Volume xx (200y), Number z, pp. 1–13

Efficient interpolation of articulated shapes using mixed shape spaces

Stefano Marras Thomas J. Cashman Kai Hormann

Università della Svizzera italiana, 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 shape space 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 (see Figure 1).

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] or by computing the deformation path with least energy dissipation [HRWW12]. 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 $\mathcal{S}_v$.

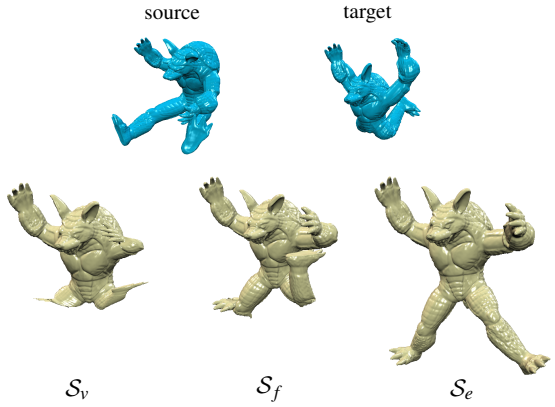


Figure 1: *An example of linear interpolation between two poses (top) using different shape spaces. Linear vertex interpolation in $\mathcal{S}_v$ leads to shrinking effects on the legs because they are pointing in opposite directions in the source and target poses (left). Linear interpolation of deformation gradients [SP04] in $\mathcal{S}_f$ prefers shortest rotation paths and rotates the left leg in the wrong direction (middle). Instead, linearly interpolating edge coordinates [WDAH10] in $\mathcal{S}_e$ gives physically plausible results (right). We propose an algorithm that produces results of a similar quality to Winkler et al. [WDAH10] but is much faster. It relies on a segmentation of the shape into rigid parts and non-rigid joints.*

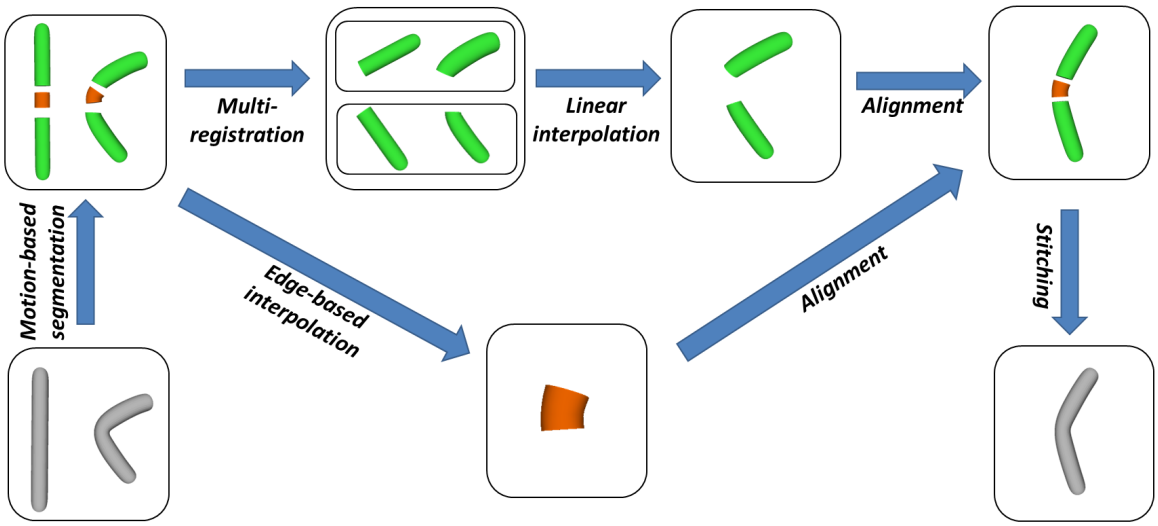


Figure 2: *Overview of our fast mesh interpolation method. Source and target mesh are first segmented into rigid parts (green) and non-rigid joints (red). We then use edge-based interpolation [WDAH10] to compute the interpolated shape of corresponding joints and linear vertex interpolation for corresponding 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.*

1.1. 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 (see Figure 1).

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 (see Figure 1). 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.

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 2 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 11). As an additional benefit, we demonstrate that decomposing the problem into rigid

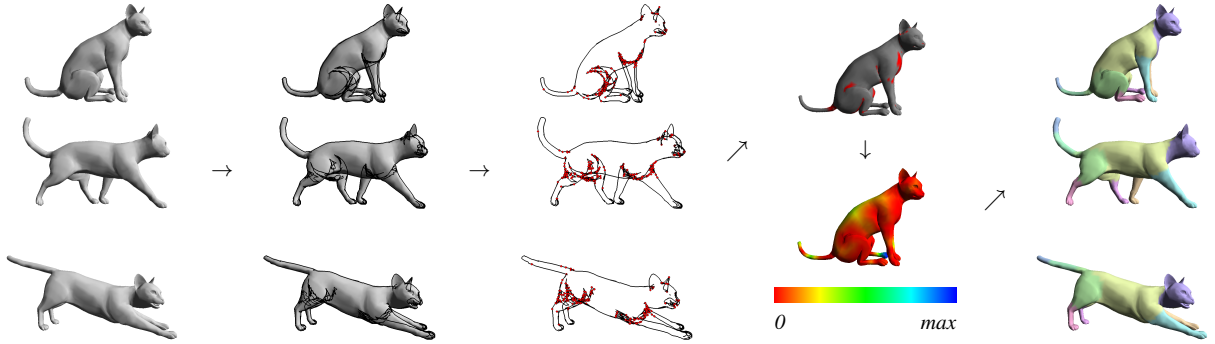


Figure 3: *Flowchart of the segmentation process described by Marras et al. [MBH*12]. Given a set of input poses of an articulated shape (a), the perceptually relevant edges (b) are extracted before studying the deformation in edge length and dihedral angle that occurs at each vertex of the shape (c). The local maxima of the minimal distance function with respect to these deforming vertices are used as seeds for a region-growing algorithm (d), which results in a segmentation of the shape into disjoint regions (e).*

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. For example, this value could be the dihedral angle, geodesic, or diffusion distance; 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.

James and Twigg [JT05] characterize each face of the source mesh by a set of rotation and scaling matrices. The faces described by similar deformations are then clustered to reveal the articulated structure of the object. Wuhrer and Brunton [WB10] investigate the behaviour of edges and dihedral angles instead, characterizing each edge of the deforming shape by its maximum dihedral angle variation. A spanning-tree of the edge graph is computed, and this tree is finally cut to detect the boundaries between rigid parts. Another edge-based method is due to Marras et al. [MBH*12], and as we decided to mainly rely on this approach for the examples presented in this paper, we discuss it in more detail in Section 2.1. In principle, however, *any* segmentation technique can be used for splitting the given poses into rigid parts and non-rigid joints, as demonstrated in Section 3.2.

The rigid parts of our segmentation often resemble the bones of a deformation skeleton, as used in animation [JT05, CBC*05, SZT*08, KP11]. Indeed, a rigged skeleton would be yet another 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 (see Figure 3). In order to find the rigid parts of the shape, this method starts by observing the N input poses from K different viewpoints. For each viewpoint, a set of *perceptually-relevant* edges are extracted: those edges which connect a visible and a hidden face without taking occlusions into account. The perceptually-relevant edges of each silhouette are then collected in a graph-based structure named the *augmented silhouette*, because it contains more edges than the standard silhouette. To reduce the size of the problem, only K' out of all the possible $K \cdot N$ silhouettes are randomly selected. For each silhouette, a further analysis identifies the vertices where the surrounding triangles deform the most, by measuring the maximum variation in both dihedral angle and length of the edges incident on each vertex. Using these values, the vertices which are characterized by a large deformation for each silhouette are marked as *significant*. Finally, counting the number of times that a vertex has been marked as significant, it is possible to identify the vertices which are affected by significant deformation.

We now consider the *diffusion distance* [BB11, CLL*05] between the significant and all other vertices, and assign to each vertex the smallest amongst all distances to the significant vertices. The vertices where this minimal distance is locally maximal are then used as *seeds* for a classical region-growing algorithm. This region-growing partitions the shape into disjoint, connected regions, with each vertex of the shape characterized by a single label. We employ the same implementation of this whole process as Marras et al. [MBH*12], taking advantage of GPU-based parallelism to extract and analyse the silhouettes.

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. In this case, the joints correspond to regions of the mesh with a high concentration of 'significant' vertices.

Instead of explicitly detecting joints in the segmentation step, our shape space infers the position of each joint by growing strips of triangles around the boundaries between segmented mesh regions (see Figure 4). 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

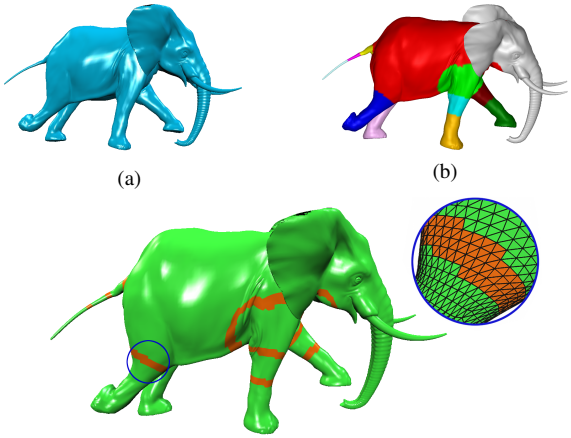


Figure 4: *In order to build the mixed shape space $\mathcal{S}_m$, we first decompose our input meshes (a) with any multiple segmentation method into meaningful articulated segments (b). We then create the joint regions by growing a triangle strip of width three around each boundary between the parts. The rigid parts are constructed by removing two triangle strips from their boundaries (c). Joints and rigid parts therefore overlap by one triangle strip.*

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 $\mathcal{R}_1, \ldots, \mathcal{R}_p$ to store the shape of each rigid part, and $\mathcal{J}_1, \ldots, \mathcal{J}_q$ 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 formally 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

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

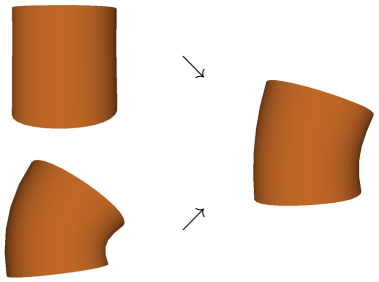


Figure 5: *For corresponding joints (left), the operator A computes a direct linear interpolation in edge space (right).*

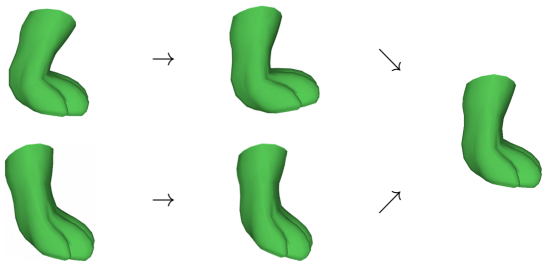


Figure 6: *For corresponding rigid parts (left), the operator A first aligns (middle) and then linearly interpolates in vertex space (right).*

The dimension of $\mathcal{S}_m$ is significantly smaller than the dimension of the full edge-based shape space $\mathcal{S}_e$, as shown in Table 2. This is because the set of joint edges is just a small subset of the whole edge set and, as a consequence, the dimension of $\mathcal{S}_m$ is influenced mainly by the number of mesh vertices.

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

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

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 $\sum_i w_l J_{k,l}$ (see Figure 5). 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 $\sum_l w_l \tilde{R}_{k,l}$ (see Figure 6).

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.

	F	V	MSGI			DSO			MixIT					
			Init.	*Interp.*	*Total*	*Iter.*	*Time/iter.*	*Total*	*Align.*	*Init.*	*Interp.*	*Stitch.*	*Total*	*Speed-up*
lion	9996	5000	112	1305	1417	25	529	13217	0	12	189	237	438	3.2
cat	14410	7207	196	1727	1923	27	919	24816	0	9	167	265	441	4.3
horse	16843	8431	257	2051	2308	22	1248	27454	1	14	277	329	621	3.7
standing elephant	79946	39969	5025	11728	16853	13	21268	276489	14	9	217	2005	2245	7.5
galloping elephant	84638	42321	5135	12395	17530	6	35933	215597	15	21	447	1716	2199	8.0
armadillo	331904	165954	77396	58581	135977	11	289830	3188130	62	73	1681	9776	11592	11.7

Table 1: *Comparison of the interpolation between two poses using MSGI, DSO and MixIT. All times are given in milliseconds. For DSO, the number of iterations and the average time per iteration are given.*

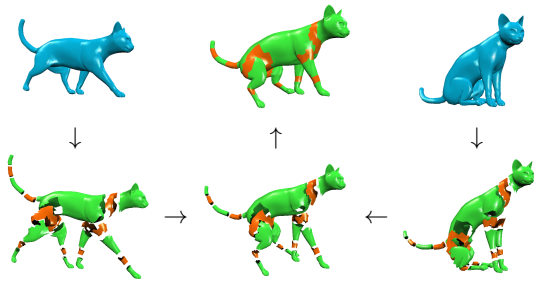


Figure 7: *Different steps of the algorithm: starting from the original shapes, we perform a segmentation. The corresponding parts are then interpolated and aligned, and finally stitched to obtain the final shape.*

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 position them in a common coordinate system. 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. The rotation invariance of this procedure is later resolved by keeping one part fixed and then finding the optimal global rotation for the remaining parts.

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]. Consider two vertices v_i and v_j, which are connected by an edge in the original shape, and suppose that this edge is also part of the overlapping region between some rigid part $\mathcal{R}_h$ and some joint $\mathcal{J}_k$. Denoting the positions of these two vertices in $\mathcal{R}_h$ by v_i' and v_j' and those in $\mathcal{J}_k$ by v_i'' and v_j'', we have $v_i' - v_j' \approx v_i'' - v_j''$ after the alignment step, with some small alignment error, and likewise for every pair of vertices connected by an edge in the overlapping region. To compensate for this slight remaining mismatch, we first compute the linearly-interpolated edge length l_{ij} and then determine the coordinates of the vertices v_i and v_j in the stitched mesh by matching all conditions

$$v_i - v_j = \frac{v_i' - v_j'}{\|v_i' - v_j'\|} l_{ij} \quad \text{and} \quad v_i - v_j = \frac{v_i'' - v_j''}{\|v_i'' - v_j''\|} l_{ij} \quad (1)$$

in a least squares sense. This amounts to solving the linear system $M^T M v = e$, where M is the matrix which represents the edge connectivity structure, v is the vector of all vertices in the overlapping region, and e is the vector containing all right-hand sides in (1). Note that M is sparse with exactly one entry 1 and one entry -1 per row.

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.

Although we have described the action of A and P_m in general terms, an interpolation pipeline using $\mathcal{S}_m$ is the composition of many simple operations. Figure 7 summarizes this complete process for the common case of interpolation between just two input meshes (i.e., $n = 2$).

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 8 and 9 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] as well as to our implementation of the *discrete shell optimization* (DSO) proposed by Fröhlich and Botsch [FB11].

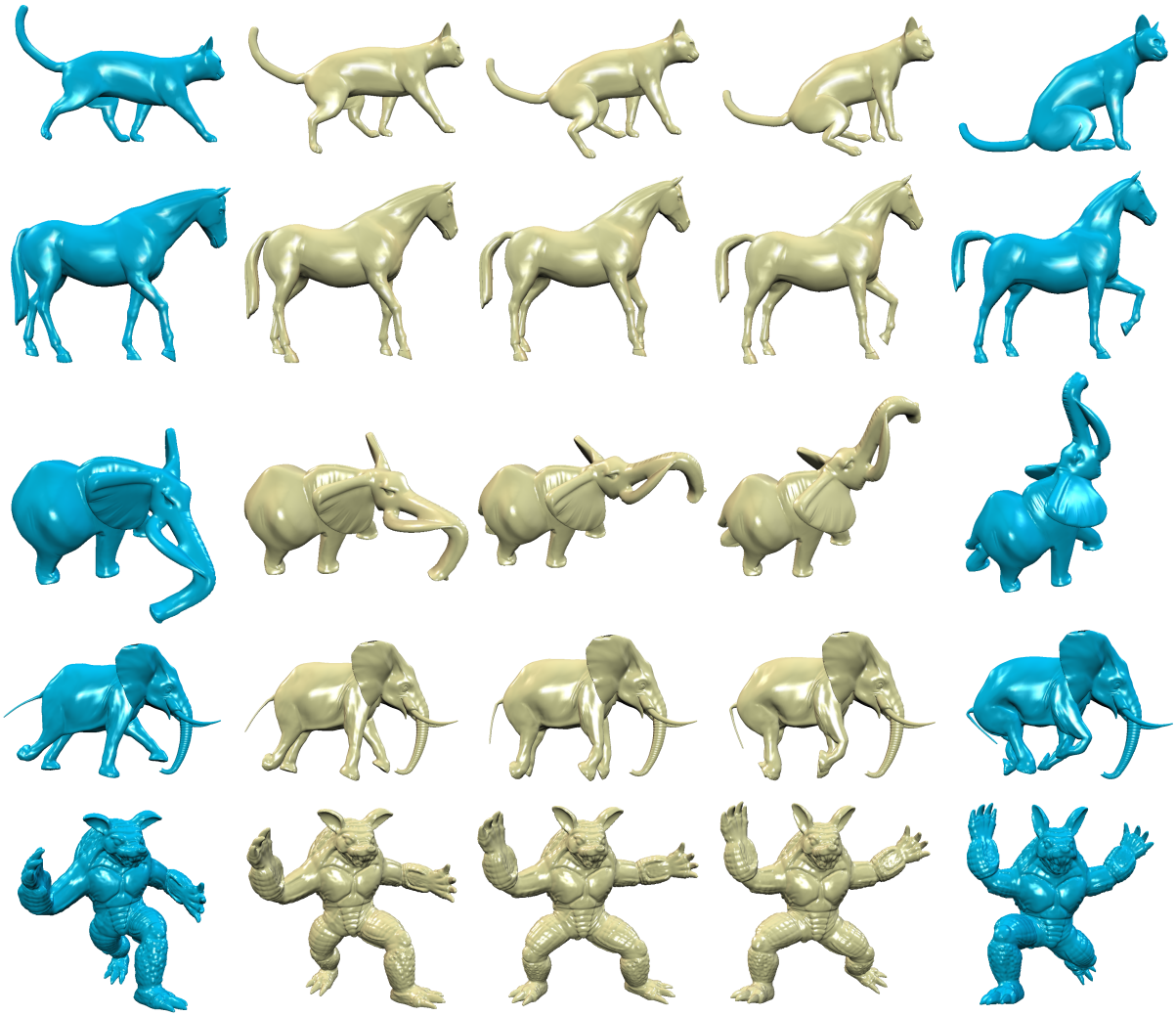
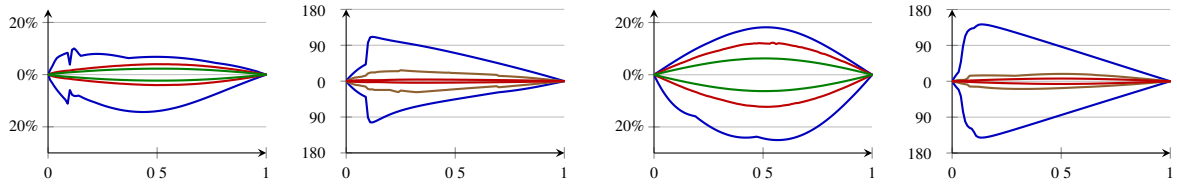


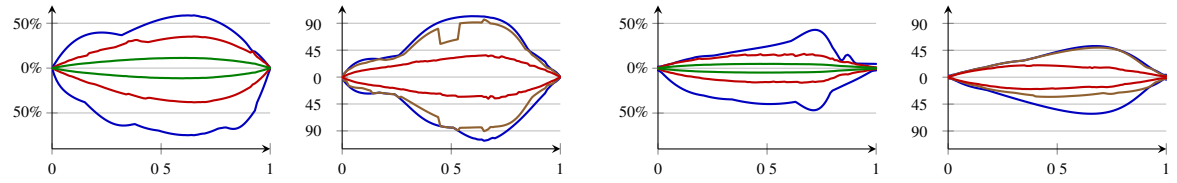
Figure 8: *Results of our mixed interpolation technique for several sets of two input poses and interpolation weight* $w_1 \in \{0.25, 0.5, 0.75\}$. *From top to bottom we show the datasets 'cat', 'horse', 'standing elephant', 'galloping elephant' and 'armadillo'.*

Figure 9: *Result of our mixed interpolation between six different input poses with interpolation weights* $w_1 = \cdots = w_6 = 1/6$.



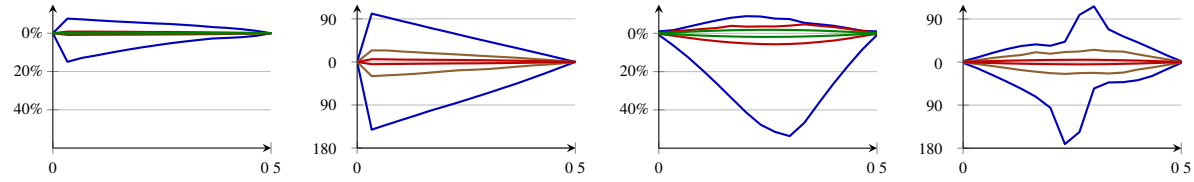
MSGI MixIT

Figure 10: *Comparison of errors for the standing elephant dataset (third row in Figure 8). 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.*



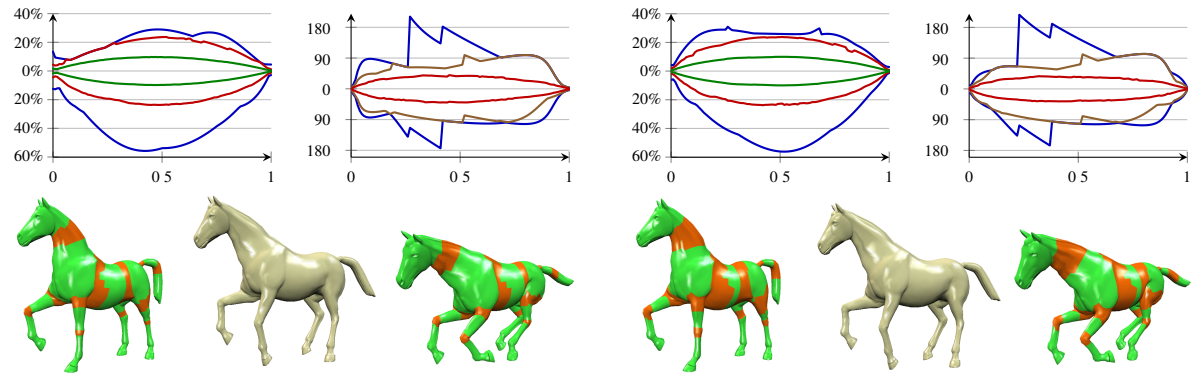
MSGI MixIT

Figure 11: *Comparison of errors for the horse dataset (second row in Figure 8); cf. Figure 10. For this example, our method MixIT gives lower error overall.*



DSO MixIT

Figure 12: *Comparison of errors for the galloping elephant dataset (fourth row in Figure 8); cf. Figures 10 and 11. For this example, DSO gives a lower error overall.*



Standard joint width Wider joints

Figure 13: *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.*

	$\dim \mathcal{S}_e$	$\dim \mathcal{S}_m$	$\frac{\dim \mathcal{S}_e}{\dim \mathcal{S}_m}$
lion	44994	22035	2.0
cat	64857	27251	2.4
horse	75847	34641	2.2
standing elephant	359751	127441	2.8
galloping elephant	380883	141217	2.7
armadillo	1493580	541528	2.8

Table 2: *Comparison between the dimensions of the edge shape space $\mathcal{S}_e$ and the mixed shape space $\mathcal{S}_m$ for the shapes listed in Table 1.*

	Width	*Align.*	*Init.*	*Interp.*	*Stitch.*	*Total*	$\dim \mathcal{S}_m$
cat	5	0	21	267	254	542	29140
	9	0	26	571	436	1033	34232
horse	5	1	87	557	308	953	37492
	9	1	91	657	410	1159	41280
galloping elephant	5	9	41	834	1909	2793	145713
	9	9	71	1606	2229	3915	153084
armadillo	5	45	497	3309	11603	15454	556309
	9	51	621	7013	15550	23235	575401

Table 3: *Comparison of times and dimensions for interpolation between two poses using different joint widths for some of the shapes listed in Table 1. Joint width is expressed as the number of triangle strips per joint. All times are expressed in milliseconds.*

The input meshes were taken from the Aim@Shape repository [aim] and the Sumner shape dataset [SP04], and we mainly used the multiple mesh segmentation described by Marras et al. [MBH*12] to create consistent segmentations of corresponding poses (see Section 2.1), although Section 3.2 also shows results for some different segmentation techniques. All tests were carried out on an Intel QuadCore Q9550 2.83GHz with 4GB onboard memory. The segmentation algorithm uses GPU parallelization as described in the original work by Marras et al. [MBH*12], while the interpolation step takes advantage of a multi-threaded implementation.

To perform a quantitative comparison between MSGI, DSO, and MixIT, we measure the interpolation error in terms of edge lengths and dihedral angle in the same way as Winkler et al. [WDAH10]. That is, we measure the difference between the linearly interpolated target lengths and angles and the actual lengths and angles of the interpolated shape. We measure the relative error for edge lengths and the absolute error for dihedral angles, and plot the minimum and maximum such errors. These metrics give an immediate idea of how far the result is from the ideal shape that fits the prescribed lengths and angles exactly. Although this shape may not be realizable, the lowest-error envelope does give an upper bound on the error of the realizable interpolation

	p	*q*	*Align.*	*Init.*	*Interp.*	*Stitch.*	*Total*	$\dim \mathcal{S}_m$
cat	7	6	0	16	154	242	412	25153
	30	12	0	54	700	421	1175	38610
horse	7	6	0	23	257	293	573	30725
	30	20	0	54	631	477	1177	41753
galloping elephant	7	6	13	18	368	1878	2277	136655
	18	14	13	37	653	1934	2637	144729
armadillo	5	4	64	55	966	11215	12300	514868
	14	13	48	131	2512	11373	14064	550432

Table 4: *Comparison of times and dimensions for interpolation between two poses using different numbers of rigid parts (p) and joints (q) for some of the shapes listed in Table 1. Notice that for increasing values of p the number of joints q tends to decrease because overlapping joints are merged together. All times are expressed in milliseconds.*

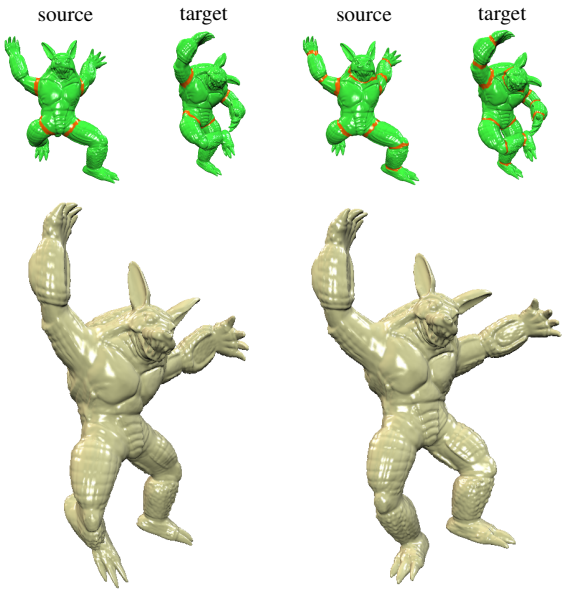


Figure 14: *Interpolation between the same poses of the armadillo using two different segmentations. A segmentation that does not capture the articulation of the legs (left) produces artefacts around the knee and the foot, which we can avoid by using a good segmentation (right).*

which follows this ideal shape as closely as possible. Our experiments show that the error behaves very similarly for all three methods: MSGI is slightly better for some examples (see Figure 10), while MixIT produces lower errors in other cases (see Figure 11). The same consideration holds for DSO (see Figure 12), since this method achieves results that are quite similar to the results using MSGI. However, the main advantage of our method is that it is significantly faster than both MSGI and DSO, in particular for large meshes (see Ta-

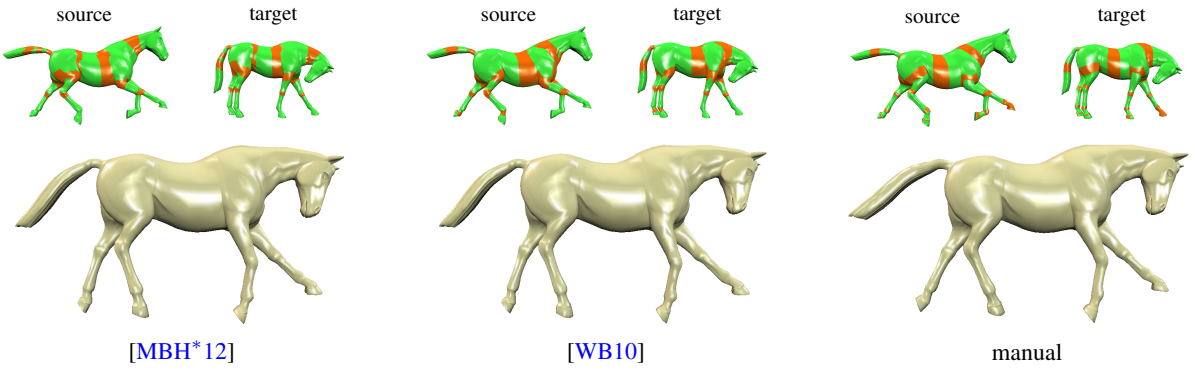


Figure 15: *Comparison of using different segmentation techniques as input to our MixIT algorithm: augmented silhouettes (left), spanning-tree cut (middle), and a manual segmentation performed by a human user (right). The interpolation results are visually similar, even though the techniques detect different numbers of joints and rigid parts. The interpolation times are similar, too: 323 milliseconds (left), 321 milliseconds (middle), and 375 milliseconds (right).*

ble 1). Moreover, the dimension of $\mathcal{S}_m$ is typically smaller than that of $\mathcal{S}_e$ by a factor between 2 and 3 (see Table 2), which greatly reduces the amount of memory needed to store shape data. The time required to compute an interpolation is strongly influenced by the dimension of $\mathcal{S}_m$ (see Tables 3 and 4). Apart from the size of a shape, this dimension depends mainly on the number of segments and on the width of the joints. In the next sections we investigate the behaviour of the algorithm with respect to these parameters.

Our DSO implementation differs from the original implementation in several aspects. For example, we do not employ a multi-resolution hierarchy and instead carry out the interpolation on the original meshes. We also use the Eigen library [GJ*10] instead of CHOLMOD [CDHR08] to handle large sparse matrices. This may account for the lower performance, shown in Table 1, compared to the implementation reported by Fröhlich and Botsch [FB11]. However, MixIT appears to be faster and less expensive than DSO, even on the basis of Fröhlich and Botsch's implementation. For example, in the case of the standing elephant (see Table 1), the original DSO implementation requires 3100 ms for a single iteration, while MixIT performs the whole interpolation in 2245 ms. Similarly, for the armadillo data set, the DSO multi-resolution approach requires approximately 21 seconds to build a coarse mesh, then 30 ms per iteration, and finally 1290 ms to go back to the original resolution, which gives a total of at least 22.5 seconds. On the other hand, MixIT computes the whole interpolation process on the full resolution mesh in 11.6 seconds.

3.1. Sensitivity 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 13) or computational speed, although performance does drop slightly for larger joints (see Table 3) 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

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) results in large errors and unwanted interpolation artefacts (see Figure 14). On the other hand, while the interpolation is resilient with respect to over-segmentation, a large number of joints means that an interpolation takes longer to compute (see Table 4).

In order to show that our method can be used with any reasonable shape segmentation algorithm, we tested our algorithm with several different initial segmentations (see Figure 15). The results show that as long as the segmentation satisfies the basic criteria discussed above, our interpolation in a mixed shape space still performs consistently in terms of both quality and computational effort.

3.3. Robustness

An interesting result from our tests is that MixIT seems to be more robust with respect to noise like vertex perturbations than MSGI. This is due to the fact that rigid parts are robustly interpolated by construction, while joints, whose interpolation may be less robust with respect to noise, cover only a small part of the shape and hence do not affect the overall result. Moreover, 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

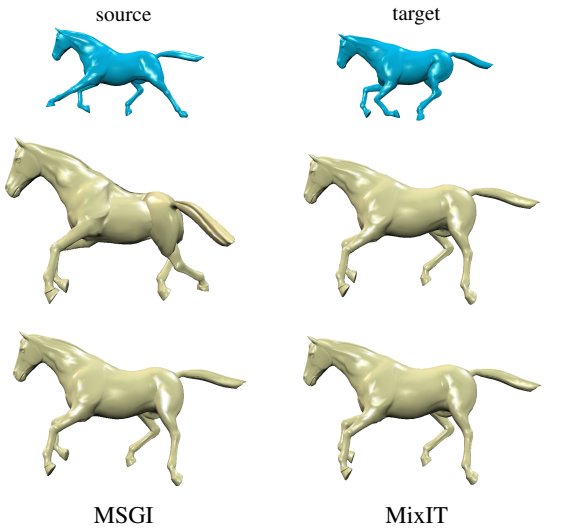


Figure 16: *Results of the interpolation between two poses of the horse (top) using MSGI and MixIT in the presence of local self-intersections (middle) and after removing them in a preprocessing step (bottom). The consistency of MixIT in the second and third rows shows that our method is more robust with respect to such artefacts.*

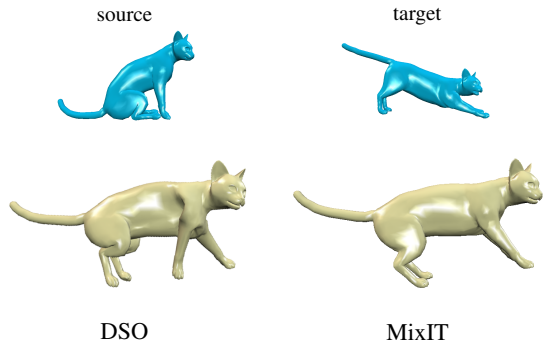


Figure 17: *Results of the interpolation between two significantly different poses of the cat (top) using DSO and MixIT. While the discrete shell optimization runs into a local minimum and the interpolated shape exhibits some artefacts, MixIT is more robust and achieves a better interpolation result.*

the self-intersections occur in the rigid parts, because the linear vertex interpolation is not affected by such artefacts (see Figure 16).

DSO also avoids these artefacts by including a volume preservation constraint in the interpolation operator. However, if the source and target poses are too different in terms of edge lengths and dihedral angles, DSO may get trapped in a local minimum, and therefore fail to achieve a suitable final configuration (see Figure 17). We have not observed any such problems with local minima using our interpolation in $\mathcal{S}_m$.

3.4. Deforming mesh sequences

Recently, Cashman and Hormann [CH12] introduced a new way to represent, visualize, and manipulate a deforming mesh sequence. The input sequence is decomposed in three separate phases, corresponding to three different and editable signals in time, pose, and shape. By modifying each of the signals it is possible, among other things, to create new frames, to resample the animation, or to transfer the deformation from one shape to another. The proposed representation can be computed for an arbitrary shape space, as long as shapes in that space can be combined with the usual Euclidean operators.

As an example application of MixIT, we show that it is possible to use $\mathcal{S}_m$ as the underlying space for this representation. This might seem difficult at first glance, as our affine combination operator A (see Section 2.2) is not a linear operator because of the alignment step that we use to combine rigid parts. This stands in contrast to simple linear interpolation in $\mathcal{S}_v$, $\mathcal{S}_f$ [SP04], or $\mathcal{S}_e$ [WDAH10]. Instead, A combines points in $\mathcal{S}_m$ in a non-linear way, as for the shape spaces described by Kilian et al. [KMP07] and Kircher and Garland [KG08].

However, if the complete set of shapes to be represented is known in advance, we can move the alignment of rigid parts into a preprocessing step. We then regain a linear affine combination operator A as every alignment transform is simply the identity. This allows us to exploit the robustness and efficiency of $\mathcal{S}_m$ in the deforming mesh sequence context. Each frame of the sequence is represented using the vertex coordinates of the aligned rigid parts plus the edge coordinates of the joint edges. Finally, to place the resulting shape in $\mathbb{R}^3$, we position each mesh using the six pose parameters explained in Section 2.3: three for the rotation of a frame face with respect to a reference triangle, plus the coordinates of one of its vertices to specify the translation. Using this representation, frames can be linearly interpolated and the resulting shape can still be properly reconstructed and positioned by the projection operator P_m. Figure 18 shows an example of applying our mixed shape space to a deforming mesh sequence and a comparison between MixIT and LRI coordinates [LSLCO05].

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. However, the class of articulated shapes is large, and our technique appears to generalize well among this class (see Figure 8).

The main advantage of our method is that it keeps the strength and the robustness of edge interpolation, while working in a shape space with significantly smaller dimension,

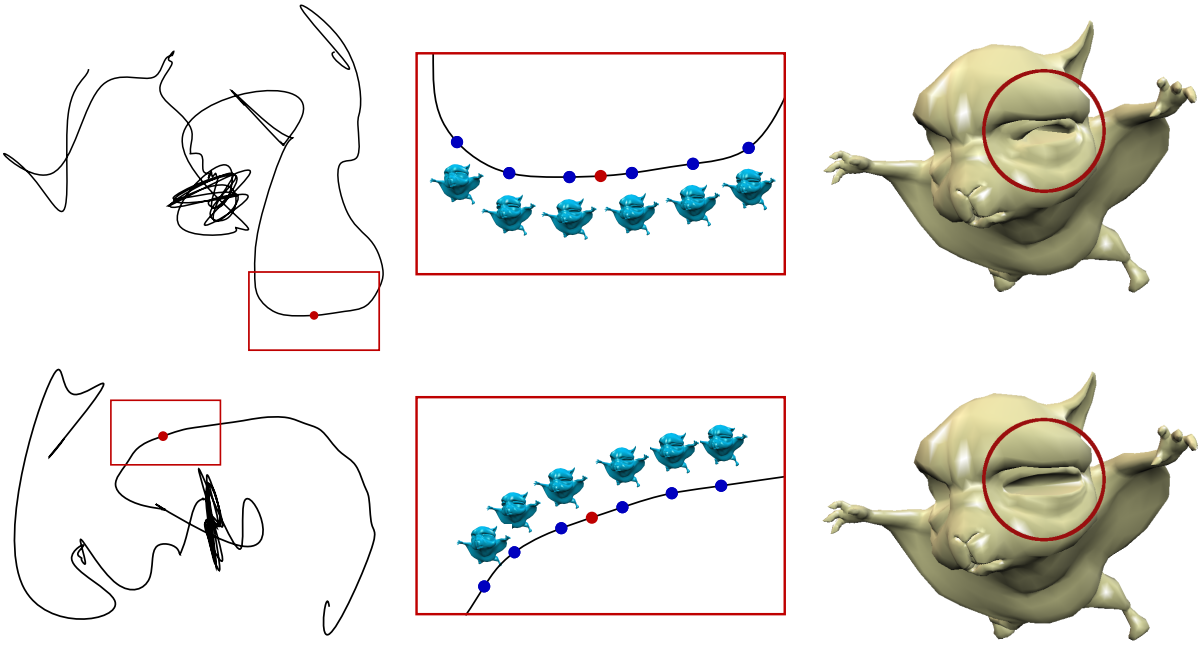


Figure 18: *Comparison between the default layout and interpolation of a given mesh sequence using LRI coordinates [LSLCO05] (top) and MixIT (bottom) in Cashman and Hormann's representation for deforming mesh sequences [CH12]. Each row shows the time signal of the sequence (left), a close-up on the region of interest (middle), and the interpolated shape (right). While interpolation with LRI coordinates may lead to artefacts in the reconstructed surface, MixIT appears to be give more reliable results, as the eye of the interpolated shape demonstrates.*

thus providing fast and high-quality interpolation. It outperforms other techniques in terms of efficiency and scalability.

In future work we plan to investigate different applications of the mixed shape space $\mathcal{S}_m$, such as compressing static and dynamic geometry, or animating static shapes to create new deforming mesh sequences.

Acknowledgements

This work was supported by the SNF under project number 200021-134639. The authors thank the anonymous reviewers for their valuable comments and suggestions, and the Blender Foundation (www.blender.org) for the sequence that appears in Figure 18 under a Creative Commons Attribution 3.0 License.

References

[aim] The aim@shape shape repository. http://shapes.aimatshape.net/. 9

[BB11] BRONSTEIN M. M., BRONSTEIN A. M.: Shape recognition with spectral distances. *IEEE Trans. Pattern Anal. Mach. Intell. 33*, 5 (2011), 1065–1071. 4

[BVGP09] BARAN I., VLASIC D., GRINSPUN E., POPOVIĆ J.: Semantic deformation transfer. *ACM Trans. Graph. 28*, 3 (2009), #36:1–6. 2

[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

[CDHR08] CHEN Y., DAVIS T. A., HAGER W. W., RAJAMANICKAM S.: Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw. 35*, 3 (2008), 2201–2214. 10

[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, 11, 12

[CLL*05] COIFMAN R. R., LAFON S., LEE A. B., MAGGIONI M., NADLER B., WARNER F., ZUCKER S. W.: Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proc. Natl. Acad. Sci. USA 102*, 21 (2005), 7426–7431. 4

[FB11] FRÖHLICH S., BOTSCH M.: Example-driven deformations based on discrete shells. *Comput. Graph. Forum 30*, 8 (2011), 2246–2257. 1, 2, 6, 10

[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. 2

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3. http://eigen.tuxfamily.org, 2010. 10

[HRWW12] HEEREN B., RUMPF M., WARDETZKY M., WIRTH

B.: Time-discrete geodesics in the space of shells. *Comput. Graph. Forum 31*, 5 (2012), 1755–1764. 1

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

[KG08] KIRCHER S., GARLAND M.: Free-form motion processing. *ACM Trans. Graph. 27*, 2 (2008), #12:1–13. 1, 2, 11

[KMP07] KILIAN M., MITRA N. J., POTTMANN H.: Geometric modeling in shape space. *ACM Trans. Graph. 26*, 3 (2007), #64:1–8. 1, 11

[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, 2, 11, 12

[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. 3, 4, 9, 10

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Comput. Graph. Forum 27*, 6 (2008), 1539–1556. 3

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (2004), 399–405. 1, 2, 9, 11

[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. 5, 6

[WB10] WUHRER S., BRUNTON A.: Segmenting animated objects into near-rigid components. *Visual Comput. 26*, 2 (2010), 147–155. 3, 10

[WDAH10] WINKLER T., DRIESEBERG J., ALEXA M., HORMANN K.: Multi-scale geometry interpolation. *Comput. Graph. Forum 29*, 2 (2010), 309–318. 1, 2, 5, 6, 9, 11

[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