Évaluation de la modélisation à base de transformation de graphes avec Jerboa

Valentin Gauthier[1], Hakim Belhaouari[1] et Agnès Arnould[1]

[1]Université de Poitiers, XLim-SIC, UMR CNRS 7252
Bât. SP2MI, Téléport 2, 11 Bd Marie et Pierre Curie, BP 30179
86962 Futuroscope Chasseneuil Cedex

Résumé

Depuis plusieurs décennies les approches topologiques permettent de modéliser des objets géométriques structurés de manière sûr et efficace. Jerboa est une jeune plateforme offrant une structure topologique et mettant en avant le développement rapide de nouvelles opérations et de modeleurs dédiés pour divers domaines. Sa force est l'utilisation des transformations de graphes pour définir les opérations et vérifier automatiquement les différentes propriétés de cohérence des objets.
Un autre enjeu de la modélisation est de pouvoir exécuter les opérations rapidement. Cet article présente une étude de performance de différentes bibliothèques à base topologique et de Jerboa, aux travers de différentes opérations typiques de la modélisation.

Abstract

Last decades, the topological approaches used to improve the reliability and efficiency of model geometric structured objects. Jerboa is a young platform that relies on a topological structure based on generalized map, and focuses rapid development of new operations and modelers dedicated to various area. Its main feature is the use of graph transformations for defining new operations, in order to check automatically the consistency of different properties of objects.
Another challenge of modeling is to perform operations efficiently. This paper presents a performance study of time execution using typical modeling operations for different topology-based libraries and Jerboa.

Mots-clés : modélisation topologique, modélisation géométrique, cartes généralisées, transformation de graphes, étude de performances, Jerboa

1. Introduction

Depuis plusieurs décennies la modélisation à base topologique a montré qu'elle permet de développer plus rapidement des opérations plus justes [DL14]. En effet, séparer la structure topologique d'un objet (sa décomposition en sommets, arrêtes, faces, etc.) de sa forme géométrique, permet le plus souvent de réduire le nombre de cas à considérer et de développer des opérations locales. Les structures topologiques combinatoires (comme les cartes [BS85], ou les cartes généralisées [Lie94])disposent de critères mathématiques de bonne formation topologique des objets, ce qui facilite la mise au point des opérations et de leurs algorithmes de mise en œuvre.

La bibliothèque Jerboa [BPA*10, BALGB14] et le langage associé [PACLG08, BALG11] permet d'aller plus loin. Grâce à des règles de transformation de graphes [EEPT06] les opérations géométriques à base topologiques peuvent être définies de manière claire et concise. Elles permettent aussi de garantir plusieurs critères syntaxique de cohérence au moment de leur édition afin d'augmenter la fiabilité de l'opération lors de son exécution. Le développement d'un modeleur à base de règles est donc particulièrement rapide et ce pour n'importe quels dimensions et n'importe quels plongements [BALGB14, BA14].

Jusqu'à peu, la principale limite de cette approche à base de règle était celle de la performance, en terme de temps d'exécution, mais surtout en terme de taille des objets mani-

pulés. Nous présentons dans cet article une nouvelle version de la bibliothèque Jerboa, développée en C++, qui permet de résoudre ce problème.

Après une présentation du langage de règles en section 2, nous allons présenter la bibliothèque logicielle Jerboa et un éventail d'opérations qui permet d'illustrer les possibilités de la bibliothèque en section 3. La section 4 est consacrée à l'évaluation de la nouvelle version C++ de Jerboa et sa comparaison à d'autres plateformes logicielles de développement d'opérations géométriques à base topologique.

2. Le langage

La bibliothèque Jerboa repose sur un langage qui permet de paramétrer entièrement un noyau de modélisation généré. Il comprend une représentation des objets à base topologique générique en dimensions et en plongements, et des règles basées sur les transformations de graphes REF.

2.1. Une structure topologique générique en dimension

La structure topologique des objets est représentée à l'aide de cartes généralisées (ou G-cartes) [Lie94]. Ces dernières ont l'avantage de pouvoir être définies par des graphes homogènes en dimension. Ce qui permet de spécifier les opérations de modification des objets à l'aide de règles de transformations de graphes uniformes [PACLG08]. Surtout les G-cartes définissent la cohérence topologique des objets à l'aide de contraintes. Nous verrons que les règles de transformation des objets, peuvent préserver ces contraintes de cohérence.

La représentation d'un objet à l'aide d'une G-carte est intuitivement issue de sa décomposition successive en cellules topologiques (sommets, arrêtes, faces, volumes, etc.). Par exemple, l'objet topologique de la Figure 1(a) peut être décomposé en G-carte de dimension 2. Tout d'abord, l'objet est décomposé en faces, Figure 1(b), qui sont liées le long de leurs arrêtes communes par des liens α_2. L'indice 2 indique que deux cellules de dimension 2 (faces) partagent une arrête. De la même manière, les faces sont découpées en arrêtes reliées par α_1, Figure 1(c). Finalement, les arrêtes sont découpées en sommets reliés par α_0 pour obtenir la 2-G-carte, Figure 1(d). Les sommets ainsi obtenus sont appelés *brins*. Chaque brin est un sommet, vu d'une face et vu d'une arrête. Une G-carte peut donc être définie comme un graphe, dont les nœuds sont les brins et les arcs sont les liaisons α_i et sont étiquetés par l'indice i. Ainsi une 2-G-carte a ses arcs étiquetés par $\{0,1,2\}$. Et plus généralement une n-G-carte est un graphe particulier dont les arcs sont étiquetés sur l'intervale $[0,n]$.

Notamment, les G-cartes sont des graphes non orientés, comme l'illustre la Figure 1(d). Formellement, les arcs non orientés sont des paires d'arcs avec l'orientation opposée et la même étiquette. Pour rendre les figures plus lisibles,

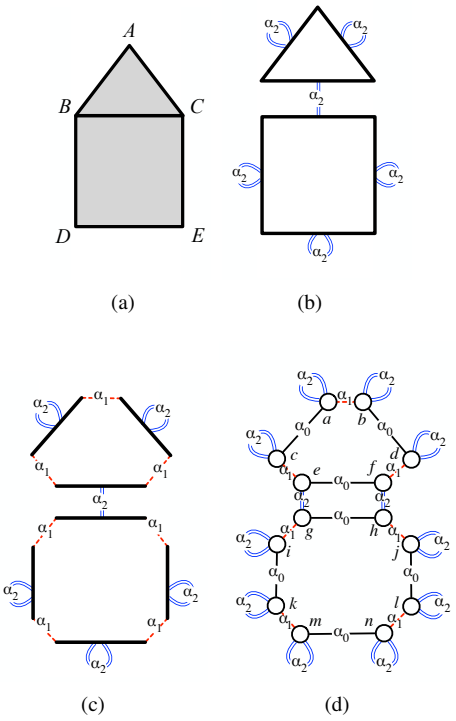
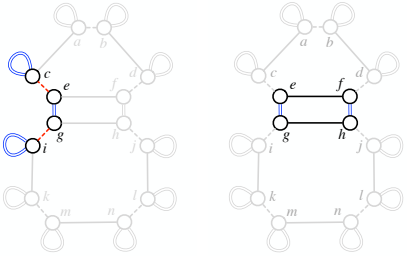


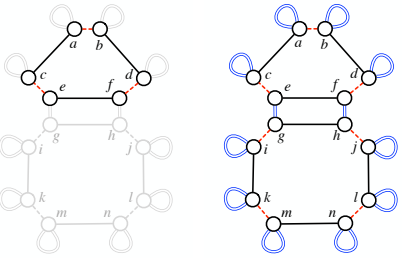
Figure 1: Décomposition d'un objet géométrique 2D

nous allons adopter le code graphique introduit Figure 1(d) : une ligne noir pour les liaisons α_0, une ligne pointillé rouge pour les α_1, une double ligne bleue pour les α_2, et une ligne discontinue verte pour les α_3. L'ensemble des figures qui suivent utiliseront ce code graphique, en omettant le nom des liaisons.

Dans les G-cartes, les cellules topologiques sont représentées implicitement à l'aide de sous-graphes, qui peuvent être calculés par parcours à partir d'un brin d'origine et d'un ensemble d'étiquettes. Par exemple, Figure 2(a), la 0-cellule adjacente à e (le sommet associé au brin e) est le sous-graphe qui contient e, les nœuds atteignables à partir de e en parcourant les arcs α_1 et α_2 (i.e. les nœuds c, e, g, et i) et les arcs eux-mêmes. Cette cellule est notée $G\langle\alpha_1\alpha_2\rangle(e)$, ou simplement $\langle\alpha_1\alpha_2\rangle(e)$. Elle modélise le sommet B de l'objet Figure 1(a). Figure 2(b), la 1-cellule adjacente à e (l'arrête associée à e) est le sous-graphe $G\langle\alpha_0\alpha_2\rangle(e)$ qui contient le brin e, tous les brins atteignables par les liaisons α_0 et α_2 (les brins e, f, g et h) et les liaisons elle-mêmes. Elle modélise l'arrête BC. Finalement, Figure 2(c), la 2-cellule adjacente à e (la face associée à e) est le sous-graphe $\langle\alpha_0\alpha_1\rangle(e)$ construit à partir du brin e et les liens α_0 et α_1, et représente la face ABC. En fait, les cellules topologiques sont des cas particuliers des *orbites* qui sont modélisés par les sous-graphes construits à partir d'un brin de départ et d'un ensemble d'éti-



(a) Sommet (b) Arrête



(c) Face (d) Composante connexe

Figure 2: Reconstruction d'orbites adjacentes à e

quettes. Par exemple, l'orbite $\langle \alpha_0\alpha_1\alpha_2 \rangle(e)$ de la Figure 2(d), représente la composante connexe incidente à e.

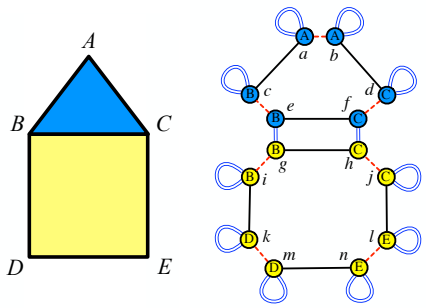
Un graphe G avec arcs étiquetés sur $[0,n]$ est une n-G-carte, si elle satisfait les contraintes de cohérence topologiques suivantes :

— *non orientation :* G est non orienté ;
— *arcs adjacents :* chaque nœud (brin) de G est la source d'exactement $n+1$ arcs respectivement étiquetés de α_0 à α_n ;
— *cycle :* pour tous indices i et j tel que $0 \leq i \leq i+2 \leq j \leq n$, il existe un cycle étiqueté par $ijij$ dans G à partir de chacun de ses nœuds (brins).

Les contraintes ci-dessus garantissent que les objets représentés par des G-cartes sont des quasi-variétés [Lie91]. En particulier, la contrainte de cycle garanti que, dans une G-carte, deux i-cellules ne peuvent être liées que le long d'une $(i-1)$-cellule. Par exemple, dans la G-carte de la Figure 1(d), la contrainte de cycle $\alpha_0\alpha_2\alpha_0\alpha_2$ impose que les faces soient liées le long des arrêtes. Notons que ces trois contraintes restent vérifiées au bord de l'objet grâce aux boucles (par exemple les boucles α_2 Figure 1(d)).

2.2. Les plongements utilisateur et leurs cohérences

Pour modéliser un objet complet, la structure topologique des G-cartes doit être complétée par différents plongements géométriques et physiques qui dépendent de l'application ciblée. Formellement, les plongements sont définis par des



(a) (b)

Figure 3: Représentation d'un objet 2D

étiquettes sur les nœuds. Plus précisément, chaque plongement à sa propre structure de donnée (une classe) et est défini pour un type particulier d'orbite. Par exemple, l'objet Figure 3(a) a deux plongements : les points géométriques 2D attachés aux sommets, et les couleurs RGB associées aux faces. Ainsi, dans la G-carte plongée Figure 3(b), chaque brin porte deux étiquettes, l'une qui précise le point géométrique, l'autre la couleur. Par exemple, le brin e porte le point B et la couleur bleu.

Une G-carte plongée porte donc un ensemble Π *d'opérations de plongement* $\pi : \langle o \rangle \rightarrow \tau$, où π est le nom du plongement, $\langle o \rangle$ est le type de l'orbite de rattachement du plongement, et τ est le type du plongement. Par exemple, pour l'objet Figure 3, l'opération de plongement *point* : $\langle \alpha_1\alpha 2 \rangle \rightarrow$ *point_2D* associe les instances A,B,C,D,E de la classe *point_2D* aux sommets, et *color* : $\langle \alpha_0\alpha_1 \rangle \rightarrow$ *color_RGB* associe des instances de la classe *color_RGB* aux faces.

Une G-carte plongée G doit vérifier la *contrainte de plongement* suivante [BALG11] : pour chacune de ses opérations de plongement $\pi : \langle o \rangle \rightarrow \tau$ de Π, tous les brins d'une même $\langle o \rangle$-orbite portent la même valeur pour le plongement π.

2.3. Développement des opérations topologiques via des règles de transformation de graphe

Comme nous l'avons déjà dis, l'originalité de Jerboa par rapport aux principales bibliothèques de modélisation géométriques à base topologique, est son langage de règles qui permet très rapidement de définir/programmer des opérations topologiques et géométriques. Ce langage est basé sur les transformations de graphes [EEPT06], et plus précisément sur les transformation de graphes partiellement étiquetés [HP02], qui permettent la modification des étiquettes de plongement, et l'utilisation de variables inspirées de [Hof05].

Pour permettre de définir des opérations pour des orbites

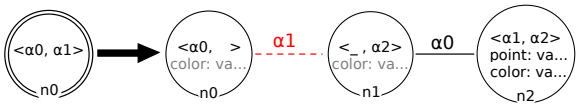
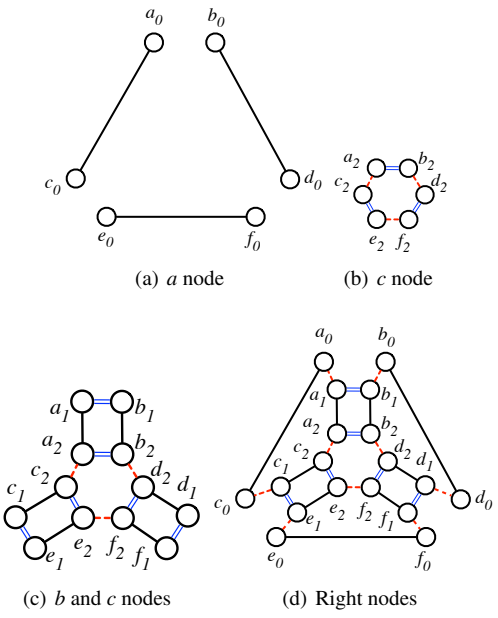


Figure 4: Règle de triangulation d'une face

de taille et de forme variable, nous utilisons des *variables topologiques* [PACLG08]. Ces dernières sont dénotées par le type de l'orbite et sont instanciées par des orbites de ce type. Par exemple, Figure 10, le nœud gauche de la règle est étiqueté par type d'orbite $\langle \alpha_0 \alpha_1 \rangle$, et peut donc filtrer n'importe quelle face d'un objet. Son instanciation par le brin a (resp. g) de la G-carte Figure 3(b) spécialise la règle pour une face triangulaire (resp. quadrangulaire).



(a) a node (b) c node

(c) b and c nodes (d) Right nodes

Figure 5: Topological instantiation

Tous les nœuds de droite de la règle portent un type d'orbite de la même longueur, mais ils varient par suppression et/ou réétiquetage des arcs. Plus précisément le caractère '_' permet de supprimer des arcs. Ainsi, par exemple, le noeud `n0` de droite est étiqueté par $\langle \alpha_0 \_ \rangle$, il est donc instancié par l'orbite face préalablement filtré, dont toutes les liaisons α_1 sont supprimées. Ce qui appliqué au triangle de la Figure 3(b), donne les trois arrêtes décousues Figure 5(a). Les arcs peuvent aussi être renommés. Par exemple, le nœud `n2` porte l'étiquette $\langle \alpha_1 \alpha_2 \rangle$, ce qui renomme les liaisons α_0 en α_1 et les liaisons α_1 en α_2. L'application de la règle de triangulation sur la face triangulaire de la Figure 3(b), instancie donc le nœud `n2` par le sommet dual de la face de départ représenté Figure 5(b). Une fois tous les nœuds de droite instanciés par des orbites, elles sont reliées deux à deux conformément aux liaisons explicites de la règle. Par exemple, l'instanciation de la liaison α_0 entre les nœuds `b` et `c` α_0-lie deux à deux les brins instances de `b` avec ceux instance de `c` pour produire l'objet Figure 5(c). Finalement, l'instanciation de la règle de triangulation Figure 10 sur la face triangulaire de la Figure 3(b) produit le motif de la Figure 5(d) qui, après exécution de la règle, remplacera le triangle d'origine par trois facettes triangulaires.

Des conditions syntaxiques sur les règles permettent de garantir la préservation de la cohérence topologique des objets [BALG11]. Par exemple, une règle préserve la contrainte d'arc incident, si ses nœuds supprimés (les nœuds qui sont uniquement à gauche) et ses nœuds ajoutés (qui sont seulement à droite) ont toutes leurs liaisons (α_0 à α_n), et si ses nœuds préservés (qui apparaissent à gauche et à droite) ont les mêmes types de liaisons à gauche et à droite. Par exemple, la règle de la Figure 10 a son nœud ajouté `n1` qui a bien ses trois liaisons : α_0 et α_1 explicitement et la liaison α_2 implicitement dans son étiquette topologique. Le nœud préservé `n0` de la règle n'a pas toutes ses liaisons, mais se sont les mêmes étiquettes à gauche et à droite de la règle. A gauche, `n0` a ses deux liaisons α_0 et α_1 implicites dans l'étiquette topologique, tandis qu'à droite α_0 est implicite, mais α_1 est explicite. Les liaisons α_2 des brins filtrés par `n0` ne sont pas filtrées par la règles, elles permettent de rattacher les motifs au reste de l'objet (au départ la face à trianguler, puis après exécution de la règle les facettes triangulaires).

2.4. Calcul des valeurs de plongements

Nous venons de voir comment les règles permettent de modifier la structure topologique des objets. Nous allons maintenant présenter comment modifier les plongements à l'aide d'expressions de plongement. Ces expressions utilisent comme *variables de plongement* les nœuds de gauche. Par exemple, la règle Figure 10 admet une seule variable `n0` qui permet d'accéder aux brins filtrés par le nœud `n0`. La bibliothèque Jerboa fourni l'accès aux plongements. Ainsi par exemple, `n0.point` permet d'accéder au plongement *point* du brin filtré par le nœud `n0`. Lors de l'instanciation de la règle, Figure 5, la variable `n0` prend la valeur du brin courant et l'expression `n0.point` celle de son plongement. Ainsi, l'instanciation de l'expression `n0.point` pour le brin a_0, est l'expression `a0.point` qui a pour valeur la point géométrique A du brin a_0.

En outre, la bibliothèque fourni des fonctions d'accès aux brins voisins. Par exemple, l'expression `n0.`α_2 permet d'accéder au brin de la face voisine, et donc l'expression `n0.`α_2`.color` accède à la couleur de la face voisine. La bibliothèque fourni aussi des fonctions de collecte, qui permettent de collecter les plongements d'une orbite. Par exemple, `point`$_{\langle \alpha_0 \alpha_1 \rangle}$`(n0)` collecte les plongements *point* de la face incidente au brin filtré. Si l'on reprend l'exemple d'instanciation précédent Figure 5, l'expression instanciée pour le brin a_0 aura pour valeur la collection des points de

la face incidente au brin a_0, c'est-à-dire la collection des trois points A, B, et C. Notons que chaque plongement n'est recueilli qu'une seule fois par orbite de plongement, ainsi chaque point n'est recueilli qu'une seule fois, même s'il possède deux brins supports dans la face.

Enfin, les expressions de plongement utilisent les méthodes fournies par les classes de plongement. Par exemple, si la classe `point_2D` fourni une méthode `middle` qui prend en argument une collection de points 2D et calcul le barycentre, alors l'expression `middle(point`$_{\langle\alpha_0\alpha_1\rangle}$`(n0))` permet de calculer le centre de la face filtrée par le nœud `n0`. Ceci est l'expression portée par le nœud `n2` de la règle Figure 10. Lors de son instanciation sur le brin a_2, elle sera instanciée par l'expression `middle(point`$_{\langle\alpha_0\alpha_1\rangle}$`(a0))` qui calculera le centre de la face. Notons que comme toutes les instances a_0 à f_0 du nœud `n0` ont la même face (l'orbite $\langle\alpha_0\alpha_1\rangle$), alors les six expressions instanciées pour les brins a_2 à f_2 calculeront bien la même valeur.

Les règles sont munies de conditions syntaxiques nécessaires à la préservation de la cohérence des plongements [BALG11]. Notamment, toute orbite de plongement a au plus une expression de plongement. Par exemple, la règle Figure 10 a une expression de plongement `point` sur son nœud `n2` qui défini à lui seul le nouveau sommet central, mais n'a pas d'expression de plongement `point` sur le nœud `n0`, car les points des sommets d'origine ne sont pas modifiés, ni d'expression de plongement `point` sur son nœud `n1` qui appartient à la même orbite sommet que `n0` (puisqu'il est relié par α_1). De même, le nœud `n2` porte l'expression de plongement `color` qui est reporté (en grisé) sur les nœuds `n0` et `n1` qui appartiennent à la même orbite face, et donc portent nécessairement la même couleur.

2.5. L'éditeur de modeleur

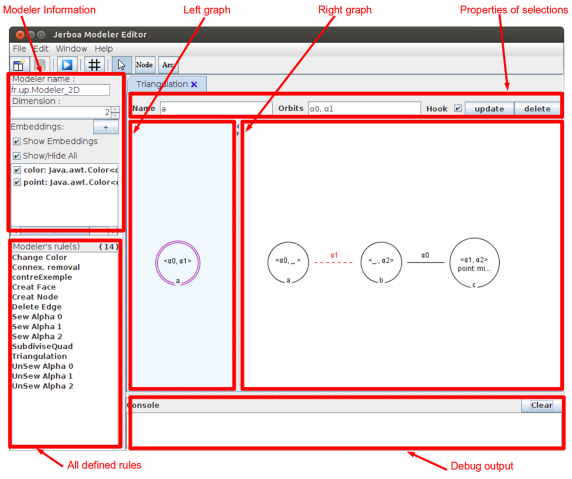


Figure 6: JerboaModelerEditor

La bibliothèque Jerboa est fourni avec un éditeur graphique qui utilise le langage précédent pour paramétrer la génération du modeleur souhaité. Comme le montre la copie d'écran Figure 6, l'utilisateur utilise le cadre en haut à gauche pour définir les paramètres généraux de son modeleur : son nom, sa dimension topologique, et ses différents plongements. Puis il édite les différentes opérations de son modeleur sous la forme de règles. Le cadre en bas à gauche liste les règles déjà créées et les fenêtres centrales permettent d'éditer la règle courante.

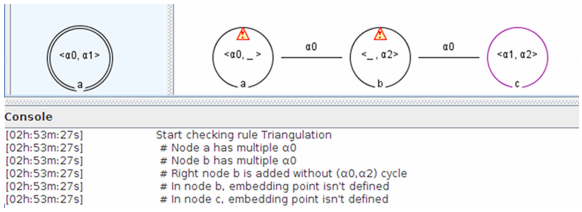


Figure 7: Erreurs détectées par l'éditeur

Au fur et à mesure de l'édition d'une règle, les conditions syntaxiques sont vérifiées et les erreurs topologiques, puis géométriques sont affichées dans la console comme le montre la Figure 7. Ce retour facilite l'apprentissage du langage et accélère largement la mise au point des opérations.

L'éditeur de Jerboa est commun aux deux noyaux de Jerboa : le noyau d'origine en Java [BALGB14] et celui en C++ que nous présentons dans cet article. Il a seulement été étendu pour permettre la génération de code pour les deux noyaux. Cela est rendu possible par un langage largement commun. Seules les classes de plongement et les expressions de plongement doivent être adaptées à chacun des deux langages de programmation cibles. En particulier, les conditions syntaxiques de préservation de la cohérence topologique et de plongement des règles sont strictement identiques.

3. Noyau et opérations en Jerboa

Depuis la sortie de la version Java en 2014 [BALGB14], Jerboa a été utilisé dans plusieurs projets académiques afin de vérifier la généricité des transformations de graphes pour le prototypage rapide d'opérations complexes. Cependant, le passage à l'échelle sur de très gros modèles (plusieurs dizaines de millions de brins) ont montré les limites de la JVM et nous avons constaté que l'application passait plus de temps à tenter de libérer la mémoire qu'à exécuter l'opération (cf. section 4).

Le temps d'exécution devenant trop lent, nous avons dû opter pour un portage de notre architecture en C++ que nous présentons dans cet article. De plus, il était important pour nous de vérifier si notre approche et notre architecture logicielle pouvaient être portables vers d'autres langages objets utilisés dans la communauté informatique graphique.

3.1. Architecture logicielle du noyau

La volonté première de Jerboa est d'éviter au développeur la connaissance de détails d'implantations. Ainsi, la connaissance des G-Cartes, des concepts et des propriétés sous-jacentes sont suffisantes pour développer des opérations simples ou complexes de manière rapide et fiable. Le premier prototype de Jerboa se focalise donc sur la facilité à expérimenter de nouvelles opérations au travers de la transformation de graphes.

Par ailleurs, l'éditeur de règle permet de générer une implantation suivant les besoins de l'utilisateur. Actuellement, notre éditeur permet de produire des modeleurs spécifiques en Java et en C++, suivant les besoins de l'utilisateur final. Cette simplicité est possible grâce à une architecture logicielle générique inspirée du *génie logiciel* [AC96].

En effet, nous avons pu identifier plusieurs usages des structures topologiques et des besoins particuliers sur chacun induisant des choix d'implantations spécifiques suivant les outils alors que les opérations ou propriétés restent les même, par exemple la non-duplication des valeurs de plongements dans le but d'optimiser la consommation mémoire (très utile dans le domaine de la géologie [PR13]), ou au contraire le partage de valeurs au sein des orbites supports [KUJ*13].

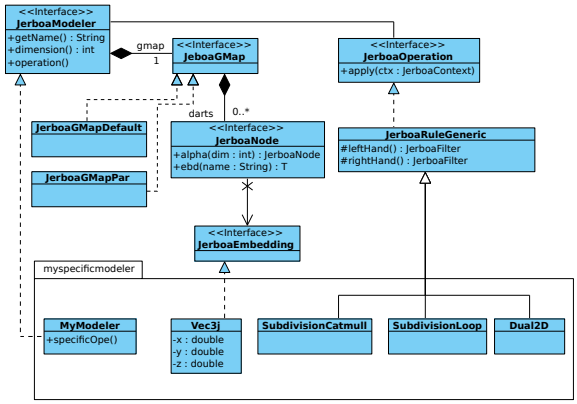


Figure 8: Architecture de Jerboa

La partie supérieure de la Figure 8 présente l'architecture du noyau de Jerboa. Elle fournit plusieurs interfaces/classes abstraites, assurant une grande flexibilité d'architecture. Cette architecture est identique entre les différents langages supportés par Jerboa. Le noyau fournit une implantation par défaut qui optimise les accès aux valeurs de plongements (`JerboaGMapDef`), et qui fournit le moteur d'application des règles (`JerboaRuleGeneric`) réalisant le traitement de toutes les opérations. Cependant, il serait possible d'ajouter facilement une implantation (`JerboaGMapPar`) supportant les accès concurrents ou proposant une version parallélisée du moteur d'application de règles.

Ainsi, l'éditeur de règle produit pour un modeleur, un ensemble de classes (partie inférieure de la Figure 8) pour un langage donné, en héritant les interfaces/classes du noyau. Cet ensemble se compose essentiellement de l'ensemble des règles (concrètes) définies à l'aide de l'éditeur (`Dual2D`, `SubdivisionCatmull`, `SubdivisionLoop`, ...) et du modeleur spécifique regroupant l'ensemble de ces informations (`MyModeler`). Il reste donc les classes de plongements développées par l'utilisateur en respectant un lien d'héritage pour intégrer simplement ses données (comme le montre la classe `Vec3`, qui représente les points géométriques 3D du modeleur).

Il est à noter que, malgré l'absence d'un *ramasse-miettes*, l'implantation C++ offre la gestion mémoire des valeurs de plongement. En effet, toujours dans le but de limiter l'intervention de l'utilisateur, la classe de base `JerboaEmbedding` s'occupe de libérer automatiquement la ressource lorsque celle-ci n'est plus utilisée par la G-carte. Dans notre implantation actuelle, nous avons vérifié que les objets étaient bien désalloués à l'aide de l'outil *valgrind* sur différents exemples de modeleur spécifique.

3.2. Exemples d'opérations en 3D

Cette section présente différentes opérations issues de la modélisation topologique et géométrique, converties en règle de transformation de graphes. Ces différentes opérations classiques nous ont permis de comparer la version C++ de Jerboa, nommée dans la suite du texte *Jerboa++*, aux autres outils existants mais également à la version Java de Jerboa, que nous nommerons *JJerboa*. Pour illustrer ces différentes opérations, nous considérons dans la suite de l'article un modeleur 3D possédant le plongement des points géométrique sur l'orbite support $\langle \alpha 1, \alpha 2, \alpha 3 \rangle$. Enfin, nous nous sommes focalisés sur l'écriture d'opérations utilisables en une seule règle Jerboa (nommée règle atomique), c'est-à-dire les opérations travaillant sur une seule *orbite filtrée*.

3.2.1. Triangulation de faces

La triangulation barycentrique d'une face est une opération standard, qui consiste à subdiviser une face en triangles en ajoutant des arêtes reliant les sommets au barycentre de la face. Cette opération comporte donc un seul calcul de plongement (la position géométrique du barycentre), et la création d'autant de faces que du nombre d'arêtes la face d'origine. Dans le langage de règles de Jerboa, cette opération est illustrée en Figure 10. Le résultat de l'application de la triangulation barycentrique s'observe en Figure 9.

3.2.2. Éponge de Menger

L'éponge de Menger est une opération fractale de dimension trois, qui subdivise un volume donné en un plusieurs petits volumes auxquels les volumes centraux (des faces et du volume) sont retirés. Comme le montre la Figure 11, un

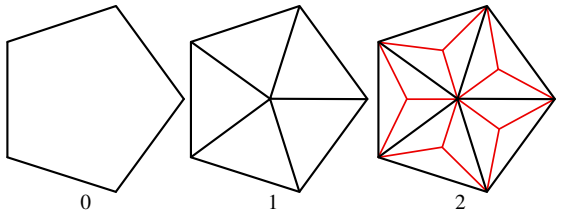


Figure 9: Triangulation barycentrique.

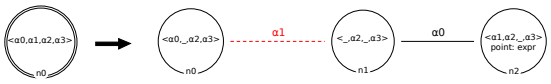


Figure 10: Règle de la triangulation de toutes les faces de la composante connexe.

cube est subdivisé en 27 petits cubes, où on retire les 6 cubes au centre des faces originelles et le cube central au volume, à la fin de l'opération il reste donc 20 petits cubes. Cette opération peut être répétée alors sur l'ensemble des nouveaux volumes augmentant rapidement le nombre de cubes (pour la i^e répétition, il y a 20^i petits cubes au total).

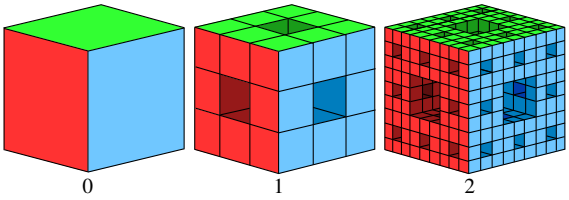


Figure 11: Éponge de Menger .

La règle Jerboa associée à cette opération repose sur les *régularités de l'opération* pour chaque brin existant. Ainsi, la règle consiste en la création des nouveaux brins et des liaisons entre eux afin de former les nouveaux petits cubes comme illustrée sur la Figure 12. Cette règle permet de tester la création de nouveaux brins au sein de notre structure dans la section 4.

3.2.3. Suppression d'une composante connexe

La suppression d'une composante connexe s'écrit par une règle ne possédant pas de membre droit (c'est-à-dire où tous les brins seront supprimés) (*cf.* Figure 13). Cette opération nous permet de tester la rapidité de suppression des brins d'une G-carte.

3.2.4. Translation

La translation d'une composante connexe est une règle ne modifiant pas la topologie de l'objet, elle se contente de

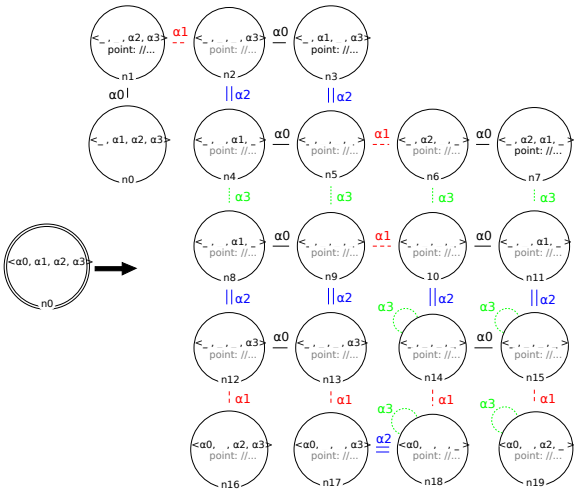


Figure 12: Règle de l'éponge de Menger.

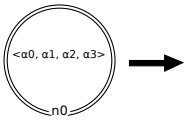


Figure 13: Règle de suppression d'une composante connexe.

mettre à jour les valeurs de plongements pour chaque sommet de l'objet. Elle nous permet de tester, la rapidité de calcul des plongements sans modification de la topologie. La Figure 14 présente la règle de translation d'une composante connexe. Le membre gauche et le membre droit possède un seul nœud avec les mêmes caractéristiques topologiques. Cependant le nœud de droite possède une ligne en plus, qui redéfinie le plongement `point`.

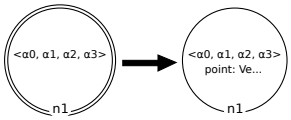


Figure 14: Règle de translation d'une composante connexe.

3.2.5. Subdivision de Loop

La subdivision de Loop [Loo87] est un schéma de subdivision pour tout polygone dont les faces ont une valence inférieure ou égale à trois. Le principe est de créer un nouveau sommet au milieu de chaque arête, et de relier ces nouveaux sommets entre eux pour découper la face originelle en plusieurs petits triangles (*cf.* Figure 15).

Cette méthode de subdivision permet de lisser des objets 3D. Pour cela, la position des points existants et des nouveaux points n'est plus un simple calcul de milieu mais un

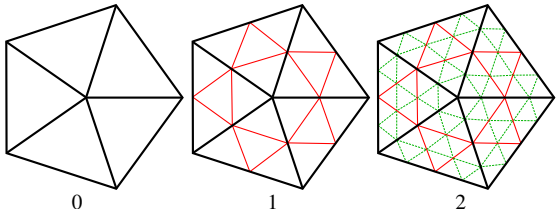


Figure 15: Subdivision de Loop.

barycentre de son voisinage avec des coefficients particuliers comme le montre la Figure 16 [Loo87].

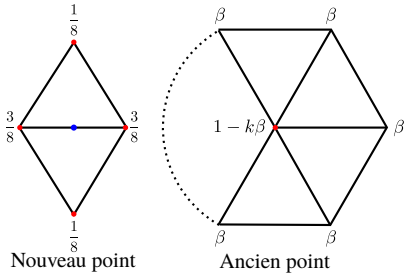


Figure 16: Masques de calcul des positions points.

La Figure 17 présente la règle Jerboa de la subdivision de Loop lissée. Le membre gauche permet de filtrer un volume isolé et de créer la subdivision souhaitée. Ensuite, le calcul des positions se fait grâce aux combinaisons de la Figure 16, à savoir le nœud `n0` représente les brins existants et le nœud `n3` représente les nouveaux brins. Il est à noter que les nœuds `n1` et `n2` font partie de la même orbite que le nœud `n3` et donc Jerboa détecte automatiquement la propagation du résultat à ces nœuds afin d'assurer la cohérence des valeurs de plongements sur l'orbite support.

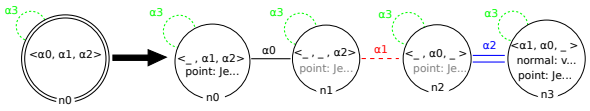


Figure 17: Règle pour la subdivision de Loop.

3.2.6. Subdivision de Catmull-Clark

L'opération de subdivision de Catmull-Clark [CC78] est une opération qui subdivise et lisse un objet surfacique isolé. Cette opération découpe chaque arête de la face, et crée un point central à cette face. Topologiquement, les sommets ajoutés sur les arêtes sont tous reliés au sommet central de la face d'origine. Il en résulte une surface où toutes les faces sont de valence quatre (uniquement composée de quadrangles) comme illustré en Figure 18.

La règle obtenue en Jerboa est donnée en figure 19. Ici, les

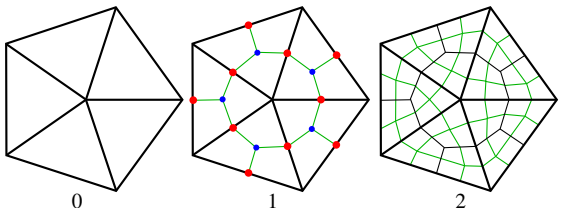


Figure 18: Subdivision de Catmull-Clark

nœuds `n1` représentent les nœuds existants, les nœuds `n3` et `n4` représentent le nouveau sommet au centre de l'arête et `n2` représente le nouveau sommet au centre de la face.

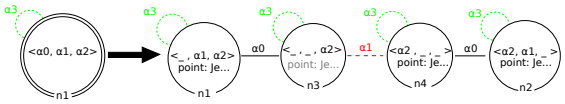


Figure 19: Règle pour la subdivision de Catmull-Clark.

Géométriquement, le calcul des plongements concerne la position des points selon les brins. La coordonnée des brins du nœud `n2` est le centre de tous les sommets de la face (obtenu par le parcours d'orbite $\langle \alpha 0, \alpha 1 \rangle$). Nous notons le calcul du barycentre par la fonction ϕ, l'équation est donc la suivante :

$$n2.point = \phi(point_{\langle \alpha 0, \alpha 1 \rangle}(n1)) \quad (1)$$

La position des points de `n4` (ou `n3`) se calcule en fonction des deux faces incidentes à l'arête et des deux extrémités comme illustrée dans l'équation suivante :

$$\begin{aligned} n4.point &= \frac{\frac{n1.point + n1.\alpha 0.point}{2} + FP_0 + FP1}{3}, \\ \text{avec } FP_0 &= \phi(point_{\langle \alpha 0, \alpha 1 \rangle}(n1)), \\ \text{et } FP_1 &= \phi(point_{\langle \alpha 0, \alpha 1 \rangle}(n1.\alpha 2)) \end{aligned}$$

Ainsi, nous illustrons l'intérêt de la structure topologique dans le calcul des valeurs de plongements et les différentes fonctions de parcours, possible dans Jerboa.

Les sommets existants (nœud `n1` de la figure 19) doivent être modifiés pour lisser le maillage. Elle se calcule en fonction des faces et arêtes incidentes au brin, selon la formule suivante (où k est le nombre d'arêtes incidentes) :

$$\begin{aligned} n1.point &= \frac{F + 2R + (k-3)n1.point}{k}, \\ \text{avec } R &= \sum_{n \in \langle \alpha 1, \alpha 2 \rangle (n1)} \frac{n.\alpha 0.point}{2k} \\ \text{et } F &= \phi(\forall n \in \langle \alpha 1, \alpha 2 \rangle (n1), \phi(point_{\langle \alpha 0, \alpha 1 \rangle}(n))) \end{aligned}$$

3.2.7. Dual 2D

L'opération de dual 2D [Kra08] s'applique aux surfaces. Cette opération transforme le maillage de manière à ce que chaque face du maillage initial devienne un point, et que chaque arête commune entre deux anciennes faces, devienne une arête reliant les deux nouveaux sommets créés au centre de chaque face (la règles Figure 20 illustre le renommage adéquate des liaisons).

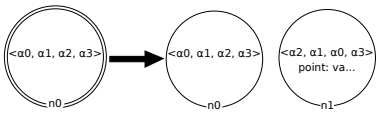


Figure 20: Règle réalisant le Dual 2D.

La Figure 21 présente deux applications successives du dual 2D à partir d'une face carré triangulée. Les applications successives montrent que la version courante pose un problème sur les bords du maillage. En effet les faces en bordure sont remplacées par leur barycentre, et les positions exactes des sommets de ces anciennes faces sont perdues. Cela implique l'impossibilité de recalculer les points qui sont au bord dans le maillage initial. En conclusion, si topologiquement cette opération est sa propre inverse, ce n'est pas le cas géométriquement.

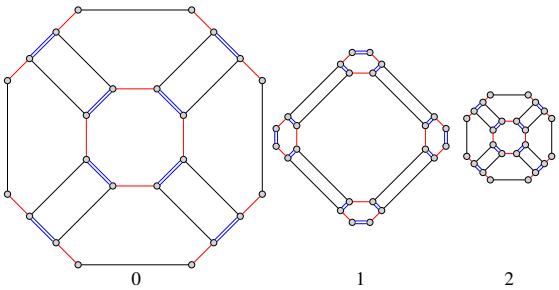


Figure 21: Applications successives du Dual 2D.

3.2.8. Dual 3D

L'opération de dual 3D transforme le maillage de manière à ce que chaque volume du maillage initial devienne un point et que chaque faces communes entre deux anciens volumes, devienne une arête reliant les deux nouveaux sommets créés au centre de chaque volume (*cf.* Figure 22).

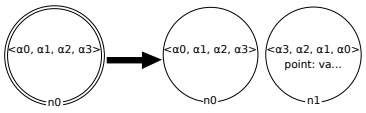


Figure 22: Règle réalisant le Dual 3D.

La Figure 23, présente un exemple de maillage sur lequel on applique l'opération de dual 3D.

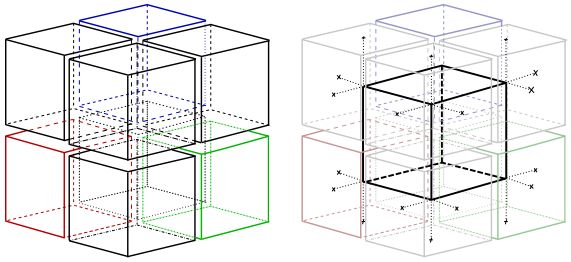


Figure 23: Illustration du calcul du dual 3D.

4. Évaluation de Jerboa

Cette section propose une comparaison des temps d'exécutions de différents outils sur quelques opérations illustrées dans la sous-section 3.2. Une étude plus approfondie sur la comparaison entre les implantations JJerboa (Java) et Jerboa++ (C++) est aussi proposée.

4.1. Présentation des autres outils

Moka [DL14,Mok] se compose d'une bibliothèque offrant la structure de donnée d'une carte généralisée de dimension trois (3-G-carte) et des opérations de bases sur cette dernière. L'implantation principale de cette 3-G-carte propose un unique plongement de point géométrique sur les cellules sommets (plus précisément porté par un unique brin de l'orbite sommet). Elle inclut aussi plusieurs sous-modules, qui enrichissent les opérations possibles. Ce sont ces derniers que nous avons exploité.

La bibliothèque *CGoGN* [KUJ*13] fournit une implantation des cartes combinatoires, des cartes généralisées et d'autres structures à base de brins, toutes reposant sur une structure indexée, ce qui lui donne un haut niveau de performance dans son utilisation. Cependant, les algorithmes sont propres à chaque modèle. Nous avons donc décidé d'utiliser pour notre expérience les cartes combinatoires, car la bibliothèque d'exemples était plus variées et correspondait à nos objectifs.

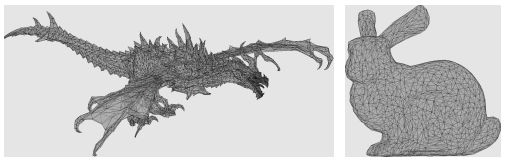
L'outil *Computational Geometry Algorithms Library* (CGAL) [CGA] est une bibliothèque complète offrant plusieurs structures de données et plusieurs opérations simples et complexes optimisées. Il inclut la classe `Polyhedron`, qui repose sur la structure topologique des arêtes ailées pour des objets 3D [**?**]. C'est cette classe que nous utilisons dans notre expérimentation avec les différents exemples fournis avec l'archive.

L'outil *OpenFlipper* [MKK13] se présente comme une interface graphique pour visualiser des objets 3D mêlant structure topologique, modélisation géométrique et différents algorithmes de rendu réaliste. La structure topologique des objets repose sur la bibliothèque *OpenMesh* [BSB*02], qui implante de manière efficace les arêtes ailées.

Les opérations choisies pour l'expérimentation sont communes à tous les outils ci-dessus, nous nous sommes inspirés en particulier des exemples fournis qui paraissaient optimaux. L'expérimentation est réalisée sur un Intel Core i7-3930K avec 32Go et OpenJDK 7 et les temps réfèrent uniquement celle de l'opération sans les temps de chargement ou de préparation des données. Nous avons sélectionné des maillages de tailles radicalement différentes. Cependant, nous avons été confrontés à un problème avec la préparation de ces derniers. En effet, le format d'échange étant un format surfacique générique, nous avons dû, grâce à Jerboa, fournir un mécanisme de sérialisation pour assurer la cohérence d'orientation propres aux cartes combinatoires lors du chargement des maillages.

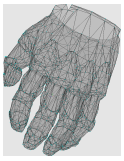
4.2. Comparaisons temporelles

La figure 24 présente l'ensemble des modèles, tirés de la littérature *informatique graphique*, utilisés pour nos expérimentations. Les maillages sont variées et le nombre de sommets varie d'une dizaine à une dizaine de milliers et est donnée à titre indicatif entre parenthèse dans les titres des modèles. Nous avons aussi utilisé un modèle nommé *mini cubes*, qui s'obtient à partir d'un cube auquel nous avons appliqué un certain nombre de subdivision en 8 petits cubes afin de grossir rapidement le nombre de brins dans nos tests.

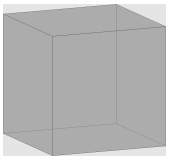


(a) Alduin (7778)

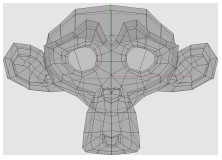
(b) Bunny (1494)



(c) Main (1626)



(d) Cube (8)



(e) Monkey (441)

Figure 24: Modèles utilisés pour les tests de performance.

Dans la plupart des tests réalisés (*cf.* Tableau 1), Jerboa++ donne des résultats dans la moyenne des autres outils. Les opérations les plus utilisées pour les tests sont les subdivisions de Catmull-Clark et de Loop, réalisant des modifications à la fois topologiques et géométriques. Sur ces opérations, Jerboa++ est plus lent que CGAL et CGoGN : jusqu'à quatorze fois pour CGAL ; entre deux et trois fois pour CGoGN. Cependant, après un certain nombre d'itérations, CGoGN n'arrive plus à réaliser les calculs (cases `NAN` dans le tableau). On peut supposer que ces erreurs peuvent être dues à un nombre trop important de sommets.

Sur ces mêmes opérations, Jerboa++ est plus efficace que MOKA ou JJerboa qui utilisent également des G-cartes : jusqu'à deux fois pour MOKA et jusqu'à cinq fois pour JJerboa. OpenMesh est également en moyenne deux fois plus lent que Jerboa++, et comme pour JJerboa cet écart s'agrandit avec la taille des modèles.

Nous avons également effectué des comparaisons de création de brins avec les règles de Dual 2D et 3D sur des G-cartes. Sur ces opérations Moka est environ deux fois plus rapide que Jerboa++. On note que le passage au C++ a réduit presque de moitié le temps de création de brins par rapport à la version Java. Il est à noter aussi que les objets obtenus par Moka sont des objets isolés et ne permettent pas d'obtenir l'objet initial après une nouvelle application.

Contrairement à sa version Java, Jerboa++ offre des temps d'exécution comparables aux autres outils existants, même si certains de ces outils (CGAL notamment) sont bien plus efficaces en temps d'exécution. Ces différences s'expliquent par le fait que nous avons un unique moteur d'application de règle générique à toutes les règles possibles (quelque soit la dimension et les plongements). Cette légère perte d'efficacité est un faible prix à payer au regard du gain en temps de développement des opérations (5 à 10 fois plus rapide).

4.3. Comparaison avancée

Cette sous-section présente une étude plus poussée entre l'implantation Java et celle en C++. Pour rappel, Jerboa possède un seul moteur d'application de règle commun à toutes les règles. Il est donc intéressant de voir les opérations engendrant des pertes de performances. Pour cela, il a été choisi de prendre l'opération de suppression d'une composante connexe, la règle de triangulation d'une face et enfin la règle de translation mettant à jour l'ensemble des plongements d'une composante connexe.

Le Tableau 2, montre les temps obtenus pour les différentes opérations réalisées. Globalement, Jerboa++ obtient des temps largement plus intéressants que JJerboa. Cependant, on peut noter que la suppression semble être plus rapide dans la version Java pour les gros objets. Une explication possible serait que le ramasse-miette de Java puisse s'exécuter en parallèle, alors que notre gestionnaire de mémoire s'exécute de manière séquentielle.

5. Conclusion et perspectives

L'approche formelle des transformations de graphes fournie un langage concis et permet de vérifier automatiquement la préservation de la cohérence des objets transformés. Ainsi, le langage de Jerboa et son éditeur graphique permet de développer très rapidement des opérations géométriques à base topologique, comme nous l'avons montré sur des opérations typiques à la sous-section 3.2.

Modèle	Opération	Itération	CGAL	CGOGN	Jerboa++	MOKA	OpenMesh	Jjerboa
Alduin	Catmull	2	138,4	806,7	2176,6		24598,5	10916,5
		4	9044,0	13680,3	38776,5		420642,7	1209119,8
	Loop	2	98,9	823,2	2052,1		24497,4	3919,6
		4	7392,3	NAN	34424,7		417229,7	161602,7
Bunny	Catmull	2	17,1	144,0	390,9		415,9	2469,5
		5	13154,9	10418,4	26882,8		27116,8	765609,8
	Loop	2	17,5	142,9	365,0		396,8	1006,7
		5	11113,9	NAN	24740,3		25447,7	119285,7
Cube	Catmull	6	15,6	154,5	246,1	448,4	365,3	1552,3
		9	1622,6	10963,6	16720,3	31092,9	23382,5	93627,2
	Loop	6	13,2	136,7	241,2		339,9	635,4
		9	1121,8	8950,7	17130,5		22031,3	36701,0
	Menger	1			1,8			17,2
		4			6075,2			8629,7
Main	Catmull	2	16,1	164,7	422,8		1303,9	2646,6
		5	6610,1	11474,9	31103,1		88950,0	822867,8
	Loop	2	12,7	170,2	399,6		1266,4	1185,5
		5	5493,6	NAN	27892,4		87348,6	122775,5
Mini cubes	Dual 2D	–			39,6	21,4		289,7
		–			3251,4	1752,7		4100,8
	Dual 3D	–			36,8	12		202
		–			3170,9	1100,8		5882,5
Monkey	Catmull	2	4,8	31,4	63,8	83,6	173,7	586,2
		5	836,1	1959,6	4840,8	6853,0	11599,8	22575,7
	Loop	2	2,3	27,0	58,8		166,6	323,0
		5	360,1	NAN	4744,9		11129,6	8995,7

Table 1: Tableau de comparaison des temps d'exécution (en ms)

Opération	Modèle	JJerboa	Jerboa++
Translation	minicubes1	10,2	0,7
	minicubes2	39,4	3,0
	minicubes3	134,6	14,0
	minicubes4	329,4	167,9
	minicubes5	1213,3	1506,7
Triangulation	minicubes1	17,5	2,2
	minicubes2	73,0	7,6
	minicubes3	273,6	42,5
	minicubes4	580,0	356,1
	minicubes5	8528,9	2827,8
Suppression	minicubes1	6,6	0,3
	minicubes2	29,4	1,8
	minicubes3	96,8	11,9
	minicubes4	182,2	144,1
	minicubes5	988,9	1208,8
Dual 2D	minicubes1	46,2	0,4
	minicubes2	182,5	3,0
	minicubes3	289,6	39,7
	minicubes4	467,7	372,4
	minicubes5	4100,8	3251,4
Dual 3D	minicubes1	14,8	0,4
	minicubes2	50,2	3,5
	minicubes3	202,0	36,8
	minicubes4	550,3	470,2
	minicubes5	5882,5	3170,9

Table 2: Temps d'exécution (en ms)

La nouvelle version C++ du noyau de Jerboa permet désormais d'approcher les performances des principaux outils de modélisation géométrique à base topologique (voir la section 4). En particulier, la taille des objets manipulables est désormais identique à celle des autres outils. La vitesse d'exécution des opérations reste légèrement inférieure, mais cet inconvénient est largement compensé par la vitesse de développement des opérations. Par ailleurs, l'utilisateur n'a pas à gérer lui-même la libération de la mémoire, car cela est fait automatiquement par Jerboa++ non seulement sur la structure topologiques des G-cartes, mais aussi sur les plongements définis par l'utilisateur.

À l'avenir nous envisageons d'améliorer davantage la rapidité d'exécution des opérations en parallélisant le moteur d'application de règles, véritable cœur de notre approche. La forme des règles de transformation de graphe rendent cette parallélisation relativement aisée. Enfin, cette dernière sera profitable à toutes les règles (pour toutes les dimensions et tous les plongements) et totalement transparente pour l'utilisateur, qui pourra continuer à exécuter ses opérations sans même changer les règles qui les définissent.

Références

[AC96] ABADI M., CARDELLI L. : *A Theory of Objects*. Monographs in Computer Science. Springer, 1996.

[BA14] BELHAOUARI H., ARNOULD A. : Jerboa : un modeleur géométrique à base de règles de transformations de graphes. In *Journées de l'Association Française d'Infor-*

matique Graphique, Reims Image 2014 (Reims, France, nov 2014).

[BALG11] BELLET T., ARNOULD A., LE GALL P. : Rule-based transformations for geometric modeling. In *6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011), Part of ETAPS 2011* (Saarbrücken, Germany, avril 2011).

[BALGB14] BELHAOUARI H., ARNOULD A., LE GALL P., BELLET T. : JERBOA : A graph transformation library for topology-based geometric modeling. In *7th International Conference on Graph Transformation (ICGT 2014)* (York, UK, juillet 2014), vol. 8571 de *LNCS*, Springer.

[BPA*10] BELLET T., POUDRET M., ARNOULD A., FUCHS L., LE GALL P. : Designing a topological modeler kernel : A rule-based approach. In *Shape Modeling International (SMI'10)* (Aix-en-Provence, France, juin 2010).

[BS85] BRYANT R., SINGERMAN D. : Foundations of the theory of maps on surfaces with boundaries. *Quart. J. Math. Oxford Ser. (2). Vol. 36*, Num. 141 (1985), 17–41.

[BSB*02] BOTSCH M., STEINBERG S., BISCHOFF S., KOBBELT L., AACHEN R. : Openmesh – a generic and efficient polygon mesh data structure. In *In OpenSG Symposium* (2002).

[CC78] CATMULL E., CLARK J. : Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design. Vol. 10*, Num. 6 (1978), 350 – 355.

[CGA] Computational geometry algorithm library (cgal). http ://www.cgal.org.

[DL14] DAMIAND G., LIENHARDT P. : *Combinatorial Maps : Efficient Data Structures for Computer Graphics and Image Processing*. A K Peters/CRC Press, September 2014.

[EEPT06] EHRIG H., EHRIG K., PRANGE U., TAENTZER G. : *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, Secaucus, NJ, USA, 2006.

[Hof05] HOFFMANN B. : Graph transformation with variables. In *Formal Methods in Software and System Modeling (Festschrift for Hartmut Ehrig on the Occasion of his 60th Birthday)* (2005), Kreowski H.-J., Montanari U., Orejas F., Rozenberg G., Taentzer G., (Eds.), vol. 3393 de *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, pp. 101–115.

[HP02] HABEL A., PLUMP D. : Relabelling in graph transformation. In *ICGT 2002 : international conference on graph transformation* (Barcelona , ESPAGNE, octobre 2002), vol. 2505 de *LNCS*, Springer, pp. 135–147.

[Kra08] KRAEMER P. : *Modèles topologiques pour la multirésolution*. PhD thesis, University Louis Pasteur - Strasbourg I, Nov 2008.

[KUJ*13] KRAEMER P., UNTEREINER L., JUND T., THERY S., CAZIER D. : Cgogn : *n*-dimensional meshes with combinatorial maps. In *Proceedings of the 22nd International Meshing Roundtable, IMR 2013, October 13-16, 2013, Orlando, FL, USA* (2013), Sarrate J., Staten M. L., (Eds.), Springer, pp. 485–503.

[Lie91] LIENHARDT P. : Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design. Vol. 23*, Num. 1 (1991).

[Lie94] LIENHARDT P. : N-dimensional generalised combinatorial maps and cellular quasimanifolds. *International Journal of Computational Geometry and Applications* (1994).

[Loo87] LOOP C. T. : *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, University of Utah, 1987.

[MKK13] MÖBIUS J., KREMER M., KOBBELT L. : Openflipper - A highly modular framework for processing and visualization of complex geometric models. In *6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems, SEARIS 2013, Orlando, FL, USA, March 17, 2013* (2013), Latoschik M. E., Reiners D., Blach R., Figueroa P., Wingrave C. A., (Eds.), IEEE, pp. 25–32.

[Mok] Moka. http ://moka-modeller.sourceforge.net/.

[PACLG08] POUDRET M., ARNOULD A., COMET J.-P., LE GALL P. : Graph transformation for topology modelling. In *4th International Conference on Graph Transformation (ICGT'08)* (Leicester, United Kingdom, septembre 2008), vol. 5214 de *LNCS*, Springer, pp. 147–161.

[PR13] PERRIN M. G., RAINAUD J.-F. : *Shared Earth Modeling : Knowledge driven solutions for building and managing subsurface 3D geological models*. Editions Technip, Paris, 2013.