EUROGRAPHICS 2011 / M. Chen and O. Deussen
(Guest Editors)

Volume 30 (2011), Number 2

Component-wise Controllers for Structure-Preserving Shape Manipulation

Youyi Zheng[1] Hongbo Fu[2] Daniel Cohen-Or[3] Oscar Kin-Chung Au[2] Chiew-Lan Tai[1]

[1]Hong Kong University of Science & Technology [2]City University of Hong Kong [3]Tel Aviv University

Abstract

Recent shape editing techniques, especially for man-made models, have gradually shifted focus from maintaining local, low-level geometric features to preserving structural, high-level characteristics like symmetry and parallelism. Such new editing goals typically require a pre-processing shape analysis step to enable subsequent shape editing. Observing that most editing of shapes involves manipulating their constituent components, we introduce component-wise controllers *that are adapted to the component characteristics inferred from shape analysis. The controllers capture the natural degrees of freedom of individual components and thus provide an intuitive user interface for editing. A typical model usually results in a moderate number of controllers, allowing easy establishment of semantic relations among them by automatic shape analysis supplemented with user interaction. We propose a component-wise propagation algorithm to automatically preserve the established inter-relations while maintaining the defining characteristics of individual controllers and respecting the user-specified modeling constraints. We extend these ideas to a hierarchical setup, allowing the user to adjust the tool complexity with respect to the desired modeling complexity. We demonstrate the effectiveness of our technique on a wide range of man-made models with structural features, often containing multiple connected pieces.*

1. Introduction

Shape editing is useful for reusing existing models and generating new shape variations. Recent editing techniques, especially for man-made models, aim to preserve structural, high-level characteristics instead of local, low-level geometric features. However, most accessible objects are represented as bare polygonal surfaces, without any semantics or structural information associated. Therefore, shape analysis is typically required prior to meaningful editing of polygonal models, demanding an *analyze-and-edit* approach [KSCOS08, GSMCO09, XWY*09, ZXTD10, CLDD09]. Generally speaking, such analysis results in a set of geometrical constraints. During editing, the manipulated object is optimized to satisfy the user-specified modeling constraints while respecting the geometric constraints derived from the analysis.

The key challenge in realizing such an analyze-and-edit approach is two-fold. First, how to analyze the given model; second, how to represent the results of the analysis to facilitate a constrained optimization. Gal et al. [GSMCO09] use 1D controllers which they call "wires". These controllers are defined at sharp features, which typically form closed flat 3D curves. The use of wires as controllers is based on the idea that most man-made shapes can be captured by a small number of 1D curves. While this is generally true, there are many man-made objects or parts which are smooth and lack characteristic sharp features. On the other hand, some objects might have many sharp features, leading to an excess of redundant controllers [GSMCO09] or constraints [KSCOS08]. This observation motivates the use of controllers that are independent of sharp features of the object and possibly hierarchically organized.

This paper presents an intuitive shape editing technique for man-made engineering models. Our technique builds on the analyze-and-edit paradigm, and introduces *volumetric* controllers derived from a hierarchical analysis of the given model. Our technique is based on two key ideas. First, the controllers are *component-wise* (Figure 1), and their shapes are adapted to the geometry of model components, enabling effective protection of the underlying shape and structure. Second, the user has easy control over the properties of the small number of individual controllers, their inter-relations as well as the hierarchical organization of controllers.

The controllers that we use are simple geometric parametric primitives (e.g., cylinders and cuboids, as shown in Figures 1 and 3). A controller forms a volumetric "cage" over the enclosed shape component and reflects the natural degrees of freedom afforded by the component geometry. We show that although such controllers have small degrees of freedom, they sufficiently express a variety of editing effects and provide intuitive guidance for shape editing.

The fact that the controller construction is hierarchical is crucial in providing the user access to *a rather small number* of controllers that capture the main components of the model to be edited. This makes tasks like manual specifi-

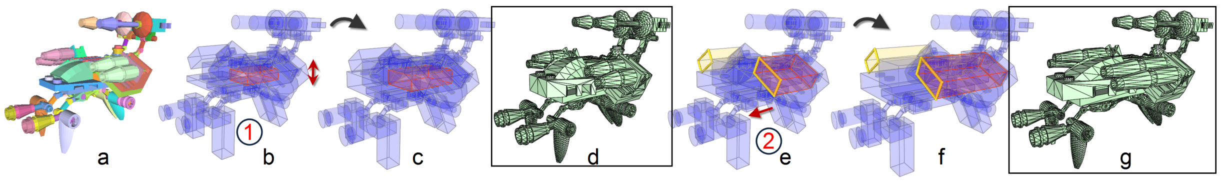

Figure 1: Our technique allows structure-preserving shape manipulation through construction of a small set of component-wise controllers (b), which are adapted to the component geometry of the given model (a) and thus capture its natural degree of freedom for editing. The editing result (g) is easily obtained by directly manipulating a subset of the controllers (highlighted in red here, with the numbers showing the editing order) while automatically computing the rest of the controllers to maintain both the individual shape characteristics and their inter-relationships.

cation or modification of the inter-relations (like symmetry and parallelism) among the high-level components easy to achieve. The user can also construct a customized controller hierarchy to easily adjust the tool complexity with respect to the desired modeling complexity.

When the user manipulates the controllers within their predefined editing degrees of freedom, the changes in the manipulated controllers are automatically propagated to the other controllers. We propose a component-wise propagation scheme that maintains both the defining characteristics of individual controllers and their mutual relations. We demonstrate that our technique allows easy manipulation of models while preserving their structures (see an editing example in Figure 1 and the interactive editing sessions in the accompanying video).

2. Related Work

In recent years, many deformation techniques have been developed for deforming polygonal meshes. These methods typically assume that the input mesh consists of homogeneous material, which works well for organic shapes with homogeneous material. The works of Popa et al. [PJS06] and Botsch et al. [BPGK06] are two exceptions: these methods can simulate surface deformation with non-homogenous material behavior by manual specification of local surface stiffness. However, all the above methods are designed for organic objects and exhibit rubber-like deformations, which would inappropriately distort structural features (e.g., orthogonality, parallelism) in man-made objects.

Surface deformation techniques can be roughly categorized into two groups: surface-based (e.g., [YZX*04, SLCO*04, BS08]) and space-based (e.g., [HSL*06, SSP07, BPWG07, LLCO08, BCWG09]). The latter group of techniques typically enclose a given object within a cage, which has roughly the same shape as the input model but with a significantly lower polygon count. Cage-based deformation techniques are mainly designed for simplicity, flexibility and speedup, regardless of the structural features of underlying objects or inter-relations among different components. Our volumetric controllers also form cages which drive the deformation of the enclosed local geometry. However, our cages are local and deliberately simple to inherit the inter-relations among the components of the underlying objects as well as to express the editing degrees of freedom required to maintain the intrinsic characteristics of the parts.

Our technique is most related to the *iWires* technique introduced by Gal et al. [GSMCO09], and builds upon their concept of analyze-and-edit. The key observation of their work is that man-made models can be characterized by a set of 1D wires with associated shape properties and inter-relationships among them (e.g., symmetry, parallelism), which are maintained during editing via a multi-stage constrained optimization. The wires are simply defined at sharp features without having control on their number and locations. Furthermore, wires as controllers are local and oblivious to the notion of components, thus making component-level shape manipulation (see an example in Figure 2) not easy to achieve. As we will explain later, our component-wise controllers also allow a simpler design of edit propagation over controllers.

There are other editing methods specifically designed for man-made objects. These methods, however, focus on specific editing operations like resizing [KSCOS08] or specific types of models like articulated models [XWY*09] and architectural scenes [CLDD09]. For example, Kraevoy et al. [KSCOS08] presented a technique for scaling complex man-made objects in a non-homogeneous way while implicitly preserving certain geometric characteristics of the original model. Observing that mechanical joints naturally define the editing degrees of freedom afforded by the geometry, Xu et al. [XWY*09] presented a joint-aware deformation framework for manipulating mechanical models with joints, which are detected through slippable motion analysis. Neither of these methods attempts to preserve the global inter-relationships among geometric features, which is crucial for preserving the shape characteristics and meaning of the manipulated object under editing. The work of Cabral et al. focuses on modeling textured architectural scenes, where angle preservation plays an important role.

3. Overview

Our system builds upon the analyze-and-edit paradigm and thus consists of two main steps: shape analysis and shape editing, as illustrated in Figure 2.

The goal of the analysis step is to construct a small set of component-wise controllers and identify their mutual relations. The challenge here is to compute the proper types of primitives that best fit the meaningful components of a given model. Since the model itself does not have prior component information, we employ a hierarchical mesh segmentation

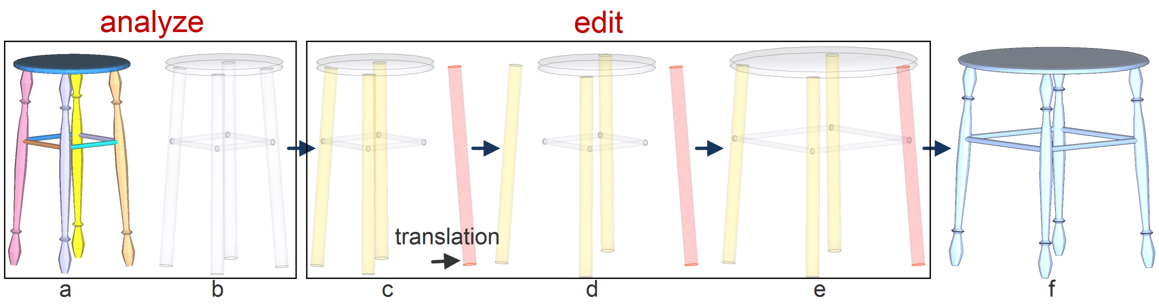

Figure 2: System overview. The given model is analyzed and automatically decomposed into meaningful components (a). Each component is equipped with a proper controller (b). During editing, the user manipulates one of the controllers (highlighted in red) and the applied change is automatically propagated to other controllers to maintain the structural relations among them, such as symmetry, coplanar and parallelism (e). (d) is an intermediate propagation result. The final model is reconstructed with respect to the modified controllers (f).

algorithm that supports simultaneous shape decomposition and primitive fitting [AFS06] (Section 4). A typical model usually leads to only dozens of components (Figure 2 (a)) and their associated controllers (Figure 2 (b)). We then identify the mutual relations between the found controllers by automatic shape analysis and possibly a small amount of user assistance for specifying alternative semantic relations.

We use four types of primitives as controllers, namely, spheres, cuboids, cylinders, and generalized cylinders (Figure 3). We focus on these four types of primitives as controllers since they are simple and still well approximate the component geometry of most man-made models. Their simplicity is necessary for intuitive user manipulation.

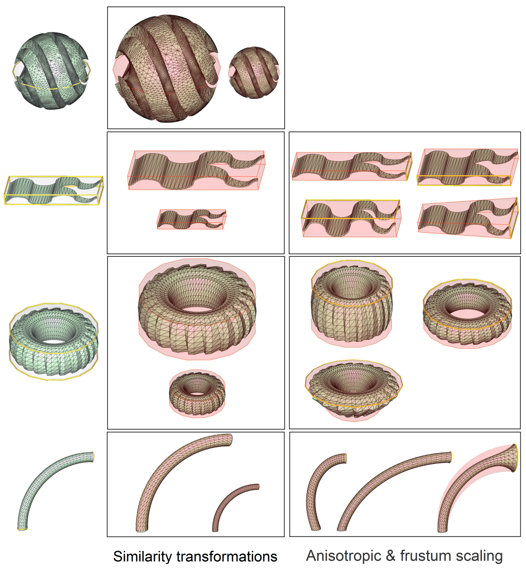

Figure 3: Left column: We use four types of primitives as controllers (i.e., sphere, cuboid, cylinder, and generalized cylinder). Each primitive is characterized by its associated feature curves (shown in yellow). Middle and Right columns: To protect the structural features of the underlying shape geometry, each type of controllers has predefined degrees of freedom for editing. In particular, the sphere primitive is the only type of primitive that does not support anisotropic scaling. Note that for the example of generalized cylinder, the controller and the original model coincide.

Since typical editing scenarios involve only a small set of component-wise controllers that reveal the editing space afforded by the associated components, our system lets the user edit a model via its controllers. This design reduces possible unexpected deformation of the engineering structures of the model since the components are prevented from deforming unnaturally. During editing, the user manipulates one of the controllers as a handle (Figure 2 (c)) and the changes are then automatically propagated from the controller being edited to the other ones (Figure 2 (d, e)). A component-wise propagation scheme (Section 5) is proposed to effectively preserve the structural information defined for the individual controllers and their interrelationships identified in the analysis step. The edited model is finally reconstructed with respect to the changed controllers (Figure 2 (f)).

The above ideas can be naturally extended to support a hierarchical organization of controllers (Section 6). The user forms a hierarchy of controllers, where the top levels of controllers are for direct manipulation and depend on the editing complexity, and the lower levels of controllers are used to protect structural features during editing and depend on the model complexity. Edits are first propagated over all the controllers at the top levels and then passed in the same fashion down to the lower levels.

4. Controller Construction and Analysis

This section presents the details of the construction of the component-wise controllers and the analysis of their interrelationships.

4.1. Controller Construction

Constructing controllers that are adapted to the component geometry demands a decomposition of a given model into meaningful components and the fitting of the components with proper types of controllers. Many man-made models consist of multiple connected components (in terms of mesh connectivity), each of which possibly comprises multiple semantically meaningful parts [MYY*10]. Therefore, our discussion below focuses on splitting a *single connected* part

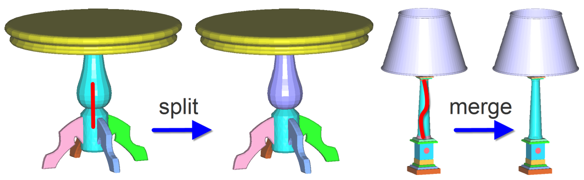

Figure 4: By drawing a stroke (i.e., red curves) over the automatically segmented results, the user can split or merge segments. Editing results of the table model are shown in Figure 17.

into multiple components and fitting each component with a proper primitive.

We use a variant of the method by Attene et al. [AFS06], which computes a hierarchical segmentation of a single-connected model by finding a best-fit primitive for each component. The hierarchy of segments is obtained by a bottom-up clustering algorithm. The key of this clustering algorithm lies in the design of the merging cost for two clusters. Besides the fitting errors of different types of primitives [AFS06], we incorporate boundary concavity and boundary length cost into the fitting error. We compute the boundary concavity as the average edge normal difference in order to encourage cuts along concave regions. The length cost is the length of the segment boundary. These two boundary measures help merge pairs of clusters with large length and concave boundaries and are found to produce fitting results that better capture the shape semantics. See Figure 4 for two decomposition results.

We observe that it is rather computationally expensive to directly include all four types of primitives for model decomposition. Hence, we adopt a two-step approach. In the first step, similar to [AFS06], we perform a model decomposition using only spheres, planes and cylinders. In the second step, we include cuboids and generalized cylinder and exclude planes. Specifically, for components fitted with planes or cylinders during decomposition, we refit them with cuboids or generalized cylinders and retain whichever best-fit primitive as the controller. To fit a generalized cylinder, we first extract a 1-D curve-skeleton of the component [ATC*08][†], then compute the skeleton-to-surface distance along the skeleton, and finally trace 2-D profile circles along the skeleton to create a generalized cylinder.

The results of the above automatic decomposition and fitting process can sometimes deviate from the user intent. Our system allows user intervention to modify the automatic segmentation results. For example, the user can modify the decomposition results using simple merging or splitting operations by sketching strokes over the extracted components (Figure 4). The splitting is implemented by separating the parent cluster into its two children in the hierarchy tree [AFS06] while the merging is achieved by finding

[†] If junction nodes or loops are detected in the extracted skeleton, we simply consider the fitting error with a generalized cylinder as infinity.

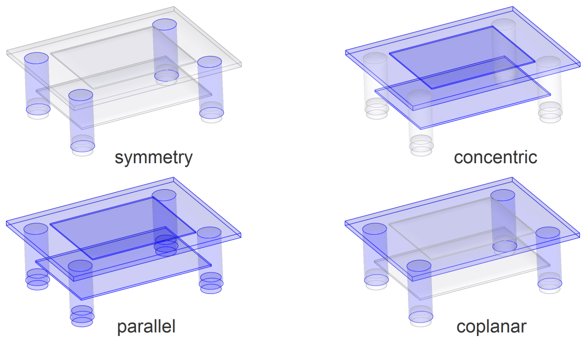

Figure 5: Examples of inter-relationships automatically established by our system. Note that the groups of controllers with common relations are not necessarily mutually exclusive. For relations like parallelism we delegate their definition to the feature curves of controllers (see the main text for more details).

the common ancestor for the clusters. Primitives are then refitted for the affected components. The merging operation is especially useful for combining spatially or topologically disconnected pieces. If necessary, the properties of individual controllers, including primitive type, orientation and position, could also be manually modified.

4.2. Establishing Mutual Relations

Defining the inter-relations among the controllers is the key to establishing a good structure-preserving editing tool. The relations of interest include symmetry, coplanarity, parallelism, concentricity, and orthogonality. Controllers sharing common relations naturally form groups, which are not necessarily mutually exclusive (Figure 5). To detect the global symmetry between each pair of controllers, we use the method of [MGP06] and take into account the geometry of both the controllers and their underlying components. Since other types of relations like coplanarity may hold only at one end of two controllers but not the other end, we delegate the definition of these relations to the feature curves of the controllers.

Feature Curves of Controllers. *Feature curves* of a controller are a set of planar circles or rectangles that define the controllers (Figure 3). Specifically, the feature curve of a sphere controller is the equator along its principle z-axis. For a cylinder or generalized cylinder, the two circles at its two ends constitute its feature curves. For a cuboid, its feature curves are the rectangles on its six sides. The feature curves uniquely characterize a primitive given their positions and orientations. Note that unlike wires in [GSMCO09], our feature curves inherently come with the controllers and do not need to be detected. Moreover, all feature curves in our system are simple planar curves that are easy to optimize during edit propagation (Section 5).

Except for the symmetry relation, in our system all other mutual relations (i.e., coplanarity, parallelism, concentricity, and orthogonality) are detected at the level of feature curves (see examples in Figure 5). However, the feature curves

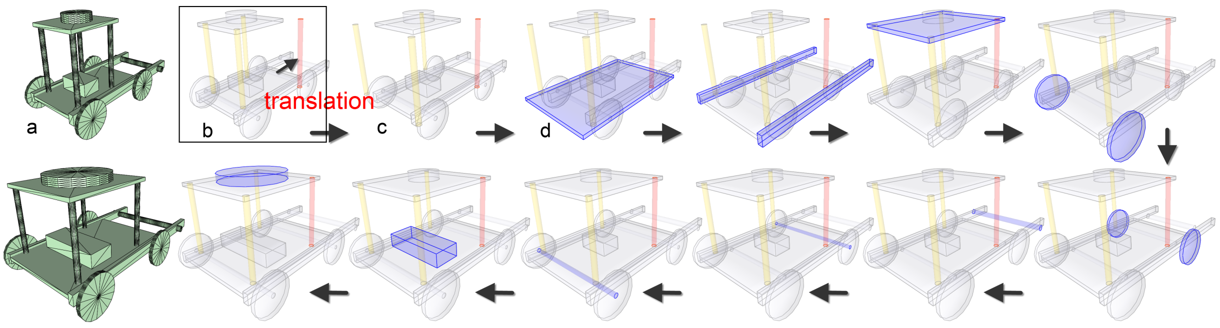

Figure 6: Component-wise propagation in action (see the accompanying video for propagation animation). The input model (a) and the controllers of the components (b). The editing effects caused by the user's translation of a pillar is first propagated to controllers which have symmetry relations with the controller being manipulated (b → c). Then the propagation goes from controller to controller by proximity (c → d). These two steps are repeated until all controllers are treated. Controllers being processed in each step are highlighted in blue.

come with the notion of components, i.e., each feature curve is associated with the controller it belongs to, which is crucial for our simpler and effective edit propagation algorithm.

We let the user manually modify the automatically found inter-relationships to achieve different editing results (Figures 11, 12, and 13). Since many man-made objects involve only a small set of component-wise controllers and inter-relationships, typically only a small amount of user interaction suffices.

5. Edit Propagation

Component-wise controllers provide intuitive guidance for interactive editing, we therefore allow the user to edit at the controller level. The user initiates editing operations by manipulating one controller (Section 5.1) or a set of controllers as a group (Section 6). The modeling constraints are then propagated to the rest of the controllers (Section 5.2). The shape characteristics of individual controllers and their mutual relations are automatically preserved during propagation to achieve high-level editing of the object. The modified controllers finally induce a deformation of the surface itself (Section 5.3).

5.1. Manipulation of Individual Controllers

To protect the underlying geometry of the model components, each type of controllers has certain predefined degrees of freedom for editing, no matter if the controllers are under direct manipulation or in the process of edit propagation. For example, it is not recommended to apply any shearing or stretching transformation to the sphere-shape controllers in order to maintain their underlying sphere-like geometry. Below we summarize all allowed local transformations for each type of controllers (Figure 3):

- ◇ **spheres:** similarity transformations (i.e., rigid transformations + isotropic scaling).
- ◇ **cuboids:** similarity transformations + frustum scaling along three main axes.
- ◇ **cylinders:** similarity transformations + frustum scaling along cylinder axes.
- ◇ **generalized cylinders:** similarity transformations + frustum scaling along main axes (e.g., the chair arms of the top-left example in Figure 17).

The user could *optionally* allow different isotropic scaling at two opposite ends of a controller, leading to a frustum scaling. For example, the two ends of a cylindrical-shape controller can be modified by different isotropic scaling transformations, forming conical frustums (Figure 3). Similarly, different similarity transformations can be specified at different circular cross sections of generalized cylinders. The specified transformations are then linearly interpolated to construct the entire generalized cylinders (Figure 3).

5.2. Component-wise Propagation

The edit propagation can essentially be viewed as an optimization problem, with the user manipulation and the inter-relations as the model constraints, seeking a new configuration of all the controllers that can fulfill both requirements. However, formulating a global optimization solution is challenging. The different natures of the inter-relations and different properties of controllers make it infeasible to design a proper objective function and weight assignments for global optimization. Hence, we opt for a component-wise propagation scheme.

Overall procedure. Our propagation scheme proceeds progressively in a *component-to-component* manner. Each controller is treated only once and will not be revisited after it is processed. Specifically, in a *breadth-first* manner, the propagation proceeds to other controllers by symmetry and proximity, as illustrated in Figure 6. Let Ω be the set of controllers that have been optimized. Initially, Ω contains the set of controllers manipulated by the user. Since symmetry relations between different components is an important shape characteristic, these relations are immediately enforced when encountered. Therefore, for each controller C in Ω, we first identify all the *untreated* controllers (i.e., those not in Ω) that are in the same symmetry group as C (e.g., controllers in yellow in Figure 6 (b)), then apply the same transformation to them as C (adjusted according to symme-

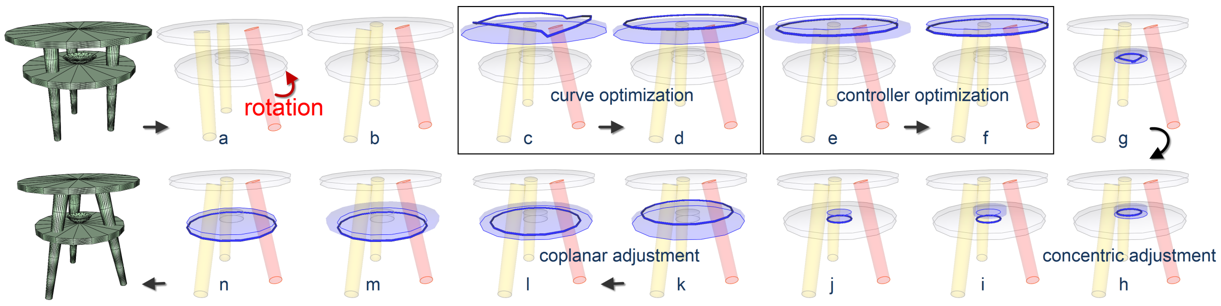

Figure 7: Various stages of the edit propagation on feature curves (see the accompanying video for propagation animation). We optimize the shape, scaling, orientation and position of feature curves of the controllers (in blue) to preserve both their individual defining characteristics (e.g., circularity) and the mutual relations with other feature curves (e.g., coplanarity). See the text for more details.

try), and finally add them to Ω. Next, we progressively select a closest controller to the ones in Ω, optimize it (by adjusting its position and orientation according to its feature curves, see details below) and then add it to Ω (Figure 6 (d)). We define the distance between a pair of controllers as the closest distance among all the feature curves of the two controllers, which is computed before editing. We repeat the above symmetry-based and proximity-based propagation steps until all the controllers are optimized.

Optimizing a new controller. Now we explain the details of how we optimize an untreated controller that is closest to the ones in Ω, denoted as U. Overall, we update the individual feature curves of U with respect to the feature curves of the controllers in Ω, denoted as Θ, and then restore the controller U. Figure 7 (c–g) illustrates the propagation process (Θ here contains the six feature curves of the cylindrical controllers of the table legs). We start with the feature curve of U closest to all the curves in Θ, denoted as c_U (e.g., the top circular feature curve of the the seat surface in Figure 7 (c)). The edit influence from Θ to c_U is found by local frame encoding [GSMCO09]. The resulting curve may violate the original characteristics of c_U (e.g., circularity). Therefore it is first optimized to retain its original characteristics (e.g., Figure 7 (c→d)). Note that all feature curves in our system are simple planar curves which are easy to optimize. Next, we adjust the position and orientation (with scaling unchanged) of c_U with respect to the feature curves in Θ that hold common mutual relations with c_U (e.g., Figure 7 (g→h) and (k→l)). The above procedure continues until all the feature curves of controller U are treated. Finally, we restore U by adjusting the position and scaling of all its feature curves (with orientation unchanged). For example, to restore a cylinder controller, we need to adjust the scaling of its two ends to be the same and align them along a common axis (Figure 7 (e→f)).

Discussions. It is possible that the user-manipulated editing constraint leads to conflict between the interrelations among controllers. In such cases, not all interrelations can be preserved after editing. In the example of Figure 7, the coplanar relation among top/bottom faces of the legs and table surfaces is violated after the legs are rotated. Our current solution to address potential conflicts is to restore mutual relations in a prescribed order: symmetry, concentricity, coplanarity, parallelism, and orthogonality (Figure 15). Besides providing users with some control over the priority of relations, a possibly better solution is to let users choose from suggestive results of enforcing relations in different priority orders.

5.3. Final Deformation

After all the controllers are treated, we update the underlying surface to reflect the changes of the controllers. Since all surface components are enclosed by our controllers, a straightforward solution might be to use the controllers as cages and apply a cage-based deformation technique (e.g., the *Green coordinates* [LLCO08]). However, it is unclear how the boundaries between adjacent surface components should be deformed as they are controlled by different cages, which might lead to noticeable artifacts. Thus, we adopt a surface-based deformation approach instead [LSLCO05], with the modeling constraints derived from the transformed feature curves.

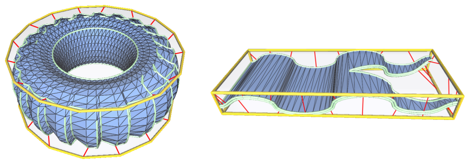

Figure 8: The final deformation is governed by the virtual edges (highlighted in red) that connect the controller with the underlying component in a feature-to-feature manner.

Our component-wise controllers have no direct connection with the underlying surface. Therefore, to transfer the edit influence from the controllers to the surface, we construct *virtual edges* between the underlying surface and the controllers. A straightforward approach is to connect sampled points of the feature curves to their nearest points on the surface. However, we found this approach sometimes causes distortion artifacts, especially around regions with sharp features. To preserve the surface features, we extract surface feature curves using the method of Ohtake et al. [OBS04] and construct virtual edges connecting points on the controller feature curves to their nearest points on

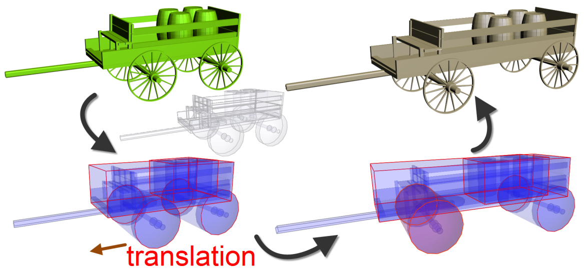

Figure 9: An editing example with two levels of controllers. Four groups of controllers (i.e., two cuboid controllers and two cylinder controllers) are formed as the top-level controllers (bottom left). Grouped controllers are highlighted with red feature curves. The user initiates edit intent by anisotropically scaling the bigger cuboid controller. The edit is propagated first to the rest of the top-level controllers, then to the original controllers (bottom right), and finally to the surface itself (top right). See the accompanying video for interactive grouping and editing results.

the surface feature curves (Figure 8). If no surface feature curve is detected in a certain range to a sampled point on a controller curve, the above-mentioned straightforward approach is used to construct virtual edges for these parts (e.g., cushions in Figure 16). The range is defined as 2 × distance from the curve point to its nearest surface point. Finally, we transform those end points of the virtual edges lying on the surface in the same way as their corresponding points on the controllers, inducing a surface deformation optimized by the method of Lipman et al. [LSLCO05]. Here the transformed end points of the virtual edges serve as boundary constraints for the surface deformation optimization. We adopt a surface-based deformation approach with virtual edges mainly as a proof of concept to demonstrate the effectiveness of our component-wise controllers in a complete editing system. We expect that a better space-based deformation approach (e.g., by improving the approach of [LLCO08]) can work well without using virtual edges.

6. Hierarchical Controller Organization

For better editing flexibility, we allow the user to group a set of controllers and edit at the group level (see interactive editing sessions in the accompanying video). By making groups of groups, we essentially obtain a hierarchical organization of controllers. The propagation scheme described in the previous section is easily adapted to support hierarchal propagation. For simplicity, below we explain the ideas under two levels of controllers: the original controllers as the *bottom level* and groups of controllers as the *top level*. The extension of these ideas to more levels is straightforward.

Although some hierarchical information is generated during automatic model decomposition (Section 4.1), it is available only within each connected part and is unreliable since it is based on low-level shape analysis only. Therefore, like many editing systems, we allow the user to interactively define and control the grouping of the controllers, which is especially crucial for grouping multiple connected components. Each set of grouped controllers is automatically fitted with a best-fit primitive as a top-level controller, using the same construction method as for the lowest-level controllers (e.g., Figures 9 and 10).

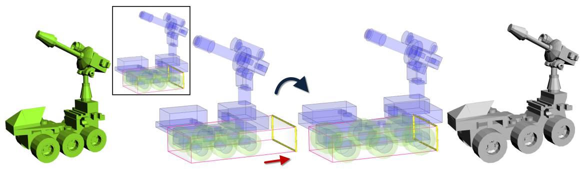

Figure 10: Another editing example with two levels of controllers. The original controllers of the 6 wheels are grouped and the resulting top-level controller is edited.

The user manipulates one of the top-level controllers to initiate an editing operation. The edit is then propagated to the underlying surface in a top-down manner: first the top-level controllers, then the bottom-level controllers and finally the surface itself. More specifically, the edit influence is first propagated from the manipulated top-level controller to the rest of the top-level controllers using the propagation scheme described in the previous section. Next, to propagate the edit from the top-level controllers to the bottom-level controllers, for each top-level controller we first find the closest bottom-level controllers inside it (according to the distance defined between feature curves) and then update their position, scale and orientation using local frame encoding and decoding [GSMCO09]. Finally, the affected bottom-level controllers trigger the edit propagation to the rest of the bottom-level controllers.

7. Results and Discussions

We have tested our techniques on a variety of man-made models, most of which exhibit many structural features and often have multiple connected components. Before editing, both the automatically constructed component-wise controllers and the automatically extracted structural relations associated with the controllers are subject to user refinement or modification (e.g., Figures 11, 12, and 13). During editing, the user can manipulate either an individual controller (Section 5) or a grouped controller (Section 6). In both cases, our propagation scheme automatically propagates the edit from the controller(s) under direct manipulation to the rest of the controllers. To control the editing effects, the user can optionally fix the position and/or orientation of certain controllers on the fly. The accompanying video shows some interactive editing sessions.

Letting the user directly manipulates the feature curves enables more flexible editing. For instance, translating one feature curve of a cylinder controller while keeping the feature curve at the other end can partially resize the cylinder along the axis. Thus, we provide such a user interface in our system. In other words, the user is able to either manipulate a controller as a whole or its individual feature curves, if desired (e.g., Figure 3). Obviously, the extra flexibility brought by the latter UI enhances the editing capacity but may lead to

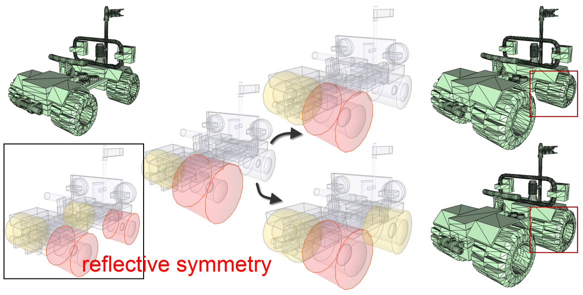

Figure 11: Changing mutual relations between controllers leads to different editing effects. The user initiates the edit by enlarging the front-left tire (top left). Top row: editing result without symmetry relation defined between front and back tires. Bottom row: editing result with manually assigned reflective symmetry relation between front and back tires.

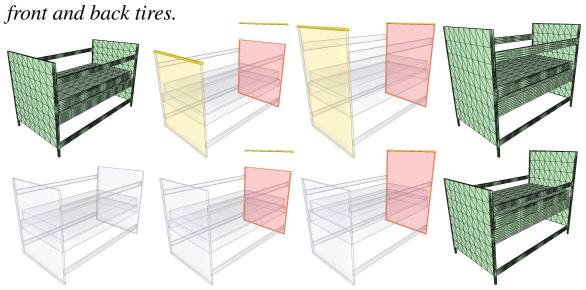

Figure 12: Editing with user-specified mutual relations. Top row: editing result with symmetry and coplanar relations defined between left and right slats. Bottom row: editing result with the relations removed.

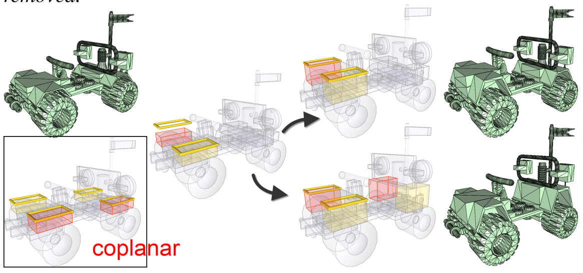

Figure 13: Editing with user-specified relations. Top row: without coplanar relations defined between the front and back covers. Bottom row: with coplanar relations manually assigned.

failure in restoring the defining characteristics of individual controllers (e.g., being frustum scaled).

Figure 17 shows a wide range of edits performed on various engineered models using our approach. We demonstrate that without reverse engineering it is possible to obtain intuitive results using our controllers with light-weight shape analysis and a small amount of user interaction. Our technique automatically adjusts the shape, size, position, and orientation of individual controllers such that the editing results satisfy the user-specified modeling constraints while preserving structural features.

Figure 14 shows a result of non-homogeneous shape resizing with our hierarchical organization of controllers. Non-homogeneous resizing lets the user control resizing effects by specifying different scaling factors for the three

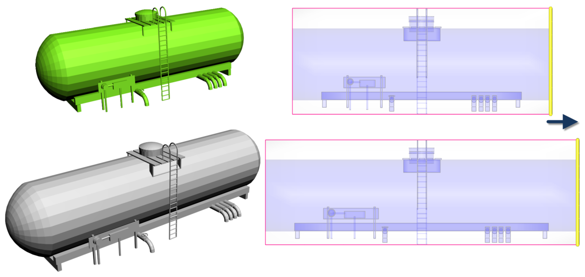

Figure 14: A non-homogeneous resizing example. To resize the entire model, we form a top-level controller which encloses all the original controllers. Such two-level representation of controllers successfully preserves the underlying surface structures (e.g., circularity) during non-homogeneous resizing.

axes [KSCOS08]. Given this low-dimensional editing space, it is unnecessary to let the user manipulate individual controllers, which exposes excessive editing degrees of freedom to users. A single top-level controller which encloses the entire model sufficiently provides the desired user control. Therefore we claim that to some extent a hierarchical organization of controllers can enhance the user's editing flexibility while minimizing the user effort towards desired editing effects.

Timings. Our technique essentially consists of two steps: shape analysis and shape editing. The bottleneck of the analysis step lies in controller construction (Section 4.1), whose time complexity depends on both the number of connected components and the granularity of the individual components. For all the models shown in the paper, it typically took less than one minute to finish all the processing described in Section 4, including the time for user intervention. In fact, for many models in our experiments, the automatic shape analysis process suffices to properly detect all the inter-relations among controllers because man-made models are usually regularly shaped and have rather distinguishable components. For some complex models like the robot ship in Figure 1, only a small amount of interaction is required to manually adjust relations such as co-planarity.

After the model is analyzed in the pre-processing step, the user is able to edit its shape through component-wise controllers at *interactive rate*. The user simply performs mouse drag-and-drop operations and the system provides immediate feedback (see the accompanying video). The user can control the editing effects *on the fly* using operations like selecting different controllers for direct manipulation, fixing different controllers, forming different controller hierarchies, etc. To achieve fast runtime performance for the final deformation of the underlying surface (Section 5.3), we perform system pre-factorization in the initialization step and do fast back-substitution at runtime [LSLCO05]. The editing sessions for all the examples in the paper took no more than two minutes.

Comparison with iWires. Our approach differs from the iWires method in several aspects. First, our technique is built

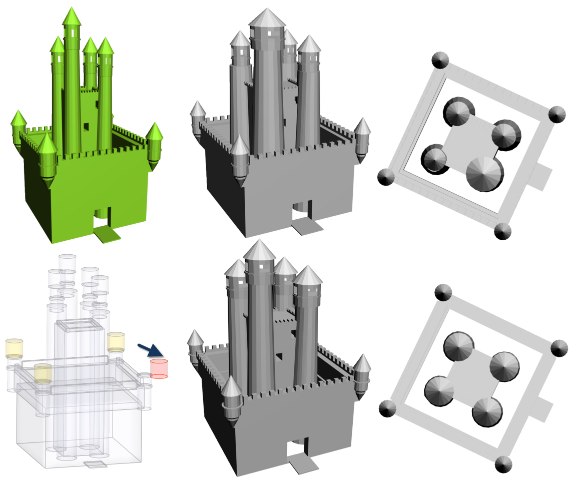

Figure 15: *Our component-wise propagation scheme (bottom row) alleviates the conflicting problem among different sets of inter-relations (symmetry vs. concentric in this example), which leads to different sizes of the four pillars (top middle & top right) by the wire-level propagation scheme of Gal et al. [GSMCO09].*

upon the concept of component-wise editing. This presents to the user only a small set of editing controllers, which come with the inherently afforded editing degrees of freedom and thus offer intuitive editing guidance (see interactive editing sessions in the accompanying video). In contrast, as iWires uses wires as controllers, which are more low-level, it is more powerful for shape editing at the level of feature curves. Therefore bringing both component-wise and wire-like controllers into an integrated system would be an interesting topic to explore in the future. Second, the component-wise controllers enable the design of a simple yet effective propagation paradigm. As mentioned in Section 5.2, iWires does not consider the object components in the design of edit propagation and thus may fail to preserve component-wise features when optimizing individual wires of a single component to maintain different inter-relations. In contrast, our approach leverages the concept of component to govern the propagation in a natural manner, leading to better proximity definition, thus alleviates this problem (Figure 15). Third, our system integrates the user control within the analysis procedure, allowing more control over the editing results. The component organization enables all the user interventions to be performed at a convenient level. Lastly, the component-wise controllers makes our technique insensitive to the local shape features. In contrast, noisy models and models with few structural features render automatic wire extraction challenging for iWires (Figure 16).

Limitations. Since our controller construction is based on model decomposition, it can fail to identify a component that crosses over multiple connected pieces. Redefining the merging cost for two clusters by considering disconnectedness might alleviate this problem. In addition, our controllers are by default defined to enclose the underlying

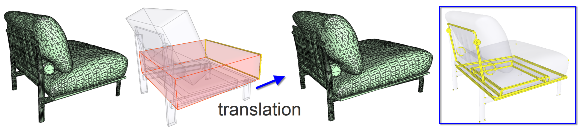

Figure 16: *Our approach does not rely on sharp features of the original geometry as controllers. This enables handling of models with few sharp features (e.g., seat and back cushions), whereas iWires cannot achieve such editing result since no wires can be extracted on these parts (right).*

components. Hence, it may fail to detect and preserve hollow features, e.g., cylindrical holes in the phone dial model in [GSMCO09]. In such cases, user assistance may be required to manually place proper controllers there. Lastly, our technique does not penalize self-intersection between components caused by editing, which is an interesting topic to explore in the future.

8. Summary

We have introduced component-wise controllers for structure-preserving editing of man-made models. Controllers are constructed to adapt to the component geometry and thus naturally reflect their editing degrees of freedom, providing intuitive user interface for editing. We design a simple but effective component-wise propagation algorithm to preserve both the characteristics of individual controllers and their mutual relations while satisfying user-specified modeling constraints. Our framework supports hierarchical organization of controllers, allowing the user to easily organize the controllers to match edit intent. The technique has already been demonstrated on a large number of man-made models full of structural features, which are challenging to edit with traditional local surface or volumetric deformation techniques.

Acknowledgement

We would like to thank the anonymous reviewers, Niloy Mitra and Kai Xu for their constructive comments and Michael Brown for the video narration. Hongbo Fu and Chiew-Lan Tai are partly supported by the Hong Kong Research Grant Council (Project No. CityU113711, GRF9041562 and GRF619908). Daniel Cohen-Or was supported in part by grants from the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities.

References

[AFS06] Attene M., Falcidieno B., Spagnuolo M.: Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput. 22*, 3 (2006), 181–193. 3, 4

[ATC*08] Au O. K.-C., Tai C.-L., Chu H.-K., Cohen-Or D., Lee T.-Y.: Skeleton extraction by mesh contraction. *ACM Transactions on Graphics 27*, 3 (2008), 44. 4

[BCWG09] Ben-Chen M., Weber O., Gotsman C.: Variational harmonic maps for space deformation. *ACM Transactions on Graphics 28* (2009), 34. 2

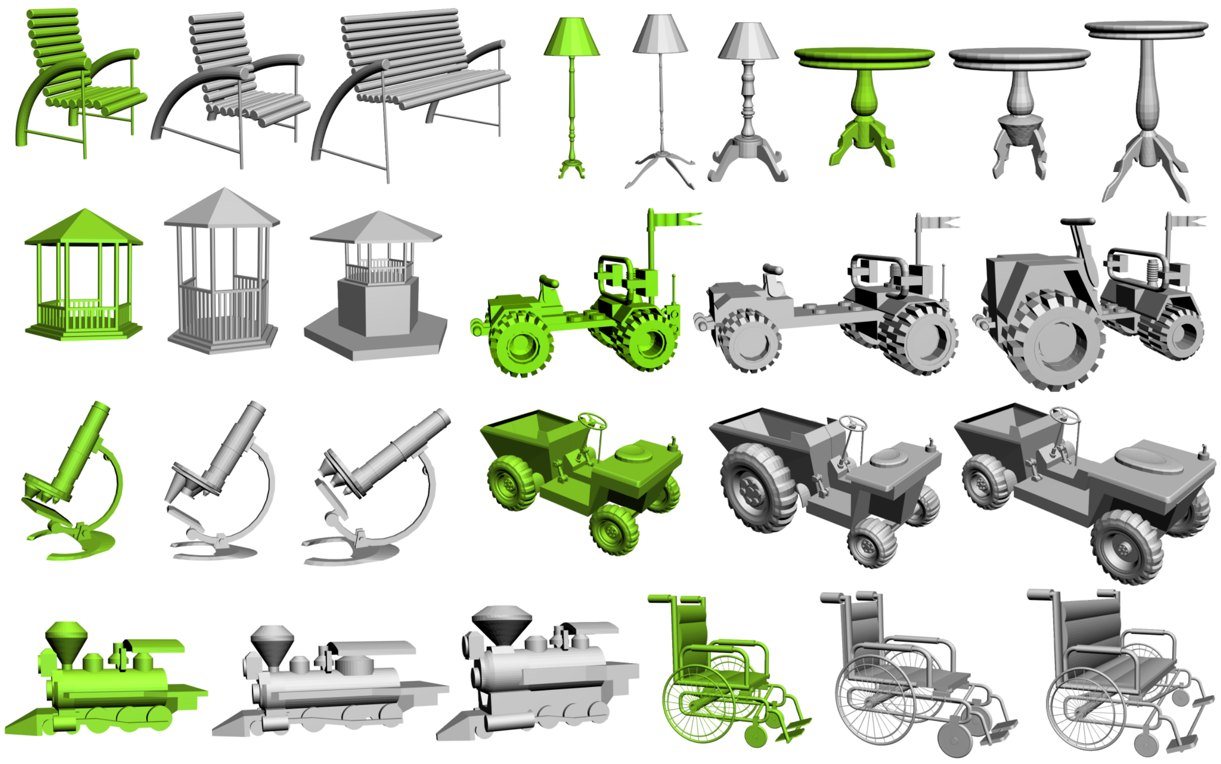

Figure 17: A variety of editing results with our component-wise controllers. The original models are shown in green, and the edited results in gray. Note that the deformation introduced to the modified models is complex, spatially varying but still respects the underlying structure, which is challenging to achieve with existing techniques.

[BPGK06] Botsch M., Pauly M., Gross M., Kobbelt L.: PriMo: coupled prisms for intuitive surface modeling. In *Symposium on Geometry Processing* (2006), pp. 11–20. 2

[BPWG07] Botsch M., Pauly M., Wicke M., Gross M.: Adaptive space deformations based on rigid cells. *Computer Graphics Forum (Proc. Eurographics 2007) 26*, 3 (2007), To appear. 2

[BS08] Botsch M., Sorkine O.: On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (2008), 213–230. 2

[CLDD09] Cabral M., Lefebvre S., Dachsbacher C., Drettakis G.: Structure preserving reshape for textured architectural scenes. *Computer Graphics Forum 28*, 2 (2009), 469–480. 1, 2

[GSMCO09] Gal R., Sorkine O., Mitra N., Cohen-Or D.: iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics 28*, 3 (2009). 1, 2, 4, 6, 7, 9

[HSL*06] Huang J., Shi X., Liu X., Zhou K., Wei L.-Y., Teng S.-H., Bao H., Guo B., Shum H.-Y.: Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3 (2006), 1126–1134. 2

[KSCOS08] Kraevoy V., Sheffer A., Cohen-Or D., Shamir A.: Non-homogeneous resizing of complex models. *ACM Transactions on Graphics 27*, 5 (2008). 1, 2, 8

[LLCO08] Lipman Y., Levin D., Cohen-Or D.: Green coordinates. *ACM Transactions on Graphics 27*, 3 (2008), 78. 2, 6, 7

[LSLCO05] Lipman Y., Sorkine O., Levin D., Cohen-Or D.: Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24* (2005), 479–487. 6, 7, 8

[MGP06] Mitra N. J., Guibas L. J., Pauly M.: Partial and approximate symmetry detection for 3D geometry. *ACM Trans. Graph. 25*, 3 (2006), 560–568. 4

[MYY*10] Mitra N. J., Yang Y.-L., Yan D.-M., Li W., Agrawala M.: Illustrating how mechanical assemblies work. *ACM Trans. Graph. 29*, 4 (2010), 1–12. 3

[OBS04] Ohtake Y., Belyaev A., Seidel H.-P.: Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3 (2004), 609–612. 6

[PJS06] Popa T., Julius D., Sheffer A.: Material-aware mesh deformations. In *SMI '06* (2006). 2

[SLCO*04] Sorkine O., Lipman Y., Cohen-Or D., Alexa M., Rössl C., Seidel H.-P.: Laplacian surface editing. In *Symposium on Geometry Processing* (2004), pp. 179–188. 2

[SSP07] Sumner R., Schmid J., Pauly M.: Embedded deformation for shape manipulation. *ACM Transactions on Graphics 26*, 3 (2007), 80. 2

[XWY*09] Xu W., Wang J., Yin K., Zhou K., van de Panne M., Chen F., Guo B.: Joint-aware manipulation of deformable models. *ACM Transactions on Graphics 28*, 3 (2009), 35. 1, 2

[YZX*04] Yu Y., Zhou K., Xu D., Shi X., Bao H., Guo B., Shum H.-Y.: Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651. 2

[ZXTD10] Zhou K., Xu W., Tong Y., Desbrun M.: Deformation transfer to multi-component objects. *Computer Graphics Forum 29*, 2 (2010). 1