Skeleton-Based Seam Computation for Triangulated Surface Parameterization

Xu-Ping Zhu[1], Shi-Min Hu[1], and Ralph Martin[2]

[1] Department of Computer Science and Technology, Tsinghua University
Beijing 100084, P. R. China
zhuxp@cg.cs.tsinghua.edu.cn, shimin@tsinghua.edu.cn
[2] Department of Computer Science, Cardiff University, Cardiff, CF24 3XF, UK
ralph@cs.cf.ac.uk

Abstract. Mesh parameterization is a key problem in digital geometry processing. By cutting a surface along a set of edges (a seam), one can map an arbitrary topology surface mesh to a single chart. Unfortunately, high distortion occurs when protrusions of the surface (such as fingers of a hand and horses' legs) are flattened into a plane. This paper presents a novel skeleton-based algorithm for computing a seam on a triangulated surface. The seam produced is a full component Steiner tree in a graph constructed from the original mesh. By generating the seam so that all extremal vertices are leaves of the seam, we can obtain good parametrization with low distortion.

1 Introduction

Due to their flexibility and efficiency, triangle meshes have been widely used in the entertainment industry to represent arbitrary surfaces for 3D games and movies during the last few years. Many new mesh techniques have been developed for different applications. In these techniques, parameterization is a key issue—surface parameterization is always an important problem in computer graphics. In this paper, we focus on the parameterization of triangle meshes. In particular, we wish to establish a mapping between a triangulated surface and a given domain.

Parameterization methods can be grouped into three categories, depending on whether the domain is a polyhedron, sphere or plane. Eck et al [6] first used a polyhedron as the parameter domain for a mesh. They called the domain polyhedron a *base-mesh*. A mesh parameterized using the base-mesh is called a semi-regular mesh; such an approach has been widely applied in multi-resolution modelling [14], remeshing [2] and morphing [13]. Recently, Khodakovsky et al [12] presented a globally smooth parameterization algorithm for semi-regular meshes. In contrast with polyhedral parameterization, there is little work [9,17] on spherical parameterization because of its limited application. Finally, we focus on planar parameterization. As is done for parametric surfaces, we want to map the mesh to a bounded region of the plane. This is trivial if the mesh surface is homeomorphic to a disc. Unfortunately, if the topology of the surface is complex, with

M.J. Wilson and R.R. Martin (Eds.): Mathematics of Surfaces 2003, LNCS 2768, pp. 1–13, 2003.

high genus, the surface is usually separated into several disc-like patches, and each patch mapped to a chart. This multi-chart method is often used in texture mapping [15]. Obviously, the more charts into which a surface is separated, the lower the distortion is, and in the extreme, if each triangle of the mesh has its own chart, the distortion is zero. Conversely, using fewer charts causes higher distortion. Hence, there exists a trade-off between the number of charts and the amount of distortion.

Our goal is to map an arbitrary topology surface mesh to a single chart with low distortion. Theoretically, any closed surface can be opened into a topological disc using a set of cut edges making up a *seam*. Cutting along the seam, we can get a disc-like patch. Unless the surface is developable, parameterization using a single chart inevitably creates distortion, which is especially great where protrusions of the surface (such as fingers of a hand and horses' legs) are flattened into the plane. Sheffer [20] and Gu et al [10] have found that to reduce the distortion, it is important for the seam to pass through the various so-called "extrema". Although extrema can be found accurately with their methods, it is still difficult to guide the seam through these extrema. In fact, it is a tough job even for human beings to choose an appropriate seam. Piponi et al [16] describe a system that allows the user to define a seam interactively. In this paper, we consider this problem as a minimal Steiner tree problem and present a novel skeleton-based method to approximate the minimal Steiner tree.

2 Related Work

Our goal is to map an arbitrary 2-manifold surface to a planar domain, so high genus surfaces must be cut by a set of edges or a seam. The problem of minimizing the length of the set of cut edges is known to be NP-hard [7]. Gu et al [10] present a method that approximates such a seam in $O(n \log n)$ time. Once a surface has been cut into a disc, additional cuts are usually necessary to reduce distortion. Choosing them includes two processes: firstly, all extremal vertices on the mesh must be detected; secondly, a path (always a tree) connecting all extremal vertices and the surface boundary must be found. Sheffer [20] detects extrema by searching for vertices with high curvature. Unfortunately, this method only solves problems associated with local protrusions. Gu et al give a more suitable algorithm for finding extrema which utilizes the shape-preserving feature of Floater's parameterization [8]. For each protrusion, he identifies the triangle with maximum stretch in the parametric domain, and picks one of its vertices as a new extremal vertex to augment the seam.

In this paper, we assume all extremal vertices are already given and concentrate on determining the layout of the seam. We pose the problem as a network (weighted graph) *Steiner tree* problem. A Steiner tree for a set of vertices (terminals) in a graph is a connected sub-graph spanning all terminals. Here terminals include all extremal vertices, and the boundary (the whole boundary is regarded as a terminal, but only one point on the boundary is required to be connected to the seam). Naturally, we want to minimize the length of the seam because it will eventually be added to the boundary of the parameterization. The problem

of finding the minimal Steiner tree in a graph is NP-complete [11]. Sheffer [20] approximates it by the minimum spanning tree (MST). The Euclidean MST is never more than $2/\sqrt{3}$ times the length of the Euclidean Steiner minimal tree; but in a graph the corresponding worst-case approximation ratio is 2, not $2/\sqrt{3}$ [11]. Gu et al use a greedy approximation method: when a new extremal vertex is detected, the shortest path between the current seam and the new extremal vertex is added to the seam. This incremental method depends heavily on the sequence of adding extremal vertices; choice of the sequence is not discussed in his paper. Consider the model of a horse in Fig. 1. The most obvious extrema are located at the ends of the four legs and the head, and can be detected by Gu's method. The seam computed using the MST method is poor (See Fig. 1(a)), as the MST passes through most extrema more than once. However, extremal vertices are always at the ends of protrusions, and so it is reasonable to require that the seam should pass through each extremal vertex as few times as possible. It will be best if all extremal vertices are leaves on the seam.

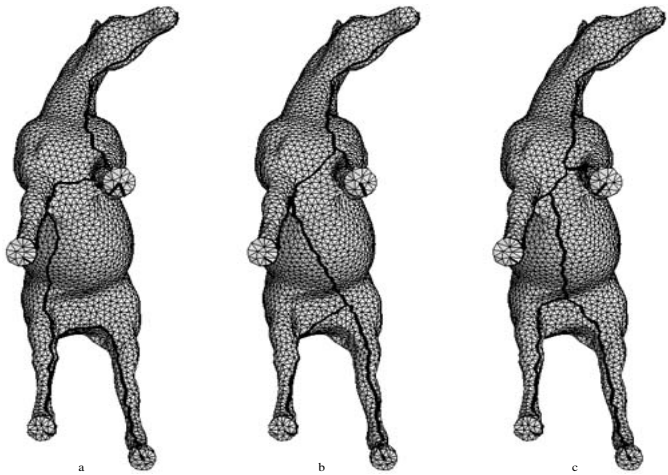

Fig. 1. Seam computed by: (a) Sheffer's method (seam length = 7.99), (b) Gu's method (seam length = 7.07), (c) our method (seam length = 5.97).

For this reason, we constrain the Steiner tree to be a *full component* of the mesh. (A full component is a subtree in which each terminal is a leaf). Robins [18] gives a method for constructing an approximation to a minimum Steiner tree in a weighted graph, with length approaching $1 + (\ln 3)/2 \approx 1.55$ times the minimal length. However, his method is not efficient if the Steiner tree is constrained to be a full component. Thus, in this paper we suggest a new method to compute an approximation to the minimal full component Steiner tree, deriving it from the *straight skeleton*. It produces good results in practice: comparisons with previous methods are given in Fig. 1.

The rest of this paper is organized as follows. In Section 3, we introduce the definition of straight skeleton. Our novel seam computation algorithm is presented in Section 4. Section 5 shows some experimental results of applying our new method to different models. Conclusions and future work are given in Section 6.

3 The Straight Skeleton

The *straight skeleton* of a planar straight-edged graph G is obtained as the interference pattern of certain wavefronts propagated from the edges of G [1]. If G is a simple polygon, the part of the straight skeleton interior to G is a tree whose leaves are the vertices of G (see Fig. 2). If all vertices of the polygon are terminals, the interior straight skeleton must be a full component Steiner tree. Thus, the main idea of our method is to extend the concept of straight skeleton from the plane to a 3D triangle mesh. To do this, we must solve two problems. First, a shortest tour that visits all terminals is found. (Of course, one can also use other heuristics—see the discussion in Section 6). Secondly, the tour is shrunk to produce the skeleton which is a full component Steiner tree of terminals.

In the plane, the straight skeleton is found by shrinking a polygon via inward, uniform speed parallel movement of all edges, and finding where they intersect. On a triangle mesh, we analogously treat the part of the tour between two neighbouring terminals as an "edge" and shrink all "edges" inward in a uniform-speed way, as explained in Section 4.3.

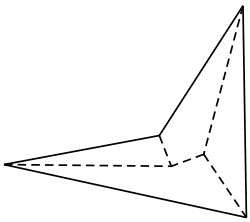

Fig. 2. The straight skeleton interior to a simple polygon.

4 Seam Computation

Before going into details of our algorithm, we now restate the problem. Given an arbitrary 2-manifold triangulated surface (closed or open) and several extremal vertices on it, we seek a minimum length seam connecting all extremal vertices which opens the surface into a disc. In practice, better results are obtained if all extremal vertices are leaves on the seam, so in principle we wish to compute a minimal full component Steiner tree. Because this is expensive, we approximate it using our new skeleton-based algorithm, which is explained in the following section.

4.1 Overview of the Algorithm

We carry out the following preprocessing before computing the seam:

- For surfaces of genus greater than zero, we use Gu's algorithm to compute an initial seam. Cutting along the initial seam, the surface is opened into a disc. After this process, the surface is either a closed genus-0 surface or a disc-like patch with one boundary loop.
- We find the shortest path between each pair of terminals, i.e. the shortest path between each pair of extremal vertices and the shortest paths between each extremal vertex and the boundary loop (if any). To compute the shortest paths, a modified Dijkstra's algorithm is used, as in [20].
- Our algorithm assumes that there are at least two terminals. If there are only two terminals, we just use the shortest path between them as the seam.

The Algorithm for computing the seam can be divided into three steps. Fig. 3 shows the results of each step.

- Find the shortest tour that visits all terminals.
- Shrink the tour to a skeleton.
- Straighten the skeleton.

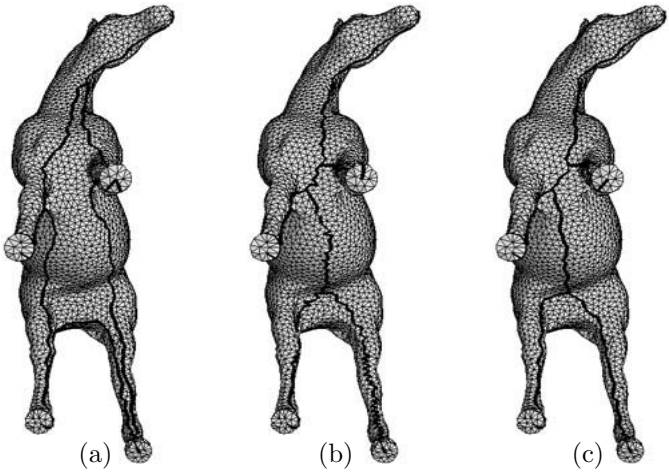

Fig. 3. Results after each step of seam computation.

4.2 Constructing a Tour with MST

For convenience, we use a half-edge data structure to describe the triangle mesh, i.e. we replace each edge with two opposed half-edges; the three half-edges inside a triangle are ordered anticlockwise (see Fig. 4(a)). With this data structure, the mesh can be represented by a directed weighted graph G where the weight is the length of the path between terminals (see Fig. 4(b)). A directed tour must be found in G which satisfies two requirements.

- The tour must visit all terminals exactly once.
- The tour must not self-intersect, i.e. the tour must separate the graph into two regions.

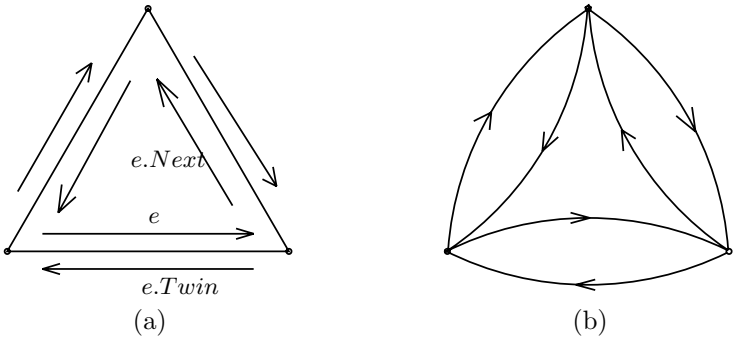

Fig. 4. Half-edge data structure and the corresponding directed weighted graph.

Since the tour is directed, we define the region on the right hand side of the tour to be the interior of the tour. The first requirement is straightforward. Consider the second requirement. Fig. 5 shows two directed tours. They both satisfy the first requirement. We say that the tour on the left-hand side of the Figure is self-intersecting while the tour on the right-hand side is not. In the Figure, nodes marked "○" are terminal nodes, while nodes marked "●" are non-terminals. We may decide whether a tour is self-intersecting by the following algorithm.

Algorithm 1 (Tour Self-intersection Test).

For each half-edge e_i on a directed tour T
{
  Let e_j be the next adjacent edge to e_i in T
  $e_k = e_i.Next$; // Next half-edge in the data structure (see Fig. 4)
  While ($e_k \neq e_j$) // If e_k is not the next edge of the tour
  {
    If e_k is a half-edge in T
      return; // T is self-intersecting
    Else
    {
      $e_k = e_k.Twin$;
      $e_k = e_k.Next$;
      // Move to the next half-edge around the vertex
      // at the end of e_i (see Fig. 6)
    }
  }
}

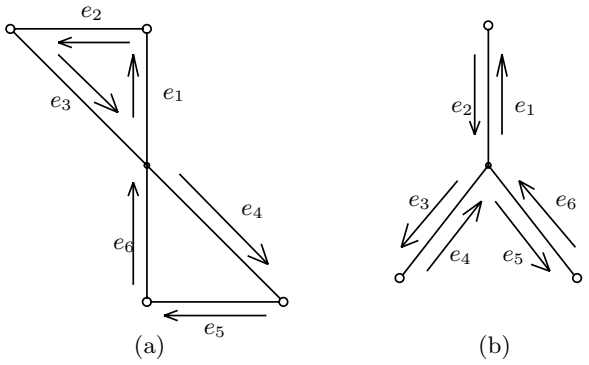

Fig. 5. Self-intersecting and non-self-intersecting tours.

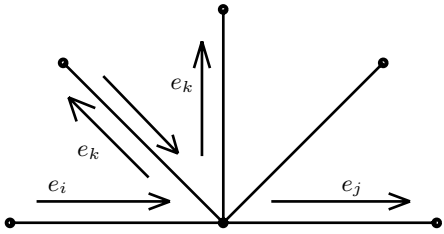

Fig. 6. Tour self-intersection testing.

We wish to find the shortest tour which satisfies the two requirements above. This is similar to the classic traveling salesman problem (TSP) which asks for a shortest tour that visits all nodes of a graph. Here we only restrict the tour to visiting all terminals. Since the part of the shortest tour between two terminals must be the shortest path between them, we can construct a complete graph G_T based on all terminals. The weight of each edge in G_T is the length of the shortest path between each pair of corresponding terminals. Thus we can reduce the problem to a traveling salesman problem in G_T. Unfortunately, TSP is also NP-Hard [11]. Thus we use an approximate method, to be able to handle a large number of terminals. Christofides's algorithm [3] for TSP approximation starts by computing a minimum spanning tree. Using the MST alone gives a tour with length no more than twice the minimal answer. To improve the approximation, Christofides [3] uses minimum-length matching on the odd-degree vertices of the MST. Although this method improves the approximation ratio to 3/2, it can not guarantee the tour is not self-intersecting.

Thus, we use the basic MST method together with a heuristic which not only approximately minimizes the tour, but also avoids self-intersections of the tour. The idea is as follows. Firstly we compute the minimal spanning tree of G_T. If we simply walk around the tree in half-edge order, we can obtain a directed tour which is not self-intersecting in G (See Fig. 6(b)). However, this tour does not satisfy the requirement of visiting each terminal only once, and repeated visits

to terminals must be eliminated from the tour. Starting from this initial tour, we remove repeat visits to terminals one-by-one, until all terminals appear in the tour only once. Let t be a terminal visited more than once in the current tour, $t \to Pre$ be the terminal before t in the tour and $t \to Next$ be the following terminal in the tour. When a visit to t is removed, the part of the tour from $t \to Pre$ to $t \to Next$ is replaced by the shortest path from $t \to Pre$ to $t \to Next$. This operation may produce a self-intersection in the tour. Thus, each time we remove a visit to a terminal, we must check whether the resulting tour is self-intersecting. To decide which terminal to update, a heuristic rule is used. We choose the terminal t, and its Pre and $Next$, with minimal shortest path between $t \to Pre$ and $t \to Next$. We now give the pseudocode of the algorithm as follows; Fig. 3(a) shows the resulting tour produced for the horse model.

Algorithm 2 (Computing a Tour Using MST).

Use Kruskal's algorithm [4] to compute the MST of G_T.
Walk around the tree starting at any terminal.
Record the sequence of terminals in a circular list $TourList$.

```
While (size of TourList > number of terminals)
//Repeated terminals remain in the tour
{
    For each repeated terminal t in TourList
    {
        If the tour is not self-intersecting when the appropriate visit
         to t is removed
            t.priority = Length of the shortest path between
             t → Pre and t → Next
        Else
            t.priority = ∞
    }
    Remove the visit with minimum priority from TourList
}
```

4.3 Shrinking the Tour to a Skeleton

As outlined in Section 3, we treat the part of the tour between two neighbouring terminals as an "edge" and shrink all "edges" inward in a uniform-speed way across the triangulation. The skeleton is the interference pattern of wavefronts propagated from these "edges". To carry out this idea, we mark each "edge" with a different number and propagate these numbers stepwise to the triangulation of the interior of the tour. Places where two different "edge" numbers meet are parts of the skeleton. Pseudocode for the algorithm for doing this follows; q is a queue and all variables are explained in Fig. 7. The principle is to process all half edges adjacent to the tour first by placing them in a queue. Then, as their neighbours are processed, they are also added to the end of the queue, then their neighbours are processed in turn, and so on. Eventually, edges both of whose

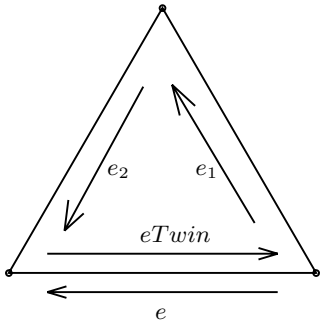

Fig. 7. Mark propagation in skeleton computation.

half-edges have already been marked with different numbers are encountered: these are parts of the skeleton (see Fig. 3(b)).

Algorithm 3 (Skeleton Computation).

Order all terminals according to their sequence in the tour.
For each terminal t_i,
    For each triangulation half-edge e on the shortest path
    between t_i and t_{i+1},
        $e.mark = i$; $q.append(e)$;
//Shrink the tour
While (q is not empty)
{
    $e = q.takefirst()$;
    If (e is a boundary edge)
        continue; // Ignore it. Otherwise
    $eTwin = e.Twin$;
    If ($eTwin$ is not marked)
    // Then propagate the mark to the twin half-edge
    {
        $e_1 = eTwin.Next$;
        $e_2 = e_1.Next$;
        $eTwin.mark = e_1.mark = e_2.mark = e.mark$;
        $q.append(e_1)$;
        $q.append(e_2)$;
    }
    Else If ($eTwin.mark \neq e.mark$)
        Export e; // This edge is part of the skeleton
}

4.4 Straightening the Skeleton

The skeleton as produced above is always jagged and should be smoothed to produce a better result. As intended, the skeleton is a tree spanning all terminals.

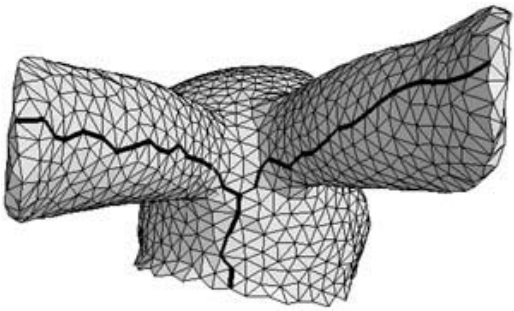

Fig. 8. Computed seam on an open model (a bunny head).

While all terminals are leaves of the tree, some interior vertices have valence greater than two. Such vertices are called *Steiner points*. We fix the positions of all Steiner vertices and all extremal vertices, and replace the skeleton path between each pair of such vertices by the shortest path connecting them in the triangulation (see Fig. 3(c)).

Finally, the set of smoothed paths is output as the seam.

5 Results

The following examples show seams computed by our method. Fig. 8 shows a bunny head model (with boundary). In this example, the three terminals include two extremal vertices at the tops of both ears, and a boundary loop. The seam computed by our method is almost the shortest path.

In Fig. 9, we demonstrate seams on a complex model produced by manually adding increasing numbers of extremal vertices. The model has 10062 facets. It only takes 1.6 seconds to compute the final seam (Fig. 9(f)). The bottleneck in our algorithm is computing the shortest path between any two terminals, which takes $O(E \log N)$ time [20], where E is the number of edges and N is the number of vertices in the mesh.

In Fig. 10(a), we can see the result is good even if the selected extremal vertices are not on protrusions. For such a head model, it is not appropriate for the seam to pass through the face. By using other heuristic rules when we construct the tour, the user can interactively guide seam generation (see the discussion in Section 6).

6 Conclusion and Future Work

The motivation for this paper was to reduce distortion when a 2-manifold triangulated surface is mapped to a single chart in the plane. Most previous approaches reduce distortion by optimizing an energy function [5,19]. Unlike these

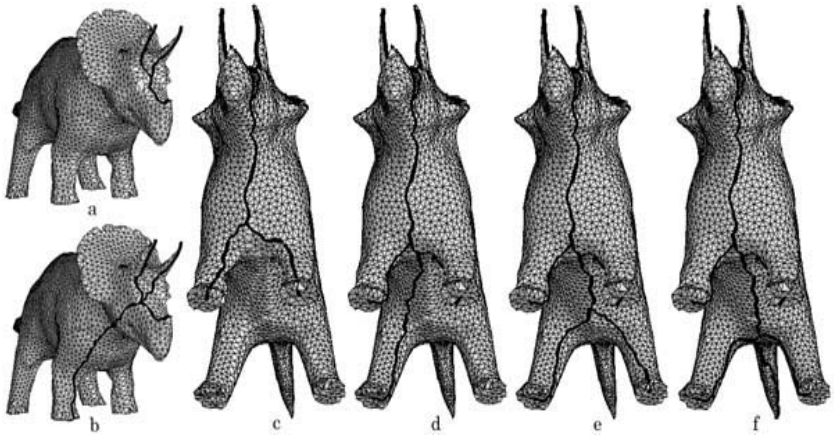

Fig. 9. Seams found by interactively adding new extremal vertices.

methods, we achieve the goal by finding an additional seam. Through experience, we have found that to get a good mapping, it is important for the seam to pass through all extrema just once. Such a seam is a constrained Steiner tree in which all terminals are leaves. Inspired by the straight skeleton of a simple planar polygon, we have presented a novel method for computing the seam. From Fig. 1, we can see that our method leads to shorter seams than previous methods. Furthermore, seams computed with our method treat extrema in a more-or-less symmetrical way, and produce seams similar to those produced by human intuition when mapping textures to a surface.

In Section 4.1, a heuristic rule was used to determine the seam: minimize the tour length. Now, the seam definitely lies on the part of the triangulation surrounded by the tour, and so we can use other heuristics based on this to find the seam. For example, we could minimize the total energy of all triangles interior to the tour, where the triangle energy may be defined in terms of area of triangle, or in terms of "visibility" [21]. Fig. 10 is the result when we define triangle energy as the dot product between the normal vector and the viewing direction. Using such a rule, the user can control the position of the seam according to the desired viewing direction.

Our method relies on correctly knowing all extremal vertices. If extremal vertices are found corresponding to all *large* protrusions, we can obtain very good results, as illustrated in Fig. 9. It is less important whether further extremal vertices corresponding to *small* protrusions are also provided. If not, the results are usually still acceptable (see Fig. 10). However, if an extremal vertex for some large protrusion is not provied, the seam may pass through it, resulting in a bad seam. Thus, in future work, we hope to develop a multi-resolution algorithm for selecting extremal vertices more carefully. We hope to first find all big protrusions at low resolution and then locate extremal vertices precisely at high resolution.

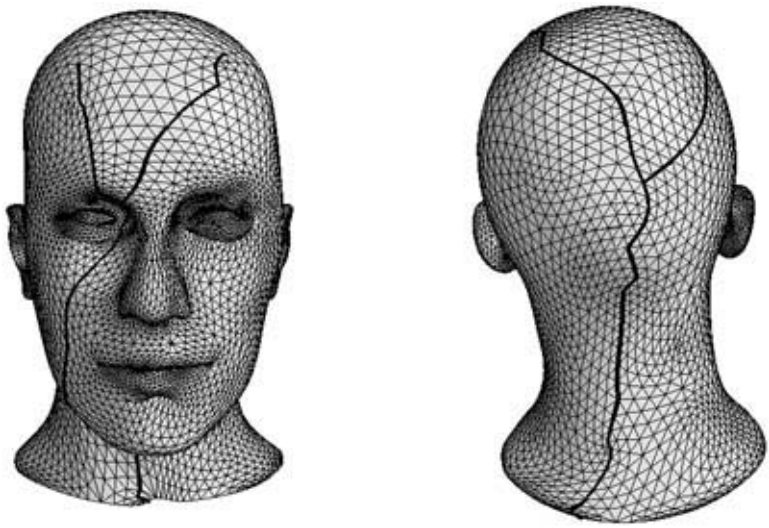

Fig. 10. Seams computed for different view points.

Acknowledgement

This work was partly supported by the Natural Science Foundation of China (Grant 60225016), the Specialized Research Fund for the Doctoral Program of Higher Education (Grant 20020003051) and the National Basic Research Project of China (Grant 2002CB312102). We especially thank the reviewers for their helpful comments.

References

1. Aichholzer O., and Aurenhammer F.: Straight skeletons for general polygonal figures in the plane. In: Proceedings of 2nd International Conference on Computing and Combinatorics, Hong Kong, (1996) 117–126.
2. Allize P., Meyer M. and Desbrun M.: Interactive geometry remeshing. In: Proceedings of SIGGRAPH 2002. ACM Transaction on Graphics, Vol. 21–3 (2002) 347–354.
3. Christofides N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1976.
4. Cormen T. H., Lieseron C. E., and Rivest R. L.: Introduction to Algorithms. MIT Press, Cambridge, 2000.
5. Desbrun, M., Meyer, M., Alliez, P.: Intrinsic parameterizations of surface meshes. In: Proceedings of Eurographics 2002. Computer Graphics Forum 21–3 (2002) 209–218.
6. Eck, M., Derose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle W.: Multiresolution Analysis of Arbitrary Meshes. In: Proceedings of SIGGRAPH 95. Computer Graphics 29 (1995) 173–182.

7. Erickson, J., and Har-peled, S.: Cutting a surface into a disk. In: Proceedings of the 18th ACM Symposium on Computational Geometry (2002), 244–253, ACM Press.
8. Floater, M.: Parameterization and smooth approximation of surface triangulations. Computer Aided Geometric Design 14–3 (1997) 231–250.
9. Gotsman, C., Gu, X., and Sheffer, A., Fundamentals of spherical parameterization for 3D meshes. In: Proceedings of SIGGRAPH 2003. ACM Transaction on Graphics, Vol. 22–3 (2003).
10. Gu, X., Gortler, S. and Hoppe, H.: Geometry images. In: Proceedings of SIGGRAPH 2002. ACM Transaction on Graphics, Vol. 21–3 (2002) 355–361.
11. Hochbaum S.: Approximation Algorithms for NP-hard Problems, PWS Publishing Company, 1997.
12. Khodakovsky, A., Litke, N. and Schröder, P.: Globally smooth parameterizations with low distortion, In: Proceedings of SIGGRAPH 2003. ACM Transaction on Graphics, Vol. 22-3 (2003).
13. Lee, A., Dobkin, D., Sweldens, W. and Schröder, P.: Multiresolution mesh morphing. In: Proceedings of SIGGRAPH 99, Computer Graphics 33(1999) 343–350.
14. Lee, A., Sweldens, W., Schröder, P., Cowsar, L., and Dobkin, D.: MAPS: Multiresolution adaptive parameterization of surfaces. In: Proceedings of SIGGRAPH 1998. Computer Graphics 32 (1998) 95–104.
15. Lévy, B., Petitjean, S., Ray, N., and Maillot, J.: Least squares conformal maps for automatic texture atlas generation. In: Proceedings of SIGGRAPH 2002. ACM Transaction on Graphics, Vol. 21-3 (2002) 362–371.
16. Piponi, D., and Borshukov, G. D.: Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In: Proceedings of SIGGRAPH 2000. Computer Graphics 34 (2000) 471–478.
17. Praum, E. and Hoppe, H.: Spherical parameterization and remeshing. In: Proceedings of SIGGRAPH 2003. ACM Transaction on Graphics, Vol. 22-3 (2003).
18. Robins, G. and Zelikovsky, A.: Improved Steiner tree approximation in graphs. Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California, (2000) 770–779.
19. Sander, S., Snyder, J., P., Gortler and Hoppe, H.: Texture mapping progressive meshes. In: Proceedings of SIGGRAPH 2001. Computer Graphics 35 (2001) 409–416.
20. Sheffer, A.: Spanning tree seams for reducing parameterization distortion of triangulated surfaces. In Geoff Wyvill(eds): Proceedings of Shape Modelling International 2002, IEEE CS Press (2002) 61–68.
21. Sheffer, A. and Hart, J. Seamster: Inconspicuous low-distortion texture seam layout. Proceedings IEEE Visualization 2002, IEEE CS Press, (2002) 291–298.