Geometric Snapping for 3D Meshes

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

[1] Dept. of Computer Education and Dept. of Information Industrial Engineering, Chungbuk National University, 48 San Gaesin-Dong Heungduk-Gu Cheongju Chungbuk 361-763 Republic of Korea
khyoo@cbucc.chungbuk.ac.kr

[2] Dept. of Computer Engineering, Woosuk University, 490 Hujongri, Samrae-Up Wanju-Kun Chonbuk 565-701 Republic of Korea
jsha@woosuk.ac.kr

Abstract. Image snapping is the technique to move a cursor position to a nearby feature such as edges in a 2D image when the cursor is located by a user. This paper proposes a new snapping technique called the *geometric snapping* that naturally moves the cursor position to a geometric feature in 3D meshes. The cursor movement is based on the approximate curvatures defined for measuring the geometric characteristics of the 3D meshes. The proposed geometric snapping can be applied to extracting geometric features of 3D mesh models in many CAD and graphics systems.

1 Introduction

The cursor snapping presented in Sketchpad systems is a well-known technique for interactively providing an exact cursor position in graphics user interfaces [12]. Many CAD and graphics systems have adopted the cursor snapping for picking 2D objects such as line segments, circles, polygons, and so on. This technique was extended into picking objects in 3D space by locating the cursor in a 2D screen [1,2,11]. Image snapping is another evolution of the cursor snapping proposed by Gleicher [5], which moves the cursor position to a nearby feature such as edges in an image when the cursor is located by a user. The image snapping can be applied to extracting edge features from an image as well as editing the image efficiently.

In order to visualize the world more realistically in CAD and graphics systems, there have been diverse 3D models describing objects in the real world. Most of the 3D models tend to be represented with 3D meshes for being effectively processed. One of the most important processing in the meshes is to detect the *geometric features* that represent the main boundaries of the 3D meshes, since they are crucial for deciding which parts of the meshes have to be processed or to be preserved in many applications such as mesh simplification, mesh compression, mesh editing, mesh morphing, and mesh deformation [4,13-16]. In mesh simplification and compression, the geometric features have to be maximally preserved. Mesh editing usually processes the parts representing geometric features in a mesh. Mesh morphing is also usually performed by using the corresponding geometric features between two meshes. Meshes can be deformed by manipulating their parts representing the geometric features.

In this paper, we propose the *geometric snapping* that can be used as a basic technique for effectively processing meshes. The geometric snapping extends the notion of image snapping to mesh models in the three-dimensional space. In other words, when a user selects an arbitrary vertex or point of a given 3D mesh model with the cursor, the cursor naturally moves to a nearby geometric feature of the mesh. There are two major issues in the extension; defining geometric characteristics on the mesh and moving the cursor onto the surface of the mesh. In this paper, the geometric characteristics are represented with the change of normals of adjacent faces or the radius of the circle passing the centers of three adjacent edges in the mesh. Moreover, we define the movement cost that is required when the cursor moves from a vertex into another vertex. Using the proposed measurements, we develop efficient techniques for the geometric snapping, and then apply them to extracting geometric features from an example mesh model.

2 Geometric Characteristics of 3D Meshes

The effectiveness of geometric snapping depends on the methods of measuring the geometric characteristics of meshes and the strategies of moving the selected vertex to a point on geometric features by the measurement. In general, the definition of geometric characteristics of a mesh may vary according to each application. In this paper, we use the approximate curvatures defined on a mesh to measure the geometric characteristics. This section introduces the methods for computing the approximate curvatures and blurring them.

2.1 Computing Approximate Curvatures

The curvature for a given point in a 3D mesh is defined as the curvature of the curve lying in a plane containing the vector tangent to the surface at the point. The curvature at a specific point on a curve in the plane is defined as the ratio of change in slope on the point. Because it is difficult to calculate exact curvatures on a surface, other curvatures are defined: *principal*, *Gaussian*, and *mean* curvatures. The principal curvatures at a point of a surface are the minimum and maximum of the curvatures at the point, the Gaussian curvature is the product of two principal curvatures, and the mean curvature is the half sum of two principle curvatures [17]. Since it is very difficult to exactly calculate these curvatures on faces of 3D mesh models, there are many attempts to obtain approximate curvatures on a point of the mesh. An important factor in approximating the curvatures is how to explain main geometric features of a mesh model. Many powerful methods [4,9,10,13-16] for obtaining the approximate curvatures have been proposed. This paper proposes new methods for reflecting the geometric characteristics of 3D meshes more exactly. In the first method, we define the approximate curvature $AC(v)$ on a vertex v in a given mesh by exploiting the normal vectors of faces containing v as:

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

In Equation (1), k is the number of faces sharing v, and f_i^v is the normal vector of the i-th face when the adjacency faces are ordered in counter-clockwise, and the operation $\cdot$ is the dot product of two vectors. The approximate curvature $AC(v)$ is defined as the subtraction of the minimum value among the inner products of normal vectors for all pairs of adjacent faces from 1.

As the second method, the curvature for a vertex v is the average of approximate curvatures on the edges incident to v. Let the ordered vertices adjacent to v be nv_i for all $i = 0, \ldots, k$–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 $C(ne_i)$ on the edge ne_i is defined as.

$$C(ne_i) = \frac{r_i^v + r_i^{nv}}{2} \tag{2}$$

In Equation (2), r_i^v is the radius of the circle passing the center of ne_i, and each center of two edges that are adjacent to ne_i while sharing v. Similarly, the radius r_i^{nv} of another circle is defined by ne_i and the two edges that are adjacent to ne_i while sharing nv_i. In addition to the proposed methods, we can compute approximate curvatures by applying other methods such as the quadric error metric [4], the inner product of two adjacent vertices [9], and the mean or Gaussian curvatures of edges or vertices [10].

2.2 Blurring Approximate Curvatures

If we regard approximate curvatures as height maps, the cursor movement of geometric snapping can be explained with a ball rolling down to valleys. The rolling ball may fall into local minimums before reaching at the deepest valley, which is caused by the limitation of computing method or the geometric characteristics themselves. In order to avoid this undesirable phenomenon, we soften up local minimums and emphasize global minimums by weighting the approximate curvature of each vertex on its nearby vertices. This technique is called the *blurring*. In this paper, we blur the approximate curvatures by using a well-known weighting factor called the *Gaussian smoothing filter*. That is, the approximate curvature on a vertex v is redefined as.

$$BAV(v) = \sum_{i=0}^{k-1} AC(nv_i) \times \frac{1}{\sqrt{2\pi}\sigma} \times e^{\frac{-(dx^2+dy^2+dz^2)}{2\sigma^2}} \tag{3}$$

In Equation (3), the vector (dx_i, dy_i, dz_i) 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 $nv_i = (nv_i^x, nv_i^y, nv_i^z)$. Since the smoothing degree of the Gaussian filter is determined by the size of σ, we assign appropriate values to σ according to the size of k; $\sigma = 0.85$ if $k \leq 7$, $\sigma = 1.7$ if $7 < k \leq 16$, $\sigma = 2.5$ if $16 < k \leq 36$, and $\sigma = 3.5$ if $k > 36$.

3 Geometric Snapping in 3D Meshes

Assume that a user selects a vertex on the 3D mesh in which each vertex has the approximate curvature computed by one of the methods described in Section 2. The cursor pointing to the selected vertex should be moved into other vertex appearing geometric features. In order to process the movement, we express a given mesh as a connected graph whose vertices and edges are just the ones of the mesh. Each vertex of the graph corresponds to that of the mesh, and has 3D coordinates and the approximate curvature. In this section, after establishing the cost function that is used as a criterion for moving the cursor vertex to other vertex, we develop several strategies how to move the cursor by using the cost function.

3.1 Move Cost Function

We estimate the cost required for moving the cursor from a vertex to another vertex over a mesh. Let u and v, respectively, be the current vertex and the next vertex to be chosen. The cost function for moving from u to v is denoted by $cost(u, v)$, which is defined similarly to that of an image pixel [8] as:

$$movecost(u, v) = \omega_z f_z(v) + \omega_d f_d(u, v) + \omega_g f_g(v). \tag{4}$$

In Equation (4), the three functions of 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 such as edges. From experimental results, we use the critical value of approximate curvatures for determining whether a vertex u represents the geometric features as; if $AC(v) > 2$ 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 defined as $f_d(u, v) = AC(u) - AC(v)$. If $f_d(u, v) > 0$, the cursor tends to move from u to v. Otherwise, the movement occurs conversely. The last function of the curvature magnitude $f_g(v)$ is the approximate curvature $AC(v)$ itself. Each ω is also 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, that is, the Laplacian zero-crossing and the curvature direction play important roles while the curvature magnitude has a little effects relatively. For nonadjacent two vertices u and v, we consider the cost function $movecost(u, v)$ for moving from u to v by using the shortest path $sp(u, v)$ from u to v. If $sp(u, v)$ consists of a sequence of k vertices, $u = v_1, \cdots, v_k = v$, then the cost function $movecost(u, v)$ can be defined as:

$$movecost(u, v) = \sum_{i=1}^{k-1} movecost(v_i, v_{i+1}) \tag{5}$$

3.2 Strategies for Moving the Cursor

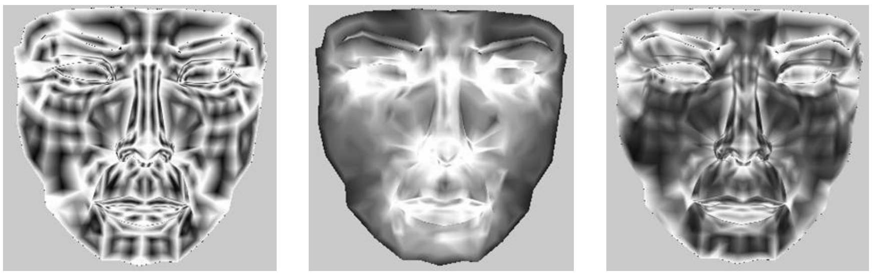
We consider three strategies for moving the cursor to a nearby geometric feature by computing the cost function. The first is to check the vertices adjacent to current vertex v. If the largest cost of the adjacent vertices is greater than 0, the cursor moves to the vertex with the largest. This movement is iterated until the costs of the vertices adjacent to current vertex are all zeros. 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 a 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 d may be determined by the adjacent vertex that is the farthest from v. An appropriate integer value n may be selected for the path length that is the minimum number of edges connecting two vertices. Hence, we check the vertices inside a sphere at origin v with the radius d, or the vertices whose path length to v is less than n. The cursor movement iterates same as the first method.

4 Experimental Results

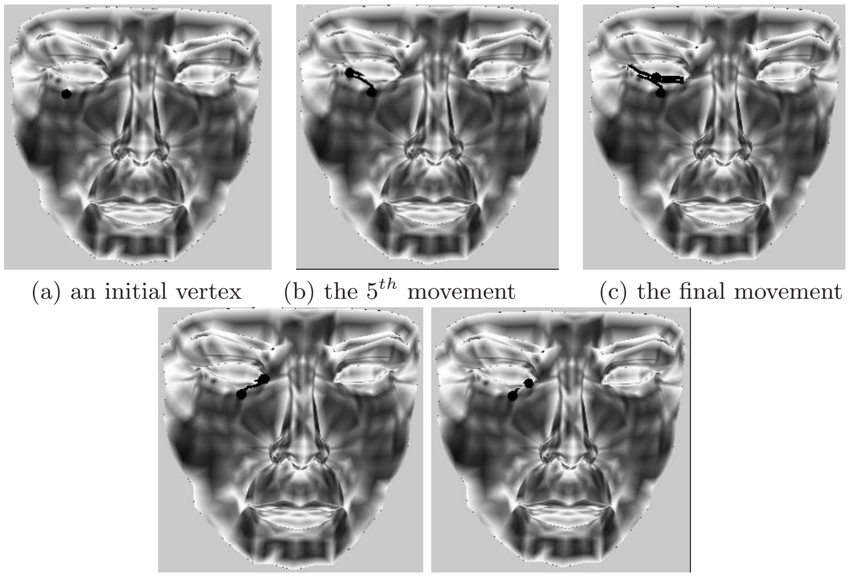
The proposed geometric snapping has been implemented on PC environments with the libraries of Microsoft Foundation Class (MFC) and OpenGL. The half-edge data structures are adopted for representing 3D meshes. We tested the implementation in the mesh of a face model. The approximate curvatures computed with Equation (1) and (2) in all vertices of the face model are visualized as grey colors in Fig. 1 (a) and (b) respectively, where the brightness depends on the magnitude of the curvatures. However, the computed approximate curvatures are discontinuous and noisy in some regions. To remove these phenomena, the approximate curvatures were blurred with Equation (3). Fig. 1 (c) illustrates the result from blurring the approximate curvatures in Fig. 1(a). For geometric snapping, a user first selects any vertex over a mesh that is preprocessed like Fig. 1(c). The located cursor will be moved into a nearby geometric feature within a certain neighboring range by using the *movecost* function in Equation (4) or Equation (5). Fig. 2 demonstrates the steps of the cursor movement in the first strategy using the adjacency for checking neighbors of the selected vertex: the 1^{st} movement (a), the 5^{th} movement (b), and the final movement (c). The final movements obtained by other strategies using a Euclidean length and a path length are shown in Fig. 2 (d) and (e) respectively. The cursor settled down after 3 movements for a determined Euclidean length, while 2 movements are needed for the path length 3. In these figures, the initial vertex selected by a user and the vertices passed by the cursor are black-colored. Solid lines represent the whole paths along which the cursor moved by the geometric snapping.

The problem of edge extraction in a 2D image [3,6,8] is very important for extracting feature boundaries in many applications. Similarly to the edge extraction in an image, we consider the extraction of geometric features such as



(a) Equation (1) (b) Equation (2) (c) Equation (3)

Fig. 1. Computing and blurring approximate curvatures



(a) an initial vertex (b) the 5^{th} movement (c) the final movement

(d) using Euclidean length (e) using path length

Fig. 2. Moving the selected cursor

eyes, eyebrows, noses, and lips in a facial mesh model. Various applications such as facial deformation and facial animation in the facial model have needed the effective process of the geometric features. To extract the geometric features for a 3D mesh model, Lee and Lee [7] proposed the geometric snake as one method that is the extension of an image snake [6]. In this paper, we use the procedure of the geometric snapping for the geometric extraction; a sequence of vertices selected in this procedure can be identified as the geometric features. Fig. 3 (a) shows the result from applying the geometric snapping to extracting the boundary of lips. The black vertex is the one selected by a user, and the black solid lines represent the trace along which the cursor moves from the selected vertex when the geometric snapping is applied iteratively. Fig. 3 (b) and (c) are the results from iteratively applying the geometric snapping to extracting the lower boundary and the whole boundary of a left eye respectively.

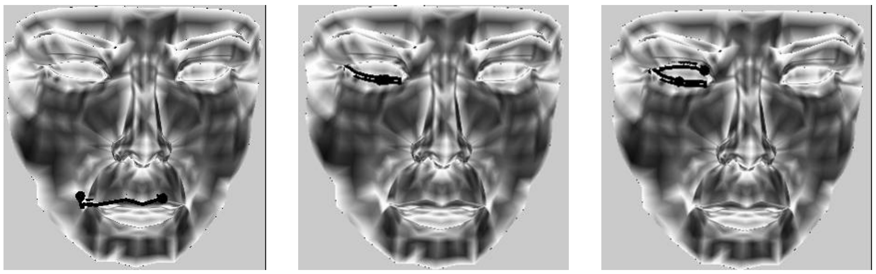

(a) lips (b) a left eye I (c) a left eye II

Fig. 3. Extracting geometric features

5 Conclusion

This paper proposed the technique of geometric snapping that naturally moves the cursor from a selected vertex to other vertex representing a geometric feature in a 3D mesh. We applied it to extracting geometric features from a face model. In the future, it is required to develop another technique of geometric snapping, which considers the relations among several vertices selected by a user.

The geometric features obtained by iteratively applying our geometric snapping are represented with a set of open or closed polygonal lines connecting a selected vertex and other vertices passed during the iteration. This is because a mesh consists of vertices, edges and faces. Hence, the obtained geometric feature may have the shape of staircases. It is also very important to remove these staircases of the extracted geometric features.

References

1. Bier, E., Snap-Dragging in Three Dimensions. Proc. Of Symposium on Interactive 3D Graphics, ACM Press, (1990), 193-204.
2. Bier, E., Stone, M., Snap-Dragging, Proc. Of SIGGRAPH'86, ACM Press, (1986), 223-240.
3. Falcao, A.X., User-Steered Image Segmentation Paradigms: Live Wire and Live Lane, Graphical Models and Image Processing **60**, (1998), 223-260.
4. Garland, M., Hecbert, P.S., Surface Simplification using Quadric Error Metric, ACM Computer Graphics (Proc. Of SIGGRAPH'97), (1997) 209-216.
5. Gleicher, M., Image Snapping, ACM Computer Graphics (Proc. of SIGGRAPH'95), (1995,) 183-190.
6. Kass, M., Witkin, A., Terzopoulos, D., Snakes, Active contour models. Int. Journal of Computer Vision **1**, (1987), 321-331.

7. Lee, Y., Lee, S., Geometric Snakes for Triangular Meshes, EuroGraphics Forum, (2002).
8. Mortensen, E., Barrett, W.A., Intelligent scissors for image composition, ACM Computer Graphics (Proc. of SIGGRAPH '95), (1995), 191-198.
9. Rosenfeld, A., Johnston, E., Angle Detection in Digital Curves, IEEE Transactions on Computers **22**, (1973), 875-878.
10. Smith, A.D.C., The folding of the human brain: from shape to function, PhD Dissertations, University of London, (1999).
11. Stork, A., An Algorithm for Fast Picking and Snapping using a 3D Input Device and 3D Cursor, CAD Tools and Algo-rithms for Product Design, (1998), 113-127.
12. Sutherland, I., Sketchpad: A Man Machine Graphical Communication System, PhD Dissertations, MIT, (1963).
13. Kobbelt, L.P., Bischoff, S., Botsch, M., Kehler, K., Ressl, C., Schneider, R., Vorsatz, J., Geometric modeling based on polygonal meshes, EUROGRAPHICS 2000 Tutorial, (2000).
14. Gu X., Gortler S., Hoppe H., Geometry images, Proceedings of SIGGRAPH' 02, (2002), 355-361.
15. Vorsatz, J., Rossl, C., Kobbelt, L., Seidel, H., Feature Sensitive Remeshing, Proc. of EUROGRAPHICS '01, (2001), 393-401.
16. Alliez, P., Cohen-Steiner, D., Levoy, B., Desbrun, M., Anisotropic Polygonal Remeshes, Proceedings of SIGGRAPH '03, (2003), 485-193.
17. Yamaguchi, F., Curves and surfaces in Computer Aided Geometric Design, Springer-Berlag, (1988).