



ELSEVIER

Computer-Aided Design 36 (2004) 665–677

COMPUTER-AIDED  
DESIGN

[www.elsevier.com/locate/cad](http://www.elsevier.com/locate/cad)

# Reduce the stretch in surface flattening by finding cutting paths to the surface boundary

Charlie C.L. Wang\*, Yu Wang, Kai Tang, Matthew M.F. Yuen

*Department of Mechanical Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, People's Republic of China*

Received 15 July 2002; received in revised form 12 February 2003; accepted 13 February 2003

## Abstract

This paper presents a method for finding cutting paths on a 3D triangular mesh surface to reduce the stretch in the flattened surface. The cutting paths link the surface boundary and the nodes where the Gaussian curvature is high, and their total length is minimized. First, a linear algorithm for computing an approximate boundary geodesic distance map is introduced; the map encapsulates the undirected geodesic distance from every triangular node to the surface boundary approximately. This is followed by determining the undirected shortest paths passing through all the nodes where the Gaussian curvature is larger than a threshold. The cutting paths walk along the triangular edges of the given surface. Compared with other similar approaches, our method reaches a faster speed, and can deal with surfaces with widely distributed curvatures.

© 2003 Published by Elsevier Ltd.

**Keywords:** Surface flattening; Gaussian curvature; Cutting path; Shortest path; Geodesic distance

## 1. Introduction

Surface flattening plays a prominent role in engineering and manufacturing applications, such as aircraft design, vehicle design, garment design, etc. Isometric surface development is also a key procedure for texture mapping in computer graphics. It is known that flattening a 3D surface into a plane introduces surface metric (angle, distance, areas) stretch unless the surface has zero Gaussian curvature everywhere [1]—Gaussian curvature is the product of the maximum and minimum normal curvatures at a given point. Thus, the stretch depends directly on the Gaussian curvature of the given surface. Cutting the surface across points of high Gaussian curvature will reduce the stretch in the flattened surface. However, since cutting paths modify and elongate the boundary of a surface, they often introduce additional constraints to the following applications (e.g. they may lead to discontinuity in texture mapping). Therefore, the length of cutting paths has to be minimized. Subdividing triangles might lead to difficulties in the following applications after surface flattening (e.g., some finite element applications), so our cutting paths walk

along the triangular edges of the given triangular mesh surface.

When introducing our algorithm, the following definitions are necessary.

**Definition 1.1.** The *boundary geodesic distance map* is a map that encapsulates the undirected geodesic distance from every node to the boundary on the given mesh surface.

**Definition 1.2.** A *flipped triangle* is a triangle whose normal direction after flattening is along the negative  $z$ -axis while the normal direction of other flattened triangles is along the positive  $z$ -axis.

**Contribution.** This paper describes a method to find the cutting paths on a 3D triangular mesh surface to reduce the stretch in the flattened surface. The shortest cutting paths connecting all nodes where Gaussian curvature is larger than a threshold are generated according to our newly defined boundary geodesic distance map. The map encapsulates the undirected geodesic distance from every triangular node to the surface boundary approximately, and can be computed in linear time. The cutting paths on surfaces with widely distributed curvatures can also be found by generating cuts during the flattening process.

\* Corresponding author.

E-mail address: [cwang@acae.cuhk.edu.hk](mailto:cwang@acae.cuhk.edu.hk) (C.C.L. Wang).

Compared with other similar approaches, our method reaches a faster speed while generating an acceptable result; it can deal with surfaces with widely distributed curvatures.

After reviewing the related work in Section 2, the rest of the paper is organized as follows. In Section 3, a linear algorithm is introduced to compute the approximate boundary geodesic distance map of a surface; this is followed by generating the approximate shortest path from a selected node to the surface boundary by the map. In Section 4, we propose a reduction of the stretch in the flattened surface by adding shortest cutting paths from high Gaussian curvature nodes to the surface boundary. During the surface flattening process, cutting paths are also incorporated to prevent flipped triangles. Finally, in Section 5, experimental results are shown and discussed.

## 2. Related work

Carmo [1] defined a developable surface as: for a ruled surface— $X(t, v) = \alpha(t) + v\beta(t)$ , it is developable if  $\beta$ ,  $d\beta/dt$  and  $d\alpha/dt$  are coplanar for all points on  $X$  (where  $\alpha(t)$  is the *base curve* and  $\beta(t)$  is the *director curve* of  $X(t, v)$ ). The simplest examples of developable surfaces are cylinders and cones, and a simple and representative example of non-developable surfaces is sphere. Every surface enveloped by a one-parameter family of planes is a developable surface. Generally, a surface is developable if and only if the Gaussian curvature of every point on it is zero. It is difficult to satisfy this condition when given a freeform polygonal mesh surface. Thus, it is important to introduce cutting paths to make the given surface as close as possible to the developable condition.

Gaussian curvature is the product of the maximum and minimum normal curvatures at a given point [1]. However, since differential geometry analyzes surfaces that are sufficiently differentiable, the equation for calculating the Gaussian curvature cannot be applied to a mesh surface directly. A discrete Gaussian curvature computing method is needed. Kobbelt et al. [2] gave the formulas of discrete Gaussian curvature based on the fact that meshes can be interpreted as approximations of smooth surfaces. The idea of their approach is to discretize a theorem for defining the Gaussian curvature on a smooth surface derived from a theorem by Rodrigues [1]. In the same way, Sheffer [3] gave another Gaussian curvature approximation, which is scale independent. In our approach, we adopt the formula of Kobbelt et al. [2] to compute the Gaussian curvature of every internal triangular node on a mesh surface.

Insertion of cuts or darts in surface flattening has been studied for a long time. Parida and Mudur [4] presented an algorithm to obtain planar development (within acceptable tolerances) of complex surfaces with cuts and overlaps only in specified orientations. Their algorithm first obtains an approximate planar surface by flattening triangles, cracks are generated while triangles are flattened one by one; and then, they reorient cracks and overlap parts in the developed

plane to satisfy orientation constraints. Their algorithm might generate many cracks and calculation errors. Aono et al. [5,6] introduced a reverse approach to surface flattening, which inserts darts to fit a woven cloth model to a curved surface. Other approaches are based on local strain energy minimization technology [7,8]. The approach of Wang et al. [9] generates the cutting line from the stretch energy distribution map; however, the length of cutting paths is not considered in their paper. Recently, Sheffer [3] tried to find the shortest cutting path that passes through the nodes with high Gaussian curvature to reduce the parameterization distortion of the triangulated surface. Unfortunately, this method is not able to find protrusions with widely distributed curvatures (e.g. looped cylindrical surfaces); and a modified Dijkstra algorithm is applied in this method to compute the shortest paths, whose running time is  $O(E \times \log N)$ , where  $N$  is the number of triangular nodes and  $E$  is the number of triangular edges. Our method presented in this paper can compute the shortest path from a node to the surface boundary in linear time; and the cutting paths on the surface with widely distributed curvatures are generated while preventing flipped triangles in surface flattening.

The single source shortest path problem (SSSP) is one of the classic problems in algorithmic graph theory [10]: given a weighted graph with a source vertex, find the shortest path from the source vertex to all other vertices in the graph. For the shortest path problem on a polygonal surface, approaches based on Dijkstra algorithm [11] are most popular solutions [3,12,13]. Their computing time is non-linear though. Thorup [14] presented a method to compute the undirected single source shortest paths in linear time; this is the fastest solution. Here, our problem is different from SSSP. We need to select a node from candidate nodes where Gaussian curvature is larger than a threshold. The final selected node should have the shortest geodesic distance to the surface boundary among the candidate nodes. The algorithm of Thorup cannot be directly applied here since it cannot give the geodesic distance from multiple points to the surface boundary in linear time. Thus, the newly defined boundary geodesic distance map is needed, which is different from the weighted graph with a source vertex in SSSP. After the node is selected, the shortest cutting path from it to the surface boundary walking along the triangular edges can also be computed in linear time.

## 3. Approximate shortest path to surface boundary

In this section, we introduce a method to approximate the shortest path from a selected node to the surface boundary in linear time. First, a boundary geodesic distance map, which indicates the undirected geodesic distance from every triangular node to the boundary on the mesh surface, is computed in linear time. After that, the approximate shortest path is generated from the map, also in linear computing time.

### 3.1. Boundary geodesic distance map

We generate the boundary geodesic distance map using a boundary advancing method, which progressively moves the event list  $L_v$  of the triangular nodes from the boundary to the center of the surface. During the movement, the geodesic distance between the event list before and after moving is used to update the boundary geodesic distance of every passed node. The given triangular mesh  $M$  is stored in a graph  $G(v, e)$ .  $G(v, e)$  is composed of the vertices and the edges of  $M$ . Every node has its adjacent nodes and edges stored; and every edge has its bounding nodes stored. The geodesic distance from every vertex  $v_i$  to the surface boundary is stored as a weight factor  $W_{v_i}$ . Before moving the event list, the  $W_{v_i}$  of every internal vertex is initialized as  $+\infty$  the  $W_{v_i}$  of every boundary vertex is set to zero; and the length of every edge  $e_j$  is calculated and stored. Our algorithm repeatedly moves  $L_v$  from the boundary to the center of  $M$ ; during the movement, the  $W_{v_i}$  of the nodes adjacent to  $L_v$  are updated. The pseudo-code for the boundary geodesic distance map generation is given as follows.

**Algorithm** MapGeneration( $G$ )

**Input:** A graph  $G(v, e)$  of the given mesh surface  $M$ .

**Output:** The updated weight factor  $W_{v_i}$  of every triangular node.

1. **for** every node  $v_i \in G$  {
2.      $W_{v_i} \leftarrow +\infty$ ;
3.     Set the passed flag of  $v_i - fp_{v_i}$  to **false**;
4.     **if** ( $v_i$  on the boundary)
5.         Add  $v_i$  to  $L_v$ ,  $W_{v_i} \leftarrow 0$ , and set the passed flag of  $v_i - fp_{v_i}$  to **true**;
6.     }
7.     Calculate the length  $l_{e_j}$  of every edge  $e_j \in G$ ;
8.      $L'_v \leftarrow \phi$ ;
9.     **do**{
10.         **for** every node  $v_k \in L_v$  {
11.             **for** every node  $v_j$  adjacent to  $v_k$  {
12.                 **if** ( $(W_{v_k} + \text{the length of edge } v_j v_k) < W_{v_j}$ ), **then**  $W_{v_j} \leftarrow W_{v_k} + \text{the length of edge } v_j v_k$ ;
13.                 **if** ( $fp_{v_j}$  is **false**), **then** add  $v_j$  to  $L'_v$  and set  $fp_{v_j}$  to be **true**;
14.             }
15.         }
16.         Replace  $L_v$  by  $L'_v$  and empty  $L'_v$ ;
17.     }**while**( $L_v \neq \phi$ );

After running MapGeneration( $G$ ), the weight factor  $W_{v_i}$  of every node indicates the approximate geodesic distance from the node  $v_i$  to the boundary of  $M$ . The complex of  $W_{v_i}$ , called  $W$ , and the graph  $G(v, e)$  comprise the approximate boundary geodesic distance map  $B = W + G$ . The visualization for

the isohypses generated from  $B$  to the surface given in Fig. 1(a) is shown in Fig. 1(c). From Fig. 1(c), we find that our boundary geodesic distance map approximately indicates the geodesic distance from every surface point to the boundary.

In Algorithm MapGeneration( $G$ ), we assume that the lengths of edges are close to each other; so the weight difference between the nodes of  $L_v$  and  $L'_v$  is a constant. However, when the edges in  $M$  do not have the same length, an approximation error will be generated by Algorithm MapGeneration( $G$ ) since the weight difference between the nodes of  $L_v$  and  $L'_v$  is no longer a constant. For example, in the surface shown in Fig. 2(a) which has the same 3D shape as the one in Fig. 1(a), the lengths of some edges are much longer than others; the isohypses of  $B$  generated by Algorithm MapGeneration( $G$ ) are shown in Fig. 2(b), which are apparently distorted. To reduce this error, when selecting a node to add to  $L'_v$ , the weight difference between the nodes of  $L_v$  and  $L'_v$  should be controlled. Only nodes that make the weight difference less than some threshold value will be added into  $L'_v$ . We

use the length of the shortest triangular edge as the incremental step of the threshold in our modified algorithm. Also, the node in  $L_v$  should also be added to  $L'_v$  if the event list has not passed any of its adjacent nodes before. The modified algorithm is adaptive with

the length of triangular edges. It is outlined below as Algorithm AdaptiveMapGeneration ( $G$ ).

in Fig. 2(c); and Fig. 3(c) shows the weight factor of nodes in Fig. 1(c). Comparing them, it is easy to find that the lengths of

**Algorithm** AdaptiveMapGeneration( $G$ )

**Input:** A graph  $G(v, e)$  of the given mesh surface  $M$ .

**Output:** The updated weight number  $W_{v_i}$  of every triangular node.

1. **for** every node  $v_i \in G$  {
2.      $W_{v_i} \leftarrow +\infty$ ;
3.     Set the passed flag of  $v_i - fp_{v_i}$  to **false**;
4.     **if** ( $v_i$  on the boundary)
5.         Add  $v_i$  to  $L_v$ ,  $W_{v_i} \leftarrow 0$ , and set the passed flag of  $v_i - fp_{v_i}$  to **true**;
6.     }
7. Calculate the length  $l_{e_j}$  of every edge  $e_j \in G$ , and store the minimum edge length as  $l_{\min}$ ;
8.  $L'_v \leftarrow \phi$ ;
9.  $\lambda \leftarrow l_{\min}$ ;
10. **do**{
11.     **for** every node  $v_k \in L_v$  {
12.         **for** every node  $v_j$  adjacent to  $v_k$  {
13.             **if** ( $(W_{v_k} + \text{the length of edge } v_j v_k) < W_{v_j}$ ), **then**  $W_{v_j} \leftarrow W_{v_k} + \text{the length of edge } v_j v_k$ ;
14.             **if** ( $(fp_{v_j} \text{ is false})$  and  $(W_{v_j} < \lambda)$ ), **then** add  $v_j$  to  $L'_v$  and set  $fp_{v_j}$  to **true**;
15.         }
16.     }
17.     **for** every node  $v_k \in L_v$
18.         **if** any  $fp_{v_i}$  of its adjacent node  $v_j$  is **false**, **then** add  $v_k$  to  $L'_v$ ;
19.     Replace  $L_v$  by  $L'_v$  and make  $L'_v$  empty;
20.      $\lambda \leftarrow \lambda + l_{\min}$ ;
21. } **while** ( $L_v = \phi$ );

After applying Algorithm AdaptiveMapGeneration ( $G$ ) to the mesh surface given in Fig. 2(a), the isohypses of the result are shown in Fig. 2(c). Fig. 3(a) shows the weight factor of nodes in Fig. 2(b); Fig. 3(b) shows the weight factor of nodes

triangular edges have less influence on the result of Algorithm AdaptiveMapGeneration ( $G$ ). Thus, the isohypses in Fig. 1(c) and 2(c) are the same except in the area where there is no any triangular node.

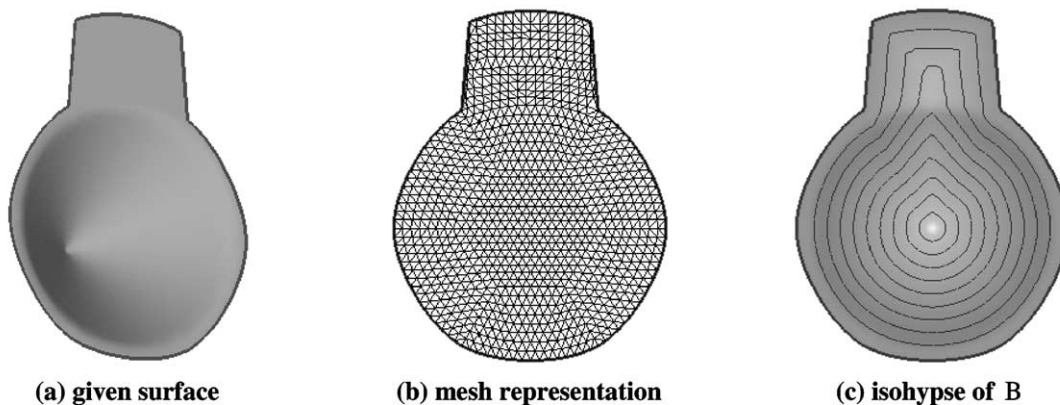


Fig. 1. Boundary geodesic distance map.

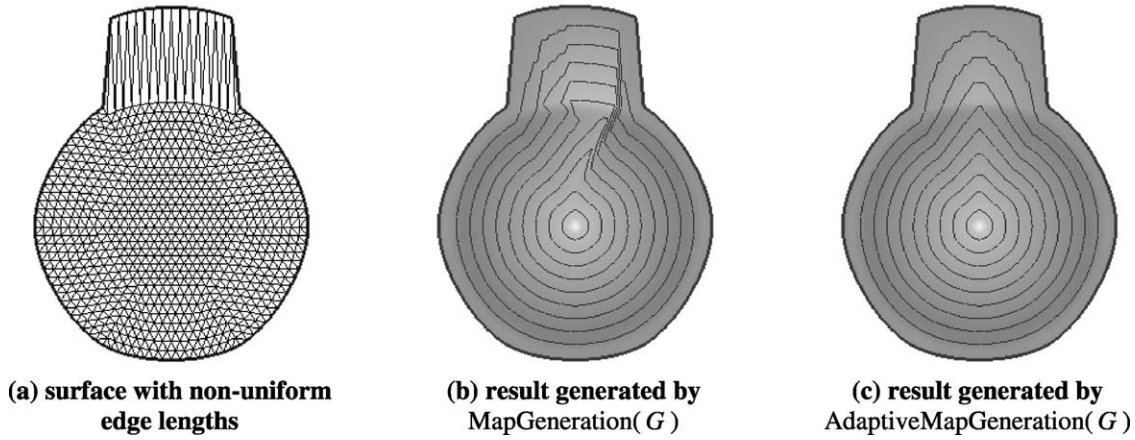


Fig. 2. Approximation error generated by non-uniform edge lengths.

**Computing time analysis.** For Algorithm MapGeneration( $G$ ), the running time of step 1–6 is  $O(N)$ , where  $N$  is the number of triangular nodes; the running time of step 7 is  $O(E)$ , where  $E$  is the number of triangular edges; and during step 9–17, since every node visits its adjacent nodes once—in other words, every edge is passed twice, the running time is  $O(E)$ . Therefore, the running time of Algorithm MapGeneration( $G$ ) is  $O(N + E)$ . For Algorithm AdaptiveMapGeneration( $G$ ), the running time of step 1–6 is also  $O(N)$ ; and the running time of step 7 is  $O(E)$ . For the running time of step 11–16, every node visits its adjacent nodes  $\text{INT}(\|v_j v_k\|/l_{\min})$  times, where  $\text{INT}(\dots)$  denotes the function to round a number down to the nearest integer. Thus, the running time in the worst case is  $O(TE)$ , where  $T = \text{INT}(l_{\max}/l_{\min})$ , and  $l_{\max}$  and  $l_{\min}$  denote the maximum and minimum edge length on the given surface  $M$ . The running time of step 17–18 is  $O(E)$  since every node visits its adjacent nodes once. In summary, the running time of Algorithm AdaptiveMapGeneration( $G$ ) is  $O(N + (T + 1)E)$ . Generally,  $T$  is a small number (e.g. from 1 to 100). Therefore, the running time of

the Algorithm AdaptiveMapGeneration( $G$ ) is also linear in  $N$  and  $E$ .

### 3.2. Generate approximate shortest path

We generate the approximate shortest path from a select node  $v_s$  to the surface boundary by the steepest descent method [15] according to the boundary geodesic distance map. For any selected triangular node  $v_i \in G$ , all its adjacent nodes are candidates for forming the path. We choose the node  $v_j$ , whose descent function  $f_d(v_i, v_j)$  has the maximum value among all the adjacent nodes of  $v_i$ . The definition of  $f_d(v_i, v_j)$  is

$$f_d(v_i, v_j) = (w_{v_i} - w_{v_j})/\|v_i v_j\|. \quad (1)$$

Starting from node  $v_s$ , after searching for the adjacent nodes with the maximum  $f_d(v_i, v_j)$  one by one, the final path is generated. For example, in Fig. 4, the circled node is  $v_s$ , and the bolded edges are the final path. The pseudo-codes of the path generation algorithm are given below as Algorithm PathGeneration( $v_s, B$ ).

---

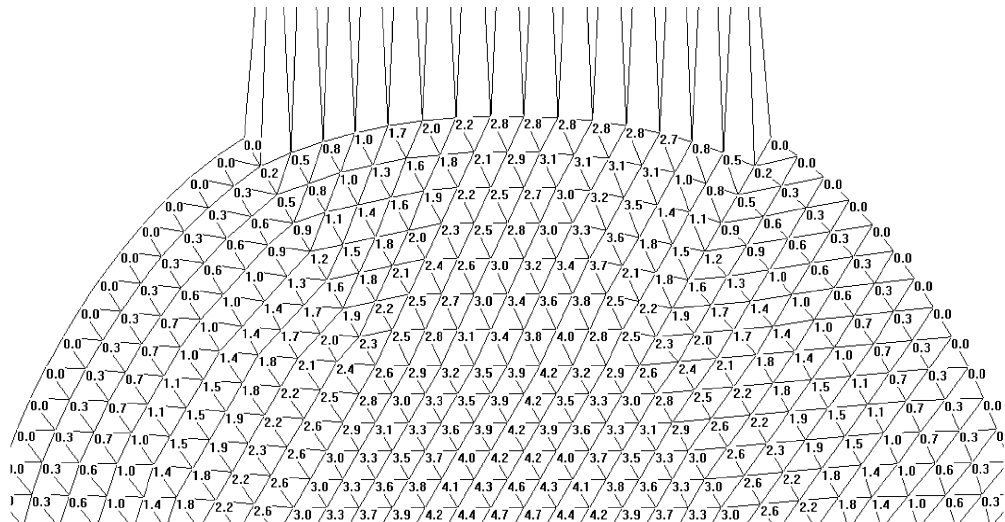
#### Algorithm PathGeneration ( $v_s, B$ )

**Input:** Boundary geodesic distance map  $B$  of the given mesh surface  $M$ .

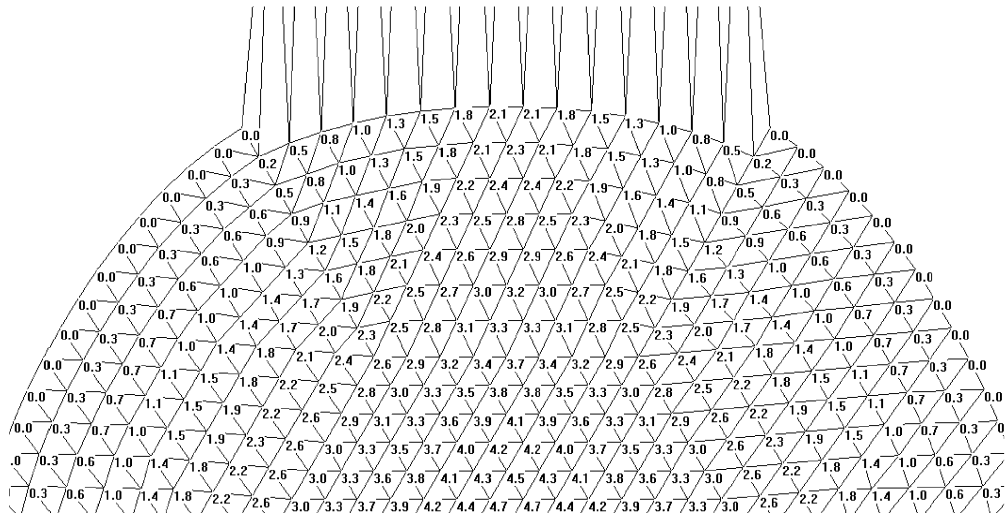
**Output:** The cutting path  $P$ , consisting of triangular edges.

1.  $P \leftarrow \emptyset$ ;
2. **while**( $W_{v_s} \neq 0$ ) {
3.    $v_{\max} \leftarrow$  any node adjacent to  $v_s$ ;
4.   **for every** node  $v_j$  adjacent to  $v_s$
5.     **if** ( $f_d(v_s, v_j) > f_d(v_s, v_{\max})$ ), **then**  $v_{\max} \leftarrow v_j$ ;
6.   Add the edge  $v_s v_{\max}$  into  $P$ ;
7.    $v_s \leftarrow v_{\max}$ ;
8. }  
- 9. **return**  $P$ ;

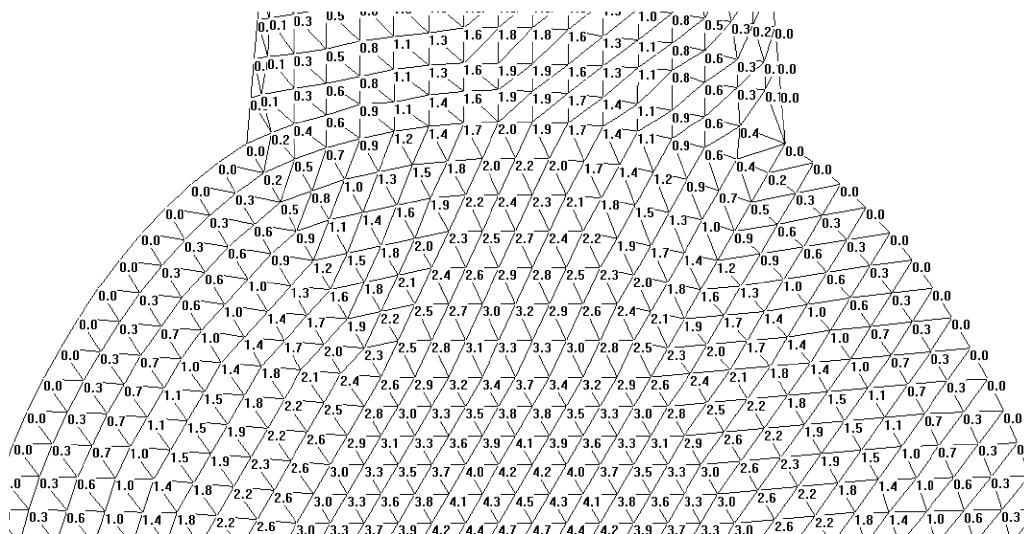




(a) weight factor of nodes in Fig. 2b



(b) weight factor of nodes in Fig. 2c



(c) weight factor of nodes in Fig. 1c

Fig. 3. Boundary geodesic distance map by weight factors.

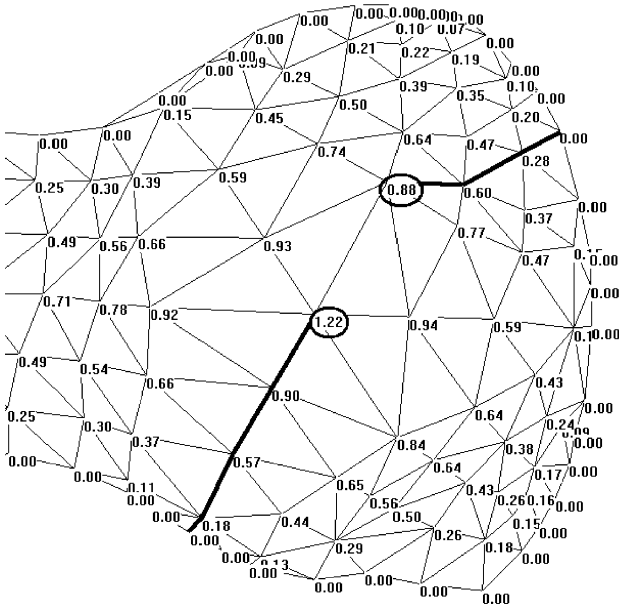


Fig. 4. Path generated from the boundary geodesic distance map.

In the worst case, every node on the given mesh surface  $M$  visits its adjacent node once; so the computing time is  $O(E)$ . In summary, we can generate the approximate shortest path from a selected node to the surface boundary in linear time.

#### 4. Reduce stretch in surface flattening

Using the technique of approximating the shortest path to the boundary, we develop a method to determine the cutting paths passing through all the nodes where the Gaussian curvature is higher than some given value. After that, cutting paths may also be incorporated in surface flattening to prevent flipped triangles.

##### 4.1. Node selection

As alluded earlier, the stretch in the flattened surface depends directly on the Gaussian curvature of the given

surface. Cutting the surface across points with a high Gaussian curvature can thus reduce the stretch in surface flattening. Gaussian curvature is not well defined mathematically on a polygonal mesh surface, so a discrete approximation is needed. Here, we conduct the approximation of Kobbelt et al. [2] on every internal triangular node  $v_i$ . The formula of Ref. [2] is

$$\kappa_{v_i} = \frac{2\pi - \sum_j \theta_j}{\frac{1}{3} \sum_j A_j}, \quad (2)$$

where  $\theta_j$  are the inner angles adjacent to  $v_i$  and  $A_j$  are the corresponding triangle areas.

The Gaussian curvature  $\kappa_{v_i}$  of the nodes on the boundary of the given surface is zero since its adjacent triangles are not closed. All the triangular nodes, whose  $\kappa_{v_i} > \varepsilon$ , are candidates. In different cases, a different  $\varepsilon$  can be chosen. A larger  $\varepsilon$  leads to shorter cutting paths, and a smaller  $\varepsilon$  always leads to longer cutting paths. In our testing examples, we usually choose  $\varepsilon = 0.8\kappa_{\max}$ , where  $\kappa_{\max}$  is the maximum Gaussian curvature among all the interior nodes on the given mesh surface.

##### 4.2. Add cutting path

After the nodes whose Gaussian curvatures are higher than  $\varepsilon$  are determined and stored in a candidate nodes list  $L_c$ , the shortest path connecting these nodes and the surface boundary is determined by an incremental method. First, we compute the boundary geodesic distance map  $B$  of the surface; secondly, we remove a node  $v_c$  from  $L_c$  whose weight factor  $w_{v_c}$  is a minimum in  $L_c$ ; thirdly, we use Algorithm PathGeneration ( $v_c, B$ ) to generate a cutting path from  $v_c$  to the surface boundary. After that, we add the newly generated path into the surface boundary, and go back to the first step to compute the new boundary geodesic distance map. We determine the cutting path through all the candidate nodes by repeating the above steps until the candidate nodes list  $L_c$  becomes empty.

In order to construct a fast algorithm, we store the candidate nodes list in a minimum heap  $H_c$ ; so the computing

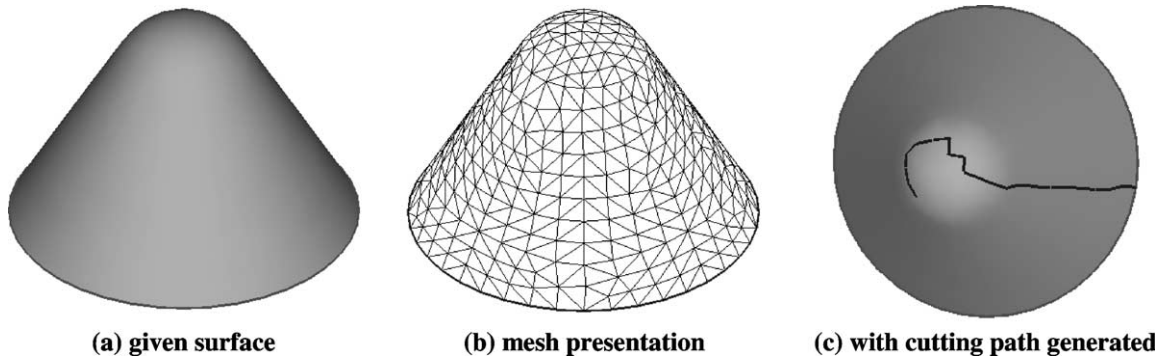


Fig. 5. Example I—umbrella surface.

time of any operation on a single node in  $H_c$  is  $O(\log N_c)$ , where  $N_c$  is the number of nodes in  $H_c$ . After computing the boundary geodesic distance map of the given surface in linear time, it only takes a time of  $O(N_c \log N_c)$  to update the minimum heap  $H_c$  and a time of  $O(\log N_c)$  to remove any node  $v_c$  from  $H_c$ . Therefore, the total computing time of our algorithm is  $O((N + (T + 1)E)N_c \log N_c)$ . The pseudo-codes of our algorithm are shown as Algorithm DetermineCuttingPathReducingStretch ( $M$ ). Two examples of cutting path generation are shown in Figs. 5 and 6, where the bolded curves are the determined cutting paths.

#### 4.3. Surface flattening

After the surface cutting path is determined, a spring-mass model based on the energy function is used to flatten the 3D mesh surface into its corresponding 2D pattern [9]. This procedure consists of triangles flattening and planar mesh deformation. During the triangles flattening phase, triangles are flattened one by one; and a partial spring-mass system containing flattened triangles is deformed to release the strain energy during the flattening. After all the triangles are flattened, the spring-mass system will have all the triangles of the given surface. The planar triangular mesh deformation

**Algorithm** DetermineCuttingPathReducingStretch ( $M$ )

**Input:** The given mesh surface  $M$  with boundary  $B_M$ .

**Output:** The cutting path  $P^*$  connecting  $B_M$  and nodes with high Gaussian curvature.

1.  $P^* \leftarrow \phi$ ;
2. **Call** AdaptiveMapGeneration ( $G$ ) to generate the boundary geodesic distance map  $B$  of  $M$ ;
3. **for** every internal nodes  $v_i \in M$
4.     Compute the Gaussian curvature  $\kappa_{v_i}$  of  $v_i$  by eq.(2);
5. **for** every boundary node  $v_b \in B_M$
6.      $\kappa_{v_b} \leftarrow 0$ ;
7. **for** every node  $v_i \in M$
8.     **if** ( $\kappa_{v_i} > \varepsilon$ ), **then** add  $v_i$  into the minimum heap  $H_c$  by the value of  $w_{v_i}$ ;
9. **while** ( $H_c \neq \phi$ ) {
10.     Remove the top node  $v_c$  from  $H_c$ ;
11.     **Call** PathGeneration ( $v_c, B$ ) to generate the cutting path  $P$ ;
12.     Add  $P$  into  $P^*$ ;
13.     Add every triangular edge in  $P$  into  $B_M$ ;
14.     **Call** AdaptiveMapGeneration ( $G$ ) to generate the updated  $B$  by the updated boundary  $B_M$ ;
15.     Update  $H_c$  by the new  $w_{v_i}$  of every node  $v_i \in H_c$ ;
16. }
17. **return**  $P^*$ ;

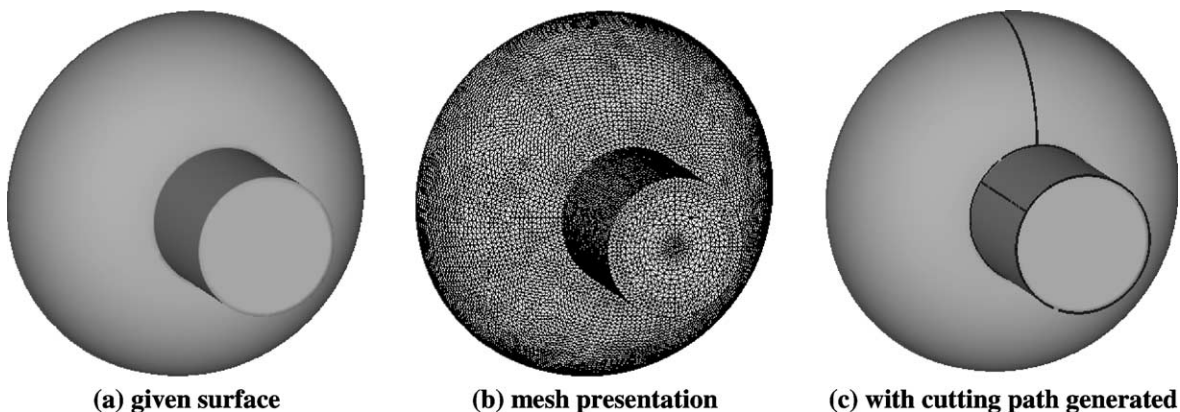


Fig. 6. Example II—bottle surface.



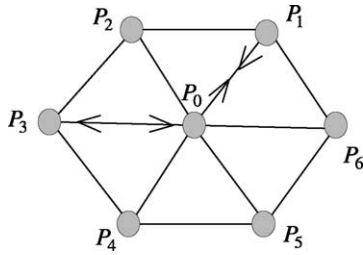


Fig. 7. Example of a node in a spring-mass system.

process is directed by the energy function of the spring-mass system. One example of a spring-mass system is shown in Fig. 7. In this example, nodes  $P_i$  are masses that correspond to the vertex  $v_i \in M$ ; and the links between masses  $P_i$  and  $P_j$  are springs. During deformation, if the distance between  $P_i$  and  $P_j$  on the planar surface is larger than the distance between them on the original spatial surface, we apply an attraction force between them (e.g. the force between  $P_0$  and  $P_1$ ); otherwise there will be a repellent force between them (e.g.  $P_0$  and  $P_3$ ).

The energy function on one single mass  $P_i$  to be minimized is

$$E(P_i) = \sum_{j=1}^n \frac{1}{2} C(|P_i P_j| - d_j^2) \quad (3)$$

where  $C$  is the spring constant,  $|P_i P_j|$  is the current distance between  $P_i$  and  $P_j$  on the planar surface, and  $d_j$  is the geodesic distance between  $v_i$  and  $v_j$  on the given mesh surface  $M$ . The energy function for the whole surface patch is

$$E(M) = \sum_{i=1}^N E(P_i). \quad (4)$$

By releasing the energy function, we can obtain the 2D pattern corresponding to the given 3D mesh surface. The surface flattening results of example I (Fig. 5) and II (Fig. 6) are shown in Fig. 8.

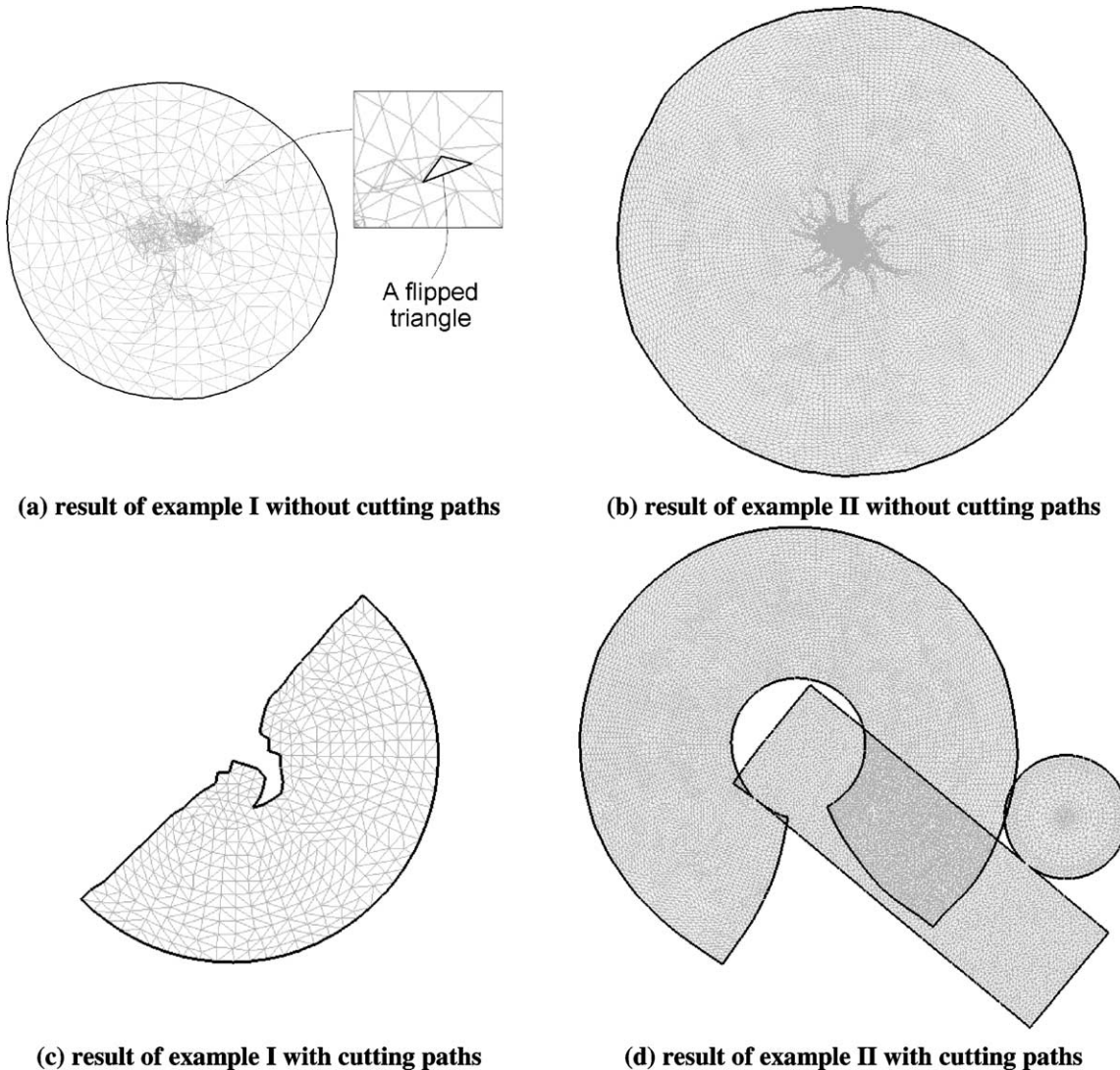


Fig. 8. Surface flattening results without vs. with cutting paths.

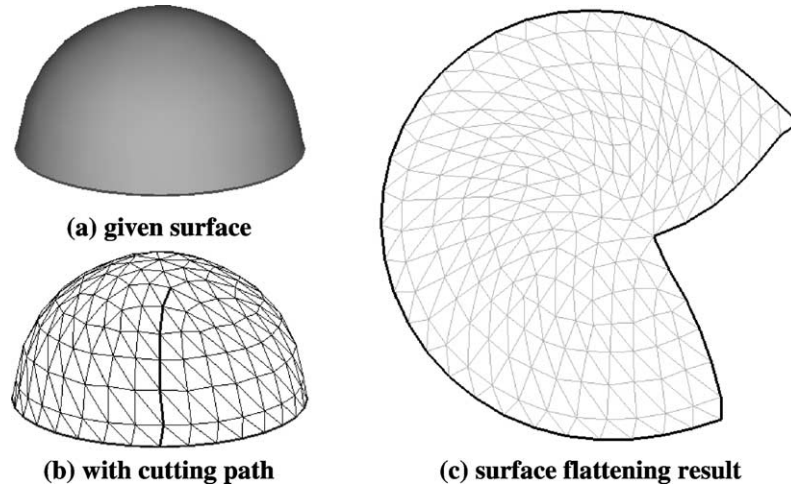


Fig. 9. Example III—hemisphere.

#### 4.4. More cuts generation

The Algorithm DetermineCuttingPathReducingStretch ( $M$ ) cannot give efficient cutting paths to the surfaces with widely distributed curvatures (e.g. a cylindrical surface or a hemisphere). Alternative methods are introduced in this section to reduce stretch in these cases.

For the case of a surface patch with widely distributed curvatures, if the surface has only one boundary loop, we incorporate the cutting path from the flipped triangles to the surface boundary. The procedure consists of 4 steps: step 1, flatten the given mesh  $M$  by the method given in Section 4.3; step 2, check the orientation of every triangle and find the flipped triangles; step 3, find the node  $v_m$  with the minimum weight factor  $w_{v_m}$  in the flipped triangles; step 4, incorporate the cutting path by calling Algorithm PathGeneration ( $v_m, B$ ); then go back to step 1 until no flipped triangle is found. The cutting path determined by this method on a hemisphere and its flattening result is shown in Fig. 9.

If a surface patch with widely distributed curvatures has multiple boundary loops, we incorporate the cutting path to connect these loops. We can determine the shortest path connecting the two loops— $B_1$  and  $B_2$  from a modified geodesic distance map  $B'$ . To determine  $B'$ , when using Algorithm AdaptiveMapGeneration ( $G$ ) to compute the geodesic distance map, in step 3–4, only the vertices on  $B_1$  are set to be of zero weight and are

passed. After determining  $B'$ , the vertex  $v_c$  on  $B_2$  with the minimum weight factor can be found. Calling this Algorithm PathGeneration ( $v_c, B'$ ), we obtain the shortest path connecting the two boundary loops. The cutting path determined by this method on a cylindrical surface and its corresponding flattening result are shown in Fig. 10.

When a given mesh surface  $M$  is closed, Algorithm DetermineCuttingPathReducingStretch( $M$ ) will fail, as there is no boundary to work on. With reference to Ref. [16], we only need to add an initial cutting edge  $e_{init}$  passing through the vertex  $v_{max}$ , which has the maximum Gaussian curvature on  $M$ .  $e_{init}$  is the shortest adjacent triangular edge of  $v_{max}$ . After  $e_{init}$  is given, we determine the final cutting path by calling Algorithm DetermineCuttingPathReducingStretch( $M$ ). A closed surface example, a cube, is shown in Fig. 11.

#### 5. Experimental results

In general, we can determine the cutting path to reduce the stretch in surface flattening by passing through the nodes with high Gaussian curvatures. In example I, the given mesh surface has 774 triangles; when we choose  $\varepsilon = 0.8\kappa_{max}$  as the threshold, 23 cutting edges passing through seven high curvature nodes are computed in less than 1 s. In example II, the given mesh surface has 21122 triangles; when we choose

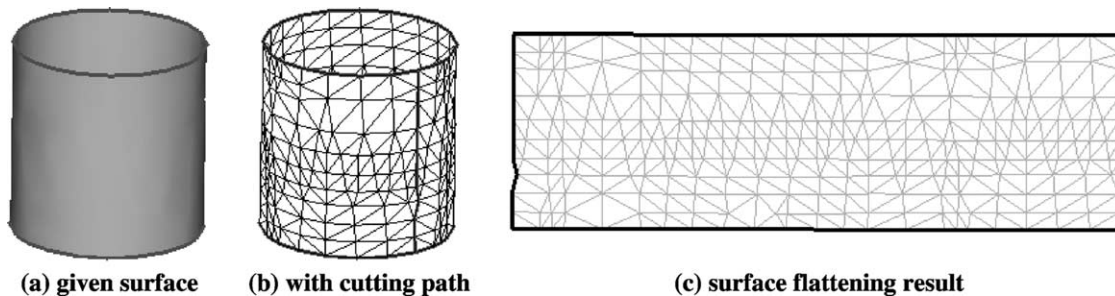


Fig. 10. Example IV—cylindrical surface.

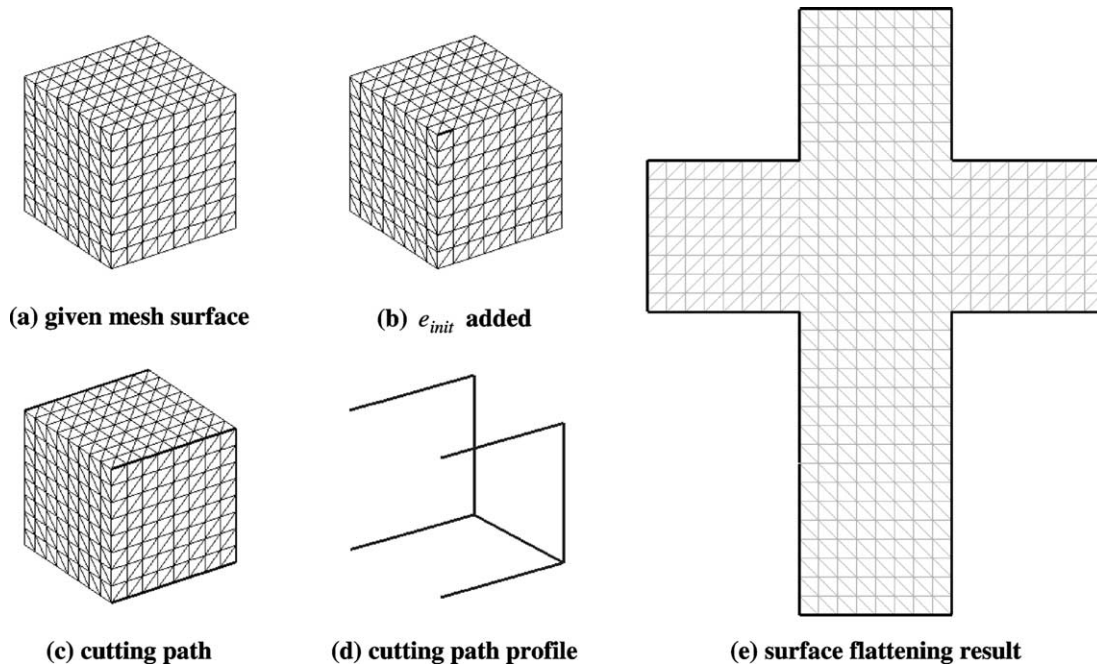


Fig. 11. Example V—closed surface.

$\varepsilon = 0.8\kappa_{\max}$  as the threshold, 268 cutting edges passing through 200 high curvature nodes are determined in 93 s.

In order to compare our method with the others that are based on the similar idea of using cutting paths, we select

a same example as [3]—the cow (shown in Fig. 12(a)), which has 5804 triangles. When we choose  $\varepsilon = 0.8\kappa_{\max}$  as the threshold, 478 cutting edges passing through 279 high curvature nodes (shown in Fig. 12(b) and (c)) are computed

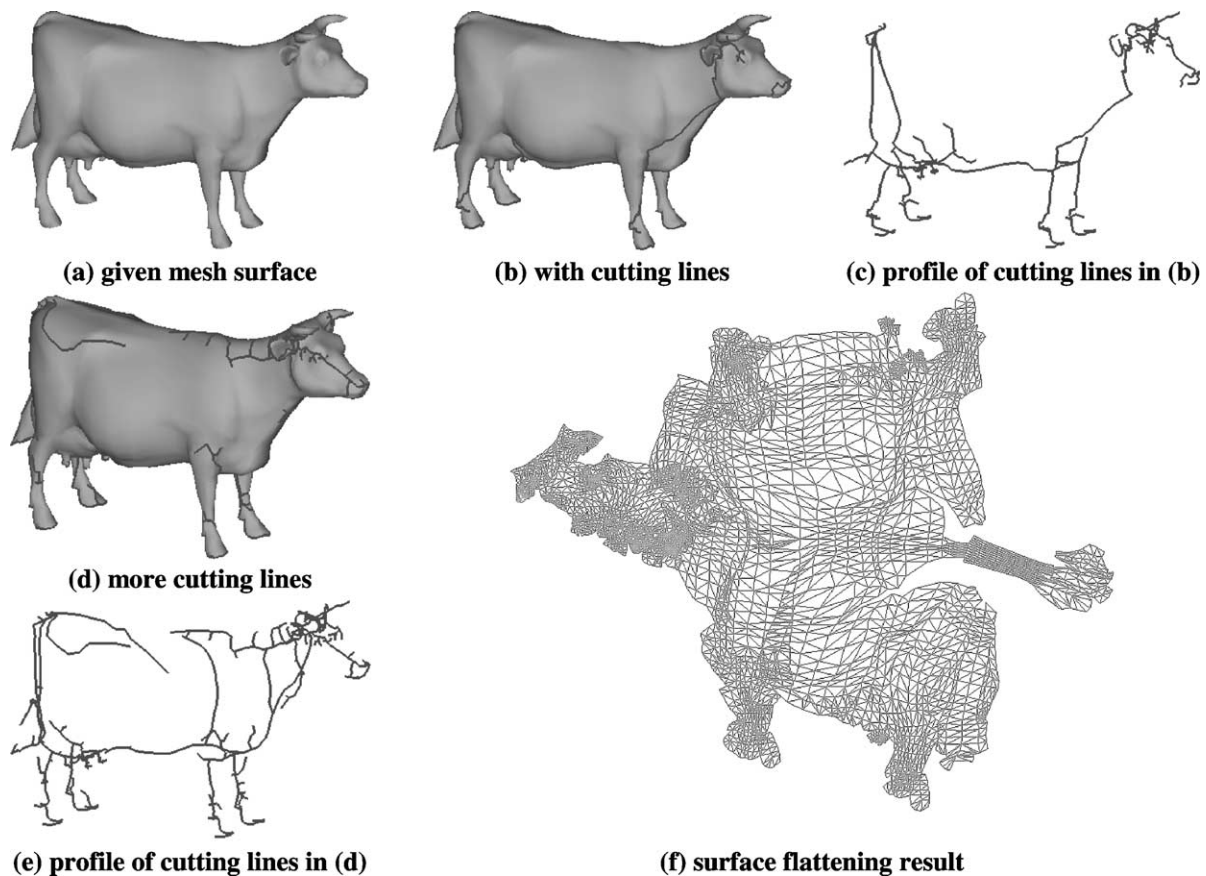


Fig. 12. Example VI—cow.



in 47 s. This compares favorably to the computing time reported in Ref. [3] which is close to two minutes when 326 cutting edges passing through 98 nodes are incorporated. Since the cow is a surface with widely distributed curvatures, more cutting edges should be introduced to reduce the stretch by the method of Section 4.4. Fig. 12(d) and (e) show the final cutting edges, and Fig. 12(f) shows its corresponding surface flattening result. All tests in this paper were conducted on a PIII 600 MHz PC with 128 MB RAM.

## 6. Conclusion

In this paper, we develop a method for finding the cutting paths on a 3D triangular mesh surface to reduce the stretch on the flattened surface. The length of cutting paths is minimized. First, a linear algorithm to compute an approximate boundary geodesic distance map is developed. The algorithm is adaptive to different lengths of triangular edges. After that, the undirected shortest path from a selected node to the surface boundary can be generated according to the map; the geodesic distance from the selected node to the surface boundary is the shortest among all the nodes whose Gaussian curvature is higher than a threshold. Using this idea, we can reduce the stretch in the flattened surface by adding shortest cutting paths from high Gaussian curvature nodes to the surface boundary one by one. The cutting paths walk along the triangular edges of the given surface. By generating more cuts from flipped triangles during flattening, even closed surfaces or surfaces with widely distributed curvatures can be cut and flattened.

As a new technique for enhancing the existing surface flattening techniques, our method has the following two advantages compared with other methods based on similar ideas of using cutting paths:

1. our method reaches a faster speed while generating an acceptable result;
2. our method can deal with surfaces with widely distributed curvatures.

Industrial applications of surface flattening impose more requirements on the cutting paths. For example, the direction of the cutting path, or the distribution of the cutting paths may be constrained. Further research can focus on how to incorporate these constraints into the surface flattening technique.

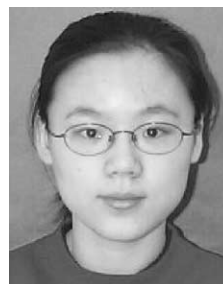
## References

- [1] Carmo MPD. Differential geometry of curves and surfaces. Prentice Hall: Englewood Cliffs, NJ; 1976.
- [2] Kobbelt LP, Bischoff S, Botsch M, Kähler K, Rössl C, Schneider R, Vorsatz J. Geometric modeling based on polygonal meshes. EUROGRAPHICS 2000 Tutorial.
- [3] Sheffer A. Spanning tree seams for reducing parameterization distortion of triangulated surface. SMI 2002: International Conference on Shape Modelling and Applications.
- [4] Parida L, Mudur SP. Constraint-satisfying planar development of complex surfaces. *Comput-Aid Des* 1993;25(4):225–32.
- [5] Aono M, Denti P, Breen DE, Wozny MJ. Fitting a woven cloth model to a curved surface: dart insertion. *IEEE Comput Graph Appl* 1996; 16(5):60–70.
- [6] Aono M, Breen DE, Wozny MJ. Modeling methods for the design of 3D broadcloth composite parts. *Comput-Aid Des* 2001;33(13): 989–1007.
- [7] McCartney J, Hinds BK, Seow BL. The flattening of triangulated surfaces incorporating darts and gussets. *Comput-Aid Des* 1999; 31(4):249–60.
- [8] Kim SM, Kang TJ. Garment pattern generation from body scan data. *Comput-Aid Des* 2003;35(7):611–18.
- [9] Wang CCL, Smith SSF, Yuen MMF. Surface flattening based on energy model. *Comput-Aid Des* 2002;34(11):823–33.
- [10] Cormen TH, Lieseron CE, Rivest RL. Introduction to algorithms. Cambridge: MIT Press; 2000.
- [11] Dijkstra EW. A note on two problems in connection with graphs. *Numerische Mathematik* 1959;1:269–71.
- [12] Lanthier MA, Maheshwari A, Sack J-R. Approximating weighted shortest paths on polyhedral surfaces. *Proceedings of 13th ACM Symposium on Computational Geometry*; 1997. p. 274–83.
- [13] Kanai T, Suzuki H. Approximate shortest path on a polyhedral surface and its applications. *Comput-Aid Des* 2001;33(11):801–11.
- [14] Thorup M. Undirected single source shortest paths in linear time. *Proceedings of 38th IEEE Symposium on Foundations of Computer Science*; 1997. p. 12–21.
- [15] Press WH. Numerical recipes in C: the art of scientific computing. Cambridge: Cambridge University Press; 1992.
- [16] Gu X, Gortler SJ, Hoppe H. Geometry images. *SIGGRAPH 2002 Conference Proceedings*; 2002.



**Charlie C.L. Wang** is currently a visiting scholar in the Department of Mechanical Engineering at the Hong Kong University of Science and Technology (HKUST). He gained his B.Eng. (1998) from Mechatronics at the Huazhong University of Science and Technology, China. He joined HKUST in 1998 to go on his study by the recommendation of the Ministry of Education P. R. China for his excellent grades in the B.Eng. He received both his M.Phil. (2000) and PhD (2002) from the Mechanical Engineering at HKUST. His

research interests include design automation and optimization, geometric modeling, reverse engineering, CAD/CAM, and computer graphics.



**Yu Wang** is a PhD student in the Department of Mechanical Engineering at the Hong Kong University of Science and Technology (HKUST). She gained her B.Eng (1999) from Mechanical Engineering and Automation of the Huazhong University of Science and Technology, China. She joined HKUST at 2000 and received her M.Phil. degree at 2002. Her research interests include CAD, geometric modeling, computer vision, computer graphics and image processing.



**Kai Tang** is currently a faculty member in the Department of Mechanical Engineering at Hong Kong University of Science and Technology. Before joining HKUST in 2001, he had worked for more than 13 years in the CAD/CAM and IT industries. His research interests concentrate on designing efficient and practical algorithms for solving real world computational, geometric, and numerical problems. Dr Tang received PhD in Computer Engineering from the University of Michigan in 1990, MSc in Information and

Control Engineering in 1986 also from the University of Michigan, and BSc in Mechanical Engineering from Nanjing Institute of Technology in China in 1982.



**Matthew M.F. Yuen** has extensive research experience in design and manufacturing automation. He spent four years in UK industry before taking up a teaching position in Hong Kong in 1979 and received the 1987 Edwin Walker Prize from the Institute of Mechanical Engineers, UK. Prof. Yuen is currently a Full Professor of the Department of Mechanical Engineering at the Hong Kong University of Science and Technology. He has provided consultancies in design and manufacturing

automation, and vibration control. His research interests include intelligent CAD/CAM system, soft object modeling, electronic packaging, rapid prototyping, vibration control and polymer processing.