Volume 32 (2013), Number 2 pp. 1–13 COMPUTER GRAPHICS *forum*

Mobility-Trees for Indoor Scenes Manipulation

Andrei Sharf[⋆∗∗] and Hui Huang[∗†] and Cheng Liang[∗] and Jiapei Zhang[∗] and Baoquan Chen[∗§†] and Minglun Gong[∗‡]

[⋆] Ben-Gurion University
[∗] Shenzhen VisuCA Key Lab / SIAT
[§] Shandong University
[‡] Memorial University of Newfoundland

Figure 1: *In a complex indoor scene (left), our method detects functional mobilities of furniture object and parts (zoom-ins) allowing easy manipulation and reorganization (right).*

Abstract

In this work we introduce the mobility-tree *construct for high-level functional representation of complex 3D indoor scenes. In recent years, digital indoor scenes are becoming increasingly popular, consisting of detailed geometry and complex functionalities. These scenes often consist of objects that reoccur in various poses and interrelate with each other. In this work we analyze the reoccurrence of objects in the scene and automatically detect their functional mobilities.* Mobility *analysis denotes the motion capabilities (i.e. degree of freedom) of an object and its subpart which typically relates to their indoor functionalities. We compute an object's* mobility *by analyzing its spatial arrangement, repetitions and relations with other objects and store it in a* mobility-tree*. Repetitive motions in the scenes are grouped in* mobility-groups*, for which we develop a set of sophisticated controllers facilitating semantical high-level editing operations. We show applications of our mobility analysis to interactive scene manipulation and reorganization, and present results for a variety of indoor scenes.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations

1. Introduction

Indoor scenes play an important role in living environments which, together with buildings, outdoor furniture and flo-

[†] Corresponding authors: Hui Huang (hhzhiyan@gmail.com), Baoquan Chen (baoquan.chen@gmail.com)

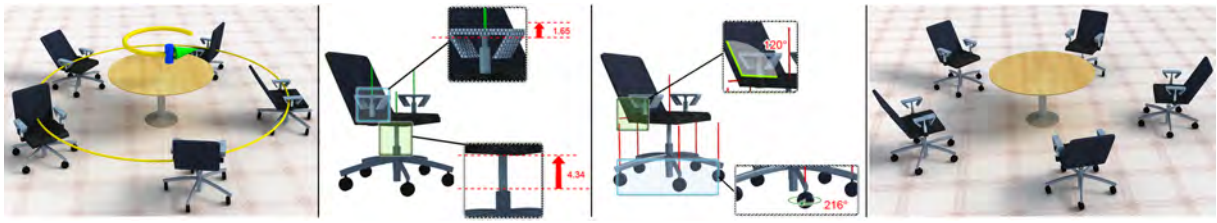

Figure 2: *Extracting mobilities from repeating chairs near a table. We detect rotational mobility of chairs around the table (left), translational mobility of armrests and chair legs (mid-left), rotational mobility of wheels and chair backs (mid-right). The regularized scene using our mobility controllers is on the right.*

ra, constitute the urban landscape. Flourishing fields such as video games, urban planing and digital cities, largely benefit from the availability of indoor scene models. Fortunately, large 3D indoor scene corpuses such as Google (Trimble) 3D Warehouse and World of Warcraft are becoming publicly available. With their popularity, virtual indoor scenes are becoming increasingly complex, in terms of their geometric detail and object functionalities.

In recent years we observe a constant increase in the number of digital indoor scenes that are generated. This poses an immediate need for efficient modeling and manipulation tools. While interior designers follow numerous high-level guidelines in producing indoor layouts [Lyo08], amateurs rely on more intuitive rules such as alignment with prominent features and positioning against walls. In both cases interior design is challenging since it requires jointly optimizing a multitude of functional and visual criteria.

Indoor scenes are typically composed of a large number of objects. The multitude of objects define a large and complex variety of object interrelations which derive from their functionalities. For example, chairs are located *near* tables, lamps are *on top of* desks, books are *organized vertically* on shelves etc. Associated with the object functionalities are often a set of predominant motions that can be performed by the object or its subparts. We denote the motions prescribed by an object's functionality as its *mobility*, which defines the specific degrees of freedom, types of motions, axes and limits. For example, furniture elements (such as doors revolving around doorposts, chair backs reclining, etc.) consist of movable parts to serve their functionalities. A key point in manipulation of large scale digital indoor scenes is the analysis and modeling of the rich functional mobility information. An immediate application of it is scene editing and reorganization (Figure 1).

The prevalence of digital 3D indoor scenes has been drawing much attention in recent years, presenting methods for automatic scene generation [MSK10, MSL*11, YYT*11, FRS*12], and reconstruction through classification [NXS12, KMYG12]. Mobility analysis of 3D objects has been previously studied [GG04, XWY*09, ZFCO*11, MYY*10, WXL*11]. In these works, mobility captures the natural degrees of freedom of individual objects and their subparts. In our work, we detect the functional mobility of objects and their subparts from their interrelations and reoccurrences in the scene. We develop a hierarchical construct denoted *mobility-tree*, which allows easy manipulation of the objects in the scene compatible with their functionalities.

Our key idea is to detect the functional mobility of parts and subparts by analyzing their poses and interrelations in a static 3D scene (Figure 2). We achieve this by segmenting the scene into a tree hierarchy based on *support* interrelations in the scene. For example, an indoor scene, is first segmented into supporting structures (floor, walls, ceiling) and supported objects (furniture, pictures, lights). Segmentation by support relations repeats recursively until reaching furniture parts that are unbreakable.

We use the tree hierarchy to detect the functional mobilities of objects and their parts. We analyze the support relations in the tree to infer certain degrees of freedom for each supported node with respect to its supporting node (e.g. apples translating on a table). We denote this as *weak mobility* constrains as it allows free movement within the scope of the supporting surface. We proceed by searching for repeated instances of the same objects and parts in the scene. We analyze pose differences of repeated instances and infer motion axes and limits. We denote this as *hard mobility* constraints since only translation and rotation motions along certain axes are permitted (e.g. drawers translating in or out of a cabinet). Finally, we derive interactive controllers for sophisticated high-level editing and manipulation of indoor scenes that conforms to objects' functionalities.

Our main contribution is an algorithm for detecting motion behaviors of objects in a general static indoor scene. This is achieved by a novel hierarchical construct which segments the scene into meaningful parts and encodes their support interrelations. We show a novel algorithm for high-level mobility detection by analyzing support interrelations as well as pose variation of reoccurring instances. The resulting

mobility-tree allows users to easily manipulate objects in a semantically meaningful way. As a result, we turn the static scene into a dynamic and easily tractable one.

2. Related work

Our work combines ideas from two research fields: functional mobility analysis and 3D indoor scene processing. We therefore divide our related work discussion into these two domains as below.

Functional Mobility Analysis The mobility of individual objects and their parts has been widely explored in the past in context of shape aware deformation and animation. These methods focus on motion detection of individual objects using geometrical considerations and different heuristics.

Kraevoy et al. [KSSCO08] presented a non-homogeneous resizing technique for man-made objects. Xu et al. [XWY*09] presented a joint-aware shape deformation technique using slippage analysis to detect joint-constraints. Cabral et al. [CLDD09] allow the user to interactively modify lengths of edges, while constraining angles for modeling textured architectural scenes. Wang et al. [WXL*11] built a graph which encodes inter-part symmetry and self-symmetries of objects for intuitive shape editing.

Similar to us, Mitra et al. [MYY*10] computed the functional mobilities of mechanical objects from a static model. Their method focuses on the visualization of inner motions of individual CAD objects. In contrast, our problem domain is indoor scenes, analyzing functional mobility from multiple objects, their repetitions and interactions.

Recently, Zheng et al. [ZFCO*11] presented a component-aware shape manipulation technique using controllers that capture the natural degrees of freedom. Motion information is automatically propagated to other components during manipulation, to preserve their interrelations. In contrast to these methods, we focus on large scenes and automatically compute model functional mobilities of parts and subparts using a hierarchical construct.

Indoor Scene Processing The problem of efficiently manipulating the vast amount of information typical in indoor scenes is a challenging task.

An early work [BS95] defined object associations to determine valid and desirable transformations for objects manipulation in the scene. Kjolass [Kjo00] presented a system for automatic placement of furniture into a given floor plan while recursively resolving spatial conflicts. To reduce efforts of 3D scene generation, automatic rule-based methods have been suggested [XSF02, AON05, GS09]. These methods generally define an energy functional representing the goodness of a layout, and minimize it using optimization techniques. Instead of looking for general constraints and global rules for scene generation, our work detects objects mobilities for scene manipulation.

More recently, Merrell et al. [MSK10] presented a method for automated generation of building layouts by synthesizing architecture using a Bayesian network. An interactive furniture layout system was presented in [MSL*11] which suggests furniture arrangements from a precomputed density function of interior layouts. Fisher et al. [FSH11] showed an efficient graph representation of semantic relationships for measuring similarity of scenes. Yu et al. [YYT*11] analyze hierarchical and spatial relationships of furniture objects to synthesize new arrangements. Fisher et al. [FRS*12] have recently presented a method for synthesizing scene arrangements from examples using an occurrence model. Their model groups objects according to their local neighborhood, thus capturing the continuous spatial relationships. Common to us, these works aim at learning indoor scene characteristics from arrangements and interrelations of scene objects. Nevertheless, their focus is on layout synthesis, while we focus on detecting objects functional mobility from the static indoor scene arrangement.

3. Overview

Our input consists of large scale 3D indoor scenes, which are typically man-made, as in Google 3D Warehouse. Our algorithm initially over-segments the scene into homogeneous patches denoted *superpatches*. In an indoor scene, objects and their parts typically function as *supporting* and *supported*. We compute a hierarchical segmentation of the scene into object parts and subparts based on their support relations. We formulate support relations as an energy function and compute the scene segmentation using graph-cut. Given a *supporting* part, we cluster together *supported* superpatches which are directly connected in the mesh. The *support tree* hierarchy is computed by recursively applying the segment–cluster process, starting from the whole scene as the root node down to parts and subparts. The result is a scene hierarchy, referred to as a *support tree*, which encodes the *supporting–supported* relations in the scene (Figure 3 (mid-right)).

We compute the mobility by analyzing support relations and reoccurrences of objects in the scene. Mobility defines the degrees of freedom of an object in the scene. We label each node in the tree as either no mobility, weak mobility, or hard mobility. Weak mobility defines a general motion of objects restricted by their supporting contact surface. Hard mobility defines restricted motions along translational or rotational axes (apples and chairs respectively in Figure 1).

Based on detected mobilities we define a set of high-level controllers, which allow sophisticated scene manipulation through a simple easy-to-use UI. In the editing step, the user can select mobility controllers and modify the scene by redistributing, aligning, reorganizing, positioning and de-

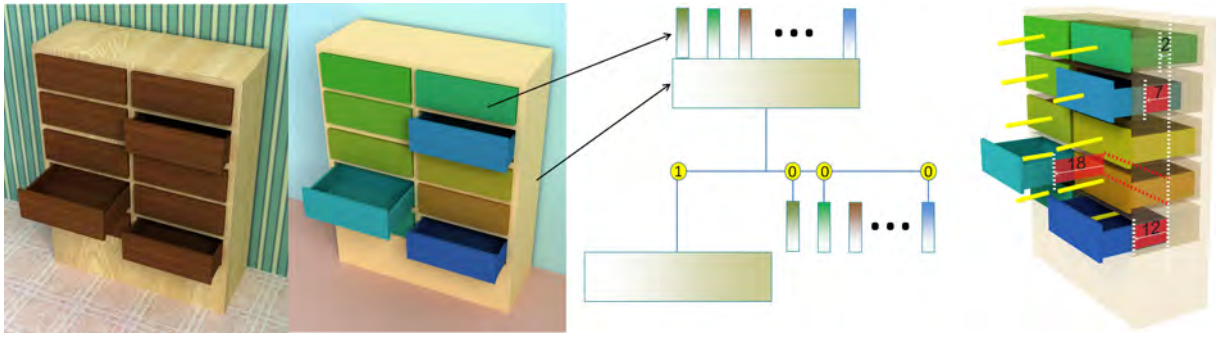

Figure 3: *Algorithm overview on a 3D cabinet scene. Left-to-right: input scene, segmentation into drawers and body, support tree consisting of one supporting node belonging to cabinet body and supported drawers, detected mobilities defined by translational axes and limits.*

forming multiple objects or subparts simultaneously (Figure 2(right)).

4. Technical details

In the following, we discuss in detail our support-based segmentation, support-tree hierarchy, mobility detection and scene manipulation.

4.1. Support-aware Segmentation

The input to our algorithm is a large scale indoor scene represented by unorganized boundary meshes. Our algorithm does not assume any mesh properties, thus, the input mesh may contain self intersections, non-manifoldness, topological inconsistencies etc. Inspired by the utilization of superpixels for image segmentation [RM03], we initially oversegment the input scene into a large number of homogeneous *superpatches*. Starting from independent triangles, we group adjacent triangles with similar orientation and shading (i.e. color and material if available) within a ε-threshold into a superpatch. We treat each superpatch as an unbreakable unit in our segmentation.

We observe that objects in indoor scenes are commonly characterized by *supporting* and *supported* interrelations (e.g. table supporting lamp and supported by floor, pictures and lights supported by wall and ceiling respectively). Our key idea is to utilize the prevalent *supporting-supported* relationships in the scene for meaningful object segmentation.

For each superpatch we compute its *support energy* with respect to neighboring superpatches and apply graph-cut [KZ04] to label superpatches into either *supporting* or *supported*, while minimizing an energy functional. We write superpatch labeling as a minimization problem of the following form:

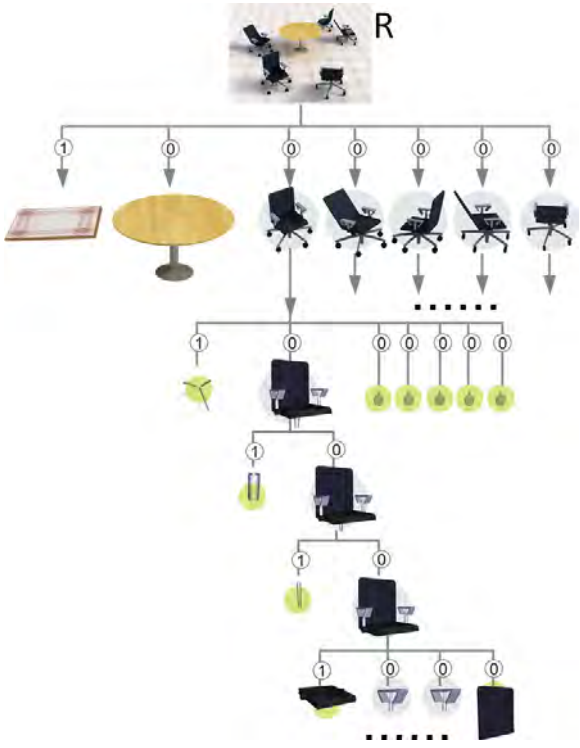

Figure 4: *Support-tree decomposition of a 3D scene.*

$$E = \sum_{p \in S} U_p(l_p) + \lambda \sum_{p,q \in S} V_{p,q}(l_p, l_q),$$

where $U_p(l_p)$ is the data cost for assigning label l_p to superpatch p, and $V_{p,q}(l_p, l_q)$ is the smoothness term for balancing labels l_p at p and l_q at q. $l_p = 1$ denotes that p has been

assigned a *supporting* role, and $l_p = 0$ indicates that p is *supported*. The parameter λ controls the relative importance between the data cost and smoothness terms.

Typically, a supporting superpatch p has the properties:

- is *closer* to its supported parts. If p is supporting q and both are directly/indirectly supported by another part r, then p is closer to q than r is. For example, in Figure 4, the seat is closer to the armrests (supported by seat) than the legs are (supporting both seat and armrests).
- is *larger* in size than the superpatches that it supports.
- supports *more than one* object, hence in contact with multiple superpatches.

Based on the above observations, we define the data cost term as:

$$\begin{aligned} U_p(1) &= 0.4D(p) + 0.3\frac{1}{A(p)} + 0.3\frac{1}{N(p)}, \\ U_p(0) &= 1 - U_p(1), \end{aligned}$$

where $D(p)$ is the *proximity* term and measures the shortest distance from p to a (previously labeled) supporting superpatch; $A(p)$ is the *size* term and measures the surface area of p's oriented bounding box (OBB); $N(p)$ is the *cardinality* term and counts the number of superpatches that are directly connected to p, by searching for superpatches that overlap with p's OBB within a threshold.

The smoothness term is used to encourage superpatches to be assigned with the same label if they are either geometrically similar or spatially proximate. It is defined as:

$$\begin{aligned} V_{p,q}(0,1) &= V_{p,q}(1,0) \\ &= 0.4S(p,q) + 0.3G(p,q) + 0.3/C(p,q), \\ V_{p,q}(0,0) &= V_{p,q}(1,1) = 1 - V_{p,q}(0,1), \end{aligned}$$

where $S(p,q)$ is the *similarity* term and measures the difference between two parts in terms of their material and size. Material similarity is computed by the difference between their texture images. Size similarity is computed by length ratios of the three PCA axes, where axes correspondence is achieved by their size ordering. $G(p,q)$ is the *alignment* term and measures how well two superpatches are aligned. We measure it by checking the overlapping percentage of the two parts' OBBs (e.g. a closed drawer aligned with its supporting cabinet frame). $C(p,q)$ is the *connectivity* term and measures the shortest connecting path between p and q, in terms of the number of superpatches in the shortest geodesic path. When there is no path between p and q, we set $C(p,q) = \infty$. All six terms are normalized and we have experimentally set the parameter $\lambda = 4$.

Once superpatches are labeled, it is common that multiple parts are supported by he same *supporting* part. For example, in Figure 4, the chair seat, armrests and back are all supported by the legs. We group *supported* superpatches into clusters based on their connectivity. That is, two superpatches p and q are in the same cluster iff $C(p,q) \neq \infty$ and their connecting path passes only through *supported* superpatches. The outcome is a segmentation where each supporting object supports one or more supported objects. In Figure 4 chairs are separated by the supporting floor thus denoted by independent nodes in the tree. However, connected chair parts are initially clustered as one supported part denoted by a single node. In deeper levels in the tree, we recursively segmented this cluster.

We enhance the segmentation process with user assistance, to allow refinement of labels in scenes that do not fully comply with the above supporting properties. We define two types of manual scribbles for assistance: graph-cut *relabeling* and cluster *breaking*. The *relabeling* scribble inverts the label of a superpatch as assigned by graph-cut, whereas the *breaking* scribble splits supported superpatches into separate clusters.

Algorithm 1 Build the tree hierarchy for input node S

```
Label superpatches in S as supporting and supported;
Cluster together connected supported superpatches;
Create a node B to hold all supporting superpatches;
Create a node D_i to hold i^th cluster of supported superpatches;
Add B and {D_i} as the children of S; see Fig. 6(b);
if S is a supporting node then
    Find S's parent node A;
    Remove S from A's children list;
    Add B and {D_i} into A's children list;
    for all A's children {T_j} that S supports do
        if T_j is solely supported by one of the subset D_k then
            Create a node C that holds both D_k and T_j;
            Remove D_k and T_j from A's children list;
            Add C to A's children list;
            Add D_k and T_j as the children of C; see Fig. 6(d);
        end if
    end for
end if
Apply Algorithm 1 recursively to build subtrees for B and {D_i}.
```

4.2. Support Tree Hierarchy

The procedure discussed in Section 4.1 segments the scene S into semantically meaningful objects. In this step, we compute the support tree hierarchy by applying the segment-cluster process in a recursive manner (Figure 4).

Starting with the whole scene, we create a root node R that holds all the superpatches in the scene (Figure 4). We manually detect typical dominant supporting parts in the indoor scene as floors, walls and ceilings. Remaining superpatches are inserted into the support tree as follows: we examine superpatches clusters in R (Section 4.1) and at each level, create one tree node for the supporting part and one or more tree nodes for each supported cluster, and add them as children of R. Next, we recursively apply segment-cluster

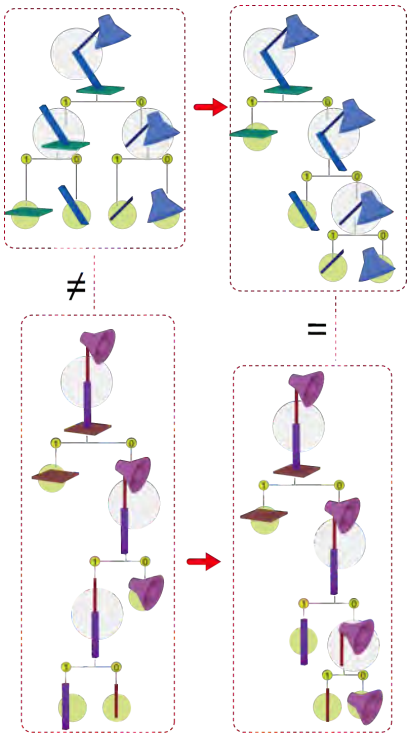

Figure 5: *Comparison between normal (a) and standardized (b) support trees for an articulated lamp.*

on each of the child nodes, further segmenting objects into parts and subparts. We repeat node creation and tree insertion for supporting and supported subparts within each child. Support tree construction proceeds until unbreakable superpatches are reached.

By construction, a support tree has the following properties which need be maintained:

1. A node in the tree has two or more children, with exactly one child being the supporting node and other siblings being supported nodes;
2. A supporting node E supports all its siblings and hence, if E moves in the scene, all superpatches in its siblings move with it;
3. If a superpatch p is contained in node E, it is also contained in all the ancestor nodes of E. The supporting/supported role of p in these nodes may be different. For example, a table resides in a supporting node and supports books on the table, while at its parent level, the table and the books reside in the same supported node which is supported by the floor.

While the above recursive approach is simple and nicely encodes support interrelations in the tree, it cannot guarantee identical tree structures for two instances of the same object.

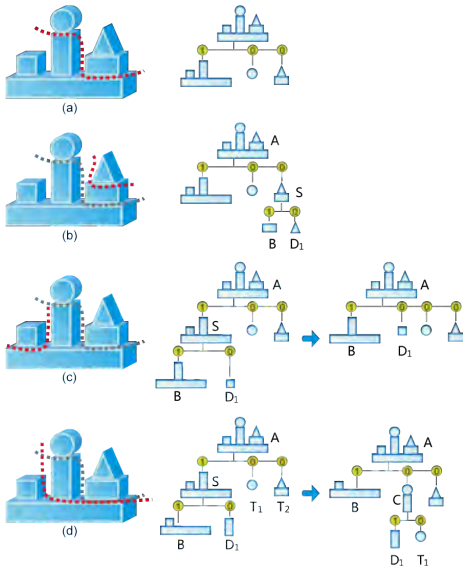

Figure 6: *Rearrangement scenarios of support tree: a) initial cut segments scene into a supporting and two supported objects; b) no rearrangement needed after segmentation of a supported object (red cut); c) and d) rearrangement when the supporting object is segmented.*

This is because the *support tree* construct depends on segmentation order (Figure 5(a)). To achieve a consistent hierarchy, we introduce the notion of *standardized support tree*. The additional constraint of a standardized version is that all supporting nodes must be leaf nodes (Figure 5(b)). To generate a standardized support tree, we use additional rearrangement steps to move supporting nodes into the leaves (See Algorithm 1 and Figure 6). Thus, two articulated objects with identical parts will yield identical trees.

4.3. Mobility Detection

In this step, we automatically extract objects mobilities from the standardized support tree. We consider two types of mobilities: *weak mobility* and *hard mobility*, the former allows an object to translate and rotate as long as it remains in contact with its supporting surface (e.g. chairs moving around on the floor), whereas the later allows an object to translate or rotate along specific axes and within some limits (e.g. drawers translating in and out of cabinets, doors rotating around their doorposts).

To automatically detect *hard mobility*, we utilize the fact that objects and parts reoccur in the scene with different poses and orientations. Based on this observation, we search for repetitive objects and parts in the scene. Two objects p_i and p_j are considered repetitive instances if there is a perfect matching between their standardized support trees and between their mesh parts at leaf level. For example,the two

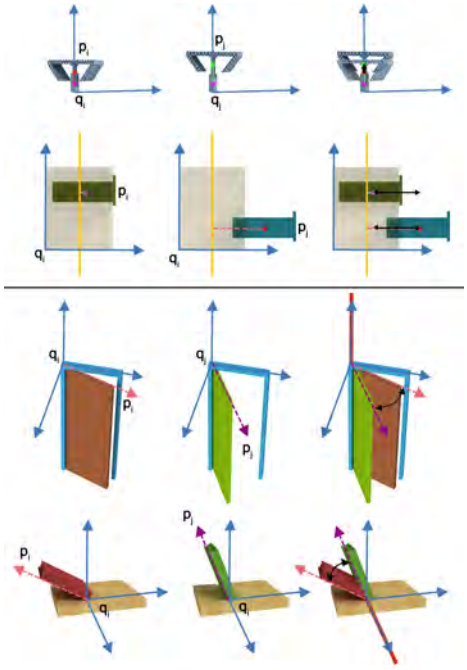

Figure 7: *Mobility extraction, top: translational (armrest, drawer), bottom: rotational (door, book). Support parts may share the same support q_i (drawers,books), or different supporting parts q_i, q_j (armrests, doors).*

lamps in Figure 5 has the same (standardized) tree hierarchies and consist of four mesh parts. For computational efficiency, we first compare tree hierarchies, and only if identical we compare shape similarity at leaf level. Parts' similarity is measured using $S(p,q)$ at leaf objects. In our example, lamp parts match exactly, thus lamps are considered repetitive instances with different poses.

We restrict our mobility detection to piece-wise rigid transformations. This does not pose a limitation since indoor scenes objects are commonly man-made and composed of rigid body functional parts. A translation transformation is defined by translation direction, origin and min/max translational offsets. Similarly, a rotation transformation (represented by quaternions) is defined by rotation axis, reference direction, and min/max rotation angles (Figure 2).

We detect the mobility of repetitive parts, say $\{p_i,\ldots,p_j\}$ w.r.t. their supporting part $\{q_i,\ldots,q_j\}$. Note that repeating parts may have the same or different supports. For example in Figure 7, drawers are supported by one cabinet body while doors are supported by different door frames. If the supporting nodes are different, i.e. $q_i \neq q_j$, we align all PCA axes of $\{q_i,\ldots,q_j\}$ with q_i's PCA axes denoted $(\mathbf{x}_{q_i},\mathbf{y}_{q_i},\mathbf{z}_{q_i})$. Axes correspondence is achieved by ordering axes by size.

To detect translational mobility we map all centroids $\{c_{p_i},\ldots,c_{p_j}\}$ to the common q_i's axes. We check w.r.t. to each axis, e.g. $\mathbf{x}_{q_i}$ if the projected centroids $\{c_{p_i}\odot\mathbf{x}_{q_i},\ldots,c_{p_j}\odot\mathbf{x}_{q_i}\}$ form a line ($\odot$ denotes dot product). Then, we define this as a translational mobility moving along $\mathbf{x}_{q_i}$. The min and max offsets can be easily computed by examining the projection extremes w.r.t. the origin, which we set as the center of $\mathbf{x}_{q_i}$ (Figure 7 (top)).

To detect rotational mobility, we consider the PCA axes of p_i and map them to the common q_i's PCA axes denoted $\mathbf{v}_i=(\mathbf{x}_{p_i},\mathbf{y}_{p_i},\mathbf{z}_{p_i}),\ldots,\mathbf{v}_j=(\mathbf{x}_{p_j},\mathbf{y}_{p_j},\mathbf{z}_{p_j})$ (axes are matched by size). For each PCA axis, if all vectors, e.g. $\mathbf{x}_{p_i},\ldots,\mathbf{x}_{p_j}$ are on the same plane, we define the rotation axis as the normal to this plane. The min/max angle limits are computed from the angles between vectors in the plane and one reference vector(Figure 7 (bottom)). If vectors are not on the same plane, it means objects rotate along more than one axis. In this case, we assume objects to be rotational symmetric around the dominant PCA axis, and simply take the other two PCA axes as the rotational mobility axes.

For supported parts without hard mobilities we detect weak mobilities instead. We use the assumption that an object can move along its supporting surface. Once an object has weak mobility, we compute the normal of its supporting surface and use it as an arbitrary rotational axis. The two directions that are orthogonal to the normal serve as the 2D translational axes. Together, they allow the object to translate and rotate along the supporting surface. A node that has neither hard nor weak mobility is considered not movable with respect to its supporting node. Embedding the detected mobilities into the standardized support tree generates the *mobility-tree*, which provides a high-level functional representation of a complex indoor scene.

4.4. Mobility Controllers (UI)

We utilize the detected mobilities for high-level editing controls of large scale scenes. Repetitive parts in the scene form groups which allow simultaneous editing operations utilizing their mobilities. Thus, we develop several *group-controls* and part controls as follows:

- **Alignment**: the user selects a group of repetitive objects simply by selecting one of its members. A click on the alignment button aligns all the objects in the group. If the group has translational mobility, objects in the group will translate to the min offset line. Similarly, in case of rotational mobility, objects will align with the circle around the rotation axis. In Figure 1, pears on the table and chairs around the table align using translational and rotational mobilities respectively.
- **Regularization**: the user selects a group and clicks the regularization button. In case of translational mobility, objects in the group are translated to regular intervals within the translational limits. Similarly, in case of rotational

Figure 8: *In a bookshelf (left) we detect translational and rotational book mobilities (mid-left). Applying mobility controllers, we get various bookshelf reorganizations: piled (center), aligned and uniformly distributed (mid-right) and randomized (right).*

mobility, objects are rotated around their rotation axis to a regular angular interval (Figure 8).

- **Randomization**: In contrast to regularization, objects within a group are repositioned following their group mobility at random intervals within the mobility limits.
- **Deformation**: the user selects a single object and moves it according to its mobility and limits simply by moving the mouse left-to-right in a 1D manner.

The generation of global object transformations from local editing operations is not trivial. E.g., it is not obvious when rotating an office chair's seat, what should be the leg wheels movement? We use our support tree to solve these problems. As the user edits a part, we first search for the *leaf node L* in the mobility-tree that contains the part. If L is a supporting node (or a supported node with no mobility), we trace its ancestor until we find a movable supported node E. Based on the mobility information stored at E, relevant controllers pop, and the user can select a controller to move the corresponding object of E within its mobility limits. All children of E shall move with E together. Thus, if we rotate the seat of an office chair, only the back and armrest move while legs stay intact (Figure 4);

5. Results and Applications

We tested our algorithm on various indoor scenes originating from range scans, manually modeled and the web. All our scenes are represented by arbitrary meshes without making any geometrical or topological assumptions. On average scene sizes are $100K$ polygons (see Table 2). We run all our results on a 2.4Ghz dual core CPU with 4GB RAM. Our processing times are as follows: automatic over-segmentation into superpatches was 20 secs on average. In the interactive step, the user analyzes and detects incorrect segmentation and labeling, and manually corrects and refines it. Interaction was 4 mins on average (8 mins max), and the user provided 4 scribbles on average. Mobility computation took 10 secs on average (30 secs max). The ϵ-threshold for triangle similarity in the superpatch growing is set to 1.0 for most

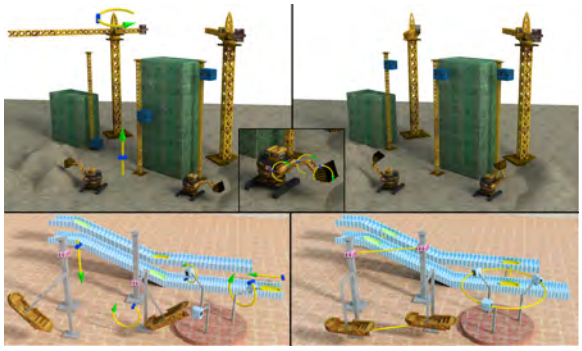

Figure 9: *Functional mobility detection and manipulation of outdoor scenes.*

scenes (i.e. if the distance between two patches is smaller than this, they are connected). Nevertheless, in the scanned scenes in Figure 11 we set the value to 0.1 due to extensive noise.

Figure 8 demonstrates a comprehensive result on a relatively simple input. Books have both rotational and translational mobilities with respect to the supporting shelf, thus allowing a high degree of control for reorganizing the scene. Through manipulating mobility controllers, users can rotate or translate books either individually or collectively, to lay them down, pile them up, or distribute them regularly or randomly.

Our algorithm is also applicable to general scenes (non-indoor), as long as repetitive objects exist. In Figure 9, we demonstrate mobility extraction in two outdoor scenes. We apply our algorithm in a straight forward manner detecting and analyzing repetitive object parts and subparts. Our algorithm was able to accurately compute mobility controllers for reorganizing and aligning these scenes.

We have tested our method on complex indoor scenes downloaded from Google Warehouse consisting of multiple ob-

Figure 10: *We compute mobilities in complex indoor scenes (left column). Using computed controllers (green arrows), we can simultaneously align and regularize objects, allowing easy manipulation of the scenes (right column).*

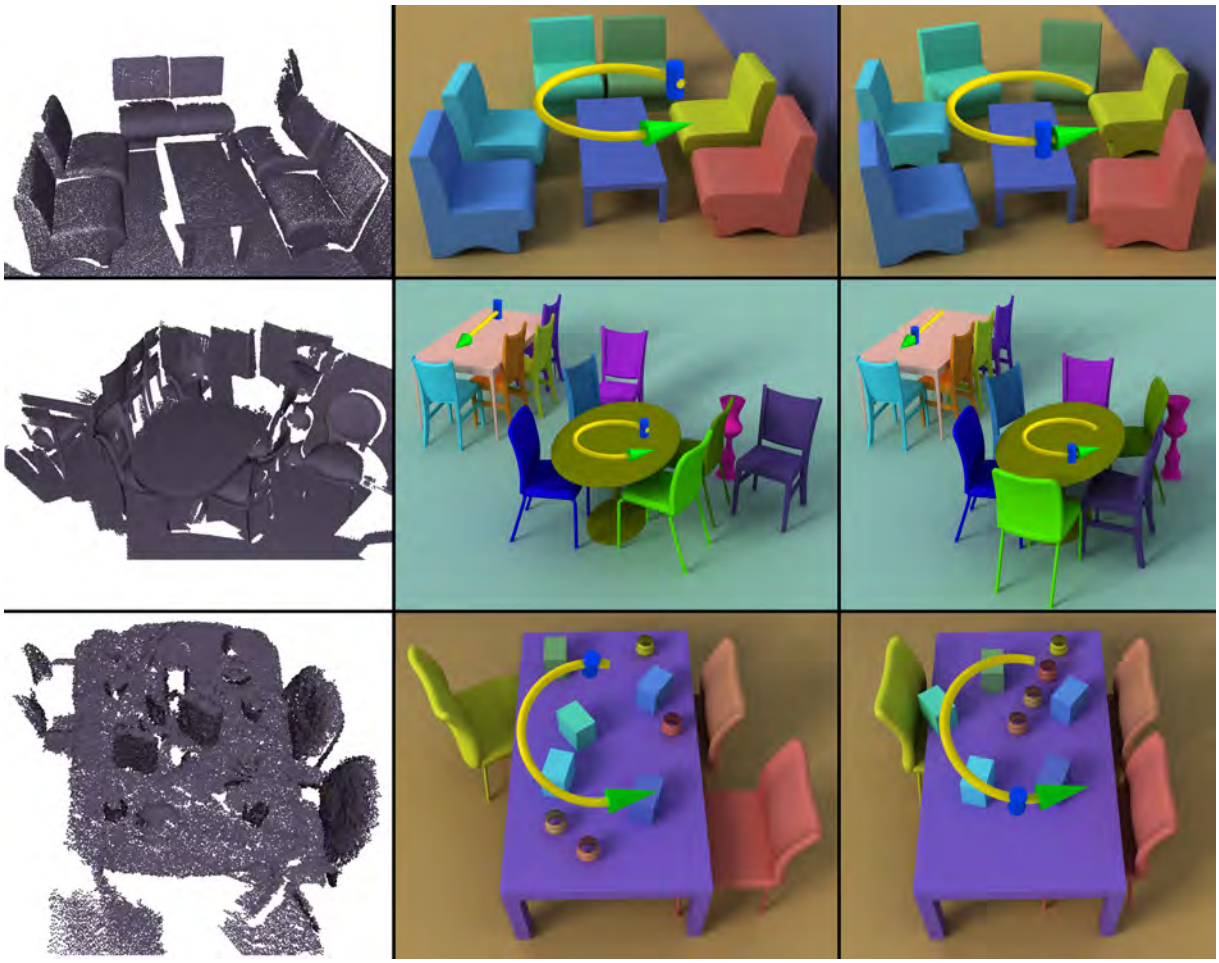

Figure 11: *Three scene models reconstructed from raw scans (left). We detect mobilities (middle) and reorganize the scenes using our algorithm (right).*

jects and parts of various types. In Figure 10, we show results on various scenes. The left column shows the scene with detected mobility controls, the right shows the reorganized scene by simultaneously aligning and distributing objects by their detected functional mobility. We show weak rotational mobility for cupboard door and chairs (top row), and weak translational mobility for telephone (2nd row from top). Hard translational mobilities are shown for curtains and drawers (mid row); rotational for pictures, computer screens and lamp joints (2nd row from bottom).

In Figure 11, we demonstrate a possible extension of our algorithm to reconstructed scanned scenes. We reconstruct three scanned indoor scenes (left, middle) and rapidly reorganize them using detected mobilities.

Figure 12 shows the result of our mobility computation and manipulation on a non-indoor scene of repeating cars. We extract several rotational mobilities for wipers, wheels and doors as well as translational mobilities in the sunroof. Note that since both front and back wheels are identical, they are grouped in the same mobility group, resulting in an unnatural rotation of the back wheels.

To detect all mobilities, we need object repetitions with sufficient pose variations. In Figure 13 we test our algorithm on a very complex robot arm with many rotational joints and degrees of freedom. The mobilities of the hand (red axes in zoom in) were detected only after adding the two rightmost arm variations.

In Table 1, we show a user study comparing between scene manipulation with and without using our mobility controllers. We have asked a group of 10 end-users, artists and CG researches to reorganize and manipulate 5 input scenes (Figure 10 left) that were segmented into individual objects in a

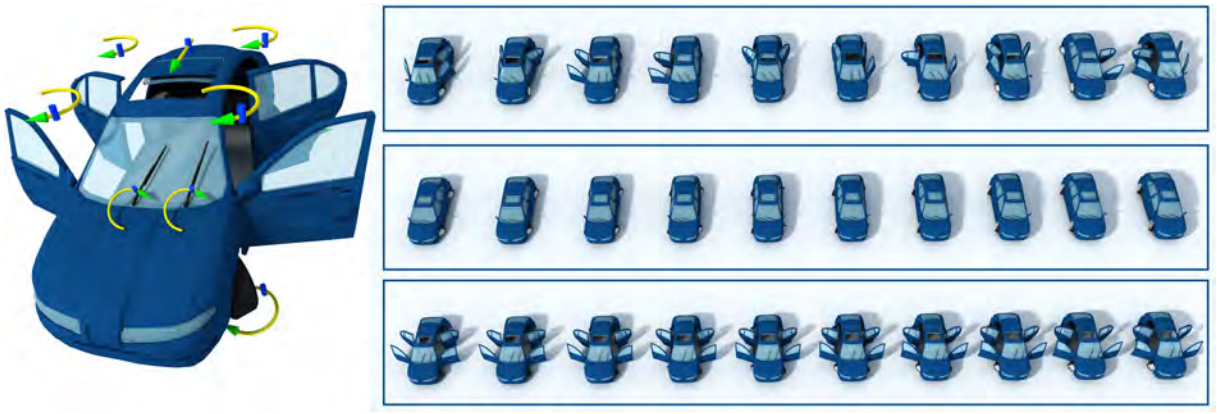

Figure 12: *Mobility extraction from repeating cars in a parking lot (top row). Middle, bottom rows show simultaneous mobility editing of car parts.*

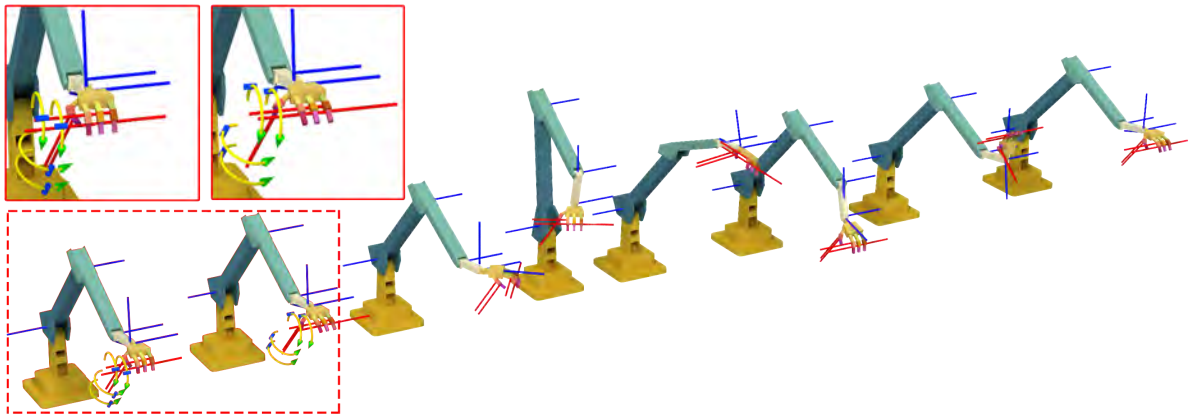

Figure 13: *Complex rotational mobilities are progressively extracted from repetitions of a robot-arm.*

preprocessing step. In our study, users were presented with the target organized scene (Figure 10 right) and were asked to manipulate their input to reach this goal. The required manipulation consisted of alignment, regularization and deformation of objects in the scene. Each scene was manipulated twice, with and without using our mobility controllers. Without mobility controllers, the user manually selects individual objects in the scene, and translates and rotates them around with the mouse. Interaction timings (columns 3 and 4) show more than an order of magnitude difference between mobility-based and manual editing. Column 2 shows preprocessing timings of our mobility tree computation.

Table 2 summarizes our results in detail. For each result, we specify its origin whether made by a modeler, downloaded from Google sketchup or reconstructed from a raw scan of a real 3D scene. Next are the number of objects and subparts in each scene and the number of detected mobilities. The rightmost column describes each of these mobilities by the part name, and number of its repetitions in the scene, followed by mobility type (translational, rotational, weak).

Limitations Our mobility-tree construct is based on two major observations: the existence of supporting–supported relations between objects and pose variations of repetitive objects. If these assumption do not hold, our algorithm will not compute the correct mobilities. For example, repetitive objects having exactly the same pose (e.g. all drawers are closed), will not have any mobility. On the other hand, object variations may be unnatural resulting in incorrect mobility (in Figure 12, back wheels). In such cases, a possible workaround is to manually introduce and adjust pose variations by editing the scene before applying our technique.

Fig	Source	#Parts	#Controls	Controllers Mobility
1	Modeler	74	7	C1: 5 X apple (translate) (weak) C2: 5 X pear (translate) (weak) C3: 7 X book (rotate) C4: 2 X lamp (top) (rotate) C5: 2 X lamp (bottom) (rotate) C6: 6 X chair (translate) (weak) C7: 6 X drawer (translate)
2	Modeler	131	6	C1: 25 X wheel (rotate) C2: 5 X seat (rotate) C3: 5 X seat (translate) C4: 5 X back (rotate) C5: 5 X arm(top) (translate) C6: 5 X arm(bottom) (translate)
3	Modeler	13	1	C1: drawer X 10 (translate)
9(a)	Modeler	22	5	C1: 2 X cantilever (rotate) C2: 2 X elevator (translate) C3: 2 X bulldozer (top) (rotate) C4: 2 X bulldozer (mid) (rotate) C5: 2 X bulldozer (bottom) (rotate)
9(b)	Modeler	30	5	C1: 2 X sliding boat (green) (translate) C2: 2 X sliding boat (yellow) (translate) C3: 2 X mega drop (translate) C4: 3 X roller (rotate) C5: 2 X pirate ship (rotate)
10(a)	Sketchup	144	10	C1: 2 X pillow (rotate) C2: 2 X lampshades (translate) C3: 2 X curtains (translate) C4: 24 X book (rotate) C5: 2 X door (rotate) C6: 6 X drawer(long) (translate) C7: 3 X drawer(short) (translate) C8: 5 X apple (translate) (weak) C9: 4 X pear (translate) (weak) C10: 5 X cup (translate) (weak)
10(b)	Sketchup	130	5	C1: 6 X droplight (translate) C2: 8 X drawer (bedside table) (translate) C3: 5 X drawer (translate) C4: 1 X disk (translate&rotate) (weak) C5: 1 X telephone (translate&rotate) (weak)
10(c)	Sketchup	170	7	C1: 4 X curtain (translate) C2: 4 X drawer (translate) C3: 2 X picture (rotate) C4: 3 X screen (rotate) C5: 2 X lamp (top) (rotate) C6: 2 X lamp (middle) (rotate) C7: 2 X lamp (base) (rotate)
10(d)	Sketchup	338	7	C1: 8 X book (rotate) C2: 2 X book-group(a) (translate) C3: 2 X book-group(b) (rotate) C4: 2 X book-group(c) (translate) C5: 2 X book-group(d) (rotate) C6: 5 X book (gray) (rotate) C7: 12 X drawer (translate)
10(e)	Sketchup	203	6	C1: 4 X drawer (short) (translate) C2: 8 X drawer (long) (translate) C3: 2 X door (long) (rotate) C4: 4 X door (short) (rotate) C5: 4 X drawer (bedside table) (translate) C6: 3 X picture (translate)
11(a)	Raw scan	7	1	C1: 6 X chair rotate) (weak)
11(b)	Raw scan	12	2	C1: 6 X chair (translate&rotate) (weak) C2: 3 X chair (translate) (weak)
11(c)	Raw scan	13	3	C1: 3 X chair (translate) (weak) C2: 4 X cup (translate)(weak) C3: 5 X box (translate)(weak)
12	Modeler	192	5	C1: 20 X door (front) (rotate) C2: 20 X door (back) (rotate) C3: 20 X wiper blades (rotate) C4: 10 X sunroof (translate) C5: 40 X wheel (rotate)
13	Modeler	137	6	C1: 8 X arm(1) (rotate) C2: 8 X arm(2) (rotate) C3: 8 X arm(3) (2D-rotate) C4: 8 X hand (3D-rotate) C5: 40 X finger(1) (rotate) C6: 40 X finger(2) (rotate)

Table 2: *Summary of mobility results*

Fig	Preprocessing	Mobility Editing	Manual Editing
1	91s	13s	519s
2	100s	9s	485s
8	30s	5s	299s
10(a)	95s	15s	613s
10(b)	63s	5s	451s
10(c)	83s	7s	536s
10(d)	114s	6s	448s
10(e)	97s	5s	513s
12	117s	7s	565s
13	90s	6s	639s

Table 1: *Mobility editing vs. manual editing user study*

6. Discussion and future work

We present an algorithm for detecting and manipulating functional mobilities in a static indoor scene. To achieve our goal, we analyze the support relationship among objects and detect their repetitions. Based on pose variation of repetitive objects, we are capable of detecting translational and rotational mobilities, and analyze the corresponding axes and limits of motions. We can also infer the mobilities of objects along their respective supporting surfaces. Finally, we integrate mobility information into a hierarchical representation and generate high-level controllers for scene editing.

In the future we plan to incorporate additional object relations, such as symmetry, dependency and regularity, into the mobility detection, as well as design additional high-level UI controllers. We would also like to pursue the idea of functional mobilities for scene understanding.

Acknowledgements: this work was partially supported by grants from NSFC (61103166, 61232011, 61025012), CAS Young Scientists (2013Y1GA0007), Guangdong Science and Technology Program (2011B050200007), Shenzhen Innovation Program (CXB201104220029A, KQCX20120807104901791, JCYJ20130401170306810, ZD201111080115A, KC2012JSJS0019A), Israel Science Foundation (ISF) and European IRG FP7.

References

[AON05] AKAZAWA Y., OKADA Y., NIIJIMA K.: Automatic 3D scene generation based on contact constraints. In *Computer Graphics and Artificial Intelligence* (2005), vol. 8, pp. 51–62. 3

[BS95] BUKOWSKI R. W., SÉQUIN C. H.: Object associations: a simple and practical approach to virtual 3D manipulation. In *Proc. of Symp. on Interactive 3D Graphics* (1995), pp. 131–138. 3

[CLDD09] CABRAL M., LEFEBVRE S., DACHSBACHER C., DRETTAKIS G.: Structure preserving reshape for textured architectural scenes. *Eurographics* (2009), 469–480. 3

[FRS*12] FISHER M., RITCHIE D., SAVVA M., FUNKHOUSER T., HANRAHAN P.: Example-based synthesis of 3d object arrangements. In *Proc. of ACM SIGGRAPH ASIA* (2012), vol. 31. 2, 3

[FSH11] FISHER M., SAVVA M., HANRAHAN P.: Characterizing structual relationships in scenes using graph kernels. *Proc. of ACM SIGGRAPH 30*, 4 (2011), 34:1–34:11. 3

[GG04] GELFAND N., GUIBAS L. J.: Shape segmentation using local slippage analysis. In *Proc. Eurographics Symp. on Geometry Processing* (2004), pp. 214–223. 2

[GS09] GERMER T., SCHWARZ M.: Procedural arrangement of furniture for real-time walkthroughs. *Computer Graphics Forum 28*, 8 (2009), 2068–2078. 3

[Kjo00] KJOLAAS K. A. H.: Automatic furniture population of large architectural models, 2000. 3

[KMYG12] KIM Y. M., MITRA N. J., YAN D.-M., GUIBAS L.: Acquiring 3d indoor environments with variability and repetition. *Proc. of ACM SIGGRAPH ASIA 31*, 6 (2012). 2

[KSSCO08] KRAEVOY V., SHEFFER A., SHAMIR A., COHEN-OR D.: Non-homogeneous resizing of complex models. In *Proc. of ACM SIGGRAPH ASIA* (2008), pp. 111:1–111:9. 3

[KZ04] KOLMOGOROV V., ZABIH R.: What energy functions can be minimized via graph cuts. *IEEE Trans. Pat. Ana. & Mach. Int. 26*, 2 (2004), 147–159. 4

[Lyo08] LYONS G.: *Ten Common Home Decorating Mistakes and How to Avoid Them*. Blue Sage Press, 2008. 2

[MSK10] MERRELL P., SCHKUFZA E., KOLTUN V.: Computer-generated residential building layouts. *Proc. of ACM SIGGRAPH ASIA 29*, 6 (2010), 181:1–181:12. 2, 3

[MSL*11] MERRELL P., SCHKUFZA E., LI Z., AGRAWALA M., KOLTUN V.: Interactive furniture layout using iterior design guidelines. *Proc. of ACM SIGGRAPH 30*, 4 (2011), 87:1–87:10. 2, 3

[MYY*10] MITRA N. J., YANG Y.-L., YAN D.-M., LI W., AGRAWALA M.: Illustrating how mechanical assemblies work. *ACM Trans. on Graphics 29*, 4 (2010), 58:1–58:12. 2, 3

[NXS12] NAN L., XIE K., SHARF A.: A search-classify approach for cluttered indoor scene understanding. *Proc. of ACM SIGGRAPH ASIA 31*, 6 (2012). 2

[RM03] REN X., MALIK J.: Learning a classification model for segmentation. In *Proc. of the Ninth IEEE International Conference on Computer Vision* (2003), vol. 1, pp. 10–17. 4

[WXL*11] WANG Y., XU K., LI J., ZHANG H., SHAMIR A., LIU L., CHENG Z., XIONG Y.: Symmetry hierarchy of man-made objects. *Eurographics 30*, 2 (2011), 287–296. 2, 3

[XSF02] XU K., STEWART J., FIUME E.: Constraint-based automatic placement for scene composition. In *Proc. Graphics Interface* (2002), pp. 25–34. 3

[XWY*09] XU W., WANG J., YIN K., ZHOU K., VAN DE PANNE M., CHEN F., GUO B.: Joint-aware manipulation of deformable models. In *Proc. of ACM SIGGRAPH* (2009), vol. 28, pp. 35:1–35:9. 2, 3

[YYT*11] YU L. F., YEUNG S.-K., TANG C. K., TERZOPOULOS D., CHAN T. F., OSHER S. J.: Make it home:automatic optimization of furniture arrangement. *Proc. of ACM SIGGRAPH 30*, 4 (2011), 86:1–86:11. 2, 3

[ZFCO*11] ZHENG Y., FU H., COHEN-OR D., AU O. K.-C., TAI C.-L.: Component-wise controllers for structure-preserving shape manipulation. In *Eurographics* (2011), vol. 30, pp. 563–572. 2, 3