EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

SAFE: Structure-aware Facade Editing

Minh Dang Duygu Ceylan Boris Neubert Mark Pauly

École Polytechnique Fédérale de Lausanne

Figure 1: Structure-aware facade editing. Using the notion of generalized grids, our system encodes various symmetry, alignment, and hierarchy relations among the elements of a facade. During incremental editing, the user can specify different grids (shown as box abstractions) for which our system proposes new configurations. Editing progresses by selecting such grids and one of the proposed configurations (shown in red).

Abstract

Many man-made objects, in particular building facades, exhibit dominant structural relations such as symmetry and regularity. When editing these shapes, a common objective is to preserve these relations. However, often there are numerous plausible editing results that all preserve the desired structural relations of the input, creating ambiguity. We propose an interactive facade editing framework that explores this structural ambiguity. We first analyze the input in a semi-automatic manner to detect different groupings of the facade elements and the relations among them. We then provide an incremental editing process where a set of variations that preserve the detected relations in a particular grouping are generated at each step. Starting from one input example, our system can quickly generate various facade configurations.

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

1. Introduction

One of the long-standing problems in computer graphics is to provide artistic control for content creation. Modeling of shapes is not trivial because it requires both artistic talent and technical skill. The design process is time-consuming and error-prone as extensive manual processing is often needed to obtain high-quality models. To address these challenges, recent research efforts focus on the *modeling-by-example* paradigm, where the goal is to modify an existing model to create new shapes while preserving certain features of the original shape. Such a paradigm is particularly useful for modeling urban spaces, since many applications (e.g. mapping and navigation, urban design, content creation for entertainment) can benefit from a fast and easy design process.

Building facades often exhibit dominant structural relations such as symmetry and regularity. When producing new shapes by editing a given example, these relations should typically be preserved. However, this is a highly ambiguous process as there are often multiple ways to maintain the structural relations that all result in plausible output shapes. Amongst these shapes, there is no definitely correct output and the desired solution depends on the intent of the user.

Therefore, instead of committing to a particular output based on certain heuristics, it is vital to be able to efficiently navigate through alternative solutions.

In this paper, we present an interactive framework for structure-preserving editing of 2D building facades that enables exploration of the structural ambiguity during the editing process. We assume as input an ortho-rectified facade image that has been hierarchically segmented with vertical and horizontal splitting lines into rectangular sub-regions. Certain sub-regions such as the windows and doors are semantic facade elements and we preserve the arrangements of these elements during editing.

It is often desirable to edit a group of related elements together. For example, identical windows arranged in a regular grid are typically expected to behave similarly. A row of windows and the door separating them might act as a grid of nonidentical elements if grouped together, making the insertion or deletion of either of the element types possible. Often, there are multiple ways to group a set of elements and the particular grouping of interest depends on the user intent. Thus, we provide a semi-automatic framework to group the facade elements. Given a particular grouping, we support editing operations such as insertion or resizing of elements, while propagating the edits to hierarchical sub-elements.

Contributions. Our main contribution is a novel incremental editing process that exposes shape ambiguities by prompting the user with a set of alternative output shapes at each editing step. A central feature of our approach is the ability to encode the structure of a facade as a general group of elements that can be nested in a hierarchy. This avoids limitations of most existing systems that restrict editing operations to regular grids only. We evaluate our framework on building facades of varying complexity and demonstrate that a large variety of plausible output shapes can easily be created from a single input example.

Related Work. Recently, a considerable amount of shape manipulation methods have been proposed that focus on high level structural relations among the parts of a shape such as symmetry and regularity. A common strategy is to utilize a two-step analyze-and-edit approach [GSMCO09]. The analysis step focuses on discovering relevant structures and the relations among them. Once detected, these relations are preserved in the subsequent editing operations.

Symmetry driven analysis plays an important role in structure aware shape processing as symmetry is ubiquitous in most man-made objects. In the iWires system, symmetry properties of the salient feature lines of an object are preserved during free-form deformation [GSMCO09]. Bokeloh et al. [BWKS11, BWSK12] explore discrete and continuous regular patterns during shape deformation. Wu et al. [WWF*10] present an image resizing method where detected regular grids are trimmed or expanded. All of these methods focus on generating an *optimal* editing result with respect to the specific energy formulation. Although this is useful for quickly generating plausible editing results, the user needs to manipulate the relevant parameters to obtain a specific output. In case of symmetry and regularity, however, often there are multiple plausible outputs that correspond to a specific edit and we focus on exploring this variation.

Another approach to represent the structure of a shape is to encode it as a procedural production. Grammar-based modeling has demonstrated its ability to create a wide range of 3D models [PM01, WWSR03, MWH*06]. In traditional procedural modeling, the user specifies new rules or modifies the parameters of the existing rules to create a desired output. Inverse procedural modeling focuses on creating procedures from example shapes which are used to synthesize similar shapes [BWS10, ŠBM*10]. Despite the effectiveness of inverse procedural modeling in generating model variations, it is notably difficult to control. Several approaches have been proposed to guide procedural modeling towards a desired output [TLL*11, LWW08]. However, it is non-trivial to determine the preference of the users or enable them to interact with the grammar in an intuitive manner. In contrast, we provide the user with a small set of plausible shapes for each editing operation and do not require a consistent preference throughout the editing session.

In the context of urban data, semi-automatic facade parsing methods provide an alternative approach to explore the structure of facades [MWA*12]. These methods focus on decomposing a facade into a hierarchy of rectangular regions with vertical and horizontal splitting lines [MWW12]. To effectively handle irregular facade configurations, methods that decompose the input into a set of 1D sequence of elements [LCOZ*11] or facade layers [ZXJ*13] have been proposed. Lin et al. [LCOZ*11] present a retargeting framework that changes the model topology by duplicating or removing the extracted 1D sequences. Recently, Zhang et al. [ZXJ*13] introduce a method to decompose a facade into different layers by maximizing the symmetry of the resulting substructures. This decomposition then can be used for editing operations. We provide comparisons with both methods in the evaluation section (Sec 6). Finally, Lefebvre et al. [LHL10] present a fully automatic method for extracting horizontal and vertical strips from architectural textures based on self-similarities. A number of such strips are reassembled to synthesize a new texture. However, this method does not explicitly explore any structural and hierarchical relations between the facade elements that may be desired to preserve.

Several strategies have been proposed for exploration of large collections of shapes [UIM12, OLGM11]. In our framework, we limit the number of alternative outputs exposed to the user at a time by adopting an incremental editing approach. In the context of exploring facade variations, our work can be considered closest to the work of Bao et al. [BSW13]. Given a hierarchical decomposition of the input facade, this work creates variations by recursively re-

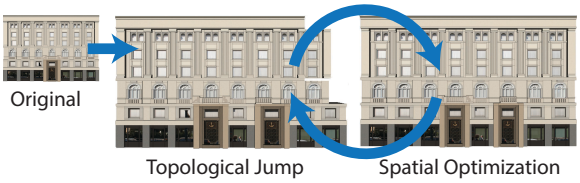


Figure 2: The proposed editing pipeline consist of two steps: The topological jump step explores structural variations of the input facade by changing the number of facade elements. Then, in the second step spatial configurations of the facade elements are optimized to generate a plausible facade.

arranging the substructures in a rectangular region. To ensure the validity of the generated facade layouts, the user needs to specify valid arrangements for each rectangular region depending on the input decomposition along with additional size and alignment constraints. In contrast, we provide an easy grouping of the facade elements independent of the input decomposition to edit them simultaneously.

2. Overview

One of the core challenges in structure-aware editing is ambiguity: there are often multiple consistent ways to maintain a set of structural relations. Simple operations, such as resizing a facade, can quickly lead to a combinatorial explosion of possible solutions. Many existing editing methods provide a single solution based on a set of heuristics that try to anticipate the intent of the user. However, the user might initially only have a vague idea of the desired output. In such a case, the final solution can be obtained in an exploratory process by iterative refinement of intermediate results. Therefore, our aim is to give access to a large space of possible solutions, while avoiding exposure to an exponential set of variations. We achieve this goal with an incremental editing process that prompts the user with a small set of variations at a time. The output is successively refined by selecting one of these variations at each step.

We distinguish between two fundamental types of modifications to a facade: *discrete* modifications that change the number of facade elements, e.g. inserting new elements or removing existing elements, and *continuous* modifications that change the size of facade elements, e.g. resizing a window. We propose an editing framework that enables such modifications in a two-step approach. Discrete modifications of the input facade are performed in the **topological jump** step, while continuous modifications are applied to the resulting facade elements in the **spatial optimization** step to compensate for the distortion introduced by the structural changes. This separation allows a stepwise exploration of structural variations while the potential variations due to continuous changes in size are ignored and expressed as constraints in the optimization. Note that only the spatially optimized facade configurations are exposed to the user.

3. Representing Spatial and Structural Relations

In this section we present the data structures that capture spatial and structural relations between facade elements. Such relations are preserved in the editing operations we present. Facade parsing algorithms provide a method to decompose a facade into smaller shapes by recursive subdivision. In the following we call this spatial subdivision structure *decomposition tree*. By introducing *parent-child* relations between facade elements, this data structure captures how changes in the size of a facade element induce changes at its parent. Besides the spatial relations, facade elements exhibit structural relations, e.g. the number of windows in one floor matches the number of windows in the second floor, which might or might not be of the same kind. We introduce an additional data structure (*facade grids*) to capture this information.

Spatial Decomposition: Decomposition Tree. Many facade parsing [MWW12, SHFH11] and element classification [RKT*12, TKS*11] algorithms result in a spatial decomposition of the input data. This decomposition is usually represented as a tree with alternating splitting directions. A node in the tree represents a *facade shape* associated with a rectangular area. The root node (representing the whole facade) is recursively subdivided into smaller rectangular shapes by splitting along the x or y directions. We store the direction of the subdivision and its relative position with respect to the size of the node (*split lines*).

The resulting spatial decomposition exhibits properties that we exploit during the optimization step to compute valid spatial configurations of new facade variations (see Sec. 5). The *total decomposition property* allows us to express the size of a node in terms of the size of its children. Specifically, if a node has x (y) splits, the width (height) of this node is equal to the sum of the widths (heights) of its children, where as the node and its children have identical heights (widths).

Several automatic or semi-automatic methods exist to define and detect splitting lines. We refer the reader to the survey on urban reconstruction for a thorough review of this topic [MWA*12]. Another source for such spatial decompositions are facade configurations resulting from shape grammars that can be naturally transformed into decomposition trees based on the grammar parsing tree of the grammar (such as [MZWG07]). We provide examples using facade decompositions resulting from both methods.

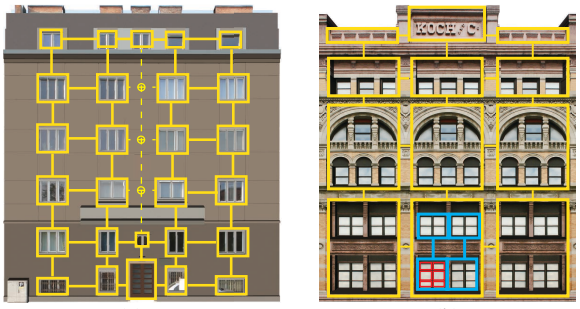
Structural Relations: General Grids. We present a data structure to capture the structure of the input facade and influence the resulting variations. Earlier attempts consider structural information in terms of symmetries, repetitions, or regularity of a model using one- or two-dimensional regular lattices (cf. [BWSK12, PMW*08]). Although these approaches can be applied to a wide range of models, they fail

to encode potential links between non-symmetric elements (e.g. non-regular element spacing) and cannot encode hierarchical relations (e.g. a regular element itself consists of a regular configuration of subelements).

Horizontal and vertical alignments are typically most important to define the structure of building facades. We encode these alignments by a grid-like data structure. Grids provide an intuitive way to specify structures in facades as well as the constraints between facade elements. Elements that are part of the same grid are meant to behave similarly under structural changes. Such relations are used in the following sections to constrain the spatial optimization.

We employ a generic definition for these grids where grid elements are nodes of the facade decomposition tree. These elements do not need to be similar, can be unevenly spaced (Fig. 3), and be part of more than one grid. A *facade grid* is defined as a group of non-overlapping facade elements, which are arranged into columns and rows. The user either manually selects elements that should be combined to a grid, which allows arbitrary elements in a grid, or nodes similar to a selected element are identified based on automatic symmetry detection methods [CML*12]. This generic definition of facade grids enables the grouping of different types of elements, and hence makes it possible to handle facades without dominant repetitions (see Fig. 13).

Given a group of facade elements, we assign each of them a unique coordinate consisting of a row and a column index. For column assignment, starting with the left-most element, we consider the next element to belong to the same column as long as we observe a vertical overlap between the elements. These elements form the first column. We repeat this process for the remaining elements to obtain additonal columns. The assignment of row indices is similar, starting with the top element. Our scheme does not require all rows or columns to have the same number of elements. In the case of missing elements we add a *phantom* element as a place holder without assigned geometry to obtain a rectangular grid configuration (see Fig. 3-a).



(a) (b)

Figure 3: (a) A general grid with different types of elements is shown. Dashed lines connect phantom elements (circles) in the middle column. (b) The grid in red is a subgrid of the blue grid, which in turn is a subgrid of the yellow grid.

Structures can be observed at different levels within a hierarchy, i.e. one structure can be contained within another structure (see Fig. 3-b). We translate this hierarchical relation into a *hierarchy of facade grids*. These hierarchical configurations are automatically assigned if all elements of a grid are included in the subtree (of the decomposition tree) of an element of another grid. The former grid is then called the subgrid of the latter one.

4. Discrete Modification – Topological Jump

The objective of our incremental editing framework is to generate plausible facade variations that preserve desired structural relations between facade elements. We enable the user to select a set of facade grids at each editing step for which the algorithm will suggest new configurations. Such selected grids are called *active grids*. When a grid is selected, our framework enables automatic selection of other grids with identical element types and counts. By selecting different active grids, the user specifies different structural relations to be preserved at each editing step and explores the resulting variations. After choosing one of the proposed variations, the user proceeds by changing the active grids or analysing further extensions of the current active grids. In this section, we describe how new facade configurations are generated by changing the number of elements in an active grid by utilizing both structural and spatial information. Note that the size of the elements in the new facade configuration are determined in the *spatial optimization* step (see Sec. 5).

Once a grid is selected to be active, our method first examines the content of the grid to determine its possible variations. Specifically, if the editing operation increases the width (height) of the facade, the unique columns (rows) as potential insertion candidates are identified. We call such unique columns (rows) the *source* columns (rows). Insertion of any of the source columns (rows) in each possible location of the active grid results in a potential variation presented to the user. In the following subsections, we describe how discrete modifications to an active grid are performed. For convenience, we only describe the case where an editing operation changes the width of a facade. Changes in the height of the facade are handled in a similar fashion.

Structure-aware Insertion Operation. Insertion of a new grid column is performed by insertion of each element in the column in a row-wise manner. Therefore, we first describe how a grid element S which we call the *source* element is inserted between two *anchor* elements A_l and A_r. Often, grid elements such as windows are most prominent elements adjoining less important non-grid elements such as walls. Thus, insertion of a new grid element requires the duplication of the surrounding content of the anchors to ensure

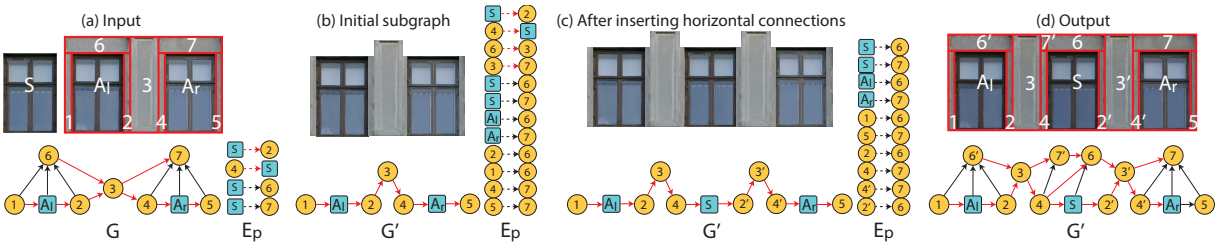


Figure 4: The source S is inserted between the anchors A_l and A_r. The neighborhood graph G' and the contents of the pending edge queue E_p is shown at each step of the insertion process. Vertical and horizontal edges are in black and red respectively.

that the source is embedded in a region similar to the neighborhood of the anchors before the insertion (see Fig. 4). Preserving such neighborhood relations requires direct access to the neighborhood information of each grid element. While the facade decomposition tree provides hierarchical decomposition links between the facade elements, it does not provide direct access to the neighborhood information as neighboring elements may be part of different subtrees depending on the order of the subdivision operations. Instead, we encode the neighboring relations between facade elements in a graph structure called the *neighborhood graph*.

A neighborhood graph $G = (N, V, H)$ is a directed graph composed of a set of nodes N where each node corresponds to a facade leaf shape in the facade decomposition tree. A vertical edge $e_v = (n_i, n_j) \in V$ (horizontal edge $e_h = (n_i, n_j) \in H$) directed from n_i to n_j connects these two nodes if the corresponding facade shapes share a vertical (horizontal) boundary and n_i is below (to the left of) n_j. The neighborhood graph can be considered as a dual structure of the facade decomposition tree which provides direct access to relative positions of the graph nodes. It is straightforward to build this graph given a decomposition tree. Conversion of a neighborhood graph G to a facade decomposition tree, on the other hand, is performed in a recursive manner. At each step of the conversion, the longest sequence of vertically (or horizontally) connected nodes $C = \{n_0, ..., n_k\}$ is extracted such that all nodes $\{n_0, ..., n_{k-1}\}$ have only one outgoing edge and all nodes $\{n_1, ..., n_k\}$ have only one incoming edge of the same type. Such a sequence of nodes, called *chain*, are collapsed to a single node and a new parent shape is added to the decomposition tree to represent the collapsed node. A chain is equivalent to a set of sibling nodes in a facade decomposition tree. Thus, if G represents a valid facade decomposition tree, it is ensured that a chain can be detected at each step. The conversion process terminates when the whole graph is collapsed to a single node which represents the root of the corresponding decomposition tree.

When inserting a source element S between the anchor elements A_l and A_r, we duplicate additional facade shapes and determine their spatial arrangement in the new facade configuration by utilizing the neighborhood graph. Specifically, we build a neighborhood graph G corresponding to the subtree of the facade decomposition tree rooted at the common parent of A_l and A_r, since this subtree contains all relevant elements. Insertion of S is then carried out by constructing a new neighborhood graph G' from G which contains S. S is embedded in G' in such a way that its neighborhood is similar to those of A_l and A_r in G. One naive approach to obtain G' is to connect the source S to all neighbors of A_l and A_r. However, this often leads to an invalid graph, i.e. a graph which does not represent a valid facade decomposition. For example, an element might end up as both right and left neighbors of another element (see Fig. 5). This observation supports the intuition that S can be embedded in a neighborhood similar to the neighborhood of both anchors only by duplicating some nodes in G. Therefore, we propose an incremental solution that takes a valid neighborhood graph as input and adds new edges one at a time while ensuring that the graph remains valid at each step. Necessary nodes are automatically duplicated during this process. Once G' is constructed, we convert it back to a decomposition subtree to replace the original subtree. Next, we describe the details of the incremental edge insertion process.

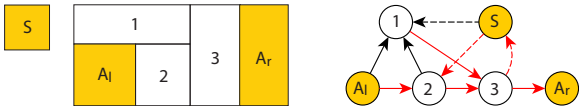


Figure 5: Connecting the source S to all the neighbors of the anchors A_l and A_r results in a conflict: the loop $(S, 2, 3)$ suggests that S is both to the left and right of 3. (Vertical and horizontal edges are shown in black and red respectively.)

Incremental Edge Insertion. Given an initial neighborhood graph G including the anchors A_l and A_r, our goal is to insert the source S between the two anchors. To achieve this goal, we incrementally construct a new neighborhood graph G' by utilizing a *pending edge queue*, E_p, consisting of the edges that need to be added to G'.

First, the edges in G that involve either A_l or A_r are added to E_p while replacing the respective anchor node with S (see Fig. 4 a). Insertion of these edges in the subsequent stages influences the neighborhood of S to be similar to the neighborhood of the anchors. We then initialize G' as the subgraph

extracted from G consisting of the longest sequence of horizontally connected nodes including A_l and A_r. Any edge of G that is not included in this subgraph is added to E_p (see Fig. 4 b). Once E_p is initialized, edges in E_p are inserted to G' incrementally. Inserting an edge requires an update of the neighborhood information to ensure a valid facade configuration at any point during the process. Please note that a new edge in E_p becomes available for insertion when at least one of the nodes it connects is present in the current graph.

Edge insertion starts with the pending horizontal edges, processing those involving S first. Next, the pending vertical edges are processed. If a node is required to be vertically connected to a sequence of horizontal nodes in G', all such vertical edges are inserted simultaneously (e.g. $e(S,6)$ and $e(2',6)$ in Fig. 4 c are inserted at once). The algorithm continues by processing the remaining pending edges in a similar manner until E_p is empty.

Once a pending edge is selected, its insertion is performed based on whether it involves one or two nodes present in the current graph G'. Assume a horizontal edge $e(n_i,n_j)$ is selected that connects a new node n_i to an existing node n_j. Intuitively, insertion of such an edge is equivalent to introducing a vertical split in the facade shape of n_j to create a tiny sub-shape and replacing this sub-shape with the facade shape of n_i. In other words, our goal is to correctly embed the new node n_i between the nodes already present in G' with coherent horizontal and vertical relations. Thus, we first extract all incoming horizontal edges of n_j and relink them to n_i (in Fig. 6, $e(1,8)$ and $e(2,8)$ are relinked to $e(1,N)$ and $e(2,N)$). If such a relinked edge was present in the original graph G, we add it to E_p so that this neighborhood information is not lost. We next establish the vertical neighborhood relations of n_i by connecting it to one of the above and one of the below neighbors of n_j. Specifically, we extract a set of nodes M_{below} that are connected to n_j with vertical incoming edges and a set of nodes M_{above} that are connected to n_j with vertical outgoing edges. Since all nodes in M_{below} (or M_{above}) are connected to n_j vertically, there is a horizontal path connecting all of them. We connect n_i vertically to the leftmost node along such a path (in Fig. 6, vertical edges $e(N,4)$ and $e(6,N)$ are established).

The described procedure so far allows to insert an edge with a new node into G' and update the present edges to ensure a valid configuration. If at any point of this process, an edge which has been previously added to G' from E_p is required to be pushed back to E_p, this insertion is discarded and we undo any update in G' triggered by this insertion. This ensures that no edge is inserted to G' and pushed back to E_p continuously.

If the selected pending edge connects two existing nodes n_i and n_j in the current graph G', its insertion might result in a conflict. In order to avoid such conflicts, we duplicate one of the edge nodes n_i and insert the pending edge between the new duplicated node n_i' and n_j instead. This edge now connects a new node n_i' to the existing node n_j and can be added as described before. When a node is duplicated, all the edges of the original node are copied and added to E_p. Note that if both nodes of a pending edge have already been duplicated, we discard it to ensure that a node is duplicated at most once.

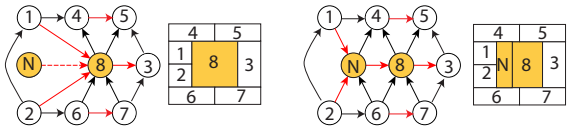


Figure 6: The edge $e(N,8)$ is inserted to the current neighborhood graph. Vertical and horizontal edges are shown in black and red respectively.

Coupled Insertion in Multiple Rows. In cases where anchor elements in different rows of a facade grid have the same common parent, performing the column insertion row-by-row will result in multiple duplications of certain facade shapes. To avoid such a scenario, we perform the element insertion in these rows in a coupled fashion. We first find the subtree rooted at the common parent of the identified rows and construct the corresponding neighborhood graph G. Intuitively, we divide G into multiple subgraphs where each subgraph consists of a single grid row. We insert source elements into corresponding subgraphs and combine the updated subgraphs. In more detail, we extract a set of subgraphs $G^s = \{G_1,...,G_m\}$ where G_k represents the longest sequence of horizontally connected nodes in G including the anchor elements in the corresponding row r_k. We then initialize a common pending edge queue E_p with any edge that is not included in any subgraph in G^s. For each source element to be inserted, we also copy the edges from the corresponding anchors and add to E_p. We then label the edges in E_p to denote an order of insertion to the extracted subgraphs. Specifically, starting from the top row G_1, we define a node set N_k as the set of nodes in G_k and the unlabeled nodes that are above any node in G_{k+1}. Edges that connect any two nodes in N_k are labeled as l_k. At the end of this grouping, a set of edges remain unlabeled which are the vertical edges later used to connect the updated subgraphs in G^s. Next, starting from the first row, the incremental edge insertion process described before is performed by updating the corresponding subgraph of the row. The only notable difference is that when updating the graph G_k, only the edges labeled l_k are processed.

Once all rows are processed, E_p contains only the unlabeled edges. We use these edges to combine the subgraphs in G^s into a common graph G'. A pair of vertical edges are used to connect two subgraphs if they are not *crossing*. Edges $e(x_1,y_1)$ and $e(x_2,y_2)$ are defined to be crossing if x_1 is located to the left of x_2 and y_1 is to the right of y_2. Once all updated subgraphs are combined into G', it is converted to a decomposition tree to replace the original subtree.

5. Continuous Modification – Spatial Optimization

In the topological jump step of our framework, new facade elements are added to the spatial decomposition tree and the respective facade grids. This process violates the total decomposition property (as described in Sec. 3) and thus requires an update of the size of the resulting nodes (see Fig. 2). The naive option to compute valid sizes for the nodes is to propagate the change due to additional elements up to the root node, effectively increasing the size to accommodate space for the new elements. Further, changes in inner nodes need to be propagated down to the children, e.g. by evenly distributing the size increment. While this re-establishes the total decomposition property of the spatial data structure, the results often violate aesthetic properties such as alignment and coherent size of similar elements (see Fig. 7). To address this issue, we propose an optimization scheme that minimizes the deviation from the original size of the elements while respecting additional constraints relating the size and the alignment of the elements in facade grids. We formulate this optimization as a *quadratic programming* problem.

Figure 7: Simple propagation of changes along the decomposition tree (a) breaks alignments between facade elements, which can be preserved by our spatial optimization (b).

Our optimization process shares some similarities with the method of Bao et al. [BSW13] with one fundamental difference. We optimize for the size of all the facade shapes at once while Bao et al. recursively solves for a sequence of 1D layout problems. The size of the already processed facade shapes impose certain layout constraints for the subsequent stages of their algorithm. Thus, backtracking is necessary if a solution cannot be found at a certain step.

At the end of each editing step, we compute the new width and height of the facade shapes by setting up two independent optimization problems respectively. In the following, we describe how the new shape widths are computed. Computation of the shape heights is performed similarly by interchanging width and height and x and y-splits.

Quadratic Programming. When computing new sizes of facade shapes, our goal is to preserve the original sizes as close as possible. Thus, we define the quadratic objective $E(\mathbf{x}) = \|\mathbf{W}(\mathbf{x}-\mathbf{x}^*)\|^2$, where $\mathbf{x}$ and $\mathbf{x}^*$ respectively denote a vector of new and original widths of the facade leaf shapes and $\mathbf{W}$ is a diagonal weight matrix. We weight the changes in shape sizes by the inverse of their areas to allow larger shapes to deviate more. We normalize the shape areas by dividing by the total facade area and cap the weights at 10.0. We minimize $E(\mathbf{x})$ for positive $\mathbf{x}$ subject to hierarchy constraints C_h, alignment constraints C_a and symmetry constraints C_s as described next:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & E(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} > \mathbf{0}, \mathbf{C}_h\mathbf{x} = \mathbf{0}, \mathbf{C}_a\mathbf{x} = \mathbf{d}_a, \mathbf{C}_s\mathbf{x} = \mathbf{0} \end{aligned} \tag{1}$$

Additional constraints specified by the user such as the target size and location of a shape can be integrated into Eqn. 1 to provide additional interaction possibilities.

Hierarchy Constraints. We specify a hierarchical link between the size of a facade shape and its children. For each leaf shape, we define a binary vector α with one non-zero entry corresponding to the width, w, of the leaf in $\mathbf{x}$, such that $w = \alpha^T\mathbf{x}$. The width of an internal node is then obtained from the binary vectors α of its children. Specifically, if S has an x split, we obtain $\alpha_s = \sum_i \alpha_{c_i}$ and $w_s = \alpha_s^T\mathbf{x}$. In case of a y split, however, any two children c_i and c_j have the same width, $\alpha_{c_i}^T\mathbf{x} = \alpha_{c_j}^T\mathbf{x}$, equal to the width of S. Thus, we set $\alpha_s = \alpha_{c_0}$, and for any pair of children (c_i, c_{i+1}), we represent the constraint $(\alpha_{c_i} - \alpha_{c_{i+1}})^T\mathbf{x} = 0$ as a row of $\mathbf{C}_h$.

Shape Alignment. Facade elements that belong to the same grid need to preserve their relative positions during the editing process. Therefore, we define constraints relating the pairwise distances between neighboring grid elements. To quantify these alignment constraints, we first associate each grid element S with an anchor point A_s. In horizontal edits, for each column, we define three sets of points consisting of (i) the centers, (ii) midpoints of the left edge, and (iii) midpoints of the right edge of the grid elements in the column. We use the point set with minimal variance in terms of x-coordinates as the anchor point set. The shape alignment constraints preserve the distance between the x-coordinates of these anchor points.

The x-coordinate, a_s, of the anchor point A_s of a shape S can be expressed in terms of the x-coordinate, l_s, of the left edge of S:

$$a_s = l_s + \varepsilon w_s, \tag{2}$$

where $\varepsilon = 0$ if A_s is the midpoint of the left edge, $\varepsilon = 0.5$ if it is the shape center, and $\varepsilon = 1$ if it is the midpoint of the right edge. Further, l_s can be computed as:

$$l_s = l_p + \sum_{s_i \in \mathbb{S}_S^-} w_{s_i}, \tag{3}$$

where l_p is the x-coordinate of the left edge of the parent of S, $\mathbb{S}_S^-$ is the set of siblings in the facade decomposition tree located to the left of S. w_{s_i} denotes the width of such sibling nodes. Setting $l_{root} = 0$, l_s is computed as a linear combination of the widths of the leaf nodes $\mathbf{x}$: $l_s = \beta_s^T\mathbf{x}$. By plugging this expression and $w_{s_i} = \alpha_{s_i}^T\mathbf{x}$ into Eqn. 3 we

(a) Input (b) Selected variation in step 1 (c) Selected variation in step 2 (d) Selected variation in step 3

Figure 8: By activating different grids (colored) at each editing step, interesting facade variations can be generated.

obtain $\beta_s = \beta_p + \sum_{s_i \in \mathbb{S}_s^-} \alpha_{s_i}$. Finally, from Eqn. 2, we have $a_s = \beta_s^T \mathbf{x} + \varepsilon \alpha_s^T \mathbf{x}$.

We then express the alignment constraints between any two consecutive grid element S_i and S_{i+1} in a column in terms of the x coordinates of the corresponding anchor points: $a_{s_i} - a_{s_{i+1}} = d_{i,i+1}$, where $d_{i,i+1}$ is the original horizontal distance between A_{s_i} and $A_{s_{i+1}}$. This translates into $(\beta_{s_i} + \varepsilon\alpha_{s_i} - \beta_{s_{i+1}} - \varepsilon\alpha_{s_{i+1}})^T \mathbf{x} = d_{i,i+1}$ and is represented as a row in the system $\mathbf{C}_a\mathbf{x} = \mathbf{d}_a$.

Shape Symmetry. Certain facade elements, either manually indicated by the user or detected by automatic symmetry detection methods, are desired to have the same size. We formulate this symmetry constraint between two such shapes S_i and S_j using their α vectors: $(\alpha_{s_i} - \alpha_{s_j})^T \mathbf{x} = 0$.

Variable Reduction. The widths of all the facade leaf shapes are considered as variables in our optimization. However, we can reduce the number of variables by analyzing the constrained nodes of the facade decomposition tree. Specifically, we collapse the subtree rooted at a node n_i if none of the remaining nodes in the subtree is involved in a constraint (see Fig. 9, yellow subtrees). In this case, we optimize only for the width of n_i and distribute the change in this width evenly to its descendants in a recursive manner. Further reduction of the variables is possible considering the split direction of the nodes. If a node n_i is subdivided into a set of leaf nodes by a y split, all of its children have a width equal to the width of n_i. In this case, we collapse the subtree rooted at n_i and express any constraint involving its children in terms of the width of n_i (see Fig. 9, green subtrees). We perform such node collapses iteratively until no further collapse is possible reducing the number of variables significantly.

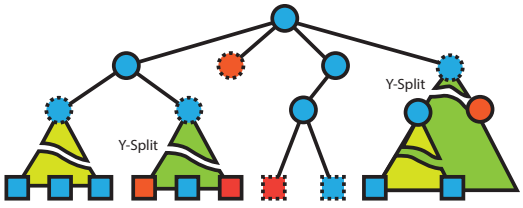
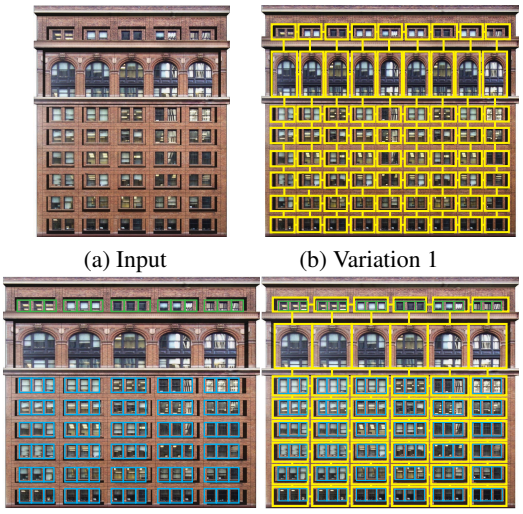


Figure 9: Variable reduction process: Yellow subtrees are collapsed as they do not contain any node involved in a constraint (shown in red). Green subtrees are collapsed as the parent node is subdivided into a set of leaf nodes with y splits. The leaf nodes in the final tree are shown as dotted and are used as variables in the optimization process.

6. Evaluation

In this section we discuss the main features of our framework for several example editing scenarios. For a larger collection of editing results, please refer to the video and the supplementary material. Note that manually specifying a general grid used in these examples takes around 20 seconds (see the accompanying video). In the following results we show the final state of the active grids selected at each editing step to highlight the changes to a facade (Fig. 8, 10-13).

Data sets. The input to our framework is a hierarchically decomposed ortho-rectified facade image. Such a decomposition can be obtained manually or in a semi-automatic manner. We evaluate our method on facade images decomposed by the semi-automatic approach of Musialski et al. [MWW12]. We also create a shape grammar for rectified facade images taken from online repositories using CityEngine [Esr13] and convert this grammar into a decomposition tree. We next describe the capabilities of our framework on these input facades of varying structural complexity.



(a) Input (b) Variation 1

(c) Variation 2 (d) Variation 3

Figure 10: Given the hierarchical grid structure, one can modify the main grid (yellow), the sub-grids (green and cyan), or both.

Results. The input to our system is a hierarchical decomposition of a facade. However, the editing operations are not restricted by the splitting levels of this decomposition. Specifically, we allow facade elements that do not belong to the same parent element to be grouped into a grid (see Fig. 8, row 2). This enables to jointly edit elements across different levels of the hierarchy and eliminates the dependency on the input decomposition.

Our editing framework is incremental where at each editing step, the elements of active grids are removed or duplicated. The elements of the remaining grids, on the other hand, are only scaled to respect the new size of the facade. By activating different grids at each step of this process, interesting variations of a facade can be generated, which is difficult otherwise (see Fig. 8).

An important feature of our framework is the support for hierarchical grids. The facade shown in Fig. 10 consists of a grid of windows each of which is composed of a sub-grid. When this facade is resized horizontally, one can change the inner sub-grids of the window frames, add a new column of windows or perform both actions simultaneously. The user can explore these options by activating or deactivating the sub-grids at each editing step.

In some facades, certain elements such as a door can span multiple rows (or columns) as shown in Fig. 11. When the elements in the rows spanned by the door are grouped into a grid, we assign the door to one of these rows resulting in fewer grid elements in the remaining row(s). In order to obtain a complete grid in such a case, we add a *phantom element* to the rows with fewer elements. The phantom element

(a) Input (b) Editing output

Figure 11: The door of the facade, that spans the bottom two rows, is assigned to the lower row, a phantom element (cyan) is added to the upper row.

(a) Input (b) Editing output

Figure 12: Different element types are grouped in a common active grid (in green) and edited together.

acts as a placeholder in the grid and can be duplicated or removed based on the editing operation. On the other hand, if an element is required to be inserted between two grid elements of which at least one is phantom, we simply discard this insertion (see Fig. 8, last row).

A unique feature of our approach is the notion of general grids that enable the grouping of different element types with varying spacings. For example, in Fig. 12 one large grid consisting of different window types has been built. When the facade is resized in the vertical direction to trigger the addition of a row, any row of the grid can be duplicated. Moreover, when a new column is inserted, the corresponding window types are duplicated in each row.

Even though the concept of general grids is capable of handling irregularity, some facades as shown in Fig. 13 exhibit no dominant grid structure. Grouping all the elements in these facades results in a grid with many phantom elements making it difficult to edit. However, editing can be performed by activating multiple small grids and preserving these local structures simultaneously.

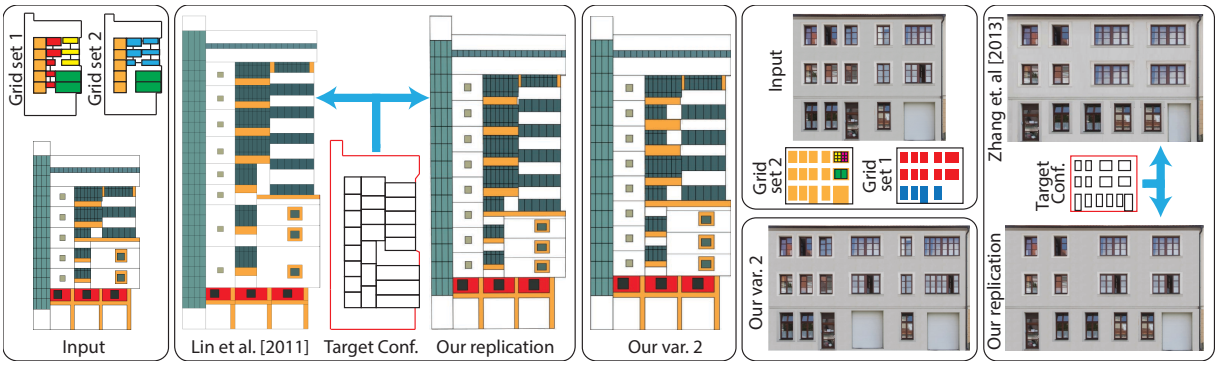
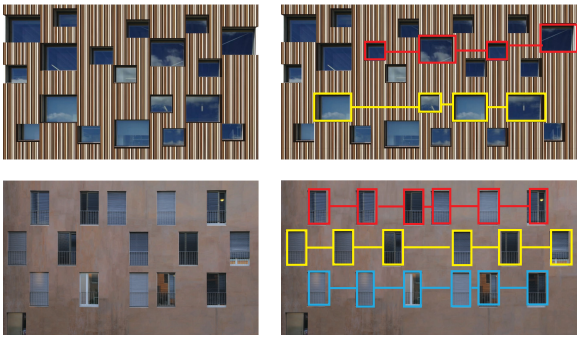


Figure 14: Editing results in comparison to [LCOZ*11] (left) and [ZXJ*13] (right). Our replications of the target configurations shown in box abstraction, are generated by activating *grid set 1*. Additional outputs are produced by activating *grid set 2*.



(a) Input (b) Editing output

Figure 13: Multiple small grids (colored) are activated to edit the facades that lack a dominant grid structure.

Comparison. In this section, we provide comparisons with the methods of Lin et al. [LCOZ*11] and Zhang et al. [ZXJ*13]. We evaluate our method on two examples taken from these works and generate a replica of the same target facade configuration. We also generate additional variations to emphasize the differences between the methods (see Fig. 14). In the retargeting framework of Lin et al. [LCOZ*11], reshuffling of elements in an extracted sequence is not supported. In contrast, we allow rows and columns to be inserted at each possible position in an active grid (see how small and large balconies in the red sequence are arranged in *variation 2* in Fig. 14). A major advantage of the layered decomposition proposed by Zhang et al. [ZXJ*13] is the ability to move a layered element to an arbitrary new position. We achieve similar editing results by duplicating such an element at the target position and deleting the original. However, directly moving around of elements might be more intuitive for the users. Interleaving grids of different element types can be edited with the method of Zhang et al. [ZXJ*13] only if they have been decomposed as different layers. In that case, the user also needs to specify additional constraints to position such grids during editing. In our method, however, such elements can be grouped into a general grid and edited easily. We also support editing of hierarchical sub-structures (see how small and big window types are grouped together in *variation 2* in Fig. 14 and edited together with their substructures).

Limitations. Even though our system is able to generate many variations of an input facade, there are several limitations we would like to address in future work. The input to our system is a decomposition of a facade with vertical and horizontal splitting lines. Such a decomposition fails to provide a tight partitioning for other polygonal and arched shapes. We represent these shapes with their axis-aligned bounding boxes. Integrating non-axis aligned splits and curved shapes will provide more plausible results for certain architecture styles with dominant curved structures.

In this work, we do not focus on smart texture synthesis, the texture of an edited facade is synthesized by duplicating, cropping, and scaling the original texture. In presence of occlusions and strong lighting variations, this results in visual artifacts. In the future, we would like to incorporate more advanced texture synthesis methods to reduce such artifacts.

7. Conclusion and Future Work

We have presented a novel structure-preserving facade editing framework. Our approach gives direct access to shape ambiguities during editing, while avoiding the combinatorial explosion of potential editing results through an incremental process. The concept of generalized grids allows capturing both regular and irregular structures, including hierarchical configurations, which greatly expands the set of possible shape manipulations. We believe that this approach provides a novel perspective on structure-aware editing that has potential for many other geometric design tasks.

There are several interesting avenues for future work. The input to our system is a hierarchical decomposition of a

facade. Conceptually it is possible to extend our method to handle layered decompositions as proposed by Zhang et al. [ZXJ*13] by enabling the grouping of elements at different layers into a common grid.

Although we only focus on 2D facade editing, our system can be extended to 3D by recursively decomposing 3D buildings using axis-aligned cutting planes similar to [LCOZ*11]. Our current framework explores structural ambiguity by supporting incremental edits and providing alternatives at each step in a sequential manner. However, defining a *structural similarity* measure between variations of an input facade and recursively clustering these variations based on this measure is an interesting research direction. Such a clustering approach, especially if built upon some properties of human perception system such as Gestalt rules [NSX*11], will enable to explore the design space more effectively. Finally, we believe extending the principles of combining continuous and discrete changes as well as neighborhood synthesis to 3D models with dominant axis-aligned structures, such as furniture, is possible and interesting.

Acknowledgement: We thank the anonymous reviewers for their valuable comments. We thank Przemyslaw Musialski, Fan Bao and Kathleen Tuite for providing some of the input examples. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 257453: COSYM.

References

[BSW13] BAO F., SCHWARZ M., WONKA P.: Procedural facade variations from a single layout. *ACM TOG 32*, 1 (Feb. 2013), 8:1–8:13. 2, 7

[BWKS11] BOKELOH M., WAND M., KOLTUN V., SEIDEL H.-P.: Pattern-aware shape deformation using sliding dockers. *ACM TOG (SIGGRAPH) 30*, 6 (2011), 123:1–123:10. 2

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM TOG (SIGGRAPH) 29*, 4 (2010), 104:1–104:10. 2

[BWSK12] BOKELOH M., WAND M., SEIDEL H.-P., KOLTUN V.: An algebraic model for parameterized shape editing. *ACM TOG (SIGGRAPH)* (2012), 78:1–78:10. 2, 3

[CML*12] CEYLAN D., MITRA N. J., LI H., WEISE T., PAULY M.: Factored facade acquisition using symmetric line arrangements. *CGF 31* (May 2012), 671–680. 4

[Esr13] ESRI: Cityengine, 3D modeling software for urban environments, 2013. URL: http://www.esri.com/software/cityengine. 8

[GSMCO09] GAL R., SORKINE O., MITRA N. J., COHEN-OR D.: iWIRES: An analyze-and-edit approach to shape manipulation. *ACM TOG (SIGGRAPH) 28*, 3 (2009), 33:1–33:10. 2

[LCOZ*11] LIN J., COHEN-OR D., ZHANG H. R., LIANG C., SHARF A., DEUSSEN O., CHEN B.: Structure-preserving retargeting of irregular 3D architecture. *ACM TOG (SIGGRAPH Asia) 30*, 6 (2011), 183:1–183:10. 2, 10, 11

[LHL10] LEFEBVRE S., HORNUS S., LASRAM A.: By-example synthesis of architectural textures. *ACM TOG (SIGGRAPH) 29*, 4 (July 2010), 84:1–84:8. 2

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *ACM TOG (SIGGRAPH) 27*, 3 (2008), 102:1–102:10. 2

[MWA*12] MUSIALSKI P., WONKA P., ALIAGA D. G., WIMMER M., VAN GOOL L., PURGATHOFER W.: A Survey of Urban Reconstruction. In *EUROGRAPHICS STAR* (2012), pp. 1–28. 2, 3

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., GOOL L. V.: Procedural modeling of buildings. *ACM TOG (SIGGRAPH) 25*, 3 (2006), 614–623. 2

[MWW12] MUSIALSKI P., WIMMER M., WONKA P.: Interactive coherence-based façade modeling. *CGF 31*, 2 (May 2012), 661–670. 2, 3, 8

[MZWG07] MÜLLER P., ZENG G., WONKA P., GOOL L. V.: Image-based procedural modeling of facades. *ACM TOG (SIGGRAPH) 26*, 3 (2007). 3

[NSX*11] NAN L., SHARF A., XIE K., WONG T.-T., DEUSSEN O., COHEN-OR D., CHEN B.: Conjoining gestalt rules for abstraction of architectural drawings. *ACM TOG (SIGGRAPH Asia) 30*, 6 (2011), 185:1–185:10. 11

[OLGM11] OVSJANIKOV M., LI W., GUIBAS L., MITRA N. J.: Exploration of continuous variability in collections of 3d shapes. *ACM TOG (SIGGRAPH) 30*, 4 (July 2011), 33:1–33:10. 2

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. *ACM TOG (SIGGRAPH)* (2001), 301–308. 2

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L.: Discovering structural regularity in 3D geometry. *ACM TOG (SIGGRAPH) 27*, 3 (2008), 43:1–43:11. 3

[RKT*12] RIEMENSCHNEIDER H., KRISPEL U., THALLER W., DONOSER M., HAVEMANN S., FELLNER D., BISCHOF H.: Irregular lattices for complex shape grammar facade parsing. In *CVPR* (2012), pp. 1640–1647. 3

[ŠBM*10] ŠT'AVA O., BENEŠ B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of l-systems. *CGF 29*, 2 (2010), 665–674. 2

[SHFH11] SHEN C.-H., HUANG S.-S., FU H., HU S.-M.: Adaptive partitioning of urban facades. *ACM TOG (SIGGRAPH Asia 30*, 6 (2011), 184:1–184:9. 3

[TKS*11] TEBOUL O., KOKKINOS I., SIMON L., KOUTSOURAKIS P., PARAGIOS N.: Shape grammar parsing via reinforcement learning. In *CVPR* (2011), pp. 2273–2280. 3

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM TOG 30*, 2 (Apr. 2011), 11:1–11:14. 2

[UIM12] UMETANI N., IGARASHI T., MITRA N. J.: Guided exploration of physically valid shapes for furniture design. *ACM TOG (SIGGRAPH) 31*, 4 (July 2012), 86:1–86:11. 2

[WWF*10] WU H., WANG Y.-S., FENG K.-C., WONG T.-T., LEE T.-Y., HENG P.-A.: Resizing by symmetry-summarization. *ACM TOG (SIGGRAPH Asia) 29*, 6 (2010), 159:1–159:10. 2

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM TOG (SIGGRAPH) 22*, 3 (2003), 669–677. 2

[ZXJ*13] ZHANG H., XU K., JIANG W., LIN J., COHEN-OR D., CHEN B.: Layered analysis of irregular facades via symmetry maximization. *ACM TOG (SIGGRAPH) 32*, 4 (2013), 121:1–121:13. 2, 10, 11