# The Level Set Tree on Meshes

Julie Digne, Jean-Michel Morel
CMLA, ENS Cachan, CNRS, UniverSud,
61 Avenue du Président Wilson,
F-94230 Cachan
digne@cmla.ens-cachan.fr
morel@cmla.ens-cachan.fr

Nicolas Audfray, Charyar Mehdi-Souzani
LURPA, ENS Cachan, Univ. Paris Sud 11,
61 Avenue du Président Wilson,
F-94230 Cachan
audfray@lurpa.ens-cachan.fr
souzani@lurpa.ens-cachan.fr

## Abstract

*Given a scalar function defined on a meshed surface, its level set component tree can be computed by a fast algorithm. This tree structure allows for an adaptation to meshes of the Maximally Stable Extremal Regions (MSER) method. Applied to the mesh curvature, this algorithm extracts significant curvature level lines and segments 3D surfaces into smooth parts separated by curves with high curvature. Segmentation results are shown on high resolution meshes of archaeological and industrial pieces. They compare favorably with MSER segmentations of pictures of the same objects.*

## 1. Introduction

Segmenting meshes or point clouds into their significant parts is a basic but still challenging problem. Robust shape descriptors are required for most applications such as facet classification, shape registration, mesh simplification or shape retrieval. To this aim, the "crest lines" detection plays a role analogous to edge detection in image analysis. Crest lines are usually defined as the loci of directional extrema of curvature. Ridge (resp. valley) points can be defined as points where the maximum principal curvature takes a positive maximum (resp. negative minimum) along the line tangent to its eigenvector.

This notion being closely linked to the surface curvature, robust curvature estimators are needed. Curvatures can be computed by surface regression [5], by discrete schemes using the mesh triangle geometry [27], or by computing the surface tensor [32]. One of the most used curvature estimation method is [34], where curvatures are estimated by drawing curves on the mesh surface (see [22] for an adaptation to point clouds). Other mesh based methods use the angles between adjacent mesh triangles to determine the curvature [10],[16].

Most crest extraction methods detect and link the points where the derivative of the curvature crosses zero. These methods are usually defined on meshes, but can be adapted to raw point clouds [23],[2],[35],[30], [8],[15]. In [7], potential features are extracted by regressing surfaces near those points, estimating the number of fitted surfaces, and deducing the feature type. The idea of using feature lines for surface segmentation was investigated in [33] and suggested in [17]. Regression free methods include [14], where the analysis of the surface local covariance matrix leads to point classification, or [31] where a multi-scale approach is introduced, yielding good results for mechanical shapes. Indeed, texture features are detected at fine scale, whereas at coarse scale the features describe shape geometry and are more robust. Recent research has also made an amazing progress in the rendering of viewpoint dependent apparent edges [9], [18].

Crest lines have, however, two limitations. The first is that these lines are often obtained by computing degree three derivatives, which is not an easy task for noisy or textured surfaces. The second is that crest points must be connected by some heuristic linkage. The final crest lines being often open or broken, they do not provide a surface segmentation.

Analogous problems arise in image analysis with *edge detection*. Image analysis therefore also uses *segmentation* methods dividing the image into regions separated by closed curves. One of the most reliable ways to define such closed curves is to extract the contrasted level lines. The level lines are the boundaries of connected components of upper (or lower) level sets. They inherit from these connected components an inclusion tree structure (fig. 1). This structure was called *Tree of Shapes* in [1], and a fast algorithm computing them, the *Fast Level Set Transform* is given in [28].

For an image $I$ defined on a domain $\mathcal{D}$ and with values in $\mathbb{R}$ the level sets with level $\lambda$ are

$$F^\lambda = \{x \in \mathcal{D} | I(x) \geq \lambda\} \text{ (upper level set)}$$

$$F_\lambda = \{x \in \mathcal{D} | I(x) \leq \lambda\} \text{ (lower level set)}$$

If $\lambda' > \lambda$, then $F^{\lambda'} \subset F^\lambda$ and each connected component of $F^{\lambda'}$ is contained in one connected component of $F^\lambda$. The set of connected components of upper (resp. lower) level sets partially ordered by inclusion is therefore a tree. The shape tree proposed in [28] is a fusion of both trees.

The level sets can be represented by their borders $\partial F^\lambda$ and $\partial F_\lambda$ which are unions of closed Jordan curves, the image *level lines*. Several methods have been proposed to select the relevant level lines. A definition of meaningful level lines is given in [3], [11]. More recently the MSER method introduced the same objects with different names: the connected components of upper or lower level sets are called *extremal regions (ER)*. The ones with best contrasted level lines are called *maximally stable extremal regions (MSER)* [25]. The extraction of significant level lines to segment data is so useful that it has been extended to 3D medical imaging to extract meaningful level surfaces [4],[26], and to video analysis [13], where extremal regions are tracked from frame to frame.

To the best of our knowledge these level line techniques have not yet been extended to meshes. The reason could be the lack of straightforward intrinsic scalar functions linked to a mesh, (such as the grey level for images). But there are actually such functions on meshes, the simplest one being the mean curvature. Several methods have already considered the curvature level lines and the umbilical points, but mostly from a theoretical point of view [19],[6] and [24]. But curvature level lines have not yet been studied as valuable feature lines, or used for surface segmentation. The goal of the present paper is to describe an algorithm computing all conspicuous curvature level lines, and to give experimental evidence that the method detects valuable mesh features.

In a way, the present work extends [21] and [36]. In these works, the surface is (implicitly) decomposed into a smooth base and a height value in the normal direction. Then the edges or iso-contours of this height are extracted. The Mesh-MSER framework considers this same situation in a more general setting : a surface with any scalar function defined on it. As will be shown in Fig. 8, results comparable to the results of [36] can be obtained by a significantly simpler and more general procedure.

The remainder of this paper is divided as follows: Section 2 recalls the image MSER method, discusses its adaptation to meshes, and gives the algorithm building the level set component tree. Section 3 describes the algorithm extracting maximally stable extremal regions from this tree. Section 4 shows results on various simple and complex scanned objects, discusses strategies for level line selection, and compares 3D Mesh-MSER results to 2D MSER results on pictures of the scanned objects.

## 2. Mesh Extremal Regions

### 2.1. Definition of MSER for 2D images [25]

Let $I$ be a real function defined on an image domain $\mathcal{D} \subset \mathbb{Z}^2$. MSER needs an adjacency relation $A$ for elements of $I$, and usually chooses a 4 or 8 connectivity. The boundary $\partial N$ of any set $N \subset \mathcal{D}$ is defined as $\{q \in \mathcal{D} \setminus Q \mid \exists p \in Q,\ pAq\}$. An *Extremal Region* $N$ is a region such that for every $p \in N$ and every $q \in \partial N$ one has $I(p) > I(q)$ (maximum extremal region) or $I(p) < I(q)$ (minimum extremal region). To define *Maximally Stable Extremal Regions (MSER)*, consider a sequence $(N_i)_i$ of nested extremal regions ($N_{i+1} \subset N_i$). A region $N_{i^*}$ in the sequence is maximally stable iff its *area change rate* $q(i) = \frac{|N_{i-\delta}| - |N_{i+\delta}|}{|N_i|}$ has a local minimum at $i^*$. The small variation $\delta > 0$ is a parameter of the method.

The detection of MSER proceeds by:

1. sorting pixels by intensity;
2. iteratively placing pixels in the image and updating the list of connected components and their areas;
3. selecting intensity levels that are local minima of the area change rate as thresholds, producing MSERs.

In a more formal way, the method uses the fact that upper level sets $F_k = \{p | I(p) \geq k\}$ are ordered by inclusion: $F_{k+1} \subset F_k$. One calls extremal region any connected component of some level set. For each connected component $N_k$ of the level set $E_k$, either $k = k_{min}$, in which case $N_k$ is the whole image domain, or there exists a connected component $N_i$ of the upper level set $F_i$ such that $i < k$ and $N_k \subset N_i$. Thus the set of extremal regions is a rooted tree [1] called (upper) *level set component tree*. Its dual tree is the (lower) level set component tree. The fastest component tree method seem to be [29]. It is its tree structure that allows the fast selection of MSERs [13].

We will now adapt these definitions to the case of meshes.

### 2.2. Level set trees: an extension to 3D meshes

Let $(V, T)$ be a set of vertices and triangles sampled from a 2-manifold $\mathcal{M}$ embedded in the 3D Euclidean space $\mathbb{R}^3$. Points of $v \in V$ are linked to other points of $V$ by edges forming triangles $t \in T$. We will assume that each edge is adjacent to either one or two triangles, so that each point belongs to at least one triangle. This means that there is no orphan edge and no orphan point, and that the mesh has no edge adjacent to three triangles. In other terms, the mesh is a manifold.

To define a level set tree we need a real function defined on the mesh. The function $H\colon V \to \mathbb{R}$ associating with each vertex $v$ its mean curvature $H(v)$ will be our example throughout the paper. Mesh regions will be defined as unions of mesh triangles. A level set tree requires a topology and therefore an adjacency relation on the mesh. Two

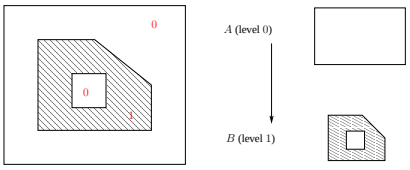



Figure 1. Example of a binary image (left) and its component tree (right). Node $A$ is the father of $B$ and $B$ contains a hole

triangles will be called adjacent if they share an edge. With this definition, analogous to 4-connectivity on 2D images, two regions sharing a vertex but no edge are disconnected. The main differences with the two-dimensional case are that the mesh itself can be disconnected, and that it usually contains scanning holes. If the mesh is disconnected the component tree is a forest. The algorithm will process independently each tree. Section 3.3 will explain how to handle scanning holes.

As for images, connected components of upper level sets can contain topological holes which are themselves connected components of lower level sets (fig. 1). Monasse [28] therefore proposed to build an *shape tree*, which is a fusion of the upper level set component tree with the lower level set component tree. Since an upper component tree is faster to build, and since it is the appropriate object to perform MSER extraction on the mesh, we will limit ourselves to the upper component tree.

### 2.3. Building the component tree for meshes

Set $\Lambda = \{h_1 < h_2 < \cdots < h_n\}$ a set of quantized values of $H$ on $V$. The triangles $t \in T$ will be ordered by setting $Level(t) = \max\{k | \min_{v \in t}(H(v)) \geq h_k\}$. Referring to the 2D case, we can say that the triangles play the roles of pixels and that the real function used for building the component tree is $Level(t)$. Algorithm 1 describes the construction of the tree of level set components on $T$.

**Theorem 1.** *Assume that the mesh $(V, T)$ is a manifold (meaning that each edge is shared by at most two triangles). Then the list of nodes and father-child relations created by Algorithm 1 gives the graph of connected components of level sets of $H$ a mapping defined on $\mathcal{M}$. This graph is a forest (meaning that the constructed graph has no cycles).*

**Sketch of Proof .** For simplicity, in the algorithm and in the comments below, $k$ denotes the level $h_k$. Only the order of the levels matters, and it is reflected by $k$. Algorithm 1 is strongly based on the triangle processing order from higher to lower levels. Indeed, while the algorithm is building nodes at level $k$, the only previously built nodes have a higher level. This fact is used for expanding the node: the expansion of a level $k$ node is performed by successively adding the not previously added neighboring triangles of level $k$ to the node triangle set. When expanding a new node $N_{new}$, we may encounter triangles already processed

**Algorithm 1**: Building the Component Tree Forest

**Data**: A list of the mesh triangles $F$ tagged with their levels
**Result**: The component tree forest $\{\mathcal{F}(\mathcal{M})\}$

```
1  Compute the levels of all triangles;
2  Sort F in decreasing level order;
3  Set all triangle markers to active;
4  for t ∈ F and t is active do
5      k = t → level;
6      Create an empty node N_new;
7      E = {t};
8      while E is not empty do
9          Remove and return the first element t of E;
10         Get t_1, t_2, t_3 the neighbors of t;
11         for i = 1 ··· 3 do
12             if t_i is inactive and t_i → level == k then
13                 Add t_i to the set E;
14             end
15             if t_i → level > k then
16                 Get the node N containing t_i;
17                 Get P the last built ancestor of N;
18                 if P → level = k then
19                     Merge node P into N_new;
20                 else
21                     P.father = N_new;
22                     Add P to N_new's children;
23                 end
24                 |N_new| ← |N_new| + |P|;
25             end
26         end
27         Mark t as inactive;
28         Add t to the set of triangles of N_new;
29         |N_new| ← |N_new| + Area(t);
30     end
31 end
```

(line 15) and belonging to a node $N$ at level $l$ (then necessarily $l > k$; otherwise the node would not have yet been created). By going up in the parent-child relation, we can retrieve $P$ (line 17), the last created ancestor of $N$ (which can be $N$ itself if it has no parent). For the same reason as before, $P$'s level must be superior or equal to $k$. If $P$ has the same level as $k$, then it belongs to the same node, and one can merge both nodes (line 19) by merging their triangle sets and adding $P$'s children to the set of $N_{new}$'s children (and then deleting $P$). If $P$ has a level larger than $k$, then $P$ is a child of $N_{new}$ (line 21), and we can set $P$'s father to be $N_{new}$ and add $P$ to the set of $N_{new}$'s children.

During the construction of each node, track is kept of the triangles which are contained in this node, but not in its children. This way, each triangle belongs to a single node.

The algorithm also keeps track of the area $A$ (sum of the areas of triangles belonging to the node and to its descendants) while building the tree. This information is used in Mesh-MSER.

Algorithm 1 being similar to [29], the complexity of building the forest is also quasi-linear. Indeed, it starts by sorting the triangles, an $O(N \log N)$ step. When expanding a node, the computationally demanding case is when the encountered triangle belongs to an already created node. This requires finding the last created ancestor, which entails some traversing node operations, merge triangle lists (constant time) and add areas (1 operation). Since each triangle is processed only once, the total complexity is roughly $N \log N$.

There are as many trees in the forest as nodes with no parent. By arguments similar to [28] one can prove:

**Theorem 2.** *Any quantized scalar function $H$ on a manifold mesh $(V, T)$ can be reconstructed from its component tree. This means that no information is lost on $H$ in the component tree.*

Assume the quantized levels of $H$ are $h_k,\ k = 1, \cdots, n$. We want to extract local maximal elements (in the MSER sense) in the level set tree of $H$. Call $\delta > 0$ the level step used for computing the stability coefficient of a node $N_{h_k}$ at level $h_k$. The set of test levels $h_1 < \cdots < h_n$ must therefore be complemented with all levels $(h_i + \delta)$ and $(h_i - \delta)$. For each node $N_{h_i}$, going up in the tree hierarchy yields its descendant $N_{h_i+\delta}$ with level $h_i + \delta$, and going down yields its ascendant $N_{h_i-\delta}$ with level $h_i - \delta$. Once these nodes are at hand, the stability coefficient is simply $q(N_{h_i}) = \frac{|N_{h_i-\delta}|-|N_{h_i+\delta}|}{|N_{h_i}|}$. For simplicity, we assumed in the previous relation that $N_{h_i}$ only has one descendant with level $h_i + \delta$, which is not necessarily true. In the case of multiple descendants with level $h_i + \delta$, the sum of their areas is used instead of $|N_{h_i+\delta}|$. Remark that $N_{h_i-\delta}$ is not necessarily the father of $N_{h_i}$, since there is no condition on the size of $\delta$ relatively to $\min_i(h_{i+1} - h_i)$. For the same reason $N_{h_i+\delta}$ is not necessarily a son of $N_{h_i}$. This is why we must go up and down father-son relations in the component tree.

# 3. Extracting maximally stable regions from the component tree

## 3.1. Mesh-MSER: the algorithm

Algorithm 2 describes how to extract maximally stable extremal regions (MSERs). Starting from the component tree, this is an easy task. The tree structure gives a quick access to the area variations between $N_{h_i+\delta} \subset N_{h_i} \subset N_{h_i-\delta}$. When computing the stability coefficient, topological changes are authorized by the algorithm, whereas the original image-MSER technique only compares nodes on branches with no bifurcation. This way more lines are found than in the original 2D image method. The node merging procedure (2) is also standard. It generates a subtree whose nodes are exclusively the maximally stable regions. Merging a node $N_{h_{i+1}}$ into its father $N_{h_i}$ requires a) removing $N_{h_{i+1}}$ from the set of $N_{h_i}$'s children, b) adding all of $N_{h_{i+1}}$'s children to $N_{h_i}$'s children, c) setting their father to be $N_{h_i}$, d) merging the list of triangles and e) updating the areas accordingly.

**Algorithm 2**: Mesh-MSER

**Data**: $\delta$ a step for computing stability coefficients and a set of levels $h_1 < \cdots < h_n$
**Result**: A labeling of the triangles and the borders of the detected MSERs

Build the component tree with levels $h_i \pm \delta$;
**for** *each tree node $N_{h_i}$ (where $h_i$ is the node level)* **do**
  Look for all descendants of $N_{h_i}$ with level $h_i + \delta$ and the sum of their areas $A_{h_i+\delta}$;
  Look for the ascendant of $N_{h_i}$ with level $h_i - \delta$ and its area $A_{h_i-\delta}$;
  $q(N_{h_i}) = \frac{A_{h_i-\delta} - A_{h_i+\delta}}{A(N_{h_i})}$;
**end**
**for** *each node $N_{h_i}$* **do**
  Get $q_{h_{i+1}}$ the minimal stability of the descendants of $N_{h_i}$ with level $h_{i+1}$;
  Get $q_{h_{i-1}}$ the stability of the $N_{h_i}$'s ascendant with level $h_{i-1}$;
  **if** $q(N_{h_i}) < q_{h_{i+1}}$ *and* $q(N_{h_i}) < q_{h_{i-1}}$ **then**
    Select Node $N_{h_i}$;
  **end**
**end**
**for** *all non selected tree nodes $N$* **do**
  Merge the node $N$ with its father;
**end**
Associate to each triangle the index of the node with highest level containing it;

## 3.2. Triangle Classification

Each selected node being given an index $l$, algorithm 2 yields a triangle classification $L$ which, with any triangle $t$ of the mesh, associates the label of the highest node containing the triangle (ie the label of the node with highest level containing the triangle). Because of the tree structure, it may occur that two triangles with label $l$ are not connected by triangles with label $l$. Then the node must be split into different parts with different labels.

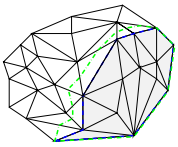


Figure 2. Extracting the border of a region (the region border with no interpolation at all is shown in the blue, and the interpolated region border is the dotted green line). Notice that a part of both borders coincide with the mesh border

### 3.3. Border Extraction from selected region

Extracting the borders of each selected region is a simple task using the available list of triangles for each node. From this list one can extract those which share an edge with a triangle of lesser level. This yields a set of border edges which can be linked. Since we are dealing with meshes built on raw point sets with a triangulation method which does not fill in the scanning holes, the component border line may encounter a scanning hole. To extract the line, we first extract the set $B$ of edges belonging to at least one triangle of the connected component which is either a hole border or a component border. Starting from an edge of $B$ which is not a hole border, a line is extended by linking edges from $B$. This way only closed contour lines are built, which are not hole borders, but can partially coincide with hole borders.

## 4. Implementation and results

In the experiments herewith, the function $H$ on the mesh will be the mean curvature. Choosing the levels $h_i$ is another question. Since curvatures are real numbers and are estimated only up to a given estimation error depending on the curvature estimation method, using all levels for the $h_i$ would not be a good solution. The adopted solution is to use equally spaced bins and $\delta$ equal to the quantization step.

The Mesh-MSER algorithm was implemented in C++. On a $1.5$ GHz dual core laptop, without any particular effort on code optimization, the whole process lasts for less than one minute for a $500000$ vertices mesh. All shapes presented in this section (with the exception of the Stanford Urbis Romae pieces) were acquired using a triangulation laser scanner which yielded a high precision dense point cloud (with some acquisition holes though, as can be noticed on fig. 4). The non-oriented point cloud was first oriented, its curvature computed at all raw points, and an interpolating mesh built using the method described in [12].

### 4.1. Results on mechanical and geometrical shapes

The first experiment is a sanity check on simple a diamond shaped volumetric pattern (fig. 3) with $150k$ vertices and $400k$ triangles. Mesh-MSER surrounds all geometrically relevant areas, namely the facets and a single connected region containing all vertices. It could be objected

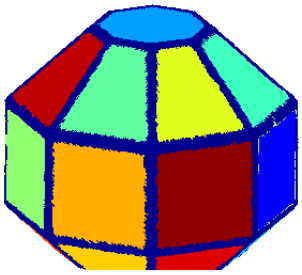
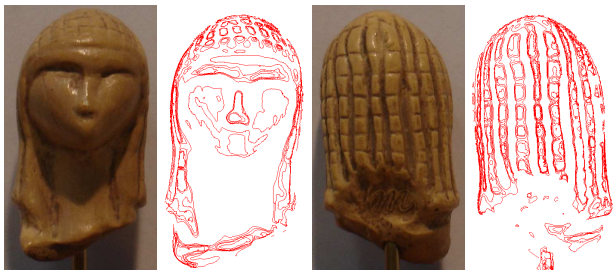


Figure 3. Classification by Mesh-MSER Analysis of a diamond shaped pattern (400k triangles).



(a) Picture (b) MSER (c) Picture (d) MSER

Figure 5. Maximally stable curvature level lines of "La dame de Brassempouy". Some lines on the above figure appear to be open because of the cropping into front and back part)

that a single threshold on the curvature would have sufficed to obtain the facets. But, even in that simple case, it was not obvious to predict the right curvature threshold. Furthermore, a simple curvature threshold would have delivered many small extremal regions due to noise inside the facets, which are actually fused to the facets by Mesh-MSER. The second industrial example is a mesh acquired from a water pump ($2.5$ million vertices, $4$ million triangles), whose the mesh was again built directly on the raw data. This object has many acquisition holes (see fig. 4). The final classification gives some $200$ regions. For better visualization random colors were given to the regions. The algorithm automatically separated edges from plane or curved parts. The segmentation of such a huge cloud into only 200 regions promises to enable a further model analysis, facet by facet.

### 4.2. Archeological pieces: comparative results

The next test (fig. 5) was performed on a good quality mould of an archaeological piece, which was subjected to a massive scan followed by Mesh-MSER. Mesh and curvatures were obtained using [12]. This small prehistorical figurine (23.000 B.C.), "La Dame de Brassempouy" is only 2 centimeters tall. The mesh has approximately $300k$ vertices and $500k$ triangles. Notice how each detail of the hair dress is segmented out.

This preliminary exploration of the capabilities of Mesh-MSER was continued on the Stanford Forma Urbis Romae

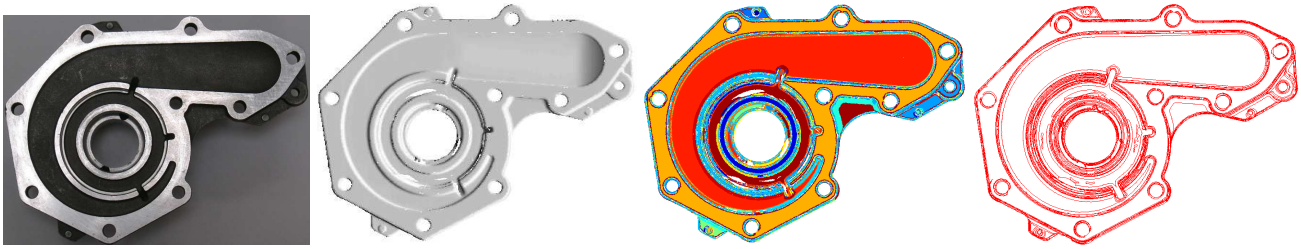


Figure 4. Classification by MSER Analysis (From top to bottom: picture of the object; obtained mesh; MSER segmentation; MSER borders)

database, containing hundreds of archaeological artifacts coming from a broken stone map of Roma (see [20]). A challenge of this project is to solve the jigsaw puzzle and rebuild the map. It is a crucial test for the Mesh-MSER method to check whether or not it extracts the engraved symbols and drawings figuring the town map, and whether it does it better than what can be done with 2D-MSER or with a Canny edge detector from simple photographs. Pictures of fragments 10g and 31u are given along with the result of MSER extraction on figs 7 and 6. The experiment of fig. 6(b) shows Mesh-MSER working on these engravings with a high performance, comparable to the best 2D image MSER performance on pictures containing high contrasted trademarks and logos [25]. Indeed, all visible symbols and all features of the map plan are faithfully extracted, with very few outliers.

This experiment can be pushed further. Indeed, the pieces being rather flat, a direct comparison of 2D- and Mesh-MSER on their main facet makes sense. The Mesh-MSER result compares advantageously to 2D level line or edge extraction methods applied to a picture of the same object (fig. 6 (a)). The comparison shows that it is far more reliable to detect boundaries on the 3D mesh.

Finally, several strategies for extracting curvature level lines on meshes are compared on fig. 7 with fragment 10g of the Stanford Urbis Romae database. Here again the Mesh-MSER results seem to be complete, accurate, and without outliers. The experiment compares the choice of curvature level lines made by MSER with a simple threshold based on level line length. Although this choice indeed removes noisy level lines, it also loses many meaningful ones, and adds anyway a spurious threshold parameter.

To compare the obtained results with those presented in [36], the same data set point (a fragment of antique vase) was used. The mentioned reference is very similar in scopes to MSER: it proposed to make a sort of two scale analysis on a mesh by defining a "base" and "height function". The lines shown in [36] are level lines of the height function. The base is implicitly defined by its gradient, by a sophisticated variational procedure. Here we used a similar height function to get a relevant comparison. The height function is defined as the difference between the surface and its smoothed out version by a large scale mean curvature motion. The Mesh MSER method can then be directly applied (fig. 8) on this scalar function.

## 5. Conclusion

This paper introduced a fast level set tree method, Mesh-MSER, applicable to any scalar function defined on a mesh. This method is a direct extension of classic 2D image analysis tools building trees of level sets components or of level lines. Using the fact that the curvature is the most straightforward scalar function defined from and on a mesh, the method was used to segment meshes into maximally stable extremal regions (MSERs) of the curvature. Future work will focus on the exploitation of this structure. Indeed the experiments clearly point out the possibility of using the detected curves and regions to perform pattern recognition of complex objects such as the Urbis Romae fragments. On the other hand the method provides automatic segmentations of industrial objects into edge parts and parts with constant or slowly varying curvature, for which spline or conical models should easily be estimated.

**Acknowledgements** Pictures and scans of the Stanford Forma Urbis Romae are used with permission of Professor Marc Levoy (Stanford Digital Forma Urbis Romae Project). The vase model is property of Laboratory of Computer Graphics & Multimedia at the Technion and the Zinman Institute of Archaeology, University of Haifa.

## References

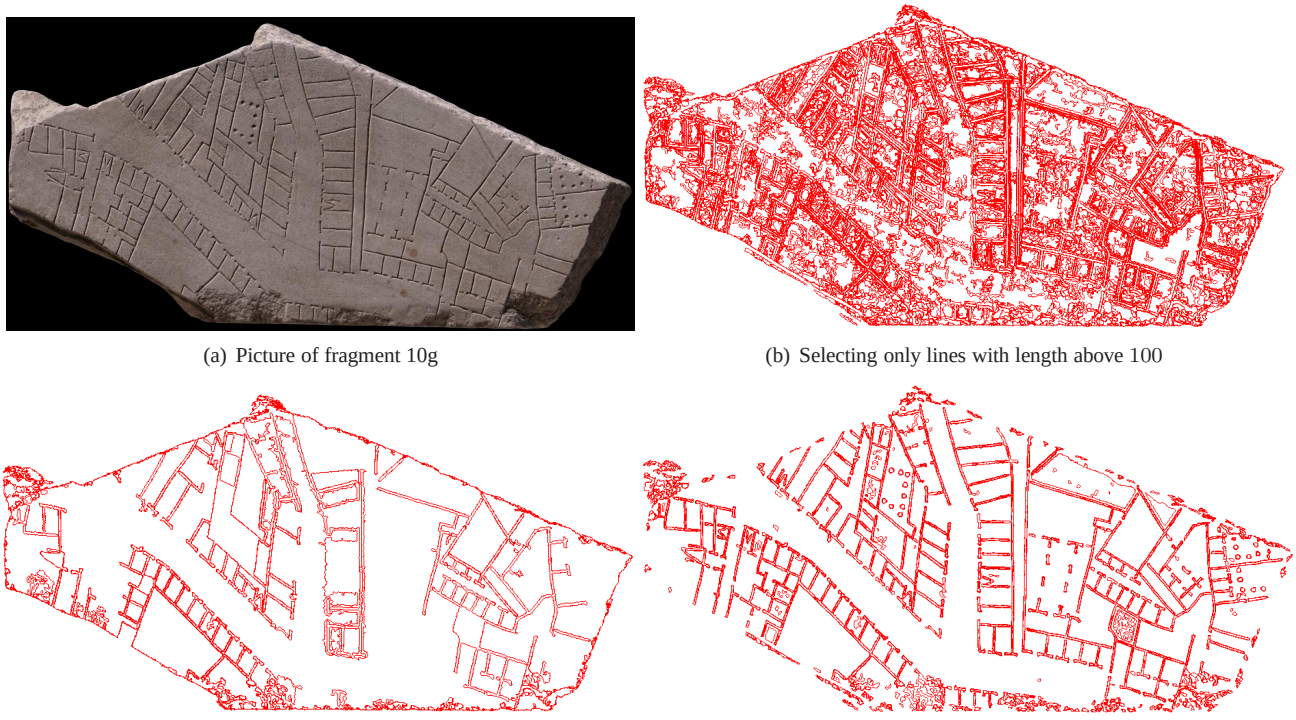
[1] C. Ballester, V. Caselles, and P. Monasse. The tree of shapes of an image. *ESAIM: COCV*, 9:1–18, 2003. 1, 2

[2] A. Belyaev and E. V. Anoshkina. *Mathematics of Surfaces XI, IMA Conf. on Mathematics of Surfaces*, chapter Detection of Surface Creases in Range Data. Springer, 2005. 1

[3] F. Cao, P. Musé, and F. Sur. Extracting meaningful curves from images. *J. Math. Im. Vis.*, 22(2-3):159–181, 2005. 2

[4] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *SODA '00*, pages 918–926, Philadelphia, PA, USA, 2000. SIAM. 2

[5] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. In *SGP '03*, pages 177–187, 2003. 1



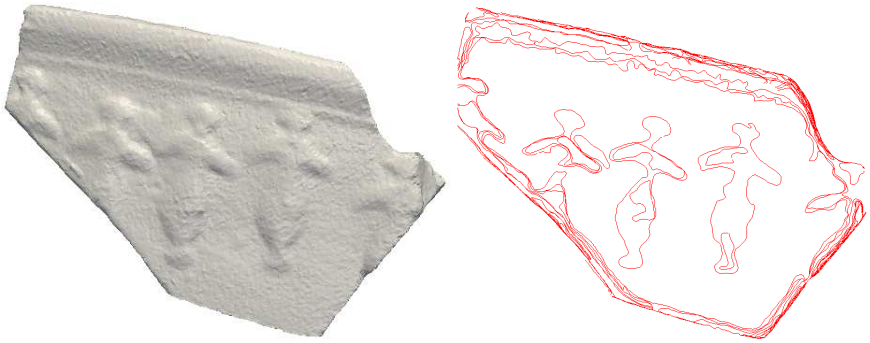
(a) Picture of fragment 10g

(b) Selecting only lines with length above 100

(c) Selecting only lines with length above 10000

(d) MSER Selection

Figure 7. Comparison between several strategies for extracting level lines: a) Picture of fragment 10g of Stanford FUR database ; b) and c) level lines with length above a given threshold; d) Mesh-MSER: its selection is definitely much more accurate, misses no apparent detail and gives very few outliers



(a) Original object

(b) Mesh-MSER selection

Figure 8. Result of Mesh-MSER on a vase model. Compare with results provided in [36] and [21]: results segment the shape into the relief and the base .


[6] W. Che, J.-C. Paul, and X. Zhang. Lines of curvature and umbilical points for implicit surfaces. *Comput. Aided Geom. Des.*, 24(7):395–409, 2007. 2

[7] J. I. Daniels, L. K. Ha, T. Ochotta, and C. T. Silva. Robust smooth feature extraction from point clouds. In *SMI '07*, pages 123–136, Washington DC, 2007. IEEE. 1

[8] J. I. Daniels, L. K. Ha, T. Ochotta, and C. T. Silva. Spline-based feature curves from point-sampled geometry. *Vis. Comput.*, 24(6):449–462, 2008. 1

[9] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003. 1

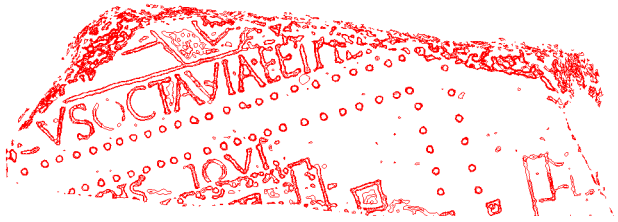
[10] K. Demarsin, D. Vanderstraeten, T. Volodine, and D. Roose. Detection of closed sharp feature lines in point clouds for reverse engineering applications. In *GMP06*, pages 571–577, 2006. 1

[11] A. Desolneux, L. Moisan, and J. Morel. Edge detection by Helmholtz principle. *J. Math. Im. Vis.*, 14(3):271–284, 2001. 2, 8

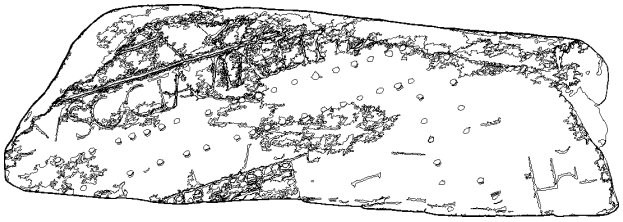
[12] J. Digne, J. Morel, C. Mehdi-Souzani, and C. Lartigue. Scale space meshing of raw data point sets. preprint CMLA - ENS Cachan, October 2009. 5

[13] M. Donoser and H. Bischof. Efficient maximally stable extremal region (MSER) tracking. In *IEEE CVPR 2006*, vol-

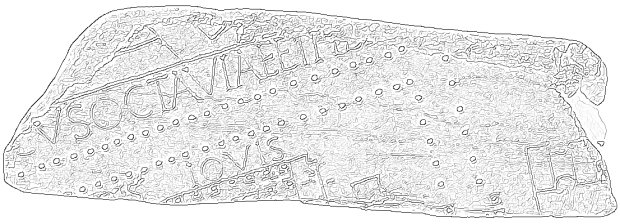
(a) Original picture of fragment 31u



(b) Mesh-MSER result



(c) Meaningful level lines [11] of the original picture



(d) Canny edge detector on the original picture

Figure 6. Picture (a) of fragment 31u of Stanford FUR database and its Mesh-MSER result (b). This result can be compared with the 2D-MSER result (c) on a good quality picture of the same fragment [11], and with the result of a Canny edge detector applied to the same picture (d). The comparison gives a sweeping advantage to Mesh-MSER. Indeed, 2D-MSER misses parts and keeps noisy level lines. Canny's detector has many outliers and yields anyway no segmentation. Similar experiments on artificial renderings of the mesh gave no better results


ume 1, pages 553–560, 2006. 2

[14] S. Gumhold, X. Wang, and R. McLeod. Feature extraction from point clouds. In *Proc. 10th International Meshing Roundtable*, 2001. 1

[15] K. Hildebrandt, K. Polthier, and M. Wardetzky. Smooth feature lines on surface meshes. In *SGP '05*, page 85, 2005. 1

[16] A. Hubeli, K. Meyer, and M. Gross. Mesh edge detection. In *Proc. Workshop Lake Tahoe*, Lake Tahoe City, California, USA, January 2000. 1

[17] V. Interrante, H. Fuchs, and S. Pizer. Enhancing transparent skin surfaces with ridge and valley lines. In *VIS '95*, page 52, Washington DC, 1995. IEEE Computer Society. 1

[18] T. Judd, F. Durand, and E. Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3):19, 2007. 1

[19] E. Kalogerakis, D. Nowrouzezahrai, P. Simari, and K. Singh. Extracting lines of curvature from noisy point cloud. *Computer-Aided Design*, 41(4):282 – 292, 2009. 2

[20] D. Koller, J. Trimble, T. Najbjerg, N. Gelfand, and M. Levoy. Fragments of the city: Stanford's digital Forma Urbis Romae project. In *Proc. Third Williams Symposium on Classical Architecture, Journal of Roman Archaeology Suppl. 61*, pages pp. 237–252, 2006. 6

[21] M. Kolomenkin, I. Shimshoni, and A. Tal. On edge detection on surfaces. In *CVPR*, pages 2767–2774, 2009. 2, 7

[22] C. Lange and K. Polthier. Anisotropic smoothing of point sets. *Comput. Aided Geom. Des.*, 22(7):680–692, 2005. 1

[23] R. Lengagne, P. Fua, and O. Monga. Using crest lines to guide surface reconstruction from stereo. In *ICPR '96: Volume I*, page 9, Washington, DC, USA, 1996. IEEE. 1

[24] T. Maekawa, F.-E. Wolter, and N. M. Patrikalakis. Umbilics and lines of curvature for shape interrogation. *Comput. Aided Geom. Des.*, 13(2):133–161, 1996. 2

[25] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *In British Machine Vision Conference*, volume 1, pages 384–393, 2002. 2, 6

[26] E. Meinhardt, E. Zacur, A. F. Frangi, and V. Caselles. 3D edge detection by selection of level surface patches. *J. Math. Im. Vis.*, 34(1):1–16, 2009. 2

[27] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. Discrete differential geometry operators for triangulated 2-manifolds. In *International Workshop on Visualization and Mathematics*, 2002. 1

[28] P. Monasse and F. Guichard. Fast computation of a contrast-invariant image representation. *IEEE I.P.*, 9:860–872, 1998. 1, 2, 3, 4

[29] L. Najman and M. Couprie. Quasi-linear algorithm for the component tree. In *In SPIE Vision Geometry XII*, pages 98–107, 2004. 2, 4

[30] Y. Ohtake, A. Belyaev, and H. Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3):609–612, 2004. 1

[31] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces. In *Computer Graphics Forum*, volume 22, pages 281–289, september 2003. 1

[32] S. Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *3DPVT '04*, pages 486–493, Washington DC, 2004. 1

[33] G. Stylianou and G. Farin. Crest lines for surface segmentation and flattening. *IEEE TVCG*, 10(5):536–544, 2004. 1

[34] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *ICCV '95*, page 902, Washington DC, 1995. 1

[35] S. Yoshizawa, A. Belyaev, and H. Seidel. Fast and robust detection of crest lines on meshes. In *SPM '05*, pages 227–232, New York, 2005. ACM Press. 1

[36] R. Zatzarinni, A. Tal, and A. Shamir. Relief analysis and extraction. *ACM Trans. Graph.*, 28(5):1–9, 2009. 2, 6, 7