Recent Advances in Mesh Morphing

Marc Alexa
Interactive Graphics Systems Group, Department of Computer Science, Technische Universität Darmstadt
Rudower Chaussee 6, 64281 Darmstadt, Germany
email: alexa@gris.informatik.tu-darmstadt.de
phone: +49 6151 155 674, fax: +49 6151 155 669

Abstract
Meshes have become a widespread and popular representation of models in computer graphics. Morphing techniques aim at transforming a given source shape into a target shape. Morphing techniques have various applications ranging from special effects in television and movies to medical imaging and scientific visualization. Not surprisingly, morphing techniques for meshes have received a lot of interest lately. This report sums up recent developments in the area of mesh morphing. It presents a consistent framework to classify and compare various techniques approaching the same underlying problems from different angles.

1. Introduction
Morphing techniques transform one shape into another\textsuperscript{1}. With the introduction in TV and movies, morphing is nowadays known to an audience beyond the computer graphics community. At the same time, morphing has established itself as an interesting research area. Recently, the focus is shifting from handling representations of space (images, volumes) to using explicit boundary representations, interpolating or blending the shape of the objects. This work concentrates on the process of preserving linear boundary representations, namely meshes.

Blending shapes rather than the space they are embedded in can lead to better results but is also more involved, because a proper mapping between the shapes is needed. Defining such a mapping is not trivial for two main reasons. First, it requires a parametrization of the boundary representation, and second, the mapping might involve shapes with different topology.

Besides the parameterization problem, which is fundamental in many areas dealing with meshes, morphing also requires to find suitable paths for the elements of a mesh. This part has an aesthetic component, however, several reasonable conditions should be observed; i.e. the shape should not self intersect or collapse as it varies from source to target configuration.

Traditionally, morphing is applied to two shapes: a source and a target shape. Morphing among more than two shapes can be seen as generating elements in a space of shapes. This has interesting applications for modeling, animation, and analysis. Analysis using well-established methods such as the principal component analysis has gained interest lately.

This report explains mainly techniques for morphing between two meshes. This avoids some clutter in the formalism. Once all methods are explained for two meshes, possible extensions to more than two meshes and their applications are discussed.

2. Terminology & Framework
Mesh morphing techniques involve computations on the geometry as well as the connectivity of meshes. For simplicity this report concentrates on triangle meshes. In the context of morphing it seems to be acceptable to triangulate polygonal meshes prior to the application of a morphing technique. To classify and understand mesh morphing techniques it is helpful to use the now widespread terminology from Spanner\textsuperscript{15}.

A mesh \( M \) is described by a pair \((K, V)\), where \( K \) is a simplicial complex representing the connectivity of vertices, edges, and faces and \( V = \{v_1, \ldots, v_n\} \) describes the geometric positions of the vertices in \( \mathbb{R}^3 \), where typically \( n \geq 3 \).

The abstract complex \( K \) describes vertices, edges, and faces as \((0, 1, 2)\)-simplices, that is, edges are pairs \( \{i, j\} \), and faces are triples \( \{i, j, k\} \) of vertices. The topological re-identification maps \( K \) to a simplicial complex \( K' \) in \( \mathbb{R}^2 \). The vertices are identified with the canonical basis of \( \mathbb{R}^2 \) and each simplex \( x \in K' \) is represented as the convex hull of the points \( (\phi_i(x), \phi_j(x), \phi_k(x)) \). Thus, each 0-simplex is a point, each 1-simplex is a line segment, and each 2-simplex is a triangle in \( \mathbb{R}^2 \).

The geometric realization \( \phi(K) \) is a linear map of the simplicial complex \( K' \) to \( \mathbb{R}^3 \), which is defined by associating the basis vectors \( e_i \in \mathbb{R}^3 \) with the vertex positions \( v_i \in \mathbb{V} \). The map \( \phi_0 \) is an embedding if \( \phi_0 \) is injective. The importance of an embedding is that every point on the mesh can be uniquely represented with a barycentric coordinate \( b_0 \), i.e. \( p = \phi_0(b_0) \). Such barycentric coordinates have at most three non-zero components and specify the position of a point relative to a simplex. If the point coincides with a vertex it is a canonical basis vector, if the point lies on an edge it has two non-zero components, otherwise it has three and lies on a face.

The neighborhood ring of a vertex \( v \) is the set of adjacent vertices \( \pi(v) = \{v_i | v_i \in K \} \) and it is the set of the star incident simplices \( S(v) = \cup_{v_i \in \pi(v)} K_i \).

In the classical setting of mesh morphing two meshes \( M_0 = (K_0, V_0) \) and \( M_1 = (K_1, V_1) \) are given. The goal is to generate a family of meshes \( M(t) = (K(t), V(t)) \), \( t \in [0, 1] \) so that the shape represented by the new connectivity together with the geometries \( V(0) \) and \( V(1) \) is identical with the original shapes, i.e. \( \phi_0(K) = \phi_0(K_0) \) and \( \phi_1(K) = \phi_1(K_1) \). Most of the time the paths \( V(t) \) are required to be smooth. The generation of this family of shapes is typically done in three subsequent steps:

1. Finding a correspondence between the meshes. More specifically, computing coordinates \( W_0, W_1 \) that lie on the other mesh, i.e. \( W_i \in \phi_i(K_1) \). \( W_0 \) and \( W_1 \) are fixed vertices in \( W \). Each coordinate in \( W_0, W_1 \) is represented as a barycentric coordinate with respect to a simplex in the other mesh. Note that \( W_0 \) will not map \( K_0 \) to \( \phi_1(K_1) \) (and vice versa), as only the vertices are mapped to the other mesh but not the edges and faces. Particularly important is the adjustment of automatically detected or user specified features of the meshes.

2. Generating a new, consistent mesh connectivity \( K \) together with two geometric positions \( V(0), V(1) \) for each vertex so that the shapes of the original meshes are reproduced. The traditional morphing approach to this problem is to create a superset of the simplicial complexes \( K_0 \) and \( K_1 \). However, replacing techniques as used in multisresolution techniques are also attractive.

3. Creating paths \( V(t), t \in [0, 1] \) for the vertices. While in general this is an aesthetic problem, several constraints seem reasonable to help in the design process. For example, in most applications the shape is not expected to collapse or self intersect and, generally, the paths are expected to be smooth.

In the following, recent work will be explained in terms of the above mentioned problem areas. This state of the art report focuses on mesh morphing, however, if believed to be instructive also techniques dealing with polygons are discussed.

3. Correspondence of shapes
In this section we aim at finding corresponding vertex positions on two or more meshes. Given two meshes \( M_0 \) and \( M_1 \), the result of this procedure is a set of barycentric coordinates \( B_0 \) so that the geometry \( W_0 = \phi_0(B_0) \) of the barycentric coordinates on \( M_0 \) is an embedding \( \phi_0 \) of \( M_0 \) on the surface of \( M_1 \) and vice versa. The idea is that this mapping of vertices from one mesh to the other accomplishes the main part of a bijection mapping between the surfaces of \( M_0 \) and \( M_1 \). After this step only the edges and faces have to be adjusted accordingly.

The process is typically done by finding a common parameter domain \( D \) for the surfaces. By mapping each surface bijectively to that parameter domain, the mapping between the shapes is established. The typical parameter domain for meshes in the context of morphing are the sphere \( S^2 \) (in case the meshes are topological spheres) or a collection of topological disks represented as a piecewise linear parameter domain \( L \). In case of the disks, the meshes have to be decomposed into isomorphic structures of disks (which requires them to be homeomorphic). A major constraint is to take into account user specified or automatically generated feature correspondences (i.e. vertex-vertex correspondences). Depending on the approach chosen, this is done by reparaterization or by decomposing the meshes, according to the feature correspondence.

In case of mapping to a sphere, an embedding \( \phi \) with \( S = \{s_0, s_1, \ldots, s_n \} \subset \mathbb{R}^3 \) is computed. The embeddings on the sphere are aligned according to the feature correspondence using a bijection map \( f \) that maps spheres into spheres.

\[ \{v_1, v_2, \ldots, v_n\} \in K_0 \]

\[ \phi_0(B_0) \]

\[ \phi_0(B_0) \]

\[ \phi_0(B_0) \]

\[ \phi_0(B_0) \]

The main problems in this approach are to compute the vertex coordinates \( S_0, S_1 \) on the sphere and the reparaterization.

The decomposition approach is more general and more difficult. In addition to generate embeddings of the topological disks one has to decompose the meshes in an isomorphic way, taking possible feature correspondences into account. Formally, an abstract simplicial complex \( L \) consisting of a subset of the vertices in \( K_0 \), \( K_1 \) is used as coarse approximation of both meshes.

\[ \phi_0(L) \approx \phi_0(K_0) \]

\[ \phi_0(L) \approx \phi_0(K_1) \]

Typically, \( L \) is topological minor of \( K_0 \) as well as \( K_1 \), i.e.
it is a partition of the meshes. Vertices in $K_1$, $K_2$ are identi-
ified with a face in $L$, and all vertices belonging to a particular
face are embedded in its planar shape. Thus, the common
parameter domain is the topological realization $[L]$, where
each vertex is represented with a barycentric coordinate with
respect to a particular face in $L$. This requires to embed
pieces of the mesh in the plane.

$$\sum_{i} w_i a_i - \frac{1}{2} \sum_{i} w_i a_i = \frac{1}{4} \sum_{(i,j,k) \in \Delta^1} a_i a_j a_k$$

Following, techniques to embed simply-connected
bounded and unbounded meshes in the plane and on the
sphere are explained. Then, approaches to dissect the
meshes into isomorphic patch-networks (or, equivalently,
inducing base-domains $[L]$ on $M_n, M_r$) are discussed.
After these basic embedding steps parametrization
for feature alignment is introduced. Finally, some comments
on rarely mentioned details in the correspondence problem
are given.

3.1. Mapping topological disks

Similarly-connected parts of the boundary of three dimen-
sional spaces are homeomorphic to a disk and, therefore,
called topological disks. In order to find a parametrization
of such pieces we need a bivariate map of a bounded, simply
connected mesh to the plane.

In our application we need to find a bivariate map between
patches. Thus, it is necessary to constrain the boundary of
the patches to a particular shape. Here, we concentrate on
mapping an arbitrary bounded and simply connected mesh
to a unit disk so that boundary vertices of the mesh lie on the
unit circle. This limits the applicability of several parametriza-
tion approaches, which allow the boundary of a triangula-
ted surface to be non-convex or not to be fixed a priori to
achieve smoother or less distorted mappings.

In a first step the boundary vertices are fixed on the unit
circle. First, the three vertices from the base domain $L$
are fixed in an equiangular way. This is necessary to make sure
that adjacent faces in the base domain have a continuous
parametrization across base domain edges. The remaining
boundary vertices are fixed so that the arc lengths between
neighboring vertices are proportional to the original edge
lengths. The remaining (interior) vertices are free and their
position is determined by a relation to neighboring vertices.

Most of the published approaches to solve this task mini-
mize a quadratic error functional expressed as the vertex po-
sition relative to its neighbors. This boils down to solving
a system of linear equations. Non-linear approaches either use
higher order functionals to be minimized or are of al-
gorthmic nature (e.g. Gregory et al.18,20, which is discussed
after the linear techniques).

More specifically, let $\{v_i\}$ be the vertices to be mapped
to the disk so that the free interior vertices have indices $0 \leq
i < n$ and the fixed boundary vertices have indices $n \leq
i < N$. We aim at finding positions $w_i$ in the plane
with $|w_i| = 1$, $3 \leq i \leq N$. The mapping is biseptive if and only
if no edges cross or, alternatively.

$$\forall(i,j,k) \in \Delta^1 \det \begin{pmatrix} w_i & w_j & w_k \end{pmatrix} \geq 0 \quad (1)$$

However, this quadratic expression is awkward to use as a cri-
terion to guarantee that the planar embedding is valid, which
is why most approaches resort to the more restrictive but suf-
ficient linear conditions.

In the following we discuss three ways to define a lin-
ear system, whose solution yields positions for the vertices.
In addition, the Divide&Conquer approach of Gregory et al.18,20
is explained.

3.1.1. Barycentric mapping

Tutte19 has shown how to embed planar graphs in the plane
using a barycentric mapping. In our restricted setting, the
idea is simply to place every interior vertex at the centroid of
its neighbors:

$$w_i = \frac{1}{2} \sum_{j} w_j a_i$$

Setting $A = \{A_{ij}\}$ with

$$A_{ij} = \begin{cases} a_i a_j & (i,j) \in K \ 
0 & (i,j) \notin K \end{cases} \quad (3)$$

can be written as the mentioned system of linear equa-
tions

$$\begin{pmatrix} I - A \end{pmatrix} \begin{pmatrix} w_i \end{pmatrix} = \begin{pmatrix} \sum_{j} w_j \end{pmatrix} \begin{pmatrix} \sum_{j} a_i a_j \end{pmatrix}$$

$$w_i = \sum_{j} w_j a_i$$

This matrix $(I - A)$ has full rank and, thus, there is exactly
one solution. Tutte19 has shown that this solution is a valid
embedding of the mesh.

Note, that the shape of the mesh has no effect on the place-
ment of vertices in the plane. All information for the embed-
ding comes from $K$ and it is clear that the embedding cannot
reflect geometric properties contained in $U$ of the mesh.
In the following we try to incorporate information about
the original shape.

3.1.2. Shape preserving parametrization

In the barycentric mapping the weights $\lambda_i$ contain only topo-
logical information. Floater18 determines weights that reflect
the local shape of the mesh. More precisely, the $\lambda_i$ are so cho-
sen that the angles and lengths of edges around a vertex are
taken into account.

To compute the weights for a particular vertex $v_j$ this ver-
tex is placed in the origin and incident edges are laid out in
the plane using the original edge lengths and angles propor-
tional to the original angles. This is assumed to be the ideal
parametrization $w_j$ of the mesh with respect to $v_j$.

The weights are computed in a way that result in placing
$w_j$ in the origin if the neighbors $w_i$ were fixed and the
system of equations had to be solved. Thus, we have

$$w_j = \left(0,0, \cdots, 0, \sum_{j} \lambda_j w_j \right)$$

and

$$1 = \sum_{j} \lambda_j \quad (5)$$

If $v_j$ has only three neighbors this exactly determines the
positive weights, for more than three neighbors a positive
solution has to be chosen from the space of possibilities solu-
tions. Note that positivity results in convex combinations,
which are necessary to assure a valid embedding. Floater
proposes a method to compute reasonable weights, which
are guaranteed to be positive. Take the cyclically ordered set
of neighbors $j_{1,2,3,\ldots}$, where $j_1 \in N(j)$. Determine sets of
weights $\lambda_{j1}(k)$ with respect to three subsequent neighbors
$j_1, j_2, j_3$. This yields non-negative $\lambda_{i,j1}(k)$ for each $k$.
These weights are averaged to yield the final weights:

$$\lambda_{j1}(k) = \frac{1}{\sum_{k} \lambda_{j1}(k)} \sum_{k} \lambda_{j1}(k) \quad (6)$$

The positions $w_j$ are computed by solving (4). Recently,
Floater18,19 has proven a generalization of Tutte’s theorem,
which shows that it is sufficient that each vertex is a convex
combination of its neighbors.

3.1.3. Discrete harmonic mappings

Harmonic mappings are a concept found in several fields in
mathematics using differentials. Harmonic maps are often
described as the function $u$ among all functions mapping to
a given domain $\Omega$ that minimize the Dirichlet energy

$$\mathcal{E}_D(u) = \frac{1}{2} \int_{\Omega} |\nabla^2 u|^2 \quad (7)$$

Pinkall and Polthier43 show how to discretize this problem
for triangles, so that weights are derived per vertex and
neighbor leading to a system of linear equations of the form
of Eq. (4). A somewhat clearer derivation can be found in a
more recent work of Polthier44. There, it is shown that the
discrete Dirichlet energy is

$$\mathcal{E}_D(v) = \frac{1}{2} \sum_{(i,j,k) \in \Delta^1} \left(\cot \alpha_{ij} + \cot \beta_{ij}\right) |w_i|^2$$

and that the minimizer solves

$0 = \frac{1}{2} \sum_{(i,j,k) \in \Delta^1} \left(\cot \alpha_{ij} + \cot \beta_{ij}\right) (v_i - v_j)$

at each vertex $i$. This leads to weights

$$\lambda_{ij} = \left(\frac{1}{2} \sum_{k \in \Delta_i} \left(\cot \alpha_{ik} + \cot \beta_{ik}\right) \right) |v_i - v_j| \quad (8)$$

which are used to obtain an embedding by solving Eq. (4).
Another formulation, which is probably better known in the
graphs community, is given by Eck et al.11.

3.1.4. Area preserving Divide&Conquer mapping

Gregory et al.18,20 describe a recursive algorithm that aims
to preserving the area of triangles in the mapping. The idea
is to induce the mapping by recursively dividing the patch
into two pieces, which are then mapped independently.
These dissections are so chosen that the ratio of areas in the
original mesh and the embedding are approximately the same.

In particular, two diametrical vertices on the boundary of
a patch are chosen. A shortest path is computed using Di-
jikstra’s algorithm. This path is mapped to the segment con-
fined by the vertices and the triangulated surface. The
segment divides the patch into two halves, which are treated
in the same way, until all ver-
tices are mapped.

3.1.5. Comparison and Conclusion

We have embedded parts of a mesh using the four ap-
proaches presented above. Note that the solution of ma-
trix equation (4) could be performed using hierarchical meth-
ods, which is equivalent to using multigrid meth-
ods. However, the matrix has sparse structure and we have
found it sufficient to use iterative solvers exploiting the
sparnesses.

Some of the results of the comparison are shown in Fig-
ure 1. It is apparent that the general structure of larger
and smaller triangles is very similar in all embeddings
generated using linear optimization techniques. This suggests that con-
nnectivity is the major factor in these type of embeddings.
Changing the weights used to compute the embedding only
changes the local behavior of the embedding. They share the
problem of area compression. Inner triangles have much less
area than outer triangles. The area preserving scheme elimi-
nates this problem at the cost of distorted triangle shapes.

In fact, all these parametrizations might be unsuitable due
submitted to COMPUTER GRAPHICS Forum

This problem: The sphere is recentered after each relaxation is depicted in Figure 3 and resulting embeddings for several models are shown in Figure 4. 27 has reported an elegant and simple solution to A relaxation process for the polyhedral model of a horse.

Once the shape is simplified to a tetrahedron, vertices. In the planar case the fixed vertices avoid that all vertices collapse to one point, which is the trivial scheme that turns arbitrary genus 0 polyhedra into convex shapes. They first simplify the shape using vertex removal until the simplified shape is a tetrahedron. Only vertices with valence 3,4, and 5 are removed. Since the mesh is triangular shaped and if so to find a star point. For piecewise linear shapes (meshes) this can be done by intersecting halfspaces induced by the face elements of the mesh. The intersection of all halfspaces is called kernel. If the kernel is non-empty the mesh is star shaped and every j as easily from the interior kernel is a suitable star point. The kernel of a mesh in 3D can be computed in \( O(n \log n) \) using standard techniques. 3.2.1. Star shapes

Kent et al. 36 were the first to present techniques to map certain classes of genus 0 meshes to a sphere. A particularly simple class of objects are convex shapes. A convex shape has the property that a straight line connecting any two boundary points of the shape lies completely inside the model. Thus, all points are visible from any interior point of the shape and a projection through an interior point onto an enclosing sphere is necessarily bijective. A generalization of this idea extends the class of shapes to star shapes. Such shapes have at least one interior point so that straight lines connecting this interior point with boundary points lie completely inside the shape. Interior points with this property are called star points. Obviously, projecting the boundary points of a shape through a star point onto an enclosing sphere is a bijective mapping. Specifically, if point \( O \) is visible from all vertices of the mesh then translate all points so that \( O \) coincides with the origin. Then normalize all vertex coordinates. These vertex coordinates are the parameterization of the mesh vertices on a unit sphere. An illustration is given in Figure 2.

Figure 2: A polygonal star shape and its projection to a circle. The kernel of a star shape is the intersection of all open half spaces over the edges (faces in case of a polyhedron). Every point in the kernel induces a bijective mapping to the circle by projection.

The only problem is to determine whether a shape is star shaped and if so to find a star point. For piecewise linear shapes (meshes) this can be done by intersecting halfspaces induced by the face elements of the mesh. The intersection of all halfspaces is called kernel. If the kernel is non-empty the mesh is star shaped and every j as easily from the interior kernel is a suitable star point. The kernel of a mesh in 3D can be computed in \( O(n \log n) \) using standard techniques. 3.2.2. Simplification

Shapiro and Tal 39 seem to be the first to present a reliable scheme that turns arbitrary genus 0 polyhedra into convex shapes. They first simplify the shape using vertex removal until the simplified shape is a tetrahedron. Only vertices with valence 3, 4, and 5 are removed. Since the mesh is triangular such vertices always exist. It follows easily from the Euler-Poincare formulas that the average degree in any triangular (surface) mesh is less than 6. Thus, at least one vertex with degree strictly less than 6 has to exist.

Once the shape is simplified to a tetrahedron, vertices are reattached making sure that the shape stays convex. More specifically, it is shown how to attach vertices with degree 3, 4 and 5 to a convex shape so that the shape stays convex. More specifically, if a vertex \( v \) has to be added to a face \( f \), its position has to be outside the convex hull of the current mesh but inside the kernel of faces adjacent to \( f \).

3.2.3. Spring embedding

Alexa 1 introduced a variation of the methods presented for planar embeddings to embed polyhedra on the unit sphere. The basic idea is the same as in barycentric mappings: Place each vertex in the centroid of its neighbors. On the sphere, however, two conditions of the planar case are violated. First, convex combinations of the neighbors’ positions are not part of the domain (the sphere) and, second, no peripheral cycle is given to support the embedding.

The approach is to use a relaxation algorithm to compute the solution to the barycentric constraints. The starting configuration is generated by computing an interior point of the solid model represented by the mesh and then projecting all vertices to a sphere, which is centered at the interior point. The relaxation algorithm repeatedly places each vertex at the centroid of its neighbors. Since the centroid is not on the sphere the coordinate is normalized:

\[
\bar{w}_{i}^{t+1} = \frac{1}{n} \sum_{j \in N(i)} w_{i}^{t+1} - \bar{w}_{i}^{t+1}
\]

The small differences in local shape do not seem to have much influence on the resulting correspondence of the shapes. This is even more true when local features of the shapes are aligned by reparameterization (see Section 3.4).

A part of a mesh parameterized on the unit disk using different mapping techniques. The original geometry is highlighted in red. A barycentric mapping (see Section 3.1.1) does not reflect the geometry of the mesh. The shape preserving embedding tries to capture the local shape of the mesh by locally approximating conformal maps (see Section 3.1.2). Discretized harmonic embeddings minimize metric distortion (see Section 3.1.3). The area preserving embedding is a recursive process, which aims at approximating the original area of triangles (see Section 3.1.4).

Figure 1: A part of a mesh parameterized on the unit disk using different mapping techniques. The original geometry is highlighted in red. A barycentric mapping (see Section 3.1.1) does not reflect the geometry of the mesh. The shape preserving embedding tries to capture the local shape of the mesh by locally approximating conformal maps (see Section 3.1.2). Discretized harmonic embeddings minimize metric distortion (see Section 3.1.3). The area preserving embedding is a recursive process, which aims at approximating the original area of triangles (see Section 3.1.4).

A relaxation process for the polyhedral model of a horse.

Figure 3: Embedding a polyhedral object on a sphere using relaxation. Initially, the vertices are projected through an interior point of the model onto a unit sphere. The relaxation is finished when all faces are oriented correctly. Incorrectly oriented faces are surrounded by red edges.

Figure 4: Sphere embeddings of the models of a giraffe, a hammerhead shark, and a swordfish.
3.2.4. Comparison and Conclusion

None of the techniques discussed above makes a particular attempt to preserve the properties of the original mesh. Additional constraints (as discussed in Section 3.4) are necessary to make these embeddings useful. The central projection is obviously limited to a small class of objects. We find that the two techniques for general genus 0 meshes have some what complementary features/problems. The simplification approach is more robust (in terms of geometric computations and sensitivity to topological defects in the mesh) while the relaxation generates smooth embeddings. Both techniques suffer from the area compression problem mentioned earlier.

3.3. Isomorphic dissection

The more general approach to establish correspondence between meshes is to dissect them into pieces. Each piece is a topological disk and can be mapped to the plane using one of the techniques discussed in Section 3.1. Of course, the shapes have to be split in such a way that the graphs representing the dissections have equivalent topologies.

This approach is not limited to a particular topology of the shape, since the dissection results in a set of topological disks. However, the shapes need to be homeomorphic so that their dissections could be topologically equivalent. With extra conditions it is possible to deal also with topologically different shapes.

3.3.1. Automatic dissection of shapes

Ideally, the dissection process would not require the user to assist. However, the fully automatic dissection of two meshes into isomorphic structures seems to be a hard problem. The approach of Kanai et al. uses a single patch and, thus, automatically decomposes into isomorphic structures. However, the approach is limited to genus 0 meshes and suffers from the already mentioned area compression problems in the embedding.

Several techniques exist for the dissection of a single mesh. In the context of multi resolution models several approaches require the mesh to be broken into patches. This problem is known as mesh partitioning and naturally related to graph theory. Some algorithms try to balance the size of patches (e.g., Eck et al., Karypis & Kumar).

In many multi resolution methods, however, the base domain (the structure of large patches) is found by simplifying the mesh using vertex removal or edge collapse.

These techniques might help in deriving a single base domain for two meshes. Lee et al. use two independently established base domains to generate one base domain for both meshes. They employ their MAPS scheme to build independent parameterizations over different base domains. These base domains are merged (see Section 4) so that the resulting merged base domain contains both independent base domains as subgraphs. Note, that in general the correspondence problem had to be solved for the geometry of the base domains. Lee et al. assume that the geometry of the base domains is so similar that this problem could be solved with simple heuristics (e.g. projecting in normal direction).

3.3.2. User specification of isomorphic dissections

The underlying idea of all works in this section is that the user specifies the topology/connectivity of the base domain and the location of the base domain vertices on the original meshes. Tracing the edges of the base domain on the mesh is more or less done automatically.

DeCarlo and Gallier do not assist the user specifying the edges. While this way of defining the dissection gives a lot of freedom to the user it is very time consuming.

Gregory et al. assist the user in defining the edges (see Figure 5). The base domain is developed while intersecting the surfaces. The user defines a pair of vertices on a mesh and the system finds a shortest path of mesh vertices connecting the defined vertices. Subsequently, feature vertices can be connected to existing feature vertices using shortest paths along the mesh. By picking corresponding vertices in the input meshes the system will construct the same graph in the input meshes. A problem could arise from the fact that only mesh vertices are used to find shortest path.

However, even using the exact shortest path can lead to problems. Praun et al. illustrate the problem and propose better solutions: If a shortest path would cross an already established edge of the base domain, the shortest possible connection avoiding the intersection is computed using a wavefront algorithm. However, also the order of vertices being connected is important, because several edges might enclose an unconnected vertex. This problem can be avoided by traversing the vertices along a spanning tree.

In our view, the underlying problem is that on non-convex domains the dissection results in a set of topological disks. However, the shapes need to be homeomorphic so that their dissections could be topologically equivalent. With extra conditions it is possible to deal also with topologically different shapes.

3.3.3. Comparison and Conclusion

The works of Bao and Peng and Zickler et al. are similar in spirit. However, it seems that they allow to use more points to define the boundary of a patch. Points are connected with the shortest paths in the vertex-edge graph as in the work of Gregory et al. In the approach of Kanai et al. the user first defines a set of corresponding feature vertices. Aware of the problems resulting from using a shortest path consisting of mesh vertices the authors compute the shortest path on the piecewise linear surface connecting the feature vertices. This path may or may not coincide with vertices and edges. Figure 6 shows the resulting dissection for two cars. Since computing exact shortest path on polyhedral surfaces is difficult and time consuming they employ an approximate method that refines the original mesh and uses Djikstra’s algorithm.

A difficult task is to identify common features in several shapes. It seems impossible to automatically find such common features as they are mostly defined in a semantic and not necessarily in a geometric way. The user can identify these features and provide information about their location and correspondence (for instance as vertex-vertex correspondence of a few vertices). The algorithm should exploit this information as much as possible.

All dissection type methods explained above offer this way of user-control. Since the user explicitly chooses corresponding patches (and, therefore, corresponding edges and vertices) they can specify which parts of the meshes correspond. However, the user is also involved in other tasks, which can make the process complicated and lengthy.

The shapes’ geometry also contain information useful to exploit. Several functions over the parameter domain of the meshes seem to be worth looking at. It is important that these functions are independent of the parameterization, i.e. they are intrinsic to the shape and do not change if the description of the shape is changed. Such functions are especially considered in differential geometry, which could be seen as exploring a shape on the shape, i.e. without a distant view. The most prominent assets for describing shapes in differential geometry are:

- normals, which are independent of translation and scaling but sensitive to rotation and
- curvature (principal curvatures, mean or gaussian curvature), which is independent of translation and rotation but sensitive to scaling.

The parameterization of the shape’s boundary allows to represent these quantities as a function in two variables, i.e. the normal n: $\mathbb{R}^2 \rightarrow S^2$ or the gaussian curvature $c: \mathbb{R}^2 \rightarrow \mathbb{R}$.

It is clear that this information about the meshes does not lead to point to point correspondences such as user selected features. Instead the quality of the match of two shapes is quantified as a function of the distance of the shape descriptors. For example, Surazhsky and Elber use the integral over the inner products of normals:

$$R = \int (n_1 \cdot n_2) dA$$  \hspace{1cm} (14)$$

Here, the inner product between normals and the integration over the surface represent particular choices. One might
choose another metric for the difference of normals as well as another method to take into account the set of differences (e.g. the maximum of the angles between normals). In order to match shapes based on such criteria the parametrization is changed so that the functional is minimized. Note that no point has an a priori optimum placement making this problem much harder to solve than aligning specified point to point correspondences.

3.4.2. Transforming to align features

As a first step in an alignment procedure the parameter domains should be transformed using affine transforms to roughly align the features. Note that this is not possible for parameterizations resulting from dissection as the orientation of each patch is determined by neighboring patches.

Alexa aligns a set of point to point correspondences by rotating the spherical embeddings of the mesh. The objective function to be minimized is the squared distance of corresponding points. The minimization problem can be solved using the techniques explained in Section 5.2.

3.4.3. Warping parameterizations to align features

In general, one could generate any parameterization of the meshes as a first step to establish correspondence. After this, the parameterization domain can be used to align user selected features or automatically generated features in terms of a re-parametrization of one or more of the initial parameterizations.

Alexa and Zöckler et al.

explicitly allow the user to select a set of point to point correspondences. Warping techniques similar to those used in image morphing (e.g. see the overview works of Ruprecht and Wolber) are used to deform the parameterization so that corresponding points coincide. Whether the parameter domain is a disk or a sphere does not make a difference for the general approach.

In contrast to image warping, it is absolutely necessary that the warp does not introduce incorrectly oriented faces. This would be less of a problem if vertices as well as edges were warped. But since the algorithm later might require edge-edge intersection tests, warping the edges is impractical. Instead, edges should be (still) defined as the shortest path between vertices. That is, we warp the vertices only. Thus, even injective warping functions might introduce foldovers.

Two solutions have been proposed in the context of morphing: Alexa warps only as much as is possible with the given triangulation. If the mapping starts to introduce foldovers in the triangulation the warp is made more local by adjusting the radius of influence. However, the features are not guaranteed to coincide after this process.

Zöckler et al. use the foldover free warping scheme of Fukumura and Makovoz. They also warp in small steps. However, if foldover occurs they change the mesh connectivity to assure that the embedding stays valid. In particular, they use edge flips for this task. This changes the original triangulation of the meshes.

Recent work in texture mapping allows to incorporate point constraints. These techniques could be applied for the problem here.

Lévy formulates the problem of satisfying given point constraints by incorporating the squared error of the point correspondences into the energy functional used to generate the parameterization. Using a scalar to weigh the importance of the point correspondence allows to trade between the regularization term for the smoothness of the parameterization and the accuracy of satisfying the constraints. On the other hand this mixed energy functional does not guarantee a valid embedding. A possible way would be to start with a valid embedding and then increasing importance of the constraints as long as the embedding stays valid.

Eckstein et al. propose a scheme that allows to exactly satisfy constraints whenever possible. It might be necessary to introduce additional vertices in the triangulation for this. The triangulation is first simplified so that it contains only the constrained vertices. These are placed accordingly and the mesh is then refined again. During the refinement process it might be necessary to insert additional vertices because straight edges connecting vertices could intersect.

3.5. Conclusions

The ideal algorithm for finding a parameterization of a mesh has not been found. In general, coarse simplifications of the original meshes are accepted as useful parameter domains. In the context of morphing they are not ideal for two reasons:

- For seemingly different shapes a common base domain might be hard to find and the decomposition of the original mesh forces the user to interact.
- The alignment of features (e.g. shape features) is restricted to corresponding patches of the base domain.

In view of these limitations the simple solution to embed arbitrary input, takes any number of user-constraints into account optimizes a reasonable resemblance of the shapes, and is sufficiently fast.

4.1. Mapping parameter values to the surface

After the meshes have been parameterized it is easy to map the position of a particular vertex on the surface of a mesh. Assume we want to find the position of vertex $v_k$ of the first mesh on the second mesh determining the vertices $\{w_{jk}\}$ comprising the face in the parameterization in which the parameter domain position $w_{jk}$ lies. Then, $w_{jk}$ is represented in barycentric coordinates with respect to $\{w_{ij}\}$:

$$w_{jk} = \sum_i \lambda_{ij} w_{ij}$$

(16)

The position of $v_k$ in the other mesh is found as

$$w_{jk} = \sum_i \lambda_{ij} w_{ij}$$

(17)

This is the exact position on the piecewise linear shape and the way used in most of the morphing literature.

However, this does not take into account the idea that piecewise linear shapes are (in most cases) just approximations of smooth shapes. Practical problems occur when normals have to be redefined from these new geometric positions. Vertices inside a face get the face's normal. If standard rendering methods are used (vertex normals and Gouraud shading) this results in deteriorated shading.

It would be advantageous to find positions which result in
a smooth surface. More specifically, we would like to use the barycentric coordinates to find positions over a triangular face and not necessarily on the face. This calls for methods defining a smooth surface from a coarse mesh. An obvious choice for such a method would be subdivision (e.g., Loop subdivision26 or Kobbelt’s S+ scheme27).

4.2. Map overlap data structure

We need a data structure to store the meshes, which allows to add and remove edges, gives quick access to topological in-formation (e.g., the ordering of edges around a vertex), and is not to heavy in terms of storage. We choose the doubly con-

ected edge list28 (sometimes called twin-edge data struc-
ture) as the basic data type of this data structure is the edg-

es are stored as two directed half edges. More specifi-
cally, the following information is stored:

Face

The face record contains a pointer to an arbitrary half edge on its boundary.

Edge

Each edge record contains pointers to:

• its originating vertex,
• the face it bounds,
• the half edge connecting the same vertices but in the opposite direction (its twin),
• the next half edge along the boundary of the bounded face.

Vertex

The vertex record contains a pointer to an arbitrary half edge originating from this vertex as well as location in space and other attributes (e.g., normal, color, texture coordinate).

Figure 8 illustrates the data structure. Note that it is particular easy to iterate along the boundaries of faces (next pointers) or through all edges incident upon a vertex in their circular order (twin → next). A good description of the doubly con-

ected edge list can be found in Berg et al.

4.3. Open meshes embedded in a disk

Several algorithms were proposed for the problem of over-

laying planar graphs - see a textbook14. In general, the planar map overlap has the complexity O(n log n + kn), where n is the number of edges and k is the number of intersections. If

two subdivisions are connected (as in our case) the planar overlay can be computed in O(n log n + kn).

The general paradigm for planar overlay is plane sweep. Sweep algorithms process the input with a virtual line moving along its normal direction. Whenever a vertex intersects the sweep line the corresponding edge is added (the vertex is the starting point of this edge) or removed (the vertex is the end point) from the list of active edges. The list of active edges is tested for intersection with added edges. To further reduce the number of necessary intersection tests the active edges are stored in their order along the sweep line. This is done by inserting edges in the correct position. In addition, the order has to be updated at intersection points. Using the ordering, only neighboring edges have to be tested for inter-

section. This processing leads to an algorithm with complicity O(n log n + k + k log n). By exploiting that two connected graphs are intersected the complexity can be reduced to O(n + k).

In the case that meshes are embedded on the disk the special care has to be taken for the boundaries of the meshes. While we assume that the embedding is surjective (i.e. fills the disk), the boundary in fact is a polygon leaving small empty regions between the disk and the polygon. However, it is clear that the boundaries of the meshes to overlay should be mapped onto each other. So in order to avoid that the bound-

ary polygons intersect with inner edges of the other mesh the boundaries have to be merged first. This is done by simply connecting the vertices of all meshes on the disk along the linear order given by the disks boundary. After this bound-

ary polygon has been established the planar mesh overlap procedure can be computed.

4.4. Closed meshes in arbitrary position

There seem to be only a few publications about the overlay of meshes in general position (i.e. the triangulated faces are close to each other but n. e. planar). Note that plane sweep solutions are not applicable in this case. For pub-

lications dealing with overlaying two subdivision of the sphere, Kent et al.28 give an algorithm for the sphere overlay prob-

lem, which needs O(n log n + kn) time. Alexa has presented a solution to this particular problem, which reports the in-

tersection of two spherical subdivisions in the optimum time of O(n + k). Also, both algorithms exploit the topologi-

cal properties of both subdivisions, which are used to guaran-
tee the correct order of intersections. Here, we generalize these algorithms to work on two arbitrary shaped meshes, which are assumed to be sufficiently close to each other. We also al-

leviate the problem that the published versions had a worse case complexity of O(n + k log n) for the construction of the merged mesh using the already reported intersections.

The algorithm consists of two main parts: First, finding all intersections, and second, constructing a representation for the merged model.

The algorithm consists of two main parts: First, finding all intersections, and second, constructing a representation for the merged model.

4.4.2. Generating the data structures

An appropriate data structure for storing the intersections is needed. Information about an intersection should be accessi-

ble from both intersecting edges at constant costs. We use a hashtable with edge indices as key values. When edge-edge intersections are found and stored in the intersection lists a pointer to the entry in the hashtable is stored. This means, both edges point to the same data structure containing in-
formation about the intersection (the intersecting edges in the beginning). The hashtable is only needed to access the entry when the intersections have already been computed by processing K1 and need to be found when intersections from K2 are generated. After reporting all intersections the hashtable is discarded.

The following two step algorithm constructs the merged mesh. First, edges in K2 are cutted. We iterate through the intersection list of an edge and cut the edge at each intersect-
tion point. Thus, a new edge (two half edges) are generated for each intersection. The new edge represents the part of the edge that has to be processed. At each intersection the data structure containing the respective information is updated to now contain the two parts of the edge incident upon the in-

tersection point. At this point only the twin pointers of the half edges are updated. The next pointers are left empty.

Second, edges in K1 are processed. As in the first step edges are cut into two pieces at each intersection point. How-

ever, this time also the next pointers are updated. This is done by using the information stored in the intersection data struc-
ture, which now contains both edges of the already cut edge in K1.

After all intersections are processed in this way we have a valid vertex and edge lists of the embedding. It remains to compute the records for the faces. Note that faces created from intersecting triangles are convex polygons with 3 to 6 sides, which should be triangulated. This is another subtlety, which is more involved as it may seem: While the polygon resulting from the intersection is convex it is not clear what shape it has in other geometric configurations, e.g. those of the source meshes. In principle one should find a triangula-
tion that is admissible in all source geometries. This might be difficult and could lead to the need for additional vertices. The problem is known as compatible triangulation and dis-
cussed in detail in another context in Section 5.5.1.

Note that this approach could be extended to bounded meshes, however, boundaries require special treatment be-
yond the scope of this report.

4.5. Remeshing

A mesh is typically just an approximation of a shape. We have already seen that the mesh overlay process together with using coordinates lying exactly on the mesh might in-
roduce artifacts into the source meshes (see Section 4.1). Thus, even if the original mesh connectivities are available as subsets of K the reproduction of the original shapes though exact is not ideal. It seems that the perfect recon-

struction of the source shapes is impossible and we could as
the features of several meshes, which do not necessarily coincide. This, again, incurs extra burden on the user, because a more complex base domain has to be induced on the input meshes. In addition, a more complex base domain limits the possibilities of automatic feature alignment methods. However, the flexible and lean representation mesh seems worth it.

5. Vertex paths

After the computation of one mesh connectivity \( K \) and two mesh geometries represented by vertex coordinates \( V(0) \) and \( V(1) \) it remains to compute vertex coordinates for the blended shapes. For a typical morphing animation, a set of vertex coordinates \( V(t), t \in [0,1] \) has to be generated.

A simple choice is linear interpolation\(^{35,36,37}\). A rigid\(^{38} \) or affine\(^{39} \) transform prior to linear vertex interpolation yields better results. More complex behavior during the transform calls for more elaborate methods. Such methods decompose the shape in to linear pieces and treat these pieces separately.

5.1. Linear interpolation of vertices

The easiest way to produce blends of corresponding shapes is to interpolate the coordinates of vertices. Given a transition parameter \( t \), the coordinates of an interpolated shape are computed by

\[
V(t) = (1-t) V(0) + t V(1)
\]

This type of interpolation produces good results if the shapes have the same orientation and are somewhat similar. Figures 11 and 12 show morph sequences obtained by linear interpolation.

Different orientation could lead to displeasing results. Imagine two squares that are rotated by 180 degrees against each other. If simple vertex interpolation is applied in this configuration, the interpolated squares will shrink until the shape is collapsed to one point and then grow again. This is not the desired result in most applications. It is advisable to interpolate the orientation separately from the vertex coordinates.

5.2. Interpolation of Orientation

Several ways exist to compute a relative orientation of two shapes. Note that the orientation part of the merged base domain has to represent

distances of corresponding vertices using the corresponding transform. The minimization problem of finding an affine transform can be solved using the pseudo inverse of the coordinate vector. Let the vertex vectors be arranged as a \( n \times 3 \) matrix

\[
V = \begin{bmatrix}
    v_{1x} & v_{1y} & v_{1z} \\
    v_{2x} & v_{2y} & v_{2z} \\
    \vdots & \vdots & \vdots
\end{bmatrix}.
\]

Then the squared distance of coordinates under an affine transform \( A \) is

\[
(V(0)A - V(1))^2
\]

and has to be minimized. This leads to linear system of equations, which can be solved using pseudo inverse \( V(0)^+ \):

\[
A = V(0)^+(V(0)^+V(0)^+)^{-1} V(0)^+V(1)
\]

Alternatively, the least squares solution (or, the pseudo inverse) could be computed using the SVD, which allows explicit control over the sensitivity to near rank deficiencies\(^{40} \).

Intermediate shapes \( V(t) = \{ v_{1}(t), v_{2}(t), \ldots \} \) are described as \( V(t) = A(t)V(0) \). The question is how to define \( A(t) \) reasonably? The simplest solution would be: \( A(1) = (1-t) I + t A \). However, some properties of \( A(t) \) seem to be desirable, calling for a more elaborate approach:

- The transformation should be symmetric.
- The rotational angle(s) and scale should change monotonically.
- The transform should not reflect.
- The resulting paths should be simple.

The basic idea is to factor \( A \) into rotations (orthogonal matrices) and scale–shear parts with positive scaling components. Alexea et al.\(^{41} \) have examined several decompositions. Through experimentation, they have found a decomposition into a single rotation and a symmetric matrix (i.e. the polar decomposition), to yield the visually-best transformations. This result is supported by Shoemake\(^{42} \) for mathematical, as well as psychological, reasons. The decomposition can be
5.3. Interpolation of intrinsic boundary representation

Linear interpolation of vertices can lead to undesirable effects such as shortening of parts of the boundary during the transition. To avoid such problems, Sederberg et al. suggest that a proper morph cannot be expressed merely as a boundary interpolation, but as a smooth blend of the interior of the objects. To achieve such an interior interpolation, they represent the interior of the 2D shapes by compatible skeletons and apply the blend to the parametric description of the skeletons. An extension of this approach to meshes - though theoretically possible - has not been presented so far. The extension of this idea to 3D has been investigated by Blending et al.

The approach of Gotsman et al. is to decompose the interior of the shapes into one rotation and a symmetric matrix and using

\[ A = \Lambda R \Phi = \Lambda R (R_1 R_2 \Phi ) R_2 = (R_1 R_2 \Phi ) R_2 R_1 = S \]

where \( A \) is the linearly interpolating the free parameters in the factorizations in (21), i.e.

\[ A_i,1 = R_{i,1} (1 - J + J) \]

Figure 13 illustrates the resulting transformations for a triangle. For comparison, 13(a) shows linear interpolation of the vertices coordinates. The transformation resulting from a singular value decomposition and linear interpolation \( A_i,1 \) is depicted in 13(b). Note that the result is symmetric and linear in the rotation angle but still unsatisfactory, since a rotation of more than \( \pi \) is unnecessary. However, if we substract \( 2\pi \) from one of the angles (depicted in 13(c)) the result is even more displeasing. We have found that decomposing \( A \) into one rotation and a symmetric matrix and using \( A_i,1 \) yields the best results (Figure 13(d)). It avoids unnecessary rotation or shear compared to the SVD and is usually more symmetric than a QR decomposition-based approach.

5.4. Smoothing paths among neighboring vertices

If every vertex is interpolated independently the paths of neighboring vertices are independent. However, one might expect a coherence among those paths. Ohbuchi et al. construct a subdivision surface connecting the two boundary representations. This is comparable to the variational approach of Turk and O'Brien for implicit shapes. Intermediate shapes are defined as cross sections of the subdivision surface. Since the subdivision surface is smooth in all directions not only the paths of single vertices are smooth but they are also smooth along local neighborhoods. By introducing trans-finite constraints on the subdivision surface it is possible to preserve sharp features of the original meshes.

5.5. Interpolation of the interior of shapes

Shapira and Rappopeter suggest that a proper morph cannot be expressed merely as a boundary interpolation, but as a smooth blend of the interior of the objects. To achieve such an interior interpolation, they represent the interior of the 2D shapes by compatible skeletons and apply the blend to the parametric description of the skeletons. An extension of this approach to meshes - though theoretically possible - has not been presented so far. The extension of this idea to 3D has been investigated by Blending et al.

Another way to represent the interior of the shapes is to decompose the shape into linear pieces, or more specifically, into simplices. The works of Floater, Goldan, and Surazhsky and Alexa et al. use this type of decompositions mainly for polygons, however, the extension to meshes has been demonstrated. The main difficulty in extending these approaches lies in the reliable computation of isomorphic dissections of meshes into simplicial complexes.

We will first discuss ways of generating such isomorphic complexes and then explain the possibilities they open for computing vertex paths.

5.5.1. Isomorphic Simplicial Complexes of Shapes

Simplicial complexes allow the local deformation of the shapes to be analyzed and controlled. Here, we explain how to construct isomorphic dissections given two shapes with identical boundary connectivity.

The problem was first discussed by Aronov et al. for polygons. They offer two general approaches: The first approach is to translate the polygons independently and then use a piecewise linear bijective map to compute a planar overlay of the triangulations. This is somewhat similar to the planar embeddings explained in Section 3.1 together with the overlay procedures in Section 4.3. The second approach of Aronov et al. is a universal triangulation that fits every isolated polygon. This approach is extended by Gottschal and Surazhsky to generate triangulations with few interior vertices. However, it is unclear how to extend the general triangulation to meshes because of the more complex boundary connectivity. For this reason, we concentrate on the first approach.

It seems that the mesh-version of compatible triangulations has not been discussed in the literature. However, the procedure is conceptually the same. The meshes are tetrahedralized independently using common techniques. Then, a piecewise linear bijective map is computed between the shapes, typically using a common parameter domain. This parameter domain is used to compute an overlay of the simplicial complexes.

In case the common parameter domain for the meshes is a sphere, the interior of the sphere could be used as the parameter domain for the tetrahedra. If a piecewise linear parameter domain is used it seems more difficult to find a mapping of the interiors. If the parameter domain is given and then induced onto the meshes one could as well prescribe a simplicial base domain and induce this simplicial complex onto the (independent) simplicial complexes of the original meshes. The resulting structures are merged using a plane sweep algorithm similar to the line sweep algorithms discussed in Section 4.3.

The resulting simplicial complex might contain many, ill-shaped simplices, which cause the following determination of the vertex paths to deteriorate. For that reason, both, Gottschal and Surazhky as well as Alexa et al. try to improve the simplicial complex while preserving the isomorphisms. Gottschal and Surazhky try to minimize the number of resulting simplices. Alexa et al. employ an approach motivated by meshing techniques, however, adapted to the situation that one connectivity has to work for two shapes.

It seems advantageous to start with Delaunay triangulations because they avoid the generation of skinny simplices, which would be inadmissible in the merged complex, and because the same connectivity is produced for similar regions, which reduces the number of extra simplices generated by the overlay. The following smoothing strategy tries to maximize the minimum angle (as the Delaunay triangulation does) by independently moving interior vertices and concurrently flipping edges. This procedure is called compatible mesh smoothing. If the result needs to be improved further, the vertex count is increased by means of splitting edges. The split operation is well-defined in terms of topology, if it is applied to both triangulations simultaneously, the isomorphism remains. The idea is to split long edges to avoid long skinny triangles. Figure 15 shows a result achieved with this approach.

5.5.2. Morphing barycentric coordinates

The approach of Gottschal et al. does not require the interior of the shapes to be decomposed but also the exterior. The exterior is bounded by a common fixed convex shape. This fixed convex shape allows to represent each interior vertex with barycentric coordinates:

\[ v_i = \sum_{j \in N_i} \alpha_i, j v_j \]

Thus, each shape is fully described by the matrix of weights \( \Lambda = (\alpha_i, j) \). The idea of the approach is to linearly interpolate these barycentric coordinates, i.e.

\[ v_i, t = \sum_{j \in N_i} \lambda_i, j v_j, t \]
to be linear in the space of barycentric coordinates. Since the linear paths of vertices has some desirable features (e.g. it is short) one could modify the original barycentric coordinates $A$ towards the original vertex representation (represented by $I$) and, thus, the linear morph. Surazhsky and Gotsman show that $A^m$ gets closer to the linear morph with increasing $m$ and that if the linear morph is invalid there is a $m^*$ dev-iding the family $\lambda^m$ into sets of valid ($m \leq m^*$) and invalid ($m > m^*$) morphs.

5.6. Composing local ideal transforms

The general idea of Alexa et al.3 is to find a transformation which is locally as similar as possible to the optimal transfor-mation between each pair of corresponding simplices. The optimal simplex transform is found by factoring the affine transform defined by the pair of corresponding simplices as explained in Section 5.2. This defines ideal trajectories for each simplex.

Now consider the simplicial complex rather than a single triangle. We define an error functional for a candidate vertex configuration $V(t) = (V_0, V_1, \ldots)$

$$E(V(t)) = \sum_i (\tilde{A}_i(t) - B_i(t))^2,$$

where $A_i(t)$ is the desired ideal transform computed for simplex $i$, $B_i(t)$ is the affine transform induced from $V(0)$ to $V(t)$, and $\parallel \cdot \parallel$ is the Frobenius norm.

We define an intermediate shape $V(t)$ as the vertex configura-tion which minimizes this error between the desired coordinates for each individual simplex and the space of admissible coordinates.

Note that the coefficients of $B_i(t)$ are in $V(t)$ and that the $A_i(t)$ are known for a fixed time $t$. Thus, $E(V(t))$ is a positive quadratic form in the elements of $V(t)$.

The functional $E(V(t))$ can be expressed in matrix form as

$$E(V(t)) = \sqrt{v}^T G^T G H \sqrt{v},$$

where $\sqrt{v} \in R$ represents the constant, $G \in R^{2n \times 1}$ is the linear, and $H \in R^{2n \times 2}$ is the mixed and pure quadratic coefficients of the quadratic form $E(V(t))$. The minimization problem is solved by setting the gradient $\nabla E(V(t))$ over the free variables to zero.

The above definition has the following notable properties:

- For a given $t$, the solution is unique.
- The solution requires only one matrix inversion for a spe-cific source and target shape. Every intermediate shape is found by multiplying the inverted matrix by a vector.
- The vertex path is infinitely smooth, starts exactly in the source shape, and ends exactly in the target shape. These are properties typically difficult to achieve in physically-based simulations.

5.7. Non-uniform interpolation

So far we have always morphed the whole mesh, i.e. the tran-sition has been defined by a scalar transition parameter $t$. Now, we want to locally morph certain features or regions of interest, i.e. the transition parameters are different for differ-ent vertices. We will call the set of transition parameters for vertices the transition state.

An idea to apply non-uniform morphs is to explicitly define the path of several vertices. Gregory et al.36 allow the user to define the paths of base mesh vertices, which are used to modify the linear interpolation of the remaining ver-tices (see Figure 17). If a multiresolution hierarchy is used paths could be defined on each of the levels and propagated to higher levels. This is a concept introduced in multiresolu-tion mesh modeling19,24,35, which has been used for morphing by Michikawa et al.38.
shape representations other than meshes, e.g. parametric or implicit functions.

The backward transformation (from relative to absolute coordinates) is, by construction, not unique. It should be uniquely determined up to a translation. This means, \( \mathbf{Z} \in \mathbb{R}^{n \times n-2} \) should have rank \( n - 1 \), which is indeed \( 2 \).

The main idea of this approach is to morph by linearly interpolating Laplacian coordinates rather than absolute coordinates. This is somewhat similar to interpolating barycentric coordinates (see Section 5.4.2), however, here coordinates cannot be convex sums of neighbors as neighbor rings

Not all techniques presented in this framework are equally suited to be extended to more meshes. The correspondence problem discussed in Section 3 seems to be relatively easy to extend. All meshes are embedded in the given parameter domain, which leads to barycentric representation of the original vertices. If each set of original vertices \( V_j \) needs to be mapped to all other meshes \( V_{i \neq j} \), the complexity would grow quadratically with the number of meshes. However, this is not necessary if a remeshing strategy is used to generate a consistent mesh connectivity (see Section 4.5). This procedure generates the same set of vertices over all shapes, thus, the complexity is linear in the number of meshes times the number of vertices used in the remesh, which is the best we can expect. Concluding, the best way to generate the set \( \{ (V_j, K_j) \} \) is to embed all meshes in a common parameter domain (spherical or piecewise linear) and then remesh to the desired accuracy. This has been demonstrated by Michikawa et al.\(^{39}\) (see Figure 20).

The vertex path problem now extends to compute combinations of several vertex vectors. Linear vertex combination is easily extended

\[
V(s) = \sum_i s_i V_i
\]

Surprisingly, any technique involving rotations such as the ones explained in Sections 5.2 and 5.4 seem to be difficult to extend. Instead of interpolating the orientation one could compute the principal components (moments) of the shapes and align them with the canonical axes of the coordinate system. To extend the local morph approach explained in Section 5.5 the linear combination has to be applied to the Laplacian coordinates.

Applications of such spaces of meshes range from modeling and analysis of shapes to animation. Praun et al. have termed the synthesis-analysis part digital geometry processing (DGIP)\(^1\). Modeling could be achieved by combining several shape (features) to yield the desired result. Using techniques such as the principal component analysis, spectral properties of the mesh family can be explored.

The space of meshes \( (V(V_i), K_i) \) allows to represent animation as a curve \( s(t) \). A classical key frame animation with \( k \) key frames could be simply models as a \( k \)-dimensional space, where the curve linearly interpolates subsequent key frames. Alexa and Müller\(^{32}\) use the PCA together with rigid motion detection to find a more compact space (see an illustration in Figure 21). In this space the main part of the animation is stored in the rigid motion of the first base vector. The additional base vectors are sorted according to their energy in the spectrum of the animation. This allows to progressively store and stream mesh animations, where the progressiveness is with respect to movements and not model fidelity. The understanding of certain features of the animation as bases of a linear space gives this representation semantics. It is possible to identify e.g. the smile in a facial animation with a particular basis and, thus, to modify only the smile without the need to work on all key frames.

7. Conclusions

Mesh morphing has reached a state where basic problems are solved, yet, a practical working system is not available. The correspondence and representation problems can be seen as the application of several techniques now common in multiresolution representations and modeling. The vertex path problem is specific to morphing applications and leaves room for improvement.

What is still missing is a robust implementation of current techniques. As with other geometric techniques, many of the approaches suffer from numerical problems. Many "detail" problems such as normal and texture coordinate interpolation can cause trouble in practice.

With the extension of mesh morphing to linear spaces of meshes several interesting avenues for future research arise. However, one might ask the question whether meshes (with fixed connectivity) are the right representation for deforming shapes, anyway.

Additional material

The web site http://www-sfb.kfa.de/Kanai/GreenMorph/ maintained by Takashi Kanai has links to many other project web pages on mesh morphing and is a great source of information and material.

Acknowledgements

Encouraging and insightful discussions with many knowledgeable people, namely Daniel Cohen-Or, Jose Encarnacao, Craig Gotsman, David Levin, Wolfgang Müller, are gratefully acknowledged. Many thanks to Arthur Lee, Ming Lu, Takashi Kanai, Peter Schröder for allowing me to re-use their original illustrations and morph sequences. Thanks to the anonymous referees for their helpful and detailed reports. This work contains (rephrased) parts of the author’s prior works.

References

Figure 21: The SVD applied to a space of meshes. Here the original space represents a key frame animation, the result allows to represent the same animation with less geometries containing most of the original animation's energy.


