Vision, Modeling, and Visualization (2012)
M. Goesele, T. Grosch, B. Preim, H. Theisel, and K. Toennies (Eds.)

A mixed shape space for fast interpolation of articulated shapes

S. Marras T. J. Cashman K. Hormann

University of Lugano, Switzerland

Abstract

Interpolation between compatible triangle meshes that represent different poses of some object is a fundamental operation in geometry processing. A common approach is to consider the static input shapes as points in a suitable shape space and then use simple linear interpolation in this space to find an interpolated shape. In this paper, we present a new interpolation technique that is particularly tailored for meshes that represent articulated shapes. It is up to an order of magnitude faster than state-of-the-art methods and gives very similar results. To achieve this, our approach introduces a novel space shape that takes advantage of the underlying structure of articulated shapes and distinguishes between rigid parts and non-rigid joints. This allows us to use fast vertex interpolation on the rigid parts and resort to comparatively slow edge-based interpolation only for the joints.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations

1. Introduction

For processing geometric mesh data, the concept of a *shape space* has proved invaluable for creating [SP04, CH12], interpolating [KMP07, WDAH10], posing [LSLCO05, FB11], and editing [KG08, YYPM11] static shapes or dynamic geometry sequences. For a given mesh connectivity graph $\mathcal{G}$, an associated shape space $\mathcal{S}$ allows every geometric realization of $\mathcal{G}$ to be represented as a point in $\mathcal{S}$. The simplest such shape space is $\mathcal{S}_v = \mathbb{R}^{3V}$, to store the coordinates of V vertices in $\mathbb{R}^3$. However, *linearly* combining shapes in $\mathcal{S}_v$ leads to artefacts for rotating parts [WDAH10].

To avoid this effect, we therefore have two options. We can choose to combine shapes in a *non-linear* way, for example by using geodesics with respect to an appropriate Riemannian metric [KMP07]. Alternatively, we can represent shapes using a space that encodes differential properties of the mesh instead [SP04, LSLCO05], so that simple *linear* combinations better preserve the shape of rotating parts. Usually, the dimension of such a *differential shape space* is greater than $3V$ and therefore not every point in $\mathcal{S}$ corresponds to a realization of $\mathcal{G}$ as a mesh in $\mathbb{R}^3$. In this case, existing approaches introduce an operator P which projects from $\mathcal{S}$ back into $\mathcal{S}_v$.

1.1. Differential shape spaces

Sumner and Popović [SP04] propose one such alternative shape space based on *deformation gradients*, which describe each face of the mesh as a linear transformation of a corresponding reference triangle. As the translational part of the transformation is omitted, the representation of a mesh in this shape space $\mathcal{S}_f$ is invariant under translations. The projection operator P_f only requires the solution of a linear system and can therefore be computed in real time. However, Kircher and Garland [KG08] observe that when combining deformation gradients, faces rotate along the shortest path in the embedding space, so interpolation between two or more poses can give unnatural, non-smooth deformations.

Several shape spaces [LSLCO05, KG08, BVGP09] address this shortcoming and give natural results even for very large rotations. Winkler et al. [WDAH10] and Fröhlich and Botsch [FB11] demonstrate excellent results based on a shape space $\mathcal{S}_e$ which stores the *edge lengths* and *dihedral angles* of a mesh, and therefore represents the mesh in a way that is invariant under both translations and rotations. The standard Euclidean metric in this space gives the discrete shell energy proposed by Grinspun et al. [GHDS03], and so linear interpolation in $\mathcal{S}_e$ results in interpolations that minimize this shell energy on $\mathcal{G}$. However, these favourable properties come at a cost, as the corresponding projection P_e requires the solution of an expensive non-linear problem, which Winkler et al. [WDAH10] tackle with a multi-scale approach and Fröhlich and Botsch [FB11] solve with a non-linear Gauss–Newton optimization.

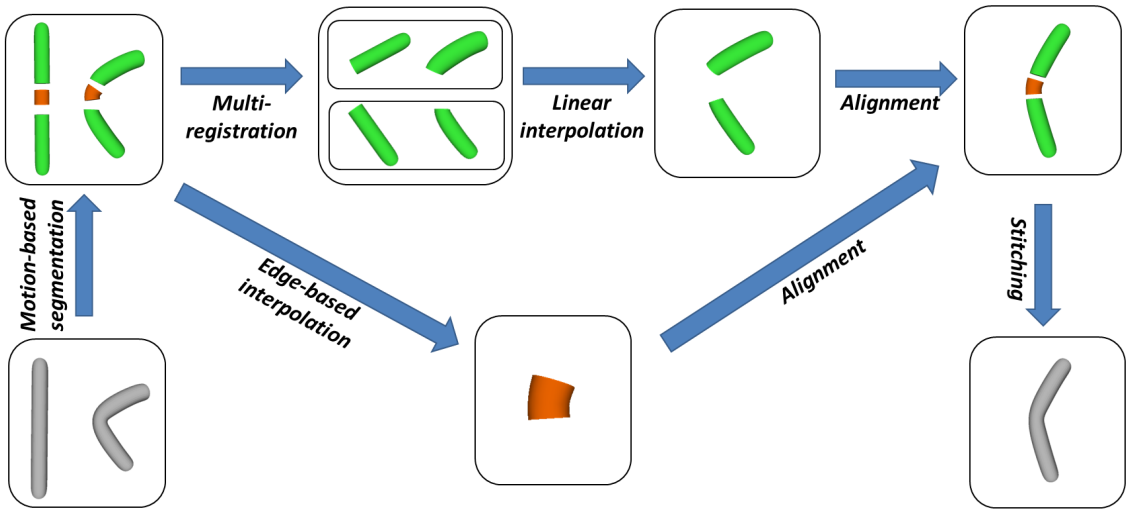

Figure 1: *Overview of our fast mesh interpolation method. Source and target mesh are first segmented into rigid parts (green) and non-rigid joints (red). We then use edge-based interpolation [WDAH10] to compute the interpolated shape of corresponding joints and linear vertex interpolation for corresponding rigid parts, after having registered the rigid parts into a common coordinate system. Finally, we align all the interpolated components and stitch them together to obtain the interpolated mesh.*

1.2. Our contribution

In this paper, we observe that for the common class of *articulated shapes*, much of the work that goes into computing P_e is unnecessary, as these shapes are composed of a number of *rigid parts* connected by *joints*, and it is only the joints that benefit from expensive non-linear handling. The majority of the shape is described by its rigid parts, and for these it is sufficient to use fast linear vertex interpolation. This observation leads to a *new shape space* $\mathcal{S}_m$, which stores edge lengths and dihedral angles only where necessary and uses vertex coordinates to represent the rest of the shape. We use the subscript m here to emphasize that $\mathcal{S}_m$ is a *mixed* rigid and non-rigid shape space.

Mixing these two representations gives a more compact shape space, as vertex coordinates require less storage than their edge-based counterparts, and a faster projection operator P_m, by avoiding expensive non-linear optimization where it is not needed. To demonstrate the advantages of $\mathcal{S}_m$, we use the application of *interpolating* or *morphing* between articulated shapes, as summarized in Figure 1 and reported in Section 3.

Our mixed shape space makes it possible to compute an interpolated mesh as much as 11 times faster than edge-based interpolation on whole shapes as described by Winkler et al. [WDAH10]. We find that interpolations in $\mathcal{S}_m$ may also better respect linearly interpolated edge lengths and dihedral angles than in $\mathcal{S}_e$ (see Figure 7). As an additional benefit, we demonstrate that decomposing the problem into rigid parts and joints leads to a more robust reconstruction in the presence of local self-intersections (see Section 3.3), which can cause distortions when using earlier approaches.

1.3. Segmentation

To decompose an articulated shape into rigid parts and joints, we rely on existing mesh segmentation techniques. Many segmentation methods fit into a common framework, which works by assigning a descriptive property value to each element of the mesh. This value could be the dihedral angle, geodesic or diffusion distance, for example; see the survey by Shamir [Sha08] for a detailed list of possibilities. The segmentation then partitions the mesh into meaningful parts using either k-means, hierarchical, or fuzzy clustering on the property values.

For interpolating between two or more poses, it is important that we use a segmentation technique that is based on multiple mesh poses. In this situation we want the segmentation to reflect the behaviour of the mesh in all available poses and to return connected subsets of $\mathcal{G}$ that *move* in a rigid fashion, and joint regions that *deform* non-rigidly. Instead of using a static metric, each element of the mesh therefore needs to be characterized by some measure which expresses the variation in that element. Common choices are rotation matrices [JT05] or dihedral angles [WB10, MBH*12], and the final clustering then reveals the articulated structure of the object. In this work, we adopt the approach by Marras et al. [MBH*12], but in principle, any other existing technique could be used instead.

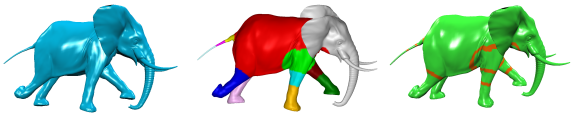

Figure 2: *In order to build the mixed shape space $\mathcal{S}_m$, we first decompose our input meshes (left) with any multiple segmentation method into meaningful articulated segments (middle). We then create the joint regions by growing a triangle strip of width three around each boundary between the parts. The rigid parts are constructed by removing two triangle strips from their boundaries (right). Joints and rigid parts therefore overlap by one triangle strip.*

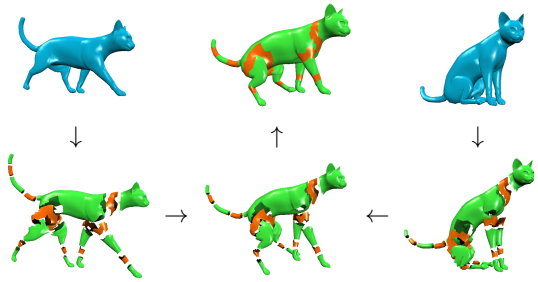

Figure 3: *Different steps of the algorithm: starting from the original shapes, we perform a segmentation. The corresponding parts are then interpolated and aligned, and finally stitched to obtain the final shape.*

The rigid parts of our segmentation often resemble the skeletal bones of an animation technique [JT05, CBC*05, SZT*08, KP11]. Indeed, a rigged skeleton would be one way to derive the initial segmentation we use in this work. However, there are some key differences between our interpolation and skeletal-driven deformation. One is that we start from a disjoint classification into separate rigid parts, rather than finding skinning weights that loosely associate vertices with a collection of skeletal bones. Another is that the range of motion possible through our interpolation is much wider, as the joints can exhibit arbitrary non-rigid motion rather than being constrained to model a single skeletal joint. This makes our interpolation better suited to high-quality mesh interpolations which are computed off-line, rather than the real-time animation targeted by linear blend skinning.

2. A shape space $\mathcal{S}_m$ for articulated shapes

We build our shape space $\mathcal{S}_m$ from a mesh segmentation that identifies the rigid parts of an articulated shape (Section 2.1). The space is defined by combining a rotation-invariant representation with values that locate a shape in space. When interpolating between shapes, we must therefore account for the structure of $\mathcal{S}_m$ that results from this rotation invariance (Section 2.2). As for many other shape spaces, it is trivial to embed a mesh in $\mathcal{S}_m$ by extracting the required edge and vertex properties. However, there is a trade off, in that complexity is moved to the operator P_m that maps a point in $\mathcal{S}_m$ back into a mesh with connectivity $\mathcal{G}$ in $\mathbb{R}^3$. We discuss our implementation of P_m in Section 2.3.

2.1. Segmentation and joint creation

As mentioned in Section 1.3, we use the method described by Marras et al. [MBH*12] to identify parts of the mesh which move in a rigid or near-rigid fashion. A mesh segmentation assigns a label to each face of the mesh so that the entire mesh is partitioned into disjoint regions. However, while artificial or man-made objects might be completely described by their rigid parts, many articulated shapes occurring in nature have rigid parts which are connected together by joints exhibiting some non-rigid deformation.

Our shape space uses an explicit segmentation for these joints, which we obtain by growing strips of triangles around the boundaries between segmented mesh regions (see Figure 2). We use a default joint width of three triangle strips for our implementation, although we also find that the result does not depend significantly on the joint width that we select (see Section 3.1). We remove triangles from the rigid parts so that they overlap with connecting joints by just one triangle strip. This overlap is important for the *stitching* described in Section 2.3 that forms a whole shape from its component parts.

After creating joints and shrinking rigid parts, the end result is p rigid parts, with V_i vertices in the i-th rigid part, and q joints, with E_k edges in the k-th joint. Our shape space now stores the shape of all of these components separately, by defining the shape spaces $\mathcal{R}_1,\dots,\mathcal{R}_p$ to store the shape of each rigid part, and $\mathcal{J}_1,\dots,\mathcal{J}_q$ for the rotation-invariant shape of each joint. The rigid part shape spaces are $\mathcal{R}_i = \mathbb{R}^{3V_i}$ as rigid parts are simply described using the coordinates of their vertices, and the joint spaces are $\mathcal{J}_k = \mathbb{R}^{2E_k}$ as joints are described by the length and dihedral angle of each of their edges.

2.2. Shape space structure

We are now able to define the complete shape space $\mathcal{S}_m$ as the Cartesian product of all its component shape spaces, with the addition of six real values to resolve translational and rotational invariance. That is,

$$\mathcal{S}_m = \mathcal{J}_1 \times \cdots \times \mathcal{J}_q \times \mathcal{R}_1 \times \cdots \times \mathcal{R}_p \times \mathbb{R}^6,$$

and the dimension of $\mathcal{S}_m$ is therefore

$$2\sum_{k=1}^{q} E_k + 3\sum_{i=1}^{p} V_i + 6.$$

We consider each component part to be specified up to a

rigid-body transform, as the rotations and translations that match the parts together are determined automatically by the projection P_m (see Section 2.3). Joints are already stored in a rotation-invariant way, but the same is not true of the rigid parts, and so we define a custom operator A to compute *affine combinations* of points in $\mathcal{S}_m$. This operator takes account of the fact that each rigid part shape space $\mathcal{R}_i$ is a redundant representation for an infinite number of equivalence classes, where shapes belong to the same equivalence class if they are related to each other by a rigid-body motion. With this view, a shape $R \in \mathcal{R}_i$ is only a representative of its equivalence class, and to combine shapes in two different classes, we must first align the representatives to each other.

Formally, A is a map

$$A : (\mathcal{S}_m \times \mathbb{R})^n \to \mathcal{S}_m$$

where each of the n shapes in $\mathcal{S}_m$ is associated with a weight in $\mathbb{R}$. For any combination

$$S = A\big((S_1, w_1), \ldots, (S_n, w_n)\big)$$

we also enforce the constraint on the weights that $\sum_l w_l = 1$, so that A computes an *affine* combination. We can now define the action of A in detail by considering its effect on each component of $\mathcal{S}_m$.

Edge-based interpolation for joints By transforming joints into an edge-based rotation-invariant representation, A is very easy to compute on each joint shape space $\mathcal{J}_k$. If $J_{k,l}$ is the shape of the k-th joint in S_l, then the new shape of that joint in S is simply $\sum_i w_l J_{k,l}$. In other words, we interpolate edge lengths and dihedral angles linearly [WDAH10], which gives the good properties described in Section 1.1 for non-rigid deformations.

Multi-registration for rigid parts We have to do more work to compute A in the rigid-part spaces $\mathcal{R}_k$, as the shapes $R_{k,l}$, $l = 1, \ldots, n$ are only representatives, in a particular orientation, of the rotation-invariant shapes they describe. While these shapes are modelled as rigid and may correspond to true rigid entities such as bones, in fact we expect to interpolate between parts that also show slight non-rigid variations.

To combine rigid parts, we must therefore align the shapes before using linear vertex interpolation, which we find to be sufficient to resolve any small non-rigid deformation. We approach the alignment in the same way as Winkler et al. [WDAH10], by using the simultaneous registration described by Williams and Bennamoun [WB00]. Their method finds the rigid transformation for each part $R_{k,l}$ ($l = 1, \ldots, n$) that minimizes the squared differences between all pairs of corresponding vertices. If we write $\tilde{R}_{k,l}$ for the result of transforming $R_{k,l}$ using the computed translations and rotations, then the final step to compute A in $\mathcal{R}_k$ is to linearly interpolate the vertices of the transformed parts. That is, the new shape of the k-th rigid part in s is $\sum_l w_l \tilde{R}_{k,l}$.

Linear interpolation for pose parameters We now have A defined on $\mathcal{J}_1, \ldots, \mathcal{J}_q$ and $\mathcal{R}_1, \ldots, \mathcal{R}_p$, so it only remains to describe its action on the pose parameters θ_l in $\mathbb{R}^6$ which define the final position and orientation of each shape S_l. For these degrees of freedom, we can use the barycentre to describe position in space and the three independent entries that result from the logarithm of a rotation matrix to describe orientation. This rotation could give the orientation of the first face with respect to a reference triangle, for example. Both of these descriptions can be combined linearly to good effect, so again A can take the simple weighted linear combination $\sum_l w_l \theta_l$ as the pose parameters for s.

2.3. The projection operator P_m

After representing a collection of shapes in $\mathcal{S}_m$ and taking affine combinations using the operator A, the last step in an interpolation is to project back into the space of meshes with fixed connectivity $\mathcal{G}$ in $\mathbb{R}^3$. This projection consists of three steps: alignment, stitching, and global positioning.

Alignment A shape in $\mathcal{S}_m$ is a collection of q rotation-invariant joints, and p rigid parts, which we also consider modulo rigid-body transforms. So the first step to convert these parts back into a coherent shape is to align the joints and rigid parts to each other. To do so, we can use the same registration technique by Williams and Bennamoun [WB00] we used in Section 2.2. The problem is the same, since we have a collection of mesh parts with corresponding vertices provided by the overlapping strips of triangles we describe in Section 2.1. This multi-registration method therefore gives us rotations and translations for each part, found to minimize the squared differences between all corresponding vertices.

Stitching Once the component parts have been placed in alignment, there is likely to still be some residual error between corresponding vertices. We therefore stitch the parts together by using the *edge blending* described by Winkler et al. [WDAH10]. In summary, their method solves a linear system that minimizes the error between each edge and a target vector, for each part in which that edge appears. Each such target vector retains the *orientation* of its corresponding edge, but sets the target *length* to be the linearly-interpolated edge length. Note that this target length always appears in a joint shape space $\mathcal{J}_k$ for some k, as we connect one or more joints to a rigid part, and never connect rigid parts to each other.

Global positioning The result of the stitching step is a coherent mesh with connectivity $\mathcal{G}$, but with arbitrary position and orientation in $\mathbb{R}^3$. The final step in P_m is therefore to use the six pose parameters to locate the projected shape in space. This involves a rotation, so that the first face is oriented correctly with respect to its reference triangle, followed by a translation to correctly locate the mesh barycentre.

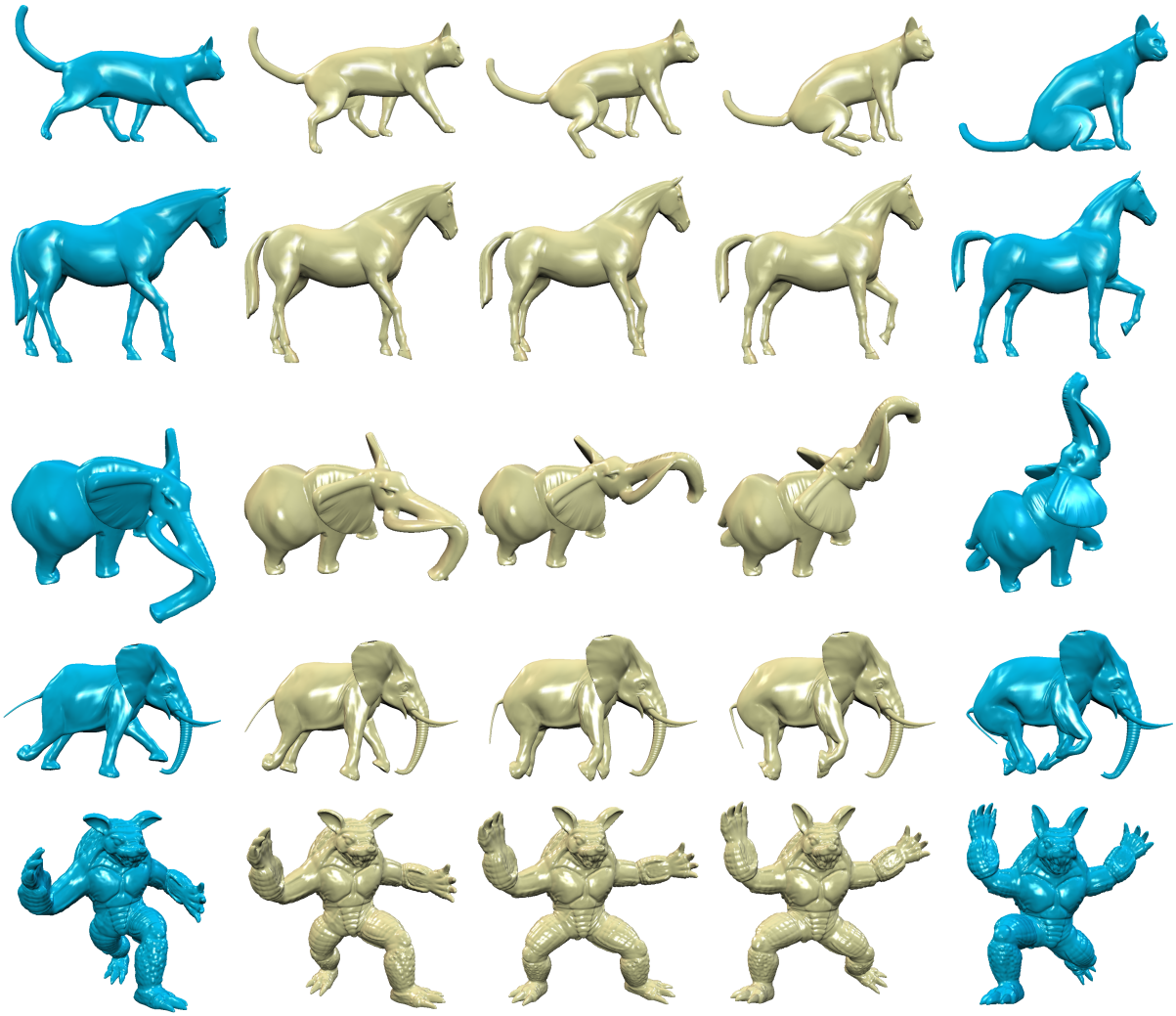

Figure 4: *Results of our mixed interpolation technique for several sets of two input poses and interpolation weight* $w_1 \in \{0.25, 0.5, 0.75\}$. *From top to bottom we show the datasets 'cat', 'horse', 'standing elephant', 'galloping elephant' and 'armadillo'.*

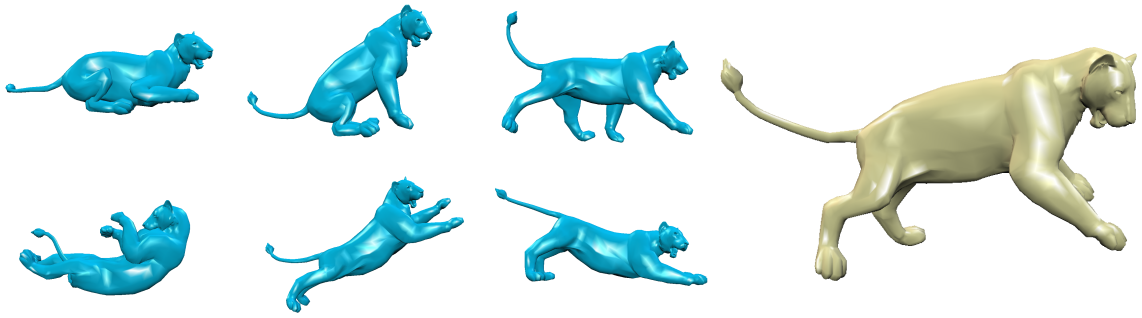

Figure 5: *Result of our mixed interpolation between six different input poses with interpolation weights* $w_1 = \cdots = w_6 = 1/6$.

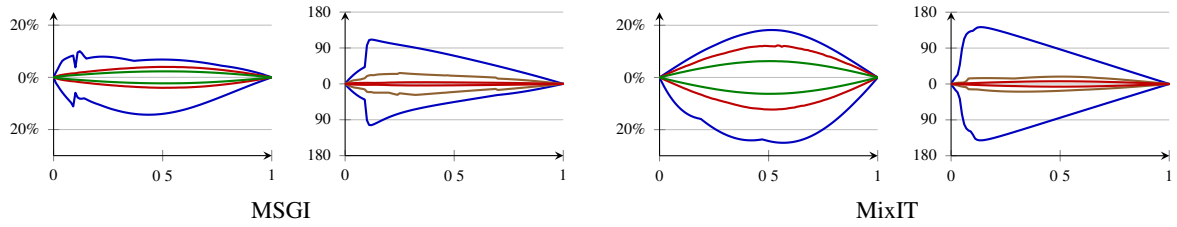

Figure 6: *Comparison of errors for the standing elephant data set (third row in Figure 4). The plots show the relative error in edge length and degree error in dihedral angle, compared to the linearly-interpolated target values for each edge and plotted over the interpolation weight $w_1 \in [0,1]$. Every plot for length error shows the envelopes for all edges —, as well as the worst 99.9% — and 99% — of edges. In every plot for dihedral angle error, the envelopes show all edges —, as well as the worst 99.99% — and 99.9% — of edges. For this example, MSGI gives lower error overall.*

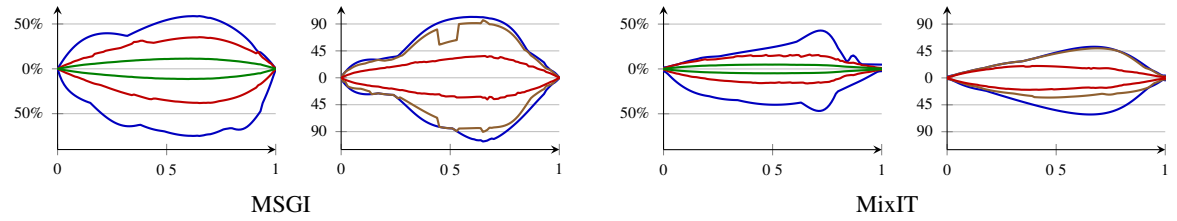

Figure 7: *Comparison of errors for the horse dataset (second row in Figure 4); cf. Figure 6. For this example, our method MixIT gives lower error overall.*

	F	V	MSGI				MixIT							$\frac{\dim \mathcal{S}_e}{\dim \mathcal{S}_m}$
			$\dim \mathcal{S}_e$	*Init.*	*Interp.*	*Total*	$\dim \mathcal{S}_m$	*Align.*	*Init.*	*Interp.*	*Stitch.*	*Total*	*Speed-up*	
lion	9996	5000	44994	112	1305	1417	22035	0	12	189	237	438	3.2	2.0
cat	14410	7207	64857	196	1727	1923	27251	0	9	167	265	441	4.3	2.4
horse	16843	8431	75847	257	2051	2308	34641	1	14	277	329	621	3.7	2.2
standing elephant	79946	39969	359751	5025	11728	16853	127441	14	9	217	2005	2245	7.5	2.8
galloping elephant	84638	42321	380883	5135	12395	17530	141217	15	21	447	1716	2199	8.0	2.7
armadillo	331904	165954	1493580	77396	58581	135977	541528	62	73	1681	9776	11592	11.7	2.8

Table 1: *Comparison of the interpolation between two poses using MSGI and MixIT. All times are given in milliseconds.*

3. Results

We demonstrate the advantages of our mixed shape space $\mathcal{S}_m$ by interpolating between two or more poses of the same object. Figures 4 and 5 and the accompanying video show several examples of our results. We implemented this *mixed interpolation technique* (MixIT) in C++ and compare it to the *Multi-Scale Geometry Interpolation* (MSGI) method described by Winkler et al. [WDAH10].

The input meshes were taken from the Aim@Shape repository [aim] and the Sumner shape dataset [SP04], and we used the multiple mesh segmentation described by Marras et al. [MBH*12] to create consistent segmentations of corresponding poses. All tests were carried out on an Intel Quad-Core Q9550 2.83GHz with 4GB onboard memory. The segmentation algorithm uses GPU parallelization as described in the original work, while the interpolation step takes advantage of a multi-threaded implementation. Our results show that MixIT is not only faster than the MSGI algorithm, but also that it scales better, achieving the highest speed-up for the largest meshes.

To perform a quantitative comparison between MSGI and MixIT, we measure the interpolation error in terms of edge lengths and dihedral angle in the same way as Winkler et al. [WDAH10]. Our experiments show that the error behaves very similarly for both methods: MSGI is slightly better for some examples (see Figure 6), while MixIT produces lower errors in other cases (see Figure 7). However, the main advantage of our method is that it is significantly faster than MSGI, in particular for large meshes (see Table 1). Moreover, the dimension of $\mathcal{S}_m$ is typically smaller than that of $\mathcal{S}_e$ by a factor between 2 and 3, which greatly reduces the amount of memory needed to store shape data.

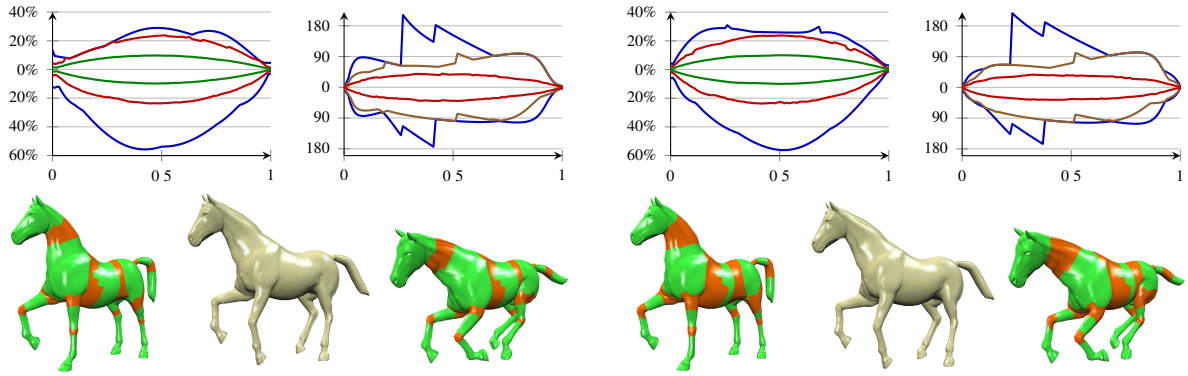

Figure 8: *Comparison of errors for interpolation between two poses of the horse using joint width three (left) and five (right). The result is relatively insensitive to this parameter, but we find a slight decrease in performance for larger joints.*

3.1. Insensitivity to joint width

Our tests found that a joint width of three triangle strips at each boundary between segmented parts usually gives good results. Increasing this size does not have a significant influence on the interpolation in terms of error (see Figure 8) or computational speed, although performance does drop slightly for larger joints because edge-based interpolation is slower than vertex interpolation. It is not advisable, however, to use smaller joints, as the stitching (see Section 2.3) can then lead to unwanted artefacts.

3.2. Sensitivity to segmentation quality

In contrast, both the quality of the results and the run-time performance depend crucially on the quality of the initial shape segmentation. On the one hand, a poor segmentation (for example, if some joints are missing due to under-segmentation) will result in large errors and unwanted interpolation artefacts (see Figure 9). On the other hand, while the interpolation is resilient with respect to over-segmentation, a high number of joints will mean that an interpolation takes longer to compute.

3.3. Robustness with respect to noise

An interesting result from our tests is that MixIT seems to be more robust with respect to noise than MSGI. In particular, if the input meshes contain local self-intersections, then MSGI can give implausible results, and so these self-intersections must be removed in a preprocessing step. Our algorithm can overcome this limitation if the self-intersections occur in the rigid parts, because the linear vertex interpolation is not affected by such artefacts (see Figure 10).

4. Conclusion

The main limitation of our fast interpolation method is its inherent restriction to articulated shapes, because the advantages of the mixed shape space $\mathcal{S}_m$ rely on a meaningful segmentation into rigid parts and non-rigid joints.

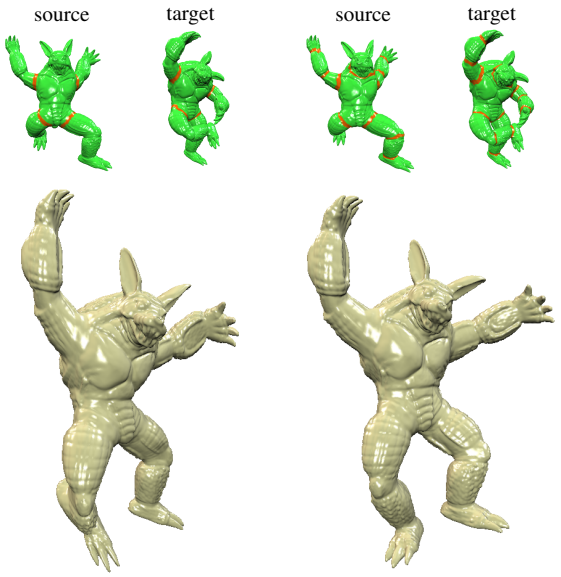

Figure 9: *Interpolation between the same poses of the armadillo using two different segmentations. A segmentation that does not capture the articulation of the legs (left) produces artefacts around the knee and the foot, which we can avoid by using a good segmentation (right).*

It is important to note that our affine combination operator A (see Section 2.2) is not a linear operator because of the rigid alignment step that we introduce for rigid parts. This stands in contrast to simple linear interpolation in $\mathcal{S}_v$, $\mathcal{S}_f$ [SP04], or $\mathcal{S}_e$ [WDAH10]. Instead, points in $\mathcal{S}_m$ are combined in a non-linear way, as for the shape spaces described by Kilian et al. [KMP07] and Kircher and Garland [KG08]. This complication appears to make it impossible to use $\mathcal{S}_m$ as

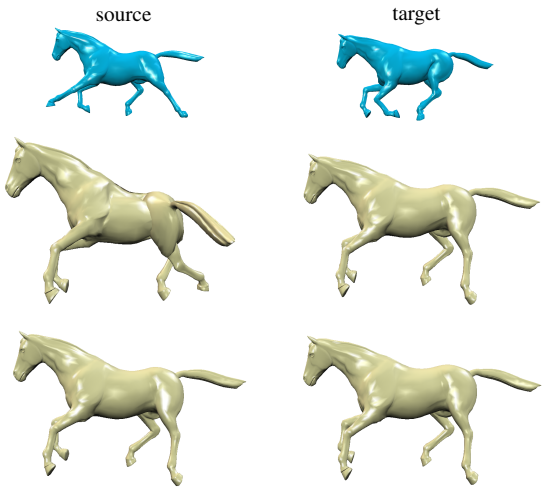

Figure 10: *Results of the interpolation between two poses of the horse (top row) using MSGI and MixIT in the presence of local self-intersections (middle row) and after removing them in a preprocessing step (bottom row). The consistency of MixIT in the second and third rows shows that our method is more robust with respect to such artefacts.*

the underlying shape space of the deforming mesh representation recently introduced by Cashman and Hormann [CH12]. However, if we are willing to consider the shape space construction as an offline process, where the total set of shapes in the space is already known (which is usually the case), then we can carry out the rigid-part alignment in a preprocessing step and subsequently, combine shapes in a purely linear way. We then recover all the tools from linear algebra that are required by Cashman and Hormann's representation for deforming mesh sequences [CH12].

Acknowledgements

This work was supported by the SNF under project number 200021-134639. The authors thank the anonymous reviewers for their valuable comments and suggestion.

References

[aim] The aim@shape shape repository. http://shapes.aimatshape.net/. 6

[BVGP09] BARAN I., VLASIC D., GRINSPUN E., POPOVIĆ J.: Semantic deformation transfer. *ACM Trans. Graph. 28*, 3 (2009), #36:1–6. 1

[CBC*05] CAPELL S., BURKHART M., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Physically based rigging for deformable characters. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005), pp. 301–310. 3

[CH12] CASHMAN T. J., HORMANN K.: A continuous, editable representation for deforming mesh sequences with separate signals for time, pose and shape. *Comput. Graph. Forum 31*, 2 (2012), 735–744. 1, 8

[FB11] FRÖHLICH S., BOTSCH M.: Example-driven deformations based on discrete shells. *Comput. Graph. Forum 30*, 8 (2011), 2246–2257. 1

[GHDS03] GRINSPUN E., HIRANI A. N., DESBRUN M., SCHRÖDER P.: Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 62–67. 1

[JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Trans. Graph. 24*, 3 (2005), 399–407. 2, 3

[KG08] KIRCHER S., GARLAND M.: Free-form motion processing. *ACM Trans. Graph. 27*, 2 (2008), #12:1–13. 1, 7

[KMP07] KILIAN M., MITRA N. J., POTTMANN H.: Geometric modeling in shape space. *ACM Trans. Graph. 26*, 3 (2007), #64:1–8. 1, 7

[KP11] KIM J., POLLARD N. S.: Fast simulation of skeleton-driven deformable body characters. *ACM Trans. Graph. 30*, 5 (2011), #121:1–19. 3

[LSLCO05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3 (2005), 479–487. 1

[MBH*12] MARRAS S., BRONSTEIN M. M., HORMANN K., SCATENI R., SCOPOGNO R.: Motion-based mesh segmentation using augmented silhouettes. *Graph. Models 74*, 4 (2012), 164–172. 2, 3, 6

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Comput. Graph. Forum 27*, 6 (2008), 1539–1556. 2

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (2004), 399–405. 1, 6, 7

[SZT*08] SHI X., ZHOU K., TONG Y., DESBRUN M., BAO H., GUO B.: Example-based dynamic skinning in real time. *ACM Trans. Graph. 27*, 3 (2008), #29:1–8. 3

[WB00] WILLIAMS J. A., BENNAMOUN M.: Simultaneous registration of multiple point sets using orthonormal matrices. In *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing* (2000), vol. 4, pp. 2199–2202. 4

[WB10] WUHRER S., BRUNTON A.: Segmenting animated objects into near-rigid components. *Visual Comput. 26*, 2 (2010), 147–155. 2

[WDAH10] WINKLER T., DRIESEBERG J., ALEXA M., HORMANN K.: Multi-scale geometry interpolation. *Comput. Graph. Forum 29*, 2 (2010), 309–318. 1, 2, 4, 6, 7

[YYPM11] YANG Y.-L., YANG Y.-J., POTTMANN H., MITRA N. J.: Shape space exploration of constrained meshes. *ACM Trans. Graph. 30*, 6 (2011), #124:1–12. 1