Merging Polyhedral Shapes with Scattered Features

Marc Alexa

Darmstadt University of Technology

Abstract

The paper presents a technique for merging two genus 0 polyhedra. Merging establishes correspondences between vertices of the models as a first step in a 3D morphing process. The technique allows for the specification of scattered features to be aligned. This is accomplished by performing the following three steps: First, initial embeddings of the polyhedra on unit spheres are computed. Second, the embeddings are deformed such that user defined features (vertices) coincide on the spheres. Third, an overlay of the subdivisions is computed and the aligned vertices are fused in the merged model.

Keywords. *Polyhedra, Scattered Features, Morphing*

1. Introduction

It is a property of nature that objects change their shape. Metamorphosis of three-dimensional objects is useful in several applications of computer graphics such as animation and modeling.

Right from the first approaches to shape metamorphosis it turned out to be difficult to handle three-dimensional objects. For the purpose of animation the task could be reduced to image morphing and a number of solutions to this problem are known. But while the metamorphosis problem seems to be solved for images it is still open for 3d-models.

Among the large number of representations for 3d-shapes, facet-based representations are popular and wide-spread. For that reason, a technique to perform transformations between facet-based representations of shapes, which also maintains this representation for the intermediate steps, is desirable.

Such a technique has to be controllable in a way that the user can provide information about what the morph should look like. More specifically, in many situations the models share common features that should be preserved during the transformation. 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 and provide an adequate transformation.

The author is with the Interactive Graphics Systems Group (GRIS), Department of Computer Science, Darmstadt University of Technology, Rundeturmstr. 6, 64283 Darmstadt, Germany
email: alexa@gris.informatik.tu-darmstadt.de
phone: +49 6151 155 674, fax: +49 6151 155 669

1.1. Related Work

One common way to generate transforms between polyhedral models is to establish correspondence between the vertices of the models. In 2D, this problem was solved by Sederberg et al. [17]. Later, Shapira and Rappoport have used the star-skeleton representation to transform between polygons [18].

The problem turned out to be more challenging in three dimensions. First attempts ([5],[10], and [11]) had strong limitations on the class of models that could be processed or the results of the transformation. Kent et al. gave the first correspondence algorithm for a wide class of genus 0 polyhedra in [13] (an earlier version [12] treated only star shaped polyhedra). The basic idea is to project the polyhedra onto a sphere and to compute the overlay of these projections. The resulting model contains the topology of both models and can easily be deformed into the shape of the source models. Any interpolation scheme yields a transformation. To date the problem of mapping an arbitrary genus 0 polyhedron to sphere remains challenging. Carmel and Cohen-Or use curve evolution for this task [3], but admit some implementation problems in the three-dimensional case.

Feature alignment in transformation of polyhedra has been considered by Lazarus and Verroust [14], who mainly align major axes. Carmel and Cohen-Or use a two part transform, an affine transformation and a deformation inspired from image warping, to align features [3]. DeCarlo and Gallier need user defined correspondences to transform between shapes of different topology [3].

1.2. Overview

The general idea of our approach follows the one of Kent et al [13]. We try to find a polyhedral shape that results from merging the two input shapes. This shape can represent both of the source shapes and a metamorphosis of the shapes might be produced by interpolating the vertices. While interpolating the vertices is interesting in its own right (the so-called vertex-path problem), we consider only the merging problem.

In contrast to other work so far our approach takes into account an arbitrary number of features (points) that must remain aligned during the transformation. For the specification of features we use vertices. This is no restriction as one might introduce additional vertices on every face of the model. The complete process consists of the following three

steps:
1. Find an embedding of the polyhedral model on the unit sphere. The embedding should reflect the spatial relationships of the model's vertices.
2. Given two sphere-embeddings, deform these embeddings so that the aligned vertices have the same position on the unit sphere.
3. Compute an overlay of the two graphs.

With this scheme aligned vertices are fused in the merging step. Thus, they have to remain aligned during the transformation. We will discuss each of the above three steps in detail in the upcoming sections.

For the purpose of transformation the merged model has to be deformed to represent the source models, which is covered in the fourth section. The last section reports on implementation issues and results of the approach.

2. Embedding the Polyhedron on a Sphere

We consider genus 0 polyhedra, because these are the only models that have an topologically equivalent embedding on a sphere. This embedding is defined by a graph reflecting the relationship between the vertices. The graph of genus 0 polyhedrons is necessarily planar (this dates back to Euler), i.e. it can be drawn in the plane such that no two edges intersect. The theory of graphs, especially planar graphs, is important for our way of finding an embedding on a sphere. We assume that the reader is familiar with some basic terms from graph theory, which can be found in every textbook (such as [9]).

It is known that every planar graph has a straight line embedding, i.e. the edges could be represented by straight, non-intersecting lines (independently established by Wagner [22], Fary [6] and Stein [19]).

As the first step of the procedure the polyhedron is triangulated. In terms of graph theory we make the graph maximal. This assures that the embedding of the graph is unique and, therefore, no extra effort has to be spent in order to preserve the topology during the mapping onto the sphere.

The general idea of our approach is to adopt an algorithm for producing a straight-line embedding of a planar graph to produce a "straight-arc" embedding of the graph on a unit sphere. We define the straight-arc embedding to be the one, where an edge connecting two vertices is represented by the shortest possible path along the surface of the sphere. This shortest path is the part of the great circle passing through the two endpoints on the sphere. The existence of such an embedding for planar graphs follows from Steinitz's theorem on convex polytopes in three dimensions [20].

2.1. Straight-Line Embedding Algorithm

Among the many algorithms for producing straight line embeddings we use a very simple one [21], which is well suited for our needs. We first describe the algorithm in the case of planar embeddings:
1. Identify a peripheral cycle in the graph and fix its vertices on a convex boundary. In the case of a triangulated polyhedron as source of the graph every triangle bounding a face is a peripheral cycle.
2. Place every vertex (except from those of the outer cycle) at the centroid of its neighbors. Neighbors are those vertices that are connected by an edge.

The positions of the vertices are naturally described by a system of linear equations. It can be proven with the use of matrix theory that this system always has a unique solution. This solution is a planar embedding of the graph.

2.2. Straight-Arc Embedding Algorithm

We want to accommodate this algorithm to produce an embedding on a sphere. The second step in the above scheme is easily adapted. Instead of placing the vertex in the centroid we project this centroid onto the sphere. We do this by simply normalizing the vector of the centroid, which is, in fact, a central projection through the center of the unit sphere. Due to this projection, the former linear system of equations is not linear anymore. The system of equations can still be solved with relaxation methods. We will describe the process more formally and in more detail after we have explained the first step.

As in the case of planar embeddings several vertices on the sphere have to be fixed. In the remainder we will call the fixed vertices *anchors*. Without a sufficient number and placement of anchors the embedding on the sphere collapses, i.e. several vertices are collinear or in the same position, because this is a state, in which vertices lie in the centroid of their neighbors.

Contrary to the situation on the plane it is not enough to fix one triangle. In this case the relaxation would result in an embedding where all vertices lie inside the triangle (see embedding a) of figure 1). More generally, if there exists a half sphere without any anchor, no vertex would be placed in that empty half sphere, because this can't be the projection of any of the others vertices centroid. Thus, at least four anchors are needed, distributed over the sphere such that every half sphere contains at least one of the fixed vertices. It seems natural to place the anchors at the edges of regular tetrahedron, which has the unit sphere as a circumsphere.

While we need anchors to avoid the collapse of the embedding, every anchor also causes a problem: The fixed vertices are not placed in the centroid of their neighbors. Experiments show that this results in foldovers of the graph (see embedding c) in figure 1 for a dramatic example). Since every anchor (except the first one) causes a potential foldover, we decide to use a regular tetrahedron as the anchor of our embedding.

We are still faced with two problems: First, even with only four anchors some foldovers might occur, and second, the re-

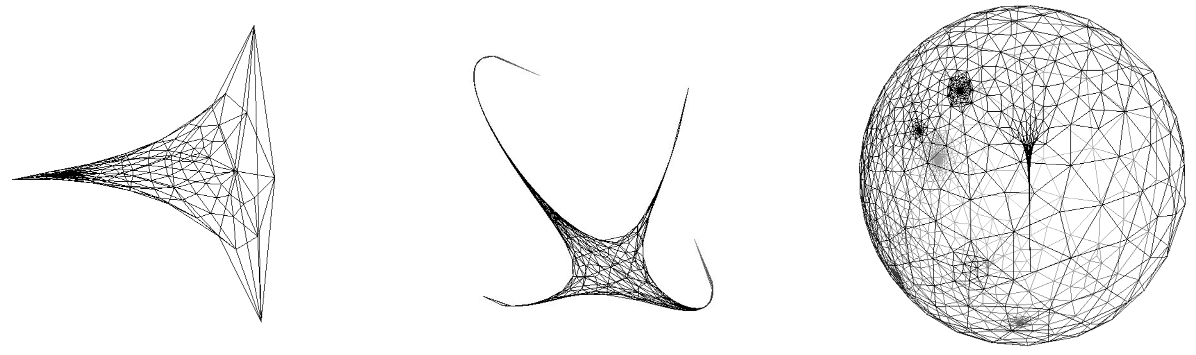
Figure 1: Problems of the sphere embedding: a) Collapsed embedding fixed with three vertices. b) Collapsed embedding fixed with the four vertices of a regular tetrahedron. c) Foldover of the graph.

laxation turns out to be sensitive to the initial state of the vertices. Even with a tetrahedron as anchor the embedding might collapse (see embedding b) in figure 1). We have solved both problems with heuristics. So far we have found a valid embedding for every Euler valid genus 0 model.

2.3. Resolving Foldovers

To resolve the foldovers from anchors the locality of the problem is exploited. Speaking again in terms of energy, the vertices far away from the anchors are in a state of very low energy, i.e. they have reached a state where every vertex is in the projection of the centroid of its neighbors. The idea to resolve the foldovers from an embedding is as simple as effective: Anchors are placed where the vertices have reached the minimum energy state. As a heuristic, the vertices diametric to those of the tetrahedron are fixed. Then the relaxation scheme is repeated.

We have to be able to judge the current state of the embedding. As a criterion we use the orientation of vertices along a face. If the embedding contains no foldovers all orientations of three consecutive vertices in clockwise order around the face have to be the same. We expect the representation of the polyhedral model to contain information about the order of vertices around a face.

2.4. Initializing the Relaxation

Again, heuristics are used to find a good initial state for the relaxation process. The general idea is to find a circumsphere of the model, whose center is translated to the origin and which is subsequently scaled to a unit sphere. We use the positions of the vertices as the initial state for the mapping to the sphere.

It remains to choose a certain circumsphere and the orientation of the tetrahedron of anchors, which together lead to a stable relaxation. In general, it causes problems if the center of the circumsphere is not inside the model. Nevertheless, the smallest enclosing sphere of the model seems to be a good starting point in search of the desired circumsphere. In cases where the center of this sphere does not lie inside the model the center is translated to an interior point. For many models the center of mass is such a point.

Experience showed that the orientation of the anchor-tetrahedron does not influence the relaxation for most models. However, in some cases there are orientations that do not lead to the desired embedding. In any case, it has proved to be sufficient to generate random orientations until the relaxation is stable.

We now give an overview of the complete algorithm in a more formal way. Note: In several places we need to normalize vectors to pull them back on the unit sphere. We will use the notation $v^* = v / \|v\|$ as an abbreviation throughout the paper.

2.5. Complete Algorithm for the Initial Embedding

Given a three-dimensional model in a facet-based representation consisting of faces F edges E and vertices $V = \{\vec{v}_0, \vec{v}_1, \ldots\}$, such that $|V| - |E| + |F| = 2$. We assume that for each face we have at least information about the vertices that are incident upon this face and their linear order along the boundary of the face.

1. Compute the smallest enclosing sphere of the model. A fast approximating approach is described in [16], an exact solution might be obtained with a randomized incremental algorithm as in [23].
2. Translate the center of the sphere to a point inside the model.
3. Transform the model such that the circumsphere is transformed to a unit sphere with its center at the origin of the coordinate space.
4. Choose a random regular tetrahedron with vertices on the unit sphere and fix the vertices of the model that are closest to those of the tetrahedron.
5. Relax the vertices V of the model by repeating the following step

$$\forall i \therefore \vec{v}_i = \left(\sum_{n \in \text{ neighbors of } \vec{v}_i} \vec{v}_n \right)^*$$

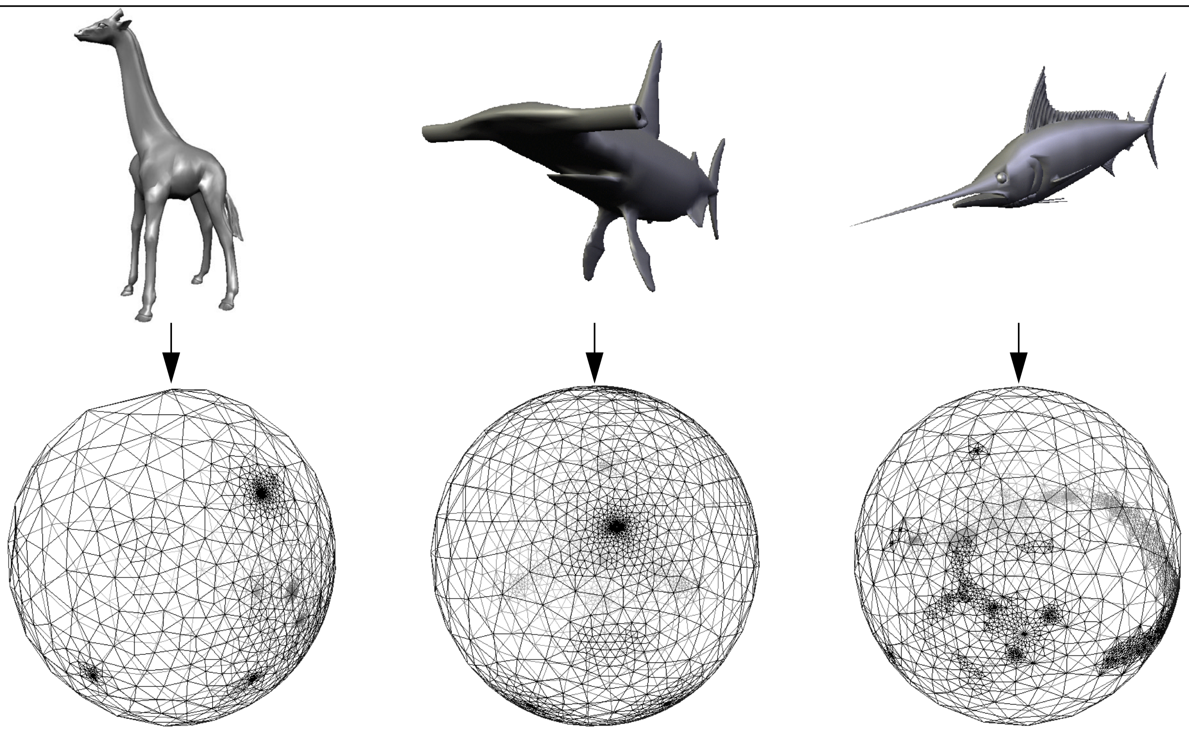

Figure 2: Sphere embeddings of polyhedral models of a) a giraffe, b) a hammerhead shark, and c) a swordfish.

until the largest movement of any of the vertices is less then a predefined epsilon:

$$max\left\|\vec{v_i^r} - \overrightarrow{v_i^{r+1}}\right\| < \varepsilon$$

6. If the embedding is collapsed go back to step 4. To check whether the embedding is collapsed we compute the distances of the nearest neighbors to the diametric points of the vertices of the tetrahedron. If this distance is not substantially less than half of the distance between the anchors we consider the embedding collapsed.
7. Fix the vertices diametric to the tetrahedron from step 4.
8. Relax the vertices according to step 5.
9. Check the orientation of every three consecutive vertices along the boundary of a face. The orientation of three vertices $\vec{v_0}, \vec{v_1}, \vec{v_2}$ on a sphere is: $\mathrm{sgn}((\vec{v_0} \times \vec{v_1}) \cdot \vec{v_2})$. If the orientation is not the same for every triple, the epsilon bound is decreased and step 8 is repeated. The embedding is complete if all orientations are the same.

Figure 2 shows the initial sphere embeddings of polyhedral models of a giraffe, a hammerhead shark, and a swordfish.

3. Aligning the Features

Of course, we could overlay the embeddings of two models retrieved from the initial step, then transform the merged model to the shape of the source models and perform a linear interpolation. The left hand side of figure 3 shows such a transformation between the models of a pig and a horse. While this is clearly a smooth transformation between two polyhedral shapes it might not be what one would expect if a pig is transformed into a horse. Instead, we would like to see the basic characteristics (head, legs, etc.) of the two mammals to remain stable during the transformation. This can only be done with feature-alignment. The right-hand side of figure 3 shows a transformation, where the basic features were aligned.

Only the user knows what features would correspond in two models. In order to specify this correspondence features are identified with vertices. A simple way to assure that features are preserved during the transformation is to place the corresponding vertices at (approximately) the same positions on the two spheres. Thus, in the merged model they are fused or at least very close by. This characteristic cannot change during any transformation. That is, with our approach, the alignment of features is independent of the actual interpolation technique, but assured by the topology of the merged model. The general idea to achieve the alignment of corresponding vertices on the spheres is to use deformation techniques. For that, we can borrow from the extensive research on image morphing and warping. Note that our deformation problem has an additional constraint: edges of the graph must not intersect. This could be achieved by using warping techniques that guarantee an injective mapping. But this is not enough: If we map only the vertices and define the edges as arcs connecting them these arcs might intersect, because they are not deformed according to the mapping. If the arcs are de-

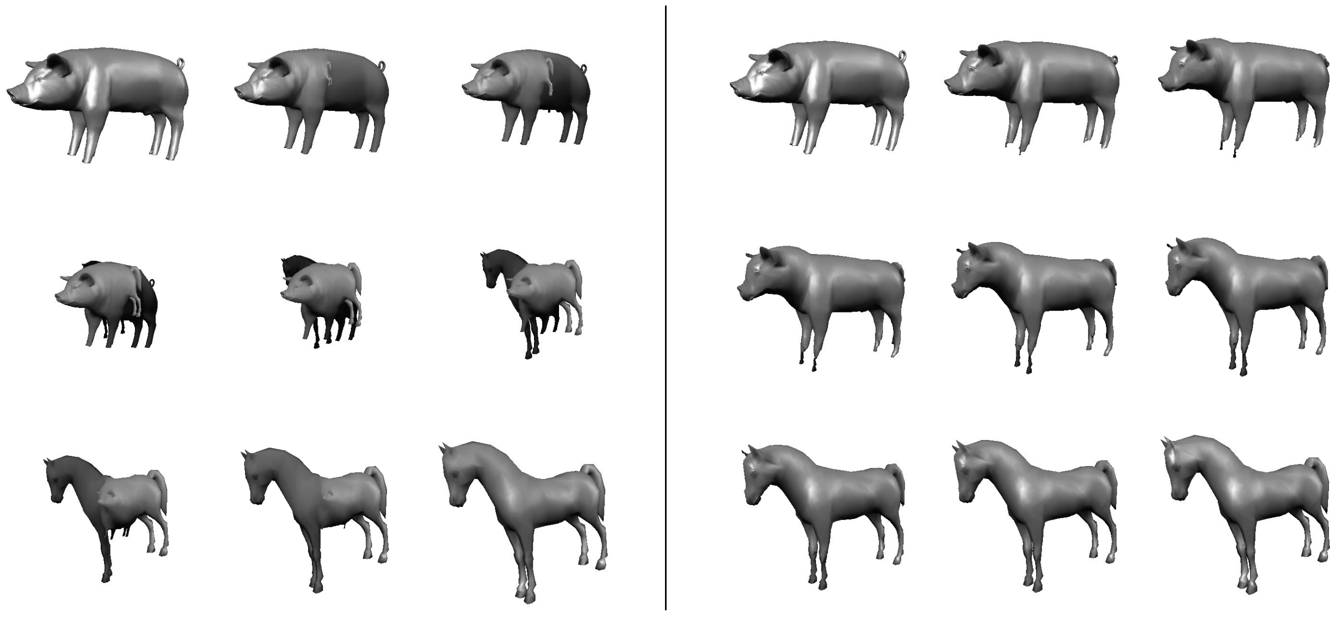
Figure 3: Transformations between the models of a pig and a horse. The sequence on the left hand side uses no features. The sequence on the right hand side aligns the head, legs, and the tail. Note the problems of linear vertex interpolation at the forelegs.

formed, too, the overlay problem becomes unsolvable, because the intersections of deformed arcs are intractable.

Because of this problem elaborate warping techniques are discarded, since one cannot take advantage of their mathematical properties. Instead, we have developed a very simple iterative scheme inspired by radial basis functions (see [1]). The following deformation technique is used to move one vertex: Given a vertex at position $\vec{v}$ that should move to $\vec{w}$, the map f is defined as

$$f(\vec{x}) = \begin{cases} (\vec{x} + (d - \|\vec{x} - \vec{v}\|)(\vec{w} - \vec{v}))^* & \text{if } \|\vec{x} - \vec{v}\| < d \\ \vec{x} & \text{if } \|\vec{x} - \vec{v}\| \geq d \end{cases}$$

In order to remap a couple of vertices into desired positions the vertices are repeatedly moved into their desired positions. Every time a vertex is placed correctly some of the other vertices are also displaced. The amount of this displacement is less than the displacement of the vertex moved, though. Thus, the displacements necessary to correct a position of a vertex are decreasing and eventually all vertices come to a stable position. We prefer this iterative scheme because the displacements are bounded and easy to control. In this way, it is possible to check the validity of the embedding in each step and decrease the deformation, if necessary. Now we explain the complete algorithm to align corresponding vertices.

3.1. Algorithm for aligning corresponding vertices

Given two embeddings of vertices V_1,V_2 and edges E_1,E_2 on the sphere as well as two ordered sets of features represented as a number of indices in the sets F_1,F_2 (we assume $|F_1| = |F_2|$). We denote the i-th element of V^S as $\vec{v}_i^S$, the i-th element of F_S as f_i^S. The goal of the following procedure could be written as

$$\forall i \therefore \vec{v}^1_{f_i^1} = \vec{v}^2_{f_i^2}$$

1. Rotate the first sphere so that the summed squared distance
$$s_d = \sum_i \left\| \vec{v}^1_{f_i^1} - \vec{v}^2_{f_i^2} \right\|^2$$
of corresponding vertices is minimized.
2. Compute the desired position $\vec{p}_i$ for each pair of vertices as the normalized sum of corresponding vertices:
$$\vec{p}_i = \left(\vec{v}^1_{f_i^1} + \vec{v}^2_{f_i^2} \right)^*$$
3. Define the displacement $\vec{t} = \vec{p}_i - \vec{v}^S_{f_i^S}$ necessary to move the i-th feature. Then apply the following map to all vertices:
$$\forall i \forall j \therefore \vec{v}_j^S = \begin{cases} \left(\vec{v}_j^S + \vec{t}\left(d - \left\| \vec{v}_j^S - \vec{v}^S_{f_i^S} \right\| \right) \right)^* & \text{if } \left\| \vec{v}_j^S - \vec{v}^S_{f_i^S} \right\| < d \\ \vec{v}_j^S & \text{if } \left\| \vec{v}_j^S - \vec{v}^S_{f_i^S} \right\| \geq d \end{cases}$$
Check after each mapping if the embedding is still valid. If not, decrease the length of the displacement vector $\vec{t}$.
4. Repeat step 3 until the positions of the vertices remain unchanged after a complete round.
5. If the features vertices are not in their desired positions and d is greater than zero decrease d and go back to step 3.

If the topologies permit, the feature vertices are now aligned.

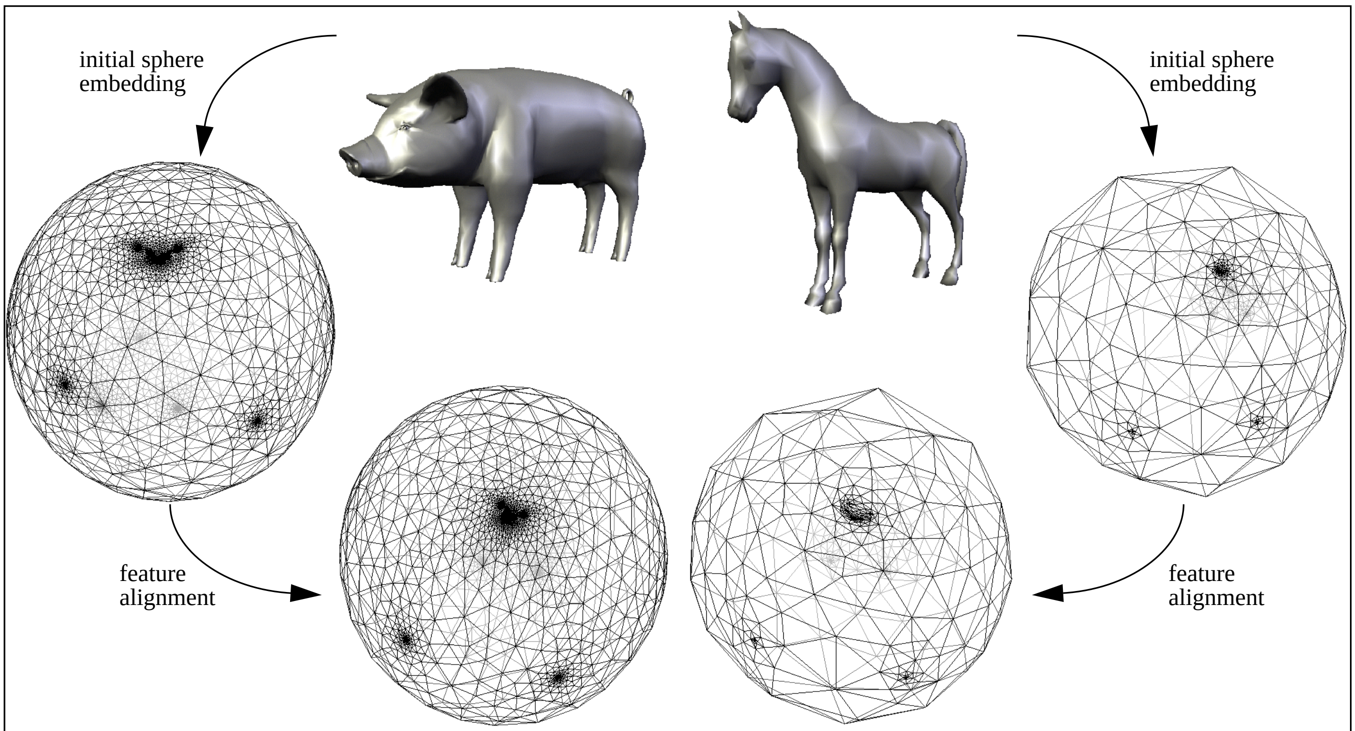

Figure 4: The feature alignment process: First, an initial sphere embedding is computed. Second, the inital sibdivision is deformed such that the common features coincide on the spheres.

Note that it is not necessarily the case: The mapping might have introduced some foldovers and has been rejected in step 3. In this case d is decreased until it is zero but the features are moved only as far as their movement did not introduce any foldover.

Figure 4 illustrates the feature alignment step for models of a pig and a horse.

4. Merging the Embeddings

Given two embeddings we need to produce an embedding that contains the faces, edges, and vertices of both source models. The problem is known as *map overlay.* Several algorithms were proposed for this problem but they handle mainly the case of planar graphs (see a textbook like [2]). In general, the planar map overlay has the complexity $O(n\log n + k)$, where n is the number of edges and k is the number of intersections. If the two subdivisions are connected (as in our case) the planar overlay can be computed in $O(n + k)$ ([7]). There seem to be only a few publications about the overlay of subdivision on the sphere and it is not clear whether (and which of the) solutions for the planar case are applicable in our context. Kent et al. [13] give an algorithm for the sphere overlay problem, which needs $O(n + k\log k)$ time. We give a new solution to this particular problem, which reports the intersection of two spherical subdivisions in the optimum time of $O(n + k)$. Also, our algorithm exploits the topological properties of both subdivisions, which are used to guarantee the correct order of intersections.

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

In both cases a sophisticated data structure is needed to represent the models. We use the doubly-connected edge list [15]. This data structure contains information about each face, each *directed* edge, and each vertex. The following information is stored for the different types:

- The face record contains a pointer to an arbitrary half edge on its boundary.
- The edge record contains pointer to: The vertex it is originating, the face it bounds, its twin (the half edge connecting the same vertices but pointing in the opposite direction), and the next half edge along the boundary of the bounded face.
- The vertex record contains information about the location in space and a pointer to an arbitrary edge that has this vertex as its origin.

The reader might consult a textbook like [2] to get more information about this structure and especially why the above data are sufficient for all algorithmic tasks.

Note: In the following we assume that none of the vertices of one embedding lies on a vertex or an edge of the other graph. We can assure this *general position assumption* by using a symbolic perturbation scheme like the one described in [8].

4.1. Finding the intersections

In the algorithm two geometric functions are needed: One to decide if and where two edges intersect on the sphere, and a second to decide whether a point lies inside a face. In both cases we cannot adopt planar techniques.

Edge-Edge Intersection. Edges were defined as the shortest path between two points $\vec{p}_1, \vec{p}_2$ on the sphere. This is the shorter arc of the circle with the same center as the sphere passing through the points (in the following we assume a unit sphere). Obviously every two of those circles intersect (or they are the same). The points of intersection can be computed as follows: Each of the circles defines a plane, the intersection of the planes cuts the sphere in the intersection points we are searching for. The intersection of the planes is orthogonal two both normals of the planes, and the normal of a plane is orthogonal to the unit vectors $\vec{p}_1, \vec{p}_2$. Thus, we find the intersection of two circles based by $\vec{p}_1, \vec{p}_2$ and $\vec{q}_1, \vec{q}_2$ as

$$\vec{r} = \pm(\vec{p}_1 \times \vec{p}_2) \times (\vec{q}_1 \times \vec{q}_2)$$

To decide whether the two shorter arcs intersect we have to solve the following two equations for s_p, s_q:

$$t_p\vec{r} = \vec{p}_1 + s_p(\vec{p}_2 - \vec{p}_1)$$

$$t_q\vec{r} = \vec{q}_1 + s_q(\vec{q}_2 - \vec{q}_1)$$

The two arcs intersect, iff $0 \le s_p, s_q \le 1$. The point of intersection is already given in $\vec{r}$.

Point-in-Face. Since the model was triangulated every face is described by three vertices $\vec{v}_1, \vec{v}_2, \vec{v}_3$. We can assume that vertices are ordered counterclockwise around the triangle as viewed from outside the sphere (the order is known from the next pointers in the edge list). We define three planes by two subsequent vertices and the origin. The intersection of the positive half-spaces of these planes contains exactly the face. A point lies in the positive half-space of a plane if the sign of the dot product with the normal of this plane is positive. Thus, a point $\vec{p}$ lies inside a spherical triangle bounded by $\vec{v}_1, \vec{v}_2, \vec{v}_3$ iff:

$$((\vec{v}_1 \times \vec{v}_2) \cdot \vec{p} \ge 0) \wedge ((\vec{v}_2 \times \vec{v}_3) \cdot \vec{p} \ge 0) \wedge ((\vec{v}_3 \times \vec{v}_1) \cdot \vec{p} \ge 0$$

Now that we can answer these basic geometric questions the algorithm will be explained: Given two embeddings consisting of vertices $\underline{V}, \bar{V}$, edges $\underline{E}, \bar{E}$, and faces $\underline{F}, \bar{F}$. For every edge $\underline{e}$ in $\underline{E}$ we need a list of edges in $\bar{E}$ that intersect $\underline{e}$ and vice versa. These lists are generated by traversing the graph. Actually, both of the graphs are traversed in almost the same way, so we describe the process only for traversing and generating the intersection lists for $\underline{E}$.

Traversing to find intersections. The basic idea is to traverse the graphs breadth first. Choose an arbitrary vertex $\underline{v} \in \underline{V}$ and search the face $\bar{f} \in \bar{F}$ that contains $\underline{v}$. Start with the edge $\underline{e}$ contained in the record of $\underline{v}$. For the traversal a stack of edges to inspect is maintained. For each half edge pushed onto the stack the face that contains the origin of the half edge is also pushed. Thus, $\underline{e}$ and $\bar{f}$ are pushed on the stack. Then the following procedure is repeated until the stack is empty:

1. Pop a working edge $\underline{e}$ and a face $\bar{f}$.
2. Mark the edge $\underline{e}$ and its twin $\underline{e}'$ used.
3. Initialize a variable $\bar{c}$ that stores the edge of $\bar{E}$ that was intersected last.
4. Test whether $\underline{e}$ intersects with one of the boundary edges $\bar{e}_1, \bar{e}_2, \bar{e}_3 \ne \bar{c}$ of $\bar{f}$.
5. If $\underline{e}$ intersects $\bar{e}_i \ne \bar{c}$ then replace $\bar{f}$ with the face stored in the twin of $\bar{e}_i$, that is, the face on “the other side” of $\bar{e}_i$ and go back to step 3. If no intersections are reported a topologically ordered list of intersections for the half edge $\underline{e}$ was generated.
6. $\bar{f}$ is (now) the face that contains the endpoint of $\underline{e}$. Thus, push all unused edges originating from this endpoint together with $\bar{f}$ on the stack.

This procedure is done a second time with the under- and upperscores exchanged. Actually, it is useful to generate the new vertices in one of the traversals, which makes the traversals slightly different. We need information about the relative positions of all vertices later. Therefore, the three vertices making up a face in one graph are stored for all vertices in the other graph (this is done when edge and face are popped from the stack). For the newly generated vertex the four vertices incident upon the two intersecting edges are stored.

Is is easy to see that the algorithm is linear in the number of edges plus intersections: For each edge three intersection tests are made and for each intersection of this edge additional three intersection tests are necessary, which are constant costs per edge and per intersection. In the initialization of the traversal it was tested for all faces whether they contain the starting vertex. This is also linear in the number of edges.

4.2. Constructing a representation

We want to compute a list of vertices $\hat{V}$, edges $\hat{E}$, and faces $\hat{F}$ from the vertices $\underline{V}, \bar{V}$, edges $\underline{E}, \bar{E}$, and faces $\underline{F}, \bar{F}$ as well as the intersection lists generated in the above procedure. Both subdivisions are given as doubly connected edge lists and a doubly connected edge list should be constructed for the merged model.

We start with initializing $\hat{V} = \underline{V} \cup \bar{V}$ and $\hat{E} = \underline{E} \cup \bar{E}$. All the vertices originating from the intersections in the first part of the algorithm are already generated. But contrary to the vertices in $\hat{V}$, these vertices do not contain valid information about a half edge.

The basic idea is to process all intersection lists for the edges in $\underline{E}$. For each intersection the intersecting edges have to be cut. Because the original edges are changed in this step the intersection lists have to be updated or new intersection lists

have to be created. The invariant we like to maintain is that the intersection lists are valid for the current list of edges. The intersections for one edge are processed in the order given by the intersection list. Thus, the intersections inserted in the edge from $\underline{E}$ are in topologically correct order. It will become clear later on, where the information about the edges from $\bar{E}$ (the intersection lists for these edges) is used, such that these edges are also cut topologically correct.
First we explain how to cut two edges (four half edges). The intersection list of $\underline{e}$ contains information about an intersection with an edge $\bar{e}$. Because the faces of the subdivisions are convex (triangles), and $\underline{e}$ was leaving the face that bounded $\bar{e}$, the situation is always the same: $\underline{e}$ and $\bar{e}$ make a clockwise turn.
The four edges already in $\hat{E}$ ($\underline{e}$, $\bar{e}$ and their twins) keep their origin. The four new edges will have the new vertex as their origin. It remains to update the information about the next edge along the boundary of a face and the twins.
The intersection list of $\bar{e}$ and of some edges in $\underline{E}$ might not be valid anymore: Some edges in $\underline{E}$ cut $\bar{e}$ in the part that now is the newly generated edge originating from the new vertex (we call this edge $\tilde{e}$). To find these edges the information in the intersection list of $\bar{e}$ is inspected. The edges in this intersection list are browsed until $\underline{e}$ is found (which has to be in the list, because $\bar{e}$ was in the list of $\underline{e}$). $\underline{e}$ can be safely removed from the intersection list. All edges in the intersection list after $\underline{e}$ now belong to $\tilde{e}$. That is, these edges are removed from the list and a new intersection list is generated for $\tilde{e}$. Then the intersection lists for all edges in this new intersection list have to be updated. These intersection lists originally pointed to $\bar{e}$ and now have to point to $\tilde{e}$. After these updates the invariant holds again.
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. Faces are easily found by following cycles of half edges: Loop through all edges. If the edge does not contain information about a face follow the next pointers of the edges until the first edge is found again. Create a new face, set the record of the edges to this face, and the record of the face to an arbitrary edge. After all edges are processed the complete information about faces is created. Note that faces created from intersecting triangles are not necessarily triangles themselves, but they are convex.

5. Reconstructing the Source Models

In most of the applications the purpose of the merged model is to serve for a transformation between the original polyhedra. Thus, the merged model has to be deformed to have exactly the shape of the source models. We refer to vertices, edges, and faces as in the previous sections.
Note first of all that the information about edges and faces remains unchanged. Only the vertex positions have to be set. Of course, the vertices of $\hat{V}$ that are elements of $\underline{V}$ can be set to their original values. For the other vertices some additional information is needed. This information was computed while finding the intersections. We distinguish between vertices of $\bar{V}$ and the new vertices.
The vertex $\bar{v}$ of $\bar{V}$ belongs to a face in $\underline{F}$. Remember that we stored the three vertices $\underline{v}_1, \underline{v}_2, \underline{v}_3$ around that face. We compute the barycentric representation of $\bar{v}$ in the basis of $\underline{v}_1, \underline{v}_2, \underline{v}_3$. This barycentric representation is used to interpolate between all attributes of $\underline{v}_1, \underline{v}_2, \underline{v}_3$. The values are found by writing the attributes in the columns of a matrix and multiplying this matrix with $\bar{v}'$. This way, not only the position is easily computed but also color values or texture coordinates might be interpolated.
If a vertex v does not belong to $\underline{V}$ nor to $\bar{V}$ it was generated due to an edge-edge intersection. The four vertices of the two intersecting edges were stored for this vertex. In each of the reconstruction processes only the two vertices that belong to the original model are needed. In this case the vertices $\underline{v}_1, \underline{v}_2$ of the intersecting edge $\underline{e}$ are used. As in the step above a barycentric representation is calculated (this time with respect to $\underline{v}_1, \underline{v}_2$). Again, this barycentric representation is used to interpolate all attributes of the supporting vertices $\underline{v}_1, \underline{v}_2$.
Also, the faces of the merged model are not necessarily triangles and more than three points are not necessarily coplanar, which might cause problems during a transformation. Thus, the models should be triangulated prior to a transformation.

6. Implementation and Results

We have implemented the above explained procedures in Java/Java3D using a native function in C for the relaxation step. Table 1 below gives some execution and timing information for the initial embedding. Times are average execution times, not CPU seconds, for a Sun Ultra 10 in a regular network and with normal workload. The initial embedding step makes use of a native function coded in C.

model	vertices	edges	faces	time
duck	629	1597	970	1.8 sec
giraffe	4197	9537	5342	142.8 sec
horse	674	1535	863	4.5 sec
pig	3522	7689	4169	45.9 sec
hh shark	2564	5365	2803	19.8 sec
shuttle	310	701	393	1.5 sec
swordfish	2930	6726	3797	42.2 sec

Table 1: Information about the initial embedding step

In order to make the feature specification feasible we have designed a GUI that allows to pick vertices of the source models. These feature vertices are aligned in the second step.

The deformation of two models towards the alignment of their features is coded completely in Java. Due to the nature of the algorithm the cost increases linear with the number of features. Table 2 below shows timing information for some of the models with different number of features to be aligned. Also, information is given, whether the features could be completely aligned without creating foldovers.

models aligned	features	aligned	time
shark / shuttle	4	no	7.0 sec
horse / giraffe	16	yes	51.5 sec
horse / pig	14	yes	34.1 sec
horse / pig	22	yes	53.0 sec
giraffe / pig	23	no	117.1 sec

Table 2: Information about the deformation step

In the third step the sphere subdivisions are merged. The algorithm is also completely coded in Java. Of course, the merging step does not depend on the number of features directly, but it depends on the number of intersecting edges, which might be influenced by the deformation of the embedding. The table below gives information about the merging step. The timing information (given in the upper triangle matrix) has to be seen with respect to the number of vertices of the output models (given in the lower triangle matrix).

	giraffe	horse	pig	shark	swordfish
giraffe		4.7 sec	8.1 sec	7.5 sec	6.8 sec
horse	7152		4.2 sec	3.6 sec	3.2 sec
pig	12509	6418		6.7 sec	8.3 sec
shark	11825	6585	11540		6.8 sec
swordfish	11365	5448	14645	11243	

Table 3: Information about the merging step.

Finally, we like to present another transformation obtained by linear interpolating the vertex positions. Figure 5 in the color section shows the transformation from a duck to a space shuttle with feature-alignment. Only four corresponding vertices were defined. Picking the vertices in the GUI was done in less than a minute.

7. Conclusions

We have presented a technique to merge arbitrary genus 0 polyhedra. The merging procedure is capable of aligning scattered features given as corresponding vertices. This allows for smooth transitions between polyhedra such that common features are preserved.

The approach has proven to be fast, especially when compared to methods proposed so far. However, we would like to deepen the understanding of the embedding and deformation process, such as to be able to better predict and control the complete process.

8. References

[1] Nur Arad and Daniel Reisfeld. Image Warping Using few Anchor Points and Radial Basis Functions. *Computer Graphics Forum,* 14, 1, 23-29, 1995

[2] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Heidelberg, 1997

[3] Douglas DeCarlo and Jean Gallier. Topological Evolution of Surfaces. *Proc. of Graphics Interface ‘96,* 194-203, 1996

[4] Eyal Carmel and Daniel Cohen-Or. Warp-guided Object-space Morphing. *The Visual Computer,* 13, 9/10, 1998

[5] Shenchang Eric Chen and Richard E. Parent. Shape Averaging and Its Applications to Industrial Design. *IEEE Computer Graphics and Applications,* 9, 1, 47-54, 1989

[6] I. Fary. On Straight Lines Representation of Planar Graphs. *Acta Sci. Math. Szeged,* 11, 229-233, 1948

[7] Ulrich Finke and Klaus Hinrichs. Overlaying simply connected planar subdivisions in linear time. *Proceedings 11th Annual ACM Symp. on Computational Geometry,* 119-126, 1995

[8] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. on Graphics,* 9, 66-104, 1990

[9] Frank Harary. *Graph Theory,* Addison-Wesley, Reading, MA, 1969

[10] T. Hong, M. Magnenat-Thalmann, and D. Thalmann. A General Algorithm for 3-D Shape Interpolation in a Facet-Based Representation. *Proc. of Graphics Interface ‘88,* 229-235, 1988

[11] Anil Kaul and Jarek Rossignac. Solid-Interpolating Deformations: Construction and Animation of PIPs. *Eurographics ‘91,* 493-505, Eurographics Association, Amsterdam, 1991

[12] James R. Kent, Richard E. Parent, and Wayne E. Carlson. Establishing Correspondences by Topological Merging: A New Approach to 3-D Shape Transformation. *Proceedings of Graphics Interface ‘91,* 271-278, 1991

[13] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape Transformation for Polyhedral Objects. *Computer Graphics (SIGGRAPH ‘92 Proceedings),* 26, 2, 47-54, 1992

[14] F. Lazarus and A. Verroust. Feature-based shape transformation for polyhedral objects. *Proceedings of the Fifth Eurgraphics Workshop on Animation and Simulation,* 1994

[15] D.E. Muller and F.P. Preparata. Finding the intersections of two convex polyhedra. *Theor. Comp. Sc.,* 7, 212-236, 1978

[16] Jack Ritter. An Efficient Bounding Sphere. In Andrew Glassner, editor, *Graphics Gems,* 301-303, Academic Press, San Diego, London, 1990

[17] Thomas W. Sederberg and Eugene Greenwood. A Physically Based Approach to 2-D Shape Blending. *Computer Graphics (SIGGRAPH ‘92 Proceedings),* 26, 2, 25-34, 1992

[18] Michal Shapira and Ari Rappoport. Shape Blending Using the Star-Skeleton Representation. *IEEE Computer Graphics and Applications,* 15, 44-50, 1995

[19] S.K. Stein. Convex Maps. *Proc. Amer. Math. Soc.,* 2, 464-466, 1951

[20] E. Steinitz and H. Rademacher. *Vorlesung über die Theorie der Polyeder*. Springer, Berlin, 1934

[21] W.T. Tutte. How to Draw a Graph. *Proc. London Math Soc,* 3, 13, 743-768, 1963

[22] K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jber. Deutsch. Math-Verein,* 46, 26-32, 1936

[23] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, New Results and New Trends in Computer Science, 359-370, Springer-Verlag, 1991