Texture Mapping with Hard Constraints Using Warping Scheme

Tong-Yee Lee, *Member*, *IEEE*, Shao-Wei Yen, and I-Cheng Yeh

Abstract—Texture mapping with positional constraints is an important and challenging problem in computer graphics. In this paper, we first present a theoretically robust, foldover-free 2D mesh warping algorithm. Then, we apply this warping algorithm to handle mapping texture onto 3D meshes with hard constraints. The proposed algorithm is experimentally evaluated and compared with the state-of-the-art method for examples with more challenging constraints. These challenging constraints may lead to large distortions and foldovers. Experimental results show that the proposed scheme can generate more pleasing results and add fewer Steiner vertices on the 3D mesh embedding.

Index Terms—Foldover, path-swapping, positional constraints, texture mapping, warping.

1 Introduction

Texture mapping technique usually assigns (u, v) texture image coordinates to 3D mesh vertices. This assignment can be computed using mesh planar parameterization. Many other methods [2], [3], [4], [5], [9], [10], [12], [13], [17], [20], [23], [25] have also been proposed. A more challenging problem is to compute this assignment with a special correspondence between the mesh geometry and the texture. This problem is called constrained texture mapping [2], [4], [10], [12], [25]. The hard constraint states that the feature vertices on the parametric domain are exactly aligned to the user-specified positions. In this paper, two major contributions are listed below:

1. For a given 2D mesh, we propose a theoretically robust and foldover-free warping algorithm to align the positional constraints.
2. We extend this new warping algorithm to handle texture mapping with *hard* constraints. In contrast to the state-of-the-art method [10], this new constrained texture mapping can handle more challenging constraints with less danger of distortions. Furthermore, our method adds a fewer number of Steiner vertices on the 3D mesh embedding.

The rest of the paper is organized as follows: Section 2 describes the most related work. We first present a foldover-free 2D mesh warping algorithm in Section 3 and then introduce our application to the constrained texture mapping in Section 4. Section 5 demonstrates the experimental results and gives discussion. Our conclusions and future work are given in Section 6.

- *The authors are with the Computer Graphics Group/Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng-Kung University, No.1, Ta-Hsueh Road, Tainan 701, Taiwan, R.O.C.*
E-mail: {tonylee, ichenyen}@mail.ncku.edu.tw, legends@csie.ncku.edu.tw.

Manuscript received 26 Oct. 2006; revised 28 June 2007; accepted 2 July 2007; published online 2 Aug. 2007.
Recommended for acceptance by C. Gotsman.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0194-1006.
Digital Object Identifier no. 10.1109/TVCG.2007.70432.

2 Related Work

Several methods in [1], [2], [4], [6], [10], [11], [12], [19], [21], [25] have addressed the problem of parameterization under constraints, and several applications are explored including morphing, compatible parameterization, remeshing, and constrained texture mapping. Desbrun et al. [2] and Lévy [12] solved constraints using Lagrange multipliers and a least square system, respectively. However, both methods failed to guarantee a bijective embedding. On the other hand, Lee and Huang [14] used RBF to warp embedding for aligning features. However, this method can create foldovers. Eckstein et al. [4] used a constrained simplification to align the constraints and reconstructed simplified vertices without foldovers by adding Steiner vertices. Fujimura and Makarov [7] presented a 2D image warping method with several constraints. Their approach repeatedly used the Delaunay triangulation and edge swaps to avoid foldovers. The examples presented in [4] and [7] were simple; so, it is not clear how these methods were able to handle more complicated constraints. In addition, Zöckler et al. [26] also used [7] to handle constraints in 3D morphing. However, the authors mentioned that this approach could potentially create foldovers when the positions of the corresponding features are very distinct between two embeddings. Kraevoy et al. [10] described a state-of-the-art matchmaker algorithm for solving constraints in texture mapping. However, this algorithm may fail because it does not consider consistent neighbor ordering as mentioned in [21]. This greedy path-matching approach fails to handle challenging constraints well, and some examples will be shown in Section 5.

Many methods for surface parameterization with constraints have been introduced to embed a 3D surface onto a simpler intermediate domain. Alexa [1] suggested a computationally intensive relaxation-based approach to align feature vertices between two spherical parameterizations. This algorithm can potentially fail for a large number of features. Later, Lin et al. [16] used edge swaps to solve this problem on spherical embeddings. However, the edge swaps can damage the geometric surface. Praun et al. [19] consistently parameterized a set of genus-0 meshes into a user-specified simplicial complex. However, this approach potentially produces swirling paths, thereby increasing the

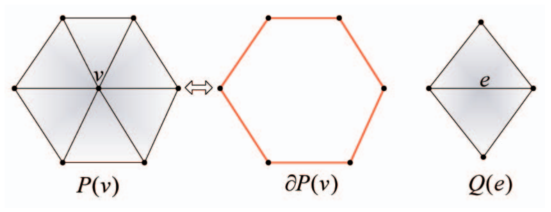


Fig. 1. $P(v)$: the union of triangles adjacent to v. $\partial P(v)$: The boundary of $P(v)$. $Q(e)$: the enclosing quadrilateral of e.

distortion. Later, Schreiner et al. [21] and Kraevoy and Sheffer [11] extended [10] and [19] to consistently parameterize models with the same genus. Recently, Lee et al. [15] proposed a method to consistently parameterize models with different genus on the spherical domain. In this method, the CSG operation potentially damages the surface, in particular, for areas near holes.

3 Foldover-Free 2D Triangle Mesh Warping

In this section, we will present a foldover-free 2D mesh warping method to satisfy positional constraints. This method uses edge swap operations to prevent triangles from folding over. The basic idea behind the proposed method is simple and well known. However, we will show that the edge swaps must be properly arranged; otherwise, they cannot solve foldovers. To the best of our knowledge, this is the first theoretically robust algorithm for solving foldovers that uses edge swaps only. Our approach, meanwhile, can detect foldovers in advance and determine where the edge swaps should be executed to resolve potential triangle foldovers.

3.1 Definitions

We consider a topologically disklike 2D triangular mesh T with a convex boundary. The following notations are used to describe our warping algorithm:

- v. An interior vertex of T.
- e. An edge of T, and let $\overleftrightarrow{e}$ denote a line containing e.
- $P(v)$. The union of triangles adjacent to$v$ (see Fig. 1).
- $\partial P(v)$. The boundary (edges) of $P(v)$.
- $Q(e)$. The quadrilateral formed by the two adjacent triangles sharing an edge e. $Q(e)$ is *convex* if none of its internal angles is greater than π. An edge e can be swapped if $Q(e)$ is convex.
- $\tilde{v}$. A positional constraint for v to align, and $\tilde{v}$ is inside triangular mesh T.
- $\overrightarrow{v_m}$. A moving direction defined as a ray starting from v and passing through $\tilde{v}$, that is, $\overrightarrow{v_m} = \overrightarrow{v\tilde{v}}$.
- v_α. The *invalid* point defined as the following equation:

$$v_\alpha = \arg\min_{p \in K} \|\overline{vp}\|, \tag{1}$$

where

$$K = \{x \mid x \text{ is the intersection of } \overrightarrow{v_m} \text{ and } \overleftrightarrow{e}, \forall e \in \partial P(v)\}.$$

- $E_\alpha(v)$. An edge set defined as

$$E_\alpha(v) = \{e \mid e \in \partial P(v), \text{ and } \overleftrightarrow{e} \text{ contains } v_\alpha\}.$$

Our foldover-free warping algorithm gradually moves v along $\overrightarrow{v_m}$ to align $\tilde{v}$. If $\|\overline{v\tilde{v}}\| \geq \|\overline{vv_\alpha}\|$, we need to move v to v_α first by appropriate edge-swaps and then continue moving it to $\tilde{v}$ gradually. The mesh therefore becomes warped. Fig. 2 shows three different possibilities of v_α: 1) v_α is on $E_\alpha(v)$, except the vertices of $E_\alpha(v)$, 2) v_α is not on $E_\alpha(v)$, and 3) v_α is on any vertex of $E_\alpha(v)$. We will show that v can move to v_α without creating foldovers under the conditions of 1) and 2). If v_α is on any vertex of $E_\alpha(v)$, we will also show that this special case can be resolved such that v can continue moving to v_α along $\overrightarrow{v_m}$.

3.2 Theoretical Analysis and Algorithm

In this section, the following theoretical analysis is given, and the results of analysis are used to derive our foldover-free warping algorithm.

3.2.1 *Theoretical Analysis*

Lemma 1. *If $|E_\alpha(v)| = 1$ and v_α is on $E_\alpha(v)$, v can move to v_α along $\overrightarrow{v_m}$ without flipping triangles after an edge swap operation.*

Proof. v_α is not on any vertex of E_α because $|E_\alpha(v)| = 1$. Let $E_\alpha(v) = e$. Then, we show that e can be swapped before v moves to v_α along $\overrightarrow{v_m}$. If $Q(e)$ is convex, then e can be swapped. If $Q(e)$ is not convex, the following statements show that e can still be swapped. In Fig. 3a, let $e = \overline{p_1p_3}$ and $vp_1p_2p_3$ be the quadrilateral $Q(e)$. Both interior angles

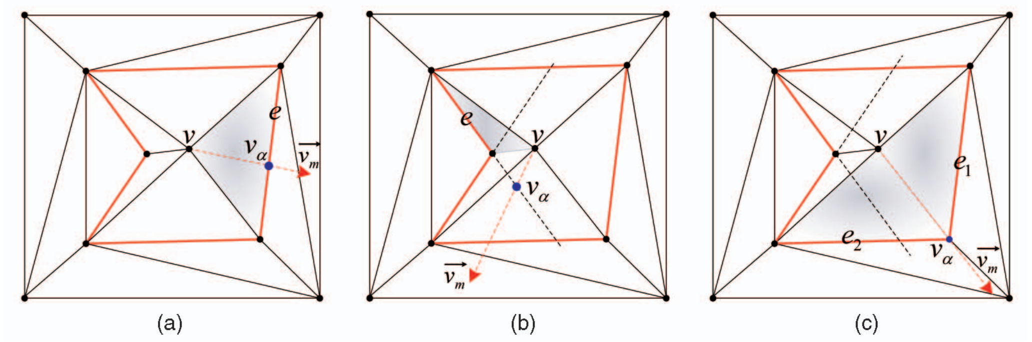


Fig. 2. $\partial P(v)$ consists of red edges. When v moves to v_α, each shaded triangle becomes degenerated, that is, folded over. (a) $E_\alpha(v) = \{e\}$. (b) $E_\alpha(v) = \{e\}$. (c) $E_\alpha(v) = \{e_1, e_2\}$.

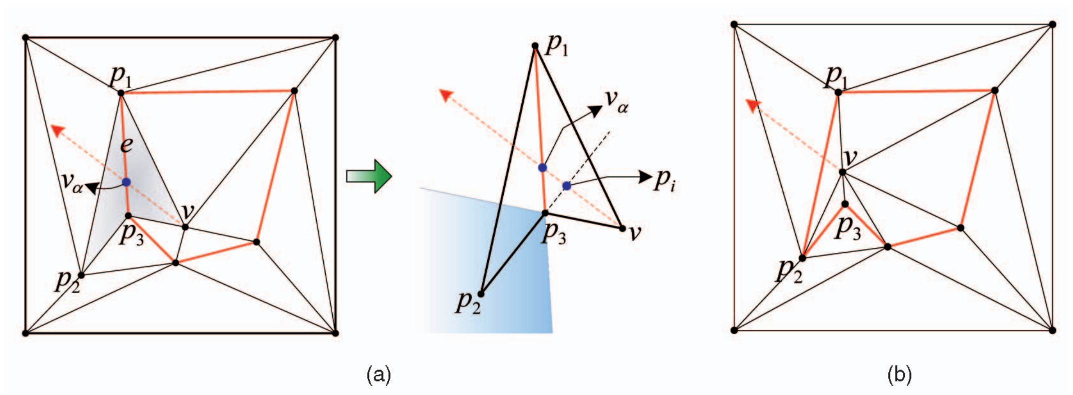


Fig. 3. (a) $E_\alpha(v) = \{e\}$. When v moves to the open interval (p_i, v_α), $Q(e)$ becomes convex and e (that is, $\overline{p_1p_3}$) can be swapped. (b) Edge e is swapped, and v moves to v_α.

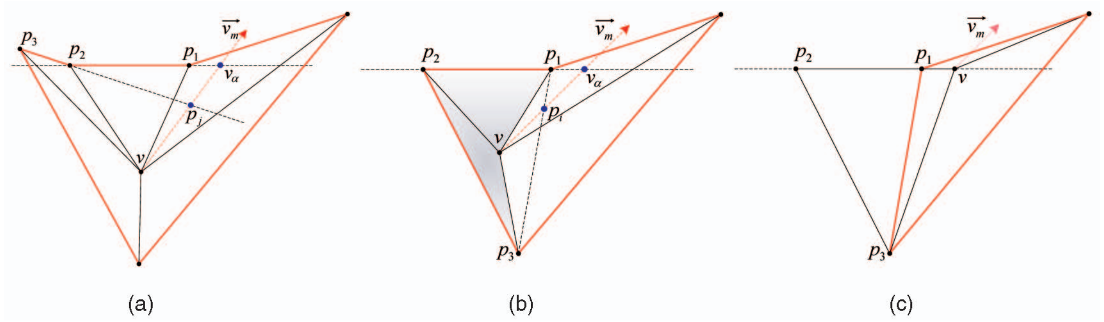


Fig. 4. (a) $\angle p_1p_2p_3 > \pi$ in $Q(\overline{vp_2})$. (b) $\angle p_3vp_1 \geq \pi$ in $Q(\overline{vp_2})$. (c) $\overline{vp_2}$ is swapped, and v moves to v_α.

$\angle p_3vp_1$ and $\angle p_1p_2p_3$ in $Q(e)$ are less than π since they are the internal angles of triangles in 2D triangular mesh T. Without loss of generality, let $\angle p_2p_3v$ in $Q(e)$ be greater than or equal to π. Then, the vertex p_2 is in the shaded blue region defined by the intersection of two half-planes. These two half-planes are formed by $\overleftrightarrow{p_1p_3}$ and $\overleftrightarrow{p_3v}$, respectively, as shown in Fig. 3a. The intersection point p_i of two lines $\overleftrightarrow{p_2p_3}$ and $\overleftrightarrow{vv_\alpha}$ is on the half-closed interval $[v, v_\alpha)$. If v moves to the open interval (p_i, v_α), $\angle p_2p_3v$ is less than π. Then, the edge e can be swapped.

Since $|E_\alpha(v)| = 1$, only the triangle Δvp_1p_3 containing e is degenerated when v moves to v_α, as shown in Fig. 3a. After e is swapped, the possibly degenerated triangle Δvp_1p_3 is removed and Δvp_1p_2, as well as Δvp_2p_3 are created. Thereafter, Δvp_1p_2 and Δvp_2p_3 are not flipped as v moves to v_α, as shown in Fig. 3b. Our result follows.□

Lemma 2. *If $|E_\alpha(v)| = 1$ and v_α is not on $E_\alpha(v)$, v can move to v_α along $\overrightarrow{v_m}$ without flipping triangles after an edge swap operation.*

Proof. Let $E_\alpha(v) = \{\overline{p_1p_2}\}$. v_α is on $\overleftrightarrow{p_1p_2}$ but not on $\overline{p_1p_2}$. Without loss of generality, we assume that p_2 is farther than p_1 from v_α. Let p_3 be a vertex adjacent to both v and p_2 such that $vp_1p_2p_3$ is $Q(\overline{vp_2})$, as shown in Fig. 4a and 4b. The $\angle p_1p_2p_3$ in $Q(\overline{vp_2})$ is not equal to π since $|E_\alpha(v)| = 1$. Let p_j be the intersection of $\overrightarrow{v_m}$ and $\overleftrightarrow{p_2p_3}$. If $\angle p_1p_2p_3$ in $Q(\overline{vp_2})$ is greater than π, v_α is farther than p_j from v, as shown in Fig. 4a. This is a contradiction to the definition of v_α. Then, we know that the vertex p_2 is convex for $Q(\overline{vp_2})$.

Next, we show that the edge $\overline{vp_2}$ can be swapped before v moves to v_α. If the $Q(\overline{vp_2})$ is not convex, we will show that $\overline{vp_2}$ can be still swapped. The $\angle vp_1p_2$ and $\angle p_2p_3v$ in $Q(\overline{vp_2})$ are less than π since they are the internal angles of triangles in a 2D triangular mesh T, and $\angle p_1p_2p_3$ in $Q(\overline{vp_2})$ is also less than π because p_2 is convex for $Q(\overline{vp_2})$. Then, let $\angle p_3vp_1 \geq \pi$ in $Q(\overline{vp_2})$, as shown in Fig. 4b. The intersection point p_i of $\overrightarrow{v_m}$ and $\overleftrightarrow{p_1p_3}$ is on the half-closed interval $[v, v_\alpha)$ since $\angle p_3vp_1 \geq \pi$. If v moves to the open interval (p_i, v_α), $\angle p_3vp_1 < \pi$. Then, edge $\overline{vp_2}$ can be swapped.

Since $|E_\alpha(v)| = 1$, only the triangle Δvp_1p_2 containing $\overline{p_1p_2}$ is degenerated when v moves to v_α, as shown in Fig. 4b. After $\overline{vp_2}$ is swapped, the possibly degenerated triangle Δvp_1p_2 is removed and Δvp_1p_3, as well as $\Delta p_1p_2p_3$, are created. Thereafter, Δvp_1p_3 and $\Delta p_1p_2p_3$ are not flipped as v moves to v_α. Our result follows. □

Lemma 3. *If $|E_\alpha(v)| \geq 1$ and v_α is not on $E_\alpha(v)$, v can move to v_α along $\overrightarrow{v_m}$ without flipping triangles after a set of edge swap operations.*

Proof. There are one or more connected components in $E_\alpha(v)$, as shown in Fig. 5. For each connected component C of $E_\alpha(v)$, we will show that the adjacent triangle of each edge in C can be removed before v moves to v_α, and then, no triangles are flipped when v moves to v_α. We prove this by induction on $|C|$ in the following:

Basis Step. $|C| = 1$, let $C = \{\overline{q_aq_b}\}$ and follow the similar procedure of Lemma 2, that is, let $\overline{q_aq_b}$ be $\overline{p_1p_2}$, as shown in Fig. 4. We can find that the triangle Δvq_aq_b containing $\overline{q_aq_b}$ can be removed and that the triangles created by the swap of $\overline{vq_b}$ are not degenerated when v moves to v_α.

Inductive Step. We assume that the statement is true while $|C| = k$. Now, consider $|C| = k + 1$. First, all edges of C are collinear since $C \subseteq E_\alpha$ and C is connected. Let $\overline{q_sq_t} \in C$. Without loss of generality, $\overline{q_sq_t}$ is the farthest

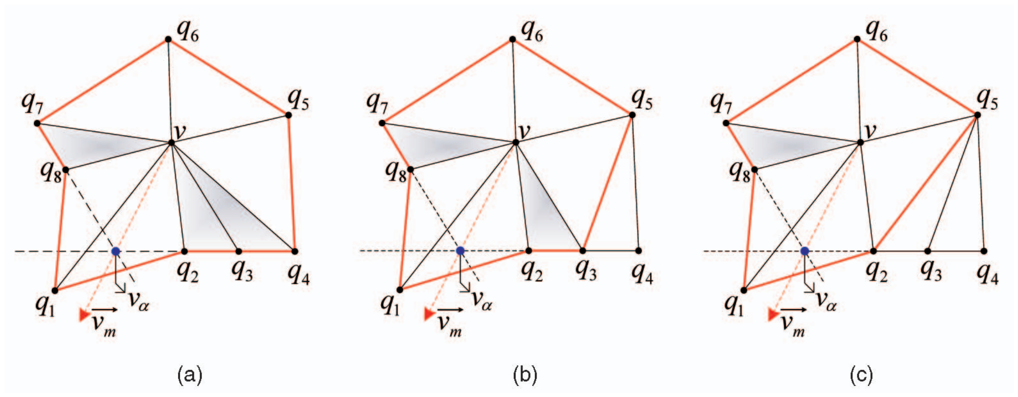


Fig. 5. Shaded triangles are degenerated when v moves to v_α. (a) $E_\alpha(v) = \{\overline{q_2q_3}, \overline{q_3q_4}, \overline{q_7q_8}\}$. The connected components of $E_\alpha(v)$ are $C_1 = \{\overline{q_2q_3}, \overline{q_3q_4}\}$ and $C_2 = \{\overline{q_7q_8}\}$. Before, v moves to v_α, we can swap $\overline{vq_4}$ to remove Δvq_3q_4, as shown in (b), and then swap $\overline{vq_3}$ to remove Δvq_2q_3, as shown in (c).

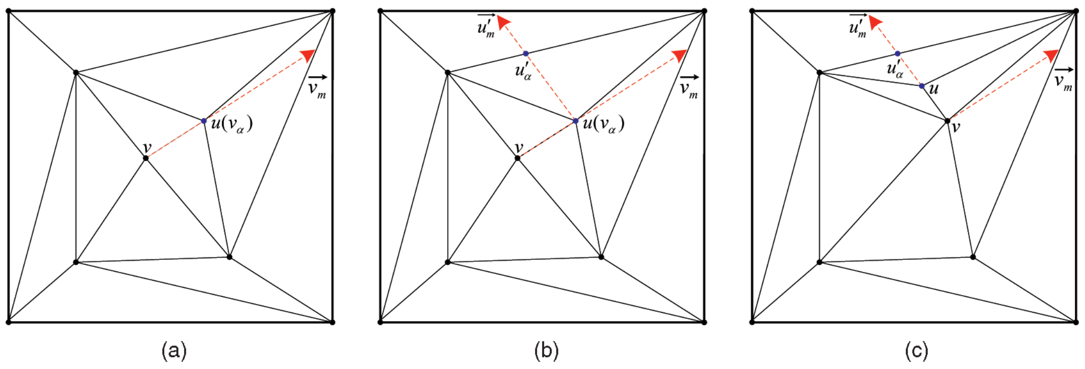


Fig. 6. (a) v_α is on a vertex u of $E_\alpha(v)$. u can find a direction $\overrightarrow{u'_m}$ in (b). Then, u can move to the interval (u, u'_α) and v can move to v_α in (c).

element of C from v_α, and q_t is farther than q_s from v_α. If we follow the similar procedure as that used in Lemma 2, that is, let $\overline{q_sq_t}$ be $\overline{p_1p_2}$, as shown in Fig. 4. The triangle Δvq_sq_t containing $\overline{q_sq_t}$ can be removed, and the new triangles created by edge swap of $\overline{vq_t}$ are not degenerated when v moves to v_α and then $|C| = k$. Using the induction hypothesis, the adjacent triangle of each edge in C can be removed, as shown in Fig. 5. Our result follows. □

Lemma 4. *If $|E_\alpha(v)| \geq 1$ and v_α is on $E_\alpha(v)$, but not on the vertices of $E_\alpha(v)$, v can move to v_α along $\overrightarrow{v_m}$ without flipping triangles after a set of edge swap operations.*

Proof. We will show that the adjacent triangle of each edge in $E_\alpha(v)$ can be removed before v moves to v_α and that no triangles are flipped when v moves to v_α. We classify the edges of $E_\alpha(v)$ into two sets B_1 and B_2.

1. $B_1 = \{\overline{q_iq_j} \mid \overline{q_iq_j} \in E_\alpha(v)$ and v_α is on $\overline{q_iq_j}\}$. Then, $|B_1| = 1$ because v_α can be on one edge of $E_\alpha(v)$ at most. Let $B_1 = \{\overline{q_mq_n}\}$, we can follow the similar procedure as that used in Lemma 1, that is, let $\overline{q_mq_n}$ be the edge e, as shown in Fig. 3. Then, we can find that $\overline{q_mq_n}$ can be swapped to remove the triangle Δvq_mq_n and that the new triangles created by the edge swap of $\overline{q_mq_n}$ are not degenerated when v moves to v_α.
2. $B_2 = \{\overline{q_iq_j} \mid \overline{q_iq_j} \in E_\alpha(v)$ and v_α is not on $\overline{q_iq_j}\}$. By following the similar procedure of Lemma 3, the adjacent triangle of each edge in B_2 can be removed, and the new triangles are not degenerated when v moves to v_α. Our result follows. □

Lemma 5. *If v_α is on a vertex u of $E_\alpha(v)$, v can move to v_α along $\overrightarrow{v_m}$ without flipping triangles.*

Proof. We first show that u can be temporarily moved, such that u is not on the direction of $\overrightarrow{v_m}$. We create a moving direction $\overrightarrow{u'_m}$ for u by randomly picking a direction that is not the same as $\overrightarrow{v_m}$ and $-\overrightarrow{v_m}$, as shown in Fig. 6a. Then, u can find a temporary invalid point u'_α by $\overrightarrow{u'_m}$. If u moves to the open interval (u, u'_α), u is not on the direction of $\overrightarrow{v_m}$, and no triangles are flipped. Then, by Lemma 3 or Lemma 4, v can continue moving to v_α, as shown in Fig. 6c. Our result follows. □

In addition, Fig. 6c shows that u can also move back to v_α along $-\overrightarrow{u'_m}$ to restore its original position by Lemma 3 or Lemma 4 after v leaves v_α along $\overrightarrow{v_m}$.

We can state the following theorem from Lemma 3, Lemma 4, and Lemma 5.

Theorem 1. *v can move to the invalid point v_α along $\overrightarrow{v_m}$ without flipping triangles.*

3.2.2 Foldover-Free Warping Algorithm

Now, we will describe a foldover-free 2D mesh warping algorithm in *MeshWarp*() based on the above analysis. Iteratively, this algorithm moves each vertex v along $\overrightarrow{v_m}$ to $\tilde{v}$. If v is required to move to v_α, the algorithm executes edge swaps to remove possibly degenerated triangles. Then, v moves to v_α using *MoveToInvalidPoint*(v). The procedure *MoveToInvalidPoint*() is derived from Lemma 3, Lemma 4, and Lemma 5. Note that the *StepSpeed* in *MeshWarp*() is a user-defined constant to control the warping speed, and *MinimalStep*(T) returns the minimal distance between each $\overline{v\tilde{v}}$ for all unaligned vertices. In our experiment, if the *StepSpeed* constant is set too large, the number of edge swaps would become bigger, and the large distortion would occur in base mesh. From our experimental experience, the value 0.1 is a good constant for the *StepSpeed*.

```
Algorithm MeshWarp (T) {
  Step ← StepSpeed * MinimalStep(T);
  for each un-aligned vertex v such that v.align == false {
    v.position ← v.position + Step * $v.\overrightarrow{v_m}$ ;
    if $v_\alpha$ is over $\tilde{v}$ then {
      v.align ← true;
      v.positon ← $\tilde{v}$ ;
      Step ← StepSpeed * MinimalStep(T); }  /* update Step */
    else if v is over $v_\alpha$ then{
      /* go to previous position */
      v.position ← v.position – Step * $v.\overrightarrow{v_m}$ ;
      MoveToInvalidPoint(v); }
  }
}
```

```
Procedure MoveToInvalidPoint(v)
{
  if $v_\alpha$ is not on any vertex of $E_\alpha(v)$ then {
    The edges of $E_\alpha(v)$ are classified into two sets $B_1$ and $B_2$ as in
    Lemma 4.
    if $B_1$ is not empty then{
      Let $B_1 = \{ \overline{q_m q_n} \}$ and $Q(\overline{q_m q_n})$ is $vq_m q_o q_n$;
      if $\angle vq_m q_o \geq \pi$ then   /* internal angle of $Q(\overline{q_m q_n})$ */
        Compute the intersection $p_i$ of $\overleftrightarrow{q_o q_m}$ and $\overleftrightarrow{vv_\alpha}$ ;
      else if $\angle vq_n q_o \geq \pi$ then
        Compute the intersection $p_i$ of $\overleftrightarrow{q_o q_n}$ and $\overleftrightarrow{vv_\alpha}$ ;
      else then $p_i$ ← v.position ;
      v.position ← 0.5 * ( $p_i + v_\alpha$ );   /* v moves to $(p_i, v_\alpha)$ */
      EdgeSwap( $\overline{q_m q_n}$ ); }
    if $B_2$ is not empty then{
      do
        for each connected component $C \subseteq B_2${
          Let $\overline{q_s q_t} \in C$, $\overline{q_s q_t}$ is the farthest element from $v_\alpha$ in C,
          and $q_t$ is farther then $q_s$ from $v_\alpha$ ;
          Let $Q(\overline{vq_t})$ be $vq_s q_t q_u$ ;
          if $\angle q_u v q_s \geq \pi$ then {  /* internal angle of $Q(\overline{vq_t})$ */
            Compute the intersection $p_i$ of $\overleftrightarrow{q_u q_s}$ and $\overleftrightarrow{vv_\alpha}$ ;
            v.position ← 0.5 * ( $p_i + v_\alpha$ ); }
          EdgeSwap( $\overline{vq_t}$ ); /* eliminate one element of C */ }
        /* for loop */
      while $B_2$ is not empty ;   /* while loop */ }
    v.position ← $v_\alpha$ ;
    UpdateInvalidPoint(v);   /* find the next $v_\alpha$ of v */ }
  else if (v is on a vertex u of $E_\alpha(v)$) then {
    create $\overrightarrow{u'_m}$ , and compute $u'_\alpha$ by $\overrightarrow{u'_m}$ ;   /* (see Fig. 6) */
    move u to open interval $(u, u'_\alpha)$ ;
    move v to $v_\alpha$ ;   /* use Lemma 3 or Lemma 4 */
    UpdateInvalidPoint(v) ;
    v.position ← 0.5 * ( $v + v_\alpha$ ) ;
    move u to its original position ; /* use Lemma 3 or Lemma 4 */ }
}
```

The warping algorithm may possibly generate very skinny triangles that would cause a numerical error in computation. From our practice, it is usually due to a very small "*Step*" in *MeshWarp*(T). We avoid this problem by limiting the value of "*Step*" to not be smaller than a predefined threshold. In addition, before an edge swap, when v is required to move to the open interval (p_i, v_α), the vertex v is always moved to $0.5 * (p_i + v_\alpha)$ in *MoveToInvalidPoint*(v). Such movement is also useful in preventing the generation of skinny triangles.

4 Texture Mapping with Hard Constraints

4.1 Overview and Definitions

We now extend our new warping algorithm to compute surface parameterization for texture mapping with *hard* constraints. Figs. 7a, 7b, 7c, 7d, 7e, 7f, and 7g show the overall stages. To present this application, we define some terminology used in the following sections. Let M be the input triangle mesh and a set of positional correspondence between M and texture specified at Figs. 7a and 7b. P^M is the planar embedding of M and can be obtained using any bijective planar parameterization. In our implementation, P^M is computed with that in [24]. We also add a set of virtual points surrounding P^M, as suggested in [10], such that the boundary of P^M is free to move in the parametric domain. Let $M+$ be the mesh M with virtual points, and the embedding of $M+$ be P^{M+}, as shown in Fig. 7c. In P^{M+}, each feature vertex (red point at the stage in Fig. 7c) corresponds to a feature vertex of M (red point in Fig. 7a). The feature vertex of P^{M+} will be aligned to a user-specified position on a texture image (red points in Fig. 7b) using our warping algorithm in Figs. 7d, 7e, and 7f. Finally, in Fig. 7g, a variant of stretch metric [20] is used to smooth embedding for reducing distortion. Later in Section 5, we will experimentally compare our method in [10] and give a discussion with the related work.

4.2 Constructing a Base Mesh

We do not directly use the warping algorithm in Section 3 to match feature vertices of P^{M+} since the edge swap operations in P^{M+} will damage the geometry of mesh M. To avoid this problem, we warp a base mesh B^{M+} instead of P^{M+}. A base mesh B^{M+} is constructed from P^{M+}. An edge swap in B^{M+} corresponds to a path swap in P^{M+}, such that a path swap does not change the geometry of mesh M. The construction of base mesh B^{M+} is described as follows:

Let G be a guiding mesh, which is used to construct the base mesh. The vertex set of G consists of feature vertices and virtual points from P^{M+}. Then, we can determine the connectivity of G by the constrained Delaunay triangulation [22]. An example of G is illustrated as a red-color-edge mesh in Fig. 7d. Each edge of G guides us to compute a path on P^{M+}. We use the following steps to find such paths (the green paths shown in Fig. 7d).

1. For each inner edge $E_G \in G$, we first determine a closed region in order to locally compute its corresponding path on P^{M+}. Initially, the boundary of the closed region is formed by the enclosing quadrilateral of E_G, as shown in Fig. 8a. If the corresponding path of E_G is found, E_G is replaced by this path. For example, a guiding edge $\overline{f_3 f_4}$ is replaced by the green path from f_3 to f_4, as shown in Fig. 8b.

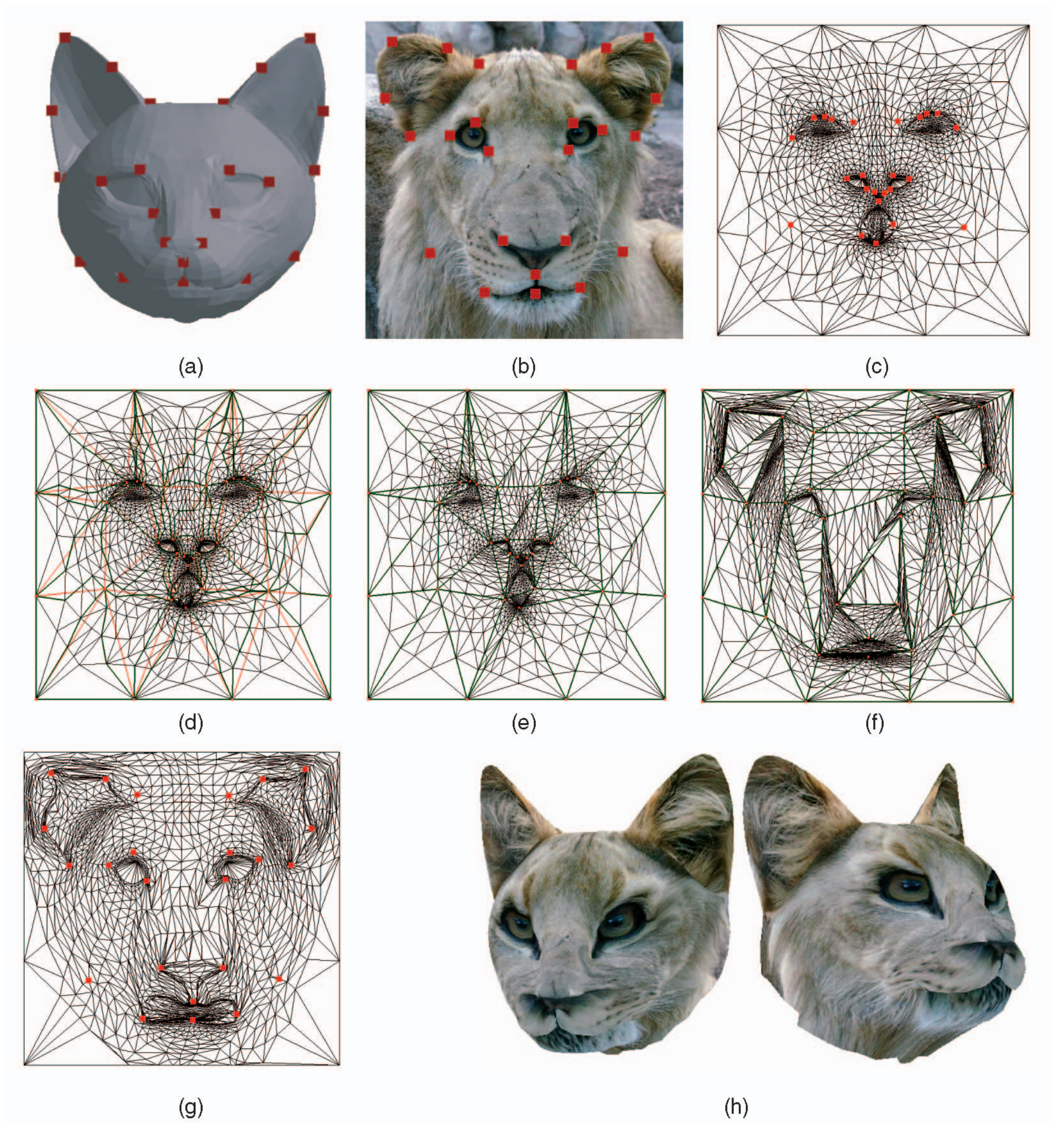


Fig. 7. (a) and (b) Input triangle mesh M and 2D texture. (c) The planar embedding of M with virtual boundary. (d), (e), and (f) Warping algorithm is used to align constraints. (g) The resulting embedding is further smoothed. (h) The resulting textured model.

2. Next, if there exists some edge $\overline{v_m v_n}$ in P^{M+}, such that $\overline{v_m v_n}$ divides the enclosing boundary of E_G into two disjoint subregions, then both end points of $\overline{v_m v_n}$ can be either outside or are right on the enclosing boundary of E_G, for example, an edge $\overline{v_1 v_2}$, as shown in Fig. 8. This edge $\overline{v_m v_n}$ potentially blocks the corresponding path of E_G. If $\overline{v_m v_n}$ is split by a *Steiner* vertex, the corresponding path of E_G can be found. Therefore, we will add a *Steiner* vertex at the midpoint of $\overline{x_m x_n}$. The following rules are used to compute x_m and x_n.

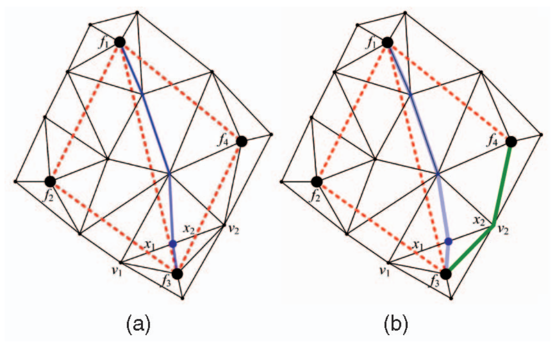


Fig. 8. f_i are feature vertices, and red dotted lines are edges of G. A corresponding path (green) of $\overline{f_3 f_4}$ was computed in (b). In (a) and (b), an edge $\overline{v_1 v_2}$ of P^{M+} divides the enclosing boundary of $\overline{f_1 f_3}$ into two subregions. A Steiner vertex (blue) is added, and the corresponding path of $\overline{f_1 f_3}$ is shown in light blue.

- If both end points of $\overline{v_m v_n}$ are outside the boundary of E_G, x_m and x_n become the intersection points of $\overline{v_m v_n}$ and the boundary of E_G, for example, x_1 and x_2 in Fig. 8a.
- If one end point v_m of $\overline{v_m v_n}$ is outside the boundary of E_G, and the other v_n is right on the boundary of E_G, x_m becomes the intersection point of $\overline{v_m v_n}$ and the boundary of E_G, and x_n is the same with v_n, for example, x_1 and x_2 in Fig. 8b.
- If both end points of $\overline{v_m v_n}$ are right on the boundary of E_G, x_m can be v_m, and x_n can be v_n.

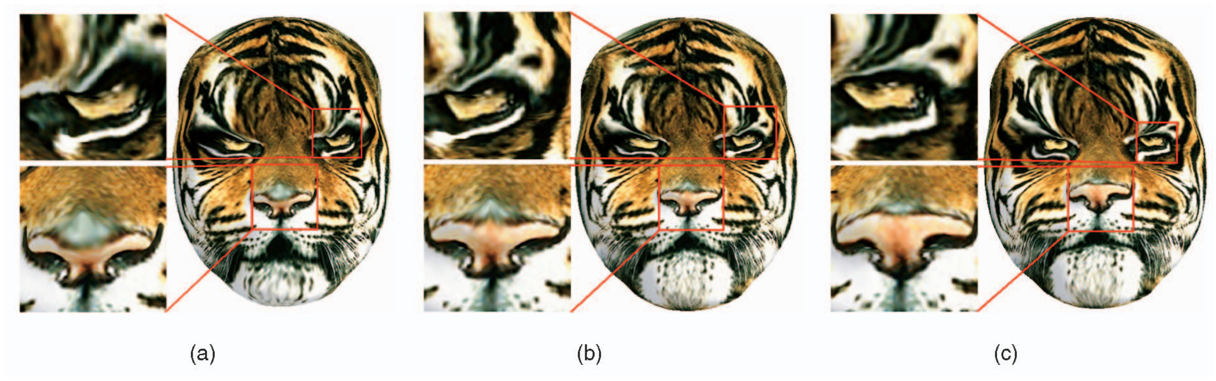


Fig. 9. Mesh smoothing using (a) [10], (b) the metric $\phi(t)$, and (c) the weighted metric $w \cdot \phi(t)$.

3. We can find the correspondent path of E_G by *Dijkstra*'s shortest path algorithm within the enclosing boundary of E_G. Only the vertices inside the enclosing boundary of E_G can be joined to the shortest path computation.

In addition, for each boundary edge of G, its corresponding path consists of the boundary edges of P^{M+}. After constructing the corresponding path of each edge in G, P^{M+} is then decomposed into a set of triangular patches, for example, in Fig. 7d, the boundary of each patch is drawn in green. Then, we can straighten the boundary of each patch in P^{M+}, and the base mesh B^{M+} would therefore be created, as shown in Fig. 7e. We parameterize the vertices inside each patch by Tutte's method [24]. Then, the vertices inside each triangular patch are encoded by the barycentric coordinates with respect to the triangle nodes of B^{M+}. Note that before we straighten the boundary of each base triangle, there may be some vertices outside the corresponding base triangle. However, after 1) straightening the boundary of each triangle and 2) parameterizing the vertices inside each triangle patch using [24], the vertex inside the triangle patch is inside the corresponding triangle of base mesh, and the resulting barycentric coordinates become valid.

4.3 Warping a Base Mesh

To match constraints, we warp a base mesh B^{M+} instead of an embedding P^{M+}. Each edge swap of B^{M+} corresponds to a path swap since the edge of base mesh is actually a path on P^{M+}. Say, the edge e of B^{M+} to be swapped. First, we find the quadrilateral $Q(e)$ and reparameterize the inner vertices contained in $Q(e)$ using barycentric parameterization [24]. From Section 3, we know that $Q(e)$ can become convex before the edge swap. Therefore, the barycentric parameterization is always valid. Next, let the other diagonal of $Q(e)$ be a guiding edge to compute a new path P for the swap of e. The new path P is computed by *Dijkstra*'s shortest path algorithm. An additional *Steiner* vertex will be added if there exists an edge in P^{M+}, such that its two end points are on the boundary of $Q(e)$. Once path P is determined, it needs to be straightened, and two new triangular patches adjacent to P are reparameterized. Therefore, the edge e of B^{M+} is swapped. Note that the inner vertices in each triangular patch of B^{M+} are encoded by the barycentric coordinates. These vertices are extracted to compute a new path before an edge swap in B^{M+}, and the barycentric coordinates of these vertices need to be updated after an edge swap in B^{M+}.

4.4 Smoothing the Embedding

After aligning the constraints, the embedding is distorted. As suggested in [10], the postembedding smoothing will be useful in reducing distortion. Our smoothing procedure is a variant in [20] and is based on a restricted and iterative relaxation procedure. All feature vertices are restricted to be stationary, whereas the positions of other nonfeature vertices are iteratively adjusted. The adjustment aims at minimizing a stretch metric $\phi(t)$ for each triangle t defined below:

$$\phi(t) = \alpha^2_{stretch} + \beta^2_{stretch},$$
$$\text{where } \alpha_{stretch} = \begin{cases} \Gamma, \Gamma \geq 1 \\ 1/\Gamma, \Gamma \leq 1 \end{cases} \text{ and } \beta_{stretch} = \begin{cases} \gamma, \gamma \geq 1 \\ 1/\gamma, \gamma \leq 1. \end{cases} \tag{2}$$

Γ and γ are singular values derived from that in [20]. In $\phi(t)$, both enlargement Γ and shrinkage γ are treated with the same importance to measure the distortion of t. To further reduce the distortion for the neighborhood of the feature vertex, we use the weighted distortion metric $w \cdot \phi(t)$ instead of $\phi(t)$ to smoothen the embedding. The value of w is computed by $1/d_f^2$, where d_f is the distance between the centroid of t and the

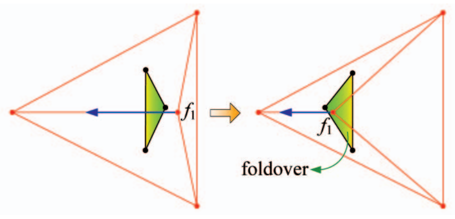


Fig. 10. Example of failure in Zöckler et al. [26]. The red nodes are feature vertices, and the black nodes are the other mesh vertices encoded by the barycentric coordinates. The shaded triangle is flipped when the feature f_1 continues moving.

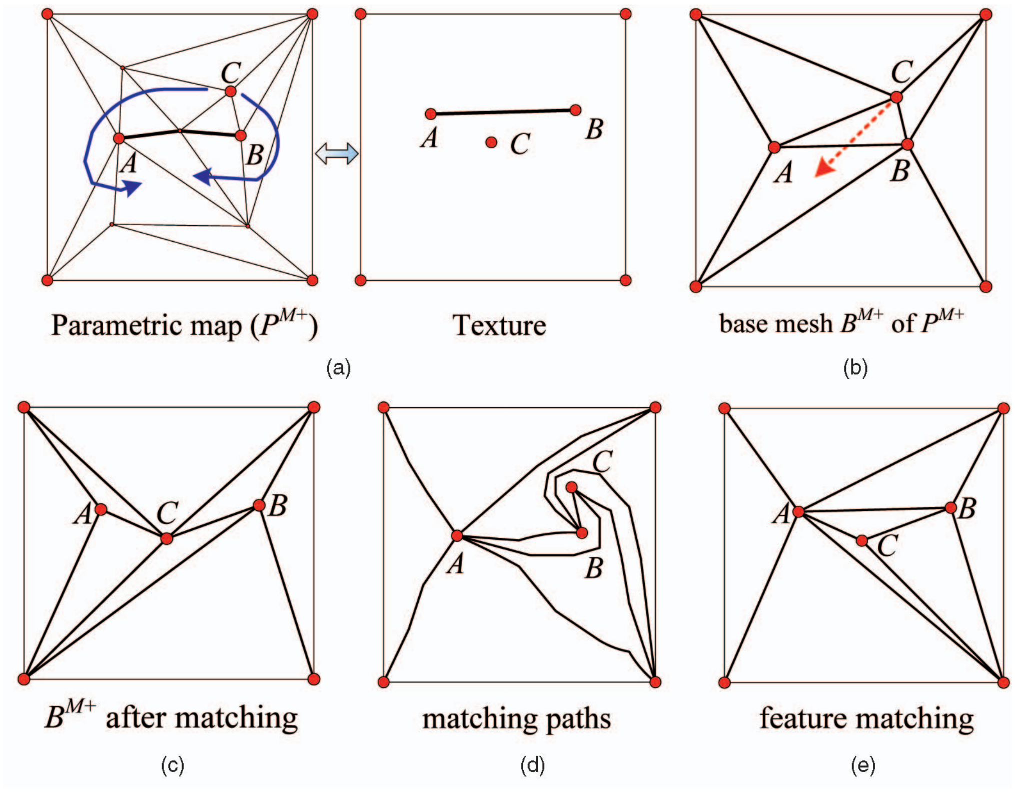


Fig. 11. (a) Feature vertex C is at the different side of path AB. (b) Path AB does not prevent C from moving since it can be swapped. (c) Feature matching in (b). (d) The matching paths using [10]. (e) Feature matching from that in (d). Note that the vertices inside each triangle patch are not shown in (b), (c), (d), and (e) for easy illustration.

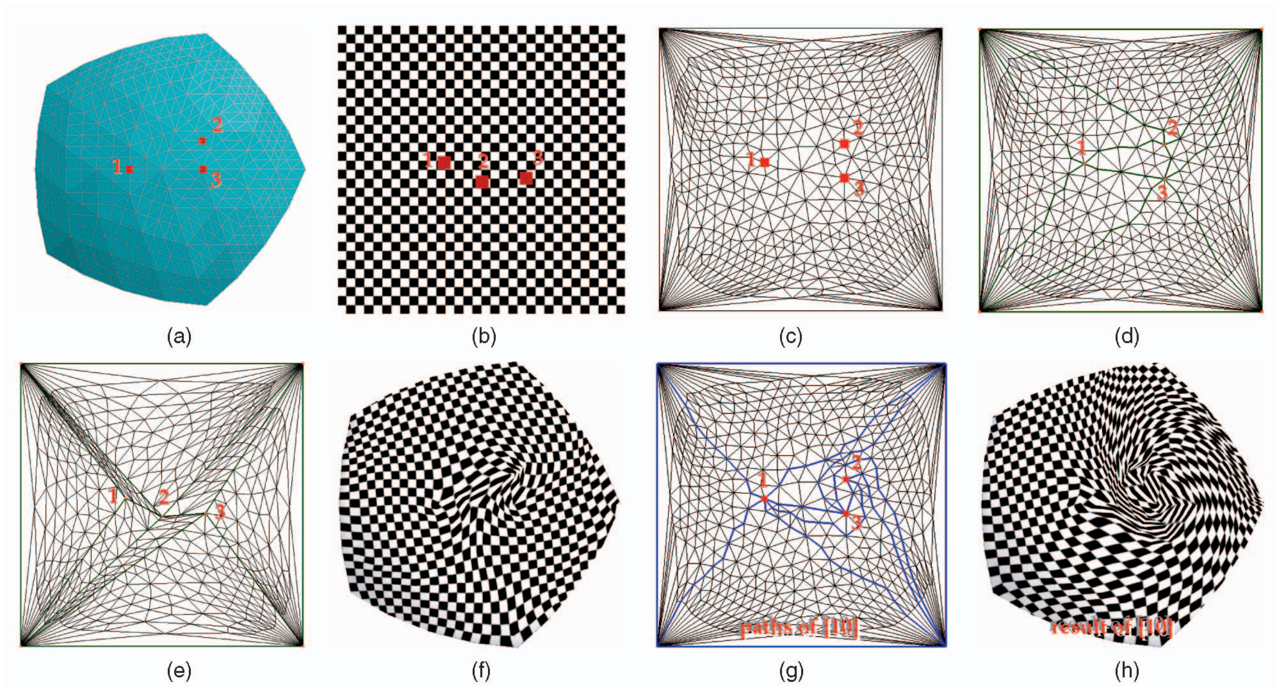


Fig. 12. (a) Input mesh M. (b) Input texture. (c) The embedding P^{M+} of M. (d) The initial paths (green) of base mesh B^{M+}. (e) B^{M+} after feature matching. (f) Texture mapping result after smoothing from (e). (g) The path matching using [10]. (h) Texture mapping result in [10].

nearest feature vertex of t. Fig. 9 shows an experimental comparison between [10] and our mesh smoothing method using $\phi(t)$ and $w \cdot \phi(t)$. The weighted metric $w \cdot \phi(t)$ performs the best in this example.

5 Experimental Results and Discussion

5.1 Results

When corresponding features are very distinct, it is easy to yield large distortions and foldovers. We experimentally

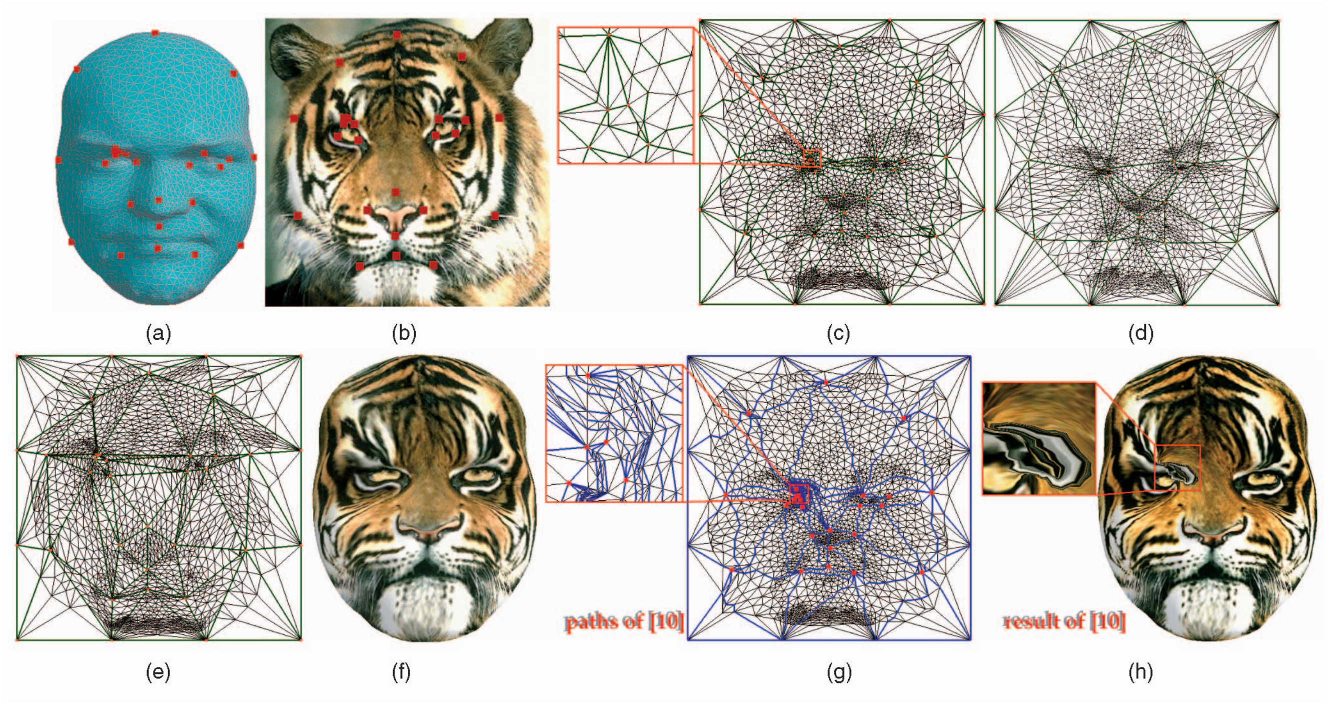


Fig. 13. (a) Input mesh M. (b) Input texture. (c) The embedding P^{M+} of M and the initial paths (green) of base mesh B^{M+}. (d) Initial base mesh B^{M+}. (e) B^{M+} after feature matching. (f) Texture mapping result after smoothing from (e). (g) The path matching using [10]. (h) Texture mapping result in [10].

compare the proposed method and that in [10] for challenging examples as follows: The initial planar embedding for each example is computed by Tutte [24]. In Fig. 12, we design a special example, where the orientation of corresponding features is different between the texture in Fig. 12b and embedding in Fig. 12c. As shown in Figs. 12g and 12h, the proposed method creates a much better result. In Fig. 12f, long swirls are created by their approach, and thus, a large distortion is made in Fig. 12h. In Figs. 12d and 12e, our method simply swaps the edge e_{13} and moves feature vertex v_2 to its positional constraint. Other two examples are evaluated, as shown in Figs. 13 and 14. In both examples, in [10], a number of Steiner vertices are added since many paths wind around some feature vertices, as shown in Figs. 13f and 14f. Figs. 13g and 14g show that our result is still better than Figs. 13h and 14h. Furthermore, we compare the number of Steiner vertices introduced by both methods. We add 0, 10, and 34 Steiner vertices for those in Figs. 12, 13, and 14, respectively. However, the number of Steiner vertices created by their approach is 12, 368, and 2,687 for Figs. 12, 13, and 14, respectively. In Figs. 12, 13, and 14, the texture mapping results in [10] are obtained after smoothing is applied by our proposed smoothing method. There are some other nice results created by our method, as shown in Figs. 15, 16, and 17. Kraevoy et al. [10] mentioned some challenging examples similar to those in Figs. 16 and 17. Our method can also handle these similar examples well. In Fig. 15, the cow example (Figs. 15c and 15d) is a very challenging one due to lots of deformations in the horns. We show the final planar embeddings after smoothing for the cow model, especially near the horns.

Table 1 shows several statistics for our experiments. We implemented the proposed algorithm on an Intel Pentium 4, 2.4-GHz PC with 1 Gbyte of RAM. In our implementation, we also removed unnecessary Steiner vertices after mesh optimization, as suggested in [10]. For Figs. 15, 16, and 17, the texture mapping results are very similar to those in [10] in visual perception, but the number of Steiner vertices added by our approach is much fewer than [10], as shown in Table 1. In this table, we show the numbers of Steiner vertices introduced by both methods before and after removing redundant Steiner vertices.

5.2 Discussion with Related Work

For a 3D morphing application, Zöckler et al. [26] used edge-swaps to solve foldovers on base mesh, also called warp mesh. There are many significant differences between our algorithm and theirs. First, each edge of their warp mesh is a direct line connecting two features on the embedding. In contrast, we need to compute a path on the embedding for each edge of warp mesh. We solve foldovers by edge swaps, and each edge swap on the base mesh corresponds to a path swap on the embedding. Second, their approach is not robust and cannot avoid foldovers, as shown in Fig. 10. In contrast, we present a theoretically robust algorithm to detect and solve foldovers. Moreover, our algorithm can robustly handle parameterization under a large set of constraints. These constraints can be very distinct in their positions.

For the path matching-based algorithm [10], [11], [19], [21], an undesired "swirl" situation may arise. In [19], swirling indicates that the matching path may wind around the other path. From our observation, this usually happens while the neighbor vertex of the matching path is located at

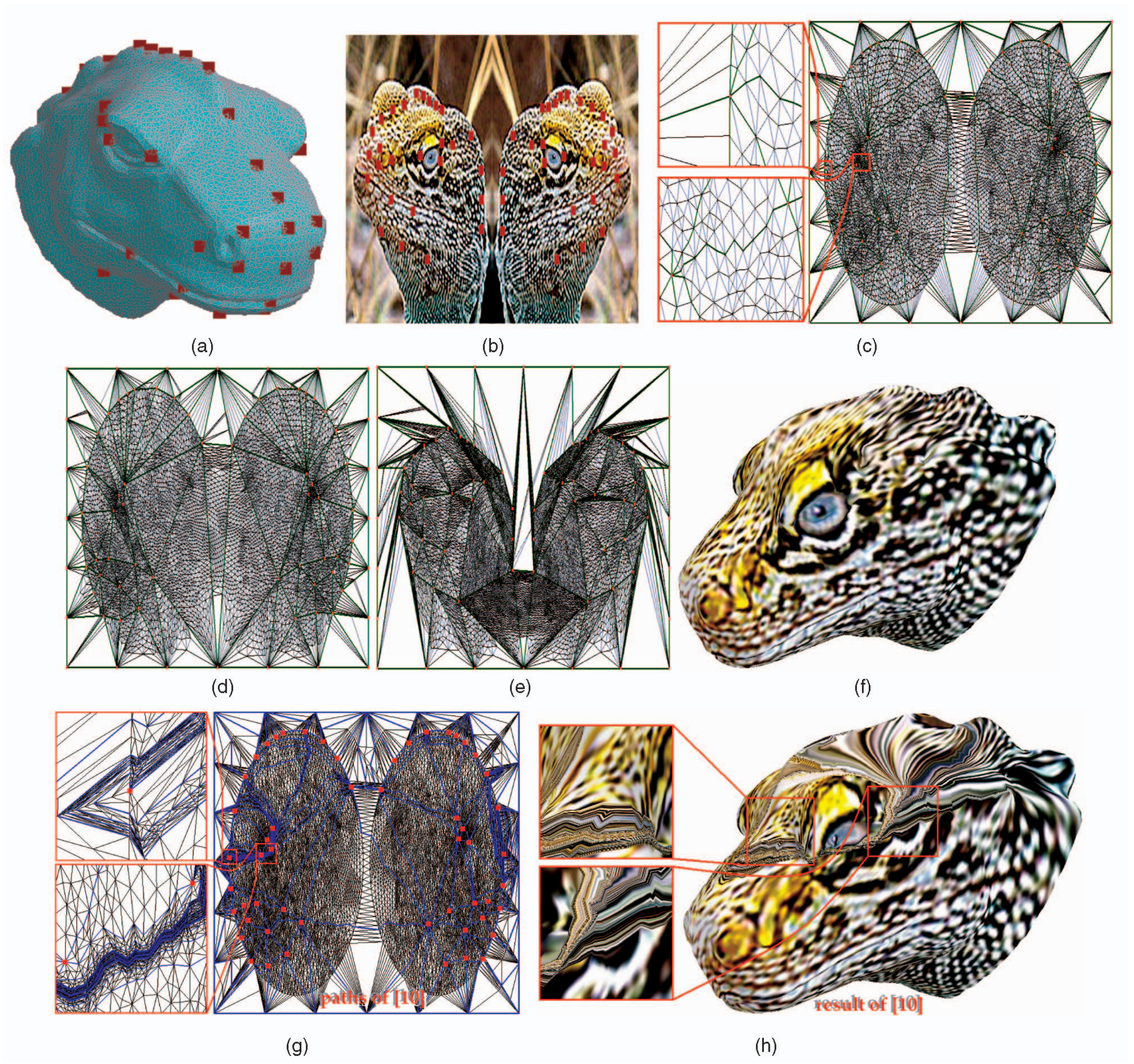


Fig. 14. (a) Input mesh M. (b) Input texture. (c) The embedding P^{M+} of M and the initial paths (green) of base mesh B^{M+}. (d) Initial base mesh B^{M+}. (e) B^{M+} after feature matching. (f) Texture mapping result after smoothing from (e). (g) The path matching using that in [10]. (h) Texture mapping result in [10].

the different side between the embedding domain and the texture. Our warping strategy can easily deal with this issue using the edge swap. We will illustrate this by a simple example, as shown in Fig. 11. Let A, B, and C be the three feature vertices, where there exists a matching path AB, such that the feature point C is at the different side between P^{M+} and the texture. The feature vertex C should move around A or B to align the destination on P^{M+} since the existence of path AB. In contrast, using our approach, this feature vertex C can move directly to the destination using the edge swap of path AB. The result of matching paths using [10] is shown in Fig. 11d. In addition, Praun et al. [19] and Schreiner et al. [21] try to avoid swirling using the heuristics that delays such bad path matching or reroutes the path to let neighbor features be at the same side between the embedding and the texture. In contrast, our approach simply uses the edge swap to avoid swirling instead of these more complicated heuristics. Finally, regarding a path finding, we should also mention the following observation. For [10], [11], [19], and [21], their path-matching algorithms require tedious tests for path blocking and path searching in a global and greedy manner. In contrast, the path finding and block checking are only executed within the local enclosing quadrilateral, as shown in the warping algorithm of Section 3.2.2.

6 Conclusion

We presented a new and theoretically robust warping algorithm to align features for the planar embedding of a triangle mesh. Our algorithm robustly detects where a foldover occurs and determines where to execute edge swaps. This new algorithm is extended to solve texture

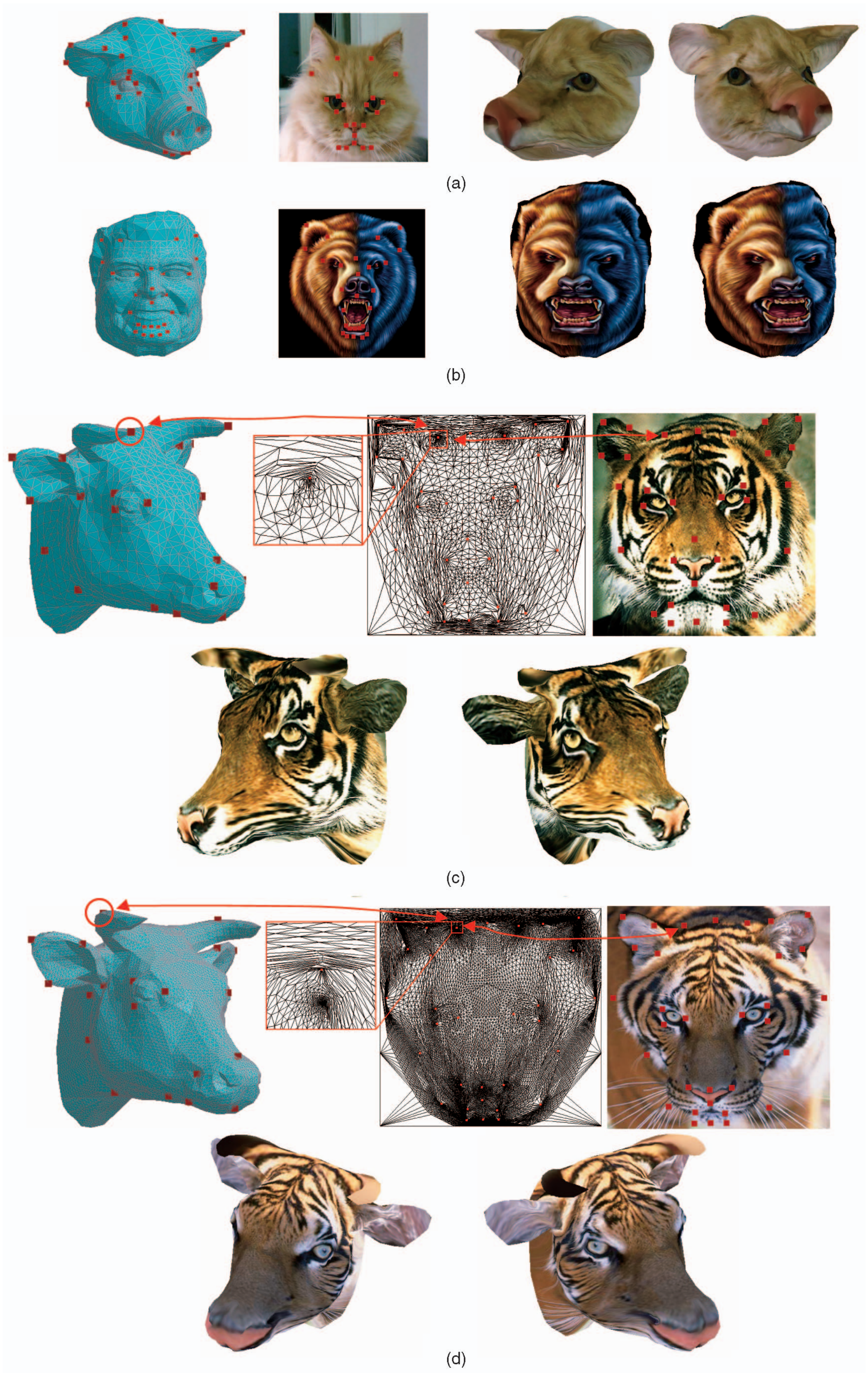


Fig. 15. More results. In (c) and (d), the embeddings are the final results after smoothing.

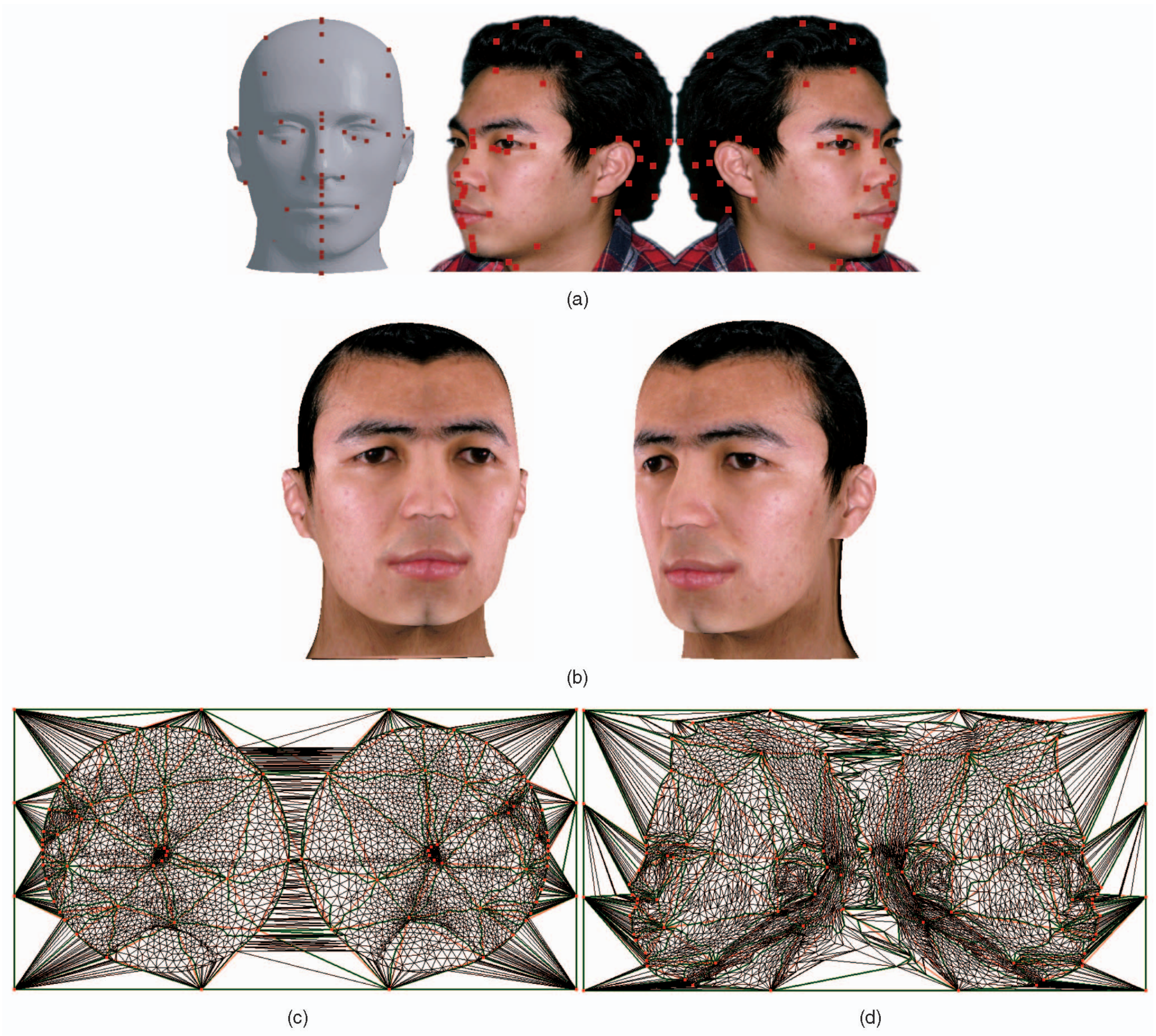


Fig. 16. (a) A hairless head model with the texture image. (b) Texture mapping results. (c) Initial parametric map. (d) Final embedding after mesh smoothing.

mapping with hard constraints. This new algorithm employs path swaps to eliminate possible triangle foldovers during the matching constrained vertices process between embedding and texture. We experimentally verify the proposed method and also compare it with the state-of-art method for some challenging examples. Our results show the advantage of the proposed method in handling challenging constraints well. In the near future, many related and interesting researches will be continued. For example, minimizing the number of edge swaps and intelligently scheduling edge-swaps to avoid skinny triangle shapes of the base mesh will pose very challenging problems. In addition, we plan to apply our method to other applications, such as morphing and consistent parameterization of several models. Our smoothing is an iterative and local relaxation scheme. It cannot be computed on the fly. We will investigate possible alternative to fast smoothing embedding. In constrained texture mapping, constraints might be localized, and therefore, they create a sort of groups that might cause a highly distorted area. A global relaxation method may reduce the problem (at least visually). We will investigate this possibility in the near future.

ACKNOWLEDGMENTS

The authors give their special thanks to Dr. Kun Zhou and Jianwei Han at Microsoft Research Asia who implemented a matchmaker program and helped them with the experimental comparison to that in [10]. They also give their special thanks to their colleague Professor Sun-Yuan Hsieh, who helped them check the correctness of proof in this paper. This project is supported by the National Science Council, Taiwan, under Contracts NSC-95-2221-E-006-368, NSC-95-2221-E-006-193-MY2, and NSC-96-2628-E-006-200-MY3.

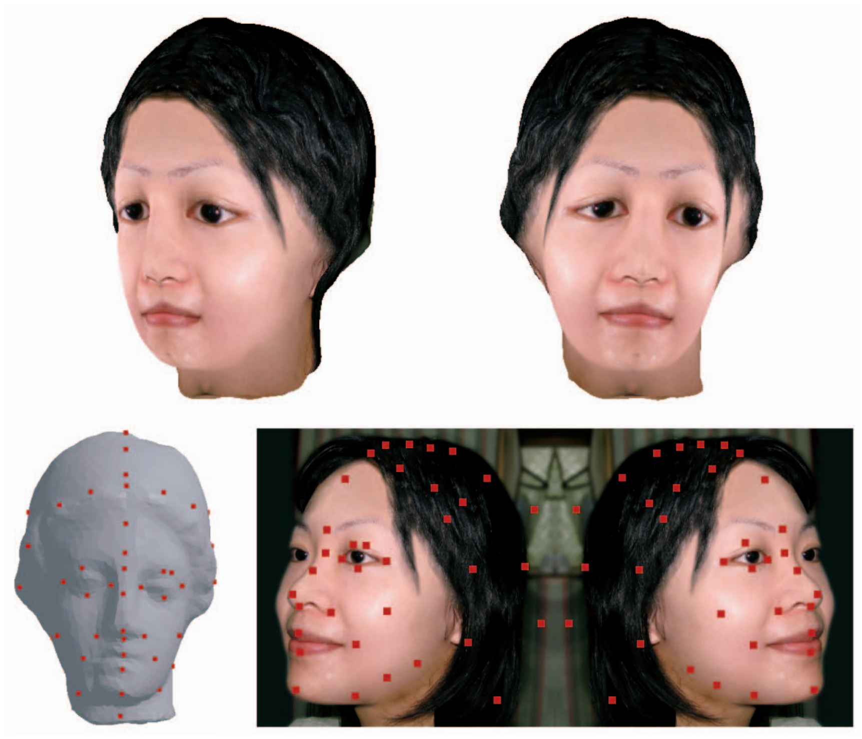


Fig. 17. An Igea model and the texture mapping results.

TABLE 1
The Statistics for the Texture Mapping Samples

Model	# vertices	# triangles	# feature points	Feature warping (seconds)	# Steiner (initial)	# Steiner (final)	# Steiner (initial) [10]	# Steiner (final) [10]
Fig. 7	1770	3526	24	0.625	77	7	596	43
Fig. 13	1808	3602	25	1.712	128	10	1488	368
Fig. 14	10017	20008	54	10.641	94	34	9365	2687
Fig. 15(a)	1772	3450	21	3.359	383	21	1343	154
Fig. 15(b)	1657	3300	27	0.75	69	2	456	51
Fig. 15(c)	1697	3384	32	4.39	223	64	1318	193
Fig. 15(d)	10736	21404	32	14.5	127	8	1358	133
Fig. 16	5184	10354	83	10.86	183	38	2620	361
Fig. 17	4149	8284	71	6.14	122	25	1425	192

#Steiner (initial) and #Steiner (final) are the number of Steiner vertices before and after removing.

References

[1] M. Alexa, "Merging Polyhedral Shapes with Scattered Features," *The Visual Computer,* vol. 16, pp. 26-37, 2000.

[2] M. Desbrun, M. Meyer, and P. Alliez, "Intrinsic Parameterizations of Surface Meshes," *Proc. Eurographics '02/Computer Graphics Forum,* vol. 21, no. 3, pp. 209-218, 2002.

[3] M. Eck, T. Derose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Proc. ACM SIGGRAPH '95,* pp. 173-182, 1995.

[4] I. Eckstein, V. Surazhsky, and C. Gotsman, "Texture Mapping with Hard Constraints," *Proc. Eurographics '01/Computer Graphics Forum,* vol. 20, no. 3, pp. 95-104, 2001.

[5] M.S. Floater, "Parameterization and Smooth Approximation of Surface Triangulations," *Computer Aided Geometric Design,* vol. 14, no. 3, pp. 231-250, 1997.

[6] C.-H. Lin and T.-Y. Lee, "Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 1, pp. 2-12, Jan./Feb. 2005.

[7] K. Fujimura and M. Makarov, "Foldover-Free Image Warping," *Graphical Models and Image Processing,* vol. 60, no. 2, pp. 100-111, 1998.

[8] C. Gotsman, X. Gu, and A. Sheffer, "Fundamentals of Spherical Parameterization for 3D Meshes," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 358-363, 2003.

[9] K. Hormann and G. Greiner, "MIPS: An Efficient Global Parametrization Method," *Proc. Curve and Surface Design Conf.,* pp. 153-162, 1999.

[10] V. Kraevoy, A. Sheffer, and C. Gotsman, "Matchmaker: Constructing Constrained Texture Maps," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 326-333, 2003.

[11] V. Kraevoy and A. Sheffer, "Cross-Parameterization and Compatible Remeshing of 3D Models," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 861-869, 2004.

[12] B. Lévy, "Constrained Texture Mapping for Polygonal Meshes," *Proc. ACM SIGGRAPH '01,* pp. 417-424, 2001.

[13] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least Squares Conformal Maps for Automatic Texture Atlas Generation," *ACM Trans. Graphics,* vol. 21, no. 3, pp. 362-371, 2002.

[14] T.-Y. Lee and P.H. Huang, "Fast and Institutive Polyhedra Morphing Using SMCC Mesh Merging Scheme," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 1, pp. 85-98, Jan.-Mar. 2003.

[15] T.-Y. Lee, C.-Y. Yao, H.-K. Chu, M.-J. Tai, and C.-C. Chen, "Generating Genus-n-to-m Mesh Morphing Using Spherical Parameterization: Research Articles," *Computer Animation and Virtual Worlds,* vol. 17, no. 3, pp. 433-443, 2006.

[16] C.-H. Lin, T.-Y. Lee, H.-K. Chu, and Z.-Y. Yao, "Progressive Mesh Metamorphosis," *J. Computer Animation and Virtual Worlds,* vol. 16, nos. 3-4, pp. 487-498, 2005.

[17] J. Maillot, H. Yahia, and A. Verroust, "Interactive Texture Mapping," *Proc. ACM SIGGRAPH '93,* pp. 27-34, 1993.

[18] U. Pinkall and K. Polthier, "Computing Discrete Minimal Surfaces and Their Conjugates," *Experimental Math.,* vol. 2, no. 1, pp. 15-36, 1993.

[19] E. Praun, W. Sweldens, and P. Schröder, "Consistent Mesh Parameterizations," *Proc. ACM SIGGRAPH '01,* pp. 179-184, 2001.

[20] P.V. Sander, J. Snyder, S.J. Gortler, and H. Hoppe, "Texture Mapping Progressive Meshes," *Proc. ACM SIGGRAPH '01,* pp. 409-416, 2001.

[21] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, "Inter-Surface Mapping," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 870-877, 2004.

[22] J.R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," *Lecture Notes in Computer Science,* vol. 1148, pp. 203-222, 1996.

[23] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski, "Bounded-Distortion Piecewise Mesh Parameterization," *Proc. IEEE Visualization,* pp. 355-362, 2002.

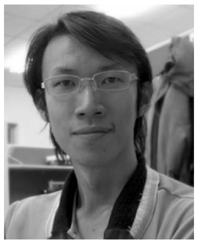
[24] W.T. Tutte, "Convex Representations of Graphs," *Proc. London Math Soc.,* vol. 10, pp. 304-320, 1960.

[25] K. Zhou, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum, "TextureMontage: Seamless Texturing of Arbitrary Surfaces from Multiple Images," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 1148-1155, 2005.

[26] M. Zöckler, D. Stalling, and H.-C. Hege, "Fast and Intuitive Generation of Geometric Shape Transitions," *The Visual Computer,* vol. 16, no. 5, pp. 241-253, 2000.

Tong-Yee Lee received the PhD degree in computer engineering from Washington State University, Pullman, in May 1995. He is currently a professor in the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan. He leads a Computer Graphics Group/Visual System Lab, National Cheng-Kung University (http://graphics.csie.ncku.edu.tw/). His current research interests include computer graphics, nonphotorealistic rendering, image-based rendering, visualization, virtual reality, surgical simulation, medical visualization and medical system, and distributed and collaborative virtual environment. He serves as an associate editor for *IEEE Transactions on Information Technology in Biomedicine* from 2000 to 2007. He is also on the editorial advisory board of the *Journal Recent Patents on Engineering*, an editor of the *Journal of Information Science and Engineering*, and a region editor of the *Journal of Software Engineering*. He is a member of the IEEE and ACM.

Shao-Wei Yen received the BS degree from the Department of Civil Engineering, National Taiwan University, Taipei, in 2000. He is currently working toward the PhD degree at the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include computer graphics and mesh parameterization.



I-Cheng Yeh received the BS degree from the Department of Computer and Information Science, National Chiao Tung University, Taiwan, in 2004. He is currently working toward the PhD degree at the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include computer graphics and mesh parameterization.