Multi-Scale Geometry Interpolation

Tim Winkler · Jens Drieseberg · Marc Alexa · Kai Hormann

Abstract

Interpolating vertex positions among triangle meshes with identical vertex-edge graphs is a fundamental part of many geometric modelling systems. Linear vertex interpolation is robust but fails to preserve local shape. Most recent approaches identify local affine transformations for parts of the mesh, model desired interpolations of the affine transformations, and then optimize vertex positions to conform with the desired transformations. However, the local interpolation of the rotational part is non-trivial for more than two input configurations and ambiguous if the meshes are deformed significantly. We propose a solution to the vertex interpolation problem that starts from interpolating the local metric (edge lengths) and mean curvature (dihedral angles) and makes consistent choices of local affine transformations using shape matching applied to successively larger parts of the mesh. The local interpolation can be applied to any number of input vertex configurations and due to the hierarchical scheme for generating consolidated vertex positions, the approach is fast and can be applied to very large meshes.

Citation Info

Journal
Computer Graphics Forum

Volume
29(2), May 2010

Pages
309–318

Note
Proceedings of Eurographics

1 Introduction

Creating vertex positions or trajectories for a triangle graph from two or more sets of vertex positions is a fundamental building block of many techniques in geometry processing and animation. In the context of morphing it has been considered a challenge in its own right and is usually referred to as the *vertex path problem*. It is generally accepted that linear interpolation of vertex positions yields undesirable results, because the local shape distorts in the presence of rotations.

Better solutions to the vertex-path problem, therefore, try to preserve the local shape throughout the interpolation. There are two fundamental approaches that can be traced back to two techniques for morphing planar figures. Sederberg et al. [21] interpolate intrinsic parameters of a polygon, namely edge lengths and angles. Alexa et al. [2] identify local affine transformations for each part of the shape and then compute preferred interpolations of these transformations. Both approaches have in common that no global vertex configuration satisfies the local constraints, so that vertex positions are found by an optimization process.

The two approaches have complementary advantages and disadvantages and, as we will see, the disadvantages become more pronounced in the three-dimensional instance of the problem: interpolating edge lengths and angles is robust and fast, but it is difficult to formulate the subsequent optimization problem in such a way that it can be solved both efficiently and uniquely. The fundamental problem is that the orientation of each element of the shape is unknown, meaning that the optimization involves rotations and is non-linear.

On the other hand, prescribing local transformations for each element in the mesh defines this orientation and consolidating the different vertex positions can be done, for example, by linear least squares. But, interpolating the rotational part of each affine transformation is ambiguous. This is mainly due to the non-Euclidean nature of rotations, which makes it impossible to distinguish the *effect* of a rotation by x degrees and another by $x+360n$ degrees (for any integer n), as they both result in the same *orientation* of the rotated object [17]. Note that picking a preferred rotation (i.e. the one with smallest angle) fails if some parts in the target differ from their corresponding parts in the source by a rotation of more than 180 degrees.

In the following section we discuss several approaches that apply the idea of factoring and interpolating local affine transformations to geometric modelling approaches in 3D. We have found, however, that the fundamental ambiguity cannot be reliably resolved. Consequently, our proposed approach follows Sederberg et al. [21] and we suggest to interpolate edge lengths and dihedral angles. Note that interpolating these quantities bears a nice connection to the local metric and directional curvatures of the surface. The main contribution of our approach is applying a *global multi-registration* and to efficiently compute it using a *hierarchical structure* of the mesh.

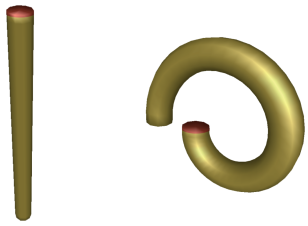

Figure 1: Example of a source mesh (left) and a target mesh (right) that illustrates the problem of interpolating large global rotations.

2 Shape interpolation based on local transformations

Sumner et al. [26] were the first to lift the idea of deriving local affine transformations to 3D and then used it in [27] for interpolation among several meshes. For interpolation, they split the matrix into a rotational and a scale/shear part. While the scale/shear part can then be interpolated linearly without any further treatment, the rotational part should be treated in log-space, which requires to compute matrix logarithms and exponentials [1].

The main drawback of this approach is that the triangles are interpolated individually, and so a situation as shown in Figure 1 cannot be handled correctly. From a global perspective, it is clear that the cylinder is deformed, but for all approaches that are based on deformation gradients [27, 11], it is not possible to distinguish between the top of the cylinder in the source mesh and the rotated top of the bent cylinder in the destination mesh. More precisely, the affine transformation between both parts is a translation. During the interpolation, this part of the mesh will therefore move in a very undesirable way, as it will not rotate at all. The same holds for Poisson shape interpolation [30], as it was shown by Botsch et al. [8] that it is equivalent to the deformation gradient approach in this setting.

The fact that deformation gradients rely upon a reference mesh can be exploited to improve the interpolation by using a global alignment in relation to the reference mesh. For example, Baran et al. [4] achieve this by factoring out the average rotation. But even then it is still not a sufficient representation method for interpolation.

What other options do we have for interpolating the rotation then? One idea is to track the rotations during a breadth-first traversal from some seed triangle. In 2D, this actually works pretty well, because all triangles are rotated about the same axis and neighbouring triangles have similar rotation angles. For example, Alexa has applied this technique to generate the results in [2]. But in 3D, this approach also requires to propagate the rotation axes, and our experiments show that it is impossible to find a globally consistent distribution of rotation axes unless the object has a very simple shape, and even then the result depends on the traversal order. Moreover, it is often the case that the natural deformation path is not a geodesic in the space of rotations and can therefore not be described correctly by a rotation about a fixed axis [13].

Another option is to take the connectivity information of the triangle mesh into account, and instead of treating all triangles individually, considering transformations that connect local frames in the mesh. Lipman et al. [18] pioneered an approach in this direction. They construct a local coordinate frame for each vertex of the mesh and then consider *connection maps* to encode the transformation between neighbouring frames. A key property of this method is that it represents the local geometry of a mesh in a rotation-invariant way, which appears to overcome the problem that all linear mesh representation (such as deformation gradients and Laplace coordinates) suffer from. Although this may cause counterintuitive results for extreme rotations (by more than 180 degrees) when used as an editing tool, it solves the orientation problem discussed above when used for interpolation. The reason is that the reconstruction process is performed in two steps: the connection maps are used to compute local frames, and based on the local frames, the vertex coordinates are reconstructed.

Kircher and Garland [16] improve upon this approach by considering affine connection maps between neighbouring triangles and storing them explicitly. This also results in a two-step linear reconstruction process, but the matrices that are involved are rather big (three times the number of triangles) and need to be factorized for each interpolation step, which in turn limits the method to rather small meshes. Baran et al. [4] reduce the rotation problem by splitting the meshes into patches such that the triangles within a single patch are not rotated by more than 180 degrees relative to the patch frame. This is a significant improvement, however, it is not clear if such a segmentation necessarily exists.

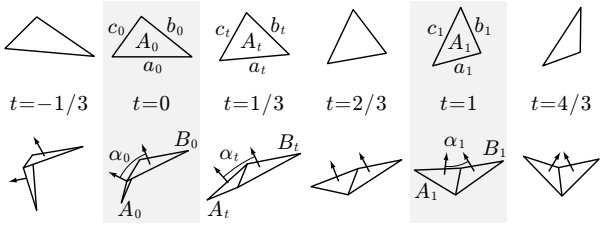

Figure 2: Linear interpolation of a single triangle (top) and a wedge (bottom).

All the approaches discussed so far try to model the non-linear nature of the problem in a linear way and require only the solution of one or two large but sparse linear systems. Clearly, the problem can also be modelled non-linearly if we accept more involved optimization. Pyramid coordinates [22] are a natural non-linear local representation of vertex positions. Kilian et al. [15] define a Riemannian metric that penalizes non-isometric deformations and search for geodesic paths in the resulting shape space. In both cases the reconstruction process is computationally intense. Lastly, another recent method [9] with very promising results tries to solve the problem of large rotations by applying a hierarchical version of the mean shift cluster algorithm.

In contrast, our approach follows the idea of Sederberg et al. [21] and interpolates the local intrinsic properties of the mesh (edges and dihedral angles). In order to derive a globally coherent solution, we utilize a hierarchical shape matching approach. The latter works for much larger meshes than any of the above mentioned techniques, and local intrinsic shape interpolation naturally applies to more than two input configurations. Moreover, our mesh representation actually provides a new kind of shape space and thus has the potential to be used for other applications beyond shape interpolation.

3 Multi-scale interpolation

Our method is based on a hierarchical approach and is similar in spirit to the shape matching in [19, 7, 24]. At the bottom of the hierarchy, we consider single triangles and linearly interpolate the local metric of the given meshes. Let A_0 and A_1 be two corresponding triangles in the source and the target mesh with edge lengths a_0, b_0, c_0 and a_1, b_1, c_1. Given some interpolation parameter $t \in [0,1]$, we construct the destination triangle A_t by linearly interpolating the edge lengths (see Figure 2). That is, A_t is a triangle with edge lengths $a_t = (1-t)a_0 + t\,a_1$, $b_t = (1-t)b_0 + t\,b_1$, $c_t = (1-t)c_0 + t\,c_1$, and it is clear by construction that such a triangle always exists. However, although this determines shape and size of A_t, we do not yet know where to place it, and that is where the hierarchy comes into play.

On the next coarser hierarchy level, we consider pairs of adjacent triangles (wedges) and linearly interpolate the local mean curvature of the given meshes. Let (A_0, B_0) and (A_1, B_1) be such corresponding wedges in the source and the target mesh with dihedral angles α_0 and α_1 at the common edge (see Figure 2). We then glue the interpolated triangles A_t and B_t together such that they form a wedge with dihedral angle $\alpha_t = (1-t)\alpha_0 + t\,\alpha_1$. Note that A_t and B_t fit together seamlessly as the common edge has the same length in both triangles. Again, this does not tell us where to place the wedge (A_t, B_t) in space, but at least it determines the *relative position* of one triangle to the other.

In some sense, this method interpolates between corresponding wedges as rigidly as possible, because both the local metric (edge lengths) and the local mean curvature (dihedral angle) are interpolated in the straightest way (linearly). It now remains to paste all interpolated wedges together in order to yield the interpolated mesh. If we consider all possible wedges, that is, one for each edge in the mesh, then it is clear that for each triangle there are three wedges that overlap, like the scales of a fish. And it is this small overlap that can be exploited in order to arrange all wedges globally. All we have to do is to find a set of rigid transformations, one for each wedge, such that the overall sum of distances (or rather squared distances) between all corresponding vertices for two overlapping wedges is minimized. Such a global alignment procedure for all wedges can in principle be solved by a multi-registration method, but for meshes with more than a few hundred triangles this can become very slow and unstable.

Thus, we take further advantage of our hierarchical approach. On the next coarser level above the wedges in our hierarchy, we cluster all the wedges around a common vertex and align them with the multi-registration method of Williams and Bennamoun [28] (see Section 3.1). Once aligned, we average the coordinates of cor-

responding vertices (see Section 3.2) and combine the wedges to form a one-ring. If we do so for all vertices of the mesh, we get a set of larger patches, which again overlap by an even bigger amount (neighbouring one-rings share two triangles), like the scales of an armadillo. Proceeding this way recursively, we create larger and larger patches by always clustering, aligning, and blending a small number (4 to 10) of overlapping and neighbouring patches. And at the top of the hierarchy we get the interpolated mesh, similar to the way the scales of a crocodile form its exoskeleton.

Note that this method clearly reproduces T_0 and T_1 for $t = 0$ and $t = 1$, respectively, because in both cases the patches can be aligned with zero distance on all levels of the hierarchy.

3.1 Aligning patches

Let us now take a closer look at how to align multiple patches, so that they form a globally consistent mesh. This problem is somewhat similar to the registration of point clouds in 3D. But our setting is even simpler, in that we already know all corresponding vertices and do not have to bother searching for closest points. Consider, for example, the alignment of two neighbouring wedges. Since they have an overlap of one triangle, there are exactly three corresponding vertex pairs (the corners of said triangle), and that suffices to optimally align them. And on all upper levels of the hierarchy, neighbouring patches overlap by even more triangles.

In order to register two neighbouring patches, we could apply the method of Besl and McKay [6] or one of its improved siblings and directly compute the optimal rotation and translation for transforming one patch such that it aligns best to the other. But unfortunately, pairwise registration leads to error accumulation in the hierarchy. We therefore need a method that allows to distribute the registration error between the patches as equally as possible.

We tested several such multi-registration methods [25, 10, 20] and found the one of Williams and Bennamoun [28] to be very well adapted to our particular problem. Their goal is to *simultaneously* determine *all* rigid transformations (translation and rotation), i.e., one for each of the patches that need to be registered. This is done by minimizing a cost function which sums up all the squared distances between corresponding vertices for neighbouring patches. The key idea of their approach is to pre-compute in an elegant way a constant matrix that encodes the whole registration problem. This matrix then allows to iteratively solve for the best rigid transformations, and only few iterations suffice to get close to the optimal solution. In our setting, we can actually stop the iterations rather early (after 4 passes), for we post-process the alignment anyway in order to smoothen the result (see Section 3.2).

Due to the iterative nature of the method, one would assume that a good starting solution is essential. Yet, we learned from our experiments that the approach is robust enough to allow for an initialization of all rotations with the identity matrix. Moreover, the result is (in principle) unique only up to a global rotation, but this can easily be adjusted by constraining one of the rotation matrices to be the identity matrix, thereby fixing the global orientation of the corresponding patch. Finally, it is possible (in case of coplanar or collinear data), that some of the resulting rotation matrices have a negative determinant and thus contain an unwanted reflection. We fix this as described in [3], that is, we simply negate the last column of these matrices and continue iterating.

3.2 Blending patches

Once we have computed the best rigid transformations for a set of m small patches $P^{[1]}, \ldots, P^{[m]}$ at some level of our hierarchy, it remains to blend them into a consistent larger patch P at the next coarser hierarchy level. In general, even an optimal alignment still leaves a small gap between corresponding vertices so that the patches do not fit together seamlessly in their overlap region. At first, we tried to simply average the coordinates of corresponding vertices, but this turned out to be insufficient as the alignment errors still tend to accumulate, yielding an unsatisfactory overall result for the whole mesh. Instead, we decided to distribute the remaining alignment errors in a more global way as follows.

Inspired by the handling of the consistency requirements in the deformation gradient setting [26], we determine the coordinates of the vertices in the large patch P such that its edges deviate as little as possible from all the corresponding edges in the small patches $P^{[k]}$. Suppose that $\boldsymbol{v} = (v_1, \ldots, v_n)$ are the vertices of P and that $[v_i, v_j]$ is one of the edges in P. Then this edge also occurs in some of the small patches $P^{[k]}$, but there it is spanned by the vertices with local coordinates $v_i^{[k]}$ and $v_j^{[k]}$ (see Figure 3). Ideally, the vertices $\boldsymbol{v}$ of P should be such that

$$v_i - v_j = v_i^{[k]} - v_j^{[k]} \tag{1}$$

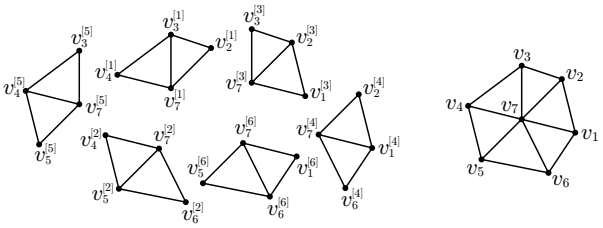

Figure 3: Notation for blending $m = 6$ wedges $P^{[1]}, \ldots, P^{[6]}$ (left) into a one-ring P (right) with $n = 7$ vertices.

for all edges $[v_i, v_j]$ in P and all its occurrences in the small patches $P^{[k]}$. Gathering all these conditions yields a linear system

$$M\boldsymbol{v} = \boldsymbol{e}, \tag{2}$$

where $\boldsymbol{v}$ are the unknown vertex positions of P, M is a large sparse matrix with exactly two entries 1 and -1 per row, reflecting the left hand side of (1), and $\boldsymbol{e}$ is the vector of corresponding edges from the small patches, reflecting the right hand side of (1). In general, this is an overdetermined linear system and we compute its least squares solution by solving the linear problem

$$\min_{\boldsymbol{v}} \|M\boldsymbol{v} - \boldsymbol{e}\|_2^2 \qquad \Longleftrightarrow \qquad M^T M\boldsymbol{v} = M^T\boldsymbol{e}.$$

Similar to the linear systems that appear in the work of Sumner et al. [27] and Kircher and Garland [16], the solution is unique only up to translation. In our setting, we resolve this by simply adding the constraint $v_1 = 0$ to the linear system.

Note that the system matrix $M^T M$ does not depend on the blending parameter t. Hence we can factorize it in a pre-processing step when building the hierarchy, so as to allow for a more efficient construction of the interpolated meshes when the user explores the shape space.

We can improve the quality of the result by replacing condition (1) with

$$v_i - v_j = \frac{v_i^{[k]} - v_j^{[k]}}{\left\| v_i^{[k]} - v_j^{[k]} \right\|} s_{ij}(t), \tag{3}$$

where $s_{ij}(t)$ is the linear interpolation of the lengths of edge $[v_i, v_j]$ in the source and the target mesh. In this way, we only keep the *orientation* of the edge from the small patch, but enforce its *desired length* on each hierarchy level. Overall, this drastically reduces the deviation of the edge lengths in the interpolated mesh from the ideal, linearly interpolated lengths that we use at the bottom of the hierarchy to assemble the interpolated single triangles. It is remarkable to note that solving the linear system (2) with the conditions from (3) seems to yield a locally optimal solution. That is, solving the system iteratively, with the edges $v_i - v_j$ from the current solution instead of $v_i^{[k]} - v_j^{[k]}$, does not improve the edge lengths.

Moreover, this approach allows to apply additional local deformations to parts of the mesh by modifying the destination lengths. For example, we can scale parts of the mesh by tagging a set of edges and multiplying the corresponding lengths $s_{ij}(t)$ with some common scaling factor, as shown in Figure 12.

3.3 Building the hierarchy

One question that we have not yet answered is how to get the adjacency information about the patches and how to set up the hierarchy. A common practice is to recursively decimate or cluster the mesh to construct a progressive hierarchy. Then a multi-grid method is used to propagate the solution from the coarsest up to the finest level. For example, Botsch et al. [7] solve their hierarchical shape matching problem this way. Our approach is different in that we recursively split the mesh into smaller and smaller patches, resulting in a hierarchy tree.

We start by taking the complete mesh as root node of the tree (level 0) and then descend one level by taking as many random seed triangles as we want the hierarchy to have patches on each level, say m. Using these seed triangles, we concurrently apply a region growing step until the patches meet and overlap by a triangle strip of width one. Our experiments show that it does not matter how the seed triangles are located, but theoretically it is better to place them such that the resulting patches have an irregular boundary and hence a larger overlap region. This creates the first m patches of the hierarchy tree at level 1.

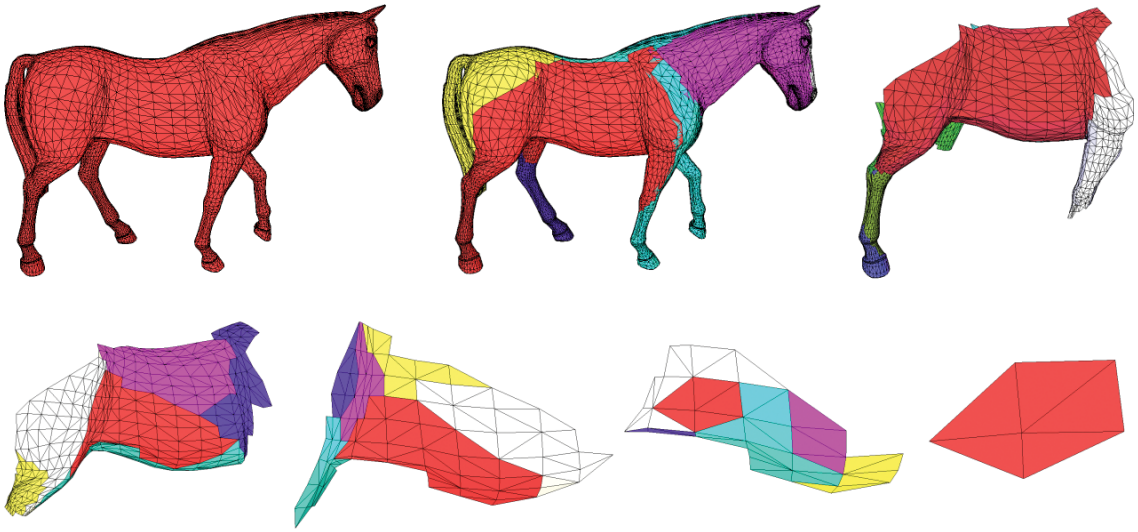

Figure 4: Building the hierarchy: the whole mesh (root node at level 0) is split into m patches at level 1, which in turn get split into m patches each at level 2 and so on, until the patches consist of less than m triangles (lowest level). From top left to bottom right: one branch of the hierarchy tree, where it is always the red patch whose split is shown in the next picture.

By recursively applying this scheme, we build the whole tree until we arrive at the lowest level (see Figure 4). Theoretically, this should be the level of triangles, but it turned out to be sufficient to stop as soon as the patches consist of less than m triangles. At this lowest level, we interpolate each of the triangles and the dihedral angles between neighbouring ones as described above and glue them together in a greedy way. That is, we start with any of them and keep attaching the others one by one, respecting the desired dihedral angles. If such a lowest level patch is a triangle strip, then this is actually the best one can do. But if it is a triangle fan (or contains one), then this simple strategy may create gaps because the fan does not necessarily close up perfectly. However, we found that the blending procedure (see Section 3.2) takes care of this and smoothes these imperfections. Overall, this speeds up the interpolation process, because it reduces the number of hierarchy levels.

Although the multi-registration step (see Section 3.1) allows any number of patches to be aligned, we found that using $m = 6$ patches per node gives the best trade-off in terms of computation time. A smaller m creates too many hierarchy levels, and a larger m slows down the multi-registration steps, because each of them requires to compute the singular value decomposition of a matrix whose size is $3m \times 3m$ (see [28] for more details).

3.4 Multiple input meshes

Since our approach is based on linear interpolation, it trivially allows for the interpolation between more than two input meshes. Let $a_1, \ldots, a_n$ be the lengths of a corresponding edge in n input meshes, and let $\boldsymbol{t} = (t_1, \ldots, t_n) \in [0,1]^n$ be an n-dimensional interpolation parameter. Then the interpolated edge length is $a_{\boldsymbol{t}} = t_1 a_1 + \ldots + t_n a_n$, and likewise for the interpolated dihedral angles $\alpha_{\boldsymbol{t}} = t_1 \alpha_1 + \ldots + t_n \alpha_n$. Once these values have been used to construct the two lowest levels of the hierarchy (single triangles and wedges), the remaining levels are constructed in the same way as described above. In the example shown in Figure 5, we use mean value coordinates [12] with respect to the corners of the control polygon as interpolation parameter $\boldsymbol{t}$.

3.5 Extrapolation

In principle, our method can also be used for extrapolating between two or more input meshes, that is, the interpolation parameter can be chosen outside the range $[0,1]$. But it is then no longer guaranteed that the interpolated edge lengths match up to form a triangle (an interpolated edge can end up being negative or bigger than the sum of the other two), and the interpolated dihedral angle may leave the valid range between -180 and $+180$ degrees. However, we found that this happens only for rather extreme extrapolations and works well in most cases (see Figure 10 and 11).

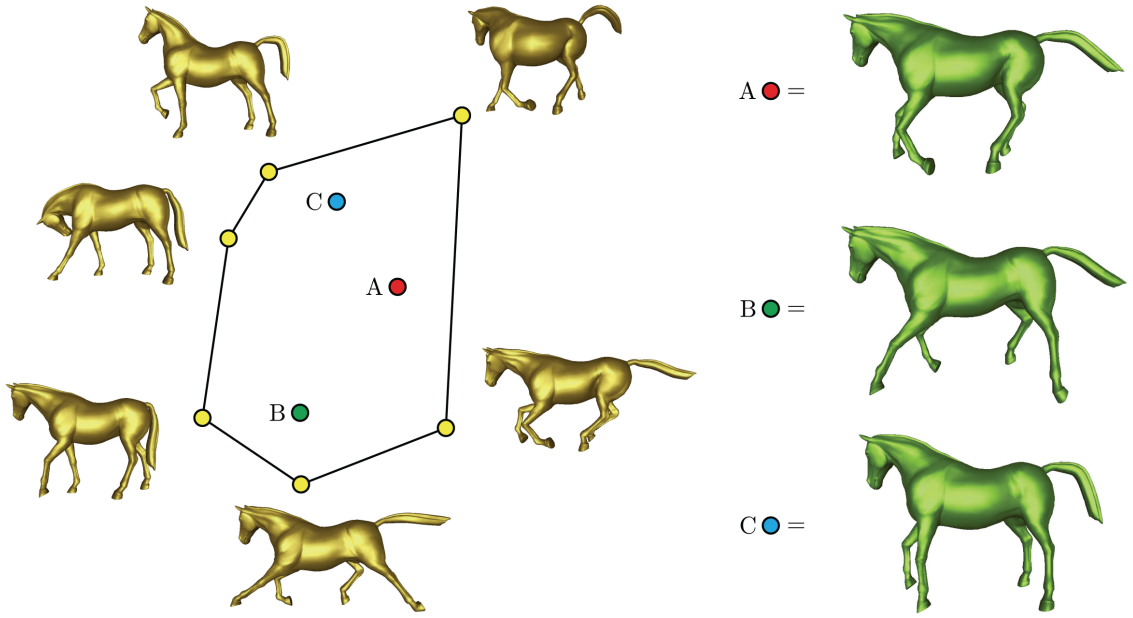

Figure 5: Interpolation between multiple input meshes. The n reference meshes correspond to the vertices of a control polygon in 2D, and any point within this polygon corresponds to an interpolated pose. The user can change the interpolated mesh by either moving the reference point inside the polygon, or by changing the shape of the control polygon.

4 Results

In general our results are comparable in quality to all state-of-the-art approaches [15, 16, 9], but our method is significantly simpler and faster. The only method that can compete in terms of speed, is the one by Kircher and Garland [16], but as mentioned above we can handle much larger meshes, because our mesh hierarchy decomposes them into digestible chunks.

4.1 Interpolation

Figure 6 confirms that our approach generates basically the same interpolation results as the ones by Kilian et al. [15] and Chu et al. [9], but we are at least an order of magnitude faster (see Table 1). The armadillo example emphasizes that very large meshes do not pose a challenge for our method, because of our hierarchical structure.

The cylinder deforming into a helix probably illustrates best what we are striving for in this paper. The interpolation follows the multiple rotations (far greater than 180 degrees) in a visually plausible way, due to the fact that the multi-registration step takes care of the global rotation. Note that the result shows a remarkable resemblance to the outcome of the physical simulations of discrete elastic rods [5] and does not suffer from the fact that the triangle density varies strongly over the mesh.

Figures 13 and 14 further show that the method is robust enough to tolerate a reasonable amount of vertex noise and able to handle long and skinny triangles.

Figure 7 illustrates how little the edge lengths in the interpolated mesh deviate from the desired lengths that we impose at the bottom of our hierarchy. The plots show the maximum and the minimum relative difference from the linearly interpolated lengths. While the global worst case is about -12% the major fraction of the edges (99%) do not differ by more than 2.5% from the ideal value. This confirms that our alignment and blending steps affect the local metric only very slightly. Figure 8 shows the equivalent plots for the angles, but this time on an absolute scale. Although the overall worst case differs by more than 100 degrees from the ideal value, such extreme deviations are very rare to happen. For 99.9% of all edges, the interpolated dihedral angle lies within ± 5 degrees from the linearly interpolated angle, and the maximum error of 99% of the angles is less than 1 degree. Again this shows how well our global alignment procedure keeps the dihedral angles that we impose on the wedge-level. Interestingly, our method also does a good job in preserving the volume of the meshes during interpolation as shown in Figure 9, although we do not directly consider this as a constraint during our reconstruction.

Figure 6: Interpolation between two input meshes (leftmost and rightmost column). The interpolated poses (in green) are shown for parameter values of $t = 0.25$, $t = 0.5$, $t = 0.75$.

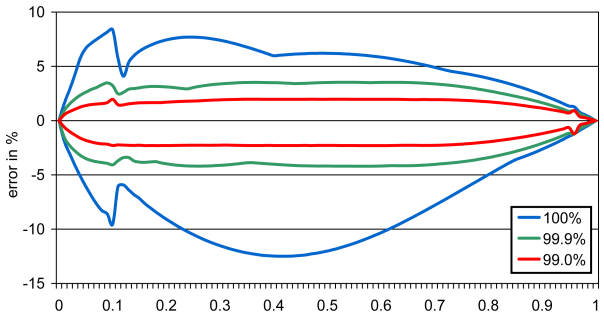

Figure 7: Relative error between the effective edge lengths in the interpolated mesh and the ideal linearly interpolated lengths, for the elephant example in Figure 11. Plotting this error for all edges yields the envelope represented by the blue curves. Neglecting the 0.1% edges with the worst deviation results in the green envelope, and the red one visualizes the envelope of 99% of all edges.

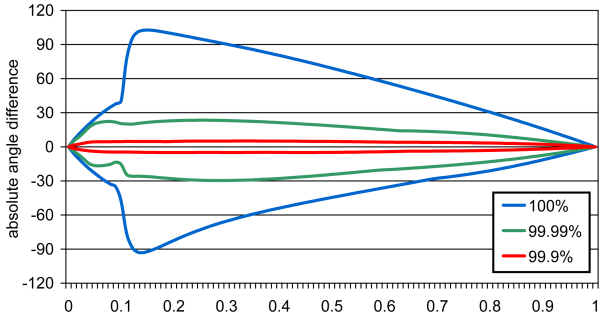

Figure 8: Relative error between the effective dihedral angles and the linearly interpolated ones for the elephant example. Compare to Figure 7.

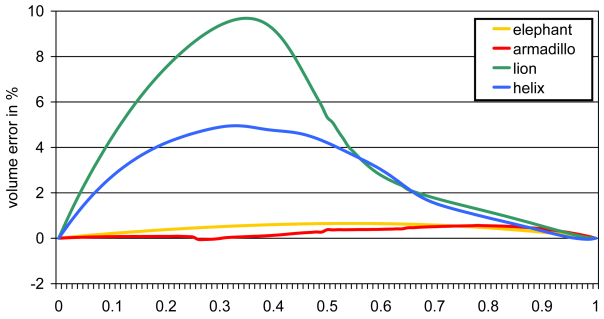

Figure 9: Relative error between the volume of the interpolated mesh and the volume of the input meshes for the examples in Figures 6 and 11.

4.2 Extrapolation and constrained interpolation

As mentioned in Section 3.5 our approach can also be used for extrapolation, according to the limits mentioned. For the elephant example in Figure 11 we cannot go much beyond the interval $[-0.25, 1.25]$, but if the edge lengths and dihedral angles in the input meshes are quite similar, then we can actually extrapolate quite far as shown in Figure 10.

Figure 12 finally shows an example of a constrained interpolation as explained at the end of Section 3.2. We applied an additional scaling factor of $t+1$ to the lions head, tail and paws (i.e., the scaling factor varies linearly from 1 at the source mesh to 2 at the target mesh).

4.3 Timings

The timings in Table 1 report that the preprocessing step is the computationally most intense part of the approach, but still reasonable even for large meshes. Once this work is done, constructing an interpolated pose is not too expensive. The cost for interpolating the edge lengths and dihedral angles at the bottom of the hierarchy is negligible and the registration time grows linearly with the number of triangles. From a certain mesh size on, the blending step becomes the most expensive part of the pipeline, since the matrices from Equation (1) are then relatively large on the top of the hierarchy. All timings were measured on a Core2Duo Laptop with 4 GB RAM and a 2.5 Ghz CPU.

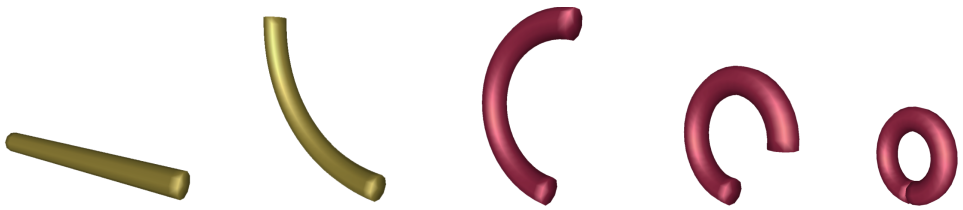

Figure 10: Extrapolation of a cylinder. The two input meshes and extrapolated poses for parameter values of $t=2$, $t=3$, $t=4$ (from left to right).

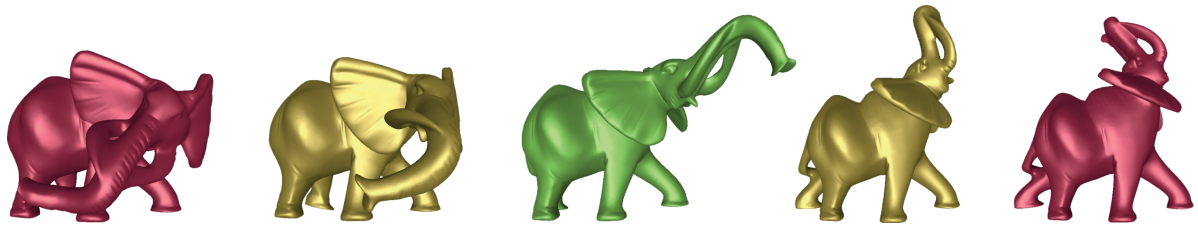

Figure 11: Extrapolating the elephant. The parameter values of the meshes are $t = -0.25$, $t = 0$, $t = 0.5$, $t = 1$, $t = 1.25$ (from left to right).

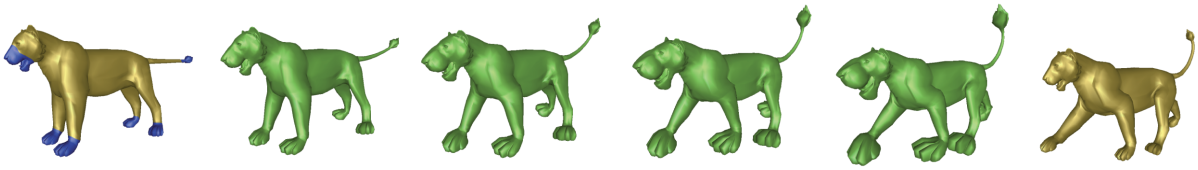

Figure 12: Example of a constrained interpolation for the lion using a scaling factor of $2t$ for the edges in the selected regions (blue). The parameter values of the interpolated meshes are $t = 0.25$, $t = 0.5$, $t = 0.75$, $t = 1$ (from left to right).

5 Conclusion

We presented a novel method for interpolating between two or more compatible meshes that is not based on local affine transformations. Instead we linearly interpolate the intrinsic local properties of the input meshes, which is the most natural and simplest thing to do on the level of wedges. The simplicity of this idea is counterweighed by the fact that putting the wedges together such that they yield a globally consistent mesh is a rather complex optimization problem. However, we found that this problem can be solved in principle by multi-registration methods and since this can be combined with a hierarchical decomposition of the mesh in larger and larger patches, it can actually be solved efficiently as well.

As the examples confirm, our approach yields very intuitive interpolations and can hence be used for exploring the natural space shape spanned by a set of key poses. This allows for a number of interesting applications. For example, approaches like [23, 29, 14] can be used on top of our method, and it can also be used for shape editing based on user-defined constraints, similar to how it is described in [15]. Moreover, our framework can be used to express animation sequences as simple 2D paths in a reference polygon (compare Figure 5), which in turn may provide an intuitive tool for character animation or even crowd generation.

Acknowledgements

We would like to thank Martin Kilian for providing the meshes used in [15] and especially John A. Williams for helpful hints on his implementation of the multi-registration method.

meshes			pre-processing		mesh interpolation		
	vertices	faces	splitting	factorizing	interpolation	registration	blending
cylinder	312	620	2.94	59.71	1.17	18.78	5.21
helix	1212	2420	12.75	243.76	3.98	84.11	24.66
lion	5000	9996	127.55	1248.22	16.05	365.99	122.45
horse	8431	16843	300.41	2140.96	72.45	544.01	205.93
elephant	39969	79946	5975.31	11954.33	126.15	2632.40	1327.28
armadillo	165954	331904	90106.71	54339.12	570.07	11273.22	7105.04

Table 1: Timings for all meshes shown throughout the paper, measured in milliseconds.

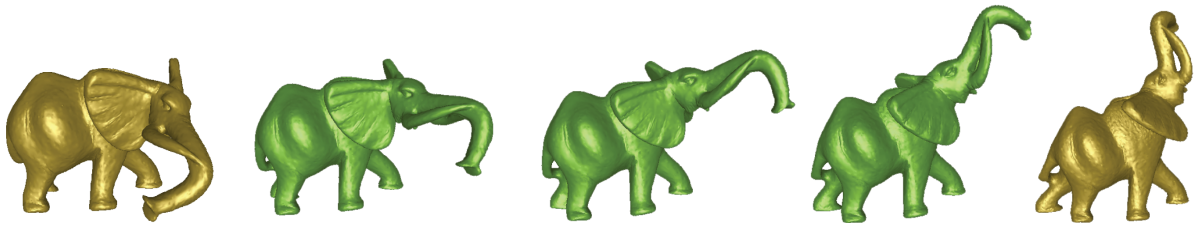

Figure 13: Interpolating the elephant with additional noise (compare with Figure 11). The parameter values of the interpolated poses are $t = 0.25$, $t = 0.5$, and $t = 0.75$ (from left to right).

Figure 14: Interpolating long and skinny triangles. The parameter values of the interpolated poses are $t = 0.25$, $t = 0.5$, and $t = 0.75$ (from left to right).

References

[1] M. Alexa. Linear combination of transformations. *ACM Transactions on Graphics*, 21(3):380–387, July 2002. Proceedings of SIGGRAPH.

[2] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Proceedings of SIGGRAPH 2000*, pages 157–164, July 2000.

[3] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, Sept. 1987.

[4] I. Baran, D. Vlasic, E. Grinspun, and J. Popović. Semantic deformation transfer. *ACM Transactions on Graphics*, 28(3):36:1–36:6, Aug. 2009. Proceedings of SIGGRAPH.

[5] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. Discrete elastic rods. *ACM Transactions on Graphics*, 27(3):63:1–63:12, Aug. 2008. Proceedings of SIGGRAPH.

[6] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb. 1992.

[7] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. PriMo: coupled prisms for intuitive surface modeling. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, pages 11–20, Cagliari, Italy, June 2006.

[8] M. Botsch, R. Sumner, M. Pauly, and M. Gross. Deformation transfer for detail-preserving surface editing. In L. Kobbelt, T. Kuhlen, T. Aach, and R. Westermann, editors, *Proceedings of Vision, Modeling, and Visualization 2006*, pages 357–364, Aachen, Germany, Nov. 2006. Aka.

[9] H.-K. Chu and T.-Y. Lee. Multiresolution mean shift clustering algorithm for shape interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):853–866, Sept. 2009.

[10] S. J. Cunnington and A. J. Stoddart. N-view point set registration: A comparison. In *Proceedings of the 10th British Machine Vision Conference*, pages 234–244, Nottingham, UK, Sept. 1999.

[11] K. G. Der, R. W. Sumner, and J. Popović. Inverse kinematics for reduced deformable models. *ACM Transactions on Graphics*, 25(3):1174–1179, July 2006. Proceedings of SIGGRAPH.

[12] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, Mar. 2003.

[13] S. Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, Mar. 1998.

[14] N. Hasler, C. Stoll, M. Sunkel, B. Rosenhahn, and H. P. Seidel. A statistical model of human pose and body shape. *Computer Graphics Forum*, 2(28), Mar. 2009. Proceedings of Eurographics.

[15] M. Kilian, N. J. Mitra, and H. Pottmann. Geometric modeling in shape space. *ACM Transactions on Graphics*, 26(3):64:1–64:8, July 2007. Proceedings of SIGGRAPH.

[16] S. Kircher and M. Garland. Free-form motion processing. *ACM Transactions on Graphics*, 27(2):12:1–12:13, Apr. 2008.

[17] J. Lee. Representing rotations and orientations in geometric computing. *IEEE Computer Graphics and Applications*, 28(2):75–83, Mar./Apr. 2008.

[18] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics*, 24(3):479–487, July 2005. Proceedings of SIGGRAPH.

[19] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Transactions on Graphics*, 24(3):471–478, July 2005. Proceedings of SIGGRAPH.

[20] H. Pottmann, S. Leopoldseder, and M. Hofer. Simultaneous registration of multiple views of a 3D object. In R. Kalliany and F. Leberl, editors, *Photogrammetric Computer Vision*, volume 34, Part 3A of *Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 265–270. ISPRS, 2002.

[21] T. W. Sederberg, P. Gao, G. Wang, and H. Mu. 2-D shape blending: an intrinsic solution to the vertex path problem. In *Proceedings of SIGGRAPH 1993*, pages 15–18, Aug. 1993.

[22] A. Sheffer and V. Kraevoy. Pyramid coordinates for morphing and deformation. In *Proceedings of the Second International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 68–75, Thessaloniki, Greece, Sept. 2004. IEEE Computer Society.

[23] R. C. Smith, R. Pawlicki, I. Kókai, J. Finger, and T. Vetter. Navigating in a shape space of registered models. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1552–1559, Nov. 2007.

[24] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, pages 109–116, Barcelona, Spain, July 2007.

[25] A. J. Stoddart and A. Hilton. Registration of multiple point sets. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 40–44, Vienna, Austria, Aug. 1996.

[26] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):399–405, Aug. 2004. Proceedings of SIGGRAPH.

[27] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popović. Mesh-based inverse kinematics. *ACM Transactions on Graphics*, 24(3):488–495, July 2005. Proceedings of SIGGRAPH.

[28] J. A. Williams and M. Bennamoun. Simultaneous registration of multiple point sets using orthonormal matrices. In *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2199–2202, Istanbul, Turkey, June 2000.

[29] T. Winkler, J. Drieseberg, A. Hasenfuß, B. Hammer, and K. Hormann. Thinning mesh animations. In O. Deussen, D. Keim, and D. Saupe, editors, *Proceedings of Vision, Modeling, and Visualization 2008*, pages 149–158, Konstanz, Germany, Oct. 2008. Aka.

[30] D. Xu, H. Zhang, Q. Wang, and H. Bao. Poisson shape interpolation. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, pages 267–274, Cambridge, MA, June 2005.