Erschienen in: ACM Transactions on Graphics ; 30 (2011), 6. - 183

Structure-Preserving Retargeting of Irregular 3D Architecture

Jinjie Lin[1] Daniel Cohen-Or[2] Hao Zhang[3] Cheng Liang[1] Andrei Sharf[4,1] Oliver Deussen[5] Baoquan Chen[1]

[1]SIAT [2]Tel-Aviv University [3]Simon Fraser University [4]Ben-Gurion University [5]University of Konstanz

Figure 1: *Retargeting of a real-world irregular 3D architectural model (photo at bottom-left corner) while preserving its structural style.*

Abstract

We present an algorithm for interactive *structure-preserving* retargeting of *irregular* 3D architecture models, offering the modeler an easy-to-use tool to quickly generate a variety of 3D models that resemble an input piece in its structural style. Working on a more global and structural level of the input, our technique allows and even encourages replication of its structural elements, while taking into account their semantics and expected geometric interrelations such as alignments and adjacency. The algorithm performs automatic replication and scaling of these elements while preserving their structures. Instead of formulating and solving a complex constrained optimization, we decompose the input model into a set of sequences, each of which is a 1D structure that is relatively straightforward to retarget. As the sequences are retargeted in turn, they progressively constrain the retargeting of the remaining sequences. We demonstrate interactivity and variability of results from our retargeting algorithm using many examples modeled after real-world architectures exhibiting various forms of irregularity.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems;

Keywords: Model retargeting, irregular 3D architecture

1 Introduction

Generating a scene with a variety of models that share a common characteristic or style is a challenging task. In the absence of a concrete description of the target style, a common approach is to synthesize by examples, where some notion of style in the example is preserved in the generated models. A straightforward means of style-preserving synthesis generates models consisting of contents taken from the example while preserving their general relation. A primary example is example-based texture synthesis [Wei et al. 2009] where the imitated styles are mainly of a local nature. Recently, such techniques have been extended to preserve more global structures in images [Risser et al. 2010; Wu et al. 2010].

Works on inverse procedural modeling can be regarded as abstracting the structure present in the examples into a set of rules and then synthesizing novel models based on the rules [Aliaga et al. 2007; Stava et al. 2010; Bokeloh et al. 2010]. For effective rule extraction, dominant presence of regularities in the examples is essential. In a rather loose sense, works on image or model retargeting [Shamir and Sorkine 2009] also represent a form of style-preserving synthesis, where the style is signified by the salient features of the source model. This direction has led to recent work on example-bases synthesis of facade images [Lefebvre et al. 2010].

In this paper, we are interested in retargeting of 3D architectural geometry which preserves the *structural style* of an input piece. Our focus is on the handling of complex structures, especially *irregularities* in the example architectural piece. Specifically, the input model does not exhibit dominant regular grids; it is typically composed of regular substructures that vary in form and are arranged in an irregular manner in 3D space. Our retargeting technique offers a modeler an easy-to-use interactive tool to quickly generate a variety of 3D models possessing a common structural style; see Figure 1. The large collection of models obtained provides a means for

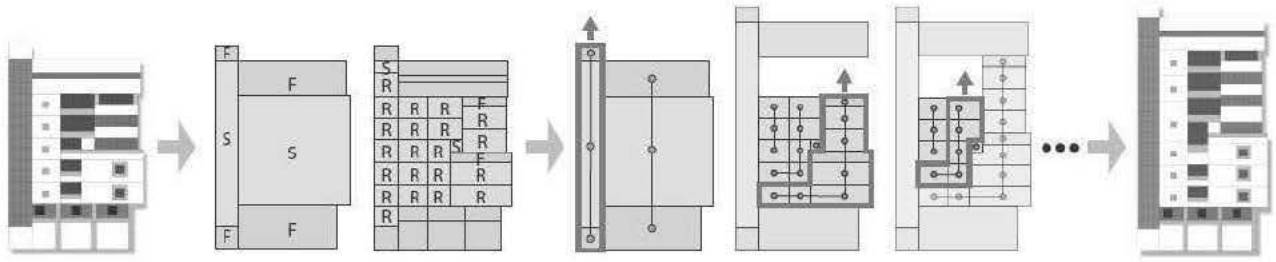

Figure 2: *Major components of our retargeting algorithm. User interactively defines (b) the box hierarchy and behavior attributes (red: replicated; green: scaled; blue: fixed). The rest of the algorithm is automatic. An ordered set of retargetable sequences is computed at each level of the hierarchy — two are shown here in (c). Retargeting is executed by a traversal of the box hierarchy and operating on the retargetable sequences in turn — from left to right, three such sequences are shown with a red border in (c).*

creativity support [Shneiderman et al. 2006; Chaudhuri and Koltun 2010]; they can be explored by artists and engineers and stimulate them during creative design and modeling.

Traditional retargeting methods focus on preserving salient features while traditional example-based synthesis are aimed at reproducing features at a local scale. Our technique works on a more global, structural level of the input, e.g., it allows and even encourages replications of structural elements such as windows so as to minimize stretching. Such minimization avoids deforming the elements beyond what their semantic or engineering constraints would allow. Existing methods on retargeting architectural scenes have operated on image data with the recent work of Wu et al. [2010] being structure-oriented. Their work focuses on retargeting detected regular grid structures only. Many modern architectural models, even the building facades alone, present amazing varieties including irregularities in diverse forms (see figures throughout the paper). Such models cannot be easily parsed by grammar-based methods [Wonka et al. 2003; Aliaga et al. 2007; Mueller et al. 2006], nor can they be handled by the method of Wu et al. [2010] since the facade cannot be expressed by a grid, an aggregation of sub-grids, nor is there a simple partition into floors of a regular structure. Our technique aims to retarget architecture scenes exhibiting general irregularities and it operates in the 3D object space.

To obtain a general solution to the structure-oriented retargeting problem, a tie to semantic analysis is inevitable. Ideally, the retargeting should preserve certain semantics of the structural elements of the input and certain semantic relationships among them. It is difficult to learn automatically the semantics from the geometry of a single input. Retargeting based purely on detected structural regularities can result in much ambiguity, since a multitude of repeated patterns can be present along any retargeted dimension; see Figure 2. Indeed, there is generally no unique way to define the most meaningful retargeting result; the results depend on semantic interpretations of the input which can differ significantly according to design needs or user preference.

To incorporate semantics and alleviate the issue of ambiguity, our retargeting algorithm starts with a semi-automatic analysis of the input architectural structure. The user interactively partitions the input and hierarchically groups the elements as a means to define the semantic interpretation and influence the retargeting behavior. Regardless of the geometric complexity of the elements, the user only needs to manipulate their axis-aligned bounding boxes. Each box is tagged with an attribute that indicates its behavior under retargeting; a box can be scaled ($\mathcal{S}$), replicated/deleted ($\mathcal{R}$), or stay intact ($\mathcal{F}$). Altering the hierarchy or behavior attributes can lead to characteristically different variations.

The retargeting step operates on the set of bounding boxes. The boxes are replicated or scaled according to the resizing parameters while respecting the user-defined semantic properties and automatically detected geometric interrelations between the boxes such as alignments. The computational challenge is to achieve interactive retargeting amid the multitude of constraints. Instead of formulating and solving a complex constrained optimization, we break down the problem into simpler-to-solve pieces. At the core of our approach is a decomposition of a given irregular arrangement of boxes into a set of disjoint sequences. Each sequence is a 1D structure that is relatively straightforward to resize. As the sequences are retargeted in turn, they progressively constrain the retargeting of the remaining sequences. We develop a scheme to find and order these sequences to more effectively simplify the retargeting problem at each step. Figure 2 illustrates the major steps of our algorithm.

We demonstrate interactivity and variability of results from our retargeting algorithm using many examples modeled after real-world architectures exhibiting various forms of irregularity. The produced model varieties can be easily plugged into a virtual scene for movie or game production, urban planning, and architectural design. Our technique is not designed for controlled editing. Instead, it can be used to quickly and easily generate many models, even including "surprises", for a modeler to choose from and be creative.

2 Related work

Several topics involving scene analysis and modeling are relevant to our work. In general, our goal of structure-preserving retargeting differs from those of existing works on image-space retargeting and example-based texture synthesis. Most works on structural analysis, particularly those on inverse procedural modeling, strive for deterministic structure inference based on various heuristics. We aim for interactive shape manipulation allowing semi-automatic analysis to resolve the inevitable ambiguity resulting from different semantic interpretation of the structural elements.

Image-space retargeting. The well established example-based texture synthesis approach [Wei et al. 2009] has led to advances in media retargeting [Shamir and Sorkine 2009], with a few recent works focusing on architectural textures [Lefebvre et al. 2010; Wu et al. 2010]. Most works on image retargeting have been saliency-driven. Lefebvre et al. [2010] cut the source image into horizontal or vertical strips to avoid important features and then reassemble them into a new size. Wu et al. [2010] resize architectural models in an image by explicitly exploiting symmetry. Their work shares similarities with our work in that it also analyzes the structure of the input prior to retargeting. However, their method is in image space

and more importantly, their symmetry summarization is designed to retarget regular structures formed by a collection of grids. Our work aims to handle general irregular structures.

Structural analysis. Many recent works attempt to detect regular patterns in a scene based on symmetry analysis, e.g., [Pauly et al. 2008; Mitra et al. 2010]. The symmetric elements can then be organized into a hierarchy [Wang et al. 2011] or to infer a generative model [Mueller et al. 2007; Stava et al. 2010; Bokeloh et al. 2010]. In particular, the work of Stava et al. [2010] on inverse procedural modeling is more relevant to our work. They analyze a more general pattern than a facade aimed at learning the irregular structure therein in order to generate an L-system that describes it. New structures can then be generated by modifying the parameters of the L-system. The automatic analysis problem is indeed quite challenging. Such an analysis typically attempts to infer, deterministically, the underlying generative model. However, the generative procedures behind a pattern, in particular an architectural piece containing irregularities, are often not unique and subject to different semantic interpretations. The goal of our work is orthogonal to that of automatic structure inference; we focus on structure-preserving interactive model generation. With user involvement in the modeling loop, we are able to obtain different forms of editable structures which would lead to more varied model creations.

Grammar-based facade modeling. In the last decade, procedural modeling has received much attention in urban architecture modeling, e.g., [Wonka et al. 2003; Mueller et al. 2006; Mueller et al. 2007; Aliaga et al. 2007; Lipp et al. 2008]. These works develop shape grammars and show their usefulness for modeling facades and architectural models in general. Varying the grammar parameters allows modifying the architectural structure. The grammar scripts can be defined manually [Wonka et al. 2003; Mueller et al. 2006], automatically [Mueller et al. 2007], or with an interactive visual editing tool [Lipp et al. 2008]. The grammars strongly exploit dominant regularities in the models. Inferring the same from irregular structures is much more challenging. With user assistance, our retargeting method is designed to handle irregular architectural geometry. Instead of working from a pre-set procedure, our tool can work directly with existing 3D models from various sources.

The model synthesis method of Merrell and Manocha [2008] takes an existing input model and generates many complex variations while respecting a local constraint, namely every point in the output is locally identical to some point in the input. Aliaga et al. [2007] present an interactive system with objectives similar to ours. They aim at creating novel buildings that preserve the style of the input model. Like [Mueller et al. 2007], they use grammar to describe the structure of a building. The global structure of the building is assumed to have a hierarchical regular structure consisting of a grid with regular floors. This assumption allows the use of a template grammar to describe the floor patterns. Their method does not maintain vertical coherence and is not designed to deal with irregular architectural structures in general.

Structure-preserving editing. Our object-space retargeting is related to structure-preserving shape editing [Kraevoy et al. 2008; Gal et al. 2009; Cabral et al. 2009; Wang et al. 2011]. Kraevoy et al. [2008] introduce an axis-aligned resizing of shapes that preserve differential properties of certain object-space features, allowing them only to rescale isotropically. Similarly, Cabral et al. [2009] deform architectural models while constraining the angles of rectilinear features. In contrast to our work, these methods are oblivious to the semantics of the model's elements. Our work shares more conceptual similarity with iWires [Gal et al. 2009] in the sense that the input shape is expressed by a set of elements, boxes in our case and wires in theirs, and the editing is aware of the element properties and their inter-relations. However, we explore a completely different problem as we allow and even encourage the replication or deletion of model elements during retargeting; iWires still remains a tool for continuous shape deformation.

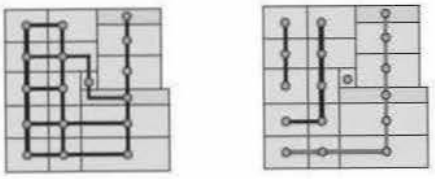

Figure 3: *The box compatibility graph of a 2D arrangement of boxes (left) and the set of retargetable sequences obtained (right). Primary sequence is in red. Note that compatibility between horizontally adjacent boxes requires common attribute and height.*

3 Retargetable sequences

One may approach the retargeting problem using constrained optimization. However, in the presence of structure irregularity, it is difficult to determine what the most meaningful result is; multiple solutions can often be deemed desirable. Moreover, we aim for an interactive tool which a complex optimization may not allow; the complexity is inherent to the discrete nature of our problem and the inhomogeneity of the retargeting constraints.

Breaking down into sequences. We have opted not to formulate the retargeting task as an optimization problem. Instead, we compute one solution that fulfills the multitude of retargeting goals. On one hand, the user imposes semantic constraints including hierarchical grouping of the structural elements of the input and their desired behavior under retargeting. On the other hand, adjacency relations, as well as alignments, between the elements or groups of elements constrain their movement.

To achieve interactivity, we break down the problem into simpler-to-solve pieces. Specifically, we decompose a given irregular arrangement of structure elements into a set of disjoint *sequences*. For lack of a better term, we call these sequences *retargetable* to signify the fact that they are relatively straightforward to retarget. We use the sequences to express intra-relations within a sequence and inter-relations among different sequences of the model. A resizing operation then redistributes the elements in a sequence onto a resized space subject to maintaining these relations. Our goal is for the sequences to progressively simplify the problem, each contributing to the simplification as much as possible. Ideally, we would like the result computed for each retargetable sequence to *maximally* constrain the retargeting of the remaining sequences.

Primary and secondary sequences. Given an arrangement of structural elements (the retargeting space), the retargeting of the very first sequence is unconstrained and its result dominates the overall end result. We call this sequence the *primary* sequence. The remaining sequences are the *secondary* sequences, which are retargeted in turn and constrained by results computed for previous sequences. To more stringently constrain the remaining sequences, the primary sequence needs to be long, i.e., encompassing more structural elements. At the same time, it is still an essentially one-dimensional structure to remain (easily) retargetable.

4 Algorithm overview

The input to our algorithm, the *source* model, is a mesh representing an architectural piece. Operating at the coarse, bounding box level allows the algorithm to work directly on possibly non-clean on-line

models, such as those from Google Warehouse. The retargeting algorithm works in two steps:

1. The first is an offline structural analysis of the source model which produces: a) an attributed hierarchical tree representation of the structural elements of the source and their grouping; b) alignment constraints between the elements; and c) a set of retargetable sequences.

2. The second step, the retargeting step, is an online, interactive process which dynamically retargets the source based on user-specified resizing parameters.

The retargeting operates on the retargetable sequences in an appropriate order based on the hierarchy while respecting various intra- and inter-sequence constraints. Details of the two steps are given in Sections 5 and 6, respectively.

Assumptions. Technically, our algorithm can take as input any 3D architectural model obtained from various sources. However, to obtain meaningful results, we assume that the structural elements of the 3D source model can be symbolically expressed by a *non-overlapping* tessellation of *axis-aligned* (to one of the canonical axis directions) boxes, e.g., see Figure 3. Except for rare instances, this is indeed the case for most contemporary architectures, as evidenced by examples shown throughout the paper most of which are modeled after real-world buildings. Note that this assumption does not preclude the handling of non-rectangular, e.g., curved, elements, e.g., see Figure 9 (third row) — the structure enclosed by a box can assume an arbitrary shape. To generate variations from the source, we assume that the resizing is performed along one dimension at a time. The retargeting result depends on the ordering of the dimensions, a positive (side) effect of which is that multiple variations may be produced towards the same final target size.

Bounding boxes and hierarchy. This step constitutes a user-directed semantic analysis of the source model. First, the user interactively defines a semantic grouping of the structural elements of the source. Each element or group of elements is characterized by an axis-aligned bounding box. Grouping of elements is intended to allow them to behave as a whole, e.g., to be replicated. In turn, the grouping can define a hierarchical tree representation of the boxes, which we call the *box hierarchy tree*; see Figure 2 for a two-level box hierarchy. Along the process of defining the hierarchy, the user tags each box in the hierarchy with one of three *behavior attributes* which indicates the box's behavior under retargeting.

Box alignment. Preservation of proper alignment between the structural elements or their corresponding boxes is a natural constraint to enforce for retargeting of architectural models. Our analysis *automatically* detects alignments between all the boxes, specifically their faces, at the bottom (finest) level of the hierarchy only. Instead of recording all alignment pairs, we use a look-up table which stores for each horizontal side of each box a unique *alignment index* — box sides sharing the same index are aligned in the input model. These alignment indexes are propagated from the bottom level up in the hierarchy as long as the box sides coincide.

Retargetable sequences. We compute a set of retargetable sequences at each node of the box hierarchy. Given a set of boxes forming a retargeting space (these boxes tessellate the bounding box associated with a node in the box hierarchy), we first define a *box compatibility graph* where two boxes are connected by an edge if they are compatible. Intuitively, two boxes are compatible if it is straightforward to retarget them in a sequence; see Figure 3 for an example and Section 5 for the definition of compatibility.

Computation of the retargetable sequences is via a *constrained longest-path* graph traversal over the compatibility graph. After finding the primary sequence which spans the whole height of the retargeting space for vertical retargeting (retargeting along the other two dimensions is similar), the secondary sequences are computed by applying the same scheme to the compatibility graph after removing all nodes and edges belonging to sequences obtained so far. Figure 3 shows the decomposition of a facade into a primary (red) and a few secondary sequences.

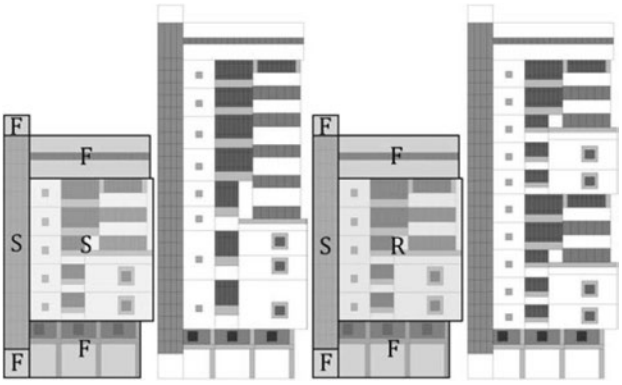

Figure 4: *Different retargeting results by changing the behavior attribute of one box (from $\mathcal{S}$ to $\mathcal{R}$). The behavior attributes of the finest-level boxes are shown in Figure 3.*

Structure-preserving retargeting. To resize along the vertical direction, we traverse a *sequence hierarchy tree* in *depth-first* order. The sequence hierarchy is constructed from the box hierarchy by reordering its nodes based on the computed retargetable sequences. Specifically, sequences at each node in the box hierarchy are sorted from primary to secondaries and within each sequence, the boxes are ordered along the sequence. The retargeting of each sequence is constrained by a) the available distributable length (positive for stretching and negative for contraction); b) the behavior attributes associated with boxes along the sequence; and c) alignment constraints enforced by boxes from previously retargeted sequences. Section 6 describes the above in detail.

The 3D picture. While our algorithm is designed to work on 3D arrangements of boxes, most illustrations are 2D to more intuitively convey the key ideas. We mainly use the example in Figure 2 to illustrate our retargeting algorithm. Furthermore, we focus on explaining concepts and procedures involved with a *vertical* retargeting session in Sections 5 and 6.

The only step of our algorithm that is *dimension-independent* is the box hierarchy construction. All boxes are axis-aligned 3D boxes and the same 3D box hierarchy is used for retargeting in all directions. The behavior attributes depend on the retargeting dimension. For example, a door can be repeated along the horizontal direction but not along the other two directions. In our system, each box is tagged with three behavior attributes, one per direction. For vertical retargeting, box alignment is in reference to the top and bottom faces of the boxes. For horizontal retargeting, top and bottom are replaced by left and right, and for retargeting along the depth, by front and back. The box compatibility graph is defined accordingly, which results in different retargetable sequences and sequence hierarchies for different retargeting directions.

5 Structural analysis

Given the source model, the user first interactively defines the box groupings to obtain a box hierarchical graph. Box alignments are

then detected and stored in a look-up table. Finally, retargetable sequences are computed over the box compatibility graph constructed over nodes in the box hierarchy. These steps are all part of the offline structural analysis, with the first step a manual process and the other steps performed automatically.

Bounding boxes and hierarchy. The user first defines an axis-aligned bounding box using a series of cutting planes for each structural element of the source including window, balcony, door, etc. These boxes together form a non-overlapping tessellation of the 3D space occupied by the input. The user then defines a hierarchical grouping over the boxes in a recursive, bottom-up fashion. This results in a box hierarchy tree whose nodes correspond to bounding boxes and edges signify box containment. The root bounds the whole model and leaves contain the actual structural elements.

Grouping of structural elements of different forms reflects an understanding of the model semantics. In our current implementation, the finest-level boxes and the box hierarchy are defined manually. The number of finest-level boxes depends on the complexity of the input and it varies from tens to a hundred. The complexity of the hierarchy is dictated by the extent of fine-grained control the user desires and the model semantics. The typical number of levels in a box hierarchy is between two to five, e.g., see Figure 2(b).

Behavior attributes. Each box in the hierarchy is tagged by the user with a behavior attribute: an $\mathcal{F}$-box must remain intact; an $\mathcal{R}$-box can only be replicated or deleted; and an $\mathcal{S}$-box can only be scaled. Changing these attributes, for even few boxes, is a quick way of generating interesting variations. Figure 4 shows two different retargeting results by changing only one box attribute.

While the box hierarchy is constructed bottom-up, tagging of the behavior attributes is executed top-down to allow early stopping, e.g., none of the lower-level boxes need to be tagged if the current box is marked as an $\mathcal{F}$. The default attribute is $\mathcal{S}$ since retargeting is a scaling transform. Assignment of the $\mathcal{R}$ and $\mathcal{F}$ attributes depends on knowing the semantics of the elements.

Box alignment. For vertical retargeting, horizontal faces (there are two per box, one top and one bottom) of the finest-level boxes determine potential alignment groups. In the source model, when a number of faces align horizontally within an ϵ-precision and their total area exceeds the average, they define an alignment group. An alignment group is given a numerical index. If a face does not belong to any alignment group, its index is null. All the alignment indexes define a look-up table T_{align} which stores all the horizontal box faces with their corresponding alignment indexes. Similarly, boxes at higher levels of the hierarchy inherit alignment indexes from the finest-level boxes they contain if their faces coincide.

Box compatibility graph. We construct a compatibility graph $C_{v_i} = (V^C_{v_i}, E^C_{v_i})$ at each node v_i of the 3D box hierarchy; the graph is a 3D structure. Let i be the box at v_i. The graph C_{v_i} is used to compute the retargetable sequences at node v_i. Specifically, these sequences form a decomposition of the arrangement of boxes which tessellate box i — these boxes are the children of v_i in the box hierarchy, forming the retargeting space. These boxes, along with two virtual terminal boxes, constitute the set of nodes of C_v. The two virtual boxes are attached to the top and bottom of box i and will serve at the start and end of the primary sequence.

An edge between two nodes in C_{v_i} corresponding to boxes b_1 and b_2 is defined if the boxes are *compatible*: b_1 and b_2 are compatible if a) one horizontal face of b_1 is adjacent to (not necessarily coincides with) one horizontal face of b_2; or b) one vertical face of b_1 coincides with one vertical face of b_2 and the two boxes b_1 and b_2 are of the same behavior attribute. Note that each axis-aligned box has two horizontal faces and the remaining four are vertical faces.

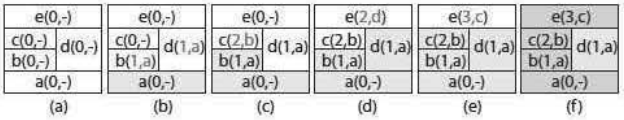

Figure 5: *A simple 2D box compatibility graph and the longest-path computation to find the primary sequence (orange). Visited nodes are in gray and the pair in parentheses give the current stored path length and the preceding node for that path.*

One can easily observe that box compatibility is defined in such a way that a vertical resizing of two compatible boxes is straightforward. In case b), regardless of the behavior attribute, the coincident vertical face will simply remain so after a resizing. Figure 3 shows a 2D box compatibility graph as an illustration.

Sequence construction. Given a 3D arrangement of boxes constituting a compatible graph C, we first describe construction of the primary sequence. Our scheme computes the longest-path in C via dynamic programming, which traverses the graph starting from the node that is at the *bottom-front-left* of the retargeting space. The traversal is *restricted* in that it cannot move to the left, down, or back to the front direction. During traversal, we record at each visited node v a pair consisting of two pieces of information: the length of the *longest* path from the starting node to v so far and the predecessor node along that path. Figure 5 illustrates longest-path computation on a simple compatible graph, showing how the pair of values are updated and the resulting primary sequence.

Computation of the secondary sequences follows the same form of computation, except that all the nodes and edges along previously computed sequences are removed from the graph C. Note that this may result in a disconnected graph C'. We still designate two virtual terminal nodes. The bottom terminal is connected to the lowest boxes in each disconnected component of C', where a lowest box is one whose bottom face has the smallest height. The top terminal is defined similarly. Refer to Figure 3 for the primary and secondary sequences computed over a simple yet full compatibility graph.

Note that each secondary sequence must keep track of the two boxes that are attached to its two ends. We call these the top and bottom *anchor boxes* for the sequence. An anchor box can belong to a previously computed sequence or be a virtual terminal. During dynamic retargeting, the anchor boxes are used to determine the distributable height a secondary sequence receives, as well as to keep track of where its top and bottom faces are as the preceding sequences are retargeted.

Sequence hierarchy The sequence hierarchy is constructed offline and it will be traversed during dynamic retargeting to determine the order in which the primary and secondary box sequences will be retargeted. Consider a node v whose associated retargetable sequences are q_1 (the primary), q_2, ..., q_k, in the order they were computed. Let the sequence of boxes for q_i be $b_{i1}, \ldots, b_{im_i}$, in order, along the sequence. Then the children of v in the sequence hierarchy are sorted in the order of $b_{11}, \ldots, b_{1m_1}, b_{21}, \ldots, b_{2m_2}, \ldots, b_{k1}, \ldots, b_{km_k}$; see Figure 6 for a simple illustrative example.

6 Structure-preserving retargeting

The on-line dynamic retargeting process is carried out one sequence at a time with an order determined by a depth-first traversal of the sequence hierarchy tree. When retargeting a particular sequence, the sequence is given a height quantity (we again only describe vertical retargeting in this section with the situation involving retarget-

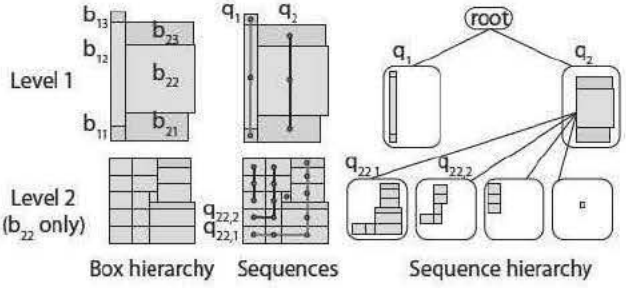

Figure 6: *Example of a (partial) sequence hierarchy tree with its corresponding box hierarchy (left). Square boxes enclosing the b's are actual boxes in the box hierarchy. Oval boxes indicate retargetable sequences, denoted by the q's. A depth-first traversal of the sequence hierarchy yields the following sequences retargeted in order: q_1, q_2, $q_{22,1}$, $q_{22,2}$, ... The box b_{21} is not processed in the retargeting order since it has a behavior attribute $\mathcal{F}$.*

Algorithm 1 Pseudocode: On-line retargeting.

```
Input: b = Bounding box of the whole model
  for each retargetable sequence q_i of b in order do
    move q_i to align its bottom with its bottom attachment
    determine the distributable height ΔH
    distribute ΔH to boxes of q_i
    maintain and update soft alignments
    for each box b_i of the sequence q_i in order do
      recursively retarget b_i
    end for
  end for
```

ing along the other two directions easy to derive) to be distributed among the boxes along the sequence. The precise positioning of the boxes is determined by the distribution subject to global alignment constraints as dictated by already retargeted boxes. See Algorithm 1 for pseudocode for the on-line retargeting process.

Retargeting order. The retargeting order is determined by traversing the sequence hierarchy in *depth-first* order, following edges connecting the nodes, i.e., the boxes. However the retargeting is done on the encountered sequences. A sequence may be considered as a "super node" in the sequence graph, consisting of a list of boxes along that sequence. Figure 6 shows a partial sequence hierarchy and its corresponding box hierarchy to provide a simple example of the depth-first traversal.

Distributable height. We define *distributable height* assigned to a box as the difference between the target height of the box under (vertical) retargeting and its current height. Since the primary sequence spans the entire height of the box it belongs to, it receives the full distributable height of the box. The distributable height given to the next secondary sequence is determined by the movement of its two anchor boxes as a result of retargeting the primary sequence — the two ends of the secondary sequence are seen as attached to the anchor boxes. This determination is applied recursively as the sequences are retargeted in turn.

Height distribution. The height assigned to a sequence is distributed among the boxes along the sequence in a cascading fashion based on certain priority. Adjacent boxes along the sequence that are horizontally (or along depth) aligned have the same height and during vertical retargeting, they also share the same distributed height — they can be regarded as one box for height distribution (see boxes marked by S_1 and S_2 in Figure 7). Recall that our

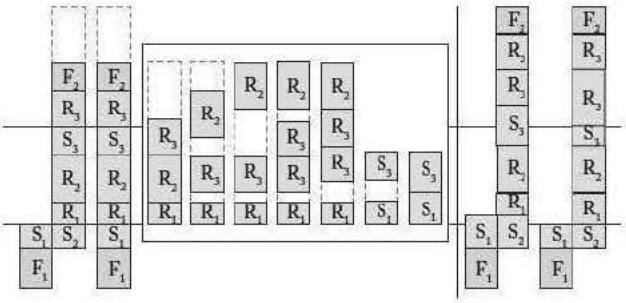

Figure 7: *An illustration of height distribution and box alignment when a sequence is (vertically) retargeted. Compatible boxes (S_1 and S_2) adjacent in a row are treated as one during height distribution. The height of the virtual boxes having red and dashed outlines represents the current distributable height. Observe how this height is distributed to the $\mathcal{R}$ and $\mathcal{S}$ boxes in a cascading fashion. On the right, we show two boxes scaled in succession (bottom to top) due to alignment constraints (given by the black line).*

retargeting encourages replication, hence all $\mathcal{R}$-boxes take precedence over all $\mathcal{S}$ boxes; $\mathcal{F}$-boxes do not resize hence receive no distributable height. Within a set of boxes having the same attribute, priority is set according to the height of the boxes — taller boxes receive higher priority. We now describe the height distribution scheme in detail, assuming the height is positive hence replication and up-scaling are possible. In the case of negative distributable height, the scheme works the same way except that box replication becomes box deletion and up-scaling becomes down-scaling.

The distributable height is first distributed proportionally (with respect to box height) among all the $\mathcal{R}$-boxes, if any. All the $\mathcal{R}$-boxes are sorted by descending heights. Let us take the first $\mathcal{R}$-box. Given its associated distributable height σ' and its current height σ, the box is repeated as much as possible, i.e., $\text{floor}(\sigma'/\sigma)$ times. Any unused height is passed onto the box with the next highest priority. If there are no $\mathcal{R}$-boxes left and there still remains some distributable height, the height is distributed proportionally to all the $\mathcal{S}$-boxes. If there are no $\mathcal{S}$-boxes remaining, then the height is distributed proportionally among all the $\mathcal{R}$-boxes — these $\mathcal{R}$-boxes are then *scaled* according to the heights they receive. Figure 7 shows one instance of height distribution.

Attachment. The attachment between the top and bottom boxes of a sequence to their respective anchor boxes is maintained through retargeting. As one sequence is retargeted, its boxes reposition and any of these boxes that is an anchor box simply "drags" the corresponding secondary sequence with it. In the case of a contraction, an anchor box belonging to sequence may be seen as deleted. However, we implement deletion of boxes in a *virtual* way. Specially, deleted boxes will be assigned a height of zero with the underlying box structure unchanged. Hence, the attachment relationship is retained just as in the case of a stretching.

Soft box alignment. As each box along a sequence determines its new height and as new boxes are created via replication, the exact positioning of the boxes, specifically their top and bottom faces for vertical retargeting, are also influenced by *soft* alignments. Among constraints given by alignment, distributed heights, and attachment, the precedence goes as follow:

Precedence: Attachment > Alignment > Distributed heights.

Note that attachment constraints are hard constraints. Two faces from two boxes are to be aligned if they share the same alignment

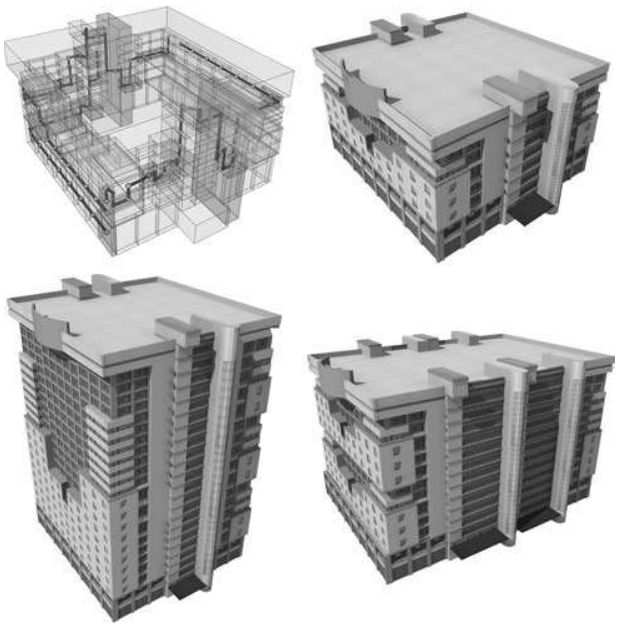

Figure 8: *On the top left, we show the 3D arrangement of boxes and sequences traversing the 3D object. Top right is the corresponding 3D model of the building used in many of the 2D illustrations. In the bottom row, we show two retargeting results.*

index in the input model. Before retargeting starts, each alignment group in the look-up table T_{align} has a status "not-updated". After a face f belonging to an alignment group is repositioned, the group's status changes to "updated" and the remaining faces in the group to be repositioned will seek to align with f, if possible. Any newly created face due to replication has an alignment index null; it does not have an alignment to maintain.

The alignments are enforced in a *greedy* manner, following the retargeting order of the sequences. Within each sequence, the positions (i.e., alignments) of the boxes are determined from bottom to top. When a face has a null alignment index, the distributed height assigned to its box dictates how it is positioned. Otherwise, alignment constraint takes effect (e.g., see Figure 7 for a simple example) unless it breaks an attachment, causes any box to have a negative height, or must force an $\mathcal{F}$-box along the sequence to resize. The latter would happen if all the boxes remaining in the sequence are of type $\mathcal{F}$. If that is not the case, the series of $\mathcal{F}$ boxes are simply "skipped" as if the face is adjacent to the next non-$\mathcal{F}$ box, while of course the height of the $\mathcal{F}$-series is maintained. Due to above conditions, not all alignment constraints are fulfilled; this is allowed since all alignments are soft constraints. At the same time, we should note that alignment constraints may lead to the rescaling of certain $\mathcal{R}$-type boxes, typically to compensate for misalignments globally, thus allowing for more alignments.

7 Results

We first show, in Figure 8, the 3D model of the building used in many of the 2D illustrations so far, as well as two retargeting results. Figure 9 is a gallery of retargeting results, where each input piece is either modeled by an artist after a real-world building or obtained from Google Warehouse. Figure 10 shows a virtual residential community set up by one family of retargeted models.

The set of architectural models that we selected is meant to demonstrate that our retargeting technique is not limited to simple regular structures, but can, and is designed to, handle complex 3D architectures exhibiting irregular structures. Structure preservation as well as the variety of geometry variations are both visually evident in all the results. Note that we do not only stretch, but also contract, e.g., as shown by the smallest resulting models in Figure 9. In theory, our technique can generate an infinite set of different variations by altering the box hierarchy, behavior attributes, and the combination of retargeting dimensions and scales.

Our algorithm was run on a machine with a 2.8GHz CPU and 3GB RAM. Given a box hierarchy and user-defined behavior attributes, our automatic structure analysis including alignment detection and sequence construction takes about one minute to complete for the models shown. For the largest model containing 300 boxes, the analysis took only 1.5 minutes. All the dynamic retargeting results were obtained in real time, with one interaction taking less than a second to complete. The most time-consuming task is the manual definition of the box hierarchies. With the application tool we created, the user first creates a partitioning of an input 3D model into boxes and then groups the boxes in a bottom-up fashion to define the box hierarchy. Typically, less than 80 boxes were sufficient to express, at a rather fine scale, the inner structure and semantics of an input building. The time taken in this typical case is about half an hour. Rather loosely defined box hierarchies, involving much fewer boxes, can also lead to interesting retargeting results.

After the user defines the box hierarchy, real-time interactive retargeting along different dimensions, to different sizes, and involving different such combinations, can commence. A typical modeling scenario then consists of the user modifying a few behavior attributes, waiting for up to one minute for the automatic structure analysis to complete, and then performing many more real-time interactive retargeting operations. In practice, constructing, from scratch, a digital 3D model of moderate complexity like those used in games, movies, or VR applications is always time-consuming. It typically takes an average modeler hours per model using tools such as Maya or Studio-Max. Our technique reuses the input model and allows fast creation of a large number variety of models via retargeting, taking up to few minutes per retargeted model.

8 Discussion, limitation, and future work

We present a technique for interactive structure-preserving retargeting of irregular architecture models. The semantic interpretation of the structural elements of the input, as well as their organization, are provided by the user. The core computational effort involves the decomposition of the retargeting task into the retargeting of a series of retargetable sequences to achieve interactivity. The retargeting results preserve the structural style of the input while retaining the consistency and coherence among the structural elements.

Figure 10: *A virtual community set up by the retargeted models.*

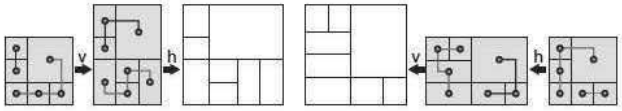

Figure 12: *Example showing non-commutativity of retargeting. Two different results with the same final size, one with horizontal (***h***) followed by vertical (***v***) retargeting, the other vice versa.*

Structural irregularity. In architectural models, irregularity does not imply a complete lack of regularity, rather, the irregularity is reflected by the presence of diverse forms of regular patterns organized in an irregular way. The multitude of regularities and their irregular organizations make it difficult to infer a unique generative model, a grammar, for example, and similarly, it induces much ambiguity when the input is retargeted. On the other hand, the presence of regularities, many of which might exist at small scales, imply strong coupling between certain structural elements. Adding to this the engineering constraints such as vertical or horizontal alignments, the retargeting problem becomes challenging.

User interaction and automated variations. The main goal of this work is to allow a casual user to quickly and easily generate many variations from an input piece. Ease of use is indeed one of the main advantages of our technique: there is solely a single operation, axial scaling, which requires no expertise. With such a simple interaction, the user can quickly generate a large number of variations; virtually any manipulation yields a valid model. Optionally, we can allow automated variations, e.g., by randomly choosing the retargeting axes and extents at each step. Figure 11 shows some results with alternate retargeting in vertical and horizontal directions; the ratio of change with respect to the current length is also randomly determined.

Figure 11: *Automated variations by randomly selecting retargeting dimensions and extents.*

Variability of results. Besides modifying the box hierarchy, the user has a few other options to vary the retargeting results. For example, a combination of horizontal and vertical retargeting steps would yield an interesting model as shown in Figure 12. Note that the two axial scaling operations are *not commutative*, a feature which increases the possibility of generating variable models. With the implementation described so far, our method does not scale well, in the sense that the more a structure is stretched the more regular it becomes, as shown in the left of Figure 13. The strengthening of the regularities is enforced since we encourage replication. This, however, can be alleviated by introducing certain probabilistic alteration of the box hierarchy to break the regularity and produce more varied results. In the right of Figure 13, we show the result produced by the same retargeting operations (as in the left) but with certain box tags altered probabilistically.

Grammar-based notation. Architectural models naturally lend themselves to grammar-based representations [Wonka et al. 2003; Mueller et al. 2007; Aliaga et al. 2007]. In this work, we deliberately do not describe our method as such, although the hierarchy of boxes can be represented in a grammar. Box replication is nothing but a simple $A \rightarrow AA$ rule, a scaling of a box is a parameter change, and a box B with n siblings can be denoted as $B \rightarrow C_1 \ldots C_n$. However, a context-free grammar is not necessarily effective under the multitude of constraints arising in our retargeting application. Our sequences can be regarded as being derived by a grammar, while being context-sensitive with respect to each other. That said, the formalism of grammar-based representations makes them attractive to study in future work.

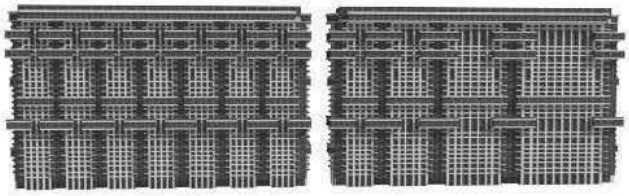

Figure 13: *When an irregular structure is overly stretched, it become regular (left). By simply altering some box tags probabilistically, less regularity and more varied retargeting results can be obtained easily (right, after the same retargeting operations).*

Limitations. The meaningfulness of our retargeting results is naturally dictated by the meaningfulness of the box hierarchy. An unnatural partitioning of the input by the user would lead to unnatural retargeting results, as shown in Figure 14 (left). On a technical level, our alignment scheme still leaves room for improvement. For example, currently the alignment is only applied to the bounding boxes and not to the salient features in the structural elements; this may lead to visible artifacts as shown in Figure 14 (right). Also, even if the input model is a watertight mesh, our algorithm is not guaranteed to produce a watertight output; possible misalignment between adjacent boxes can be a cause.

Another limitation of our method is inherent to the fact that it is developed for retargeting and not generic modeling which supports arbitrary editing. Retargeting alone does not provide the user a means to synthesize new creations by cut-and-paste or reshuffling of architectural pieces. Nevertheless, conceptually, such editing operations can be integrated with our technique; at any point, the geometry of any structural element can be altered and then retargeted. This, however, compromises the simplicity of retargeting which we advocate in our work for generating variations.

Finally, our retargeting algorithm is not based on optimizing a particular objective function; it merely computes one solution, among possibly many solutions, that would fulfill user-defined semantics and softly enforced alignment constraints.

Future work. Improving our interactive retargeting algorithm at the technical level is possible, e.g., by enhancing the height distribution and alignment schemes. Feature alignment can be considered in addition to box alignment. Defining appropriate objective functions may lead to less greedy computations, but at the expense of more costly processing. Other selection criteria for the retargetable sequences are also possible. Our current choice tends to maximize the number of hard constraints involving box compatibility and attachments. This facilitates alignment enforcement and has been shown to produce effective results with efficiency. Other choices may result in different retargeting behavior and model variations.

We regard our work as an early step in analyzing and synthesizing irregular structures. So far regular structures have been extensively researched, much more so than semi-regular structures [Chen et al. 2008]. The synthesis of semi-regular and irregular structures deserve more attention and we plan to conduct such studies in domains involving structures other than architectural models. Many automatically created model variations can be offered to a user as

Figure 14: *Results showing certain limitations of our method. Left: unnatural partitioning (bottom-left insert) of the input results in unnatural retargeting results. Right: aligning only the bounding boxes but not edge features in the structural elements can result in visible artifacts; bottom-left insert again shows the partitioning used.*

a gallery to choose from, i.e., as in a design gallery [Marks et al. 1997]. Finally, we would like to develop other interactive tools to create variations. Our motivation is to enlarge the scope of transformations while keeping interactivity and ease of use.

Acknowledgments We thank the anonymous reviewers for their valuable suggestions. This work was supported in part by NSFC (60902104, 61025012, 61003190), 863 Program (2011AA010500), CAS One Hundred Scholar Program, CAS Visiting Professorship for Senior International Scientists, CAS Fellowship for Young International Scientists, Shenzhen Science and Technology Foundation (JC201005270329A, JC201005270340A), China Postdoctoral Science Foundation (201104146), the Israel Science Foundation, Lynn and William Frankel Center for Computer Sciences and the Tuman Fund, and the Natural Sciences and Engineering Research Council of Canada (No. 611370).

References

ALIAGA, D. G., ROSEN, P. A., AND BEKINS, D. R. 2007. Style grammars for interactive visualization of architecture. *IEEE Trans. Vis. & Comp. Graphics 13*, 4, 786–797.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. on Graph 29*, 4, 104:1–10.

CABRAL, M., LEFEBVRE, S., DACHSBACHER, C., AND DRETTAKIS, G. 2009. Structure preserving reshape for textured architectural scenes. *Computer Graphics Forum (Eurographics) 28*, 2, 469–480.

CHAUDHURI, S., AND KOLTUN, V. 2010. Data-driven suggestions for creativity support in 3D modeling. *ACM Trans. on Graph 29*, 6, 183:1–10.

CHEN, G., ESCH, G., WONKA, P., MULLER, P., AND ZHANG, E. 2008. Interactive procedural street modeling. *ACM Trans. on Graph 27*, 3, 103:1–10.

GAL, R., SORKINE, O., MITRA, N., AND COHEN-OR, D. 2009. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Trans. on Graph 28*, 3, 33:1–10.

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. *ACM Trans. on Graph 27*, 5, 111:1–9.

LEFEBVRE, S., HORNUS, S., AND LASRAM, A. 2010. By-example synthesis of architectural textures. *ACM Trans. on Graph 29*, 4, 84:1–8.

LIPP, M., WONKA, P., AND WIMMER, M. 2008. Interactive visual editing of grammars for procedural architecture. *ACM Trans. on Graph 27*, 3, 102:1–10.

MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J., KANG, T., MIRTICH, B., PFISTER, H., RUML, W., RYALL, K., SEIMS, J., AND SHIEBER, S. 1997. Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proc. of SIGGRAPH*, 389–400.

MERRELL, P., AND MANOCHA, D. 2008. Continuous model synthesis. *ACM Trans. on Graph 27*, 158:1–7.

MITRA, N. J., BRONSTEIN, A., AND BRONSTEIN, M. 2010. Intrinsic regularity detection in 3d geometry. In *ECCV*, 398–410.

MUELLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Trans. on Graph 25*, 3, 614–623.

MUELLER, P., ZENG, G., WONKA, P., AND GOOL, L. V. 2007. Image-based procedural modeling of facades. *ACM Trans. on Graph 26*, 3.

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. 2008. Discovering structural regularity in 3D geometry. *ACM Trans. on Graph 27*, 3, 43:1–11.

RISSER, E., HAN, C., DAHYOT, R., AND GRINSPUN, E. 2010. Synthesizing structured image hybrids. *ACM Trans. on Graph 29*, 4, 85:1–6.

SHAMIR, A., AND SORKINE, O. 2009. Visual media retargeting. In *ACM SIGGRAPH ASIA 2009 Courses*, 11:1–11:13.

SHNEIDERMAN, B., FISCHER, G., CZERWINSKI, M., AND ET AL. 2006. Creativity support tools: Report from a u.s. national science foundation sponsored workshop. *Int. J. Hum. Comput. Interaction*, 61–77.

STAVA, O., BENES, B., MECH, R., ALIAGA, D., AND KRISTOF, P. 2010. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum (Eurographics) 29*, 2, 665–674.

WANG, Y., XU, K., LI, J., ZHANG, H., SHAMIR, A., LIU, L., CHENG, Z., AND XIONG, Y. 2011. Symmetry hierarchy of man-made objects. *Computer Graphics Forum (Eurographics) 30*, 2, 287–296.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. *Eurographics'09 State of the Art Reports EGSTAR*, 93–117.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. on Graph 22*, 3, 669–677.

WU, H., WANG, Y., FENG, K.-C., WONG, T.-T., LEE, T.-Y., AND HENG, P.-A. 2010. Resizing by symmetry-summarization. *ACM Trans. on Graph 29*, 6, 159:1–10.

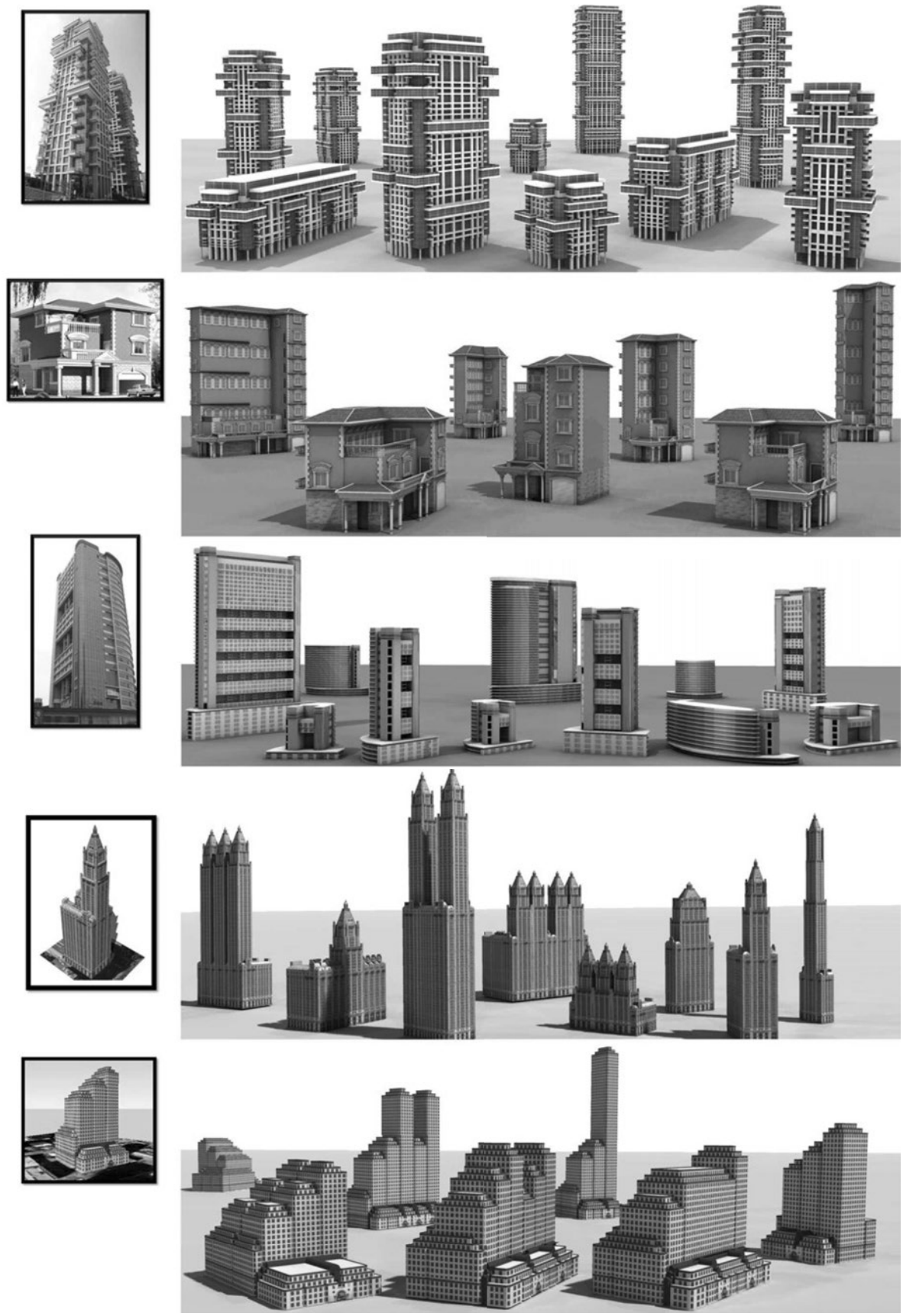

Figure 9: *Retargeting results demonstrating our method's ability to handle irregular structures and the model variety it generates. The first three inputs were modeled by an artist after real-world buildings (shown on the left) and the last two were from Google Warehouse.*