

# A Novel Scheme for Efficient Cross-Parameterization

Jia Pan, Huaiyu Wu, Chunhong Pan, Qing Yang

National Laboratory of Pattern Recognition  
Institute of Automation  
Chinese Academy of Sciences  
{jpan, hywu, chpan, qyang}@nlpr.ia.ac.cn

**Abstract** Existing cross-parameterization approaches suffer from several problems: low robustness, slow speed, hard to balance compatible mesh's detail and complexity. These difficulties reduce its usefulness. In this paper, we propose a new scheme to solve these problems. Utilizing the inherent relationship between patch-generation and cross-parameterization, we compute compatible triangle patch layouts on input models. Compared with previous approaches, our method is more robust and much faster. Then we use our active cross-parameterization to obtain high-quality inter-surface mappings and compatible remeshing directly and quickly. Only areas around high-stretch and salient vertices are refined, so compatible mesh's complexity can be well controlled. Moreover, our scheme survives several extreme cases that can cause failure in previous approaches.

## 1 Introduction

Many digital geometry processing applications benefit from *cross-parameterization* (also called *inter-surface mapping*), which builds a bijective mapping between models with different connectivity and transforms them into compatible meshes. Using it as a preprocessing, many originally difficult problems can be greatly simplified. Such advantage is well known in some important applications such as morphing, multi-model shape blending, simultaneous model editing, etc. Recently, skeleton and its parameters' transfer among animation models are also accelerated by this technique [16].

Generally, a good cross-parameterization algorithm should perform well in following aspects: 1)it should be robust enough to survive extreme topological and geometric conditions; 2)compatible mesh should not have too many elements; 3)detail loss caused by different sampling modes should be limited; 4)the inter-surface mapping should be semantically meaningful; 5)the procedure

should have low time-complexity. Though previous approaches try to do well in above aspects, these problems remain unsolved, which prevent the widespread usage of cross-parameterization technique.

Existing cross-parameterization techniques can be classified due to following features: 1)whether it is a local or global scheme; 2)whether parameterization is constructed directly or indirectly; 3)how the compatible mesh's connectivity is constructed. Here, we give a brief overview of previous techniques according to this taxonomy.

Cross-parameterization can be implemented either globally or locally. The former constructs mapping between surfaces as a whole, while the latter splits surfaces into compatible patches and cross-parameterization is operated within each patch separately. One representative of global scheme is *spherical cross-parameterization*, which is used in many works, e.g. [7,9]. Recent work [6] is another global method. These methods' high distortion makes them unsuitable for cross-parameterization between complex models. Many recent works use local scheme, whose distortion is smaller, because each patch can locally be approximated with flat. However, it also brings another difficulty: how to compute compatible patches efficiently. Method able to guarantee an optimal compatible partition is not available currently. A common alternative is to obtain an acceptable partition through certain heuristic techniques, whose complexity greatly influences partition's robustness and speed. Earlier works such as [1,8] are faster but less robust, because they use heuristic techniques work only when models are of nearly identical shapes. Recent works [2-4] use more complex heuristic tools to improve robustness, but they are quite time-consuming: when partition number is large ( $> 80$ ), several hours are needed to compute valid patch layouts, which is unacceptable for interactive applications. Another downside of local scheme is that though continuous within each patch, the mapping may be discontinuous when transiting inter-patch boundaries. So postprocessing, such as *smoothing* in [3], is necessary.

Almost all existing cross-parameterization methods use indirect scheme, i.e. an intermediate domain is first constructed, sub-mapping between each model and the domain is then built separately, finally mapping between models is constructed through sub-mappings' composition. Difference among algorithms lies in the type of intermediate domain to be chosen. Except global methods using sphere, most local methods, e.g. [1–3,8], use a coarser polyhedron mesh. One advantage of indirect scheme is that sub-mappings can be effectively constructed with linear system, such as *mean-value parameterization* [17]. Moreover, for local methods, as surfaces are split into small patches, only a series of small size linear systems need to be solved instead of a large one, which greatly accelerates the solving process. To our knowledge, two direct cross-parameterization algorithms have been proposed. One is the *coarse-to-fine* described in [4]. After a coarse mapping is constructed with edge-collapse constrained mesh simplification, the mapping is refined in a per-vertex way, using nonlinear stretch-minimum optimization, which results in its slow speed. The other is essentially the least-squares approximation using large numbers of feature vertices automatically generated [5]. Despite its high speed, it is not a *cross-parameterization* in strict sense, because its compatible mesh's vertices may not lie exactly on the target surface. Furthermore, it needs pre-subdivision and parameterization's quality is low when processing complicated models.

Another important difference among existing algorithms is the way to construct compatible mesh's connectivity. One widely used technique is *meta-mesh* [1,6], which captures the geometry of input models quite well, but the output can be 10 times larger than the input. An alternative is to remesh the models using base mesh's connectivity [1,2,8]. However, for features not reflected by base mesh, it also needs a rather dense subdivision. Another technique is *adaptive-smoothing and refinement* in [3,16]. Compared with the former two, it can balance output's geometric detail and connectivity complexity. However, it is a *passive* method, i.e. it passively refines compatible mesh when large approximation errors between compatible meshes and input meshes happen. It needs to check errors of all input meshes' vertices and edge midpoints, so the process is slow and its parameter is hard to set. Moreover, to capture salient features, vertices much more than necessary are added.

In this paper, we propose a novel scheme for cross-parameterization. Our scheme is local and indirect, because we prefer cross-parameterization with low distortion and high efficiency.

First, we introduce a novel framework to split surfaces into compatible triangle patches. Compared with previous methods, it is robust and can obtain meaningful segmentations. More importantly, it greatly accelerates the speed, which is always a bottleneck of local methods.

After necessary adjustment, then comes our new cross-parameterization algorithm. Unlike previous passive meth-

ods, it is an *active* one, i.e. it actively updates input meshes' connectivity in key areas to avoid distortion, rather than passively repairing low-quality cross-parameterization after distortion happens. In detail, we first propose two criteria to decide which vertices on input meshes are important for a good parameterization, and process them carefully to make sure that they will be well approximated by compatible meshes. Then a high-quality compatible remeshing is obtained directly, without the need of the slow relaxation and refinement.

## 2 Scheme Overview

### 2.1 Definitions

The following terms are used in algorithm description:

◊ Algorithm inputs are two meshes  $M_s$  and  $M_t$ , with vertex sets  $\mathbf{V}_s = \{V_i^s\}_{1 \leq i \leq N_s}$ ,  $\mathbf{V}_t = \{V_i^t\}_{1 \leq i \leq N_t}$ . The initial  $C$  vertices of  $\mathbf{V}_s$  and  $\mathbf{V}_t$  are feature vertex pairs selected by the user.

◊ A *patch layout*  $\mathbf{P}$  is a partition of mesh into simply connected, non-overlapping *patch* where the boundary of each patch is constructed by non-intersecting edge *path* connecting feature vertices.  $\mathbf{P}$  is *triangular* if all its patches' boundary has three paths, and each patch is marked as  $P_{ijk}$ , where  $V_i, V_j, V_k$  are three feature vertices. Path connecting  $V_i, V_j$  is marked as  $p_{ij}$ .

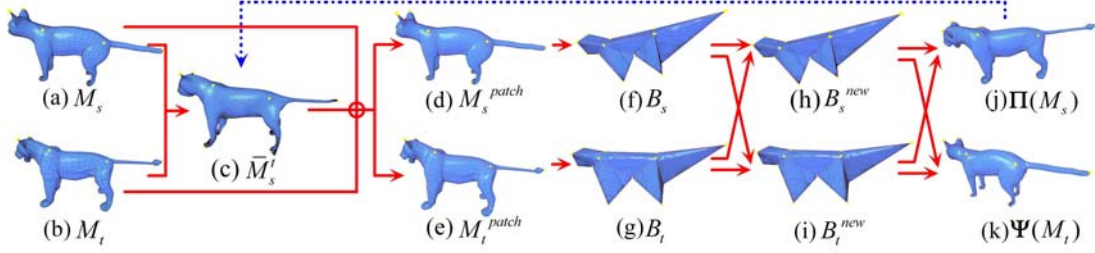
◊ Patch layouts  $\mathbf{P}_s, \mathbf{P}_t$  are *compatible*, if each path  $p_{ij}^s$  of  $P_s$  matches path  $p_{ij}^t$  of  $P_t$ , and vice versa. Such pairs of paths are called *compatible paths*.

◊ Patch  $P_{ijk}$ 's *base triangle* is the planar triangle formed by  $V_i, V_j, V_k$ . A mesh's *base mesh*  $B$  is the union of all its patches' base triangles.

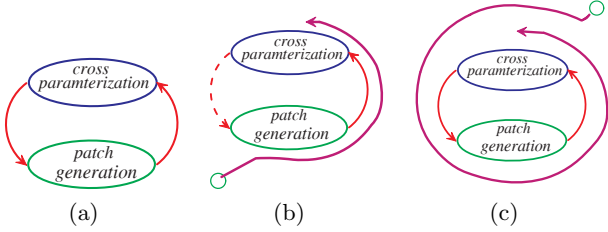
◊ A *cross-parameterization* or *inter-surface mapping* is a bijective mapping  $\Pi: M_s \rightarrow M_t$ . *Compatible mesh*  $\Pi(M_s)$  is of  $M_s$ 's connectivity and  $M_t$ 's geometry.  $\Pi$  is a composition of three *sub-mappings*:  $\Pi_s: M_s \rightarrow B_s$ ,  $\Pi_{ts}: B_s \rightarrow B_t$ ,  $\Pi_t: M_t \rightarrow B_t$ , and  $\Pi = \Pi_t^{-1} \cdot \Pi_{ts} \cdot \Pi_s$ . Similarly, we can define mapping  $\Psi: M_t \rightarrow M_s$  and compatible mesh  $\Psi(M_t)$  with  $M_t$ 's connectivity and  $M_s$ 's geometry. Generally  $\Psi \neq \Pi^{-1}$ .

### 2.2 Motivation

The motivation of our new scheme is the fact that patch-generation and cross-parameterization are two interdependent problems (Fig 2(a)). If surfaces are already split into valid and meaningful compatible patches, cross-parameterization can be solved with method such as [3]. Contrarily, if a good cross-parameterization  $\Pi$  is on hand, meaningful compatible patches can easily be constructed by first building patches on  $M_s$  and then mapping them on  $M_t$  using  $\Pi$ . To get a good cross-parameterization, this interdependence should be utilized. However, all previous approaches work as Fig 2(b), i.e. cross-parameterization's



**Fig. 1** Flow chart of our scheme. Using input meshes (a) and (b), initial compatible mesh (c) is computed. Guided by path hypotheses obtained from (c), compatible patch layouts (d)(e) and base meshes (f)(g) are obtained. Then, using active parameterization algorithm, base meshes are updated to (h)(i), and compatible meshes (j)(k) are calculated as output. Here, (h)(i)'s updates to (f)(g) are not obvious, though updates do happen in tail base meshes. For more obvious updates, please refer Fig 8(a)(c). The dashed line is the optional further iteration, which can produce better results for complex models.



**Fig. 2** (a) Interdependent relations between patch generation and cross-parameterization; (b) is traditional scheme, which does not fully utilize these relations; (c) is our scheme.

potential assistance to patch-generation is ignored, resulting in slow and not robust patch-generation and thus a poor cross-parameterization. Fig 2(c) shows our scheme. We first compute a low-quality cross-parameterization in form of a rough compatible mesh; using it as initial hypothesis, we can get compatible patch layouts robustly and rapidly; based on these patch layouts, a cross-parameterization is then computed, which is used as a better hypothesis. The procedure repeats until a high-quality cross-parameterization is obtained. Regarding cross-parameterization as a non-linear optimization problem, the coarse compatible mesh can be viewed as a good initial solution quite near the optimal solution, which results in our scheme's fast convergence: only  $1\frac{1}{2}$  cycles are enough to get good results in most cases, though more cycles are useful for complex models. Our scheme inherently is a linear coarse-to-fine strategy, more efficient than the nonlinear one of [4].

### 2.3 Algorithm Stages

Flowchart of our novel cross-parameterization scheme is shown in Fig 1, which has main stages as follows:

**Hypothesis-based patch generation:** First, an initial compatible mesh  $\bar{M}_s^t$  is constructed using least-squares method. Based on it, hypotheses to ideal pathes are then computed. With the guidance of these hypotheses, compatible patch layouts  $\mathbf{P}_s$  and  $\mathbf{P}_t$  are calculated effectively and robustly.

**Active cross-parameterization:** For all vertices in  $M_t$ , we estimate how well they can be approximated by  $\Pi(M_s)$ , and detect vertices with too large approximation error. Moreover, using the salient detection algorithm similar to [14], salient vertices of  $M_t$  determining its profile are also detected. Then,  $B_s$  and  $M_s$  are simultaneously updated around these vertices, resulting in a better cross-parameterization  $\Pi^{new}$  and  $M_s$ 's compatible mesh

$\Pi^{new}(M_s)$ . Similarly,  $\Psi$  and  $\Psi(M_t)$  are computed, by updating  $M_t$  and  $B_t$ . We use  $\Psi(M_t)$  instead of  $\Pi^{-1}(M_t)$  as  $M_t$ 's compatible mesh because when  $|\mathbf{V}_s| > |\mathbf{V}_t|$ ,  $\Psi(M_t)$  usually has fewer elements than  $\Pi^{-1}(M_t)$  does.

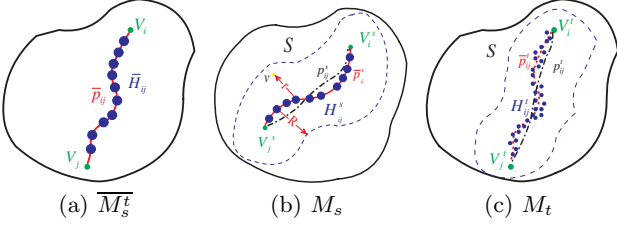
## 3 Hypothesis-based Patch Generation

### 3.1 Constructing Initial Compatible Mesh

The initial compatible mesh serves as a coarse guess for final output, which is solved using the least-squares method. As introduced in [15], we solve the new positions of  $M_s$  ( $\mathbf{V}'_d = [v'_{1d}, v'_{2d}, \dots, v'_{N_d}]^T, d \in \{x, y, z\}$ ) separately by forcing its feature vertices to have identical position with  $M_t$ 's, using a quadratic energy minimization:  $\min_{\mathbf{V}'_d} \|\mathbf{L}\mathbf{V}'_d\|^2 + \sum_{1 \leq i \leq C} w_i^2 |v'_{id} - v_{id}^t|^2$ , where  $\mathbf{L}$  is the uniform Laplacian coefficient matrix of  $M_s$ ;  $w_i$  is the belief weight of  $(V_i^s, V_i^t)$  marker-point pair. This optimization can be computed by a linear system:  $\begin{bmatrix} \mathbf{L} \\ \mathbf{W} \end{bmatrix} \mathbf{V}'_d = \begin{bmatrix} \mathbf{0} \\ \mathbf{w}\mathbf{V}_d^t \end{bmatrix}$ . Its solution is denoted as  $M_{lsm}^s$ , usually whose geometry still differs a lot from  $M_t$ 's. To improve it, we use the simple approximation algorithm introduced in [5] to project  $M_{lsm}^s$  on  $M_t$ . Then an initial compatible mesh  $\bar{M}_s^t$  is achieved.

### 3.2 Adding Paths

With  $\bar{M}_s^t$ , we now compute compatible patches for  $M_s$  and  $M_t$ , which is a four-step scheme. First, for every two feature vertices, a path hypothesis is computed; then compatible edge paths are added due to hypotheses;



**Fig. 3**  $\mathcal{H}_{ij}^s$ ,  $\mathcal{H}_{ij}^t$  (blue points) are hypotheses of  $p_{ij}^s$  and  $p_{ij}^t$  (black dashed curves).  $S$  (blue dashed curve) is propagated vertices set,  $\forall v \in S$ ,  $|v - \mathcal{H}| \leq R$ . The weighted Dijkstra search is limited in  $S$ .  $\forall v \in S$ , its punishment is  $\exp(\lambda r)$ , where  $r = |v - \mathcal{H}|$  and  $\lambda$  is computed by letting  $\exp(\lambda R) = 5$ . Notice that  $p_{ij}^t$  is just position on  $M_t$ , i.e. it is a virtual path.

next, compatible face paths are added similarly; finally, patch adjustment is implemented to improve partitions.

**Compute path hypothesis** For each pair of feature vertices in  $M_s^t$ , a shortest path  $\overline{p_{ij}}$  is computed using Dijkstra search and the union of its vertices makes the set  $\overline{\mathcal{H}}_{ij}$  (Fig 3(a)). As  $M_s$  is compatible to  $\overline{M}_s^t$ , there exists a corresponding path  $\overline{p_{ij}}^s$  on  $M_s$ , and the union of its vertices is  $\mathcal{H}_{ij}^s$  (Fig 3(b)).  $M_t$ 's connectivity is different from  $\overline{M}_s^t$ , so a direct path correspondence is not available. But as they have similar geometry, for each vertex in  $\overline{\mathcal{H}}_{ij}$ , there exist vertices in  $M_t$  whose distances from it are within a given threshold. The union of such vertices makes the counterpart of  $\overline{\mathcal{H}}_{ij}$  on  $M_t$ , represented as  $\mathcal{H}_{ij}^t$  (Fig 3(c)). The sets  $\mathcal{H}_{ij}^s$  and  $\mathcal{H}_{ij}^t$  are called the *hypothesis* for actual surface paths  $p_{ij}^s$  and  $p_{ij}^t$ . This denomination is based on the fact that these vertices are on or near the initial compatible paths of  $M_s$  and  $\overline{M}_s^t$ , so they are most likely to be on the compatible paths  $p_{ij}^s$  and  $p_{ij}^t$ . So  $\mathcal{H}_{ij}^s$  and  $\mathcal{H}_{ij}^t$  can be considered as a good guess to ideal positions of  $p_{ij}^s$  and  $p_{ij}^t$ .

**Add edge paths** This step adds matching paths on the edge graphs of  $M_s$  and  $M_t$ , with the guidance of  $\mathcal{H}_{ij}^s$  and  $\mathcal{H}_{ij}^t$  (Algorithm 1).

We use  $\overline{M}_s^t$ 's path length as the criterion to decide the order of path adding. Unlike [3,4], the models need not be scaled to similar sizes. In our experiment, we find this criterion can give more reasonable adding order.

For any pair of paths to be added, we first compute *punishment fields* around their hypotheses using a limited vertex propagation according to exponential distribution (Fig 3(b)(c)). The field's construction is based on the fact that the farther is a vertex from a path's hypothesis, the less likely it will be on the path, so it is given larger punishment. In weighted shortest path algorithm, this punishment field can pull path closer to hypothesis.

Then, we construct compatible paths using weighted Dijkstra algorithm. If neither  $V_a, V_b$  are on paths added before, edge  $(V_a, V_b)$ 's weight is set as the product of two end-vertices' punishments; otherwise, is set as  $\infty$ .

---

**Algorithm 1:** EdgePathAdding guided by  $\overline{M}_s^t$ 


---

**Data:**  $\overline{M}_s^t(\{\overline{p_{ij}}\}, \{\overline{\mathcal{H}}_{ij}\})$ ,  $M_s(\{\mathcal{H}_{ij}^s\})$ ,  $M_t(\{\mathcal{H}_{ij}^t\})$

**Result:** Compatible edge path sets  $EP_s$  and  $EP_t$

**begin**

$ST \leftarrow \{\overline{p_{ij}}\}$ ,  $EP_s \leftarrow \emptyset$ ,  $EP_t \leftarrow \emptyset$

**while**  $ST \neq \emptyset$  **do**

$\overline{p_{mn}} \leftarrow ST.removeShortestPath()$

/\* generate punishment fields \*/

$M_s.punishmentFieldGen(\mathcal{H}_{mn}^s)$

$M_t.punishmentFieldGen(\mathcal{H}_{mn}^t)$

/\* compute path  $p_{mn}^s$ ,  $p_{mn}^t$  using weighted Dijkstra algorithm \*/

$p_{mn}^s \leftarrow M_s.minEdgePath(m, n)$

$p_{mn}^t \leftarrow M_t.minEdgePath(m, n)$

**if**  $CyclicalOrderAdjust(p_{mn}^s, p_{mn}^t)$  **then**

**if**  $NonSwirling(p_{mn}^s, p_{mn}^t)$  **then**

**if**  $NonBlocking(p_{mn}^s, p_{mn}^t)$  **then**

$EP_s \leftarrow p_{mn}^s$

$EP_t \leftarrow p_{mn}^t$

**end**

---

As Dijkstra search is limited in propagated vertices set  $S$  (Fig 3(b)(c)), with  $|S| \ll |M|$ , shortest path generation is rather fast. Compared with previous approaches, our patch-generation procedure can be at least 4 times faster. On large models, its speed advantage is more obvious.

The obtained compatible paths must be checked to judge whether they are valid, as introduced in [2–4]. Our check processes are as follows:

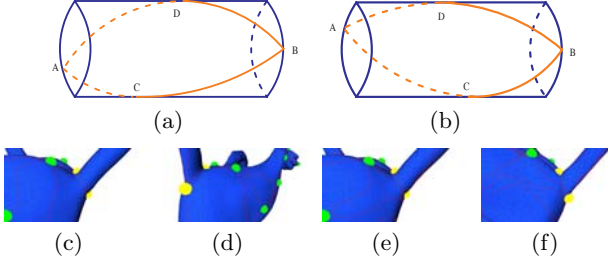
**Intersection:** New paths must not intersect paths already added. Weighted Dijkstra search guarantees it, for edges intersecting with previous paths are assigned  $\infty$  punishment.

**Cyclical Order:** New paths should have identical cyclical order around feature vertices. Procedure  $CyclicalOrderAdjust$  checks it using the same method as in [3]. If the condition is not met, the procedure recomputes the paths, starting in the correct segment.

**Blocking:** The new paths should never block future paths, which is checked by procedure  $NonBlocking$  similar to [3].

**Swirls:** As defined in [2,4], swirl is a meaningless geometric configuration in which paths between feature vertices will take unnecessary long routes around existing paths. [2] points out that swirl cannot be fixed with local relaxation. Previous solutions to this problem are either complex or not robust. [2] uses a complicated parametrical trace method to detect swirl. [4] avoids swirl by delaying paths against two heuristics, which need extra calculations and cannot guarantee success in many cases.

We cope with swirl in a more natural and effective way. We notice that swirl happens when paths' lengths are much longer than their normal lengths. So we first compute a reference length for each path, and then check

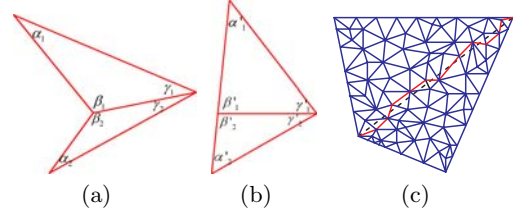


**Fig. 4** Both (a) and (b) have two *local shortest paths*  $ACB$  and  $ADB$ .  $A$ 's location change makes them choose different branches as shortest path,  $ACB$  for (a) and  $ADB$  for (b). In (c) and (d), branch mismatch happens between yellow vertices, and the procedure of compatible patch generation fails. Our scheme avoids mismatch and gets correct results (e)(f).

whether the length of path obtained by weighted Dijkstra algorithm deviates too much from it. In our scheme, path's reference length is easy to obtain. As  $\overline{M}_s^t$ 's geometry is similar to  $M_t$ 's, its path  $\overline{p}_{mn}$  should be of similar length as its counterpart on  $M_t$ . So  $length(\overline{p}_{mn})$  can be considered as reference length for  $p_{mn}^t$ . For  $p_{mn}^s$ , its hypothesis path  $\overline{p}_{mn}^s$ 's length is a good reference. Our strategy to avoid swirl is: if  $p_{mn}^s$  and  $p_{mn}^t$  are so long that  $length(p_{mn}^s) > k \cdot length(\overline{p}_{mn}^s)$  and  $length(p_{mn}^t) > k \cdot length(\overline{p}_{mn})$ , swirl happens and the pair of paths are delayed to add. In theory,  $k$  should be 1, but hypothesis's error with ideal path and mesh's discrete sampling mode require a bigger  $k$ . We find  $k = 1.5$  is a good choice. Fig 13 (e)(f) show that our method can avoid swirl and obtain reasonable partition, compared with [3].

**Circle Branch Mismatch:** This is an important condition ignored by previous approaches. We first introduce a new definition: *local shortest path*, which is a path shorter than all the paths deviating a little from it. If there are more than one local shortest path connecting two vertices, a small deviation of end-vertices' position can cause the unstableness of shortest path's location (Fig 4(a)(b)). This unstableness will bring the serious problem of branch mismatch, i.e. compatible paths of  $M_s$  and  $M_t$  belong to different circle branches. This mismatch can cause failure in previous approaches (Fig 4(c)(d)). (The essential reason for this failure is that *UnBlocking* may omit some invalid paths in the early step of path adding procedure.) Our scheme can completely avoid this problem, because the two path hypotheses correspond to the same path on  $\overline{M}_s^t$ , they are inherently of the same circle branch (Fig 4(e)(f)).

**Add face paths** Algorithm 1 terminates when no more matching paths can be added on edge graphs. Therefore, we have to trace matching paths on meshes' face graph instead. We use the method similar to [3], the only difference is that path hypotheses are used to guide face path adding, as introduced above.



**Fig. 5** (a) non-convex quadrangle (b) convex unfolding with  $\alpha'_1 = \pi \frac{\alpha_1}{\alpha_1 + \alpha_2}$ ,  $\alpha'_2 = \pi \frac{\alpha_2}{\alpha_1 + \alpha_2}$ ;  $\beta'_1 = (\pi - \alpha'_1) \frac{\beta_1}{\beta_1 + \gamma_1}$ ,  $\beta'_2 = (\pi - \alpha'_2) \frac{\beta_2}{\beta_2 + \gamma_2}$ ;  $\gamma'_1 = (\pi - \alpha'_1) \frac{\gamma_1}{\beta_1 + \gamma_1}$ ,  $\gamma'_2 = (\pi - \alpha'_2) \frac{\gamma_2}{\beta_2 + \gamma_2}$  (c) The red curve is the shortest path on parameter domain.

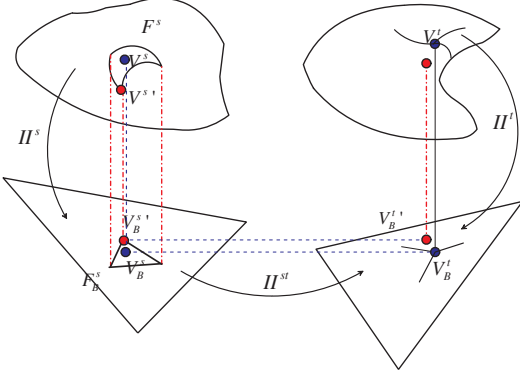
### 3.3 Patch Adjustment

When all possible paths have been added, two input surfaces are partitioned into compatible triangle patches. However, as pointed out in [3], to generate a globally continuous, low-distortion parameterization, we need to relocate vertices from one patch to another. We implement it using our new patch adjustment algorithm. We first compute the base triangles for two adjacent patches sharing a path  $p$ . Then we unfold the two base triangles to the same plane and get a quadrangle. Since we need convex quadrangle in following steps, we scale the concave quadrangle as Fig 5(a)(b). Then we map the exterior bounding paths of the two patches to corresponding edges of the quadrangle using length-size parameterization. To parameterize the interior vertices, the stretch-minimizing scheme proposed by [11] is used. On the obtained quadrangle parameterization domain, we find the shortest path connecting  $p$ 's two end-vertices (Fig 5(c)). Mapping this path back to the mesh domain, we can get a new path  $p'$  which can replace  $p$  as the new patch boundary. This shortest path algorithm in parameter domain needs no extra vertices and can adjust the unreasonable path remarkably.

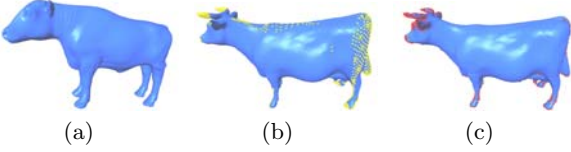
Our coarse-adjustment is an iterative method. In each cycle, it implements the above operation on every two adjacent patches. The iteration uses Gauss-Seidel strategy, i.e. once the new paths are computed, they are immediately used in following adjustments. Compared with the Jacobi strategy, which uses paths in last cycle for current cycle's adjustment, our method is faster. By the way, we notice that in [12], a method similar to ours is used. However, because extra Steiner vertices are needed, it is not suitable for reiterative adjustment. Fig 13(f)(g) show the effectiveness of our patch adjustment strategy.

## 4 Active Cross-Parameterization

Using the minimum-stretch parameterization proposed in [11], we can construct base meshes  $B_s$  and  $B_t$ . Then initial cross-parameterization  $\Pi$  can be constructed, where  $\Pi = \Pi_t^{-1} \cdot \Pi_{ts} \cdot \Pi_s$ .  $\Pi$  can approximate ideal inter-surface mapping well in most cases. However, if  $M_t$  has complex



**Fig. 6** Variables used in *active cross-parameterization*.



**Fig. 7** (b) shows *high-stretch vertices* in yellow, which are features not available in (a), e.g. the udders, horns and tail. (c) shows *salient vertices* essential for mesh's profile.

features not available in  $M_s$ ,  $\Pi$  may lose these features. Moreover, some profile vertices are not captured by  $\Pi$ . So the improvement to  $\Pi$  is necessary. [3] updates it using *adaptive-smoothing* and *refinement*. As introduced in Section 1, it has many disadvantages, which root from its strategy of treating all vertices with the same importance during remeshing. In our method, we distinguish vertices that are important for feature capture, profile keeping and error reducing, and then pay them special attention. We define two types of special vertices: high-stretch vertices and salient vertices.

#### 4.1 High-stretch Vertices

Stretch is a metric used to measure distortion of unit length vector  $\mathbf{u}$  during affine mapping  $f$  from 2D to 3D, e.g.  $\Pi_s^{-1}$  or  $\Pi_t^{-1}$  in our cases. It has many forms and we use the  $L^2$  metric proposed by [10], i.e.  $L^2(T) = \sqrt{\frac{\Gamma^2 + \gamma^2}{2}}$ , where  $\Gamma$  and  $\gamma$  are the largest and smallest singular values of  $f$ 's Jacobian matrix. This is  $\mathbf{u}$ 's average distortion in triangle  $T$ . Then we define  $\mathbf{u}$ 's average distortion in the neighborhood of vertex  $V$  as  $L^2(V) = \sqrt{\frac{\sum_{T_i \in 1-ring(V)} (L^2(T_i))^2 Area_{3D}(T_i)}{\sum_{T_i \in 1-ring(V)} Area_{3D}(T_i)}}$ . For every position  $P$  in certain triangle face  $F_{ABC}$  we can compute  $L^2(P)$  using the  $A, B, C$ 's stretches:  $L^2(P) = w_A \cdot L^2(A) + w_B \cdot L^2(B) + w_C \cdot L^2(C)$ , where  $\{w_A, w_B, w_C\}$  are the barycentric coordinates of  $P$  relative to  $F_{ABC}$ . Then for  $\mathbf{u} = \overrightarrow{PQ}$ ,  $\|f(\mathbf{u})\|$  can be estimated by  $L_f^2(P) \cdot \|\mathbf{u}\|$ .

To get a high-quality cross-parameterization  $\Pi$ ,  $M_t$ 's vertices that are not approximated well by  $\Pi(M_s)$  should be treated carefully. As shown in Fig 6, two series of

vertices are given:  $V^s \xrightarrow{\Pi_s} V_B^s \xrightarrow{\Pi_{ts}} V_B^t \xrightarrow{\Pi_t^{-1}} V^t$  and  $V^{s'} \xrightarrow{\Pi_s} V_B^{s'} \xrightarrow{\Pi_{ts}} V_B^{t'} \xrightarrow{\Pi_t^{-1}} V^{t'}$ , where  $V_B^t$  is a mesh vertex on  $B_t$ , and  $V_B^{s'}$  is the vertex closest to  $V_B^s$  on  $B_s$ . Here only  $V_B^t, V^t, V_B^{s'}$  and  $V^{s'}$  are mesh vertices, the others are all positions on the meshes.

Now we need to estimate the error between  $\Pi$  and  $\Pi_{ideal}$ . At  $V^t$ , the error is defined as  $V^t$ 's stretch, which can be estimated as:

$$\begin{aligned} & |\Pi(V^s) - V^t| \\ &= |\Pi(V^s) - \Pi_{ideal}(V^s)| \\ &\approx |\Pi(V^{s'}) - V^t| \\ &= |\Pi_t^{-1}(\Pi_{ts}(\Pi_s(V^{s'} - V^s)))| \\ &\approx stretch_t(V_B^t) \cdot \lambda \cdot \frac{1}{stretch_s(V_B^s)} |V^{s'} - V^s| \end{aligned} \quad (1)$$

where  $stretch_t(V_B^t)$  is the stretch of  $\Pi_t^{-1}$  at  $V_B^t$ , and  $stretch_s(V_B^s)$  is the stretch of  $\Pi_s^{-1}$  at  $V_B^s$ ,  $\lambda$  is the stretch of unit length vector during affine mapping  $\Pi_{ts}$ , which is constant for patch inner vertices because  $\Pi_{ts}$  keeps constant within a patch. The last approximation uses  $\|f(\mathbf{u})\|$ 's estimation introduced above.

The physical meaning of Eq 1 is quite interesting: to reduce the error, we need to reduce  $\frac{stretch_t(V_B^t)}{stretch_s(V_B^s)}$  and  $|V^{s'} - V^s|$ . The first form is the feature match term, which is large only when  $M_t$ 's feature is not available in  $M_s$ . The second term is the sampling term. If  $M_s$  is dense enough, then  $|V^{s'} - V^s|$  can be small enough to make approximation error  $|\Pi(V^s) - V^t|$  very small, even when feature match term is large.

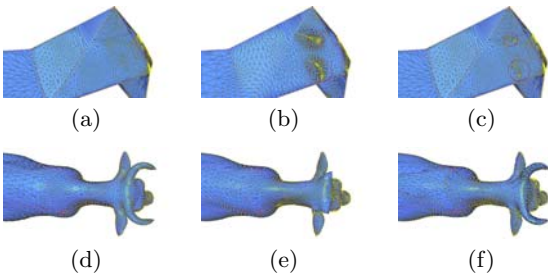
To improve the cross-parameterization's quality, we limit all  $M_t$ 's vertices' stretch under certain threshold. To make the threshold independent of models' scale, we compute it using the *box plot test* in statistics. For vertices with high-stretch (Fig 7(b)), i.e. stretch larger than threshold, we need to reduce their approximation error. From Eq 1, we should reduce the sampling term, because the feature match term is decided by models themselves, which is hard to adjust. The simplest way to reduce sampling term is to upsample  $M_s$  near the position of high-stretch vertices. It is an iterative procedure: suppose  $V^t$  in Fig 6 is a high-stretch vertex, then one edge of  $F_B^s$  is split. The split is done simultaneously on  $M_s$  and  $B_s$ . The iteration stops when all vertices have stretches less than given threshold.

#### 4.2 Salient Vertices

Salient vertices, as introduced in [14], are a cluster of surface vertices that can approximate the whole surface with local patches associated with them. These vertices should be accurately reached, otherwise model's profile will have obvious collapse.

$M_s/M_t$	input sizes(#v)	#feature vertices	$T_{previous}$	$T_{ours}$	$\Pi(M_s)/\Psi(M_t)$ sizes(#v)
cat/lion <sup>1</sup>	21617/14996	63	>2hours	20min	21834/15285
cat/lion <sup>1*</sup>	21617/14996	63	FAILED	21min	21878/15300
cat/lion <sup>2</sup>	7207/5000	63	15min	7min	7414/5212
cat/lion <sup>3</sup>	7207/5000	14	72sec	75sec	7383/5028
cat/lion <sup>3*</sup>	7207/5000	14	FAILED	72sec	7390/5043
run cat/lion	7207/5000	17	2min	2min	7286/5136
camel/horse	9770/19851	35	25min	10min	10826/20736
bull/buffalo	12941/8708	17	7min	4min	13392/9013
torus/star	16815/5192	20	12min	6min	16846/8408
Egea/David	8268/10113	27	12min	5min	8417/10292

**Table 1** Cross-parameterization statistics. The only difference between models marked with and without \* is the small deviation of feature vertices’ locations.



**Fig. 8** Active parameterization: (a)initial base mesh  $B_s$  (b)base mesh  $B_t$  (c)updated base mesh  $B_s^{new}$  (d) $M_t$  (e)compatible mesh  $\Pi(M_s)$  (f)compatible mesh  $\Pi^{new}(M_s)$ .

We first compute salient vertices on  $M_t$ , then find  $M_s$ ’s vertices which can map onto salient vertices by slightly changing their positions. If such vertices are not available, we add new vertices in  $M_s$ .

To compute the salient vertices, we adopt technique similar to [14], except that our local patch is sphere patch instead of quadric patch. Quadric patch is more accurate, but it requires a dense enough mesh for linear system to compute quadric coefficients. So for models with lower sample rate, it is not robust. Our sphere patch is simple but more robust. We use [13] to compute mean curvature normal  $\mathbf{K}(V)$  at  $V$ , and define  $\mathbf{n} = \frac{\mathbf{K}(V)}{\|\mathbf{K}(V)\|}$  as unit normal, and  $R = \frac{1}{\|\mathbf{K}(V)\|}$  as curvature radius. Neighborhood of  $V$  can be approximated by a sphere with radius  $R$ , centering at  $V - R \cdot \mathbf{n}$ . To judge whether vertices nearby  $V$  are in its patch, a scale independent threshold, e.g.  $10^{-4}$  bounding box diagonal length, is used. Fig 7(c) shows detected salient vertices.

After  $M_t$ ’s salient vertices are detected, we then find or create vertices on  $M_s$  to make salient vertices have exact correspondence on  $\Pi(M_s)$ . Suppose  $V^t$  in Fig 6 is a salient vertex. Our method is a two-step procedure. First, we move  $V_B^s$ ’s closest vertex  $V_B^{s'}$  to its location. The movement should be valid, i.e. 1-ring triangles of  $V_B^{s'}$  should keep their normal direction unchanged. If not, the movement is discarded. Moreover, if  $V_B^{s'}$  is the closest vertex shared by several  $\{V_{B_i}^s\}$ , we sort  $\{V_{B_i}^s\}$  ac-

ording to their distance to  $V_B^{s'}$  in ascending order, and move  $V_B^{s'}$  to the first valid position. After  $V_B^{s'}$ ’s movement,  $V^{s'}$  also moves according to barycentric coordinate. After movement step, there may still exist salient vertices having no exact correspondence. For these vertices, we add new vertices in  $M_s$  and  $B_s$  to approximate them exactly. If  $V^t$  is a such vertex, a new vertex is added in  $B_s$  at  $V_B^s$  and its counterpart is added in  $M_s$  at  $V^s$  (Fig 6).

After the treatment to high-stretch and salient vertices,  $M_s$  and  $B_s$  are updated to  $M_s^{new}$  and  $B_s^{new}$ , and sub-mappings are also updated. Then new inter-surface mapping  $\Pi^{new} = (\Pi_t^{new})^{-1} \cdot \Pi_{t_s}^{new} \cdot \Pi_s^{new}$  is obtained, which can approximate ideal mapping quite well. Fig 8 shows updated base mesh and the final compatible mesh, and we can see features and profiles lost in parameterization without update (Fig 8(e)) are all captured by our active cross-parameterization (Fig 8(f)). Another mapping  $\Psi^{new}$  can be updated from  $\Psi$  in the same way. With these updated mappings, high-quality compatible remeshings  $\Pi^{new}(M_s)$  and  $\Psi^{new}(M_t)$  are obtained.

## 5 Experimental Results

We test our scheme on a series of benchmark model pairs (Fig 9-13). In each pair, one model has features not available in the other. Our algorithm can give quite good results and the output compatible meshes’ sizes are well controlled (Table 4.2). The only exception is Fig 10, in which  $M_t$  is much more complex than  $M_s$ , so  $\Pi(M_s)$  is large to capture  $M_t$ ’s features. However, increasing the threshold in our active cross-parameterization can produce  $\Pi'(M_s)$  (Fig 10(e)) with fewer vertices ( $\#v = 6090$ ), and the detail loss is still small.

We compare our method’s speed and robustness with [3]. The experiments run on P4 1.8GHz/1GRAM laptop. Shown in Table 4.2, our method’s advantage is obvious, especially to large models with many feature vertices, where [3] may fail with branch mismatch mistakes as mentioned in Section 3.2.

The effectiveness of our method is not a hit-and-miss. In previous works, using heuristic techniques, only local information about the mapping is known. The lack of global guidance makes path adding and patch generation a great difficulty, so that algorithm's robustness and effectiveness can hardly be guaranteed. The most important improvement of our method is that it makes use of the prior knowledge which comes from a coarse correspondence computing with a simple real-time method. Though this prior knowledge is far from perfect and many correspondence mistakes exist, it gives an overall guidance about the way in which the bijective mapping between surfaces should be constructed. The introduction of global information results in our method's advantages in robustness and effectiveness.

We also apply our algorithm to *direct pose transfer*, i.e. directly transferring  $M_s$ 's pose onto  $M_t$ . First,  $\Psi(M_t)$  is computed with our method. Using this compatible mesh as intermediate,  $M_s$ 's pose can be transferred to  $M_t$ , as proposed in [18]. Results are shown in Fig 14.

## 6 Conclusions and Future works

In this paper, we introduce a novel scheme for a fast, robust and high-quality cross-parameterization. Our hypothesis-based patch generation greatly accelerates algorithm's speed and can obtain reasonable mesh partition robustly. Our active cross-parameterization adds necessary vertices in necessary locations, making a better trade-off between result's complexity and precision.

The problem to build one-to-one mapping between two meshes of different connectivity and geometry is far from being solved. The correspondence between meshes with not-zero genus is still a great challenge. Though [6] gives an elegant and uniform framework to handle genus problems, it can not utilize geometry properties. Further work is necessary to unify [6]'s framework with ours scheme.

**Acknowledgements:** This research work is supported by the National Natural Science Foundation of China (NSFC No. 60675012).

## References

1. Lee A. W., Dobkin D., Sweldens W., and Schroder P.: Multiresolution mesh morphing. In *SIGGRAPH* (1999) 343-350
2. Praun E., Sweldens W., Schroder P.: Consistent mesh parameterizations. In *SIGGRAPH* (2001) 179-184
3. Kravevov V., Sheffer A.: Cross-parameterization and compatible remeshing of 3D models. In *SIGGRAPH* (2004) 861-869
4. Schreiner J., Asirvatham A., Praun E., and Hoppe H.: Inter-surface mapping. In *SIGGRAPH* (2004) 870-877
5. L. Zhang, L. Liu, Z. Ji, G. Wang: Manifold parameterization. In *CGI* (2006) 160-171
6. C. Carner, M. Jin, X. Gu, and H. Qin: Topology-driven surface mappings with robust feature alignment. In *Visualization* (2005) 543-550
7. Gotsman C., Gu X., and Sheffer A.: Fundamentals of spherical parameterization for 3D meshes. In *SIGGRAPH* (2003) 358-363
8. T. Michikawa, T. Kanai, M. Fujita, H. Chiyokura: Multiresolution interpolation meshes. In *PG* (2001) 60-69
9. Praun E., Hoppe H.: Spherical parametrization and remeshing. In *TOG* 22(3) (2003) 340-349
10. Sander P. V., Snyder J., Gortler S. J., and Hoppe, H.: Texture mapping progressive meshes. In *SIGGRAPH* (2001) 409-416
11. S. Yoshizawa, A. Belyaev, and H.P. Seidel: A fast and simple stretch-minimizing mesh parameterization. In *SMI* (2004) 200-208
12. Kravevov V., and Sheffer A.: Template-based mesh completion. In *SGP* (2005) 13-22
13. Meyer M., Desbrun M., Schröder P., and Barr A. H.: Discrete differential geometry operators for triangulated 2-manifolds. In *VisMath* (2002)
14. Gal R. and Cohen-Or D.: Salient geometric features for partial shape matching and similarity. In *TOG* 25(1) (2006) 130-150
15. Sorkine O., and Cohen-Or D.: Least-squares meshes. In *SMI* (2004) 191-199
16. Y.T. Chang, B.Y. Chen, W.C. Luo and J.B. Huang: Skeleton-driven animation transfer based on consistent volume parameterization. In *CGI* (2006) 78-89
17. Floater M.S.: Mean value coordinates. In *CAGD* 20(1) (2003) 19-27
18. H.Y. Wu et al.: Model transduction. In *IEEE Trans. on Visualization and Computer Graphics* (submitted) (2006)

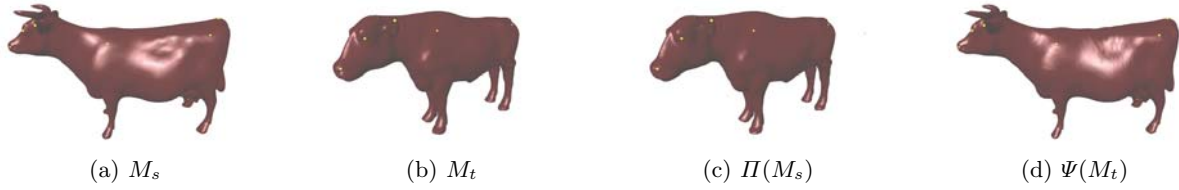


Fig. 9 Compatible remodeling between *bull* and *buffalo*.

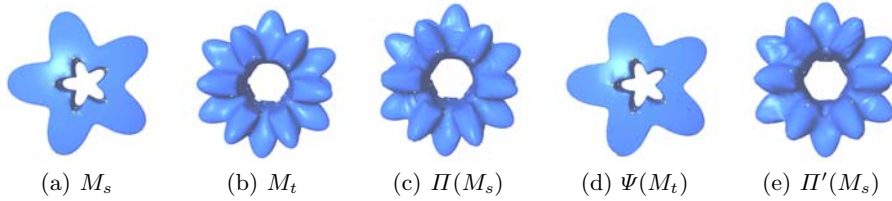


Fig. 10 Compatible remodeling between *trim-star* and *bumpy-torus*.

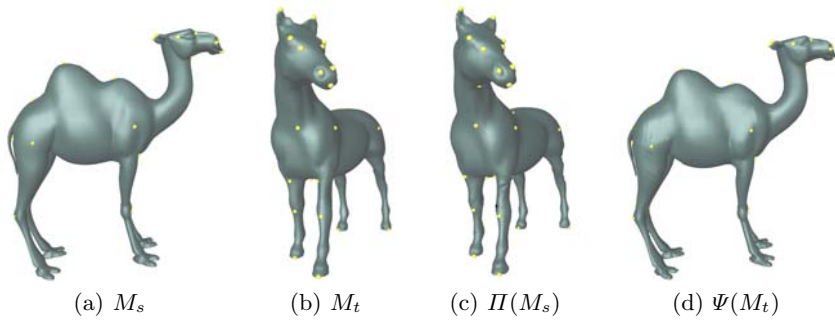


Fig. 11 Compatible remodeling between *camel* and *horse*.

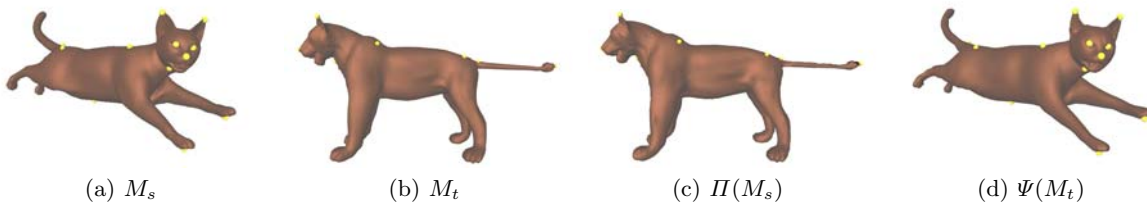


Fig. 12 Compatible remodeling between *running cat* and *lion*.

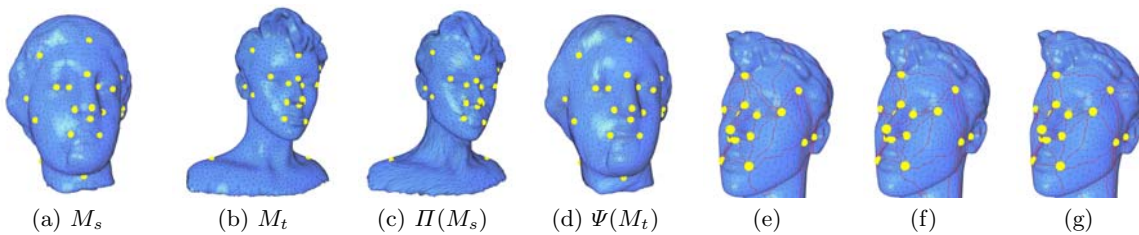


Fig. 13 Compatible remodeling between *Egea* and *David*((a)-(d)). (e)(f)(g) are all partitions of  $M_t$ . (e) is obtained by using method in [3]. (f)(g) are obtained with our scheme: (f) is before patch-adjustment, (g) is after patch-adjustment.

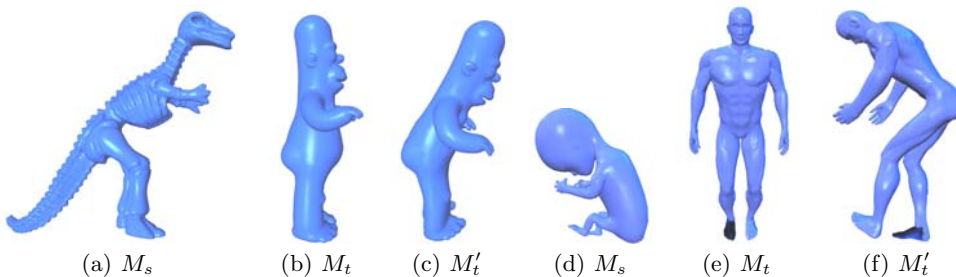


Fig. 14 Direct pose transfer: (a)(d) transfer their pose to (b)(e) and get (c)(f).