Calcul parallèle de l'arbre de contour augmenté *via* une forêt d'arbres

Parallel computation of the augmented contour tree using forests of trees

Charles Gueunet, Pierre Fortin, Julien Jomier, Julien Tierny

English Abstract—This abstract is a short version of [6]. It presents an algorithm to efficiently compute the contour tree of scalar fields on tetraedral meshes on a multi-threaded, shared memomy architecture. This approach uses a value-driven partitionning to create a forest of trees in a first step. Then we show how to stitch them together efficiently to obtain the final contour tree. We demonstrate the utility of our in several data analysis and visualisation tasks.

1 Introduction

L'analyse topologique est un domaine fournissant des outils très utilisés en visualisation scientifique tels que le graphe de Reeb [12], le complexe de Morse-Smalle [7] ou encore l'arbre de contour [3]. Cependant ces algorithmes reposent généralement sur une vue globale des données et un traitement séquentiel qui les emêchent d'exploiter pleinement la puissance des architectures parallèles modernes.

Ce travail s'interesse plus particulièrement au cas de l'arbre de contour, abstraction topologique permettant entre autre l'extraction rapide et la simplification d'ensembles de niveaux [4], [13] et le suivi de zones d'intérêt [10]. Des versions parallèles du calcul de cette structure existent [2], [8], [9], mais ne permetent pas le calcul efficace de l'arbre de contour avec sa segmentation sur un maillage tétraédrique.
Ce travail présente un algorithme parallèle efficace pour le calcul de l'arbre de contour augmenté sur maillages tétraédriques et grille réguilère. Il est adapté aux machines à mémoire partagée. L'implémentation correspondante utilise VTK et OpenMP et est disponible au sein de la plateforme Open-Source: the Topology Toolkit [11].

2 Prérequis

2.1 Les données

L'entée de notre algorithme est un champ scalaire linéaire par morceaux: $f : \mathcal{M} \to \mathbb{R}$ défini sur une variétée linéaire par morceaux et simplement connexe. Nous considèrerons ici sans perte de généralité que nous sommes en dimension 3 (maillage tétraédrique).

Le champ scalaire f peut être étendu à l'ensemble de la géométrie *via* une interpolation linéaire barycentrique. On définit alors un ensemble de niveau comme étant la pré-image d'une valeur scalaire $i \in \mathbb{R}$ dans $\mathcal{M}$ par $f : f^{-1}(i) = \{p \in \mathcal{M} \mid f(p) = i\}$. Un ensemble de niveau peut être constitué de plusieurs composantes connexes qui sont appelées *contours*. La figure 1(a) nous montre un ensemble de niveaux constitué de cinq contours.

2.2 L'arbre de contour

Prenons $f^{-1}(f(p))_p$ le contour contenant p. L'*arbre de contour* tel qu'illustré en 1(b) est un complexe simplicial de dimension 1 définit comme l'espace quotient $\mathcal{C}(f) = \mathcal{M}/\sim$ par la relation d'équivalence $p_1 \sim p_2$:

$$\begin{cases} f(p_1) = f(p_2) \\ p_2 \in f^{-1}(f(p_1))_{p_1} \end{cases}$$

Si chaque arc de l'arbre connait la liste des sommets de $\mathcal{M}$ se projetant sur lui par la relation d'équivalence $\sim$, on peut obtenir une segmentation de la géométrie en zone où le nombre de contour est de 1. L'arbre de contour est alors dit *augmenté* (voir les zones de couleurs sur la Figure 1(b)).

3 Algorithme

Afin de pouvoir partager le travail sur les différents threads, notre approche se base sur un partitionnement des données par valeur scalaire. S'il y a n_t threads disponibles, alors l'image du domaine $f(\mathcal{M})$ est divisée en $n_t/2$ intervalles $\mathcal{I}_i$ de même taille, séparés par des ensembles de niveaux, formant une partition de $f(\mathcal{M})$ telle qu'illustrée par la figure 2(a). On crée ensuite un chevauchement fin entre ces segments $\mathcal{I}_i$ en ajoutant dans chacun les sommets qui lui sont voisins, c'est a dire les sommets qui ne sont pas

- *Gueunet Charles: Kitware, UPMC*
  E-mail: charles.gueunet@kitware.com.
- *Fortin Pierre: UPMC*
  E-mail: pierre.fortin@lip6.fr.
- *Julien Jomier: Kitware*
  E-mail: julien.jomier@kitware.com.
- *Tierny Julien: UPMC*
  E-mail: julien.tierny@lip6.fr.

dans $\mathcal{I}_i$ mais qui sont liés par une arête à un sommet de $\mathcal{I}_i$.
On calcule alors l'arbre de contour de chacun des segments ainsi augmentés (figure 2 [b]) en utilisant l'algorithme de rérérence [3]. Une partie de ce calcul peut s'effectuer en utilisant deux threads par segment. Pour finir, on regroupe le travail de ces segments efficacement en identifiant les arcs de chaque contour de l'ensemble de niveau à l'interface avec l'arc correspondant du segment opposé pour obtenir l'arbre de contour final.

4 Résultats

Les résultats présentés ici sont ceux obtenus avec l'implémentation présente dans TTK [11], sur un processeur Intel Xeon CPU E5-2630 v3 (2.4 GHz, 8 cores) et 64 GB de RAM.
Afin d'illuster les performances de notre méthode nous l'avons comparée à Libtourtre, une implémentation reconnue de l'algorithme séquentiel de référence dans le tableau 1. Pour ce faire, nous avons séléctionné 7 champs scalaires plus ou moins bruités que nous avons échantionnés sur une grille 3D de 256 sommets de coté. Ces résultats nous montrent une efficacité parallèle sur Elevation de $5.38/8 = 67\%$ et dans le cas général cette efficacité avoisine les 45%. De plus notre version parallèle calcule l'arbre de contour approximativement trois fois plus vite que Libtourtre.

4.1 Limitations

La figure 3 nous montre le passage à l'échelle en fonction du nombre de threads. La pente de cette courbe semble dépendre du chanp scalaire et plus particulièrment il semblerait que plus l'arbre en sortie est complexe et plus on s'éloigne de l'idéal.

Des tests non présentés ici pour des raisons de place nous ont permis de trouver les raisons de cette baisse de performances. Ces test sont détaillés dans l'artice [6].
Tout d'abord, le chevauchement entre nos différents segments induit du travail redondant qui n'est pas nécessairement équilibré. Il dépend de la taille de l'ensemble de niveau à l'interface entre deux segments. Par exemple dans le cas du Foot, cette taille varie d'un facteur 3 (Figure 5 de l'article [6]).
Nous avons également observé une saturation du bus mémoire. Les threads traitent des données éparses ne bénéficiant donc pas du "cache prefetch". Ils passent alors beaucoup de temps à attendre que le bus rapatrie les données dont ils ont besoin.
Enfin, le tableau 3 de l'artice [6] nous permet d'affimer que la vitesse de traitement des sommets est différente d'un segment à l'autre.

5 Application

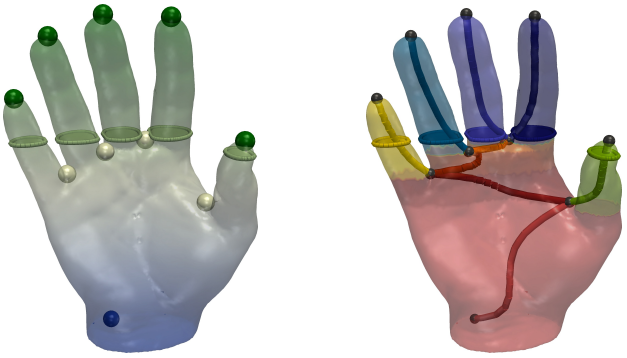
Notre implémentation est disponible au sein de la plateforme (Open Source) TTK [11] dont l'utilisation est illustrée par la figure 4. On voit ici qu'il est possible de calculer l'arbre de contour d'un jeu de données avec sa segmentation. D'autres plugins reposent également sur le calcul efficace de cette structure, tels que le diagramme ou la courbe de persistance également présent sur l'image, afin de permettre une exploration interactive de différents seuils de simplification par persistence.
Un exemple d'application concrète de l'arbre de contour au sein de la plateforme est la participation au *Visualisation Contest 2016* [1] (mention honorable), montré par la figure 5. Le but du concours était d'analyser les formes de doigts que l'on obtients en diluant un bloc de sel dans de l'eau, phénomène connus sous le nom de digitation visqueuse. Cette partitcipation est décrite en détail dans un autre document [5].

6 Conclusion

Dans ce papier, nous avons présenté un algorithme efficace pour calculer l'arbre de contour d'un maillage tétraédrique en parallèle sur une machine à mémoire partagée, avec sa segmentation.
L'implémentation correspondante est disponible au sein d'une plateforme Open Source permettant autant la reproduction des résultats que l'utilisation simple de cette méthode.

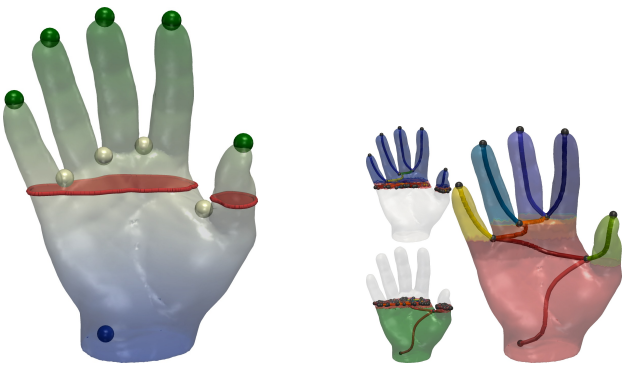
References

[1] IEEE Visualization Contest 2016. http://www.uni-kl.de/sciviscontest/.
[2] A. Acharya and V. Natarajan. A parallel and memory efficient algorithm for constructing the contour tree. In *PacificVis*, 2015.
[3] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Symposium on Discrete Algorithms*, 2000.
[4] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proc. of IEEE VIS*, pages 497–504, 2004.
[5] G. Favelier, C. Gueunet, and J. Tierny. Visualizing ensembles of viscous fingers. *IEEE SciVis Contest 2016*.
[6] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Contour forests: Fast multi-threaded augmented contour trees. *IEEE Large Data Analysis and Visualization*, 2016.
[7] A. Gyulassy, P.-T. Bremer, B. Hamann, and P. Pascucci. A practical approach to Morse-Smale complex computation: scalabity and generality. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, pages 1619–1626, 2008.
[8] S. Maadasamy, H. Doraiswamy, and V. Natarajan. A hybrid parallel algorithm for computing and tracking level set topology. In *International Conference on High Performance Computing*, 2012.
[9] V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. *Algorithmica*, 2003.
[10] B. S. Sohn and C. L. Bajaj. Time varying contour topology. *IEEE Transactions on Visualization and Computer Graphics*, 2006.
[11] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. Technical report, UPMC, https://topology-tool-kit.github.io/stuff/ttk.pdf.
[12] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 15:1177–1184, 2009.
[13] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pasucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. of ACM Symposium on Computational Geometry*, 1997.



(a) (b)

Fig. 1. Jeu de données d'une main, avec le champ scalaire étant l'élévation (du bleu vers le vert). (a) Visualisation du champ scalaire et des points critiques, ainsi qu'un ensemble de niveau constitué de cinq contours. (b) L'arbre de contour correspondant avec la segmentation induite.



(a) (b)

Fig. 2. Illustration du fonctionnement de notre approche sur le jeu de données de la main. (a) Le champ scalaire élevation partitionné en deux segments par l'interface rouge. (b) Les arbres de chacun des deux segments puis le résultats final.

Data-set	$\|\mathcal{C}\|_A$	CF seq.	CF para.	accel.	LT	accel. LT
Elevation	1	29.18	5.42	5.38	20.63	3.81
Ethande Diol	29	33.09	7.81	4.37	23.47	3.00
Combustion	3649	28.04	7.31	3.83	21.26	2.91
Boat	3235	29.94	7.44	4.02	23.26	3.13
Jet	4171	26.82	7.21	3.72	20.60	2.86
Enzo	282800	39.63	12.40	3.20	32.51	2.62
Foot	644463	18.09	9.72	1.86	13.52	1.39

TABLE 1
Comparaison de notre approche avec Libtourtre sur différents champs scalaires d'un jeu de données 3D cubique de 256 sommets de coté. $|\mathcal{C}|_A$ est la taille de l'arbre en nombre d'arcs. CF signifie Contour Forests et correspond à notre implémentation. LT signifie Libtourtre.

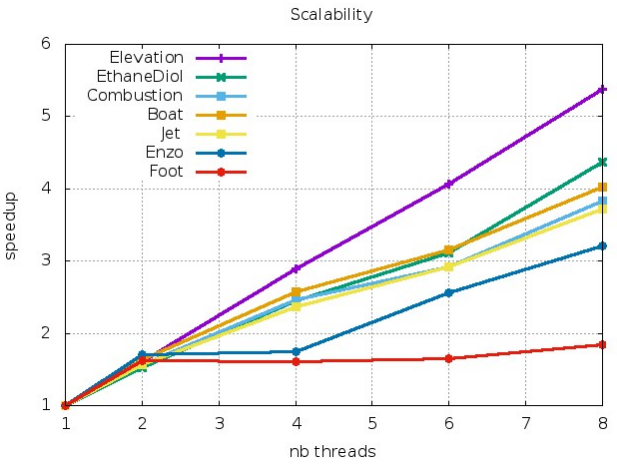


Fig. 3. Accélération obetnus avec notre algorithme en fonction du nombre de coeurs (une coubre par champ scalaire).

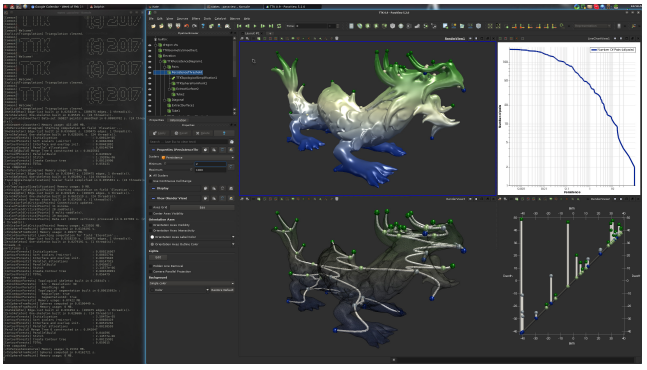


Fig. 4. Visualisation et analyse du jeu de donées Dragon avec le champ scalaire élévation au sein de TTK [11]. On peut voir sur cette image l'arbre de contour de ce jeu de données (centre bas), qui est également utilisé pour calculer le diagramme et les paires de persistances visibles dans la partie droite.

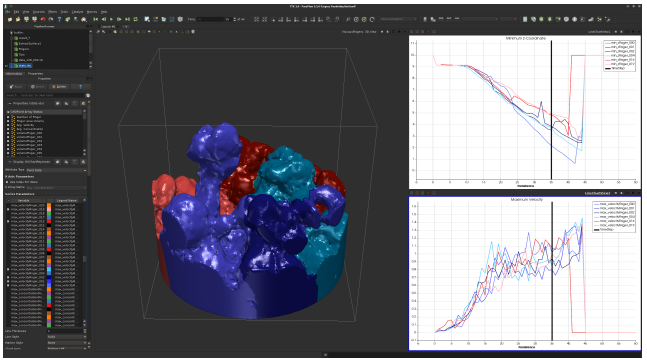


Fig. 5. Visualisation des différends doigts de sel dans un jeu de données de Vis Contest 2016 [1] au sein de notre plateforme. Chaque doigt de sel est associé à une couleur et les courbes de droites montrent leurs évolutions spatiale et dynamique.