Cross-Parameterization and Compatible Remeshing of 3D Models

Vladislav Kraevoy   Alla Sheffer

University of British Columbia, {vlady | sheffa}@cs.ubc.ca
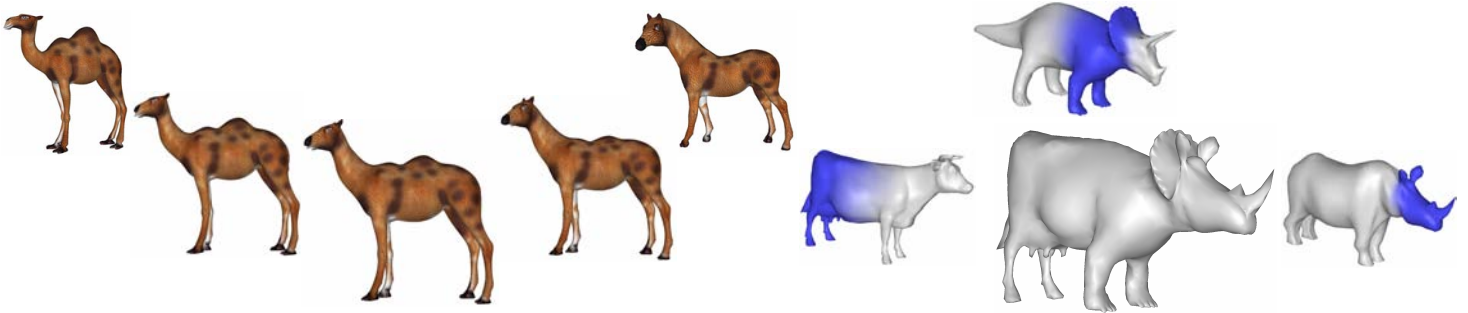
Figure 1: Applications: (left) texture transfer and morphing; (right) three-sided blending.

Abstract

Many geometry processing applications, such as morphing, shape blending, transfer of texture or material properties, and fitting template meshes to scan data, require a bijective mapping between two or more models. This mapping, or *cross-parameterization,* typically needs to preserve the shape and features of the parameterized models, mapping legs to legs, ears to ears, and so on. Most of the applications also require the models to be represented by *compatible* meshes, i.e. meshes with identical connectivity, based on the cross-parameterization. In this paper we introduce novel methods for shape preserving cross-parameterization and compatible remeshing. Our cross-parameterization method computes a low-distortion bijective mapping between models that satisfies user prescribed constraints. Using this mapping, the remeshing algorithm preserves the user-defined feature vertex correspondence and the shape correlation between the models. The remeshing algorithm generates output meshes with significantly fewer elements compared to previous techniques, while accurately approximating the input geometry. As demonstrated by the examples, the compatible meshes we construct are ideally suitable for morphing and other geometry processing applications.

Keywords: Modeling – Shape blending/Morphing; Modeling – Surface parameterization; Modeling – Polygonal modeling; Modeling – Mesh generation

1 Introduction

The ability to bijectively map one surface model to another is useful for many applications. Smooth morphing of one geometry into another is one of the oldest and most popular special effects in movies. The first stage of any morphing algorithm is the establishment of a bijective mapping between the models [Alexa 2000, Kanai et al. 2000, Lee et al. 1999]. Recent research in digital geometry processing suggests multiple new applications for such a mapping, including pair-wise model editing [Bier-

mann et al. 2002], transferring texture and surface properties (BRDFs, normal maps, etc…) [Praun et al. 2001], fitting template meshes to multiple data sets [Marshner et al. 2000, Allen et al. 2003], and principal component analysis. Many of these applications, such as blending and morphing, require *compatible* input meshes, i.e. meshes with identical connectivity. Therefore, given the cross-parameterization, the models must be remeshed with a common connectivity.

The models which need to be cross-parameterized usually have similar features (there is little use for mapping a phone onto a cow) and the parameterization must respect those. For example, when mapping between two humans, the legs must map to the legs, the ears to the ears, and so on. This is typically achieved by enforcing the correspondence for a small set of *feature vertices* and using a cross-parameterization that preserves the shape of the models as much as possible (in terms of angles and area).

1.1 Previous Work

Much of the previous work done in this area focused on morphing as the target application. Alexa [2002] gives a good recent review of cross-parameterization and compatible remeshing techniques developed for morphing. The cross-parameterization is typically computed by parameterizing the models on a common base domain. One popular choice is the sphere. There are a number of algorithms for spherical parameterization, e.g. [Alexa 2000, Gotsman et al. 2003, Praun and Hoppe 2003]. Of those, only Alexa's method addresses feature correspondence. However, it does not guarantee a bijective mapping and is not always capable of matching the features. An inherent limitation of a spherical parameterization is that it can only be applied to closed, genus zero surfaces.

A more general approach is to parameterize the models over a common base mesh [Lee et al. 1999, Lin et al. 2003, Michikawa et al. 2001, Praun et al. 2001]. This approach splits the meshes into matching patches with an identical inter-patch connectivity. After the split, each set of matching patches is parameterized on a common convex planar domain. One advantage of this approach is that it naturally supports feature correspondence by using feature vertices as corners of the matching patches. The main challenge in mapping the models to a single base mesh is to construct identical inter-patch connectivities. The vast majority of the methods use heuristic techniques that work only when the models have nearly identical shape. Praun et al. [2001] provide a robust method for partitioning both meshes into patches

given user-supplied base mesh connectivity. A common downside of existing base mesh techniques is that the patch structure severely restricts the freedom of the parameterization. As a result, the shape of the patches has a huge influence on the amount of mapping distortion.

Kraevoy et al. [2003] introduce an algorithm that, given a mesh, a set of feature vertices, and a set of corresponding positions in the plane computes a patch layout and a triangulation of the points that have identical connectivity. Since the method operates in 2D, the authors were able to use standard parameterization techniques to achieve low distortion. As explained in Section 3, our work extends this general framework to partition meshes into patches with identical connectivity.

When cross-parameterization is used for geometry processing, it is sometimes possible to limit the computation to disk-like parts of the surface [Bierman et al. 2002]. In this case, standard planar parameterization techniques (e.g. [Desbrun et al. 2002, Levy et al. 2002, Sheffer and de Sturler 2000]) can be used to construct the cross-parameterization. However, this limitation restricts the spectrum of available editing operations.

Given the cross-parameterization, many techniques [Alexa 2000, Kanai et al. 2000] generate the common connectivity for the models by overlaying the meshes in the parameter domain and computing a common intersection mesh. The new mesh captures the geometry of the models. However, typically the new mesh is a factor of ten larger than the input meshes and has very badly shaped triangles. The overlaying algorithm is also extremely tricky to implement, as it requires multiple intersection and projection operations. Another alternative is to remesh the models using a regular subdivision connectivity derived from the base mesh [Lee et al. 1999, Michikawa et al. 2001, Praun et al. 2001]. Due to the rigid connectivity structure, the shape of the mesh triangles reflects the shape of the base mesh. Hence, if the shape of the triangles is poor (because, for example, if the user picked unevenly spaced feature vertices) the shape of the mesh triangles will reflect this. More importantly, a model that contains features interior to the base mesh triangles (e.g. David's hair in Figure 9) will require a very dense subdivision mesh over the entire model. Lin et al. [2003] partly rectify this by introducing adaptive subdivision. However, their meshes still contain a very large number of elements.

Allen et al. [2003] use the connectivity of one mesh to approximate the geometry of another, avoiding explicit parameterization. Their solution is limited to very specific inputs and can introduce severe approximation errors when the input models have significantly different geometry.

A concurrent work by Schreiner et at. [2004] uses a procedure similar to the one below for base mesh construction, however, handling models of arbitrary genus more robustly. To generate a smooth cross-parameterization, they use a symmetric, stretch based relaxation procedure, which trades high computational complexity for quality of the mapping. The common mesh is generated using an overlay of the input meshes, described above. To avoid artifacts, the method has to relax feature vertex correspondence in some cases.

1.2 Contribution

In this paper we propose new methods for cross-parameterization and compatible remeshing. Our cross-parameterization method is the first one guaranteed to find a

bijective parameterization which satisfies the feature vertex correspondence between any number of genus zero models without additional input from the user. The method works for higher genus models as well, if the user specifies a sufficient number of feature vertices to define a correspondence between the handles (typically four vertices per handle).

We introduce a novel local framework for mesh parameterization over a base mesh domain. Our algorithm uses this framework to compute both shape-preserving and adaptive cross-parameterizations. The framework can be used to optimize other criteria, providing a powerful stand-alone parameterization tool.

After the cross-parameterization is computed, a novel adaptive method for compatible remeshing is applied. It generates meshes that accurately approximate the input geometry with significantly fewer elements than those generated by previous techniques. As a consequence, the method can be applied to generate compatible meshes for any number of models.

Thanks to the combination of shape preservation and good approximation, the resulting compatible meshes are well-suited for morphing and other geometry processing applications.

The rest of the paper is organized as follows. Section 2 provides an overview of the algorithm. Section 3 describes the construction of the common base mesh. Section 4 defines the cross-parameterization function between the models. Section 5 describes the compatible remeshing algorithm. Section 6 demonstrates the resulting cross-parameterizations and showcases several of their applications. Finally, Section 7 summarizes the algorithm and suggests future research directions.

2 Algorithm Overview

The algorithm as described below can handle any number of input meshes simultaneously. However for efficiency reasons it is simpler to perform the procedure on a pair-wise basis (Section 2.3). To simplify the notations, in the rest of the paper we will describe the process for two input models.

2.1 Definitions

The following terms are used in the algorithm description:
- A *cross-parameterization* of two surface meshes M_s and M_t is a one-to-one and onto mapping between the two surfaces.
- A *patch layout P* of a mesh M induced by a set of vertices V is a partition of the mesh into simply connected, non-overlapping patches where the boundary of each patch is defined by non-intersecting edge paths in M between vertices in V. A layout is *triangular* if the boundary of each patch is defined by three paths.
- Given two meshes M_s and M_t with corresponding sets of vertices V_s and V_t ($|V_s|=|V_t|$), the layouts P_s and P_t are *topologically identical* if each path (v_s^i,v_s^j) in P_s corresponds to a path (v_t^i,v_t^j) in P_t and vice-versa. We refer to such pair of paths as *matching* pair.
- A *base triangle b* for a triangular patch $p=(v^i,v^j,v^k)$ is the straight-line planar triangle in 3D formed by the coordinates of v^i,v^j, and v^k. A *base mesh B* of the patch layout is the union of the base triangles.

2.2 Algorithm Stages

The input to the algorithm consists of two closed manifold meshes M_s and M_t and the corresponding sets of matching fea-

ture vertices V_s and V_t, typically selected by the user. The algorithm has three main stages. First, we construct a common base domain. Second, a low distortion cross-parameterization is computed. The final stage compatibly remeshes the input models using the parameterization.

Common base domain construction: The goal of this stage is to construct a common base mesh domain for M_s and M_t based on the matching feature vertices V_s and V_t. To construct it, we compute topologically identical triangular layouts of the two meshes (Section 3). The layouts are constructed incrementally by adding pairs of matching paths between feature vertices.

Cross-parameterization: After the base mesh domain is constructed, each patch is mapped to the corresponding base mesh triangle. This provides two parameterizations F_s and F_t between the meshes M_s and M_t and their corresponding base domains B_s and B_t. By mapping each triangle in B_s to the matching triangle in B_t we obtain the initial cross-parameterization F (Section 4.1). The algorithm then smoothes the mappings F_s and F_t from the source and target meshes to their respective base mesh domains (Section 4.2). This improves the shape of the mesh patches and drastically reduces the distortion of the combined cross-parameterization F.

Compatible remeshing: The algorithm described in Section 5 constructs compatible meshes for the two models. It uses the source mesh M_s as the basis for remeshing. The method generates a mesh M_{st} for the geometry of the target mesh M_t with the same connectivity as M_s. This is done by mapping the vertices of M_s onto M_t using F. Initially, M_{st} does not necessarily provide a good approximation of the target geometry. Therefore the algorithm adapts M_{st} to the target geometry using a combination of smoothing (Section 5.2) and refinement (Sections 5.3 and 5.4) operations.

The result of this procedure is a shape-preserving cross-parameterization between two compatible meshes M_s and M_{st} where M_{st} closely approximates the geometry of the target mesh M_t. The cross-parameterization maps the feature vertices of one model to the corresponding ones on the other and is guaranteed to be bijective. The applications of the parameterization are demonstrated in Section 6. The stages of the algorithm are described in detail in the following sections.

2.3 Multiple Inputs

To efficiently and robustly handle multiple input models we apply a sequential procedure which incrementally generates the common connectivity for all the models. The method selects one model as the source (arbitrarily, or as the one with highest resolution, or best mesh quality). It then computes the cross-parameterization (Sections 3 and 4) between the source model and each of the other models. A mapping between any pair of models is defined by combining appropriate cross-parameterization functions. Finally a single common connectivity is computed incrementally, starting with the connectivity of the source and adding one model at a time. At each stage the remeshing procedure (Section 5) is used to fit the common mesh to a new model refining the connectivity as necessary. This procedure was used to generate the triceratops/rhino/cow blends in Figures 1 and 10.

## 3 Constructing a Common Base Domain

The common base domain is the basis for the subsequent cross-parameterization. It is constructed by computing topologically

identical triangular patch layouts of the input meshes M_s and M_t. The layouts define a common base mesh connectivity.

A related, but simpler, problem of constrained planar parameterization was addressed by Kraevoy et al. [2003]. The authors described an algorithm that computes a triangular patch layout and a triangulation of a set of positions in the plane such that both patch layout and triangulation have the same connectivity. We extend the proposed framework to compute the topologically identical patch layouts.

The layouts are computed by incrementally adding pairs of matching edge paths between feature vertices. As defined above, edge paths match if both their start and end vertices correspond to one another. The addition process is explained in detail in Sections 3.1 and 3.2. When adding each pair of matching paths, the following conditions have to be satisfied:
1. **Intersection:** The new paths must not intersect existing paths in either of the mesh layouts P_s and P_t.
2. **Cyclical Order:** The paths must have the same cyclical order around the end vertices (Figure 2 (a)). Consider a feature vertex v_s^i which has the paths $s^1,....,s^n$ emanating from it, in clockwise order. For a topologically identical layout to exist, the vertex v_t^i must have the same clockwise path order as v_s^i.
3. **Blocking:** The new paths must not block necessary future paths. Blocking happens when a path splits a patch in two and a feature vertex v_s^i and the corresponding vertex v_t^i are placed in topologically different patches (Figure 2 (b,c)). Blocking is tested by propagating fronts from the same side of both paths (as determined by mesh orientation) and comparing the feature vertices encountered by the front.
4. **Handle Blocking:** For models of genus greater than zero two other types of blocking must be considered:
   4.1 If a patch has interior handles a path connecting two boundary vertices can reduce the genus of the patch without splitting it. One case of handle blocking is therefore when the path on one model splits the patch in two and on the other only reduces the genus. The regular blocking test, above, locates this type of handle blocking.
   4.2 The second case of handle blocking happens when a path splits a patch into two on both models and the resulting matching patches have different genus. This is tested by simply comparing the genus of the new patches using the Euler formula. In practice this test is redundant if there are enough feature vertices around each handle (in this case the regular blocking test will fail as well).

(a)   (b)   (c)
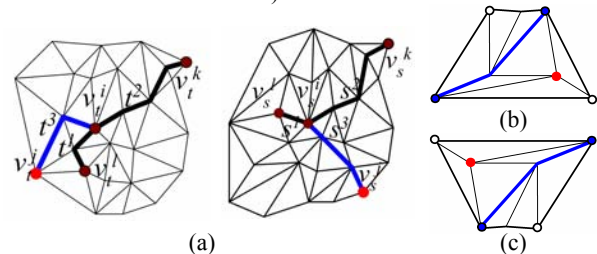
Figure 2: Path validity conditions. (a) Matching paths s^3 and t^3 (blue) violate cyclical order. Blocking paths (blue): (b) path in P_s; (c) matching path in P_t. This pair of paths places the red feature vertex on different patches on P_s and P_t.

To satisfy these conditions we introduce matching paths in two stages: first, adding as many paths as we can without modifying the connectivity of the source and target meshes; then introduc-

ing the necessary remaining paths refining the meshes when needed as we go along.

3.1 Adding Edge Paths

This stage of the algorithm traces matching paths on the edge graphs of the source and target meshes.

```
Algorithm PathMatch
    M_s' = Ms
    M_t' = Mt
    Compute the shortest paths s^ij for each pair of vertices in V_s
    Compute the shortest paths t^ij for each pair of vertices in V_t
    ST = ∅
    foreach s^ij
        ST ← <s^ij,t^ij> /* pairs of matching paths */
    while ST ≠ ∅
        < s,t> =ST.RemoveShortest()
        if NonBlocking(s,t)
            Add s to P_s;   Add t to P_t
            Remove all interior vertices of s from M_s'
            Remove all interior vertices of t from M_t'
            Update(ST, s, t)
        end
    end
end
```

The function *NonBlocking* tests the blocking condition as described above. The method *ReturnShortest* returns a pair of paths with the smallest length sum ($|s^{ij}| + |t^{ij}|$). The *Update* function modifies *ST* as follows:
1. Remove all paths s^{ij} containing interior vertices of s and recompute them in M_s'.
2. Remove all paths t^{ij} containing interior vertices of t and recompute them in M_t'.
3. For each path s^{ij} emanating from one of the end vertices of s, check that s^{ij} and t^{ij} are in the same segment with respect to the cyclical ordering around the end vertex. If they are not, recompute the paths, starting in the correct segment.

Note that given the new constraints, the *Update* procedure can find that it may not be able to locate replacement paths for s^{ij} or t^{ij}. In this case, the pair is simply removed from *ST*. Using the *Update* function eliminates the need to check intersection and cyclical order conditions explicitly when the paths are added. As a result of the on-the-fly recomputation performed by *Update,* many more paths can be added at this stage of the algorithm.

At the end of the *PathMatch* algorithm we have a topologically identical partition of both meshes into patches (Figure 4 (a),(b)). If all the patches are triangular, we are done; otherwise, the procedure described in Section 3.2 is applied.
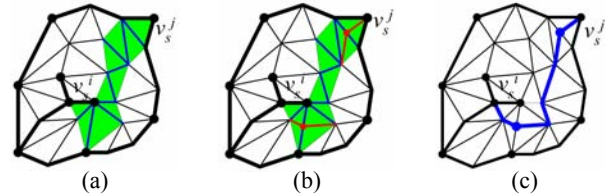
3.2 Adding Face Paths

The *PathMatch* algorithm terminates when no more matching paths can be traced on the edge graphs of the input meshes. Therefore, the second stage of the algorithm traces matching paths on the face graphs instead. We use the restricted brushfire algorithm [Praun et al. 2001] for tracing the paths. Since the face graph of each mesh patch is connected, the brushfire can find a correctly cyclically oriented path between any given pair of vertices. Paths are added until all the patches are triangulated. We guarantee termination by specifying an order for adding paths that avoids blocking [Kraevoy et al. 2003]. First, if any of the patches are not simple, namely they contain multiple boundary loops, the method connects the loops generating simple

patches. Then the algorithm triangulates the resulting simple patches.

The procedure for tracing paths on faces can fail if patches have genus bigger than zero. In our tests, this happened only if there were not enough feature vertices around each handle.

Make simple: The algorithm connects all boundary loops for each pair of non-simple corresponding patches p_s and p_t as follows:
1. Compute the shortest paths between each pair of vertices on different boundary loops of p_s. Select the shortest path among those.
2. Compute a matching path on p_t.
3. Convert both face paths into edge paths (Figure 3). Each step of the conversion traces an edge from a corner vertex of a face to a vertex on the opposite, consecutive face in the path. If the edge shared by the faces has a non-boundary vertex, the method uses the edge from the corner to it. Otherwise, the edge is split (Figure 3(b)) and the edge between the corner and the midpoint is used.

(a)   (b)   (c)
Figure 3: Constructing an edge path (v_i, v_j) from a face path: (a) face path; (b) adding new vertices; (c) derived edge path.

4. Repeat the process if the patches have multiple loops.

Triangulation: Simple patches are triangulated by recursively adding paths between non adjacent boundary vertices. Each addition of a pair of paths splits the patches into two. The procedure is then called recursively for both halves until triangular patches are obtained. The face paths are converted into edge path as described above.

The procedure produces topologically identical triangular layouts of the two input meshes.

3.3. Flipping Paths

The partitioning method described above tries to introduce the shortest paths, which typically results in reasonably shaped patches. The smoothing (Section 4) improves the shape of the patches further. However, the smoothing effectiveness is restricted by the connectivity of the mesh.

To improve the connectivity we introduce a simple path flip procedure:
1. For each path $s=(v^i, v^j)$ in P_s check if

$$deg(v^i) + deg(v^j) > deg(v^r) + deg(v^l) + 2$$

where v^r and v^l are the feature vertices opposite s in the right and left patches, and *deg* returns the valence of the vertex (number of emanating paths).
2. If the condition is satisfied, the path s is removed from P_s. The flipped path (v^r, v^l) is added instead. The same change is applied to the layout P_t.
3. The path flipping loop terminates when no more paths can be flipped.

The construction method is guaranteed to terminate and to compute topologically identical triangular layouts for closed genus zero surfaces. For genus greater than zero, as explained above, the method will terminate successfully if enough vertices are placed around each handle. The computed layouts are used to generate the cross-parameterization as explained in the next section.

(a)   (b)

(c)   (d)

(e)   (f)
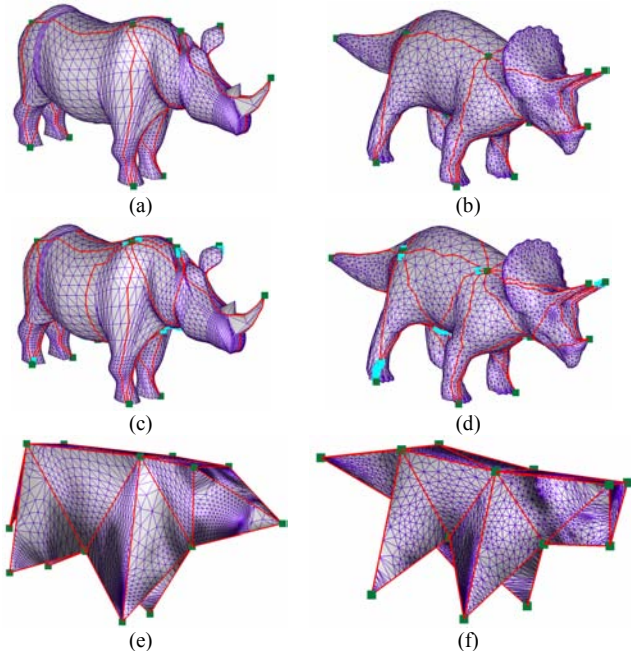
Figure 4: Base domains construction (feature vertices are dark green): (a),(b) edge paths; (c),(d) face paths, new vertices are highlighted (turquoise); (e),(f) base meshes.
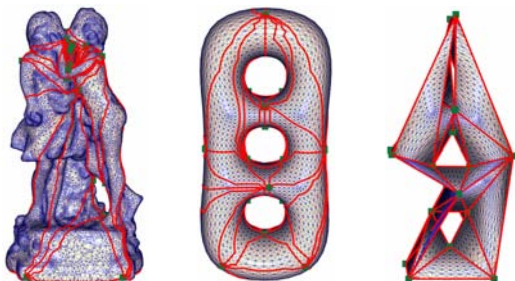
Figure 5: Base domain construction for genus 3 models (19 feature vertices).
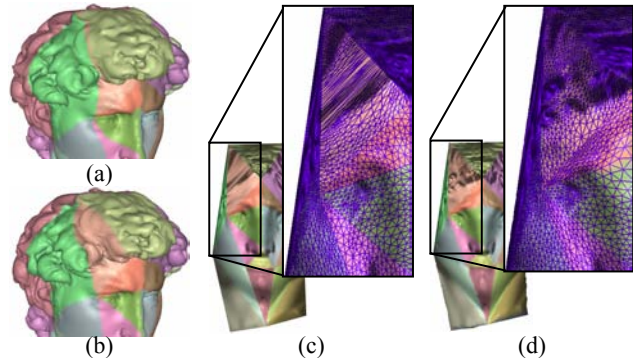
4 Cross-Parameterization

4.1 Initial Cross-Parameterization

Given the patch layouts P_s and P_t, we construct the two base meshes B_s and B_t. The initial cross-parameterization is defined as follows:

1. Map each patch p in P_s to the corresponding base triangle b using the mean value parameterization [Floater 2003]. This defines a mapping F_s from M_s to B_s.
2. Define F_t from M_t to B_t in the same way.
3. Map each triangle b_s in B_s to the matching triangle b_t in B_t, mapping the corner vertices to the corresponding corners and using barycentric coordinates for the interior. This defines a mapping F_{st}.
4. The mapping F from M_s to M_t is

$$F = F_t^{-1} \cdot F_{st} \cdot F_s.$$

Since each component of the mapping is bijective, the combined parameterization is bijective as well. Mean value parameterization computes a shape preserving (conformal) parameterization. However if the patches are not well shaped the resulting mapping can still be quite distorted (Figure 6 (c)). To reduce the parameterization distortion we apply a smoothing procedure described in Section 4.2.

(a)

(b)   (c)   (d)

Figure 6: Parameterization on a base mesh: zoom-in on patches (a) and mapping to base mesh (c) before and after smoothing (b) and (d).

4.2 Smoothing

The quality of any parameterization between a set of patches and a base mesh is highly dependent on the shape of the patches. Therefore any parameterization that preserves the original patches will have low distortion only if the patches are well shaped. We introduce the following smoothing algorithm which drastically reduces the parametric distortion by letting vertices migrate from one patch to another, improving the patch shape (Figure 6 (a), (b)).

To guarantee a bijective mapping, the smoothing procedure requires an **adjacency** constraint to be satisfied for all edges in the mesh. An edge satisfies the adjacency constraint if its end vertices belong to the same patch or to two adjacent patches that share a common boundary path. The following pre-processing procedure modifies the meshes M_s and M_t to comply with this constraint. After the initial cross-parameterization the paths are aligned with the edges of the base mesh (Figure 6 (c)). Hence, the only edges that violate the adjacency constraint at this stage are edges whose vertices lie on two bounding paths of a single patch. Prior to running the smoothing procedure, each such edge is split into two. During smoothing, vertices are prevented from migrating from one patch to another if one of their edges will violate the adjacency constraint.

The smoothing is performed independently for M_s and M_t, modifying the current mapping from the mesh to the base domain. This is done by changing the locations on the base domain to which the mesh vertices are mapped. The base mesh location of a vertex v is defined as a pair $\langle b, v_b \rangle$ consisting of base triangle index b and the barycentric coordinates v_b defined with respect to the base triangle.

The smoothing procedure iterates over the mesh vertices, repeatedly modifying their locations on the base mesh. Given a vertex v located at (b, v_b), each smoothing iteration relocates the vertex based on the locations of the neighboring vertices using the fol-

lowing local parameterization procedure (demonstrated in Figure 7):

1. Set b_1, b_2 and b_3 to the three triangles adjacent to b.
2. Map b and the three adjacent triangles (b_1, b_2 and b_3) to a planar equilateral triangle E. Each triangle is mapped to a corresponding quarter of E (b_0', b'_1, b'_2 and b'_3 respectively).
3. Map v to b_0' using the barycentric coordinates v_b. Compute the coordinates v_E of the vertex with respect to E.
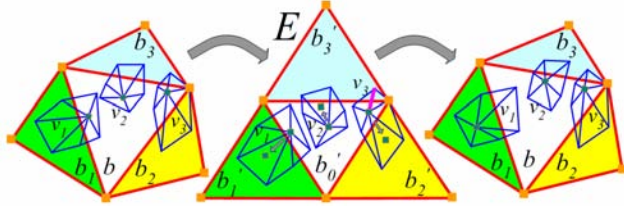

Figure 7: Relocating vertices on base mesh.

4. Map each neighbor vertex of v to E. Due to the adjacency constraint, each neighbor of v is located on one of the four base triangles b, b_1, b_2 and b_3. Hence, this mapping is well defined. Each neighbor vertex u located at $<b_u, u_b>$ is first mapped to b_u' using its barycentric coordinates u_b. The algorithm than computes the vertex's coordinates u_E with respect to E.
5. Given the locations of the neighbor vertices, compute a new location v_E based on the neighbors:

$$v_E = \frac{1}{|Nei(v)|} \sum_{u \in Nei(v)} w_{uv} u_E.$$

The weights w_{uv} are the mean value edge weights [Floater 2003] computed on the original mesh.

6. Check if moving the vertex to the new location will flip any triangles. If yes, restore previous coordinates.
7. Find the quarter b_i' that v_E is in.
8. Compute the barycentric coordinates v_i of v_E with respect to b_i'.
9. If i is zero the vertex remains in b. Set the location of v to $<b, v_i>$, (e.g. Figure 7, vertex v_2).
10. Otherwise:
    10.1 Check if moving v from b to b_i will violate the adjacency constraint for any of its edges, (e.g. Figure 7, highlighted edge incident on vertex v_3).
    10.2 If the constraint is satisfied for all the edges, update the location of v to $<b_i, v_i>$, (e.g. Figure 7, vertex v_1).

The procedure is repeated for each of the mesh vertices.

The vertex relocation in E explicitly checks for flipped triangles. Thanks to the adjacency condition, the mapping between the base mesh and E is bijective. In this way, the mapping between the meshes and the base domains is guaranteed to remain bijective after the smoothing.

The smoothing is repeated until the vertices no longer move or until a fixed number of iterations is reached. The procedure redistributes the vertices between patches, straightening the patch boundaries and significantly reducing the parametric distortion (Figure 6). Since paths are no longer mapped to base mesh edges, we no longer experience the artifacts resulting from mapping curved paths to straight lines.

The use of four base triangles for smoothing the mesh on the interior triangle provides a flexible and robust parameterization tool. Its impact is quite similar to the global parameterization tool used by Khodakovsky et al. [2003]. However, using one local

parameterization at a time provides a significantly cheaper means of achieving this effect compared to the global linear solver used by Khodakovsky et al. The parameterization framework introduced here is used in the following section to remesh the models compatibly.

5 Compatible Remeshing

The cross-parameterization, computed in Section 4, is used to generate compatible meshes for the input models. Existing compatible remeshing techniques are based on mesh overlaying or subdivision connectivity construction. For complex models both approaches tend to generate meshes that are an order of magnitude larger than the input meshes. We now describe a novel remeshing algorithm that computes significantly more compact compatible meshes.

The algorithm first remeshes the target model with the connectivity of the source mesh. The algorithm uses the cross-parameterization to map the vertices of the source mesh M_s to the target model M_t, creating a mesh M_{st} with the same connectivity as M_s. This yields compatible meshes for both models. However the mesh M_{st} may be a very poor approximation of the target geometry (Figure 8 (b)) in curved regions. The algorithm than improves the geometry approximation by vertex relocation and refinement. Since we want to minimize the final vertex count, refinement is performed only when strictly necessary.
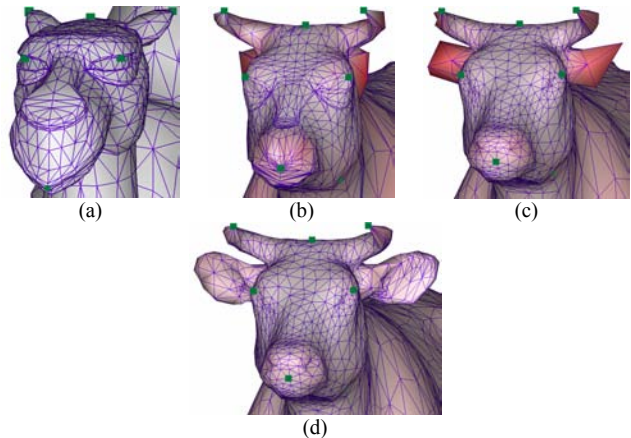

Figure 8: Compatible remeshing: (a) source mesh. (b) M_{st} after initial projection. (c) M_{st} after smoothing. (d) final mesh (after smoothing and refinement). There are no features on the camel corresponding to the cow's ears (mapping the camel's ears to the cow's horns provides a more natural morphing sequence).

The stages of the remeshing algorithm are as follows:
1. Compute the distance approximation error across the surface (Section 5.1).
2. While error is above a given threshold:
    2.1. Relocate the vertices of M_{st} on the target model using a smoothing procedure (Section 5.2). Recompute the distance approximation error.
    2.2. Refine the meshes where necessary using edge splits (Section 5.3).
3. Compute normal approximation error and perform "pseudo" edge-flip refinement (Section 5.4).

All connectivity changes are applied to both M_{st} and M_s. Note that, compared with regular remeshing, this procedure does not remove mesh vertices or flip edges, since both vertex removals

and edge flips can potentially distort the geometry of M_s. We now describe each of the algorithm stages in more detail.

5.1 Error Computation

At each stage of the remeshing we need to measure the approximation error between the target model M_t and its approximation M_{st}. Such error is typically measured using a discrete Hausdorff metric, which measures the distance from the vertices of one surface to the other surface. In our case, the vertices of M_{st} lie on M_t so the one-sided error is zero. We therefore need to measure only the distance between the vertices of the target mesh M_t and the approximation surface. Computing the exact distance is very time consuming. Instead we use our parameterization to efficiently compute a tight upper bound. We define the mapping F' between M_s and M_{st} by mapping each vertex of M_s to the corresponding vertex of M_{st} and by using barycentric coordinates inside each triangle. Note that F' is **not** identical to F, as it is based only on the mapping of the vertices of M_s. The distance error $e(v)$ at each vertex v is defined as

$$e(v) = (F'F^{-1}(v) - v) \qquad (1)$$

To eliminate high frequencies and spread the error around a larger region of the mesh, we apply Gaussian smoothing, averaging the error at each vertex with that of the neighboring vertices. The error inside each triangle of M_t is computed from the vertex errors using barycentric coordinates. For any point p on the new mesh M_{st}, we map p to the target mesh using $F(F')^{-1}$ and use the error at that location. The face coloring in Figure 8 shows the approximation error throughout the remeshing procedure, with red denoting areas of higher error.

5.2 Adaptive Smoothing

The adaptive smoothing procedure modifies the parameterization of M_s on the base mesh B_s. This modifies the mapping F and as a consequence moves the vertices of the projected mesh M_{st} along the target mesh M_t. We apply the same smoothing procedure that was used for computing shape-preserving cross-parameterization (Section 4.2). The only difference between the two procedures is the choice of weights. The adaptive smoothing algorithm assigns weights w_{uv} to the edges (u,v) using the following formula.

$$e_{uv} = \left(\frac{e(u) + e(v)}{2} + e(\frac{u+v}{2})/2 \right) \qquad (2)$$
$$w_{uv} = (e_{uv} + w_{uv})/2$$

The edge error e_{uv} is based on a combination of the error at the end vertices and the midpoint of the edge (in curved regions, the mid-point error can be high even if it is low at both ends). The weight is the combination of the current error with the previous weight. This choice of weights attracts vertices to areas of higher error. The averaging with previous value of w_{uv} is used to avoid high weight fluctuations. The result is gradual mesh smoothing which prevents vertices from jumping back and forth when the error increases in one location and decreases in another.

Those weights are passed to the smoothing algorithm (Section 4.2) which computes the new mapping F. After each smoothing iteration, the positions of the vertices of M_{st} are recomputed.

5.3 Refinement

Adaptive smoothing goes a long way towards accurately approximating the target mesh with the projected mesh M_{st}. Clearly, however, if the target contains complex features not available in the source model (e.g. the cow's ears in Figure 8), they cannot be captured by the source connectivity. Therefore, when smoothing alone can not approximate the geometry accurately, the local mesh needs to be refined. This is done by performing edge split operations. The refinement uses the same error function e_{uv} (Equation 2) as the smoothing. An edge is refined if

$$e_{uv} > \max(\frac{1}{2} \max_{(u',v')}(e_{u'v'}), \varepsilon),$$

where ε is the user-defined approximation error threshold. We found that refining only the edges with error larger than half of the current maximal error provides sufficient approximation accuracy without adding too many vertices to the final mesh. When the computed error exceeds the current threshold, the edge is split by placing a vertex at its mid-point. The split is performed simultaneously on M_{st} and M_s, thus preserving the mesh compatibility.

To avoid unnecessary refinement edge splitting is performed only when smoothing alone no longer decreases the global error.

5.4 Pseudo Edge Flips

The smoothing and refinement loop of the remeshing algorithm terminates when the approximation error at the vertices of M_t is sufficiently low. Therefore, in terms of absolute distance between the surfaces, the approximation is sufficiently accurate. However there can still be large deviation between the face normals of M_t and M_{st}. This is a well-known problem in remeshing and is typically resolved by edge flips. However, since the meshes M_s and M_{st} have to remain compatible, an edge flip can potentially distort the geometry of M_s. We detect the edges to be flipped using a standard normal deviation test. For each such edge we compute the midpoints of the flipped edge. Instead of flipping the edge we split it, placing a new vertex at the computed location. This operation reproduces the change in normals achieved by a flip.

After all the flips are completed, we have two compatible meshes describing the source and target geometries: the mesh M_s is an exact replica of the source geometry; the projected mesh M_{st} approximates the target geometry up to a user prescribed tolerance. New meshes, providing visually acceptable morphing and blending results, typically have about ten to twenty percent more vertices than the source mesh (Table 2), compared to an order of magnitude increase using previous methods.

| (a) | (b) | (c) | (d) |
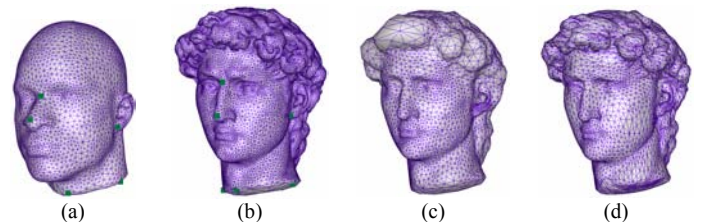
Figure 9: Mapping a coarse template mesh (a) to a fine target geometry (b). The remeshing approximates David's detailed hair starting from a coarse bold headed source. (c) M_{st} before remeshing; (d) final mesh.

6 Experimental Results

Figures 1, 9, 10, and 11 demonstrate the applications of the algorithm. In Figure 9, a coarse source mesh (4K vertices) is used to remesh a highly detailed target (20K vertices). The final connectivity contains 6764 vertices and approximates the target model with an L_2 error of 0.13% (measured as a percentage of the bounding box diagonal). This is well within the range of errors computed by regular (non-compatible) remeshing methods. This example demonstrates the method's ability to approximate dense models accurately with a much coarser template mesh, providing a generic alternative to template fitting methods such as [Allen et al. 2003] which use techniques carefully tailored to a particular type of models.
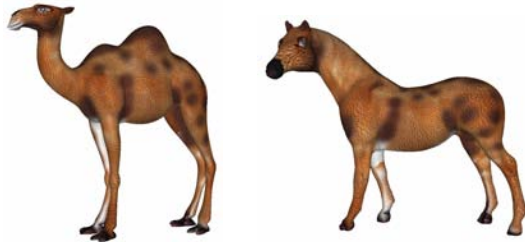

Figure 10: Texture transfer from camel to horse.

Figure 10 demonstrates texture transfer performed using the cross-parameterization. By texturing the compatible meshes the texture can be used throughout a morphing sequence (Figure 1).

Figures 1 and 11 showcase the use of the computed compatible meshes for blending and morphing. The three-sided blends of a cow, rhinoceros and triceratops highlight the method's ability to simultaneously cross-parameterize and remesh multiple models. The cup/teapot and feline/triceratops examples demonstrate the method's ability to handle high genus models. The genus 2, feline/triceratops example required 28 constrained vertices. The examples include weighted local blends that paste parts of different models together. This feature is very useful for model editing.

Some cross-parameterization statistics (before remeshing) are shown in Table 1. The conformality is measured using the formula introduced by [Sheffer and de Sturler 2000]. The L_2 stretch is measured as described in [Sander et al. 2001]. Note that ideal L_2 stretch is 1 while ideal E_{Ang} is 0. Clearly, the distortion is highly dependent on the level of similarity between the models. Note that the cross-parameterization uses mean-value coordinates aimed at reducing angular distortion.

Models (source/target)	Input sizes(#v)	# Feature vertices	E_{Ang}	L_2 stretch
teapot/cup	6988/1725	25	0.12	0.05
head/David (Figure 9)	4000/20000	10	0.04	0.86
triceratops/rhinoceros	4000/3552	15	0.18	0.49

Table 1: Parameterization statistics.

Models (source/target)	Input sizes (#v)	Output size	L_2 error (%)	PSNR
camel/cow	2444/2904	3357	0.12%	58.4
teapot/cup	6988/1725	7114	0.14%	57.1
triceratops/rhinoceros/cow	4000/3552/2904	4541	0.13%	57.7
camel/horse	39074/3500	39740	0.016%	75.9

Table 2: Remeshing statistics.

Table 2 shows remeshing statistics, including the L_2 error computed using $e(v)$ (Equation 1) and the Peak Signal to Noise Ratio (PSNR) derived from it. Note that $e(v)$ approximates the Hausdorff metric from above. The algorithm takes about two minutes to run on models of up to 10K triangles on a 3GHz Pentium IV.

7 Conclusions

In this paper, we have introduced new methods for computing shape preserving cross-parameterizations and compatible remeshing. The parameterization method is guaranteed to find a bijective, feature-preserving parameterization for any set of models. The smoothing mechanism we propose significantly reduces the mapping distortion compared to other parameterization methods. Our remeshing method generates compatible meshes for both models with fewer vertices than previous methods while accurately approximating the input geometry.

Compared to previous techniques, this method is significantly less dependent on the shape of the patches. However, visible artifacts can still occur when patch vertices have very high valence, requiring special care during feature selection. Our future research will explore ways to alleviate this problem. Another problem we plan to address is automatic selection of matching features.

Acknowledgements

References

ALEXA, M. 2000 Merging Polyhedral Shapes with Scattered Features, *The Visual Computer 16*, 26-37.

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, *ACM Transactions on Graphics*, 22, 3, 587-594.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste Editing of Multiresolution Surfaces. *ACM Transactions on Graphics*, *21*, 3, 312-321.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic Parameterizations of Surface Meshes. *In Proceedings of Eurographics 2002*, Blackwell Publishing, Saarbrucken, G. Drettakis and H.-P. Seidel, Eds., Computer Graphics forum, 21, 3, 210-218.

FLOATER, M.S. 2003. Mean-value coordinates. *Computer Aided Geometric Design*, 20, 19-27.

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

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

KHODAKOVSKY, A., LITKE, N., SCHRÖDER, P., 2003, Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics*, 22, 3, 350-357.

KRAEVOY, V., SHEFFER, AND A., 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, Multiresolution Mesh Morphing. In *Proceedings ACM SIGGRAPH 1999*, 343-350.

LÉVY, 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.

LIN, J.L., CHUANG, J.H., LIN, C.C., AND CHEN, C.C., 2003, Consistent parametrization by quinary subdivision for remeshing and mesh metamorphosis, In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia*, 151-158.

MARSCHNER, S., GUENTER, B., AND RAGHUPATHY, S., 2000, Modeling and Rendering for Realistic Facial Animation. *Rendering Techniques 2000*, 231-242.

T. MICHIKAWA, T. KANAI, M. FUJITA, H. CHIYOKURA, 2001. Multiresolution Interpolation Meshes, *In Proc. 9th Pacific Graphics International Conference*, IEEE CS Press, 60-69.

PRAUN, E., HOPPE, H., 2003, Spherical parametrization and remeshing. *ACM Transactions on Graphics* 22, 3, 340-349.

PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent Mesh Parameterizations. In *Proceedings of ACM SIGGRAPH 2001*, E. Fiume, Ed.., Computer Graphics Proceedings, Annual Conference Proceedings, 179-184.

SANDER, P. V., SNYDER, J., GORTLER, S., AND HOPPE, H. 2001. Texture Mapping Progressive Meshes. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Proceedings, 409-416.

SCHREINER, J., PRAKASH, A., PRAUN, E., HOPPE, H., 2004, Inter-Surface Mapping, *ACM Transactions on Graphics*, to appear.

SHEFFER, A., AND DE STURLER, E. 2000. Surface Parameterization for Meshing by Triangulation Flattening. In *Proceedings of the 9th International Meshing Roundtable*, 161-172.
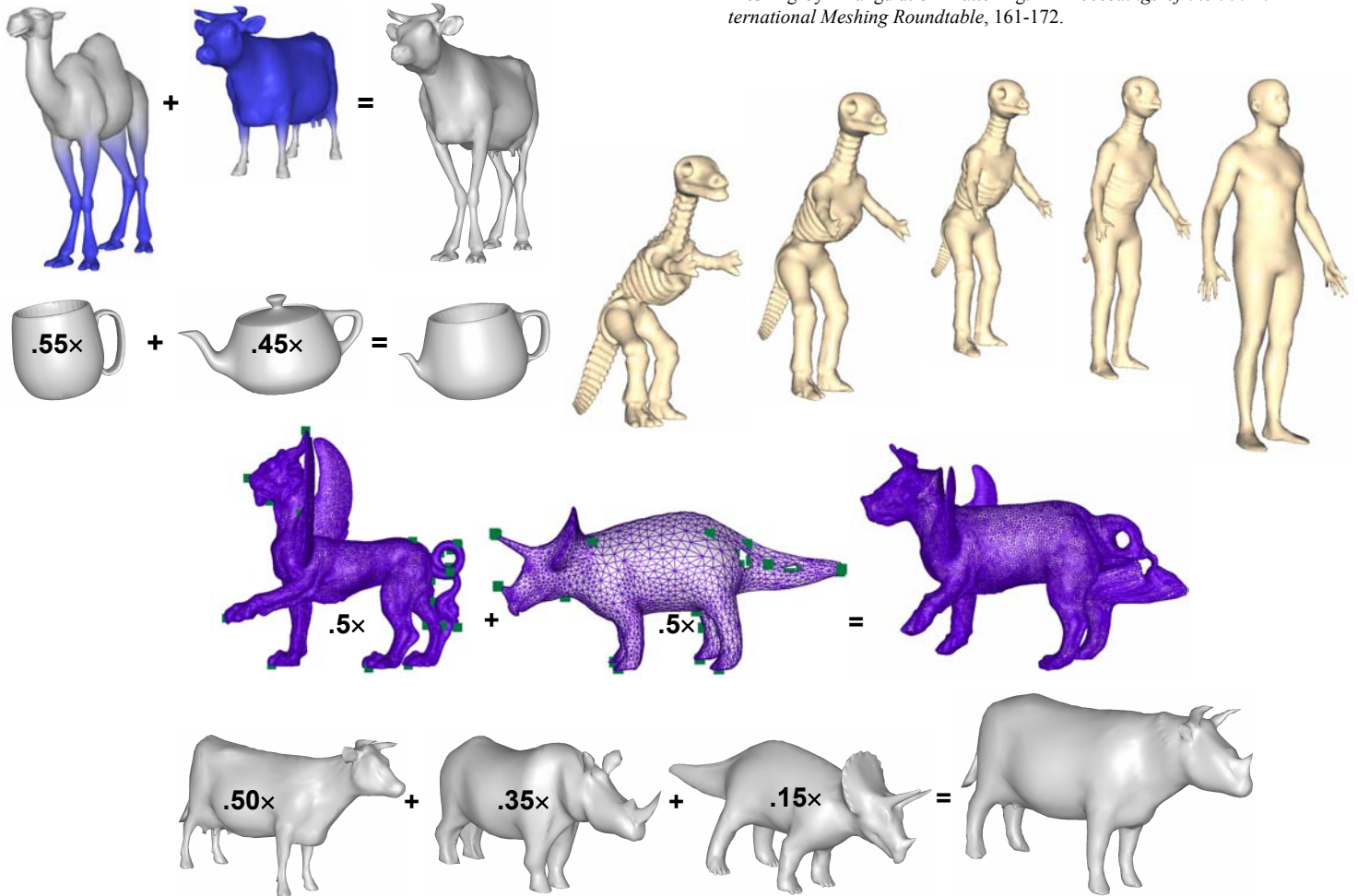
Figure 11: Blending and morphing of different models. The numbers in the blends are the affine combination weights of each model.