Model Composition from Interchangeable Components

Vladislav Kreavoy Dan Julius Alla Sheffer
University of British Columbia, Vancouver, B.C., Canada

Abstract

Following the increasing demand to make the creation and manipulation of 3D geometry simpler and more accessible, we introduce a modeling approach that allows even novice users to create sophisticated models in minutes. Our approach is based on the observation that in many modeling settings users create models which belong to a small set of model classes, such as humans or quadrupeds. The models within each class typically share a common component structure. Following this observation, we introduce a modeling system which utilizes this common component structure allowing users to create new models by shuffling interchangeable components between existing models. To enable shuffling, we develop a method for computing a compatible segmentation of input models into meaningful, interchangeable components. Using this segmentation our system lets users create new models with a few mouse clicks, in a fraction of the time required by previous composition techniques. We demonstrate that the shuffling paradigm allows for easy and fast creation of a rich geometric content.

1 Introduction

In recent years there has been an increase in demand for design and modeling tools aimed at non-expert users, driven in part by the increasing popularity of computer games that support player-driven development of new content [20, 23]. As a result there is an emerging effort to make creation and manipulation of 3D geometry accessible to a wider range of users by simplifying the user interfaces and reducing the amount of time users need to spend to create interesting models. Two popular approaches for simplifying model creation are sketch based modeling and mesh composition. While sketch based modeling is often restricted to creation of relatively simple models from scratch [15], composition tools [22, 10, 7] allow users to create more sophisticated models by combining parts of existing ones. State-of-the-art methods, such as Modeling by Example [7] and SnapPaste [22], have made the composition process faster and simpler. However, researchers acknowledge that it still takes the user an hour or more to create new non-trivial models using these systems [7]. We propose to take composition based modeling a step further, eliminating the need for the user to perform any time-consuming geometry manipulations and thus reducing the total time of a modeling session to minutes instead of hours.

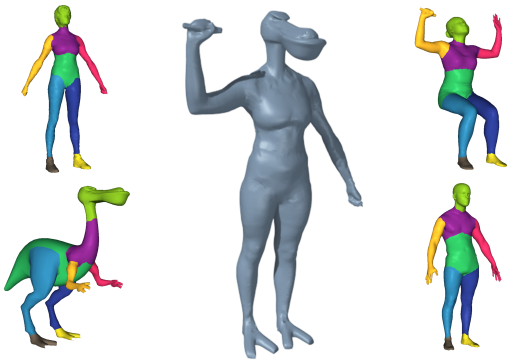


Figure 1. Interchangeable component correspondences computed by our system. An alien generated by shuffling the parts.

We observe that users of modeling tools often concentrate on generating new instances within specific classes of models. This is true for movie and game design where artists create crowds of people or other creatures, player-driven game content creation where players design avatars, and even home setups where users personalize everyday objects. Models within each class typically have a natural decomposition into interchangeable, meaningful components. For instance, all quadrupeds have a similar body part structure: four legs, a body, a head, and, typically, a tail. We utilize these observations, introducing a modeling system, *Shuffler*, where users can create new models by composing interchangeable components from objects within the same class. The system employs a shuffling paradigm where users start from a base or *target* model and modify it by replacing components of the model with corresponding components from other models. We observe that even with a small number of input models, this paradigm allows us to create a large number of new models. Specifically, given n input models with k components in each, n^k models can be synthesized by shuffling. Since numerous models from

the commonly used model classes are readily available in web databases, shuffling is an effective modeling metaphor permitting the creation of countless new models.

To facilitate the proposed modeling paradigm we require a compatible segmentation of input models into interchangeable, meaningful components. While such segmentation has been mentioned as open problem by many researchers [21, 16], we are not familiar with any algorithms addressing it. In our system, we take a two step approach to computing compatible segmentations. We first segment the input models into meaningful *parts* and then find a correspondence between interchangeable *components*, or groups of parts, on different models (Sections 4-5). To generate a meaningful compatible segmentation we use similar notions of meaningfulness in both stages of the algorithm. The combined method is able to identify and match the interchangeable components for a large variety of models from different classes, including humans, quadrupeds, planes, chairs, and tables.

Utilizing the interchangeable part structure we completely eliminate the need for a novice user to perform any geometric operations, reducing the interface to a number of mouse click operations (Section 6). For expert users we provide several ways to influence the composition process and adjust the results to their liking. As we demonstrate, Shuffler supports the creation of sophisticated 3D shapes and characters in minutes (Section 7).

2 Previous Work

Commercial modeling and design tools geared toward non-expert users typical utilize a procedural modeling approach where users can select shape components or properties from a restricted pre-processed dataset [20, 23]. Sketch based modeling interfaces [15] allow users to create more diverse content, but are so far useful mostly for creating relatively simple shapes.

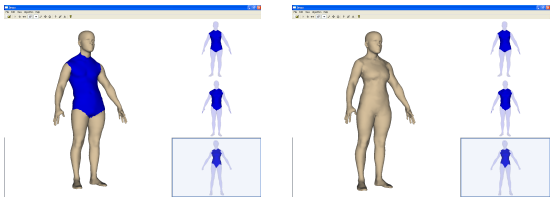
Mesh composition is emerging as an alternative simple modeling metaphor that allows even non-expert users to create complex models within a reasonable timeframe [7, 22]. In a typical composition setup, users first cut the two composed components from their respective input models and then align them such that the corresponding cut boundaries match reasonably well. The composition software is then used to connect the two components into a single model and define the geometry in the boundary region using a blending mechanism.

Several researchers proposed ways to simplify the user interaction during the boundary cutting and alignment [10, 7, 18, 22]. Smart scissoring methods [7, 18, 22] simplify the specification of the cut boundaries. Sharf et al. [22] require the user to provide only a coarse alignment and then improve the alignment further using a variation of deformable ICP. Funkhouser et al. [7] use the correspondence between the shuffled-in and out components to compute the alignment. This approach can lead to unintuitive alignment if the shuffled-in and out components differ significantly. By relying on the pre-computed interchangeable segmentation, Shuffler fully automates both the cutting and the alignment.

All the existing composition systems operate on one pair of components at a time. Thus composition involving multiple components, such as replacing a torso from one model by a torso from another (Figure 2) which requires composing together five components, can only be performed as a sequence of several composition operations. In our system, only a single operation is required to shuffle the two torsos, using the interchangeable component structure to facilitate the composition. Combining this ability with automatic alignment and component boundary extraction, Shuffler reduces the amount of user time required to create non-trivial models by one to two orders of magnitude, compared to state of the art methods [7]. Regrettably, more recent publications do not list user times for comparison.

Our pre-processing mechanism segments the input models into meaningful, interchangeable components. We are not familiar with any methods that provide such compatible segmentation. There is a large number of methods for meaningful segmentation of individual meshes, surveyed by Shamir [21] and Attene et al. [1]. Most methods rely on definitions of meaningful parts based on a study showing that humans segment models in regions of high negative curvature [11]. Geodesic distances are commonly used to steer the segmentation and negative curvature is taken into account when generating the actual cuts [17, 25, 16]. Amato et al. [19] consider the depth of the concave, negative-curvature, regions on the model to determine where to generate the cuts between the components. Chaselle et al. [3] use a dual approach, where instead of generating cuts in concave regions they generate parts which correspond to convex regions. The method searches for exact convex decomposition and shows that this problem is NP-hard. None of the methods have obvious extensions to compatible segmentation, though some mention it as a topic of future research [16].

In recent years, several authors addressed the problem of establishing point-wise or part-wise correspondences between models. Several methods do this by matching models skeletons [24, 5] or feature vertices [9]. Others compute complete point-to-point correspondences [2, 14, 8]. Most feature based methods are robust only under rigid transformations. Skeleton based methods [24, 5] are more robust to changes in pose but typically do not distinguish between symmetric skeleton branches such as arms or legs. Global point-to-point correspondence methods are closest to the task we address, and successfully find maps between models within the same class. The more recent methods



(a) (b)

Figure 2. Shuffling interface: (a) shuffled-in and out components highlighted; (b) composition result computed automatically. Note that the shuffled-in component shares multiple boundaries with the rest of the target model.

[14, 8, 2] correctly map major features, however the mapping they provide may not map the natural part boundaries in a consistent manner and thus may not always preserve the geometric meaning of the parts. These methods operate on watertight models and thus cannot be used on many available models of man-made objects, such as chairs, which contain multiple connected components. Partial matching [7] focuses on finding models that contain a part similar to one particular part in the given model. Since selecting the best match for one part may enforce poor matches for other parts, our algorithm uses a global approach that considers all the correspondences at once.

3 Overview

The Shuffler editing system operates on commonly used classes of both natural and man-made objects. It works on both watertight and non-watertight meshes including models with any number of connected components, which are quite common in online databases. The system consists of a modeling interface and a pre-processor that segments the input meshes into meaningful interchangeable components. The pre-processing can be performed on the fly once the user selects the set of models they want to work with or can be carried out beforehand for a database of models.

Compatible segmentation: When considering a compatible segmentation of a large set of models, one option is to consider all the models at once, locating the interchangeable components which are present on all the models. However, this setup is very restrictive, as any component not present on one of the models would no longer be considered as an interchangeable component. Considering each pair of models completely individually is even more problematic, as it can lead to pairwise segmentations which have nothing in common and thus can not be combined in a shuffling setup. Therefore, we opt for a hybrid solution. We first segment all the models in each class into meaningful parts using the same level-of-detail threshold, providing very similar segmentations. We then use a matching algorithm that computes pairwise correspondences between models using the located parts. The algorithm aims at finding a one-to-one correspondence between parts of the two models, if one exists. For instance, it finds a complete part correspondence including that of geometric facial features between a lion and a cat (Figure 6). Since such a correspondence may not always exist, the method automatically groups parts into larger meaningful components and establishes a one-to-one correspondence between these larger components when necessary. For example, for a lion and a bull where a detailed matching is ambiguous (e.g., the lion's ears could correspond to either the bull's ears or its horns (Figure 5)), Shuffler groups the facial parts into a single "head" component and matches one head to the other. This approach allows us to use multiple pairwise correspondences during shuffling, enabling the user to shuffle components between models with different degrees of similarity. In the example of shuffling components between a cat, a bull, a camel and a lion (Figure 9 (a)) the detailed correspondence between the lion and the cat enabled the user to replace the ears of the lion with cat ears. At the same time, the user was able to use a much coarser correspondence between a lion and a camel to swap the legs. While theoretically, grouping can lead to contradictory components formed by individual matches, such as grouping a neck with the body in one match and with the head in another, we found that the constraint that the components represent meaningful pieces of the model makes such situations extremely rare.

Shuffling Interface: After loading a set of models the user first selects one model as *target* and then picks any number of components which should be shuffled-in from the other models. Given each shuffled-in component, the system uses the precomputed correspondences to locate the corresponding shuffled-out component on the target (Figure 2(a)). The algorithm then automatically swaps the components (Figure 2(b)) using the correspondences to align and blend the shuffled-in component with the rest of the model (Section 6). For expert users we provide a number of additional controls to fine tune the alignment.

4 Segmentation

Our segmentation method is designed to create a perceptually meaningful decomposition of input meshes into parts that can be successfully matched by the subsequent correspondence computation step. The approach we use is inspired by the work of Chaselle et al. [3], who used convexity as a measure of meaningfulness and generated exact convex decompositions. In our setup we search for approximate convex decomposition, where the approximation tolerance, or threshold, reflects the level of detail required in the segmentation. The NP-hardness proof of Chaselle et al. for computing the most compact convex decomposi-

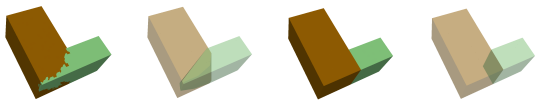


Figure 3. Part compactness (the semi-transparent images on the left show the part convex hulls): (left) convex parts created without the compactness metric; (right) parts generated by the segmentation using the combined part cost.

tion extends to the approximate setup. However, we found that using an incremental constrained Lloyd-type method we obtain segmentations which are close to optimal in terms of the number of parts generated and the part compactness (Figure 4).

4.1 Metrics

Before describing the algorithm we define the metrics of part convexity and compactness used to obtain the desired segmentation.

Convexity: Part convexity is the main criterion for our algorithm. We measure it as the distance between the part P and its convex hull $H(P)$. The distance is defined as an area weighted average of the distances from the part triangles t to the convex hull:

$$conv(P) = \frac{\sum_{t \in P} dist(t, H(P)) \cdot area(t)}{\sum_{t \in P} area(t)}, \qquad (1)$$

where $area(t)$ is the area of the triangle t, and $dist(t, H(P))$ is the distance from the triangle t to the convex hull $H(P)$. Specifically, $dist(t, H(P))$ 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.

Compactness: Our primary objective is to segment the mesh into nearly convex parts. However, to achieve a useful segmentation, it is not enough for the parts to be convex, they must also be compact. In particular, we want to prevent random segmentation in areas that can be equally well occupied by more than one neighboring part, thereby avoiding the possibility of complex part boundaries (Figure 3 (left)). To evaluate the compactness we consider the convex hull of the part. When the part is close to convex, its convex hull captures well the volumetric shape of the part. Therefore, we introduce a metric aimed at the generation of compact,

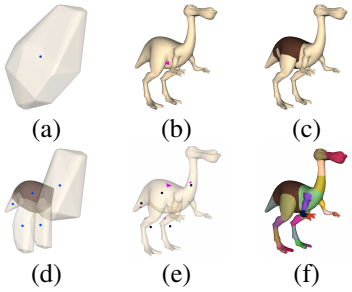


Figure 4. Segmentation stages: (a) the hull of the first potential part (the entire model) and its center; (b) first seed triangle; (c) first part after convergence; (d) hulls of the existing part and the new potential parts; (e) new seed triangles; (f) final result.

relatively spherical convex hulls. We calculate the compactness as an area to volume ratio of the convex hull $H(P)$,

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

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 part. We discuss the implications of this observation on the algorithm in the next section.

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

$$pCost(P) = (1 + conv(P)) \cdot (1 + comp(H(P)))^{\alpha}, \quad (3)$$

where α controls the trade-off between the two. We experimented with different values of α and found that $\alpha = 0.007$ appears to provide an optimal balance between convexity and compactness measures. We use this value in all the examples in this paper. 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.

4.2 Part Formation

Given these metrics, the goal of our decomposition algorithm is to segment the model into a small number of nearly convex, compact parts such that the convexity error (Equation 1) for each part is below the specified threshold D_{max}. The threshold values we used for each class of models are listed in Table 1. To generate the parts we use a Lloyd iteration scheme [17, 4]. In contrast to previous Lloyd-type methods we do not expect the user to provide an estimate number of parts for the segmentation. Instead we use the threshold to control the number of

parts. Starting with zero parts, our algorithm generates the parts incrementally using the following four step procedure.

1. **Potential part generation** collects the unassigned triangles, which do not belong to any part, into connected components and classifies each component as a potential new part. A triangle can be unassigned either at the beginning of the algorithm or during part growing when all parts have reached the convexity bound D_{max} (Figure 4 (c)). Both possibilities are a good indication that a new part is required in the unassigned region. This method of formation of new parts allows us to implicitly handle models with multiple components (Figure 10 (d) and (e)). After forming the potential parts, the method computes the convex hull for each of them (Figure 4 (a) and (d)).

2. **Seed generation** for both existing and potential parts selects the seed triangle $t \in P$ that minimizes the following cost function:

$$sCost(t) = (1 + dist(t, H(P))) \cdot (1 + comp(h))^{\alpha},$$

where h is the seed convex hull and $H(P)$ is the current convex hull (Figure 4 (b) and (e)). We define the seed convex hull h as the tetrahedron formed by the seed triangle and the center of the current convex hull, highlighted in Figure 4 (a) and (d). The described choice of seed hull construction explicitly avoids the formation of zero-volume hulls preventing instability in the behaviour of our compactness metric. The minimized function is nearly identical to the part cost function (Equation 3), facilitating the convergence of the Lloyd iterations.

3. **Part growing** is performed on all the parts simultaneously using a Dijkstra search algorithm to find at each step the best adjacent vertex to add to one of the parts. The insertion cost of a vertex is defined as the cost of the potential part formed by adding v to P, $pCost(P + v)$. The growth of each individual part is terminated when any vertex addition will cause the convexity error $conv(P+v)$ to go above the threshold D_{max}. Growing is repeated until all triangles are assigned to parts or until parts can no longer grow without violating the threshold.

4. **Termination:** If the new parts differ from the ones grown in the previous iteration, the algorithm repeats the reseeding and growing loop (Stages 2 and 3). Once parts no longer change we check if they cover the entire model. If they do the algorithm terminates, otherwise the algorithm returns to Stage 1.

Our algorithm typically takes only two to three iterations to converge each time new parts are added. Thus the dominant component of the algorithm runtime is the part growing, which requires the use of incremental convex hull construction to measure the cost of adding a vertex to a part. On a 3 GHz Pentium IV the method takes 25 seconds to segment the $10K$ faces lion (Figure 5). The algorithm can be sped up by using a progressive mesh, using a similar approach to [16].

4.3 Improving Compactness

The parts generated by the Lloyd algorithm satisfy the convexity threshold, and are on average well shaped. However we observe that in some cases the compactness of the patches can be significantly improved while only slightly worsening convexity. After the algorithm's convergence the level of compactness is directly linked to the amount of convex hulls overlap (Figure 3 left). Reducing the overlap between the hulls is tantamount to improving compactness. To measure the extent of the overlaps we consider the sum of volumes of the convex hulls of adjacent parts. Since this sum double counts the volume of the overlapping region, reducing it implicitly reduces the overlap. The improvement algorithm considers the triangles adjacent to the boundary between the parts and recursively reassigns triangles from one part to the other if the reassignment reduces the sum of convex hull volumes and does not violate the convexity threshold for either part.

When the overlap between hulls is minimal the boundaries between the parts are typically fairly straight. If users are interested in even straighter boundaries, methods which modify the connectivity of the input [18] can be applied as a postprocessing step. As explained in Section 6 our shuffling interface uses boundary midpoints and part hulls to facilitate component alignment and uses the actual location of the boundary only to assign blending weights. Thus, the shape of the part boundaries has little influence on the modeling results, and such straightening is unnecessary.

4.4 Hierarchical Segmentation

Hierarchical segmentation, where coarse parts are unions of finer parts, is useful for several processing applications [21]. In our setting, it helps speed up the part correspondence computation and increase its robustness. Our algorithm computes a hierarchical segmentation from the bottom up. Given a set of increasing convexity threshold values, it first computes a fine segmentation using the smallest threshold. It computes the subsequent hierarchy layers incrementally, using as input the segmentation in the previous layer. Given the new convexity threshold it recursively merges adjacent parts if the combined part satisfies

this threshold (Figure 5 (a),(b)). Our matching algorithm uses a two level hierarchical segmentation, where the coarse segmentation uses $D_{max} = 1\%$ for all the example models in this paper.

5 Component Correspondence

The correspondence algorithm matches the interchangeable components between pairs of segmented models. It does this by optimizing a cost function which takes into account both the meaningfulness of each component and the quality of the matching between components. Like in the segmentation stage we use convexity as a measure of meaningfulness. We define matching as high quality if the approximated geodesic distances between matching components on both models are roughly the same. The method uses the convex hulls of the parts as proxies for all computations. Since we have no prior registration of the models, to enable comparison between them, we scale the models to be of equal size, based on volume for articulated models, or bounding boxes for rigid ones. The volume is computed as the sum of part hull volumes.

5.1 Matching Cost

To describe the cost function we first introduce some terminology. We denote the parts of the first model as P_{11} to P_{1n}, and those of the second as P_{21} to P_{2m}. The components C_{11} to C_{1k} on the first model have a one-to-one correspondence to the components C_{21} to C_{2k} on the second, where the matching maps C_{1i} to C_{2i}, $\mu(C_{1i}) = C_{2i}$. A component C_{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 (d)). Our cost function is based on two main components: *midpoint graph distances* and *component convexity*.

Midpoint Graph Distances: We define the distance between two components $d(C_{ia}, C_{ib})$ as the shortest distance between their parts on the midpoint graph. Note that using this definition the distance between adjacent components 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. In contrast to Euclidean distances between part centers these distances are well preserved between different poses. The difference of distances metric is defined as

$$mgd(\mu) = \sum_{a=1,b=1}^{k} (d(C_{1a}, C_{1b})/s_1 - d(C_{2a}, C_{2b})/s_2)^2,$$

$$s_i = \sum_{a=1,b=1}^{k} d(C_{ia}, C_{ib}).$$

Due to the normalization by s_i, the metric remains comparable across matches with a different number of components or different groupings.

Component Convexity: This metric measures the quality of the part grouping and is measured independently on each model. Since components should remain meaningful, they should be as convex as possible. Each component already consists of a number of nearly convex parts, hence there is no need to explicitly compute its convexity by measuring the distance from the actual mesh component to its hull. Instead, it is much faster to evaluate convexity by measuring the volume of each component's convex hull, which indicates how much larger the component is than the sum of its parts,

$$cconv(\mu) = \sum_{j=1}^{k} volume(H(C_{1j})) + \sum_{j=1}^{k} volume(H(C_{2j})).$$

Cost function: The cost function we use also includes a volume similarity metric defined as

$$vol(\mu) = \frac{1}{k} \sum_{j=1}^{k} \left(\frac{volume(C_{1j}) - volume(C_{2j})}{volume(C_{1j}) + volume(C_{2j})} \right)^2$$

to evaluate the correspondence in terms of component volumes $volume(C_{ij})$, defined as the sum of volumes of the convex hulls of the parts in the component. This metric is not a good indicator of similarity on its own, as models in the same class can have very different proportions, however it is often useful for matching fine-level details. Thus, we include this metric in the combined cost function, but give it a relatively small weight.

The combined cost function is

$$mCost(\mu) = (1+mgd(\mu))(1+cconv(\mu))(1+vol(\mu))^{\beta}. \quad (4)$$

Note that the first two terms are always given equal weight. This proportion was determined empirically as optimal. We investigated the influence of β, the weight given to the volume similarity, on the obtained matches and found that providing different values for different classes of models leads to better results (Table 1). To find the best correspondence the algorithm described below looks for the global minimum of the cost function.

5.2 Coarse Level Matching

Finding a local minimum of the cost function is insufficient for a shuffling application as it may contain unacceptable choices of interchangeable parts. Therefore we employ a global approach known as stochastic local search [12] to locate the global minimum. The stochastic local search

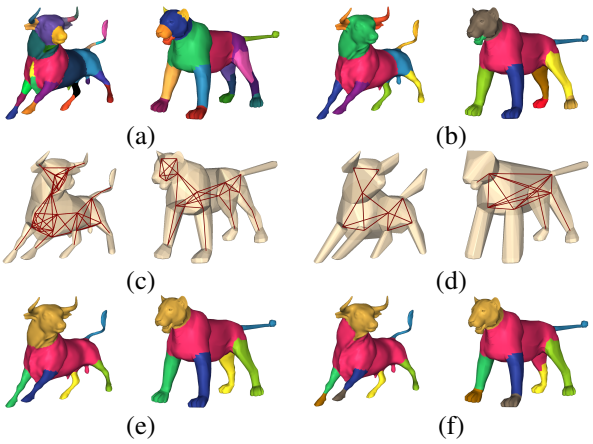


Figure 5. Segmentations and matches: (a)(b) fine and coarse level segmentations; (c) (d) midpoint graphs for the fine and coarse segmentations; (e) coarse level correspondence; (f) fine level correspondence.

combines randomized selection of multiple initial guesses with a steepest decent minimization to compute local minima starting from those guesses. At the limit, when run an infinite number of iterations, the search is guaranteed to find the global minimum of the cost function. In practice, a small number of initial guesses is typically sufficient. To make the process more robust and efficient, we utilize the hierarchical segmentation of the models. The algorithm uses stochastic local search to establish correspondences between the components at the coarse level of the hierarchical segmentation (Figure 5 (e)) and then refines those.

Randomized Initial Guess: To initiate a new search we start by randomly selecting an initial guess that consists of three parts: the number of components, the components themselves, and an initial correspondence. During the coarse level matching the system uses upper and lower bounds on the number of components, defined per class of models. The upper bound is slightly higher than the typical number of anticipated coarse components for the class. For instance, for quadrupeds we typically have seven coarse components: body, head, tail, and four legs (Table 1). The upper bound we use is nine. The lower bound is provided to speed up the process and avoid testing all groupings between two and the upper bound. Note that the upper bound is enforced only during coarse level matching. The number of components generated by the fine level matching is unbounded. The number of components is set randomly with equal probability for any number between the upper and lower bounds. The number of components is the same for both models, and remains constant during the local search. The selection of parts that form each initial component is also randomized, but with higher probability given to better groupings based on component convexity $mConv(\mu)$. 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 switches the correspondences between two pairs of components $C_{2i} = \mu(C_{1j})$ and $C_{2j} = \mu(C_{1i})$. The regroup operation has two stages and is performed separately for each model. First, a new component is formed by separating one part from a component containing multiple parts. Next, two components are merged; thus the total number of components remains constant. The local search uses steepest decent, at each step performing the atomic operation that improves the cost the most. If neither swaps nor regrouping can improve the current match, the process terminates.

5.3 Refinement

After obtaining the optimal match at the coarse level, we switch to the fine level in the segmentation hierarchy (Figure 5 (a),(c)). Since the coarse parts were obtained by merging fine parts, the component structure of the match remains unchanged while the number of parts in each component increases. Following the switch, the algorithm performs one iteration of the local search using the final coarse correspondences as an initial guess. This iteration typically improves the part grouping, moving some of the fine parts from one component to an adjacent one (Figure 6 (c)).

Finally, the method tests if the matching can be refined by constructing correspondences between sub-components of the matching coarse components. For each pair of corresponding components and each possible number of sub-components the method finds the sub-component match that is optimal in terms of both the overall cost and sub-component convexity. If this match is an improvement on the coarse match in terms of cost then the algorithm updates the match to include the sub-component correspondences. The process is repeated recursively for each pair of matching components (Figure 6 (d)). We found that the constraint of considering both overall cost and convexity improves the matching results at the fine level, preventing undesirable fine correspondences.

5.4 Registration

Our inputs are not aligned prior to processing. Given the variety of poses and shapes for many of our models, global registration beforehand is unreliable. However, to use the results of the matching to facilitate composition, we need to be able to position the model components with respect to one another. To align the models, we use the point-to-point correspondence between the centers of the hulls of the matching components. The algorithm finds the best similarity transformation between the centers using the quaternion

(a) (b) (c) (d)

Figure 6. (a) Top coarse match for the cat and lion, prior to registration. (b) Top coarse match after registration. (c) Top match on the fine level after one iteration of local search (the matching of the thighs is now correct). (d) Final fine level match.

method of Horn [13].

The registration serves another important purpose, as it allows us to detect matches with global mirroring (Figure 6 (a)). For symmetric models, a cost function based on distances and volumes, like ours, typically assigns low cost to both the correct and the mirrored results. During the stochastic search instead of only storing the best match found, we store the top one hundred matches obtained. After the search terminates, for each of the stored results, we use the correspondences to align the models. We then use the alignment to recompute the cost of the stored matches taking registration cost into account,

$$reg(\mu) = \frac{1}{k}\sum_{j=1}^{k} \|c_{1j} - c_{2j}\|^2$$

$$mCost'(\mu) = mCost(\mu)(1 + reg(\mu))^{\gamma}.$$

Following empirical testing, we used $\gamma = 0.5$ for nearly all model classes. For planes, which are both flat and symmetric, a one hundred and eighty degree rotation gives a very small registration error. Hence, to avoid such rotations, we used a higher registration weight (Table 1). We then select the best match as the one that minimizes $mCost'$. To speed up processing the registration and subsequent re-evaluation are performed using the coarse level matches, prior to refinement (Figure 6(a),(b)).

The matching algorithm complexity is a function of the number of model parts, which is typically in the double digits. While the search space is exponential, we found that our stochastic local search approach efficiently locates the global optima using just a hundred initial guesses. For example, the algorithm only 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 Shuffling Interface

The shuffling interface utilized the component correspondences to provide a mouse-click based composition interface. After a user loads the models of interest, they select a target model from which to start the processing. They can subsequently shuffle-in components from the other *source* models by simply clicking on them. As the components are swapped, the system automatically aligns each shuffled-in component with the rest of the target model. If desired, it

(a) (b) (c) (d)

Figure 7. Chair seat shuffle: (a) source (top) and target (bottom) chairs showing the components to be shuffled in and out in dark blue; (b) a common adjacency graph for both models constructed by our algorithm; (c) initial component positioning; (d) position after automatic alignment.

then zippers and blends the composed components together to create a watertight output model.

Alignment: The global alignment of source and target models (Section 5.4) establishes a fixed common frame which can be used as-is to specify the rotation and scale for the shuffled-in component. Alternatively, rotation and scaling can be performed when individual components are shuffled in and out, optimally aligning the convex hulls of the shuffled in and out components using geometric moments [6]. Since the components are nearly convex, the hulls provide a good approximation of their shape. We observe that using the global alignment better preserves the pose of the models typically leading to more visually intuitive results. Shuffler supports both alternatives, using the global alignment as the default.

Shuffler adjusts the translational alignment for each individual shuffling operation to ensure that the constructed model remains connected despite differences in individual component sizes. The alignment is computed using the *common adjacency graph* of the source and target models (Figure 7 (b)), which has a node for each pair of corresponding components. The graph has an edge between two nodes if and only if the corresponding components are adjacent on both models. Each edge of the graph is associated with a pair of midpoints defined as in Section 5.1. The midpoints serve as the connection points between neighbouring components. We observe that removing the node that corresponds to the shuffled-out component from the adjacency graph can break it into multiple connected components or branches (Figure 7). The algorithm translates each individual branch of the target model to align the shared mid-

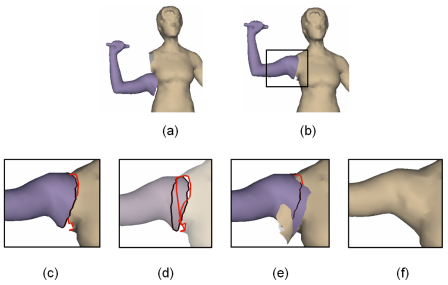


Figure 8. Component shuffle: (a) initial position of the shuffled-in component; (b) position after automatic midpoint alignment; (c) and (d) zoom in onto the boundary region; (e) formation of overlap region for blending; (f) final result.

points between the branch and the shuffled-in component (Figure 7 (c) and (d)). In case there is more than one midpoint between the branch and the shuffled-in component it uses the average of the midpoints to calculate the translation. For additional user control we provide an interface to manually adjust the alignment if desired. We observe that using the common adjacency graph Shuffler can compose multiple mesh components in a single operation, something other existing composition algorithms do not support.

For models which do not need to be watertight, such as tables or chairs, the process ends here. For watertight models we use a method similar to Sharf et al. [22] to blend the components along shared boundaries. We first extend each boundary by a few layers of triangles (Figure 8) forming an overlap region and then use a variant of soft ICP to snap the two meshes together and generate a common connectivity.

7 Results

In Figures 1 and 9 we demonstrate several models created using our system. For each of the models it took the user only a few mouse clicks to specify the combination of components they want. In the camow (Figure 9 (b)) and two of the chair models (Figure 9 (i)) the user adjusted the alignment of the components to achieve the desired composition effect.

We tested the combined segmentation and matching approach for compatible segmentation on several dozen models, computing over a hundred pair-wise correspondences using the per-class parameters defined in Table 1. Several representative examples are shown in Figure 10. Figure 10 (a) shows three compatible segmentations computed between the cat and several other animal models with different degrees of similarity. As demonstrated the level-of-details in the obtained compatible segmentations depends on the level of similarity between the models, as desired. At the same time the obtained segmentations can be easily com-

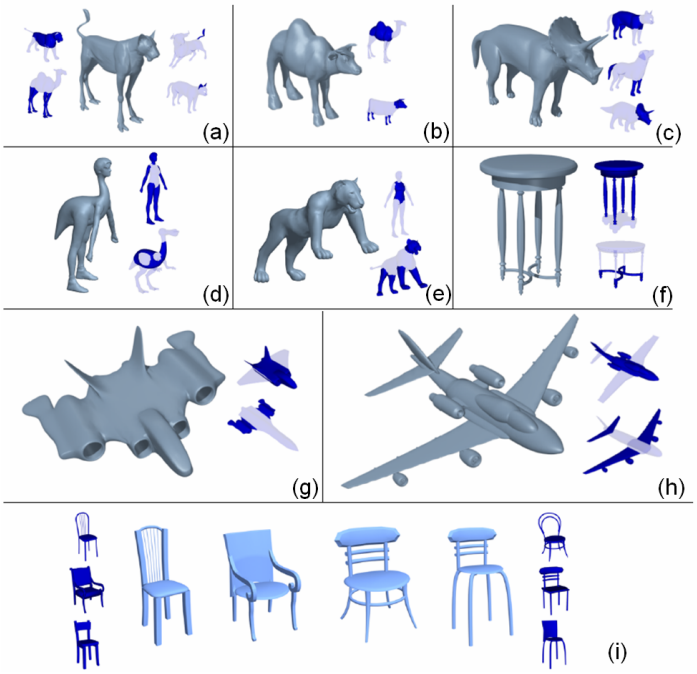


Figure 9. Shuffling: (a) cambuliot; (b) camow; (c) triceradog; (d) dinowoman; (e) shelion; (f) a table; (g) a stealth-jet; (h) a six engine super-jumbo; (i) a bunch of chairs.

Class	D_{max}	bounds on group # (coarse level)	β	γ
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.

bined by the shuffling interface as they share common component boundaries. Figures 10 (b)–(e) demonstrate a similar effect on compatible segmentations of animals, humanoids, planes and chairs. Note that our algorithm correctly computed compatible segmentation for the male and the David models (Figure 10 (c) bottom) despite the two models having different genus. In several shuffling examples (Figure 9) we created chimeras combining components from models in different classes for which perceptually correct correspondences exist. Our algorithm successfully computed the compatible segmentations for such models as shown in Figure 10 (f).

8 Summary

We presented a prototype modeling system, Shuffler, which allows users to generate detailed geometric models with a few mouse cicks. As part of the system we introduced a method for automatically computing compatible segmentation of models into interchangeable components. We believe such segmentation can be used by many other modeling operations, beyond shuffling. We plan to investigate using the established correspondences to compute

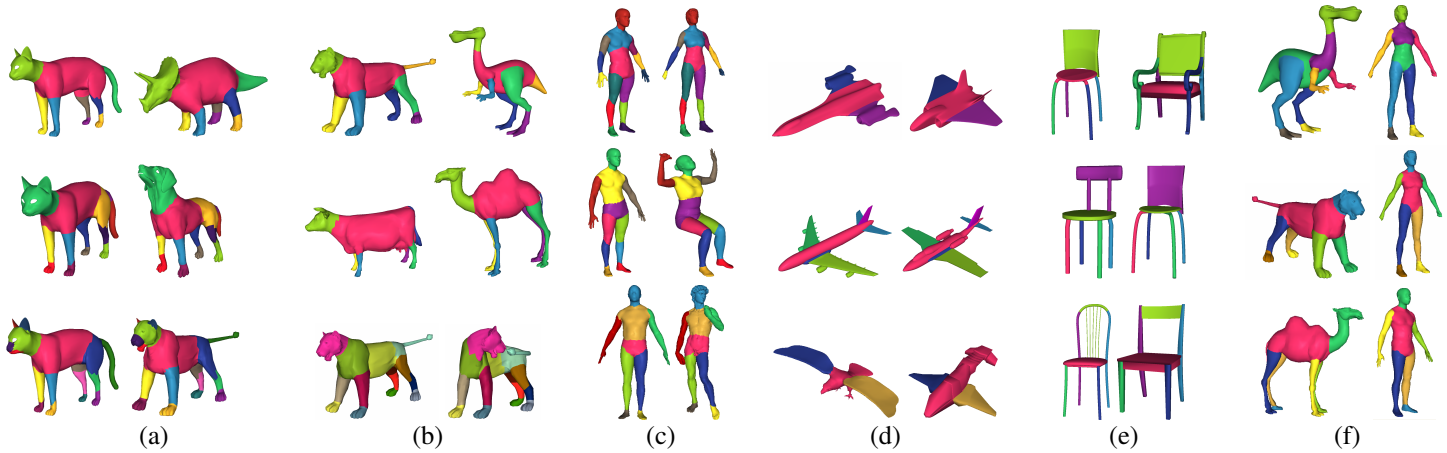


Figure 10. Compatible segmentations.

cross-parameterization between the components enabling local blending and morphing, as well as transfer of skeletons and associated animations.

We observe that our definition of parts is purely geometric and does not account for semantics, thus it is not suitable for processing shapes such as faces where some parts (cheeks, forehead, chin) have no clear geometric boundaries in the sense defined by Hoffman and Richards [11]. In our work we did not explicitly consider symmetry, thus the computed segmentations are not necessarily symmetric. Given the recent advances in detecting symmetries in 3D, it would be interesting to introduce symmetry constraints into our system.

Acknowledgments

We appreciate the editing help and comments of Ciaran Llachlan Leavitt. We thank the Stanford Graphics Laboratory for providing the David model. We thank NSERC, MITACS NCE, and IBM for their generous support.

References

[1] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation - a comparative study. In *Proc. Shape Modeling International*, 2006.

[2] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching. *Proc. National Academy of Sciences*, 103(5):1168–1172, 2006.

[3] B. Chazelle, D.-P. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. In *Symposium on Computational Geometry*, pages 297–305, 1995.

[4] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, 2004.

[5] N. Cornea, M. F. Demirci, D. Silver, A. Shokoufandeh, S. Dickinson, , and P. Kantor. 3d object retrieval using many-to-many matching of curve skeletons. In *Shape Modeling and Applications (SMI' 04)*, pages 366–371, 2005.

[6] M. Elad, A. Tal, and S. Ar. Content based retrieval of vrml objects: an iterative and interactive approach. In *Proceedings of the sixth Eurographics workshop on Multimedia 2001*, pages 107–118, New York, NY, USA, 2002. Springer-Verlag New York, Inc.

[7] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3):652–663, 2004.

[8] R. Gal, A. Shamir, and D. Cohen-Or. Pose oblivious shape signature. *IEEE Transactions of Visualization and Computer Graphics*, 2006.

[9] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust global registration. In *Symposium on Geometry Processing*, 2005.

[10] T. Hassner, L. Zelnik-Manor, G. Leifman, and R. Basri. Minimal-cut model composition. In *Shape Modeling and Applications (SMI' 05)*, pages 72–81, June 2005.

[11] D. D. Hoffman and W. A. Richards. Parts of recognition. *ACM Trans. Graph.*, 18(1–3):65–96, 1984.

[12] H. H. Hoos and T. Stutzle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[13] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, 1987.

[14] V. Jain and R. Zhang. Robust 3d shape correspondence in spectral domain. In *Shape Modeling and Applications (SMI' 06)*, 2006.

[15] O. A. Karpenko and J. F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Transactions on Graphics*, 25/3:589–598, 2006.

[16] S. Katz, G. Leifman, and A. Tal. Segmentation using feature point and core extraction. *The Visual Computer*, 21(8–10):865–875, 2005.

[17] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, 22(3):954–961, 2003.

[18] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des.*, 22(5):444–465, 2005.

[19] J.-M. Lien, J. Keyser, and N. M. Amato. Simultaneous shape decomposition and skeletonization. In *Proc. ACM symposium on Solid and physical modeling*, pages 219–228, 2006.

[20] Second Life. http://secondlife.com/.

[21] A. Shamir. Segmentation and shape extraction of 3d boundary meshes. In *State-of-the-Art Report, Proc. Eurographics*, 2006.

[22] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or. Snappaste: An interactive technique for easy mesh composition. *Visual Computer (Pacific Graphics 2006)*, 2006.

[23] Spore. http://www.spore.com/.

[24] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proc. Shape Modeling International*, page 130, 2003.

[25] H. Zhang and R. Liu. Mesh segmentation via recursive and visually salient spectral cuts. In *Proceedings of Vision, Modeling and Visualization*, pages 429–436, 2005.