Metamorphosis of Polyhedral Surfaces using Decomposition

Shymon Shlafman, Ayellet Tal[†] and Sagi Katz

Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel

Abstract
This paper describes an algorithm for morphing polyhedral surfaces based on their decompositions into patches. The given surfaces need neither be genus-zero nor two-manifolds. We present a new algorithm for decomposing surfaces into patches. We also present a new projection scheme that handles topologically cylinder-like polyhedral surfaces. We show how these two new techniques can be used within a general framework and result with morph sequences that maintain the distinctive features of the input models.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computational Geometry and Object Modeling]: Boundary representations I.3.7 [Three-Dimensional Graphics and Realism]: Animation

Keywords: Metamorphosis, shape transformation, surface decomposition

1. Introduction

Metamorphosis of three-dimensional polyhedral models has been a lively topic of research for many years. To generate a pleasing morph sequence, it is usually required to find a good correspondence between the models before an interpolation is applied.

A common approach for finding a correspondence between two given polyhedra is to look for a common embedding of their topologies (i.e., their one-skeleton graphs). This is done by projecting the models onto a common parameterization domain, merging their one-skeleton graphs in this domain, and projecting the merged topology back to the original models. Various projection domains and projection techniques have been proposed. For instance, in [8, 5] the polyhedra are projected onto the plane. In [10] the polyhedra are projected onto the surface of a sphere. In [14] the polyhedra are projected onto the surfaces of convex polyhedra.

This general approach has a couple of drawbacks. First, fine correspondence is hard to achieve since the projection is global. This can result with visible artifacts when features in one object are transformed into completely different features on the other. Second, it is necessary to assume that the input

models are either genus-zero polyhedra or disk-like polyhedral surfaces, for the above algorithms to be applicable. Most models found in VRML libraries, however, are neither. Models are rarely two manifolds, let alone genus-zero. Usually, models are "polygon soups", consisting of sets of triangles without any restrictions applied.

To overcome the first shortcoming, it is proposed in [1] that the user specifies corresponding feature points on the polyhedra's surfaces. This algorithm is shown to generate pretty morph sequences. However, specifying many points might make the global embedding impossible, in addition to being a burden on the user. And, it is still required that the input models are genus-zero polyhedra.

One way to get over both shortcomings is to decompose the objects compatibly prior to their projection. Then, a common embedding is found for each compatible pair of patches. This approach was first proposed by [7, 16], where impressive morph sequences are produced.

Both [7] and [16] require that the models are decomposed manually, which might not always be a simple task. Moreover, it is required that the resulting patches are all topologically disks. However, for many models, meaningful decompositions necessarily include topologically *cylinder-like* patches (i.e., polyhedral models having two closed polygonal boundaries). Think for instance on legs of furniture or legs of animals. In fact, rotational sweep models or general sweep objects all consist of cylinder-like patches. In [16] it is

proposes to cut cylinder-like patches along a cutting line, thus converting them into disk-like patches. This scheme works, but might result with visible artifact along the cut.

In this paper we follow [7, 16] and add two novel aspects. First, we propose a novel decomposition algorithm for decomposing polyhedral models. Second, we propose a new algorithm for projecting cylinder-like patches.

The main idea that underlies our scheme is that given a model, its components and the way they relate to each other characterize this model and portray its distinctive features. This is supported by observations that the visual system tends to segment complex objects at regions of matched concavities [2]. Thus, it is important to morph the meaningful components of the models to each other.

This is in particular the case when the models belong to the same family of objects and are highly similar to each other. In this case the viewer expects that the metamorphosis maintains this similarity throughout the morph sequence. Figure 1 illustrates this case, by showing a few snapshots from a movie that morphs a cheetah into a tiger, as generated by our system. As can be seen, the gradual changes are hardly noticeable. (The animals are colored according to their decompositions.)

The rest of the paper is organized as follows. In Section 2 we give some definitions and describe the general algorithm. The next four sections describe the various steps of our algorithm. Section 3 describes the decomposition algorithm. Section 4 describes the algorithm for projecting cylinder-like patches. Section 5 reviews some well-known disk-like projection algorithms and presents a comparative study. Preserving continuity across patch boundaries is described in Section 6. Results are presented in Section 7. Finally, conclusions are drawn in Section 8.

2. General Algorithm

We begin this section with a few definitions and then outline our algorithm. Let S be a polyhedral surface with n vertices.

Definition 2.1 *Decomposition:* $S_1, S_2, \cdots S_k$ is a decomposition of S iff (i) $\forall i, 1 \leq i \leq k$, $S_i \subset S$, (ii) $\forall i$, S_i is connected. (iii) $\forall i, j, i \neq j, 1 \leq i, j \leq k$, S_i and S_j are face-wise disjoint (i.e, the patches can only intersect in a vertex or along an edge) and (iv) $\cup_{i=1}^{k} S_i = S$.

Definition 2.2 *Decomposition graph:* Given a decomposition $S_1, S_2, \cdots S_k$ of a surface S, a graph $G(V, E)$ is its representative decomposition graph iff each patch S_i is represented by a vertex $v_i \in V$ and there is an arc between two vertices in the graph iff the two corresponding patches share an edge in S.

Definition 2.3 *Compatible decomposition:* Given two surfaces S and T, their decompositions $S_1, S_2, \cdots S_k$ and $T_1, T_2, \cdots T_k$ are called compatible if their decomposition graphs are isomorphic.

Definition 2.4 *Disk-like polyhedral surface:* A polyhedral surface is called disk-like if the following requirements hold: (i) The faces are either disjoint, or they have a single vertex in common, or they have two vertices and the edge joining them in common, (ii) Every internal point is homeomorphic to a disk, and every boundary point is homeomorphic to half a disk. (iii) The surface is connected. (iv) The boundary of the surface is a single simple (3D) polygon.

Definition 2.5 *Cylinder-like polyhedral surface:* A polyhedral surface is called cylinder-like if the following requirements hold: (i)-(iii) as in the disk-like case. (iv) The boundary of the surface consists of two disjoint simple (3D) polygons.

Given two models and their compatible decompositions, our goal is to morph each pair of corresponding patches, while preserving continuity across patch boundaries. Recall that the underlying assumption is that the patches represent the meaningful parts of the objects, thus the boundaries between them represent the essential features which characterize the objects. A decomposition-based morph maintains these essential features. Consequentially, having a good decomposition algorithm is of vital importance to the approach pursued in this paper.

The algorithm consists of the following stages. First, the models are decomposed, as described in Section 3. A novel aspect of our decomposition algorithm is that the level of the decomposition can be controlled by the end user. Thus, it is possible to decompose the objects top-down. In other words, the system first decomposes the objects into a small number of patches, and then selected patches are further decomposed. For instance, the animals are first decomposed into their major organs (e.g., head, body, tail, legs), and then each part is further decomposed (e.g. the head is decomposed into the nose, ears etc.). This minimizes the need for manual corrections to the decompositions.

Once the polyhedra are decomposed compatibly, the problem of finding a global parameterization is broken down into finding a parameterization for each pair of corresponding patches. At this stage each patch is classified as disk-like or as cylinder-like, and accordingly, the patches are embedded onto their corresponding parameterization domains, as described in Sections 4–5. Disk-like patches are projected to the plane while cylinder-like patches are projected to an ideal cylinder. (In case we get other types of patches, they are further decomposed.) It is essential to maintain continuity across the boundaries of the patches, as described in Section 6.

The projected topologies are merged on their parameterization domains in a conventional manner, i.e. [10, 14], and the merged topology is projected back onto the original objects. This merged topology is the correspondence we were seeking.
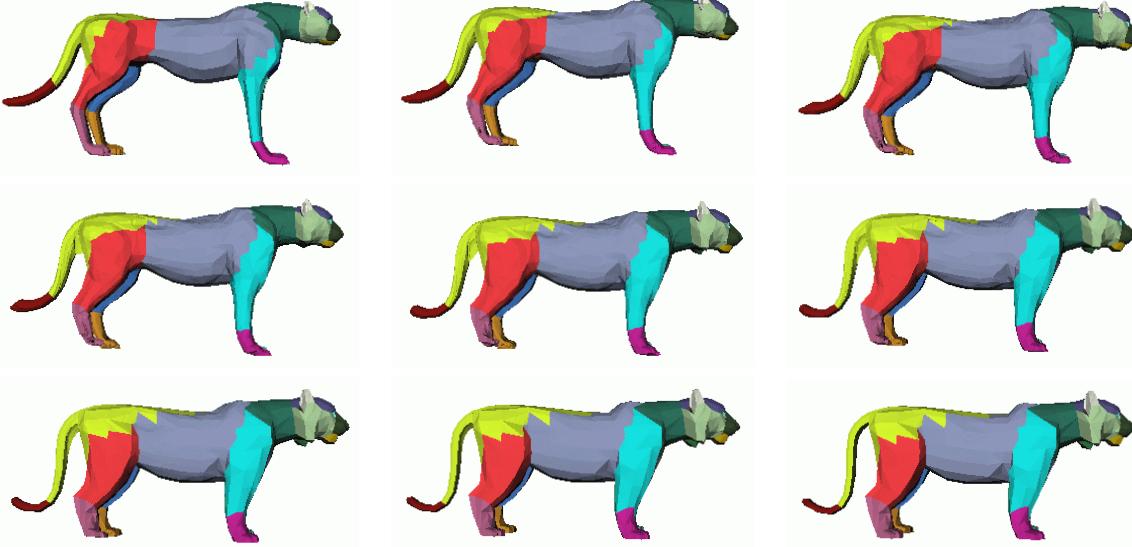
Figure 1: *Metamorphosis of a cheetah into a tiger*

Having established a correspondence, the models are interpolated, to produce the metamorphosis, using any standard interpolation scheme. In the examples, we use linear interpolation. In the next sections we elaborate on each step of the algorithm.

3. Surface Decomposition

Given S, a polyhedral surface with n vertices, the goal is to decompose S into k disjoint patches $S_1, \cdots S_k$ whose union gives S.

In the past, convex decompositions [3] and watershed decompositions [12] were proposed. In this section we describe a new algorithm for surface decomposition. A major benefit of our algorithm is that the number of output patches can be controlled, thus avoiding over-segmentation. The algorithm is particularly suitable for morphing since a small number of meaningful patches is needed, and in our case the user can determine this number.

The algorithm gets as input a three-dimensional model (e.g., in VRML) and a parameter specifying the upper bound on the number of final patches. This parameter can be ∞ in case the user prefers that the system determines the suitable number of patches.

The major decision the algorithm needs to make is whether two given faces should belong to the same patch or not. Possible considerations are convexity [3] and curvatures [12]. Another possible consideration is the proximity of the faces.

Our underlying assumption is that distant faces, both in terms of physical distance and in terms of angular distance, are less likely to be in the same patch than faces which are close together. We therefore define the distance between two faces F_1 and F_2 as follows. If F_1 and F_2 are adjacent, then:

$$Distance(F_1, F_2) = (1 - \delta)(1 - cos^2(\alpha)) + \delta Phys\_Dist(F_1, F_2)$$

The first part of the distance definition measures the angular distance, where α is the dihedral angle between the faces. Note that the expression $(1 - cos^2(\alpha))$ reaches its maximum at $\Pi/2$ and its minimum at 0 (or Π). Thus, coplanar (or close to coplanar) faces are considered close to each other and are more likely to belong to the same patch. The second part of the formula measures the "physical" distance. It is the sum of the distances between the centers of mass of the two faces and the midpoint of their common edge. Note that we choose not to take the distance between the face centers which depends on the dihedral angle. The latter peaks at 0 and gets its minimum at $\Pi/2$, which would be the opposite of what we are trying to achieve. The δ is a weight parameter that allows the user to trade off the two distances.

The distance definition is extended to non-adjacent faces in the following manner. If F_1 and F_2 are non-adjacent, then:

$$Distance(F_1, F_2) = \min_{F_3 \neq F_1, F_2} (Distance(F_1, F_3) + Distance(F_3, F_2))$$

The main idea of the algorithm is to iteratively improve the decomposition by transferring faces from one patch to another. In other words, unlike previous algorithms in which the decomposition is determined and cannot be changed, our

algorithm iterates on switching faces locally as long as the value of some global function is being improved. This is similar to the concept that underlies the K-means clustering algorithm.

The algorithm consists of four steps: preprocessing, electing the initial representatives of the patches, determining the decomposition, and reelecting the representatives. Steps 3 and 4 are iterated until convergence or until some predefined number of iterations is performed. We elaborate on each of these steps below.

1. Preprocessing: During preprocessing the distances between all the adjacent faces are computed. In addition, the faces' normals are fixed when the model is wrongly oriented, and the disconnected components of the model are found. Obviously, the number of patches in the final decomposition should be at least the number of disconnected components.

2. Electing the initial representatives of the patches: Each patch is represented by one of its faces. In principle, the initial representatives of the patches could be chosen randomly. In practice, however, there are a couple of reasons for carefully choosing these representatives. First, our algorithm converges to local minima, which makes the initial decomposition critical for the the final quality of the decomposition. Second, good initial representatives mean that the algorithm will converge after a small number of iterations.

The goal is to maximize the distances between the initial representatives. Initially, one representative is chosen for each disconnected component. It is the face having the minimal distance between its center of mass and the center of mass of its component. If the number of required patches is less or equal to the number of representatives, we are done.

Otherwise, the model should be further decomposed. We calculate, for each representative, the minimum distances to all the faces (e.g., using Dijkstra's algorithm). A new representative is added so as to maximize the average distance to all the existing representatives on the same connected component. New representatives are added one by one, until the required number of representatives (i.e., patches) is reached. In case the user specifies that the system should automatically determine the number of patches, new representatives are added as long as their distance from any existing representative is larger than a pre-defined ε-distance.

3. Determining the decomposition: For each face, the distances to all the representatives within its connected component are calculated. Each face is assigned to the patch whose representative is the closest. This procedure creates a decomposition of the given model.

4. Re-electing the representatives: The goal of the algorithm is to minimize the function

$$F = \Sigma_p \Sigma_{f \in patch(p)} Dist_{fp}$$

where $Dist_{fp}$ is the shortest distance from a patch representative p to a face f belonging to the patch p represents. Therefore, $\Sigma_p \Sigma_{f \in patch(p)} Dist_{fp}$ is the sum on the shortest distances of all the faces to their patch representatives.

In order to converge to a solution, new patch representatives are being elected. This is done by minimizing the sum of the shortest distances from each representative to the faces which belong to the relevant patch. In other words, for each patch a new representative p_{new} is elected as the face (belonging to the patch) that optimizes the function $\min_p \Sigma_f Dist_{fp}$.

In practice, another option is to choose as a new representative the face whose center of mass is closest to the center of mass of the patch, as was done in the initialization step. Obviously, the complexity of the latter is much better. Moreover, our experiments have shown that the decompositions produced by this technique are often better.

If any patch had its representative changed in Step 4, the algorithm goes back to Step 3.

Lemma 3.1 All the faces which belong to a single patch are connected, thus the algorithm produces a legal decomposition.

4. Embedding Cylinder-like Patches

The result of the surface decomposition algorithm need not necessarily consist of disk-like patches, nor genus-zero polyhedra. In fact, it is often the case that the resulting patches are topologically cylinder-like. This is the case, for instance, with most of the patches of rotational sweep or general sweep objects, as well as with animals, furniture etc. For instance, most of the patches of the cheetah in Figure 1 are cylinder-like.

In [16] cylinder-like patches are "cut" so that they are transformed into disk-like patches. This, however, might distort the objects along the cutting line. Figure 2 illustrates this distortion by showing a morph of a bishop and a cylinder, both without their tops and bottoms (i.e., they are cylinder-like). Our algorithm avoids these distortions, as can be seen in Figure 3.

In [11] the user controls the global evolution of the deformation of cylinder-like models by specifying two skeletal structures. This works very well, however, it might not always be easy for the user.

We propose a novel scheme where the parameterization domain is an *ideal cylinder*, i.e. a cylinder with radius 1 and height 1. Our algorithm is based on a recursive divide-and-conquer scheme and consists of two steps: establishing boundary correspondence and cylinder embedding.

1. Establishing boundary correspondence: Given a pair of cylinder-like patches C_1 and C_2, a correspondence between their boundaries is first established. Let the boundaries of C_1 and C_2 be c_{11}, c_{12} and c_{21}, c_{22} respectively (see Figure 4).
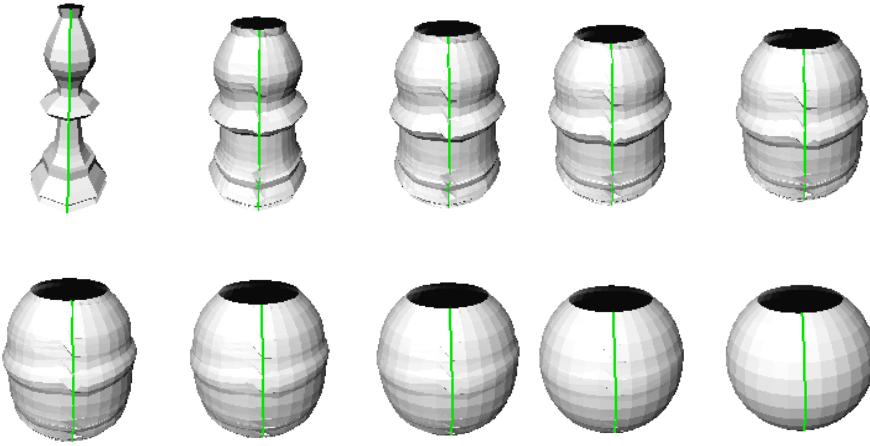
Figure 2: *Morphing a Bishop and a Cylinder – Cutting the cylinder along a cutting line*
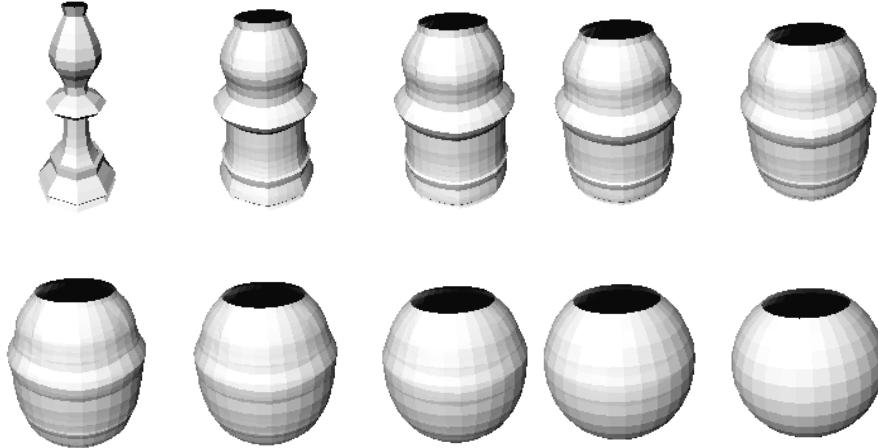

Figure 3: *Morphing a Bishop and a Cylinder – Our algorithm*

The user should first choose two sets of corresponding pairs of boundary vertices on the opposite boundaries of both patches, called the *anchor vertices*. These are the vertices $a_1 \in c_{11}$, $a_1' \in c_{12}$ and their corresponding vertices $a_2 \in c_{21}$, $a_2' \in c_{22}$ and also vertices $b_1 \in c_{11}$, $b_1' \in c_{12}$ and their corresponding vertices $b_2 \in c_{21}$, $b_2' \in c_{22}$ in Figure 4.

The anchor vertices are first mapped to the boundary curves of the parameterization cylinder. Then, the other boundary vertices are mapped according to their relative arclength distance from the anchor points on the patches.

The user is allowed to mark more *corresponding vertices* on the boundaries of the patches. These correspond-ing vertices are placed on the boundaries of the ideal cylinder according to the average of their arc-lengths relative to the overall lengths of the boundaries. In this case, all the other vertices are mapped according to their relative distances from their nearest marked corresponding vertices.

Since the number of boundary vertices on the given patches may differ, in order to establish a full correspondence between the boundaries, it is necessary to add vertices on the boundaries of the patches. These vertices are added by merging the boundaries, similarly to way sorted lists are merged (see Figure 5).

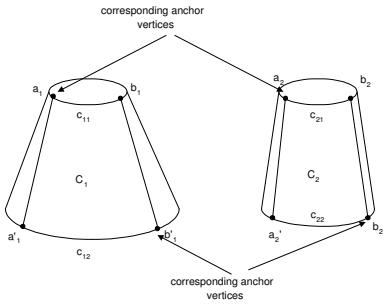*2. Cylinder embedding (full parameterization):* Mapping a
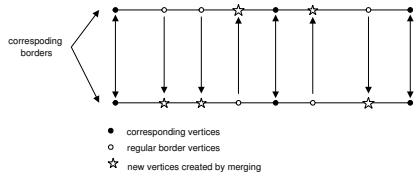
Figure 4: *The anchor vertices*

Figure 5: *Merging the boundaries*

cylinder-like patch to to an ideal cylinder is done using a divide and conquer approach. We begin with a couple of definitions.

Definition 4.1 *A dividing path* of a cylinder-like patch is a non self-intersecting polyline consisting of a subset of the surface edges, such that: (i) The path connects two vertices, one on each boundary of the cylinder-like patch, and (ii) The path does not intersect the boundary of the patch except at its end-vertices.

Definition 4.2 *A strip* is a subsurface of a cylinder-like surface confined by two non-intersecting dividing paths.

The major idea of the algorithm is to decompose the cylinder-like patch into a set of adjacent narrow *strips*. These strips can then be mapped onto the parameterization cylinder or onto a planar rectangle. It is done as follows.

Given a strip, the two middle vertices (one on each boundary) are found. These are vertices c and c' in Figure 6. The shortest path between these vertices is constructed, thus "vir-
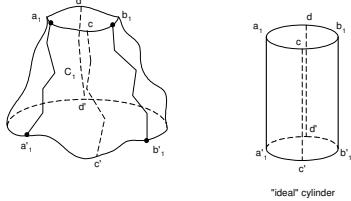

Figure 6: *A recursive step*

tually" cutting the strip into two sub-strips. In the shortest path algorithm, the weight of each edge is set to its length. Initially, the strips are defined by the anchor points.

If this shortest path is a legal dividing path and it does not intersect the strip's boundaries except at its end-vertices, the vertices which lie along it are mapped to straight lines connecting the corresponding boundary vertices, on the ideal cylinder or on a rectangle. (Recall that the boundary vertices were already mapped in Step 1.) The locations of the vertices along the straight line are found proportionally to their edge lengths, as illustrated in Figure 7.
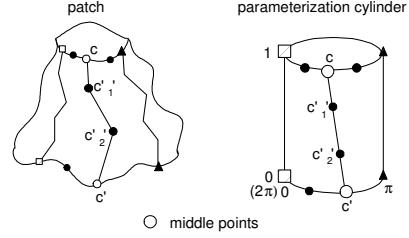

Figure 7: *Mapping the inner vertices of a dividing path*

Once the vertices along the dividing path are mapped, the algorithm recurses on the two sub-strips. The recursion is performed simultaneously on both strips, which belong to the two input patches. That is to say, simultaneously two dividing paths are found, one on each strip. Then, the vertices of the dividing paths are mapped to the same curve on the ideal cylinder. Finally, the sub-strips are the input to the next recursive calls.

At some stage of the recursion it becomes impossible to find a dividing path, i.e., each path between the two boundaries of the strip intersects previously-found dividing paths. In this case the recursion stops. Note that the recursion stops on both patches, and not only on the patch the stopping condition is satisfied.

Once the recursion ends, the given cylinder-like patches are compatibly divided into strips by a set of dividing paths. The next step of the algorithm is to map the inner vertices of each strip onto an appropriate strip on the parameterization domain. The mapping of each strip is done independently on the other strips. Each strip has a disk topology and can be mapped using any of the three parameterization methods for disk-like patches discussed in Section 5 (i.e., barycentric, harmonic or shape-preserving). Since the strips are typically narrow and almost planar, it is advantageous to use the shape preserving parameterization.

Finally, on that common domain, the vertex/edge/face connectivity graphs of the two patches are merged in a conventional way (e.g. [9]). This merge establishes the full correspondence between the given patches. The only difference between our merge step and previously proposed merge

steps is that before merging the inner vertices of the strips, it is necessary to merge the vertices of the dividing paths. This is done, as before, similarly to merging sorted lists (Figure 5).

Figure 8 demonstrates the mapping of a cylinder-like bishop (i.e., a bishop whose bottom and top were removed) onto the ideal cylinder parameterization domain.
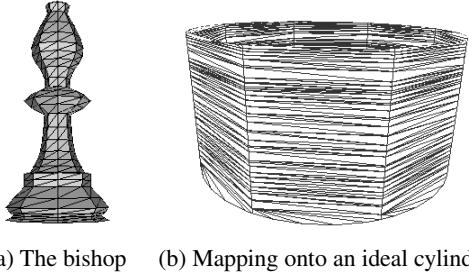

   (a) The bishop   (b) Mapping onto an ideal cylinder

Figure 8: *Mapping a cylinder-like bishop onto the parameterization cylinder*

5. Embedding Disk-like Patches

Various methods have been proposed for embedding disk-like patches onto the plane. *Harmonic mapping*, which attempts to preserve the aspect ratios of the edges, was proposed in [4, 8]. A different approach is to use convex combinations for parameterization. This can be done either by *barycentric mapping* [16] or by *shape-preserving mapping* as proposed in [5, 6]. In Barycentric mapping, every internal vertex is defined as the barycentre of its neighbors. In shape-preserving mapping, the convex combination is chosen such that if the triangulation is planar to start with, the location of the vertices will not change.

We experimented with the above three methods, used them to parameterize various disk-like patches, and compared their resulting embeddings. To measure the quality of the parameterization techniques, we define three criteria: preservation of areas, preservation of angles, and stretching. These distortion parameters aim at measuring how well the original geometry is maintained after the embedding.

Let F_i and F_j be two faces of the original surface. Let f_i and f_j be their corresponding faces on the parameterization domain. It is desirable that the ratio of the areas in the original surface is preserved in the parameterization domain. The following parameter measures it

$$area\_parameter = \left| \log\left(\frac{Area(F_i)/Area(F_j)}{area(f_i)/area(f_j)} \right) \right|.$$

The average value and the maximum value of the above parameter are calculated over all possible pairs.

The next parameter measures the preservation of the angles. For a given face F_i and its mapped face f_i, the angle

preservation parameter is calculated as:

$$angle\_parameter = \Sigma_{j=1}^{3} (\alpha_{ij} - \phi_{ij})^2 w_{ij}$$

where $\alpha_{ij}, j = 1, 2, 3$ are the angles of f_i, w_{ij} are weights and

$$\phi_{ij}(k) = \begin{cases} \beta_{ij}^k \frac{2\pi}{\Sigma_i \beta_{ij}^k} & \text{if } v_k \text{ is an interior vertex} \\ \beta_{ij}^k & \text{if } v_k \text{ is an boundary vertex} \end{cases}$$

where β_{ij}^k is the angle in the original mesh [15]. Again, both the average value and the maximum value are used to evaluate the parameterization.

The last parameter measures the stretch. i.e. how much the sampling direction in the parameterization domain is stretched on the mesh surface, and is described in [13].

Results of using the three parameterization methods are shown in Figure 9, where a head of an android and a head of a cheetah are parameterized.

Table 1 summarizes the distortion measures for the above two models. It can be seen that the harmonic mapping produces the best results in all three categories. Nevertheless, the shape-preserving method is very competitive. A major advantage of the shape-preserving method is that it is proven to always work. In addition, it should be the method of choice when the given patch is close to planar. Barycentric mapping, on the other hand, introduces large distortions.

6. Handling the Boundaries

Since the parameterization of each patch is done separately, it is important to assure continuity along the boundaries of the patches. Continuity is essential for avoiding distortions and cracks along the boundaries during the metamorphosis.

Recall that the parameterization methods we use (both for disk-like patches and for cylinder-like patches) are based on placing the boundary vertices prior to placing the inner vertices on the parameterization domain. Thus, placing the vertices lying on the boundaries of the patches should be done such that each vertex is mapped compatibly in the parameterization domain, for each patch for which it belongs. That is to say, the relative arc-length between the vertices should be maintained on the parameterization domain.

A boundary vertex is called a *branching point* if it is shared by at least three patches [16]. Given two 3D models decomposed compatibly, the number of branching points are the same for each pair of corresponding patches. The corresponding branching points are first found and mapped to the parameterization domain. In the case of disk-like patches, it can either be done by placing them uniformly on the boundary of a regular polygon [9] or by placing them proportionally to the average lengths of the boundary segments in the original 3D models [16], which is preferable. The other boundary vertices, residing between the branching points, are then mapped proportionally to their average arc lengths on the original models.
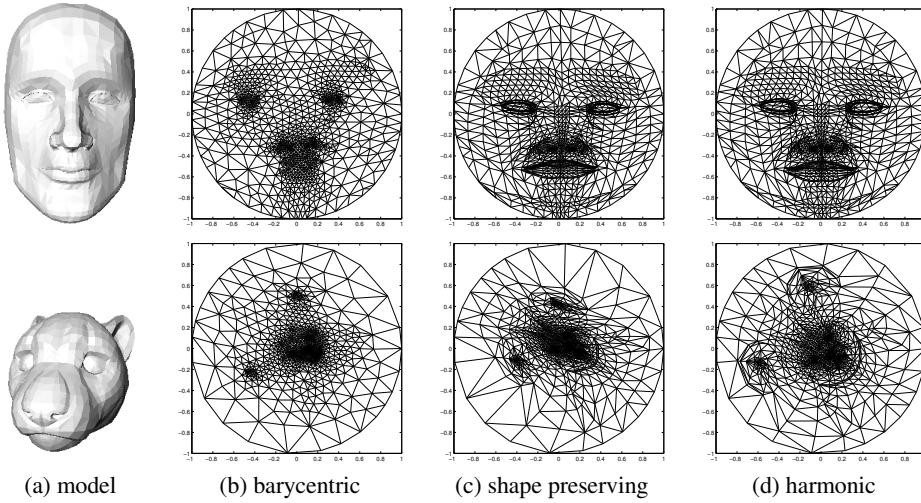
| (a) model | (b) barycentric | (c) shape preserving | (d) harmonic |

Figure 9: *Mappings using various parameterization techniques*

	Barycentric		Shape-preserving		Harmonic	
	average	maximum	average	maximum	average	maximum
Android	1.098	8.340	0.388	3.164	0.361	3.155
Cheetah	1.509	8.212	1.304	6.325	1.272	5.423

(a) Area preserving parameter

	Barycentric		Shape-preserving		Harmonic	
	average	maximum	average	maximum	average	maximum
Android	6.732	2149.693	0.036	6.686	0.012	1.302
Cheetah	0.861	194.189	0.145	13.632	0.008	0.494

(b) Angle preserving parameter

	Barycentric		Shape-preserving		Harmonic	
	average	maximum	average	maximum	average	maximum
Android	2.105	20.950	1.555	20.950	1.544	20.950
Cheetah	4.173	24.316	3.704	13.899	2.779	9.579

(c) Stretch parameter

Table 1: *Comparisons of the parameterization methods according to various distortion measures*

This method can be extended to handle the boundaries of cylinder-like patches. An additional requirement is that the anchor points, which are used for mapping the cylinder-like patches, should also be placed compatibly on the parameterization domain. We therefore handle the anchor points similarly to the way branching points are handled.

There is one special case that must be addressed. This is the case where a patch (either a disk-like or a cylinder-like) is entirely surrounded by a cylinder-like patch. In this case there are no branching points on the boundaries since only two patches share the whole boundary. See Figure 10. There are two possible ways to handle this case. Either the user chooses *user-defined branching points* on the source and on the target models, or the anchor points (used for mapping the cylinder-like patches) are used as branching points.

7. Results

Figure 1 shows a few snapshots from a movie that morphs a cheetah into a tiger. The images also show the decompositions. This example is selected because a cheetah and a tiger belong to the same family of animals and thus resemble each other. As such, the viewer is more likely to notice deformations in the sequence. This is exactly the case our algorithm
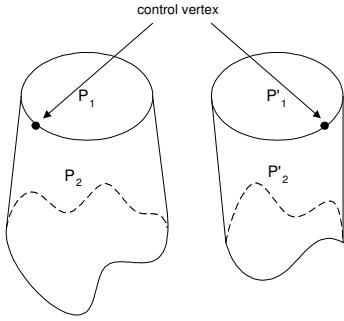
Figure 10: *Disk-like patches P_1 and P'_1 surrounded by cylinder-like patches P_2 and P'_2, respectively*

intends to handle well, since it can take advantage of the similarity of the decompositions.

Another example is shown in Figure 11 where a duck is being transformed into a dove. Again, the gradual changes are hardly noticeable. Figure 12 demonstrates the metamorphosis of an align into a dino-pet. To illustrate the final results, this example is shown without coloring the patches in different colors. See also the color section.

8. Conclusion

We have described in this paper an algorithm for establishing a correspondence for metamorphosis of polyhedral models. The algorithm is based on decomposing the input models into their inherent components. A full correspondence is found for each pair of compatible patches, while taking care to preserve continuity across the boundaries.

Our algorithm has two novel aspects. First, a new decomposition algorithm is presented. This algorithm lets the user control the number of outcome patches and avoids oversegmentation. Second, a new parameterization scheme is described, which embeds cylinder-like patches onto an ideal cylinder. The latter algorithm avoids distortions and well maintains the symmetry of the patches due to its divide-and-conquer nature. We also reviewed and compared some well-known embedding algorithms for disk-like patches.

We have shown a few results that demonstrate the quality of the metamorphosis produced by our algorithm. Of course, as is always the case with metamorphosis, the expectations and the evaluation of the metamorphosis are in the eye of the beholder.

References

1. Alexa M. *Merging polyhedral shapes with scattered features*, The Visual Computer, 16:26–37, 2000.

2. Biederman I. *Visual Object Recognition*, In An Invitation to Cognitive Science, Vol. 2: Visual Cognition. S. Kosslyn, D. Osherson, Eds. MIT Press, pp. 121-65, 1995.

3. Chazelle B., Dobkin D.P., Shouraboura N., Tal A. *Strategies for Polyhedral Surface Decomposition: An Experimental Study*, Computational Geometry: Theory and Applications, 7(4-5): 327-342, 1997.

4. Eck M., DeRose T., Duchamp T., Hoppe H., Lounsbery M., Stuetzle W. *Multiresolution Analysis of Arbitrary Meshes* ACM Siggraph, 1995, 173–182.

5. Floater M.S., *Parameterization and smooth approximation of surface triangulations*, Computer Aided Geometric Design 14:231–250, 1997. 231–250

6. Floater M.S., *Parametric Tilings and Scattered Data Approximation*, International Journal of Shape Modeling 4:165–182, 1998.

7. Gregory A., State A., Lin M.C., Manocha D., Livingston M.A. *Interactive surface decomposition for polyhedral morphing*, The Visual Computer, 15:453–470, 1999.

8. Kanai T., Suzuki H., Kimura F. *3D geometric metamorphosis based on harmonic maps*, Proceedings of Pacific Graphics '97, 97–104, October 1997.

9. Kanai T., Suzuki H., Kimura F. *Metamorphosis of arbitrary triangular meshes*, IEEE Computer Graphics and Applications, 20:62–75, 2000

10. Kent J.R., Parent R.E., Carlson W.E. *Shape transformation for polyhedral objects*, Computer Graphics, 26(2):47–54, July 1992.

11. Lazarus F., Verroust A. *Metamorphosis of cylinder-like objects* J. Visualization and Computer Animation, 8:3, 131–146, 1997.

12. Mangan A.P., Whitaker R.T. *Partitioning 3D Surface Meshes Using Watershed Segmentation*, IEEE Transactions on Visualization and Computer Graphics, Vol. 5 No. 4 (1999), 308-321.

13. Sander P.V., Snyder J., Gortler S.J., Hoppe H. Texture mapping progressive meshes. *ACM SIGGRAPH 2001*, pages 409–416, 2001.

14. Shapiro A., Tal A. *Polyhedron Realization for Shape Transformation*, The Visual Computer, 14 (8-9): 429–444, December 1998

15. Sheffer A., de Sturler E. Surface parameterization for meshing by triangulation flattening. In *Proceedings of 9th International Meshing Roundtable*, pages 161–172, 2000.

16. Zockler M., Stalling D., Hans-Christian Hege H.-C. *Fast and Intuitive Generation of Geometric Shape Transitions*, The Visual Computer, 16:5, 241–253, 2000.
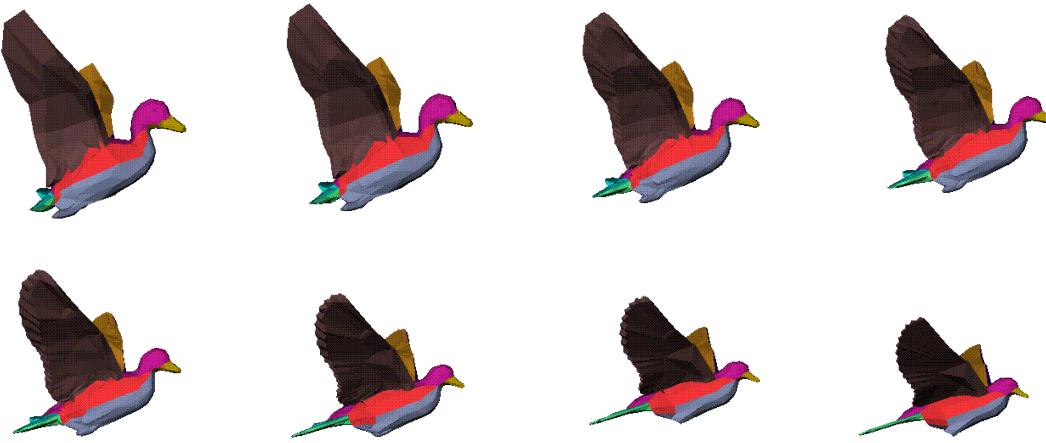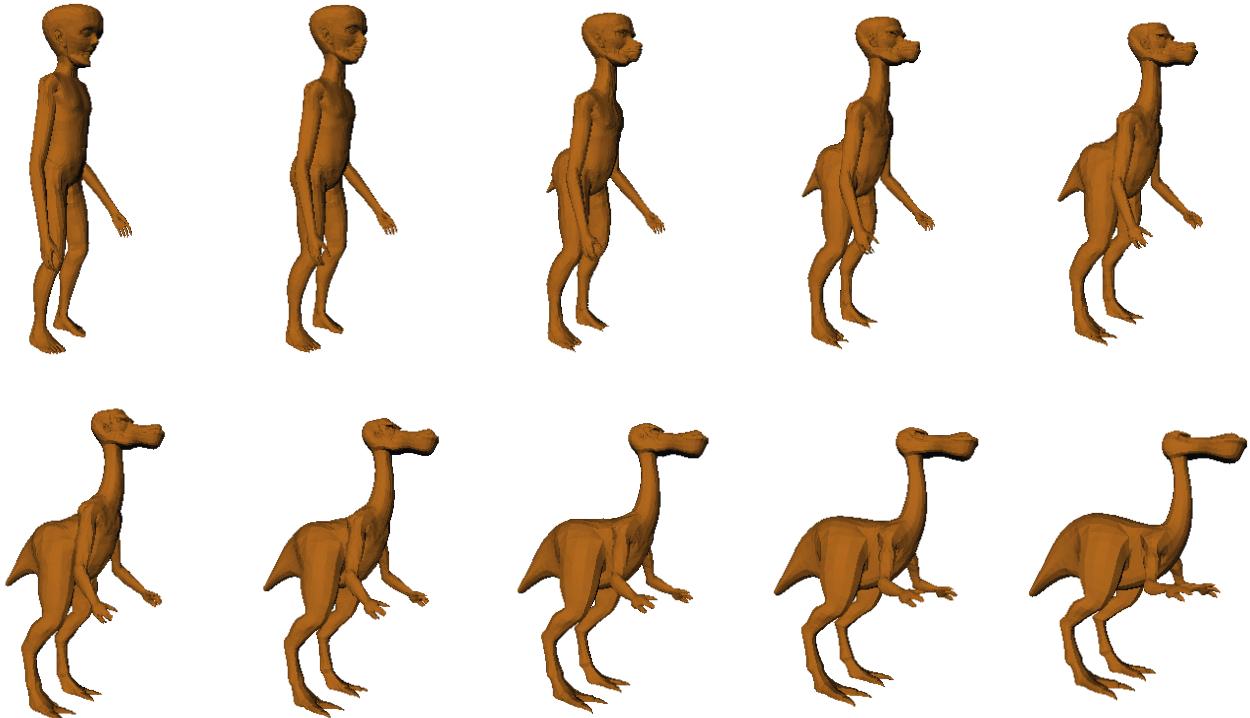
Figure 11: *Metamorphosis of a duck into a dove*

Figure 12: *Metamorphosis of an alien into a dino-pet*