Une approche pour inférer les expressions de calcul géométrique en modélisation à base topologique

Romain Pascual, Pascale Le Gall, Hakim Belhaouari, Agnès Arnould

Une approche pour inférer les expressions de calcul géométrique en modélisation à base topologique

Romain Pascual[1], Pascale Le Gall[1], Hakim Belhaouari[2], and Agnès Arnould[2]

[1]MICS, CentraleSupélec, Université Paris-Saclay
[2]Université de Poitiers, Univ. Limoges, CNRS, XLIM, Poitiers

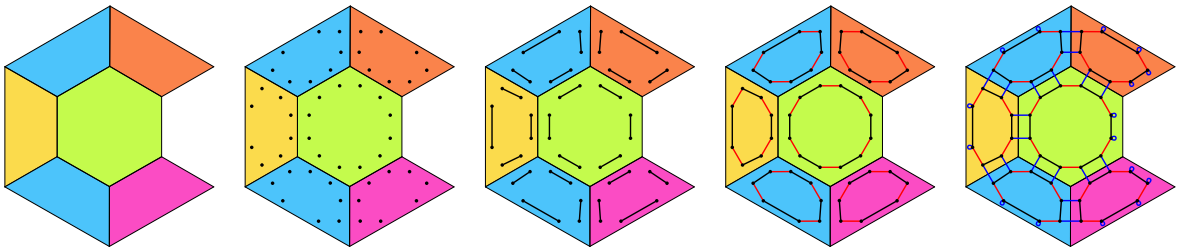
Résumé

La conception d'opérations de modélisation géométrique repose sur leur implantation dans un langage de programmation. Même si cette tâche peut être simplifiée en exploitant une description de haut niveau de ces opérations, la difficulté de les implanter contraste avec l'apparente simplicité de la description d'une opération sur un exemple. Nous proposons une méthode d'inférence d'opérations à partir d'un exemple représentatif. Plus précisément, nous nous plaçons dans le domaine de la modélisation géométrique à base topologique qui propose une représentation d'objets nD par décomposition en cellules topologiques (sommets, arêtes, faces, volumes, etc.) sur lesquelles sont ajoutées des informations géométriques. Ce domaine admet une spécification de la topologie par des structures combinatoires qui peuvent être représentées à l'aide de graphes, de sorte qu'une opération se formalise comme une règle de transformation de graphes. Dans cet article, nous complétons notre algorithme d'inférence du calcul topologique avec une approche pour la déduction des expressions de calcul géométrique. Notre approche exploite deux idées principales : des abstractions topologiques des expressions géométriques pour assurer la généricité des calculs retrouvés et une représentation comme un problème de satisfaction de contraintes de l'expression recherchée.

Mots-clés : modélisation géométrique à base topologique ; inférence d'opérations ; synthèse de code ; langage dédié ; CSP ; règles de transformations de graphes.

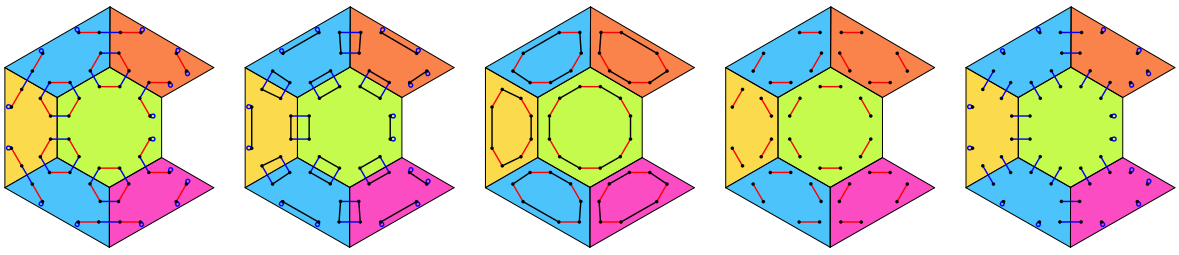
1 Introduction

Concevoir une nouvelle opération de modélisation géométrique suppose généralement l'élaboration d'un algorithme qui décrit les modifications à effectuer et l'implantation de l'algorithme dans un langage de programmation efficace. Nous proposons une approche exploitant l'intuition que l'on peut avoir de l'illustration d'une opération sur un exemple pertinent. Plus précisément, nous travaillons dans le domaine de la modélisation géométrique à base topologique qui sépare les modifications topologiques et géométriques de l'opération. Nous réalisons l'inférence d'opération en exploitant le langage dédié de la plateforme Jerboa [2] qui exploite les cartes généralisées comme représentation des objets et les règles de transformations de graphes comme formalisation des opérations. L'approche décrite ici permet de compléter l'algorithme de repliement topologique [7] en étudiant l'inférence d'expressions de calcul pour les informations dites de plongement qui encodent la géométrie de l'objet. L'inférence d'expressions géométriques associée à l'inférence topologique ainsi qu'à la compilation des règles permet ainsi la synthèse de code pour des opérations de modélisation géométrique à partir d'un simple exemple représentatif d'utilisation. Nous étudierons principalement les calculs géométriques liés aux positions des sommets pour lesquels nous supposerons que les nouvelles positions géométriques sont obtenues par combinaisons affines de barycentres topologiques.



(a) (b) (c) (d) (e)

Figure 1: G-carte associée à un objet géométrique: (a) objet géométrique 2D, (b) brins (points noirs), (c) ajout des 0-arcs (en noir), (d) ajout des 1-arcs (en rouge), (e) ajout des 2-arcs (en bleu).



(a) (b) (c) (d) (e)

Figure 2: Cellules topologiques et orbites: (a) orbites $\langle 1, 2 \rangle$ (sommets), (b) orbites $\langle 0, 2 \rangle$ (arêtes), (c) orbites $\langle 0, 1 \rangle$ (faces), (d) orbites $\langle 1 \rangle$ (sommets de faces), (e) orbites $\langle 2 \rangle$ (sommets d'arêtes).

2 Représentation des objets et leur manipulation dans un langage dédié

2.1 Représentation topologique et géométrique d'un objet nD

Nous représentons les objets géométriques avec le modèle des cartes généralisées [3] vues sous forme de graphes [8].

Une *carte généralisée* de dimension n (avec $n \geq 0$), n-G-carte ou simplement G-carte, est un graphe non orienté, étiqueté sur les arcs par les dimensions de $0..n$ (où $i..j$ est l'ensemble des entiers entre i et j inclus) et vérifiant deux conditions de consistance topologique:

- *contrainte d'arcs incidents* : tout nœud possède un unique arc incident par dimension.
- *contrainte de cycles* : pour toutes dimensions d et d' telles que $d \leq d' + 2$, tout chemin étiqueté par $dd'dd'$ est un cycle.

Par héritage de la vision combinatoire, on appelle *brin* d'une G-carte les nœuds du graphe. Un brin représente un n-uplet de cellules topologiques. Ainsi, un brin d'une 2-G-carte représente un sommet, une arête et une face. Les arcs du graphes expliquent alors comment les cellules topologiques sont reliées entre elles. Plus précisément, un arc de dimension d explicite que deux cellules de dimension d sont adjacentes tout en partageant les cellules des autres dimensions. Ainsi, un arc de dimension 0, aussi appelé 0-arc, sépare deux sommets d'une même arête et d'une même face.

A partir de ces informations, on peut reconstruire la G-carte associée à un objet, comme illustré sur la Figure 1. Partant d'un objet 2D (Figure 1a), on ajoute un brin pour chaque triplet (sommet, arête, face) valide (Figure 1b). On relie ensuite les paires de brins séparant les sommets d'une même arête vue d'une même face à l'aide de 0-arcs (en noir sur la Figure 1c). Cette construction est ensuite itérée par dimension croissante, on ajoute des 1-arcs entre les brins d'un même sommet et d'une même face mais d'arêtes différentes (en rouge sur la Figure 1d) et finalement des 2-arcs pour relier les faces le long d'une arête commune (en bleu sur la Figure 1e). On notera qu'un brin du bord topologique de dimension d est relié à lui-même par un d-arc. Par exemple, les faces du bord engendrent des 2-boucles sur la Figure 1e.

Cette représentation permet de définir les cellules topologiques comme des sous-graphes induits par un ensemble de dimensions. Etant donné un brin v d'une G-carte G et un ensemble de dimensions

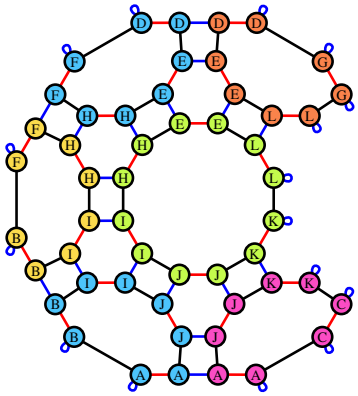


Figure 3: G-carte plongée par l'ajout des positions et des couleurs (représentés dans les brins).

$o \subset 0..n$, l'*orbite* $G\langle o\rangle$(v) est le sous-graphe maximal induit par o et contenant v. On dit alors que $G\langle o\rangle$(v) est de type $\langle o\rangle$ ou est une $\langle o\rangle$-orbite. Par exemple, l'orbite $G\langle 1,2\rangle(v)$ contient l'ensemble des brins auquel on peut accéder en n'empruntant que des arcs de dimension 1 et 2 à partir du brin v. Vu autrement, les $\langle 1,2\rangle$-orbites sont les composantes connexes du graphe construites à partir de la G-carte en supprimant les arcs de dimension 0. Ces orbites encodent les sommets de l'objet, comme illustré sur la Figure 2a. De même, on encode les arêtes avec les $\langle 0,2\rangle$-orbites (Figure 2b) et les faces avec les $\langle 0,1\rangle$-orbites (Figure 2c). Les orbites décrivent aussi des graphes qui ne sont pas des cellules topologiques. Par exemple, la Figure 2d illustre les $\langle 1\rangle$-orbites qui encodent les sommets de faces et la Figure 2e les $\langle 2\rangle$-orbites qui encodent les sommets d'arêtes. D'autres orbites ont été illustrées implicitement lors de la construction de la G-carte comme les orbites vides de type $\langle\rangle$, réduites à un brin (Figure 1b), les $\langle 0\rangle$-orbites (Figure 1c), qui encodent les arêtes de faces, ou les orbites de type $\langle 0,1,2\rangle$, qui encodent les composantes connexes, ici l'objet complet (Figure 1e).

Les informations géométriques ajoutées sur cette structure topologique sont encodées par des fonctions de plongement qui associent une valeur d'un type donné aux cellules topologiques. Ainsi, nous pouvons ajouter des positions aux sommets ou des couleurs aux faces. Comme les cellules topologiques ne sont pas des éléments atomiques de la G-carte mais des sous-graphes, un plongement π est décrit par un type d'orbite $\langle o\rangle$ et un type de donnée τ (au sens des types abstraits algébriques). On obtient alors des fonctions de plongement π de profil $\langle o_\pi\rangle \rightarrow \tau_\pi$. Par exemple, les positions sont encodées par une fonction `position` : $\langle 1,2\rangle \rightarrow$ `Point3` et les couleurs par une fonction `color` : $\langle 0,1\rangle \rightarrow$ `ColorRGB` où `Point3` et `ColorRGB` désignent les types de données usuels utilisés en informatique graphique. Formellement, nous utilisons la description par des graphes attribués [4] qui permet de manipuler les valeurs attachées aux nœuds du graphe [6]. Dans notre cas, chaque nœud de la G-carte possède une unique valeur pour chaque plongement avec la condition que tous les brins d'une $\langle o_\pi\rangle$-orbite possède la même valeur pour le plongement π. L'ajout des plongements à l'objet de la Figure 1 est illustré en Figure 3 où les informations de position et de couleur sont visibles sur chacun des brins de la G-carte.

2.2 Un langage fondé sur les transformations de graphes

Nous exploitons une formalisation des opérations de modélisation géométrique sous la forme de règles de réécriture de graphes [4, 6]. De telles règles sont une extension aux structures non-linéaires de la réécriture de mots ou de termes. Une règle se présente donc sous la forme $L \rightarrow R$ où L et R sont des graphes, décrivant respectivement le motif à modifier et le motif réécrit. Pour des questions de généricité, nos règles se présentent sous la forme de schémas de règles paramétrés par un type d'orbite $\langle o\rangle$ afin d'abstraire les changements topologiques. Ces schémas de règles sont au cœur de la conception d'opérations dans la plateforme Jerboa [2]. Un exemple de schéma de règles est donné en Figure 4a, illustrant la triangulation barycentrique d'une face. Ce schéma de règles modifie une $\langle 0,1\rangle$-orbite, c'est-à-dire une face. Plus précisément, le nœud étiqueté $\langle 0,1\rangle$ du membre gauche de la règle est appelé à être instancié par une face avec un nombre arbitraire de brins. Les différents nœuds

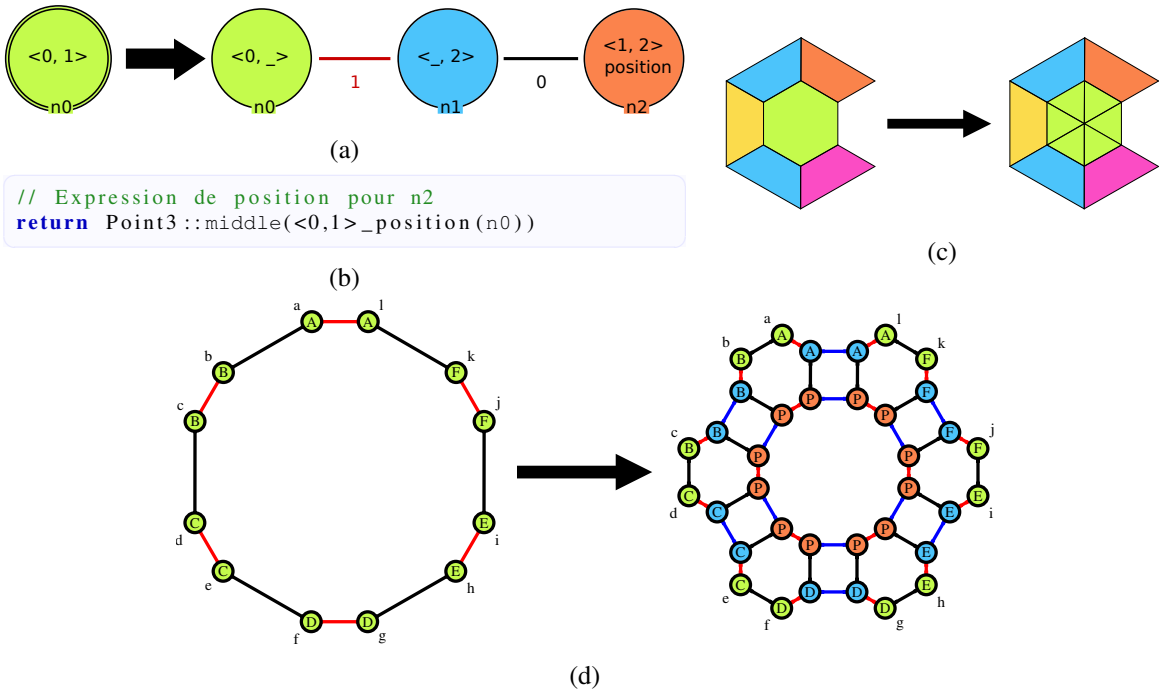


Figure 4: Schéma de règles pour la triangulation d'une face: (a) schéma de règles, et (b) expression géométrique associée au nœud $n2$.

du motif droit décrivent alors les différentes orbites à modifier ou à ajouter avec les modifications topologiques à effectuer. Par exemple, le nœud $n0$ possède le type d'orbite $\langle 0, \_ \rangle$. Les brins et les 0-arcs de la face filtrée sont préservés tandis que ses 1-arcs sont supprimés, ce qui déconnecte les arêtes de la face. De même, le nœud $n2$ possède le type d'orbite $\langle 1, 2 \rangle$ pour construire le sommet dual de la face, ajouté pour trianguler cette dernière : l'orbite d'origine est dupliquée en renommant les 0-ars en 1-arcs et les 1-arcs en 2-arcs. Le nœud $n1$ et les deux arcs du membre droit permettent de connecter la face initiale au sommet dual. L'instanciation du schéma de règles sur une face hexagonale est donnée en Figure 4d où la couleur des brins décrit l'association avec le nœud du schéma de la Figure 4a: par exemple les brins $\{a, b, c, d, e, f, g, h, i, j, k, l\}$ sont associés au nœud $n0$.

L'annotation `position` sur le nœud $n2$ à droite indique qu'il porte une expression géométrique. Le calcul correspondant est donné en Figure 4b. L'expression renvoie un terme de type `Point3` dont la valeur est donnée par l'expression $\texttt{middle}(\langle 0, 1 \rangle \texttt{\_position}(n0))$. Dans cette expression, `middle` retourne le barycentre d'une collection de positions et $\langle 0, 1 \rangle$`_position` exprime que l'on doit récupérer l'ensemble des positions dans une orbite $\langle 0, 1 \rangle$. Lors de l'application de la règle, l'identifiant $n0$ sera substitué par les brins associés au nœud $n0$ avant modification. Dans le cas présent, il s'agit des brins de la face que l'on triangule. En substituant $n0$ par l'identifiant d'un des brins de la face, par exemple b, on obtient alors $\texttt{middle}(\langle 0, 1 \rangle \texttt{\_position}(b))$ qui encode le barycentre des positions des sommets de l'orbite $G\langle 0, 1 \rangle(b)$, c'est-à-dire le barycentre de la face. Soulignons que n'importe quel choix parmi les brins $a, \ldots, l$ encoderait le même barycentre.

3 Inférence d'opérations

Dans cet article, nous nous intéressons à l'inférence d'opérations de modélisation géométrique. Tout comme la description des objets et la formalisation du langage, cette inférence est nécessairement double, c'est-à-dire topologique et géométrique. La reconstruction des modifications topologiques a été étudiée dans [7] et repose sur un algorithme de repliement. Il s'agit intuitivement d'un parcours de graphe qui reconstruit le schéma de règles à partir d'un exemple représentatif. Un tel exemple consiste en une instance de l'objet avant modification, une instance de l'objet après modification, d'informations associant les parties non modifiées de l'objet et enfin d'un type d'orbite spécifiant la

généricité de l'opération. Nous présentons ici une méthode pour déduire les expressions de calcul des modifications géométriques à associer à la règle afin de compléter l'inférence topologique. Nous disposons toujours d'un exemple représentatif de l'opération, du résultat de l'algorithme de l'inférence topologique [7] augmentés de résultats intermédiaires obtenus lors des calculs de repliements.

3.1 Espace de recherche

Nous commençons par le cas des positions associées aux sommets, plongement nécessaire pour afficher des objets polyédriques. Puisque nous disposons d'une instance de l'objet avant et après modification, nous pouvons accéder aux valeurs de plongement des brins associés à un nœud du membre droit comme du membre gauche. En particulier, nous pourrons vérifier que le résultat du calcul de l'expression ajoutée à un nœud du membre droit de la règle correspond à la valeur de plongement des brins associés. Dans un premier temps, nous proposons une approche pour déterminer le placement des expressions géométriques sur les nœuds du membre droit :

1. Si v_R est un nœud préservé du membre droit et que tous les brins qui lui sont associés ont la même position dans les instances avant et après, alors aucun calcul ne doit être réalisé, et nous laissons l'expression du nœud droit vide. Nous marquons l'orbite $\langle 1, 2\rangle(v_R)$ comme résolue.
2. Pour chaque $\langle 1, 2\rangle$-orbite non résolue, nous choisissons un nœud référent pour accueillir une expression de plongement.

Le marquage des $\langle 1, 2\rangle$-orbites dans l'étape 1 et le choix d'un référent par $\langle 1, 2\rangle$-orbite non résolue dans l'étape 2 repose sur la propagation des valeurs de plongement calculées lors de l'application d'un schéma de règles. On réduit ainsi en amont le nombre d'expressions géométriques à résoudre. Dans le cas où nous ne pourrions pas trouver de solution pour le nœud référent choisi à l'étape 2, la contrainte de cohérence des plongements [1] assure que nous n'en trouverions pas pour les autres nœuds de son orbite puisque les brins ont nécessairement la même position.

Une fois que nous avons identifié les nœuds référents pour accueillir une expression de plongement, il faut retrouver une expression géométrique valide pour l'ensemble des brins associés aux nœuds. Plus encore, l'expression doit rester valide qu'importe l'instanciation. Autrement dit, le résultat du calcul ne doit pas dépendre du brin choisi pour effectuer le calcul et nous avons besoin d'une solution permettant d'encoder l'invariance du brin dans le calcul induite par l'abstraction topologique des schémas de règles. La solution que nous proposons est d'utiliser des opérateurs *adhoc* en lien avec les informations topologiques comme l'opérateur de collecte qui récupère l'ensemble des valeurs d'une orbite. De manière plus générale, nous appelons *point d'intérêt* toute solution qui encode une abstraction de la topologie sous-jacente invariante au choix du brin.

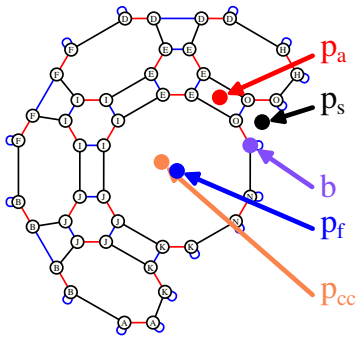
Bien que les points d'intérêt éliminent la question du choix du brin dans le calcul, il convient encore de spécifier l'ensemble des fonctions à considérer pour résoudre concrètement l'inférence des expressions géométriques. Nous appelons *famille de fonctions* cet ensemble de fonctions. Nous allons considérer la famille des combinaisons affines qui présentent le double avantage d'être souvent utilisées pour les opérations de modélisation géométrique tout en étant simples à résoudre. Concrètement, nous construisons, pour chacun des nœuds référents, une équation symbolique à partir des points d'intérêts. L'idée est ensuite d'imiter l'instanciation du schéma de règles en évaluant cette équation sur chaque brin associé au nœud de la règle. On remplace donc chaque point d'intérêt par sa valeur calculée en fonction des positions de l'instance avant modification, ce qui permet de générer un système d'équations affines. Ce système définit un problème de satisfaction de contraintes (ou CSP) où les variables sont les poids de la combinaison de domaine $\mathbb{R}$, liées par une unique contrainte déduite de l'évaluation de l'équation symbolique. Nous déléguons ensuite la résolution du CSP à un solveur dédié (nous avons expérimenté avec les solveurs OR-tools de Google et Z3 de Microsoft).

3.2 Points d'intérêt

Comme évoqué précédemment, une règle est symétrique par rapport aux orbites portées par ses nœuds. Ainsi, une expression de plongement doit permettre de calculer les positions de tous les

Point d'intérêt	Nom	Expression
sommet	p_s	middle($\langle 1,2\rangle$_position(b))
arête	p_a	middle($\langle 0,2\rangle$_position(b))
face	p_f	middle($\langle 0,1\rangle$_position(b))
cc	p_{cc}	middle($\langle 0,1,2\rangle$_position(b))

(a)



(b)

Figure 5: Points d'intérêt: (a) expressions des points d'intérêt , (b) calculs sur un objet.

brins instances du nœud tout en s'évaluant de manière identique pour tous les brins d'une même orbite de plongement. Pour assurer ces propriétés, nous exprimons les équations par des points d'intérêt invariants par symétrie sur les différentes orbites. Nous proposons d'utiliser les barycentres des positions des différentes orbites. En exploitant que le type d'orbite $\langle 1,2\rangle$ du plongement position induit une relation d'équivalence sur l'ensemble des orbites (dépendant également de la contrainte de cycle), on peut se ramener à considérer seulement 4 orbites pour un objet 2D. Ces orbites correspondent aux sommets, arêtes, faces et composantes connexes. Les barycentres d'une orbite $\langle o\rangle$ sont encodés par les expressions middle($\langle o\rangle$_position(v)) où v est un nœud gauche du schéma. Il s'agit de l'expression déjà introduite pour la triangulation barycentrique (cf. Section 2.2) mais en anonymisant l'orbite. Les points d'intérêt sont donnés dans le tableau 5a et illustrés en Figure 5b.

Nous généralisons ces expressions comme des expressions de plongement pour pouvoir les utiliser sur un schéma de règles quelconque et notons $\mathrm{PI}(v)$ pour l'ensemble des expressions des points d'intérêt du nœud v. A partir de ces points, on obtient donc l'équation symbolique suivante :

$$\texttt{position}(v_R) = \sum_{v_L \in V_L} \sum_{p \in \mathrm{PI}(v_L)} w_{p,v_L} p(v_L) + t \tag{1}$$

où V_L est l'ensemble des nœuds du membre gauche du schéma de règles, $(w_{p,v_L})_{v_L \in V_L}$ sont des poids (inconnus), et t encode une translation intrinsèque (inconnue).

3.3 Génération automatique du code des expressions géométriques

Nous générons automatiquement le code de l'expression géométrique à partir de la solution trouvée par le solveur CSP. Cette expression exploite le langage de script de Jerboa qui étend la vision purement algébrique des calculs [1] dans un langage impératif [5] dont la syntaxe est similaire au langage `Java`. Voici le modèle générique du code de l'expression déduite :

```
Point3 res = new Point3( #translation# );                          // translation

// pour tout noeud #node# du motif gauche
Point3 pS_#node# = Point3::middle(<1,2>_position( #node# ));       // sommet
pS_#node#.scaleVect( #w(s,#node#)# );                              // poids
res.addVect(pS_#node#);

Point3 pA_#node# = Point3::middle(<0,2>_position( #node# ));       // arete
pA_#node#.scaleVect( #w(a,#node#)# );                              // poids
res.addVect(pA_#node#);

Point3 pF_#node# = Point3::middle(<0,1>_position( #node# ));       // face
pF_#node#.scaleVect( #w(f,#node#)# );                              // poids
res.addVect(pF_#node#);

Point3 pCC_#node# = Point3::middle(<0,1,2>_position( #node# ));    // composante connexe
pCC_#node#.scaleVect( #w(cc,#node#)# );                            // poids
res.addVect(pCC_#node#);

return res;
```

Les expressions générées permettent l'ajout des différents points d'intérêt au vecteur de translation. Les expressions `Point3::middle(< #orb# >_position( #node# ))` code le calcul des points d'intérêt qui sont ensuite multipliés par le poids calculé lors de la résolution du CSP via l'expression `#poi#.scaleVect( #w(#poi#,#node#)# )`. Finalement, la contribution du point d'intérêt est ajouté au calcul global avec l'expression `res.addVect(#poi#);`. Dans le cas où le poids calculé est nul (avec un seuil de tolérance), le code associé n'est pas généré.

Pour conclure, nous mentionnons aussi que l'on exploite la condition de cohérence des plongements pour réduire le nombre de variables du système. En effet, si deux nœuds sont dans la même $\langle 1, 2\rangle$-orbite du membre gauche, leurs brins associés auront la même position. On peut donc remplacer V_L par $V_L/\langle 1, 2\rangle$ dans l'équation 1.

3.4 Résolution de la triangulation barycentrique

Nous illustrons notre méthode en reprenant le schéma de règles pour la triangulation (cf. Figure 4a) et en considérant les instances avant et après de la face hexagonale (cf. Figure 4d). Le nœud $n0$ est préservé et les brins qui lui sont associés $(a, b, \ldots, l)$ possèdent les mêmes positions $(A, B, \ldots, F)$ avant et après modification. Nous pouvons en déduire que $n0$ est résolu sans avoir besoin d'expression. Nous marquons donc l'orbite $\langle 1, 2\rangle(n0)$ comme résolue, ce qui permet d'éliminer le nœud $n1$ (cf. étape 1). Il reste alors le nœud $n2$ à résoudre via l'équation symbolique 1. Puisque le membre gauche ne contient que le nœud $n0$, on obtient l'équation:

$$\texttt{position}(n2) = \underbrace{w_{(s,n0)} p_s(n0)}_{\text{sommet}} + \underbrace{w_{(a,n0)} p_a(n0)}_{\text{arête}} + \underbrace{w_{(f,n0)} p_f(n0)}_{\text{face}} + \underbrace{w_{(cc,n0)} p_{cc}(n0)}_{\text{composante connexe}} + t$$

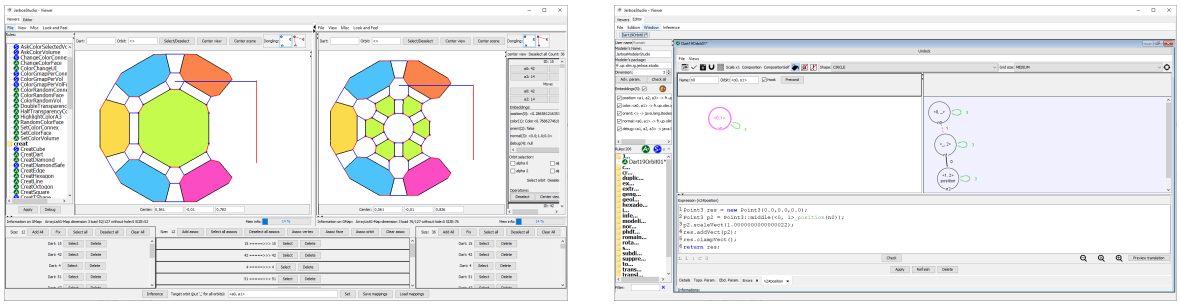
On évalue cette équation sur les 12 brins de l'hexagone (cf. Figure 4d). En projetant le système sur les axes x, y et z, nous obtenons un système à 36 équations et 7 variables. A l'aide d'un solveur, nous obtenons la solution $w_{(s,n0)} = 0.0$, $w_{(a,n0)} = 0.0$, $w_{(f,n0)} = 1.0$, $w_{(cc,n0)} = 0.0$, $tx = 0.0$, $ty = 0.0$, et $tz = 0.0$. A partir de cette solution, nous générons le code suivant :

```
Point3 res = new Point3(0.0,0.0,0.0);              // translation nulle
Point3 p2 = Point3::middle(<0,1>_position(n0));    // face
p2.scale(1.0);                                     // poids
res.addVect(p2);
return res;
```

3.5 Généralisation à un espace vectoriel

Bien que nous ayons expliqué l'inférence des expressions pour les positions associées aux sommets, cette approche fonctionne pour d'autres types de plongement. En effet, dès lors que les valeurs de plongement appartiennent à un espace vectoriel, notre méthode par combinaison affines de barycentre fonctionne. On étend la notion de points d'intérêt en *valeurs d'intérêt*, qui collectent les valeurs dans une orbite et calculent le barycentre du multi-ensemble obtenu. En pratique, il suffit d'encapsuler les plongements vectoriels en codant des opérateurs de somme et de multiplication par un scalaire d'une part et de généraliser le calcul des orbites pertinentes pour un plongement donné.

Nous avons ainsi expérimenté cette approche sur le type de données `ColorRGB` porté par le type d'orbite $\langle 0, 1\rangle$, c'est-à-dire les faces. La principale difficulté liée aux couleurs RGB provient de leur encodage comme des vecteurs dans $[0, 1]^3$ et non dans $\mathbb{R}^3$. Une première solution consiste à laisser l'expression déduite inchangée. L'application des règles peut alors conduire à des couleurs en dehors de l'espace représentable, ce qui convient lorsque l'utilisateur dispose d'un mécanisme pour traiter ces cas limites. Une deuxième solution est de modifier automatiquement la solution déduite afin de restreindre le résultat du calcul. Cette modification se produit sur l'expression déduite et ne modifie pas la résolution du système d'équations. Nous pouvons aussi retourner une couleur par défaut ou une couleur aléatoire comme solution finale.



(a) (b)

Figure 6: JerboaStudio : (a) viewer et (b) éditeur.

3.6 JerboaStudio

Nous avons implanté notre méthode dans la plateforme Jerboa [2]. Jerboa contient : (a) un éditeur qui permet la conception d'opérations sous forme de règles, (b) un noyau qui permet de traduire ces règles en code efficace et (c) un viewer qui permet d'afficher des objets géométriques représentés sous forme de G-cartes et de leur appliquer les opérations obtenues par synthèse de code à partir des règles. L'éditeur permet à un spécialiste de la modélisation géométrique de construire un logiciel pour un expert d'un domaine applicatif qui sera amené à utiliser uniquement le viewer. L'approche de conception d'opération par inférence à partir d'un exemple présentée ici pour les aspects géométriques et dans [7] pour les aspects topologiques vise à rendre caduque cette séparation des tâches et donc des différents composants de Jerboa. Nous avons donc conçu un outil entièrement intégré appelé JerboaStudio et illustré en Figure 6. Une version de JerboaStudio est disponible librement en ligne.[1]

4 Une application en 3D: l'éponge de Menger

L'éponge de Menger est une extension en 3D de l'ensemble de Cantor (1D) ou du tapis de Sierpinski (2D). Une étape de subdivision peut être construite comme suit : prendre un cube et diviser chaque face en 9 carrés pour obtenir 27 cubes, puis enlever tous les cubes du milieu (milieu des faces et centre du cube initial). L'itération suivante applique cette subdivision à chacun des 20 cubes restants. Afin de pouvoir inférer les éléments géométriques de cette opération, nous avons, dans un premier temps, construit l'éponge de Menger (Figure 7b) à partir du cube (Figure 7a) à l'aide d'une règle écrite par nos soins (Figure 7e). Il s'agit ici d'une opération volumique sur une 3-G-carte qui suppose que le schéma de règles associé utilise le type d'orbite $\langle 0, 1, 2, 3 \rangle$, utilisé ici pour annoter le nœud $n0$. Nous avons ensuite retiré les expressions géométriques de la règle (Figure 7e).

À partir des deux premiers objets (Figure 7a et 7b) et du schéma de règles épuré, nous avons appliqué la méthode d'inférence décrite dans cet article. Notons que chaque nœud du schéma de règles encode 48 brins (8 brins par face carrée, et 6 faces carrées pour un cube) de sorte qu'il ne manque que 3 calculs de positions. Ces expressions sont portées par les nœuds $n1$, $n7$, et $n16$ dans la Figure 7e. Des exemples de sommets associés à ces nœuds sont donnés en Figure 7f. Le brin de référence pour ces trois sommets appartient au sommet inférieur droit de l'objet. Cette démarche a été motivée par l'idée de pouvoir comparer la règle inférée avec la règle de référence construite manuellement. A titre d'illustration, nous donnons ci-dessous les expressions inférées pour les 3 nœuds $n1$, $n7$ et $n16$.

[1] Dépôt consulté le 22 mai 2023 : https://gitlab.com/jerboateam/jerboa-studio

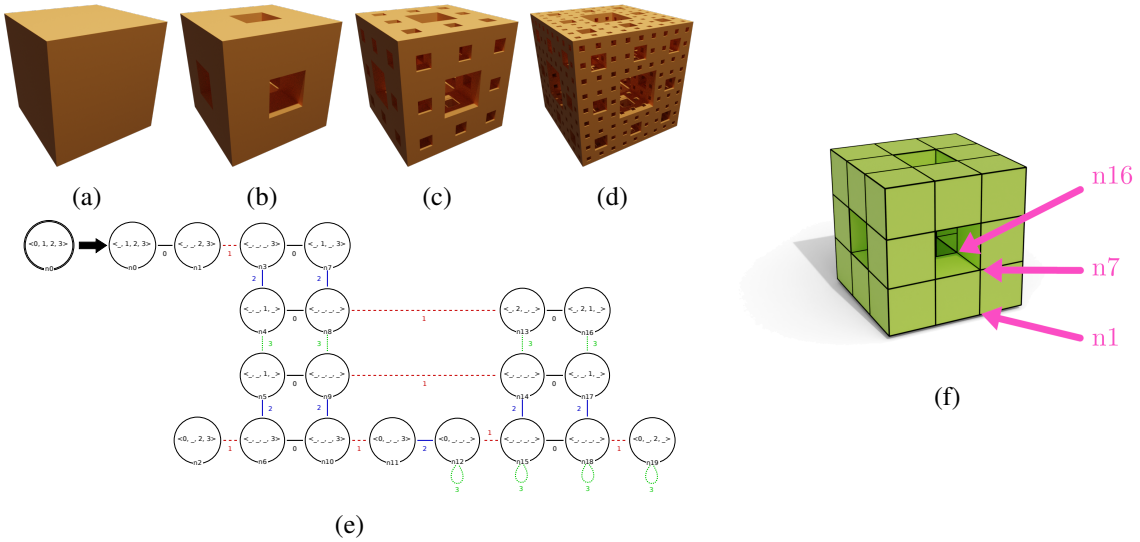


Figure 7: Éponge de Menger: un cube (a), première (b), seconde (c) et troisième (d) itérations, schéma de règles associé (e) et mise en évidence de sommets encodés par les nœuds $n1$, $n7$, and $n16$.

Nœud $n1$:

```
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.3333333134651184);
res.addVect(p0);
Point3 p1 = Point3::middle(<0>_position(n0));
p1.scale(0.6666666865348816);
res.addVect(p1);
return res;
```

Nœud $n7$:

```
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.3333333134651184);
res.addVect(p0);
Point3 p2 = Point3::middle(<0,1>_position(n0));
p2.scale(0.6666666865348816);
res.addVect(p2);
return res;
```

Nœud $n16$:

```
Point3 res = new Point3(0.0,0.0,0.0);
Point3 p0 = Point3::middle(<>_position(n0));
p0.scale(0.3333333134651184);
res.addVect(p0);
Point3 p3 = Point3::middle(<0,1,2>_position(n0));
p3.scale(0.6666666865348816);
res.addVect(p3);
return res;
```

Lorsque nous appliquons les deux versions de l'opération - celle conçue manuellement et celle inférée à l'aide de l'approche proposée - sur un cube, elles fournissent le même le résultat : la première itération de l'éponge de Menger de la Figure 7b. Lorsque nous les appliquons sur d'autres objets, nous n'avons plus de garantie que le résultat sera le même, comme illustré sur la Figure 8. Soulignons que même si l'opération inférée ne correspond pas aux attentes de l'utilisateur, elle présente l'avantage d'être facilement disponible pour aider l'utilisateur à ajuster l'opération en cours de conception. Pour conclure, puisque l'éponge de Menger n'est formellement définie que dans le cas du cube, nous laissons le choix de la version préférée au goût esthétique du lecteur.

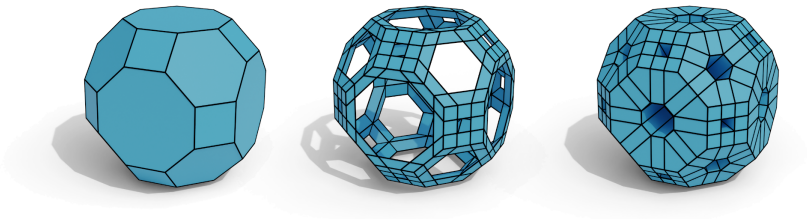


Figure 8: Illustration des deux calculs géométriques : (de gauche à droite) objet initial, résultat de la règle conçue manuellement, et résultat de l'opération inférée.

5 Conclusion

Nous avons décrit une approche générique pour l'inférence d'expressions géométriques afin d'être en mesure de générer automatiquement des opérations de modélisation géométrique prêtes à l'emploi. Pour nous guider dans la recherche de solutions, nous utilisons un langage dédié à base de règles de transformations de graphes. Nous avons introduit la notion de valeur d'intérêt (ici les barycentres des cellules topologiques) comme une valeur dont l'expression se calcule indépendamment de la topologie sous-jacente. Ces valeurs d'intérêt, couplées aux familles de fonctions (ici les combinaisons affines), permettent une représentation sous la forme d'un problème de satisfaction de contraintes que nous résolvons à l'aide de solveurs génériques. Par ailleurs, nous avons implanté notre méthode d'inférence géométrique au sein de l'outil JerboaStudio, comme extension de la plateforme Jerboa. Afin d'étoffer notre approche, nous avons l'intention d'étendre les familles des types de données et des valeurs d'intérêt associées.

References

[1] Agnès Arnould, Hakim Belhaouari, Thomas Bellet, Pascale Le Gall, and Romain Pascual. "Preserving consistency in geometric modeling with graph transformations". In: *Mathematical Structures in Computer Science* (Oct. 18, 2022). Publisher: Cambridge University Press, pp. 1–48. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129522000226.

[2] Hakim Belhaouari, Agnès Arnould, Pascale Le Gall, and Thomas Bellet. "Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling". In: *Graph Transformation*. ICGT 2014. Ed. by Holger Giese and Barbara König. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 269–284. ISBN: 978-3-319-09108-2. DOI: 10.1007/978-3-319-09108-2_18.

[3] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, Sept. 19, 2014. 407 pp. ISBN: 978-1-4822-0652-4.

[4] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Berlin Heidelberg: Springer-Verlag, 2006. ISBN: 978-3-540-31187-4. DOI: 10.1007/3-540-31188-2.

[5] Valentin Gauthier. "Développement d'un langage de programmation dédié à la modélisation géométrique à base topologique, application à la reconstruction de modèles géologiques 3D". These de doctorat. Poitiers, Jan. 17, 2019.

[6] Reiko Heckel and Gabriele Taentzer. *Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering*. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-43915-6. DOI: 10.1007/978-3-030-43916-3.

[7] Romain Pascual, Hakim Belhaouari, Agnès Arnould, and Pascale Le Gall. “Inferring topological operations on generalized maps: Application to subdivision schemes”. In: *Graphics and Visual Computing* 6 (May 14, 2022), p. 200049. ISSN: 2666-6294. DOI: 10.1016/j.gvc.2022.200049.

[8] Romain Pascual, Pascale Le Gall, Agnès Arnould, and Hakim Belhaouari. “Topological consistency preservation with graph transformation schemes”. In: *Science of Computer Programming* 214 (Feb. 1, 2022), p. 102728. ISSN: 0167-6423. DOI: 10.1016/j.scico.2021.102728.