Shuffler: Modeling with Interchangeable Parts

Vladislav Kraevoy Dan Julius Alla Sheffer

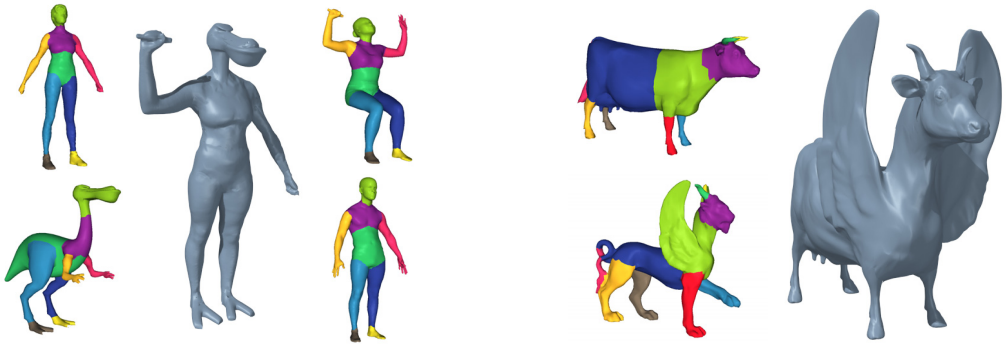

Figure 1: Interchangeable part correspondences computed by our system. An alien and a flying cow generated by shuffling the parts.

Abstract

Many man made and natural objects are easily classified into families of models with a similar part-based structure. Example families include quadrupeds, humans, chairs, and airplanes. In this paper we present *Shuffler* – a modeling system that automates the process of creating new models by composing interchangeable parts from different existing models within each family. Our system does not require the users to perform any geometric operations; they simply select which parts should come from which input model, and the system composes the parts together. To enable this modeling paradigm, Shuffler precomputes the interchangeable parts across each input family of models by first segmenting the models into meaningful components and then computing correspondences between them. We introduce two new algorithms to perform the segmentation and to establish part correspondences that can also be used for many other applications in computer graphics.

CR Categories: I.3.5 [Computer Graphics]**:** Computational Geometry and Object Modeling - Curve, surface, solid, and object representations

Keywords: Modeling – Digital Geometry Processing; Modeling – Shape Blending/ Morphing

1 Introduction

The creation of novel 3D content is one of the major bottlenecks of modern computer graphics. Despite recent advances in geometry processing, modeling remains time consuming and requires both technical and artistic expertise. Hence, much of the emphasis in recent research is on simplifying model creation. One approach is to provide simpler interfaces, such as sketching tools [Igarashi et al., 1999] which are good for modeling simple shapes. Another recent trend focuses on reuse of existing models by providing robust editing tools [Sorkine et al. 2004; Funkhouser et al., 2004; Yu et al., 2004].

Reuse techniques take advantage of the large set of pre-existing 3D data and allow for the creation of models with rich detailing. Most existing reuse techniques require a considerable amount of user interaction. Blending tools, such as Schreiner et al. [2004] and Kraevoy et al. [2004], require users to specify feature correspondences. Composition tools, such as Yu et al. [2004], typically require the users to cut the relevant parts from the input models and manually reposition them with respect to one another. To reduce the amount of work, researchers proposed tools that simplify the cutting interface [Lee et al., 2005] or the positioning process [Hassner et al., 2005]. Funkhouser et al. [2004] provide a tool that searches a database for potential parts to compose, but still requires the user to draw the actual cuts.

The *Shuffler* system we introduce automates the process of model composition within families of models. In our context a family is a set of models that have a natural decomposition into interchangeable, meaningful parts. For instance, all quadrupeds have a similar body part structure: four legs, a body, a head, and, typically, a tail. Other examples include such diverse classes of objects as humans, chairs, tables, and airplanes. Shuffler allows the users to create new models by composing interchangeable parts from models within a family. For example, the alien in Figure 1 was created by combining the body and arm of the standing woman, with the head and feet of the dinopet, the man's legs, and an arm of the sitting woman. Using Shuffler, the users do not need to perform any geometric operations. To generate new models they simply select which components should come from which input model. The system does the rest. We believe that this composition framework would be particularly useful in a production setting, where designers are asked to create a variety of objects within a given family, for instance the creation of a hundred human models to represent a crowd, or populating a palace with twenty types of tables. Using our system these tasks can be achieved in a minimal amount of time and with only a few mouse clicks.

Our system provides this modeling functionality by first automatically segmenting the models into meaningful parts and then establishing the correspondences between them. Our segmentation algorithm uses a definition of meaningful parts based on human perception [Hoffman and Richards, 1985]. Based on this definition we develop a number of quantifiable metrics, leading to a robust segmentation algorithm that does not require model-specific fine-tuning. Using a single threshold parameter for each family of models, the algorithm successfully segments them into meaningful components for which we can later establish good correspondence. The method operates on models with boundaries, with arbitrary genus, and with any number of connected components.

Our second algorithm computes part correspondences between pairs of models. In the past, part correspondences were mostly used to evaluate global shape similarity [Tangelder and Veltkamp, 2004]. As such, the methods used to find the correspondence did not need to compute a good match when the structural or shape variation between the models was high. Since our goal is very different, we strive to compute the optimal correspondences despite significant shape dissimilarities between the models. To obtain correct correspondences we perform optimal inexact graph matching [Conte et al., 2004] using a hierarchical approach combined with stochastic local search. As shown in

Figure 1, our method correctly establishes the correspondences between the cow and the feline, and the four bipeds despite very significant structural and shape differences, such as different genus, dissimilar poses, and different proportions.

By combining the two techniques we are able to identify and match the interchangeable parts for a large variety of models (Figure 9). Using the computed correspondences we provide a simple to use, and robust model composition system. Our approach significantly simplifies the generation of 3D content, making modeling faster, more accessible, and less expert driven.

The rest of the paper is organized as follows. Section 2 reviews previous work. Section 3 provides an overview of the method. Sections 4 and 5 describe the segmentation and matching algorithms respectively. Section 6 describes the composition interface. Section 7 presents the results. Finally, Section 8 provides a summary of our work.

2 Previous Work

Most commercial modeling systems, such as Maya [Alias] and 3D Studio Max [3DMax], are targeted toward expert users and require significant time, expertise and artistic talent to generate complex models. Sketch based modeling interfaces [Igarashi et al., 1999] are much easier to manipulate and are successfully used to create simple shapes, but encounter difficulty when rich details are needed. Blending techniques [Schreiner et al., 2004; Kraevoy et al., 2004; Allen et al., 2003] provide a mechanism for the creation of detailed shapes that reuses existing models. These methods are based on a mapping between input models, and require the manual specification of numerous feature point correspondences.

Most tools for modeling-by-composition focus on correct ways to combine the geometries of the parts in order to generate a smooth transition between them [Sorkine et al., 2004; Yu et al., 2004] and rely on the user to cut the parts from the input models. Recent approaches for simplifying the cutting process include intelligent scissoring [Funkhouser et al., 2004; Lee et al., 2005], which optimizes approximate cuts generated by the user, and cut computation based on part alignment [Hassner et al., 2005].

The mesh composition system described by Funkhouser et al. [2004] is conceptually the closest to our work. Given an input model with a selected part, their method searches a database for similar models, focusing on models containing parts that resemble the selected one in terms of shape. Next, users define cuts on both models with an intelligent scissoring interface. Finally, the system automatically aligns the parts and stitches them together. In contrast, by automatically establishing part correspondences between input models, Shuffler makes the composition process fully automatic, eliminating the need for the user to generate cuts.

The Shuffler modeling system combines new algorithms for meaningful mesh segmentation and part matching. We now review relevant existing methods in these categories.

Meaningful Segmentation: The accepted notion of meaningful parts relies on human perception and is based on the observation that human vision defines part boundaries along negative minima of principal curvatures [Hoffman and Richards, 1985]. This definition implies that the meaningful parts are in some sense convex. Mesh segmentation into meaningful components has many applications [Shamir, 2004] and has received a lot of attention in the last few years [Katz and Tal, 2003; Lee et al., 2005; Zhang and Liu, 2005; Katz et al., 2005; Lien and Amato, 2005].

Many of the methods define parts by locating farthest points using geodesic distances, either directly [Katz and Tal, 2003; Zhang and Liu, 2005] or using the MDS transform [Katz et al., 2005]. Due to the reliance on surface distances, both types of methods do not perform well on models with concavities, boundaries, or high genus. Our method can handle all three (Figure 7).

Katz et al., [2003] use the Lloyd iterations framework for meaningful mesh segmentation. Other researchers [Cohen-Steiner et al., 2004; Julius et al., 2005] used this approach for different types of mesh segmentation. In our work we also use the Lloyd framework, but in contrast to most previous methods we automatically determine the necessary number of patches.

Lee et al. [2005] focus on locating part boundaries using local curvature estimates. Their method performs best in a semi-automatic setting. Lien and Amato [2005] measure the distance between the model and its convex-hull and cut the model in regions where the distance is larger than some threshold value. Changing the threshold lets the user control the extent of the decomposition. Similar to many other techniques they encounter problems on models with concavities.

Most other segmentation methods remain somewhat vague as to how to measure the extent of the decomposition. Many [Katz and Tal, 2003; Zhang and Liu, 2005; Katz et al., 2005] use recursive algorithms to control the degree of decomposition; however, it is not clear how to quantify the 'meaningfulness' achieved by each level in the recursion. Katz et al. [2005] demonstrate that using the same level in the recursion hierarchy, their method tends to generate similar segmentations for different animal models.

Part Correspondence: The problem of part correspondence is closely related to graph matching [Conte et al. 2004]. In particular, our problem falls into the category of inexact matching, namely matching between graphs with different structures. This problem is known to be extremely hard.

Approximate matching of part adjacency graphs and skeletons is sometimes used to measure the degree of similarity between 3D models for shape retrieval purposes [Tangelder and Veltkamp, 2004; Cornea et al. 2005; Tal and Zuckerberger 2006]. In that context, since the output of the methods is a matching score, an approximate match that, for instance, interchanges symmetric parts or flips a tail and a leg is often adequate. Furthermore, it is common to assume that global shape similarity implies very similar connectivity of the part graphs, and to simply assign a low matching score to models with different structure. The geometry processing algorithms that require correct part correspondences, such as Zuckerberger et al. [2002], typically assume that those are provided by the user.

Partial matching [Funkhouser et al., 2004] finds models that contain a part similar to a particular part in the given model. In our case we need to find a correspondence that works best for all the parts, rather than just one. Selecting the best match for one part, may enforce very bad matches for some of the other parts. Thus we need to use a global approach that considers all the correspondences at once.

3 Overview

The Shuffler editing system consists of an offline and an online component. The first component segments the input meshes into meaningful parts, or patches, and computes the correspondences between them. The second component provides an interface for creating new models by composing parts from the different input models using the established correspondences.

Segmentation: The mesh segmentation algorithm is based on a convexity metric that measures the distance between a mesh patch and its convex hull. This metric is combined with a volu-

metric measure of compactness to generate patches which are both convex and compact. The metrics are described in Section 4.1. The method consists of two main stages.

- **Patch formation:** The first stage introduces patches one by one and positions them in a way that optimizes a combination of the convexity and compactness metrics using a bounded Lloyd iteration framework. The algorithm uses a convexity threshold to determine the necessary number of patches in the segmentation. The process is described in Section 4.2 and demonstrated in Figure 3.
- **Improvement:** The second stage improves the segmentation. It merges adjacent patches, if the combined patch satisfies the convexity threshold, and optimizes patch boundaries (Figure 4). The procedure is described in Section 4.3.

Correspondence: The part correspondence is computed separately for each pair of models. Given a set of models, we can either compute all the possible correspondences, or, in a more practical setting, compute the correspondences between one of the models, typically the most canonical one, and the remainder.

Given two models, we need to find the best correspondence between their parts based on a set of geometric and topological considerations. We note that different models can have significant structural differences, resulting in different meaningful decompositions. For instance the feline's wings (Figure 1), which are correctly detected by our segmentation method (Figure 7 (f)), have no corresponding parts on any other model in our data set. In order to handle such structural differences, rather than searching for one-to-one part correspondences, we search for one-to-one group correspondences, where a group can contain any number of parts. The need to handle group-to-group correspondences makes the problem drastically harder, as it introduces both the number of groups and the actual grouping as additional unknowns.

The first challenge we face is to find a cost function to evaluate the correspondences and compare matchings across different groupings and a different number of groups (Section 5.1). We use the cost function to compute a good coarse level correspondence between the parts and then further refine it if possible. We use local stochastic search [Hoos and Stützle, 2004] to find the coarse level matches with low cost (Section 5.2). We then refine them, increasing the number of groups (Section 5.3).

The final solution contains the set of low cost fine level correspondences found. By default, the algorithm returns the best one among them. Since our metrics may sometimes disagree with human perception, we provide the user the option to see all the low cost matches and to select another match, if desired. In nearly all our examples the best result provided a correct correspondence from a human perspective. In two cases, shown in Figure 9 (h) and (o), the perceptually correct result was ranked as second best.

Composition Interface: The interface provides a simple modeling-by-composition environment where users can select which parts should come from each model. After the parts are selected, the system automatically constructs the new model from the parts, computing optimal alignment and, if desired, blending and stitching the parts together to create a watertight model when possible, as described in Section 6.

We now describe each component of the algorithm in more detail.

4 Convex Segmentation

Our method is aimed at creating a perceptually meaningful segmentation of input meshes. To describe the method, we first introduce a number of metrics to evaluate the quality of a given segmentation, and then proceed to describe the algorithm.

4.1 Metrics

Convexity: The notion of meaningful parts is closely related to the notion of convexity [Hoffman and Richards, 1985]. Thus, we base our algorithm on a convexity metric. In contrast to Lien and Amato [2005], we measure convexity rather than concavity, providing a local metric, which can be evaluated independently for any mesh patch. We evaluate the convexity of a mesh patch by measuring the distance between the patch P and its convex hull $C(P)$. The distance is defined as an area weighted average of the distances from the patch triangles t to the convex hull:

$$dist(P, C(P)) = \frac{\sum_{t \in P} dist(t, C(P)) \cdot area(t)}{\sum_{t \in P} area(t)}, \quad (1)$$

where $area(t)$ is the area of the triangle t, and $dist(t,C(P))$ is the distance from the triangle t to the convex hull $C(P)$. Specifically, $dist(t,C(P))$ is defined as the distance along the imaginary line that proceeds in the direction of the triangle's normal, and connects the center of the triangle to the convex hull. While there are other ways to measure distances between meshes, we found that the normal distance yields excellent results and is cheap to compute. The distances are measured as percentages of the bounding box diagonal of the object. Combined with the area based averaging, this gives us a dimensionless measure, which is independent of the model and patch size.

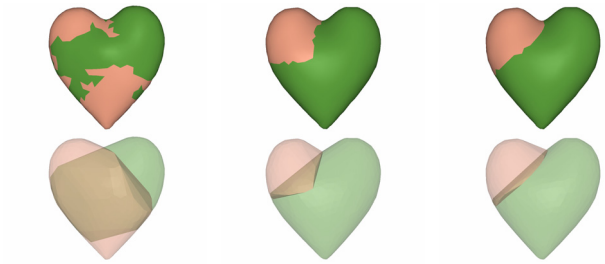

Figure 2: Patch compactness: (left) convex patches created without the compactness metric; (middle) patches generated by the segmentation using the combined patch cost; (right) patches after boundary improvement. The bottom row shows the patch convex hulls.

Compactness: Our primary objective is to segment the mesh into nearly convex patches. However, to achieve a useful segmentation, it is not enough for the patches to be convex, they must also be compact. In particular, we want to prevent random segmentation in areas that can be equally occupied by more than one neighboring patch, thereby avoiding the possibility of extremely complex patch boundaries (Figure 2 (left)). To evaluate the compactness we consider the convex hull of the patch. When the patch is close to convex, its convex hull captures well the volumetric shape of the patch. We therefore introduce a metric aimed at the generation of compact, relatively spherical convex hulls. We calculate the compactness as an area to volume ratio of the convex hull C,

$$comp(C) = \frac{area(C)}{volume(C)^{2/3}}. \quad (2)$$

This value is minimal for a perfect sphere and increases for all other shapes. Since the hull is convex, its volume is trivial to compute. We note that this metric is close to infinity for a nearly planar patch. We discuss the implications of this observation on the algorithm in the next section.

Volume: The last metric we use is the sum of volumes of the convex hulls of the patches. In some cases this metric complements the other two. If the sum of volumes is large, this always implies that the segmentation is suboptimal. The sum is large either if the distance from the patches to their hulls is large, namely the patches are not convex, or if there is a significant overlap between the hulls (Figure 2). However, a small sum does not guarantee a convex or compact segmentation. For instance, flat patches with a large area, can have a very small volume. We therefore do not use this metric during the segmentation process itself. The metric is used in the improvement stage once we already have a segmentation which is convex and relatively compact. In this setting the sum of volumes reflects the amount of overlap between the convex hulls.

Patch Cost: We define the cost function of a potential patch as a combination of convexity and compactness,

$$cost(P) = (1 + dist(P, C(P))) \cdot (1 + comp(C(P)))^{\alpha}, \qquad (3)$$

where α controls the tradeoff between the two. In all our examples α is set to 0.007 which appears to provide a good balance between convexity and compactness measures. We use multiplication to avoid the need to normalize both components to the same scale. The addition of 1 prevents the cancellation of one component as the other approaches zero.

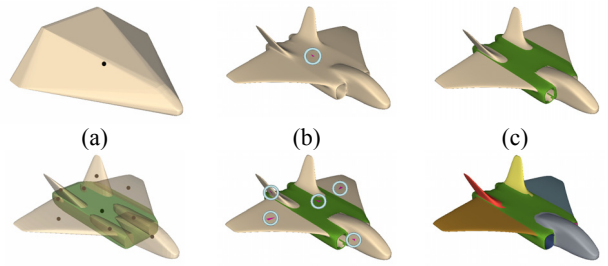

Figure 3: Segmentation stages: (a) the hull of the first potential patch (the entire model) and its center; (b) first seed triangle; (c) first patch after convergence; (d) hulls of the existing patch and the new potential patches; (e) new seed triangles; (f) patches after convergence.

4.2 Patch Formation

Given these metrics, the goal of our decomposition algorithm is to segment the model into a small number of nearly convex, compact patches such that our convexity error (Equation 1) for each patch is below the user specified threshold D_{max}. The threshold values used for each family of models are listed in Table 1. To generate the patches we use a bounded Lloyd iteration scheme [Julius et al. 2005; Katz and Tal, 2003; Cohen-Steiner et al., 2004]. In contrast to previous methods we do not expect the user to provide an estimated number of patches for the segmentation. Starting with zero patches, our algorithm generates the patches one-by-one using the following four step procedure.

1. **Potential patch generation:** The method collects the unassigned triangles, which do not belong to any patch, into connected components and classifies each component as a potential new patch. A triangle can be unassigned either at the beginning of the algorithm or during patch growing when all the patches have reached the convexity bound D_{max} (Figure 3 (c)). Both possibilities are a good indication that a new patch is required in the unassigned region. This method of formation of new patches allows us to implicitly handle models with multiple components. After forming the potential patches, the method computes the convex hull for each of them (Figure 3 (a) and (d)).

2. **Seed generation:** This stage finds seeds for re-growing existing and potential patches. We view the convex hull as the proxy of each patch, thus to facilitate convergence, the choice of seeds should minimize the change between the hulls computed at each iteration. A seed consists of a seed triangle and a seed convex hull. We define the seed convex hull as the tetrahedron formed by the seed triangle and the center point of the current convex hull, highlighted in Figure 3 (a) and (d). This selection reduces the difference between the proxies obtained at consecutive iterations. To further minimize the difference we select a seed triangle which fits well the current proxy, namely one that is close to the current convex hull. We also want the new patch to be compact from the start. Thus the algorithm selects the seed triangle $t \in P$ that minimizes the following cost function:

   $$cost(t, c, P) = (1 + dist(t, C(P))) \cdot (1 + comp(c))^{\alpha},$$

   where c is the seed convex hull and $C(P)$ is the current convex hull (Figure 3 (b) and (e)). This function is nearly identical to the patch cost function (Equation 3) and uses the same value for the compactness weight α. This seed construction explicitly prevents the formation of zero-volume hulls that could cause potential problems to the behavior of our compactness metric. After the seeds have been selected we reset the patches by marking the rest of the triangles as unassigned.

3. **Patch growing:** For each vertex v that shares an edge with a current patch, the algorithm computes the insertion cost as the cost of the updated patch formed by adding v to P, $cost(P+v)$. At each growing step, the algorithm uses the cost function to find the best adjacent vertex to add to one of the patches. If the convexity error $dist(P+v, C(P+v))$ of the updated patch is below the threshold D_{max}, the vertex is added to the patch. Otherwise, the algorithm proceeds to the next best vertex. If no such vertex is found the growing is terminated (Figure 3 (c) and (f)) and the algorithm proceeds to Stage 4. After a vertex is added to a patch, the patch convex hull is updated accordingly and the insertion cost for the vertices adjacent to the patch is recalculated. The growing is repeated until all triangles are assigned to patches or until patches can no longer grow without violating the threshold.

4. **Termination:** The algorithm now tests for termination conditions. If the new patches differ from the ones grown in the previous iteration, the algorithm repeats the reseeding and growing loop, Stages 2 and 3. Patches are defined as static if 95% percent of their triangle content is identical to that of the previous iteration. Once all the patches are static, we check if the patches cover the entire model. If not, the algorithm returns to Stage 1, otherwise the algorithm terminates.

   Our algorithm typically takes only two to three iterations to converge each time new patches are added. We do note, that theoretically the Lloyd algorithm has no termination guarantees in the discrete mesh setting [Cohen-Steiner et al., 2004].

Concavities: Many models such as the plane (Figures 3 and 4) contain concave regions. Identifying such regions is useful for many applications of meaningful segmentation [Shamir, 2004]. However, most existing segmentation methods do not identify them as separate parts. In our setting it is straightforward to detect patches placed in concave regions. We classify a patch as concave if the normals of more than 90% of its triangles point towards the center of the patch convex hull. If a patch is classified as concave we invert the normal direction when computing the distance to the hull for its triangles. The rest of the process-

ing is unchanged. This simple modification gives us well shaped patches for each concavity.

The dominant component of the algorithm runtime is the patch growing, which requires the use of incremental convex hull construction, to measure the cost of adding a vertex to a patch. Using efficient convex hull updates, on a 3GHz Pentium IV the method takes 25 seconds to segment the 10K faces lion (Figure 5) and 120 seconds to segment the 22K faces sitting woman (Figure 7 (e)). Running the algorithm first on a simplified model and then adding the rest of the vertices one by one can significantly reduce these times. In this case the cost will only be computed when adding vertices on the boundary between patches of the simplified mesh. However, this can provide less accurate results.

4.3 Patch Improvement

After the segmentation terminates, we have roughly the correct number of patches, placed at the correct locations on the model (Figure 3). The algorithm now performs local improvement, merging redundant patches and improving the boundaries between adjacent patches (Figures 2 and 4).

Merging: Due to the incremental nature of the algorithm, it sometimes creates a small number of redundant patches. To eliminate those, we check each pair of adjacent patches to see if they can be merged. We compute a common convex hull for the two patches and merge them if the combined patch satisfies the convexity threshold.

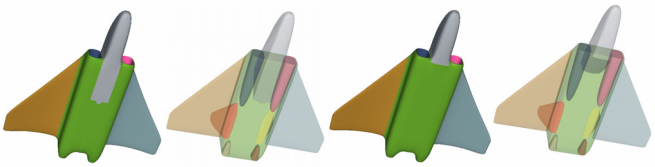

Figure 4: Boundary improvement for the plane in Figure 3, view from the bottom: (left) patches and hulls before, (right) patches and hulls after.

Boundary Improvement: After segmentation and merging all patches are nearly convex, and reasonably compact. However, it is possible to further improve their compactness and to reduce overlaps without sacrificing convexity. We iterate between improving the compactness (Equation 2) and the sum of hull volumes. At this point, since we have a segmentation that is both convex and relatively compact the sum of volumes reflects mostly the amount of overlap between the convex hulls. We found that iterating between improving the compactness and the volume gives us the fastest convergence to well shaped boundaries. We use the same procedure for improving both.

The improvement algorithm processes the vertices on shared boundaries one at a time. The algorithm tests if the boundary can be improved by removing the vertex from one patch and adding the vertex and all the incident triangles to the other patch. If the update improves the optimized metric and the convexity error for both patches remains below threshold, the new configuration is accepted; otherwise the algorithm reverts back to the previous state. The procedure is repeated as long as the boundaries can be improved.

The boundary improvement increases the compactness of the patches, without compromising the convexity. Figure 4 shows the plane segmentation before and after boundary improvement.

At the end of the process we have a decomposition of the mesh into nearly convex compact patches, where the convexity error of each patch is bounded by the threshold D_{max}. The patches accurately capture the meaningful parts of the model. In addition to the segmentation itself, as part of our output we obtain a union of convex hulls that closely approximates the model. These can be used for efficient collision detection [Shamir, 2004]. In our system we also use the convex hulls as an approximate representation of the parts when establishing the correspondences between them.

5 Part Correspondence

We now establish the correspondences between the parts computed by the segmentation. As noted, since different models have different shapes, their segmentations may differ in terms of the number of parts, the part shape, and the connectivity between the parts. Thus we are unlikely to have a one-to-one correspondence between individual parts. Instead we search to establish a one-to-one correspondence between groups of parts.

Our first challenge is to define a cost function that evaluates the quality of a given correspondence. Since the grouping and the number of groups are unknown, the cost must be comparable between different numbers of groups and different groupings.

To find the best match, given the cost function, we apply a hierarchical procedure that takes advantage of the geometric information available, particularly by relying on the convexity of our parts. We start by computing a coarse level correspondence and then refine it.

5.1 Matching Cost

We now define the cost of a one-to-one match between groups on two models. We do not assume a priori registration of the models, thus to enable comparison between the models, we scale them to be of equal size. For rigid objects such as furniture or planes, the scaling equates the lengths of the diagonals of the minimum volume bounding boxes. For articulated models such as humans or animals bounding boxes are unreliable, due to pose variation. Therefore we scale those based on their volume, obtaining models with the same "weight". The volume is computed as the sum of part hull volumes.

The cost function we optimize consists of several components with different degrees of importance. To describe those we first introduce some terminology. We denote the parts of the first model as p_{1l} to p_{1n} and of the second as p_{2l} to p_{2m}. The groups g_{1l} to g_{1k} on the first model have a one-to-one correspondence to the groups g_{2l} to g_{2k} on the second, where the matching maps g_{1i} to g_{2i}, $m(g_{1i})=g_{2i}$. A group g_{ij} consists of any number of parts p_{ia}. Given two adjacent parts p_{ia} and p_{ib} we define their *midpoint* as the center of the intersection between their convex hulls. For watertight models, the midpoints are efficiently estimated as the average of the vertices on the shared boundary between the parts. We define the midpoint graph by connecting all the midpoints adjacent to each part with straight line edges (Figure 5 (c) and (e)).

Group Convexity: This metric measures the quality of the part grouping and is measured independently on each model. While we group parts to obtain the correspondences, the groups should still represent meaningful components of the model. Thus they should still be as convex and compact as possible. The convexity metric we used during segmentation is relatively expensive to compute, and does not apply to groups of parts taken from different mesh components. To measure the convexity of a grouping, we therefore measure the volume of the group hulls, which indicates how much larger the group is than the sum of its parts. We compute the convex hull of each group, as the hull of the hulls of the individual parts, and then sum those on both models

$$c_{cost} = \sum_{j=1}^{k} volume(C(g_{1j})) + \sum_{j=1}^{k} volume(C(g_{2j})).$$

Graph Distances: Given a grouping on one of the models, we define the distance between groups $d(g_{ia},g_{ib})$ as the shortest distance between their parts on the midpoint graph. Note that using this definition the distance between adjacent groups is zero. We found that the midpoint distances capture best the global shape of our models. For instance, on mammals the distances clearly distinguish the legs from the head and the tail. These distances are also well preserved between different poses. The second important component of our matching function is the difference between distances between corresponding groups:

$$g_{cost} = \sum_{a=1}^{k}\sum_{b=1}^{k}(d(g_{1a},g_{1b})/s_1 - d(g_{2a},g_{2b})/s_2)^2, s_i = \sum_{a=1}^{k}\sum_{b=1}^{k} d(g_{ia},g_{ib}).$$

We found that due to the normalization by s_i, the metric remains comparable across matches with a different number of groups or different groupings. Additionally, when switching from coarse to fine segmentation, there is no appreciable change in the metric value.

Euclidean Distance: Our second distance metric measures the difference between distances between corresponding groups in the Euclidean space. The Euclidean distance between two groups is defined as the distance between the centers c_{ij} of the group convex hulls.

$$e_{cost} = \frac{1}{k^2}\sum_{a=1}^{k}\sum_{b=1}^{k}\left(\frac{\|c_{1a}-c_{1b}\| - \|c_{2a}-c_{2b}\|}{\|c_{1a}-c_{1b}\| + \|c_{2a}-c_{2b}\|}\right)^2.$$

Since we measure the proportion of distances we capture the differences on the different size scales of the model. This metric helps to avoid twisted matches, such as mapping the right arm to the right arm and left arm to left arm, while mapping the right leg to the left one and vice versa. Depending on the grouping, the graph distances alone can give the same score to both twisted and correct matches. Despite the fact that the Euclidean distance prefers rigid correspondences, we found that the combined cost function works well on models in different poses such as the man and the sitting woman, or the lion and the dinopet (Figure 9 (j) and (e)).

Group Volume: The group volume $v(g_{ij})$ is defined simply as the sum of volumes of the convex hulls of the parts in the group. Thus the degree of similarity of a match in terms of volume is simply

$$v_{cost} = \frac{1}{k}\sum_{j=1}^{k}\left(\frac{v(g_{1j}) - v(g_{2j})}{v(g_{1j}) + v(g_{2j})}\right)^2.$$

This metric provides only a very coarse evaluation of the match, as many different parts have a similar volume. Moreover, models within one family may have very different proportions. Thus, while we include the metric in the combined cost function, we give it a very small weight.

Registration: As noted, we do not assume that the inputs are aligned prior to processing. Given the variety of poses and shapes for many of our models, global registration beforehand is unreliable. However, after we already have the coarse matching, we have a point-to-point correspondence between the centers of the group hulls. We use these correspondences to obtain a more reliable alignment, using the quaternion method of Horn [1987]. After the alignment, we can measure the registration cost as

$$r_{cost} = \frac{1}{k}\sum_{j=1}^{k}\|c_{1j} - c_{2j}\|^2.$$

The use of this metric in the coarse to fine transition is explained in Section 5.2.

Combined cost function: The combined cost function is

$$M_c = (1+c_{cost})^{\alpha}(1+g_{cost})^{\beta}(1+e_{cost})^{\gamma}(1+v_{cost})^{\delta}(1+r_{cost})^{\varepsilon}.$$

For all the models we used $\alpha=\beta=\gamma=1$. The weight of the volume varies between different families of models (Table 1). For the coarse level matching we use $\varepsilon=0$, since the registration cost is only measured after we obtain a coarse match, at which point we set $\varepsilon=0.5$, except as listed in Table 1.

We now describe the two stages of the matching algorithm.

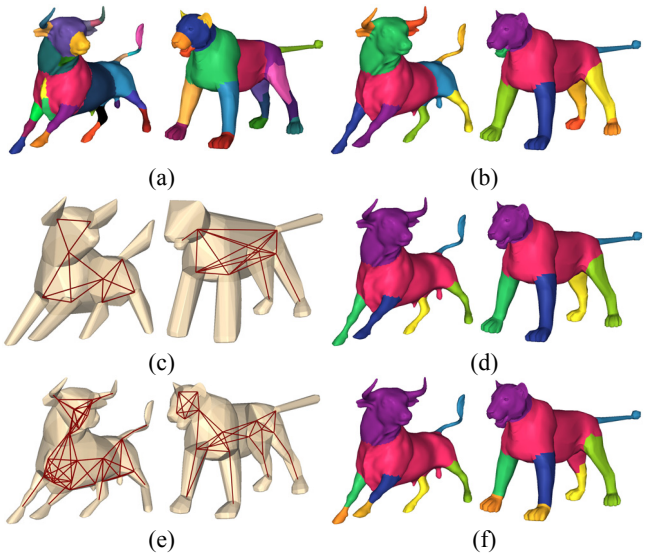

Figure 5: Part correspondence algorithm: (a) fine segmentation; (b) coarse, nested segmentation; (c) midpoint graphs for coarse segmentation; (d) coarse level correspondence; (e) fine level midpoint graphs; (f) fine level correspondence.

5.2 Coarse Level Matching

To facilitate coarse level correspondence, we first coarsen the segmentation we obtained (Figure 5 (a),(b)). The coarsening takes the output of the segmentation algorithm and merges patches using a higher convexity threshold than the original one. We used D_{max}=1% for all the models. As part of the preprocessing, concave parts, such as the engines on the plane (Figure 3), are merged with their surroundings.

To facilitate coarse level matching the user provides upper and lower bounds on the number of groups in the coarse correspondence. The upper bound is slightly higher than the typical number of coarse parts for the family. For quadrupeds we have seven coarse parts: body, head, tail, and four legs. The upper bound we use is nine. The lower bound is provided to simply speed up the process, and avoid testing all groupings between two and the upper bound. The lower bound we use for quadrupeds is six.

We use stochastic local search [Hoos and Stützle, 2004] to find the best coarse level matches. Our search combines randomized selection of multiple initial guesses and deterministic greedy search for local minima starting from those guesses. At the limit, when run an infinite number of iterations, the stochastic search is guaranteed to find the global minimum of the cost function. In practice, we clearly detect only a subset of the minima. However, in our implementation we found that a hundred randomized initial guesses were sufficient to find good matches for all our examples. We store the hundred lowest cost matches that we obtain during the search.

After the search terminates, for each of the stored results we use the correspondences to align the models by aligning the group

centers, as described above. We then use the alignment to recompute the cost of the stored matches taking registration cost into account. The cost function without registration allows global mirroring and typically assigns low cost to both the correct and the mirrored results. The registration allows us to distinguish between the two. The best coarse match for the lion and the bull is shown in Figure 5 (d).

Randomized Initial Guess: To initiate a new search we start by randomly selecting an initial guess which consists of three components: the number of groups, the groups themselves, and an initial correspondence. The number of groups is set randomly with equal probability for any number between the user provided upper and lower bounds. Note that the number of groups is the same for both models, as at any point in the algorithm we have one-to-one group correspondence. The selection of parts that form each initial group is also randomized, but with higher probability given to better groupings based on group convexity c_{cost}. The initial correspondence is set at random with equal probability for any assignment.

Local Search: The local search starts from an initial guess and searches for a local minimum using two atomic operations, swap and regroup. The swap operation simply switches the correspondences between two pairs of groups $g_{2i} = m(g_{1j})$ and $g_{2j} = m(g_{1i})$. The regroup operation is more complex, and is performed separately for each model. First, a part is removed from a group containing more than one part, to form a new group. Next, two groups are merged; thus the total number of groups remains constant.

Using these two operations the search is performed using the following cycle.

1. Compute the cost of the current match. Test all possible swaps and evaluate their cost. Select the swap that improves the cost the most and perform it. Repeat the process.
2. If none of the swaps can improve on the current match, test all possible regroupings and evaluate their cost. Select the one that improves the cost the most and perform it. Return to Step 1.

If neither swaps nor regrouping can improve the current match, the process terminates.

5.3 Fine Level Matching

After we have obtained the coarse level correspondence, we refine it, by increasing the number of groups to get a fine level correspondence (Figure 5 (f)). The algorithm first switches from the coarse segmentation back to the fine one (Figure 5 (a), (b)). Since the coarse parts were obtained using merging, the group structure of the match remains unchanged while the number of parts in each group increases. The method then runs the local search using the current correspondences as an initial guess. In our experience the new search maintains the coarse correspondences up to local adjustments, typically moving some of the finer parts from one group to an adjacent one (see attached video). This stability is an important property of our cost function, indicating that the function remains nearly invariant to the number of parts.

Since we consider the coarse matches as accurate, the fine level matching focuses on constructing sub group correspondences. It checks if the matching can be improved by splitting the coarse groups into a number of fine groups. Given a pair of corresponding groups g_{1i} and g_{2i}, the maximal number of fine groups l is the number of parts in the smaller group. For each i between two and l, the system finds the best split as follows.

Each coarse group is first split independently into i fine groups, selecting the best fine groups in terms of convexity c_{cost}. The algorithm then uses either exhaustive or local search to find the best correspondence between the fine groups on the two models in terms of the overall cost M_c. If the number of fine groups is small (less than ten) we perform exhaustive search. In the rare cases when the number is higher we switch to local search.

If any of the splits improve the overall cost, the algorithm performs the split that improves the cost the most. The process is repeated for all the coarse groups. We found that the constraint of considering only optimal splits in terms of convexity, drastically reduces the possibility of generating fine-level correspondences that disagree with human perception. Figure 6 shows two examples of refinement where all possible splits were considered, independent of convexity. As can be seen, in some cases this allows us to achieve a finer degree of correspondence, while in others the method finds wrong results. The user can choose whether this constraint should be enforced.

Figure 6: Unconstrained refinement gives a correct finer correspondence for the lion-camel match; for the lion-bull match, it does not. The constrained algorithm returns a head to head match for both (Figures 5 and 9 (b)).

The refinement is performed for all top coarse matches. Thus, the final solution contains the top fine matches, and by default we return the best one. We give the user the option to see all the top matches and to select another match, if desired. The algorithm took 25 seconds to find the coarse level correspondence for the lion and the bull (Figure 5) and another 15 seconds to refine it.

6 Composition

The composition process starts by loading a set of models. The user then selects a base model from which to start the processing. By clicking on a part of this model and then selecting any other model, the user indicates that this part should be replaced by the corresponding part on the second model (see attached video). To facilitate the processing, the composition first performs global alignment of the input models based on the correspondences. It uses the method of Horn, [1987] to maximize the alignment of the hull centers of the corresponding groups. When performing each individual part replacement it scales and translates the part to fit best with the rest of the model. We found that automatic rotation of individual parts to better align the boundaries is often undesirable as it modifies the poses of the models. We provide an interface to manually adjust the alignment if desired.

For watertight models, after all the parts are joined together we perform zippering and blending to generate the new model. Zippering is performed only if boundary topology allows it. For instance, since the feline's tail has a different number of boundary loops than the cow's one, zippering is not performed when replacing one by the other. More advanced blending can be performed using one of the recent methods [Yu et al., 2004; Sorkine et al., 2004].

7 Results

Throughout the paper we demonstrate several examples of models generated using our system as well as examples of the results of the two algorithms we developed. Figure 1 shows two examples of correspondences between interchangeable parts computed by our method and two fictional creatures generated using these. In both cases, despite significant differences in shape, connectivity,

and pose, the correspondences we establish are perceptually correct. Given the correspondences, we easily generate visually interesting models using our composition interface.

Figure 7 shows a number of segmentations generated using our method. The segmentation correctly detects the perceptually meaningful parts of the models, identifying features like ears and horns, the crown of the triceratops, and the fists of the two women. Our method correctly handles high genus models such as the feline, the fishing reel, and the jet, which cause problems to many previous methods. It also works on models with boundaries, such as the cat and the dog that have boundary loops around the eyes and mouth, not handled by most other methods. We note that our method does sometimes generate over-segmented results, such as the feline, however since our matching is not constrained to a particular group size, this is not a problem in our context. The last two examples in the figure are CAD models. Segmentation of CAD models is very useful for reverse engineering [Shamir, 2004]. Since our method works exceptionally well on such models, it can be successfully used for this purpose. Figure 8 demonstrates segmentations of models with multiple connected components.

Figure 9 shows examples of correspondences computed using our method. The examples include models from different families, including quadrupeds, humans, chairs, and planes. For the models of planes and chairs we used only the coarse level correspondence algorithm, as the expected number of parts was small a priori. The parameters used for the segmentation and matching are listed in Table 1.

As shown our algorithm establishes perceptually correct correspondences between the interchangeable parts. Note that the level of refinement in the correspondences depends on the degree of similarity between the models. For the lion and cat (Figure 9 (a)), the method finds an extremely detailed correspondence, including top and bottom jaws, and ears. For the cow and camel (Figure 9 (d)), where the correspondence is far less obvious, it obtains a correct but coarser match. The method correctly establishes correspondences between the chairs and planes rightly grouping parts from different connected components. The chair models are particularly challenging as their adjacency structures differ quite drastically.

Family	D_{max}	bounds on group # (coarse level)	δ	ε
quadrupeds	0.3%	6-9	0.2	0.5
humans	0.23%	6-9	0.2	0.5
chairs	0.3%	4-6	0.1	0.5
planes	0.6%	3-6	0.01	2

Table 1: Algorithm parameters

Figure 10 shows a number of new models generated using Shuffler and the parts used to construct them. For each of the models it took only a few mouse clicks for the user to specify the combination of parts they want. In some of the models, such as the camow, the users adjusted the alignment to achieve the desired composition effect.

8 Summary

We presented a modeling system for simple and robust creation of models by composing interchangeable parts. Our system allows users to generate detailed geometric models using a very simple interface, within a matter of minutes. Thus it makes the creation of 3D content much faster, and less expert driven. As part of the system we introduced two novel algorithms: one for meaningful mesh segmentation and one for part matching. The segmentation method uses a single threshold value to control the convexity of each generated part, automatically determining the number of parts. The part matching method computes perceptually correct part correspondences between pairs of models. It operates on both natural and man-made models.

For future work, we would like to further investigate the perceptual aspects of part correspondence and the correlation between those and measurable geometric properties of the matches.

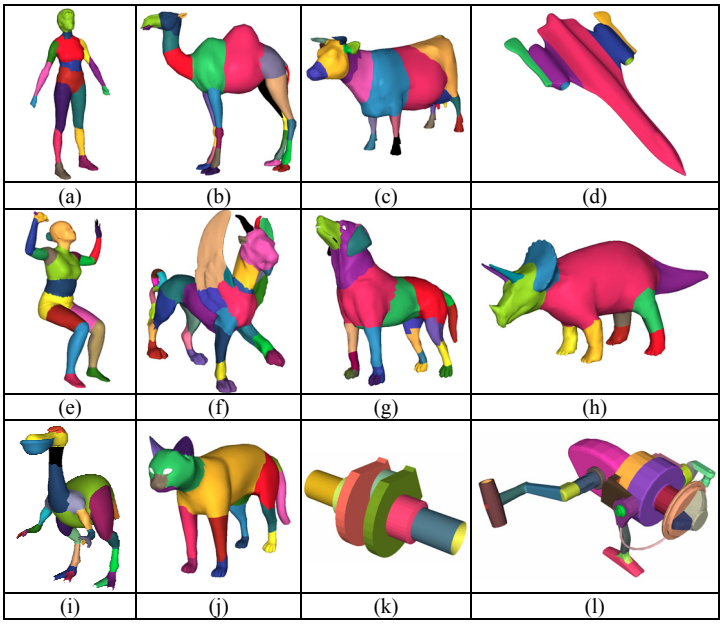

Figure 7: Segmentations of meshes with one component.

Figure 8: Segmentations of meshes with multiple components: (top) input models with each component shown in a different color, (bottom) segmented models.

References

3DMax , www.3dmax.com/

Alias Wavefront, www.alias.com/

ALLEN, B., CURLESS, B., POPOVIC, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Transactions on Graphics*, 22, 3, 587-594.

COHEN-STEINER D., ALLIEZ P., DESBRUN, M., 2004. Variational shape approximation. *ACM Transactions on Graphics*, 23, 3, 905–914.

CONTE, D., FOGGIA, P., SANSONE, C., VENTO, M.,2004, Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence* 18, 3, 265-298.

CORNEA, N. D., DEMIRCI, M. F., SILVER, D., SHOKOUFANDEH, A., DICKINSON, S., J., KANTOR, P. B., 2005, 3D Object Retrieval using Many-to-many Matching of Curve Skeletons, In *Shape Modeling and Applications*, 366-371.

FUNKHOUSER, T. A., KAZHDAN, M. M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., DOBKIN, D. P., 2004. Modeling by example, *ACM Transactions on Graphics*, 23, 3, 652-663.

T. HASSNER, L. ZELNIK-MANOR, G. LEIFMAN, R. BASRI, 2005. Minimal-Cut Model Composition, In *Shape Modeling and Applications,* 72–81.

HOOS, H. H., AND STÜTZLE, T., 2004. Stochastic Local Search Foundations and Applications, Morgan Kaufmann / Elsevier.

HOFFMAN, D. D., AND RICHARDS, W. A., 1985, Parts of Recognition, *Cognition,* 18, 65-96.

HORN, B. K. P.,1987. Closed-form Solution of Absolute Orientation Using Unit Quaternions, *Journal of the Optical Society of America*, 4, 629-642.

IGARASHI, T., MATSUOKA, S.,AND TANAKA, H., 1999. Teddy: A sketching interface for 3D freeform design. In *SIGGRAPH 99 Conference Proceedings*, 409-416. ACM SIGGRAPH.

JULIUS, D., KRAEVOY, V., SHEFFER, A., 2005. D-Charts: Quasi-Developable Mesh Segmentation, *Computer Graphics Forum*, 24, 3, 981-990.

KATZ, S., LEIFMAN, G., AND TAL, A., 2005, Mesh Segmentation using Feature Point and Core Extraction, *The Visual Computer*, 21, 8-10, 865-875.

KATZ, S., AND TAL, A., 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics,* 22, 3, 954-961.

KRAEVOY, V., SHEFFER, A., 2004. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics*, 23, 3, 861-869.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., SEIDEL, H.-P., 2005. Mesh Scissoring with Minima Rule and Part Salience, *Computer Aided Geometric Design,* 22, 5, 444-465.

LIEN, J.M., AND AMATO, N. M., 2005, Approximate Convex Decomposition of Polyhedra, *Technical Report TR05-00*1, Parasol Lab, Department of Computer Science, Texas A&M University,

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., ROESSL, C., AND SEIDEL, H.-P., 2004. Laplacian Surface Editing, In *Proc. Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 179-188.

SCHREINER, J., ASIRVATHAM, A., PRAUN, E., HOPPE, H., 2004 Inter-surface mapping. *ACM Transactions on Graphics*, 23, 3, 870-877.

SHAMIR, A., 2004, A Formulation of Boundary Mesh Segmentation, In *2nd International Symposium on 3D Processing, Visualization, and Transmission*, 82-89.

H. SUNDAR, D. SILVER, N. GAGVANI, AND S. DICKINSON, 2003 Skeleton based shape matching and retrieval. In *Shape Modeling International*, 130–139.

TAL, A., ZUCKERBERGER, E., 2006. Mesh Retrieval by Components, In *International Conference on Computer Graphics Theory and Applications*.

TANGELDER, J. W. H., AND VELTKAMP, R. C, 2004. A Survey of Content Based 3D Shape Retrieval Methods. In *Shape Modeling International*, 145-156.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. , 2004. Mesh editing with Poisson-based gradient field manipulation. *ACM Transactions on Graphics*, 23, 3,

ZHANG, H., AND LIU, R., 2005. Mesh Segmentation via Recursive and Visually Salient Spectral Cuts, In *Proceedings of Vision, Modeling and Visualization*, 429-436.

ZUCKERBERGER, E., TAL A., AND SHLAFMAN, S., 2002. Polyhedral Surface Decomposition with Applications, *Computers and Graphics*, 26, 5, 733-743.

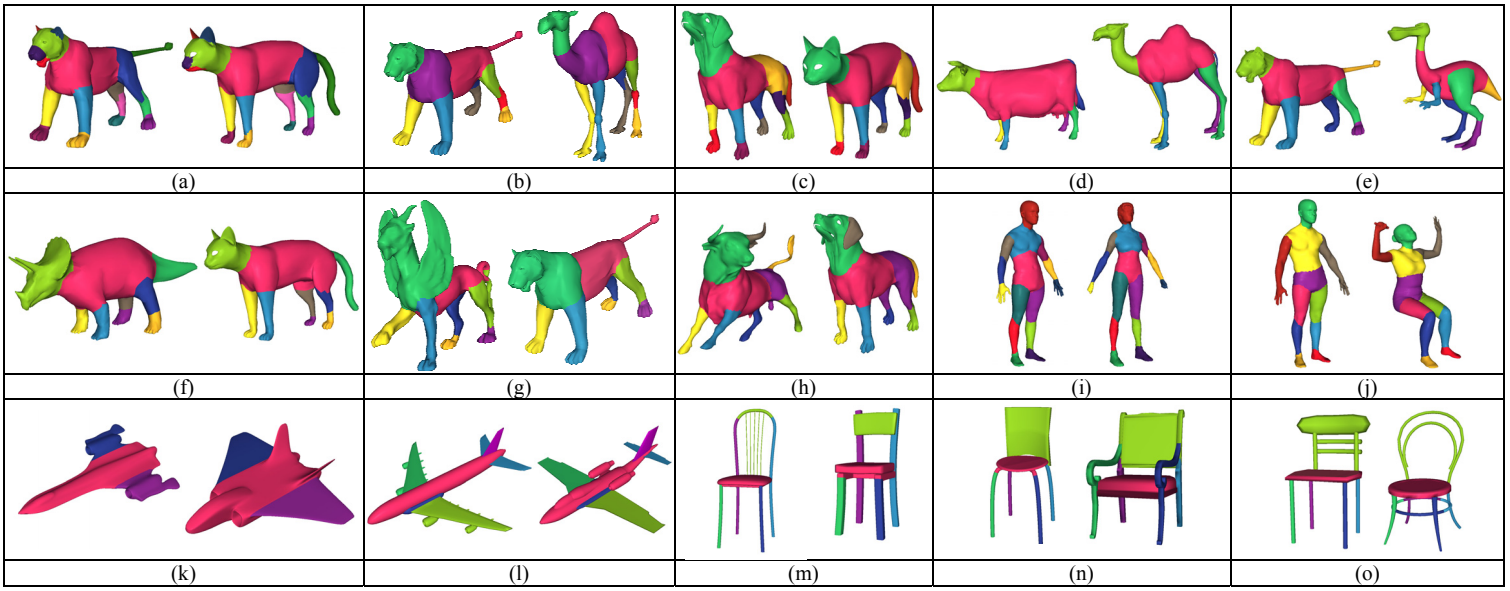

Figure 9: Pairwise matches.

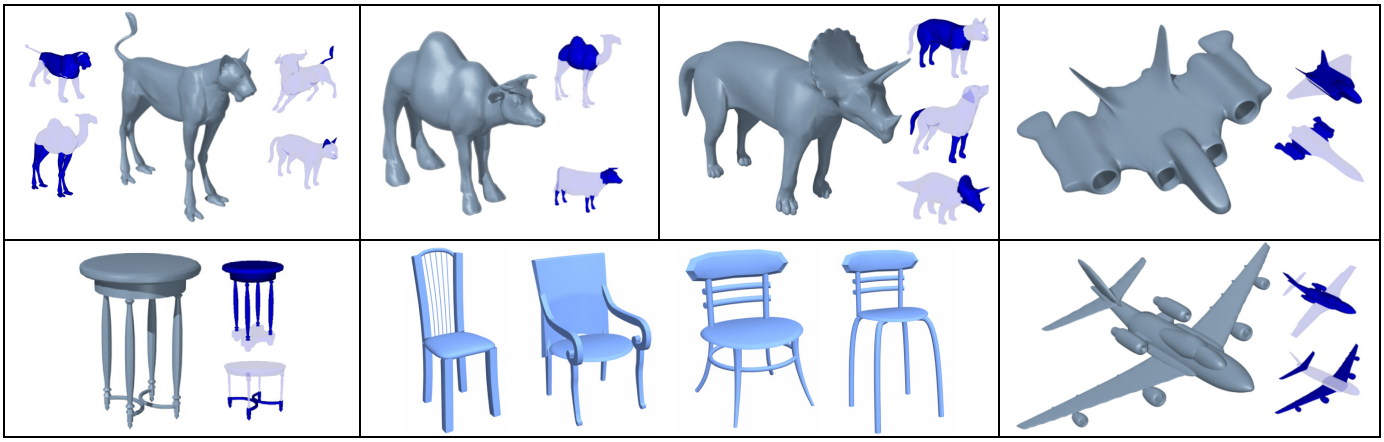

Figure 10: Shuffling: cambuliot, camow, triceracadog, a stealth-jet, a six engine super-jumbo, a table, and a bunch of chairs.