Snakes on triangle meshes

Stephan Bischoff, Tobias Weyand, Leif Kobbelt

Lehrstuhl für Informatik VIII, RWTH Aachen, 52062 Aachen
Email: {bischoff|kobbelt}@informatik.rwth-aachen.de

Abstract. In this work we introduce a new method for representing and evolving snakes that are constrained to lie on a prescribed surface (triangle mesh). The new representation allows to automatically adapt the snake resolution to the surface tesselation and does not need any (unstable) back-projection operations. Furthermore, it enables efficient and robust collision detection and gives us complete control on the topological behaviour of the snakes, i.e. snakes may split or merge depending on the intended task. Possible applications include enhanced mesh scissoring operations and the detection of constrictions of a surface.

1 Introduction

Active contour models (snakes) have been used in a wide variety of applications in computer vision and image analysis ranging from motion capturing to image segmentation. Traditionally snakes are only applied in uniform setups, i.e. curves for segmenting images and surfaces for segmenting volume data. However, there are many applications that would benefit from more general settings. In this work we consider in particular the case of curves that are embedded on arbitrary surfaces. Note that this setup is much more complex than a simple image segmentation problem as the embedding surface can be arbitrarily curved in the surrounding three-space.

As an example consider the problem of accurately locating and measuring (e.g. vascular) constrictions [1]. In this case, the embedding surface is just the surface of the vessel. Initially, the user places a curve on this surface which runs around the vessel. If we let evolve this curve according to its curvature it will become a locally shortest curve (geodesic) and its length will be a measure for the vessel's diameter. Other application scenarios include automatic mesh partitioning and interactive mesh editing.

Early attempts for modeling embedded snakes were limited to particular applications and suffered from low accuracy due to the restriction of snaxels to mesh vertices [2] or supported only expanding fronts [3]. Only recently attempts were made to fully support geometric snakes on triangles meshes [4,5]. These approaches, however, do not offer topological flexibility and furthermore rely on an elaborate piecewise parameterization of the underlying meshes. To avoid difficulties that are due to parameterization artifacts, we base our framework on the parameterization-free active contour models presented in [6].

2 Contribution

We introduce a new representation for snakes that are embedded on a given surface. For the sake of clarity, the surface will be represented by a triangle mesh, however, the method also easily extends to arbitrary polygonal meshes. The main features of this representation are:

1. Adaptivity: The sampling of the snake automatically and locally adapts to the resolution of the triangle mesh. In particular there is no need for elaborate resampling strategies based on intrinsic properties of the snake or of the surface like curvatures which are susceptible to parameterization artifacts.
2. Collision detection and topology control: The topological behavior of the snakes can be adjusted to fit the requirements of the application. In addition to a fixed connectivity, our model also supports merging or splitting of snakes. Collision detection can be performed efficiently and robustly.
3. Robustness: Our model avoids any numerically unstable (back-) projections of snaxels onto the mesh. In particular, the snake is guaranteed to always lie exactly on the mesh.

3 Methods

3.1 Representation

We represent a snake by a (possibly open) polygon in space. However, we enforce two *consistency constraints* which guarantee that the snake is actually embedded on the underlying triangle mesh (Fig. 1):

1. The vertices of the snake *(snaxels)* have to lie on mesh edges.
2. The segments of the snake have to lie in the interior of triangles.

Furthermore, we assume that the snaxels are *oriented,* i.e. each snaxel s can be represented as

$$s = (1-d)v_{from} + dv_{to}, \quad d \in [0,1)$$

where v_{from} and v_{to} are the endpoints of the supporting edge. It is important that this orientation is consistent along the snake, i.e. that all snaxels point to the same side of the snake. In particular, no two consecutive snake segments should lie in the same triangle. Snaxels adjacent to two such segments are called *invalid* and can iteratively be removed in a *cleaning conquest* linking their respective neighboring snaxels.

Note that such a piecewise linear representation is in general sufficient to capture any detail defined on the mesh. Consider e.g. a level set of a scalar field which is evaluated at the mesh vertices. As the scalar field is interpolated linearly over each triangle, the level set is just a straight line segment on the triangle, which can be exactly represented by our model.

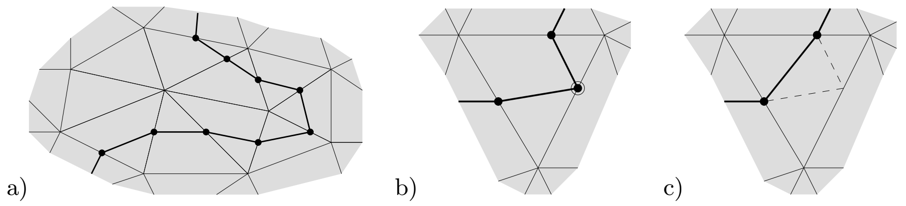

Fig. 1. Snake representation: A valid piecewise linear snake on a triangle mesh (a). Invalid snaxels (b) are removed in a cleaning conquest (c).

3.2 Evolution

The evolution of the snakes is governed by internal and external forces that are usually derived from their bending energy or the curvature distribution of the underlying mesh. Without loss of generality, we will think of the forces as a user-defined black box that assigns to each snaxel s a scalar-valued velocity v_s. We then model the snake propagation according to Huygen's principle which states that every snaxel should move along its geodesic normal, i.e. the projection of its (spatial) normal onto the tangent plane of the underlying surface. We estimate the geodesic normal at a snaxel s as the angle bisector of the two adjacent snake segments after locally flattening the configuration into a plane using a hinge map or an exponential map [7], depending on whether the snaxel lies on an edge or on a vertex, resp., cf. Fig. 2. Let α_s be the angle between the geodesic normal and the supporting edge of s. We then compute the projected velocity $\hat{v}_s$ of s along its supporting edge as

$$\hat{v}_s = \frac{v_s}{\cos \alpha_s}$$

If multiple consecutive snaxels $s_1, \ldots, s_k$ coincide at a vertex, the above computation fails, and we instead estimate the common geodesic normal from the snaxels s_0, s_1, s_{k+1} but compute the projected velocities $\hat{v}_1, \ldots, \hat{v}_k$ for each snaxel individually (depending on the angle α_i to its supporting edge). In practice, we do not need to explicitly compute the hinge or exponential map, but just sum up and normalize angles between snake segments and mesh edges. After calculating the snaxel speeds, we move the snaxels forward in time using an Euler scheme

$$d_s \leftarrow d_s + \Delta t \, \hat{v}_s \,/||e_s||$$

where e_s is the supporting edge of s and where we limit the timestep Δt such that the snaxels just do not cross mesh vertices,

$$\Delta t = min_s(1 - d_s)/\hat{v}_s$$

When a snaxel runs into a vertex of valence n, it is split into $(n-1)$ new snaxels that are put on the outgoing edges. Their distance values are reset to 0, so they all lie at the same spatial position (although their supporting edges are different). This split may result in a consistency violation, which is easily resolved by a cleaning conquest as described above.

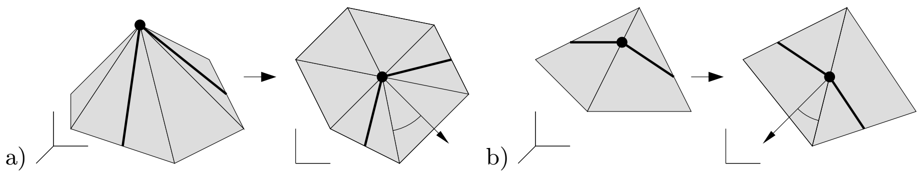

Fig. 2. Flattening. As the snakes are embedded on a surface, snaxel velocities have to be computed by locally flattening the configuration around a snaxel into a plane and then scaling the velocity magnitude according to the projection of the geodesic normal onto the snaxel's supporting edge. This is done using an exponential map (a) or a hinge map (b).

3.3 Collision detection and topology control

The consistency constraints in section 3.1 imply that two snakes (or two parts of the same snake) can only collide at snaxels, i.e. it is impossible for a snaxel to cross the interior of a snake segment. Hence collision detection can be efficiently performed by storing for each mesh edge the snaxels that lie on that edge and testing before and after each snaxel move whether the order of the snaxels on the edge has changed. If so, we determine the point of contact by linearly intrapolating the snaxel positions. In case of a collision, we may then choose depending on the application whether to join the two colliding snakes or not.

- In case of a join, we relink the snakes by removing the colliding snaxels and by connecting their predecessors and successors respectively. We then perform a cleaning conquest to remove spurious invalid snaxels (Fig. 3).
- In case of a collision, we mark the colliding snaxels as "frozen" and exclude them from the remaining update steps.

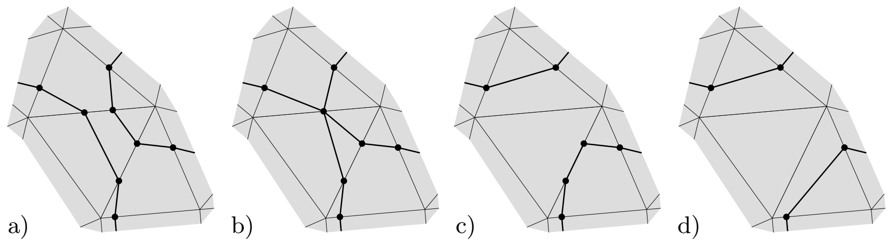

Fig. 3. Collision detection and topology control. Collisions can only happen on mesh edges (a) and hence can easily be detected (b). Two snakes can be merged by removing the colliding snaxels and relinking their neighbors (c). Spurious invalid snaxels are removed in a cleaning conquest (d).

4 Results

We have implemented our new representation and tested it on a variety of geometric models. The implementation is straightforward and runs at interactive speeds even on a standard PC. In addition to standard segmentation problems we considered the problem of detecting constrictions on a model: First the user depicts a sequence of vertices on a given input mesh. These vertices are then linked using a discrete shortest path algorithm (Dijkstra) along the edges of the mesh. The result is the initial snake which we then let evolve according to its curvature and hence minimize its length, cf. Fig. 4. The final snake is a locally shortest path whose length measures the diameter of the constriction.

As a snake can only detect local minima, we plan to distribute a number of snakes on the vessel by performing cross sections orthogonal to the skeleton of the vessel. By letting all these snakes minimize their length and selecting the shortest one it should be possible to reliably and globally detect any constriction. We also plan to exploit the topological flexibility of our snakes to evaluate the structure of whole vessel trees. Here the snake runs along the branches of the tree and splits at each furcation thereby revealing the structure of the tree.

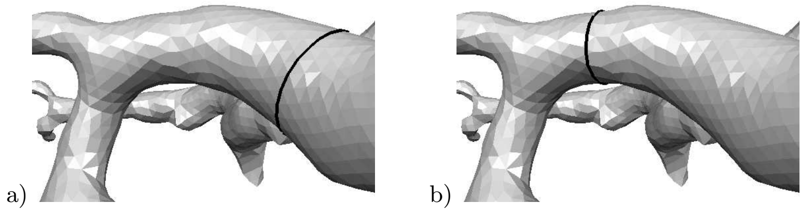

Fig. 4. Detecting constrictions. An initial snake is placed interactively around the vessel (a). After a few iterations the snake has contracted around the locally narrowest part of the vessel (b).

References

1. Hetroy, F., Attali, D.: From a closed piecewise geodesic to a constriction on a closed polyhedral surface. In: Pacific Graphics Proceedings. (2003) 394–398
2. Milroy, M.J., Bradley, C., Vickers, G.W.: Segmentation of a wrap around model using an active contour. Computer Aided Design **29** (1997) 299–320
3. Lee, H., Kim, L., Meyer, M., Desbrun, M.: Meshes on fire. In: Eurographics Workshop on Computer Animation and Simulation. (2001) 75–84
4. Lee, Y., Lee, S.: Geometric snakes for triangular meshes. Computer Graphics Forum **21** (2002) 229–238
5. Lee, Y., Lee, S., Shamir, A., Cohen-Or, D., Seidel, H.P.: Intelligent mesh scissoring using 3d snakes. In: Pacific Graphics Proceedings. (2004) 279–287
6. Bischoff, S., Kobbelt, L.: Parameterization-free active contour models. The Visual Computer **20** (2004) 217–228
7. Lee, A.W.F., Sweldens, W., Schröder, P., Cowsar, L., Dobkin, D.: Maps: Multiresolution adaptive parameterization of surfaces. In: SIGGRAPH 98. (1998) 95–104