanciées et c'est donc par la construction d'un unificateur que la contrainte à ajouter est formée.

Les instanciations doivent se faire dans l'ensemble des paramètres pour des contraintes n'existant pas encore. Nous avons choisi d'ajouter des contraintes de distance car elles sont de degré 2 et ne sont donc pas occlusives pour les contraintes retenues dans notre univers euclidien 3D. En effet, il suffit de considérer les contraintes de distances au carré :

$$(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 = k^2$$

pour que l'on ait une solution pour toute valuation de k.

Lorsqu'il ne reste plus d'inconnues, les contraintes n'ayant pas été utilisées sont celles à rattraper.

`Compléter l'instanciation de` M_m
`Appliquer le motif et ajouter le terme dans le plan de construction`
`Déplacer l'objet inconnu vers les paramètres`

Contrairement à l'heuristique par chaînage arrière, on voit qu'il n'est pas nécessaire de considérer des règles de formation de repère trop complexes. De plus, on ne choisit pas de contraintes à enlever, le solveur travaille sur un énoncé sur-contraint.

▷ **Exemple 3.3.5**
Avec les mêmes règles que dans l'exemple précédent, nous appliquons dans cet exemple l'heuristique par chaînage avant.

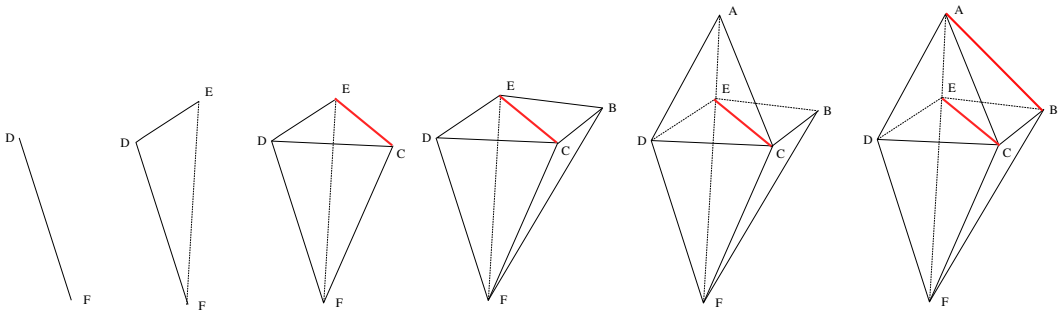


FIG. 3.4 – Reparamétrisation de l'octaèdre par chainage avant

Les deux premières images de la figure 3.4 montrent la construction d'un repère 3D au sens donné dans l'annexe B. La contrainte $dist_{pp}(C, E, k_{13})$ est ajoutée à l'étape 3 afin de pouvoir construire le point C à partir des points D, E et F.

Ensuite, le point B est classiquement construit à partir de E, C, et F et le point A à partir des points C, D et E.

À la fin de la résolution, il reste une contrainte non satisfaite : la contrainte $dist_{pp}(A, B, k_1)$. ◁

Discussions

Dans les deux heuristiques précédentes, le système S' obtenu est un système bien contraint car il est résolu de manière formelle au fur et à mesure de la k-transmutation.

Bien sûr, les deux heuristiques ne donnent pas forcément la même k-transmutation ni même une transmutation minimale, même si c'est le cas dans l'exemple de l'octaèdre.

Les avantages de ces heuristiques sont :
- la constructibilité dans U est assurée ;
- les k contraintes sont trouvées à la volée.

Il existe pourtant des inconvénients parmi lesquels on peut citer :
- comme toutes les heuristiques locales, on n'est pas assuré de trouver le coût minimal ;
- lorsque k devient grand, la taille du voisinage peut augmenter rapidement et ralentir la recherche de la $i^{\text{ème}}$ contrainte à choisir surtout lorsqu'on considère un grand nombre de règles ;

– utiliser une contrainte non occlusive comme une contrainte de distance peut conduire à produire des branches d'évaluation en plus pour des solutions n'existant pas dans S.

3.3.2 Rattrapage Numérique

Les méthodes numériques tentent d'approcher par approximations successives les solutions d'un système de contraintes géométriques.

L'application d'une technique de force brute, comme l'échantillonnage utilisé par Gao *et al.*, permet de trouver toutes les solutions mais devient prohibitif pour $k > 1$.

Nous proposons d'utiliser l'itération de Newton-Raphson (NR) présentée dans le chapitre 2, qui a une convergence quadratique quand elle converge.

Dans le cadre de la reparamétrisation, le système à considérer S'' dépend d'une fonction paramétrée $f(u_1, \ldots u_k)$. Dans notre mise en œuvre, cette fonction paramétrée est un plan de construction qu'il faut utiliser dans l'itération de NR. Ceci pose quelques problèmes que nous proposons de résoudre.

De plus, l'évaluation des multi-fonctions du plan de construction permet d'obtenir plusieurs solutions. Le calcul de toutes les solutions est relativement long et nous proposons d'avoir recours à la méthode du gel de branche [EV01] pour choisir une solution « proche » de l'esquisse.

De plus, nous proposons de corriger les instabilités numériques dues à la reparamétrisation au niveau de :
– l'initialisation ;
– la discontinuité de l'espace des solutions.

Gestion du plan de construction

La méthode de Newton-Raphson classique accepte en entrée :
– un système de contraintes sans paramètres ;
– et une valuation initiale des inconnues.

Dans notre cadre, nous devons prendre en compte les paramètres du plan de construction dans la méthode et donc plusieurs solutions, et calculer une valuation initiale pour S'' que nous avons appelé *l'esquisse virtuelle*.

Paramètres Les inconnues $u_1, \ldots, u_k$ sont les paramètres du plan de construction. À chaque itération de Newton-Raphson, ces paramètres sont modifiés et une nouvelle évaluation du plan de construction doit être effectuée.

Le plan de construction passé en paramètre à NR contient des multi-fonctions. On peut le représenter par un arbre où les nœuds sont des multi-fonctions. Chaque branche de cet arbre est une possible solution au système S'.

On peut donc lancer la correction numérique pour chaque branche et ainsi obtenir des solutions différentes pour S.

Ce nombre de branches est majoré par le nombre de Bézout [Wik] qui donne le nombre maximal de solutions d'un système d'équations. Ce nombre constitue une borne maximum grossière qui correspond à la multiplication des degrés des équations mis en jeu dans le système de contraintes. Comme nous le montrons sur les exemples de la section suivante, le nombre de branches peut être grand et le temps de résolution peut devenir prohibitif si l'on cherche toutes les solutions.

Nous proposons de ne calculer qu'une seule solution la plus proche possible de l'esquisse.

La technique du gel de branche [EVSD02] est utilisée pour effectuer le choix de la branche à évaluer. Le gel de branche consiste à utiliser le système de contraintes de départ avec toutes les valuations des inconnues et des paramètres lues sur l'esquisse.

Mais, dans notre cadre, ce gel ne peut s'effectuer directement sur l'esquisse car elle ne constitue pas une solution initiale du système S'' mais du système S.

Esquisse virtuelle Dans la figure 3.5, S_0 est l'esquisse associée à S, S'_0 est l'esquisse reparamétrée associée à S', et S''_0 est l'esquisse virtuelle associée à S''.

$$\begin{array}{ccccc} S & \rightarrow & S' & \rightarrow & S'' \\ \downarrow & & \downarrow & & \downarrow \\ S_0 & \rightarrow & S'_0 & \rightarrow & S''_0 \end{array}$$

FIG. 3.5 – Esquisse, esquisse reparamétrée et esquisse virtuelle

S_0 est récupérée à partir des valuations numériques de l'interface graphique. S'_0 est calculée à partir de S_0 en mesurant les valeurs des contraintes ajoutées et en supprimant celles des contraintes supprimées. Enfin, S''_0 contient toutes les valuations numériques issues de S_0 et S'_0.

L'itération de Newton-Raphson adaptée à la reparamétrisation, notée NR_k, accepte donc en entrée :

– le système $S'' = \langle C_{S''}, U_{S''}, A_{S''} \rangle =$

$$\langle \begin{cases} c_1(\overline{o_1}(u_1, \ldots u_k), \ldots, \overline{o_n}(u_1, \ldots u_k)) \\ \ldots \\ c_k(\overline{o_1}(u_1, \ldots u_k), \ldots, \overline{o_n}(u_1, \ldots u_k)) \end{cases} , \{u_1, \ldots u_k\}, A_S - A'_S \rangle$$

avec $\overline{o_1}, \ldots, \overline{o_n}$ les objets paramétrés résultats de $f(u_1, \ldots u_k)$;

– le plan de construction $f(u_1, \ldots, u_k)$ permettant de produire

$$\overline{o_1}(u_1, \ldots u_k), \ldots, \overline{o_n}(u_1, \ldots u_k)$$

– l'esquisse virtuelle S''_0.

En considérant un nombre d'itérations maximal I_0, `gel` la fonction de gel de branche, `extract` une fonction qui extrait les valeurs des inconnues dans S''_0, et `jacobian` qui calcule le jacobien du système de contraintes, l'algorithme NR_k s'écrit :

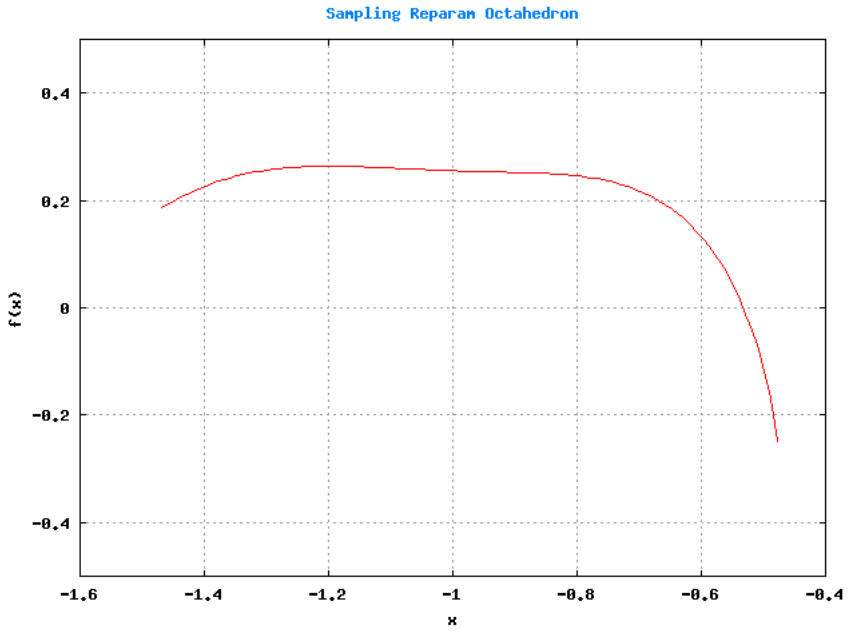


FIG. 3.7 – Échantillonnage d'une branche de l'octaèdre, avec x la valeur de la distance CE et $f(x)$ la variation de la contrainte de distance AB

$f_b(u_1, \ldots, u_k) =$ `gel(`S_0''`,`$f(u_1, \ldots, u_k)$`)`
$x_1, \ldots, x_k$ `= extract(`$u_1, \ldots, u_k$`,` S_0''`)`
$i = 1$
`while (`$\|f_b(x_1, \ldots, x_k)\| > \varepsilon$ `&&` $i \leq I_0$`)`
`{`
    `J=jacobian(`$C_{S''}$`)`
    $(x_1, \ldots, x_k) = (x_1, \ldots, x_k) - J^{-1} \times f_b(x_1, \ldots, x_k)$
    $i = i + 1$
`}`

FIG. 3.6 – Algorithme de correction numérique par l'itération de Newton-Raphson

Si l'on désire calculer l'ensemble des solutions, il suffit de parcourir l'arbre des multi-fonctions du plan de construction à la place du gel de branche.

Initialisation

La figure 3.7 est un échantillonnage d'une branche de la solution formelle trouvée pour l'octaèdre générique avec l'heuristique par chaînage avant. Le domaine de validité de u pour que $f(u)$ ait des solutions réelles, est, dans la pratique, très restreint. Lorsqu'il faut considérer le plan de construction $f(u_1, \ldots, u_k)$, les domaines des u_i sont d'autant plus difficiles à choisir.

L'initialisation du solveur numérique est donc une phase critique.

L'initialisation que nous avons mis en place se base sur l'esquisse S_0 et l'esquisse reparamétrée S'_0 pour construire l'esquisse virtuelle S''_0. Ainsi, pour certaines valeurs des paramètres de S, l'évaluation totale du plan de construction avec les valeurs de l'esquisse virtuelle échoue, *i.e.* au moins un objet $o_i \in (o_1, \ldots, o_n)$ n'est pas constructible avec ces valeurs de $u_i \in (u_1, \ldots, u_k)$.

Il faut donc modifier l'esquisse virtuelle S''_0 pour qu'elle soit valide pour la fonction de construction de S''. Pour cela, nous proposons de calculer un facteur d'échelle pour que les valeurs des paramètres ajoutés soit ramenées dans un domaine où il existe des solutions pour le plan de construction.

Ce facteur d'échelle se traduit par un cœfficient qui dépend du plan de construction pour chaque paramètre.

Dans un plan de construction, on appelle *paramètres ancêtres* d'un objet o, l'ensemble des paramètres qui sont intervenus dans la construction de o, ou d'un objet dont dépend o.

▷ **Exemple 3.3.6**
Dans le plan de construction suivant :

$$\begin{cases} o_2 = f_1(o_1, \mathbf{k_1}) \\ o_3 = f_2(o_1, o_2, k_2, k_3) \\ o_4 = f_3(o_2, \mathbf{k_4}) \\ o_5 = f_4(o_4, \mathbf{k_5}) \end{cases}$$

Les paramètres ancêtres de o_5 sont les paramètres k_5, k_4 et k_1. ◁

Ainsi, nous calculons une moyenne des quotients entre valeurs sur l'esquisse virtuelle K_s et valeurs K_v des paramètres dans S'' pour chacun des a paramètres ancêtres non ajoutés lors du remplacement de contraintes. Cela donne un cœfficient du facteur d'échelle pour un paramètre donné :

$$C_{scale}(u_i) = \frac{\sum \frac{K_v}{K_s}}{a}$$

Ce cœfficient est bien sûr différent pour chaque paramètre. Dans les expériences que nous avons menées, ce facteur d'échelle permet d'obtenir une esquisse virtuelle proche d'une évaluation numérique valide pour le plan de construction.

Stabilité : Ajustement du pas d'itération

Si l'initialisation a permis de se retrouver proche d'une solution, il se peut malgré tout que l'itération de Newton-Raphson ne la trouve pas.

D'une part, lorsqu'il existe plusieurs solutions dans une branche (*cf.* figure 3.8), l'utilisation de Newton-Raphson ne permet d'en trouver qu'une et ce n'est pas forcément celle dont l'esquisse virtuelle est la plus proche.

D'autre part, le fait que le domaine de définition des inconnues soit restreint peut amener qu'un x_i calculé à l'itération précédente sorte du domaine de validité.

Par exemple, sur la figure 3.8, si lors de l'initialisation, la valeur du paramètre vaut plus de 0.6, la correction va trouver la solution en 2.6, alors que si sa valeur est en deçà, la pente de la tangente va conduire à sortir rapidement du domaine et la solution en 0.2 ne sera pas trouvée.

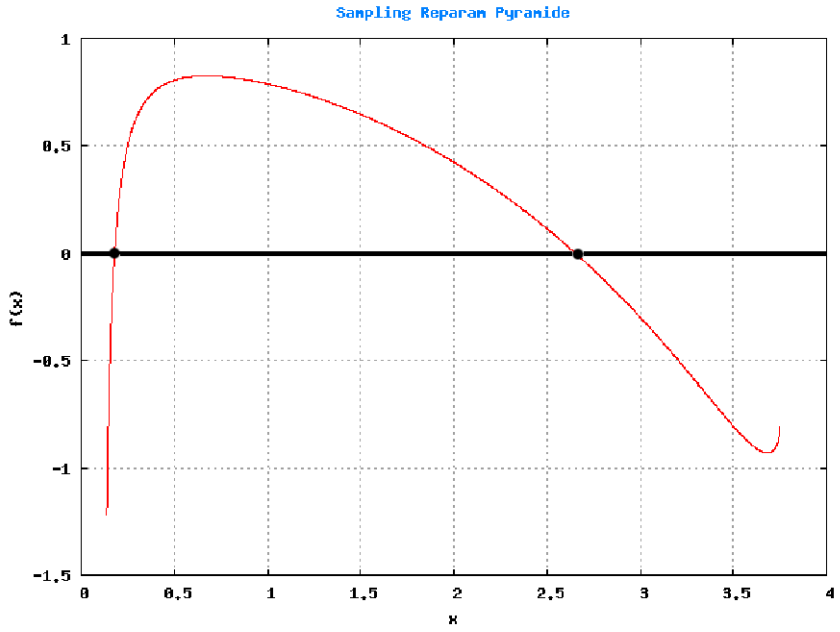


FIG. 3.8 – Échantillonnage d'une branche de la solution à la pyramide

Une stratégie adaptative de détermination du pas de l'itération a donc été mise en œuvre. Nous avons procédé en diminuant arbitrairement de 50% le pas de l'itération lorsque Newton échoue et en revenant à la dernière bonne valeur. Si au bout de deux diminutions ($\frac{1}{4}$ de la correction normalement appliquée), la solution n'a pas été trouvée, nous estimons qu'il n'en existe pas.

Saut vers une branche similaire

Comme le montre l'échantillonnage d'une branche dans le problème de l'octaèdre (*cf.* figure 3.9), chaque branche ne contient pas forcément une solution. En effet, suivant les valeurs des paramètres, on peut tomber dans des cas dégénérés (en général sur-contraints), où l'ajout d'une contrainte de degré supérieur à celle enlevée, a introduit une solution de S' ne vérifiant pas S.

Du coup, le gel de branche peut ne pas conduire forcément à une solution numérique.

On remarque que sur la plupart des échantillonnages réalisés (*cf.* figures 3.7,3.8), des tangentes horizontales ou verticales apparaissent souvent en extrémité des domaines de validité. Ces tangentes indiquent le passage par des points singuliers dont l'évaluation suit en général une branche proche.

De plus, nous avons expérimentalement remarqué que souvent, deux branches proches dans l'arbre de solutions semblaient former des morceaux de la même courbe

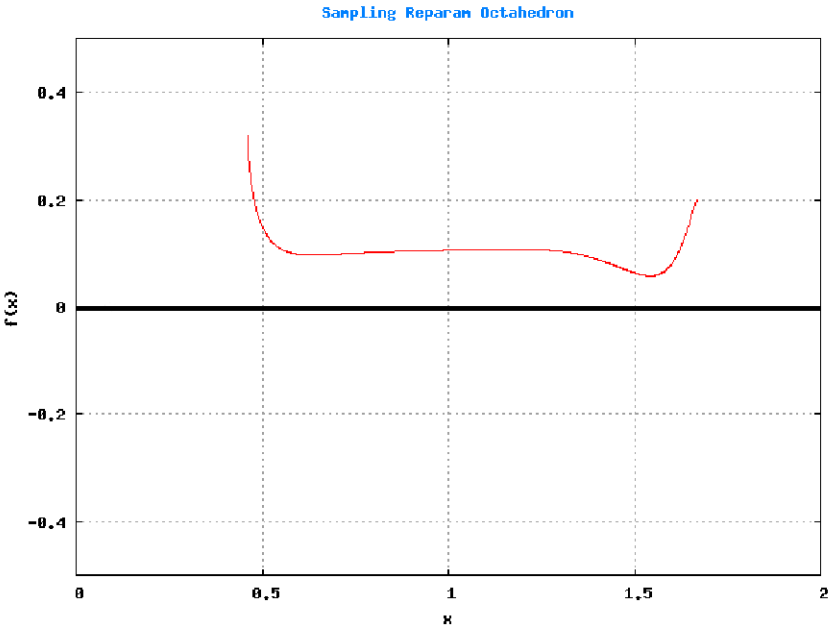


FIG. 3.9 – Echantillonnage d'une branche sans solutions

algébrique (*cf.* figure 3.10).

Dans le cadre de la CAO, où une seule solution est désirée, considérer la branche la plus proche de l'esquisse peut ne pas conduire à une solution. S'il n'existe pas de solution sur cette branche particulière, il faut en explorer une nouvelle et nous proposons de sauter de la branche considérée vers la branche la plus proche au sens du vecteur de multi-fonction, pour calculer une solution relativement proche de l'esquisse.

Ce saut de branche en branche est une forme de « back-jumping » classique issu des CSP [Dec90]. Nous l'appliquons sur l'arbre d'indice des multi-fonctions à utiliser, et la condition de saut est l'impossibilité d'évaluer complètement le plan de construction à partir d'une branche donnée.

▷ **Exemple 3.3.7**
On considère dans cet exemple des multifonctions de degré 2 avec l'arbre des solutions de la figure 3.11.

Soit le vecteur 0011 représentant une branche d'évaluation avec les choix effectués à chaque multifonction : 0 pour la première solution et 1 pour la seconde.

Si la quatrième multi-fonction échoue, la branche 0010 ne sera jamais évaluée et le saut par branche similaire va permettre d'arriver à la branche 0001. ◁

Si les solutions numériques produites par l'évaluation du plan de construction sont bien des solutions alors notre méthode est correcte. Bien sûr, elle n'est pas complète car l'utilisation de Newton-Raphson nous fait perdre des solutions.

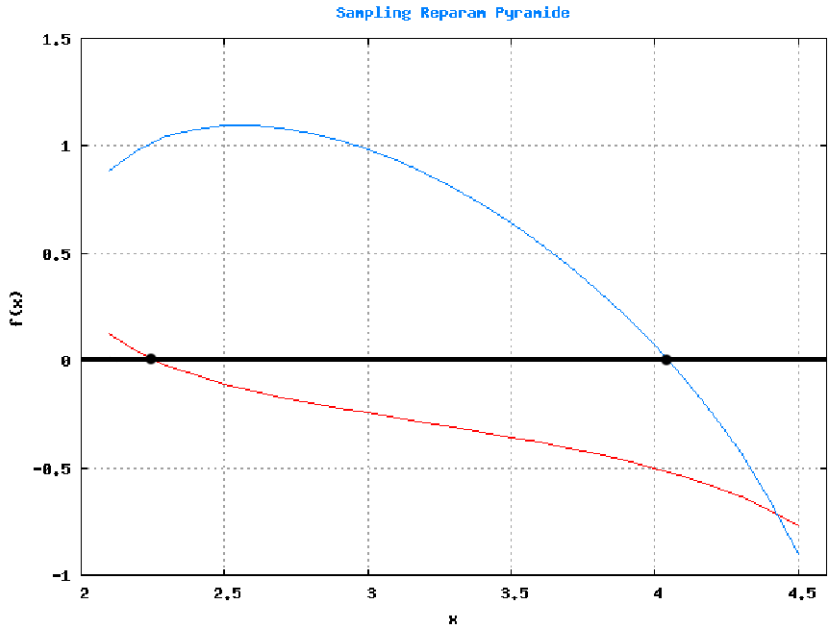


FIG. 3.10 – Échantillonnage de deux branches proches pour la pyramide

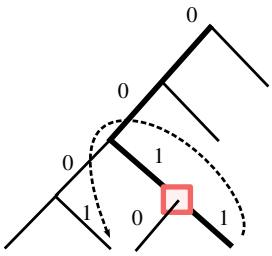


FIG. 3.11 – Saut vers la branche similaire la plus proche.

3.4 Exemples

Nous avons réalisé un prototype nous permettant de valider notre mise en œuvre de la reparamétrisation. Les tests présentés ici portent tous sur l'heuristique de chaînage avant avec gel de branche lorsque cela n'est pas précisé. Les exemples de tests sont exprimés dans l'univers géométrique des polyèdres.

Cet univers permet notamment de représenter la classe des problèmes des molécules (points et distances) très étudiée en chimie et biologie moléculaire (*cf.* [BP07]), et la classe des problèmes de polyèdres qui présente en plus des contraintes de coplanarité.

L'univers géométrique des polyèdres est décrit dans l'annexe A.2. C'est une restriction de l'univers euclidien 3D aux objets points et plans. Cette réduction d'univers nous permet de définir simplement les polyèdres par des points et des contraintes de distance et d'incidence point/plan.

Le lecteur pourra trouvera dans l'annexe C les problèmes décrits en *Gcml* et les figures obtenues dans *Axel*, un modeleur algébrique existant, développé par Julien

Wintz à l'INRIA de Nice Sophia-Antipolis, auquel nous avons ajouté une interface de modélisation par contraintes. Ce modeleur propose entre autres la visualisation d'objets composés de points et de plans mais surtout de courbes et de surfaces décrites par des équations algébriques. Nous avons ajouté la possibilité de visualiser et de poser des contraintes de distance entre points et des coplanarités entre points.

3.4.1 Polyèdres élémentaires k-quasi-décomposables

Nous commençons par considérer des exemples de polyèdres élémentaires nécessitant une transmutation pour être résolus :

- la pyramide classique,
- le camembert générique,
- la pyramide à base pentagonale,
- la pyramide à base hexagonale,
- l'antiprisme d'ordre 4,
- l'étoile de mer.

La pyramide classique est composée de 5 points dont 4 forment une base quadrilatérale.

La pyramide à base pentagonale (figure 3.12) est composée de 6 points dont 5 forment la base et la pyramide hexagonale de 7 points dont 6 forment la base.

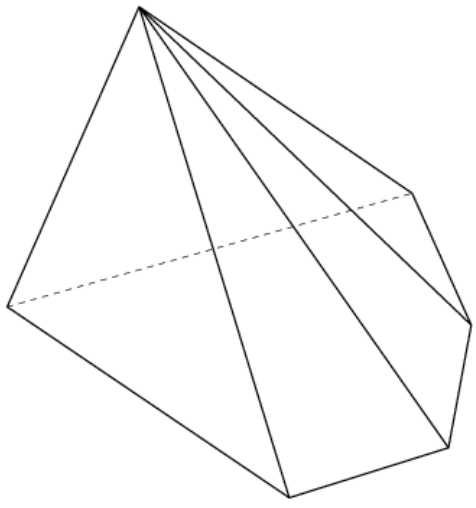


FIG. 3.12 – Pyramide à base pentagonale

Le camembert générique est ainsi nommé car c'est un pentaèdre avec 3 faces quadrilatérales qui ressemble à une part de camembert (*cf.* figure 3.13). Si les trois quadrilatères sont des parallèlogrammes, alors le système de contraintes est dégénéré et non rigide.

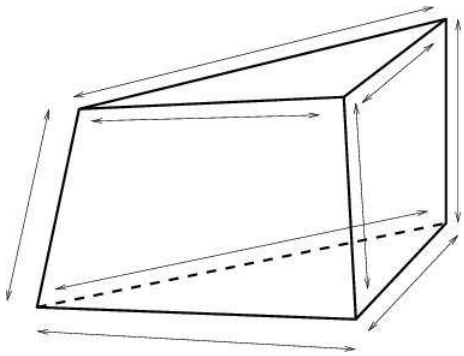


FIG. 3.13 – Le camembert

L'étoile de mer (figure 3.14) est un polyèdre à 7 sommets et 10 faces triangulaires. Deux sommets sont de degré 5 et les autres sont de degré 4. Ces derniers ne sont pas forcément coplanaires.

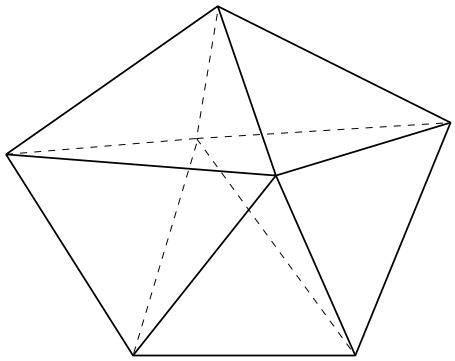


FIG. 3.14 – L'étoile de mer

L'antiprisme d'ordre 4 (figure 3.15) est composé de deux faces quadrilatérales et de 8 faces triangulaires de telle sorte que les sommets soient tous de valence 4.

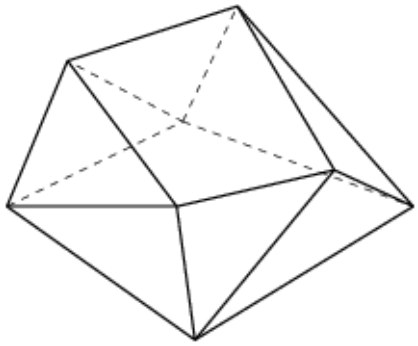


FIG. 3.15 – L'antiprisme d'ordre 4

Pour ces polyèdres élémentaires, nous avons les nombres de Bézout suivant :

Polyèdres élémentaires	**Nombre de Bézout**
Pyramide	2^8 soit 256
Camembert	2^9 soit 512
Pyramide à base pentagonale	2^{10} soit 1024
Pyramide à base hexagonale	2^{12} soit 4096
Étoile de mer	2^{14} soit 16 384
Antiprisme d'ordre 4	2^{16} soit 65 536

Avec la technique du gel de branche, nous obtenons une solution pour chacun de ces polyèdres minimaux non triviaux :

Polyèdres	Objets	Contraintes	k	Tps (ms)	Nb. sol.	Tps (min)
Pyramide	5	9	*1*	41	16	0.0.134
Camembert	6	12	*2*	46	16	0.0.142
Pyramide/pentagonale	6	12	*2*	55	16	2.10.941
Pyramide/hexagonale	7	15	*3*	108	48	0.3.099
Étoile de mer	7	14	*1*	103	16	0.0.409
Antiprisme d'ordre 4	9	20	2	216	71	5.44.830

En appliquant une recherche exhaustive, on remarque que les k-transmutations obtenues sont des transmutations minimales. Ainsi, la pyramide et l'étoile de mer sont des problèmes quasi-décomposables pour notre méthode. Les problèmes de la pyramide à base pentagonale, de la part de camembert et de l'antiprisme d'ordre 4 sont 2-quasi-décomposables et seule la pyramide à base hexagonale est 3-quasi-décomposable.

Les temps de résolution pour obtenir une unique solution sont indiqués dans la $5^{\text{ème}}$ colonne. Ils sont tous inférieurs à la seconde ce qui permet d'espérer une résolution quasi-interactive dans un modeleur par contraintes.

Le nombre de solutions trouvées pour chacun des éléments est très éloigné des nombres de Bézout qui indiquent le nombre de branches à traiter pour l'évaluation.

Les polyèdres présentés sont très symétriques et l'on pourrait s'attendre à ce qu'il y ait un nombre pair de solutions. Ce n'est pas le cas de l'antiprisme où l'évaluation d'une branche de l'arbre des solutions n'a pas permis à la méthode de Newton-Raphson de trouver une solution numérique.

On peut noter que les temps pour plusieurs solutions sont relativement corrects sauf pour la pyramide à base pentagonale et l'antiprisme. Dans ces cas là, en effet, de nombreuses branches possèdent des points de singularité qu'il faut explorer en diminuant le pas d'itération.

Les décompositions obtenues s'apparentent à un découpage en tétraèdres. Avec des structures adéquates et en découpant combinatoirement un polyèdre en tétraèdres, on peut penser obtenir une méthode efficace pour ces polyèdres. Mais, la « tétraèdrisation » n'est intéressante que pour des problèmes de petite taille.

La figure 3.16 représente une solution au problème de la pyramide à base pentagonale affichée dans le prototype *Axel*.

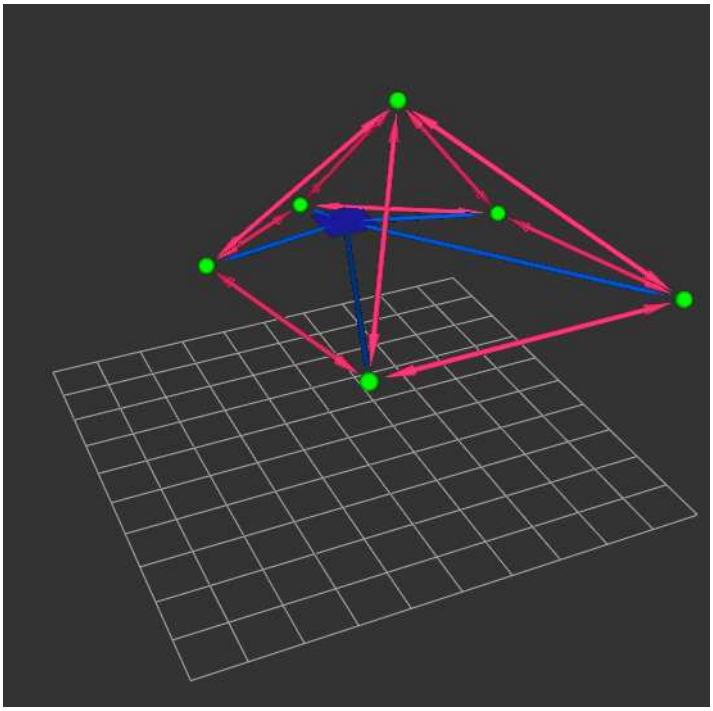


FIG. 3.16 – Pyramide à base pentagonale

3.4.2 Polyèdres pseudo-platoniciens

Les polyèdres pseudo-platoniciens sont les 5 polyèdres réguliers classiques auxquels on enlève la propriété de régularité :

- le tétraèdre,
- l'octaèdre,
- l'hexaèdre,
- l'icosaèdre,
- le dodécaèdre.

Pour les polyèdres pseudo-platoniciens, nous avons les nombres de Bézout suivant :

Polyèdre Pseudo-Platonicien	**Nombre de Bézout**
Tétraèdre	2^6 soit 64
Octaèdre	2^{12} soit 4096
Hexaèdre	2^{12} soit 4096
Icosaèdre	2^{30} soit 1 073 741 824
Dodécaèdre	2^{30} soit 1 073 741 824

Le tétraèdre ne nécessite aucune transmutation. C'est une exemple 3D minimal et trivial, ce qui n'est pas le cas des 4 autres qui nécessitent l'application de la reparamétrisation.

Polyèdres	Objets	Contraintes	k	Temps (ms)
Octaèdre	6	12	*1*	*75*
Hexaèdre	8	12	*3*	*div (1000)*
Icosaèdre	12	30	*3*	*1136*
Dodécaèdre	32	90	*7*	*div (13000)*

L'octaèdre est un cas très simple ne nécessitant qu'une 1-transmutation. L'octaèdre est intuitivement la réunion de deux pyramides à ceci près, que la base n'est pas forcément plane.

L'hexaèdre et l'icosaèdre sont des problèmes 3-quasi-décomposables, alors que le dodécaèdre est un problème 7-quasi-décomposable.

Les temps de résolution de l'icosaèdre est d'environ une seconde, ce qui reste acceptable. Par contre, pour l'hexaèdre et le dodécaèdre, la méthode de Newton-Raphson n'arrive pas à rattraper les contraintes et nous indique au bout d'un temps raisonnable qu'aucune solution n'a été trouvée. La raison pour laquelle l'algorithme de Newton-Raphson diverge (*div*) pour chaque branche est que l'esquisse utilisée ne respecte pas les contraintes de coplanarité. En effet, les valeurs calculées sur l'esquisse pour les paramètres ajoutés sortent du domaine de validité lorsque les contraintes d'incidences ne sont pas respectées. L'algorithme présenté est donc intimement dépendant de l'interface.

3.4.3 D'autres dodécaèdres

Nous présentons dans cette section deux exemples de polyèdres à douze faces moins connus que le dodécaèdre commun. Ce sont deux polyèdres de Catalan, duaux des solides archimédiens dont les faces sont toutes identiques et les sommets tous réguliers (de même degré).

Triaki-tétraèdre

Le triaki-tétraèdre est composé de 8 points et de 18 contraintes de distances (voir annexe C.3.1) et son nombre de Bézout est 2^{18} =262 144. La figure 3.17 montre le triaki sous forme d'un problème de molécule et la figure 3.18 sous forme d'une projection planaire éclatée au niveau d'un sommet. Sa représentation en 3D (*cf.* figure C.7), permet de se rendre compte de la relative simplicité de ce dodécaèdre.

En effet, la reparamétrisation du triaki-tétraèdre nécessite au minimum une 1-transmutation (p_1-p_5, p_1-p_6, ou p_1-p_7 dans la figure figure 3.18). Et une solution numérique est trouvée en 168 ms lors du gel de la branche topologiquement la plus proche de l'esquisse.

En cas de recherche de l'ensemble des solutions disponibles, le solveur propose 128 solutions en 362 ms. Deux solutions sont présentées dans les figures C.8 et C.9.

Pseudo-Dodécaèdre Rhombique

Le pseudo-dodécaèdre rhombique est un plus gros dodécaèdre que l'exemple précédent (voir annexe C.3.2) puisqu'il est formé par 14 points, 12 plans, 24 contraintes

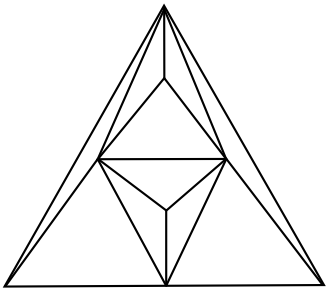


FIG. 3.17 – Représentation sous forme de molécule du triaki-tetraèdre

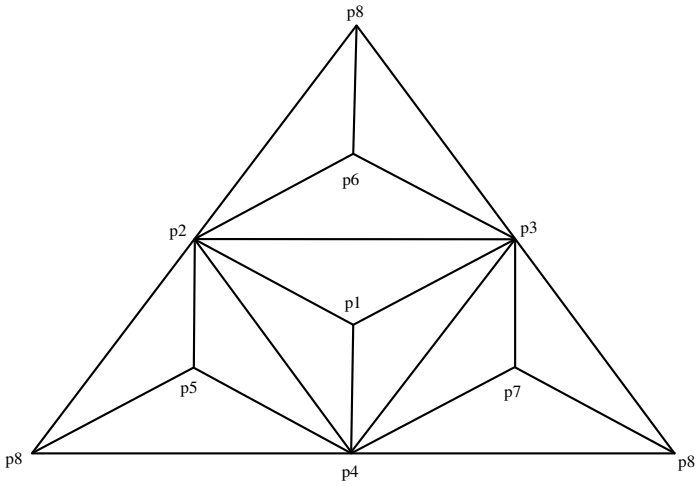


FIG. 3.18 – Le projection planaire éclatée du triaki-tétraèdre

de distances et 48 incidences points/plans. Son nombre de Bézout est de 2^{24} =16 777 216.

Il est composé de 12 quadrilatères qui sont des losanges lorsque les contraintes de distances sont positionnées pour former un vrai dodécaèdre rhombique C.10.

La reparamétrisation propose une 5-transmutation mais la correction numérique à partir d'une esquisse où les coplanarités ne sont pas respectées échoue en 8s.

Conclusion du chapitre

La résolution formelle de systèmes de contraintes géométriques 3D est un problème difficile. D'une part, les méthodes à base de connaissances géométriques sont difficilement applicables du fait de l'explosion du nombre de cas particuliers à considérer en 3D. D'autre part, la décomposition de systèmes de contraintes, qui est la stratégie la plus efficace en 2D, se heurte à des problèmes irréductibles qui sont potentiellement grands. Nous avons proposé dans ce chapitre une méthode hybride, nommée reparamétrisation, qui permet premièrement d'appliquer les méthodes formelles à base de connaissances géométriques à la 3D, et deuxièmement de décomposer certains gros systèmes irréductibles 3D que nous avons nommés systèmes quasi-décomposables pour une méthode de décomposition donnée. Notre méthode permet de résoudre ces systèmes et de proposer un sous-ensemble des solutions possibles.

Même si les heuristiques que nous proposons permettent d'appliquer cette méthode à de gros problèmes de contraintes, il reste que la correction numérique pour les systèmes dont les esquisses virtuelles ne respectent pas les contraintes d'incidences échouent. De plus, pour cette correction numérique, nous avons utilisé l'algorithme de Newton-Raphson qui outre les problèmes d'initialisation et de stabilité que nous avons corrigés, nous fait perdre la complétude. Parmi les améliorations que nous comptons mettre en œuvre, l'utilisation de méthodes de correction complètes comme l'homotopie ou l'arithmétique des intervalles (nous pensons aux méthodes de suivi de courbe et aux méthodes de Newton-Raphson par intervalles) est une priorité. Il en est de même des heuristiques de choix des contraintes à remplacer qui pourrait se baser sur d'autres stratégies de recherche.

Des exemples 3D de polyèdres ont été résolus efficacement par notre méthode. Mais la validation de ce type d'algorithme, ainsi que la découverte de nouveaux points d'achoppements dans la recherche de méthodes efficaces en 3D passe par la mise en commun des algorithmes proposés et surtout de l'univers dans lequel ils s'appliquent. Dans ce but, la réalisation de nos prototypes a été effectuée au sein d'une plateforme (présentée dans le chapitre suivant) capable de prendre en compte l'intégration rapide d'univers géométriques, et de nouvelles méthodes de résolution.

Chapitre 4

Mise en œuvre

Les prototypes décrits dans les sections précédentes prennent en paramètre un univers géométrique qui permet de spécifier la syntaxe et la sémantique utilisée par l'interface et la résolution.

Dans la pratique, ce paramètre doit être décrit dans un format de fichier adapté pour pouvoir faciliter la définition et l'échange d'univers géométriques mais aussi de systèmes de contraintes géométriques et de solutions. La recherche d'un tel format a conduit au développement d'un langage basé sur XML : le Geometric Constraints Mark-up Language ou Gcml.

Le développement de solveurs prenant en compte ce paramètre est facilité par l'utilisation d'une architecture orientée objet. Celle-ci permet d'accélérer leur intégration au sein de modeleurs par contraintes comme celui décrit au chapitre 2.

Pour permettre une utilisation plus efficace et plus simple de cette plate-forme logicielle, un outil est proposé permettant de générer automatiquement du code optimisé pour la résolution et l'interface à partir de la description d'un univers géométrique.

Sommaire

Dans les chapitres précédents, le lien qui unit univers géométrique d'une part et interface et résolution d'autre part a été mis en évidence. Si la formalisation des notions d'univers, de système, et de solution a permis de montrer comment guider l'interaction et les méthodes de résolution, il n'en reste pas moins que cette approche reste abstraite. Le but de ce chapitre est de montrer comment ce formalisme est pratiquement mis en œuvre.

Dans la première section, la structure du langage *Gcml* est détaillée et l'accent est mis sur la différenciation des descriptions de la syntaxe et de la sémantique des univers géométriques. Une architecture orientée objet où s'inscrivent des solveurs géométriques, et équationnels est ensuite décrite. Enfin, un outil pour la compilation automatique d'univers géométriques et de ses sémantiques vers différents domaines d'application est présenté.

4.1 Geometric Constraint Markup Language

Le *Gcml* [WSMF06] est un langage basé sur XML qui permet de décrire à la fois des univers et des systèmes de contraintes géométriques dans le cadre défini au chapitre 1. La description de la syntaxe et de la sémantique d'univers géométriques se passe au niveau méta : la syntaxe du *Gcml* est une méta-syntaxe. Cependant, pour éviter d'alourdir le discours, nous parlerons simplement de schéma du *Gcml*, et de la syntaxe des univers géométriques.

Le schéma du *Gcml* est définie par un ensemble de mots-clés (les balises) formant un cadre dans lequel l'univers géométrique n'est pas figé, celui-ci pouvant être modifié et étendu par simple modification du fichier *Gcml*. Les systèmes de contraintes géométriques sont décrits également en *Gcml*, en respectant la syntaxe de l'univers géométrique associé.

La distinction entre la syntaxe et les sémantiques d'univers géométriques permet de mettre en valeur les interprétations possibles d'un même univers et de combiner ainsi plusieurs descriptions sémantiques complémentaires. C'est aussi ce qui permet de séparer résolution formelle et résolution numérique.

4.1.1 Description de la syntaxe

Dans cette section, nous présentons, au travers d'un exemple, la partie du langage *Gcml* qui permet de définir la syntaxe d'un univers géométrique et des systèmes de contraintes qui s'y rapportent.

Univers géométrique

Conformément à la définition formelle présentée dans le chapitre 2, la syntaxe de l'univers géométrique se décompose en :

- un ensemble de sortes ;
- un ensemble de symboles fonctionnels ;

- un ensemble de symboles prédicatifs ;
- un ensemble d'axiomes.

Les balises traduisant ces notions sont respectivement :

- `sorts`, qui contient un ensemble de chaînes de caractères définissant les sortes ;
- `fsymbols`, qui contient un ensemble de symboles et leur arité et coarité ;
- `psymbols`, qui contient un ensemble de symboles et leur arité ;
- `axioms`, qui contient l'axiomatique de l'univers.

La figure 4.1 illustre cette partie du schéma *Gcml* avec un exemple d'univers géométrique 3D simple, composé de points, de longueurs et de contraintes de distance entre points.

Pour des raisons pratiques, le *Gcml* est prévu pour distinguer plusieurs types d'axiomes :

- les propriétés, pour décrire les relations propres à un symbole, comme par exemple la commutativité ;
- les axiomes constructifs, qui décrivent par exemple les constructions géométriques ;
- les axiomes de repères, indiquant les éléments à fixer pour produire des solutions particulières.

Les balises traduisant ces différentes parties de l'axiomatique sont :

- `properties` ;
- `construction` ;
- `location`.

Ces axiomes sont structurés par deux balises exprimant une condition : *if* et *then* permettant de différencier les prémisses des conclusions d'une règle.

Pour plus de souplesse dans la définition de problèmes de contraintes géométriques, les parties syntaxiques et sémantiques d'un même univers sont référencées par un lien vers un fichier externe. Par exemple, si l'univers précédent est contenu dans le fichier appelé `simple3d_syntax.gcml`, on peut alors y référer par :

```
<gcml>
    <universe>
        <syntax ref="simple3d_syntax.gcml" />
    </universe>
</gcml>
```

Système de contraintes géométriques

Un système de contraintes est constitué d'inconnues, de paramètres et de contraintes. Le langage *Gcml* regroupe donc dans la partie délimitée par la balise *gcs* (geometric constraint system), la partie du schéma suivant :

- `unknowns`, l'ensemble des inconnues typées par des sortes de l'univers géométrique considéré ;
- `parameters`, l'ensemble des paramètres typés également ;
- `constraints`, l'ensemble des contraintes portant sur les entités définies par les deux balises précédentes.

```
<gcml>
    <universe>
        <syntax>
            <sorts>
                point
                scalar
            </sorts>
            <fsymbols>
                <!-- Coordonnées d'un point -->
                initPoint : scalar scalar scalar -> point
                <!-- Intersection de deux sphères et du plan xOy -->
                interxOy : point scalar point scalar -> point
                <!-- Intersection de trois sphères -->
                mkPoint : point scalar point scalar point scalar -> point
            </fsymbols>
            <psymbols>
                <!-- Distance entre deux points -->
                distpp : point point scalar
            </psymbols>
            <axioms>
                <properties>
                    <!-- Propriété de commutativité de la distance -->
                    <property> distpp(p1,p2,l)=distpp(p2,p1,l) </property>
                </properties>
                <location>
                    <!-- Formation d'un repère -->
                    <if>
                        distpp(p1,p2,l1)
                        distpp(p2,p3,l2)
                        distpp(p3,p1,l3)
                    </if>
                    <then>
                        p1=initPoint(0,0,0)
                        p2=initPoint(l1,0,0)
                        p3=interxOy(p1,l1,p2,l2)
                    </then>
                </location>
                <construction>
                    <!-- Règle de construction -->
                    <if>
                        distpp(p1,p2,l1)
                        distpp(p1,p3,l2)
                        distpp(p1,p4,l3)
                    </if>
                    <then>
                        p1=mkPoint(p2,l1,p3,l2,p4,l3)
                    </then>
                </construction>
            </axioms>
        <syntax>
    </universe>
</gcml>
```

FIG. 4.1 – Univers géométrique 3D simple

```
<gcml>
    <universe>
        <syntax ref="simple3d_syntax.gcml" />
    </universe>
    <gcs>
        <syntax>
            <parameters>
                scalar l1 l2 l3 l4 l5 l6
            </parameters>
            <unknowns>
                point p1 p2 p3 p4
            </unknowns>
            <constraints>
                distpp(p1,p2,l1)
                distpp(p2,p3,l2)
                distpp(p3,p1,l3)
                distpp(p4,p1,l4)
                distpp(p4,p2,l5)
                distpp(p4,p3,l6)
            </constraints>
        </syntax>
    </gcs>
</gcml>
```

FIG. 4.2 – Description d'un tétraèdre

Dans l'univers de la figure 4.1, un tétraèdre dont les longueurs des six côtés sont imposées, peut être décrit syntaxiquement par le système de contraintes de la figure 4.2.

4.1.2 Description de sémantiques

Univers géométrique

Plusieurs sémantiques différentes peuvent être associées à une même syntaxe (*cf.* chapitre 1). Il suffit de faire correspondre aux sortes et symboles de la signature des éléments qui pourront être interprétés/compilés par la suite. Voici quelques sémantiques possibles :

- une sémantique combinatoire, indiquant les degrés de liberté et les degrés de restriction ;
- une sémantique cartésienne, qui exprime les variables et les équations cartésiennes des sortes et des symboles ;
- une sémantique graphique, qui définit l'affichage des objets et des contraintes ;
- une sémantique textuelle, qui traduit textuellement un problème de contraintes géométriques exprimée en *Gcml* ;
- une sémantique programmatoire ou opérationnelle, qui décrit une implantation dans un langage de programmation particulier.

```
<semantics>
    <semantic name="textuelle">
        <point>Soit un point %name.</point>
        <scalar>Soit un réel %name.</scalar>
        <distpp>La distance entre %arg1 et %arg2 est %arg3.</distpp>
    </semantic>
    <semantic name="cartésienne">
        <point>
            réel @x,@y,@z
        </point>
        <length>
            réel @l
        </length>
    </semantic>
    <semantic name="C++">
    <use name="cartésienne"/>
    <point>
        class Point
        {
            private :
            float %x, %y, %z;
            public :
            Point(float x,y,z) : %x(x), %y(y), %z(z)
        };
    </point>
    <initPoint>
        Point initPoint(float x, float y, float z)
        {
            return Point(x,y,z)
        }
    </initPoint>
    </semantic>
    <semantic name="graphique">
        <use name="C++"/>
        <point>
            glVertex(%x, %y, %z)
            glPrint(%name, %x + 0.1, %y, %z)
        </point>
        <length>
            glPrint("%name=%l", 0,0,0)
        </length>
        <distpp>
            glBegin(GL_LINE)
                $point(%arg1)
                $point(%arg2)
            glEnd()
            glPush()
                glTranslate((%arg1.%x + %arg2.%x)/2,
                            (%arg1.%y + %arg2.%y)/2,
                            (%arg1.%z + %arg2.%z)/2)
                $length(%arg3)
            glPop()
        </distpp>
    </semantic>
</semantics>
```

Ainsi, la balise *semantics* englobe des balises *semantic* dans lesquelles le nom de la sémantique associée est défini par un attribut. Les balises suivantes portent le nom des sortes et des symboles définis dans la signature.

En reprenant l'univers géométrique de la figure 4.1, la figure 4.3 donne quatre sémantiques (textuelle, cartésienne, C++ et graphique en pseudo-code OpenGL) dans le format *Gcml*.

Des accesseurs faisant parti du langage d'interprétation de la sémantique sont définis. Pour les sortes, il s'agit de `%name` qui correspond au nom du littéral. Pour les symboles, les noms des littéraux associés aux arguments sont définis par `%arg1`, `%arg2`, ..., `%argi`.

L'utilisation de `@` devant une chaîne de caractères permet de définir un élément interprétable dans une autre sémantique. Et l'utilisation de `$` devant une chaîne qui est soit une sorte, soit un symbole fait référence à la sémantique associée à l'élément passé en argument. Ainsi, certaines sémantiques peuvent être dépendantes d'autres sémantiques, ce qui est spécifié par l'utilisation de la balise *use*.

Système de contraintes géométriques

Le pendant sémantique des systèmes de contraintes équivaut à une valuation des variables. Dans le format *Gcml*, deux types de valuations sont considérés selon qu'elles correspondent aux informations tirées de l'esquisse (balise *sketch*) ou à des valeurs produites par un solveur (balise *valuation*).

Dans l'exemple de la figure 4.4, la partie *sketch* contient les valeurs réelles des distances entre les points, calculées par le modeleur lors de la saisie de l'esquisse.

Ces valeurs sont calculées grâce à une sémantique opérationnelle associée à l'univers géométrique.

La partie *valuation* contient un plan de construction qui contient des distances dont les valeurs numériques ont été proposées par l'utilisateur.

Précisons que les valeurs associées aux variables sont conformes à la signature annoncée dans l'univers géométrique et que les termes peuvent être de profondeur quelconque.

4.1.3 Vers un format d'échange

Le schéma du *Gcml*, que nous venons de décrire, peut être utilisée comme un format d'échange. En effet, la plupart des formats d'échanges existant en CAO [WSMF06] ne sont pas pensés pour manipuler des systèmes de contraintes géométriques. Les normes issues du dessin technique sont elles-mêmes insuffisantes pour spécifier un problème de contraintes car elles n'englobent pas les contraintes implicites, les ambiguïtés dues aux représentations et à la définition de certains objets, comme par exemple les angles (*cf.* chapitre 2 section 1).

De plus, les acteurs de la communauté des contraintes géométriques utilisent des formalismes internes différents et il n'existe pas de base de données commune

```
<semantic>
        <valuation>
            l1=3
            l2=4
            l3=5
            l4=1
            l5=2
            l6=1
            p1=initPoint(0,0,3)
            p2=initPoint(l1,0,3)
            p3=initPoint(l1,l2,3)
            p4=mkPoint(p1,l4,p2,l5,p3,l6)
        </valuation>
        <sketch>
            p1=initPoint(0,0,3)
            p2=initPoint(1.67,0,3)
            p3=initPoint(1,3,4)
            p4=initPoint(0,2,1)
            l1=2.5777
            l2=3.4564
            l3=4.2311
            l4=1.0121
            l5=0.1223
            l6=3.8778
        </sketch>
    </semantic>
```

FIG. 4.4 – Sémantique du SCG décrit à la figure 4.2

d'exemples permettant à tous de tester, valider et comparer les solveurs proposés. Ces tests, quand ils ont lieu, sont issus de conversions « manuelles » qui nécessitent un investissement conséquent.

La principale qualité du format proposé, est de favoriser les échanges en permettant de définir formellement le passage d'une signature à une autre en indiquant le morphisme adapté.

De plus, baser *Gcml* sur XML présente de nombreux avantages : la lisibilité, l'existence et la facilité d'écriture de parsers, la capacité d'y associer des feuilles de styles pour le visualiser, ou des grammaires pour en contrôler la validité syntaxique, *etc.* [Win05]

Dans la section suivante, nous décrivons l'architecture pour la résolution de contraintes géométriques que nous avons mise au point avec le *Gcml*. L'automatisation du processus de passage d'un univers à une application est présenté dans la dernière section de ce chapitre. L'implantation que nous avons retenue est en C++ mais il est possible de s'affranchir de ce langage et d'en utiliser un autre en redéfinissant la sémantique opérationnelle dans l'univers et le méta-compilateur. Ainsi, un utilisateur final peut se concentrer sur la conception de son solveur à l'aide de nos outils à la condition qu'il utilise le C++, mais peut très bien changer de langage

en redéfinissant simplement la sémantique dont il a besoin.

4.2 Architecture pour la résolution de contraintes

L'architecture proposée ici reprend et étend celle déjà mise en place dans l'équipe IGG du LSIIT. Elle repose sur un ensemble de bibliothèques permettant de tirer pleinement parti de la modularité de l'architecture.

Le schéma de la figure 4.5 en présente les différentes briques logicielles. Au centre se trouve le noyau, nommé *GcLib*, constitué d'interfaces permettant de faire le lien entre les données issues du *Gcml*, *via* la *GcmlLib*, et les différentes sémantiques que nous avons développées sous forme de bibliothèques :

- une sémantique géométrique contenant un ensemble courant d'objets géométriques en 2D et en 3D et les différentes opérations associées, nommée *GeomLib* ;
- une sémantique algébrique définissant des systèmes d'équations, nommée *AlgebraicLib* ;
- une sémantique numérique contenant des structures pour le calcul numérique, nommée *NumericLib* ;
- une sémantique combinatoire, contenant des structures et des opérations sur les graphes, nommée *GraphLib* ;
- une sémantique constructive, permettant la résolution par des systèmes experts, nommée *EsLib*.

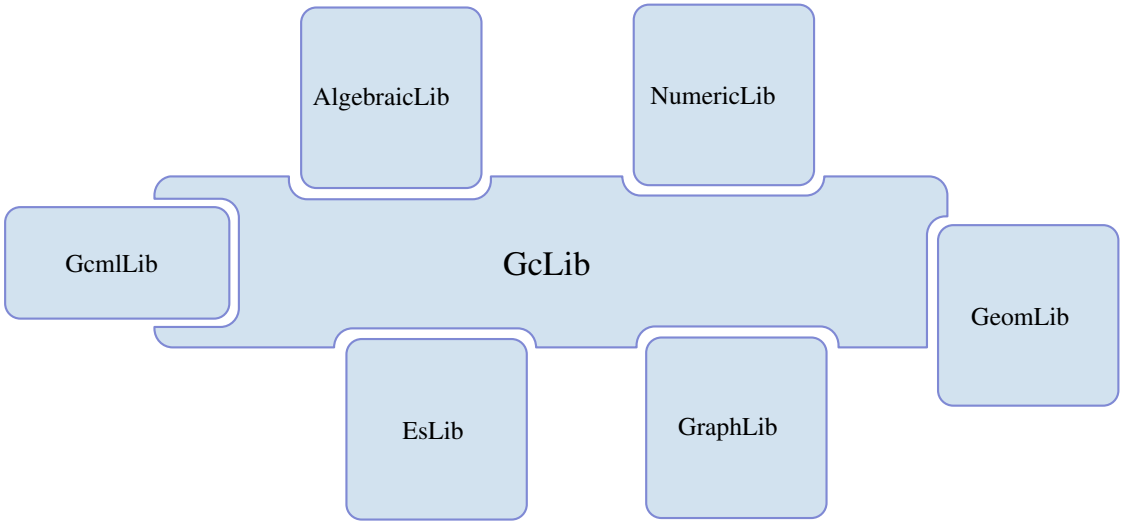


Fig. 4.5 – Schéma de la plate-forme pour la résolution de contraintes

Nous allons, dans cette section, décrire le noyau et détailler les solveurs géométriques et équationnels développés à partir des sémantiques algébriques, numériques, combinatoires, et constructives.

Les interactions avec le *Gcml* sont décrites à la section suivante et notamment, l'expression des mécanismes nécessaires à la traduction automatique de sémantiques normalisées vers une sémantique opérationnelle.

La description de chaque élément est faite en 2 niveaux : nous exposerons à chaque fois les principes mis en jeu d'un point de vue génie logiciel puis nous décrirons certains détails d'implantation. Dans un souci de simplification de l'exposé, certains aspects plus techniques concernant l'implantation sont décrits dans des paragraphes portant la mention « Détails techniques », qui pourront être omis en première lecture.

4.2.1 Noyau

Le noyau se présente sous la forme d'une bibliothèque composée d'un ensemble de conteneurs pour les éléments syntaxiques et sémantiques.

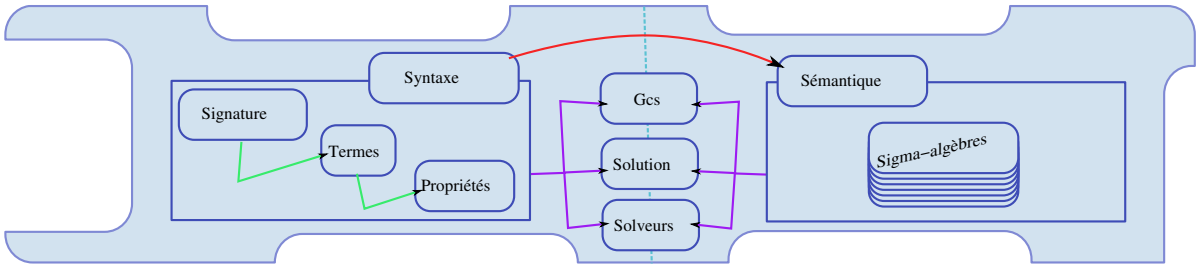


FIG. 4.6 – Schéma du noyau pour la résolution de contraintes

Le schéma de la figure 4.6 permet d'y distinguer :
- la syntaxe de l'univers géométrique avec sa signature, ses termes et son axiomatique ;
- le système de contraintes (Gcs) défini par rapport à l'univers géométrique ;
- les solutions ;
- une normalisation de l'interface des solveurs ;
- les sémantiques de l'univers géométrique (Σ-algèbres).

Détails techniques
Le noyau est principalement composé d'interfaces développées en C++ pour permettre de combiner efficacité et réutilisation.

L'utilisation de motifs de conception (Design Pattern) permet de rendre le code plus lisible et surtout réutilisable même en cas de modifications internes des structures. De plus, toute classe hérite d'une classe mère unifiant les entrées/sorties du programme pour plus de simplicité dans la gestion de l'interface.

Le travail d'implantation représente au total environ 35 000 lignes de code C++. □

Syntaxe d'un univers géométrique, d'un système de contraintes, et d'une solution

Suivant la définition classique, la signature est composée de trois ensembles : un ensemble de sortes S, et deux ensembles de symboles F et P avec les fonctions d'arité et de co-arité.

L'implantation d'une classe de construction des termes permet la génération automatisée de termes *valides* de profondeur quelconque respectant une signature donnée.

La signature peut bien sûr être enrichie ou appauvrie dynamiquement. Dans ce dernier cas, un pré-traitement des termes est appliqué pour supprimer les termes non-valides.

Détails techniques

Les sortes et les symboles sont simplement décrits par une chaîne de caractères.

On distingue plusieurs termes héritant d'une classe *Term* commune :

- le terme vide qui vérifie le motif de création *singleton*, qui restreint l'instanciation d'une classe à un seul objet ;
- les variables qui sont représentées par une chaîne de caractères associée à une sorte ;
- et les termes composés vérifiant le motif structurel *composite*, dont le dénominateur commun est un symbole et un tableau de termes ;
- *etc.*

La construction d'un terme n'est possible qu'à travers l'utilisation d'un motif *fabrique* dont le rôle est de produire le terme clos ou incomplet de manière correcte par rapport à la signature, suivant les arguments passés en paramètre. □

L'univers géométrique est composé d'une signature et d'un ensemble d'axiomes limités à l'expression des propriétés internes des symboles. Les propriétés que nous avons retenues sont des instances identifiées de propriétés comme par exemple la commutativité.

Détails techniques

Une propriété est implantée par le motif *monteur* qui permet de construire des objets complexes communs suivant des structurations différentes. Ainsi, la propriété de commutativité consiste en un monteur abstrait qui contient les éléments fonctionnels pour fabriquer des termes commutativement équivalents. Pour chaque type de commutativité, un monteur concret construit explicitement les termes suivant une structuration qui lui est propre. Ainsi, pour chaque propriété décrite dans l'univers géométrique, il est possible de construire automatiquement un monteur concret basé sur la syntaxe des termes. □

Un système de contraintes (Gcs pour geometric constraint system) est une conjonction finie de termes prédicatifs construits sur des variables réparties en deux ensembles : les paramètres et les inconnues.

Remarque

Ces variables sont vues commes des constantes dans l'univers géométrique logique considéré. Les « vraies » variables apparaissent dans les règles.

Ainsi, chaque description de système de contraintes peut se baser sur la syntaxe d'un univers géométrique défini lui aussi dynamiquement.

Les opérations classiques sur les systèmes de contraintes sont définies :

- l'union ;
- la différence ;
- la restriction par rapport à un ensemble de variables.

On peut noter bien sûr qu'un Gcs ne peut être défini qu'à partir d'un univers géométrique ou à la rigueur d'une signature.

Comme nous l'avons fait remarquer dans la description de la sémantique d'un système de contraintes géométriques, une solution peut être représentée par un plan de construction où valeurs numériques et paramètres se côtoient. D'un point de vue implantation, nous avons pris le parti de différencier les valuations formelles des valuations numériques.

Ainsi, un plan de construction est naturellement composé d'une liste de termes clos par rapport à la syntaxe de l'univers géométrique considéré. Évidemment, ces termes clos ne peuvent être que des termes fonctionnels.

Détails techniques
À cause de la gestion de termes de profondeur non fixée, une opération de factorisation est utilisée pour pouvoir comparer deux plans de constructions en ne comparant que deux termes. Cette factorisation consiste à écrire un plan de construction sous forme d'un terme de profondeur égale à la taille du plan de construction.
Par exemple, $\begin{cases} o_1 = f_1() \\ o_2 = f_2(o_1) \\ o_3 = f_3(o_1, o_2) \end{cases}$ donne le terme factorisé : $o_3 = f_3(o_1 = f_1(), o_2 = f_2(f_1()))$.
La fonction de recherche dans un plan potentiellement factorisé, du terme définissant une variable donnée, est disponible au sein de cette structure. □

Gestion des sémantiques

Le « créateur d'univers géométrique » décrit dans le fichier *Gcml* la sémantique qu'il désire en utilisant les mécanismes mis en place. Ainsi, la réalisation de nouvelles sémantiques des sortes et des symboles mais aussi des termes est laissée au soin du développeur utilisant le noyau. Seuls des conteneurs permettant de gérer les différentes Σ-algèbres sont disponibles.

Détails techniques
Ainsi, nous avons choisi de proposer des interfaces dont les classes implantant la sémantique devront hériter pour pouvoir bénéficier de la gestion d'une partie de la sémantique au niveau du noyau. Ces sémantiques sont alors stockées sous forme de tables de hachage et regroupées dans un conteneur nommé *SigmaAlgebra* pour une signature donnée. Une chaîne de caractères permet de les distinguer.

Une interface nommée *Values* permet d'uniformiser la représentation des solutions dont la sémantique n'est pas un plan de construction au travers d'une table de hachage nommée *AssignationMap* associant variables et Values.

□

Solveurs

L'interface des solveurs est normalisée : un solveur est perçu comme un conteneur qui autorise l'accès à une solution et le parcours de ses solutions.

Cette interface permet de combiner les solveurs de manière uniformisée au sein d'une approche par décomposition en favorisant le parcours de l'ensemble des solutions locales pour explorer toutes les solutions.

Détails techniques

L'implantation suit le motif de conception classique d'itérateur. Ce motif comportemental permet d'uniformiser l'accès et le parcours des solutions :

```
template <class TYPE>
class Iterator
{
public :
    Iterator() {}
    virtual ~Iterator() {}

    virtual void begin()=0;
    virtual bool end()=0;
    virtual void next()=0;
    virtual TYPE currentItem()=0;
};
```

On peut noter qu'il est souvent préférable d'un point de vue efficacité de rendre cet itérateur dynamique, c'est-à-dire de ne générer une solution qu'au moment où elle est demandée.

□

Pour coller à notre approche différenciant résolution symbolique et résolution numérique, nous distinguons deux types de solveurs : les solveurs numériques, que nous appelons assignateurs, et les solveurs symboliques.

Détails techniques

L'assignateur est alors un itérateur d'*AssignationMap* qui fonctionne dans un univers précis : une syntaxe d'univers géométrique et une Σ-algèbre, et un solveur symbolique est par héritage un itérateur d'assignateur. □

4.2.2 Sémantique d'univers euclidien

La *Geomlib* est une sémantique opérationnelle pour l'univers euclidien (*cf.* chapitre 1) contenant la sémantique équationnelle et la sémantique combinatoire. Nous décrivons ici cette sémantique associée aux sortes de l'univers euclidien 3D : *point*, *droite*, *cercle*, *sphere* et *plan*. Les fonctions décrites correspondent généralement à des constructions à la règle et au compas. Seule une partie de cette sémantique est décrite.

La sémantique combinatoire associée à une sorte ou à un symbole n'est pas forcément égale aux nombres de variables ou d'équations dans la sémantique équationnelle retenue. Effectivement, nous avons choisi, pour des raisons de facilité et d'intuition géométrique, de représenter les objets par des coordonnées et un ensemble de contraintes permettant de rendre canonique la représentation choisie.

Sortes

point : Un point est représenté classiquement par ses trois coordonnées cartésiennes :

$$P = (x_p, y_p, z_p)$$

Un point en 3D a 3 degrés de liberté conformément à ses coordonnées cartésiennes.

plan : Nous avons choisi de représenter un plan par son vecteur normal $\overrightarrow{n} = (a, b, c)$ et sa distance d signée à l'origine. Avec cette représentation, il existe plusieurs valeurs a, b et c déterminant le même plan. Le vecteur normal doit donc être normalisé pour avoir une représentation unique. Ainsi, les coordonnées du plan sont :

$$\Pi = (a, b, c, d)$$

et l'ensemble des points (x, y, z) décrivant le plan Π vérifient l'équation

$$ax + by + cz + d = 0$$

La normalisation est effectuée en considérant la contrainte suivante :

$$|a| + |b| + |c| = 1$$

Ce n'est pas la normalisation classique qui fait intervenir des puissances carrées. Nous avons préféré cette forme pour simplifier les calculs pour la correction numérique dans la reparamétrisation.

Un plan en 3D a 3 degrés de liberté, représentés par 3 réels : d et deux angles sur la sphère unitaire pour représenter le vecteur normal $\overrightarrow{n}$. Nous avons préféré une représentation avec 4 nombres réels, plus pratique pour les calculs mais qui introduit une équation en plus.

droite : Une droite en 3D est représentée par un point $A(x_A, y_A, z_A)$ et un vecteur direction $\overrightarrow{v}(\alpha, \beta, \gamma)$. Comme pour le plan, il existe une infinité de représentations possibles pour une même droite. Nous ajoutons donc les contraintes supplémentaires suivantes :

- le point doit être à une distance minimale de l'origine, *i.e.* le produit scalaire de $\overrightarrow{OA}$ par $\overrightarrow{v}$ doit être égal à zéro ;
- le vecteur est normalisé.

Ainsi, une droite est représentée par :

$$\Delta = (\alpha, \beta, \gamma, x_A, y_A, z_A)$$

Le système d'équations s'écrit naturellement :

$$\begin{cases} \beta(x - x_A) = \alpha(y - y_A) \\ \gamma(x - x_A) = \alpha(z - z_A) \end{cases}$$

Les équations de normalisation supplémentaires à considérer sont :

$$\begin{cases} \alpha x_A + \beta y_A + \gamma z_A = 0 \\ |\alpha| + |\beta| + |\gamma| = 1 \end{cases}$$

Nous n'avons pas choisi la représentation canonique d'une droite par deux équations de plans. En effet, cette représentation comportant 8 variables n'est pas très pratique à utiliser si on considère l'affichage d'une droite.

Une droite en 3D a 4 degrés de liberté. Ces degrés correspondent à une représentation à 4 réels : la distance à l'origine, deux angles sur la sphère unitaire et un angle sur le plan tangent à cette sphère.

cercle **:** Un cercle en 3D est représenté par un point $A(x_A, y_A, z_A)$, un rayon r, et un plan Π. Ainsi, le cercle a pour coordonnées :

$$C = (x_A, y_A, z_A, r, a, b, c, d)$$

Le système d'équations associé est :

$$\begin{cases} (x - x_A)^2 + (y - y_A)^2 + (z - z_A)^2 = r^2 \\ ax + by + cz + d = 0 \end{cases}$$

Avec l'équation supplémentaire :

$$|a| + |b| + |c| = 1$$

Un cercle dispose de 4 degrés de liberté qui sont les 3 degrés du plan Π plus le rayon r.

sphere **:** Une sphère en 3D est classiquement représentée par : un rayon r et un point $A(x_A, y_A, z_A)$.

Les coordonnées sont :

$$\Sigma = (x_A, y_A, z_A, r)$$

L'équation associée est :

$$(x - x_A)^2 + (y - y_A)^2 + (z - z_A)^2 = r^2$$

Une sphère dispose de 4 degrés de liberté qui représentent exactement sa représentation analytique.

Symboles fonctionnels

Les symboles fonctionnels de l'univers 3D décrits dans la *GcmlLib* sont des constructions géométriques classiques.

Associer une sémantique combinatoire à ces symboles n'a aucune utilité pour l'application de méthodes à base de graphes. Ainsi, nous détaillons ici la sémantique équationnelle de quelques symboles, en montrant les calculs numériques effectués, et les équations à résoudre. Certains calculs peuvent être décrits par une suite de constructions, comme par exemple le symbole $inter_{3s}$ qui dénote l'intersection de trois sphères.

$inter_{2pl}$ **:** L'intersection de deux plans non parallèles donne une droite qu'il est facile de représenter par les équations canoniques. Notre choix de représentation par un point et un vecteur de direction, même si elle est pratique pour l'affichage d'une droite, nécessite le calcul explicite de cette intersection.

Soit $\overrightarrow{n_1}$ et $\overrightarrow{n_2}$ les vecteurs normaux de π_1 et π_2, les plans considérés.

Le produit scalaire de deux vecteurs est noté « . » et le produit vectoriel « $\wedge$ ».

Alors, la droite d'intersection est définie par :

$$\delta = \begin{cases} A = \frac{-d_1 \times t_2 + d_2 \times t_3}{t_1 \times t_2 - t_3^2} \times \overrightarrow{n_1} + \frac{-d_2 \times t_1 + d_1 \times t_3}{t_1 \times t_2 - t_3^2} \times \overrightarrow{n_2} \\ \overrightarrow{v} = \overrightarrow{n_1} \wedge \overrightarrow{n_2} \end{cases}$$

avec $t_1 = \overrightarrow{n_1}.\overrightarrow{n_1}$, $t_2 = \overrightarrow{n_2}.\overrightarrow{n_2}$ et $t_3 = \overrightarrow{n_1}.\overrightarrow{n_2}$.

$inter_{ds}$ **:** L'intersection d'une droite $\delta = (A, \overrightarrow{v})$ et d'une sphère $\sigma = (B, r)$ peut donner au maximum deux points. Dans ce cas là, il suffit de résoudre une équation de degré 2 : $ax^2 + bx + c = 0$. Dans cette équation, nous avons :

$$\begin{cases} a = \overrightarrow{v}.\overrightarrow{v} \\ b = \overrightarrow{v}.\overrightarrow{OA} - \overrightarrow{v}.\overrightarrow{OB} \\ c = \overrightarrow{OA}.\overrightarrow{OA} + \overrightarrow{OB}.\overrightarrow{OB} \end{cases}$$

Les solutions de cette équation sont la distance entre A et les points d'intersection. Si le discriminant est négatif, il n'y a pas de solution ; si le discriminant est nul, il n'y a qu'une seule solution et s'il est positif, il y a 2 solutions.

$plan_{rad}$ **:** Lorsque deux sphères de centres et de rayons respectifs A et B et r_1 et r_2 se coupent, le plan radical contient le cercle de l'intersection. Ce plan se calcule aisément par :

$$\Pi_{rad} = \begin{cases} a = -2(x_A - x_B) \\ b = -2(y_A - y_B) \\ c = -2(z_A - z_B) \\ d = (x_A^2 + y_A^2 + z_A^2 - r_1^2) - (x_B^2 + y_B^2 + z_B^2 - r_2^2) \end{cases}$$

$inter_{3s}$ **:** L'intersection de trois sphères s_1, s_2 et s_3 est facilement calculable en appliquant la construction suivante :

- calculer le plan radical π_1 des sphères s_1 et s_2 ;
- calculer le plan radical π_2 des sphères s_1 et s_3 ;
- calculer la droite δ intersection de π_1 et π_2 ;
- calculer les points d'intersection entre δ et s_1.

Symboles prédicatifs

Les symboles prédicatifs de l'univers euclidien 3D s'expriment sous forme d'équations faisant intervenir des opérateurs trigonométriques pour les contraintes d'angle

et de degré deux pour les distances.

Bien sûr, nous ne présentons pas les contraintes d'incidence puisqu'elles ne sont que l'application des systèmes d'équations définissant un objet, présentés plus haut, combinées à certaines contraintes d'angle et/ou de distance. Par exemple, un point (x, y, z) appartenant à un plan π doit vérifier l'équation du plan présentée ci-dessus. Une droite $A, \overrightarrow{v}$ appartenant à un plan π doit vérifier une contrainte de parallélisme et l'appartenance de A à π.

Contraintes d'angles Trois types de contraintes d'angle sont considérés : entre plans, entre droites, et entre plan et droite. Ces contraintes ne font pas intervenir le placement relatif des objets les uns par rapport aux autres, mais seulement le positionnement selon un axe. Le degré de restriction de ces contraintes est donc égal à 1.

$angle_{2pl}$ **:** Soient $(\overrightarrow{n_1}, d_1)$ et $(\overrightarrow{n_2}, d_2)$, deux plans et α un angle. L'équation représentant la spécification d'un angle entre deux plans est :

$$|\overrightarrow{n_1}|.|\overrightarrow{n_2}| - cos(\alpha) = 0$$

$angle_{2d}$ **:** Soient deux droites $(A_1, \overrightarrow{v_1})$ et $(A_2, \overrightarrow{v_2})$ et l'angle α. La contrainte d'angle entre deux droites s'exprime dans l'univers analytique par :

$$|\overrightarrow{v_1}|.|\overrightarrow{v_2}| - cos(\alpha) = 0$$

$angle_{dpl}$ **:** Soit le plan $(\overrightarrow{n}, d)$, et la droite $(A, \overrightarrow{v})$ contraints par l'angle α. L'équation associée est :

$$|\overrightarrow{n}|.|\overrightarrow{v}| - sin(\alpha) = 0$$

Perpendicularité et parallélisme Les contraintes de perpendicularité et de parallélisme sont des cas particuliers des contraintes d'angles ci-dessus, à la différence que les degrés de restriction engendrés sont différents.

Nous ne considérons que les droites 3D coplanaires et perpendiculaires ou parallèles. Cette contrainte supplémentaire implique de considérer un degré de restriction égal à 1 en plus des équations présentes.

$perp_{2d}$ **:** Deux droites $(A_1, \overrightarrow{v_1})$ et $(A_2, \overrightarrow{v_2})$ sont perpendiculaires si et seulement si :

$$|\overrightarrow{v_1}|.|\overrightarrow{v_2}| = 0$$

Le degré de restriction total de cette contrainte est 2.

$parall_{2d}$ **:** Les deux droites précédentes sont parallèles si et seulement si :

$$|\overrightarrow{v_1}|.|\overrightarrow{v_2}| - 1 = 0$$

Le degré de restriction pour cette contrainte vaut aussi 2.

$parall_{2pl}$ - $perp_{2pl}$: Les contraintes de perpendicularité et de parallélisme entre plans sont représentées identiquement aux droites dans l'univers analytique lorsque l'on considère les vecteurs normaux. Mais, le degré de restriction est égal à 1 pour la contrainte de perpendicularité et 2 pour le parallélisme.

$perp_{dpl}$: Soit le plan $(\overrightarrow{n}, d)$, et la droite $(A, \overrightarrow{v})$. L'équation associée à la perpendicularité est :

$$|\overrightarrow{n}|.|\overrightarrow{v}| - 1 = 0$$

$parall_{dpl}$: Pour le parallélisme, l'équation est :

$$|\overrightarrow{n}|.|\overrightarrow{v}| = 0$$

Le degré de restriction de ces deux contraintes est égal à 2.

Contraintes de distance Les contraintes de distances ont pour degré de restriction le nombre 1, sauf $dist_{dpl}$ dont le degré vaut 2 et $dist_{2pl}$ et $dist_{dd}$ dont les degrés sont égal à 3.

$distp_{pp}$: La distance D entre deux points $A(x_A, y_A, z_A)$ et $B(x_B, y_B, z_B)$ est traduite par l'équation :

$$(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 - D^2 = 0$$

$dist_{ppl}$: La distance D entre un point $A(x_A, y_A, z_A)$ et un plan $P(\overrightarrow{n}(x_n, y_n, z_n), d)$ est associée à l'équation :

$$x_A \frac{x_n}{\|\overrightarrow{n}\|} + y_A \frac{y_n}{\|\overrightarrow{n}\|} + z_A \frac{z_n}{\|\overrightarrow{n}\|} + d - D = 0$$

$dist_{pd}$: La distance D entre un point $A(x_A, y_A, z_A)$ et une droite $(B(x_B, y_B, z_B), \overrightarrow{v}(x_v, y_v, z_v))$ a pour équation :

$$(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2 - (x_A \frac{x_v}{\|\overrightarrow{v}\|} + y_A \frac{y_v}{\|\overrightarrow{v}\|} + z_A \frac{z_v}{\|\overrightarrow{v}\|})^2 - D^2 = 0$$

$dist_{ps}$: L'équation représentant la distance entre un point $A(x_A, y_A, z_A)$ et une sphère $(B(x_B, y_B, z_B), r)$ est :

$$(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 - r^2 - D^2 = 0$$

$dist_{2pl}$: La contrainte de distance entre deux plans $\pi_1(\overrightarrow{n_1}, d_1)$ et $\pi_2(\overrightarrow{n_2}, d_2)$ si les deux plans sont parallèles est :

$$(d_1 - d_2)^2 - D^2 = 0$$

Le degré de restriction de cette contrainte est 3 car le parallélisme de deux plans implique une restriction supplémentaire de degré 2.

$dist_{dpl}$ **:** Soit un plan $\pi = (\overrightarrow{n} = (x_n, y_n, z_n), d)$ et une droite $\delta(A, v)$. L'équation traduisant la distance entre π et δ lorsqu'ils sont parallèles est :

$$x_A \frac{x_n}{\|\overrightarrow{n}\|} + y_A \frac{y_n}{\|\overrightarrow{n}\|} + z_A \frac{z_n}{\|\overrightarrow{n}\|} + d - D = 0$$

$dist_{pls}$ **:** La distance entre un plan $P(\overrightarrow{n}, d)$ et une sphère (A, r) est exprimée par :

$$x_A \frac{x_n}{\|\overrightarrow{n}\|} + y_A \frac{y_n}{\|\overrightarrow{n}\|} + z_A \frac{z_n}{\|\overrightarrow{n}\|} + d - D - r = 0$$

$dist_{dd}$ **:** La distance entre deux droites parallèles $l_1(A, \overrightarrow{v_1})$ et $l_2(B, \overrightarrow{v_2} = (x_v, x_y, x_z))$ est exprimée par :

$$(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2 - (x_A \frac{x_v}{\|\overrightarrow{v_2}\|} x_v + y_A \frac{y_n}{\|\overrightarrow{v_2}\|} + z_A \frac{z_n}{\|\overrightarrow{v_2}\|})^2 - D^2 = 0$$

Le degré de restriction vaut 3, car le parallélisme de deux droites restreint 2 degrés.

$dist_{ds}$ **:** Entre une droite $l(A, \overrightarrow{v})$ et une sphère $s(B, r)$ la distance est traduite par :

$$(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 - D^2 - r^2 = 0$$

$dist_{ss}$ **:** La distance entre deux sphères $s_1(A, r_1)$ et $s_2(B, r_2)$ correpond à la distance entre A et B moins la somme des deux rayons :

$$(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2 - (D^2 + r_1 + r_2)^2 = 0$$

Contraintes de tangence Les contraintes de tangence sont des cas particuliers de distances nulles. Leur degré de restriction est égal à 1. On les retrouve notamment pour les symboles prédicatifs suivants :

- $tang_{pls}$ en fonction de $dist_{pls}$;
- $tang_{ds}$ en fonction de $dist_{ds}$;
- $tang_{ss}$ en fonction de $dist_{ss}$.

4.2.3 Solveurs

Nous présentons ici la mise en œuvre de deux catégories de solveurs : les solveurs géométriques et les solveurs équationnels. Des bibliothèques adaptées à ce type de solveurs nous permettent de proposer deux solveurs géométriques à base de systèmes expert, deux versions d'un solveur numérique utilisant l'itération de Newton-Raphson.

Solveurs géométriques

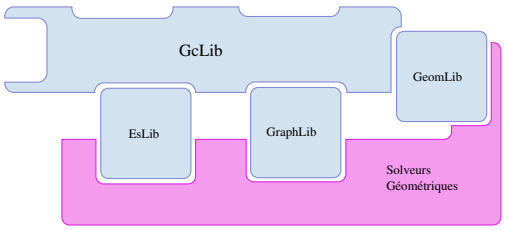


FIG. 4.7 – Solveurs géométriques

Les solveurs géométriques développés ont pour fondement la librairie *Eslib* qui propose des briques permettant de composer des solveurs à base de connaissances. Ces briques se traduisent par des composants de systèmes experts.

Les différentes manières de structurer un système expert sont bien connues et nous avons choisi de le décomposer en trois parties :
- la base de règles ;
- la base de faits ;
- le moteur d'inférence.

Les règles D'un point de vue pratique, la base de règles est une liste de règles composées de :
- un ensemble de termes qui sont les prémisses de la règle (si) ;
- un ensemble de termes qui sont les conclusions (alors) ;
- un ensemble de variables paramètres ;
- un ensemble de variables inconnues.

Ces deux derniers éléments sont utilisés pour vérifier les conditions d'application d'une règle après filtrage et unification. Nous verrons dans la section suivante, la manière de les déduire automatiquement des prémisses et des conclusions pour chaque règle.

Base de faits La base de faits contient un ensemble de termes prédicatifs. Lorsque le moteur d'inférence recherche un motif à appliquer, il applique un filtrage sur les faits de la base. Lors de ce filtrage, les propriétés des termes, comme par exemple la commutativité, sont exploitées pour générer l'ensemble des termes correspondant au motif recherché.

Le moteur d'inférence Le moteur d'inférence procède en deux étapes :
- la recherche de faits pouvant satisfaire l'unification ;
- et la vérification des conditions d'application de la règle.

La dernière étape est elle-même décomposée en 4 sous-étapes :
- les variables instanciées doivent être différentes ;
- les variables indéfinies dans la prémisse de la règle doivent être indéfinies dans le Gcs ;

- les variables utilisées dans chaque terme de la conclusion doivent avoir été définies préalablement ;
- la règle n'a jamais été appliquée avec les mêmes faits.

Nous avons vu dans la description du *Gcml*, que certaines règles sont applicables pour la recherche d'un repère. Dans ce cas, il faut prévoir deux modes de fonctionnement du moteur d'inférence : l'un permettant de s'arrêter lorsqu'un motif est trouvé, et l'autre lorsque le fait demandé a pu être déduit des faits initiaux. Un repère est décrit par un motif à rechercher dans un Gcs. On peut ainsi trouver plusieurs repères en étudiant une base de connaissances. L'ordre des règles sert d'heuristique pour orienter le choix d'un repère.

Inférence par chaînage avant Un solveur constructif utilisant le chaînage avant applique une recherche des règles appliquables sur la base de faits jusqu'à ce qu'il n'y ait plus d'inconnues dont la valeur soit calculable.
L'algorithme retenu est celui du parcours en largeur.

Contrôle de la rétro-propagation des degrés Comme nous l'avons vu au chapitre 2, la rétro-propagation des degrés de liberté consiste à partir d'un sommet d'un graphe vérifiant la formule $\sum ddl = \sum ddr$, puis à itérer sur le graphe privé des éléments ainsi déterminés. À la fin de cette rétro-propagation, nous tombons dans le cas d'un SCG bien contraint *modulo* les déplacements, sur un ensemble de nœuds racines censés constituer plus ou moins un repère. Dans ce cadre, on appelle plutôt cela une *ancre*, puisqu'il peut ne pas y avoir de contraintes communes entre ces objets.

Un solveur par chaînage arrière agit de la même manière en partant d'un objet quelconque mais sans considérer le critère combinatoire. La correction est dans ce cas assurée mais le choix des nœuds pour la propagation est fait au hasard.

Nous avons donc choisi une implantation hybride du solveur par chaînage arrière. Ainsi, nous disposons d'une fonction `formel` qui vérifie s'il existe effectivement une construction possible dans le cadre d'une rétro-propagation des degrés de libertés.

La rétro-propagation est effectuée sur un graphe de contraintes construit à partir du *Gcs*, avec des degrés de liberté et des degrés de restriction. Nous utilisons une fonction de *déficit local* `deficit` sur les sommets : $\texttt{deficit}(s) = ddr(s) - ddl(s)$ avec s un sommet, ddr une fonction donnant la somme des degrés de restriction associés aux arêtes adjacentes, et ddl le degré de liberté associé au sommet.

L'algorithme associé à cette rétro-propagation hybride est :

```
Tant que !fin
{
   s=tête(l)
   Tant que deficit(s) == 0 et formel(s)
   {
       supprime(s)
```

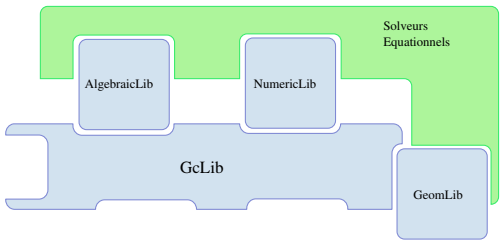


FIG. 4.8 – Solveurs équationnels

```
        maj(l)
    }
}
```

Nous utilisons une structure de graphe particulière, puisqu'elle permet de :
- mettre à jour les degrés lors d'une modification du graphe, *maj* ;
- calculer une liste l de sommets triés suivant leur déficit, *deficit* ;
- supprimer un sommet ainsi que toutes les arêtes incidentes à ce sommet, *supprime*.

Evaluation du plan de construction Les solveurs géométriques décrits ici fournissent des solutions sous forme de plan de construction. Il faut donc prévoir un assignateur capable d'évaluer numériquement les plans de constructions.

Un assignateur reposant sur les structures et fonctions définies dans la *GeomLib* a été développé. Pour tout symbole fonctionnel, une fonction est associée, par exemple, par le biais de la sémantique du *Gcml*, et elle est évaluée sur un ensemble d'objets de la GeomLib correctement initialisés, et qui forment les arguments du symbole.

Solveurs équationnels

Les solveurs équationnels proposés utilisent soit une sémantique équationnelle décrite dans le fichier *Gcml* (cf. section suivante), soit une sémantique opérationnelle comme la *GeomLib*.

La description syntaxique d'un système d'équations s'effectue grâce à la bibliothèque *AlgebraicLib* qui contient notamment les opérations de traduction de contraintes en équations, de dérivées formelles, *etc.*

La bibliothèque numérique *NumericLib*, elle, propose un ensemble d'outils numériques et principalement la manipulation de matrices de taille $m \times n$.

Deux solveurs implantant la méthode de Newton-Raphson ont été développés.

Le premier utilise au maximum la bibliothèque algébrique et notamment la représentation sous forme de système d'équations et le calcul d'une Jacobienne formelle.

Les calculs sont effectués par l'évaluation des équations associées.

Le second n'utilise que la bibliothèque *NumericLib* en calculant une dérivée nu-

mérique à l'aide de la formule classique :

$$f'(x) \simeq \frac{f(x+h) - f(x-h)}{2h}$$

avec h très petit. Les calculs sont effectués à l'aide de la sémantique opérationnelle contenue dans la bibliothèque *GeomLib*.

4.3 Méta-compilation d'un univers géométrique 3D

À chaque modification d'une des librairies ou d'extension de l'univers géométrique considéré, une phase de mise en conformité des solveurs est nécessaire. Celle-ci se trouve être répétitive et rébarbative. De plus, pour pouvoir se passer d'une description opérationnelle dans le *Gcml*, un mécanisme de traduction des sémantiques est nécessaire. C'est pourquoi nous avons mis en place un outil permettant d'automatiser ces tâches.

C'est donc à partir de la description en *Gcml* d'un univers géométrique que nous avons automatisé la génération d'applications pour résoudre des contraintes.

Le méta compilateur, nommé *muc* pour Meta-Universe Compiler, traduit le langage géométrique de haut-niveau *Gcml* en un programme via un langage de programmation de plus bas niveau.

Ce compilateur prend en entrée une description de la syntaxe et de la sémantique d'un univers géométrique en *Gcml* et un squelette d'application écrit en C++ complété par des macros spécifiques à *muc*.

Ce compilateur est écrit en Perl. Il génère les structures de données à partir de l'univers, la sémantique ainsi que les liens nécessaires entre cette sémantique et les structures de données formelles. Ces structures et ce code sont alors incorporés dans le squelette d'application par le biais de macros.

Le code est ensuite compilé automatiquement pour générer un exécutable pouvant traiter des systèmes de contraintes définis dans cet univers.

Les sémantiques suivantes sont disponibles :

1. opérationnelle, pour évaluer automatiquement des plans de constructions ;
2. combinatoire, pour spécifier les degrés de liberté et de restriction pour l'analyse de graphes ;
3. algébrique, pour paramétrer la résolution équationnelle en précisant les formules d'évaluation de chaque relation.

On peut remarquer que la description des deux dernières sémantiques est en fait une description des morphismes d'univers présentés dans le chapitre 1.

Les sémantiques doivent respecter une interface stricte pour pouvoir être branchées au noyau de résolution. Nous présentons donc ces interfaces permettant la génération de :

- solveurs constructifs ;
- solveurs combinatoires ;
- solveurs équationnels.

4.3.1 Génération de solveurs constructifs

Les solveurs constructifs prennent en paramètre les règles définies dans l'axiomatique de l'univers géométrique. Les règles de repères permettent de débuter la résolution lorsque le système est bien-contraint *modulo* les déplacements. Les propriétés permettent de ne pas surcharger inutilement la base de règles et sont utilisées seulement lorsque le moteur d'inférence en a besoin.

Mais la description des règles en *Gcml* ne précise pas de manière explicite quelles sont les inconnues et les paramètres. On peut facilement les découvrir en appliquant la convention suivante :

Les règles

Par convention, deux noms différents dans une règle désignet deux variables différentes. Il est facile de connaître les éléments qui doivent être découverts de ceux qui sont connus. Les premiers sont ceux qui appartiennent à la partie gauche des termes de la conclusion. Il faut aussi faire attention aux termes temporaires et aux termes qui utilisent des variables connues. Ainsi, les variables qui doivent être instanciées pour que la règle puisse être appliquée sont issues de l'application du petit algorithme ci-dessous.

La première règle décrit la construction du lieu géométrique d'un point (ici, un cercle) lorsque l'on sait qu'il est à une distance donnée d'un autre point :

```
<if>
    distpp(p1,p2,k)
</if>
<then>
    c = mkcircle(p1,k)
    center(c,p1)
    radius(c,k)
</then>
```

La seconde règle décrit le point intersection de deux cercles dont les centres sont deux points différents :

```
<if>
    distpp(p1,p2,k1)
    distpp(p1,p3,k2)
</if>
<then>
    c1 = mkcircle(p2,k1)
    center(c1,p2)
    radius(c1,k1)
    c2 = mkcircle(p3,k2)
    center(c2,p3)
```

```
    radius(c2,k2)
    p1=intercc(c1,c2)
</then>
```

Au départ, les éléments temporaires sont les éléments gauches n'appartenant pas aux prémisses :

- c (règle 1) ;
- $c1$ et $c2$ (règle 2).

Les paramètres sont les éléments droits des termes fonctionnels qui ne sont pas des éléments temporaires :

- $p1$ et k (règle 1) ;
- $p2$, $k1$, $p3$, $k2$ (règle 2).

Les inconnues dans la base sont les éléments gauches des termes de la conclusion appartenant aux prémisses mais pas aux paramètres de symboles fonctionnels :

- $p2$ (règle 1) ;
- $p1$ (règle 2).

Après l'application d'une règle, les inconnues de départ et les éléments temporaires ont une valeur connue.

En conclusion :

```
Temp := var(conclusion) - var(premisses)
param := RightVarFunc(conclusion) - Temp
unknowns := var(premisses) - param
```

Interprétation numérique

Pour faciliter l'évaluation numérique de plans de constructions, le noyau met à disposition un méta-solveur numérique qui permet à partir de la valuation des paramètres de fournir des solutions numériques, à la condition que la sémantique des termes fournie respecte une interface prédéfinie.

Cette interface prédéfinie pour les termes est constituée de deux descriptions, à savoir :

- l'une pour la sémantique des variables ;
- l'autre pour la sémantique des symboles fonctionnels et prédicatifs.

La sémantique pour les variables est la fonction définie dans le chapitre 1 permettant de passer d'un univers géométrique à un univers analytique. La sémantique pour les symboles correspond de même à la fonction associant équation et symbole fonctionnel ou prédicatif.

Les multi-fonctions sont exprimées au niveau de la sémantique des symboles en permettant de renseigner le nombre de solutions.

> **Détails techniques**
> Cet assignateur, nommé *Eval*, fournit une *AssignationMap* en sortie et gère les multi-fonctions à l'aide d'un vecteur d'indices représentant les différents embranchements possibles.
>
> De plus, il gère le gel de branche si l'*AssignationMap* de départ contient les valeurs numériques des inconnues issues de l'esquisse.

Un *monteur abstrait* est fourni pour pouvoir construire des sémantiques de variables suivant une sorte.

```
/**
 * Semantic for Variable
 */
template<class TYPE>
class SemVar : public gc::Value
{
public:
/**
 * Number of arguments
 */
virtual int getNbArg(void) const  = 0;
/**
   * Accessors
 */
virtual void setArg(int i, TYPE d) = 0;
virtual TYPE getArg(int i) const = 0;

virtual void draw() const {};
virtual void drawWithNames() const {};
};

/**
 * Class to build Variables
 */
template<class TYPE>
class VarBuilder : public gc::SortAttr
{
public:
virtual SemVar<TYPE> *build(const gc::Sort *sort) const = 0;

protected:
void print(std::ostream &out) const
{
out << "SemanticLib : Variable builder" << std::endl;
}
};

/**
 * Semantic for Symbols (Composite Terms)
 */
template<class TYPE>
class  SemSym : public gc::SymbolAttr
{
public:

/**
```

```
     * Multi-function information
     */
    virtual int multi(void) const { return 0; }
    /**
     * Evaluation function with valuated arguments and multi-functionnal
     * number
     */
    virtual gc::Value *evaluate(std::vector<gc::Value *>, int =0) const
    { return NULL; }
    /**
     * Predicative evaluator
     */
    virtual bool verify(std::vector<gc::Value *>, std::vector<TYPE> &)
    const { return false; }

    virtual void draw(std::vector<gc::Value *>) const {};
    virtual void drawWithNames(std::vector<gc::Value *>) const {};
    };
```

□

4.3.2 Génération de solveurs à base de graphes

Les solveurs à base de graphes utilisent les degrés de liberté et de restriction pour la résolution combinatoire.

Ces degrés sont représentés par des nombres entiers qu'il suffit de renseigner dans la sémantique combinatoire pour chaque objet et pour chaque contrainte.

▷ **Exemple 4.3.1**
Une partie de la sémantique combinatoire associée à l'univers géométrique euclidien 3D est :

```
<semantic name="combinatoire">
<scalar>1</scalar>
<measure>1</measure>
<point>3</point>
<line>4</line>
<plane>4</plane>
<onl>2</onl>
<onP>1</onP>
<norm>1</norm>
<onlP>2</onlP>
<angle2l>1</angle2l>
<angle3p>2</angle3p>
<distpp>1</distpp>
<distpl>1</distpl>
<distll>2</distll>
<distpP>1</distpP>
<distlP>1</distlP>
<distPP>3</distPP>
```

```
</semantic>
```

◁

4.3.3 Génération de solveurs à base d'équations

En *Gcml*, les équations sont décrites sous forme de formules respectant la syntaxe LaTeX. Ces équations sont alors analysées syntaxiquement et instanciées dans les structures de la bibliothèque *AlgebraicLib*.

Pour les sortes, un parenthèsage indique l'appartenance à un même objet. Pour les symboles, la balise *param* sert à indiquer la relation entre les noms des variables et les objets de l'arité. La balise *equations* permet de remplir les équations à raison d'une par ligne.

▷ **Exemple 4.3.2**
Pour les sortes *measure* et *point* et la contrainte $dist_{pp}$, la sémantique équationnelle est :

```
<semantic name="equations">
<measure>(m)</measure>
<point>(x,y,z)</point>
<distpp>
<param>(x_1,y_1,z_1) (x_2,y_2,z_2) (m)</param>
<equations>
(x_1-x_2)^2 + (y_1-y_2)^2 + (z_1-z_2)^2 - m^2
</equations>
</distpp>
</semantic>
```

◁

Ces équations seront soit interprétées directement par un solveur numérique, soit d'abord pré-calculées par un solveur polynomial en un système triangulaire du type bases de Gröbner.

L'utilisation de la syntaxe de LaTeX permet un affichage des équations par l'utilitaire *texvc* [Med] écrit en CAML.

Conclusion du chapitre

La mise en place d'une plateforme de modélisation par contraintes est un projet ambitieux. Le noyau que nous proposons permet de développer n'importe quel solveur en commençant par spécifier l'univers géométrique dans lequel il se situe. La spécification de cet univers est réalisée au travers d'un format de fichier basé XML que nous avons nommé Gcml. *Celui-ci permet de facilement décrire la syntaxe et les sémantiques correspondantes.*

Le passage de cette spécification à un programme éxécutable a été automatisé et nous proposons un méta-compilateur qui permet de spécialiser et d'optimiser certains solveurs existant à partir des sémantiques décrites dans le Gcml.

Les sémantiques prises en compte pour le moment sont la sémantique cartésienne, la sémantique combinatoire et une sémantique opérationnelle dans le langage C++. À partir de ces sémantiques, les solveurs que nous proposons sont : des solveurs formels à base de connaissances, des solveurs combinatoires, un solveur à base de l'algorithme de Newton-Raphson, et un solveur par reparamétrisation.

Conclusion

Ce travail, mené dans une équipe d'informatique graphique, porte sur l'interaction et la résolution de contraintes géométriques dans l'espace. Plusieurs domaines ont été explorés, que ce soit l'informatique géométrique et graphique bien sûr, mais aussi les domaines connexes et complémentaires comme la réalité virtuelle et l'intelligence artificielle. Tous ces domaines ont été abordés avec la volonté de produire des briques logicielles pratiques et réutilisables. Ces briques nous ont permis d'esquisser ce que pourrait être une interaction en 3D avec des logiciels de modélisation par contraintes en réalité virtuelle, et de formaliser une technique de résolution de contraintes en 3D permettant de s'attaquer à des classes de problèmes qui n'étaient pas résolubles de manière constructive. La mise en place d'outils permettant de simplifier le passage d'une description de l'univers géométrique considéré à un programme de modélisation par contraintes va nous permettre de multiplier les expérimentations dans ce domaine.

Sommaire

Bilan

Le travail exposé porte sur les contraintes géométriques de l'espace. Plus particulièrement, deux aspects ont été étudiés : d'une part la saisie d'une esquisse et de contraintes 3D et d'autre part la résolution des contraintes afin de produire les objets qui les satisfont. Aussi, dans ce travail, deux champs qui n'ont jusqu'alors pas été envisagés ensemble sont rapprochés : la réalité virtuelle et la résolution formelle de contraintes géométriques.

L'équipe d'informatique géométrique et graphique de Strasbourg disposant d'une expérience en réalité virtuelle, nous avons bénéficié de ce cadre et l'avons exploité pour aborder le problème de la saisie de contraintes 3D.

Nous avons développé une interaction gestuelle en réalité virtuelle qui comprend un dictionnaire de gestes permettant de désigner, sélectionner et manipuler des objets et des contraintes géométriques, et un ensemble d'outils pour manipuler ces objets contraints et naviguer dans la scène. Les contraintes qui expriment des dépendances entre objets, comme par exemple l'incidence d'un point à une droite, sont prises en compte par l'interface au niveau d'un gestionnaire permettant de guider la manipulation. Ce gestionnaire sert de lien entre l'esquissage de problèmes et la résolution de contraintes géométriques.

Nous avons expérimenté trois univers pour la modélisation en réalité virtuelle. Le premier permet d'effectuer des constructions géométriques dynamiques en 3D.

Le prototype Coyote-géomètre réunit l'interaction gestuelle que nous avons définie et un ensemble d'outils permettant de mieux gérer les constructions géométriques dans un environnement de réalité virtuelle, comme le Workbench dont nous disposons.

Une étude des univers isothétiques 3D est présentée. Elle montre comment les 3 univers isothétiques à base de contraintes de distance sont définis de manière formelle et comment les univers composés de droites ou de points placés isothétiquement se ramènent à l'univers des plans isothétiques. Pour ce dernier, la résolution et une interaction gestuelle adaptées sont décrites.

Enfin, les gestes peuvent être utilisés pour poser intuitivement des contraintes sur une esquisse 3D par le dessin de contraintes. Le problème de l'entrée des valeurs numériques peut se résoudre facilement soit par un widget dédié, soit par l'utilisation d'un geste permettant de modifier la valeur de l'esquisse.

La résolution de contraintes géométriques a principalement été étudiée en 2D avec des méthodes algébriques ou utilisant des constructions géométriques classiques. Ces méthodes sont d'ailleurs fréquemment employées dans un mécanisme de décomposition-recomposition. Les méthodes de résolution de contraintes 3D sont, pour les méthodes géométriques, des extensions des méthodes 2D. Malheureusement, si en 2D la plupart des problèmes sont solubles par ces méthodes, en 3D peu de situations se résolvent avec ces méthodes étendues. Notre approche, initiée simultanément par Gao [GHY04], consiste à transformer le problème initial en un problème plus simple à résoudre puis, après résolution, à rattraper les approximations faites lors de la transformation.

Nous avons formalisé cette voie et proposé de considérer les problèmes quasi-

décomposables suivant une méthode donnée. Nous avons appelé reparamétrisation la résolution de tels problèmes.

Cette technique transforme un système de contraintes géométriques S en un système S' par k-transmutation. Ces deux systèmes sont similaires du point de vue des solutions géométriques mais k contraintes sont différentes de telle manière que le système S' soit géométriquement finement décomposable, au sens où l'élaboration d'un plan de construction est garantie. Puis, une résolution numérique du système S'' obtenu par combinaison des systèmes S et S' et d'une fonction paramétrique donnant des solutions de S', permet dans la plupart des cas d'obtenir les solutions au système initial S.

Les véritables difficultés que nous avons mises à jour concernant cette méthode, se situent au niveau de la transformation de S en S', et de la résolution numérique.

Les heuristiques proposées pour appliquer une k-transmutation à la volée permettent dans bien des cas de retomber sur l'ensemble des solutions du système initial. Mais la correction numérique à l'aide de l'itération de Newton-Raphson fait perdre la complétude de cette méthode en lui faisant gagner en efficacité.

Nous avons montré par l'application sur des systèmes de contraintes de l'univers polyédrique l'efficacité de cette méthode et la possibilité de récupérer plusieurs solutions similaires à l'esquisse initiale.

Notre méthode permet de conserver le plus possible un caractère formel à la résolution. Cette méthode est ainsi une première tentative de méthode 3D permettant de s'approcher des qualités suivantes :

1. correction : toutes les figures trouvées par le solveur sont des solutions ;
2. complétude : toutes les solutions existantes sont trouvées ;
3. extensibilité : le solveur doit pouvoir s'adapter à tous les univers géométriques ;
4. convivialité : le solveur doit pouvoir
   - prendre en compte des énoncés sur- et sous-contraints ;
   - proposer la solution la plus « naturelle » ou la plus proche de l'esquisse ;
   - proposer toutes les solutions « acceptables » (prise en compte de contraintes métier) ;
   - expliquer pourquoi il n'y a pas de solutions ;
   - agir en temps réel.

Le lien entre les deux domaines, résolution et réalité virtuelle, passe par la définition précise des univers géométriques en jeu.

Un univers géométrique est composé d'une signature hétérogène décrivant la syntaxe des systèmes de contraintes géométriques, une axiomatique décrivant les propriétés de l'univers ainsi que des règles de construction, et un ensemble de sémantiques précisant le sens des différents termes exprimés dans cet univers.

Nous avons ainsi présenté un état de l'art des principales méthodes existantes suivant le critère de l'univers géométrique. Les trois principaux univers géométriques sont l'univers euclidien, l'univers analytique et l'univers combinatoire. Des restrictions de ces univers ont été intensivement étudiées en 2D et la plupart des travaux en 3D sont des généralisations des méthodes 2D. La résolution de systèmes de

contraintes 3D passe à notre avis par une étude plus poussée des univers restreints 3D correspondant aux univers 2D. Nous l'avons d'ailleurs mis en œuvre en étudiant successivement les problèmes isothétiques et les problèmes polyédriques en 3D.

L'utilisation de l'univers géométrique comme paramètre pour la résolution de systèmes de contraintes nous permet de mieux nous approcher d'une résolution multi-solveurs qui, à notre sens, est le meilleur moyen de résoudre les problèmes de contraintes 3D. Des solveurs hybrides qui combinent à la fois de la résolution formelle et de la résolution numérique approchée, constituent autant de pistes de recherche que nous aimerions développer.

Enfin, un important travail de génie logiciel a été fourni pour montrer que l'étude que nous avons réalisée en considérant un univers géométrique quelconque se traduit expérimentalement par un prototype d'une grande modularité.

Ainsi, un noyau pour le développement de solveurs permet de prendre en compte un univers géométrique. La signature peut être enrichie de manière dynamique et la gestion automatisée des termes facilite le développement de solveurs formels. Nous avons développé des solveurs constructifs basés sur des systèmes experts à chaînage avant ou arrière mais aussi des solveurs numériques basés sur l'itération de Newton-Raphson.

Un langage de description d'univers et de systèmes de contraintes, nommé *Gcml*, a été présenté. Il permet une description précise à la fois des éléments de syntaxe et des éléments de sémantique. Nous avons décrit une sémantique combinatoire et une sémantique équationnelle spécifiques, et leur utilisation au sein des solveurs précédents. Enfin un outil de compilation permet de passer automatiquement d'un univers géométrique à une application de résolution de contraintes optimisée pour les sémantiques décrites.

Quelques chiffres permettent de mesurer l'étendue des réalisations en C++ effectuées en collaboration avec Pascal Mathis, et Julien Wintz :

- une plate-forme pour les contraintes géométriques : *GcLib* et les sémantiques, représentant $\sim$ 12 500 lignes ;
- une bibliothèque pour la gestion du *Gcml* : *GcmlLib*, représentant $\sim$ 4 000 lignes ;
- un ensemble de solveurs : représentant $\sim$ 18 000 lignes ;
- un méta-compilateur : *muc*, représentant $\sim$ 1 000 lignes.

Perspectives

Les résultats exposés peuvent assez naturellement se prolonger dans plusieurs directions.

Le développement d'autres formes d'interactions à partir d'autres techniques issues de la réalité virtuelle fait bien sûr partie des perspectives futures à mettre en place. Par exemple, une interaction avec retour haptique pourra être étudiée grâce à l'utilisation d'un Spidar [TCH+03, CTH+03], installé sur le Workbench disponible à l'Université Louis Pasteur.

Nous envisageons aussi le recours à une interaction vocale pour faciliter les constructions géométriques et la pose de contraintes.

Pour la résolution de contraintes, plusieurs pistes sont actuellement en cours d'exploration. Tout d'abord, il nous semble nécessaire de trouver et de comparer d'autres heuristiques pour la reparamétrisation afin de déterminer une stratégie optimale de k-transmutation. Ensuite, la correction numérique doit être améliorée avec l'utilisation d'une méthode complète, comme l'homotopie ou l'analyse par intervalles par exemple.

En ce qui concerne le solveur isothétique, nous aimerions l'étendre pour prendre en compte les angles. De plus, nous aimerions prendre en compte des contraintes de plus en plus complexes pour la manipulation d'objets sous-contraints.

La description d'univers géométriques permettant de prendre en compte des « connaissances métiers » que ce soit en architecture, en usinage, ou bien dans d'autres domaines comme la chirurgie, ou la chimie, pourrait nous permettre d'adapter ou bien de trouver de nouvelles méthodes de résolution adaptées à ces univers.

Les avantages d'avoir développé le noyau et le méta-compilateur sont multiples :

- une spécification rigoureuse de l'univers considéré est moins sujette aux ambiguïtés ;
- tout le monde peut partager une spécification écrite dans ce langage,
- de nouvelles méthodes sont facilement et rapidement implantables, et il est possible de les comparer entre elles ;
- il est facile de partager et d'échanger de l'information et des problèmes.

Mais le principal désavantage d'un tel langage est qu'il peut devenir éprouvant de développer un univers de manière écrite. Aussi, une perspective à court terme serait d'élaborer une application permettant l'écriture rapide d'univers géométriques.
Ceci nous permettrait de tenter de propager à l'ensemble de la communauté, nos outils et notre langage dans le but de construire une base d'exemples et une base de méthodes de résolutions de contraintes géométriques. Ainsi, nous aimerions permettre de comparer pratiquement l'ensemble des méthodes de résolution existantes sur cette base d'exemples commune.

La description de morphismes de signature pour pouvoir faire correspondre des univers géométriques entre eux, est aussi une piste que nous comptons explorer pour permettre une communication inter-solveurs dans une approche multi-solveurs.

Pour parvenir à compléter les études entamées ici, plusieurs expériences peuvent être conduites.

Elles concernent l'apprentissage de la géométrie dans l'espace, et la modélisation par contraintes. En effet, nous aimerions dans un premier temps comparer une application sur support 2D traditionnel avec un support de réalité virtuelle. Dans un deuxième temps, nous sommes intéressés par la mise au point d'un outil de réalité virtuelle portatif à faible coût. Doté de ce matériel et des logiciels que nous avons développés, nous projetons d'expérimenter dans un cadre réel le comportement d'utilisateurs professionnels dans cet environnement. Par exemple, la découverte des constructions géométriques 3D peut être améliorée dans des classes du secondaire, et un outil de modélisation isothétique peut être testé dans des bureaux d'architecture.

Bibliographie

[AAM93] S. Ait-Aoudia and D. Michelucci. Reduction of Constraint Systems. In *Compugraphics Conference, Alvor*, pages 83–92, 1993. 18, 21, 41, 54

[ADT92] R. Allen, C. Desmoulins, and L. Trilling. Tuteurs intelligents et intelligence artificielle : problèmes posés en construction de figures géométriques. In *Proceedings of the ITS Conference,Montréal*, pages 325–334. Springer-Verlag, 1992. 21

[Ald88] B. Aldefeld. Variation of geometries based on a geometric-reasoning method. *Computer Aided-Design*, 30(3) :117–126, 1988. 17, 20, 43

[ARMP02] R. Joan Arinyo, A. Soto Riera, S. Vila Marta, and J. Vilaplana Pasto. Revisiting decomposition analysis of geometric constraint graphs. *Computer Aided-Design*, 36 :123–140, 2002. 36, 39, 46, 47

[Bau90] Y. Baulac. *Un micro-monde de géométrie - Cabri-Géomètre*. PhD thesis, Université Joseph Fourier, Grenoble, 1990. 21

[Bau91] Y. Baulac. Cabri-géomètre, A tool for computer aided geometry. *Wheels for the mind. 5 (1), 30-34.*, 1991. 68

[BFH+95] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A Geometric Constraint Solver. *Computer Aided-Design*, 27(6) :487–501, 1995. 18, 20, 40

[BH97] D. A. Bowmann and L. F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Symposium on interactive 3D graphics*, 1997. 62

[BNT98] C. Bliek, B. Neveu, and G. Trombettoni. Using graph decomposition for solving continuous CSPs. In *Principles and practice of constraint programming*, volume 1520. Lecture Notes on Computer Science, 1998. 54

[Bow99] D. A. Bowmann. *Interaction Techniques for grabbing and manipulating remote objects in immersive virtual environnements*. PhD thesis, 1999. 61

[BP07] Dominique Bechmann and Bernard Péroche, editors. *Informatique graphique, modélisation géométrique et Animation*, chapter Modélisation géométrique par contraintes par C. Jermann, D. Michelucci et P. Schreck. Hermes - Traité IC2, à paraître, 2007. 17, 111

[Brü86] B. Brüderlin. Constructing three-dimensional geometric objects defined by constraints. In *Proceedings of the ACM Siggraph Workshop on Interactive 3D Graphics*, pages 111–129, 1986. 44

[Brü88] B. Brüderlin. Automatizing Geometry Proofs and Constructions. In *Computational Geometry and its Applications*, Lecture Notes in Computer Science. Springer-Verlag, 1988. 44

[Buc85] B. Buchberger. *Multidimensional Systems Theory : Theory and applications*. N.K. Bose, 1985. 50

[But79] M. Buthion. Un programme qui résout formellement des problèmes de constructions géométriques. *RAIRO Informatique*, 13(1) :73–106, 1979. 20, 43

[Cau13] A.-L. Cauchy. *2° Mémoire - Recherches sur les polygones et les polyèdres*. 1813. 44

[CBG02] C.Jermann, B.Neveu, and G.Trombettoni. A new structural rigidity for geometric constraints systems. In *4th International Workshop on Automated Deduction in Geometry (ADG'02)*, number 2930, pages 87–105. Springer LNAI, 2002. 43

[CDG85] U. Cugini, C. Devoti, and P. Galli. System for parametric definition of engineering drawings. In *MICAD*, 1985. 19

[Cha99] S. Channac. *Conception et mise en oeuvre d'un système déclaratif de géométrie dynamique*. PhD thesis, Université Joseph Fourier - Grenoble 1, 1999. 68

[Che92] G. Chen. Les constructions à la règle et au compas par une méthode algébrique. Technical Report Rapport de DEA, Université Louis Pasteur, 1992. 51

[Cho88] S.C. Chou. *Mechanical geometry theorem proving*. Mathematics and its Applications. D. Reidel, Dordrecht, 1988. 17, 50, 51

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. MIT press, 1990. 54

[CTH+03] S. Coquillart, N. Tarrin, S. Hasegawa, L. Bouguila, and M. Sato. The Stringed Haptic Workbench. In *Sketches and Applications, SIGGRAPH*, 2003. 152

[DC] D-Cubed. http://www.d-cubed.co.uk. 39

[Dec90] R. Dechter. Enhancement schemes for constraint processing : Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 2(41) :273–312, 1990. 110

[Del00] F. Delobel. *Résolution de systèmes de contraintes réelles non linéaires*. PhD thesis, Université de Nice Sophia-Antipolis, 2000. 53

[DH99] C. Durand and C.M. Hoffmann. Variational constraints in 3D. In *Internatinal Conference on Shape Modeling and Applications, Aizu, Japan*, pages 90–97, 1999. 53

[DH00] C. Durand and C.M. Hoffmann. A Systematic Framework for Solving Geometric Constraints Analytically. *Journal of Symbolic Computation*, 30(5) :493–519, 2000. 53

[DM58] A.L. Dulmage and N.S. Mendelsohn. Covering of bipartite graphs. *Canadian Journal Of Mathematics*, 10 :517–534, 1958. 54

[DMS98] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence Journal*, 99(1) :73–119, 1998. 6, 17, 21, 43

[Doh95] M. Dohmen. A survey of constraint satisfaction techniques for geometric modeling. *Computer and Graphics*, 19(6) :831–845, 1995. 2

[Dur98] C. Durand. *Symbolic and Numerical Techniques for Constraint Solving.* PhD thesis, Purdue University, 1998. 53

[Eid] Eidos. Le jeu vidéo hitman 2 : Silent assassin. http://www.hitman2.com/. 59

[EV01] C. Essert-Villard. *Sélection dans l'espace des solutions engendrées par un plan de construction géométrique.* PhD thesis, Université Louis Pasteur, Strasbourg, 2001. 6, 22, 105

[EVSD02] C. Essert-Villard, P. Schreck, and J.-F. Dufourd. Combination of Automatic and Interactive Tools for Solution Space Browsing. In *Solid Modeling and Applications*, 2002. 106

[EY88] L.W. Ericson and C.K. Yap. The design of LINETOOL, a geometric editor. In *SCG '88 : Proceedings of the fourth annual symposium on Computational geometry*, pages 83–92, New York, NY, USA, 1988. ACM Press. 50

[FCMS03] A. Fabre, H. Chreif, P. Mathis, and P. Schreck. Constructions géométriques en 3 dimensions : Visualisation et manipulation dans un environnement de réalité virtuelle. In *AFIG'03*, 2003. 69

[FH97] I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Trans. Graph.*, 16(2) :179–216, 1997. 21, 40

[FHZ96] A. Forsberg, K. Herdin, and R. Zelesnik. Aperture Based Selection for immersive virtual environments. In *ACM symposium on user interface software and technology*, 1996. 62

[Fit81] W. Fitzgerald. Using axial dimensions to determine the proportions of line drawings in computer graphics. *Computer-Aided Design*, 13(6), 1981. 33

[FMJ05] S. Foufou, D. Michelucci, and J.-P. Jurzak. Numerical decomposition of geometric constraints. In *SPM '05 : Proceedings of the 2005 ACM symposium on Solid and Physical Modeling*, pages 143–151, New York, NY, USA, 2005. ACM Press. 53, 55

[FMS04] A. Fabre, P. Mathis, and P. Schreck. 3d geometric constructions in virtual reality. In *Virtual Reality International Conference 2004 - Laval Virtual*, 2004. 69

[FS06] A. Fabre and P. Schreck. Solving 3d quasi-decomposable geometric constraint systems. In *Automatic Deduction in Geometry, Pontevedra, Spain*, 2006. 98

[FSSB06] Arnaud Fabre, Ludovic Sternberger, Pascal Schreck, and Dominique Bechmann. Constrained gesture interaction for 3d geometric constructions. In *Gesture Workshop 05*, volume 3881, pages 324–334. Lecture Notes in Artificial Intelligence, 2006. 70

[Fuc03] P. Fuchs. *Le traité de la réalité virtuelle*. Ecole des mines de Paris, 2003. 59

[Geo98] Geospacw. http ://www2.cnam.fr/creem/, 1998. 69

[GHY02] X.-S. Gao, C.M. Hoffmann, and W.-Q. Yang. Solving spatial basic geometric constraint configurations with locus intersection. In *SMA '02 : Proceedings of the seventh ACM symposium on Solid Modeling and Applications*, pages 95–104, New York, NY, USA, 2002. ACM Press. 88

[GHY04] X.-S. Gao, C.M. Hoffmann, and W.-Q. Yang. Solving spatial basic geometric constraint configurations with locus intersection. *Computer-Aided Design*, 36 :111–122, 2004. v, 45, 47, 48, 88, 150

[GS01] J. Garloff and A.P. Smith. Investigation of a subdivision based algorithm for solving systems of polynomial equations. *Journal of nonlinear analysis : Series A Theory and Methods*, 47(1) :167–178, 2001. 53

[GSS97] J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity*. Graduate Studies in Mathematics. American Mathematical Society, 1997. 19

[GWM+00] J. Goguen, T. Winkler, J. Mesguer, K. Futatsugi, and J.-P. Jouannaud. *Software Enginneering with OBJ : algebraic specification in action*. Kluwer, 2000. 12

[GZ03] X.-S. Gao and G.-F. Zhang. Geometric Constraint Solving Based on Connectivity of Graph. *AMSS*, 2003. 46

[HB78] R. Hillyard and I. Braid. Analysis of dimensions and tolerances in computer-aided mechanical design. *Computer Aided Design*, 10(3), 1978. 52

[HC02a] C. M. Hoffmann and C.-H. Chiang. Variable-radius circles of cluster merging in geometric constraints (part ii). *Computer-Aided Design*, 34 :799–805, 2002. 43

[HC02b] C.M. Hoffmann and C.-H. Chiang. Variable-radius circles of cluster merging in geometric constraints (part i). *Computer-Aided Design*, 34 :787–797, 2002. 19, 43

[Hil71] D. Hilbert. Edition critique préparée par P. Rossier avec le concours du CNRS, Dunod, 1971. 14

[HLS98] C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Geometric Constraint Decomposition. In B. Bruderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 170–195. Springer, 1998. 20

[HLS01a] C. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition Plans for Geometric Constraint Problems, part II : New Algorithms. *J. Symbolic Computation*, 31 :409–427, 2001. 43

[HLS01b] C. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition Plans for Geometric Constraint Systems, part I : Performance Measures for Computer-Aided Design. *J. Symbolic Computation*, 31 :367–408, 2001. 43

[HT73] J.E. Hopcroft and R.E. Tarjan. Dividing a graph into triconnected components. *SIAM J.Comput.*, pages 135–158, 1973. 47

[HV95] C.M. Hoffmann and P.J. Vermeer. A spatial constraint problem. In J.-P. Merlet and B. Ravani, editors, *Computational Kinematics '95*, pages 83–92. Kluwer Academic, 1995. 20

[Jac95] N. Jackiw. The Geometer's Sketchpad. http://www.dynamicgeometry.com/, 1991-1995. 68

[Jer02] C. Jermann. *Résolution de contraintes géométriques par rigidification récursive et propagation d'intervalles.* PhD thesis, Université de Nice Sophia Antipolis, 2002. 19, 20, 43, 54

[JNT03] C. Jermann, B. Neveu, and G. Trombettoni. Algorithms for Identifying Rigid Subsystems in Geometric Constraint Systems. In *Int Joint Conference on Artificial Intelligence, IJCAI 2003 Acapulco, Mexique.*, pages 233–238, 2003. 43

[JTNR00] C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A Constraint Programming Approach for Solving Rigid Geometric Systems. In *proceedings of CP2000*, volume 1894 of *Lectures Notes in Computer Science*, pages 233–248. Springer, 2000. 54

[Kal91] K. Kalkbrenner. Elimination theory. Technical report, Research Institute for Symbolic Computation, Johannes Kepler Universität Linz, Austria, 1991. 49

[Kau02] H. Kaufmann. Construct3D : An Augmented Reality Application for Mathematics and Geometry Education. In *ACM Multimedia Conference*, 2002. 69

[KGC97] G. Kwaiter, V. Gaildrat, and R. Caubet. Interactive constraint system for solid modeling objects. In *SMA '97 : Proceedings of the fourth ACM symposium on Solid Modeling and Applications*, pages 265–270, New York, NY, USA, 1997. ACM Press. 2

[Kon92] K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer Aided Design*, 24(3), 1992. 17, 50

[Kor99] U. Kortenkamp. *Foundations of Dynamic Geometry.* PhD thesis, Swiss Federal Institue of Technology, Zurich, 1999. 68

[KR87] A. Kanevsky and V. Ramachandran. Improved algorithms for graph four-connectivity. In *28th Ann. IEEE Symp. Foundations of Computer Science*, pages 252–259, 1987. 47

[Kra90] G.A. Kramer. Solving Geometric Constraint Systems. In *National Conference on Artificial Intelligence*, pages 708–714, 1990. 45

[Kra91] G.A. Kramer. Using degrees of freedom analysis to solve geometric constraint systems. In *SMA '91 : Proceedings of the first ACM symposium on Solid modeling foundations and Computer-Aided Design/CAM applications*, pages 371–378, New York, NY, USA, 1991. ACM Press. 45

[Kra92] G.A. Kramer. A geometric constraint engine. *Artificial Intelligence*, 58 :327–360, 1992. 45

[KTY96] K. Kiyokawa, H. Takemura, and N. Yokoya. An empirical study on two-handed/collaborative virtual assembly. In *International Display Workshops*, 1996. 59

[Lab98] N. Van Labeke. Calques 3D : a microworld for spatial geometry learning. *ITS'98 - System Demonstrations, San Antonio (Texas), August 16-19*, 1998. 69

[Lab99] N. Van Labeke. *Prise en compte de l'usager enseignant dans la conception des EAIO.* PhD thesis, Université Henri Poincaré à Nancy, 1999. 69

[LG82] R. Light and D.C. Gossard. Modification of geometric models through variational geometry. *Computer-Aided Design*, 14(4) :209–214, 1982. 52

[Lig85] R. Light. Symbolic dimensioning in Computer-Aided Design. Master's thesis, Massachussets Institute of Technology, June 1985. 52

[LL92] C. Laborde and J.-M. Laborde. Problem solving in geometry : from microworlds to intelligent computer environments. *Ponte J. et al. (eds.) Mathematical Problem Solving and New Information Technology (NATO ASI Series vol. 89, pp.177-192). Berlin : Springer Verlag*, 1992. 68

[LLG81] R. Light, V. Lin, and D. Gossard. Variational Geometry un Computer-Aided Design. *Computer-Aided Design*, 15(3), 1981. 52

[LM94a] H. Lamure and D. Michelucci. Decomposition of 2D constraints graphs. Technical report, Ecole des Mines, Saint-Etienne, 1994. 20

[LM94b] H. Lamure and D. Michelucci. Résolution de contraintes géométriques par homotopie. *Actes de la Conférence AFIG*, 1994. 53

[LM95] H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. In *SMA '95 : Proceedings of the third ACM symposium on Solid modeling and applications*, pages 263–269, New York, NY, USA, 1995. ACM Press. 53

[LM96] R.S. Latham and A.E. Middleditch. Connectivity analysis : a tool for processing geometric constraints. *Computer-Aided Design*, 28(11) :917–928, 1996. 54

[LM98] H. Lamure and D. Michelucci. Qualitative study of geometric constraints. In *Geometric Constraint Solving and Applications*, pages 234–258. B. Brderlin and D. Roller Ed., Springer, 1998. 55

[Mat97] P. Mathis. *Constructions géométriques sous contraintes en modélisation à base topologique.* PhD thesis, Université de Strabourg, 1997. 20, 44

[Mec94] R. Mechling. Euklid, logiciel de géométrie allemand (shareware). http://www.mechling.de, 1994. 68

[Med] MediaWiki. Texvc, tex validator and converter. http://en.wikipedia.org/wiki/Texvc. 146

[MF06] D. Michelucci and S. Foufou. The Witness Configuration Method. *Computer-Aided Design*, 38(4) :284–299, 2006. 55

[Min95] M. R. Mine. Virtual environment interaction techniques. Technical Report TR95-018, UNC Chapell Hill Computer Science, 1995. 62

[MJ04] D. Michelucci and J.-P. Jurzak. An implementation of Jurzak's prover. In *isiCAD : Constraint-based Approaches and Methods of Mathematical Modelling for Intelligent CAD/CAM/CAE systems : From Methods to Applications*, 2004. 55

[MP05] Bernard Mourrain and Jean-Pascal Pavone. Subdivision methods for solving polynomial equations. Technical Report RR-5658, INRIA Sophia-Antipolis, 2005. 53

[Nel85] G. Nelson. Juno, a constraint-based graphics system. In *Proceedings of the ACM Siggraph'Conference*, volume 19, pages 235–243, 1985. 52

[Nin06] Nintendo. La manette wii. http://wii.nintendo.com/fr/controller.html, 2006. 59

[OH99] H. Ouhaddi and P. Horain. Hand tracking by 3D model registration. *Laval Virtual*, 1999. 66

[Owe91] J. Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of the 1th ACM Symposium of Solid Modeling and Computer-Aided Design/CAM Applications, Austin, Texas*, pages 397–407. ACM Press, 1991. 18, 20, 21, 38, 46

[PBWI96] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The Go-go Interactive technique : non-linear mapping for direct manipulation in VR. In *ACM symposium on user interface software and technology*, 1996. 62

[PR86] H.O. Peitgen and P.H Richter. *The beauty of fractals, Images of Complex Dynamical Systems.* Springer-Verlag, 1986. 52

[PSP99] J. S. Pierce, B. C. Stearns, and R. Paush. Voodoo dolls : seamless interaction at multiple scales in virtual environments. In *Symposium on interactive 3D graphics*, 1999. 62

[Qas97] S. Qasem. *Conception et Réalisation d'Une Interface 3D Pour Cabri-Géomètre.* PhD thesis, Université Joseph Fourier - Grenoble 1, 12 décembre 1997. 69

[Qt] Qt. (prononcer "cute"). http://www.trolltech.com/. 63

[R.E96] R.E.Moore. *Interval Analysis.* Prentice-Hall, 1996. 53

[Rit50] J.F. Ritt. Differential algebra. In *American Mathematical Society, Colloquium publications, XXXIII.* Reprinted by Dover Publications, Inc (1966), 1950. 50

[RSV88] D. Roller, F. Schonek, and A. Verroust. Dimension driven geometry in Computer-Aided Design : A survey. Technical report, LIENS, Ecole Normale Supérieure, Paris, 1988. 3, 17

[SB04] L. Sternberger and D. Bechmann. Deformable Ray-Casting Interaction Technique. In *Young'VR*, 2004. 75

[Sch93] P. Schreck. *Automatisation des constructions géométriques à la règle et au compas*. PhD thesis, Université Louis Pasteur - Strabourg, 1993. 11, 20, 21, 25, 43

[Sch01] P. Schreck. Robustness in Computer-Aided Design Geometric Construction. In *Proceedings of the fifth International Conference on Information Visualisation, IV2001 (London)*. IEEE, 2001. 22

[Sch02] P. Schreck. *Résolution Symbolique de contraintes géométriques – Habilitation à Diriger des Recherches*. PhD thesis, Université Louis Pasteur - Strasbourg, 2002. 6, 41

[SCP95] R. Stoackley, M. J. Conway, and R. Paush. Virtual Reality on WIM, Interactive World in Miniature. In *CHI*, 1995. 62

[Sit03] M. Sitharam. FRONTIER, an opensource gnu geometric constraint solver : version 3 (2003) for general 2D and 3D systems. http://www.cise.ufl.edu/~sitharam, 2003. 20

[SSK01] Y. Sato, M. Saito, and H. Koike. Real-time Input of 3D Pose and Gestures of a User's Hand and Its Applications for HCI. *IEEE Virtual Reality*, 2001. 66

[Ste06] Ludovic Sternberger. *Interaction en Réalité Virtuelle*. PhD thesis, Université Louis Pasteur - Strasbourg, 2006. 7, 59, 62

[Sun86] G. Sunde. Specification of shape by dimensions and other geometric constraints. In *IFIP WG 5.2 on Geometric Modeling, Rensselaerville*, 1986. 19, 21, 34

[Sun87] G. Sunde. A Computer-Aided Design system with declarative specification of shape. In *Eurographics Workshop on Intelligent Computer-Aided Design systems, Noordwisjkerout*, pages 90–101, 1987. 20, 23, 34

[Sut63] I.E. Sutherland. Sketchpad : A man-machine graphical communication system. In *Proceedings of the IFIP Spring Joint Computer Conference*, 1963. 2, 17

[SW00] R. Salomon and J. Weissmann. Evolutionary Tuning of Neural Networks for Gesture Recognition. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1528–1534, Piscataway, NJ, 2000. IEEE Service Center. 66

[TCH+03] N. Tarrin, S. Coquillart, S. Hasegawa, L. Bouguila, and M. Sato. The Stringed Haptic Workbench : a New Haptic Workbench Solution. In *Eurographics*, 2003. 152

[TN97] G. Trombettoni and B. Neveu. Computational Complexity of Multiway, Dataflow Constraint Problems. In *Fifteenth International Joint Conference on Artificial Intelligence*, 1997. 20

[Tro97] G. Trombettoni. *Algorithmes de maintien de solution par propagation locale pour les systèmes de contraintes.* PhD thesis, Université de Nice - Sophia Antipolis, 1997. 41

[Ver90] A. Verroust. *Etude de problemes liés à la définition, la visualisation et l'animation d'objets complexes en informatique graphique.* PhD thesis, Thèse d'Etat, Université de Paris-Sud, Orsay, 1990. 35

[vrj] VRJuggler, Open Source Virtual Reality Tools. http://www.vrjuggler.org/. 63

[VSR92] A. Verroust, F. Schonek, and D. Roller. Oriented method for parametrized computer-aided design. *Computer-Aided Design*, 24(10) :531–540, 1992. 17, 35, 36, 43

[Wan02] D. Wang. Geother 1.1 : handling and proving geometric theorems automatically. In *Automated Deduction in Geometry*, 2002. 50

[Wik] Wikipedia. Nombre de bézout, théorème de bézout. http://fr.wikipedia.org/wiki/Théorème_de_Bézout. 106

[Win05] J. Wintz. Compilation de systèmes à base de règles pour la résolution symbolique de contraintes géométriques. Master's thesis, Université Louis Pasteur, Strasbourg, 2005. 126

[Wir90] M. Wirsing. Algebraic specification. *Handbook of theoretical computer science*, pages 677–780, 1990. 12

[WSMF06] J. Wintz, P. Schreck, P. Mathis, and A. Fabre. A framework for geometric constraint satisfaction problem. In *ACM Symposium on Applied Computing*, 2006. 7, 23, 120, 125

[Wu84] W.T. Wu. Basic principles of mechanical theorem proving in elementary geometries. *Journal of Symbolic Computation*, 4 :207–235, 1984. 17, 50

[ZBM94] S. Zhai, W. Buxton, and P. Milgram. The "Silk cursor" : investigating transparency for 3D target acquisition. In *CHI*, 1994. 62

[ZHH96] R.C. Zeleznik, K.P. Herndon, and J.F. Hughes. SKETCH : An Interface for Sketching 3D Scenes. In *ACM Transactions on Graphics, Proceedings of SIGGRAPH'96*, 1996. 69

Annexe A

Univers géométriques 3D

Cette annexe comporte la description en *Gcml* de deux univers géométriques euclidiens 3D. La première section comporte l'univers général décrit au chapitre 2. La seconde section contient l'univers utilisé pour la résolution de problèmes de polyèdres du chapitre 4.

A.1 Univers euclidien 3D

L'univers géométrique suivant est composé de 5 sortes dénotant les points, les droites, les cercles, les plans et les sphères. Une dernière sorte, nommée *measure*, représente la sorte *elem* de l'univers analytique.

```
<universe>
    <syntax>
        <sorts>
            measure
            point
            line
            circle
            plane
            sphere
        </sorts>
```

Les symboles préfixés par *init* servent à initialiser des repères pour les déplacements. Les symboles préfixés par *mk* dénotent la construction d'un objet à partir d'autres objets. Les symboles préfixés par *inter* désignent une intersection d'objets géométriques.

```
        <fsymbols>
            <!-- Point à l'origine -->
            initp_origin : -> point
            <!-- Droite parallèle à l'axe Ox passant par un point -->
            initl_x : point -> line
            <!-- Droite parallèle à l'axe Oy passant par un point -->
```

```
initl_y : point -> line
<!-- Droite parallèle à l'axe 0z passant par un point -->
initl_z : point -> line
<!-- Plan parallèle au plan x0y passant par un point -->
initP_xy : point -> plane
<!-- Plan parallèle au plan y0z passant par un point -->
initP_yz : point -> plane
<!-- Plan parallèle au plan x0z passant par un point -->
initP_xz : point -> plane
<!-- Point à partir de ces 3 coordonnées -->
mkpoint_3m : measure measure measure -> point
<!-- Point sur une droite à une distance donnée d'un
autre point -->
mkpoint_plm : point line measure -> point
<!-- Droite passant par deux points -->
mkline_2p : point point -> line
<!-- Droite faisant un angle donné avec une droite
et passant par un point dans un plan -->
mkline_plPm : point line plane measure -> line
<!-- Droite à une distance donnée d'une droite dans
un plan -->
mkline_lPm : line plane measure -> line
<!-- Droite passant par un point
et faisant deux angles avec deux autres droites -->
mkline_p2l2m : point line line measure measure -> line
<!-- Cercle dans un plan à partir de son centre
et d'un point sur le cercle -->
mkcircle_2pP : point point plane -> circle
<!-- Cercle à partir de son centre
et d'un point définissant son orientation et son rayon -->
mkcircle_2p : point point -> circle
<!-- Cercle dans un plan à partir de son centre et
de son rayon -->
mkcircle_pmP : point measure plane -> circle
<!-- Plan défini par 3 points -->
mkplane_3p : point point point -> plane
<!-- Plan défini par un point sur le plan
et un point définissant sa normale -->
mkplane_2p : point point -> plane
<!-- Plan passant par une droite
et faisant un angle donné avec un autre plan -->
mkplane_lPm : line plane measure -> plane
<!-- Plan passant par un point
et faisant deux angles avec deux autres plans -->
mkplane_p2P2m : point plane plane measure measure -> plane
<!-- Sphère à partir de son centre et d'un point sur
```

```
    la sphère -->
    mksphere_2p : point point -> sphere
    <!-- Sphère à partir de son centre et de son rayon -->
    mksphere_pm : point measure -> sphere
    <!-- Point intersection de deux droites -->
    inter2l : line line -> point
    <!-- Point intersection de deux cercles -->
    inter2c : circle circle -> point
    <!-- Droite intersection de deux plans -->
    inter2P : plane plane -> line
    <!-- Cercle intersection de deux sphères -->
    inter2s1 : sphere sphere -> circle
    <!-- Point intersection de deux sphères-->
    inter2s2 : sphere sphere -> point
    <!-- Point intersection d'une droite et d'un cercle -->
    interlc : line circle -> point
    <!-- Point intersection d'une droite et d'un plan -->
    interlP : line plane -> point
    <!-- Point intersection d'une droite et d'une sphère -->
    interls : line sphere -> point
    <!-- Point intersection d'un cercle et d'un plan -->
    intercP : circle plane -> point
    <!-- Point intersection d'un cercle et d'une sphère -->
    intercs : circle sphere -> point
    <!-- Point intersection d'un plan et d'une sphère -->
    interPs1 : plane sphere -> point
    <!-- Cercle intersection d'un plan et d'une sphère -->
    interPs2 : plane sphere -> circle
</fsymbols>
```

Les contraintes considérées dans cet univers sont des contraintes d'incidence, préfixées par *on*, des contraintes d'angles, préfixées par *angle*, et des contraintes de distance préfixées par *dist*.

```
<psymbols>
    <!-- Coplanarité de quatre points -->
    cop : point point point point
    <!-- Incidence d'un point sur une droite -->
    onl : point line
    <!-- Incidence d'un point sur un plan -->
    onP : point plane
    <!-- Incidence d'un point sur un cercle -->
    onc : point circle
    <!-- Incidence d'un point sur une sphère -->
    ons : point sphere
    <!-- Incidence d'une droite sur un plan -->
    onlP : line plane
```

```
        <!-- Incidence d'un cercle sur une sphère -->
        oncs : circle sphere
        <!-- Non incidence d'un cercle sur un plan -->
        notoncP : circle plane
        <!-- Angle entre deux plans -->
        angle2P : plane plane measure
        <!-- Angle entre deux droites -->
        angle2l : line line measure
        <!-- Angle entre deux bi-points partageant un point -->
        angle3p : point point point measure
        <!-- Distance entre deux points -->
        distpp : point point measure
        <!-- Distance entre un point et un plan -->
        distpP : point plane measure
        <!-- Distance entre un point et une droite -->
        distpl : point line measure
        <!-- Distance entre un point et une sphère -->
        distps : point sphere measure
        <!-- Distance entre deux plans -->
        distPP : plane plane measure
        <!-- Distance entre un plan et une droite -->
        distPl : plane line measure
        <!-- Distance entre un plan et une sphère -->
        distPs : plane sphere measure
        <!-- Distance entre deux droites -->
        distll : line line measure
        <!-- Distance entre une droite et une sphère -->
        distls : line sphere measure
        <!-- Distance entre deux sphères -->
        distss : sphere sphere measure
    </psymbols>
```

Les axiomes se décomposent en propriétés, *properties*, et en règles, *rules*. Les propriétés considérées dans cet univers sont des propriétés de commutativité. Les règles sont divisées en règles de construction, *construction*, et en règles pour former des repères, *location*.
Ces règles de formation de repères sont illustrées dans l'annexe suivante.

```
    <axioms>
        <properties>
            <property>
                distpp(p1,p2,k) = distpp(p2,p1,k)
            </property>
            <property>
                angle3p(p1,p2,p3,k) = angle3p(p3,p1,p2,k)
            </property>
            <property>
```

```
            angle2l(l1,l2,alpha) = angle2l(l2,l1,alpha)
        </property>
        <property>
            angle2P(p1,p2,alpha) = angle2P(p2,p1,alpha)
        </property>
        <property>
            cop(p1,p2,p3,p4) = cop(p4,p1,p2,p3) =
            cop(p3,p4,p1,p2) = cop(p2,p3,p4,p1) =
            cop(p1,p2,p4,p3) = cop(p4,p3,p1,p2) =
            cop(p3,p1,p2,p4) = cop(p2,p4,p3,p1) =
            cop(p1,p4,p2,p3) = cop(p4,p2,p3,p1) =
            cop(p3,p1,p4,p2) = cop(p2,p3,p1,p4) =
            cop(p1,p4,p3,p2) = cop(p4,p3,p2,p1) =
            cop(p3,p2,p1,p4) = cop(p2,p1,p4,p3) =
            cop(p1,p3,p2,p4) = cop(p4,p1,p3,p2) =
            cop(p3,p2,p4,p1) = cop(p2,p4,p1,p3) =
            cop(p1,p3,p4,p2) = cop(p4,p2,p1,p3) =
            cop(p3,p4,p2,p1) = cop(p2,p1,p3,p4)
        </property>
    </properties>
    <rules>
        <construction name="On a line and on a sphere">
            <if>
                onl(p,l)
                ons(p,s)
            </if>
            <then>
                p=interls(l,s)
            </then>
        </construction>
        <construction name="Line by two points">
            <if>
                onl(x,l)
                onl(y,l)
            </if>
            <then>
                l=mkline_2p(x,y)
            </then>
        </construction>
        <construction name="Incidence relation">
            <if>
                onl(x,l)
                onlP(l,pl)
            </if>
            <then>
                onP(x,pl)
```

```
    </then>
</construction>
<construction name="Point on a sphere">
    <if>
        distpp(x,y,k)
    </if>
    <then>
        s=mksphere_pm(x,k)
        ons(y,s)
    </then>
</construction>
<construction name="Intersection of a sphere
and a plane">
    <if>
        ons(x,s)
        onP(x,p)
    </if>
    <then>
        c=interPs(p,s)
        onc(x,c)
    </then>
</construction>
<construction name="Intersection of 2 spheres">
    <if>
        ons(x,s1)
        ons(x,s2)
    </if>
    <then>
        c=inter2s(s1,s2)
        onc(x,c)
    </then>
</construction>
<construction name="On a sphere and on a circle">
    <if>
        onc(p,c)
        ons(p,s)
    </if>
    <then>
        p=intercs(c,s)
    </then>
</construction>
<construction name="Intersection of two circles">
    <if>
        onc(x,c1)
        onc(x,c2)
    </if>
```

```
    <then>
        x=inter2c(c1,c2)
    </then>
</construction>
<construction name="Plane by three points">
    <if>
        onP(p1,pl)
        onP(p2,pl)
        onP(p3,pl)
    </if>
    <then>
        pl=mkplane_3p(p1,p2,p3)
    </then>
</construction>
<construction name="Line by angles with two lines">
    <if>
        onl(p,l1)
        onl(p,l2)
        onl(p,l3)
        angle2l(l2,l1,a1)
        angle2l(l3,l1,a2)
    </if>
    <then>
        l1=mkline_p2l2m(p,l2,l3,a1,a2)
    </then>
</construction>
<construction name="In a plane">
    <if>
        cop(p1,p2,p3,p4)
        onc(p1,c)
    </if>
    <then>
        plane=mkplane_3p(p2,p3,p4)
        p1=intercP(c,plane)
    </then>
</construction>
<location name="Angle between two lines">
    <if>
        angle2l(l1,l2,alpha)
        onl(p1,l1)
        onl(p1,l2)
    </if>
    <then>
        p1=initp_origin()
        l1=initl_x(p1)
        P=initP_xy(p1)
```

```
        l2=mkline_plPm(p1,l1,P,alpha)
    </then>
</location>
<location name="Distance point/line">
    <if>
        distpl(p,l,k)
    </if>
    <then>
        origin=initp_origin()
        l=initl_x(origin)
        l1=initl_z(origin)
        p=mkpoint_plm(origin,l1,k)
    </then>
</location>
<location name="Angle between 2 planes and distances
to a point">
    <if>
        angle2P(pl1,pl2,alpha)
        distpP(p,pl1,k1)
        distpP(p,pl2,k2)
    </if>
    <then>
        origin=initp_origin()
        pl1=initP_xy(origin)
        lx=initl_x(origin)
        pl2=mkplane_lPm(lx,pl1,alpha)
        ly=initl_y(origin)
        plyz=initP_yz(origin)
        l=mkline_lPm(ly,plyz,k1)
        l2=inter2P(pl2,plyz)
        l3=mkline_lPm(l2,plyz,k2)
        p=inter2l(l,l3)
    </then>
</location>
<location name="Angles between 3 planes">
    <if>
        angle2P(pl1,pl2,a1)
        angle2P(pl2,pl3,a2)
        angle2P(pl1,pl3,a3)
    </if>
    <then>
        origin=initp_origin()
        pl1=initP_xy(origin)
        lx=initl_x(origin)
        pl2=mkplane_lPm(lx,pl1,a1)
        pl3=mkplane_p2P2m(origin,pl1,pl2,a2,a3)
```

```
    </then>
</location>
<location name="3 points and 3 distances">
    <if>
        distpp(p1,p2,k1)
        distpp(p2,p3,k2)
        distpp(p1,p3,k3)
    </if>
    <then>
        p1=initp_origin()
        lx=initl_x(p1)
        p2=mkpoint_plm(p1,lx,k1)
        pl=initP_xy(p1)
        c1=mkcircle_pmP(p1,k3,pl)
        c2=mkcircle_pmP(p2,k2,pl)
        p3=inter2c(c1,c2)
    </then>
</location>
<location name="2 points and 1 distance in a plane">
    <if>
        distpp(p1,p2,k)
        onP(p1,pl)
        onP(p2,pl)
    </if>
    <then>
        p1=initp_origin()
        lx=initl_x(p1)
        p2=mkpoint_plm(p1,lx,k1)
        pl=initP_xy(p1)
    </then>
</location>
<location name="2 points a plane and 3 distances">
    <if>
        distpp(p1,p2,k1)
        distpP(p1,pl,k2)
        distpP(p2,pl,k3)
    </if>
    <then>
        origin=initp_origin()
        pl=initP_xy(origin)
        lz=initl_z(origin)
        p1=mkpoint_plm(origin,lz,k1)
        lx=initl_x(origin)
        plxz=initP_xz(origin)
        l=mkline_lPm(lx,plxz,k2)
        c=mkcircle_pmP(p2,k3,plxz)
```

```
                    p2=interlc(l,c)
                </then>
            </location>
            <location name="A point on a line on a plane">
                <if>
                    onl(p,l)
                    onlP(l,pl)
                </if>
                <then>
                    p=initp_origin()
                    l=initl_x(p)
                    pl=initP_xy(p)
                </then>
            </location>
        </rules>
    </axioms>
</syntax>
</universe>
```

A.2 Univers 3D atomique

L'univers décrit ici est un univers 3D restreint aux sortes *point*, *droite* et *plan*. Dans cet univers, la sorte `measure` est distinguée de la sorte `scalar` pour souligner le morphisme venant de l'univers analytique. Une mesure ne sert que pour paramétriser une contrainte dimensionnelle alors qu'un scalaire est utilisé pour associé des coordonnées aux objets.

```
<gcml>
<universe>
    <syntax>
        <sorts>
            scalar
            measure
            point
            line
            plane
        </sorts>

        <fsymbols>
            mkMeasure : scalar -> measure
            mkPoint : scalar scalar scalar -> point
            mkLine : point point -> line
            mkPlane : scalar scalar scalar scalar -> plane
            initPoint: -> point
            initPlane : -> plane
            initLine_2p : point point -> line
```

```
    initLine_x : point -> line
    initLine_y : point -> line
    initLine_z : point -> line
    initPlane_xy : point -> plane
    initPlane_yz : point -> plane
    initPlane_xz : point -> plane
    mkPlane_3p : point point point -> plane
    mkPoint_3p3m : point point point measure measure measure
    -> point
    mkPoint_3P : plane plane plane -> point
    mkPoint_plm : point line measure -> point
    mkPoint_2pP2m : point point plane measure measure
    -> point
    mkPoint_p2Pm : point plane plane measure -> point
    mkPlane_2l : line line -> plane
</fsymbols>

<psymbols>
    onl : point line
    onP : point plane
    norm : plane
    distpp : point point measure
</psymbols>

<axioms>
    <properties>
        <property>
            distpp(p1,p2,k) = distpp(p2,p1,k)
        </property>
    </properties>

    <rules>
        <construction name="Point by 3 planes">
            <if>
                onP(p,pl1)
                onP(p,pl2)
                onP(p,pl3)
            </if>
            <then>
                p=mkPoint_3P(pl1,pl2,pl3)
            </then>
        </construction>
        <construction name="Point by 3 distances">
            <if>
                distpp(p1,p2,k1)
                distpp(p1,p3,k2)
```

```
        distpp(p1,p4,k3)
    </if>
    <then>
        p1=mkPoint_3p3m(p2,p3,p4,k1,k2,k3)
    </then>
</construction>
<construction name="Plane by 3 points">
    <if>
        onP(p1,p)
        onP(p2,p)
        onP(p3,p)
        norm(p)
    </if>
    <then>
        p=mkPlane_3p(p1,p2,p3)
    </then>
</construction>
<construction name="Point in a plane by 2 distances">
    <if>
        onP(p1,pl)
        distpp(p1,p2,k1)
        distpp(p1,p3,k2)
    </if>
    <then>
        p1=mkPoint_2pP2m(p2,p3,pl,k1,k2)
    </then>
</construction>
<construction name="Point by 2 planes and 1 distance">
    <if>
        onP(p1,pl1)
        onP(p1,pl2)
        onP(p2,pl1)
        onP(p2,pl2)
        distpp(p1,p2,k1)
    </if>
    <then>
        p1=mkPoint_p2Pm(p2,pl1,pl2,k1)
    </then>
</construction>
<location name="3 points and 3 distances">
    <if>
        distpp(p1,p2,k1)
        distpp(p2,p3,k2)
        distpp(p1,p3,k3)
    </if>
    <then>
```

```
                    p1=initPoint()
                    lx=initLine_2p(p1,p2)
                    px=mkPlane_3p(p1,p2,p3)
                    p2=mkPoint_plm(p1,lx,k1)
                    p3=mkPoint_2pP2m(p1,p2,px,k3,k2)
                </then>
            </location>
            <location name="3 points 1 plane">
                <if>
                    onP(p1,pl)
                    onP(p2,pl)
                    distpp(p1,p2,k1)
                    norm(pl)
                </if>
                <then>
                    p1=initPoint()
                    lx=initLine_2p(p1,p2)
                    pl=initPlane()
                    p2=mkPoint_plm(p1,lx,k1)
                </then>
            </location>
        </rules>
    </axioms>
</syntax>
</universe>
</gcml>
```

Annexe B

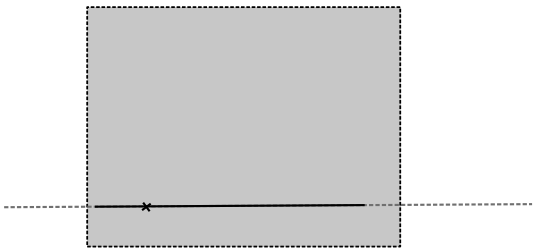
Sémantique des repères 3D

Un repère 3D est un système de contraintes bien-contraint *modulo* les déplacements, qui a six degrés de liberté (trois en translations et trois en rotations). Cette annexe présente les repères 3D que nous avons retenus.

B.1 Contraintes d'incidence

Le repère que l'on peut extraire de contraintes d'incidence est celui basé sur un point incident à une droite dans un plan.

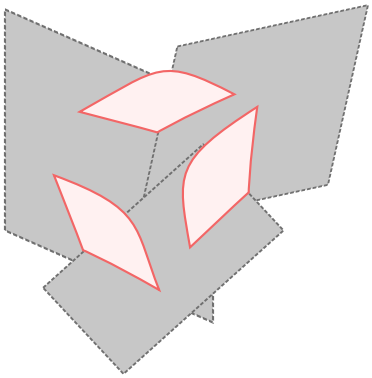
```
point p
line l
plane pl
onl(p,l)
onlp(l,pl)
```



B.2 Contraintes dimensionnelles

Le repère suivant est extrait de 3 plans contraints en angles deux à deux.

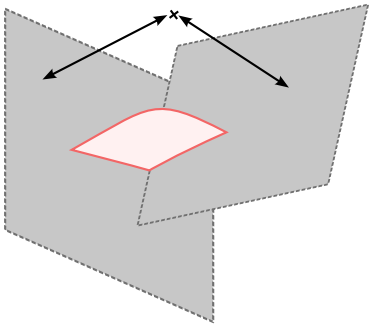
```
plane pl1,pl2,pl3
measure a1,a2,a3
angle(pl1,pl2,a1)
angle(pl2,pl3,a2)
angle(pl1,pl3,a3)
```



Un autre repère utilisant des contraintes dimensionnelles est celui déduit à partir d'un point, de deux plans, d'une contrainte d'angle entre ces deux plans et deux

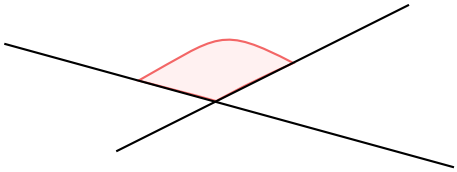
distances point-plan.

```
point p
plane pl1,pl2
angle(pl1,pl2)
distppl(p,pl1)
distppl(p,pl2)
```



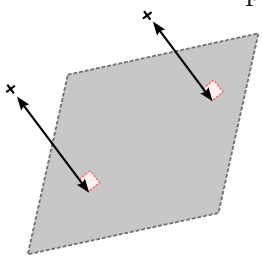
Un troisième repère est constitué d'un angle entre deux droites.

```
line l1,l2
measure alpha
angle(l1,l2,alpha)
```



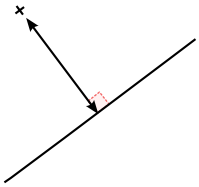
Le repère suivant se base sur deux points, un plan et deux distances point-plan.

```
point p1,p2
plane pl
distppl(p1,pl)
distppl(p2,pl)
```



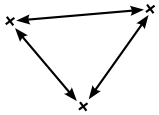
Une distance point-droite peut fournir un repère :

```
point p
line l
measure k
distpl(p,l,k)
```



Enfin, trois points contraints par la donnée de trois contraintes de distances donnent le repère :

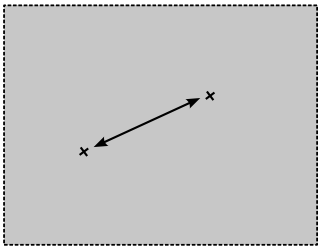
```
point p1,p2,p3
measure k1,k2,k3
distpp(p1,p2,k1)
distpp(p2,p3,k2)
distpp(p1,p3,k3)
```



B.3 Contraintes d'incidence et dimensionnelles

Deux points contraints par une distance dans un plan donnent un repère.

```
point p1,p2
plane pl
measure k
distpp(p1,p2,k)
onp(p1,pl)
onp(p2,pl)
```



Annexe C

Exemples

Dans les exemples suivants, la partie sémantique a été supprimée, prenant trop de place par rapport à son importance réelle. Nous l'avons juste gardée pour le camembert et le tétraèdre à titre indicatif.

C.1 Minimaux non triviaux

C.1.1 Pyramide

```
<gcml>

  <universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
  </universe>

  <gcs>
   <syntax>
    <unknowns>
        point p1 p2 p3 p4 p5
        plane pl1
    </unknowns>
    <parameters>
        measure k1 k2 k3 k4 k5 k6 k7 k8 k9
    </parameters>
    <constraints>
        distpp(p5,p1,k1)
        distpp(p5,p2,k2)
        distpp(p5,p3,k3)
        distpp(p5,p4,k4)
        distpp(p4,p3,k5)
        distpp(p3,p2,k6)
        distpp(p2,p1,k7)
        distpp(p1,p4,k8)
```

```
        distpp(p1,p3,k9)
        onP(p3, pl1)
        onP(p2, pl1)
        onP(p1, pl1)
        onP(p4, pl1)
    </constraints>
   </syntax>
  </gcs>
</gcml>
```

C.1.2 Le camembert

```
<gcml>
<universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
</universe>
<gcs>
    <syntax>
        <unknowns>
            point p1 p2 p3 p4 p5 p6
            plane pl1 pl2 pl3
        </unknowns>
        <parameters>
            measure k1 k2 k3 k4 k5 k6 k7 k8 k9
        </parameters>
        <constraints>
            distpp(p2,p5,k1)
            distpp(p5,p1,k2)
            distpp(p1,p2,k3)
            distpp(p3,p4,k4)
            distpp(p4,p6,k5)
            distpp(p3,p6,k6)
            distpp(p1,p3,k7)
            distpp(p2,p4,k8)
            distpp(p5,p6,k9)
            onP(p1, pl1)
            onP(p2, pl1)
            onP(p4, pl1)
            onP(p3, pl1)
            onP(p2, pl2)
            onP(p5, pl2)
            onP(p6, pl2)
            onP(p4, pl2)
            onP(p1, pl3)
            onP(p5, pl3)
```

```
            onP(p6, pl3)
            onP(p3, pl3)
            </constraints>
        </syntax>
        <semantic>
            <valuation>
            </valuation>
        <sketch>
            p3=mkPoint(0,0,0)
            p4=mkPoint(1,0,0)
            p6=mkPoint(0.265953,0.739256,0)
            p1=mkPoint(0,0,1)
            p2=mkPoint(0.730375,-0.0217405,0.983264)
            p5=mkPoint(0.296812,0.784927,0.835793)
            pl1=mkPlane(-0.506215,-0.0969415,-0.856941,-0.364424)
            pl2=mkPlane(0.581612,0.163423,-0.796882,-0.270511)
            pl3=mkPlane(0.268996,0.2211,-0.93742,-0.364424)
            k1=mkMeasure(0.927598)
            k2=mkMeasure(0.862229)
            k3=mkMeasure(0.695355)
            k4=mkMeasure(0.905195)
            k5=mkMeasure(0.976332)
            k6=mkMeasure(0.78259)
            k7=mkMeasure(1.04898)
            k8=mkMeasure(1.00364)
            k9=mkMeasure(0.853576)
            </sketch>
        </semantic>
</gcs>
</gcml>
```

C.1.3 La pyramide à base pentagonale

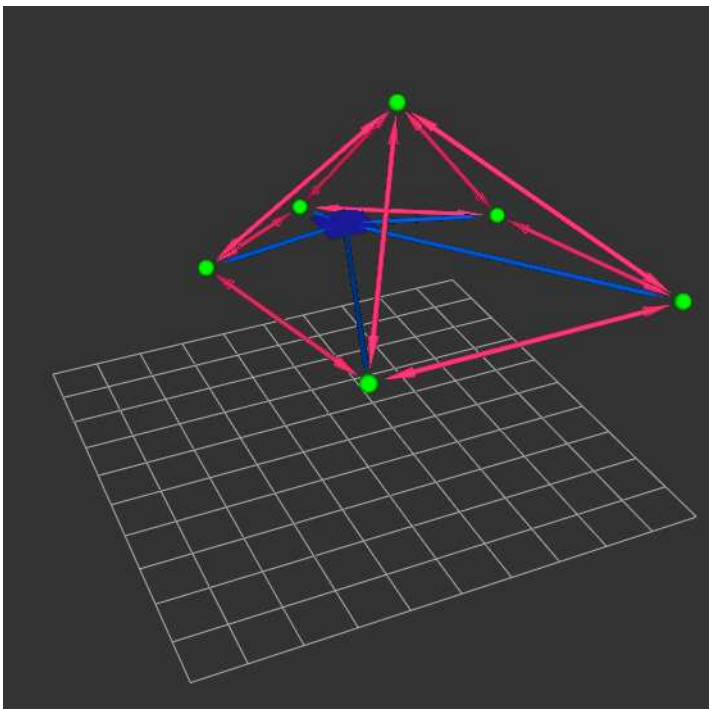


FIG. C.1 – Pyramide à base pentagonale

```
<gcml>
  <universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
  </universe>

  <gcs>
    <unknowns>
        point p1 p2 p3 p4 p5 p6
        plane pl1
    </unknowns>
    <parameters>
        measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10
    </parameters>
    <constraints>
        distpp(p1,p2,k1)
        distpp(p1,p3,k2)
        distpp(p1,p4,k3)
        distpp(p1,p5,k4)
        distpp(p1,p6,k5)
        distpp(p6,p2,k6)
        distpp(p2,p3,k7)
        distpp(p3,p4,k8)
```

```
        distpp(p4,p5,k9)
        distpp(p5,p6,k10)
        onP(p6, pl1)
        onP(p2, pl1)
        onP(p3, pl1)
        onP(p4, pl1)
        onP(p5, pl1)
    </constraints>
  </gcs>
</gcml>
```

C.1.4 La pyramide à base hexagonale

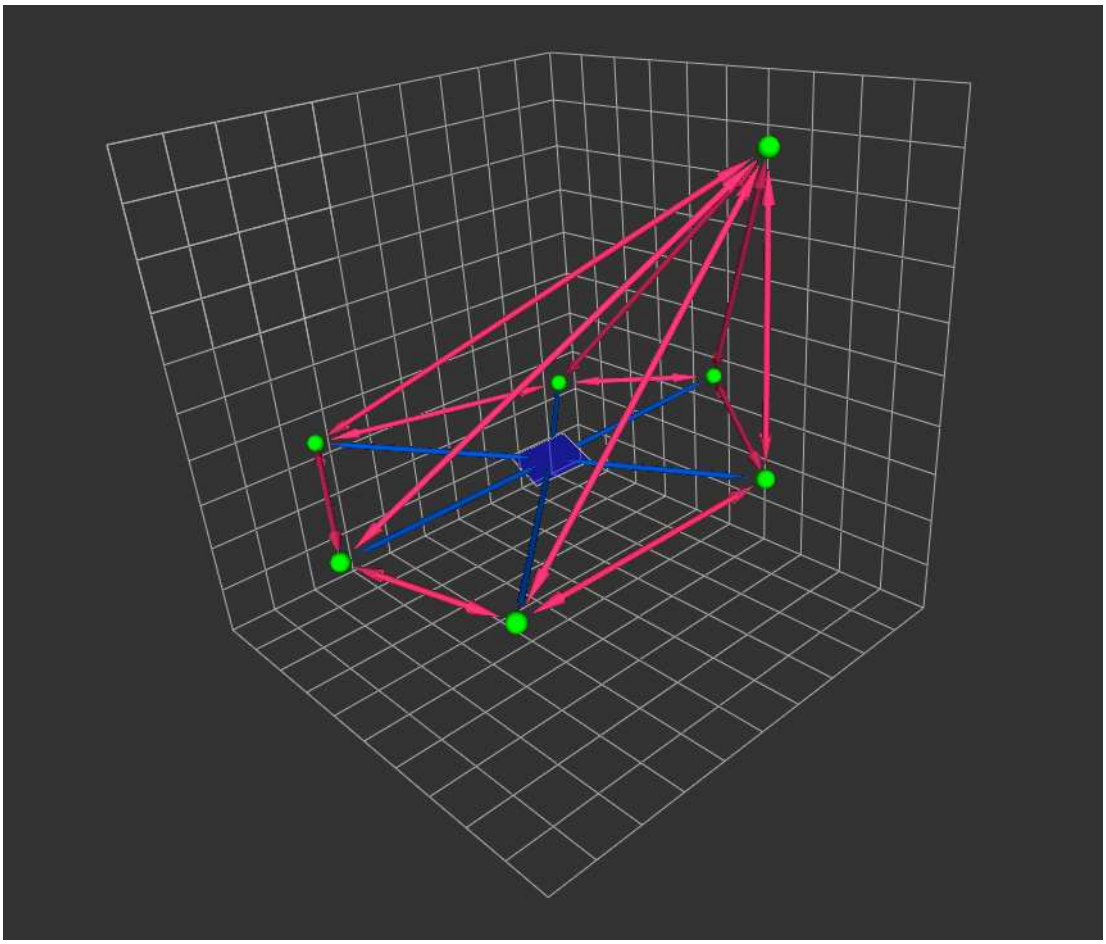


FIG. C.2 – Pyramide à base hexagonale

```
<gcml>
 <universe>
  <syntax ref="atomic3d_syntax.gcml" />
  <semantic ref="atomic3d_semantic.gcml" />
 </universe>
```

```
 <gcs>
  <syntax>
   <unknowns>
point p1 p2 p3 p4 p5 p6 p7
plane pl
</unknowns>
   <parameters>
measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12
</parameters>
   <constraints>
distpp(p1,p2,k1)
distpp(p1,p3,k2)
distpp(p1,p4,k3)
distpp(p1,p5,k4)
distpp(p1,p6,k5)
distpp(p1,p7,k6)
distpp(p2,p3,k7)
distpp(p3,p4,k8)
distpp(p4,p5,k9)
distpp(p5,p6,k10)
distpp(p6,p7,k11)
distpp(p7,p2,k12)
onP(p2,pl)
onP(p3,pl)
onP(p4,pl)
onP(p5,pl)
onP(p6,pl)
onP(p7,pl)
norm(pl)
</constraints>
  </syntax>
</gcs>
</gcml>
```

C.1.5 L'étoile de mer

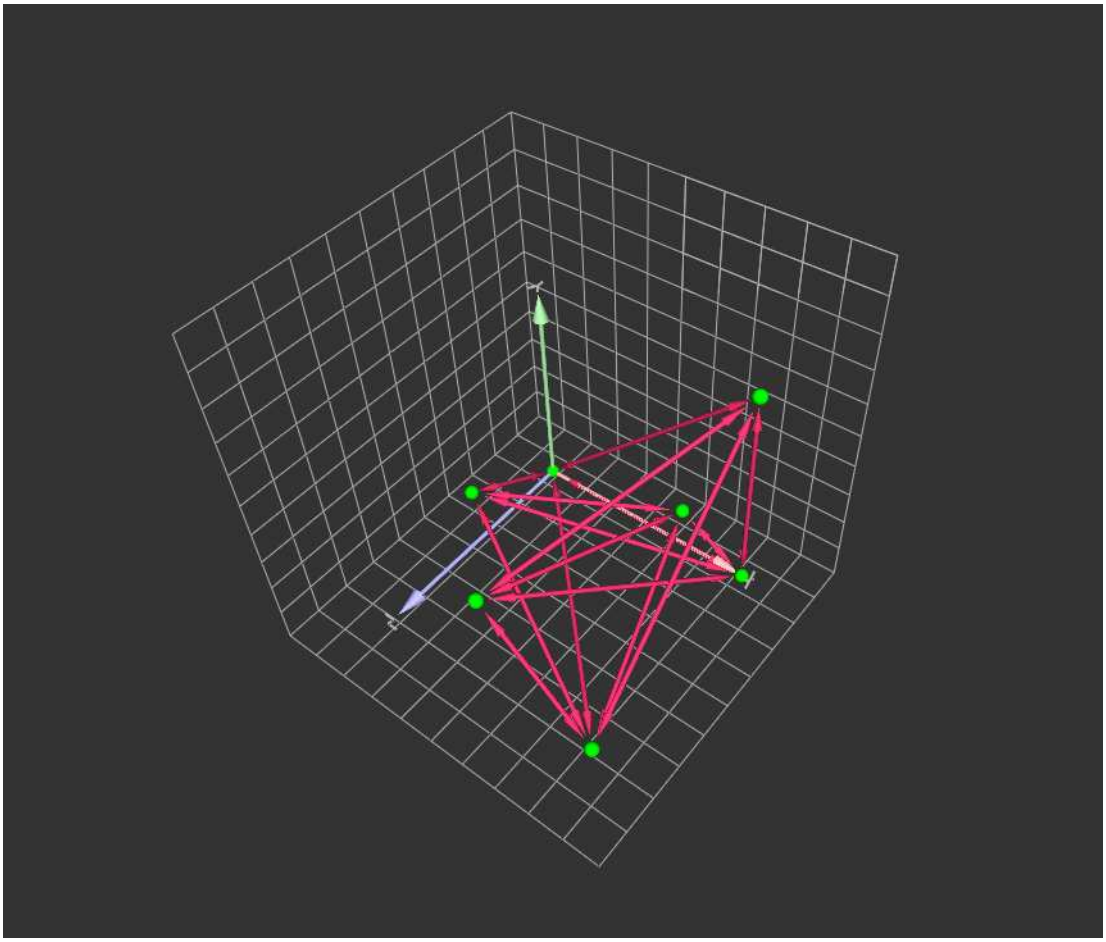


FIG. C.3 – Étoile de mer

```
<gcml>
  <universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
  </universe>

  <gcs>
    <unknowns>
        point p1 p2 p3 p4 p5 p6 p7
    </unknowns>
    <parameters>
        measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 k13 k14
    </parameters>
    <constraints>
        distpp(p4,p1,k1)
        distpp(p2,p1,k2)
        distpp(p1,p3,k3)
```

```
        distpp(p1,p5,k4)
        distpp(p1,p6,k5)
        distpp(p7,p2,k6)
        distpp(p7,p3,k7)
        distpp(p7,p4,k8)
        distpp(p7,p5,k9)
        distpp(p7,p6,k10)
        distpp(p6,p5,k11)
        distpp(p5,p4,k12)
        distpp(p4,p3,k13)
        distpp(p3,p2,k14)
    </constraints>
  </gcs>
</gcml>
```

C.1.6 Antiprisme d'ordre 4

```
<gcml>
 <universe>
  <syntax ref="atomic3d_syntax.gcml" />
  <semantic ref="atomic3d_semantic.gcml" />
 </universe>
 <gcs>
  <syntax>
   <unknowns>
   point p1 p2 p3 p4 p5 p6 p7 p8
   plane pl1 pl2
</unknowns>
   <parameters>
   measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 k13 k14 k15 k16
</parameters>
   <constraints>
    distpp(p1,p2,k1)
    distpp(p2,p3,k2)
    distpp(p3,p4,k3)
    distpp(p4,p5,k4)
    distpp(p1,p5,k5)
    distpp(p1,p6,k6)
    distpp(p2,p6,k7)
    distpp(p3,p6,k8)
    distpp(p4,p6,k9)
    distpp(p1,p7,k10)
    distpp(p2,p7,k11)
    distpp(p5,p7,k12)
    distpp(p8,p7,k13)
    distpp(p3,p8,k14)
```

```
    distpp(p8,p5,k15)
    distpp(p8,p4,k16)
    onP(p2,pl1)
    onP(p3,pl1)
    onP(p7,pl1)
    onP(p8,pl1)
    norm(pl1)
    onP(p1,pl2)
    onP(p4,pl2)
    onP(p5,pl2)
    onP(p6,pl2)
    norm(pl2)
</constraints>
</syntax>
<gcs>
<gcml>
```

C.2 Solides Pseudo-Platoniciens

C.2.1 Tétraèdre

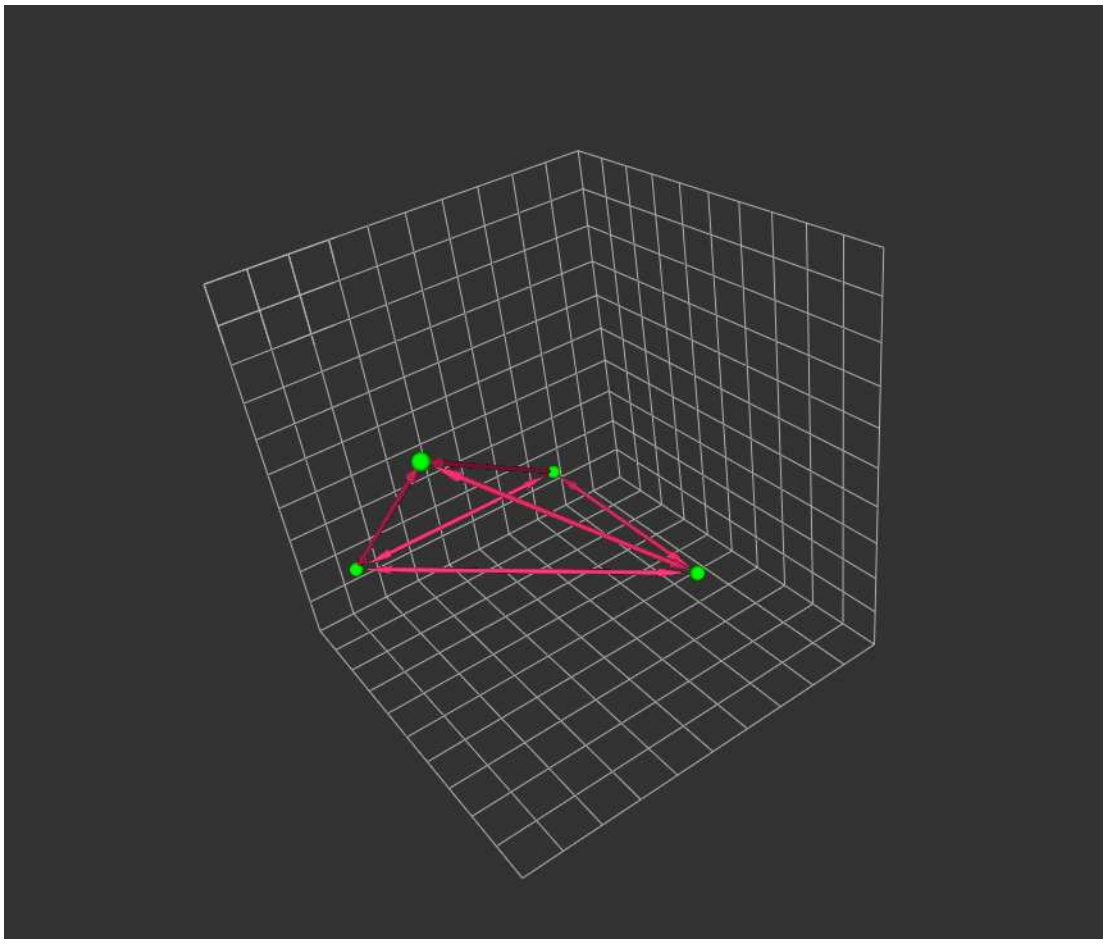


Fig. C.4 – Tétraèdre

```
<gcml>
<universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantics ref="atomic3d_semantic.gcml"/>
</universe>

<gcs>
    <syntax>
        <unknowns>
            point p1 p2 p3 p4
            line l1 l2 l3 l4 l5 l6
        </unknowns>

        <parameters>
            measure k1 k2 k3 k4 k5 k6
```

```
        </parameters>

        <constraints>
            distpp(p1,p2,k1)
            distpp(p2,p3,k2)
            distpp(p1,p3,k3)
            distpp(p1,p4,k4)
            distpp(p2,p4,k5)
            distpp(p3,p4,k6)
            onl(p1,l1) onl(p2,l1)
            onl(p2,l2) onl(p3,l2)
            onl(p3,l3) onl(p1,l3)
            onl(p1,l4) onl(p4,l4)
            onl(p2,l5) onl(p4,l5)
            onl(p3,l6) onl(p4,l6)
        </constraints>
    </syntax>
    <semantic>
        <valuation>
            p1=mkPoint(1,0.3)
        </valuation>
        <sketch>
            p1=mkPoint(0,0,3)
            p2=mkPoint(1.67,1.5,2.22)
            p3=mkPoint(1,3,4)
            p4=mkPoint(0,2,1)
            l1=mkLine(p1,p2)
            l2=mkLine(p2,p3)
            l3=mkLine(p3,p1)
            l4=mkLine(p1,p4)
            l5=mkLine(p2,p4)
            l6=mkLine(p3,p4)
            k1=mkMeasure(1)
            k2=mkMeasure(1)
            k3=mkMeasure(1)
            k4=mkMeasure(1)
            k5=mkMeasure(1)
            k6=mkMeasure(1)
        </sketch>
    </semantic>
</gcs>
</gcml>
```

C.2.2 Octaèdre

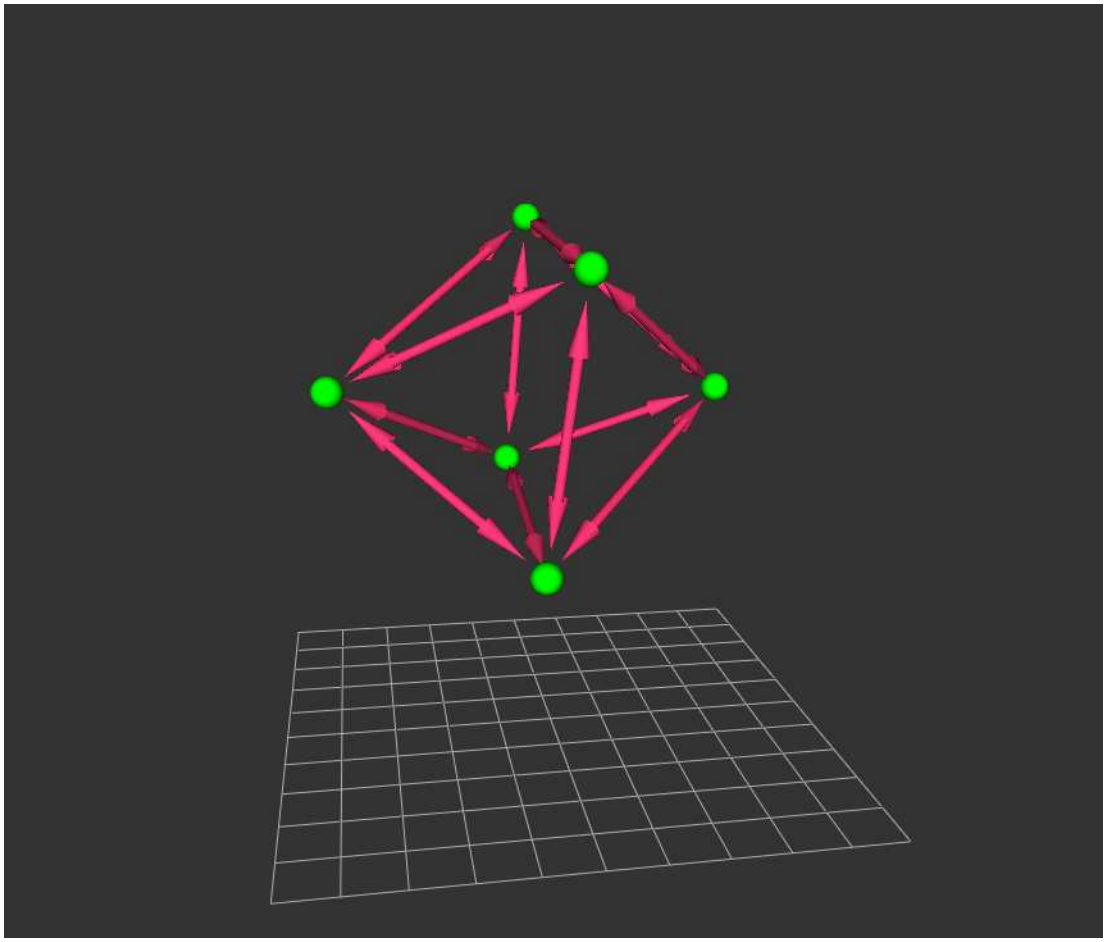


FIG. C.5 – Octaèdre

```
<gcml>
<universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
</universe>

<gcs>
    <syntax>
        <unknowns>
            point p1 p2 p3 p4 p5 p6
            line l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12
        </unknowns>

        <parameters>
            measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12
        </parameters>
```

```
        <constraints>
            distpp(p1,p2,k1)
            distpp(p2,p3,k2)
            distpp(p3,p4,k3)
            distpp(p4,p1,k4)
            distpp(p1,p5,k5)
            distpp(p2,p5,k6)
            distpp(p3,p5,k7)
            distpp(p4,p5,k8)
            distpp(p1,p6,k9)
            distpp(p2,p6,k10)
            distpp(p3,p6,k11)
            distpp(p4,p6,k12)

            onl(p1,l1) onl(p2,l1)
            onl(p2,l2) onl(p3,l2)
            onl(p3,l3) onl(p4,l3)
            onl(p1,l4) onl(p4,l4)
            onl(p1,l5) onl(p5,l5)
            onl(p2,l6) onl(p5,l6)
            onl(p3,l7) onl(p5,l7)
            onl(p4,l8) onl(p5,l8)
            onl(p1,l9) onl(p6,l9)
            onl(p2,l10) onl(p6,l10)
            onl(p3,l11) onl(p6,l11)
            onl(p4,l12) onl(p6,l12)
        </constraints>
    </syntax>
</gcs>
</gcml>
```

C.2.3 Hexaèdre

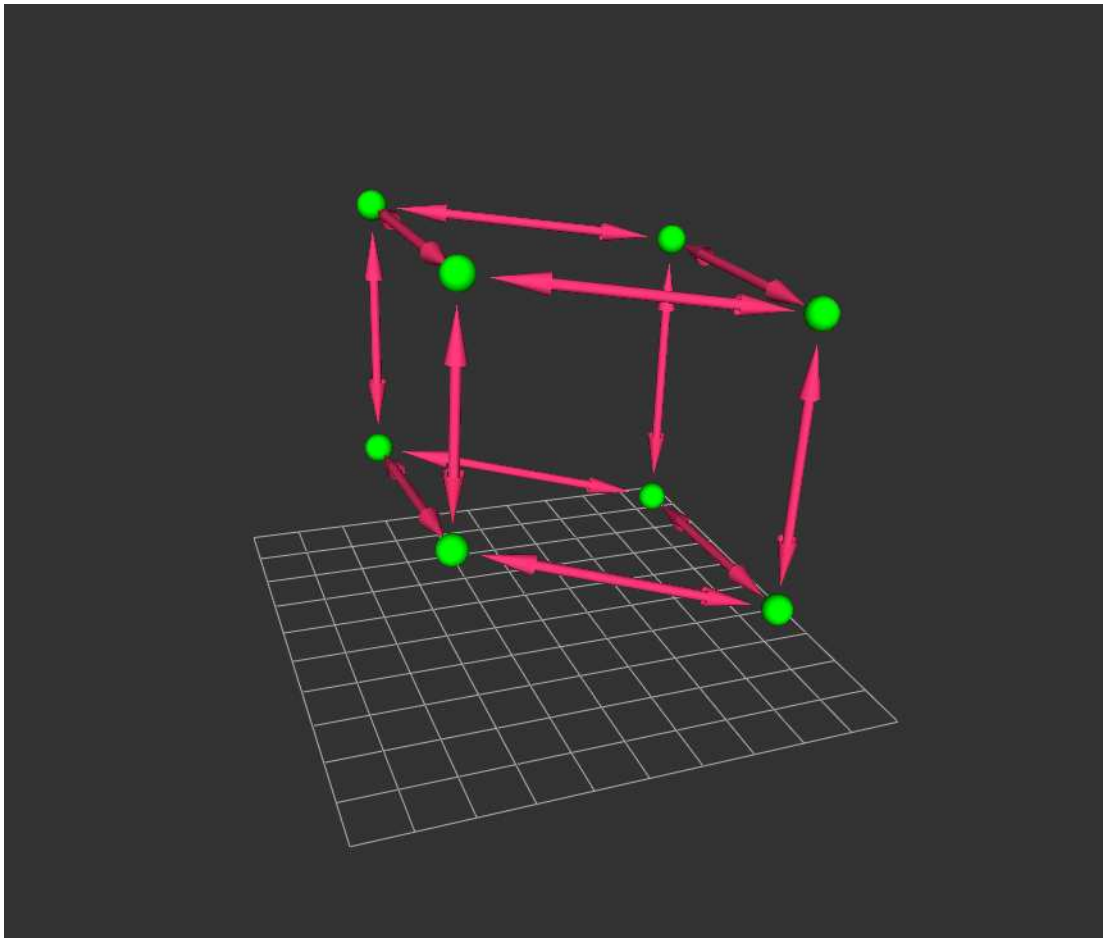


FIG. C.6 – Héxaèdre

```
<gcml>
<universe>
    <syntax ref="atomic3d_syntax.gcml" />
    <semantic ref="atomic3d_semantic.gcml" />
</universe>
<gcs>
<syntax>
    <unknowns>
        point p1 p2 p3 p4 p5 p6 p7 p8
        plane pl1 pl2 pl3 pl4 pl5 pl6
    </unknowns>
    <parameters>
        measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12
    </parameters>
    <constraints>
        distpp(p1,p2,k1)
        distpp(p2,p3,k2)
```

```
        distpp(p3,p4,k3)
        distpp(p4,p1,k4)
        distpp(p5,p6,k5)
        distpp(p6,p7,k6)
        distpp(p7,p8,k7)
        distpp(p8,p5,k8)
        distpp(p1,p5,k9)
        distpp(p2,p6,k10)
        distpp(p3,p7,k11)
        distpp(p4,p8,k12)
        onP(p1,pl1) onP(p2,pl1) onP(p3,pl1) onP(p4,pl1)
        onP(p1,pl2) onP(p2,pl2) onP(p5,pl2) onP(p6,pl2)
        onP(p5,pl3) onP(p6,pl3) onP(p7,pl3) onP(p8,pl3)
        onP(p7,pl4) onP(p8,pl4) onP(p4,pl4) onP(p3,pl4)
        onP(p1,pl5) onP(p5,pl5) onP(p8,pl5) onP(p4,pl5)
        onP(p2,pl6) onP(p6,pl6) onP(p7,pl6) onP(p3,pl6)
        norm(pl1)
        norm(pl2)
        norm(pl3)
        norm(pl4)
        norm(pl5)
        norm(pl6)
    </constraints>
</syntax>
</gcs>
</gcml>
```

C.2.4 Icosaèdre

```
<gcml>
<universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
</universe>

<gcs>
    <syntax>
        <unknowns>
            point p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12
        </unknowns>

        <parameters>
            measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12
                    k13 k14 k15 k16 k17 k18 k19 k20 k21 k22
                    k23 k24 k25 k26 k27 k28 k29 k30
        </parameters>
```

```
        <constraints>
            distpp(p1,p2,k1)
            distpp(p1,p3,k2)
            distpp(p1,p4,k3)
            distpp(p1,p5,k4)
            distpp(p1,p6,k5)
            distpp(p2,p3,k6)
            distpp(p3,p4,k7)
            distpp(p4,p5,k8)
            distpp(p5,p6,k9)
            distpp(p2,p6,k10)
            distpp(p7,p8,k11)
            distpp(p7,p9,k12)
            distpp(p7,p10,k13)
            distpp(p7,p11,k14)
            distpp(p7,p12,k15)
            distpp(p8,p9,k16)
            distpp(p9,p10,k17)
            distpp(p10,p11,k18)
            distpp(p11,p12,k19)
            distpp(p8,p12,k20)
            distpp(p2,p8,k21)
            distpp(p8,p3,k22)
            distpp(p3,p12,k23)
            distpp(p12,p4,k24)
            distpp(p4,p11,k25)
            distpp(p11,p5,k26)
            distpp(p5,p10,k27)
            distpp(p10,p6,k28)
            distpp(p6,p9,k29)
            distpp(p9,p2,k30)
        </constraints>
    </syntax>
</gcs>
</gcml>
```

C.2.5 Dodécaèdre

```
 <gcml>
<universe>
    <syntax ref="atomic3d_syntax.gcml"/>
    <semantic ref="atomic3d_semantic.gcml"/>
</universe>
<gcs>
    <syntax>
```

```
<unknowns>
    point p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12
          p13 p14 p15 p16 p17 p18 p19 p20
    plane pl1 pl2 pl3 pl4 pl5 pl6 pl7 pl8
          pl9 pl10 pl11 pl12
</unknowns>

<parameters>
    measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12
            k13 k14 k15 k16 k17 k18 k19 k20 k21 k22
            k23 k24 k25 k26 k27 k28 k29 k30
</parameters>

<constraints>
    onP(p1,pl1) onP(p2,pl1) onP(p3,pl1) onP(p4,pl1)
    onP(p5,pl1) norm(pl1)
    onP(p1,pl2) onP(p2,pl2) onP(p6,pl2) onP(p7,pl2)
    onP(p8,pl2) norm(pl2)
    onP(p1,pl3) onP(p5,pl3) onP(p6,pl3) onP(p9,pl3)
    onP(p10,pl3) norm(pl3)
    onP(p4,pl4) onP(p5,pl4) onP(p10,pl4) onP(p11,pl4)
    onP(p12,pl4) norm(pl4)
    onP(p3,pl5) onP(p4,pl5) onP(p12,pl5) onP(p13,pl5)
    onP(p14,pl5) norm(pl5)
    onP(p2,pl6) onP(p3,pl6) onP(p14,pl6) onP(p15,pl6)
    onP(p8,pl6) norm(pl6)
    onP(p7,pl7) onP(p8,pl7) onP(p15,pl7) onP(p16,pl7)
    onP(p17,pl7) norm(pl7)
    onP(p6,pl8) onP(p7,pl8) onP(p17,pl8) onP(p18,pl8)
    onP(p9,pl8) norm(pl8)
    onP(p9,pl9) onP(p10,pl9) onP(p11,pl9) onP(p18,pl9)
    onP(p19,pl9) norm(pl9)
    onP(p11,pl10) onP(p12,pl10) onP(p13,pl10) onP(p19,pl10)
    onP(p20,pl10) norm(pl10)
    onP(p13,pl11) onP(p14,pl11) onP(p15,pl11) onP(p16,pl11)
    onP(p20,pl11) norm(pl11)
    onP(p16,pl12) onP(p17,pl12) onP(p18,pl12) onP(p19,pl12)
    onP(p20,pl12) norm(pl12)

    distpp(p1,p2,k1)
    distpp(p2,p3,k2)
    distpp(p3,p4,k3)
    distpp(p4,p5,k4)
    distpp(p5,p1,k5)

    distpp(p1,p6,k6)
```

```
            distpp(p6,p7,k7)
            distpp(p7,p8,k8)
            distpp(p8,p2,k9)

            distpp(p6,p9,k10)
            distpp(p9,p10,k11)
            distpp(p10,p5,k12)

            distpp(p10,p11,k13)
            distpp(p11,p12,k14)
            distpp(p12,p4,k15)

            distpp(p12,p13,k16)
            distpp(p13,p14,k17)
            distpp(p14,p3,k18)

            distpp(p14,p15,k19)
            distpp(p15,p8,k20)

            distpp(p15,p16,k21)
            distpp(p16,p17,k22)
            distpp(p17,p7,k23)

            distpp(p17,p18,k24)
            distpp(p18,p9,k25)

            distpp(p18,p19,k26)
            distpp(p19,p11,k27)
            distpp(p19,p20,k28)
            distpp(p20,p13,k29)
            distpp(p20,p16,k30)
        </constraints>
    </syntax>
</gcs>
</gcml>
```

C.3 Les dodécaèdres

C.3.1 Triaki-tétraèdre

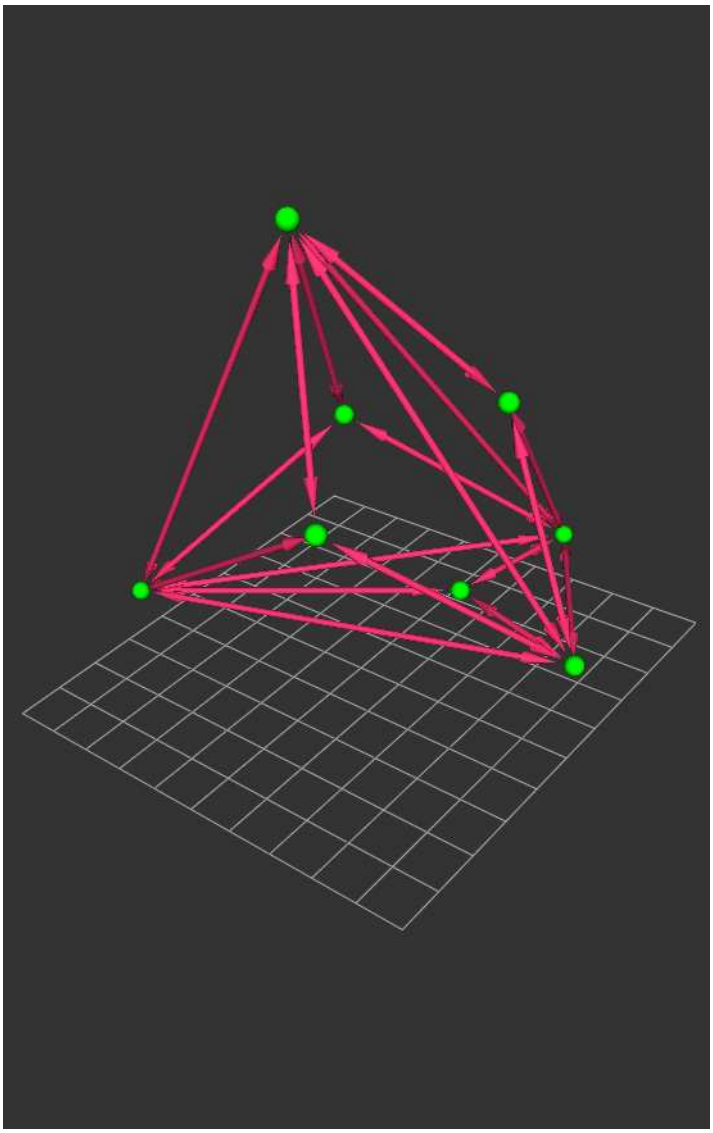


FIG. C.7 – Le triaki-tétraèdre dans l'espace

```
<gcs>
    <syntax>
        <unknowns>
            point p1 p2 p3 p4 p5 p6 p7 p8
        </unknowns>
        <parameters>
            measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 k13 k14
            k15 k16 k17 k18
        </parameters>
        <constraints>
```

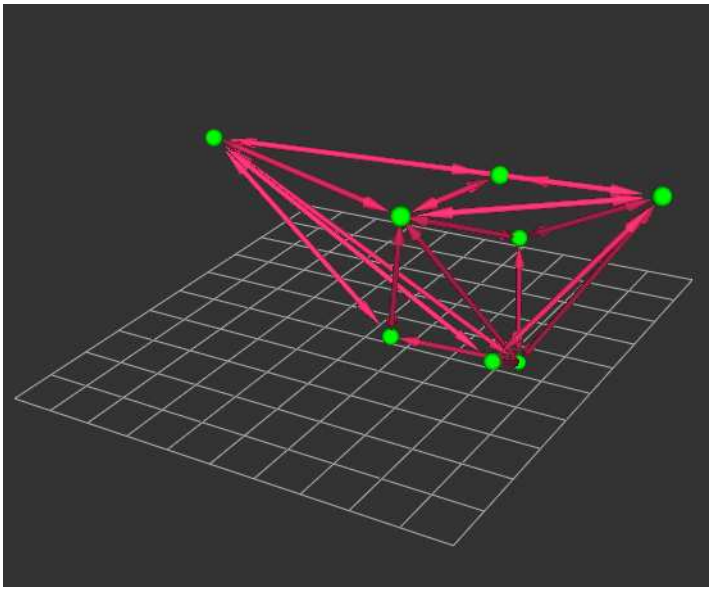


FIG. C.8 – Une solution au problème du triaki-tétraèdre

```
            distpp(p1,p2,k1)
            distpp(p1,p3,k2)
            distpp(p1,p4,k3)
            distpp(p2,p3,k4)
            distpp(p3,p4,k5)
            distpp(p2,p4,k6)
            distpp(p2,p6,k7)
            distpp(p6,p3,k8)
            distpp(p3,p7,k9)
            distpp(p4,p7,k10)
            distpp(p2,p5,k11)
            distpp(p4,p5,k12)
            distpp(p2,p8,k13)
            distpp(p3,p8,k14)
            distpp(p4,p8,k15)
            distpp(p6,p8,k16)
            distpp(p7,p8,k17)
            distpp(p5,p8,k18)
        </constraints>
    </syntax>
</gcs>
</gcml>
```

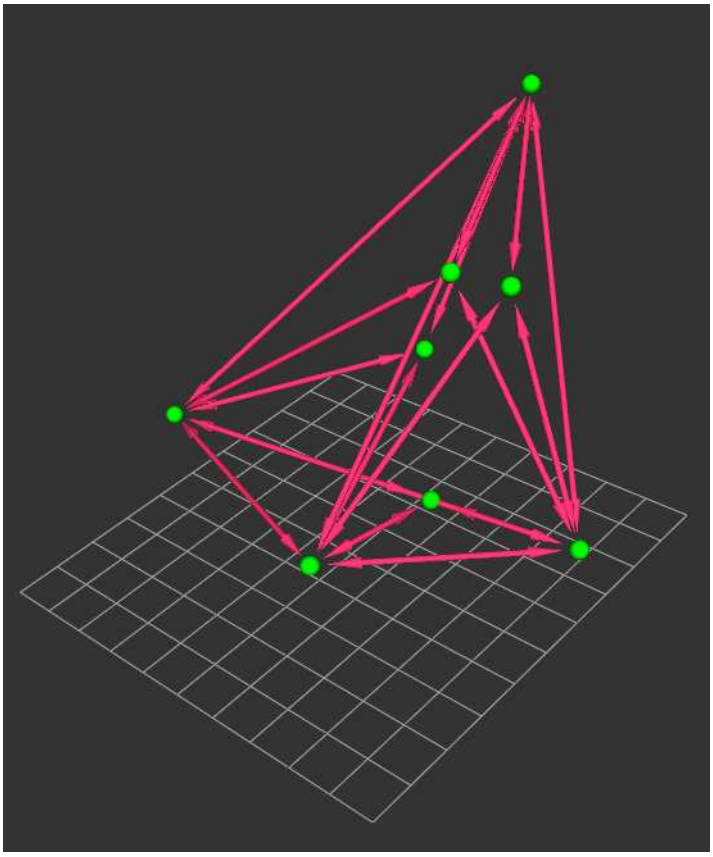


FIG. C.9 – Une deuxième solution au problème du triaki-tétraèdre

C.3.2 Dodécaèdre rhombique

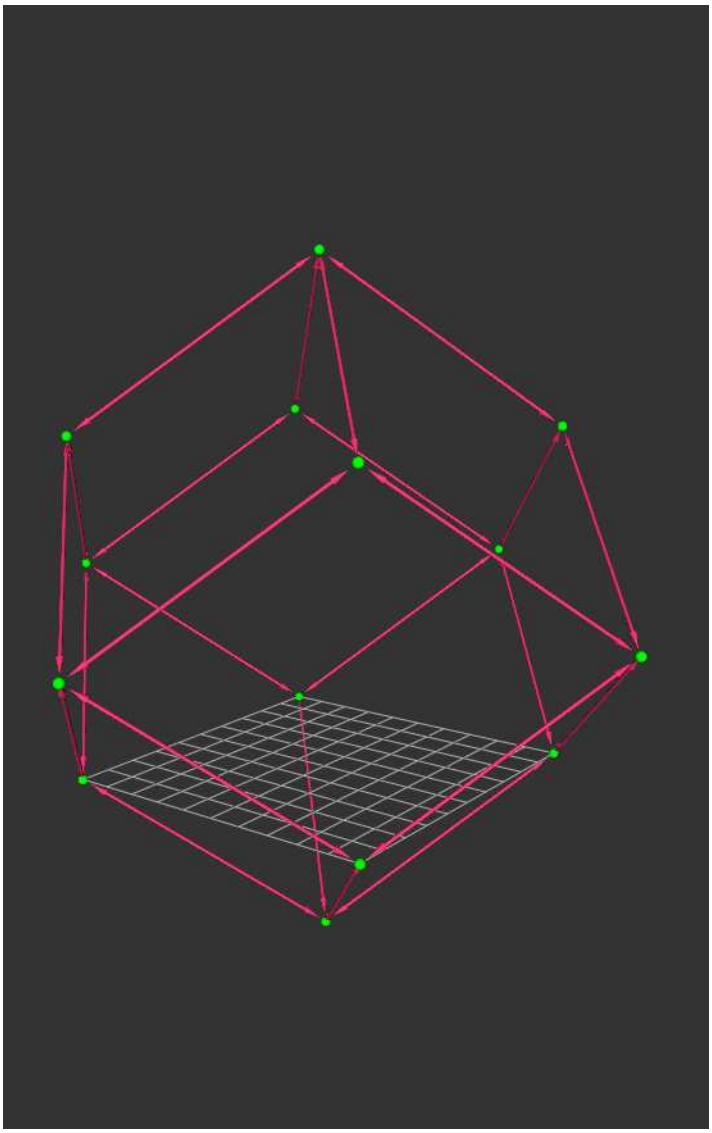


FIG. C.10 – Dodécaèdre Rhombique

```
<gcs>
    <syntax>
        <unknowns>
            point p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14
            plane pl1 pl2 pl3 pl4 pl5 pl6 pl7 pl8 pl9 pl10 pl11 pl12
        </unknowns>
        <parameters>
            measure k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 k13 k14
            k15 k16 k17 k18 k19 k20 k21 k22 k23 k24
        </parameters>
        <constraints>
            distpp(p13,p3,k1)
```

```
distpp(p3,p9,k2)
distpp(p9,p1,k3)
distpp(p1,p13,k4)
distpp(p11,p4,k5)
distpp(p4,p13,k6)
distpp(p1,p11,k7)
distpp(p2,p11,k8)
distpp(p9,p2,k9)
distpp(p4,p10,k10)
distpp(p10,p6,k11)
distpp(p6,p11,k12)
distpp(p2,p14,k13)
distpp(p6,p14,k14)
distpp(p9,p5,k15)
distpp(p5,p14,k16)
distpp(p3,p12,k17)
distpp(p5,p12,k18)
distpp(p7,p12,k19)
distpp(p7,p10,k20)
distpp(p7,p13,k21)
distpp(p8,p10,k22)
distpp(p8,p12,k23)
distpp(p8,p14,k24)
onP(p13,pl1) onP(p3,pl1) onP(p9,pl1) onP(p1,pl1)
norm(pl1)
onP(p11,pl2) onP(p4,pl2) onP(p13,pl2) onP(p1,pl2)
norm(pl2)
onP(p2,pl3) onP(p11,pl3) onP(p9,pl3) onP(p1,pl3)
norm(pl3)
onP(p4,pl4) onP(p10,pl4) onP(p6,pl4) onP(p11,pl4)
norm(pl4)
onP(p6,pl5) onP(p4,pl5) onP(p2,pl5) onP(p14,pl5)
norm(pl5)
onP(p6,pl6) onP(p10,pl6) onP(p8,pl6) onP(p14,pl6)
norm(pl6)
onP(p4,pl7) onP(p10,pl7) onP(p7,pl7) onP(p13,pl7)
norm(pl7)
onP(p4,pl8) onP(p7,pl8) onP(p12,pl8) onP(p3,pl8)
norm(pl8)
onP(p12,pl9) onP(p5,pl9) onP(p9,pl9) onP(p3,pl9)
norm(pl9)
onP(p5,pl10) onP(p14,pl10) onP(p8,pl10) onP(p12,pl10)
norm(pl10)
onP(p5,pl11) onP(p14,pl11) onP(p2,pl11) onP(p9,pl11)
norm(pl11)
onP(p7,pl12) onP(p12,pl12) onP(p3,pl12) onP(p13,pl12)
```

```
            norm(pl12)
        </constraints>
    </syntax>
<gcs>
```