Functionality Preserving Shape Style Transfer

Zhaoliang Lun[1] Evangelos Kalogerakis[1] Rui Wang[1] Alla Sheffer[2]

[1]University of Massachusetts Amherst [2]University of British Columbia

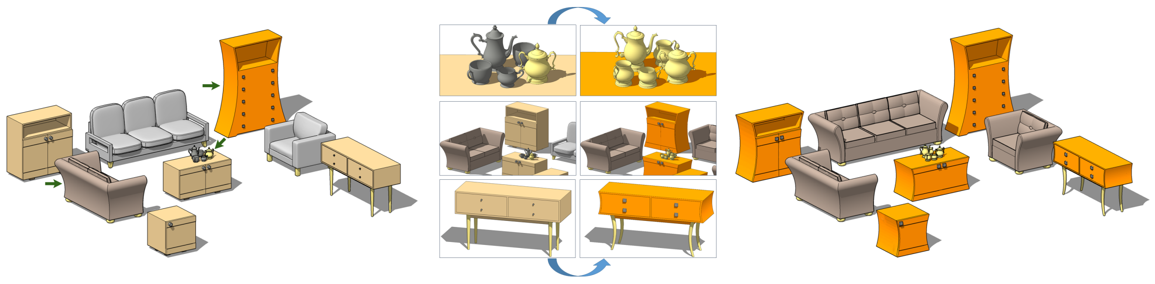

Figure 1: *Our algorithm transfers the geometric style of three exemplars, cabinet, loveseat, and sugar pot (left, highlighted with arrows), to the rest of the objects in the scene, while preserving the target functionality (right). Please zoom-in to see more details.*

Abstract

When geometric models with a desired combination of style and functionality are not available, they currently need to be created manually. We facilitate algorithmic synthesis of 3D models of man-made shapes which combines user-specified style, described via an *exemplar* shape, and functionality, encoded by a functionally different *target* shape. Our method automatically transfers the style of the exemplar to the target, creating the desired combination. The main challenge in performing cross-functional style transfer is to implicitly separate an object's style from its function: while stylistically the output shapes should be as close as possible to the exemplar, their original functionality and structure, as encoded by the target, should be strictly preserved. Recent literature point to the presence of similarly shaped, salient *geometric elements* as a main indicator of stylistic similarity between 3D shapes. We therefore transfer the exemplar style to the target via a sequence of element-level operations.We allow only *compatible* operations, ones that do not affect the target functionality. To this end, we introduce a cross-structural element compatibility metric that estimates the impact of each operation on the edited shape. Our metric is based on the global context and coarse geometry of evaluated elements, and is trained on databases of 3D objects. We use this metric to cast style transfer as a tabu search, which incrementally updates the target shape using compatible operations, progressively increasing its style similarity to the exemplar while strictly maintaining its functionality at each step. We evaluate our framework across a range of man-made objects including furniture, light fixtures, and tableware, and perform a number of user studies confirming that it produces convincing outputs combining the desired style and function.

Keywords: style analysis, shape synthesis, style-vs-function

Concepts: •**Computing methodologies** → **Shape modeling;**

SA '16 Technical Papers, December 05-08, 2016, Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2980237

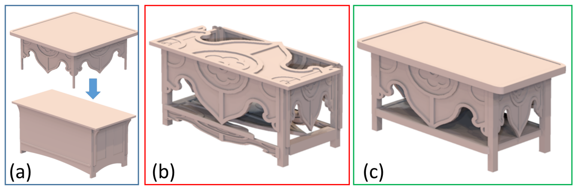

Figure 2: *Transferring a style from a table to a TV stand (a) without (b) and with (c) functionality constraints in place.*

1 Introduction

Human living spaces are typically populated with groups of functionally different household objects which share a common style that is reflective of both their temporal and geographic context and the personal preferences of their owners. To model believable virtual 3D scenes, artists aim to similarly populate them with stylistically coherent sets of objects and to select a style reflective of the desired scene context. Unfortunately, while large databases of 3D man-made objects already exist, sets of functionally diverse shapes in a particular style are often not available. Synthesizing 3D objects with desired functionality and style from scratch is time-consuming and expertise-intensive. Instead, we propose a novel shape synthesis algorithm that transfers the style of an *exemplar* shape to structurally and functionally different *target* objects while retaining target functionality (Figure 1). Using our framework, instead of locating or generating an entire set of functionally different objects in a desired style, users need to locate or create a single *exemplar*, a much easier task. By enabling algorithmic, cross-functional, exemplar-to-target style transfer we significantly simplify and speed up the modeling of coordinated virtual environments.

The main challenge in style transfer between objects with different functionality is to implicitly separate style from function: while each output should stylistically look similar to the exemplar, it should fully retain the functionality of the target (Figure 2). Previous attempts at achieving this goal addressed only very narrow special cases of 3D style transfer: transfer of part scales across co-segmented shapes [Xu et al. 2010] and analogy-based transfer which employs an extra source model as input [Ma et al. 2014]. Our framework is much more general than these approaches and requires significantly less user input (Section 2). Most of the results shown in this paper, including those in Figure 1, are not attainable by previous methods.

Conceptually, we can formulate style transfer as a constrained opti-

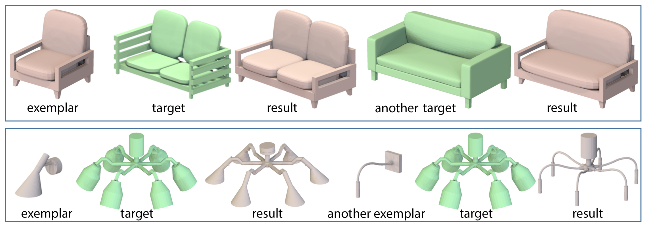

Figure 3: *(top) Using the same exemplar with two different same-class targets may lead to different outputs due to variations in target structure. (bottom) Using different exemplars and the same target predictably generates outputs with distinct styles.*

mization problem: given an exemplar and a target shape, construct an output that preserves the target shape's structure and functionality, while minimizing the style distance between the output and the exemplar. Recent research [Lun et al. 2015; Liu et al. 2015b] as well as descriptions of style common in art history literature [Nutting 1928; Lewis 2008] argue for encoding style via fine-level geometric properties of shape elements, or separable object parts and sub-parts. Shapes are deemed to be more stylistically similar if they share more geometrically similar elements, and have fewer unshared decorative features. Following these observations, we define our style transfer method around element level operations - substitution, addition, removal, and deformation.

We first hierarchically segment the models (Section 4) and then apply a sequence of operations that minimizes the style distance [Lun et al. 2015] between the target and the exemplar, using a tabu search framework (Section 3). We only permit *compatible* operations, or ones that maintain the original functionality of the output model. To this end we introduce new element and shape compatibility metrics (Section 5) inspired by design literature [Norman 1988] and recent research [Liu et al. 2015b] that suggest that gross form and arrangement of elements within a shape are strongly correlated to its functionality.

We consequently predict that preserving gross element form and arrangement indirectly preserves shape functionality. Furthermore we expect that the more similar these structural properties are, the more likely are the elements and the shapes containing them to be functionally compatible. Thus, our compatibility measure is based around element overall shape and their context within their respective shapes. We encode these properties by considering the features of the individual elements, their neighbors, and their symmetric counterparts, and accumulating those into a graph kernel based formulation [Laga et al. 2013]. Contrary to Laga et al. who seek to compute semantic part correspondences within the same class, the features we encode within the graph kernel are selected to evaluate functional compatibility across different classes. We employ a learning based approach to derive robust metric parameters using automatically generated training datasets. Specifically we note that sets of coordinated same style shapes in online shape databases can provide insights on element compatibility. In particular the contexts of similarly shaped, and hence likely compatible, elements located on differently structured set shapes provide valuable cues as to the intercalation between compatibility and context. Our resulting compatibility metric is validated to be well aligned with human perception (Section 9).

The compatible operations supported by our framework are specifically designed to maintain the target structure by preserving element connectivity during edits (Figure 9). In particular, our substitution and addition operations (Section 6) assemble the new outputs by applying a restricted set of linear transformations to the newly added-in element and the elements of the current model. The imposed restrictions are designed to simultaneously maintain the output structure and preserve exemplar element style.

The output geometry we produce is impacted by the choice of the target shape (Figure 3, top), with different target shapes, even within the same narrow class, sometimes being more or less amenable to style transfer from a particular exemplar. To facilitate the selection of the most adaptable target for each exemplar we allow users to specify a target shape class rather than a specific target shape. We then employ a variant of our compatibility measure to select the most adaptable target object within a database of shapes in the specified class (Section 8).

Our algorithm's settings are learned from perceptual and geometric data and aim to achieve results that suit a typical user. Since style-function separation is necessarily subjective, to account for personal preferences we support computation of multiple alternatives solutions, with a range of style similarity scores and compatibility thresholds. We then provide the user with both the best result, generated with parameters empirically determined to be optimal, as well as a ranked list of alternatives.

Contribution. Our main contribution is a new algorithm for geometric style transfer between models of man-made household objects with different structure and functionality. Our method is significantly more general and requires less input than previous techniques. It successfully maintains fine-grained output functionality without specifically identifying functional parts or elements (Figure 1). The three technical components that form this method are the general framework, based around element-level operations, a new element compatibility measure, and a slot alignment algorithm that balances functionality and style. Our major technical contribution, which has many applications beyond style transfer, is the new compatibility metric that measures functional similarities between objects across a range of granularities, from fine elements, to coarse geometric parts and all the way to the entire models.

We validate our work in a number of ways. We provide a gallery of results both in the main paper and the supplementary material, including both the top scoring outputs and the suggested alternatives. To evaluate the effectiveness of our method we conduct three user studies validating our method's ability to plausibly transfer the exemplar style to the target; confirming its ability to retain the target functionality; and comparing our functional compatibility metric against prior work.

2 Related Work

We build upon previous work in shape style and function analysis, part correspondence, part-based shape synthesis, and style transfer.

Style analysis. A large number of works explore fine-grained and style similarities between shapes in the same functional class [Xu et al. 2010; Kalogerakis et al. 2012; Huang et al. 2013; Yumer and Kara 2014; Kleiman et al. 2015]. They offer few insights on style similarity evaluation between models in different functional classes, or categories. More recent works aim to fill this gap: Liu et al. [2015b] propose a metric for stylistic compatibility between furniture learned from collections of 3D scenes, while Lun et al. [2015] introduce a more generic structure-invariant style similarity measure applicable to multiple categories of man-made shapes. Both measures are driven by analysis of variation in fine details across model parts and sub-parts. Specifically, Liu et al. observe that the functionality of objects is strongly correlated to the gross shape and arrangement of their major parts, while style is strongly linked to the fine geometric details of these parts. Similarly, the work of Lun et al. is driven by observations in art history literature [Nutting 1928; Lewis 2008] that point to the presence of similarly shaped, salient, geometric elements across analyzed shapes as a key indicator of stylistic similarity.

While neither paper makes any suggestions as to how to reduce style distance between shapes, the insights they provide inspire our

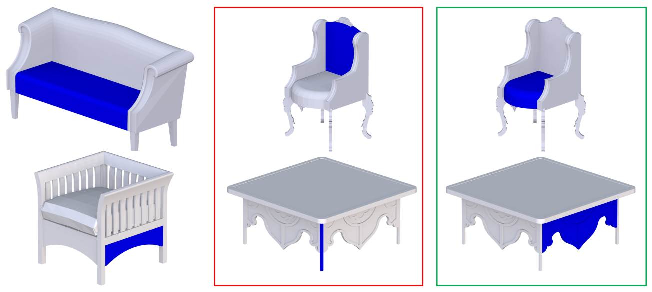
Figure 4: *Part compatibility: [Laga et al. 2013] wrongly identify the highlighted parts (center) as compatible to those on the left. Compatible parts correctly identified by our method (right) can share same functionality (top) but can also have different function (bottom) yet be safe to swap with those on the left without reducing object functionality.*

work. We note that regardless of which of these two style distance measures we use, the distance between the exemplar and target can be decreased by increasing the proportion of similarly shaped elements on them, e.g. by replacing elements on the target with ones from the exemplar, or by deforming target elements to be more geometrically similar to exemplar ones. However, unconstrained target element editing can easily break target functionality. The key contribution of our work is a method that preserves target functionality while performing element-level style transferring edits that minimize exemplar-target style distance.

Part Correspondence. Numerous existing methods compute correspondences between compatible parts of objects within the same class or co-segment such objects into such corresponding parts, see [van Kaick et al. 2011; Xu et al. 2016] for recent surveys. Recent methods leverage structural similarities between shapes to extract functional part correspondences. Zheng et al. [2013] match specific types of shape substructures (called SFARR) that have the form of part triplets: two symmetric parts and a third part connecting the two. Such special substructures do not exist in many man-made shape categories (e.g., lamps, cutlery) and span only a small subset of compatible parts in others. Liu et al. [2015a] extract common substructures on shapes through manual annotation of corresponding parts. Our method instead uses more general structural relationships to measure functional compatibility between elements and does not require any user interaction.

Laga et al. [2013] use graph kernels to evaluate functional part correspondences between shapes within the same class. Our work adopts some ideas from this work, and in particular the use of graph kernels to measure element compatibility. As shown by our comparisons to [Laga et al. 2013] (Figure 4, Section 9), applying graph kernels successfully to evaluate element compatibility on structurally different shapes requires automatic kernel parameter adaptation, different geometric feature sets and different encoding of graph edge relationships. In particular, instead of hand-tuning graph kernels, we employ a learning approach detailed in Section 5 to automatically adapt weights of appropriate feature descriptors and graph kernel parameters impacting cross-functional compatibility.

A number of recent works compute functional correspondences between points, parts, or fuzzy regions on different shapes by analyzing their potential interactions with either other objects in the scene [Hu et al. 2015; Hu et al. 2016], or with posed human avatars [Jiang et al. 2013; Kim et al. 2014; Savva et al. 2014; Zhu et al. 2014; Savva et al. 2016]. The detected interactions relate parts with similar gross functionality across objects within the same broad shape class and with sufficient input can be extended to handle patch correspondences with similar interaction types across

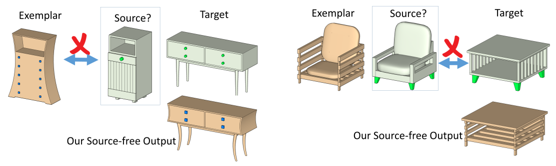

Figure 5: *Example triplets that violate the assumptions of [Ma et al. 2014]: (left) source and exemplar with misaligned decorative elements (right) only a small portion of the target's surface (legs) has meaningful source counterparts. (bottom) We plausibly transfer style in both cases, without using source models.*

shapes with different functionality [Hu et al. 2016]. However, such correspondences can only be estimated for patches with specific types of object interactions and require large amounts of external context data. Consequently it is likely unrealistic to extend these methods beyond detecting gross functional part correspondences. Our work uses readily available coordinated object scenes as the only training input, and computes fine-level element compatibility, necessary for synthesis of detailed functional geometric shapes in a given style. We demonstrate that compatibility can be successfully evaluated without explicit functionality detection.

Part-based shape synthesis. Interactive methods for part-based shape synthesis rely on users to specify and edit parts to assemble a shape, either directly [Funkhouser et al. 2004; Kreavoy et al. 2007] or indirectly through semantic handles, attributes and suggestion mechanisms [Chaudhuri et al. 2011; Yumer et al. 2015], and to explicitly control output style and function.

Shape grammars are used to generate new models, either by repeating structural patterns specified manually by the user, or through automatic inference of such patterns from a set of examples [Bokeloh et al. 2010; Talton et al. 2012]. The shape structure and functionality are determined by the grammar, which operates within a particular shape class, or sub-class.

Multiple methods generate new shapes by combining parts from objects within the same functional class [Kalogerakis et al. 2012; Huang et al. 2015a; Xu et al. 2012; Huang et al. 2015b], employing part correspondences generated via co-segmentation and labeling methods designed to operate on shapes with common functionality and coarse structure. After assembling models using co-segmented same class-shapes as input, Huang et al. [2015b] subsequently deform them to best fit input images. [Yumer and Kara 2014] facilitates structure preserving shape deformation by providing users with deformation handles trained on co-segmented shapes within a class. All these methods rely on co-segmentation or correspondences between parts of objects within a single class. Our framework is designed for transferring element style between shapes in vastly different classes, requiring a cross-functional element compatibility measure and does not require any prior co-segmentation or part labeling.

Style transfer. Researchers have explored style transfer for 2D curves, e.g. [Hertzmann et al. 2002; Li et al. 2013] and images [Hertzmann et al. 2001]. While insights from these frameworks are useful for understanding the conceptual notion of style transfer, as explained by [Xu et al. 2010; Ma et al. 2014], algorithms developed for the 2D setting cannot be readily extended to 3D space.

There had been no attempts to address generic style transfer for 3D shapes. However two recent methods address special cases of this problem [Xu et al. 2010; Ma et al. 2014]. Given a predefined coarse segmentation of the exemplar and target shapes into corresponding parts, Xu et al. [2010] use these correspondences to anisotropically scale target parts to fit the proportions of the match-

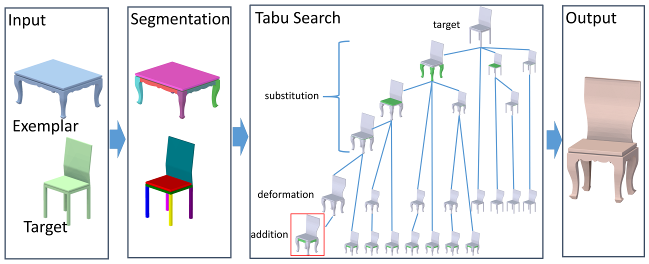

Figure 6: *Framework overview.*

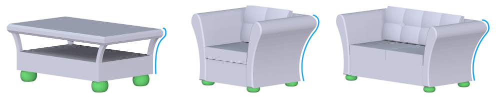

Figure 7: *Same style objects frequently have both similarly shaped compatible elements (here legs) as well as similar curves positioned on incompatible parts (here curved corner features and contours) and similar element proportions.*

ing parts on the exemplar. The method has limited applicability, as it assumes a meaningful part level correspondence between the exemplar and target, and cannot handle style properties beyond scale.

Following [Hertzmann et al. 2001; Hertzmann et al. 2002], Ma et al. [2014] require a triplet of inputs: an exemplar, a source, and a target, where the source and target are expected to share the same style but have different functionality, while the source and exemplar are expected to have the same functionality and structure. They assemble the output by combining exemplar and target surface patches guided by a combination of a dense exemplar to source mapping and a piece-wise similarity transformation between the source and target. Their method makes a number of strong assumptions, that rarely hold even when a source model which fits the generic criteria above is available (Figure 5). For the transfer to be successful, they implicitly assume that decorative elements on the exemplar and source are co-located, and assume most target surfaces to have meaningful source counterparts, related via simple similarity transformations (target surfaces with no source counterparts are left untouched by the transfer). Our framework has none of these limitations: it does not require a source or a compatible segmentation, and can handle a far wider range of inputs than either of the methods above.

3 Style Transfer Framework

Our framework takes an exemplar shape and a target shape in a different functional class as input. Users can either provide a specific target shape, or alternatively point the algorithm to a database of shapes within a target class. In the latter case our method automatically analyzes the database and extracts the target shape that is anticipated to be easiest to transfer exemplar style to while still maintaining its functionality (Section 8).

Given the exemplar and target shapes, we search for modifications to the target shape which bring it stylistically closer to the exemplar. In evaluating style we follow previous work which focuses on fine-level geometric features of model elements, or parts and sub-parts, as major style cues. In particular, [Lun et al. 2015] identify three factors affecting element-level style similarity: similar geometric surface features across elements, similarity between dominant feature and contour curves, and element proportions (Figure 7). Using this measure the style distance between two model decreases as the number of similar elements on them increases, or the number of dissimilar elements decreases.

Our method is consequently designed around element level operations. It begins by hierarchically segmenting both the exemplar and the target into potential geometric elements (Section 4) and then employs a set of element-level target modifications that reduce the style distance between the output and the exemplar (Figure 6). To measure this distance we use the style measure of [Lun et al. 2015] which is demonstrated to work well on a large range on household object categories. We perform only compatible operations, ones that do not violate the target functionality. We evaluate compatibility as discussed in Section 5.

Operations. The simplest and most common operation we employ is substituting elements on the current shape with (appropriately scaled) exemplar elements. By definition any such substitution reduces the style distance between shapes; however, it is rarely possible to replace every single portion of the target shape with exemplar elements without violating functionality. We therefore use three additional operations that can improve style similarity once substitution is no longer possible: curve-based element deformation, element addition, and element removal. Our deformation step takes as input a pair of dominant, decorative curves, or *curve handles*, on pairs of exemplar and current models, and deform the current model by substituting the current curve with its (appropriately scaled) exemplar counterpart adjusting the rest of the model's geometry to match the new curve (Section 7). Our addition and removal operations are intended to process purely decorative elements (i.e., ones whose presence does not affect overall shape compatibility) occasionally present on either the exemplar or the target models which do not have a counterpart on the other model. Addition of decorative exemplar elements to the target shape, and removal of decorative target elements facilitate further improvement of style similarity between the two models.

When a target shape contains symmetric elements (such as four legs in a chair), processing these elements separately can break this symmetry, potentially degrading output functionality and negatively affecting its look. To ensure that our operations preserve target shape symmetries, we detect all replicated elements and curve handles on the target shape, and apply each operation to an entire symmetry group instead of to a single element.

Tabu Search. Our optimization procedure is designed to select from a large set of possible target shape modifications the modifications which will provide maximal style adaptation while preserving target functionality. Our algorithm follows the concept of tabu search [Glover and Laguna 1997]. A detailed pseudocode of our method is provided in the Appendix.

Throughout the optimization, we maintain a list of seed shapes which is initialized with the given target shape. At each iteration, we remove a seed shape from the list and attempt to bring it closer, style-wise, to the exemplar using one of the four supported operations. To explore the most promising solutions first, we always select the seed shape currently closest to the exemplar in terms of style. If the attempt to improve it through allowable operations succeeds, then the improved shape is inserted into the seed list. If the attempt yields a shape which has been generated before, or if all evaluated operations violate the functionality preservation constraints, then we discard and forbid the output. In this manner, the tabu search only searches the space of valid target shape modifications (a much smaller subset of the set of all possible modifications), and avoids performing redundant operations on previously generated shapes or performing operations on functionally implausible shapes. If no more operations can be performed on any shapes in the seed list, the optimization loop terminates.

To reduce search space and avoid redundant operations we first perform tabu search using only element substitutions, then once compatible style-distance-reducing substitutions are exhausted we repeat the search using curve-based deformation, and finally use the same process for adding and then for removing decorative elements.

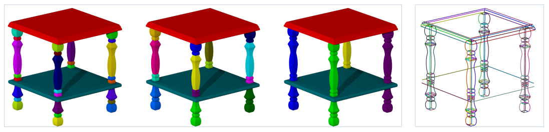

Figure 8: *Hierarchical segmentation and extracted curves.*

Improvement Step. Given a seed shape, we aim to perform a compatible editing operation that will maximally reduce the style distance from the seed to the exemplar. We thus cycle through all possible operations of the currently examined type and select the one that reduces style distance the most. We can only compute the exact impact of each operation after it is performed, since most operations change the geometry of both the elements involved and their surroundings. However, for both substitution and deformation we can reasonably predict beforehand if the operation is incompatible, or if it does little to reduce style distance. We rely on such predictions to avoid unnecessary computations. To examine if an operation is *a priori* incompatible, we use the compatibility measure and the threshold learned from training data discussed in Section 5, and only proceed with full computation if the predicted compatibility is higher than the threshold. We also avoid operations predicted to reduce the style distance by less than a minimum threshold. Style distances are normalized to the $[0, 1]$ interval and we use 1% as a threshold. Once the operation is performed we reassess both compatibility and style distance, rejecting operations that after the fact violate compatibility or do not sufficiently reduce style distance. We add the compatible result with the smallest distance to the exemplar to the list of seeds.

Substitution. Give a pair of compatible elements in the exemplar and seed shape, we seek to replace the seed shape element, including any of its symmetric counterparts, with the corresponding element on the exemplar shape. This step requires alignment or coverage of *slots* [Kalogerakis et al. 2012], i.e. areas on the elements which are in contact with the rest of the shape (Figure 9). The alignment step, Section 6, balances two potentially contradictory goals: it seeks to preserve output functionality and to minimally change the style, and specifically the proportions, of both the substituted-in and pre-existing output elements (Figure 9).

Curve-Based Deformation. Our curve based deformation step only considers seed elements originating from the target, and evaluates the compatibility of each possible deformation using a variant of the compatibility measure designed for curves (Section 5). The deformation embeds the exemplar curve into the candidate shape replacing its target counterpart and adapting the surrounding surface, see Section 7.

Element Additions. Addition is performed after substitution and deformation, the two operations expected to be most effective at minimizing exemplar-output style distance. To add a new element to a model one needs to know the attachment point or slot to place it at. We thus only consider adding elements that on the exemplar are immediately adjacent to an element previously substituted-in or added to the seed. To add the element we first place it next to this prior neighbor using the slots they shared on the exemplar. If after this step the element has any uncovered slots left, we perform slot alignment to cover them (Section 6).

Element removals. Finally, we perform removal of target elements, deemed decorative, i.e. having only marginal impact on functionality as measured by our per-object compatibility function. We only remove elements if the slots they share with the rest of the model are closed surfaces, i.e. if no gaping holes are left after removal. Typically only a tiny fraction of elements fits these constraints.

Termination. Once a seed shape can no longer be improved it is considered to be a terminal solution, or in other words it corresponds to a local minimum. In this case, we store it in an output list and proceed to examine the next best seed shape in our search list, terminating when this list is empty. The tabu search typically computes and evaluates a few dozen solutions.

Output. If the user seeks a single output, then at the end of the process we return the shape in the output list closest to the exemplar in terms of style distance. To provide multiple results with different emphasis on target functionality preservation versus exemplar style adaptation, we add to the output list all the seed models produced during the tabu search, as well as models generated with more lax compatibility thresholds, see supplementary material.

4 Pre-Processing and Segmentation

Our style transfer framework requires that models are first hierarchically segmented into meaningful geometric elements. We assume that the input shapes are represented as partially connected meshes (polygon soups), are consistently scaled and oriented (have the same upright orientation, and face the same front direction when that direction is well defined). Consistent orientations can be found manually or by using the automatic method by Huang et al. [2013]. Consistent scalings are computed through bounding box-based alignment.

Segmentation. We follow previous works that use convexity as a criterion for shape segmentation into meaningful parts or elements. At the finest hierarchy scale, we generate geometric elements by using the approximate convex decomposition technique by [Asafi et al. 2013]. We generate elements at a number of scales by repeating the segmentation with different convexity thresholds (0.3, 0.5 and 0.7). Since when designing 3D models of man-made objects artists often leave functionally meaningful parts as separate components we also add such separate connected components as potential elements. We introduce additional larger elements by merging neighboring elements when they jointly approximate a portion of a primitive (box, sphere, or cylinder). The primitive-based element grouping is based on [Yumer and Kara 2012]. The result of this step is a collection of a few dozen elements at a range of scales (Figure 8, left). We also detect symmetry groups of elememts by approximately matching them through ICP.

For each shape we build a graph representing its structure, which is consequently used for compatibility evaluation (Section 5). The nodes of the graph are the different elements (typically 10-50 elements per shape). The graph has three types of edges: we connect nodes by an *adjacency* edge if their corresponding elements are adjacent; we create a *symmetry* edge connecting nodes whose corresponding pairs of elements are related under a reflective, translational or rotational symmetry; and we create *containment* edges between nodes corresponding to pairs of elements where one element directly contains the other in the hierarchical segmentation.

Curve Handles. Our element deformation operation uses matching curve handles on target and exemplar elements. Following the style formulation of Lun et al [2015], which points to both feature and contour curves as reflective of object style, we extract two types of curve handles (Figure 8, right): view-independent ridges and valleys [Ohtake et al. 2004], and occluding contours. To compute the latter we use 12 views uniformly distributed about the upright axis at elevation angles of 0, 30, and 60 degrees above the horizontal plane. We extract the feature curves as described in [Kalogerakis et al. 2009], and hierarchically segment them along element boundaries.

5 Compatibility

To effectively transfer style we need to evaluate the impact of each editing operation on the functionality of the edited shape. We answer this question by using a set of compatibility measures that predict the impact of our most commonly used operations - element substitution and deformation before those are fully executed and also assess the a posteriori impact of any operation on the output functionality once completed.

We formulate substitution and deformation compatibility by considering the contextual similarity between the substituted elements or deformation handles, and formulate shape-level similarity by analyzing compatibility between pairs of elements on them.

5.1 Formulation

Previous research, e.g. [Liu et al. 2015b] as well as insights from design literature [Norman 1988] point to the context and gross shape of geometric elements as important features in determining an object's functionality. While we do not aim to detect functionality, we speculate that elements with similar context and shape features are more likely to be compatible, i.e. replacing one with the other is less likely to negatively affect the functionality of the resulting shape. For example consider the element pairs in Figure 4 - the seats on the sofa and armchair share similar relative locations with respect to the overall object center, have comparable orientations, and similar local neighborhood structures; the same applies to the "skirts" on the table and armchair at the bottom. Replacing one with the other, using appropriate scaling will preserve the functionality of the containing shapes. In contrast the skirt and the table leg, or the sofa seat and the chair back may be geometrically similar but have very different context. Our metric is designed to reflect these similarities and differences.

We encode each element's *functional* and *contextual* properties using the element relation graph contracted as described in Section 4. We then use a graph kernel-based similarity evaluation framework, inspired by [Laga et al. 2013] to combine those into a single compatibility metric. In contrast to Laga et al. we design our graph kernels to measure cross-class functional compatibility by choosing a different set of feature descriptors and then learning their individual importance and kernel parameters from training data. Our procedure offered dramatic improvements in performance compared to using the formulation of [Laga et al. 2013] as-is (Section 9).

Per-Element Descriptors. We encode each element's gross geometry and context within the overall shape using the following set of descriptors: the element's relative position with respect to its containing shape, encoded by the location of its markers such as its center of mass, lowest and highest points with respect to global object's markers (center of mass and its projections on supporting planes); its relative dimensions with respect to the object's dimensions; its mass distribution; and the relative orientation of the element's major axis with respect to the object's coordinate axes. The full set of the detailed 13 descriptors is provided in the Appendix.

Pairwise Descriptors. For each pair of elements connected by an edge in our graph we compute two sets of relative pairwise descriptors, using the same measurement as for individual elements, but computed for each element with respect to its graph neighbor rather than with respect to the containing object.

We assemble these descriptors into an element compatibility measure and learn their respective weights as discussed in Section 5.2. Intuitively, the learned weights indicate which geometric descriptors are more relevant for evaluating functional compatibility between elements on functionally different shapes.

Element Compatibility. We evaluate compatibility between pairs of elements on two shapes by comparing the graph walks initiated at their corresponding nodes in the respective graphs. Given an element p in a shape S, an n^{th} order (length) walk $W_S^{(n)}(p)$ is defined as a finite sequence of $n+1$ vertices and n edges forming a continuous path in the graph. Given another element q in another shape E, the n^{th} order similarity $K^{(n)}(p,q)$, defined for the n^{th} order walks starting at p and q, is given by the recursive formula:

$$K^{(n)}(p,q) = K_{node}(p,q) \cdot \sum_{\substack{p' \in \mathcal{N}(p) \\ q' \in \mathcal{N}(q)}} K_{edge}(e_{pp'}, e_{qq'}) \cdot K^{(n-1)}(p',q') \tag{1}$$

where $K_{node}(p,q)$ is a kernel function comparing node descriptors for elements p and q; $\mathcal{N}(p)$ and $\mathcal{N}(q)$ represent the set of neighboring elements for p and q respectively; and $K_{edge}(e_{pp'}, e_{qq'})$ is a kernel function comparing edge (i.e. pairwise) descriptors that represent relationships between elements. For $n = 0$ (0^{th} order walk), the kernel function only evaluates $K_{node}(p,q)$.

We define the node and edge kernels as a weighted combination of Radial Basis Function (RBF) kernels with learned parameters. The kernels evaluate node similarity and edge similarity respectively as follows:

$$K_{node}(p,q) = \sum_k w_k \cdot \exp\left\{ -\frac{D_k^2(p,q)}{2\sigma_k^2} \right\} \tag{2}$$

$$K_{edge}(e_{pp'}, e_{qq'}) = \delta(e_{pp'}, e_{qq'}) \sum_l w_l \cdot \exp\left\{ -\frac{D_l^2(e_{pp'}, e_{qq'})}{2\sigma_l^2} \right\} \tag{3}$$

where $D_k(p,q)$, $D_l(e_{pp'}, e_{qq'})$ are distances between the descriptors of nodes and edges respectively, $\delta(e_{pp'}, e_{qq'})$ is a binary function that returns 1 when the edges $e_{pp'}, e_{qq'}$ are of the same type (symmetry, containment, or adjacency) and 0 otherwise.

Compatibility between elements p and q is then defined as a weighted combination of n^{th} order similarities between them across a range of walk lengths n:

$$K_{func}(p,q) = \sum_n w_n K^{(n)}(p,q) \tag{4}$$

where w_n is a learned weight for each different walk length. For computational efficiency, in our implementation we use walks up to length 5, as in our experiments the learned weights assigned to longer walks were negligible. The above similarity function is a kernel function itself, and can therefore be normalized to ensure consistent similarity values for graphs of different size [Lanckriet et al. 2004]:

$$\hat{K}_{func}(p,q) = \frac{K_{func}(p,q)}{\sqrt{K_{func}(p,p) \cdot K_{func}(q,q)}} \tag{5}$$

Given positive weights $\{w_k\}$, $\{w_l\}$, $\{w_n\}$, our kernel is guaranteed to be positive definite, thus distances between elements can be derived from the above kernel as follows [Schölkopf 2001]:

$$D_{func}(p,q) = \sqrt{K(p,p) - 2K(p,q) + K(q,q)} \tag{6}$$

where $K(p,p), K(q,q)$ represent the self-similarities of elements in the graphs used for normalization.

To evaluate compatibility between pairs of symmetric group of elements G_E, G_S, we find the best pairwise element match. If the best match is compatible for substitution or deformation, this indicates that at least one pair of elements are interchangeable. The rest of the elements within their respective symmetric group can be substituted or deformed under symmetry constraints. Thus, we use the compatibility of the best element match for measuring group compatibility $D_{func}(G_E, G_S)$:

$$D_{func}(G_E, G_S) = \min_{p \in G_E, q \in G_S} D_{func}(p, q). \tag{7}$$

Shape Compatibility. We employ the shape compatibility measure after each editing operation, to evaluate whether the resulting new shape S' is functionally compatible with the original target one T . We define compatibility as the maximal compatibility distance between corresponding elements on the two shapes, seeking the worst-case influence on shape compatibility:

$$D_{func}(S', T) = \max_{p \in T, p' \in S'} D_{func}(p, p') \tag{8}$$

where p is an element on the original target shape, and p' is its corresponding substituted or original element on the generated shape. Note that while added or removed parts are not explicitly accounted for by this metric, their presence or absence will be reflected in the graph kernels of their neighboring elements.

Curve Compatibility. To evaluate curve compatibility for curve-based deformation, we take into account the compatibility of the elements they belong to, and the similarity between the shape, relative location and size of the curves within the overall shape. The list of curve descriptors is provided in the Appendix.

For a pair of *view-independent* curves e, f belonging to elements p, q respectively, their compatibility is expressed as follows:

$$K_{curve}(e, f) = K_{func}(p, q) + \sum_m w_m \cdot \exp\left\{ -\frac{D_m^2(e, f)}{2\sigma_m^2} \right\} \tag{9}$$

and $D_m(e, f)$ represent distances between curve descriptors and $\{w_m\}, \{\sigma_m\}$ are learned parameters. We note that the curves are segmented according to our hierarchical element segmentation such that the curve segments can be associated with the corresponding element compatibilities.

For a pair of *view-dependent* curves e, f , we additionally take into account the distance between the views they are generated from:

$$K_{viewcurve}(e, f) = K_{curve}(e, f) + w_v \exp\left\{ -\frac{||\mathbf{v}(e) - \mathbf{v}(f)||^2}{2\sigma_v^2} \right\} \tag{10}$$

where $\mathbf{v}(e)$ and $\mathbf{v}(f)$ represent given 3D viewpoint location for these two curves, and w_v, σ_v are learned parameters.

Curve compatibility is defined by converting kernel similarity to distance (as in Equation 6). Our deformation only pairs same-type curves; that is, we do not match view-dependent curves on one shape with view-independent curves on the other.

5.2 Parameter Learning

We algorithmically learn the parameters of our element and curve compatibility measures. For element compatibility these include the kernel weights $\{w_k\}_{k=1...K}$, $\{w_l\}_{l=1...L}$, $\{w_n\}_{n=1...N}$ and RBF variances $\{\sigma_k\}_{k=1...K}$, $\{\sigma_l\}_{l=1...L}$ (58 parameters in total). For curve compatibility these include the weights $\{w_m\}_{m=1...M}$, $\{\sigma_m\}_{m=1...M}$ and w_v, σ_v (8 parameters in total). We use the same learning procedure for both. We note that these parameters can vary across object classes - compatibility criteria for chairs and sofas may differ from those for beds and cabinets. We consequently learn these parameters separately for each pair of shape classes. In our experiments we used coarse class classification with up to five classes per broad shape category (e.g. tables, chairs, sofas, cabinets, and beds for furniture).

Clearly, learning requires training data. One possibility to create a training dataset is to manually specify pairs of compatible or incompatible elements or curves across shapes. However, creating such a dataset requires human labor and supervision. Instead, we developed an automatic procedure to create training data. Specifically, we observe that online repositories such as Google Warehouse already contain a significant number of coordinated sets of objects in the same style. Since these shapes are designed to have the same style, many of the objects in these scenes contain elements which are identical up to an affine transformation. By construction these elements are compatible, since they can be clearly substituted (subject to appropriate scaling) without affecting shape functionality. Consequently, detected pairs of such compatible elements across different models yield valuable training data for learning compatibility parameters. We clearly detect only a subset of compatible pairs since compatible elements may have different geometry even on same set shapes. However, our compatibility function is based on coarse-scale element properties and context and does not consider fine-level element geometry. Thus, restricting our training data to elements identical up to affine transformation, does not, in our experience, bias our learning setup. On the assumption that most random element substitutions would lead to structurally or functionally invalid results, we complement our compatible pairs with less compatible ones using random pair assignment.

Given a dataset of scenes downloaded from Google Warehouse, we first segment each model, extracting elements and curves (Section 4); we then use an ICP based alignment to compute all pairs of elements approximately identical up to an affine transformation. Given these training pairs, the goal of parameter learning is to compute the set of parameters with which our compatibility function will, on average, deem these pairs p, q more compatible than element pairs which contain one of the elements in our compatible pair and a randomly selected one - p, r or q, r. We use a probabilistic framework that is well suited to handle such relative comparisons for training and is known to be robust to outliers [Burges et al. 2005; Tamuz et al. 2011]. We express the probability that a pair $\{p, q\}$ is more compatible than $\{p, r\}$ (or more compactly $p_q \triangleright p_r$) as:

$$P(p_q \triangleright p_r) = \sigma\Big(D_{func}(p, r) - D_{func}(p, q)\Big) \tag{11}$$

where $\sigma(x)$ is a sigmoid function that converts the functionality differences into probabilities. We also include an L^1 norm as regularization term that minimizes the weights assigned to the different descriptors. The L^1-norm regularization, proposed by Tibshirani [1996], promotes sparsity by allowing some weights to dominate while pushing others toward zero. In addition, when the number of training pairs is small relative to the number of parameters, the regularization encourages more zero weights, leading to a simpler model with better predictive performance. Our regularizer is formulated as follows:

$$P(\mathbf{w}) = \exp\Big(-\lambda ||\mathbf{w}||_1 \Big) \tag{12}$$

where the weight vector $\mathbf{w}$ includes all kernel weights. The regularization parameter λ controls the degree of regularization and is automatically estimated through 10-fold cross-validation on the training set.

Given T training triplets p, q, r, we learn the parameter values that maximize the following likelihood:

$$L(\mathbf{w}, \boldsymbol{\sigma}) = \ln P(\mathbf{w}) + \sum_{t=1}^{T} \ln P(p_q[t] \triangleright p_r[t]) \tag{13}$$

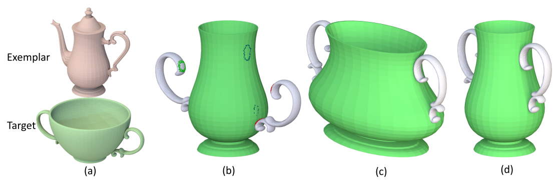

Figure 9: *Element alignment: (a) exemplar and target; (b) seed model and substituted-in element with identified slots and their covers; (c) alignment using non-uniform scaling across the board; (d) style and structure aware alignment.*

where vector $\boldsymbol{\sigma}$ includes all variances, $p_q[t] \triangleright p_r[t]$ refers to the automatically generated training triplet t. For element correspondences, we train the weights and variances by maximizing the above regularized likelihood function on the element training data for input shapes per each pair of classes. Then for curve correspondences, we train the weights and variances by again maximizing the same likelihood function, but this time using curve training data for input shapes per each pair of classes. We use bound constraints to enforce all parameters to be positive. To maximize our regularized likelihood function, we use the the L-BFGS-B method [Zhu et al. 1997]. We note that analytic gradients of our kernel functions with respect to weights can be derived following a recursive formulation explained in the Appendix. In our datasets, the number of our training inputs based on the ICP-aligned pairs varied from 25 to 300 depending on the pair of classes (most were above 100).

Automatic Threshold Selection. We use the detected element and curve correspondences to algorithmically select the compatibility threshold ϵ used in our tabu search. For each pair of classes c, c', we set the threshold $\epsilon_{c,c'}$ for element correspondence to the maximum distance between corresponding elements in the training data. We similarly use the maximum distance between corresponding curves in our training data as the threshold for curve compatibility. We note that we can safely use these maximum distances as thresholds since any outlier matches are pruned by the ICP matching step.

6 Element Alignment

Part and element adjacencies within an object obviously impact its functionality. In particular the locations of contact areas, or slots, connecting each element to the rest of the model are likely to reflect on this element's role within the larger whole. To preserve target functionality when adding or substituting elements into an edited seed shape we aim to, whenever possible, preserve all previously existing slots on both the incorporated element and the seed model, i.e. to keep previously covered, or in contact areas, similarly covered. We detect all slots on the seed shape and exemplar element, using the algorithm developed by [Kalogerakis et al. 2012] for part-based model synthesis. The identified slots include shared boundary loops, in-contact surfaces, and part-intersections. To preserve functionality, we treat object contacts with the ground plane as additional slots. By construction, within a model each slot has an opposite matching, or *cover*, slot. To assemble the new model, we need to compute such covers for slots at the interface between the seed shape and the new element, and then transform the elements to bring all pairs of matching slots into contact (Figure 9).

Aligning, or bringing slots into contact, often requires changes to element geometry, e.g. incorporating an armchair back into a sofa requires stretching it. Yet, unconstrained changes to element shape, can decrease the output functionality and negatively affect style similarity with the exemplar. Thus in performing alignment we seek to achieve the balance of changing element geometry enough to provide coverage but with minimal style and function degradation. While the method of Kraevoy et al. [2008] seeks to preserve geometric features when non-uniformly resizing models, adapting it to our setting and applying it on a per element basis using coverage constraints is too computationally expensive. Instead, we facilitate an effective yet efficient alignment computation using the following framework. We first restrict the set of allowable per-element transformations to translation, rotation, and axial scaling. By preventing non-axial shear, and penalizing deviations from pure translation we seek to weakly preserve element proportions and orientation. However, applying a penalty approach to all elements uniformly is insufficient. Even small non-uniform scaling can lead to visible artifacts by breaking element symmetry (Figure 9c); and even small rotations of anisotropic elements can affect their look and functionality. We therefore disallow symmetry violating scaling and rotations that change the direction of the major axis on anisotropic elements. To detect both scenarios we use the element's oriented bounding box (OBB) . When an element has two or more OBB axes with roughly similar length we constrain our transformations to maintain their length ratio (we use 20% deviation as conservative threshold). We only allow rotations for elements that are either isometric or that have two near identical axis lengths, in the later case rotation is allowed only within the plane span by these axes, using the same threshold as above to detect similar axis lengths. We also note that from a style perspective changes in thickness of thin elements are particularly undesirable, and disallow such changes (an element is considered thin if one of its axes is shorter than 10% of the sum of all axis lengths).

As in many alignment settings we face a chicken-and-egg problem, we need correspondences to perform the desired transformations, but correspondences computed when two objects are far apart are not reliable. We consequently use an iterative-closest-point (ICP) strategy, iterating alignment and correspondence steps. We first approximately align the new element to the seed shape. For substitution we transform the incoming element to align its OBB with that of the outgoing one. During addition, the added-in element by construction has at least one adjacent exemplar element that had been incorporated into the seed. We therefore similarly transform the added-in element to align the slots it shares with those elements. We then locate and pair seed and element slots nearest to one another. For any unpaired slot we treat the closest points on the opposite model as the matching covers.

At each subsequent alignment step, to minimize changes in element proportions and orientations we first solve for closest slot alignment using only translations. If this step is unsuccessful, we use the set of permissible scaling constraints per element to perform a restricted scale plus translation closest-point alignment of all participating elements. For each element we restrict the scalings to the permissible ones, while seeking to distribute the amount of scaling evenly between all elements. If and when this step fails we repeat the closest-point alignment allowing restricted element rotations and scales. For all symmetric groups of incorporated or seed elements, we constrain the transformations to preserve these symmetries. We iterate between correspondence computation and alignment till distances no longer improve or full coverage is achieved.

7 Curve Based Deformation

The input to our curve based deformation is a handle curve on the currently processed seed shape and a corresponding exemplar curve. Our deformation step modifies the seed by replacing the handle with the exemplar curve while smoothly deforming the seed surface so as to conform to the new curve geometry while preserving local surface details (Figure 10). While multiple surface deformation methods exist, we found that the ARAP framework [Sorkine and Alexa 2007] works well in our setup, as it supports curve deformation handles and preserves local geometric features under significant handle deformations. To facilitate deformation, we first align the endpoints of the exemplar curve with those of the handle curve through translation and uniform scaling and use arc-length parameterization to define curve-to-curve correspondences. We then deform the seed model by moving handle vertices to corresponding

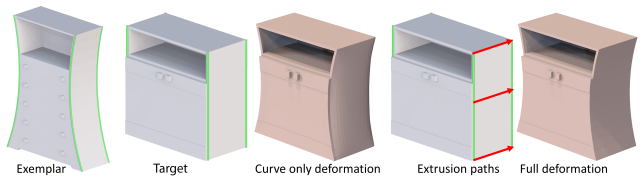

Figure 10: *Curve based deformation without (center) and with (right) swept surface edits.*

locations on the transformed exemplar curve. Using, the original, surface-based ARAP formulation as-is for large curve deformations can cause surface self-intersections. We therefore implemented ARAP on a volumetric graph, following the graph construction described in [Zhou et al. 2005] shown to prevent self-intersections in the case of Laplacian deformations. In our experiments, this modification allows for the intersection-free large deformations necessary to modify curve style.

We seek to impact not only the features but also the contours of the output shape, and note that man-made objects are frequently dominated by swept surfaces. We implement the desired contour changes by editing the sweep profiles on these surfaces (Figure 10, right). For each handle contour curve we examine whether its underlying surface is well defined by sweeping the contour handle along a path curve, and maintain these swept surfaces during deformation. For simplicity we only implemented this mechanism for the most common sweep cases, revolution and extrusion, where this structure is easiest to detect and preserve. Specifically, for each pair of similarly shaped and oriented handle curves on an element, we interpolate the curves and compare the distance from the resulting surface to the element. If the generated surface is close to the mesh surface, then we infer that it is a swept surface. To detect extrusions we use linear interpolation, and to detect surfaces of revolution we interpolate handle normals and positions.

8 Automatic Target Selection

Our output is dependent on the choice of a particular target shape (Figure 3). Typically the more similar the exemplar and the target are structurally, the more compatible their elements are, and the more complete, or compelling, the style transfer. Thus when users specify a database of shapes within a particular class as a target for style transfer, we use structural compatibility as a criterion for selecting the target shape to operate on within the database.

Intuitively one shape is more compatible with a given exemplar than another when a larger share of its elements are more compatible with exemplar elements. Given the exemplar shape E and a shape D within a target class, we compute their compatibility by first locating for each shape element the most compatible exemplar element, and then summing up the degrees of compatibility between them using the normalized kernel of Equation 5:

$$\hat{K}(D,E) = \sum_{p\in D, q\in E, q=s(p)} \hat{K}_{func}(p,q)$$

where p is an element on the database shape, and q is its most compatible element on the exemplar shape. A simple brute-force approach for selecting the most compatible shape is to evaluate these similarities across all database shapes and select the best one.

However, for large shape collections, this brute force approach is too slow. We speed up the process by leveraging the observation that in practice shape databases frequently contain clusters of structurally similar shapes. We first find such clusters, then select a representative shape per cluster, and finally perform the above computation only for those representative shapes, selecting one of them as

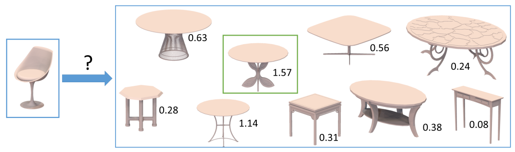

Figure 11: *Among all possible tables on the right we selected the highlighted one as most compatible target for the exemplar chair.*

the target. We perform clustering using affinity propagation [Frey and Dueck 2007] with the following similarity metric between two database shapes D_1, D_2:

$$\hat{K}(D_1,D_2) = \frac{1}{|P|}\sum_{p\in D_1, q\in D_2, q=s(p)} \hat{K}_{func}(p,q) + \frac{1}{|Q|}\sum_{q\in D_2, p\in D_1, p=s(q)} \hat{K}_{func}(p,q)$$

where $|P|$ is the number of elements in shape D_1, $|Q|$ is the number of elements in shape D_2, $s(p)$ returns the most similar element in D_2 to element p, $s(q)$ returns the most similar element in D_1 to element q. The affinity propagation method automatically infers both the number of clusters and their representative shapes.

9 Validation

We evaluate our method by synthesizing over a hundred new shapes using style transfer, see Figures 1 and 12 for representative examples, and supplementary material. We tested our method on four broad categories of everyday objects: furniture, lamps, cutlery, and coffee and tea sets. Our choice of categories was motivated both by availability and by diversity of functions and styles within each category. We use as inputs models from publicly available databases, 3D warehouse and TurboSquid. Throughout the paper and the accompanying video we demonstrate a diverse range of style transfer results. In our supplementary material, we include all our synthesized shapes including the exemplar along with the manually or automatically selected target shapes. We used automatic selection for 37 of the 126 generated models. Our results convincingly combine exemplar styles with target functions.

Perceptual Validation. We validate the key properties of our method via three user studies: one designed to evaluate the functionality of the output models, one designed to evaluate the degree of style similarity between the outputs and the exemplars, and one designed to specifically evaluate our compatibility metric against the most similar prior work [Laga et al. 2013]. We summarize those below. The full study details are provided in the appendix and supplementary material.

Style similarity. Style similarity is an inherently relative notion, thus asking if two shapes have the same style is often inconclusive. We consequently use relative comparison to assess our results. We asked participants to compare style similarity between an exemplar model and our output generated from it, against style similarity between the exemplar and a range of alternatives, aiming to ascertain the degree of success our method has at believably transferring style. We used questionnaires based on triplets of models, laid out with one shape image on the top and two on the bottom. The shape on the top (A) is an exemplar shape and one of the two shapes on the bottom (B or C, assigned randomly) is the top result synthesized by our method using this exemplar and a target in a different functional class. The second shape on the bottom is in the

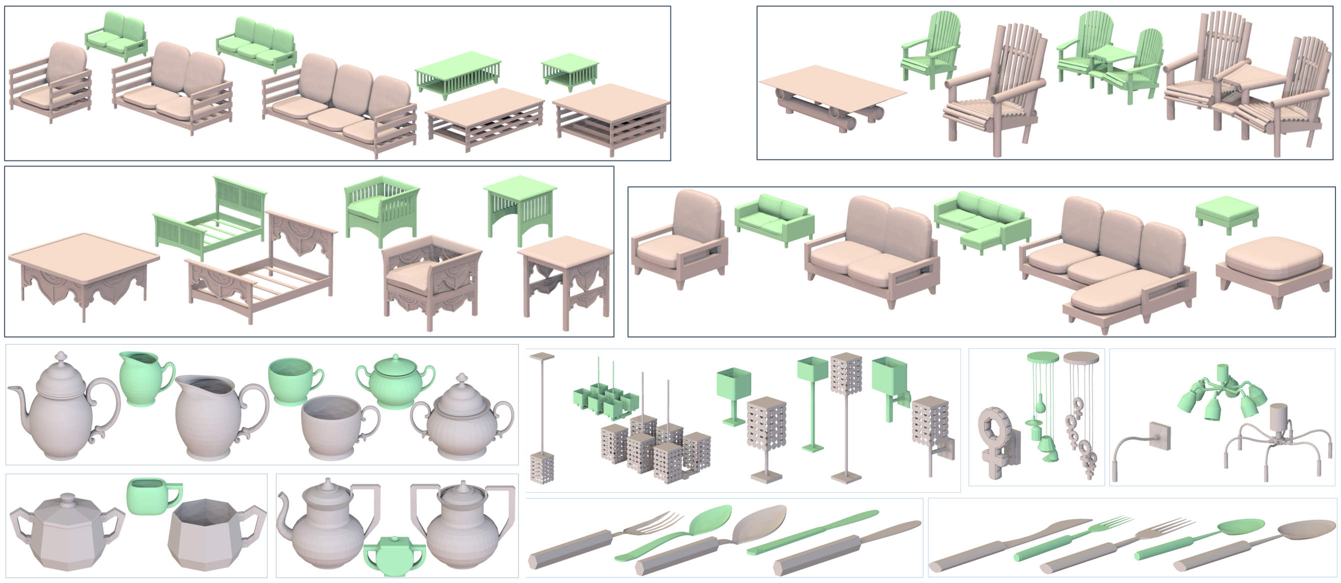
Figure 12: *Typical style transfer results. For each group we show the exemplar first then, multiple synthesized outputs in the same style with targets shown as insets.*

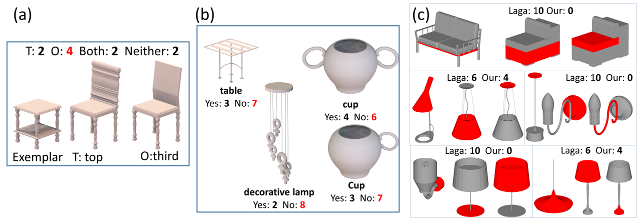

Figure 13: *Study outlier examples: (a) the* only *query where participants deemed the third ranked result more stylistically similar to exemplar than the top; (b) the only four of our top results participants deemed non-functional; (c) examples of 5% of queries where participants ranked Laga et. al correspondences as superior to ours.*

	plurality					raw votes			
	T	O	both	neither	draw	T	O	both	neither
top (T) vs target (O)	100.0%	0.0%	0.0%	0.0%	0.0%	93.2%	0.4%	0.5%	6.0%
top (T) vs third (O)	78.9%	1.4%	9.9%	5.6%	4.2%	68.6%	5.5%	15.1%	10.8%
top (T) vs original (O)	38.8%	26.9%	19.4%	1.5%	13.4%	39.1%	32.5%	21.2%	7.2%

Table 1: *Style similarity study results: per-query plurality responses (left) and raw vote percentages (right).*

same functional class as the output and is randomly selected among the following alternatives: a shape from a style-coordinated preexisting scene which included the exemplar A - these shapes can be viewed as plausible ground truth for style transfer; a shape synthesized by our method using the same exemplar, but ranked as third in terms of its stylistic similarity to the exemplar - this shape is useful to evaluate the meaningfulness of our ranking; and the target shape used for style transfer which serves as a random baseline, expected to be arbitrarily different style-wise from the exemplar. Subjects were asked the question "Which of the two shapes on the bottom (B or C) is more similar, style-wise, to the shape on the top (A)?" and were asked to select one of the following answers: "(i) B, (ii) C, (iii) can't tell - both B and C, (iv) can't tell - neither B nor C".

We assembled a total of 264 queries, up to three per each of our generated outputs comparing each output to all available alternatives. We gathered answers to each query from 10 different, reliable users

	plurality			raw votes	
	yes	no	draw	yes	no
target	96.0%	1.6%	2.4%	89.8%	10.2%
top	92.1%	3.2%	4.8%	86.7%	13.3%
third	90.1%	7.0%	2.8%	86.3%	13.7%
lax compatibility	69.8%	23.0%	7.1%	68.4%	31.6%
Laga et al.	65.3%	29.8%	5.0%	65.2%	34.8%
exemplar	12.2%	86.1%	1.7%	17.0%	83.0%

Table 2: *Functionality study results: per-query plurality responses (left) and raw vote percentages (right).*

	plurality	raw votes
ours	93.3%	91.8%
Laga et al.	5.0%	8.2%
draw	1.7%	

Table 3: *Element compatibility study results: per-query plurality responses (left) and raw vote percentages (right).*

using the procedure described in the Appendix. Vote distribution by query and raw vote percentages for each answer are listed in Table 1. Participants perceived our synthesized shapes as at least as similar style-wise to the exemplars as the ground truth models. Furthermore the top-ranked shapes were perceived as more stylewise similar to the exemplar compared to the third-ranked ones, and drastically more similar when compared to the baseline target shapes. The supplementary material contains the full results. The only query for which our top result was ranked below the third one is shown in Figure 13. These results strongly validate our claim of consistently successful style transfer across shapes with different functionality.

Functionality. Functionality is a largely boolean property, thus to evaluate how well our outputs preserve target functionality we show participants one model at a time and ask "Is this a functional X?" where X is the name of the specific, narrow, target class used for synthesis, e.g. coffee table, loveseat, side table, etc. Users were asked to choose either "yes" or "no". To provide a baseline to compare against, in addition to showing participants our top and third ranked results, we also included equal numbers of models from the following groups: original target models - intuitively one would expect a near 100% positive response on these models, with the actual positive response rate providing a good upper bound to compare against; top-ranking results synthesized using our framework but with either our compatibility metric but with a 10-times more lax compatibility threshold, or with the original threshold but with the similarity metric of Laga et al. [2013] (based on the graph encoding, edge relationships and kernel parameters described in their paper) - intuitively we expect these two sets of results to produce less positive responses than ours; and last exemplar models - these serve as the lower bound, as they do not share target functionality.

We assembled a total of 611 queries and gathered answers to each query from 10 different, reliable users using the procedure described in the Appendix. The responses are reported in Table 2. The results demonstrate that our synthesized shapes are deemed to fulfill their function at nearly the same rate as the ground-truth target models. The only four outputs deemed non-functional by respondent majority are shown in Figure 13. If we relax our learned threshold for element compatibility, the functional plausibility of shapes drops significantly. Similarly, over a third of the shapes synthesized using Laga et al.'s metric are found to violate functionality considerations. The results validate the second goal of our method - the ability to reliably preserve target functionality during transfer. The comparisons to alternative methods also confirm that our compatibility metric and the automatic threshold setting we employ (Section 5) are key to this success.

Element compatibility metric. We directly evaluate our metric's effectiveness by comparing the correspondences it computes against those produced using the metric of [Laga et al. 2013]. To compare the methods we randomly selected pairs of an exemplar and a target across our inputs, and then selected a random element on the exemplar. We ran both methods to find its corresponding, or most compatible element on the target. Of the 660 queries assembled this way, the methods were in agreement 54.7% of the time.

Our user study consequently focused on the remaining queries. We used questionnaires based on triplets of models, laid out with one shape image on top and two on the bottom. The shape on the top (A) is an exemplar shape with the selected element highlighted, one of the two shapes on the bottom (B or C, assigned randomly) is the compatible target element selected by our method and the second shape is the element selected by the method of Laga et al. Subjects were asked "Which of the two highlighted parts on the bottom (B or C) is MORE similar functionality wise to the highlighted part on the top (A)?", and were asked to select either B or C. The user study had the same format and filters as the first study.

Study participants selected our result 93% of the time, and only on 5% or 15 queries did a plurality of respondents prefer the correspondences computed by Laga et. al (1.7% were a draw). Most of the outliers (12) were on queries which compared elements on lamps with different attachment mechanisms (floor vs ceiling vs wall) . Representative outliers are shown in Figure 13. These results confirm that our metric is significantly better aligned with human perception of functional part compatibility. At the same time additional features may be useful to consider to address attachment diversity when processing hanging shapes.

Implementation and Runtimes. Our method is implemented in C++ and source code is publically available on our project web page. Our method takes on average 6 min to synthesize a new model, with roughly 2 min out of this time spent pre-processing the models. The rest of the time is spent in the tabu search. Tabu search runtime depends on the complexity and number of operations, and ranges from 2 min for typical models to up to 10 min for the slowest ones. Learning the parameters of our compatibility measure requires about one hour for each pair of shape classes. This learning step is an offline process: once the compatibility measure is learned, evaluating the compatibility between all pairs of elements on two shapes takes only a few seconds. All running times are reported on an Intel E5-2697 v2 processor.

10 Conclusion

We have described the first algorithm for synthesizing shapes by transferring style between man-made objects with different structure and functionality. As demonstrated by our results, given a single exemplar model, our method is able to successfully generate functional, plausible, similar-style models in a wide range of shape classes. Key to our success is a novel, learned metric designed to assess element compatibility across shapes with different structure and function.

There are many exciting directions for future work. Our algorithm requires as input an exemplar 3D model that represents the desired style to be transferred to other objects. It would be interesting to explore other input modalities that describe style, such as sketches and natural language. Furthermore, our algorithm leverages the structure of a target shape, either specified manually or retrieved automatically, to synthesize new shapes. Instead of relying on a pre-existing target shape structure, it would be interesting to employ generative models that are capable of generating plausible shape structure and accurate surface geometry automatically. Such models could also avoid the need of slot-based part alignment that may fail when slots largely differ in number, size and orientation. Finally, another interesting direction would be to combine our structure-based functional compatibility metric with functionality models [Kim et al. 2014; Hu et al. 2016] that consider part interactions with agents and other objects in a scene to improve correspondences especially for parts where such interactions are meaningful.

11 Acknowledgments

Kalogerakis and Wang gratefully acknowledge support from NSF (CHS-1422441, CHS-1617333, IIS-1423082). Sheffer gratefully acknowledges support from NSERC discovery and DAS grants. We thank Nicholas Vining for proofreading the paper and narrating the video, Mikhail Bessmeltsev for useful suggestions on the figures, and the anonymous reviewers for their comments.

References

ASAFI, S., GOREN, A., AND COHEN-OR, D. 2013. Weak convex decomposition by lines-of-sight. In *Proc. SGP*.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graphics 29*, 4.

BURGES, C., SHAKED, T., RENSHAW, E., LAZIER, A., DEEDS, M., HAMILTON, N., AND HULLENDER, G. 2005. Learning to rank using gradient descent. In *Proc. ICML*.

CHAUDHURI, S., KALOGERAKIS, E., GUIBAS, L., AND KOLTUN, V. 2011. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph. 30*, 4.

FREY, B. J., AND DUECK, D. 2007. Clustering by passing messages between data points. *Science 315*.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *ACM Trans. Graphics 23*, 3.

GLOVER, F., AND LAGUNA, M. 1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proc. SIGGRAPH*.

HERTZMANN, A., OLIVER, N., CURLESS, B., AND SEITZ, S. M. 2002. Curve analogies. In *Proc. Eurographics workshop on Rendering*.

HU, R., ZHU, C., VAN KAICK, O., LIU, L., SHAMIR, A., AND ZHANG, H. 2015. Interaction context (icon): Towards a geometric functionality descriptor. *ACM Trans. Graph. 34*, 4.

HU, R., VAN KAICK, O., WU, B., HUANG, H., SHAMIR, A., AND ZHANG, H. 2016. Learning how objects function via co-analysis of interactions. *ACM Trans. Graph., to appear*.

HUANG, Q.-X., SU, H., AND GUIBAS, L. 2013. Fine-grained semi-supervised labeling of large shape collections. *ACM Trans. Graph. 32*, 6.

HUANG, H., KALOGERAKIS, E., AND MARLIN, B. 2015. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Computer Graphics Forum 34*, 5.

HUANG, Q., WANG, H., AND KOLTUN, V. 2015. Single-view reconstruction via joint analysis of image and shape collections. *ACM Trans. Graph. 34*, 4.

JIANG, Y., KOPPULA, H., AND SAXENA, A. 2013. Hallucinated humans as the hidden context for labeling 3d scenes. In *Proc. CVPR*.

KALOGERAKIS, E., NOWROUZEZAHRAI, D., SIMARI, P., MCCRAE, J., HERTZMANN, A., AND SINGH, K. 2009. Data-driven curvature for real-time line drawing of dynamic scene. *ACM Trans. Graph. 28*, 1.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. 31*, 4.

KIM, V. G., CHAUDHURI, S., GUIBAS, L., AND FUNKHOUSER, T. 2014. Shape2pose: Human-centric shape analysis. *ACM Trans. Graph. 33*, 4.

KLEIMAN, Y., VAN KAICK, O., SORKINE-HORNUNG, O., AND COHEN-OR, D. 2015. Shed: Shape edit distance for fine-grained shape similarity. *ACM Trans. Graph. 34*, 6.

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. *ACM Trans. Graphics 27*, 5.

KREAVOY, V., JULIUS, D., AND SHEFFER, A. 2007. Model composition from interchangeable components. In *Proc. Pacific Graphics*, 129–138.

LAGA, H., MORTARA, M., AND SPAGNUOLO, M. 2013. Geometry and context for semantic correspondences and functionality recognition in man-made 3d shapes. *ACM Trans. Graph. 32*, 5.

LANCKRIET, G. R. G., CRISTIANINI, N., BARTLETT, P., GHAOUI, L. E., AND JORDAN, M. I. 2004. Learning the kernel matrix with semidefinite programming. *J. Machine Learning Research 5*.

LEWIS, M. 2008. *Architectura: elements of architectural style*. Barrons Educational Series.

LI, H., ZHANG, H., WANG, Y., CAO, J., SHAMIR, A., AND COHEN-OR, D. 2013. Curve style analysis in a set of shapes. *Computer Graphics Forum 32*, 6.

LIU, H., VIMONT, U., WAND, M., CANI, M.-P., HAHMANN, S., ROHMER, D., AND MITRA, N. J. 2015. Replaceable substructures for efficient part-based modeling. *Comp. Graph. Forum 34*, 2.

LIU, T., HERTZMANN, A., LI, W., AND FUNKHOUSER, T. 2015. Style compatibility for 3d furniture models. *ACM Trans. Graphics 34*, 4.

LUN, Z., KALOGERAKIS, E., AND SHEFFER, A. 2015. Elements of style: Learning perceptual shape style similarity. *ACM Trans. Graph. 34*, 4.

MA, C., HUANG, H., SHEFFER, A., KALOGERAKIS, E., AND WANG, R. 2014. Analogy-driven 3D style transfer. *Computer Graphics Forum 33*, 2.

NORMAN, D. 1988. *The Design of Everyday Things*. Basic Books.

NUTTING, W. 1928. *Furniture Treasury*. Gr Macmillan Publishing.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *Proc. Siggraph*.

SAVVA, M., CHANG, A. X., HANRAHAN, P., FISHER, M., AND NIESSNER, M. 2014. Scenegrok: Inferring action maps in 3d environments. *ACM Trans. Graph. 33*, 6.

SAVVA, M., CHANG, A. X., HANRAHAN, P., FISHER, M., AND NIESSNER, M. 2016. PiGraphs: Learning Interaction Snapshots from Observations. *ACM Trans. Graph., to appear*.

SCHÖLKOPF, B. 2001. The kernel trick for distances. In *Proc. NIPS*.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. SGP*.

TALTON, J., YANG, L., KUMAR, R., LIM, M., GOODMAN, N. D., AND MĚCH, R. 2012. Learning design patterns with bayesian grammar induction. In *Proc. UIST*, 63–74.

TAMUZ, O., LIU, C., BELONGIE, S., SHAMIR, O., AND KALAI, A. 2011. Adaptively learning the crowd kernel. In *Proc. ICML*.

TIBSHIRANI, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society 58*.

VAN KAICK, O., ZHANG, H., HAMARNEH, G., AND COHEN-OR, D. 2011. A survey on shape correspondence. *Computer Graphics Forum 30*, 6, 1681–1707.

XU, K., LI, H., ZHANG, H., COHEN-OR, D., XIONG, Y., AND CHENG, Z.-Q. 2010. Style-content separation by anisotropic part scales. *ACM Trans. Graph. 29*, 6.

XU, K., ZHANG, H., COHEN-OR, D., AND CHEN, B. 2012. Fit and diverse: Set evolution for inspiring 3d shape galleries. *ACM Trans. Graph. 31*, 4.

XU, K., KIM, V. G., HUANG, Q., AND KALOGERAKIS, E. 2016. Data-driven shape analysis and processing. *Computer Graphics Forum, to appear*.

YUMER, M., AND KARA, L. 2012. Co-abstraction of shape collections. *ACM Trans. Graphics 31*, 6, 166:1–166:11.

YUMER, M., AND KARA, L. 2014. Co-constrained handles for deformation in shape collections. *ACM Trans. Graph. 32*, 6.

YUMER, M. E., CHAUDHURI, S., HODGINS, J. K., AND KARA, L. B. 2015. Semantic shape editing using deformation handles. *ACM Trans. Graph. 34*, 4.

ZHENG, Y., COHEN-OR, D., AND MITRA, N. J. 2013. Smart variations: Functional substructures for part compatibility. *Comp. Graph. Forum 32*, 2.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3.

ZHU, C., BYRD, R. H., LU, P., AND NOCEDAL, J. 1997. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw. 23*, 4.

ZHU, Y., FATHI, A., AND FEI-FEI, L. 2014. Reasoning about object affordances in a knowledge base representation. In *Proc. ECCV*.

User study	style similarity	functionality	element compatibility
# total users	155	341	140
# reliable users	140	268	139
# rejected users	15	73	1
# male	83	175	78
# female	72	165	62
# unknown gender	0	1	0
# age 18-35	103	221	84
# age 36-50	40	86	42
# age > 50	12	33	14
# unknown age	0	1	0
# no post-secondary education	17	30	14
# yes post-secondary education	138	307	125
# other education level	0	4	1

Table 4: *Participant statistics*

	majority					raw votes			
	B	C	both	neither	draw	B	C	both	neither
top (B) vs target (C)	126	0	0	0	0	1174	5	6	75
top (B) vs third (C)	56	1	7	4	3	487	39	107	77
top (B) vs original (C)	26	18	13	1	9	262	218	142	48

Table 5: *Detailed style similarity study results.*

	majority			raw votes	
	yes	no	draw	yes	no
different class	14	99	2	196	954
original target	121	2	3	1131	129
top ranking	116	4	6	1092	168
third ranking	64	5	2	613	97
lax compatibility	88	29	9	862	398
Laga et al.	79	36	6	789	421

Table 6: *Detailed functionality study results.*

	majority	raw votes
ours	280	2755
Laga et al.	15	245
draw	5	

Table 7: *Detailed element compatibility study results.*

Appendix

Study Format

All three studies were conducted via Mechanical Turk. Each questionnaire contained 25 unique queries. For studies with triplet based questions (style and compatibility) each question was repeated twice, with B and C flipped, to measure participant persistence. For the functionality questionnaire, we similarly repeated the same query twice. To collect a diverse set of answers per query and avoid any individual bias, we allowed each participant to complete only one questionnaire. Participants were rewarded $\$0.75$ for each completed questionnaire with a triplet based question, and $\$0.5$ for filling the functionality questionnaire. Since any large-scale, study faces the risk of attracting unreliable respondents, we detected and discarded outlier responses using a two stage filter. Participants who gave two different answers to more than 8 out of the 25 unique queries in the questionnaire, or took less than 3 minutes for triplet based questioners and 2 minutes for the functionality one to complete it, were classified as unreliable and all their answers were discarded. For the functionality questionnaire we also classified as unreliable participants who picked the same yes/no answer for 20 queries or more. For all other participants, we ignored non-persistent answers, where a participant answered the same question differently. We gathered answers to each query from 10 different, reliable users.

Extra Study Statistics

Table 4 shows statistics about the participants in each user study, including gender, age and education level. The number of reliable participants and rejected participants are also listed. The number of raw votes in each user study are shown in Tables 5, 6 and 7.

Descriptors

In this section we described the descriptors used in the element compatibility formulation in section 5.1.

Per-Element descriptors. In total, we have 13 element descriptors yielding 13 distance measures. The first set of descriptors capture the relative location of the markers on an element, including its centroid, its centroid projected on the ground plane, then its highest point, lowest point, and its centroid projected onto the upright axis (i.e., representing height from the ground plane). When comparing nodes in our graph, element locations are expressed with respect to the object's coordinate system. When comparing edges in our graph, element locations are expressed with respect to the local corresponding system of its neighboring element in the graph (the local coordinate system is formed by the neighboring element's corresponding marker locations and object's axes). Each of the five relative locations yields a Euclidean distance when comparing two elements. The next three descriptors store the proportions of the element's axis-aligned bounding box, relative to the object's bounding box proportions when comparing nodes, and relative to the neighboring element's bounding box proportions when comparing edges. These proportions (one per each axis) yield three more distances. The next three descriptors are similar to the previous three, but instead of the bounding box proportions, we use the variance of the element point positions along the object's axes. The next descriptor stores the major orientation of the element estimated via PCA. When comparing nodes, we measure the angle difference between the major orientations of the two corresponding elements. When comparing edges, we measure the relative angle difference between the major orientations of the two corresponding elements with respect to their neighboring elements major axes. The last descriptor is a histogram that approximately captures the distribution of point samples in an element. We build a $4 \times 4 \times 4$ grid and compute a histogram by counting how many sample points on the element surface are inside each bin. When comparing nodes, we compute the euclidean distance between the histograms for the corresponding elements. When comparing edges, we compute a histogram for each edge measuring the absolute difference of bin values between the corresponding histograms of neighboring elements, then measure the euclidean distance between the resulting histograms.

Curve descriptors. In total, we get 3 distance measures between curve descriptors. The first one represents distance between centroids of the two curves and the second one represents differences between their arc lengths. The last one represents the average point-to-point distance after aligning the two input curves via ICP.

Gradient for learning

Learning the compatibility metric requires computing the analytic gradient of our objective function (Equation 13) with respect to our parameters. The loss function evaluates the compatibility metric, which is defined through the recursive formula of Equation 1. Interestingly, it turns out that the gradient also follows a similar recursive definition, which makes it possible to compute it efficiently. For clarity, we provide here the formulas that evaluate the partial derivatives of our objective function with respect to the node kernel parameters $\{w_k\}_{k=1...K}$ and RBF variances $\{\sigma_k\}_{k=1...K}$. The partial derivatives for the rest of the parameters follow a similar recursive computation. We begin by computing the gradient of the loss function with respect to the node kernel parameters:

$$\frac{\partial L(\mathbf{w},\boldsymbol{\sigma})}{\partial w_k} = -\lambda \cdot \text{sign}(w_k) + \sum_{t=1}^{T} \frac{\partial \ln P(p_q[t] \triangleright p_r[t])}{\partial w_k} \quad (14)$$

The gradient of the log likelihood per training example t can be expressed as:

$$\frac{\partial \ln P(p_q[t] \triangleright p_r[t])}{\partial w_k} = \Big(1 - \sigma\big(D_{func}(p,r) - D_{func}(p,q)\big)\Big) \cdot \cdot \Big(\frac{\partial D_{func}(p,r)}{\partial w_k} - \frac{\partial D_{func}(p,q)}{\partial w_k}\Big)$$

The partial derivatives of the distance function $D_{func}(p,r)$, and similarly for $D_{func}(p,q)$, are in turn computed as:

$$\frac{\partial D_{func}(p,q)}{\partial w_k} = \frac{\big(\frac{\partial K^{(n)}(p,p)}{\partial w_k} - 2\frac{\partial K^{(n)}(p,q)}{\partial w_k} + \frac{\partial K^{(n)}(q,q)}{\partial w_k}\big)}{2D_{func}(p,q)}$$

The above formula requires computing partial derivatives of our graph-based compatibility function. The derivatives also follow a recursive definition :

$$\frac{\partial K^{(n)}(p,q)}{\partial w_k} = \frac{\partial K_{node}(p,q)}{\partial w_k} \cdot \sum_{\substack{p' \in \mathcal{N}(p) \\ q' \in \mathcal{N}(q)}} K_{edge}(e_{pp'}, e_{qq'}) \cdot K^{(n-1)}(p',q') + K_{node}(p,q) \cdot \sum_{\substack{p' \in \mathcal{N}(p) \\ q' \in \mathcal{N}(q)}} K_{edge}(e_{pp'}, e_{qq'}) \cdot \frac{\partial K^{(n-1)}(p',q')}{\partial w_k}$$

To evaluate the above formula, the partial derivatives of the node similarity functions with respect to the kernel node parameters are required. These are computed as follows:

$$\frac{\partial K_{node}(p,q)}{\partial w_k} = \exp\Big\{ - \frac{D_k^2(p,q)}{2\sigma_k^2} \Big\}$$

Computing the partial derivatives of our objective function with respect to the RBF variances follow the same procedure as above with only two differences: the sign term in Equation 14 is omitted (no L^1-norm regularization is used for variances since sparsity is not required for them) while the partial derivatives of the kernel node functions are instead expressed as follows:

$$\frac{\partial K_{node}(p,q)}{\partial \sigma_k} = \frac{w_k D_k^2(p,q)}{\sigma_k^3} \exp\Big\{ - \frac{D_k^2(p,q)}{2\sigma_k^2} \Big\}$$

Framework Pseudocode

For clarity we include the detailed pseudocode of the tabu search algorithm described in Section 3.

```
input  : Exemplar shape $E$ in class $c'$, Target shape $T$ in class $c$
output: An output list $\mathcal{O}$ of new shapes
Initialize search list $\mathcal{L} = \{T\}$
repeat
    Choose shape $S = \arg\min_{T' \in \mathcal{L}} D_{style}(E, T')$
    Remove shape $S$ from search list $\mathcal{L}$

    // Search for element substitutions
    for each element (or symmetric group of elements) $G_S$ in $S$ do
        Find elements $G_E$ in shape $E$ with $D_{func}(G_E, G_S) < \epsilon_{c,c'}$
        for each retrieved element $G_E$ do
            if replacing $G_S$ with $G_E$ drops $D_{style}(E, S)$ then
                Construct new shape $S'$ by aligning $G_E$
                if $D_{func}(S', T) < \epsilon_{c,c}$ and alignment is successful
                 then
                    insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                    (unless a copy of $S'$ already exists in the output list)
                end
            end
        end
    end

    // Search for curve-based deformation
    for each curve (or symmetric group of curves) $C_S$ in $S$ do
        Find curve $C_E$ in shape $E$ with $D_{curve}(C_E, C_S) < \epsilon^{curve}_{c,c'}$
        for each retrieved curve $C_E$ do
            Construct new shape $S'$ by deforming $S$ to align with $C_E$
             if the constructed new shape $S'$ drops $D_{style}(E, S)$ then
                if $D_{func}(S', T) < \epsilon_{c,c}$ then
                    insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                    (unless a copy of $S'$ already exists in the output list)
                end
            end
        end
    end

    // Search for element additions
    for each non-used element (or group) $G_E$ in $E$ do
        if adding $G_E$ to $S$ drops style distance then
            Construct new shape $S'$ by aligning $G_E$ with $S$
            if $D_{func}(S', T) < \epsilon_{c,c}$ and alignment is successful then
                insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                (unless a copy of $S'$ already exists in the output list)
            end
        end
    end

    // Search for element removals
    for each non-substituted/added element (or group) $G_S$ in $S$ do
        if removing $G_S$ from $S$ drops style distance then
            Construct new shape $S'$ by removing $G_S$
            if $D_{func}(S', T) < \epsilon_{c,c}$ then
                insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                (unless a copy of $S'$ already exists in the output list)
            end
        end
    end
until search list $\mathcal{L}$ is empty
```

Figure 14: *Tabu search pseudo-code.*