Pacific Graphics 2009
S. Lee, D. Lischinski, and Y. Yu
(Guest Editors)

Volume 28 (2009), Number 7

Mesh Segmentation Refinement

Lotan Kaplansky and Ayellet Tal

Technion – Israel Institute of Technology

Abstract

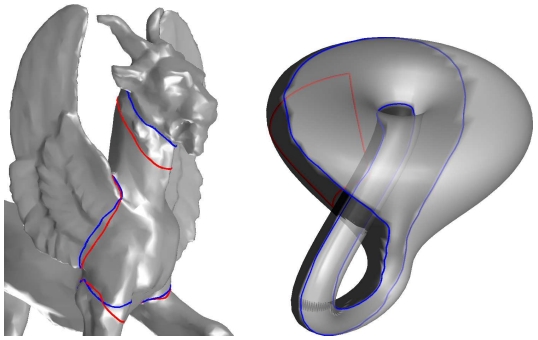
This paper proposes a method for refining existing mesh segmentations, employing a novel extension of the active contour approach to meshes. Given a segmentation, produced either by an automatic segmentation method or interactively, our algorithm propagates the segment boundaries to more appropriate locations. In addition, unlike most segmentation algorithms, our method allows the boundaries to pass through the mesh faces, resulting in smoother curves, particularly visible on coarse meshes. The method is also capable of changing the number of segments, by enabling splitting and merging of boundary curves during the process. Finally, by changing the propagation rules, it is possible to segment the mesh by a variety of criteria, for instance geometric-meaningful segmentations, texture-based segmentations, or constriction-based segmentations.

1. Introduction

Mesh segmentation is an important problem in computer graphics, with applications as diverse as modeling, animation, retrieval, texture mapping, and more [AKM*06, Sha08]. While many algorithms show appealing results, some result in segment boundary curves that pass in undesirable locations. Moreover, these curves are often jagged, being constrained to pass through the vertices. Some algorithms apply additional post-processing, such as minimal graph-cuts, in an attempt to refine the segmentation curves. However, the curve positions often remain subpar and the effect of constraining the curve to the edges cannot be overcome.

This paper proposes an algorithm for advancing and deforming segment boundaries directly – aiming at shorter, smoother, and better feature-aligned curves – while respecting the underlying mesh geometry. It is able to overcome the above mentioned drawbacks, by attracting the curves to relevant mesh features, such as local concavity, while allowing them to cut through the mesh edges. This refinement is most evident on coarse or irregular meshes, in which the mesh triangulation inherently introduces curve jaggedness.

Our approach is based on the *active contour* model, aka *snakes*. An active contour is a dynamic curve that deforms by minimizing an energy functional, relevant to the object's features and to the application. For instance, in "meaningful" segmentation, concavity and boundary length are considered important features (Figure 1(a)), while in color-based mesh segmentation, only the assigned color at each vertex should influence the boundary position (Figure 1(b)).



(a) Refining [LLS*05] (b) Segmentation-by-color

Figure 1: *Refining segmentations by different criteria (the initial curves are in red and the resulting curves are in blue). Left: meaningful segmentation of the geometry; note how the neck/head boundary is advanced to a better location. Right: segmentation by color of a non-orientable Klein bottle.*

Some previous active contour schemes for mesh segmentation are *parameterization-dependent*, i.e., they associate the mesh with a planar parameterization, and apply the algorithms on this plane [LL02, LLS*05]. This flattening process can introduce deformations. In contrast, our method is

parameterization-free, avoiding this problem by advancing the curve directly on the mesh. As such, it provides a general framework for deforming mesh curves directly, in a manner that is respectful of the underlying mesh geometry.

The contribution of this paper is hence threefold. First, instead of a new segmentation algorithm, we propose a method of refining an existing segmentation (Section 4). This method simultaneously propagates the segment boundaries to better locations, creating smoother, shorter boundaries regardless of the underlying triangulation. The method also handles segment merging and splitting automatically.

Second, this paper provides a general approach for computing level-set flows on meshes. To this end, it presents algorithms for performing gradient and divergence operators on the mesh vertices (Section 5). These operators can find uses in many other applications.

Last but not least, the method's utility is demonstrated in other segmentation tasks – interactive segmentation, texture-based segmentation, and constriction detection (Section 7).

2. Related work

Many algorithms have been recently suggested for shape-based mesh segmentation [AKM*06, Sha08]. These include clustering techniques [KT03, LZ04, ZH04, KLT05] and variants thereof [GF08], random walks [LHMR09], skeleton-based methods [MPS*04], and snake-based methods [LL02, LLS*05, MBV97, JK04], to name a few.

Most of these algorithms, particularly clustering algorithms, define the segment boundaries implicitly, by the segments themselves. The active contours based methods, on the other hand, extract the boundaries first, and define the segments implicitly. This work focuses on the latter.

A few attempts have been made to use active contours for segmentation of meshes. In [LL02, LLS*05] a *parameterization-dependent* curve deformation is performed on an underlying parameterization plane. This plane is determined at each step as a 2D projection of the mesh surface. Although their results are appealing, the projection deformations might introduce inaccuracies.

In [SK07], the parameterization plane is given and the flow calculation is accurate. In real applications, however, the input mesh typically lacks such a parameterization. Furthermore, since closed 2-manifold surfaces are not homeomorphic to a disk, the parameterization typically requires the "tailoring" of (at least) two patches, a non-trivial task.

In contrast, [MBV97, JK04] perform snake-based segmentation utilizing a *parameterization-free* approach, which advances the curve directly on the mesh. These attempts restrict the curve to reside only on the mesh edges, and update the supporting vertices at each iteration. Since the curve accuracy is limited by the mesh resolution, mesh subdivision is required to increase the curve's accuracy. [LMLR06] focuses on reliefs. and allows the curve points to reside anywhere on the mesh faces.

This paper proposes a general snake-based algorithm for segmentation refinement, which avoids the above drawbacks by being both parameterization-free and implicit.

3. Preliminaries

Given a mesh M and its initial segmentation (produced either interactively or automatically by a segmentation algorithm), our goal is to modify this segmentation according to a given rule. This modification may not only change the segment boundaries, but also merge segments or split them. Moreover, while most segmentation boundaries do not pass through the mesh faces, we allow them to. This section details the definitions required for the algorithm's formulation.

Commonly, mesh segmentation is defined as the union of disjoint sets of mesh faces [KT03]. We will give a different definition, which relies on the boundaries between segments, and thus allows the curves to pass through the mesh faces.

Definition 3.1 Mesh curve (curve) $\mathscr{C}$: A 3-dimensional polygon whose vertices lie on the mesh edges, and whose edges reside within the mesh faces.

Definition 3.2 Supporting edges: The set of mesh edges the curve passes through.

Definition 3.3 Mesh segmentation: $\mathscr{C}_1, \mathscr{C}_2, \ldots \mathscr{C}_k$ is a segmentation of M iff $\forall i, 1 \leq i \leq k, \mathscr{C}_i$ is a mesh curve, $\forall i \neq j, 1 \leq i, j \leq k, \mathscr{C}_i$ and $\mathscr{C}_j$ do not intersect, and the removal of any $\mathscr{C}_i$, $1 \leq i \leq k$, decreases the number of disjoint segments defined by $\mathscr{C}_1, \mathscr{C}_2, \ldots \mathscr{C}_k$.

A mesh curve can be represented and manipulated *explicitly* through its intersections with its supporting mesh edges – *snaxels* [BWK05]. While intuitive, this representation requires special splitting and cleaning operations to keep the curve valid when moving snaxels across vertices. Moreover, curve topology changes such as splitting and merging, need to be checked for and addressed at every iteration.

To avoid these limitations, we extend the *level-set* approach for images [CKS97] to meshes. In this framework the curves are defined and advanced *implicitly*, ensuring curve validity and eliminating the need for consistency checks, while handling splitting and merging automatically.

Definition 3.4 Level-set function U: A scalar function defined on the mesh, whose zero level-set coincides with the mesh curves.

An update to U implicitly propagates its zero level-set and hence the curves. This definition automatically handles curve movement across vertices, splitting, and merging. Note that this implicitly constrains all $\mathscr{C}_i$ to be closed.

It suffices to assign values of U at the mesh vertices, and linearly interpolate them across the mesh. This defines

the curves implicitly, setting their supporting edges as those linking vertices of opposite sign in U.

Curves sometimes propagate with respect to specific external features on the given mesh. One way to relate these features to the curve propagation is by defining an *indicator function* g on the mesh, as follows:

Definition 3.5 Indicator function g: A scalar function in the range [0,1], defined on the mesh, approaching 0 near the mesh features and 1 far from the features. (As before, it suffices to assign values to g at the mesh vertices.)

Evidently, the definition of the indicator function depends on the mesh features that the application attempts to find, and can have a profound effect on its output. For mesh segmentation, it should depend on the surface curvature [Bie87].

Definition 3.6 Level-set flow: An update equation to the level-set function $\Delta U = time\_step \times F(U,g)$, which implicitly advances the current curves $\mathscr{C}_1, \mathscr{C}_2, \ldots \mathscr{C}_k$.

Explicit curve flows can be uniquely (up to an isometry) represented by the form $\frac{\partial \mathscr{C}_i}{\partial t} = \beta \vec{\mathscr{N}}$ [CKS97], where $\frac{\partial \mathscr{C}_i}{\partial t}$ is the time derivative of curve $\mathscr{C}_i$ and β is the propagation speed in direction $\vec{\mathscr{N}}$, which is the unit normal to the curve on the tangent plane of the surface at each point [Do 76].

A simple geometric derivation shows that when the flow is given in its explicit form $\frac{\partial \mathscr{C}_i}{\partial t} = \beta \vec{\mathscr{N}}$, the equivalent level-set representation is given as $\Delta U = time\_step \times \beta |\nabla U|$ [CKS97]. Hence, $F(U,g) = \beta |\nabla U|$, with ∇U representing the gradient of U on the surface and β is expressed via the level-set function U and the indicator function g. The exact form of β depends on the application at hand.

4. Segmentation refinement algorithm

This section describes the segmentation refinement algorithm, whose goal is to better adhere the segment boundaries to their appropriate locations on the mesh, based on the minima rule (concavity) as well as boundary length [Sha08]. The algorithm realizes a specific flow from Definition 3.6. Outlined in Algorithm 1, its inputs are the initial segment boundaries $\mathscr{C}_1, \mathscr{C}_2, \ldots \mathscr{C}_k$ at time $t = 0$ and β's formulation.

Algorithm 1 Segmentation refinement (flow) algorithm

1: Initialize the level-set function U, so that its zero level-set coincides with the curves $\mathscr{C}_1, \mathscr{C}_2, \ldots \mathscr{C}_k$.
2: Determine a (possibly unconnected) region Ω around the current curves.
3: Advance the level-set function U in Ω, according to the level-set flow equation (Equation 1):

$$\Delta U = time\_step \times \beta |\nabla U|. \quad (1)$$

4: From the updated U function, determine the new curves as its zero level-set.
5: Repeat Steps 2–4 until convergence.

Step 1 calculates the initial level-set function U as the signed distance to the curves. This is done by defining U at each vertex as its minimal geodesic distance to all the curves. This distance can be calculated using [MR08] or approximated by the Fast Marching method [KS98, SSK*05]. The sign of U is then set to alternate across boundaries.

Step 2 limits the region Ω over which ΔU is calculated. This step is not essential to the algorithm; the propagation could be applied on the whole mesh. Ω is used since it can significantly lower the number of vertices in which the updates must be calculated, and thus refine performance.

Step 3 is the key step of the algorithm. It computes the update to U by applying Equation 1. Computing ΔU involves the gradient operator ∇U and the divergence operator (required for β, as will be later discussed). Section 5 describes how these operators are performed directly on the mesh.

After Step 4, which determines the new curves from the updated U, the algorithm iterates until the curves do not change between subsequent iterations.

Many flows can be realized using Algorithm 1. For mesh segmentation refinement, we use the *geodesic flow*, first introduced in [CKS97] for images. This flow operates by deforming the boundary curves towards the segment boundaries, while attempting to minimize curve lengths.

The geodesic flow is derived from the minimization of an *energy functional* $J(\mathscr{C})$ defined using a curve $\mathscr{C}$ with respect to a feature indicator function g (Definition 3.5):

$$J(\mathscr{C}) = \int |\mathscr{C}'(q)| \cdot g(|\nabla I(\mathscr{C}(q))|) dq. \quad (2)$$

Here, $\mathscr{C}(q)$ is the given parameterization of the curve, I denotes the scalar quantity whose features are searched for, and the indicator function g is defined as a monotonically decreasing function of the gradient. The intrinsic curve length $\int |\mathscr{C}'(q)| dq$ is thus weighted by $g(|\nabla I(\mathscr{C}(q))|)$, which is indicative of the length significance at each curve point. In effect, this functional penalizes the curve length in the non-feature regions (having high g values).

Explicitly, it can be shown that the flow minimizing J is:

$$\frac{\partial \mathscr{C}}{\partial t} = g\kappa \vec{N} - (\nabla g \cdot \vec{N})\vec{N} = (g\kappa - (\nabla g \cdot \vec{N}))\vec{N}. \quad (3)$$

The effect of the flow is to deform the curve $\mathscr{C}$ using a combination of forces. The term $-(\nabla g \cdot \vec{N})$ attracts the curve towards regions of low g values, hence towards the features. The term $g\kappa$ (κ being the curve's local curvature) attempts to reduce the curve's overall length, thereby smoothing the curve. In regions where g is low (probable features), the feature attraction forces dominate, while around high g values (non-features) the curve shortening forces dominate. Thus, g acts as a weighting term as well as the feature indicator.

In the level-set formulation, Equation 3 is given by:

$$\Delta U = time\_step \times |\nabla U| (div(g \frac{\nabla U}{|\nabla U|})), \quad (4)$$

where div is the divergence operator applied to the vector function $g\frac{\nabla U}{|\nabla U|}$. (See [CKS97] for a derivation.)

As g should act as an attractor around areas of conceptual part boundaries, the minima rule [Bie87] suggests that it should take on low (attractive) values around the mesh's deep concavities. To achieve this, we use:

$$g = \frac{1}{1+(\frac{|K|}{\gamma})^n}, \tag{5}$$

with K chosen as the minimal surface curvature. Since convex regions typically do not influence the segmentation, K values corresponding to convex areas are set to zero. γ is a user-defined parameter, which accounts for K's dependence on the model's scale. In all our examples, $n = 2$.

Implementation issues: Two issues arise when implementing Algorithm 1. The first regards the choice of the time step at each iteration (Equation 1). The second concerns the region Ω over which U and ΔU are calculated.

The difficulty in determining the time step arises from the fact that the expressions that govern the flow speed, such as $\kappa, \nabla g \cdot \vec{N}, div(\cdot)$ etc. do not scale linearly with re-scaling of the mesh. For example, scaling the mesh by a factor of a scales κ by a^{-1}, leading to a relative speed change of a^{-2}. To compensate for this, a simple heuristic, which turns out to be satisfactory, is to set the time step relative to a small fraction (typically $\frac{1}{100}$) of the square of the curve's total length.

In general, Ω is defined as the set of vertices whose distance to the curves is less than a given value. This set can be either defined as an n-ring neighborhood around the curves, or as the set of vertices whose geodesic distance to the curves (already computed during initialization) is below a threshold. In our implementation a 7-ring neighborhood is used.

As the curves propagate into new regions, the values of U in the new vertices need to be updated accordingly. In addition, the U values in Ω are re-initialized (similarly to Step 1) every few iterations as a regularizing process. This prevents accumulation of numerical inaccuracies.

5. Calculating the gradient & divergence

To perform the geodesic flow (Equation 4), we derive the differential operators of gradient and divergence on meshes.

5.1. The gradient operator

Given a scalar function f defined on a manifold M, the gradient $\nabla_M f$ at a point p is a vector on M's tangent plane at p, with the same direction and magnitude as the maximum rate of change of f at p. If a parameterization of M is given,

$$\chi = (u, v, f^{geom}(u,v)) : \Lambda \subset \mathbb{R}^2 \rightarrow M,$$

with $\Lambda = \{(u,v)|u,v \in \mathbb{R}\}$, $\nabla_M f$ on $p \in \chi(\Lambda)$ is [Do 76]:

$$\nabla_M f = \frac{f_u G - f_v F}{EG - F^2}\chi_u + \frac{f_v E - f_u F}{EG - F^2}\chi_v, \tag{6}$$

with f_u, f_v representing the partial derivatives of f *in* Λ at p, and E, F, G are given as the coefficients of the first fundamental form of M at p.

Since such a parameterization is not given, we calculate $\nabla_M f$ at each mesh vertex v_0 by approximating its local neighborhood, as described in Algorithm 2.

Algorithm 2 Gradient calculation algorithm

1: Approximate the local geometry of M, $\hat{f}^{geom}$ at v_0.
2: Calculate the gradient of the geometry $\hat{\chi}_u$ and $\hat{\chi}_v$.
3: Approximate the local level-set function $\hat{U}$ on the local parameterization plane Λ.
4: Calculate the gradient of $\hat{U}$: $\hat{U}_u$ and $\hat{U}_v$.
5: Calculate the gradient of the level-set function$\nabla_M U$ according to Equation 6, using the approximations $\hat{U}_u$, $\hat{U}_v$, $\hat{\chi}_u$, $\hat{\chi}_v$.

Since the methods used for performing Stages 1-2 are similar to those used in Stages 3-4, we discuss only the former, which addresses the geometry of the mesh. Stages 3-4 have the function $\hat{f}^{geom}$ describing the geometry (the mesh z values) replaced by a function $\hat{U}$ describing the level-set.

Let $\underline{\hat{f}^{geom}} = (\hat{f}_0^{geom}, \cdots, \hat{f}_N^{geom})^T$ be the values of $\hat{f}^{geom}$ at v_0 and its N neighbors. Assume that v_0 is at the origin and its normal is aligned with the z-axis. We distinguish between two cases – fitting the local geometry to either a linear or a quadratic form. In the linear form, $\hat{f}^{geom}$ is found by fitting the local geometry to $\hat{f}^{geom}(u,v) = w_1 u + w_2 v + w_3$. The quadratic form, $\hat{f}^{geom}(u,v) = w_1 u^2 + w_2 uv + w_3 v^2 + w_4 u + w_5 v + w_6$, is more accurate and is used when the linear approximation does not conform well enough to the geometry.

After choosing the function type, its approximated coefficients $\underline{\hat{w}} = [\hat{w}_1 \cdots \hat{w}_d]^T$ are determined by least-squares fitting, as follows. For the linear case:

$$\begin{aligned} \hat{f}^{geom}(u,v) &= w_1 u + w_2 v + w_3 = (u,v,1)\underline{w} \\ \underline{f}^{geom} &= (f_0^{geom} \ldots f_N^{geom})^T \cong \phi^T \underline{\hat{w}} \\ \phi &= \begin{pmatrix} u_0 & \cdots & u_N \\ v_0 & \cdots & v_N \\ 1 & \cdots & 1 \end{pmatrix} \\ \underline{\hat{w}} &= (\phi\phi^T)^{-1}\phi \underline{f}^{geom} = \Phi \underline{f}^{geom}, \end{aligned} \tag{7}$$

where the coefficients matrix $\Phi \in R^{3\times(N+1)}$ represents the approximation used.

In the quadratic case, $\hat{f}^{geom}(u,v) = (u^2, uv, v^2, u, v, 1)\underline{w}$,

$$\phi = \begin{pmatrix} u_0^2 & \cdots & u_N^2 \\ u_0 v_0 & \cdots & u_N v_N \\ v_0^2 & \cdots & v_N^2 \\ u_0 & \cdots & u_N \\ v_0 & \cdots & v_N \\ 1 & \cdots & 1 \end{pmatrix}, \tag{8}$$

and Equation 7 gives $\Phi \in R^{6\times(N+1)}$.

For Step 2, we need to differentiate $\hat{f}^{geom}$. In the linear case, $\nabla \hat{f}^{geom} = (\hat{f}_u^{geom}, \hat{f}_v^{geom}) = (w_1, w_2)$ (Equation 7), hence they can be calculated by extracting the first two rows in Φ (i.e. Φ_1, Φ_2) and multiplying by $\underline{f}^{geom}$:

$$\nabla \hat{f}^{geom} = (\hat{f}_u^{geom}, \hat{f}_v^{geom}) = (w_1, w_2) = (\Phi_1^T, \Phi_2^T)^T \underline{f}^{geom}.$$

The quadratic case is similar. Since v_0 is defined at the origin, differentiating $\hat{f}^{geom}(u,v)$ results by Equation 8 in:

$$\nabla \hat{f}^{geom} = (\hat{f}_u^{geom}, \hat{f}_v^{geom}) = (w_4, w_5) = (\Phi_4^T, \Phi_5^T)^T \underline{f}^{geom}.$$

Steps 1-2 are performed only once in a pre-processing step, since the geometry of the mesh does not change. Steps 3-5 are performed at each iteration, as the level-set function keeps changing. The key property in performing Steps 3-5 is that together they result in a linear operator that can be expressed as a product of a $3\times(N+1)$ matrix W^{grad} with the function's values. Plugging Equation 7 into Equation 6 yields (we omit the complete derivation for brevity):

$$W^{grad} = \frac{1}{EG-F^2}((G\hat{\chi}_u - F\hat{\chi}_v)\Phi_1 + (E\hat{\chi}_v - F\hat{\chi}_u)\Phi_2).$$

(It can be shown that for the linear case, $E = 1 - {w_1}^2$, $F = w_1 w_2$, $G = 1 - {w_2}^2$, and similarly for the quadratic case.)

Each row in W^{grad} performs a partial derivative in the corresponding axis. Thus,

$$\nabla_M U = W^{grad} \underline{U}.$$

As all derivations above are done locally, without a need for a global parameterization, this method frees us from providing an external parameterization. Moreover, it can be shown that invariance to rigid transformations results from the construction of $\nabla_M U$. Finally, since the computation at a vertex depends on its valence, the complexity of this stage is linear wrt the number of vertices.

5.2. The divergence operator

The planar divergence operator, applied to a vector function $\vec{f}(u,v) = (f_1(u,v), f_2(u,v))$, is defined as:

$$div\,\vec{f} = \frac{\partial}{\partial u} f_1 + \frac{\partial}{\partial v} f_2. \tag{9}$$

On surfaces, the divergence can be quite tricky [Ros97]. Below we propose a simple first-order approximation.

We are given a vertex v_0 and its N 1-ring neighbors $\{v_0, v_1, \ldots v_N\}$, the respective function vectors $\{\vec{f}_i, i = 0\ldots N\}$ and vertex normals $\{\vec{n}_i, i = 0\ldots N\}$. We assume that $\vec{n}_0$ is aligned with the z-axis. We initially apply a rotation R^i to each f_i independently, so as to align $\vec{n}_i$ with the z-axis, resulting in $\vec{\varphi}_i = R^i \vec{f}_i = (\varphi_{i,1}, \varphi_{i,2}, 0)^T$.

Given a vector function $\vec{\varphi} = (\varphi_1, \varphi_2, 0)$ defined on the plane Λ, whose values are given at the vertices, we can calculate its divergence by Equation 9. In order to relate it to the divergence on the manifold χ, we associate the partial derivatives $\frac{\partial}{\partial u}\varphi_1, \frac{\partial}{\partial v}\varphi_2$ with χ using W^{grad} (Section 5.1):

$$\begin{aligned} div_M \vec{f} &= W_1^{grad} \underline{\varphi}_1^T + W_2^{grad} \underline{\varphi}_2^T = \\ &= \sum_i W_{1,i}^{grad} e_u^T R^i \vec{f}_i + W_{2,i}^{grad} e_v^T R^i \vec{f}_i = \\ &= \sum_i W_i^{div} \vec{f}_i = \langle W^{div}, \underline{\vec{f}}^T \rangle, \end{aligned} \tag{10}$$

where $e_u = (1,0,0)^T$, $e_v = (0,1,0)^T$, $W_{i,1\ldots3}^{div} = W_{1,i}^{grad} e_u^T R^i + W_{2,i}^{grad} e_v^T R^i \in \mathbb{R}^{1\times3}$, $W^{div} \in \mathbb{R}^{(N+1)X3}$, and the matrix inner product is defined as $\langle A, B\rangle = \sum_{i,j} A_{ij} B_{ij}$.

The divergence operator has several benefits. It is performed in a manner respectful of the underlying geometry; it is linear; and the coefficients matrix W^{div} can be calculated once per vertex, in a pre-processing step.

5.3. Approximation accuracy

To assess the accuracy of our approximations, we measured the error between the approximate and analytical gradient on a number of model test cases. The geometries used are identical to those in [Xu04], with the scalar function set as $f = xy$. The experiments were performed on two manifolds: $z = sqrt(1-(x-0.5)^2-(y-0.5)^2)$ and $z = tanh(9y-9x)$.

Our gradient approximation was compared both to the analytical gradient and to the DEC approximation [Hir03] using both regular triangulations and unstructured, non-uniform tessellations. Figures 2-3 relate the average error magnitude, *err*, to the average edge length *h*, for the first manifold. The results for the other manifold are similar.

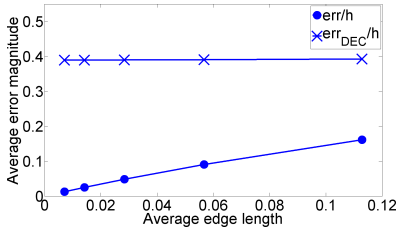


Figure 2: *Error convergence on a structured mesh*

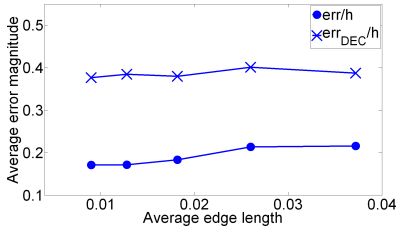
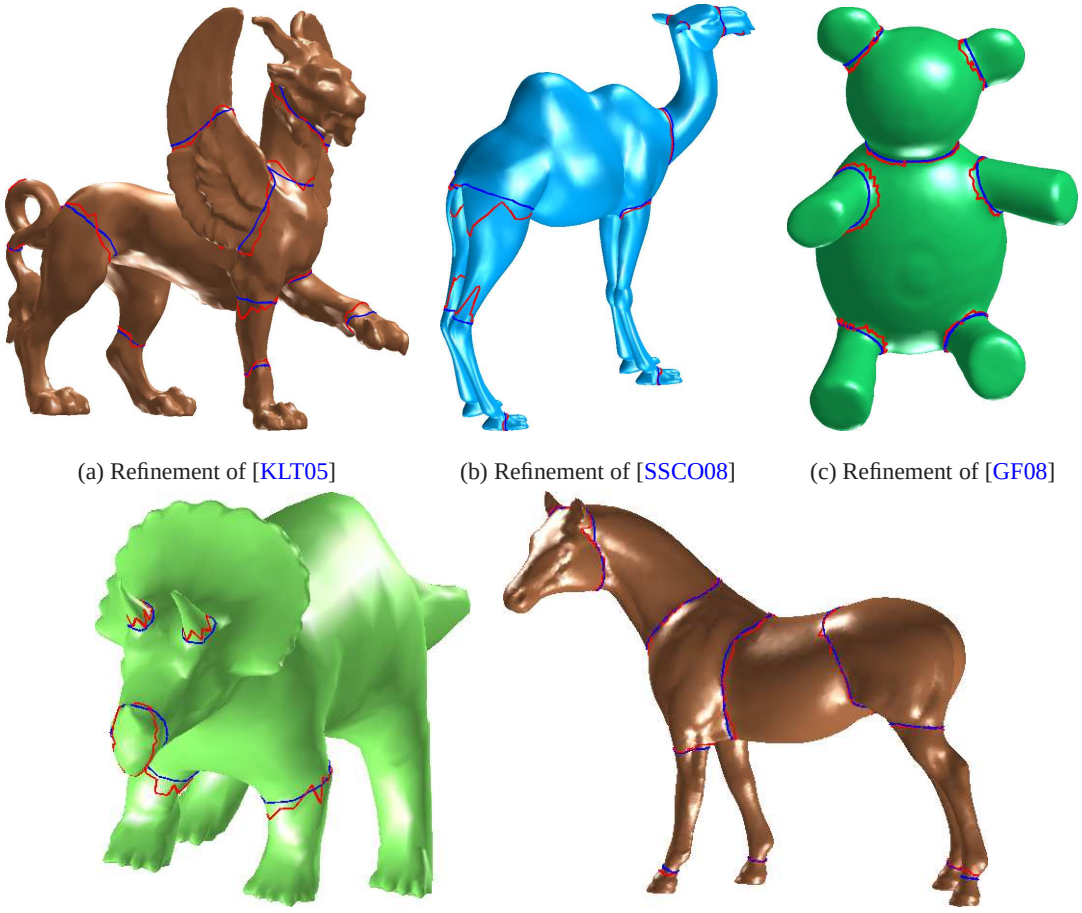


Figure 3: *Error convergence on an unstructured mesh*



(a) Refinement of [KLT05] (b) Refinement of [SSCO08] (c) Refinement of [GF08]

(d) Refinement of [ABBP05] (e) Refinement of [ZL05]

Figure 4: *Refinements of segmentation boundaries produced by other segmentation algorithms (original – red, refined – blue). The boundaries were not only smoothed but also propagated to better locations.*

It is clear that as the mesh is made finer, the error of our approximation converges to zero at least linearly with h, even on the irregularly-sampled meshes.

Moreover, comparison of the results to DEC shows that our method results in a better approximation. DEC proposes a thorough, coordinate-free formulation for mesh differential operators [MDSB02, Hir03, ES06] . Our approach focuses on the gradient and divergence. Similarly to DEC, our operators are coordinate-free, parameterization independent, pose invariant, linear wrt the vertex values, and easy to implement. However, in addition to producing a better approximation, our scheme is free from DEC's orientability requirement, as illustrated by the Klein bottle in Figure 8.

6. Results

Figures 1 and 4 show examples of segmentation refinements. The initial boundary curves are produced by known segmentation algorithms [ABBP05, GF08, KLT05, LLS*05, SSCO08, ZL05]. As seen, the original boundaries might not only be jagged (Figure 4), but sometimes pass through undesirable locations, as evident in the boundary between the head and the neck in Figure 1. Nevertheless, these boundaries are close enough to the appropriate location to serve as the initial curves for the refinement process. While there exist algorithms that resolve jaggedness [LLP05], they do not handle the problem of advancing the boundaries to positions commonly considered "meaningful" – concave and short.

It is important to note that some of these algorithms (e.g. [KLT05, SSCO08]) include post-processing stages in which the boundaries are refined, for instance using minimal cuts. Regardless, our algorithm is able to further refine upon their results.

Our process not only "smooths" the original curves, but also moves them to locations that better conform to the mesh

features. These refinements are more pronounced for moderately sized models (Figure 4(d)), where it is visible that the improvement gained by passing the curve through the faces more than makes up for the error introduced by the local geometry fitting. Figure 1 also demonstrates refining a segmentation produced by a parameterization-dependent flow.

An additional application of this algorithm is in interactive segmentation [FKS*04, LLS*05] which enables the user to manually segment the model in accordance with a particular goal in mind. The user quickly draws an initial coarse estimate of the boundary curves, which then deforms wrt the features relevant to this goal. Our algorithm proposes a ready solution for this problem .

Figure 5 shows an example of deforming a manually-defined curve. It also demonstrates the algorithm's ability to automatically handle changes in the curve topology, performing splitting during the flow.

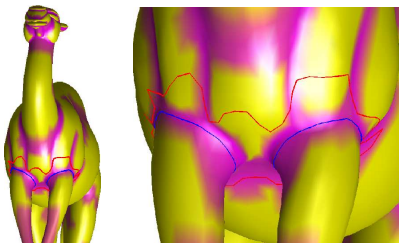


Figure 5: *Interactive segmentation: Starting from a single user-input curve (red), the curve splits, conforming to the two leg segments (blue).*

Limitations: One limitation of this process is its inability to handle instances where the segmentation graph (mapping segments to nodes) has an odd-length cycle. This stems from a sign conflict, since the curve should reside between opposite signs of U. In practice, if the regions are sufficiently far apart, limiting the size of Ω takes care of this situation. Additional limitations, inherent to the use of the level-set formulation, are the inability to handle sharp corners as well as open curves.

Performance: Curve advancement (Steps 3-4, Algorithm 1) is linear in the size of the support region Ω at each time step, owing to the linearity of the gradient and divergence operators. In our examples a single step on an Intel Core2 2.6GHz 2GB RAM, using Matlab, takes 5.4 ms, with Ω consisting of 1410 vertices (a 5K-vertex model). For a support region Ω consisting of 3135 vertices (a 20K model), it takes 14.8 ms. Convergence typically requires 1000–2000 steps.

The geodesic distances (Step 1) are approximated using Fast Marching, having a complexity of $O(N \log N)$ (N being the size of Ω). This step takes 50 ms for a 1410-vertex Ω and 130 ms for a 3135-vertex Ω, and is performed at re-initialization, typically every 20–40 advancement steps.

7. Additional applications

Our algorithm can be utilized to aid in additional applications. This section demonstrates two of these: texture segmentation on manifolds and local constriction detection.

7.1. Texture-based mesh segmentation

Most mesh segmentation algorithms are designed from a purely geometrical viewpoint – the conceptual "parts" they aim at finding are assumed to be defined only by the model's shape. However, there are cases in which the texture carries additional information about the partitioning. This can occur in models where the part's significance is better defined by its color rather than by its geometrical saliency, or when we wish to segment a texture that is already mapped on a mesh.

Our algorithm takes advantage of this additional information in a straightforward manner. By defining the edge indicator g using the texture information rather than the geometry information, the flow will conform to the texture features while still being mindful of the underlying mesh geometry. We define g using Equation 5 with K set as the angle in polar coordinates of the Lab color space of the texture.

Figure 6 shows an example of how texture information can assist facial segmentation. Starting from an initial (e.g., manually drawn) curve, the final curve settles smoothly along the lip boundaries. This enables us to overcome the inherent difficulty of this mesh segmentation task, caused by having the outer boundary of the lips lie in a convex region.

Figure 7 shows how multiple texture components are segmented on a colored sphere. Starting from a single initial curve, topology changes are automatically handled, segmenting all the gray elements. In this example, a 2D parameterization is difficult to provide, hence it demonstrates the benefit of performing the segmentation directly on the mesh.

Figure 8 compares the result of our algorithm with [SK07]. The latter is a parameterization-dependent level-set flow method, which requires a surface parameterization to be provided. Unfortunately, this requirement is hard to satisfy, since an arbitrary mesh cannot easily be mapped to a regular 2D grid. Despite the lack of a given parameterization, our algorithm is able to achieve a comparable result in terms of color separation, while converging to a smoother curve.

In Figures 7–8 an additional "external" force $\nu\vec{N}$ is added, in order to overcome the curvature-induced force in the intermediate stages of the process. This force is weighted using the g indicator to obtain: $\frac{\partial \mathscr{C}}{\partial t} = (g\kappa - (\nabla g \cdot \vec{N}) + \nu g)\vec{N}$.

7.2. Finding local constrictions

Finding local constrictions of a thin structure often plays an important part in its analysis [HA03, BWK05]. Examples include weakness detection in mechanical strength analysis, blood vessel stenosis location in arteriostenosis patients, and

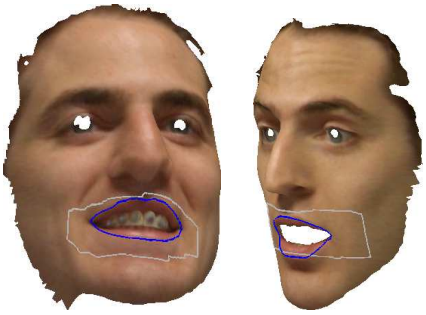


Figure 6: *Texture-based lip segmentation, starting from the white curve and converging to the blue.*

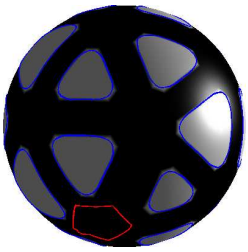
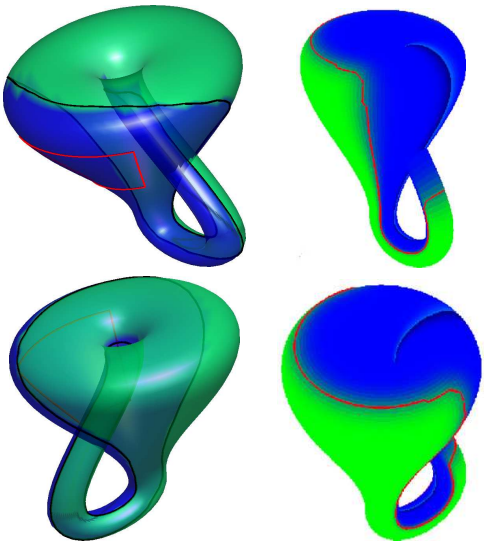


Figure 7: *Texture-based segmentation: starting from a single curve (red) and ending with multiple curves (blue).*



(a) Our method (b) [SK07]

Figure 8: *Texture-based segmentation of a (non-orientable) Klein bottle*

collagen fiber strength analysis in osteoporosis patients. We propose to use a degenerate case of the geodesic flow within the general approach of Algorithm 1, in which $g = 1$, effectively resulting in the *curvature flow* to achieve this.

In Figure 9, the user-defined curves advance while cutting through the mesh edges, unconstrained by the vertices. This illustrates cases in which multiple curves are propagated simultaneously (using one global U function), each independently settling at its locally narrowest part of the mesh.

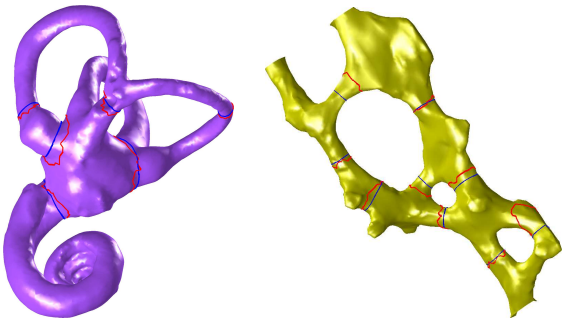


Figure 9: *Local constriction detection. The initial curves (red) are deformed until reaching a steady state (blue). Left: Inner ear model; Right: Bone section tissue.*

Figure 10 compares our result with [HA03]'s, in which the algorithm performs by progressive mesh simplification until a seed curve is located and then expanded back. By allowing the curve to cut through the mesh edges, our approach results in a smoother, shorter curve than would be possible when limiting the curve to pass through the vertices.

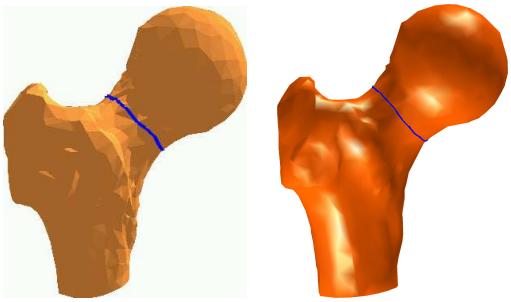
The algorithm's robustness to noise is demonstrated in Figure 11. We have performed constriction detection on increasingly noisy versions of the original model (with uniform noise added in the normal direction). The correct constrictions were located even for the very noisy version.

8. Conclusion

This paper has presented an algorithm for refining given segmentation boundaries, resulting in smoother, shorter curves, which lie nearer to surface features. It is based on an extension to meshes of the level-set approach for implicitly propagating curves. The algorithm supports automatic topology changes of the curves, while allowing them to cut through the mesh edges, significantly reducing the triangulation effects. In addition, the paper has described a technique for calculating the surface gradient and divergence operators on the mesh vertices, which can find uses in other applications.

The algorithm was applied to segmentation results produced by five state-of-the art segmentation algorithms and exhibited refinements. It was also shown that our algorithm can be used for related curve-driven applications – interactive segmentation, texture-based mesh segmentation, and local constriction detection. In these applications refinements over previous approaches was shown.

In the future, we intend to investigate other curve flows and their usefulness for additional applications. Moreover,



(a) [HA03] (b) Our method

Figure 10: *Local constriction detection on a human femur. Our algorithm detects the constriction, while [HA03] exhibits minor jaggedness in the mid section.*

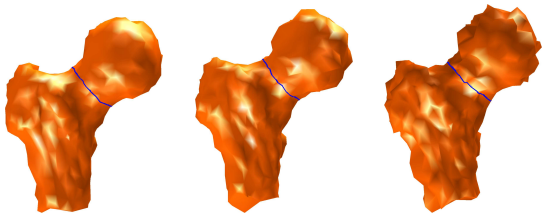


Figure 11: *Robustness to added noise in the mesh geometry*

other indicator functions, that attract the boundaries to different types of features (such as convex sharp edges) relevant for other segmentation tasks, can also be studied.

Acknowledgments: This work was supported by the Israeli Ministry of Science, Culture & Sports, grant 3-3421.

References

[ABBP05] ANTINI G., BERRETTI S., BIMBO A. D., PALA P.: 3D mesh partitioning for retrieval by parts applications. In *IEEE Multimedia and Expo* (2005), pp. 1210–1213.

[AKM*06] ATTENE M., KATZ S., MORTARA M., PATANE G., SPAGNUOLO M., A.TAL: Mesh segmentation - a comparative study. In *Shape Modeling Int.* (2006), p. 7.

[Bie87] BIEDERMAN I.: Recognition-by-components: A theory of human image understanding. *Psych. Rev. 94* (1987), 115–147.

[BWK05] BISCHOFF S., WEYAND T., KOBBELT L.: Snakes on triangle meshes. In *Bild. für die Medizin* (2005), pp. 208–212.

[CKS97] CASELLES V., KIMMEL R., SAPIRO G.: Geodesic active contours. *Int. J. Comput. Vision 22*, 1 (1997), 61–79.

[Do 76] DO CARMO M.: *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[ES06] ELCOTT S., SCHRÖDER P.: Building your own DEC at home. In *ACM SIGGRAPH Courses* (2006), pp. 55–59.

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. *ACM Trans. Graph. 23*, 3 (2004), 652–663.

[GF08] GOLOVINSKIY A., FUNKHOUSER T.: Randomized cuts for 3D mesh analysis. *ACM Trans. Graph. 27*, 5 (2008), 145.

[HA03] HÉTROY F., ATTALI D.: Detection of constrictions on closed polyhedral surfaces. In *Data Vis.* (2003), pp. 67–74.

[Hir03] HIRANI A.: *Discrete Exterior Calculus*. PhD thesis, California Institute of Technology, 2003.

[JK04] JUNG M., KIM H.: Snaking across 3D meshes. In *12th Pacific Graphics* (Oct. 2004), pp. 87–93.

[KLT05] KATZ S., LEIFMAN G., TAL A.: Mesh segmentation using feature point and core extraction. *The Visual Computer 21*, 8-10 (2005), 865–875.

[KS98] KIMMEL R., SETHIAN J.: Computing geodesic paths on manifolds. *Proc. of Natl. Acad. Sci. 95*, 15 (1998), 8431–8435.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3 (2003), 954–961.

[LHMR09] LAI Y.-K., HU S.-M., MARTIN R., ROSIN P.: Rapid and effective segmentation of 3D models using random walks. *Computer Aided Geometric Design 26*, 6 (2009), 665–679.

[LL02] LEE Y., LEE S.: Geometric snakes for triangular meshes. *Comp. Graphics Forum 21*, 3 (2002), 229–238.

[LLP05] LI W., LEVY B., PAUL J.-C.: Mesh editing with an embedded network of curves. In *Shape Modeling Int.* (2005).

[LLS*05] LEE Y., LEE S., SHAMIR A., COHEN-OR D., SEIDEL H.: Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des. 22*, 5 (2005), 444–465.

[LMLR06] LIU S., MARTIN R., LANGBEIN F., ROSIN P.: Segmenting reliefs on triangle meshes. In *ACM symposium on Solid and physical modeling* (2006), pp. 7–16.

[LZ04] LIU R., ZHANG H.: Segmentation of 3D meshes through spectral clustering. In *Pacific Graphics* (2004), pp. 298–305.

[MBV97] MILROY M., BRADLEY C., VICKERS G.: Segmentation of a wrap-around model using an active contour. *Computer-Aided Design 29*, 22 (1997), 299–320.

[MDSB02] MEYER M., DESBRUN M., SCHRÖDER P., BARR A.: Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and Mathematics III* (2002), 35–57.

[MPS*04] MORTARA M., PATANÈ G., SPAGNUOLO M., FALCIDIENO B., ROSSIGNAC J.: Plumber: A multi-scale decomposition of 3d shapes into tubular primitives and bodies. *Solid Modeling and Apps.* (2004), 139–158.

[MR08] MACDONALD C., RUUTH S.: Level set equations on surfaces via the closest point method. *J. Sci. Computing* (2008).

[Ros97] ROSENBERG S.: *The Laplacian on a Riemannian Manifold*. Cambridge University Press, 1997.

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6 (2008), 1539–1556.

[SK07] SPIRA A., KIMMEL R.: Geometric curve flows on parametric manifolds. *J. of Comp. Physics 223*, 1 (2007), 235–249.

[SSCO08] SHAPIRA L., SHAMIR A., COHEN-OR D.: Consistent mesh partitioning and skeletonization using the shape diameter function. *The Visual Computer 24*, 4 (2008), 249–259.

[SSK*05] SURAZHSKY V., SURAZHSKY T., KIRSANOV D., GORTLER S., HOPPE H.: Fast exact and approximate geodesics on meshes. *ACM Trans. Graph 24*, 3 (2005), 553–560.

[Xu04] XU G.: Convergent discrete Laplace-Beltrami operators over triangular surfaces. In *Geometric Modeling and Processing* (2004), pp. 195–204.

[ZH04] ZHOU Y., HUANG Z.: Decomposing polygon meshes by means of critical points. In *Int. Multimedia Modelling* (2004), pp. 187–195.

[ZL05] ZHANG H., LIU R.: Mesh segmentation via recursive and visually salient spectral cuts. In *Proc. of Vision, Modeling, and Visualization* (2005), pp. 429–436.