CyberTape: an interactive measurement tool on polyhedral surface

Charlie C. L. Wang*
*E-mail: cwang@acae.cuhk.edu.hk; Tel: (852) 2609 8052; Fax: (852) 2603 6002
Department of Automation and Computer-Aided Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Abstract

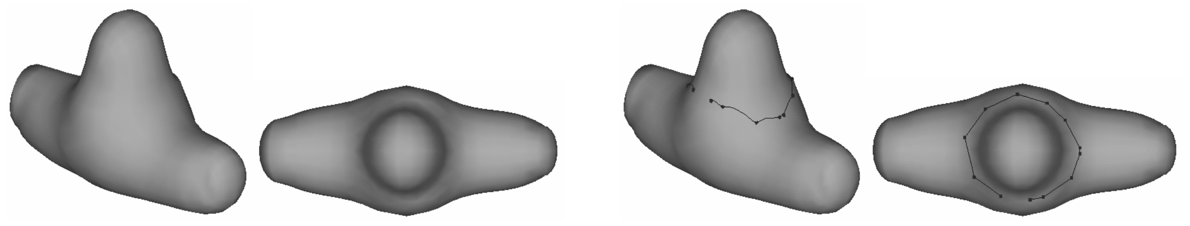
Polyhedral mesh surfaces are widely utilized to represent objects reconstructed from 3D ranged images. In computer-aided engineering, it is desired to not only observe but also measure these three-dimensional objects. This paper presents an approach to measure the curve distance between two points on a polyhedral surface in the manner that simulates dragging a tapeline at the two points. After generating the initial measurement curve through the leading points in linear computing time, an iteration algorithm is presented to approximate stretching the measurement curve on the given polyhedral surface; as an option, the obtained measurement curve can be further stretched to leave the measured surface in some concave places – this likes what a tapeline behaves in reality. This novel interactive tool allows users to perform measurement tasks in an intuitive and natural way in virtual space. Our implementation algorithm can be completed in real time on a standard PC. At the end of the paper, applications of this tool are given to demonstrate its functionality.

Keywords: virtual reality; interaction techniques; computer-aided design; manufacturing.

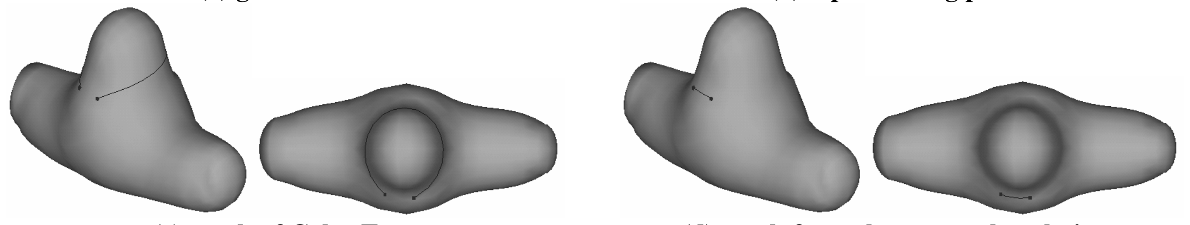
1. Introduction

A lot of techniques for constructing a polyhedral mesh surface by the point cloud from three-dimensional ranged images have been developed in the last fifteen years [1-5]. After obtaining the polyhedral mesh representation of an object in computer, tools for manipulating the object in a virtual space are expected. In literature, there are some techniques for cutting 3D mesh surfaces [6, 7] or dragging a feature on the surface of a mesh [8]. However, there is no method for measuring a polyhedral surface in the manner of a tapeline like. In this paper, we present a novel approach to measure the curve distance between two points on a polyhedral surface in the manner that simulates dragging a tapeline at the two points – the tool is named as CyberTape. The function of our CyberTape tool is illustrated in Fig.1. In this example, the given model is a closed polyhedral surface (Fig. 1a). Users can interactively specify some leading points (small cubes in Fig. 1b) on the surface;

after applying our approach, the result measurement curve that simulates dragging a tapeline is generated as shown in Fig. 1c, where the starting and ending points are fixed – they are called *location points* in this paper. From this example, it is easy to find that the interactive measurement tool presented here is very intuitive and efficient.



(a) given model **(b) input leading points**



(c) result of CyberTape **(d) result from shortest path technique**

Fig.1 CyberTape – an example to illustrate its behavior

There are many algorithms for finding the shortest path between two points on 3D surfaces. The algorithms for determining the *extract* shortest path on a polyhedral surface [9, 10] (including the non-convex case) usually take high time costs. It is impractical to apply these algorithms to the dense polyhedral surface in a real-time system. Instead, the techniques [11-13] appear, which focus on generating the *approximate* shortest path on a polyhedral surface. During them, the time cost of the fastest approach is $O(n \log n)$. They are based on using the Dijkstra's algorithm [14] to compute the undirected shortest path. If the approach of Thorup [15] is used to compute the undirected shortest path, the computing time could be reduced. However, these shortest path techniques cannot be directly adopted in our approach. It is because that the shortest path techniques do not have the ability to reflect users' intention of how to measure the given surface. For example, when computing the shortest path between the two location points given in Fig.1b, the shortest path between them is as shown in Fig.1d, which is opposite to the user wanted direction. Also, if the shortest path through all leading points is computed, all the leading points are fixed on the surface – this does not follow the behavior of a physical tapeline. The result of our method is encourage – the whole measurement curve can slip on the surface, which is much closer to using a tapeline to measure the surface of a physical object in practice.

Our paper is organized as follows. After given necessary preliminaries in section 2, the detail algorithm of our method is given in three steps. Firstly, the initial measurement curve is generated through the input leading points in linear time in section 3; compared to [14], the algorithm given here is in more implementation detail.

After determining the initial measurement curve through leading points, the section 4 presents an iteration algorithm to simulate stretching the measurement curve on the given surface. In section 5, as an option, the stretched curve is further improved to leave the concave surface parts like a physical tapeline. Finally, example industrial applications of our tool are given.

2. Preliminaries

Before introducing the algorithms in detail, some preliminaries are first given in this section.

Definition 2-1 A polyhedral surface M is defined as a pair (K,V), where K is a simplicial complex specifying the connectivity of the vertices, edges, and faces (in other words, the topological graph of M), and $V=\{v_1,\cdots,v_m\}$ is the set of vertices defining the shape of polyhedral surface in $\Re^3$.

The above definition follows the notation in [16]. In this paper, to simplify the algorithm, every polygonal face in M is subdivided into triangles by the method of [17]. From K, it is very easy for our algorithm to get the adjacent nodes, edges, and faces of a triangular node in constant time; the same, the left/right faces of an edge and the three nodes/edges of a triangle can also be obtained by constant time cost. The geometric coverage of our method is confined to the domain of two-manifold polyhedral surface. The definition is as follows.

Definition 2-2 For every point on the surface of a two-manifold object, there exists a sufficiently small neighborhood that is topologically the same as an open disk in $\Re^2$; if there is any points on the boundary that do not satisfy the two-manifold condition, the object is classified as *non-two-manifold*, or simply *non-manifold*.

For a non-manifold polyhedral object, it should be converted into several two-manifold polyhedral patches before applying our measurement tool. The measurement curves, which are generated in this approach, are not allowed to change the topology and shape of the given surface. Thus, they cannot be stored in M; they are stored as attribute curves attached on M. In order to represent attribute curves, four kinds of attribute elements are conducted in our data structure. They are defined in Tables 1 and 2, where ATTRIB_EDGENODEs and ATTRIB_FACENODEs are derived form ATTRIB_NODEs. A ATTRIB_EDGENODE is an attribute node on a triangular edge, whose coordinates depend on the position of the triangular edge by a parameter *u*; and a ATTRIB_FACENODE is an attribute node in a triangular face, whose coordinates are given by (*u, v, w*) that relates to the three nodes of the triangle. An ATTRIB_EDGE is an ordered collection list of ATTRIB_NODEs, which are ATTRIB_EDGENODEs or ATTRIB_FACENODEs.

After giving the above definitions, we will go into the detail algorithm description parts in the following of the paper.

Table 1 Representational attributes

Attribute	**Comprises**	**Represents physically**
ATTRIB_NODE	A point	An attached point on the given polyhedral surface.
ATTRIB_EDGE	Complex of ATTRIB_NODEs	A curve attached on the given surface. It is an ordered list of ATTRIB_NODEs.
ATTRIB_EDGENODE	A point	An attached point on a triangular edge. Its position depends on the positions of the two endpoints of the triangular edge.
ATTRIB_FACENODE	A point	An attached point in a triangular face. Its position depends on the positions of the three nodes of the triangular face.

Table 2 Pseudo-code of attributes

- ATTRIB_NODE

```
ATTRIB_NODE {
    FLAGS            flg;              // Status flags
    ATTRIB_EDGE      *attr_edge;       // ATTRIB_EDGEs contain this node
};
```

- ATTRIB_EDGE

```
ATTRIB_EDGE {
    FLAGS            flg;              // Status flags
    MESHSURFACE      *mesh_surface;    // Polyhedral surface contain this edge
    ATTRIB_NODE      **attr_node;      // Pointer of ATTRIB_NODEs list
};
```

- ATTRIB_EDGENODE

```
ATTRIB_EDGENODE : public ATTRIB_NODE {
    Double           u;                // Parameter coordinate of this node
    TRGLEDGE         *trgl_edge;       // TRGLEDGE contain this node
};
```

- ATTRIB_FACENODE

```
ATTRIB_FACENODE : public ATTRIB_NODE {
    Double           u, v, w;          // Areal coordinate of this node
    TRGLFACE         *trgl_face;       // TRGLFACE contain this node
};
```

3. Initial Measurement Curve

As the first step of the CyberTape tool, the algorithm in this section will generate the initial measurement curve linking the leading points. The leading points are interactively specified on the given polyhedral surface using the picking tool, which is an existing function in many popular graphics systems. First, a *geodesic distance map* that approximately indicates the geodesic distance from every triangular node to a leading point on the polyhedral surface is computed in linear time. After that, the approximate shortest path walking along triangular edges between two adjacent leading points is generated from the map, also by linear time cost.

3.1. Geodesic distance map

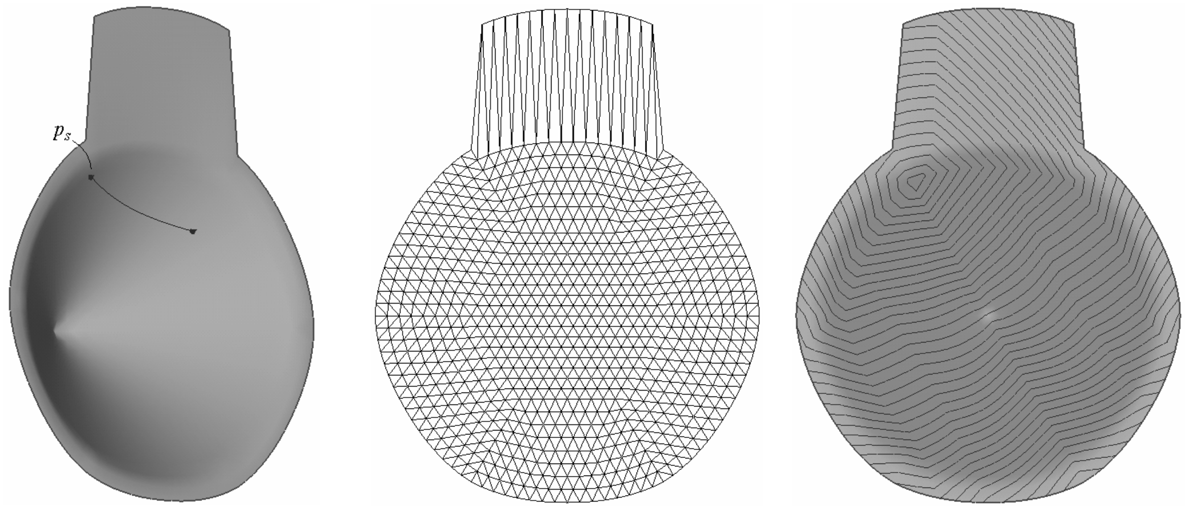
If the point p_s is a leading point, the geodesic distance map of p_s – $G_M(p_s)$ is generated by an advancing method, which progressively moves the event list L_v of nodes away from p_s on the given surface M. The geodesic distance form every vertex v_i to the point p_s is stored as a weight factor W_{v_i}. Before starting to move the event list, the W_{v_i} of every internal vertex is initialized as $+\infty$, and the length of every triangular edge e_j is calculated and stored. Our algorithm repeatedly moves L_v away from p_s on M; during the movement, the weight factors W_{v_i} of the nodes neighboring L_v are updated. The pseudo-code for the algorithm to generate the geodesic distance map is given in Table 3.

After running ***Algorithm*** MapGeneration(M , p_s), the weight factor W_{v_i} of every node indicates the approximate geodesic distance from the vertex v_i to p_s. The complex of W_{v_i}, called *W*, and the pair (*K, V*) comprise the approximate geodesic distance map $G_M(p_s) = W + (K, V)$. Given the surface and the point p_s given in Fig.2a, the visualization for isohypses is generated from $G_M(p_s)$ as shown in Fig.2c. The mesh presentation (Fig.2b) of the given surface shows that the shape of triangles in M is not uniform – the upper triangles are much longer than the other triangles; however, our algorithm can still generate a quasi-uniform $G_M(p_s)$ shown in Fig.2c. Consequently, the shape of triangles in M has less influence on $G_M(p_s)$ derived by ***Algorithm*** MapGeneration(M , p_s).

For ***Algorithm*** MapGeneration(M , p_s), 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 only once – in other words, every edge is passed twice, the running time is $O(E)$. Therefore, the running time of ***Algorithm*** MapGeneration(M, p_s) is $O(N + E)$.

Table 3 Pseudo-code of *Algorithm* MapGeneration(M , p_s)

```
Algorithm MapGeneration( $M$ , $p_s$ )
Input: The given polyhedral surface $M$ and the source point $p_s$.
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 triangle containing $p_s$ )
5.          Add $v_i$ to $L_v$, $W_{v_i} \leftarrow$ distance between $v_i$ and $p_s$, and set $fp_{v_i}$ to true;
6.  }
7.  Calculate the length $l_{e_j}$ of every edge $e_j \in M$;
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 (( $W_{v_k}$ + the length of edge $v_j v_k$ ) < $W_{v_j}$ ), then $W_{v_j} \leftarrow W_{v_k}$ + 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( $L_v \neq \phi$ );
```



(a) given surface with p_s **(b) mesh representation** **(c) isohypse of $G_M(p_s)$**

Fig.2 Geodesic distance map

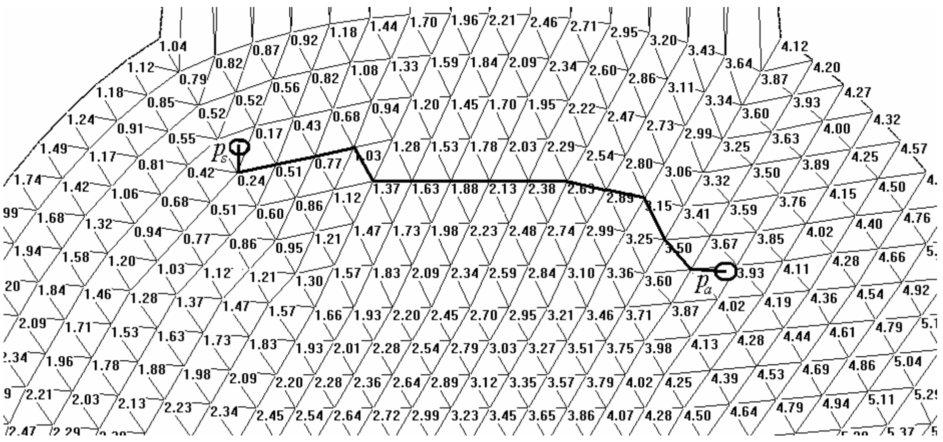


Fig.3 Determine the undirected shortest path by $G_M(p_s)$

3.2. Curve generation

The user input leading points are stored as ATTRIB_FACENODEs in a list L_0. In order to construct an initial measurement curve, we generate the approximate shortest paths between the adjacent leading points in L_0. For any leading point $p_a \in L_0$, the approximate shortest path P between p_a and p_s is formed by the steepest descent method [18] according to the geodesic distance map $G_M(p_s)$ of p_s, where p_a and p_s are neighboring points in L_0. The approximate shortest path walks along the triangular edges of M.

During the path searching, for any triangular node $v_s \in \mathrm{P}$, all its adjacent nodes are candidates for forming the new part of the path. We choose the node v_j, whose descent function $f_d(v_s, v_j)$ has the maximum value among the candidates. The definition of $f_d(v_s, v_j)$ is

$$f_d(v_s, v_j) = \frac{w_{v_s} - w_{v_j}}{\left\| v_s v_j \right\|} \tag{1}$$

Among the three nodes of the triangle containing p_a, we choose the one - v_s with smallest weight factor to start the path searching. The major part of P is formed incrementally by adding the nodes with the maximum $f_d(v_s, v_j)$ one by one. The path P stops at the triangle containing p_s. After linking P with p_a and p_s, the approximate shortest path between p_a and p_s is finally determined. For example, in Fig.3, the circled nodes are p_a and p_s, and the bolded edges are the determined path. The pseudo-codes of the path generation algorithm are given in Table 4 as ***Algorithm*** PathGeneration ($p_a, G_M(p_s)$). In the worst case (e.g., the given mesh is a triangle strip with a band shape), every node on the given mesh surface M visits its adjacent node once; so the time cost of ***Algorithm*** PathGeneration ($p_a, G_M(p_s)$) is $O(E)$. In summary, we can generate the approximate shortest path between two leading points on M in linear time.

The initial measurement curve through all leading points is formed by linking the approximate shortest paths between the adjacent leading points in L_0. In section 4, we will describe an iterative algorithm to simulate stretching the measurement curve on the surface of M .

Table 4 Pseudo-code of *Algorithm* PathGeneration ($p_a, G_M(p_s)$)

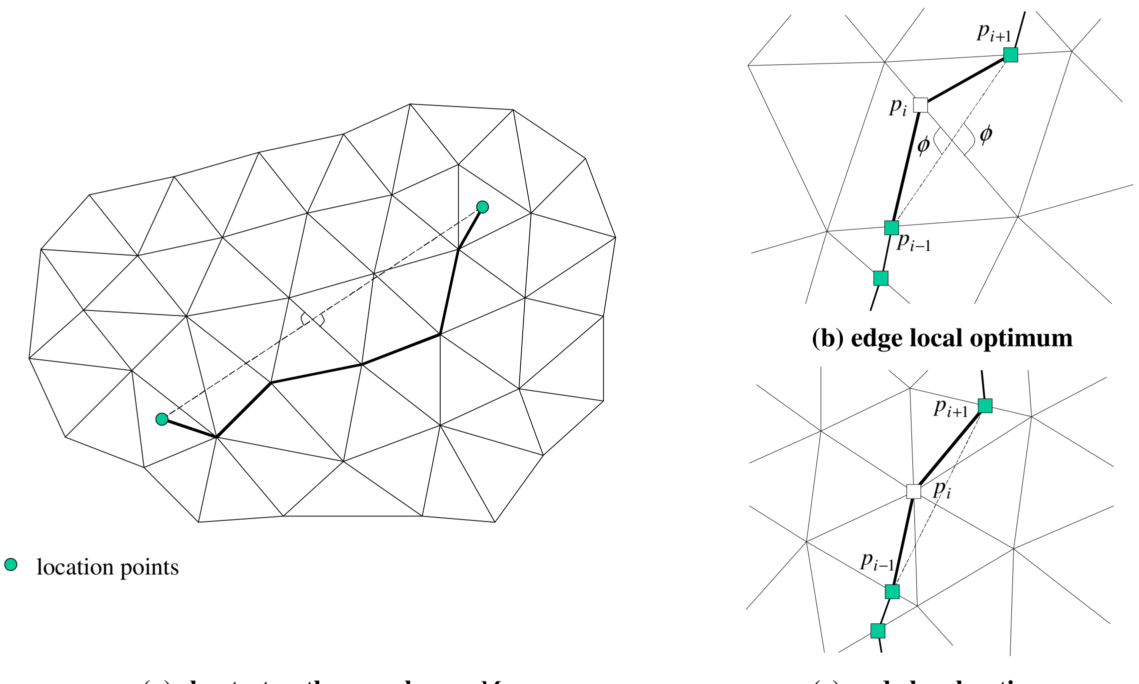
```
Algorithm PathGeneration ( $p_a, G_M(p_s)$ )
Input: Geodesic distance map $G_M(p_s)$ and a leading point $p_a$.
Output: The initial measurement curve $\mathrm{P}$.
1.  $W_{\min} \leftarrow +\infty$;
2.  for every node $v_j$ in the triangle containing $p_a$
3.        if $w_{v_j} < W_{\min}$, then $v_s \leftarrow v_j$ and $W_{\min} \leftarrow w_{v_j}$;
4.  Add $p_a v_s$ into $\mathrm{P}$;
5.  while( $v_s \notin$ the triangle containing $p_s$ ) {
6.        $v_{\max} \leftarrow$ any node adjacent to $v_s$;
7.        for every node $v_j$ adjacent to $v_s$
8.              if ( $f_d(v_s, v_j) > f_d(v_s, v_{\max})$ ), then $v_{\max} \leftarrow v_j$;
9.        Add the edge $v_s v_{\max}$ into $\mathrm{P}$;
10.       $v_s \leftarrow v_{\max}$;
11. }
12. Add $v_s p_s$ into $\mathrm{P}$;
13. return $\mathrm{P}$;
```

4. Stretching Simulation

This section presents an iteration algorithm, which simulates the stretching activity of a tapeline on the surface of a given object. The basic idea of the stretching simulation algorithm is to make the measurement curve locally shortest at every passed triangular edge. Based on this, two local operators are derived in section 4.2. During the iteration, the redundant points should be eliminated before the next iteration step; otherwise, the measurement curve will stick at the local optimum. The elimination of two types redundant points is described in section 4.3. At last, the pseudo-code of the iteration algorithm is given.

4.1. Basic idea

Let us first assume that the given polyhedral surface M is planar (as shown in Fig.4); if we stretch the initial measurement curve as the location points, the final optimum will be a straight line linking the two location points (Fig.4a). Now, considering only the part around an intersection point of a triangular edge and the final optimized measurement curve, the optimized curve maintains a constant angle with the triangular edge (Fig.4b), which leads the curve to be a straight line on a planar surface – we call it the *constant angle condition*. When every node on the measurement curve satisfies the constant angle condition, the measurement curve is a straight line between the two location points – the shortest curve between them. If M is non-planar, two adjacent faces of a triangular edge on a three-dimensional polyhedral surface can be flattened into a plane by rotating around the edge. The mapping between the faces before and after flattening is isometric; when moving a node of the measurement curve along the triangular edge, the local optimum position is still the position that makes the measurement curve and the triangular edge have a constant angle. Using an iteration procedure, we can achieve a global shortest curve on the given polyhedral surface when every node on the measurement curve satisfies the constant angle condition. When a vertex p_i of the measurement curve is on a triangular node of M (Fig.4c), if $\angle p_{i-1} p_i p_{i+1} \neq \angle p_{i+1} p_i p_{i-1}$, moving p_i to the position that makes $\angle p_{i-1} p_i p_{i+1} = \angle p_{i+1} p_i p_{i-1}$ will achieve the local optimum. In the following section, the detail operators for getting local optimum are given.



(a) shortest path on a planar M **(b) edge local optimum** **(c) node local optimum**

Fig.4 Basic idea illustration

4.2. Local operators

There are two local operators adopted in our approach: the *edge operator* and the *node operator*. After linking the leading points by approximate shortest paths, the initial measurement curve is stored in an ATTRIB_EDGE – L_0 as a list of points. Every leading point is stored as an ATTRIB_FACENODE, every approximate shortest path is firstly converted into a list of triangular nodes, and all triangular nodes are further converted into ATTRIB_EDGENODEs, where each ATTRIB_EDGENODE is attached to one of the triangular node's adjacent edges. The parameter u of every ATTRIB_EDGENODE is either zero or one (In detail, if the triangular node is the start node in its adjacent edge, we set $u = 0$; if it is the end node, set $u = 1$). After that, all ATTRIB_FACENODEs except the two location points are removed from L_0. In our iterative stretching algorithm, either of the two local optimum operators is applied to every internal node in L_0 to achieve the shortest measurement curve on M.

Edge operator

For an internal node $p_i \in L_0$, if its parameter $u \in (0,1)$, the following edge operator is applied on it to obtain the local optimum. As illustrated in Fig.5, if p_i is on the edge $v_{st}v_{ed}$, the edge operator adjusts its parameter u to make the length of $\|p_{i-1}p_i\| + \|p_i p_{i+1}\|$ shortest. First, we map the points p_{i-1}, p_i, p_{i+1}, v_{ed} into an *x*-*y* plane maintaining the angle ϕ_1 and ϕ_2 not changed, where v'_{ed} is on the positive *x*-axis, $\|v'_{ed}o\| = \|v_{ed}v_{st}\|$, $\|p'_i o\| = \|p_i v_{st}\|$, $\|p'_{i+1}p'_i\| = \|p_{i+1}p_i\|$, and $\|p'_i p'_{i-1}\| = \|p_i p_{i-1}\|$ (see the right part of Fig.5). Then, the intersection point p^*_i of the line $p'_{i+1}p'_{i-1}$ and the *x*-axis is computed on the *x*-*y* plane. The new parameter of p_i is defined by

$$u^* = \begin{cases} 0, & \textit{if } \dfrac{x[p^*_i]}{x[v'_{ed}]} < 0 \\ 1, & \textit{if } \dfrac{x[p^*_i]}{x[v'_{ed}]} > 1 \\ \dfrac{x[p^*_i]}{x[v'_{ed}]}, & \textit{otherwise} \end{cases} , \qquad (2)$$

where $x[\ldots]$ returns the *x* coordinate of a point. After updating the parameter of p_i by u^*, the edge operator on p_i is completed. In our algorithm, we utilize the symbol $O_e[p_i]$ to represent the edge operator on p_i.

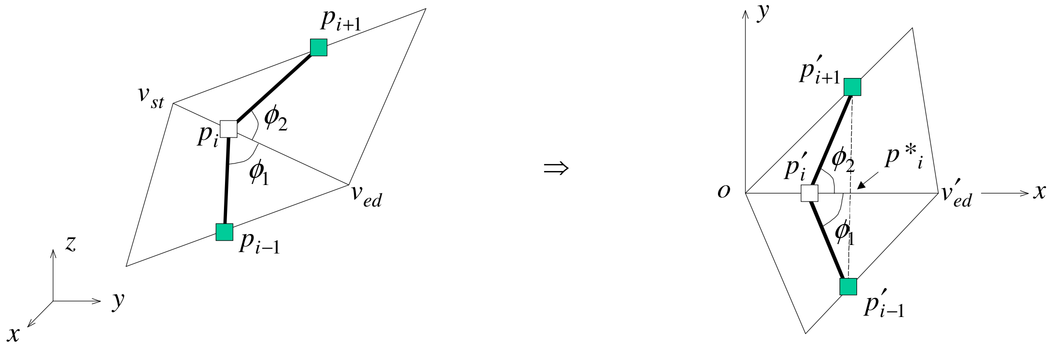


Fig.5 Edge optimize operator

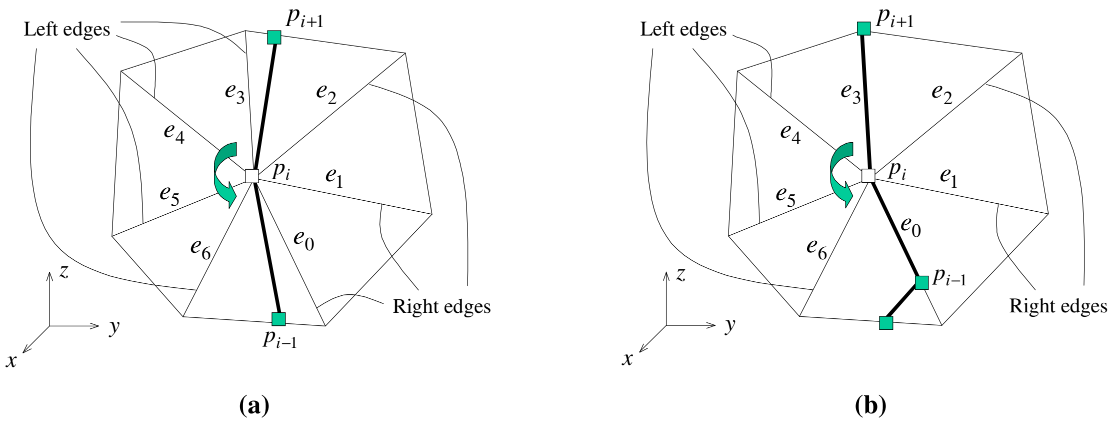


Fig.6 The left edges and the right edges around a triangular node

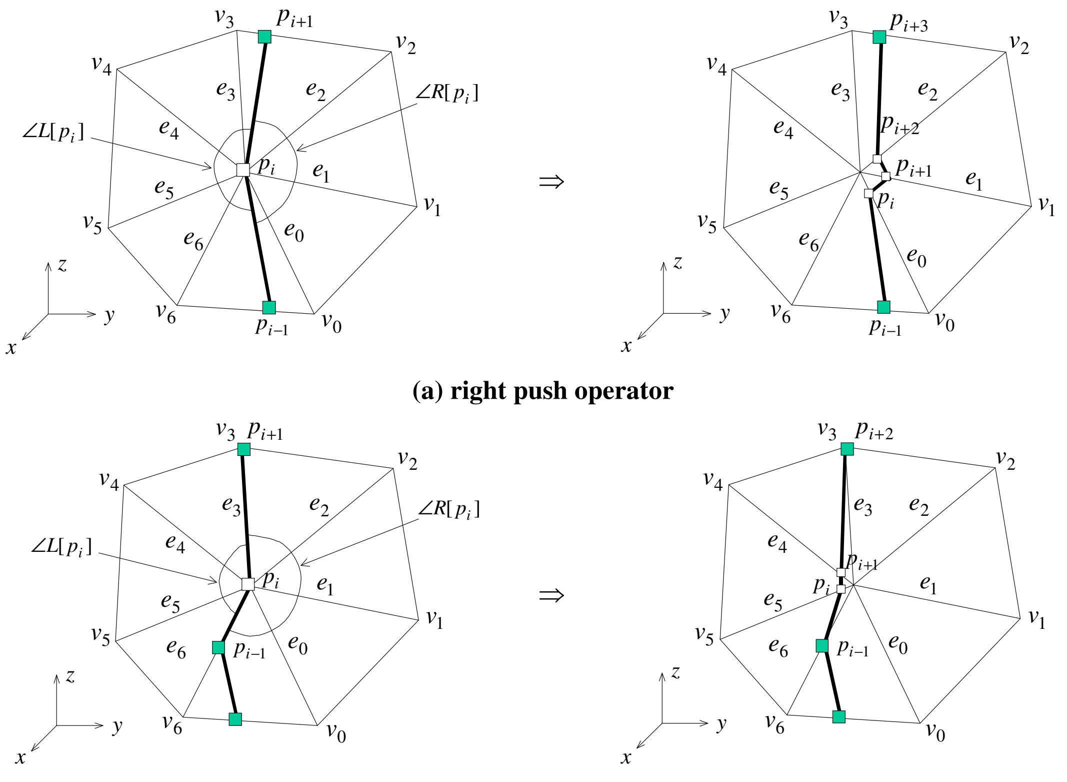


Fig.7 Node operators

Node operators

If the parameter u of an internal node $p_i \in L_0$ does not belong to $(0,1)$, p_i is coincident with a triangle node v_i; we conduct the following node operator to adjust the points in L_0. First, the edges adjacent to v_i are sorted in an anti-clockwise order around v_i; so the edges between $p_{i-1}p_i$ and $p_i p_{i+1}$ in the anti-clockwise direction are defined as the *left edges* of v_i, and the edges between $p_i p_{i+1}$ and $p_{i-1}p_i$ are defined as the *right edges* of v_i (see Fig.6a). If a triangular edge coincides with the line $p_{i-1}p_i$ or the line $p_i p_{i+1}$, it is neither a left edge nor a right edge of v_i (e.g., e_0 and e_3 in Fig. 6b). We can replace the node p_i in L_0 either by a list of points on the right edges of v_i (Fig.7a) – which is called *right push operator*, or by a list of points on the left edges of v_i (Fig. 7b) – called *left push operator*. In both the push operators, the positions of the new inserted points are very close to v_i, where the distance to v_i is usually set to $\frac{1}{100}$ of the edge length in our implementation. The order of the inserted new edge nodes is according to the order of the related triangular edge around v_i.

The measurement curve $p_{i-1} - p_i - p_{i+1}$ separates the tessellation at v_i into two parts: left and right (the curve direction pointing from p_i to p_{i+1}); the total angle in the left part is represented by $\angle L[p_i]$, and the total angle in the right part is represented by $\angle R[p_i]$. For example, in Fig.7a,

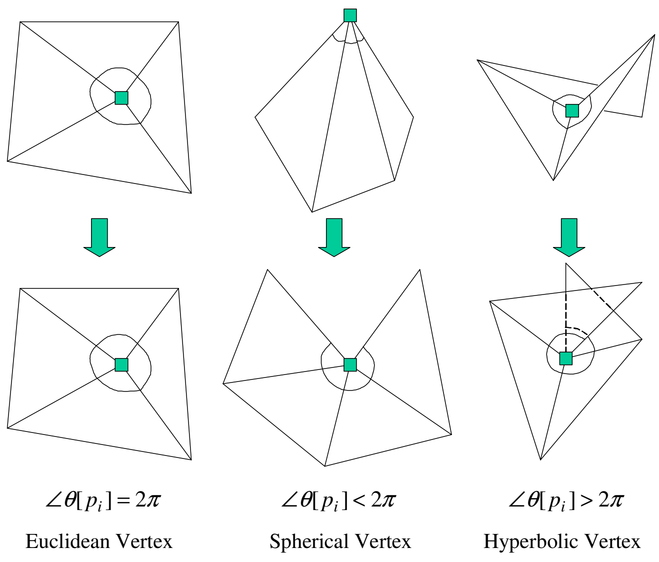
$$\angle L[p_i] = \angle p_{i+1} p_i v_3 + \angle v_3 p_i v_4 + \angle v_4 p_i v_5 + \angle v_5 p_i v_6 + \angle v_6 p_i p_{i-1}$$

and

$$\angle R[p_i] = \angle p_{i-1} p_i v_0 + \angle v_0 p_i v_1 + \angle v_1 p_i v_2 + \angle v_2 p_i p_{i+1}.$$

The sum of $\angle L[p_i]$ and $\angle R[p_i]$ is represented by $\angle \theta[p_i]$. By the values of $\angle L[p_i]$, $\angle R[p_i]$, and $\angle \theta[p_i]$, we choose either left push operator or right push operator to process the node p_i in L_0.

The triangles adjacent to a vertex can be isometrically unfolded to an Euclidean plane – partial or multiple area around the vertex on the plane is covered by the unfolded triangles. There are three situations as shown in Fig.8, which metrically characterize the vertex into a *Spherical vertex*, a *Euclidean vertex*, or a *Hyperbolic vertex* [19]. For example, the tip of a convex cone is a spherical vertex and a saddle point is a hyperbolic vertex.



$\angle\theta[p_i] = 2\pi$
Euclidean Vertex
$\angle\theta[p_i] < 2\pi$
Spherical Vertex
$\angle\theta[p_i] > 2\pi$
Hyperbolic Vertex

Fig.8 Classification of vertices on a polyhedral surface

On a Euclidean vertex

If p_i coincides to a Euclidean vertex, on the unfolded adjacent triangles, the local shortest curve between p_{i-1} and p_{i+1} is the straight line between them. Thus, when $\angle L[p_i] > \angle R[p_i]$, we push the point p_i to the right side by applying the right push operator. The right push operator will not immediately make $p_{i-1} - p_i - p_{i+1}$ be a straight line; however, after iteratively applying the edge operators to them, they converge to a straight line. When $\angle L[p_i] < \angle R[p_i]$, pushing the point p_i to the left side by applying the right push operator will give the collinear tendency to $p_{i-1} - p_i - p_{i+1}$ (see the dash lines illustrated in Fig. 9). For the case with $\angle L[p_i] = \angle R[p_i]$, the three points p_{i-1}, p_i, and p_{i+1} have already been collinear – no change is required on p_i.

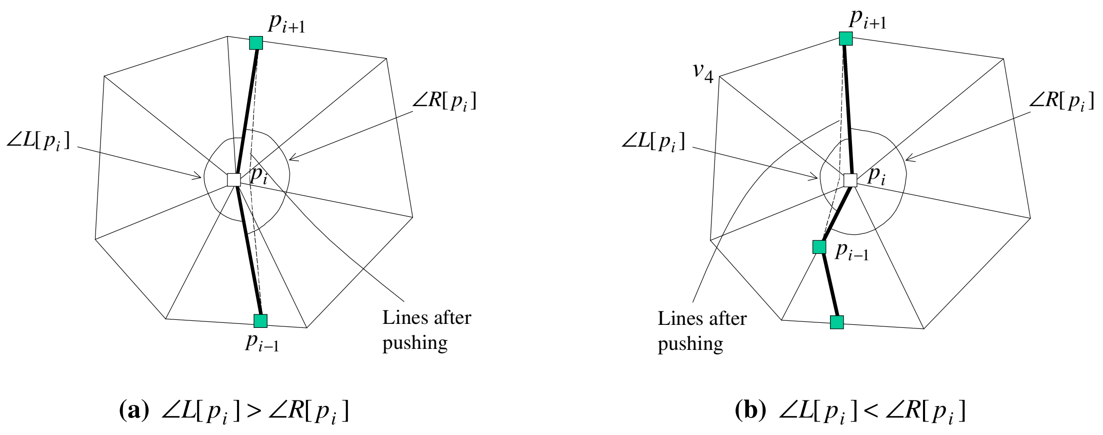
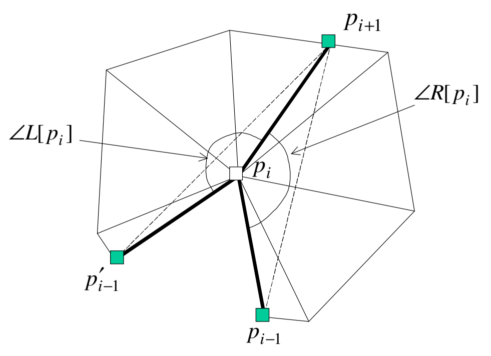
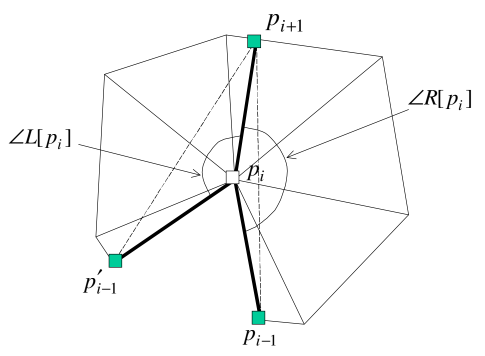


Fig.9 Operator chosen criterion on a Euclidean vertex

On a spherical vertex

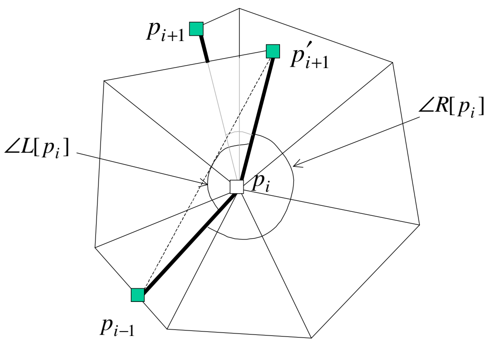
For the situation of p_i on a spherical vertex, we first cut its adjacent faces along the line $p_{i-1}p_i$, and then unfold them onto a Euclidean plane – the unfolding result is generally as shown in Fig.10. The point p_{i-1} linked to the left part triangles after cutting and unfolding is renamed as p'_{i-1}. When $\angle L[p_i] > \angle R[p_i]$, since $\|p_{i-1}p_i\| = \|p'_{i-1}p_i\|$, the distance between p_{i+1} and p_{i-1} is smaller than the distance between p_{i+1} and p'_{i-1} on the plane; when $\angle L[p_i] < \angle R[p_i]$, we have $\|p_{i-1}p_{i+1}\| > \|p'_{i-1}p_{i+1}\|$. Therefore, for the $\angle L[p_i] > \angle R[p_i]$ case, we apply the right push operator on p_i to make it have the tendency of being coincident with the $p_{i+1}p_{i-1}$ line; for the $\angle L[p_i] < \angle R[p_i]$ case, the left push operator is adopted to lead p_i to be collinear with $p_{i+1}p'_{i-1}$. When $\angle L[p_i] = \angle R[p_i]$, the lengths of $\|p_{i-1}p_{i+1}\|$ and $\|p'_{i-1}p_{i+1}\|$ are equal – we just randomly choose either the left push operator or the right push operator on p_i.

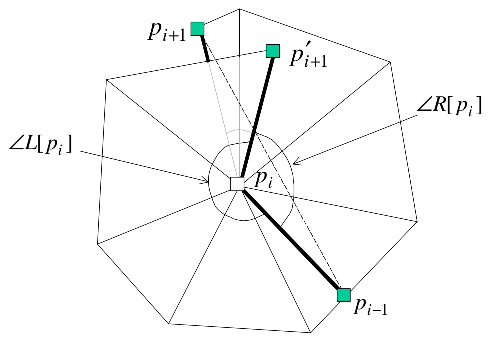




(a) $\angle L[p_i] > \angle R[p_i]$ **(b)** $\angle L[p_i] < \angle R[p_i]$

Fig.10 Operator chosen criterion on a spherical vertex





(a) $\angle L[p_i] < \pi$ **(b)** $\angle R[p_i] < \pi$

Fig.11 Operator chosen criterion on a hyperbolic vertex

On a hyperbolic vertex

When p_i is on a hyperbolic vertex, similar to dealing with the spherical case, we cut its adjacent faces along the line $p_{i+1}p_i$ and unfold them onto a Euclidean plane. The point p_{i+1} linked to the left part triangles after cutting and unfolding is renamed as p'_{i+1}. When $\angle L[p_i] < \pi$, as shown in Fig.11a, moving p_i be collinear with $p_{i-1}p'_{i+1}$ will generate the shortest path from p_{i-1} to p'_{i+1} - the left push operate gives this possibility. When $\angle R[p_i] < \pi$ (see Fig.11b), applying the right push operator will lead the curve $p_{i-1} - p_i - p_{i+1}$ to be shortened. If both $\angle L[p_i] \geq \pi$ and $\angle R[p_i] \geq \pi$ are satisfied, it is difficult to find a mathematical support to choose the left or the right push operators, we just randomly choose one. From experiments, we find that the edge operator will pull the measurement curve back after iterations if the random change makes the curve elongated.

Table 5 Chosen method of push operators

Vertex Type	$\angle L[p_i] > \angle R[p_i]$	$\angle L[p_i] < \angle R[p_i]$	$\angle L[p_i] = \angle R[p_i]$
Euclidean vertex	Right push operator	Left push operator	None
Spherical vertex	Right push operator	Left push operator	Random

Vertex Type	$\angle R[p_i] < \pi$	$\angle L[p_i] < \pi$	**others**
Hyperbolic vertex	Right push operator	Left push operator	Random

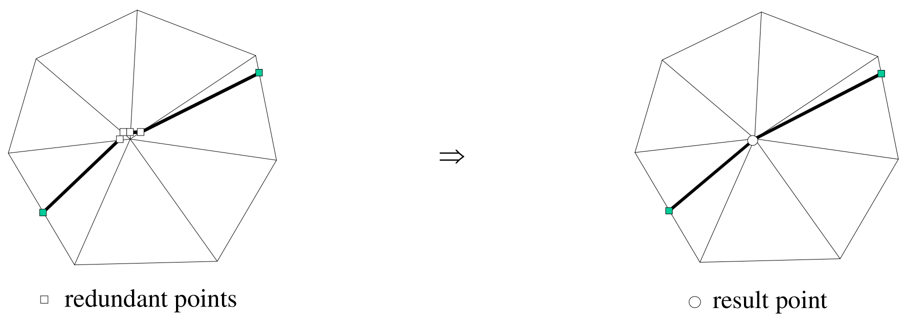
The chosen method of the left and right push operators is summarized in Table 5. This chosen method and the two push operators are generally called a *node operator*, which is represented by the symbol $O_v[p_i]$ in our iterative stretching algorithm.

4.3. Remove redundant points

After repeatedly applying the local operators to the nodes in L_0, redundant points may appear. These redundant points must be removed from the measurement curve efficiently; otherwise, the measurement curve will stick on some points so that the global optimum cannot be reached. There are two types of redundant points: Type I – redundant points around a triangular node, and Type II – redundant points in a triangle.

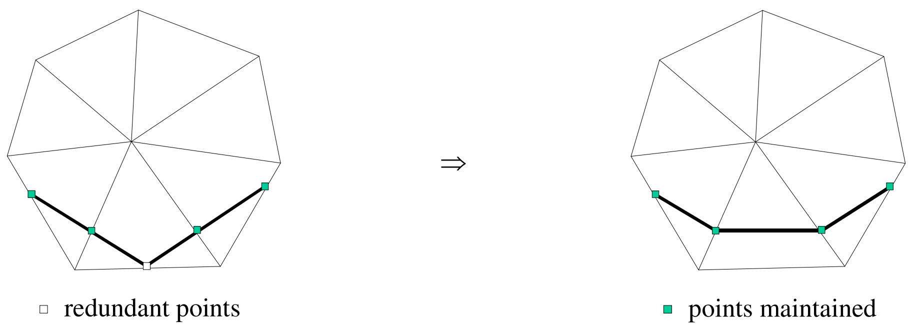
As shown in Fig.12, Type I redundant points are the points that are very close to the endpoints of a triangular edge but not exactly coincident to the point. Thus, even after applying the edge operator to it for many times, it still stick at the triangular node. Fig.14a gives an example with Type I redundant points not eliminated during the iteration. Our solution is that if $u < \varepsilon$, let $u = 0$; if $u > 1 - \varepsilon$, let $u = 1$. ε is a very small threshold number, in our approach, we choose $\varepsilon = 10^{-5}$. Also, for any point $p_i \in L_0$, if p_i and p_{i+1} are coincident, p_i will be removed from L_0.

A Type II redundant point p_i is the point whose two neighboring point p_{i-1} and p_{i+1} in L_0 are in the same triangle (e.g., the one in Fig.13a). If this is the case, p_i should be removed from L_0; otherwise, the measurement curve will stick on the triangular edge of p_i. Fig.14b gives an example that Type II redundant points are not removed.



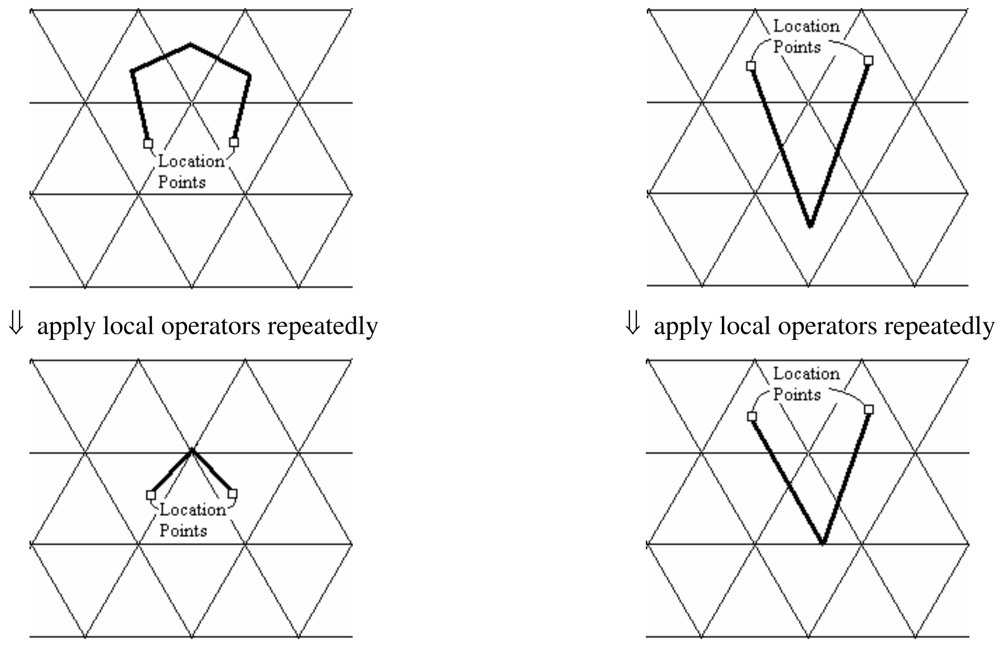
(a) before removing the redundant points **(b) after removing the redundant points**

Fig.12 Remove Type I redundant points



(a) before removing the redundant points **(b) after removing the redundant points**

Fig.13 Remove Type II redundant points


Location Points
Location Points
⇓ apply local operators repeatedly
⇓ apply local operators repeatedly
Location Points
Location Points

(a) Type I redundant points **(b) Type II redundant points**

Fig.14 Examples of the redundant points not removed

4.4. Algorithm summary

To sum up, the iteration algorithm to simulate stretching the measurement curve has two operations in each iteration step: 1) applying local operators on every internal nodes of the measurement curve, and 2) removing redundant points. During the iteration, the length of the measurement curve decreases while the step number of iteration increases. Usually, two criteria are utilized to give the terminal condition: the curve length and the step number. Here, we employ a mixture of them. Either $\|L_j - L_{j-1}\| < \mu$ or the iteration steps is greater than $N_{\max}$, the iteration stops, where L_j is the length of the measurement curve in the *j*th iteration (current value), $N_{\max}$ is the maximum iteration number, and μ is a small number. We usually choose $N_{\max} = 5000$ and $\mu = 10^{-5}$ in our testing examples. The pseudo-code of the iterative stretching algorithm is given in Table 6 as ***Algorithm*** IterativeStretching (L_0).

Table 6 Pseudo-code of *Algorithm* IterativeStretching (L_0)

```
Algorithm IterativeStretching ( $L_0$ )
Input: The initial measurement curve $L_0$.
Output: The stretched measurement curve $L_0$.

1.  Convert all internal nodes of $L_0$ into ATTRIB_EDGENODEs;
2.  $j \leftarrow 1$;
3.  do {
4.      for every internal node $p_i \in L_0$ {
5.          if (the parameter of $p_i$ - $u \in (0, 1)$ )
6.              Apply $O_e[p_i]$ on $p_i$;
7.          else
8.              Apply $O_v[p_i]$ on $p_i$;
9.      }
10.     Remove all Tape I nodes from $L_0$;
11.     Remove all Tape II nodes from $L_0$;
12.     if ( $\|L_j - L_{j-1}\| < \mu$ ), then break;
13.     $j \leftarrow j+1$;
14. }while( $j \le N_{\max}$ );
```

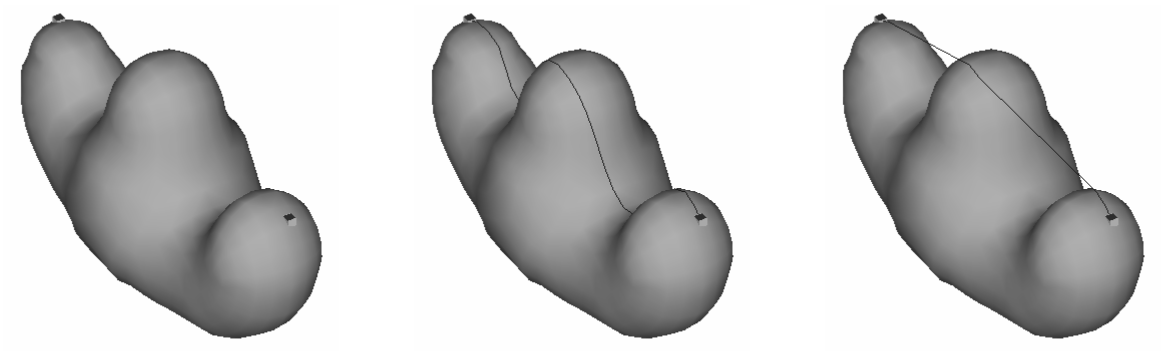
5. Further Improvement

The resultant stretched measurement curve of ***Algorithm*** IterativeStretching (L_0) in the above section lies on the surface of M . Sometimes, in reality, users intend to further stretch the tapeline to let it leave the measured surface. The additional algorithm introduced in this section will simulate this performance of a physical tapeline. For example, to get the tape measurement between the two location points shown in Fig.15a, the result of ***Algorithm*** IterativeStretching (L_0) is as shown in Fig.15b; after further stretching L_0, the result is as in Fig. 15c. The further stretching algorithm is an incremental method, which detects every internal node, $p_i \in L_0$, whether p_i can be removed from L_0. If the line segment $p_{i-1}p_{i+1}$ has any intersection point with M (not including p_{i-1} and p_{i+1}), the point p_i must be maintained in L_0; otherwise, we can remove p_i from L_0. In detail, the algorithm for further stretching L_0 is listed in Table 7.

Table 7 Pseudo-code of *Algorithm* FurtherStretching (L_0)

```
Algorithm FurtherStretching ( $L_0$ )
Input: The measurement curve $L_0$.
Output: The further stretched measurement curve $L_0$.

1.  for every internal node $p_i \in L_0$ {
2.      do {
3.          if the line segment $p_{i-1}p_{i+1}$ has any intersection with $M$ , then break;
4.          Remove $p_i$ from $L_0$;
5.          $i \leftarrow i-1$;
6.      } while( $i > 0$ );
7.  }
8.  return $L_0$;
```

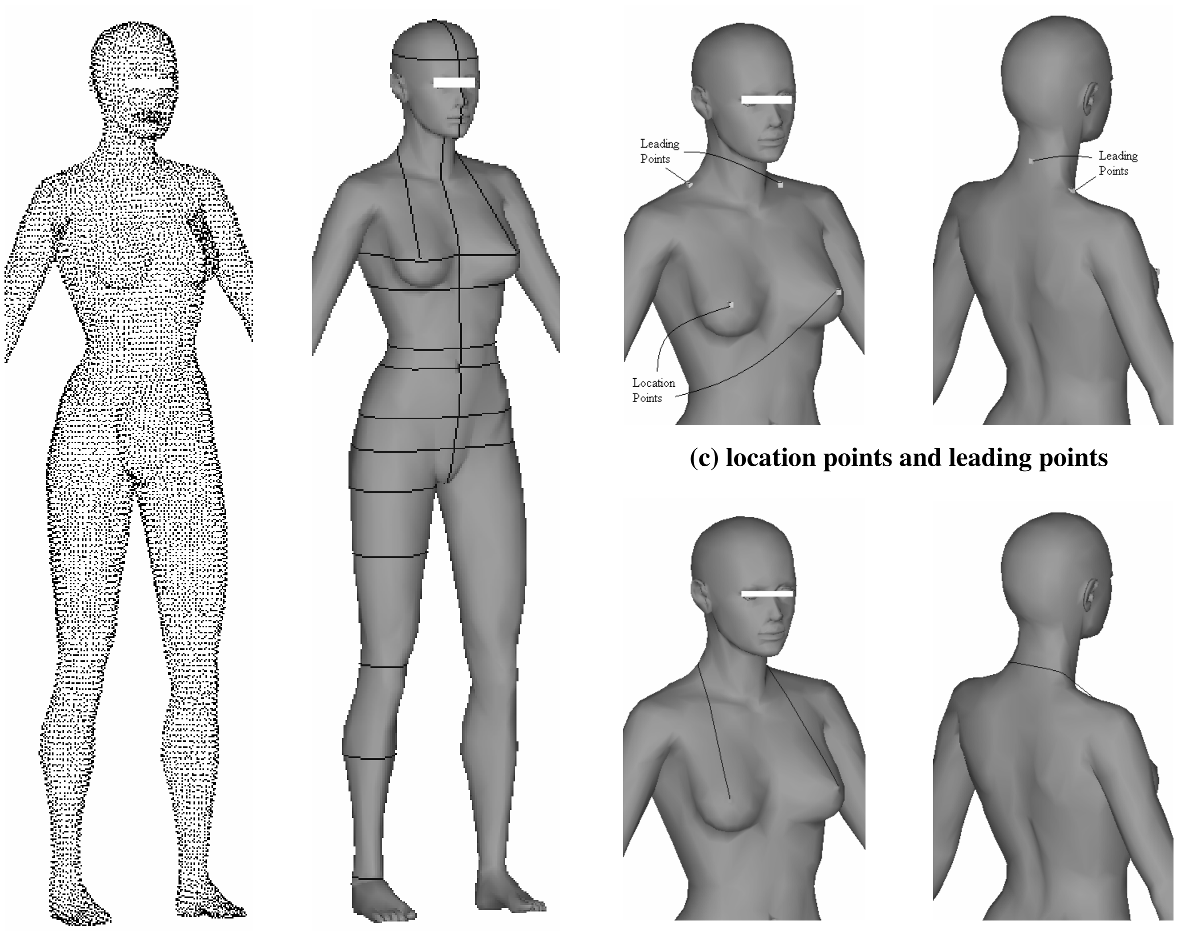


(a) given surface with two location points | **(b) result of *Algorithm*** IterativeStretching (L_0) | **(c) after applying *Algorithm*** FurtherStretching (L_0)

Fig.15 Further stretching the measurement curve

6. Results

The interactive measurement tool presented in this paper can be widely used in industry. One example application is in the appeal industry, where the design and manufacturing comes more and more worldwide. A general case is that the service centers, the design centers, and the manufacturing workshops for customized cloth design and manufacturing are not located in the same region. Thus, the body data of customers should be transferred between them. However, problems occur when the human dimensions measured at the service center is not enough for the operations at the design center. At present, the popular solution is to transfer the 3D full body data but not the dimensions between the centers. The ranged scanner in a service center captures the point cloud for the body shape of a customer; after that, the point cloud is converted into a polyhedral surface to reduce the data size while maintaining the accuracy [20]; the result polyhedral surface is finally transferred from the service center to the design center for customized design and manufacturing. For example, Fig.16a gives an example of a point cloud captured in a service center, and the transferred polyhedral surface is shown in Fig.16b. In the design center, various dimensions are measured on the virtual human body by the requirements of different designers (example measurement curves are shown in Fig.16b). The tool presented in this paper will be a most important measurement tool in the design center, which simulates the behavior of a tapeline in cyber space. For example, one key dimension for women garments is the apex-to-apex measurement around neck. As shown in Fig.16c and 16d, after interactively specifying the location points at the busty points and giving the leading points around the neck, the approach presented in this paper determines the stretched measurement curve, which passes through the two location points.



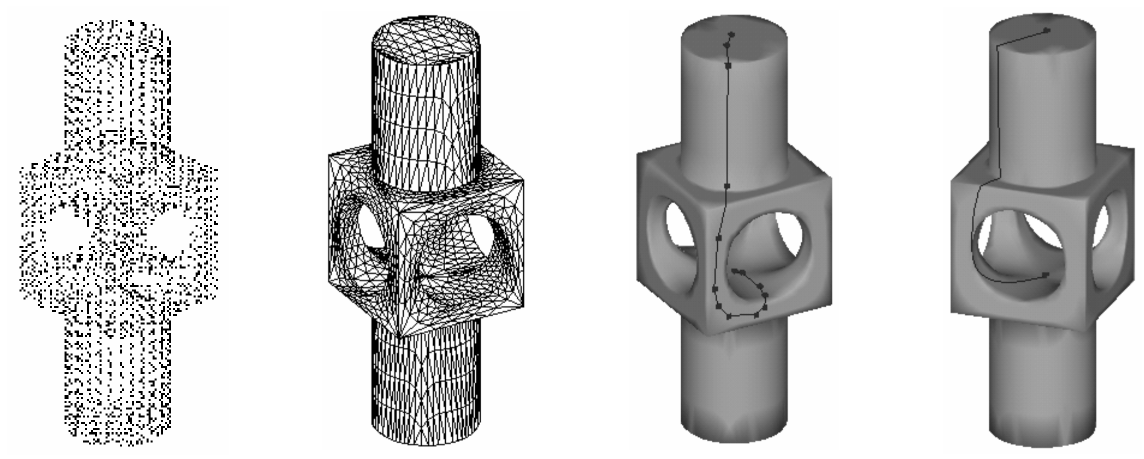
(c) location points and leading points

(a) point cloud **(b) polyhedral surface** **(d) result measurement curve**

Fig.16 Application in appeal industry

Fig.17 gives another example application of CyberTape in the reverse engineering of a mechanical part. For example, after the point cloud of a mechanical part is obtained from CMMs, the related polyhedral surface is constructed by the algorithm in [2]; then, our CyberTape tool can be applied to determine the measurement curves, which helps us give manufacturing parameters before making physical prototypes.

The computation statistics of the examples are listed in Table 8. From the statistics, it is not hard to find that the computation of CyberTape can be finished in real time on a dense polyhedral surface using a standard desktop PC.



(a) point cloud **(b) polyhedral surface** **(c) location points and leading points** **(d) result measurement curve**

Fig.17 Application in the reverse engineering of mechanical part

Table 8 Computation statistics of the examples

Example	Figure	Length of L_0 (unit: CM)		Tessellation		Iteration steps number	Computing time
		Before stretching	After stretching	Node number	Triangle number		
I	11	2.75	1.96	667	1330	246	< 1 sec.
II	12	67.33	60.74	3774	7508	806	1 sec.
III	13	9.21	7.84	1900	3808	135	< 1 sec.

* All with $N_{\max} = 5000$ and $\mu = 10^{-5}$ on a PIII 500 PC with a program written in C++.

7. Conclusion and Discussion

This paper presents an interactive virtual measurement tool – CyberTape on a polyhedral surface in the manner that dragging a tapeline between two location points, which is intuitive and efficient to measure a polyhedral surface in virtual space. Our tool allows not only location points but also leading points to be specified on the given surface, where the leading points give the expected direction of the measurement curve. Compared to existing shortest path techniques, our method is more flexible to the inclination of users. The process consists of three steps: 1) generating the approximate shortest paths through the leading points as an initial measurement curve; 2) applying local operators on every internal points to approximate a stretched curve on the given surface; and 3) further stretching the measurement curve to leave the measured surface in concave places (this is an optional step). Our implementation algorithm of the CyberTape tool can be completed in real time on a standard PC.

The current implementation of CyberTape is robust and efficient enough for experimental use. However, to enhance its functionality, the following problem needs to be solved. Since ***Algorithm*** FurtherStretching (L_0) performances as a post-processing step in our current implementation, the result of ***Algorithm*** FurtherStretching (L_0) may not be optimal enough. For example, as shown in Fig.18, the measurement curve is stuck at the bellybutton of a human body since there is a little cavity. If ***Algorithm*** FurtherStretching (L_0) and ***Algorithm*** IterativeStretching (L_0) are integrated together, the measurement curve can easily slip over the bellybutton. Simply putting them together will decrease the efficiency of our approach; further research can focus on how to mix the two algorithms into a single algorithm while maintaining the efficiency.

(a) location points and leading points on an initial measurement curve **(b) local optimum – bellybutton sticks the measurement curve**

Fig.18 The measurement curve is stuck on the bellybutton

References

[1] Hoffman R, Jain A. Segmentation and classification of range images, IEEE transactions on pattern analysis and machine intelligence, vol.9, no.5, pp.608-620, 1987.

[2] Hoppe H, DeRose T, Duchamp T, Halstead M, Jin H, McDonald J, Schweitzer J, Stuetzle W. Piecewise smooth surface reconstruction, SIGGRAPH 1994 Conference Proceedings, pp.295-302, 1994. ACM., New York, NY, USA.

[3] Várady T, Martin RR, Cox J. Reverse engineering of geometric models – an introduction, Computer-Aided Design, vol.29, no.4, pp.255-268, 1997.

[4] Barhak J, Fischer A. Parameterization for reconstruction of 3D freeform objects from laser-scanned data based on a PDE method, The Visual Computer, vol.17, no.6, pp.353-369, 2001.

[5] Allen B, Curless B, Popović Z. The space of human body shapes: reconstruction and parameterization from range scans, SIGGRAPH 2003 Conference Proceedings.

[6] Bielser D, Volker A, Gross M. Interactive cuts through 3-dimensional soft tissue, Computer Graphics Forum, vol.18, no.3, pp.31-38, 1999.

[7] Bruyns CD, Senger S. Interactive cutting of 3D surface meshes, Computers & Graphics, vol.25, no.4, pp.635-642, 2001.

[8] Suzuki H, Sakurai Y, Kanai T, Kimura F. Interactive mesh dragging with an adaptive remeshing technique, The Visual Computer, vol.16, no.3-4, pp.159-76, 2000.

[9] Mitchell JSB, Mount DM, Papadimitriou CH. The discrete geodesic problem, SIAM Journal on Computing, vol.16, no.4, pp.647-668, 1987.

[10] Chen J, Han Y. Shortest paths on a polyhedron. Proceedings 6th ACM Symposium on Computational Geometry, pp.360-369, 1990.

[11] Lanthier MA, Maheshwari A, Sack JR. Approximating weighted shortest paths on polyhedral surfaces. Proceeding 13th ACM Symposium on Computational Geometry, pp.274-283, 1997.

[12] Mata CS, Mitchell JSB. A new algorithm for computing shortest paths in weighted planar subdivisions. Proceeding 13th ACM Symposium on Computational Geometry, pp.265-273, 1997.

[13] Kanai T, Suzuki H. Approximate shortest path on a polyhedral surface and its application, Computer-Aided Design, vol.33, no.11, pp.801-811, 2001.

[14] Cormen TH, Leiserson CE, Rivest RL. Introduction to Algorithms, New York: McGraw-Hill, 1990.

[15] Thorup M. Undirected single source shortest paths in linear time, Proceeding 13th ACM Symposium on Computational Geometry, pp.12-21, 1997.

[16] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Mesh optimization, SIGGRAPH 93 Proceeding, ACM., 1993, pp. 19-26, New York, USA.

[17] Ganapathy S, Dennehy TG. A new general triangulation method for planar contours, Computer Graphics, vol.16, no.3, pp.69-75, 1982.

[18] Chapra SC, Canale RP. Numerical methods for engineers: with software and programming applications (4th ed), Boston: McGraw-Hill, 2002.

[19] Polthier K, Schmies M. Straightest Geodesics on Polyhedral Surfaces, in: Mathematical Visualization, Ed: H.C. Hege, K. Polthier, Springer Verlag, Berlin, 1998.

[20] Wang CCL, Chang TKK, Yuen MMF. From laser-scanned data to feature human model: a system based on fuzzy logic concept, Computer-Aided Design, vol.35, no.3, pp.241-253, 2003.