EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (2002), Number 3

Geometric Snakes for Triangular Meshes

Yunjin Lee† and Seungyong Lee‡

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, 790-784, Korea

Abstract

Feature detection is important in various mesh processing techniques, such as mesh editing, mesh morphing, mesh compression, and mesh signal processing. In spite of much research in computer vision, automatic feature detection even for images still remains a difficult problem. To avoid this difficulty, semi-automatic or interactive techniques for image feature detection have been investigated. In this paper, we propose a geometric snake as an interactive tool for feature detection on a 3D triangular mesh. A geometric snake is an extension of an image snake, which is an active contour model that slithers from its initial position specified by the user to a nearby feature while minimizing an energy functional. To constrain the movement of a geometric snake onto the surface of a mesh, we use the parameterization of the surrounding region of a geometric snake. Although the definition of a feature may vary among applications, we use the normal changes of faces to detect features on a mesh. Experimental results demonstrate that geometric snakes can successfully capture nearby features from user-specified initial positions.

1. Introduction

Triangular meshes are widely used to represent object shapes and many techniques have been developed for processing triangular meshes. Feature detection is important in mesh processing because features can be used to specify the target region of a mesh to be processed and/or the peculiar parts to be preserved in processing. In mesh editing, the edited parts are usually the features of a mesh. In mesh morphing, features and their correspondence should be specified between two meshes. In mesh simplification and mesh compression, an important goal is to represent the features of a mesh with a small amount of data. In remeshing an irregular mesh to achieve subdivision connectivity, the resulting mesh should preserve the features of the original mesh as exactly as possible.

In spite of much research in image processing and computer vision, the problem of automatic feature detection is not yet completely solved, even for images. Semi-automatic or interactive techniques have been investigated which rely on user intervention for the detection of image features [14, 6, 7, 20, 21, 29, 22]. In these techniques, the user input is utilized for initially approximating the features and/or guiding the feature detection process. With this interactive approach, we can precisely detect the features of an image such as contours and edges with a simple user input.

In this paper, we propose a *geometric snake* as an interactive tool for detecting the features of a 3D triangular mesh. A geometric snake is an extension of the active contour model for an image called snakes [14]. After its initial position is specified by a user, a snake can automatically slither to a nearby feature by minimizing an energy functional. The energy functional consists of the internal energy to maintain the smoothness of a snake and the external energy that has local minimums at features. In this paper, we call the original snake model presented by Kass *et al*. an image snake to distinguish it from our technique proposed for a 3D mesh. A geometric snake can be used to precisely specify the features of a mesh in mesh processing, such as mesh editing and mesh morphing.

In extending an image snake to a geometric snake, two major problems involve constraining the snake movements onto the surface of a mesh and defining the features on a

† jin@postech.ac.kr, http://home.postech.ac.kr/∼jin
‡ leesy@postech.ac.kr, http://www.postech.ac.kr/∼leesy

mesh. To resolve the first problem, we use parameterization that embeds the surrounding region of a geometric snake to a 2D plane. The geometric snake is moved on the plane while minimizing its energy functional and the new position is mapped back to the surface of the mesh. The definition of a feature on a mesh may vary depending on the application. In this paper, we use the normal changes of faces to determine the feature energy functional. Since the framework of a geometric snake is independent of the definition of a feature, other feature energy functionals can be used without changing the framework.

The remainder of this paper is organized as follows. In Section 2, we review related work. Section 3 summarizes the definition and energy minimization process of an image snake. In Section 4, we give an overview of a geometric snake and its application to interactive feature detection on a mesh. Sections 5 and 6 explain the details of a snake movement and the feature energy functional, respectively. Section 7 shows experimental results. We conclude this paper in Section 8.

2. Related Work

2.1. Interactive feature specification

Snakes are an active contour model proposed by Kass *et al*., with which we can semi-automatically detect features in an image [14]. A snake is represented by a parametric curve in an image and the desired snake position is obtained by energy minimization. Several extensions of the original snake model were investigated in image processing and computer vision [6, 7, 29, 22].

In addition to parametric snake models, implicit snakes and geometric active contour models were proposed which use curve evolution and geometric flows to detect features [3, 20, 4, 30]. These models have an intrinsic nature and can automatically manage topological changes using a level set evolution technique [25]. However, with the models, it is difficult to handle open active contours as well as to incorporate additional controls such as external forces interactively specified by a user. Although the geometric contour model proposed in reference [30] was named geometric snakes, the model is concerned with 2D images and unrelated with the geometric snakes proposed in this paper which deal with 3D triangular meshes.

In computer graphics, semi-automatic feature detection techniques for images were used to provide interactive tools for image handling. Image snapping [10] gives the functionality of snapping a mouse position to the nearest feature point in an image. When applied to a sequence of user-specified points, image snapping works in a similar way to a snake and can capture a nearby feature curve. In intelligent scissors [24], feature detection is formulated as a graph search problem whereby a feature is captured by finding the shortest path from the start point to the current mouse position.

Active contour models for images were extended to extract features from 3D surfaces. Milroy *et al*. applied the original snake model to 3D surfaces for surface segmentation [23], where the snake position is updated directly on a 3D surface by searching positions with less energy. In contrast, the geometric snakes proposed in this paper determine updated positions by energy minimization on a 2D embedding plane, which is computationally efficient. Andrews [1] presented an interactive feature detection technique for 3D triangular meshes, which is based on a minimal path approach [5]. A feature curve detected by the technique is uniquely determined between the source and destination points and cannot be controlled by a user even though the curve does not capture a desired feature. In contrast, the shapes of our geometric snakes can easily be controlled by initial positioning and interactive editing in the energy minimization process.

2.2. Mesh signal processing

Recently, the application of signal and image processing techniques to 3D meshes has attracted much attention in computer graphics [17, 28]. Guskov *et al*. developed basic signal processing filters, such as low and high pass filters, for 3D triangular meshes [11]. Praun *et al*. applied signal processing algorithms to 3D meshes, such as shape blending and principal component analysis [27].

The geometric snakes proposed in this paper can be regarded as another application of an image processing technique to 3D meshes. In image processing, advanced and complex techniques, such as image segmentation and restoration, require feature detection as a preprocessing step. Geometric snakes can be used for the feature detection process, which enables advanced image processing techniques to be applied to 3D meshes. Hubeli and Gross presented a multiresolution technique that automatically extracts features from 3D triangular meshes for the purpose of a preprocessing step in mesh signal processing [13].

3. Image Snakes

An image snake is an active contour on an image which moves by minimizing an energy functional E_{snake} [14]. The functional E_{snake} contains internal and external energy terms. The internal energy E_{spline} concerns with the smoothness of a snake and minimizing E_{spline} makes a snake act like a spline curve. The external energy E_{image} is related to image features and has local minimums at the features. Hence, a snake obtained by minimizing E_{snake} is a smooth curve that passes through a feature in an image.

In this section, we summarize the energy minimization process of a snake, which will be referred to in later sections. The details of the process can be found in reference [14].

The position of a snake is represented in a parametric

form, $v(s) = (x(s), y(s))$, where $s \in [0,1]$. Then, the energy functional $E_{snake}(v)$ can be written as

$$E_{snake}(v) = \int_0^1 \left[E_{spline}(v) + E_{image}(v)\right] ds. \quad (1)$$

The internal spline energy E_{spline} is defined by

$$E_{spline} = (\alpha(s)\|v_s(s)\|^2 + \beta(s)\|v_{ss}(s)\|^2)/2, \quad (2)$$

which consists of the first- and second-order derivatives of $v(s)$ with weights $\alpha(s)$ and $\beta(s)$. The external feature energy E_{image} is determined by the types of features to be detected. For example, $E_{image} = -|\nabla I(x,y)|^2$ can be used to detect edges in an image, where $\nabla I(x,y)$ is the image gradient at a pixel (x,y).

In implementation, a snake $v(s)$ is represented by a sequence of n sample points; that is, $v(i) = (x(i), y(i))$ for $i = 1, \ldots, n$. Then, the energy E_{snake} can be minimized by solving a matrix equation;

$$\begin{aligned} \mathbf{A}\mathbf{x} + \mathbf{f}_\mathbf{x}(\mathbf{x},\mathbf{y}) &= 0 \\ \mathbf{A}\mathbf{y} + \mathbf{f}_\mathbf{y}(\mathbf{x},\mathbf{y}) &= 0 \end{aligned} \quad (3)$$

In Eq. (3), the matrix $\mathbf{A}$ comes from the finite difference method that approximates the derivatives of $v(s)$. The column vectors $\mathbf{x}$ and $\mathbf{y}$ consist of x and y positions of the snake points, respectively. The column vectors $\mathbf{f}_\mathbf{x}(\mathbf{x},\mathbf{y})$ and $\mathbf{f}_\mathbf{y}(\mathbf{x},\mathbf{y})$ represent partial derivatives of E_{image} with respect to $\mathbf{x}$ and $\mathbf{y}$, respectively.

The matrix equation in Eq. (3) cannot be directly solved for $\mathbf{x}$ and $\mathbf{y}$ because $\mathbf{f}_\mathbf{x}$ and $\mathbf{f}_\mathbf{y}$ depend on $\mathbf{x}$ and $\mathbf{y}$. Hence, an iterative process is used in which $\mathbf{x}_t$ and $\mathbf{y}_t$ converge to the solution of Eq. (3) as time t progresses. Assuming that $\mathbf{f}_\mathbf{x}$ and $\mathbf{f}_\mathbf{y}$ are constant within a unit time interval γ, $\mathbf{x}_t$ and $\mathbf{y}_t$ can be obtained from $\mathbf{x}_{t-1}$ and $\mathbf{y}_{t-1}$ by solving two linear equations;

$$\begin{aligned} \mathbf{x}_t &= (\mathbf{A} + \gamma\mathbf{I})^{-1}(\gamma\mathbf{x}_{t-1} - \mathbf{f}_x(\mathbf{x}_{t-1}, \mathbf{y}_{t-1})) \\ \mathbf{y}_t &= (\mathbf{A} + \gamma\mathbf{I})^{-1}(\gamma\mathbf{y}_{t-1} - \mathbf{f}_y(\mathbf{x}_{t-1}, \mathbf{y}_{t-1})) \end{aligned} \quad (4)$$

In summary, after the initial position of a snake is specified by the user, the position is incrementally updated by repeatedly solving the linear equations in Eq. (4). In each iteration, the snake approaches a nearby feature by moving in the direction of $(-\mathbf{f}_\mathbf{x}, -\mathbf{f}_\mathbf{y})$, which reduces the external feature energy E_{image}. In the movement, the shape of the snake is kept smooth by solving linear equations that contain the matrix $\mathbf{A}$ derived from the internal spline energy E_{spline}.

4. Geometric Snakes

4.1. Representation

A geometric snake for a triangular mesh is represented by $v(s) = (x(s), y(s), z(s))$, where s is the parameter such that $s \in [0,1]$. Similar to Eq. (1) for an image snake, the energy functional of a geometric snake consists of the internal spline energy E_{spline} and the external feature energy E_{mesh};

$$E_{snake}(v) = \int_0^1 \left[E_{spline}(v) + E_{mesh}(v)\right] ds \quad (5)$$

In Eq. (5), the spline energy E_{spline} is concerned with the smoothness of a geometric snake and can be computed in the same way as that of an image snake with Eq. (2). The feature energy E_{mesh} is determined by the definition of a feature on a mesh and should have local minimums at the features. In this paper, we use the normal variations of the neighbor faces to determine the feature energy at a vertex of a mesh. The feature energy at the internal points of a face is computed by linear interpolation. We explain the details of the feature energy E_{mesh} in Section 6.

In implementation, similar to an image snake, we represent a geometric snake $v(s)$ by a sequence of n sample points $v(i)$ on the mesh. With this representation, the line segment connecting two sample points may not lie on the mesh surface. For simplicity, in the energy minimization process, we only consider the sample points that lie on the mesh surface. When the final position of a geometric snake is determined, we project the line segments between sample points to obtain a piecewise linear curve on the mesh surface.

4.2. Movement

After its initial position on a mesh is specified by a user, a geometric snake can detect a nearby feature by minimizing Eq. (5). For the minimization process, we can use the same numerical technique as for an image snake, summarized in Section 3. That is, the matrix equations in Eq. (3) is obtained for $v(s) = (x(s), y(s), z(s))$ and the position is incrementally updated by repeatedly solving the linear equations in Eq. (4). In this approach, the updated position of a geometric snake will be in the 3D space near the old position and is not guaranteed to lie on the surface of the mesh. However, since the feature energy E_{mesh} is defined only on the mesh surface, it is impossible to solve Eq. (4) for the next step if the current position of a geometric snake is not on the mesh surface.

A simple solution to resolve this problem is to project a geometric snake onto the surface of a mesh every time when its position is updated by solving Eq. (4). We implemented and tested this approach, but the approach was found to be computationally expensive because the projection must be performed at every update. This drawback may prohibit a geometric snake from having a speed fast enough to be used as an interactive tool.

In this paper, to constrain a geometric snake onto the surface of a mesh, we use mesh parameterization that embeds a part of the mesh onto a 2D plane. With the parameterization, a geometric snake $v(s) = (x(s), y(s), z(s))$ is mapped to a curve $v^*(s) = (x^*(s), y^*(s))$ in the plane. Then we can use the same equations in Section 3 to compute the position

updates of the curve $v^*(s)$ in the plane. Since the feature energy E_{mesh} is a scalar field on the surface of a mesh, it can be easily mapped onto the embedding plane. The updated position of the geometric snake $v(s)$ is obtained by mapping the curve $v^*(s)$ back onto the mesh surface. Smoothness of $v(s)$ can be maintained by the spline energy E_{spline} applied to $v^*(s)$.

In the proposed approach, we can perform a number of iterations to update the position of a geometric snake after the mesh parameterization has been derived. In spite of the overhead to compute parameterization, this benefit makes the proposed approach more efficient than the simple approach with the projection onto the mesh surface. In our experiment, a geometric snake implemented with the proposed approach could detect the feature of a mesh from the initial position much faster than the simple approach.

To reduce the computational overhead of parameterization, we do not embed the whole mesh but only the local region surrounding a geometric snake to a 2D plane. In addition to efficiency, local parameterization has the advantage of a small distortion in the embedding. When the position of the curve $v^*(s)$ is updated by solving Eq. (4), the new position is usually contained in the embedded part of the mesh. This is because the initial position of a geometric snake is specified near a desired feature in most cases and the snake moves slightly in each update. If the curve $v^*(s)$ moves out of the embedded part in an update, we again compute the local parameterization of the mesh with the current position of the geometric snake $v(s)$.

If we embed the whole mesh onto a 2D plane, a single parameterization is sufficient regardless of the updated positions of the curve $v^*(s)$ in the energy minimization process. However, this global parameterization requires more computation than the local one used in this paper. More importantly, when a large mesh is parameterized, a severe distortion may happen in the embedding [2, 19, 26]. Since the smoothness of a geometric snake $v(s)$ comes from that of the curve $v^*(s)$, the distortion should be minimized in the embedding. By using local parameterization, we can avoid large distortion and preserve the smoothness of a geometric snake.

4.3. Overall process

The overall process for feature detection on a mesh with a geometric snake proposed in this paper is as follows. Fig. 1 summarizes the process.

1. The initial position of a geometric snake $v(s)$ is interactively specified by the user (Section 5.5).
2. The local region containing $v(s)$ is determined on the mesh (Sections 5.1 and 5.2).
3. The local region is embedded onto a 2D plane while minimizing the distortion (Section 5.3).
4. Snake $v(s)$ is converted to a curve $v^*(s)$ on the plane (Section 5.4).

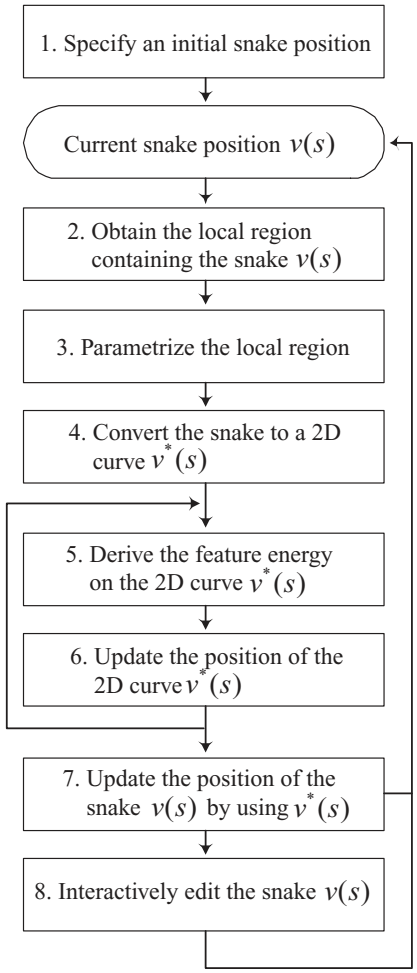


Figure 1: *Feature detection process*

5. The external feature energy E_{mesh} is derived at the points on the curve $v^*(s)$ (Sections 6.2 and 6.3).
6. The updated position of $v^*(s)$ in the plane is determined by solving the linear equations in Eq. (4). If the position update needs to be repeated, go to step 5.
7. The final position of the curve $v^*(s)$ is mapped onto the mesh surface to update the current position of snake $v(s)$ (Section 5.4).
8. If the updated current position of snake $v(s)$ is not satisfactory, the user can interactively edit the parts of $v(s)$ (Section 5.5). If the user wishes, proceed to step 2.

In the process, the position update of the curve $v^*(s)$ by steps 5 and 6 can be iterated in the fixed number of times or until a part of $v^*(s)$ moves out of the embedded local region of the mesh. Note that the local parameterization is performed only once in step 3 for the iteration. If the updated position of the snake $v(s)$ obtained from the result of the it-

eration does not capture the desired feature yet, the user can reapply the process to $v(s)$ starting from step 2.

5. Movements of Geometric Snakes

In this section, we present the component techniques that are used to move a geometric snake on the surface of a mesh. We also explain the user interaction techniques that can guide a geometric snake to capture a desired feature.

5.1. Partitioning of a snake

In general, the local surrounding region of a geometric snake is much smaller than the entire surface of a mesh. However, if a snake widely stretches over the surface (see Figs. 10(b) and 10(c)), the surrounding region to be parameterized becomes large, which increases the computation time. Moreover, in the case of a closed snake, such as shown in Fig. 10(b), the surrounding region contains a hole, which complicates the parameterization process. To reduce the size and avoid a hole of a parameterized region, we partition a geometric snake into two or more curve segments. Then, the steps 2 through 7 of the overall process in Fig. 1 are applied to each curve segment as if it is a single geometric snake.

Partitioning a geometric snake into curve segments may introduce a discontinuity into the shape of the snake. Without loss of generality, we assume that a snake is partitioned into two segments $v_A(s)$ and $v_B(s)$ (see Fig. 2). The snake may become discontinuous at the border point p_j when the positions of $v_A(s)$ and $v_B(s)$ are independently updated. To overcome this problem, we recompute the snake position around p_j by using a curve segment $v_C(s)$ that partially overlaps with $v_A(s)$ and $v_B(s)$. In other words, we first update the positions of $v_A(s)$ and $v_B(s)$ while fixing the border point p_j. Then, the position of $v_C(s)$ is updated with its end points p_i and p_k fixed at the previously computed positions in $v_A(s)$ and $v_B(s)$. This adjustment of the snake position around p_j makes the two segments $v_A(s)$ and $v_B(s)$ be connected smoothly.

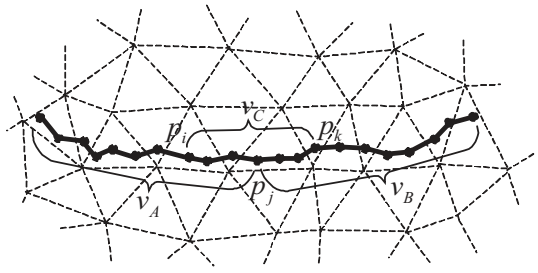


Figure 2: *Partitioning of a geometric snake*

It may be possible to partition a geometric snake into curve segments so that distortions are minimized in parameterizing the surrounding regions of the segments. In most cases, however, since the shape of a snake is smooth due to the internal energy E_{spline}, a simple partitioning may not incur severe distortions in the embedding of a surrounding region. In this paper, we partition a snake into curve segments that contain the same number of snake sample points. If the current shape of a snake contains sharp corner points with small incident angles, the snake is partitioned to place the corner points at the middles of curve segments. Rather than arbitrarily placing the corner points, for example, at the borders of curve segments, this approach enables the shape of a snake to rapidly smoothen around the corner points in a position update. A sharp corner point may happen in the initial positioning and interactive editing (see Section 5.5) of a geometric snake. Usually, a snake contains either few, or most often, no sharp corner points, and two or at most several curve segments are obtained from the partitioning.

5.2. Local surrounding region of a snake

After partitioning a snake, we determine the local surrounding region on the mesh for each segment, which will be embedded onto a 2D plane. The selection of a too small region decreases the number of position updates possible with a single parameterization. If a large region is selected, it will increase the computation time and distortions in the embedding.

To obtain the surrounding region with a reasonable size, we first obtain the face sequence F on the mesh on which the line segments between snake sample points are projected. Then, the surrounding region is determined as the set of faces which contains the face sequence F and the neighbor faces sharing a vertex with a face in F (see Fig. 3). Finally, we fill the holes that may exist in the resulting region.

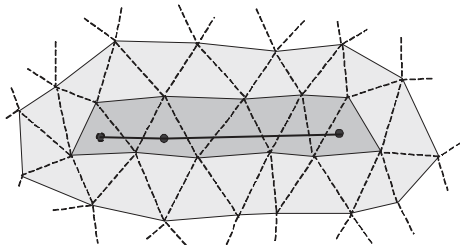
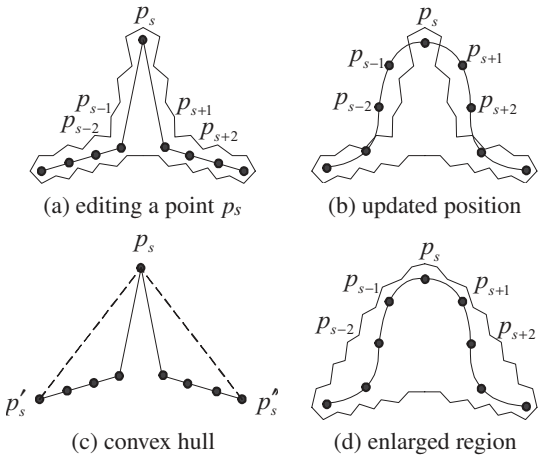


Figure 3: *Surrounding region of a geometric snake*

A geometric snake is initially placed near a feature and its position is incrementally updated in the energy minimization process. Hence, with the proposed method to select the local surrounding region, several position updates are possible with a single parameterization, as demonstrated in Section 7. Also, the selected region contains a small number of triangles and can be parameterized quickly.

In interactive editing of a geometric snake which will be explained in Section 5.5, the user can move a snake sample

point p_s far from the current position. See Fig. 4(a) for an example. Then, the parts of the snake near the point p_s will be rapidly changed to maintain the smoothness in the following position update. In this case, the updated position of the snake may not belong to the local region determined from the current position, as shown in Fig. 4(b). To avoid such a case, we first map the current snake curve to a 2D plane and find the two vertices p_s' and p_s'' adjacent to p_s in the 2D convex hull of the snake. Then, the local region is enlarged to include the projections of two 3D line segments, from p_s to p_s' and p_s'', onto the mesh surface. See Figs. 4(c) and 4(d) for an illustration.



(a) editing a point p_s (b) updated position (c) convex hull (d) enlarged region

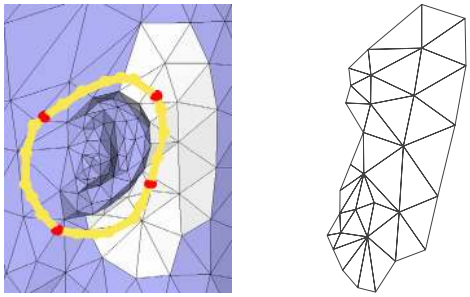
Figure 4: *Problem in region selection for interactive editing: The polygons enclosing the snake curves indicate the local regions selected for the snake.*

5.3. Parameterization of a local region

Since geometric snakes are an interactive tool to detect features, the parameterization of the selected local region should be performed quickly. In addition, the distortions in the parameterization should be minimized so that the positions of a geometric snake $v(s)$ and the 2D curve $v^*(s)$ are properly mapped to each other.

In this paper, for parameterization, we use the convex combination approach extended with virtual boundaries [18]. The convex combination approach [8] obtains the parameterization of a mesh by solving a linear system. The approach runs fast and generates an one-to-one embedding. However, the approach requires the boundary of a mesh to be fixed onto a convex polygon, which causes high distortion near the boundary. The extension [18] of the approach used in this paper reduces the distortions by using virtual vertices attached to the real boundary.

Fig. 5(a) shows the surrounding region of a curve segment from a snake that is placed near an ear. Fig. 5(b) shows the embedding result of the region generated by the convex combination approach with a virtual boundary.



(a) surrounding region (b) embedding of the region

Figure 5: *Parameterization of a surrounding region*

5.4. Update of the snake position

After the surrounding region is embedded onto a 2D plane, we convert the snake $v(s)$ to a curve $v^*(s)$ in the plane. A snake point inside a 3D triangle is mapped to a point in the corresponding 2D triangle by using barycentric coordinates. Then, the position of the curve $v^*(s)$ is incrementally updated by iteratively solving the linear equations in Eq. (4). The iteration continues until the curve $v^*(s)$ moves out of the embedded region or the preset number of iterations are performed. The final position of the curve $v^*(s)$ is mapped back onto the mesh surface to determine the updated position of the snake $v(s)$, again by using barycentric coordinates.

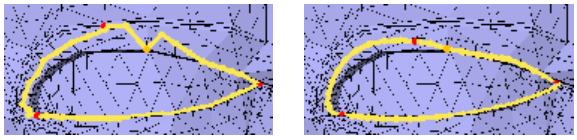
5.5. User Interaction

Our geometric snake model provides three user interaction techniques: snake initialization, point editing, and point fixing. The snake initialization is used to specify the initial position of a snake. When the user selects a sequence of vertices on a mesh near a desired feature, a closed or open curve is automatically built by connecting adjacent vertices in the sequence with the shortest paths through the edges of the mesh.

With point editing, the user can guide a point of a snake to a desired feature. For example, in Fig. 6, the user wants to detect the eyelid using the snake, but some snake points have been attracted to the eyebrow. Then, the user can edit a snake point to change its position from the eyebrow to the eyelid, as shown in Fig. 6(a). After the editing, the snake points around the edited point moves together toward the eyelid due to the spline energy E_{spline}. Fig. 6(b) shows the resulting position from which the snake can finally capture the eyelid.

The point fixing allows a point of a snake to be fixed at a specific position on the mesh in the energy minimization process. For example, in Fig. 6, the edited snake point is

fixed at the moved position after editing. The point fixing is useful to protect snake points that already capture parts of a desired feature from position changes in minimizing the spline energy E_{spline}.



(a) editing a point (b) updated position

Figure 6: *Editing a snake*

6. Feature Energy for Geometric Snakes

6.1. Feature energy definition

To define an external feature energy E_{mesh} for a mesh, we must have a numerical representation of features which has a local minimum at a feature. In image snakes, for example, image gradients at pixels are used to define the features on an image [14]. In the same manner, we can consider the features on a mesh as the points on the mesh where the mesh property changes drastically. For example, peaked corners, sharp edges, and color discontinuities can be used for mesh features.

In this paper, we use normal variations of the neighbor faces to determine the feature energy at a vertex of a mesh. To compute the normal variation, we adopt the opening angle of the normal cones at a vertex [16]. Then, the feature energy E_{mesh} at a vertex v is defined by

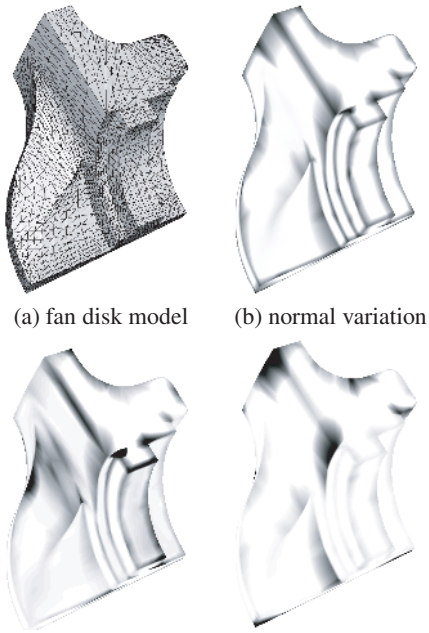
$$E_{mesh}(v) = \min_f (n_v^T n_f). \quad (6)$$

In Eq. (6), n_v is the normal at a vertex v and n_f is the normal of a face f adjacent to the vertex v. With this definition, the feature energy at a vertex v has values between 0 and 1. The feature energy at points inside a face is determined by linearly interpolating the energy at vertices of the face.

In addition to normal variation, we experimented with the discrete curvature norm [15] and the quadric error metric [9] to define mesh features. With the discrete curvature norm, we can consider a feature as a point at which the sum of principal curvatures κ_1 and κ_2 is large. By using the quadric error metric, we can define the feature energy at a vertex v as the negation of the maximum of the edge collapse errors for the edges adjacent to v. As shown in Fig. 7, all three definitions of feature energy give similar results and have local minimums at features such as edges and corners. In this paper, we chose the normal variation for feature definition because of its simplicity.

6.2. Feature energy on a face

Before starting a feature detection process, we compute the feature energy values for the vertices of a mesh by using Eq. (6). The feature energy inside a face is determined by linear interpolation, as mentioned before. However, to update the position of a snake, we do not need the feature energy itself but the derivatives of feature energy, that is, $\mathbf{f_x}$ and $\mathbf{f_y}$ in Eq. (4). Inside a face, the derivatives $\mathbf{f_x}$ and $\mathbf{f_y}$ are constant because the feature energy linearly varies.



(a) fan disk model (b) normal variation

(c) discrete curvature (d) quadric error

Figure 7: *Comparison of feature energy definitions*

Therefore, in the energy minimization process to update the position of the curve $v^*(s)$, the derivatives $\mathbf{f_x}$ and $\mathbf{f_y}$ at a point p on $v^*(s)$ can be obtained as follows:

1. The feature energy values of the vertices in the surrounding region are mapped onto the embedding plane.
2. The derivatives $\mathbf{f_x}$ and $\mathbf{f_y}$ are computed and saved for each face in the plane as a preprocessing step.
3. During the energy minimization process, the values saved for the face containing the point p are used for the derivatives $\mathbf{f_x}$ and $\mathbf{f_y}$ at p.

With this approach, the derivatives $\mathbf{f_x}$ and $\mathbf{f_y}$ need not be dynamically computed in the energy minimization process, which reduces the computational overhead.

6.3. Feature energy smoothing

In order to enable a distant feature to attract a snake and to reduce the effects of noises, image snake models use low-pass filters that blur feature energy values at image pixels [14, 10]. Similarly, we can smooth the feature energy by weighted averaging the energy value of a vertex with those of neighbor vertices. With smoothing, a geometric snake can

be less affected by noises and capture a strong feature that is distant from the initial position.

7. Experimental Results

Fig. 8(a) shows examples of feature detection with geometric snakes, where the boundaries of eyes and the nose are captured in the skull model. Figs. 8(b)-(d) illustrate the feature detection process for the right eye. First, the user selects a sequence of points to specify the initial position of a snake, which are the red dots in Fig. 8(b). Next, the initial position is determined by finding the shortest paths between consecutive points in the sequence, as shown in Fig. 8(b). After five iterations of the energy minimization step in Eq. (4), we obtain the position of the snake shown in Fig. 8(c). Fig. 8(d) shows the final result after 40 iterations. Each step in the feature detection process including energy minimization is performed fast enough for the user to interact with the snake in almost real time (see Table 1).

Fig. 9 illustrates the detection of the necklace in the 'Happy Buddha' model using a geometric snake. From the initial position in Fig. 9(b), the geometric snake successfully detects the shape of the necklace, as shown in Fig. 9(c), after 55 iterations of the energy minimization step. Fig. 9(d) shows the final result in Fig. 9(c) from a different viewpoint. In the feature detection process, we edited and fixed one snake point near the lowest part of the necklace to guide the geometric snake.

Fig. 10 shows other feature detection examples with geometric snakes. In Fig. 10(a), feature curves in a bunny model are detected. Fig. 10(b) shows a geometric snake that captures the outline of the stomach in the 'Happy Buddha' model, where points near the necklace were fixed in the feature detection process. In Fig. 10(c), eye brows and the nose in a mask model are captured, which demonstrates the usefulness of a geometric snake for feature specification in mesh morphing.

Table 1 summarizes the statistics of the examples in Figs. 8 through 10. For each example, only a small number of points were specified by the user for the initial positioning of a snake. The snake sample points were selected from the edges in the shortest paths between the user-specified points. In the current implementation, when the user clicks the snake operation button, the local region surrounding a snake is parameterized and at most five position updates are performed. Parameterization and the position update by Eq. (4) run fast enough to provide a prompt response, as shown in Table 1, where parameterization dominates the computation time. The number of snake operations required for feature detection depends on the initial position of a geometric snake and a user input helps a geometric snake to move toward a feature faster. For the examples in Figs. 8 through 10, the final positions of geometric snakes could be obtained by several snake operations. Several minutes were sufficient for the whole feature detection process including user interaction.

8. Conclusion

In this paper, we proposed a geometric snake as an interactive tool for feature detection on a 3D triangular mesh. Besides the parametric snake model [14] used in this paper, implicit active contour models based on level sets were investigated in medical image processing and computer vision [3, 20]. For future work, we will verify the possible benefits of the implicit models over the parametric model when they are applied to feature detection on a 3D triangular mesh. The final goal in this direction will be the integration of several active contour models to provide an effective feature detection tool for 3D meshes.

Acknowledgements

The authors would like to thank Igor Guskov for a 2-manifold 'Happy Budda' mesh model [12]. The bunny model and the original 'Happy Budda' models are courtesy of the Stanford Computer Graphics Laboratory. The skull model is courtesy of Headus (www.headus.com.au) and Phil Dench. We would also like to thank the anonymous reviewers for their valuable comments. This work was supported in part by the Korea Ministry of Education through the Brain Korea 21 program, the KIPA Game Animation Research Center, the KOSEF Virtual Reality Research Center, and the Korea Ministry of Information and Communication under grant 1NJ0129601.

References

1. S. Andrews. Interactive generation of feature curves on surfaces: A minimal path approach. Master's thesis, University of Toronto, 2000. 2
2. C. Bennis, J.-M. Vézien, and G. Iglésias. Piecewise surface flattening for non-distorted texture mapping. *ACM Computer Graphics (Proc. of SIGGRAPH '91)*, pages 237–246, 1991. 4
3. V. Caselles, F. Catte, T. Coll, and F. Dibos. A geometric model for active contours in image processing. *Numerische Mathematik*, 66(1):1–31, 1993. 2, 8
4. V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79, 1997. 2
5. L. Cohen and R. Kimmel. Global minimum for active contour models: A minimal path approach. *International Journal of Computer Vision*, 24(1):57–78, 1997. 2
6. L. D. Cohen. On active contour models and balloons. *Computer Vision, Graphics and Image Processing: Image Understanding*, 53(2):211–218, 1991. 1, 2
7. L. D. Cohen and I. Cohen. Finite element methods for active contour models and balloons for 2d and 3d images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(11):1131–1147, 1993. 1, 2

model	# vertices	# edges	# faces	feature	# user points	# sample points	embedding (sec)	pos. update (sec)	snake op. (sec)
Skull	19,288	57,858	38,572	right eye	5	100	0.198	0.005	0.223
Buddha	15,033	45,141	30,094	necklace	7	91	0.224	0.001	0.229
Bunny	34,834	104,288	69,451	leg	5	117	0.222	0.003	0.238
Mask	15,003	44,760	29,758	eye brows & nose	8	288	0.266	0.007	0.300

Table 1: *Statistics: The first three columns show the numbers of vertices, edges, and faces of mesh models. The 'feature' column designates the feature detected in each experiment. The '# user points' column gives the number of points selected by the user to specify the initial position of a snake. The '# sample points' column gives the number of snake sample points. The 'embedding' and 'pos. update' columns show the computation times for parameterizing the surrounding region of a snake and for one position update by Eq. (4), respectively. The final column gives the computation time for one snake operation that consists of an embedding and five position updates. The computation time is measured in seconds on a Pentium IV 1.5GHz PC.*

8. M. S. Floater. Parameterization and smooth approximation of surface triangulation. *Computer Aided Geometric Design*, 14(3):231–250, 1997. 6

9. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *ACM Computer Graphics (Proc. of SIGGRAPH '97)*, pages 209–216, 1997. 7

10. M. Gleicher. Image snapping. *ACM Computer Graphics (Proc. of SIGGRAPH '95)*, pages 183–190, 1995. 2, 7

11. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. *ACM Computer Graphics (Proc. of SIGGRAPH '99)*, pages 325–334, 1999. 2

12. I. Guskov and Z. J. Wood. Topological noise removal. In *Proc. Graphics Interface 2001*, pages 19–26, 2001. 8

13. A. Hubeli and M. Gross. Multiresolution feature extraction from unstructured meshes. In *Proc. IEEE Visualization 2001*, San Diego, California, 2001. 2

14. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988. 1, 2, 7, 8

15. S.-J. Kim, C.-H. Kim, and D. Levin. Surface simplification using discrete curvatre norm. In *Proc. the 3rd Israel-Korea Binational Conference On Geometrical Modeling and Computer Graphics*, pages 151–157, 2001. 7

16. L. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature-sensitive surface extraction from volume data. *ACM Computer Graphics (Proc. of SIGGRAPH 2001)*, pages 57–66, 2001. 7

17. L. Kobbelt and G. Taubin. Geometric signal processing on large polygonal meshes. *Course Notes # 17 for Siggraph'2001*, 2001. 2

18. Y. Lee, H. S. Kim, and S. Lee. Parameterization of triangular meshes with virtual boundaries. In *Proc. the 3rd Israel-Korea Binational Conference On Geometrical Modeling and Computer Graphics*, pages 105–113, 2001. 6

19. J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. *ACM Computer Graphics (Proc. of SIGGRAPH '93)*, pages 27–34, 1993. 4

20. R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(2):158–175, 1995. 1, 2, 8

21. T. McInerney and D. Terzopoulos. Topologically adaptable snakes. In *5th International Conference on Computer Vision (ICCV'95)*, pages 840–845. IEEE Computer Society Press, 1995. 1

22. T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000. 1, 2

23. M. J. Milroy, C. Bradley, and G. W. Vickers. Segmentation of a wrap-around model using an active contour. *Computer-Aided Design*, 29(4):299–320, 1997. 2

24. E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. *ACM Computer Graphics (Proc. of SIGGRAPH '95)*, pages 191–198, 1995. 2

25. S. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988. 2

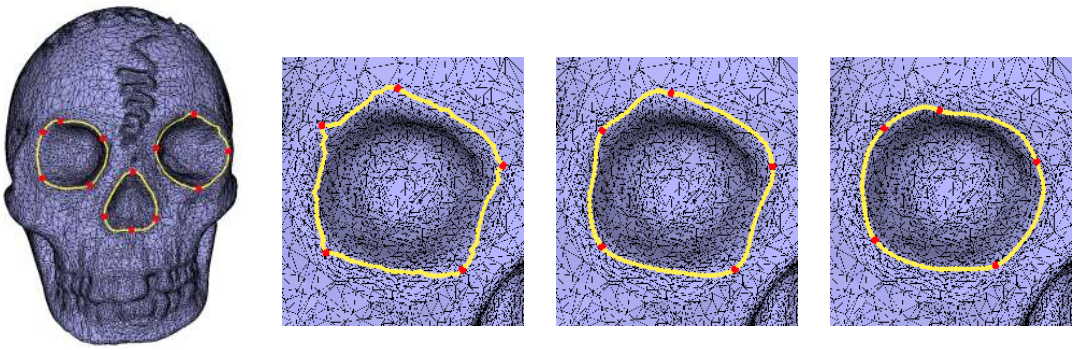
26. E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. *ACM Computer Graphics (Proc. of SIGGRAPH 2000)*, pages 465–470, 2000. 4

27. E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterizations. *ACM Computer Graphics (Proc. of SIGGRAPH 2001)*, pages 179–184, 2001. 2

28. P. Schröder. Digital geometry processing. *Course Notes # 50 for Siggraph'2001*, 2001. 2

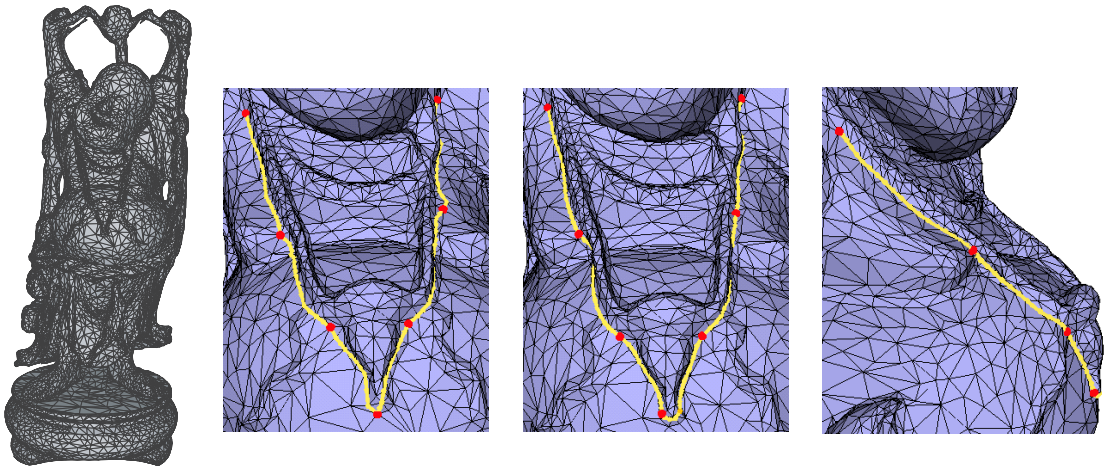
29. C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Trans. Image Processing*, 7(3):359–369, 1998. 1, 2

30. A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum. A geometric snake model for segmentation of medical imagery. *IEEE Trans. Medical Imaging*, 16(2):199–209, April 1997. 2



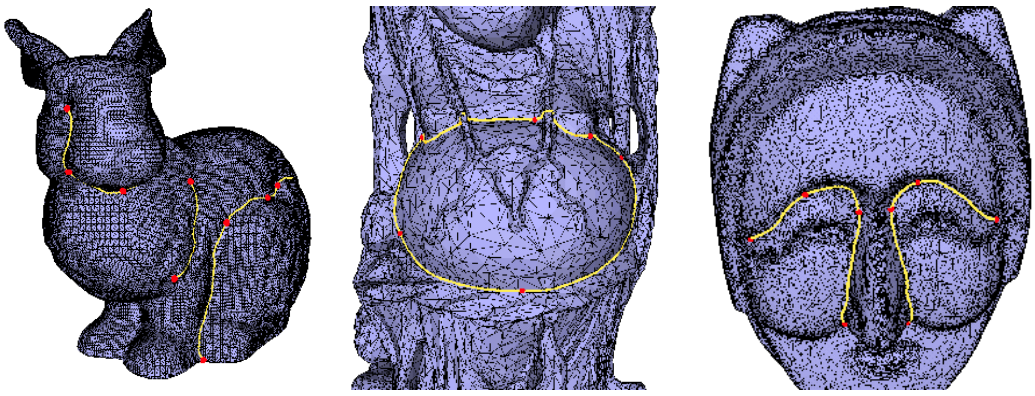
(a) (b) (c) (d)

Figure 8: *Feature detection for eyes and the nose of a skull model: (a) final results; (b) initial position of a snake to detect the right eye; (c) after five iterations; (d) final captured position*



(a) (b) (c) (d)

Figure 9: *Feature detection for the necklace of the 'Happy Buddha' model: (a) original model; (b) initial position of a snake; (c)-(d) final detection result seen from different viewpoints*



(a) (b) (c)

Figure 10: *Other feature detection examples: (a) feature curves in a bunny model; (b) outline of the stomach in the 'Happy Buddha' model; (c) eye brows and the nose in a mask model*