Polyhedral Surface Decomposition with Applications*

EMANOIL ZUCKERBERGER† AYELLET TAL‡ SHYMON SHLAFMAN§
Department of Electrical Engineering
Technion – Israel Institute of Technology

Abstract

This paper addresses the problem of decomposing a polyhedral surface into "meaningful" patches. We describe two decomposition algorithms – flooding convex decomposition and watershed decomposition, and show experimental results. Moreover, we discuss three applications which can highly benefit from surface decomposition. These applications include content-based retrieval of three-dimensional models, metamorphosis of three-dimensional models and simplification.

Key words: *Polyhedral surface decomposition, retrieval of three-dimensional models, metamorphosis, simplification.*

1 Introduction

Decomposition of polyhedra into solids has been a lively topic of research in computational geometry. It is generally the case that a decomposition into convex solids is sought (e.g, [2, 3, 6, 9, 11, 12, 23]), since convex shapes are considered useful for representation, manipulation and rendering. Most algorithms proposed are hard to implement and debug and they all suffer a quadratic blow-up, which is often prohibitive in practice.

Real applications often do not need to partition the polyhedron itself but only its boundary. Although not as versatile as solid decompositions, boundary decompositions have several advantages, including simplicity of implementation and the complexity of the output which is always linear in size. This paper addresses the problem of decomposing a polyhedral surface into patches.

We discuss a couple of algorithms for decomposing polyhedral surfaces: a flooding convex decomposition algorithm and a watershed segmentation algorithm. We also show some results of applying the two algorithms and illustrate their utility. We showed in [10] that the minimization problem of surface decomposition is NP-complete. Thus, heuristics are necessary. Flooding heuristics refer to the incremental strategy of starting from a node in the dual graph and traversing this graph, collecting faces along the way as long as they do not violate a pre-defined property (i.e., convexity). Watershed segmentation algorithms segment a given object into "catchment basins" or "watersheds" [19].

A main emphasis of the current work is the applicability of surface decomposition. In the past, major candidate applications mentioned were collision detection and rendering, both can greatly benefit from the convexity of the patches. In this paper we look at other applications – ones which do not take advantage of convexity per se, but rather benefit from the decomposition itself (which as we will show need not necessarily be convex).

More specifically, the principle that underlies this work is that given an object, its patches and the way they relate to each other characterize this object and portray its distinctive features. This is supported by observations that the visual system tends to segment complex objects at regions of matched concavities [8]. Thus, the applications we choose to explore are ones that take advantage of the structure of the decomposition.

The first application we experimented with is searching a database of three-dimensional models (e.g., given in VRML) for objects similar to a given model. As VRML objects are becoming more popular on the World-Wide Web, this problem is expected to have many uses in computational biology, CAD, e-commerce etc. Since similar objects have similar decompositions, the structure of the decomposition can be used in the matching algorithm. We ran our search algorithm on a database containing 388 VRML objects and achieved good results.

The second application concerns metamorphosis of three-dimensional models. The idea here is to decompose the given models compatibly and to morph each pair of compatible patches. This algorithm has a couple of benefits. First, there is no need to make any assumption regarding the

*This research was supported by Technion V.P.R. fund – Bernstein Research Fund

†emanuel@tx.technion.ac.il
‡ayellet@ee.technion.ac.il, http://www.ee.technion.ac.il/~ayellet
§shymon@tx.technion.ac.il

topology of the given model. Any "polygon soup" model is valid. Second, since similar objects have similar decompositions, the morph sequence maintains the overall distinctive features of the models. Similar objects are important to handle correctly because these are the objects our visual system is most sensitive to. Most automatic correspondence algorithms either ignore similarity between models or require that the end–user carefully specifies the correspondence.

The final application is simplification of three-dimensional models. Given a model of n faces we generate another model of many fewer faces that captures the characteristics of the original model. The idea is to add a pre-processing step in which the model is decomposed into patches, and to apply a simplification scheme only within the patches, thus maintaining the distinctive features of the model.

Section 2 discusses algorithms for surface decomposition. Sections 3–5 present the applications in detail. In particular, database retrieval, metamorphosis and simplification are described in Sections 3, 4 and 5, respectively. We conclude in Section 6.

2 Surface Decomposition

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

In the sequel we present two algorithms. The first decomposes S into convex patches [10, 13]. We have shown in [10] that in this case, the minimization problem is NP-complete. Nevertheless the family of *greedy flooding* heuristics achieves good results. The second algorithm is a watershed decomposition algorithm which is inspired from segmentation algorithms used in image processing and does not necessarily decompose the model into convex patches [26, 19].

Convex decomposition – flooding algorithm: A polyhedral surface is called convex if it lies entirely on the boundary of its convex hull. Let G be the dual graph of the polyhedral surface S, where nodes represent faces and arcs join nodes associated with adjacent faces. The class of greedy flooding heuristics refers to the incremental strategy of starting from some node and traversing G, collecting faces as long as they form a convex patch. When traversal cannot be continued because convexity is violated, a new patch is started, and the traversal is resumed.

Heuristics in this class vary according to the method of traversal used. The simplest heuristics use either a DFS traversal or a BFS traversal. Note that the flooding scheme can be applied using different properties of the surface and is not limited to convexity.

In the case of convexity, the traversal of a patch cannot be continued if either a *local failure* or a *global failure* occurs. A local failure means that an edge at which a face is attached to the patch is concave. A global failure means that even though the patch is locally convex everywhere, some faces do not reside on the boundary the convex hull of the vertices of the patch.

For some applications it is vital to get only a handful of patches. However, when the given model is large, it is often the case that there are many "small" concavities, i.e. local failures by very small angles. The result of flooding models having many small concavities is a decomposition into many small patches, each consists of a few faces.

We propose two ways to get over this problem. First, the user can set a parameter ϵ, and an angle less or equal to $\Pi + \epsilon$ between adjacent faces is considered convex. Obviously, the resulting patches will not necessarily be convex. However, as we will see, meaningful decompositions will be attained.

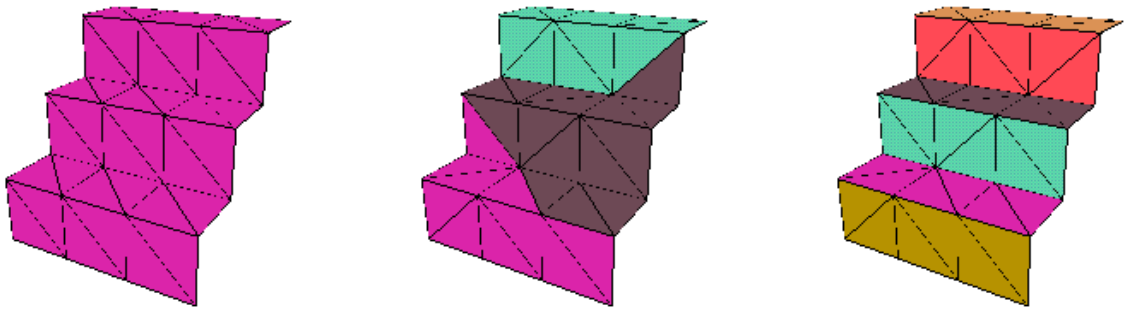
Second, we add a post-processing step in which small patches (area-wise) are merges with larger ones, thus decreasing the overall number of patches. A user-defined parameters, q, is introduced. Let A be the total area of the given object. A patch is considered *small* if its surface area is less than $q * A$.

The post-processing step proceeds as follows. The patches are first sorted by their surface area. Starting from the smallest patch, every *small* patch is considered in turn. A small patch is merged with its neighbor having the largest surface area, regardless of whether this patch should be merged as well. We will show in the sequel that the above post-processing step highly improves the quality of the resulting decomposition.

An alternative, faster, post-processing step will be described in Section 3, for a specific application where this new method is appropriate.

Watershed decomposition algorithm: The watershed segmentation algorithm was first proposed for image segmentation [26]. Let $h(x, y) : M \rightarrow R$ be a height function defined over the image domain M. A catchment basin is the set of points whose path of steepest descent ends in the same local minimum of h. Note that various height functions h can be used within this general framework. After locating the local minima of h, the algorithm associates catchment basins with the minima.

The algorithm proceeds as follows. First, all the local minima are found and labeled. Then, the flat areas are found. Flat areas can be either minima or plateaus. In the first case, the areas are labeled and treated like the local minima found in the first step. At this point, the steepest gradient descent is used to loop through the plateaus and allow each one to descend until a labeled region is encoun-



(a) Applying [19] to a uniform triangulation

(b) Applying [19] to a non-uniform triangulation

(c) Applying the new algorithm to a non-uniform triangulation

Figure 1. The Steps Model

tered. The remaining unlabeled vertices similarly descend until joining labeled regions. The idea is that of following a drop of water "downhill".

A major problem with this algorithm is that it results with over segmentation. To handle it, a post-processing merging step is applied where regions whose watershed depths are below a certain threshold are merged.

This basic algorithm has been extended to handle three-dimensional polyhedral models [19]. The major issue is how to choose the height function. In [19], the height function is defined over the polyhedral model. It is proposed to use various curvature estimations of the surface defined at each vertex of the model, as height functions.

A major problem with this scheme is that curvature estimations defined on mesh vertices get different values depending on the number of faces adjacent to each vertex. In other words, the topology of the model, and not merely its geometry, affects the final segmentation. For instance, in Figures 1(a)-(b), a model of steps is triangulated in two different ways. In Figure 1(a) all the internal vertices have the same height function values and thus only one patch results. In Figure 1(b), the internal vertices have different height function values, resulting with a decomposition into three patches. The most disturbing visual effect of this example is that adjacent planar faces end up in different patches. This algorithm works well for models created from range data, however general VRML models have less pleasing decompositions.

To get over this problem we propose to define a height function over the edges of the polyhedral mesh. Let the height function be $h = 1 - \cos(\alpha)$ where α is the dihedral angle defined on this edge (the angle between the faces adjacent to the edge). The basic watershed algorithm can now be applied to the edges (rather than to the vertices). To get the final decomposition of the surface, each face is associated with its edge with the lowest height. Thus, all adjacent planar faces end up in the same patch. Moreover, these flat regions are minima (having a height function value 0).

The result of running the watershed algorithm with the above height function on the steps model is demonstrated in Figure 1(c). Note that in this case, all the faces that reside on the same plane end up in the same patch. Thus, every step is decomposed into two patches, one horizontal and one vertical.

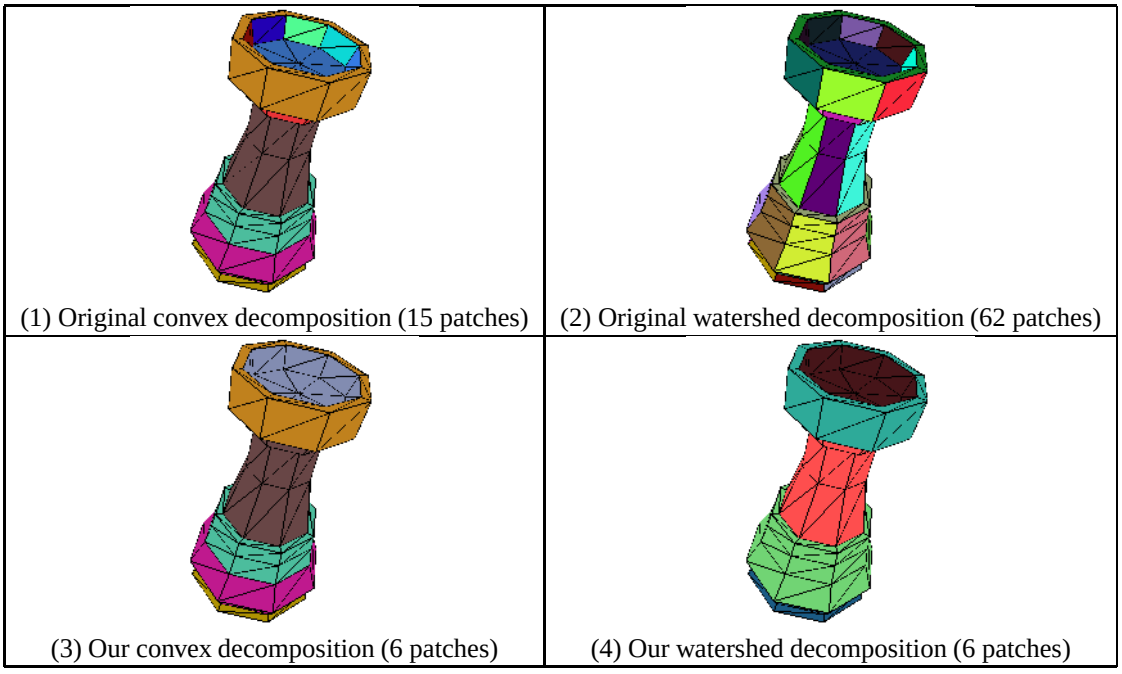
Finally, a similar post-processing step that was described for convex decomposition is applied here as well. In other words, small patches are merged with their larger neighboring patches, thus reducing the overall number of outcome patches.

We have tested various versions of the above algorithms on several objects. Figure 2 illustrates typical decompositions. It can be seen that our decompositions (Figures 2a(3)-(4) and 2b(3)-(4)) outperform previously proposed algorithms, producing less patches and resulting with more meaningful structures. Note also that the two algorithms, convex decomposition and watreshed decomposition, generate different decompositions even when the number of output patches is identical, as illustrated in Figures 2a(3)-(4).

In the following we discuss a few applications of polyhedral surface decomposition.

3 Application I: Content-Based Retrieval

Given a database of three-dimensional objects in a standard representation, such as VRML, and one specific object O, the goal is to retrieve objects similar to O from the database. As VRML objects are becoming more popular



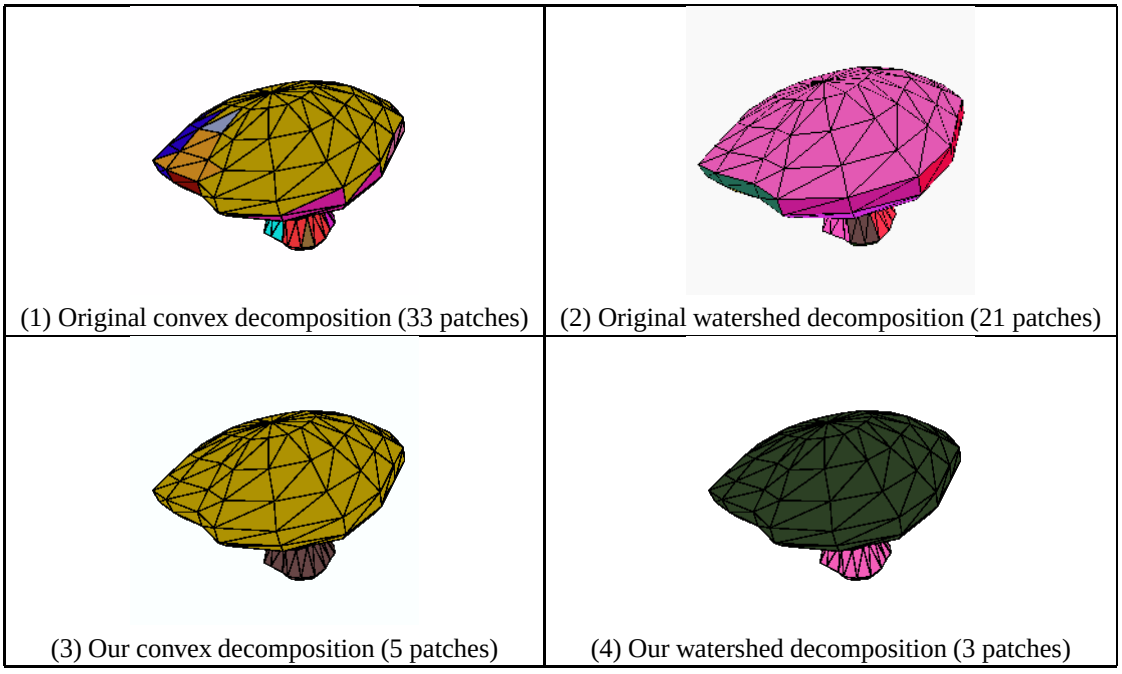
(1) Original convex decomposition (15 patches)

(2) Original watershed decomposition (62 patches)

(3) Our convex decomposition (6 patches)

(4) Our watershed decomposition (6 patches)

(a) Decompositions of a rook



(1) Original convex decomposition (33 patches)

(2) Original watershed decomposition (21 patches)

(3) Our convex decomposition (5 patches)

(4) Our watershed decomposition (3 patches)

(b) Decompositions of a mushroom

Figure 2. Decompositions

on the World-Wide Web, the retrieval of three-dimensional models is expected to have many applications.

Few papers have dealt with shape based retrieval of general polyhedral models. In [7, 21, 22] histograms of properties of the models are built and compared, sometimes using user-defined weight functions. Properties used include normals, angles, distances between points, colors, and material. In [14] shape moments are used to represent the objects, and retrieval is done within a relevance feedback framework.

We propose here a different approach. The idea is to decompose each three-dimensional model into a small number of meaningful patches. Then, the shape of each patch is evaluated and the relations between the patches are determined. Each such decomposition is represented by a attributed graph, which is viewed as a *signature* associated with each VRML object. This is the same idea as using a few key-words associated with a document as the document's signature. When searching a database for objects similar to a given object, we are basically searching for graphs similar to a given graph.

This approach is supported by psycho-physical observations, noticing that the visual system tends to segment complex objects at regions of deep concavities [8]. It is thus proposed that recognition of images can be done by recognizing the "components" found by segmentation. We claim here that similar ideas can be done in three dimensions, where surface decomposition replaces image segmentation. Moreover, the three-dimensional problem seems more informative than its two-dimensional counterpart, for a few reasons. First, no effects of reflections and shadows get in the way of segmentation. Second, three dimensional objects are seen "wholly" and do not suffer occlusions or self-occlusions. Third, recognizing the shape of each component (or patch) is easier since there is no projection involved.

Our algorithm for associating a "signature" with a three-dimensional model proceeds in three steps. First, the surface is decomposed as described in Section 2. Second, each patch is recognized as a basic shape. Third, the relationships between the patches are determined and a graph representing the model is constructed. Two points require explanation: how do we determine basic shapes and what are the meaningful relationships between patches that we use.

We consider four shapes as basic: a sphere, a cylinder, a cone and a plane. Given a patch, the goal is to determine for this patch on which type of basic shape it is more likely to reside. To do it, we sample the patch and solve a non-linear least-squares problem, which fits the sample points to the equations of each basic shape. The shape with the minimal fitting error is chosen as the basic shape the given patch has been drawn from.

Once the neighboring relationships between the patches are identified, we determine the relative size of neighboring patches. As observed in [8], the relative size of the components is of a vital importance. For instance, a small sphere adjacent to a big cylinder belongs to a different class of objects than that of a big sphere adjacent to a small cylinder.

To retrieve objects similar to a given object, we need to compare signatures. In our case, comparing signatures is related to graph isomorphism. The latter problem is a hard problem (not known to be in NP-complete [24]). It is thus important that the surface decomposition algorithms used do not produce many patches. Recall that we adjusted the decomposition algorithms accordingly, by ignoring small concavities in the case of convex decomposition, and by controlling the depth considered for merging in the case of watershed segmentation.

Moreover, in both cases we added a post-processing step in which small patches (area-wise) are merged into large patches. In fact, we can even apply a simpler post-processing step. Rather than merging small patches with large patches, we can just ignore, or eliminate, the small patches. This approach is supported by psycho-physical observations that "recognition can be fast and accurate" even if "only two or three geons of a complex object are visible" [8]. Thus, we can remove small patches and not consider them for similarity at all. Our experiments have shown that this is a very good option.

For the latter post-processing step, a patch is defined *small* if two conditions hold: (1) its surface area is less than $q*A$, where A is the surface area of the whole object, and (2) after eliminating the patch, the surface area of the remainder patches is greater than $p * A$. Both p and q are user-defined parameters. As before, the patches are first sorted. Then, *small* patches are eliminated in ascending order until either condition is not satisfied.

Attaining only a few patches is vital, because only for small graphs there are heuristics that solve sub-graph isomorphism. In our case, we use the Graph Matching Toolkit which finds subgraph isomorphism of attributed graphs [20].

To test our retrieval algorithm, we ran it on a database consisting of 388 VRML objects. We experimented with four versions of the algorithm: (1) convex decomposition with a post-processing step in which small patches are eliminated; (2) convex decomposition with a post-processing step in which small patches are merged with neighboring large ones; (3) watershed decomposition with a post-processing step in which small patches are eliminated; (4) watershed decomposition with a post-processing step in which small patches are merged with neighboring large ones. Tables 1– 2 and Figures 3– 4 demonstrate some of our results.

Table 1 shows the top 20 results of a search for objects similar to a cat, where the good results are emphasized. The class of four-legged animals consists of 18 animals. As can

be seen, all four algorithms exhibit good results, retrieving $11-16$ animals. Among the four algorithms, the watershed algorithm, where small patches are merged with larger one, achieved the best results.

Similar results can be observed in Table 2 which shows the top 20 objects retrieved when searching the database for objects similar to a human. The class of humans consists of 19 members. Again, all four algorithms exhibit good results, finding $12-18$ humans. Among the four algorithms, the watershed algorithm, where small patches are eliminated, achieved the best results (retrieving 18 humans). However, the other version of the watershed algorithm, as well as convex decomposition with merge, are almost as good, retrieving 17 similar objects.

Figure 4 demonstrates an important advantage of our algorithm. Since objects are decomposed into their meaningful parts prior to their comparisons, the objects can be in many possible positions and yet considered similar. For instance, the human can be sitting or standing, can fold her legs and arms or not etc. The graphs, representing the decompositions, are similar in all these cases. This is to be contrasted with geometric similarity, where a sitting figure would be considered dissimilar to a standing one.

To conclude. the algorithms are competitive and all of them perform well. However, the Watershed algorithm is slightly better.

4 Application II: Metamorphosis

A common approach to find a correspondence between two given polyhedra for metamorphosis is to look for a common embedding of the topologies of the given polyhedra. For instance, in [16] the polyhedra are projected onto the plane using harmonic mapping. In [17], the polyhedra are projected onto the surface of a sphere. In [27], the polyhedra are projected onto the surfaces of convex polyhedra. In all these cases, the projection is done in order to facilitate the merge of the $1-$skeleton graphs of the polyhedra.

This approach has a couple of drawbacks. First, fine correspondence is hard to achieve, since the projection is global. This can result in visible artifacts when features in one object are transformed into completely different features on the other. To overcome this shortcoming, it is proposed in [1] that the user specifies some corresponding feature points on the polyhedra's surfaces. The algorithm then tries to compute an overlay of the two $1-$skeleton graphs of the polyhedra, taking this correspondence into account. Computing this overlay, however, is not always possible, nor is it easy to know in advance when this is the case. In addition, specifying many points can become a burden on the end–user, and specifying only a handful of points is often insufficient.

The second disadvantage of the general approach is the necessary assumption that the input models are polyhedra (rather than "polygon soups"), and often even genus-0 polyhedra. This assumption cannot be made for arbitrary models found in VRML libraries over the web. Many objects are rarely two-manifolds and often have cracks and intersecting triangles.

To overcome these shortcomings, we propose here a different solution for finding a correspondence for metamorphosis. First, the given polyhedra's surfaces are decomposed into patches. The decompositions are then transformed so that their connectivity graphs become isomorphic. Finally, a common parameterization is found for each pair of corresponding patches. This parameterization is the correspondence we are seeking.

Since the objects are decomposed, they can have any topology. They need neither have genus zero, nor even be two-manifolds. Moreover, rather than carefully specifying corresponding vertices, the boundaries between the patches are considered the corresponding features. This guarantees that similar parts (e.g., organs) are transformed to each other (i.e., a head is transformed into a head and a leg into a leg).

Note that after applying a surface decomposition algorithm, we get two sets of patches whose number is not necessarily equal and whose connectivity graphs are not necessarily isomorphic. We built a tool that lets the user easily and quickly "fix" the resulting patch configuration. This tool allows the user to divide patches into smaller pieces, to unite existing patches, and to move faces from one patch to a neighboring patch. Since usually there are not many patches involved, very little manual work is needed.

Once the polyhedra are decomposed into isomorphic sets of patches, the problem of finding a global parameterization is broken down into finding a parameterization for each pair of corresponding patches. Various parameterization methods have been discussed in the literature. They include barycentric parameterization (e.g., [17, 27]), harmonic parameterization [16] and shape-preserving parameterization [15]. In all these cases, the boundary vertices are first placed on a two-dimensional polygon. Only then can the inner vertices be placed according to the specific method used.

In Figure 5 we show a few snapshots from a movie that morphs a cheetah into a tiger. The images are shown along with the decompositions. This example has been 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 intends to handle well. When the models are alike, the algorithm can take advantage of the similarity of their decompositions. As can be seen, the intermediate results look very convincing. In fact, in the movie, where more frames are used, the gradual changes are hardly noticeable. This movie, as

	Convex decomp., eliminate		Convex decomp., merge		Watershed decomp., eliminate		Watershed decomp., merge	
Rank	Object	Distance	Object	Distance	Object	Distance	Object	Distance
1.	**Cat2**	0.00	**Cat2**	0.00	**Cat2**	0.00	**Cat2**	0.00
2.	**Calf**	0.40	**Cow1**	0.44	**Calf**	2.11	**Horse2**	1.98
3.	**Cow1**	1.69	**Calf**	0.47	**Horse2**	2.87	**Tiger3**	2.52
4.	**Tiger2**	2.41	Pump	1.67	**Cow2**	4.02	**Camel2**	3.76
5.	**DogSt1**	2.48	**Tiger2**	2.03	**Tiger3**	4.54	**Calf**	3.91
6.	**DogSt2**	4.12	**DogSt2**	3.36	Shuttle	4.62	**Cow**	4.09
7.	Chicken	4.68	**DogSt1**	3.61	**Camel2**	4.78	**Cow2**	4.11
8.	Knifepr	4.76	**Goat2**	3.67	P51	4.87	Fontanin	4.39
9.	Part02	4.82	Rocktshp	3.88	**Cow**	4.89	**Cow1**	4.63
10.	**Cow2**	4.87	**Horse2**	4.22	Pump	5.02	**Goat2**	4.81
11.	**Horse1**	4.92	Chicken	4.67	**Deer**	5.15	**DogSt2**	5.09
12.	Ship2	5.30	**Cow2**	4.72	**Camel1**	5.71	**Camel1**	5.13
13.	Shuttle	5.39	**Camel2**	4.85	**Goat2**	5.72	P51	5.66
14.	Knifecl	5.55	Part01	4.87	**Cow1**	6.23	Shuttle	5.78
15.	**Horse2**	5.72	Pump1	4.94	TennisSh	6.26	**Deer**	5.97
16.	Pump	6.14	**Camel1**	5.09	**Donkey**	6.68	**DogSt1**	6.81
17.	**Camel2**	6.19	**Cow**	5.15	**Horse**R	7.58	**Tiger2**	7.20
18.	Sandal1	6.21	Sandal1	5.17	**DogSt2**	7.70	Excalibe	7.24
19.	**HorseR**	6.28	Shuttle	5.24	Pump1	7.78	**HorseR**	7.46
20.	Fontanin	6.49	**Horse1**	5.31	Fontanin	7.96	**Donkey**	7.48

Table 1. Retrieval of top 20 objects similar to Cat2

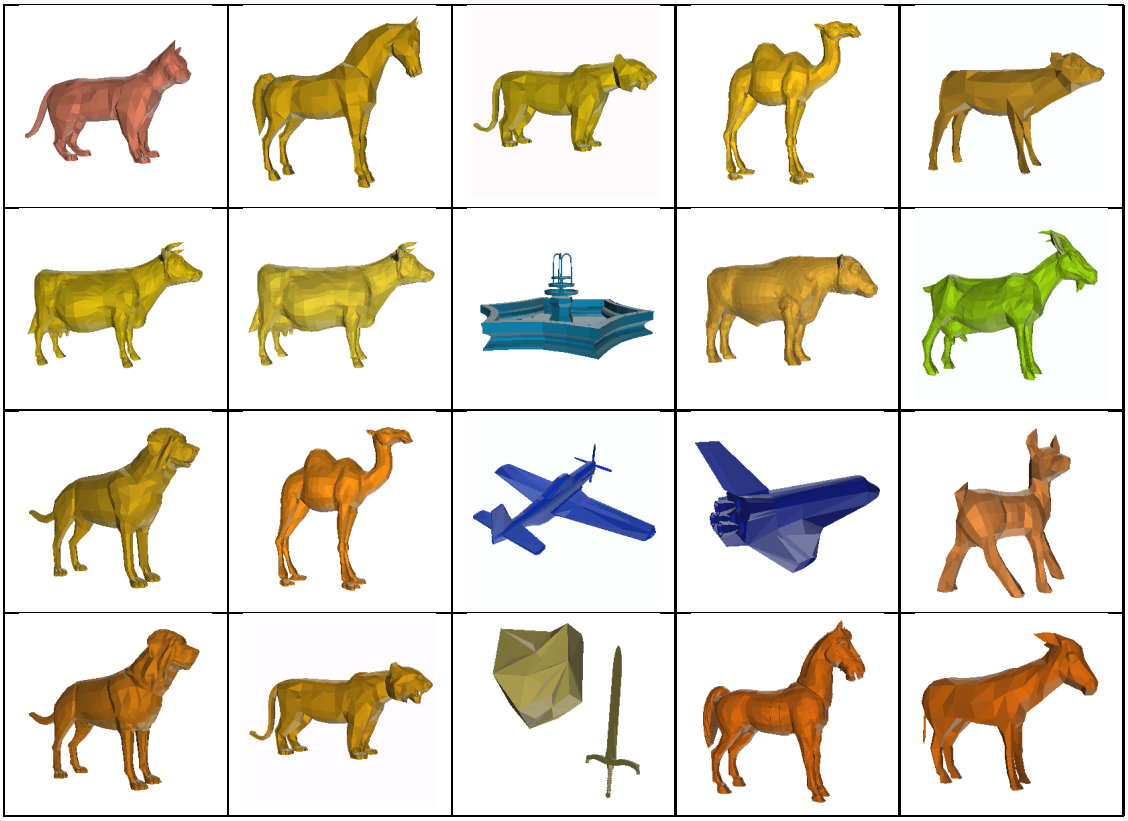


Figure 3. Retrieval of top 20 objects similar to Cat2 – watershed, merging patches

	Convex decomp., eliminate		Convex decomp., merge		Watershed decomp., eliminate		Watershed decomp., merge	
Rank	Object	Distance	Object	Distance	Object	Distance	Object	Distance
1.	**Woman2**	0.00	**Woman2**	0.00	**Woman2**	0.00	**Woman2**	0.00
2.	**Woman7**	0.02	**Woman8**	0.20	**Woman4**	0.00	**Child7y**	0.32
3.	**Woman4**	0.22	**Man7**	0.43	**Woman6**	0.00	**Woman7**	0.60
4.	**Woman5**	0.30	**Woman3**	0.47	**Woman3**	0.06	**Man1**	0.61
5.	**Woman6**	0.37	**Man3**	0.48	**Woman5**	0.06	**Man4**	0.61
6.	**Man5**	0.37	**Woman6**	0.52	**Woman7**	0.06	**Man5**	0.62
7.	**Woman3**	0.52	**Man6**	0.64	**Man3**	0.15	**Woman5**	0.64
8.	Cat2	0.97	**Man4**	0.65	**Man5**	0.15	**Child9y**	0.78
9.	Reel	1.42	**Man5**	0.66	**Woman8**	0.16	Cat2	0.93
10.	**Woman8**	1.50	**Woman5**	0.67	**Man1**	0.16	**Child3y**	1.02
11.	**Man7**	1.71	**Child3y**	0.76	**Man6**	0.16	**Child5y**	1.02
12.	Flintcar	2.21	**Child5y**	1.01	**Man4**	0.17	**Woman6**	1.14
13.	DogSt4	2.52	Billboar	1.65	**Man7**	0.18	**Man7**	1.15
14.	Toydog	2.59	**Child9y**	1.89	**Child3y**	0.48	**Man6**	1.15
15.	Tabasco	2.80	**Woman4**	1.91	**Child9y**	0.80	**Woman4**	1.55
16.	Chair11	2.99	Newtable	3.08	**Child5y**	0.98	**Woman8**	1.55
17.	**Man3**	3.14	Stool2	3.09	**Man2**	1.62	**Woman3**	1.55
18.	Ofbldng1	3.16	**Woman7**	3.12	Toydog	1.86	**Child12y**	1.72
19.	**Man4**	3.20	**Man1**	3.23	Cat2	2.02	DogSt4	1.81
20.	**Man6**	3.22	**Child12y**	3.27	**Child12y**	2.16	Manhand	2.11

Table 2. Retrieval of top 20 objects similar to Woman2

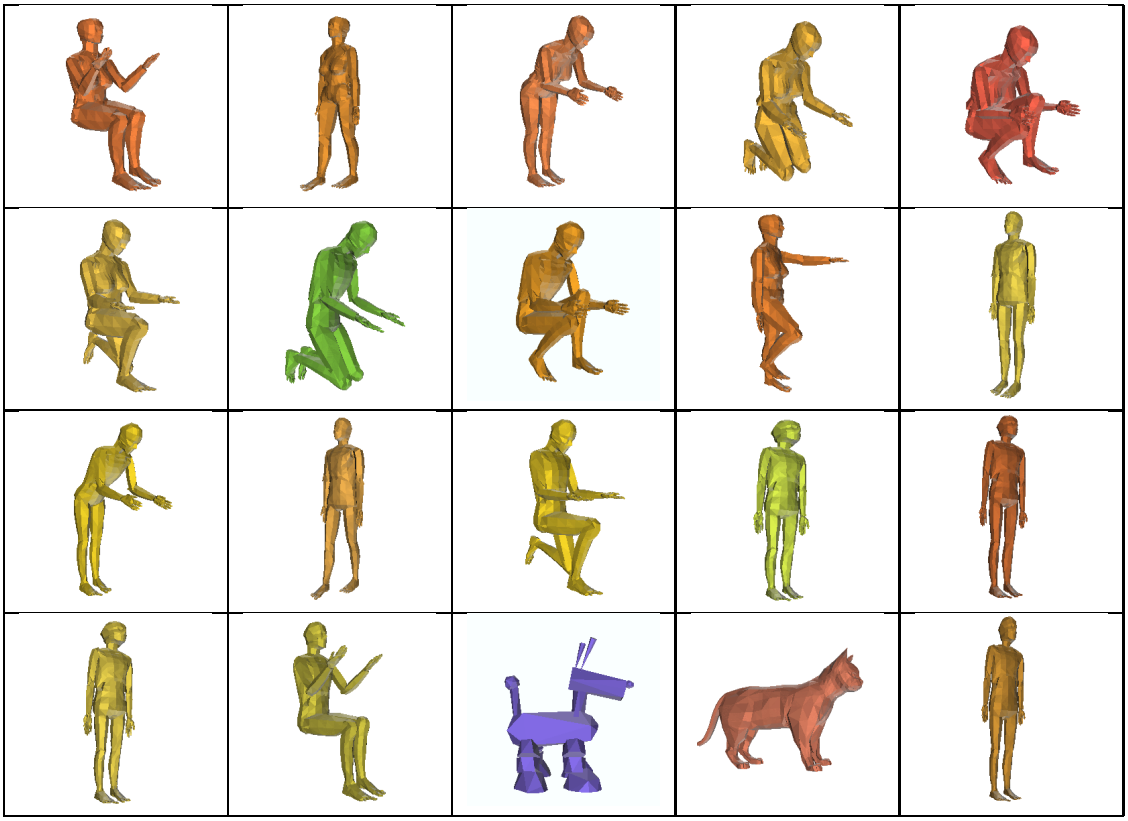


Figure 4. Retrieval of top 20 objects similar to Woman2 – watershed, eliminating small patches

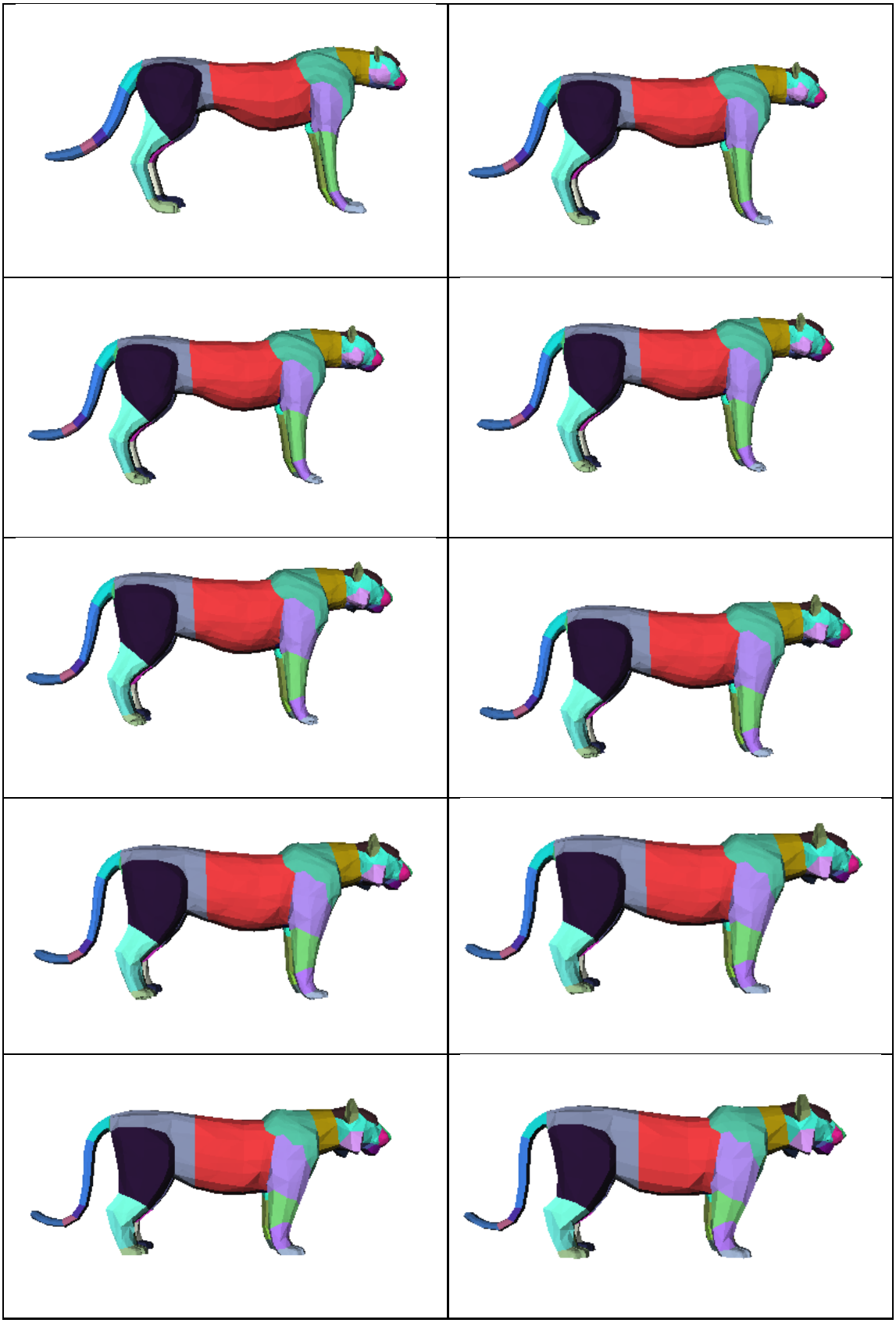


Figure 5. Metamorphosis of a cheetah and a tiger

well as others created by the algorithm, can be found in http://www.ee.technion.ac.il/~ayellet/Morph-movies.

5 Application III: Simplification

Polygonal surface simplification seeks to reduce the size of three-dimensional models in order to speed up rendering or other three-dimensional applications [18].

Decimation is one of the common simplification techniques [25]. The main idea is to iteratively remove vertices and re-triangulate the resulting holes, thus reducing the number of vertices and faces of the model. This operation is repeated until a desired simplification is achieved. Decimation has a few desirable properties. It is generally fast; it is topology tolerant; and it works on non-manifold models, which are common on the World-Wide Web.

Decimation techniques need to handle two sub-problems. First, a criterion for selecting candidate vertices for removal should be determined. Second, a triangulation scheme should be selected and applied to the holes.

In the current work we build upon the general decimation technique. The main departure of our approach from previous work is that we solve the above sub-problems in a *shape dependent* manner. More specifically, in a pre-processing step the given model is decomposed into patches, as described in Section 2. Now, decimation is applied within the patches only, thus maintaining the distinctive features of the model, which are represented by the patch graph structure.

Picking a vertex for removal is done in [25] by choosing the vertex whose distance to a plane which is the average plane of the neighboring polygons is minimal. We suggest, instead, to select a vertex whose distance to the shape it resides on is minimal. Recall that each patch is tagged as some basic shape, thus we can remove a vertex whose distance to that shape is minimal. For instance, if the patch is determined to reside on a sphere $S(C_x, C_y, R)$ with a center (C_x, C_y) and a radius R, the vertex for removal will be such that its distance to S is minimal among all the internal vertices of the patch. Note that this scheme deviates form previous schemes in two manners. First, since simplification is done only within a patch, vertices which reside on curves which "characterize" the model (i.e., the curves of the boundaries of the patches) cannot be selected. As a result, the overall structure of the model is maintained. Second, the criterion for vertex selection aims at maintaining the global shape of each patch.

The second sub-problem of decimation is the triangulation of the resulting hole. In general, two families of algorithms have been used. The first strategy projects the polygon to two dimensions, constructs a triangulation in the plane (i.e., by Delaunay triangulation), and maps this triangulation back to three dimensions. The advantage of this strategy is its efficiency. The drawback is that it is not always feasible. The other strategy finds a triangulation in three dimensions using dynamic programming [4]. Flexibility is the main advantage of dynamic programming, since many optimization functions can be used within the general algorithm. A common optimization function is a minimization of the surface area [5]. For the shape-dependent scheme that we are pursuing, the distance from the optimal basic shape can be utilized. For instance, if the patch is tagged as a sphere $S(C_x, C_y, R)$, the optimization function used within dynamic programming minimizes shape distortion of the simplified patch to S. The optimization function we use is basically a Hausdorff distance of the simplified patch to the basic shape.

$$F(Shape, Surface) = \max_{v \in Shape} \min_{w \in Surface} dist(v, w)$$

where v, w are points on the the basic shape and on the simplified surface, respectively.

To demonstrate the results of the algorithm, we show in Figure 6 the simplification of a horse model, starting with a model containing 39697 faces in Figure 6(a), and ending with a model simplified to 20% of the original size. The latter model contains 7936 faces and is shown in Figure 6(f). The original model is decomposed into 104 patches (on average, 381 faces per patch), each drawn in a different color. Note that the decomposition does not change throughout the simplification process.

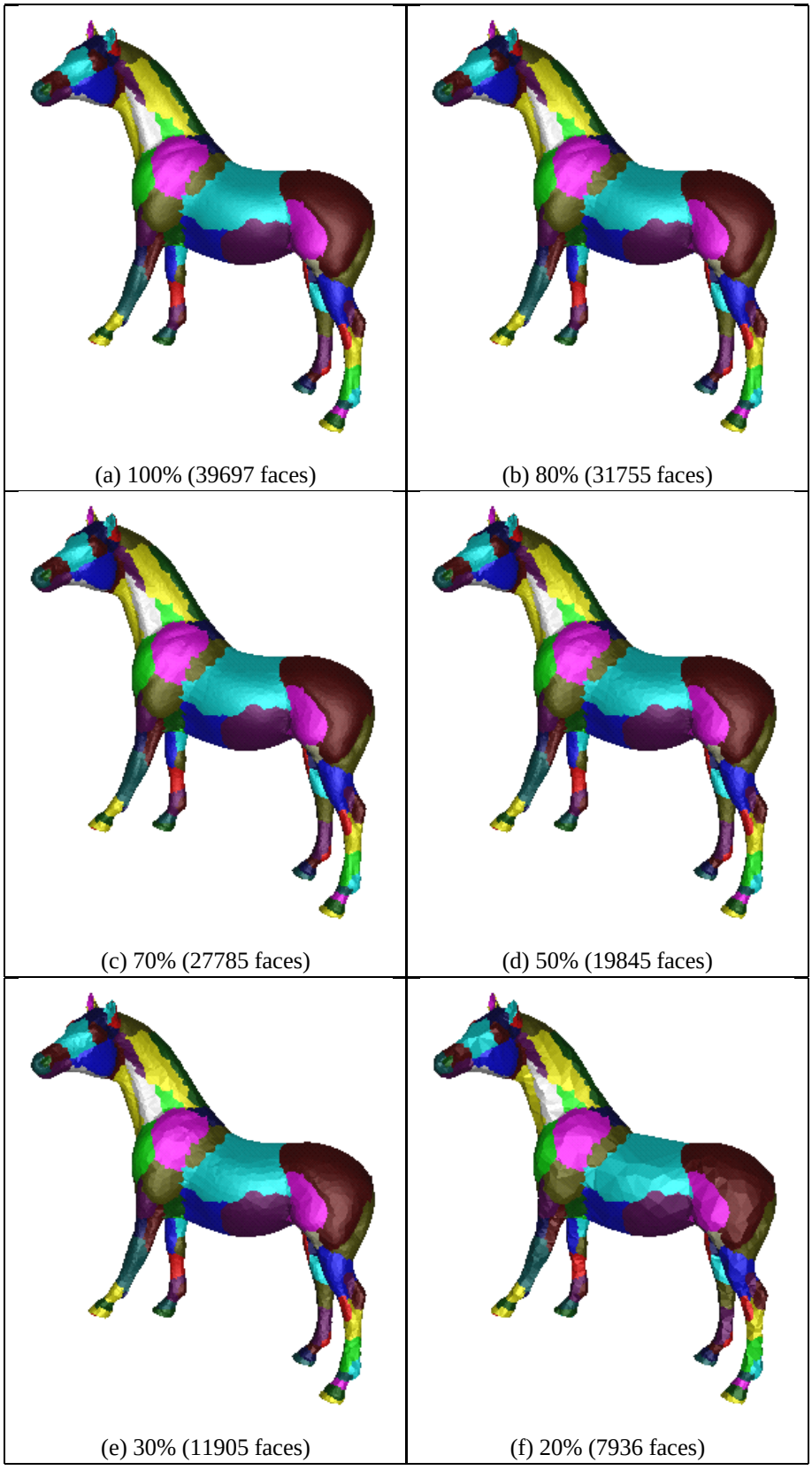
6 Conclusion

This paper has addressed the problem of decomposing a polyhedral surface into patches. The main idea that underlies this work is that decomposition results with the inherent components of the model. Various applications can take advantage of the structure of the decomposition rather than of the shape of the patches, as was proposed in the past.

In particular, we experimented with three applications of polyhedral surface decomposition. The first application is a retrieval of three-dimensional models based on shape similarity. We achieved good results running our algorithm on a database of 388 VRML objects.

The second application concerns correspondence for metamorphosis of three-dimensional models. Our algorithm handles the problem of lack of fine correspondence and gets over the restrictions on the allowed topologies of the given models. The idea is to decompose the objects compatibly and then parameterize each patch separately. We have shown that very pleasing morph sequences are obtained.

The final application deals with simplification of three-dimensional models. Using decimation only within a patch, allows us to achieve a large degree of simplification while maintaining the distinctive features of the given model.



(a) 100% (39697 faces) (b) 80% (31755 faces)

(c) 70% (27785 faces) (d) 50% (19845 faces)

(e) 30% (11905 faces) (f) 20% (7936 faces)

Figure 6. Simplification - from 39697 faces to 7936 faces

We consider a couple of directions for future research. First, we are looking at other decomposition algorithms. Second, we are considering further applications of surface decomposition. Possible applications include collision detection, surface re-parameterization and model modification.

Acknowledgments:

We would like to thank Vadim Rogol for implementing the simplification algorithm.

References

[1] Alexa, M. *Merging Polyhedral Shapes with Scattered Features*, The visual computer, 16(1), 2000, 38–46, .

[2] Aronov, B., Sharir, M. *Castles in the air revisited*, Disc. Comput. Geom 12 (1994), 119–150.

[3] Bajaj, C.L., Dey, T.K. *Convex decompositions of polyhedra and robustness*, SIAM J. Comput., 21 (1992), 339–364.

[4] Barequet, G., Shapiro, D., Tal, A. *Multi-Level Sensitive Reconstruction of Polyhedral Surfaces from Parallel Slices*, The Visual Computer, Vol. 16, No. 2, March 2000, 116-133.

[5] Barequet, G. and Sharir, M. *Piecewise-linear interpolation between polygonal slices*, Computer Vision and Image Understanding, Vol. 63 (2), pp. 251-272, March 1996

[6] Bern, M. *Compatible tetrahedralizations*, Proc. 9th Ann. ACM Symp. Comput. Geom. (1993), 281–288.

[7] Besl P. Triangles as a primary representation. Object Recognition in Computer Vision, LNCS 994:191–206.

[8] 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.

[9] Chazelle, B. *Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm*, SIAM J. Comput., 13 (1984), 488–507.

[10] 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.

[11] Chazelle, B., Palios, L. *Triangulating a nonconvex polytope*, Disc. Comput. Geom., 5 (1990), 505–526.

[12] Chazelle, B., Palios, L. *Decomposition algorithms in geometry*, in Algebraic Geometry and its Applications, C. Bajaj, Ed., Chap.27, Springer-Verlag, 1994, pp. 419–447.

[13] Chazelle, B., Palios, L. *Decomposing the boundary of a nonconvex polytope*, Proc. 3rd Scandinavian Workshop on Algorithm Theory (1992), 364–375.

[14] Elad, M., Tal, A. and Ar, S. *Content Based Retrieval of VRML Objects - An Iterative and Interactive Approach* EG Multimedia, September 2001.

[15] Floater, M.S. *Parameterization and smooth approximation of surface triangulations*, Computer Aided Geometric Design 14 (1997) 231-250.

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

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

[18] Luebke, D.P. *A Developer's Survey of Polygonal Simplification Algorithms*, IEEE Computer Graphics and Applications, Vol. 21 No. 3 (2001), 24-35.

[19] 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.

[20] Messmer, B.T. *GMT - Graph Matching Toolkit*, PhD Thesis, University of Bern, 1995.

[21] Osada R., Funkhouser T., Chazelle B. and Dobkin, D.P. Matching 3D Models with Shape Distributions, Shape Modeling International, May, 2001.

[22] Paquet E. and Rioux M. *A Content Based Search Engine for VRML Databases*, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1998.

[23] Ruppert, J., Seidel, R. *On the difficulty of triangulating three-dimensional non-convex polyhedra*, Disc. Comput. Geom., 7 (1992), 227–253.

[24] Schoning, U. *Graph isomorphism is in the low hierarchy*, Journal of Computer and System Sciences, 37:312–323, 1988.

[25] Schroeder, W.J., Zarge, J.A., Lorensen, W.E. *Decimation of Triangle Meshes*, Computer Graphics, Volume 25, No. 3, (Proc. SIGGRAPH '92), July, 1992.

[26] Serra, J.P. *Image Analysis and Mathematical Morphology*, London: Academic Press, 1982.

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