EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

Volume 31 (2012), Number 2

Exploring Shape Variations by 3D-Model Decomposition and Part-based Recombination

Arjun Jain[1] Thorsten Thormählen[1] Tobias Ritschel[2] Hans-Peter Seidel[1]

[1]MPI Informatik [2]Télécom ParisTech (CNRS-LTCI)

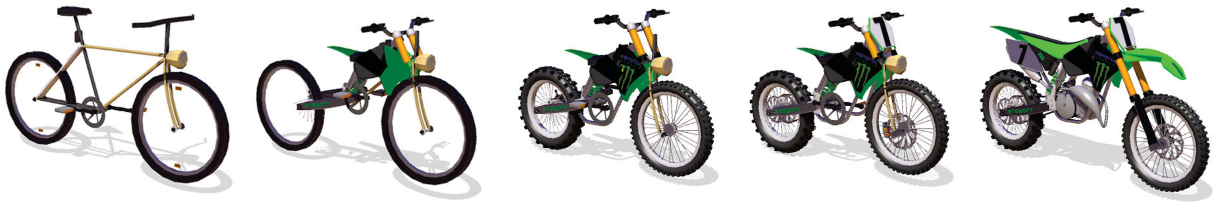

Figure 1: *A blend between a bike and a motorbike with different structure, produced without user intervention by our technique.*

Abstract

We present a system that allows new shapes to be created by blending between shapes taken from a database. We treat the shape as a composition of parts; blending is performed by recombining parts from different shapes according to constraints deduced by shape analysis. The analysis involves shape segmentation, contact analysis, and symmetry detection. The system can be used to rapidly instantiate new models that have similar symmetry and adjacency structure to the database shapes, yet vary in appearance.

1. Introduction

Easy-to-use shape modeling systems that produce custom detailed 3D models are hard to come by. Professional shape modeling tools are difficult to master, and detailed shapes take a long time to create. Thus, many non-expert users resort to simply choosing the most suitable existing 3D model from the increasing number of databases that are available on the Internet. Yet, it is often the case that no model in the database is entirely suitable, or that the user is looking for a custom model.

This work presents a system that can be used to synthesize new 3D models from a database of many-part shapes. The system is well suited for non-expert users because a blend between two database shapes, as shown Fig. 1, can be controlled via a single slider. Professional users can employ the system to conveniently and quickly generate a large number of shape variations, e. g., it is possible to produce a crowd of hundreds of re-combined individual robots from only a few database shapes. Mixing shapes of different classes could help artists to brainstorm new shapes, e. g., by mixing a boat and an airplane to produce a fast-looking boat (cp. Fig. 12).

The system aims at blending between models that are highly dissimilar from a conventional geometry processing point of view; they can have different numbers of parts, different mesh connectivity, and different topological structure. Our key simplification is to avoid varying the geometry inside individual parts that constitute the shape. Instead, we segment the database models into parts and synthesize new models by recombining these parts.

The key challenge of part-based recombination is to minimize the synthesis of undesirable variations. The number of possible models that can be synthesized from given parts grows combinatorially, and most such models are undesirable. The key assumption behind our system is that preservation of symmetries and contacts found in the source models can increase the desirability of the synthesized models. This constrains the models that the system generates, leading to more visually pleasing blends.

Our system processes a database by segmenting the shapes into parts and computing symmetries and contacts between these parts. Contacts between the parts describe the adjacency structure of the shape. The adjacency structure is employed during the creation of a hierarchy that groups connected parts in a coarse-to-fine manner. At runtime, given two database shapes that the user wishes to blend, the system performs hierarchical matching between the shapes, remixing parts that have similar positional information in the nodes of the hierarchy. The computed matching is used to interpolate the adjacency structure of the models. Individual parts are exchanged during the interpolation and are positioned by a mass-spring system that enforces contacts.This produces new models that incorporate parts from given shapes yet vary in appearance.

2. Related Work

Creating a new shape that interpolates two given shapes is called blending, or morphing. Morphing involves solving a correspondence problem and requires a blending operator. Finding correspondences is a difficult problem both between images [Wol98] and surfaces [BN92, LDSS99]. Blending becomes easier when choosing a suitable representation, such as distance fields [COSL98] and achieves more natural results when maintaining as-rigid-as-possible deformations [ACOL00].

Interpolating between multiple given models is even more challenging. The problem is simplified when a parametric model is available, as is the case for human faces and bodies [BV99, ACP03], but it is not known how to construct such parametric spaces given general models composed of many meaningful parts.

Another approach to shape synthesis is to generate instances based on statistical models. This is a popular approach for texture synthesis, where example instances are decomposed into multiple texture elements and then recombined into new instances [EL02]. This approach has been extended to synthesize large models given a smaller one [Mer07]; however, when synthesized instances must have a complex hierarchical structure, such as symmetry and preservation of physical constraints, statistical models based on local similarity are less effective, and some modeling of the global structure is required.

Procedural modeling systems have been used to describe the global hierarchical structure in shapes [Ebe03], but it is difficult to automatically derive a grammar for a given geometric model, despite promising recent steps [BWS10].

Modeling by Example [FKS*04] is a modeling system that can be used to generate a shape by manually cutting and gluing parts from existing shapes. Chaudhuri et al. [CKGK11] propose an example-based modeling system that expedites the modeling process by presenting relevant components to the user. Our goal is to further drastically simplify the modeling interface by synthesizing complete models automatically and efficiently, allowing the user to explore a whole family of new models with no direct manipulation of geometry. The *Shuffler* system [KJS07] enables users to replace selected parts in compatible models. Section 5 discusses similarities and differences to our system. To efficiently explore large

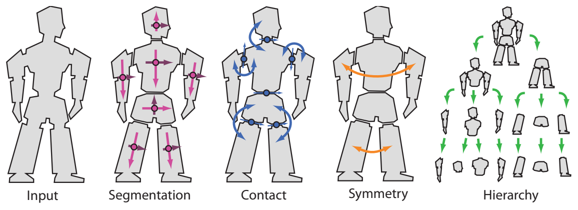

Figure 2: *After segmenting the input shape, we detect symmetry and contacts on multiple levels of a shape hierarchy.*

databases of 3D shapes, Ovsjanikov et al. [OLGM11] propose a navigation interface that allows users to find the desired shape in the database by interacting with a deformable shape template.

Higher-level shape analysis has received much interest with applications that include deformation [GSMCO09, ZFCO*11], abstractions [MZL*09], automated layout [LACS08], or upright orientation [FCODS08]. A crucial component here is the segmentation of meshes [SBSCO06, GF09, KHS10] and the detection of symmetry [MGP06, PMW*08]. Symmetries can also be organized hierarchically [WXL*11]; joints between shape parts can be automatically extracted [XWY*09]; and mechanical assemblies can be animated given only raw shapes as input [MYY*10].

3. Shape Analysis and Synthesis

Our approach consists of two phases discussed in this section. The offline phase is an analysis of a database of many 3D objects leading to a representation of shape-part relationships based on the hierarchical structure and contacts between parts. During the online phase, this representation is then used to synthesize new shapes from parts with relations similar to those in the database.

3.1. Shape Analysis

Shape analysis is used to find the relations between parts that constitute a shape. We start from $\mathcal{S} := \{S_i | S_i \in \mathcal{M}, i = 1, \ldots, n^{\mathrm{s}}\}$, where $\mathcal{M}$ is the set of n^{s} shapes S_i in the database. Each shape S_i is represented as a polygonal mesh. Our database currently comprises 280 different man-made objects, providing no symmetry or hierarchy information, which are taken from 3D model repositories on the Internet. These models typically have different scales, but it is a prerequisite that they have a consistent alignment to the global coordinate axes (as is typically the case for 3D models from Internet repositories). In particular, all 280 of our models have a consistent upright orientation.

The analysis is run on every shape in the database independently. It consists of segmentation, contact analysis, symmetry detection, and hierarchy generation (see Fig. 2). This section provides the details of every step.

Segmentation The i-th shape S_i is decomposed into n_i^{p} parts $S_i = \bigcup_{j=1}^{j=n_i^{\mathrm{p}}} P_{i,j}$ which are again polygonal meshes. In our

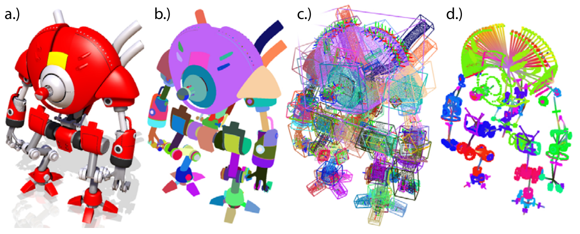

Figure 3: a.) *Shape,* b.) *Segmentation,* c.) *PCA,* d.) *Contacts.*

case, segments are connected components of the polygonal input mesh, which are generated by region growing.

Next, every part $P_{i,j}$ is re-sampled to a point cloud $\bar{P}_{i,j}$ for further processing. The individual points of $\bar{P}_{i,j}$ are placed on the surface in such a way, that their distance is roughly equal (blue noise). Now, a principal component analysis (PCA) of $\bar{P}_{i,j}$ is performed, which provides a transformation $\mathtt{T}_{i,j}$ from the global into the local coordinate system of the part. A point $\mathbf{p}'$ in the local coordinate system is given by a transformation $\mathtt{T}_{i,j}$, which combines a translation, a rotation and a scaling: $\mathbf{p}' = \mathtt{T}_{i,j}\mathbf{p} = \mathtt{S}_{i,j}\mathtt{R}_{i,j}(\mathbf{p} - \mathbf{c}_{i,j})$. The geometric center of the part's point cloud in the world coordinate system defines the center $\mathbf{c}_{i,j}$ of the local coordinate system. The local 3×3 rotation matrix $\mathtt{R}_{i,j}$ is given by the three PCA basis vectors and defines the local rotation axes. The diagonal 3×3 matrix $\mathtt{S}_{i,j} = \mathrm{diag}(1/s_\mathrm{x}, 1/s_\mathrm{y}, 1/s_\mathrm{z})$ describes the local non-uniform inverse scaling using the three singular-values $\mathbf{s}_{i,j} = (s_\mathrm{x}, s_\mathrm{y}, s_\mathrm{z})^\top$.

Contact Analysis. During contact analysis all intersections of all parts for a shape are found. For each part $P_{i,j}$ of shape S_i it is evaluated, if it is in contact with another part $P_{i,k}$. We call the subset of points in the point cloud $\bar{P}_{i,j}$ for which a point with a distance of less than 0.1 % of the bounding box diameter exists in $\bar{P}_{i,k}$ the *contact* $C_{i,j,k}$ of part j and k. In practice, an axis-aligned bounding box tree [vdB98] on all points of shape S_i is used to compute the set of contact points efficiently. The set of contacts describes the adacency structure of a shape.

In summary, the above two steps produce a segmentation of each shape into parts, as well as a list of contacts between the parts of a shape. An example is shown in Fig. 3.

Symmetry Detection. Next, the dominant global symmetry transformation $\mathtt{H}_i$ (a reflection, rotation, or translation) for the i-th shape is found using a RANSAC approach [BBW*08]. The approach randomly samples a number of potential symmetry transformations. In every trial, one symmetry transformation candidate $\mathtt{K}$ is generated. Then, the support $\alpha(\mathtt{K})$ of all parts for this symmetry candidate is computed. To this end, the center $\mathbf{c}_{i,j}$ of every part $P_{i,j}$ is mapped to $\mathbf{p}'_{i,j} = \mathtt{K}\,\mathbf{p}_{i,j}$, and if a matching part $P_{i,j'}$ is found at $\mathbf{p}'$, a support counter is incremented. Two parts match if their eigenvalues $\mathbf{s}_{i,j}$ and $\mathbf{s}_{i,j'}$ are similar, i. e., they are of similar shape. After all trials, the symmetry with the highest support count $\mathtt{H}_i = \arg\max_\mathtt{K} \alpha(\mathtt{K})$ is assumed to be the dominant symmetry.

To generate candidate transformations, the following procedure is used: for reflective symmetry, two random parts $P_{i,j}$ and $P_{i,k}$ are selected. The difference vector $\mathbf{d}_{i,j,k} = \mathbf{c}_{i,k} - \mathbf{c}_{i,j}$ between the part centers $\mathbf{c}_{i,j}$ and $\mathbf{c}_{i,k}$ defines the normal of a reflective symmetry plane, and a reference point on the plane is given by $(0.5\,\mathbf{d}_{i,j,k} + \mathbf{c}_{i,j})$. Translational symmetries are found by directly using $\mathbf{d}_{i,j,k}$ as a translational offset. For rotations, a third part $P_{i,l}$ is selected and a circle is fitted to the center of all three parts defining a rotation around a point by an angle.

In practice, we first compute the best reflective symmetry. If its support is below a threshold of 80 %, we assume no reflective symmetry and compute the best translational symmetry. If the support of this symmetry is below 80 % as well, the best rotational symmetry is computed. If rotational symmetry is supported by less than 80 %, no symmetry is assumed.

The result of this step is the one most dominant global symmetry for each shape (if present). Most shapes in our database exhibit a dominant reflective symmetry, which is therefore preferred by the described approach over other forms of symmetry. Since the manually modelled 3D objects are almost noise-free, the detection of symmetries is typically very robust.

Hierarchy. Finally, a hierarchy is generated for each shape S_i. This hierarchy is constructed in a coarse-to-fine manner and has approximately $\log(n_i^\mathrm{p})$ levels. On each level, parts are grouped into hierarchy nodes N, which know their children and store this information. The j-th node of the i-th shape at level k is denoted as $N_{i,j,k}$. On each level each node computes and stores its corresponding eigen-transformation $\mathtt{T}_{i,j,k}$ as well as its symmetry transformation $\mathtt{H}_{i,j,k}$, if it exists.

On the coarsest level, $k = 0$, all n_i^p parts are added to a single root node $N_{i,0,0}$. As the single root node on level 0 comprises all the parts of the shape, its eigen-transformation $\mathtt{T}_{i,0,0}$ corresponds to the eigen-transformation of the complete shape S_i and its symmetry transformation is given by the dominant global symmetry.

When going from a coarser level $(k-1)$ to a finer level k, for each parent node $P_{i,j,k-1}$ on the coarser level, two child nodes on the finer level are generated. Only those parts that belong to the parent node are now split into two sets and those sets are assigned to the two child nodes on the finer level. If a symmetry was detected, the splitting into children takes this information into account. For reflective symmetry, the reflective symmetry plane splits the parts of the parent node into two sets that are assigned to the two children. In a similar fashion, splitting planes can be defined for translational and rotational symmetry, e. g., for translational symmetry the splitting plane is located halfway on the translational offset vector. A third child is added in those cases where the centroid of a part is located in close proximity to the splitting plane. The splitting operation is successful

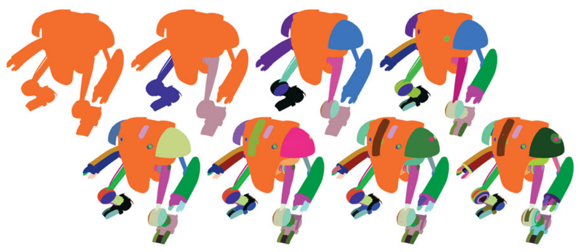

Figure 4: *Generation of a hierarchy that takes symmetry into account: Symmetric parts are grouped to nodes on the same level. The coarsest level of the hierarchy is shown top left, the initial part segmentation at the bottom right.*

if at least two child nodes have at least one part assigned. If the splitting based on the symmetry information is not successful or if no symmetry is available, the splitting plane $x = 0$ in the local coordinate system of the parent (defined by the parent's eigen-transformation $\mathtt{T}_{i,j,k-1}$) is used. If this splitting operation fails as well, the $y = 0$ and $z = 0$ planes are tested. If the operation is unsuccessful for any splitting plane, only a single child is generated for the parent node, where the child contains all the parts of the parent. Otherwise, if the splitting operation was successful, each child's corresponding eigen-transformation is computed and stored. Furthermore, the symmetry detection algorithm described in the last paragraph is applied to all the parts of the child node. This process is repeated for subsequent levels until a level is reached, where no further splitting operations can be performed. Fig. 4 shows the hierarchy for an example shape.

Enforcing Nodes without Disconnected Parts. At this stage, the hierarchy generation does not take into account contact information between parts. Consequently, it can happen that a child node contains parts that are disconnected, i. e., that there is no connection path between the parts over one or multiple contacts. In this paragraph we describe an algorithm which ensures that each node N of the hierarchy contains only non-disconnected parts. This algorithm is executed after the creation of each new level k in the hierarchy. Let's assume that after its creation the level k has n_k^{N} nodes.

In a *separating step*, a region-growing algorithm on the contact adjacency structure is used to decompose the parts of a child node on the current level k into sets of non-disconnected parts. For each set, we generate a new child node in the current level that contains the parts of the set. Thus, the original child node may be replaced by multiple new ones. After the separating step is performed for all original child nodes, we execute a *merging step* that tries to merge the child nodes (in order to obtain a similar number of nodes as the number n_k^{N} of nodes on the current level k in the original configuration). All the nodes of the current level are sorted by area, and the n_k^{N} largest nodes are kept. The other nodes are merged with a kept node with which they share at least one contact. If the merged nodes share contacts with multiple

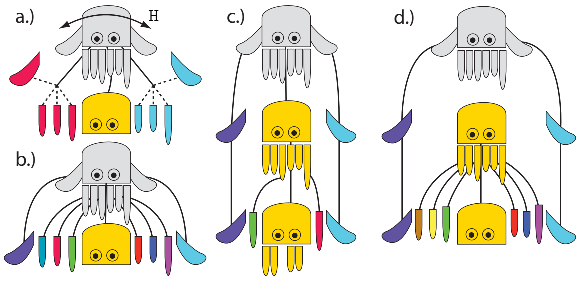

Figure 5: *Enforcing non-disconnected nodes:* a.) *Split along the dominant (reflective) symmetry* H *during the creation of the hierarchy may result in disconnected nodes (dashed lines).* b.) *In the separating step disconnected nodes are split into individual non-disconnected nodes* c.) *The merging step ensures that there are only a low number of additional nodes for each level. However, this leads to arbitrary decomposition of tentacles on the second levels (if the tentacles have very similar sizes).* d.) *In this particular case our merging step make an exception and separates all tentacles, which have similar size, at the same level into separate nodes.*

kept nodes, the smallest of the kept nodes is selected for the merge operation. During the merge operation, all parts from each merged node are added to the selected kept node and the merged nodes subsequently are deleted.

After the separating and merging step, we have the same number of nodes as before but it is ensured that each node only contains non-disconnected parts. There is, however, an exception. In cases where merged nodes are available that have very similar size as the smallest kept node (we used a threshold of 95 %), those nodes are also kept and not merged. This is necessary for the algorithm to produce the same results for different branches of the tree hierarchy because the selection by size would otherwise be arbitrary. Fig. 5 shows an example of the enforcement of non-disconnected nodes.

All of the above analysis is pre-computed and serialized to disk. It will be used to synthesize new shapes in the next step.

3.2. Shape Synthesis

The synthesis of new shapes is performed online at interactive speed using the pre-computed analysis results obtained from the previous steps. It consists of three steps: shape matching (Sec. 3.2.1), interpolation (Sec. 3.2.2), and contact enforcement (Sec. 3.2.3).

3.2.1. Shape Matching

While all previous steps were performed on individual shapes, in this step, a matching between two shapes S_1 and S_2 is established.

In general, the number of parts is not the same for each shape; thus, a one-to-one mapping for parts cannot be established. This problem could be resolved to some extent by

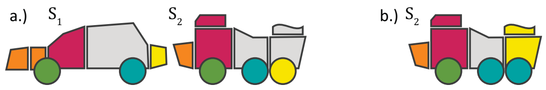

Figure 6: *Re-grouping parts (represented by polygons) into nodes (represented by the same color) of a new hierarchy can help to find a better match between two shapes S_1 and S_2:* a.) *original hierarchy for both shapes;* b.) *updated hierarchy of the target shape S_2.*

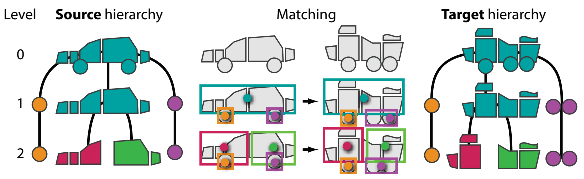

Figure 7: *During shape matching the target hierarchy is re-generated. The structure of the source hierarchy is reproduced for the target shape (if possible). The one-to-one mapping of the nodes on each level directly defines the matching between the two shapes.*

selecting a hierarchy level for both shapes where the number of nodes on both sides are the same, and performing matching between nodes on those levels. The shapes in our database, however, typically have a very different structure, resulting in a different hierarchy. As an example, let's assume the source shape is a car and the target shape is a truck, as shown in Fig. 6a. Both models have the same number of nodes, but the back of the car (yellow) has to be used as one of the truck's tires to generate a one-to-one mapping of nodes. Instead, we propose to rebuild a new hierarchy for the target shape that adapts to the observed segmentation of the source shape during matching (as can be seen in Fig. 6b).

Algorithm 1 Shape Matching

```
Build the target hierarchy root node
for all source hierarchy levels starting from level $k = 0$ do
    1. Copy the child node structure of the current source level into the target (empty target child nodes);
    2. Assign target parts to target child nodes by nearest neighbor matching, which compares the positions of the corresponding source child nodes with the positions of the target parts (all given in the local coordinate system of their parent node);
    3. If any target child node is empty, remove it from the target hierarchy and merge the corresponding source child node with another source child node;
    4. Enforce nodes without disconnected parts in the target hierarchy;
end for
```

An overview of the proposed shape matching approach is given in Algorithm 1. In the beginning, a target hierarchy root node is generated. As illustrated in Fig. 7, all target parts are assigned to the root node, which defines the new coarsest level ($k = 0$) of the target shape S_2. We then calculate the eigen-transformation $\mathtt{T}_{2,0,0}$ of the target root node. We also define that the source and the target root nodes are in correspondence, which means that they match onto each other.

The algorithm now iterates over all levels of the source shape. We start from the coarsest level ($k = 0$) of the source shape and, in each iteration, the next finer source level ($k+1$) is processed. For each iteration, a corresponding target level is created, so that the source and target hierarchies ultimately have the same number of levels.

In order to generate a level k of the hierarchy for the target shape S_2, the following steps are executed (cp. Algorithm 1): First, the same number of child nodes in the target hierarchy is generated as on the current level of the source shape. The child nodes in the source and in the target are defined to be in exact correspondence, i. e., the child nodes with the same indices match onto each other: $N_{1,j,k} \leftrightarrow N_{2,j,k}$. The nodes in the target shape are empty at the moment, i. e., they currently do not have assigned parts. The parent and child nodes in the target are now linked exactly the same, as the source parents are linked with their child nodes in the source hierarchy. This means each target parent links to the target child nodes with the same indices as its corresponding node on level $(k-1)$ in the source hierarchy links to its children.

In the second step, all parts of a target node are distributed among the children. Each target child node knows its corresponding source child node. The part assignment is based on nearest neighbor matching of the positions of a target part's center in the local coordinate system of the target parent, and the position of the corresponding source child node in the local coordinate system of the source parent. To be more explicit, let $\mathtt{T}_{2,p(j),k-1}$ be the eigen-transformation of the target parent node. The position of the part's center $\mathbf{c}_{2,j}$ in the local coordinate system of the parent is then calculated by $\mathbf{c}'_{2,j} = \mathtt{T}_{2,p(j),k-1}(\mathbf{c}_{2,j})$. Similarly, let $\mathtt{T}_{1,p(m),k-1}$ be the eigen-transformation of the parent of source child node $N_{1,m,k}$. The position of the source child node's center $\mathbf{c}_{1,m}$ in the local coordinate system of the source parent is given by $\mathbf{c}'_{1,m} = \mathtt{T}_{1,p(m),k-1}(\mathbf{c}_{1,m})$. During nearest neighbor-matching, target parts are assigned to the target child node with the smallest distance between the local position of the corresponding source child node $\mathbf{c}'_{1,m}$ and the local part position $\mathbf{c}'_{2,j}$ (cp. Fig. 7). Afterwards, the eigen-transformation for each child node of the target is calculated.

In the third step, the special case where a target child node has no attributed target part is considered. Here, the target child node is removed from the target hierarchy. Consequently, to ensure an exact one-to-one mapping, the corresponding source node has to merge with the closest child node of its parent that still has at least one target part in the corresponding target node. As a result the source hierarchy is modified.

Similar to the shape analysis procedure, nodes in the target

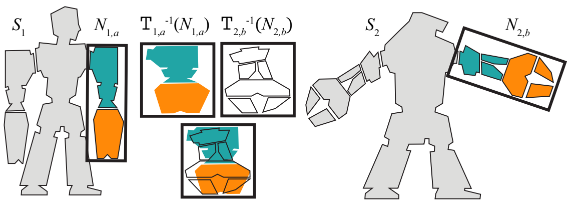

Figure 8: *Matching parts inside their parent nodes $N_{1,a}$ and $N_{2,b}$ (black boxes) of two shapes S_1 and S_2 (left and right humanoid). The parts in each node are transformed into the unit cube by the inverse eigen-transformation $\mathtt{T}_{1,a}^{-1}$ and $\mathtt{T}_{2,b}^{-1}$ and matched to the nearest neighbor.*

shape hierarchy might have disconnected components. Thus, in the fourth step, we run the exact same algorithm as in the shape analysis to ensure that parts in a target node are not disconnected, with one exception: it is always ensured that the number of nodes on the same level does not change (during the shape analysis, we allow to create more nodes if nodes selected for merging have very similar sizes as kept nodes).

In summary, after the algorithm has processed all source levels, the source and target hierarchy have the same number of levels and the same number of corresponding nodes on each level; however, the individual nodes $N_{1,j,k}$ and $N_{2,j,k}$ may have different numbers of assigned parts. The matching between source and target shape is directly given by the correspondence between the nodes $N_{1,j,k} \leftrightarrow N_{2,j,k}$.

This algorithm only takes positional information into account; however, the positions are compared in the local coordinate systems of the parent node. Thus, the shape of the parts that were assigned to the parent play an important role, as those define the parent's local coordinate system. Let's assume we want to match the arms of two humanoid shapes as shown in Fig. 8. If the matching on the previous level was successful, each part of the arm is now defined in its local coordinate system and part matching and conjoined re-grouping will return a reasonable result. This approach of conjoined matching and re-grouping is able to handle the difficult problem of matching parts that have different amounts of detail (like the arm in Fig. 8). This is demonstrated in Fig. 9 with two 3D aeroplane models from our database. The source shape, a lear jet, has no turbines on the wings, and thus a wing is modelled with only a few parts. The target shape, an MD-11, has turbines at the wing, and the complete wing (including the turbine) exhibits a large number of parts. As can be seen in the matching results of Fig. 9, our algorithm handles this situation by just grouping the wing and the turbine of the target shape in a single node in order to adapt to the source shape.

This approach can be extended to not only consider spatial information when attributing a target part to a target node. For example, parts with similar shape or size can be assigned with higher likelihood; however, as the models in our database have rather different shapes, especially in the fine details, we resorted to taking only positional information into account.

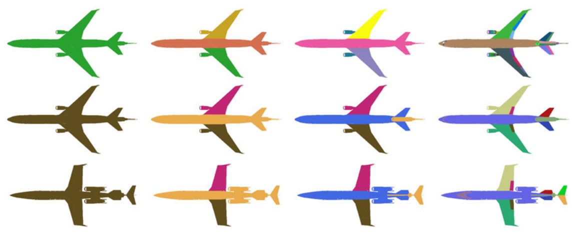

Figure 9: *Result of the matching and conjoined re-grouping of nodes. The source shape is a Learjet and the target is a McDonnell Douglas MD-11;* Top row: *Input hierarchy of the target,* Left: *The three coarsest levels,* Right: *finest level;* Middle row: *resulting hierarchy of the target;* Bottom row: *resulting hierarchy of the source (same colors identify matching parts for the lower two rows).*

3.2.2. Shape Interpolation

Using the shape matching described previously, allows to interpolate shapes composed of parts, i. e., to generate „in-between" composition of parts into shapes.

Linear Interpolation. A new shape $S(w)$ can be generated depending on a *weight* parameter $w \in [0,1]$ to blend between two shapes S_1 and S_2. As it is customary for interpolations, it is clear that $S(0) = S_1$, and $S(1) = S_2$. The question is what would be expected, e.g., at $S(0.5)$?

In our approach, shapes are interpolated using the nodes on the finest level of the hierarchy. The employed shape matching ensures that matching nodes have the same index j in the source shape S_1 and in the target shape S_2. During a blend, every part from S_1 should be replaced only once by a part from S_2 and never change back. This can be achieved by instantiating the first $w \cdot n_l^{\mathrm{N}}$ nodes (and all containing parts) from S_2 and all others from S_1:

$$P_j(w) = \begin{cases} P_{2,j} & \text{if } j < w \cdot n_l^{\mathrm{N}} \\ P_{1,j} & \text{else} \end{cases}$$

If two nodes in the source or in the target are symmetric, it is ensured that either both or neither are instantiated. The result depends now to some extent on which index j was assigned to a node because, during a blend, a node with a small index is instantiated earlier for an increasing w. Thus, we have chosen to reassign the indices by sorting the corresponding nodes by the conjoined source and target node size. Obviously, the reassignment of indices must be done similarly in the source and the target hierarchy to ensure that the matching of nodes (which is encoded in the index) is not lost. Our chosen reassignment places the node with the largest conjoined size in the middle of the blend, at $w = 0.5$, and node sizes are decreasing towards both sides $w = 0.0$ and $w = 1.0$. This choice typical gives visually pleasing blends. However, a user of our system can choose from several different sorting criteria, if desired.

Incremental interpolation of multiple shapes. Linear interpolation generates classic morphs between two shapes. Interpolation between more than two shapes is performed by executing multiple consecutive blends between two shapes. Starting from an existing shape S_1, this shape is modified by an incremental step of size $w_{1\to2}$ to become more similar to another shape S_2. This is achieved by simply exchanging parts in S_1 by a number of parts in S_2 proportional to $w_{1\to2}$, as described above. The blended shape $S_{1\to2}$ can then serve as a source shape for the next blend operation with a third shape S_3, and so on.

3.2.3. Contact Enforcement

While the interpolation rule tells us which nodes from which model should be instantiated, the exact positioning of the node's parts in the new shape must still be optimized. A good placement can be found by enforcing the contacts between nodes as these were observed in the source and in the target shape. This may impose multiple conflicting positioning constraints and a consensus needs to be found. Nodes that were in contact in the source and target shape should be attached to each other in the interpolated shape as well (Fig. 10). A node (from the source or from the target) usually tends to be in contact with its contact partner part that originated from the same shape. If this partner is unavailable, the node will try to enforce a contact with a node from the other shape that forms a match with the original contact partner. As shown in Fig. 10, it will often occur that nodes have a mutual desire to connect to each other. These bi-directional contacts will be enforced in any case. If the wish to form a connection is only unidirectional, such contacts are only enforced if they are not in conflict with bidirectional contacts (cp. Fig. 10).

We employed a simple mass-spring system to enforce the contact constraints between nodes. This system is a set of masses with locations and a set of springs that connect masses. The system to enforce our constraints is set up as follows: one mass is created in the node's center and one for every contact to another node. A spring is then generated between every node's center and each of its contact points. The contact point of each node with a connected node is calculated by averaging all the contacts points of the individual part contacts $C_{i,j,k}$, where part j is a member of the current node and part k is a member of the connected node. The springs within a node want to keep their length and enforce that the distance between the center of a node and its node contacts is maintained. We then add zero-length springs between contacts of different nodes that should be enforced. These springs seek to reduce their length to zero and are consequently pulling nodes towards each other. The mass-spring setup for a simple example is shown in Fig. 11. The mass-spring system is solved in a Jacobi fashion by successive over-relaxation [MHHR07]. Here, in each iteration, both masses connected by a spring are moved to make the spring come closer to its rest length.

Note that a mass-spring system (or an equivalent optimization procedure) is required for contact enforcement, as

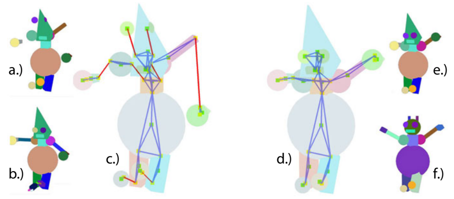

Figure 11: *A simple example demonstrating the mass-spring contact enforcement:* a.) *interpolated shape without mass-spring,* b.) *source shape,* c.) *mass spring system,* d.) *solved mass spring system,* e.) *solved shape and* f.) *target shape.*

a simple top-down relation between nodes (and their parts) does not exist. The contacts between nodes do not, in general, form a tree but a graph. For shapes that have the structure of a tree, children would just need to follow their parents. While many man-made objects appear as trees on a coarse level, in their details they are indeed graphs.

4. Results

In this section, we present the results that were generated with our system. More results are shown in the supplemental video. Users of our system can create new shapes by blending between database shapes. To this end, we offer a simple and very intuitive user interface. When the user clicks the *new* tool button, icons of available models in the database are shown. The shapes are organized into categories (e.g., "robots" or "ships") in order to help browsing the database. The user then selects a shape that should be used as the starting point of the exploration process. To create a blend, the user clicks the *blend* tool button and selects the target shape of the blend operation. Now a single slider appears which lets the user control the blending process. The slider varies the weight w of the linear interpolation between the source and the target shape (cp. Section 3.2.2). Fig. 12 shows some results of such blending operations for different source and target shapes. The results shown here are blends for complex shapes with several hundreds of parts. More results for shapes with only a few parts and from different classes (e. g., tables, lamps, or cars) are given in the supplemental material. Due to the precomputed shape analysis, the blends can be performed at interactive rates with high-quality rendering feedback. Interactive user sessions are shown in the supplemental video. The two robots in the top row of Fig. 12 both have ≈ 2.5 M polygons. Recomputing their shape analysis takes approx. 10 s each, shape matching 720 ms, and the rest of the shape synthesis runs at 5 fps. Fig. 13 shows a matrix of blend results for a weight of $w \approx 0.5$. The matrix contains blends for all combinations that are possible to generate between five different robots in our database. More matrices for different shapes are provided as supplemental material. Fig. 14 shows a result for incremental interpolation of multiple shapes.

It is also possible to reproduce to some extent the functionality known from the *Shuffler* system [KJS07], which allows

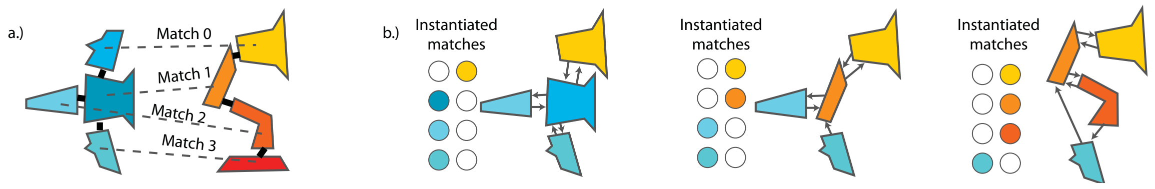

Figure 10: *Finding contacts for interpolated shapes:* a.) *for the source shape and the target shape it is known which nodes are in contact by the shape analysis;* b.) *to generate an interpolated version between the source and the target shape, either the node from the source or from the target shape can be instantiated for each match. An instantiated node wants to connect to its contact partners that originated from the same shape; however, if a contact partner is not available, a node connects to the node of the other shape with which the original contact partner forms a match.*

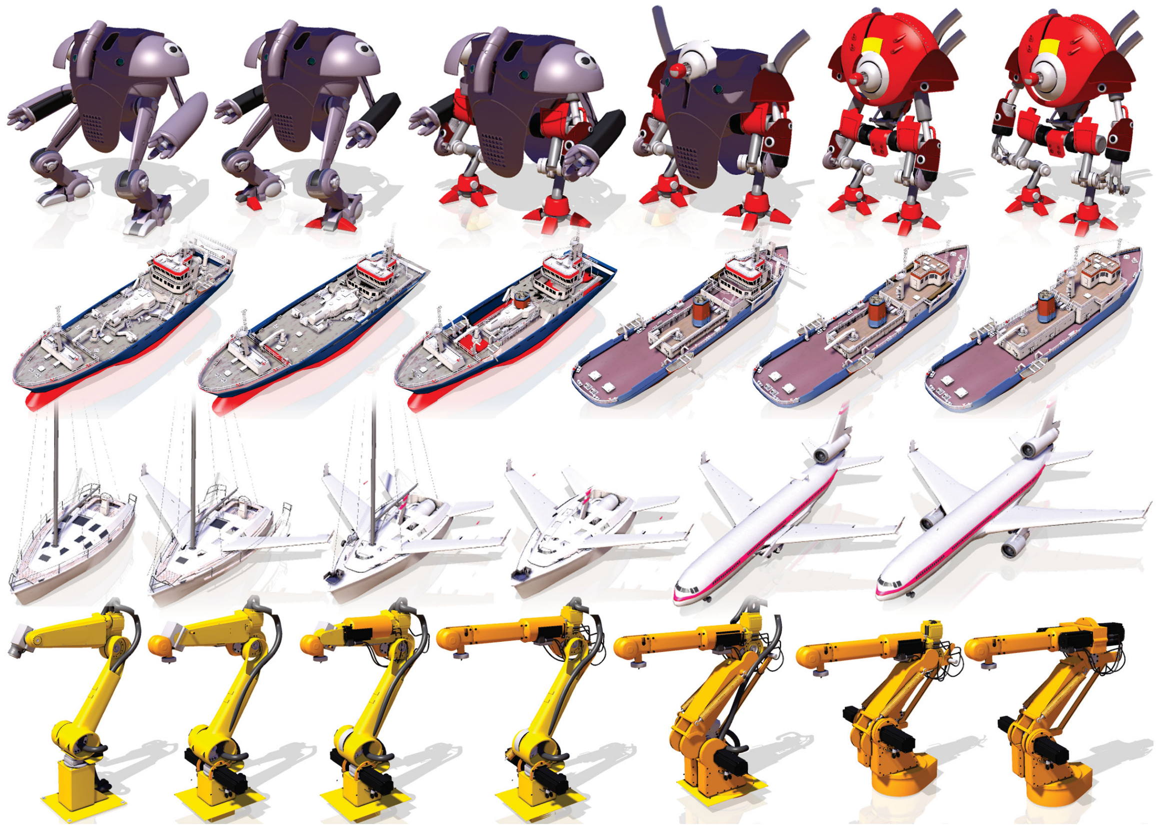

Figure 12: *Results of the blend operation (far left source shape, far right target shape)*

the user to control which part of the current shape is replaced. We call this functionality the *object brush* as it is possible to paint the parts of a selected new shape onto the current shape. In Fig. 15 the result of such a brush operation is shown. The algorithms employed during the brush operation are exactly the same as those for blending; the only difference is that the user has full control over which node j is flipped from the source to the target, i. e., when the user clicks on a source node the algorithm simply instantiates the corresponding target node. We evaluated our system with an informal user study with 14 computer science students who had never used the system before. On average the participants needed 01:41 min:sec to create a new shape with the blend tool, and it took 01:37 min:sec to reproduce a given result with the object brush tool. When asked if our system was helpful to solve the given task, the average rating over three tasks was 6.05 on a 7-point Likert scale (where 1 is worst and 7 is best). Details of the user study are provided as supplemental material.

5. Discussion

Not all the shapes generated by our system are useful or visually pleasant. Nevertheless, most of the results are of high quality, especially when it is taken into account that there was no manual intervention in the complete processing pipeline.

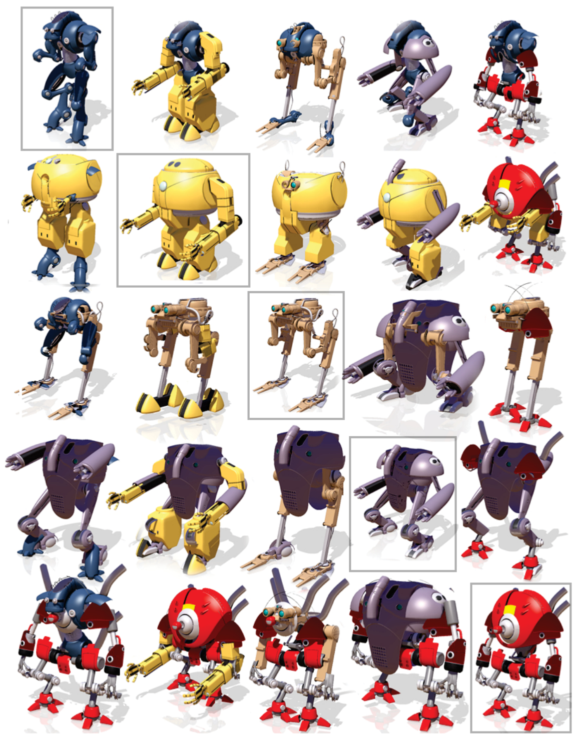

Figure 13: *Matrix of blend results between 5 different robot shapes in our database. The diagonal contains the original shapes, the off-diagonal the blends for a weight of $w \approx 0.5$. As described in Section 3.2.1 the blend is not symmetric.*

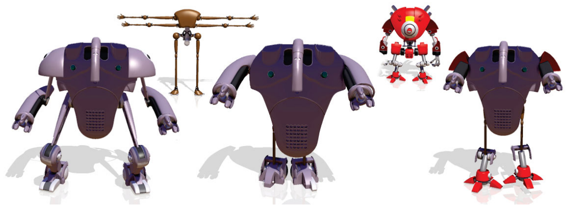

Figure 14: *Blending three shapes: (from left to right) source shape, first target shape, intermediate result, second target shape, final result that contains parts from all three inputs.*

As can be verified in Fig. 13, it is possible to generate a large number of blended shapes that have similar quality as those that come directly from the database.

Currently, we only provide a very high-level user-interface with very easy-to-use tools; however, our system currently has no tools to repair a shape, e.g., when an undesired placing of a part occurs. One option would be to provide our system as a plug-in for a conventional modeling package. A falsely placed part can then be corrected easily with the editing tools provided by the conventional modeling package.

Most of the time, unpleasant or strange-looking recombinations occur if the structure of the source and the target shape are very different or objects of different classes are combined,

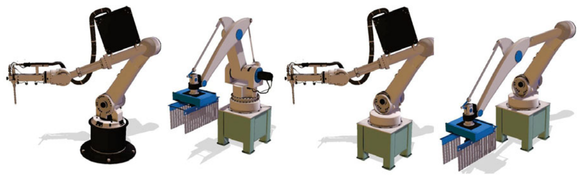

Figure 15: *Exchanging parts of a shape using the object brush: (from left to right) the source shape, the target shape, the user clicked on the black base of the source shape, the user then clicked on the extended arm.*

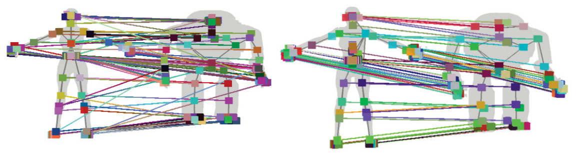

Figure 16: *Graph matching between parts of two humanoid figures.* Left*: spectral graph matching.* Right*: our approach. Note that our approach prevents, e. g., parts of the foot from being mapped to the hand. The spectral graph matching finds a good global solution; however, as the resolution hierarchy is not updated, it cannot perform as well as an approach that combines matching and re-grouping of parts.*

i. e., a morph between two boats does typically look more convincing than a morph between a boat and a plane (cp. Fig. 12). Nevertheless, our system produces reasonable results even in those situations. This works only because our algorithm updates the resolution hierarchy during matching. Fig. 16 compares our approach to spectral graph matching [Leo09], which does not re-group parts to form a new resolution hierarchy. Spectral graph matching additionally allows to consider pairwise scores between two matches. For example, a high score can be given to two matches if there is a contact between the two involved parts in the source shape and there is also a contact between the two involved parts in the target. Furthermore, it is possible to assign higher pairwise scores to two matches if two source parts are symmetric and the corresponding targets parts are symmetric as well. However, for our particular application, the algorithm presented above usually gives better results (a comparison is shown in Fig. 16).

In comparison to Shuffler [KJS07], where users specify indivdual parts to be replaced, our system performs complete blends where all parts have to be subsequently exchanged. Furthermore, our approach differs in the employed shape segmentation and shape matching strategy. In the Shuffler approach segments are matched that have a similar midpoint graph distance and similar geometry. The goal of our method is to allow matching of parts that have rather different geometries (cp. Fig. 13); consequently, our approach relies on positional (and contact) information instead.

The quality of the blended results depends on the way segments are grouped into shapes. If they are coupled loosely, with few contacts (e. g., limbs of robots, or features attached to the body of a vehicle), the approach works best. For too complex contact constraints, such as segments inside a car, the recombination might fail. Our approach does not allow the deformation of individual parts. Consequently, shapes that have puzzle-like structures (i.e., where each part has to fit into another) typically also produce visible artefacts. Furthermore, as we do not perform any functional analysis, some intermediate shapes do not look plausible.

6. Conclusion

In this work, we addressed the problem of interactively composing complex shapes from individual parts such as those found in the world that surrounds us every day. We exploit the constraints of symmetries and physical contacts, as analysed from the example shapes, to restrict the produced shapes.

As a result, visually pleasing blends can be rapidly generated between a large number of database shapes of different structures. Combining our approach with a more explicit functional analysis would be a challenging but interesting extension of our system that may improve the plausibility of the results. In future work, we would like to apply our approach to organic shapes, images, animations, or audio.

References

[ACOL00] Alexa M., Cohen-Or D., Levin D.: As-rigid-as-possible shape interpolation. In *Proc. SIGGRAPH* (2000), pp. 157–64. 2

[ACP03] Allen B., Curless B., Popović Z.: The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Trans. Graph. (Proc. SIGGRAPH) 22*, 3 (2003), 587–594. 2

[BBW*08] Berner A., Bokeloh M., Wand M., Schilling A., Seidel H.-P.: A graph-based approach to symmetry detection. In *Proc. Volume and Point-based Graphics* (2008), pp. 1–8. 3

[BN92] Beier T., Neely S.: Feature-based image metamorphosis. *Computer Graphics (Proc. SIGGRAPH) 26*, 2 (1992), 35–42. 2

[BV99] Blanz V., Vetter T.: A morphable model for the synthesis of 3D faces. In *Proc. SIGGRAPH* (1999), pp. 187–94. 2

[BWS10] Bokeloh M., Wand M., Seidel H.: A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 1–10. 2

[CKGK11] Chaudhuri S., Kalogerakis E., Guibas L., Koltun V.: Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4 (2011). 2

[COSL98] Cohen-Or D., Solomovic A., Levin D.: Three-dimensional distance field metamorphosis. *ACM Trans. Graph. 17*, 2 (1998), 116–141. 2

[Ebe03] Ebert D.: *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann Pub, 2003. 2

[EL02] Efros A., Leung T.: Texture synthesis by non-parametric sampling. In *Proc. ICCV* (2002), vol. 2, IEEE, pp. 1033–1038. 2

[FCODS08] Fu H., Cohen-Or D., Dror G., Sheffer A.: Upright orientation of man-made objects. *ACM Trans. Graph. (Proc. SIGGRAPH) 27*, 3 (2008). 2

[FKS*04] Funkhouser T., Kazhdan M., Shilane P., Min P., Kiefer W., Tal A., Rusinkiewicz S., Dobkin D.: Modeling by example. In *Proc. SIGGRAPH* (2004), pp. 652–663. 2

[GF09] Golovinskiy A., Funkhouser T.: Consistent segmentation of 3D models. *Proc. SMI 33*, 3 (2009), 262–69. 2

[GSMCO09] Gal R., Sorkine O., Mitra N., Cohen-Or D.: iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. (Proc. SIGGRAPH) 28*, 3 (2009), 1–10. 2

[KHS10] Kalogerakis E., Hertzmann A., Singh K.: Learning 3D mesh segmentation and labeling. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 1–12. 2

[KJS07] Kraevoy V., Julius D., Sheffer A.: Model composition from interchangeable components. In *Proc. Pacific Graphics* (2007), pp. 129–138. 2, 7, 9

[LACS08] Li W., Agrawala M., Curless B., Salesin D.: Automated generation of interactive 3D exploded view diagrams. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3 (2008), 1–7. 2

[LDSS99] Lee A. W. F., Dobkin D., Sweldens W., Schröder P.: Multiresolution mesh morphing. In *Proc. SIGGRAPH* (1999), vol. 99, pp. 343–350. 2

[Leo09] Leordeanu M.: *Spectral Graph Matching, Learning, and Inference for Computer Vision*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2009. 9

[Mer07] Merrell P.: Example-based model synthesis. In *Proc. I3D* (2007), pp. 105–112. 2

[MGP06] Mitra N., Guibas L., Pauly M.: Partial and approximate symmetry detection for 3D geometry. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3 (2006), 560–568. 2

[MHHR07] Müller M., Heidelberger B., Hennix M., Ratcliff J.: Position based dynamics. *J. Vis. Comm. Image Rep. 18*, 2 (2007), 109–118. 7

[MYY*10] Mitra N., Yang Y., Yan D., Li W., Agrawala M.: Illustrating how mechanical assemblies work. *ACM Trans. Graph. (Proc. SIGGRAPH) 29*, 4 (2010), 1–12. 2

[MZL*09] Mehra R., Zhou Q., Long J., Sheffer A., Gooch A., Mitra N.: Abstraction of man-made shapes. *ACM Trans. Graph. (Proc. SIGGRAPH) 28*, 5 (2009), 137. 2

[OLGM11] Ovsjanikov M., Li W., Guibas L., Mitra N. J.: Exploration of continuous variability in collections of 3D shapes. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4 (2011). 2

[PMW*08] Pauly M., Mitra N. J., Wallner J., Pottmann H., Guibas L.: Discovering structural regularity in 3D geometry. *Proc. SIGGRAPH 27*, 3 (2008). 2

[SBSCO06] Sharf A., Blumenkrants M., Shamir A., Cohen-Or D.: SnapPaste: An interactive technique for easy mesh composition. *The Visual Computer 22*, 9 (2006), 835–44. 2

[vdB98] van den Bergen G.: Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools 2* (1998), 1–13. 3

[Wol98] Wolberg G.: Image morphing: A survey. *The Visual Computer 14*, 8 (1998), 360–372. 2

[WXL*11] Wang Y., Xu K., Li J., Zhang H., Shamir A., Liu L., Cheng Z., Xiong Y.: Symmetry hierarchy of man-made objects. In *Proc. Eurographics* (2011). 2

[XWY*09] Xu W., Wang J., Yin K., Zhou K., Van De Panne M., Chen F., Guo B.: Joint-aware manipulation of deformable models. In *ACM Trans. Graph (Proc. SIGGRAPH)* (2009), pp. 1–9. 2

[ZFCO*11] Zheng Y., Fu H., Cohen-Or D., Au O. K.-C., Tai C.-L.: Component-wise controllers for structure-preserving shape manipulation. In *Proc. Eurographics* (2011). 2