Hierarchical Mesh Segmentation Editing Through Rotation Operations

Federico Iuricich[1(✉)] and Patricio Simari[2]

[1] Department of Computer Science and UMIACS,
University of Maryland, College Park, USA
iuricich@umiacs.umd.edu
[2] Department of Electrical Engineering and Computer Science,
The Catholic University of America, Washington DC, USA
simari@cua.edu

Abstract. Hierarchical and multi-resolution models are well known tools used in may application domains for representing an object at varying levels of detail. In the case of segmentations computed on a mesh, a hierarchical model can be structured as a binary tree representing the hierarchy of the region merging operations performed on the original segmentation for reducing its resolution. In this paper, we address the problem of modifying a hierarchical segmentation in order to augment its expressive power. We adapt two well-known operators defined for modifying binary trees, namely left and right rotation, to the case of hierarchical segmentations. Such operators are then applied to modifying a given hierarchy based on a user-defined function and based on a user-defined segmentation.

1 Introduction

The area of mesh segmentation algorithms research is a mature one, having inherited and adapted many of its approaches from the area of image segmentation. In the many years since its inception, segmentation algorithms have proliferated, each based on various distinct approaches and metaphors, including watershed, graph-cut, and hierarchical methods, to name a few. In all cases, once a segmentation is produced by the algorithm, it remains static and the client application must use the obtained result. If a different segmentation is desired, for example a refinement based on new objectives, an entirely new segmentation must be computed.

What if, rather than remaining static, a segmentation could be automatically refined in a principled way? By having segmentation-editing operations at our disposal, we could conceive of each segmentation as living in a space of candidate solutions, each connected to its neighbors by an edit operation application. In this fashion, we could start from a candidate segmentation obtained through the application of well-known and efficient algorithms. From here, with the algorithmically-guided application of edit operations, we could explore the space of nearby segmentations, i.e., the "segmentation neighborhood", in search

V. Murino and E. Puppo (Eds.): ICIAP 2015, Part I, LNCS 9279, pp. 398–409, 2015.
DOI: 10.1007/978-3-319-23231-7_36

of a result that better conforms to an objective that is new or more nuanced than the first algorithm was suited to optimizing.

The above motivates a "black box" approach to segmentation, decoupling the segmentation algorithm from the segmentation metric or objective function. This effectively allows us to cast the segmentation problem into the effective *metaheuristic search* framework that has been successfully applied in the Artificial Intelligence and Optimization communities. This brings a number of advantages. On the one hand, it enables research into effective domain-specific objective functions, which users are free to collect into libraries and reuse as needed. Simultaneously, the decoupled nature of the framework allows research into metaheuristic search algorithms in a domain-independent way. Any theoretical and practical improvements can then be "swapped in", immediately translating into improved performance across domains in a way that will be effortless and transparent to most users.

In this paper, we take a first step into the area of algorithmic segmentation editing. Our proposal focuses on hierarchical segmentations and the guarded application of rotation operations analogous to those applied for the rebalancing of binary trees. After introducing our framework, we demonstrate its use by applying it to two use case: the modification of a segmentation hierarchy based on a changed objective function, and based on a segmentation outside the space of segmentations encoded in the hierarchy.

2 Related Work

Hierarchical models for geometrical objects support the representation and processing of spatial entities at different levels of detail (*LOD*) [13,4]. Such representations are especially interesting because of their potential impact on applications such as terrain modeling in Geographic Information Systems (GIS) and scientific data visualization.

The basic ingredients of a hierarchical model for a spatial object are a *base complex*, that defines the coarsest representation of the object, a set of *updates* that provide variable resolution representations of the base complex when applied to it, and a *dependency relation* among updates which allow them to be combined to extract consistent intermediate representations.

The process of building a hierarchical model depends on the simplification of a cell complex. Usually, such an operation is time consuming because sophisticated techniques are required to optimize the shape of the cells and to bound the approximation error. However, such structure-building operations can be performed off-line so that the structure can then be efficiently queried on-line.

Hierarchies and multi-resolution models have been applied in a plethora of areas, including geometric modeling [6,4], morphological analysis of 2D and 3D images [11], and shape analysis [2]. Within the are of shape analysis, mesh segmentation algorithms cover a huge area of this field [16,19].

Among the existing segmentation methods, we are particularly interested in those which are often used to create an oversegmentation as a preprocessing

step. These include mean-shift clustering [18], normalized cuts [10,9,22,14,21], and, recently, an iterative approach which scales to high resolution meshes [20].

Complementary to the above algorithms is the idea of obtaining a hierarchical clustering of elements. The construction of this hierarchy can be driven by an error metric defined over the edges of the dual graph [7], or on fitting primitives of the regions [2], for example.

The hierarchical organization can provide a representation of the functional or semantic structure of a shape while reducing the emphasis on geometric details. We find various examples of these kinds of hierarchies which differ in the nature of the error function used in their construction. The hierarchy defined in [8] and applied to triangulated meshes is based on the diffusion distance on surfaces. In [17] a mesh partitioning and the corresponding hierarchy is built based on combining the well known Shape Diameter Function and the k-way graph-cut algorithm to include local geometric properties of the mesh. In [15] Reuter introduced a method to hierarchically segment articulated shapes into meaningful parts and to register these parts across populations of near-isometric shapes. The hierarchical relation applies the notion of persistent homology [5] for the elimination of topological noise.

Recently, in [12], the notion of co-hierarhical analysis has been applied to shapes. At the base of the method is the construction of a family of hierarchies for each single object. Among these hierarchies only the most representative ones are selected and studied in order to identify similarities between objects.

3 Update Operators

A tree rotation is an update operator used for changing the shape of a binary tree without changing the resulting in-order traversal of its elements. Two symmetric operators have been defined in the literature, right rotation and left rotation, which we will indicate as $rotationR(\cdot)$ and $rotationL(\cdot)$, respectively.

Let p be a node of the binary tree and q the left child of p, denoted $l(p)$. The $rotationR(p)$ operator changes the structure of the tree as follows:

- q becomes the new parent of p,
- p is the new right child of q,
- the old right child of q, denoted $r(q)$, becomes the new left child of p.

The latter can be seen as a clockwise rotation of the root p using the node q as pivot. The left rotation $rotationL(\cdot)$ is entirely dual and can be seen as a counter-clockwise rotation. These two operators are at the base of the definition of efficient data structure such as AVL, red-black, and splay trees.

Recall that working with a hierarchy of segmentations we can efficiently represent such a hierarchy as a proper binary tree T (i.e. a binary tree in which internal nodes have always two children) having the leaf nodes in one-to-one correspondence with the regions of the segmentation at finest resolution. Each internal node of T represents a merging operation between two regions (i.e. the regions represented by the left and right child) and consequently it represents

the new created region. Moreover, this structure also guarantees that, given two nodes with the same parent, the corresponding regions are adjacent in the segmentation extracted at such a level. In terms of the nodes of a tree, we will say that two nodes n_1 and n_2 are *adjacent* ($n_1 \diamond n_2$) if there exists one region in the n_1 subtree that is adjacent to at least one region in n_2 subtree.

Given the above, the rotation operators defined for general binary trees could potentially bring about two inconsistent representations when applied to a segmentation hierarchy. The two cases are shown in Figure 1b. Applying *rotationL*(p), pivoting on node 3, two internal nodes are created (nodes 3 and p) having only one child. The second problem is generated applying *rotationR*(p) pivoting on node q. The rotation applied is valid from the point of view of the node's connections. However, it results in an inconsistent representation of the segmentation. In particular, the merging between nodes 1 and 3 cannot be applied to the segmentation (see Regions 1 and 3 in Figure 1a).

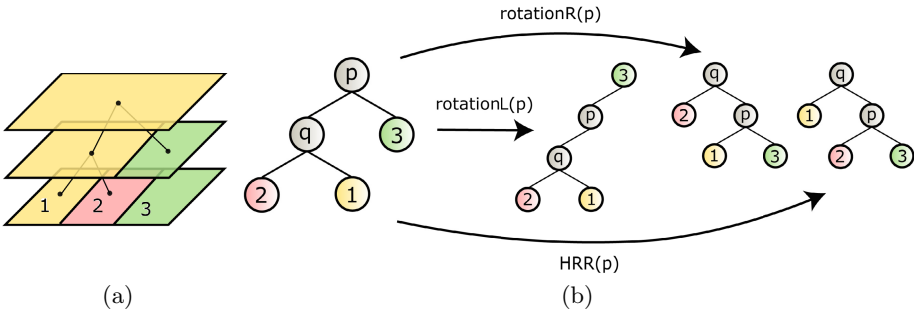


Fig. 1. (a) Binary tree and related segmentations extracted at different resolution levels. (b) Binary tree rotations resulting in invalid hierarchies and proper binary tree obtained after applying $HRR(p)$.

Thus, we have adapted the rotation operators in order to guarantee two invariants. (1) A rotation maps a proper binary tree into a proper binary tree with a different structure and, (2) for each internal node p, the left and right children are adjacents ($l(p) \diamond r(p)$).

Let q be the left child of an internal node p. The Hierarchical segmentation Right Rotation of p (denoted as $HRR(p)$), is a valid rotation if q is an internal node. Let $u = l(q)$, $v = r(q)$ and $s = r(p)$. As a consequence of $HRR(p)$, q becomes the parent of p. If v is adjacent to the region represented by s, then v becomes the left child of p. Otherwise u and v are swapped (along with the related subtrees) and u becomes the new left child of p. Note that at least one of the two regions represented by u and v is adjacent to the region represented by s since they have a common ancestor. When both u and v are adjacent to s the best node to be moved can be chosen based on application-dependent criteria. In Figure 1b we show the result of $HRR(p)$ performed on the tree depicted in

Figure 1a. The Hierarchical segmentation Left Rotation operator, indicated in the following as $HLR(\cdot)$ is performed in a dual fashion.

In contrast to the classical rotation operators, HRR and HLR rotations maintain invariant (1) as a consequence of the fact that node q, chosen as pivot, is, by definition, an internal node with non-empty subtrees.

Considering other modification operators for proper binary trees, the deletion and insertion of a single node is forbidden since it would make the binary tree not proper. The swap of two subtrees is a modification operator that could be considered in addition to rotations. Since the ordering between the left and right children of a node is irrelevant for the hierarchy, the swap of two nodes sharing the same parent, used in the rotation operators, is not a true update operator (i.e. it does not change the space of segmentations represented). Swapping two subtrees (not sharing the same parent) may cause inconsistencies in the hierarchical structure in general.

Before distinguishing between valid and non-valid swaps we introduce some basic definitions. Given the lowest common ancestor of two nodes n_1 and n_2 and its subtree T, we say that n_1 *uniquely depends* on n_2 ($n_1 \triangleright n_2$) if n_2 is the only node adjacent to n_1. As a consequence, we can say that:

- if $n_1 \triangleright n_2$ then either they share the same parent, or the parent of n_1 is the root of T,
- if $n_1 \triangleright n_2$, then the depth of n_1 is lower or equal to the depth of n_2.

The second property in particular implies that there does not exist a sequence of rotations (HRR or HLR) for moving n_2 above n_1. In Figure 2a, for example, node 5 uniquely depends on node 2 ($5 \triangleright 2$) since 2 is the only node adjacent to 5.

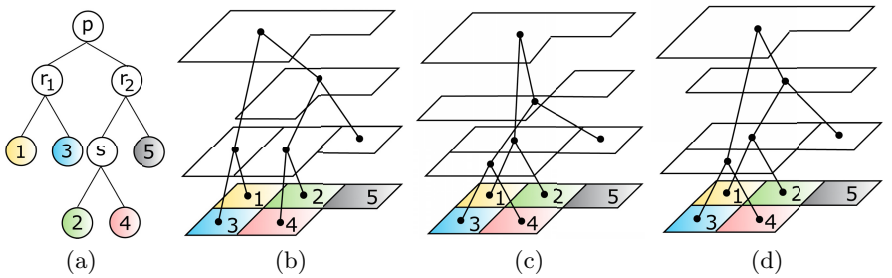


Fig. 2. (a) Binary tree representing the space of segmentations depicted in (b). (c) Inconsistent hierarchy resulting from the non-valid swap of nodes 2 and 3. (d) Hierarchy resulting from the valid swap of nodes 1 and 4

In the following, we define the conditions necessary to identify a valid swap. Let p be the common ancestor of two nodes n_1 and n_2. Let s_1 and s_2 be the siblings of n_1 and n_2 respectively. Let T_1 and T_2 be the left and right subtrees of p, rooted at r_1 and r_2 respectively.

A swap between n_1 and n_2 is a *valid swap* if $n_1 \diamond s_2$ and $n_2 \diamond s_1$. Moreover, let U be the set of nodes $n \in T_1$ such that $n \triangleright n_1$. The swap is valid if either:

i - U is empty, or
ii - for each $n \in U$, $n \diamond n_2$. (Conditions for T_2 and n_2 are dual.)

In Figure 2a for example, the swap between 1 and 4 is valid because $1 \diamond 2$, $3 \diamond 4$, and none of the nodes in the subtree of r_2 uniquely depend on 4 (condition i), while the node in the subtree of r_1 that uniquely depends on 1 (i.e. 3) is also adjacent to 4 (condition ii). In Figure 2d, the resulting hierarchy is shown.

Recalling the properties of $\diamond$ and $\triangleright$, we observe that if either condition i or condition ii are not satisfied, there does not exist a sequence of rotations for moving one node in place of the other. This leads us to conjecture that a valid swap can always be represented as a sequence of HRR and HLR rotations. In this case the rotation operators here defined would form a minimal complete set of modification operators for a segmentation hierarchy.

4 Implementation and Preliminary Results

We have studied the rotation operators HRR and HLR in the context of hierarchies built from the segmentation of triangular meshes. In our naive implementation we are encoding both the binary tree structure and the mesh. Given a triangle mesh Σ, we are indexing vertices and triangles in two separate arrays. For each vertex, we store its coordinates while, for each triangle, we store the indexes to its three vertices and the indexes to its three adjacent triangles. Let $|\Sigma_0|$ the total number of vertices and $|\Sigma_2|$ the total number of triangles the whole mesh representation takes $3|\Sigma_0| + 6|\Sigma_2|$.

Considering the proper binary tree implementation, each node encodes a pointer to its right and left children and an application-dependent value (generally a float). Differently from a binary search tree, the time complexity for finding a node in the hierarchical segmentation is linear in the number of nodes in the hierarchy. Thus, for each node, we are also encoding a pointer to its parent in order to improve the navigation efficiency. If N is the number of nodes in the binary tree, we are encoding $3N$ pointers and N float values in total.

Leaf nodes encode additional information. The set of top simplices (triangles) belonging to the corresponding region and an adjacency list pointing to the adjacent regions in the segmentation at finest resolution. If A is the set of pairs of adjacent regions we encode $2|A|$ pointers and $|\Sigma_2|$ indexes in total.

We have conducted our experiments on four benchmark meshes. All the results have been obtained on a MacBook Pro with 2.8Ghz Quad-core Intel Core i7 processor and 16GB of RAM. For all cases, the storage cost required by the hierarchy is at least two orders of magnitude smaller than the storage cost required by the triangle mesh. The Female dataset is composed of 4K vertices and 9K triangles and its over segmentation is formed by 100 regions. Hierarchies are built on this dataset in about 0.26s. The Vase daset is composed by 14K vertices, 29K triangles and 200 regions, as is the case for the Armadillo dataset, which is composed of 25K vertices and 50K triangles. Hierarchies on these datasets are built in 1.7 and 2.8 seconds respectively. The Neptune dataset

is formed by 250K vertices, 500K triangles and 300 regions. The entire encoding for the mesh and the hierarchy requires approximately 18MB and building the hierarchy takes 42.1 seconds.

4.1 Hierarchy Update Based on an Input Function

For our first application, we have considered the possibility of modifying a pre-built hierarchy based on a function given as input. Specifically, we are aiming at constructing a hierarchy that adaptively combines different functions together. The first function involved computes the volume of the axis-aligned bounding box of the region resulting from the merging operation. Regions whose composition results in a small bounding box are merged before regions creating a larger bounding box. The second function evaluates the concavity along the border of two regions using the distance function described in [20]. Regions having an almost flat behavior along their boundary are the first to be merged.

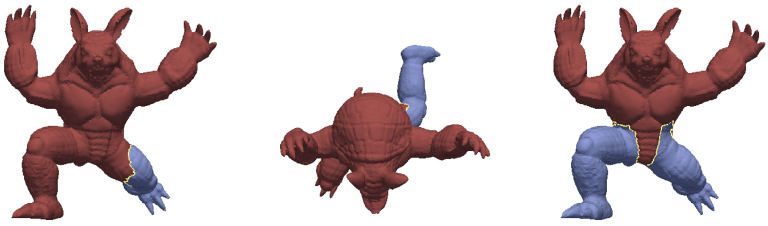
In all of our experiments our starting segmentation is produced with the superfacets algorithm presented in [20]. The hierarchy is then the result of the simplification sequence created based on the function computing the bounding box volume. Based on this sequence of merging operations, nodes are created starting from the leaves and ending with a single node (the root) representing the single region obtained at the end.

Segmentations are extracted from the latter hierarchy using a threshold value indicating the desired resolution. Let t be the threshold value. The resulting segmentation is the one obtained navigating from the root and considering only those nodes having associated value greater than or equal to t. In our experiments we have always chosen a value of t resulting in a limited number of nodes (and thus a low number of regions) in order to help the visual comparison between the methods.

Once the first hierarchy is obtained, a new hierarchy is computed based on the simplification sequence created by the concavity function. The resulting hierarchy is then balanced using the bounding box function. New values for each node are computed based on the latter function. Nodes are then considered in a post-order sequence. Given a node q and its parent p, a rotation is applied on p pivoting on q if the value of p after the rotation will be lower with respect to the value of q before the rotation. After the rotation, a depth-first visit is recursively performed on the subtree of q in order to search for new rotations that could be triggered by the latter. Note that the only values changed during a rotation are the value of p and q. The aim of the rotations in this case is to minimize the increase of the function values navigating toward the root, thus simulating the hierarchical structure that would result using the bounding box function from the beginning.

At this point we have performed extractions on both hierarchies comparing the segmentations obtained. In Figures 3a-3b we show the segmentation for the Armadillo dataset, extracted from the original hierarchy, splitting the root only. The segmentation at the finest resolution is composed of 300 regions. The leg is last to be merged with the body since its expansion is in

the opposite direction. In Figure 3c we show the segmentation obtained from the second hierarchy. Since the two legs are connected by a flat area (under the body) the influence of the concavity function makes those two parts become a single component in the first level of the hierarchy, thus improving the semantic of the result.



(a) (b) (c)

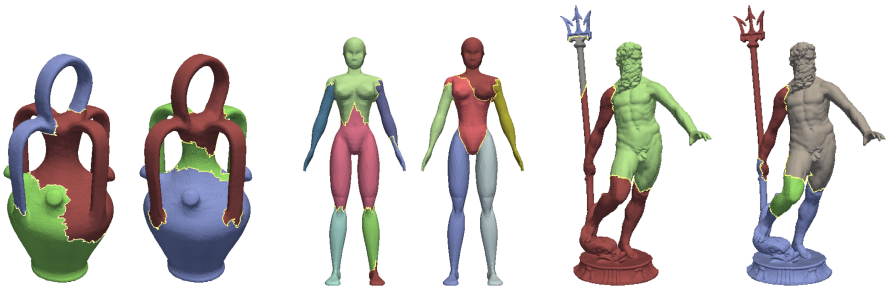
Fig. 3. (a-b) Segmentations for the ARMADILLO dataset obtained from the hierarchy based on the bounding box function and (c) the segmentation obtained combining two functions. Rotations are applied in 1.03 seconds

In Figure 4 we show three other examples were the semantic segmentation obtained from the first hierarchy (on the left) is improved by composing the two functions. In Figure 4a, the handles of the VASE dataset are better separated from the body. In Figure 4b, instead, the head of the FEMALE dataset is distinguished from the body while legs and calves are treated as single objects. For the NEPTUNE dataset shown in Figure 4c, the head of the trident is correctly treated as a single object in the early levels. However this is still a delicate operation. The legs in 4c or the torso of 4b, for example, are less well segmented using our framework.

The study of more complex state-of-the-art functions will be at the center of future developments.

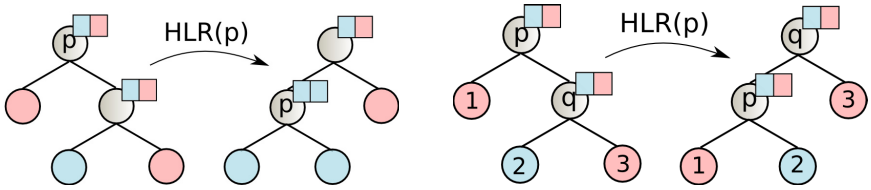
4.2 Hierarchy Update Based on an Input Segmentation

For our second application we have considered the possibility of adapting a hierarchy to an input segmentation. For our preliminary investigations we have restricted our problem. The building blocks of the input segmentation are the same as those of the original regions on which the hierarchy has been built. This means that a leaf node in the hierarchy is always contained in only one region of the input segmentation (for general purpose implementations, building the hierarchy starting from the triangles guarantees that any segmentation defined on the same dataset respects this condition). The second restriction is given by the number of regions composing the segmentation given as input. We are using input segmentations with only two regions. However any segmentation could be treated with the same algorithm grouping the regions in two main sets and then calling the algorithm recursively on their subsets.



(a) (b) (c)

Fig. 4. Comparison of the results obtained with the two hierarchies for (a) the VASE, (b) FEMALE and NEPTUNE datasets. Rotations take 0.9, 0.1 and 10.3 seconds, respectively



(a) (b)

Fig. 5. (a) Rotation $HRR(p)$ reducing the number of undetermined nodes. (b) The same operation is useless if regions corresponding to nodes 1 and 3 are not adjacent. Leaf nodes are colored with red or blue depending on the region to which they belong. Colored squares over internal nodes distinguish between determined nodes (having the same color) and undetermined nodes (having both colors).

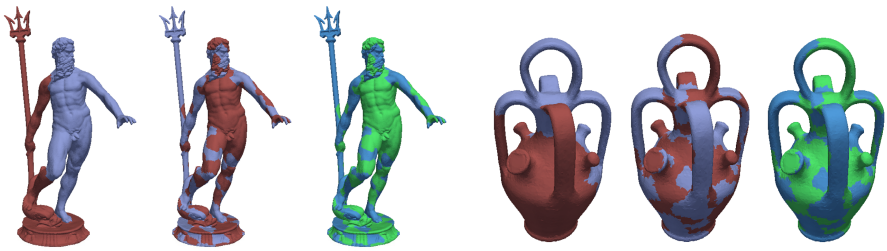
In this case the value associated with each leaf node corresponds to an index (1 or 2), indicating the region in the input segmentation to which the node belongs. The value associated with each internal node is then the set of labels of its children (such a set can be $\{1\}$, $\{2\}$ or $\{1, 2\}$). We will call *determined* a node having only one label and *undetermined* the node having both labels.

Once the values have been computed, a post-order visit on the tree is performed. The objective of the algorithm is to reduce the number of undetermined nodes by means of rotations. The input binary segmentation is successfully encoded in the hierarchy when the root is the only node undetermined.

A rotation is applied to a node p if the number of undetermined nodes decrease after the rotation (see Figure 5a). Note that the nodes the state of which can change are limited to p and $l(p)$ for an HRR rotation and to p and $r(p)$ for an HLR rotation. When a rotation is performed the subtree of the new root is visited in post-order sequence, but subtrees of determined nodes are never visited twice.

There are degenerate configurations in which the latter criterion is not sufficient. An example of this configuration is depicted in Figure 5b. The adjacency

relation between the regions actually obstructs some rotations that would reduce the number of undetermined nodes thus resulting in a "stuck" configuration. To overcome this problem, when at the end of the visit either one of the children of the root is still undetermined, we do the following on that respective subtree: using a depth-first visit, the deepest undetermined node is identified and, with a sequence of rotations, it is brought to the root of the subtree. Intuitively, we can think about the distance of a node from the root as the scope of the node with respect to the other regions. The nearer a node is to the root, the higher the number of regions with which it can be merged through rotations. This corresponds to augmenting the scope of the node, thus enabling new rotations.



(a) (b)

Fig. 6. From left to right, binary segmentation encoded in the hierarchy, input segmentation and binary segmentation obtained by adapting the hierarchy for the NEPTUNE (a) and VASE (b) datasets.

After that, a new visit is triggered on the entire tree. Note that a subtree of a determined node does not need to be visited. In Figure 6, we show some of the results obtained mapping with different colors the regions of the input segmentation and the one obtained after training the hierarchy.

5 Conclusions

In this paper we have addressed the problem of modifying the structure of a hierarchical segmentation, allowing for an expansion of said hierarchy into a space of related segmentations.

We have defined two update operators related to the well known rotation operators devised for binary trees. We have applied these operators for modifying a hierarchical segmentation based on an input function and on an input binary segmentation.

Our future work will concentrate on testing our framework with different kinds of functions, also combining the modifications triggered by a new objective function with the modifications triggered by a new objective segmentation. We would like to apply the framework also to scalar fields, typically represented as triangle meshes with a scalar value defined on each vertex, where the combination

of functions computed on the shape with functions computed based on the scalar values could bring new insights in this area.

Even though here we have described our implementation based on triangle meshes, our framework can be applied to any simplicial mesh, including, for example, tetrahedral meshes discretizing 3D volumes.

Acknowledgments. Datasets used are courtesy of the Princeton Mesh Segmentation Benchmark [3] (FEMALE, VASE and ARMADILLO datasets) while the NEPTUNE dataset is courtesy of the aim@shape repository [1]. This work has been partially supported by the National Science Foundation under grant number IIS-1116747. The authors wish to thank professor Leila De Floriani for the fruitful discussions.

References

1. AIM@Shape. http://visionair.ge.imati.cnr.it/ontologies/shapes/
2. Attene, M., Falcidieno, B., Spagnuolo, M.: Hierarchical mesh segmentation based on fitting primitives. The Visual Computer **22**(3), 181–193 (2006)
3. Chen, X., Golovinskiy, A., Funkhouser, T.: A benchmark for 3D mesh segmentation. ACM Transactions on Graphics (Proc. SIGGRAPH) **28**(3), 73:1–73:12 (2009)
4. De Floriani, L., Kobbelt, L., Puppo, E.: A survey on data structures for level-of-detail models. In: Dodgson, N., Floater, M., Sabin, M. (eds.) Advances in Multiresolution for Geometric Modelling. Mathematics and Visualization, pp. 49–74. Springer, Heidelberg (2005)
5. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. Discrete & Computational Geometry **28**(4), 511–533 (2002)
6. Garland, M.: Multiresolution modeling: Survey and future opportunities (1999)
7. Garland, M., Willmott, A., Heckbert, P.S.: Hierarchical face clustering on polygonal surfaces. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics, I3D 2001, pp. 49–58. ACM, New York (2001)
8. de Goes, F., Goldenstein, S., Velho, L.: A hierarchical segmentation of articulated bodies. In: Proceedings of the Symposium on Geometry Processing, SGP 2008, pp. 1349–1356. Eurographics Association, Aire-la-Ville (2008)
9. Hu, R., Fan, L., Liu, L.: Co-segmentation of 3d shapes via subspace clustering. Comput. Graph. Forum **31**(5), 1703–1713 (2012)
10. Huang, Q.X., Koltun, V., Guibas, L.J.: Joint shape segmentation with linear programming. ACM Transactions on Graphics **30**(6), 125 (2011)
11. Iuricich, F.: Multi-resolution shape analysis based on discrete Morse decompositions. Ph.D. thesis, University of Genova - DIBRIS, Italy (2014)
12. van Kaick, O., Xu, K., Zhang, H., Wang, Y., Sun, S., Shamir, A., Cohen-Or, D.: Co-hierarchical analysis of shape structures. ACM Trans. Graph. **32**(4), 69:1–69:10 (2013)
13. Magillo, P.: Spatial Operations on Multiresolution Cell Complexes. Ph.D. thesis, PhD thesis, Department of Computer Science-University of Genova (1998)
14. Meng, M., Xia, J., Luo, J., He, Y.: Unsupervised co-segmentation for 3D shapes using iterative multi-label optimization. Computer-Aided Design **45**(2), 312–320 (2013)
15. Reuter, M.: Hierarchical shape segmentation and registration viatopological features of laplace-beltrami eigenfunctions. International Journal of Computer Vision **89**(2–3), 287–308 (2010)

16. Shamir, A.: A survey on mesh segmentation techniques. Computer Graphics Forum **27**, 1539–1556 (2008)
17. Shapira, L., Shamir, A., Cohen-Or, D.: Consistent mesh partitioning and skeletonisation using the shape diameter function. Vis. Comput. **24**(4), 249–259 (2008)
18. Sidi, O., van Kaick, O., Kleiman, Y., Zhang, H., Cohen-Or, D.: Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. ACM Transactions on Graphics **30**(6), 126:1–126:9 (2011)
19. Simari, P.D.: Algorithms in 3D Shape Segmentation. Ph.D. thesis, Toronto, Ont., Canada, Canada, aAINR61094 (2009)
20. Simari, P.D., Picciau, G., De Floriani, L.: Fast and scalable mesh superfacets. Comput. Graph. Forum **33**(7), 181–190 (2014)
21. Wu, Z., Shou, R., Wang, Y., Liu, X.: Interactive shape co-segmentation via label propagation. Computers and Graphics **38**(C), 248–254 (2014)
22. Wu, Z., Wang, Y., Shou, R., Chen, B., Liu, X.: Unsupervised co-segmentation of 3D shapes via affinity aggregation spectral clustering. Computers and Graphics **37**(6), 628–637 (2013)