EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

Layer-Based Procedural Design of Façades

Martin Ilčík Przemyslaw Musialski Thomas Auzinger Michael Wimmer

Vienna University of Technology, Austria

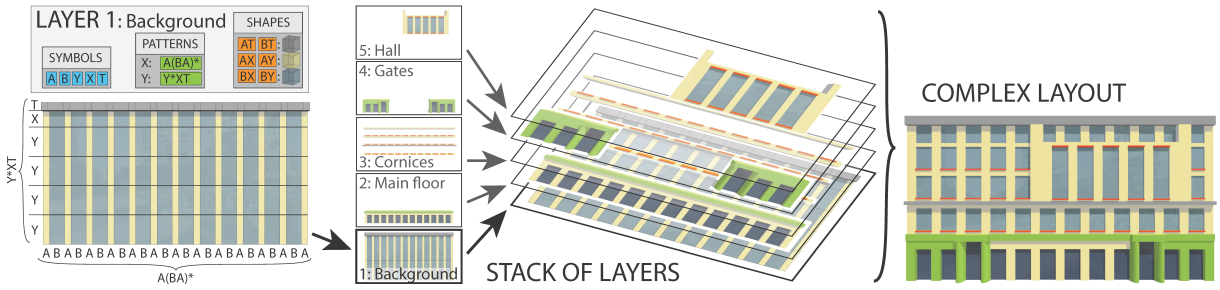

Figure 1: Complex façade layouts *(right)*, which are hard to model in conventional shape grammars, can be intuitively designed with the help of our framework by decomposing the layout into multiple layers *(center)*. We ensure sensible combinations of the various layers via an automatic process. This allows the user to model simple patterns on each layer *(left)* instead of one complex arrangement.

Abstract

We present a novel procedural framework for interactively modeling building façades. Common procedural approaches, such as shape grammars, assume that building façades are organized in a tree structure, while in practice this is often not the case. Consequently, the complexity of their layout description becomes unmanageable for interactive editing. In contrast, we obtain a façade by composing multiple overlapping layers, where each layer contains a single rectilinear grid of façade elements described by two simple generator patterns. This way, the design process becomes more intuitive and the editing effort for complex layouts is significantly reduced. To achieve this, we present a method for the automated merging of different layers in the form of a mixed discrete and continuous optimization problem. Finally, we provide several modeling examples and a comparison to shape grammars in order to highlight the advantages of our method when designing realistic building façades.

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

1 Introduction

Procedural modeling is a successful technique for quickly modeling architectural content. It is highly versatile and allows specifying a large variety of arrangements of architectural elementa, including stochastic variations. Models created by such methods can exhibit a high degree of detail and realism. However, the main disadvantage of procedural techniques is that the design space can be very large and thus difficult to explore. In particular, organizing, editing and combining a large quantity of procedural rules is still a complex and daunting task. It can also be hard to exert fine-grained control over the appearance of the generated models. Furthermore, in the context of building façades, there are structures that cannot be easily expressed in a hierarchical rule set as employed by existing solutions. This may be seen in Figure 1, left, which is *not* a trivial grid-like arrangement of elements, but exhibits a more complex appearance.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

Contribution. In order to be able to model such arrangements, we introduce a novel layer-based editing approach, where the user specifies the structure independently for individual layers. Each layer consists of two simple generator patterns that span a rectilinear grid (cf. Figure 1, middle and right). However, naively merging the layers leads to corrupt results (Figure 4). To solve this problem, our main technical contribution is an automated alignment method to satisfy inter-layer and intra-layer relations and constraints.

The use of layers is an essential technique in graphical design and can be found in many major 2D and 3D content-creation tools. Apart from an intuitive way to organize scene elements, they often provide means for interactions between them. The application of the layer paradigm to façades brings about a number of interesting modeling abstractions that were not possible with traditional tools that organize the data in tree structures.

Inspired by these observations, our framework enables convenient modeling of variable façade designs. Our approach is not inverse, i.e., the façades are designed from scratch. The user can interact with the model *(i)* by adding or removing layers, *(ii)* by changing the patterns of a layer, *(iii)* by specifying the size and appearance of façade elements. The layer-based approach reduces the complexity of managing procedural rulebases, and also allows direct control of individual façade elements. The automation of the actual merging process of procedural layers is the main computational challenge. We present a solution that is accomplished by a mixed discrete and continuous optimization. In particular, the contributions of this paper are:

- A novel approach for modeling of façades by providing multiple layers instead of a single top-down tree structure. This allows the user to introduce changes on different layers without the necessity to alter the whole tree. Moreover, allowing layers to overlap gives the ability to model features that do not fit into a hierarchy, like patterns that intersect other structures (see Figure 1).
- We provide a novel technical solution based on combined discrete and continuous optimization in order to combine all layers into a single consistent layout. It identifies valid alignments of interacting elements across layers to deliver plausible façade models.

In Section 6, we compare our approach to classical shape-grammar modeling and show that it is easy to control and allows modeling a large variety of façades.

2 Related Work

Procedural generation of architectural models is an approach to model urban environments by means of rules defined by production systems like shape grammars [Sti75] or set grammars [WWSR03]. We refer to the report by Vanegas et al. [VAW*10] for a comprehensive review.

Several works tried to make shape grammars better suitable for modeling: Mueller et al. [PM01, MWH*06] introduced a procedural generation framework called CGA-shape; Finkenzeller [Fin08] introduced a method for procedural modeling of and modifying detailed building façades that adapt to geometry automatically; and Hohmann et al. [HHKF10] provided a procedural shape grammar to describe façades within a GML-framework. These systems are quite complex, difficult to control, and it is laborious to design the grammar rules. Lipp et al. [LWW08] introduced an interactive editing system allowing the creation of rulebases without text-file editing and proposed first concepts of local control. G^2 by Krecklau et al. [KPK10] improved the local control and generalized geometry of non-terminals. Later, they focused on high-quality rendering of façades for large city scenes using GPU evluation of split grammars [KBK13].

Procedural methods are used in the field of reconstrucion as well. Aliaga et al. [ARB07] automatically extract simple patterns from façade models and enable easy exploration of novel variations. Further solutions for façade reconstruction from images utilize either interactive modeling [XFT*08, MWW12] or inverse procedural modeling [WFP10, SHFH11]. Musialski et al. [MWA*13] provide a comprehensive survey of the topic.

Different concepts of layering have been recently utilized, but only within the reconstruction domain. Zhang et al. [ZXJ*13] proposed a framework that maximizes the symmetry of pre-segmented façades by dividing the façade into different layers of symmetric substructures. Fan et al. [FMLW14] partition façade elements into interleaving grids. Driven by supervised learning, their method automatically completes a partially occluded façade. Variations and retargeting of façades from pre-segmented images is mostly based on hierarchical decompositions [LCOZ*11, BSW13]. The most recent method by Dang et al. [DCNP14] introduces topological jumps in the hierarchy.

The remainder of the paper describes our framework in Section 3 and an example modeling process in Section 4. After the technical description of our alignment solver in Section 5, we present results in Section 6.

3 Layer-Based Modeling

First, we give an overview of the basic concepts that are employed in our modeling framework. Please refer to Figure 1 for an accompanying illustration and Figure 2 for a summary and the interrelations of the introduced concepts.

3.1 Concepts

Canvas: Similar to general image editing, the modeling process starts with an empty *canvas*. By setting the size of this rectangular region, the user determines the extent of the façade.

Layer: A stack of *layers* fills the canvas. Each layer is a rectilinear 2D grid of shapes (cf. Figure 1, center) that covers the canvas entirely.

Symbol: Each layer is generated by an outer product of abstract 1D elements (in horizontal and vertical direction),

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

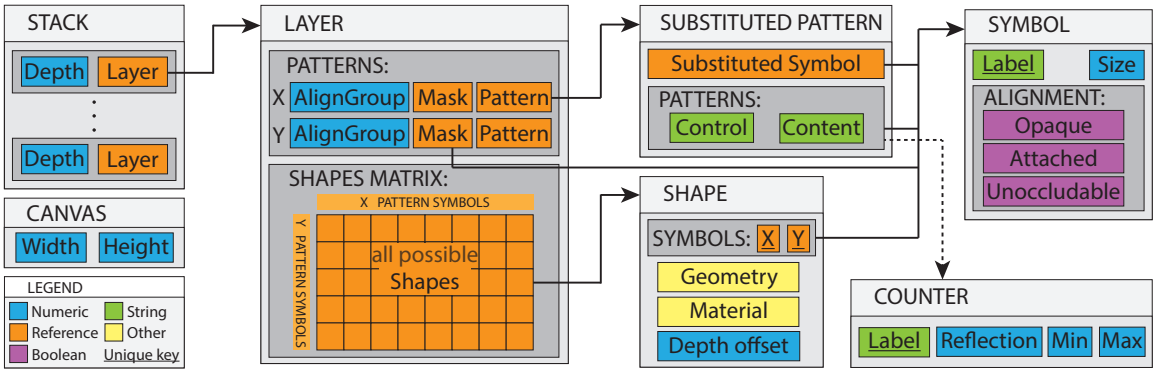

Figure 2: A simplified overview with the most important properties of components introduced in Section 3. The user interface provides access to all fields listed in this diagram.

called *symbols*, each identified by a *label* string. The user sets the *preferred size* of the symbol, providing the geometric width or height of the associated shapes. Symbols can be marked as *transparent* and used as placeholders.

Shape: A cell in the layer, uniquely determined by a pair of symbols, is called *shape*. These are the geometric building blocks of the façade and are equipped with 3D meshes and mapped with materials (i.e., shading, textures). Furthermore, a depth offset allows specifying their position along the normal of the canvas. Transparent symbols generate void gaps instead of shapes.

As shown by the example in Figure 1, the abstract layout description of the background layer is given by the outer product of the symbols $\{A,B\}$ (in the horizontal direction) and the symbols $\{Y,X,T\}$ (in the vertical direction), generating the shapes AY, AX, AT, BY, BX, and BT. In general, a layer with m unique horizontal and n unique vertical symbols exhibits mn shapes). The associated preferred width (resp. height) of shape BY is given by the preferred size of symbol B (resp. Y).

3.2 Procedural Layers

Words. The façade elements (i.e., shapes) of each layer are given by the outer product of a horizontal and a vertical sequence of symbols, denoted as *words*. In order to provide the user with an easy and efficient way to specify suitable words, we use formal languages [HMU07].

Patterns. In most cases, façade layouts exhibit a high degree of repetitions and reflective symmetries. These can be efficiently encoded with *patterns*. Each pattern generates a set of words that the user identified as suitable candidates for the façade layout. We specify patterns by regular expressions, using the common operations:

$\|$	alternation	$A\|B$	$\rightarrow$	$\{A, B\}$
$\cdot$	concatenation	$A\|B \cdot C$	$\rightarrow$	$\{AC, BC\}$
$()$	grouping	$A\|(B \cdot C)$	$\rightarrow$	$\{A, BC\}$

Regular languages are too simple to encode complex symmetries on façades, while fully context-sensitive languages are needlessly complex for this task. Therefore, we start with a regular language and extend it with the required functionality by adding a *parametric counting* operation

$$A^k = \underbrace{AA\ldots A}_{k \text{ times}},$$

where $k \in \mathbb{Z}$ is a globally defined parameter. A negative k yields a reflection of the base using one of the reflection modes (see supplementary material). We denote a reflected instance of a symbol B as $\mathfrak{B}$. For $k = 0$, the empty word ε is obtained. Nesting several operations of the same type is not supported. Note that by generating $a^k b^k c^k$, the concept of patterns is neither regular nor context-free [Jaf78]. Several works already studied pattern languages [Ang80] and regular expressions with exponentiation [MS72] and counting [Gel10].

Factorization of Dimensions. As a fundamental aspect of our modeling framework, we restrict the content of each layer to be defined *separately and independently* for the horizontal and vertical direction. Thus, the content of each layer is given by the outer product of the words that are specified in horizontal and vertical directions, which essentially defines a *grid* of design elements (cf. Figure 1, left). Such a composition has several advantages, both from a computational perspective, as the combination of different layers can be computed for each direction separately, as well as from a user perspective, as the complexity of the layout description is significantly reduced and unintended results can be edited for each direction independently.

Stack of Layers. Similar to the well-established paradigm in general image editing, layers can be stacked using a user-specified depth value. The resulting ordered set is denoted as the *stack of layers*. The depth ordering of the layers plays an essential role in the design process, since layers 'closer to the user' occlude layers below. We infer valid combinations of layers from their relative ordering (e.g., a column should

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

not cover a window, unless explicitly allowed), as well as visibility information for the final model (e.g., a window fully covered by a wall can be omitted).

3.3 Substituted Patterns

The structure of a façade often needs to be refined during the design process. Instead of editing existing patterns, we provide a mechanism to substitute parts of them with new content. A *substituted pattern* is given as a pair of two patterns, e.g.,

$$xm \tag{1}$$

$$m := A(BA)^k \tag{2}$$

where we differentiate between *control pattern* (1) and *content pattern* (2). All instances of the substituted symbol in (1) are replaced by a single word produced by (2), i.e.,

$$x\underline{A},\, x\underline{ABA},\, x\underline{ABABA},\, \ldots \tag{3}$$

Unsized Symbol. A symbol that is exclusively used in control patterns (x in previous examples) does not require a user-specified preferred size. It is referred to as *unsized*. During the layer-combination phase, our proposed alignment process automatically determines suitable sizes for such symbols. Unsized symbols that remain unsubstituted are typically used as wildcards to match arbitrarily large portions of the canvas without changing the patterns themselves. In Eq. (3), x aligns content to the right. In Eq. (6), two instances of y are used to center the content.

3.4 Transparency and Masking

Unsized symbols are meant to be used as invisible placeholders to improve the alignment of the substituted content. Therefore, we implicitly set all unsubstituted letters in a control pattern to be *transparent*. As an example, in words generated by

$$yBmAm^{-1}By \tag{4}$$

$$m := (AB)^k, \tag{5}$$

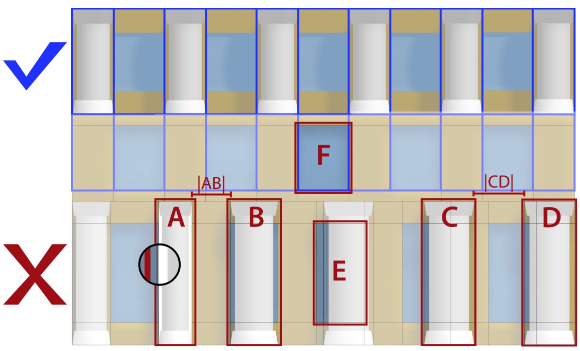

Figure 3: Flattening the stack of layers without alignment constraints may produce various types of errors. We demonstrate them using a background layer and a front layer with columns. Errors marked by red letters are explained in Section 3.5. The top floor shows the correct blue alignment.

only the underlined content is *not* a transparent placeholder:

$$yB\underline{AB}A\underline{\reflectbox{B}\reflectbox{A}}By,\, yB\underline{ABAB}A\underline{\reflectbox{B}\reflectbox{A}\reflectbox{B}\reflectbox{A}}By,\, \ldots \tag{6}$$

Transparency is a fundamental element in determining the alignment of symbols in the optimization phase (Section 5). It controls the visibility of façade elements on the pattern-level by hiding all shapes produced by transparent letters in all layers where the respective pattern is used.

On the other hand, tools to control the visibility locally for each layer are necessary for efficient design. In our framework we provide such functionality by introducing *layer masks*, which override the visibility of shapes in the respective layer. To specify a layer mask, the user marks a subset of symbols used in the layer to be visible; shapes *not* generated by visible symbols become masked and thus hidden. Since masking controls visibility on the layer-level, a different mask may be applied to the same pattern in each layer. The alignment of façade elements is only affected by the transparency of symbols, but not by layer masks.

3.5 Alignment

Our layered design concept follows a simple philosophy: Let the user specify and manipulate layers independently of each other, while our framework automatically integrates shapes from all layers to a single mesh. In Figure 3, a foreground layer with columns is directly combined with a regular background. However, such a naïve combination of layers results in multiple problems:

- **Misalignment (extrinsic)**: Column C is not aligned with the background layer elements.
- **Inconsistent sizing**: Column B is wider than column A.
- **Inconsistent placement**: Distances between neighboring columns vary, e.g. $|AB| \neq |CD|$.
- **Misalignment (intrinsic)**: Ground floor and first floor windows are not aligned (e.g., E with F), despite being produced by the same pattern.

Therefore, relations between the layers must be taken into account to achieve a proper *alignment* before merging them. Our framework automatically creates an axis-aligned grid in the xy-plane and distributes the content of layers into its cells. Elements of a cell are then constrained to snap to its borders, enforcing an alignment across layers. The computation of such alignments is described in Section 5.

Alignment Qualifiers. For a feasible layer-based façade design, it is paramount to consider alignment semantics in a way that is both robust to conflicting layout specifications and that can be intuitively manipulated by the user. Otherwise, different failures may occur (cf. Figure 3):

- **Occlusion**: Column B completely occludes a window, which is generally not intended.
- **Holes**: Column A is missing a corresponding background, which creates an unintended hole.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

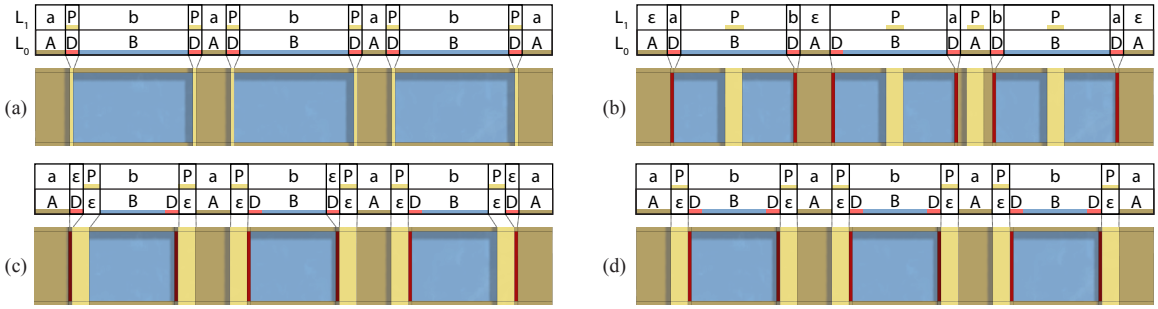

Figure 4: Overview of the alignment qualifiers. Above the façade layouts, the associated words of the two involved patterns are given: $A\,(DBDA)^m$ for layer L_0 and $a\,(dbda)^n$, $d := P$ for L_1. By default, layers may occlude the underlying ones, e.g., P occludes D (a). After marking D as *unoccludable*, P aligns with the next best option B (b). Marking P as *fully occluding* aligns it in-between elements of L_0 (c), and marking B as *attached* keeps its spatial relationship with the neighboring Ds intact (d).

Common design requirements are to keep certain elements unoccluded, or to fit them visually between content of an underlying layer. Figure 4a shows a practical example where automatic alignment based only on placement and sizing of symbols interferes with the user's intentions. To this end, we provide for each symbol three Boolean *alignment qualifiers* to control its behavior (see Figure 4b-d).

- **Opaque symbols** generate shapes which always cover a whole alignment grid cell. Therefore, opaque symbols can be placed in cells lacking content from underlying layers without a risk of introducing unintended holes in the façade. Non-opaque symbols that, e.g., represent statues, require a fully opaque content to be placed behind them.

- **Attached symbols** do not allow the creation of holes between themselves and their neighboring symbols, e.g., doors are usually attached to the door frame around them.

- **Unoccludable symbols** do not permit overlying symbols to occlude them. The alignment process has to create suitable holes in the overlying layout to permit this. Façade windows are usually defined this way.

Alignment Groups. Façade designs often contain intentional misalignments (e.g., the first floor of the Hotel in the supplementary material). We support modeling such layouts by the introduction of *alignment groups*. Assigned to the horizontal and vertical pattern of each layer separately, only patterns in the same group are aligned. This allows, for example, the replacement of whole floors of a façade with a different layout.

Synchronization. By default, all instances of a symbol/counter/shape share the same attributes. The size of a symbol is the same in all patterns, and in all alignment groups, the same holds for the value of a counter. The appearance of shapes and visibility of symbols may be overridden by patterns or layers.

4 Design Session Example

We demonstrate the layer-based workflow using a very simple synthetic design idea. A more elaborate example of a non-trivial building is presented in the supplementary material. In a typical session the user builds a stack of layers. Each new layer requires the following steps:

1. Create new patterns, if necessary
2. Set attributes for new symbols and counters
3. Create a layer, assign patterns, set layer attributes
4. Set global shape attributes or override them locally

Residential House. In this very simple example, we show a part of a residential house being designed. Please follow Figure 5 for input details. A single layer would be sufficient to represent this façade layout. However, fragmentation of the building structure into patterns as simple as possible is beneficial for later design changes. The user starts with an empty canvas and sets the size to 6×3 units.

Basic Layer. The basic layer will contain only repeating windows and walls in the horizontal direction. Let the symbol A represent a wall and B a window. The user creates a repetitive pattern B_x by entering the string `A(BA)^k`. There is no value given for the counter k. Our framework automatically chooses the count of BA repetitions which fits the canvas size the best. The building should have two floors – a ground floor P with windows and a lowered first floor Q. Thus, the basic vertical pattern B_y is given only by PQ. After entering both patterns, the preferred size for each symbol is set. Note that B_x does not fit the given canvas size for any integer k. Our framework is robust enough to handle such imprecise input automatically.

The first layer $Layer_0$ is created with B_x assigned as the horizontal and B_y as the vertical pattern. The solver described in Section 5 evaluates the patterns and finds a suitable façade layout. All shapes are brown cubes by default. The user changes their properties to match the original design idea. Windows represented by BQ are only textured.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

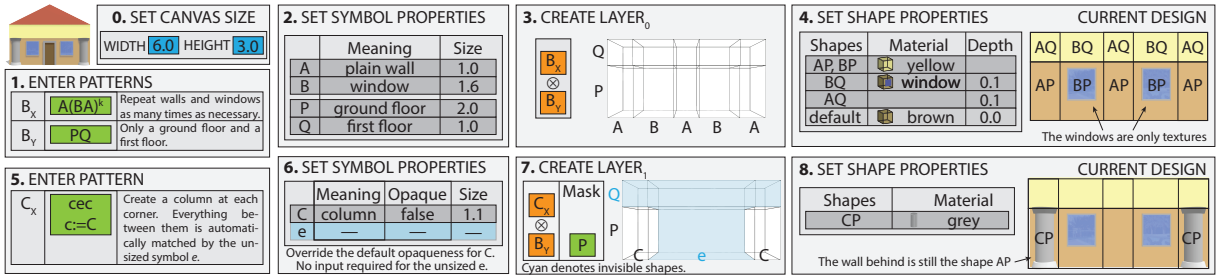

Figure 5: This example demonstrates the nine steps of user interaction required to create the first two layers for a residential house (reference design top left). The first layer contains the basic structure. Columns are added in the second layer. The modeling process is described in Section 4. For the remaining design steps please refer to supplementary material.

Columns Layer. Columns should be placed at the corners of the house, the content in the middle can be arbitrary. The user creates a new pattern C_x as cec with $c := C$. The unsized symbol e is a wildcard that represents any content from other layers. Only the preferred size of C has to be set, as no input is required for unsized symbols. The second layer is created with C_x as the horizontal pattern, while B_y is reused as vertical pattern. Since the user wants to add columns only to the ground floor, she sets a layer mask setting only P to visible. Thus, only CP and Ce stay visible. Moreover, e is an unsubstituted symbol from the control pattern, i.e., any shape produced by e will be transparent as well. Ultimately, only the CP shapes are visible in $Layer_1$. Finally, the user sets the geometry of CP to be a grey column.

For a more complex example of a design session with a non-trivial façade, please refer to the supplementary material. Advantages of our method and a critical comparison with other approaches are presented in Section 6.

5 Façade Layout Solver

User interaction in our framework is focused on the design of single layers, while their merging to determine a final façade layout is solved automatically. In this section, we give details on this solver. The user input to the solver is summarized in Figure 2. Mainly, the user defines:

- canvas width and height
- stack of layers, each with a vertical and horizontal pattern
- properties for the symbols used in the patterns, in particular the preferred size and the opaqueness

Given this input, the goal of the solver is to find a *layout* that best respects the user's input. A layout consists of two items: first, a so-called *candidate*, which is formed by instantiating a single word for each layer from its associated pattern, and second, an *alignment* of the layers, which is defined as a partition of the words on each layer such that the partition boundaries on each layer are at identical geometric positions (in other words, the partition boundaries line up). A concrete layout will usually not exactly respect the preferred symbol sizes, and thus the solver will try to minimize the overall deviation of the symbol sizes from the given values, which constitutes the goal function. In principle, there can be a large number of partitions in the layout, however, larger partition numbers can lead to higher deviations from the goal. On the other hand, at a minimum, each transition between a transparent and a non-transparent symbol will introduce a partition boundary. Our solver has three steps:

Candidate generation: In the first step, layout candidates are generated by instantiating the patterns on each layer so that the resulting words potentially fit the canvas size.

Alignment: In the second stage, starting with a candidate from the first step, possible partitions of the candidate are evaluated to find the best one with respect to the goal function. This step also takes into account a number of constraints given by alignment qualifiers in order to assure that only meaningful alignments are produced.

Sizing and placement of shapes: The third step selects the best layout produced by the first two steps and applies a continuous solver to obtain the final size and position for each symbol on each layer.

The solver is applied for the horizontal and vertical direction independently, then an outer product gives a rectangular grid of shapes, generating the final façade mesh.

The combinatorial complexity of the first two steps is very high. Therefore, the solution space is explored in a greedy fashion, featuring specialized heuristics to assess the expected quality of the potential solution. Furthermore, several individual steps make use of an error tolerance μ in order to make the optimization robust to minor variations of the symbol sizes. μ is then increased until a solution is found. In the following, each step is described in more detail. Figure 6 shows a simple example of the pipeline.

5.1 Candidate Generation

The goal of this solver stage is to generate a set of *candidates* for the alignment stage. A candidate is a tuple $(w_1, \ldots, w_m)$, where w_i is one word (instance) from the pattern for layer i, and $i = 1$ denotes the uppermost layer.

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

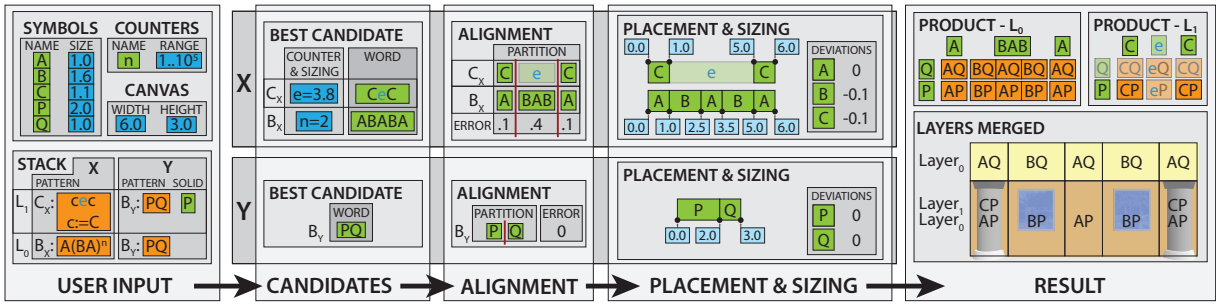

Figure 6: The solver pipeline is presented in a highly simplified way using the example from Section 4 and Figure 5. Cyan boxes and red lines denote results of the respective optimization stage.

Enumeration. For a particular pattern, variation is possible through alternation and through counting operators. We gather the different instances of a pattern by systematically enumerating the different value combinations of counters and alternation choices in lexicographical order. For patterns shared between layers, this is only done once.

Pruning by Canvas Size. Only words which fit into the canvas are relevant. This is checked using the preferred symbol sizes and a given tolerance:

$$l_i \leq \sum_i w_i[k].size \leq \text{canvasSize} + \mu, \tag{7}$$

where $w_i[k]$ denotes the kth symbol in word w_i. While intuitively, a lower bound of $l_i = \text{canvasSize} - \mu$ should apply, we note here that our system allows lower layers to "make space" for opaque symbols of upper layers. Thus, the lower bound of lower layers must account for the potential insertion of empty space by upper layers. Concretely,

$$l_i = \text{canvasSize} - \sum_{j=1}^{i-1} \sum_k \left(w_j[k].size * w_j[k].opaque \right).$$

Further examples are given in the supplementary.

Sizing Unsized Symbols. To evaluate Eq. (7), the size of all symbols needs to be known. If an enumerated word contains unsized symbols, we create several differently sized words by uniformly sampling the sizes of unsized symbols so that the resulting word size respects Eq. (7). Figure 7 gives an example of such counting and sizing operations.

Candidate Generation. Having obtained words that respect Eq. (7) for each layer, we generate candidates by combining these words into tuples and then pass them to the next stage of the solver. In practice, we do this candidate by candidate, and only pass further candidates if a solution is not found. In the supplementary material we describe how *pattern analysis* can significantly reduce the complexity of counting and sizing.

5.2 Alignment

In this solver stage, potential alignments of a given candidate $(w_i)_{i=1}^m$ are formed by partitioning the words of the candidate into an equal number of substrings. We denote this as an $m \times n$ matrix A, where for each row i the elements $a_{i,j}$ represent the substrings of word w_i, i.e., $w_i = a_{i,1} \ldots a_{i,n}$. The following shows possible alignments of the candidate $(CeC, ABABA)$:

$$\begin{pmatrix} Ce & C \\ ABA & BA \end{pmatrix} \quad \begin{pmatrix} C & e & C \\ \varepsilon & ABABA & \varepsilon \end{pmatrix} \quad \begin{pmatrix} C & e & C \\ A & BAB & A \end{pmatrix} \tag{8}$$

Note that in our framework, a substring can also be empty, denoted as ε. This represents the case that a lower layer "makes space" for a symbol in an upper layer.

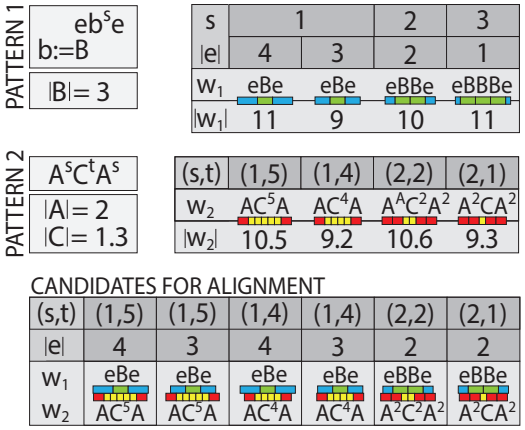

Figure 7: Counting and sizing options for two patterns and the resulting candidates. A canvas size of 10 and error tolerance $\mu = 1$ implies the word size limit $9 \leq |w| \leq 11$.

Alignment Error. An alignment means that geometrically, all elements in a column need to be the same size so that the partition boundaries are aligned. To achieve this, symbols need to be resized. This leads to deviations from the specified sizes, given by the *column error* ω_j and the *alignment error* $\Omega = \sum \omega_j$ for the whole matrix. The column error basically measures the maximum disagreement of substrings in a column:

$$\omega_j = \left| \max_i \sum_l \left(a_{i,j}[l].size \right) - \min_i \sum_l \left(a_{i,j}[l].size \right) \right| \tag{9}$$

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

Combinatorial Optimization. Finding the optimal alignment for a given candidate is a combinatorial optimization problem. We observe that an alignment can be generated iteratively by a state machine: in each step, the state machine consumes a substring for each word to form a column and leaves the remaining string for the next iteration. The rightmost alignment in Eq. (8) can be obtained in 3 steps:

$$\begin{matrix} CeC \\ ABABA \end{matrix} \rightarrow \begin{pmatrix} \boldsymbol{C} \\ \boldsymbol{A} \end{pmatrix} \begin{matrix} eC \\ BABA \end{matrix} \qquad \begin{matrix} eC \\ BABA \end{matrix} \rightarrow \begin{pmatrix} \boldsymbol{e} \\ \boldsymbol{BAB} \end{pmatrix} \begin{matrix} C \\ A \end{matrix} \qquad \begin{matrix} C \\ A \end{matrix} \rightarrow \begin{pmatrix} \boldsymbol{C} \\ \boldsymbol{A} \end{pmatrix} \begin{matrix} \varepsilon \\ \varepsilon \end{matrix}$$

The possible decisions the state machine can make at each step can be represented by a weighted graph, and we will show that finding the best alignment for a candidate is equivalent to finding the shortest path through it.

To construct this graph, we start with a simpler state machine, an m-head one-way finite state machine [Ros66], which recognizes exactly the given tuple of words, and consumes only one symbol at a time (a proper definition is given in the supplementary material). The set of states is made up of all suffix combinations of the input tuple:

$$Q = \left\{ (Q_i)_{i=1}^{m} \mid Q_i \text{ is suffix of } w_i \right\},$$

and each state has m outgoing edges, each representing the consumption of one symbol of one word w_i. The corresponding transition graph is an m-dimensional square lattice with a single source and sink node. Figure 8 shows the transition graph of the previous example with edges in orange.

The state machine we are looking for needs to be able to consume multiple symbols in each word. This can be achieved by building the *transitive closure* of the transition graph, which we call *alignment decision network* (ADN). Each edge in the ADN represents a decision on which symbols to consume in each word, and thus corresponds to a column, and each path from the initial state to the final state gives one alignment matrix. Consequently, the set of all alignments for a candidate $(w_i)_{i=1}^{m}$ is given by all traversals of its corresponding ADN. Figure 8 shows the edges added by the transitive closure in blue, and one particular traversal in red.

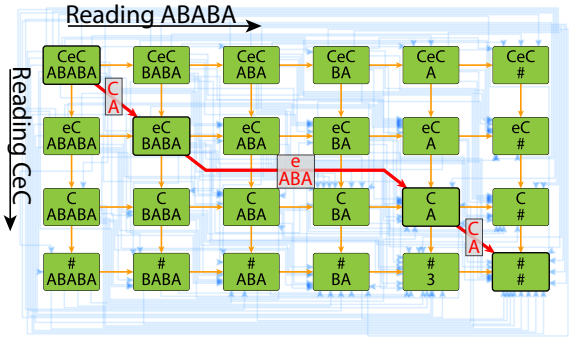

Figure 8: Alignment decision network for $(CeC, ABABA)$. The optimal path is marked red, transitions of the automaton are orange, blue arcs were added by the transitive closure.

Cost Function and Algorithm. In order to rank the possible traversals, the edges of the ADN are weighted by the column error ω_j. Thus, the edge weights in a path accumulate to Ω, the alignment error, and the best alignment corresponds to the shortest path in the ADN.

Unfortunately, the number of outgoing edges per node is exponential in the number of symbols per word ($O(n^m)$ for n symbols per word and m words). Therefore, breadth-first algorithms like A* [HNR68] are infeasible. Instead, we employ the informed version of iterative deepening with a transportation table for visited nodes [RM94]. This algorithm operates in depth-first instead of breadth-first order, thus a global ranking by edge weights is not required. Instead, each node builds a local priority queue of outgoing edge weights when visited. However, even the evaluation of the weights of all outgoing edges is too costly. Instead, we observe that the ADN describes a partial order, which allows us to apply a dynamic-programming approach, reducing the time to compute edge weights to $O(n \log(m))$ for a node.

To break ties when two outgoing edges have the same weight ω_j, we prefer the edge with the lower total number of symbols in the respective column to favor higher partition numbers.

Constraints. Apart from the column errors, the solver also evaluates a number of constraints that may disallow edges. The following holds for each alignment column j:

- A fully opaque element must exist in each column
  $\exists i \ (\forall k \, a_{i,j}[k] \text{ is opaque})$
- Transparent symbols must be separated from others
  $\forall i \ (\exists k \, a_{i,j}[k] \text{ is transparent}) \Rightarrow (\forall k \, a_{i,j}[k] \text{ is transparent})$
- Only opaque symbols may cover unoccludable symbols
  $\exists i,k \, a_{i,j}[k] \text{ is unoccludable} \Rightarrow$
  $\forall h > i: \ (a_{h,j} = \varepsilon) \vee (\forall k \, a_{h,j}[k] \text{ is transparent or opaque})$

and for each pair of subsequent columns j and $j+1$

- No "insertions" (empty string) for attached neighbors
  $(\exists i \, a_{i,j}[last] \text{ is attached}) \Rightarrow a_{i,j+1} \neq \varepsilon$
  $(\exists i \, a_{i,j+1}[first] \text{ is attached}) \Rightarrow a_{i,j} \neq \varepsilon$

In particular, the second constraint implies that transitions between transparent symbols and non-transparent ones should always form an alignment boundary. This is motivated by the fact that these boundaries typically constitute the design intent of the user when using layers. The distribution of transparent symbols determines a minimal set of columns for a candidate.

Transparent Splits. In practice, we encountered many situations where an alignment as defined above is not possible. Let us add a third word to the previous example: $(fAf, CeC, ABABA)$. Note that transparent symbols are those which are *not* underlined. The best options for alignment are:

	$\begin{pmatrix} f & A & f \\ C & e & C \\ A & BAB & A \end{pmatrix}_{\Sigma}$				$\begin{pmatrix} \varepsilon & f & A & f & \varepsilon \\ C & \varepsilon & e & \varepsilon & C \\ A & B & A & B & A \end{pmatrix}_{\Sigma}$						$\begin{pmatrix} \varepsilon & fAf & \varepsilon \\ C & e & C \\ A & BAB & A \end{pmatrix}_{\Sigma}$			
Size	5	8	5	18	2	5	8	5	2	20	2	12	2	16
Error	3	6	3	15	0	2	6	2	0	10	0	4	0	4

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

None of them results in a proper alignment. The first accumulates an error larger than the façade size, the second alignment spans too much in size, and the third one is invalid due to transparent and solid symbols mixed in a single alignment matrix entry. In the final layout, symbols from the control pattern are invisible, thus we can *split* them into smaller parts where necessary. This operation can be done during the alignment or as a preprocessing step. The optimal alignment contains the split symbols with $\text{size}(f_1) = \text{size}(e_2) = 2, \text{size}(f_2) = \text{size}(e_1) = 3$.

$$\begin{pmatrix} f_1 & f_2 & \underline{A} & f_2 & f_1 \\ \underline{C} & e_1 & e_2 & e_1 & \underline{C} \\ \underline{A} & \underline{B} & \underline{A} & \underline{B} & \underline{A} \end{pmatrix}_{\Sigma}$$

Size	2	3	2	3	2	12
Error	0	0	0	0	0	0

Heuristics and Optimizations. Façades often contain reflective symmetries. This motivated us to switch to a bidirectional variant of the shortest-path search algorithm. Another important observation is that several substituted patterns often use the same control patterns, and the control words are usually considerably shorter than the full words after the substitution. Hence, prealignment of the control patterns is mostly cheap since the words are fewer and shorter. The information can be then used to reduce the complexity for alignment of substituted words to almost constant time. The cost function is evaluated only for the edges inside of the corresponding prealigned column.

5.3 Sizing and Placement of Shapes

The alignment A obtained in the previous step is a discrete approximation of the final façade layout. Before combining the solutions for the X and Y directions, the façade elements, represented by symbols arranged in A, have to be distributed in the continuous 1D geometric domain of the façade in the respective direction. Therefore, we need to distinguish between each occurrence of a symbol and assign it a dedicated placement information. We define a *letter* to be an instance of a symbol with additional attributes s and t, the lower and upper bound of the interval assigned to the letter in the geometric domain. Letters inherit all symbol properties as given in Figure 2. The goal of this optimization stage is to find the values of s and t for all letters in $A_{i,j}$. The deviation of the letter size $(t-s)$ from the preferred symbol size should be kept minimal.

We solve the problem using a constrained system of linear equations. In particular we are looking for a solution vector which approximates a solution of an overdetermined linear system. This problem is known as least-squares with linear equality and inequality constraints. In fact it is the linear version of a convex quadratic programming problem, and we solve it using the active-set algorithm [Bjö88]:

$$\min_x \|Ax-b\|_2 \quad \text{subject to } Bx = d \text{ and } Cx \leqq e,$$

where x denotes the vector of interleaved s and t values for all letters.

Continuous Objective Function. The quality of the solution is measured by the quadratic error produced by the symbols when they deviate from the user-given preferred size. Each symbol should be counted only once, thus, in a preprocessing step we store an arbitrarily selected "prototype" letter for each symbol in a new property p. The resulting objective function is:

$$\min \sum_{i,j,k} \left(\left(a_{i,j}[k].t - a_{i,j}[k].s \right) - a_{i,j}[k].size \right)^2$$
$$\text{for} \quad \left(a_{i,j}[k] \neq \varepsilon \right) \wedge \left(a_{i,j}[k].p = a_{i,j}[k] \right)$$

Epsilons are excluded from the objective function as they fit to any required size.

Basic Constraints. The distribution of letters must follow some basic principles:

- Let s be the lower and t the upper bound: $a_{i,j}[k].s < a_{i,j}[k].t$
- Stitch the letters of $a_{i,j}$ in correct order: $\forall k > 0\ a_{i,j}[k-1].t = a_{i,j}[k].s$
- Stitch the letters at column borders in correct order: $\forall j > 0\ a_{i,j-1}[\text{last}].t = a_{i,j}[0]$
- Always fill the whole façade: $a_{i,0}[0] = 0 \wedge a_{i\,\text{last}}[\text{last}] = \text{canvasSize}$

Alignment Constraints. So far, no synchronization exists to enforce equal sizing of all instances of a symbol. Using the reference to a prototype p, it is added by:

- $\forall a_{i,j}[k] \neq \varepsilon : \quad a_{i,j}[k].t - a_{i,j}[k].s = a_{i,j}[k].p.t - a_{i,j}[k].p.s$

The alignment imposes a further constraint for each column of the alignment matrix which aligns its first and last letters in all rows to the same position:

- $\forall i > 0 \ \left(a_{i,j}[0].s = a_{0,j}[0].s \wedge a_{i,j}[\text{last}].t = a_{0,j}[\text{last}].t \right)$

This constraint indicates that all rows of any column j have to be sized equally.

Cross Group Constraints. So far we have discussed the optimization of patterns from a single alignment group. If the user assigns a pattern to several alignment groups, the alignment is performed separately for each group, leading to a set of alignment matrices. Letters from all alignment matrices are inserted into the linear system independently, but additional constraints are then introduced to make the sizing of symbol instances identical across alignment groups.

6 Results and Discussion

Our framework provides a means for efficiently modeling a large variety of façades. In Figure 10 we present selected layouts designed in our framework, which are inspired by real buildings of different architectural styles and periods. Results of retargeting and structure modifications are presented in the second row of the Figure. The computation time for our method was measured on a desktop PC with an AMD A8-3850 2.9 GHz processor and 8 GB RAM.

The most complex model with 26 layers is the Tonhalle in

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

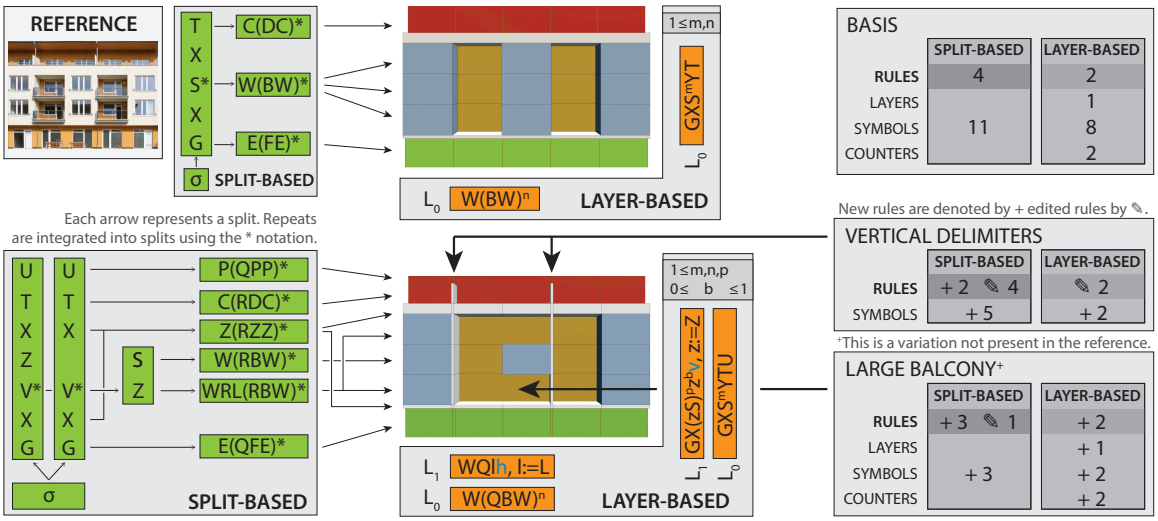

Figure 9: For a comparison of our layer-based method with the split-based CGA-Shape grammar, we construct a rough partition of the reference building (top-left). Colors of the blocks indicate content types: windows – blue, balconies – brown, terraces – red and shops – green. Adding vertical separators and large balconies with our layer-based approach requires less than a half of the split operations. Figure 10 shows some variations of the final façade.

Zürich. It requires only a second to compute. The apartment house, a much simpler building, takes almost 5 seconds to compute. The reason is that in the former case, the designer took care of precise symbol sizing for all elements and limited the usage of counters, whereas in the latter case, inconsistent symbol sizing increases the frequency of constraint violations, i.e., the shortest-path search in the ADN is more complex. The modern-looking layout of the teaser-image building is actually a branch of the bookstore building (built 1837 in Vienna). In the comparison in the following section, we present a contemporary terraced house.

6.1 Comparison to Split-Based Methods

For a comparison of our method with a layer-less approach, characteristic modeling operations were performed both with our framework and with a grammar based on split rules (see Figure 9). The modeling process for obtaining the prototype as well as the complexity of further changes are described and evaluated. Creation and management of rules is the most demanding task for grammar-based systems, therefore we consider the number of rules and their edits as the relevant comparison parameter.

Our Method. A single layer is sufficient to create a basic layout of the reference façade in Figure 9. It is specified using the vertical pattern GXS^mYT and the horizontal pattern $W\,(BW)^n$. W stands for a window block, B for a balcony block. G represents the ground floor, S the other floors and T the terrace. X and Y represent thin ledges.

The first layout modification inserts vertical delimiters. There are two ways of adding them to the design: either by changing the horizontal pattern to $W\,(QBW)^m$ or by adding a new layer. Ignoring the following tasks, modification of the existing layer seems more convenient. It is accomplished by inserting a new symbol into the pattern.

The second layout modification adds a large balcony to each odd floor using a new layer L_1. Selection of the odd-numbered floors is done by a substituted pattern $GY\,(zS)^p\,z^b v$ with $z := Z$. Floors to be altered are represented by a new symbol Z with the same size as S. The subpattern $(zS)^p\,z$ would imply an odd number of floors, thus we add z^b with $b \in \{0,1\}$. The unsized symbol v automatically covers terraces on the last floor. In the horizontal direction, we add a new symbol L matching the size of BW. The substituted pattern $WQlh$ with $l := L$ aligns the first BW from the left with the large balcony L, and the symbol h aligns with the rest.

Split-Based Method. The derivation process in Figure 9 left utilizes a compound subdivision rule which is not a standard part of split-based approaches. It performs a split with optional repetitions (using the Kleene star) at once. Although this reduces the number of rules up to 50%, our method still remains considerably more expressive.

For the basic layout, we subdivide an axiom shape σ vertically into rows representing floors. The pattern is nearly the same as for the layered approach. Each symbol (G, S and T) represents a dissimilar part of the façade, therefore we have to write three separate rules with the same structure but using different output symbols. Our method does not require such redundant rules, since it utilizes factorization of dimensions and deals with the semantics also at the shape level.

Façade modifications are significantly more involved wi-

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

thout using layers and factorization. Adding the vertical delimiter requires changing rules for all floors. The number of edit operations is twice as high as for our method. The only option for improvement would be to start over with a horizontal split applied to σ, creating the same issue in the next step when adding the balconies. The ambiguity of split ordering in linearly independent directions is often a problem for split-based approaches.

Lipp et al. [LWW08] proposed *semantic selections* within a visual editing system, which performs simple rule changes automatically. However, insertion of the balconies is a good example where their method fails to update the grammar to include the new pattern (cf. Figure 9, bottom). The problem with even or odd number of floors needs to be resolved by a pair of non-deterministic vertical split rules $\sigma \rightarrow GXV^*ZXTU$ and $\sigma \rightarrow GXV^*XTU$. The new symbol V then develops into ZS with a new horizontal split rule for Z. Similar to previous tasks, our approach again required only half the rules.

6.2 Comparison to Structure-Aware Methods.

Our method also has the ability to create further variations of the façade (see Figure 10), similar to Bao et al. [BSW13], Dang et al. [DCNP14] or Zhang et al. [ZXJ*13]. They all supply only high-level user interaction modalities. Our system is not limited to the data of an input image. It allows designing façades from scratch. Not only automatic retargeting of existing designs, but also insertion of new elements, patterns and layers are supported. Switching the layers on/off supports a non-linear workflow and allows exploring various configurations.

The main difference from our concept can be seen in the structure representation. In the case of Bao et al., an initial hierarchical segmentation with split lines has to be performed, on which various constraints, such as frequency, symmetry, size and others can be defined. Such constraints can only be set between regions, which in turn are defined by the segmentation. This forces the user to perform the initial segmentation such that the desired variations can be achieved. This makes it hard to design intentional misalignment in both directions, which can be simply added with an additional layer in our case.

The work of Dang et al. [DCNP14] uses a hierarchy of generalized grids to describe the interrelations of façade elements. The user then specifies a set of grids at possibly different levels of the hierarchy, which are then removed or duplicated during façade resizing. By choosing different sets of grids at subsequent editing sessions, a large number of variations can be generated. However, it can be difficult to anticipate the final result of such iterative modeling.

Our framework keeps the layers in a flat stack, avoiding complex hierarchies and structural dependencies. Users can focus on editing single layers, while our system aligns and merges the layers. Independent editing of horizontal and vertical patterns is simpler compared to split-based approaches like [DCNP14]. However, Bao et al. or Dang et al. provide simpler user interactions when compared to our framework, as patterns do not have to be explicitly specified. On the other hand, the expressive power of our concept is higher, providing means for semi-occluded and overlapping structures, complex repetitive and reflective symmetries, and large elements spanning the whole façade.

6.3 Limitations

While our framework works in most cases we encountered, we also observed that severely underconstrained user input can lead to impractical runtimes of our solver. This is especially the case when the user specifies layouts where symbols with significantly different preferred sizes are meant to be aligned. Thus, the solver has to allow for a large error margin, which only happens after a large number of alignments with smaller error margin are discarded. If many unbounded parametric counters are used in the layout definition, the search space for possible alignments can also become too large to be traversed at interactive speed. We believe that further research on pattern analysis as described in a simple form in Section 5 could eliminate these issues.

7 Conclusion

We proposed a layered approach for façade modeling, inspired by the layer stack used in common image editing. Our main contribution is the concept of pattern-based modeling combined with a layered organization of related (but not necessarily adjacent) regions. On each layer, simple patterns control the arrangement of façade elements and masks, and our optimization routine automatically combines the active layers into meaningful layouts. By specifying repetitions, reflections and alternations, the layout is able to adapt to different façade sizes while still respecting the preferred sizes of the façade elements. Our method reduces the complexity of rule-based methods and also facilitates direct control. The measurements show that our concept is at least twice as efficient as split-based methods.

We hope that our research results can also help to improve automatic façade reconstruction. While primarily focused on façades, we believe that our approach can be successfully transferred into other application domains that utilize a grid-based layout of elements like web design. When extending to the curved domain, jewelry design is also a promising application field.

Acknowledgements. Our research was financed by the Austrian Science Fund (FWF) projects Nr. FWF P24600-N23, FWF P23700-N23 and FWF P23237-N23.

References

[Ang80] ANGLUIN D.: Finding Patterns Common to a Set of Strings. *J. Comput. Syst. Sci. 21*, 1 (1980), 46–62. 3

[ARB07] ALIAGA D. G., ROSEN P. A., BEKINS D. R.: Style grammars for interactive visualization of architecture. *IEEE Trans. Vis. Comp. Gr. 13*, 4 (2007), 786–97. 2

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.

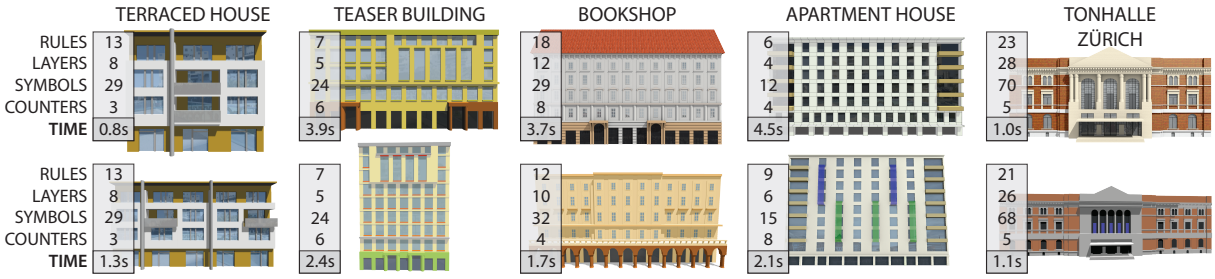

Figure 10: Various façades that were modeled with our layer-based framework. Apart from simple adaption to size changes of the façade (Teaser Building, Kongresshaus), we also support changes of the façade element sizes (Apartment house) or complex editing operations to alter significant parts of the façade (Terraced House, Bookshop).

[Bjö88] BJÖRCK Å.: A Direct Method for Sparse Least Squares Problems with Lower and Upper Bounds. *Numerische Mathematik 54*, 1 (1988), 19–32. 9

[BSW13] BAO F., SCHWARZ M., WONKA P.: Procedural Facade Variations from a Single Layout. *ACM Trans. Gr. 32*, 1 (Jan. 2013), 1–13. 2, 11

[DCNP14] DANG M., CEYLAN D., NEUBERT B., PAULY M.: SAFE: Structure-aware Facade Editing. *Comp. Gr. F. 33*, 2 (May 2014), 83–93. 2, 11

[Fin08] FINKENZELLER D.: Detailed Building Facades. *IEEE Comp. Gr. App. 28*, 3 (May 2008), 58–66. 2

[FMLW14] FAN L., MUSIALSKI P., LIU L., WONKA P.: Structure completion for facade layouts. *ACM Trans. Gr. 33*, 6 (Nov. 2014), 210:1– 210:11. 2

[Gel10] GELADE W.: Succinctness of Regular Expressions with Interleaving, Intersection and Counting. *Theoretical Comp. Sci. 411*, 31-33 (2010), 2987–2998. 3

[HHKF10] HOHMANN B., HAVEMANN S., KRISPEL U., FELLNER D.: A GML Shape Grammar for Semantically Enriched 3D Building Models. *Comp. & Gr. 34*, 4 (2010), 322–334. 2

[HMU07] HOPCROFT J. E., MOTWANI R., ULLMAN J. D.: *Introduction to Automata Theory, Languages and Computation*. Pearson Addison-Wesley, Upper Saddle River, NJ, 2007. 3

[HNR68] HART P., NILSSON N., RAPHAEL B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Sys. Sci. Cyb. 4*, 2 (July 1968), 100–107. 8

[Jaf78] JAFFE J.: A Necessary and Sufficient Pumping Lemma for Regular Languages. *SIGACT N. 10*, 2 (July 1978), 48–49. 3

[KBK13] KRECKLAU L., BORN J., KOBBELT L.: View-Dependent Realtime Rendering of Procedural Facades with High Geometric Detail. *Comp. Gr. F. 32*, 2 (May 2013), 479–488. 2

[KPK10] KRECKLAU L., PAVIC D., KOBBELT L.: Generalized Use of Non-Terminal Symbols for Procedural Modeling. *Comp. Gr. F. 29*, 2 (May 2010), 1–12. 2

[LCOZ*11] LIN J., COHEN-OR D., ZHANG H., LIANG C., SHARF A., DEUSSEN O., CHEN B.: Structure-preserving Retargeting of Irregular 3D Architecture. *ACM Trans. Gr. 30*, 6 (Dec. 2011), 1–8. 2

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive Visual Editing of Grammars for Procedural Architecture. *ACM Trans. Gr. 27*, 3 (Aug. 2008), 1. 2, 11

[MS72] MEYER A. R., STOCKMEYER L. J.: The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. *IEEE Symp. F. Comp. Sci.* (1972), 125–129. 3

[MWA*13] MUSIALSKI P., WONKA P., ALIAGA D. G., WIMMER M., VAN GOOL L., PURGATHOFER W.: A Survey of Urban Reconstruction. *Comp. Gr. F. 32*, 6 (Sept. 2013), 146–177. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural Modeling of Buildings. *ACM Trans. Gr. 25*, 3 (July 2006), 614. 2

[MWW12] MUSIALSKI P., WIMMER M., WONKA P.: Interactive Coherence-Based Façade Modeling. *Comp. Gr. F. 31*, 2 (May 2012), 661–670. 2

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural Modeling of Cities. In *Proc. of ACM SIGGRAPH '01* (New York, New York, USA, 2001), ACM Press, pp. 301–308. 2

[RM94] REINEFELD A., MARSLAND T. A.: Enhanced Iterative-Deepening Search. *IEEE Trans. Pattern Anal. Mach. Intell. 16*, 7 (July 1994), 701–710. 8

[Ros66] ROSENBERG A. L.: On Multi-head Finite Automata. *IBM J. Res. Dev. 10*, 5 (Sept. 1966), 388–394. 8

[SHFH11] SHEN C.-H., HUANG S.-S., FU H., HU S.-M.: Adaptive Partitioning of Urban Fcades. *ACM Trans. Gr. 30*, 6 (Dec. 2011), 184–191. 2

[Sti75] STINY G.: *Pictorial and Formal Aspects of Shape and Shape Grammars and Aesthetic Systems*. Phd thesis, University of California, Los Angeles, 1975. 2

[VAW*10] VANEGAS C. A., ALIAGA D. G., WONKA P., MÜLLER P., WADDELL P. A., WATSON B.: Modelling the Appearance and Behaviour of Urban Spaces. *Comp. Gr. F. 29*, 1 (Mar. 2010), 25–42. 2

[WFP10] WU C., FRAHM J.-M., POLLEFEYS M.: Detecting Large Repetitive Structures with Salient Boundaries. In *Comp. Vision - ECCV 2010* (Crete, Greece, 2010), vol. 6312, Springer Berlin / Heidelberg, pp. 142–155–155. 2

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant Architecture. *ACM Trans. Gr . 22*, 3 (2003), 669. 2

[XFT*08] XIAO J., FANG T., TAN P., ZHAO P., OFEK E., QUAN L.: Image-based Façade Modeling. *ACM Trans. Gr. 27*, 5 (Dec. 2008), 161:1–161:10. 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 Trans. Gr. 32*, 4 (2013). 2, 11

© 2015 The Author(s)
Computer Graphics Forum © 2015 The Eurographics Association and John Wiley & Sons Ltd.