User-Steered Methods for Extracting Geometric Features in 3D Meshes

Kwan-Hee Yoo[1] and Jong-Sung Ha[2]

[1]Chungbuk National University, khyoo@chungbuk.ac.kr
[2]Woosuk University, jsha@woosuk.ac.kr

ABSTRACT

In order to extract geometric features in 3D meshes with effective interactions according to user-steering, we generalize the 2D algorithms of snapping and wrapping that, respectively, move a cursor to a nearby feature and construct feature boundaries in images. First, we define two numerical values measuring the geometric characteristics of meshes: approximate curvatures and cost functions. By exploiting the defined measuring values, the techniques of **geometric snapping** and **geometric wrapping** in 3D meshes are developed and implemented. We also visualize the results obtained from applying the techniques to extracting geometric features in the general meshes modeled for human faces, cows, and single teeth.

Keywords: geometric snapping, geometric wrapping, geometric features, user-steered methods, 3D mesh.

1. INTRODUCTION

In CAD and graphics systems, diverse 3D models are represented with 3D meshes in order to be effectively processed. One of the most important functions in mesh applications is to detect **geometric features** representing their principal boundaries appearing in the meshes. Similarly to edges in images, geometric features are crucial for deciding which parts of the meshes have to be processed or to be preserved in many applications such as simplification, compression, editing, morphing, and deformation of the meshes. In mesh simplification and compression, the geometric features have to be maximally preserved. Mesh editing usually tends to process the parts appearing as geometric features in a mesh, and mesh morphing can also be performed by morphing the corresponding geometric features between two meshes. Meshes can be deformed by manipulating their parts appearing as the geometric features. In this paper, we consider user-steered methods for extracting geometric features from a given mesh with effective interactions.

Many researches [5-6][12-13][15] for extracting edges in 2D images have been reported for the last decade. Recently, 2D image snake technique [13] was extended to extracting geometric features in 3D meshes, which is called the **geometric snake** [14]. The concept of the geometric snake is to lock up geometric features with curves after mapping some parts of a given 3D mesh into the plane [7]. Initial curves are drawn for approximating the boundary of user-steered features, and then final curves are determined by changing the initial curves to minimize their energy functions. Since the geometric snake is dependent on only automatic boundary extraction after the initial curves are drawn, it often has difficulty in interactively extracting boundary shape. In this paper, we propose methods for extracting geometric features in 3D meshes according to user-steering by importing powerful interaction techniques, which are called the **user-steered geometric feature extraction**.

The user-steered feature extraction is composed of two important steps snapping and wrapping; a user moves a cursor to locate a sequence of seed points, and then boundary paths connecting the seed points are automatically constructed to form an entire boundary. These two steps can be iterated by moving or deleting some seed points, and by inserting more seed points to the sequence of seed points. In order to exactly move the cursor position to an intended point, a technique called the **snapping** is used, which moves the cursor to a nearby feature. The process of constructing the boundary paths enclosing geometric features after snapping the seed points is called the **wrapping**. 2D image snapping [9-10] is an evolution of the cursor snapping [2-3][18]. A few of practical methods have been presented for image wrapping: intelligent scissors [15], livewire and livelane [5], livewire on the fly [6], and enhanced lane [12]. In this paper, we propose two steps of **geometric snapping** and **geometric wrapping** for the user-

steered geometric features extraction in 3D meshes, which are the extensions of image snapping and image wrapping, respectively.

In Section 2, we give the outlines of image snapping for locating an exact cursor position and image wrapping for enclosing the boundary paths of features in 2D. Section 3 introduces the notion of approximate curvatures measuring the geometric characteristics of 3D meshes, a method for blurring the approximate curvatures, and a method for computing cost functions that provide the directions for moving a cursor and locating geometric features. We also propose the techniques of geometric snapping and geometric wrapping in 3D meshes, and post-processing for smoothing extracted geometric features. In Section 4, our proposed techniques are applied to extracting geometric features from the models of a face, a cow and a tooth respectively. Finally, we summarize the results of this paper in Section 5.

2. USER-STEERED EDGE EXTRACTION IN 2D IMAGES

2.1 Image Snapping

The cursor snapping was presented in Sketchpad system [19] for interactively providing an exact cursor position in graphical user interfaces, and it has been adopted in many CAD and graphics systems. Image snapping [9-10] is an evolution of the cursor snapping, which moves the cursor position to a nearby feature such as edges in 2D images when the cursor is located by a user. If we regard the images as height maps, for which the measuring values of gradient are used, the image snapping can be explained as the notion of a ball rolling down to valleys. The rolling ball may fall into local minimums before reaching at the deepest valley, which are caused by the limitation of computing method or the image characteristics themselves. In order to avoid this unexpected phenomenon, a technique called blurring is used to soften up local minimums and emphasize global minimum by weighting the gradient of each pixel on its nearby images

2.2 Image Wrapping

For developing the wrapping methods of user-steered edge extraction, an image is represented with a weighted directed graph where each node corresponds to a pixel in the image and a directed edge exists between every pair of neighboring nodes. After a cost function is defined to assign weighting values to the directed edges, the shortest paths from a start node to goal nodes are searched by dynamic programming techniques. A typical wrapping based on the graph search is the intelligent scissors presented by Mortenson *et. al* [15]; intensity difference between a pair of neighboring pixels is assigned to the directed edge connecting the pixels, and next the shortest paths from a given seed point to all nodes are searched. They suggested that some of these shortest paths are just the boundary paths of image edges. However, the shortest paths have to be post-processed since they do not always construct the right feature boundaries in an image with noises or complicated objects.

In order to get stronger user-steerability and more efficiency, other wrapping methods of livewire, livelane, and livewire on the fly have been presented by Falcao *et. al* [5-6]. They use other methods for graph search and user interaction differently to each other. In the livewire, a global graph is searched same as the intelligent scissors, but users can add seed points interactively with moving a cursor near to feature boundaries. Each time a seed point is added, a new shortest map is computed. Features are extracted by connecting the shortest paths among the seed points. However, this is inefficient since the shortest paths from each seed point to all nodes are computed by searching the global graph frequently. The livelane and the livewire on the fly are other wrapping algorithms developed for increasing the efficiency. In the livelane, the region for graph searching is restricted within a local window and the searching can be iterated with interactive feedbacks. As a cursor moves, the shortest path from a seed point to the cursor is visualized interactively. When the cursor escapes from the local window, a new seed point is generated automatically at the position where the window boundary intersects a feature boundary. Feature boundaries are extracted only when it exists within the width of a lane. In other words, users have to move the cursor near to the feature boundary within the permitted width of a lane. This concept of lanes makes it possible to construct the feature boundaries even in an image with noises or complicated objects. In the enhanced lane proposed by Kang and Shin [12], the cursor becomes the center of a local window while a seed point is used for setting up a local window in the live lane. The local window also moves according to the cursor, and the shortest path map is incrementally extended to the nodes within the moving local windows from a seed point. Similarly to the live wire on the fly, the extracted features are visualized and a point designated as new seed by a user is set up when the cursor escapes from the target boundary. Hence, fewer seed points can be set up while the graph searching is restricted within the region of local windows similarly to the live lane.

Furthermore, the response time varies within a uniform range differently to the live wire on the fly in which the time increases as the cursor goes away from seed points.

3. EXTENSION TO 3D MESHES

Before continuing, we discuss about a straightforward extension of image snapping and image warping into extracting geometric features in a given 3D mesh. First the geometry image of the mesh is generated by using parameterization mapping techniques [7]. Next we apply the image snapping and image wrapping to extract features on the generated image, and the results are mapped back to the mesh. This approach has been taken by the geometric snake [14], however, distance and angle may be distorted as well as it takes much time when geometry images are generated from 3D meshes.

The image snapping and wrapping in Section 2 represent an image with a weighted directed graph, and exploit the well-known graph searching algorithms. Our main idea is that the algorithms in 2D images can be naturally extended to 3D meshes if new numerical values measuring geometric characteristics in the meshes can be defined for replacing the measured values in the images since the meshes with vertices, edges, and faces can be directly represented as a weighted directed graph. Finally we can develop the algorithms of geometric snapping and geometric wrapping based on the pre-computed graph.

3.1 Measured Values

In order to reflect characteristics of each pixel in a given 2D image before extracting its features, the measuring values of gradients and energy functions are defined by being based on gravity. These values are weighted to the directed edges of a graph representing the image. Since the orientation of a 3D object can be changed, we have to define other values reflecting geometric characteristics of each vertex in a 3D mesh, instead of the gradients and energy functions. The geometric characteristics of the mesh are measured with approximate curvatures and cost functions.

3.2 Approximate Curvatures

In general, a curvature in a 3D model is defined as the ratio of change in slope on a particular point. The curvature is defined for a particular point on any surface of mesh models not as exactly as on curves or surfaces since the meshes are composed of planar faces and discontinuous at each vertex and edge. An important factor in computing curvatures on a mesh is how to explain the geometric features of the mesh model as well. Many powerful methods [1][8][11][16-17][20-22] for computing better approximate curvatures have been presented for mesh models. This paper adopts two results presented for computing approximate curvatures so that they reflect the geometric characteristics of 3D meshes more exactly. The first method defines the approximate curvature *AC(v)* on a given vertex v by using the normal vectors of faces containing v as.

$$AC(v) = 1.0 - \underset{i=0}{\overset{k-1}{Min}}(f_i^v \cdot f_{(i+1) \bmod k}^v) \tag{1}$$

In Equation (1), k is the number of faces sharing v, and f_i^v is the normal vector of the i-th face in the faces when they are in the counter-clockwise order. The approximate curvature *AC(v)* is determined by the minimum value among the inner products of normal vectors for all pairs of adjacent faces. The second method defines approximate curvatures on the edges incident to a given vertex v, and takes their average. Let the ordered vertices adjacent to v be nv_i for all $i = 0, \cdots, k-1$, where k is the number of vertices adjacent to v. We denote the edge connecting v and nv_i with ne_i. Then, the curvature $AC(ne_i)$ on the edge ne_i is defined as.

$$AC(ne_i) = (1/r_i^v + 1/r_i^{nv})/2.0 \tag{2}$$

In Equation (2), r_i^v is the radius of the circle passing the centers of ne_{i-1}, ne_i, and ne_{i+1}. Here each of ne_{i-1} and ne_{i+1} is contained into one of two faces sharing ne_i. The radius r_i^{nv} is similarly defined by ne_i and the two edges

adjacent to nv_i while sharing ne_i. The approximate curvature *AC(v)* on the vertex *v* can be determined by computing the average of the curvatures on ne_i for all $i = 0, \cdots, k-1$ as.

$$AC(v) = \sum_{i=0}^{k-1} AC(ne_i) / k \tag{3}$$

Since the range of approximate curvatures is dependent on the shape of meshes, we normalize *AC(v)* for a vertex *v* by redefining it as *AC(v)* / *MAX_AC*, where *MAX_AC* is the maximum of all approximate curvatures. In addition to the above methods, the approximate curvatures can be computed by using area and face adjacency [1], quadric error metric [8], and the mean or Gaussian curvatures of edges or vertices [20].

3.2.1 Blurring Approximate Curvatures

The approximate curvatures may have some extreme values as well as noises which are caused by the limitation of computing method or the geometric characteristics themselves. In order to avoid this unexpected phenomenon, we soften up local minimums and emphasize global minimum by weighting the approximate curvature of each vertex on its nearby vertices. This technique is called the blurring. This paper blurs the approximate curvatures by using a well-known weighting factor called Gaussian smoothing filter [15]. That is, the approximate curvature on a vertex *v* is redefined as.

$$BAC(v) = \sum_{i=0}^{k-1} AC(nv_i) \times \frac{1}{\sqrt{2\pi}\sigma} \times e^{\frac{-(dx^2+dy^2+dz^2)}{2\sigma^2}} \tag{4}$$

In Equation (4), the vector (dx, dy, dz) is $(v^x - nv_i^x, v^y - nv_i^y, v^z - nv_i^z)$ for a vertex $v = (v^x, v^y, v^z)$ and its adjacent vertex $nv_i = (nv_i^x, nv_i^y, nv_i^z)$. As shown in Table 1, we assign appropriate values to σ according to the size of *k* since the smoothing degree of the Gaussian filter is determined by the size of σ.

Range k	Assigned value σ
$k \le 7$	0.85
$7 < k \le 16$	1.7
$16 < k \le 36$	2.5
$36 < k$	3.5

Tab. 1 Smoothing degree of Gaussian filter

3.3 Cost Functions

For two adjacent vertices *v* and *u* in a given mesh, two directed edges <*v, u*> and <*u, v*> will be added into the corresponding graph, where <*v, u*> is a directed edge directing from *v* to *u*. The weighting value of <*v, u*> is defined as the cost required for moving the cursor from a vertex *v* to a vertex *u*. Without loss of generality, let *u* and *v*, respectively, be the current vertex and the next vertex to be chosen or moved. The cost function *cost(u, v)* for moving from *u* to *v* is defined similarly to that of an image pixel [15] as.

$$\cos t(u, v) = \omega_z f_z(v) + \omega_d f_d(u, v) + \omega_g f_g(v) \tag{5}$$

In Equation (5), the three functions, f_z, f_d and f_g are Laplacian zero-crossing, curvature direction, and curvature magnitude respectively. The Laplacian zero-crossing $f_z(v)$ is used for representing whether or not a vertex *v* is on geometric features in a mesh. In other words, it is defined $f_z(v) = 1$ if *v* is on a geometric feature, and $f_z(v) = 0$ otherwise. From our experimental results, we use the critical value of approximate curvatures for determining whether

a vertex v represents a geometric feature; If *AC(v)* is greater than 0.75, then $f_z(v) = 1$ else $f_z(v) = 0$. Since the vertex with a larger curvature represents the geometric feature better than other vertices with smaller curvatures, the curvature direction $f_d(u,v)$ is used to indicate how the cursor moves between v and u. Let *d(u, o)* be the vector starting at u and going to o such that o is a vertex adjacent to u with maximal curvature. Then, the curvature direction $f_d(u,v)$ can be defined as.

$$f_d(u,v) = 1.0 + d(u,o)\cdot d(u,v)/(|\,d(u,o)\,||\,d(u,v)\,|) \quad \text{if } AC(v)\text{-}AC(u) > 0,$$
$$1.0 + d(u,o)\cdot d(v,u)/(|\,d(u,o)\,||\,d(v,u)\,|) \quad \text{otherwise}$$

If $f_d(u,v)$ is almost zero, the cursor tends to move from v to u. Otherwise, the movement occurs conversely. The last function of the curvature magnitude $f_g(v)$ is the approximate curvature *AC(v)*. In Equation (5), we can give each ω the weight of the corresponding function. We set the weights as $\omega_z = 0.43$, $\omega_d = 0.43$, and $\omega_g = 0.14$, respectively, from the experimental results; the Laplacian zero-crossing and the curvature direction play important roles while the curvature magnitude has less effect relatively. In order that the costs of all directed edges are processed constantly, we normalize *cost(u,v)* for a directed edge *<u, v>* by redefining it as *(MAX_COST-cost(u,v))* / *MAX_COST*, where *MAX_COST* is the maximum of all costs.

3.4 User-Steered Feature Extraction

3.4.1 Geometric Snapping

The cost function defined in Section 3.3 is used for moving a cursor to a nearby geometric feature in 3D meshes. We consider three strategies for checking the cost functions of neighboring vertices [23]. The first is to check the vertices adjacent to current vertex v. If the largest cost of the adjacent vertices is greater than zero, the cursor moves to the vertex with the largest. This movement is iterated until the costs of all vertices adjacent to current vertex are less than one of the current vertex. This is a simple and convenient method, but it would take too much time in a dense mesh, i.e., lots of vertices are connected near to each other. To enhance the performance of moving the cursor in the dense mesh, it is possible to check farther vertices with a certain range from v instead of its adjacent vertices. The second strategy is to use the range of Euclidean distance, while the third one is to use the range of path length. The Euclidean distance is the distance between v and its adjacent vertex that is the farthest from v. An appropriate integer value n is selected for the path length that is the minimum number of edges connecting two vertices. Hence, we are checking the vertices inside a sphere at origin v with the radius d, or the vertices whose path lengths to v are less than n. The cursor movement iterates same as the first method.

3.4.2 Geometric Wrapping

As we did in geometric snapping, the user-steered algorithms of image wrapping are extended for extracting geometric features boundaries in a 3D mesh by using the set of given seed points. First, we can apply the algorithm of intelligent scissors to a weighted directed graph *DG* with the cost functions assigned to directed edges as presented in Section 3.2. In this paper, this algorithm is called the **geometric intelligent scissors**. Assume that a user has selected a point in a projected mesh on the screen space with a cursor, and a seed point p on a feature boundary has been determined automatically by geometric snapping, which is the nearest to a feature from the selected point. Then, the shortest paths from p to all vertices in *DG* are computed by dynamic programming with the cost functions. In this situation, whenever the user selects another vertex q, the shortest path from q to p becomes the geometric feature selected according to user-steering in the mesh.

In order to extract geometric features in a 3D mesh more precisely, a user can select a few of seed points in succession. Let $(p_1, p_2, \cdots, p_k)$ be a succession of seed points selected by the user in the mesh. After the shortest paths from p_1 to all vertices are computed, the shortest path from p_2 to p_1 is found, which will be denoted by $ge(p_1, p_2)$. After computing the shortest paths from each p_i to all vertices in sequence, the path $ge(p_i, p_{i+1})$ from p_i for each $i = 2, \cdots, k-1$ is found as same as $ge(p_1, p_2)$. The path $ge(p_1, p_2) + ge(p_2, p_3) + \cdots + ge(p_{k-1}, p_k)$ obtained by connecting all the paths found by iterating the above process is the geometric feature that the user has intended to

extract in the mesh. We call this method the **geometric livewire**. This algorithm has a shortcoming in the computation time since all vertices are considered each time; the more seed points the user has selected, the longer the computation time is.

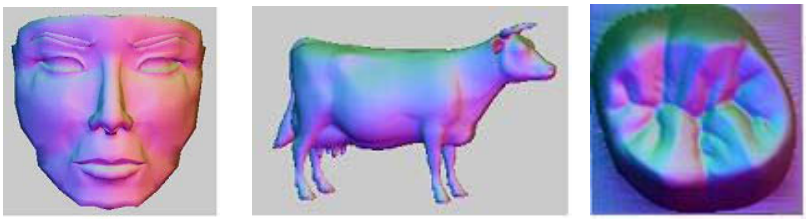
In order to reduce computation time in the geometric livewire, an enhanced algorithm called the **geometric livelane** is considered similarly to the livelane in 2D images. The livelane iterates an incremental process for locating seed points automatically with light computation times; the shortest paths are computed from a seed point to only the pixels that are in a region bounded by a window with a fixed size, the feature boundary within the fixed width is visualized by using the shortest paths as a user moves a cursor, and a new seed point is located automatically when the cursor escapes from the window. In the geometric live lane, the shortest paths are computed also from the seed points within a uniform range to other vertices. The window can be generalized as a cube in 3D. However, a sphere is more useful in 3D meshes, since the mesh is not grid-sized and the sizes of faces are different to each other. In this paper, we define a virtual window that is a sphere with the center of the seed point p and a fixed radius wd. The shortest paths from the current seed point p to the vertices within the fixed distance wd can be computed by the depth first traversal or breadth first traversal of the vertices starting from p. Among vertices within a fixed width, the vertices with the shortest paths are connected and the feature boundaries are visualized as the user moves the cursor. A new seed point is also located automatically when the cursor escapes from the neighboring range of the Euclidean distance wd.

3.4.3 Post-Processing

The geometric features obtained by applying the geometric wrappings to a 3D mesh are represented with open or closed polygonal lines connecting a selected vertex and other vertices passed during the iteration. Furthermore, the polygonal lines obtained by connecting the vertices in the mesh may have the shape of staircases. We remove such aliases by applying two kinds of curve fitting techniques [4][22]. The first fitting curve is the B-Spline curve that interpolates the start and end vertices of the obtained geometric feature, but approximates other vertices. The second one is another spline curve that also interpolates the vertices selected by a user, but approximates other vertices. Since the start point and the end point of these curves are always contained in the set of vertices of the given mesh, we can obtain the new points mapped onto the surfaces of the mesh from the points of the generated curves. The vertices of the new polygonal lines are used to represent the geometric feature better.

4. EXPERIMENTAL RESULTS

The proposed geometric snapping and geometric wrapping have been implemented in PC environments with the libraries of Microsoft Foundation Class (MFC) and OpenGL. Half-edge data structures are adopted for representing 3D meshes. We tested our implementation in the mesh models of a human face, a cow, and a tooth. Fig.1(a)-(c) are the rendered meshes of the three test models, respectively.

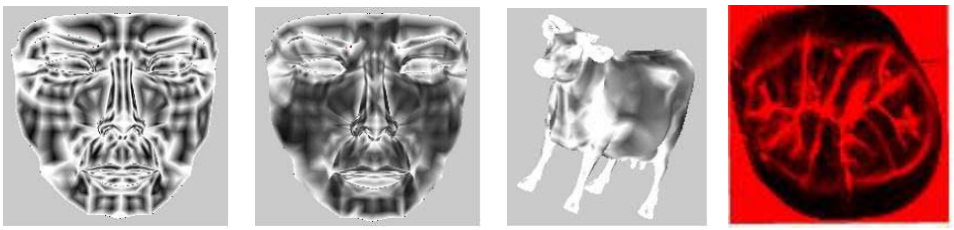


(a) (b) (c)

Fig. 1. Mesh models of a human face, a cow, and a tooth.

4.1 Approximate Curvatures and Blurring Them

The approximate curvatures of all vertices of a human face model, which were computed with Equation (1), are visualized with their magnitudes as shown in Fig.2(a). The brighter parts in the figures represent larger approximate curvatures. As explained in Section 3.2.1, however, the computed approximate curvatures are discontinuous and noisy in some regions. To get rid of these phenomena, the approximate curvatures were blurred with Equation (4). Fig.2(b)-(d) illustrate the results obtained by blurring the computed approximate curvatures that change smoothly in neighboring ranges.

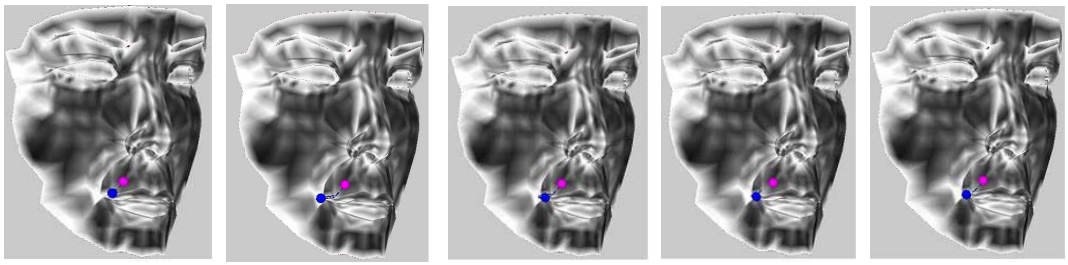


(a) (b) (c) (d)

Fig. 2. Blurred approximate curvatures

4.2 Geometric Snapping

In order to move a cursor from a selected vertex to a nearby geometric feature, we computed the cost function in Equation (5) with the blurred approximate curvatures for all vertices within each neighboring range. Fig.3(a)-(c), respectively demonstrate the steps of the cursor movement in the first strategy using the adjacency for checking neighbors: the 1'st movement, the 5'th movement, and the 7'th (final) movement. The results obtained by other strategies using an Euclidean length and a path length are presented in Fig.3(d) and (e). The cursor has settled down after 4 movements for a determined Euclidean length (see Fig.3(d)), while 2 movements were needed for the path length 3 (see Fig.3(e)). The initial vertex selected by the user is pink-colored, while the vertices passed by the cursor are blue-colored. Solid lines with the blue color represent the whole paths along which the cursor moved by the geometric snapping.

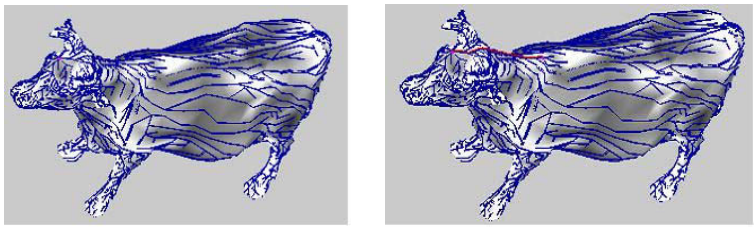


(a) (b) (c) (d) (e)

Fig. 3. Geometric snapping.

4.3 Geometric Wrapping and Post-Processing

A user can apply the geometric intelligent scissors to extracting geometric features in a mesh, by designating a seed point near to the geometric features. The red-colored sphere in Fig.4(a) represents the seed point. The shortest paths from the seed point to all other vertices are generated by applying the Dijstra's algorithm to the weighted directed graph constructed by the method in Section 3, which are visualized with blue-colored lines in Fig.4(a). Whenever a user selects another vertex, the shortest path from the vertex to the seed point can be obtained directly and the path is analyzed as a geometric feature. Solid red-colored lines in Fig.4(b) represent the geometric features obtained from two points selected by the user.



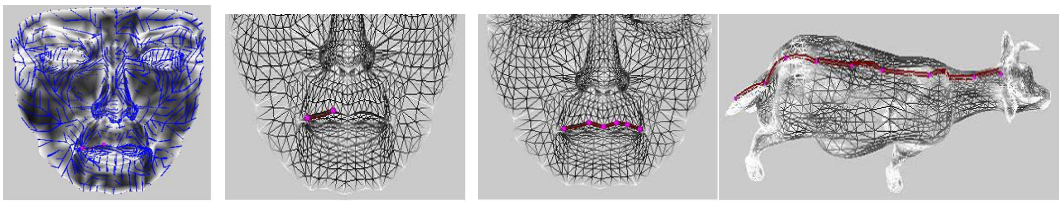
(a) (b)

Fig. 4. Geometric intelligent scissors.

For applying the proposed geometric livewire, a sequence of seed points near to the geometric features has to be provided by the user. Assume that the user has provided a sequence of seed points in the right direction from the left

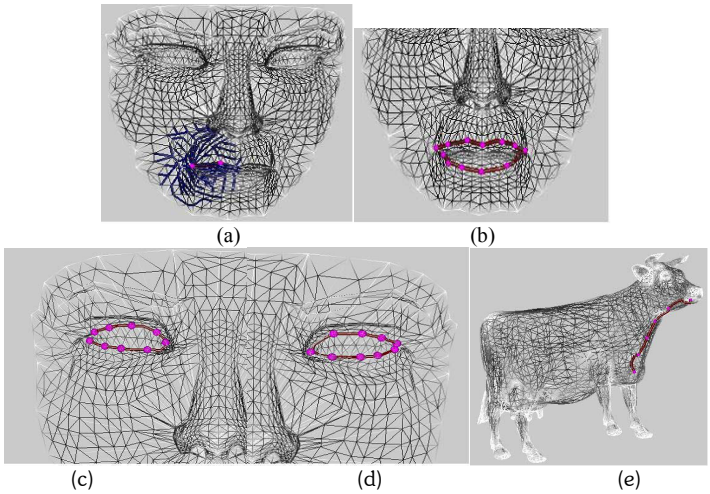
boundary of lips. In Fig.5(a), spheres are used to show the selected seed points, and the shortest paths from the first seed point to all vertices are drawn with blue-colored solid lines. Fig.5(b) illustrates the detailed geometric features between the first and the second seed points that are computed by the geometric livewire algorithm. The finally obtained geometric features are shown in Fig.5(c). Solid red-colored lines in Fig.5(d) illustrate the extracted geometric features in a cow model.

We apply the geometric live lane to extracting such geometric features from a face model. When one seed point is provided by a user, the algorithm computes the shortest paths from the seed point s to other vertices within a virtual window designated by the user, which are shown in Fig.6(a). In this experiment, the Euclidean distance 5.0 is used for the virtual window. Next the user selects a vertex v on the shortest paths and then geometric features between s and v are directly calculated using the pre-computed shortest paths. If the user wants other geometric features, the algorithm treats v as a new seed point and computes the shortest paths from v to other vertices within the distance 5.0. This procedure will be repeated until some satisfied geometric features are obtained. The most typical features of a face model are the boundaries of eyes, eyebrows, noses, lips, etc. Fig.6(b) is the result from applying geometric livelane to extracting the boundary of lips. Fig. 6(c) and (d) are the results from extracting the boundaries of a left eye and of a right eye, respectively. Input information and the selected vertices are displayed with spheres and solid lines respectively. Solid red-colored lines in Fig.6(e) illustrate the extracted geometric feature in a cow model.



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

Fig. 5. Geometric live wire.



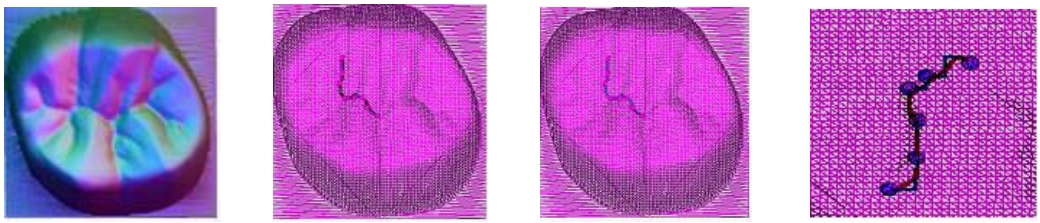
(a) (b)

(c) (d) (e)

Fig. 6. Geometric live lane.

In general, a tooth touches another tooth at the opposite side when the upper and lower jaws occlude. During the occlusion, the touched surface at the opposite side is said to be an occlusal surface. In order to exactly model artificial prostheses with the occlusal surfaces, we have to extract geometric features such as cusp, ridge, fissure, and pit [24] from teeth models, which are illustrated in Fig.7(a). The protruded part like a peak and the part descending from it are called the cusp and the ridge respectively, while the concave parts sinking in randomly and the deepest one like a small point among them are called the fissure and the pit respectively. The thick solid lines of Fig.7(b) are the fissures extracted from the tooth. In most cases, we use the 3D scanners of touch type to guarantee the accuracy of tooth model. Sampled points with the 3D touched scanners tend to produce an isotropic mesh in which the staircase

phenomenon can also appear in the extracted geometric features. Finally, we apply the post-processing to the extracted geometric features in order to remove the staircase, as shown in Fig.7(c). Fig.7(d) illustrates the final geometric feature that is visualized with thicker and softer solid lines.



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

Fig. 7. Removing staircase on geometric features

5. CONCLUSION

In order to extend 2D image snapping and wrapping algorithms into user-steered feature extraction in 3D meshes, we defined the approximate curvatures and cost functions that are numerically measured values for reflecting the geometric characteristics of the meshes. The technique of geometric snapping was presented for moving a cursor position to a nearby feature such as vertices in the meshes when the cursor located by a user. We also presented the techniques of geometric wrapping: geometric intelligent scissors, geometric live wire, and geometric live lane, which extract geometric features by constructing the boundary paths enclosing features. Since the boundaries of the extracted features have the shape of staircases, better ones were obtained by removing such aliases with curve fitting methods.

6. REFERENCES

[1] Alliez, P., Cohen-Steiner, D., Levoy, B. and Desbrun, M., Anisotropic polygonal remeshes, *ACM Computer Graphics (Proc. of SIGGRAPH '03)*, 2003, pp.485-193.

[2] Bier E., Snap-dragging in three dimensions, *Proc. of Symposium on Interactive 3D Graphics*, ACM Press, 1990, pp.193-204

[3] Bier E., and Stone M., Snap-dragging, *ACM Computer Graphics (Proc. of SIGGRAPH '86)*, 1986, pp.223-240

[4] Hearn D., and Baker M.P., *Computer Graphics*, Prentice-Hall, 1994

[5] Falcao A.X et. al, User-steered image segmentation paradigms: livewire and livelane, *Graphical Models and Image Processing*, Vol. 60, 1998, pp.223-260

[6] Falcao, A.X, et al, An Ultra-fast user-steered image segmentation paradigm: live wire on the fly, *IEEE Tr. on Medical Imaging*, Vol.19, No.1, 2000, pp.55-62

[7] Floater, M.S., Parameterization and smooth approximation of surface triangulation, *Computer-Aided Geometric Design*, Vol.14, No.3, 1997, pp.231-250

[8] Garland, M. and Hecbert, P.S., Surface simplification using quadric error metric, *ACM Computer Graphics (Proc. of SIGGRAPH '97)*, 1997, pp.209-216

[9] Gleicher M., Image snapping, *ACM Computer Graphics (Proc. of SIGGRAPH'95),* 1995, pp.183-190

[10] Gleicher M., *A Differential Approach to Graphical Manipulation*, Ph.D. Carnegie Mellon University, 1994

[11] Gu, X., Gortler, S. and Hoppe H., Geometry images, *ACM Computer Graphics (Proc. of SIGGRAPH '02)*, 2002, pp.355-361

[12] Kang H.W. and Shin S.Y., Enhanced lane: interactive image segmentation by incremental path map construction, *Graphical Models*, Vol.64, No.5, 2002, pp.282-303

[13] Kass, M., Witkin, A., and Terzopoulos, D., Snakes, active contour models, *International Journal of Computer Vision*, Vol.1, 1987, pp.321-331

[14] Lee, Y., and Lee S., Geometric snakes for triangular meshes, *Computer Graphics Forum*, Vol.21, No.3, 2002, pp.229-238

[15] Mortensen E. and Barrett W.A., Intelligent scissors for image composition, *ACM Computer Graphics (Proc. of SIGGRAPH '95)*, 1995, pp.191-198

[16] Rosenfeld A. and Johnston E., Angle detection in digital curves, *IEEE Transactions on Computers*, Vol.22, 1973, pp 875-878

[17] Smith A.D.C., *The Folding of the Human Brain: from Shape to Function*, University of London, PhD Dissertations, 1999

[18] Stork A., An algorithm for fast picking and snapping using a 3d input device and 3d cursor, *CAD Tools and Algorithms for Product Design*, 1998, pp. 113-127
[19] Sutherland I., *Sketchpad: A Man Machine Graphical Communication System*, Ph.D. Thesis, MIT, 1963
[20] Turk G., Re-tiling polygonal surfaces, *ACM Computer Graphics (Proc. of SIGGRAPH '92)*, Vol.26, No.2, 1992, pp.55—64
[21] Vorsatz, J., Rossl, C., Kobbelt, L. and Seidel, H., Feature sensitive remeshing, *Proc. of EUROGRAPHICS '01*, 2001, pp.392-401.
[22] Yamaguchi, F., *Curves and surfaces in Computer Aided Geometric Design*, Springer-Berlag, 1988
[23] Yoo K.H. and J. S. Ha, Geometric snapping for 3d meshes, *Workshop on Computer Graphics and Geometric Modelling (Lecture Notes on Computer Science 3039)*, 2004, pp.90-97
[24] Yoon C.G., Kang D.W., and Chung S.M., *State-of-the-art in Fixed Prosthetics*, Jongee press, 1999