Duplicate-Skins for Compatible Mesh Modelling

Yu Wang*
The Hong Kong University of Science and Technology

Charlie C.L. Wang†
The Chinese University of Hong Kong

Matthew M.F. Yuen‡
The Hong Kong University of Science and Technology

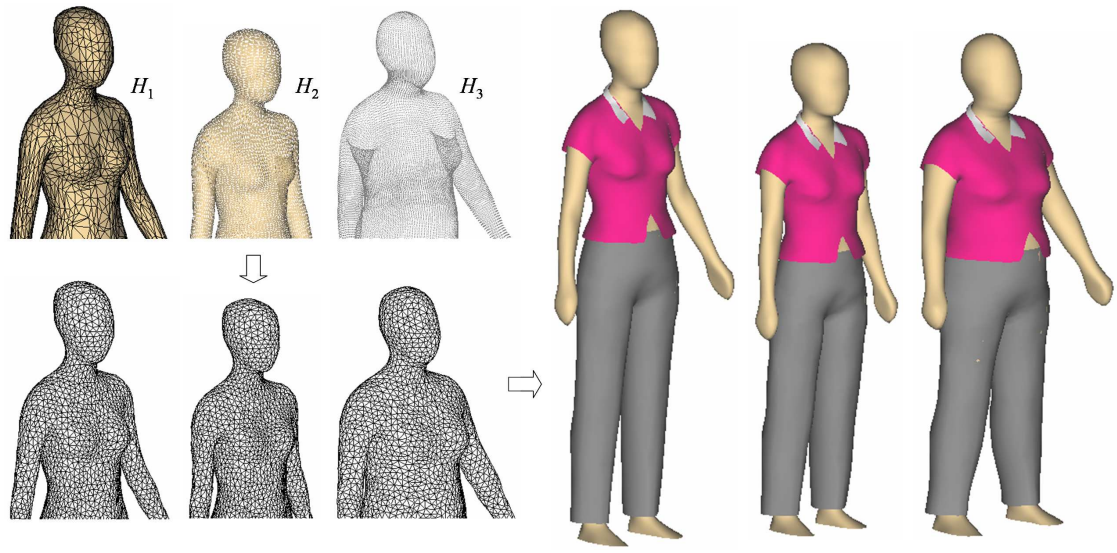

Figure 1: A design automation application — after modelling the compatible meshes by our duplicate-skins algorithm on three human models given in various representations (where H_1 is a two-manifold mesh, H_2 is a polygon soup with many holes, and the shape of H_3 is represented by a point cloud), the clothes designed around H_1 can be automatically "graded" to fit the body shape of H_2 and H_3.

Abstract

As compatible meshes play important roles in many computer-aided design applications, we present a new approach for modelling compatible meshes. Our compatible mesh modelling method is derived from the skin algorithm [Markosian et al. 1999] which conducts an active particle-based mesh surface to approximate the given models serving as skeletons. To construct compatible meshes, we developed a duplicate-skins algorithm to simultaneously grow two skins with identical connectivity over two skeleton models; therefore, the resultant skin meshes are compatible. Our duplicate-skins algorithm has less topological constraints on the input models: multiple polygonal models, models with ill-topology meshes, or even point clouds could all be employed as skeletons to model compatible meshes. Based on the results of our duplicate-skins algorithm, the modelling method of *n*-Ary compatible meshes is also developed in this paper.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Boundary representations—Curve, surface, solid, and object representations; J.6 [COMPUTER-AIDED ENGINEERING]: Computer-aided design (CAD)—Computer-aided design (CAD)

Keywords: Compatible meshes, skin algorithm, free-form modelling, deformation, design automation

*e-mail: mewangyu@ust.hk
†e-mail: cwang@acae.cuhk.edu.hk
‡e-mail: meymf@ust.hk

1 Introduction

Representing the models in compatible meshes is a fundamental problem for a large class of applications, such as mesh metamorphosis [Alexa 2002; Kanai et al. 2000; Lee et al. 1999], *n*-way shape blending/editing [Biermann et al. 2002; Kraevoy and Sheffer 2004; Praun et al. 2001], detail and texture transferring [Kraevoy and Sheffer 2004], parametric design of free-form models [Wang 2005; Seo and Magnenat-Thalmann 2004; Allen et al. 2003; Praun et al. 2001; Marschner et al. 2000], and design automation [Wang et al. 2005]. Compatible meshes, i.e. meshes with an identical connectivity, support bijective mapping between two or more models which establish immediate point correspondences between models. Therefore, each vertex in one mesh has a unique corresponding vertex in every other mesh. The research presented in this paper develops a new algorithm to construct compatible meshes between given models. For two given models M_1 and M_2, our duplicate-skins algorithm manipulates two skin meshes with consistent connectivity to approximate the geometry of M_1 and M_2. Note that the model here means the geometry represented by various representations (e.g., a polygonal mesh or a point cloud — see H_1, H_2 and H_3 in Fig.1). The correspondences of semantic features on the given models M_1

and M_2 are specified by users or by a feature recognition algorithm. Our duplicate-skins algorithm can construct identical entities on the resultant meshes for corresponding pairs of position markers.

Compatible meshes are usually requested on the models with similar features (i.e., we seldom have the need to build a compatible mesh between a tori and a cube); also, we assumed that the corresponding features have been correctly specified. It is meaningless to map the leg of a human H_1 to the head of another human body H_2 and correlate the bellybutton of H_1 to the shoulder of H_2 at the same moment.

1.1 Related work

The work presented in this paper is closely related to the so-called cross-parameterization technique, which established bijective maps between models. Alexa gave a good review of cross-parameterization and compatible remeshing techniques developed for morphing in [Alexa 2002]. The general ways are to parameterize the different models to a common domain. Classifying these techniques according to the types of parameterization domain, there are three categories commonly used: planar, spherical and simplicial parameterization.

The traditional surface parameterization problem considers the case where the domain is a planer region. Kraevoy et al. [Kraevoy et al. 2003] introduced a Matchmaker scheme for satisfying corresponding feature point constraints in both the planer domain and the model's surface. When cross-parameterization is used for geometry processing, it is sometimes possible to limit the computation to disk-like parts of the surfaces [Biermann et al. 2002; Desbrun et al. 2002]. After the entire surface is cut to disk-like parts, each part is parameterized independently. In some techniques, the surface is cut into a single chart [Sheffer and Hart 2002; Sorkine et al. 2002], while in others, it is cut into an atlas of parts (e.g., [Julius et al. 2005; Sander et al. 2003; Levy et al. 2002; Sander et al. 2002]). In either case, the cuts break the continuity of the parameterization, and make it difficult to use a planar parameterization approach to construct a low distortion bijective mapping between two different models.

Another popular choice is spherical parameterization, which uses a sphere as the base domain [Alexa 2002; Gotsman et al. 2003; Praun and Hoppe 2003]. An important limitation of spherical parameterization is that it can only deal with a closed and genus zero surface. One more general approach is to let the domain be a coarse base mesh, called simplicial parameterization. The surface is partitioned into matching patches with a consistent inter-patch connectivity [Praun et al. 2001; Kraevoy and Sheffer 2004; Schreiner et al. 2004]. The challenge in this way is that it is difficult to globally optimize the parameterization. In addition, all of these techniques requires the meshes on given models are valid and two-manifold. Our duplicate-skins algorithm does not have this constraint so that the range of models to be processed is broadened. Although, several researches in literature (e.g., [Nguyen et al. 2005; Ju 2004]) mentioned techniques that can repair the meshes with ill-topology, an operation with less constraints is always welcome.

Avoiding explicit parameterization, [Allen et al. 2003] employed a mesh surface as a template and the connectivity of this template is fixed to approximate the geometry of the input point cloud. They formulated an optimization problem in which the degrees of freedom are an affine transformation at each template vertex. However, their solution is limited to very specified inputs and this can introduce severe approximation errors when the input models have a significantly different geometry.

To explicitly and rapidly construct compatible meshes with different geometries, the skin algorithm presented in [Markosian et al. 1999] is adopted to develop our duplicate-skins algorithm. The original purpose of Markosian et al. is to rapidly design a rough free-form shape via direct interactivities. A user could interactively sculpt a free-from surface (skin) that approximates the underlying given models. The mesh connectivity of skin is updated in the iterations of skin evolution. Inspired by their work, we developed a new algorithm, *duplicate-skins*, to grow over various geometries and obtain compatible meshes.

When updating the conductivity of a mesh, three mesh optimization operators in [Welch and Witkin 1994; Hoppe et al. 1993] are iteratively used: *edge swap*, *edge split*, and *edge collapse*. In the sense of mesh optimization, our approach is related to many remeshing approaches in literature, which reconstruct high-quality meshes for given surfaces. A complete review of the remeshing techniques for surfaces can be found in [Alliez et al. 2005]. As the dynamic optimization manner is adopted in our duplicate-skins algorithm, some of our ideas are borrowed from [Surazhsky and Gotsman 2003; Kartasheva et al. 2003; Ohtake et al. 2003; Vorsatz et al. 2003; Ohtake and Belyaev 2002; Botsch and Kobbelt 2001; Ohtake and Belyaev 2001; Vorsatz et al. 2001] — particularly when the topological changes neighboring to mesh entities are associated with sharp features on the given model. However, all the above approaches consider single meshes while our approach extends the strategies to duplicate meshes.

The same as other dynamic optimization approaches, a good starting point is usually helpful to the convergency. Therefore, in our algorithm, the *radial basis function* (RBF) based shape interpolation techniques are employed to create desirable initial skin meshes to fit the underlying skeleton models. An RBF offers a compact functional description of a set of surface data. Interpolation and extrapolation are inherent in the functional representation. The benefits of modeling surfaces with RBFs have been recognized in [Yngve and Turk 2002; Cohen-Or et al. 1998; Carr et al. 1997; Savchenko et al. 1995]. The radial basis functions associated with a surface can be evaluated at any location to produce a mesh at the desired resolution. [Carr et al. 2001] suggested a RBF-based approach, which can be used for reconstructing the incomplete scan data. Similar to [Cohen-Or et al. 1998], we adopted the global RBF in our approach to deform initial skin meshes into desirable shapes and orientations.

1.2 Definition of Terms

Surface representation based on polygonal meshes has become a standard in many geometric modelling applications. The mesh representation is flexible and general with respect to shape and topology as well as conducive to efficient algorithm processing on meshes. We use the now widespread terminology of mesh from [Spanier 1966]. A triangular mesh is described by a pair (K,V), where $V = (v_1,\ldots,v_n)$ describes the geometric position of the vertices in $\Re^d$ (typically $d = 3$) and K is a simplicial complex representing the connectivity of vertices, edges, and faces. The abstract complex K describes vertices, edges and faces as $0,1,2$ simplicies, that is, edges are pairs $\{i,j\}$, and faces are triples $\{i,j,k\}$ of vertices. The neighborhood ring of a vertex $\{i\}$ is the set of adjacent vertices $N(i) = \{j|\{i,j\} \in K\}$ and its *star* is the set of incident simplices $star(i) = \bigcup_{i\in\varsigma,\varsigma\in K}\varsigma$.

Skeletons are the given models and regarded as a geometric reference, they may be closed mesh surfaces, surfaces with boundaries, non-manifold surfaces, polylines or even isolated points, as shown in Fig.2. In the duplicate-skins algorithm, a pair of skeletons are

assigned as *source* and *target* skeletons respectively. Our method does not require these two skeletons share the same number of vertices or triangles, or have identical connectivity. *Sharp edges* on the given skeletons are edges with relatively large curvatures.

Skin is the mesh growing over a skeleton. We refer to the vertices of the skin as *particles*. For a skin, a *target edge length* is defined which is the expected skin edge length. We take the target edge length as the criterion of skin connectivity modification. The target edge length is measured in terms of the average edge length of skeletons, i.e. $L_{tag} = ratio \times L_{avg}$. Each particle p should track to one position locally closest to p on the relative skeleton. This position is called the *tracking position*. We refer to the face containing the tracking position as the *tracking face*. Our duplicate-skins algorithm simultaneously constructs two skins respectively for the *source* and *target* skeletons. These two skins should be guaranteed compatible and we call these two skins as *duplicate-skins*.

Like most of the technologies used to build the correspondence between meshes [Allen et al. 2003; Sumner and Popović 2004; Kraevoy and Sheffer 2004; Schreiner et al. 2004], a small set of *position markers* are necessary to be specified on the source and target skeletons. These markers are enforced as mapping constraints, taking two head models as an example, the markers constrains the correspondence of ear, nose and eyes as well as other facial elements/features.

1.3 Contribution

We propose a new method for compatible mesh modelling — a duplicate-skins algorithm, which simultaneously grows two skins with identical connectivity over two skeleton models while satisfying the feature correspondences. Compared to other recent approaches for the same purpose, the method presented in this paper shows almost no topological constraint on the models to be approximated (i.e., the input models can be in various geometry representations).

Based on the results of our duplicate-skins algorithm, *n*-Ary compatible meshes also can be easily determined. As the feature vertices (i.e., position markers) are correlated to particles on the skin meshes, the feature correspondences among all *n* skeleton models are satisfied.

Sharp edges are well preserved on the resultant compatible meshes from our algorithm, which is important for many applications. A new sharp feature tracking method is developed to guarantee that the sharpness-preserved results can be given on the compatible meshes.

Thanks to the connectivity optimization in the duplicate-skins algorithm, the resultant compatible meshes are relatively regular, so that they can serve as good inputs for the downstream geometry processing applications where the irregularity usually leads to unsatisfactory results.

The rest of the paper is organized as follows. Section 2 gives a more detailed description of the original skin algorithm [Markosian et al. 1999]. Section 3 details our duplicate-skins algorithm. Section 4 presents the method to construct *n*-Ary compatible meshes. In section 5, several applications of the compatible meshes are demonstrated — the applications fall into two categories: free-form modelling and design automation for customized free-form products. Finally, section 6 discusses the limitations of our algorithm and suggests future research directions.

2 Skin Algorithm

We give a more detailed description of the original skin algorithm [Markosian et al. 1999] as below which works as the basis of our duplicate-skins algorithm. With a skeleton mesh M as the input, the skin algorithm governs a skin mesh S growing over M to approximate its shape with a smooth mesh whose connectivity is more regular. The algorithm consists of five steps: 1) the construction of the first skin S; 2) search for the tracking position and face for each particle on S; 3) reposition particles; 4) modify the connectivity of S; 5) update the tracking position and face for particles on S. The 3rd to 5th steps are iteratively applied on S until the movement of all particles on S is less than a small value ε and no further modification of mesh connectivity is needed.

There is no limitation to the topology of the first skin S. The only requirement is that S should show the shape that can vary into the source/target model by an elastic deformation. For example, for genus-0 models, a mesh with a box shape or a spherical shape bounding them is a good initial skin. However, for genus-1 models, we must introduce tori-like initial skins.

Before entering the iteration of skin evolution, the closest point and face of each particle on the skeleton S must be found to serve as the tracking point and face. The global searching strategy is applied to accurately obtain the first tracking. In the later iteration steps, to speed up, a local search strategy replaces the global one.

To gradually attract skin toward the skeleton, the movement of each particle is measured from its tracking position, its current position and its neighbors on the skin. The new location $v_{new}(p)$ of the particle p is computed by

$$v_{new}(p) = \alpha v_0(p) + \beta v_c(p) + \gamma v_t(p) \quad (1)$$

where $v_0(p)$ is the current location of the particle, $v_c(p)$ is the center of p's 1-ring neighbors, and $v_t(p)$ is the target position that computed by

$$v_t(p) = v_0(p) + w_m(d_s - r_s)v_{trk}(p) \quad (2)$$

where $v_{trk}(p)$ is the unit tracing direction to the closest point on skeleton. d_s is the distance from p to its tracking position, r_s is the user specified offset between the skeleton model and the final skin, and w_m is the moving ration in the range $[0,1]$ which controls the amount of movement. α, β and γ are positive coefficients, and $\alpha+\beta+\gamma=1$. β controls the smoothness of the skin. The trade-off for selecting β determines the behavior of the skin, e.g. a large β leads to the over smoothed skin; on the other hand, if a very small β is adopted, uneven particle distribution and sliver triangles could be produced. γ is calculated by $\gamma = 1-\alpha-\beta$. From our experiments, we choose $\alpha = 0.3$. A non-linear and attenuating function is used to evaluate the value of β where the iteration step is the function variable.

In the connectivity modification step of the skin algorithm, three operators: *edge swap*, *edge split*, and *edge collapse* from [Welch and Witkin 1994; Hoppe et al. 1993] are iteratively applied to remove extreme long and short edges, and at the same time increase the minimal angle in triangles. Through this, the shape quality of the triangles on the skin mesh is optimized.

As claimed by [Markosian et al. 1999], the skin algorithm can work with the skeleton models in the form of closed mesh surfaces, surfaces with boundaries (more generally saying - non-manifold models), polylines or even isolated points. Examples are given in Fig.2. Benefitting from this characteristic of the basic skin algorithm, our duplicate-skins algorithm can model compatible meshes approximating various skeleton models. Details are addressed below.

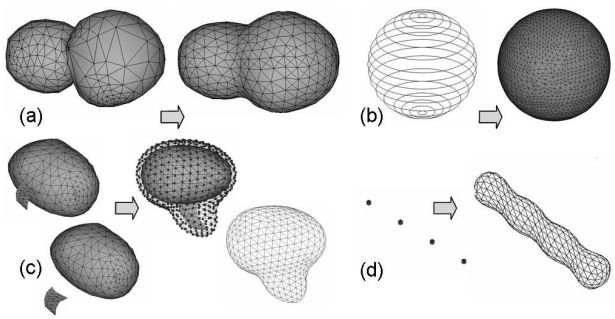

Figure 2: Various models could be employed as skeletons: (a) a closed mesh surface, (b) a wire-frame, (c) a non-manifold structure assembled from several mesh patches, and (d) isolated points.

3 Duplicate-Skins

To seek a method to construct compatible meshes for various free-form models, we propose a new technique and name it *duplicate-skins* which is derived from the original skin algorithm. The most important difference is that we can now simultaneously manipulate two skins in various geometries but with identical mesh connectivity. Benefitting from the desirable high-quality of skin meshes, the resultant compatible meshes give a bijective mapping between them that is guaranteed to be smooth and continuous. The following addresses the details of our duplicate-skins algorithm for generating compatible meshes between a pair of skeletons.

3.1 Algorithm overview

The input to the algorithm consists of two skeletons $M_0 = (K_0, V_0)$ (source) and $M_1 = (K_1, V_1)$ (target) and two sets of position markers (including $P_0 = \{(x_i, y_i, z_i), i = 1, \ldots, n\}$ defined on M_0 and its correspondence $P_1 = \{(x_i', y_i', z_i'), i = 1, \ldots, n\}$ defined on M_1). A point $(x_i, y_i, z_i) \in P_0$ should be mapped to the point $(x_i', y_i', z_i') \in P_1$. The duplicate-skins algorithm is then outlined as below, after which the key phases of the algorithm are detailed successively.

1: Construct the first skin S_0 for M_0;
2: Construct the first skin S_1 for M_1 by copying S_0 to S_1 and deform S_1 to be around M_1 by the markers P_0 and P_1;
3: Initialize the tracking position to every particle globally;
4: Determine the particles that should track to the position markers;
5: **repeat**
6: Reposition the particles in tandem with S_0 and S_1;
7: Modify the mesh connectivity;
8: Update the tracking position of a particle if it does not track to a marker;
9: **until** no change occurs;
10: **if** M_0 or M_1 has sharp features **then**
11: Find corresponding sharp-tracking-edges on skins S_0 and S_1;
12: **repeat**
13: Reposition the particles in tandem with S_0 and S_1;
14: Modify the mesh connectivity;
15: Update the tracking position of a particle if it tracks neither a position marker nor a sharp edge;
16: **until** no change occurs;
17: **end if**

To let the resultant meshes have the correct correspondence on the position markers, if a particle on S_0 tracks to a position marker $\tau = (x_m, y_m, z_m) \in P_0$, the identical particles (in terms of topology) on S_1 should track to its corresponding marker (i.e., $\tau' = (x_m', y_m', z_m') \in P_1$). For searching the tracking positions at the very beginning, space subdivision techniques (e.g., Octree or kD-tree) could be used to speed up the algorithm. We employ an Octree with a fixed depth in our implementation. Different from the particles tracking to markers, the other particles can freely track to either a vertex or a surface point (i.e., an interior point on a triangle) on skeletons. However, even if the space partition strategy is employed, it is still inefficient to conduct a global closest point search in every iteration step. In [Markosian et al. 1999], a local update strategy is conducted: for a particle p, the new tracking point is only searched on a limited number of faces – the set of entities to be searched, Γ, includes the current tracking face f of p and the faces containing any vertex of f on the skeleton; and in order to get out of a local minimum, for any particle $p' \in N(p)$, the tracking face f' of p' and the faces $f_i \in star(j)$ for $j \in f'$ are all added into Γ. The above strategy relies on the local connectivity on skeletons, thus fails on models with ill-meshes or point cloud models. To overcome the limitation, we change the strategic rules for locally updating tracking points to the follows:

- for a particle p, if an Octree node Υ_d contains the current tracking point of p, its new tracking point is searched among the faces/vertices held by Υ_d and the spatial neighboring nodes of Υ_d;
- for the purpose of jumping out of the local minimum, again the Octree nodes containing the tracking points of the particles $p' \in N(p)$ are added into the range of searching.

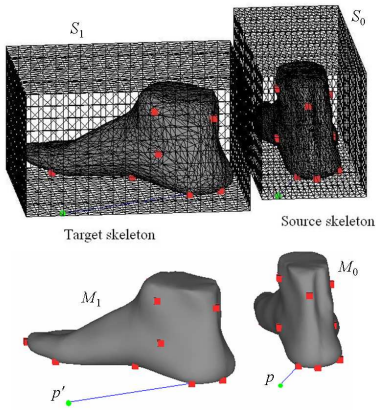

Figure 3: Illustration to explain why the RBF-deformation is needed for constructing initial skins. On the source skin and skeleton (right), the green particle p is the point on the skin S_0 closest to the marker at heel on M_0 (they are linked by the tracking vector in blue); however, the corresponding particle of p, $p' \in S_1$, is not the closest point to the corresponding marker on M_1, so that the dispersion of tracking directions occurs around p' since the particles near to p' all track to their closest points on S_1 but p' does not. Note that the topologies of S_0 and S_1 are always identical.

3.2 Initial Skins

To describe how we generate the initial skins in more detail, let us use the example shown in Fig.3. For two feet models, as the source and target skeletons (i.e., M_0 and M_1 respectively), are placed in different positions and orientations. As explained previously, the easiest way to generate initial skins for genus-0 models is to construct skin meshes as the bounding boxes orthogonal to x,y,z-axes.

However, problems may be encountered when position markers are specified on skeletons. For example, see Fig.3, after searching through the skin for the source skeleton, the green particle p on skin S_0 is closest to the marker point (in red) at the heel. Since the particles tracking the markers on the source and target skeletons should be identical, the corresponding particle of p on the target skin - $p' \in S_1$ is *not* the closest point to the heel marker. Figure 3 gives the tracking directions of p and p' in blue color. However, all the particles around p' will still track to their closest points on M_1; in other words, the tracking directions are dispersed, which easily leads to poor or even invalid meshes (e.g., face flipped). Therefore, to eliminate the occurrence of the above situation, the construction problem of S_1 after obtaining S_0 is reformulated as follows.

Problem: Given a set of position markers P_0 defined on M_0 and P_1 with respect to M_1 and a surface S_0, find a surface S_1 which is transformed from S_0 and the deformation from S_0 to S_1 is equivalent to the deformation from P_0 to P_1.

The above problem is solved by defining a deformation function $\Psi(\ldots)$ letting $P_1 = \Psi(P_0)$ so that $S_1 = \Psi(S_0)$ is easily obtained. The radial basis function (RBF) is the most suitable candidate for this deformation function [Botsch and Kobbelt 2005; Turk and O'Brien 2002]. In general, a RBF is represented in the piecewise form

$$\Psi(x) = p(x) + \sum_{i}^{n} \lambda_i \phi(\|x - \tau_i\|) \tag{3}$$

where $p(x)$ is a linear polynomial that accounts for the rigid transformation, the coefficients λ_i are real numbers to be determined and $\|\cdot\|$ is the Euclidean norm on $\Re^3$. To achieve a global deformation, the basis function $\phi(t)$ is chosen as $\phi(t) = t^3$ (the triharmonic spline as [Yngve and Turk 2002]). The coefficients λ_i and the coefficients of $p(x)$ can be easily determined by letting $\Psi(\tau_i) \equiv \tau_i'$ for all pairs of $\tau_i \in P_0$ and $\tau_i' \in P_1$ plus the compatibility conditions $\sum_i^n \lambda_i = \sum_i^n \lambda_i \tau_i = 0$. The formulated linear equation system has been proven to be positive definite unless all the points in P_0 and P_1 are coplanar.

The use of RBF guarantees smooth geometric deformation. Thus, the geometry of target skin S_1 can be determined by smoothly blending the positions of the vertices on the source skin S_0 as

$$(p' \in S_1) = \Psi((p \in S_0)). \tag{4}$$

The result of the foot example by this deformation is shown in Fig.4(a), where the initial skin S_1 follows the orientation of skeleton M_1. Also, compared to Fig.3, the green particle on S_1 is much closer to the marker at heel. This greatly reduces the chance to generate self-overlapped skins. However, uneven triangulation could still happen (see Fig.4(b)), since the somewhat conflicting tracking directions still exist among the particle which are enforced to track markers and its neighbors. Therefore, to completely eliminate the conflicting tracking directions, an alternative way with pre-skinning is proposed to construct the initial duplicate skins. Firstly, we perform several runs of the single skin algorithm for the source skeletons. The result skin is applied as S_0 and then this skin is deformed to fit the target skeleton by Eq.(3) and (4), so that S_1 is created. Figure 4(c) shows the result from this change. The source and target skeletons are respectively approximated by their corresponding skins. In this way, the green particle is almost the closest particle to its corresponding markers through the whole skin S_1. Consequently, a significant feature is introduced here: our duplicate-skins algorithm is independent of the placement of input skeletons.

The above RBF-deformation based method also enables our method to work for those genus-k models ($k \neq 0$). If enough number of position markers are well defined around each handle, we can duplicate a mesh from the source skeleton M_0 to serve as S_0 and employ

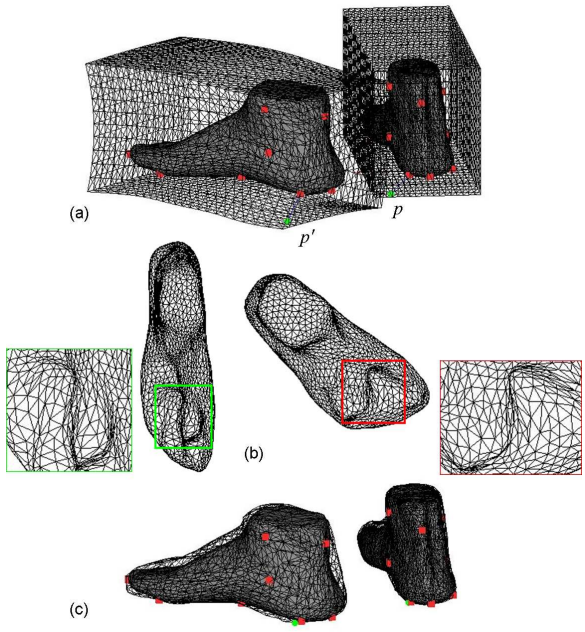

Figure 4: Illustration of the method with pre-skinning to construct initial duplicate skins. (a) Initial duplicate skins in box shape; after S_0 have been obtained, the S_1 is obtained by RBF-deformation. (b) Beginning with the initial duplicate skins in (a), the result of the duplicate-skins algorithm is shown in very uneven triangulations (see two close-up views) — this is because that some faraway particles are enforced to track markers on the target skeleton. (c) Construct initial duplicate skins by the method with pre-skinning, where the shape of S_0 after several (5 to 8) runs of the single-skin algorithm is adopted to create S_1 instead of the box shape in (a).

the above RBF-deformation function to create S_1 around M_1. In our experience, four to eight pairs of uniformly distributed markers for each handle will be enough. For instance, as the example shown in Fig.10, red points are the position markers. The initial skins consist of one tori mesh and another mesh deformed from tori by RBF. Therefore, starting from the genus-1 initial skins, our duplicate skins are iteratively evolved to take the form of the skeleton shape with mesh optimized.

3.3 Optimize connectivity on duplicate skins

In our duplicate-skins algorithm, two skins are adopted to approximate their relative skeletons. In the course of skin evolving, the mesh topology updating is identical on duplicate skins, even though they interpolate different underlying geometries. That is the primary ingredient for the generation of compatible skins. Obviously, when we apply the edge-based optimization operators, the measurements on two skins should both be under consideration.

To evaluate the criteria of edge splitting, the edge lengths on S_0 and S_1 are both taken into account. We perform the edge split if either edge $\{i, j\} \in S_0$ or its corresponding edge $\{i', j'\} \in S_1$ satisfies the splitting condition. Note that the duplicate skins share one target edge length L_{tag}. For two skeletons with sizes that differ significantly, we scale M_1 by the ratio $\rho = DL_0/DL_1$ before applying our algorithm where DL_0 and DL_1 are the diagonal lengths of the bounding boxes of M_0 and M_1. After computing the compatible meshes, we scale M_1 and S_1 back to the original dimension by the

ratio ρ^{-1}. When $\{i,j\}$ and $\{i',j'\}$ are both less than half of L_{tag}, these two edges can be collapsed.

Criterion 1: A pair of edges $\{i,j\} \in S_0$ and $\{i',j'\} \in S_1$ are allowed to be split if $\|v_iv_j\| > 1.5L_{tag}$ or $\|v'_iv'_j\| > 1.5L_{tag}$.

Criterion 2: A pair of edges $\{i,j\} \in S_0$ and $\{i',j'\} \in S_1$ are included in the edge-collapse candidates if and only if $\|v_iv_j\| < 0.5L_{tag}$ and $\|v'_iv'_j\| < 0.5L_{tag}$.

To prevent over-optimization in one iteration step, the numbers of splits and collapses are limited in each run. In fact, the available edges for either split or collapse are sorted by the ratio of their edge length to the target length. The ratio is defined by $\max(\|v_iv_j\|, \|v'_iv'_j\|)/L_{tag}$ for split and $(\|v_iv_j\| + \|v'_iv'_j\|)/2L_{tag}$ for collapse. We do the split or collapse operations on only the first 10% of edges in priority. Analogous to the basic skin algorithm, any edge that is too long (or too short) is still guaranteed to be split (or collapsed) eventually.

Furthermore, we propose the following scheme to estimate the edge-swap criterion. As shown in Fig.5, if we swap the two edges in red color, we need to compare the maximum opposite angles shown in the triangles before and after swapping, where the opposite angles are respectively denoted by α_i and β_i $(i = 1,2,3,4)$. The criterion for edge-swapping on duplicate meshes is given as follows.

Criterion 3: Defining $\alpha_{\max} = \max\{\alpha_i\}$ and $\beta_{\max} = \max\{\beta_i\}$, if and only if $\alpha_{\max} > \beta_{\max}$, the pair of edges $\{i,j\} \in S_0$ and $\{i',j'\} \in S_1$ are considered to be swapped.

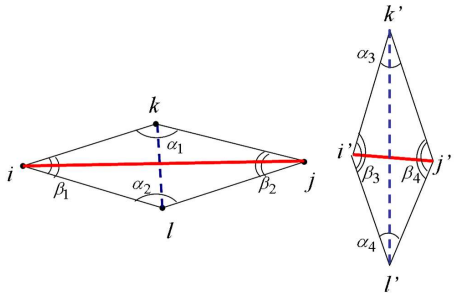

Figure 5: Edge-swap for duplicate-skins: left part – the edge $\{i,j\}$ and its adjacent triangles on S_0, right part – the edge $\{i',j'\}$ and its adjacent triangles on the target skin S_1.

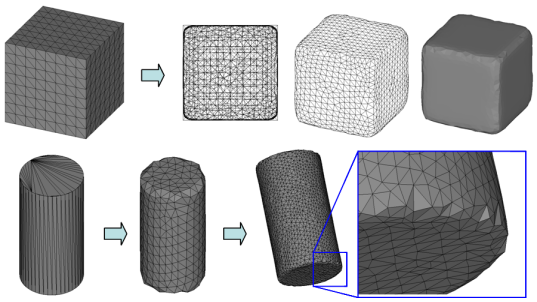

Figure 6: Aliasing errors are introduced by the skin algorithm: top row, for the box skeleton given in the most left, the resultant mesh from the skin algorithm has a high quality but degenerates in the sharp edges and corners; bottom row, for the given cylinder with a coarse and irregular mesh, even if a denser mesh is employed, the original skin algorithm can hardly recover the aliased sharp curves.

3.4 Sharp edge recovering

The original skin algorithm cannot give a correct construction at sharp features of a skeleton. If a skeleton has sharp geometric features (i.e., creases where the surface does not have continuous tangent planes) on it, the skin always aliases the sharpness with approximation artifacts. As pointed out in [Kobbelt et al. 2001], increasing the sampling rate of surfaces will not cause the skin to converge to the sharp edges and corners if no special treatment is given. Our tests also prove this (see Fig.6). In this section, we propose an efficient scheme that recovers sharp features on the resultant skins. Note that this only works for the skins interpolating skeletons (i.e., $r_s = 0$ in Eq.(2)).

On the skeleton meshes, all the endpoints of sharp edges are defined as *sharp vertices* and all the triangle faces belonging to the stars of sharp vertices are called *sharp faces*. After the sharp edges have been identified on the skeleton meshes, several edges of the skin, named as *sharp-tracking-edges*, are enforced to align to these features. Also, to prevent breaking the sharpness on the resultant skins, the meshes around sharp tracking edges should be specially treated during connectivity optimization. As mentioned in the algorithm overview, the sharp edge recovering procedure is regarded as a type of post-processing. When applying the algorithm, the skins have almost interpolated the given skeletons, so only several runs are needed and can be completed in a short time. The most important phase is to determine the sharp-tracking-edges on S_0 and S_1 which can be decomposed into three steps:

- Find the sharp edges, vertices and faces on skeletons;
- Determine the particles tracking to sharp vertices;
- Compute the shortest path on the skins between each pair of particles which track to the two endpoints of a sharp edge, where the path passes along the skin edges.

For extracting the sharp edges on skeletons, we can either manually pick the edges or automatically detect them by the discrete sharp operator referring to the discrete mean curvature at the mesh edges. For an edge e with dihedral angle θ_e, $H_e = 2\|e\|\cos\frac{\theta_e}{2}$ (ref. [Hildebrandt and Polthier 2004]) is given as the mean curvature on e. If H_e exceeds a threshold, the edge is labelled as a sharp edge.

Next, we need to find the corresponding sharp-tracking-edges on the skins. More specifically, a list C_e of edges are enforced to track each of the edges e with '*sharp*' label. To successfully recover sharp features on skeletons, the curve formed by C_e must be single-wide. Thus, the particles tracked to sharp vertices in the bijective manner (i.e., without repeating) are found first. The shortest path linking the two particles tracking to a pair of endpoints are then determined by the Dijkstra's algorithm. To speed up the searching, we filtered out most of the edges on the skin – only the edges with their endpoint tracking to sharp vertices/edges/faces are regarded as legal paths. Also, the edges that have been labelled as sharp-tracking-edges in previous searches are prevented from the searching (by assigning their length to ∞). As a result, for every list of sharp-tracking-edges, C_e, its starting and ending particles track to the endpoints of e, and the interior particles of C_e are restricted to the inertia points on e by the proportion of lengths. In other words, the particles in C_es are tracked to the sharp edges exactly. Therefore, the sharp edges are preserved list by list. See the models in Fig.7, the red edges on a skin are the sharp-tracking-edges on the skin while the blue ones are sharp edges on the skeleton.

Finally, as pointed out in the algorithm overview, when we iteratively optimize the connectivity on a skin mesh, several configurations need to be discussed if any entity related to sharp features is under consideration.

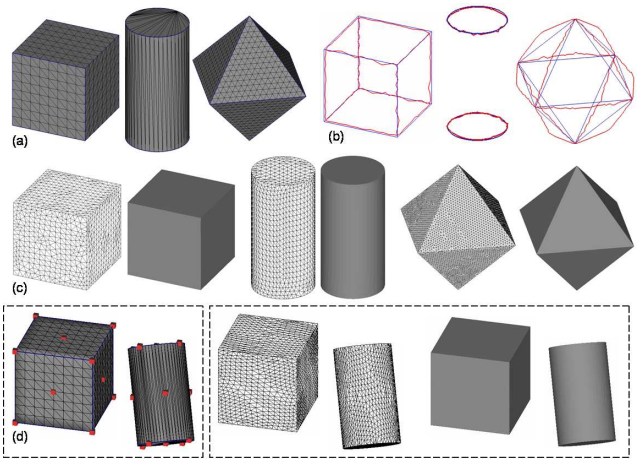

Figure 7: Shape edge recovering: (a) given skeleton meshes, (b) the sharp edges (in blue) on skeletons and their corresponding sharp-tracking-edges (in red) on skins, (c) the sharp edge recovering results on individual skeletons, and (d) the result of duplicate-skins with sharp edges recovered.

Criterion 4: If an edge on skin is a sharp-tracking-edge, this edge is prevented from swapping.

Criterion 5: For the convenience of implementation, if an edge e is a sharp-tracking-edge, we prevent edge-split on it; otherwise, the list C_e holding e needs to be updated and the tracking position of the newly inserted vertex needs to be searched.

Criterion 6: If both two particles of an edge are endpoints of sharp-tracking-edges, collapse on this edge is not allowed.

The configuration considered by the above criterion could be either of the two cases shown in Fig.8(a), where the first one eliminates the sharp-tracking-edge and the second one merges two diverse sharp-tracking-edges into one.

Criterion 7: If an edge $\{i,j\}$ satisfies the collapse-length criterion and only one of its endpoints $\{i\}$ tracks to sharp features, this edge could be collapsed.

The degenerated vertex $\{h\}$ from $\{i,j\}$ and the edges $\in star(i)\bigcup star(j)$ must be carefully processed. The position of $\{h\}$ is not located at the middle of $\{i,j\}$ after collapsing, but should inherit the position of the particle tracking to the sharp features (i.e., the position of $\{i\}$ in Fig.8(b)). In addition, if either $\{k,i\}$ or $\{k,j\}$ is a sharp-tracking-edge, the edge $\{k,h\}$ degenerated from the triangle $\{i,k,j\}$ must be assigned as a sharp-tracking-edge as illustrated in Fig.8(c). Similarly, the edge $\{l,h\}$ should be a sharp-tracking-edge, if either $\{l,i\}$ or $\{l,j\}$ is sharp tracking edge.

Criterion 8: For a pair of edges $\{i,j\} \in S_0$ and $\{i',j'\} \in S_1$, if either of them satisfy one of the above four criteria, the topology update of the edges should follow the above rules.

Criterion 9: During the iteration of sharp edge recovering, we update the tracking position and tracking face of the particles, only if they do not track to any sharp vertex/edge.

As mentioned in the outline of our algorithm, the loop for interpolating sharp features works as a post-processing procedure; hence, the iteration should have a small number of steps and this additional loop does not degenerate the efficiency of our algorithm.

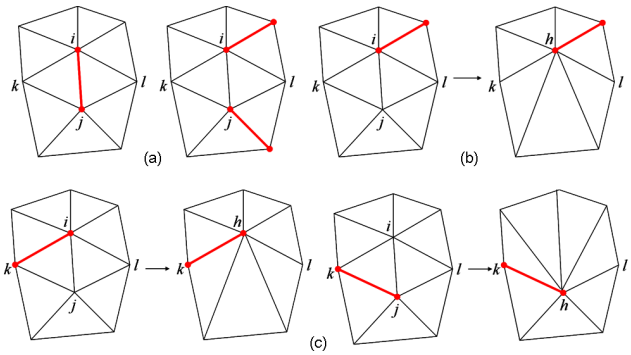

Figure 8: Cases that should be specially treated in the sharp-edge-preserved mesh optimization, where red edges denote the sharp-tracking-edges and red points are the particles tracking to sharp features. (a) Two cases that the edge $\{i,j\}$ cannot be collapsed. (b) If the particle $\{i\}$ tracks to sharp features, the degenerated particle $\{h\}$ maintains the position and tracking information of $\{i\}$. (c) The configurations for collapsing $\{i,j\}$ if either $\{k,i\}$ or $\{k,j\}$ is a sharp-tracking-edge.

4 n-Ary Compatible Meshes

The duplicate-skins algorithm above can successfully generate the compatible meshes S_0 and S_1 to approximate the given two skeleton models M_0 and M_1. However, for some applications (e.g., n-way blending, sample based parametric design of freeform models, etc.), the compatible meshes are requested for more than two skeleton models. The n-Ary compatible meshes can be generated through the vertex transformations on the results from our duplicate-skins algorithm.

Suppose that the compatible meshes are requested on n skeleton models M_i $(i = 0,\ldots,n-1)$, the duplicate-skins algorithm is first applied $n-1$ times on the pairs of skeletons — M_0 and M_j $(j = 1,\ldots,n-1)$. Thus, $n-1$ pairs of skins are obtained; for the convenience of description they are denoted by $S_{0^{(j)}}$ and $S_{(j)}$ respectively. Letting $S_0 = S_{0^{(1)}}$ and $S_1 = S_{(1)}$, we conduct the following algorithm to determine the meshes S_j $(j > 1)$ compatible to S_0 on M_j.

1: Duplicate a skin mesh S_j with S_0;
2: **for all** vertex $p \in S_j$ **do**
3: The closest point p_c of p on $S_{0^{(j)}}$ is found;
4: p_c must be inside a triangle $T \in S_{0^{(j)}}$, the barycentric coordinates of p_c on T - $(\alpha_c,\beta_c,\gamma_c)$ is then recorded;
5: By the pair of skins $S_{0^{(j)}}$ and $S_{(j)}$, T could easily find its corresponding triangle T' on $S_{(j)}$;
6: Applying the braycentric coordinate $(\alpha_c,\beta_c,\gamma_c)$ on T', the new position of p on $S_{(j)}$ can be computed;
7: Move p to the new position;
8: **end for**

Repeating the vertex transformation, the new shape of S_j approximating M_j is easily determined. It is easy to find that all steps of the above algorithm can be finished in a short time except the closest point search step. For this step, the space partition strategy which has been previously employed in the tracking point search is used again to accelerate the process.

About the approximation error. Although easy to implement, the above algorithm for n-Ary compatible meshes enlarge the L^2 approximate error (see Fig.9a). This enlargement is led by the closest point projection, where $S_{(j)}$ is employed to approximate M_j. The

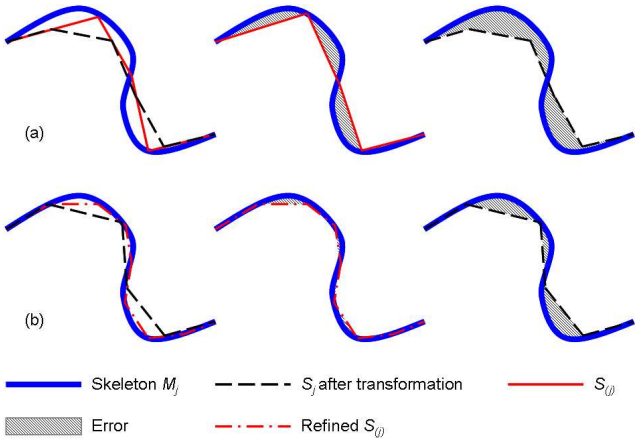

Figure 9: L^2 approximate error analysis: (a) L^2 is enlarged on the compatible meshes generated using vertex transformation (comparing the approximation error on $S_{(j)}$ and S_j), and (b) the approximation error (both $S_{(j)}$ and S_j) can be reduced by refining $S_{(j)}$.

Table 1: Computational Statistics

Examples	Fig.10	Fig.11	Fig.12
Number of Triangles (skeleton I)	3,356	3,270	4,638
Number of Vertices (skeleton I)	1,128	1,648	1,821
Number of Triangles (skeleton II)	616	2,228	8,030
Number of Vertices (skeleton II)	308	1,165	4,017
Number of Triangles (skin)	2,634	13,654	19,880
Number of Vertices (skin)	1,317	6,829	9,942
Comp. Time (in sec.)	∼10	∼83	∼40

error can be reduced by refining $S_{(j)}$ while keeping the same connectivity on S_j. By the refinement, the approximation error shown on $S_{(j)}$ is decreased so that the error given on S_j is also reduced (comparing the errors shown in different rows of Fig.9). We can also apply the repositioning step of our duplicate-skins on S_j for several runs to decrease the approximation error.

5 Results and Applications

All the examples shown in this paper are tested on a standard PC with AMD 1.6 GHz mobile CPU and 480MB RAM. For the examples shown in Fig.10, 11 and 12, the computational statistics are listed in Table 1.

5.1 Free-form modelling

Our first example is to apply our duplicate-skins algorithm on genus-1 models (a torus and a mechanical-part with sharp edges). As seen in Fig.10, twelve pairs of manually specified position markers (red points) govern our algorithm and establish correct correspondences on the resultant compatible meshes. Sharp edges are well recovered. Then, the compatibility of the meshes are employed to partially deform the mechanical part into the torus. Our duplicate-skins algorithm is applied to the design of a toy bear in the example shown in Fig.11. The original bear model consists of 1 hemisphere, 3 spheres and 3 cylinders. Although the bear model is not a single manifold surface, benefited from the duplicate-skins algorithm, we can still generate compatible meshes on a rabbit model and the bear model. The body of bear was selected and reshaped to the shape of its corresponding part on the rabbit, so that the final bear model is obtained. Figure 12 gives the original head models and the compatible meshes for two head models. The position markers in red take the role of defining the correspondences of semantic features. After the compatible meshes have been constructed, it is very easy to change the female's nose by the shape of the male model's (see Fig.13). The shape variation performed in the above three examples are all with the help of the displacement-map technique which is widely used in computer graphics applications. Briefly, the detail geometry of a mesh surface M (or part of a mesh surface) is encoded on a low-pass filtered $\overline{M}$ of M, and then the encoded surface details are added onto another filtered mesh $\overline{M'}$ so that the details of M are shown on M'.

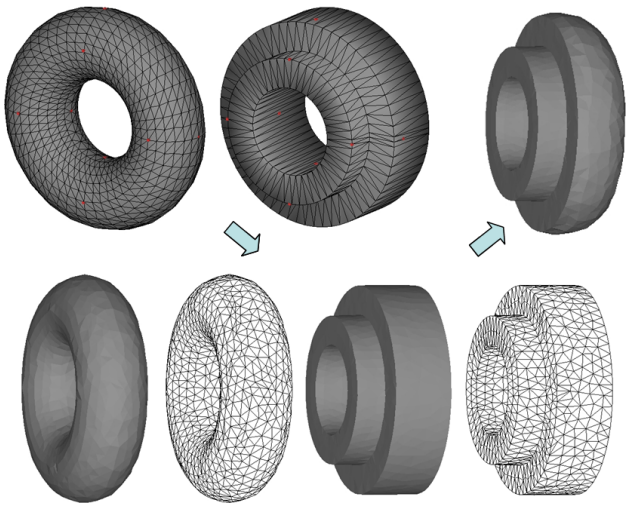

Figure 10: Tori-MechPart example: our duplicate-skins algorithm can generate compatible meshes on genus-1 models with sharp-edges well recovered.

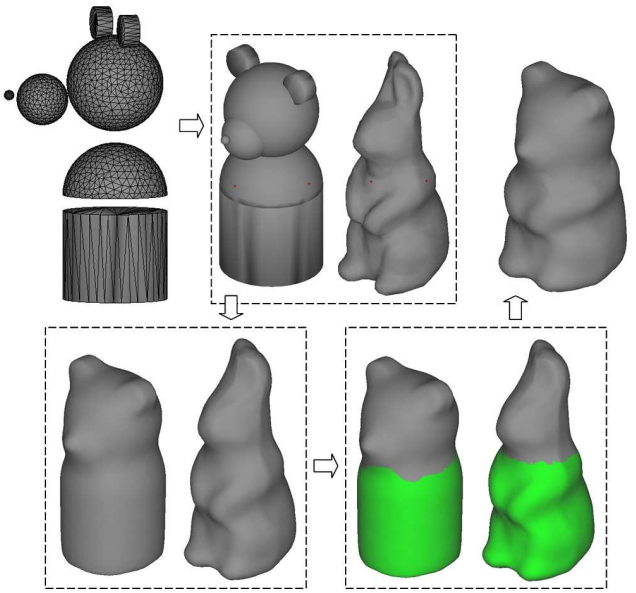

Figure 11: Toy bear design with our duplicate-skins algorithm.

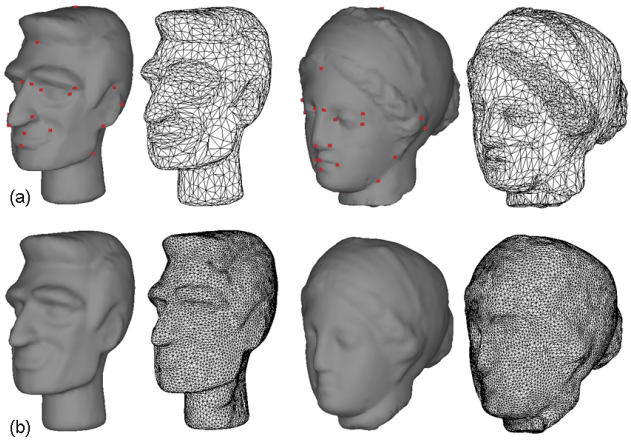

Figure 12: Compatible meshes generated on head models: (a) given heads models and the position markers defined on it (red ones), and (b) resultant compatible meshes.

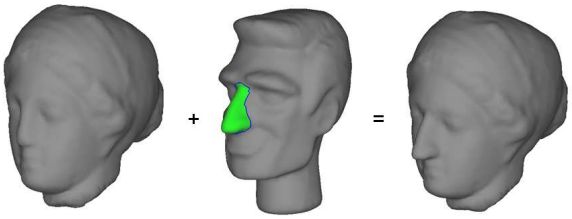

Figure 13: Cut-and-Paste modelling for changing the nose on a head model.

Figure 14 demonstrates the compatible meshes on 3 head models so that the interpolation triangle among three head models could be determined. The *n*-Ary compatible meshes are usually employed in the application of the parametric design of free-form models (e.g., the parametric design of human models [Wang 2005]). As shown in Fig.15, after obtaining the input parameters from users, a set of human models are selected from the human model database, where all human models stored in the database come with surfaces have compatible meshes. By a numerical optimization scheme, we can determine the synthesis weights of this set of human models so that the result synthesized models give the user specified parameters, where the synthesis is eventually a weighted blending procedure.

5.2 Design automation for customized products

The application of the resultant compatible meshes is not limited to the variation of mesh surfaces themselves. The compatible meshes are also very important to the design automation problem in several industries (e.g., apparel industry, shoe industry, jewelling industry, eye-wear industry, etc.). In all these industries, there exists a common quest for design automation: after carefully designing a product's geometry around a model in standard size and shape, it is desired to automatically transform the geometry of the product to other models with customized shapes while maintaining the originally spatial relationship between the product and the model (i.e., preserving the fitness). For example, Fig.1 shows this application in the apparel industry. After constructing the compatible meshes from various input models (two-manifold mesh model, a polygon soup with holes, or a point cloud), we can apply the t-FFD or p-FFD technique [Kobayashi and Ootsubo 2003; Wang et al. 2005] to

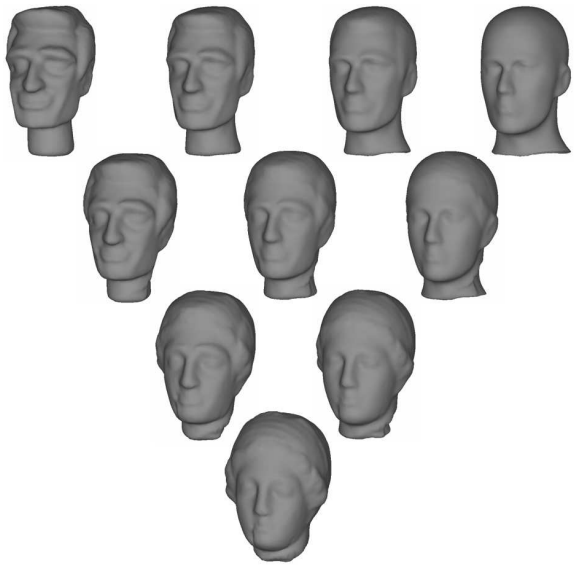

Figure 14: Shape interpolation among three head models.

encode the coordinate of each vertex on clothes by the compatible mesh H_1. Then, based on the correspondences between triangles among S_1, S_2, and S_3, we can easily fit the clothes to the shape around S_2 and S_3. Figure 16 shows a similar design automation application in the shoe industry, where the foot models have been previously shown in Fig.4.

6 Limitations and Discussion

This paper presents an approach for modelling compatible meshes on give models. Our duplicate-skins algorithm works on the models represented by polygonal meshes, a polygon soup, or a point cloud. The algorithm drives two active particle-based mesh surfaces (i.e., skins) with identical connectivity to approximate the given models that serves as the skeleton. One major limitation of the algorithm is the shrinking effects: when using skins with a relatively large triangle size to model compatible meshes on the skeleton with complex details, the shrinking effect occurs around the details since the sampling rate on skins is low. Also, the sharp edge can hardly preserved if the resolution of skin mesh is lower than the resolution of skeletons. This is in fact the problem addressed by the sampling theory. Increasing the sampling frequency can solve this problem to a certain degree but cannot guarantee the preservation of details since there is no mechanism in our current approach to ensure detail preservation. This is definitely an area we should consider in the future. One possible approach for solving this problem is to employ the compatible meshes generated by our method as the control network of subdivision surfaces, then the detail geometry could be recovered in the later mesh refinement procedure. The approximation error of our approach also needs to be further studies, and a curvature-based mesh refinement could be considered in our framework to reduce the error.

Another limitation of this approach and also all the other cross-parameterization algorithms (e.g., [Kraevoy and Sheffer 2004; Schreiner et al. 2004]) is that the algorithm relies too heavily on the position markers when modelling the genus-*k* models. For instance in the Tori-MechPart example in Fig.10, if all the markers on the tori M_1 are mapped to the markers all being placed on the left

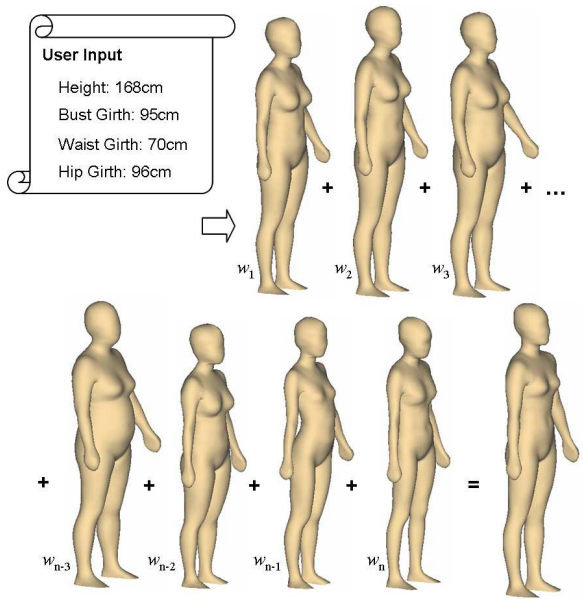

Figure 15: Parametric design of human models, where the human models with compatible meshes are synthesized into a model satisfying the user input parameters.

part of the MechPart M_2 (instead of the similar positioning of markers shown on M_2 in Fig.10), compatible meshes for these genus-1 models will not be correctly constructed. Therefore, an important area of future research is how to develop a method to add markers to ensure correct mapping. Also, the positions of markers effect on the quality of triangles and the result of shape approximation. For example, if two skeletons are much different from each other, the over-strict constraints on position markers may lead a result with great distortion. Or if there is no marker defined in the cavity of a U-shaped skeleton, our current method can hardly pull the skin mesh into the cavity. All these problems are to be investigated in our future research to improve our approach.

The last limitation in our current implementation is that the generation of n-Ary compatible meshes does not preserve sharp-edges. However, since the sharp-edges have already been recovered in our basic duplicate-skins algorithm, it is possible for us to find some methods for preserving the sharp-edges when generating n-Ary compatible meshes. Our first idea is that after we have found the corresponding paths of sharp-edges (similar to what we did in section 3.4), we restrict the mapping of such paths to be along the triangle edges (instead of triangle faces) in the mapping of n-Ary compatible meshes. This will be investigated in our future research.

Although there are several limitations, our approach shows an important advantage comparing to other approaches for the same purpose — there is less topological constraint on input models: multiple polygonal models, models with ill-topology meshes, or even point clouds can be employed as skeletons.

Acknowledgement

The authors from the Hong Kong University of Science and Technology are partially supported by the HKUST6234/02E project. The author from the Chinese University of Hong Kong would like to thank the support from the projects CUHK/2050341.

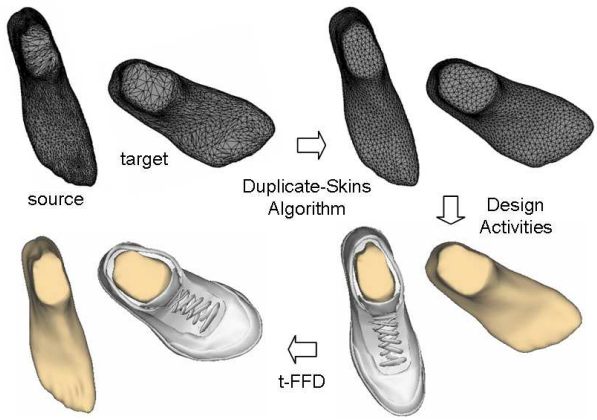

Figure 16: Design automation of shoes: for the given two foot models with position markers (top-left), our duplicate-skins can generate a pair of compatible meshes (top-right); after designing the shoe around the source model, we can automatically re-warp the shoe around the target model by t-FFD using the compatible meshes.

References

ALEXA, M. 2002. Recent advances in mesh morphing. *Computer Graphics Forum 21*, 2, 173–196.

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. *Computer Graphics Forum 22*, 3, 612–619.

ALLIEZ, P., UCELLI, G., GOTSMAN, C., AND ATTENE, M., 2005. Recent advances in remeshing of surfaces. Part of the state-of-the-art report of the AIM@SHAPE EU network.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste editing of multiresolution subdivision surfaces. *ACM Transactions on Graphics 21*, 3, 312–321.

BOTSCH, M., AND KOBBELT, L. 2001. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In *Proceedings of Eurographics 2001*, 402–410.

BOTSCH, M., AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. *Computer Graphics Forum 24*, 3, 611–621.

CARR, J., FRIGHT, W., AND BEATSON, R. 1997. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging 16*, 1, 96–107.

CARR, J., BEATSON, R., CHERRIE, J., MITCHELL, T., FRIGHT, W., MCCALLUM, B., AND EVANS, T. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of SIGGRAPH 2001*, 67–76.

COHEN-OR, D., SOLOMOVIC, A., AND LEVIN, D. 1998. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics 17*, 2, 116–141.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum 21*, 3, 209–218.

GOTSMAN, C., GU, X., AND SHEFFER, A. 2003. Fundamentals of spherical parameterization for 3d meshes. *ACM Transactions on Graphics 22*, 3, 358–363.

HILDEBRANDT, K., AND POLTHIER, K. 2004. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum 23*, 3, 391–400.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUDTZLE, W. 1993. Mesh optimization. In *Proceedings of SIGGRAPH 93*, 19–26.

JU, T. 2004. Robust repair of polygonal models. *ACM Transactions on Graphics 23*, 3, 888–895.

JULIUS, D., KRAEVOY, V., AND SHEFFER, A. 2005. D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum 24*, 3, 981–990.

KANAI, T., SUZUKI, K., AND KIMURA, F. 2000. Metamorphosis of arbitrary triangular meshes. *IEEE Computer Graphics and Applications 20*, 2, 62–75.

KARTASHEVA, E., ADZHIEV, V., PASKO, A., FRYAZINOV, O., AND GASILOV, V. 2003. Discretization of functionally based heterogeneous objects. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, 145–156.

KOBAYASHI, K., AND OOTSUBO, K. 2003. t-ffd: freeform deformation by using triangular mesh. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, 226–234.

KOBBELT, L., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001*, 57–66.

KRAEVOY, V., AND SHEFFER, A. 2004. Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics 23*, 3, 861–869.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: Constructing constrained texture maps. *ACM Transactions on Graphics 22*, 3, 326–333.

LEE, A., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresoltion mesh morphing. In *Proceedings of SIGGRAPH 99*, 343–350.

LEVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics 21*, 3, 362–371.

MARKOSIAN, L., COHEN, J., CRULLI, T., AND HUGHES, J. 1999. Skin: a constructive approach to modeling free-form shapes. In *Proceedings of SIGGRAPH 99*, 393–400.

MARSCHNER, S. R., GUENTER, B. K., AND RAGHUPATHY, S. 2000. Modeling and rendering for realistic facial animation. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, 231–242.

NGUYEN, M., YUAN, X., AND CHEN, B. 2005. Geometry completion and detail generation by texture synthesis. *The Visual Computer 21*, 8-10, 669–678.

OHTAKE, Y., AND BELYAEV, A. 2001. Mesh optimization for polygonized isosurfaces. *Computer Graphics Forum 20*, 3, 368–376.

OHTAKE, Y., AND BELYAEV, A. 2002. Dual/primal mesh optimization for polygonized implicit surfaces. In *Proceedings of the seventh ACM symposium on Solid modeling and Applications*, 171–178.

OHTAKE, Y., BELYAEV, A., AND PASKO, A. 2003. Dynamic mesh optimization for polygonized implicit surfaces with sharp features. *The Visual Computer 19*, 2-3, 115–126.

PRAUN, E., AND HOPPE, H. 2003. Spherical parameterization and remeshing. *ACM Transactions on Graphics 22*, 3, 340–349.

PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent mesh parameterization. In *Proceedings of SIGGRAPH 01*, 179–184.

SANDER, P., GORTLER, S., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parameterization. In *Proceedings of Eurographics Workshop on Rendering 2002*, 87–100.

SANDER, P., WOOD, Z., GORTLER, S., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 146–155.

SAVCHENKO, V., PASKO, A., OKUNEV, O., AND KUNII, T. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14*, 4, 181–188.

SCHREINER, J., PRAKASH, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. *ACM Transactions on Graphics 23*, 3, 870–877.

SEO, H., AND MAGNENAT-THALMANN, N. 2004. An example-based approach to human body manipulation. *Graphical Models 66*, 1, 1–23.

SHEFFER, A., AND HART, J. 2002. Seamster: Incospicuous low-distortion texture seam layout. In *Proceedings of IEEE Visualization 2002*, 291–298.

SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. 2002. Seamster: Incospicuous low-distortion texture seam layout. In *Proceedings of IEEE Visualization 2002*, 355–362.

SPANIER, E. 1966. *Algebraic Topology*. McGraw-Hill, New York.

SUMNER, R., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics 23*, 3, 399–405.

SURAZHSKY, V., AND GOTSMAN, C. 2003. Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 20–30.

TURK, G., AND O'BRIEN, J. 2002. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4, 855–873.

VORSATZ, J., RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. 2001. Feature sensitive remeshing. *Computer Graphics Forum 20*, 3, 393–401.

VORSATZ, J., RÖSSL, C., AND SEIDEL, H.-P. 2003. Dynamic remeshing and applications. In *Proceedings: 8th ACM Symposium on Solid Modeling and Applications (SM-03)*, 167–175.

WANG, C., WANG, Y., AND YUEN, M. 2005. Design automation for customized apparel products. *Computer-Aided Design 37*, 7, 675–691.

WANG, C. 2005. Parameterization and parametric design of mannequins. *Computer-Aided Design 37*, 1, 83–98.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surface. In *Proceedings of SIGGRAPH 94*, 247–256.

YNGVE, G., AND TURK, G. 2002. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics 8*, 4, 346–359.