Layer-Based Procedural Design of Facades Supplementary Material

Martin Ilčík Przemyslaw Musialski Thomas Auzinger Michael Wimmer
Vienna University of Technology, Austria

Contents

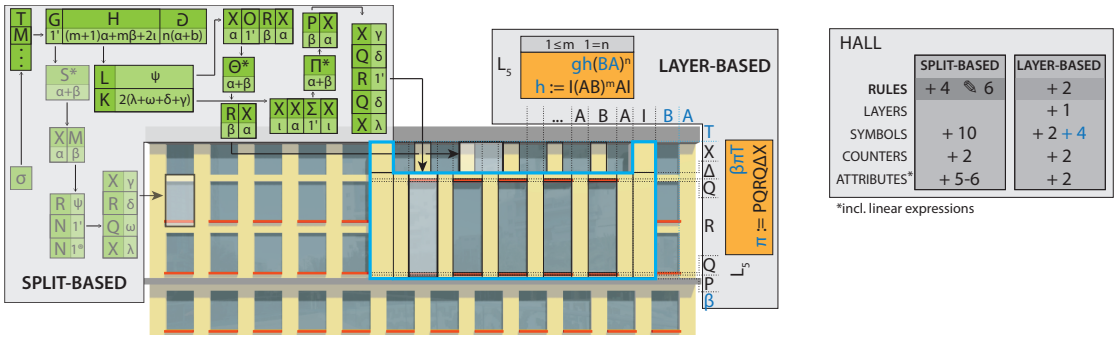


Figure 1: Comparison of designs for the top layer of the teaser. The cyan outline marks the boundary of a new layer with a large hall. While our approach requires only two new patterns (orange), the hierarchy of splits (green) must be extensively changed to produce the same result. Dimmed green rule chains starting attached to G were previously connected to M before adding the hall. Light boxes with attributes (Greek letters) denote shapes sizing, $1'$ denotes relative sizing.

1 Additional Definitions

1.1 Reflection Modes for Counters

We employ patterns that are regular expressions with common operations: concatenation, alteration, grouping. Complex repetitive and reflective symmetries require a more powerful concept than regular grammars. Therefore, we add a *parametric counting* operation with a set of reflection modes as proposed in Table 1.

Notation	Mode	Input		Result
	none	$(EkG)^{2}$	=	$EkG\ EkG$
$-$	normal inner	$(EkG)^{-2}$	=	$EkG\ \mathfrak{GAE}\ EkG\ \mathfrak{GAE}$
$\vert$	normal outer	$(EkG)^{\vert 2}$	=	$EkG\ EkG\ \mathfrak{GAE}\ \mathfrak{GAE}$
$\cdot$	pivot inner	$(EkG)^{\cdot 2}$	=	$Ek\ \mathrm{G}\ \mathfrak{AE}\ Ek\ \mathrm{G}\ \mathfrak{AE}$
$\circ$	pivot outer	$(EkG)^{\circ 2}$	=	$EkG\ Gk\ \mathrm{E}\ \mathfrak{AG}\ \mathfrak{GAE}$
$\odot$	pivot total	$(EkG)^{\odot 2}$	=	$Ek\ \mathrm{G}\ \mathfrak{A}\ \mathrm{E}\ k\ \mathrm{G}\ \mathfrak{AE}$

Table 1: Reflection modes applied to the pattern $(EkG)^{2}$.

While the reflection mode is set for each counter manually by the user, the repetitions count is automatically determined in the first stage of the facade solver. The repetition counts are constrained to be equal for all instances of the counter across all patterns and alignment groups.

1.2 Multi-head Finite State Automaton

In order to derive the oriented network encoding all possible alignments of a candidate $(w_i)_{i=1}^{m}$ in Section 5.2, we construct a one-way finite state automaton with m heads, where m is the number of patterns to be aligned.

The automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Σ is the alphabet, i.e. the set of symbols defined in our system and $\# \notin \Sigma$.
- q_0 is the initial state $(w_1\#, \ldots w_m\#)$ with $\#$ being the end marker.
- F is the final state $(\#, \ldots, \#)$
- Q is a finite set of states $Q = \{(Q_i)_{i=1}^{m} \,|\, Q_i \text{ is suffix of } w_i\}$
- δ is a finite set of transitions $(q, a_1, \ldots, a_m) \to q'$ with $q, q' \in Q$, being the current and next state, respectively, and $a_1, \ldots, a_m \in \Sigma \cup \{\varepsilon\}$, such that $\exists a_i \neq \varepsilon$, being the symbols currently read by the m heads.

Note that we add # to the end of each word in the candidate tuple so that we can assure acceptance in a single final state once all heads finished reading. The set of states is made up of all suffix combinations of the input tuple:

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

The cardinality of Q is given by the product of word lengths $|Q| = \prod_{i=1}^{m} |w_i|$. It is possible to separate the read operations for each head and construct the transition relation as $\delta = \delta_1 \cup \ldots \cup \delta_m$ with

$$\delta_i = \left\{ \begin{array}{c} ((q_1, \ldots, xq_i, \ldots, q_m), a_1, \ldots, a_m) \to (q_1, \ldots, q_i, \ldots, q_m) \\ \text{where} \qquad x \in \Sigma \ \wedge \ q \in Q \wedge \\ a_k = (x \text{ for } k = i \text{ else } \varepsilon) \end{array} \right\}$$

2 Modeling Examples

2.1 Residential House - Continued

In order to finish the design session with a simple residential building from Section 4 of the paper, a door and a roof need to be added, each in a new layer. Figure 2 lists all steps of user interaction.

Doors Layer. Creation of the door element closely follows the steps 5 – 8 which create the columns. The horizontal pattern for this layers is slightly different, since the content should now be placed in the center. The door is represented by the substitution $d := D$. Its centering is achieved by a symmetric control pattern fdf. Since the door is added only on the ground floor, the vertical pattern B_y can be reused again with P set to be visible in the mask.

Roof Layer. Up to now, only the ground floor P and the first floor Q were considered in the vertical patterns. In order to add a roof R to the top, we could alter the base pattern B_y. It is

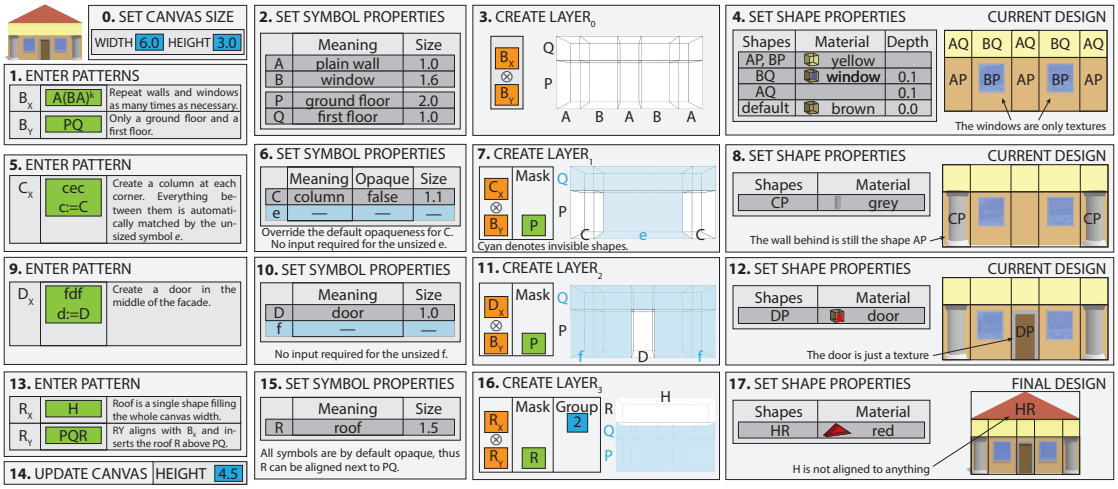


Figure 2: The whole modeling process for the residential building. Section 4 of the paper explains steps 0 – 8, here we continue with steps 9 – 17 in Section 2.

Figure 3: An apartment house in Vienna as reference for the design in Figures 4 and 5.

easy to accomplish in this trivial design, but for complex scenarios we have a better option. The user adds a new pattern R_y with PQR and expects PQ to align with the existing content. R should be placed on the top, even without any content in the underlying layers. Our framework supports such design thinking by the alignment qualifiers (see section 3.5 of the paper). All symbols are *opaque* by default, since the default shape geometry is a box. We assume that any shape produced by a pair of opaque symbols will cover the whole cell of the alignment grid, so that unintended holes in the facade are avoided. In this specific case the roof has a triangular shape, which does not cover all the space behind. However, there will be no more elements above the roof to cause problems, thus we can keep the default opacity property of the symbol R. Opaque symbols are allowed to be aligned without any content in underlying layers. Therefore, we expect the roof to be aligned separately above P and Q. After the vertical pattern is created, the user updates the canvas height.

For the horizontal direction a special pattern must be created as well. The triangular roof should stretch horizontally over the whole canvas. The new pattern R_x reflects this by being only a single symbol H. Note that it is course opaque by default. However, H is so large that it would destroy any fragmentation of the alignment, resulting in all content from all horizontal patterns being aligned in a single cell, i.e. not aligned at all. For completeness, we note that only transparent symbols (since they produce no geometric shapes) may split to achieve a better alignment. The user solves the problem by setting R_y to a new *alignment group* within $Layer_3$. The roof spans horizontally over the whole canvas, so there is no need to align it with anything. Content from layers $0-2$ can be normally aligned in the default alignment group without being limited by H. At last, a mask is applied to R in $Layer_3$ so that only the roof stays visible.

2.2 Apartment House

Additionally to the design session example in the paper, we provide a more complex example with an apartment house. The user follows the design of a photographed building in the Figure 3. It takes approximately 12 minutes to create the design from scratch. A 21×10 units canvas will be covered by at least three layers. Apart from the basic layer, the ground floor and the first floor both contain individually aligned structures.

Basic Layer – Factorized Patterns. Starting from scratch, the user enters the patterns for the first layer. The main part of the building in the horizontal direction follows

$$B_x = GC\ A(BA)^k\ CG\,.$$

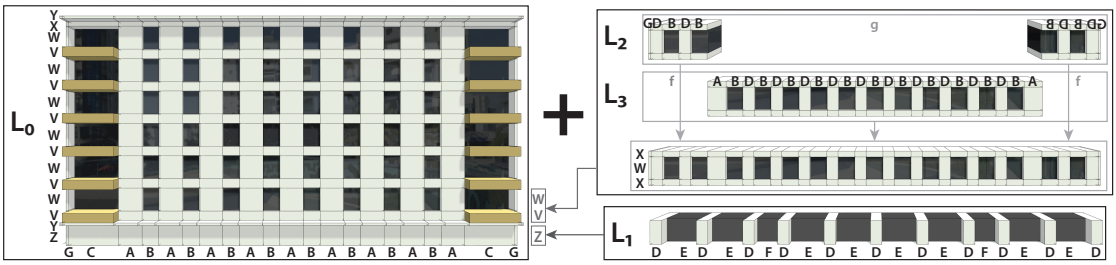


Figure 4: Irregularities in common facades require misalignment of elements. This layout uses a basic layer L_0 (left) and three more layers (right) with horizontal patterns assigned to different alignment groups. This way, the windows in the first two floors do not align with balconies and windows of the upper floors. The result is shown on Figure 5, for details see Section 2.2.

Please note that for symbols related to the horizontal/vertical direction, we use letters from the begin/end of the alphabet. In a similar way, the basic vertical pattern is given as

$$B_y = ZY\ (VW)^l\ XY\ .$$

Then, desired sizes for symbols introduced in these patterns need to be set. Their values are shared among all other patterns that use the respective symbols. Sizing and meaning of the symbols is listed in Figure 5, right. The counters k and l stay unrestricted. At this point, the basic layer L_0 can already be created.

A grid appears, with all shapes being bright boxes. By assigning materials and depth offsets to the shapes, we introduce details to the layout. A vast majority of shapes keep the default appearance, the rest should be changed according to Figure 5 right.

In contrast to split-based grammars [WWSR03, MWH$^+$06], our factorized approach to patterns is free of decisions on which direction to give preference to. Producing directly the grid with final shapes makes instancing and combining of patterns easier.

Ground Floor Layer – Masking. The second layer L_1 will contain the ground floor – denoted by Z in B_y. Hence, only shapes produced by Z should be visible in L_1. The user can directly set B_y as the vertical pattern of L_1 and assign Z to the respective mask. Our framework automatically overrides the geometry of all shapes created by the remaining vertical symbols (V, W, X, Y) to be empty.

A generalized horizontal pattern for the ground floor is more difficult to estimate, our design proposal is

$$G_x = (DE)^n D\ F\ D(ED)^m\ F\ D(ED)^n \quad \text{with} \quad n < 3\,.$$

B_x and G_x are different in structure. A common alignment would not work well. Our concept allows for efficiently dealing with several independent alignments at once. The ground floor in L_1 can be easily made independent to other layers by using a separate alignment group for G_x.

First Floor Y – Substitutions. Structured selection of facade regions is often needed to assemble more complex patterns. Lipp et al. [LWW08] introduced instance locators for split-based grammars to reduce the complexity of local changes. Instead, we utilize the more versatile substituted patterns, which suit the layered concept the best.

A vertical pattern should provide content for the first floor only. Adapting the existing B_y pattern is not possible in this case, since there is no way to distinguish between the floors

represented by $(VW)^l$. The user will take advantage of the layered concept by creating a new substituted pattern:

$$F_y = \left\{ \begin{array}{ll} & ZYts \\ t := & XWX \end{array} \right.$$

The unsized symbol s automatically covers everything above the first floor. We set the preferred size of X to be approximately half of the size of V, i.e. $X \approx 0.5V$.

First Floor X – Multiple Alignments. Two corner windows on the first floor align with balconies in L_0, whereas the middle part with thinner walls should be independent. Multiple alignments can be realized only in distinct patterns, so the user creates a pair of patterns F_x and M_x with complementary substitutions. Together they yield

$$GDBDB \quad AB(DB)^jA \quad \text{ꟻ}\text{ꟻ}\text{ꟻ}\text{ꟻ}\text{Ɔ}$$

$$F_x = \left\{ \begin{array}{ll} & f\ g\ \text{ꟻ} \\ f := & GDBDB \end{array} \right. \qquad M_x = \left\{ \begin{array}{ll} & f\ g\ \text{ꟻ} \\ g := & AB(DB)^jA \end{array} \right.$$

Layers L_2 and L_3, both with F_y as vertical pattern, utilize F_x and M_x, respectively. M_x gets assigned to a new alignment group in layer L_3. Notice that despite of being in different alignment groups, F_x and M_x will always fill the whole canvas. The reason is that both use the same control pattern. Since the size of f and g is the same across all patterns and alignment groups, an explicit alignment is forced.

Layer Ordering. During the design process, the user created the following stack of layers with L_0 being at the bottom.

	Pattern	Group	Mask		Pattern	Group	Mask
L_3	M_X	3	—	⊗	F_Y	1	—
L_2	F_X	1	—	⊗	F_Y	1	—
L_1	G_X	2	—	⊗	B_Y	1	Z
L_0	B_X	1	—	⊗	B_Y	1	—

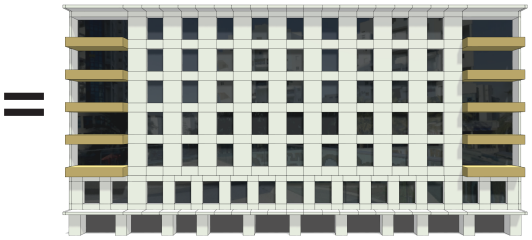
Ordering of layers is always important for visibility of elements. If L_0 were on top of the stack, it would wrongly cover elements form all other layers.

We created a non-trivial facade layout using only a few design elements. More results showing the advantages of our method and a critical comparison with other approaches are presented in Section 5 of the paper.

3 Details of the Solver

3.1 Pattern Analysis

The complexity of counting and sizing can be significantly reduced even before any words get generated. Control patterns sometimes contain the same subpatterns. This can be recognized first by substituting the content pattern into the control pattern and flattening the counters. Let us demonstrate it on the following simple example with two patterns:



Symbol	Meaning	Size
G	corner wall	0.1
C	balcony	2.2
A	wall	1
B	window	0.7
D	thin wall	0.3
E	large shop	1.8
F	small shop	1.2
Z	ground floor	1
Y	thin console	0.1
V	wall	0.4
W	window	1
X	thin wall	0.2

Shapes	Depth
CW, EZ,FZ	- 0.3
BW	- 0.1
GY, GG, CY, CV, CG AY, AG, BY, BG	0.3
default	0.0

Shapes	Material
CW, BW	dark glass
CV	orange
EZ, FZ	glass
default	white

Figure 5: Continuation of the Figure 4 with the resulting facade. Additional layers L_1, L_2 and L_3 introduce windows following a new pattern, which is not aligned with the layer L_0. The alignment in both directions is shown with transparent letters in blue. The list of all symbols and shapes is given right.

$$Pattern_0 = \begin{cases} & A^k c C^m c A^k \\ c := & C \end{cases} \qquad Pattern_1 = A^k C^l A^k$$

After trimming away the common parts from a pair of patterns, the sum of symbol sizes in the two remaining subpatterns should be equal because all layers span over the whole canvas. Thereby, we get a linear expression with either counters or control symbols as variables. After flattening, $Pattern_0$ and $Pattern_1$ reveal a dependency of the counter values

$$\begin{array}{lccc} Pattern_0 = & A^k & \underline{C} C^m \underline{C} & A^k \\ Pattern_1 = & \underline{A^k} & \underline{C^l} & \underline{A^k} \end{array} \Rightarrow m + 2 = l \ .$$

A pairwise analysis of all patterns for the given direction gains two systems of linear equations – one for counters and one for unsized symbols. Some counters and unsized symbols can be directly resolved by solving the system, others become constrained by the equations so that the number of possible counter values and sizing options stays low. In case one of the systems is infeasible, the user is notified to check the consistency of the facade design.

3.2 Pruning Candidates by Size of Opaque Symbols

When the solver generates words for alignment as described in Section 5.1 of the paper, the upper bound for the word size is determined by the canvas size. The lower bound l_i is more difficult to compute, since it is determined by the count of opaque symbols in patterns above in the stack. The reason is that an opaque symbol may push the content from underlying layers to the sides so that it becomes the deepest non-ε element in the respective column of the alignment matrix.

I opaque	false			$\sum$	true					$\sum$
$\lvert A\rvert = 4$	IB	IB	I	11	I	B	I	B	I	11
$\lvert B\rvert = 4$	A	A	A	12	ε	A	ε	A	ε	8
$\lvert I\rvert = 1$										
error	1	1	3	5	0	0	0	0	0	0

In the given example, A represents a wall, I a column and B a balcony. Only A is opaque. The patterns $I\,(BI)^l$ and A^k cover a canvas of size 12. When I is not opaque, the shapes produced by I occlude the content behind them only partially. Thus, I must be aligned to an opaque symbol from a pattern deeper in the stack in order to avoid gaps in facade. But once I is set opaque, it *can* be aligned in a way that it becomes the deepest non-ε element in the

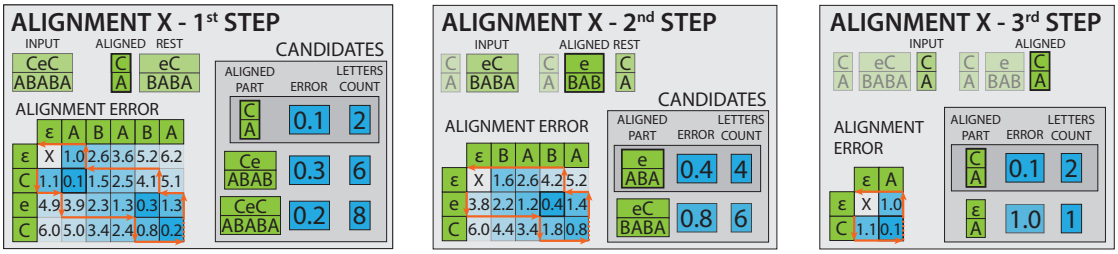


Figure 6: Weighting of edges for the three steps of the shortest path search in Figure 8 of the paper. X denotes the current node, other matrix elements list alignment errors δ for all reachable nodes. Orange arrows track boundary of the $\delta \leq 1.0$ tolerance region .

aligned column. The columns $\binom{I}{\varepsilon}$ are actually inserted in between the content of the bottom pattern. This often reduces the local alignment error, but size of the resulting alignment grows. If the words produced by deep patterns were already approaching the canvas size, the inserted columns would cause the alignment to exceed the error tolerance. Therefore, shorter words need to be considered for patterns deeper in the stack. Once a pattern contains opaque symbols, also words smaller than canvasSize $- \mu$ must be considered for all patterns below.

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

If w_2 generates AA instead of AAA in the previous example, then the alignment fits perfectly.

3.3 Weighting of Alignment Columns

The search for an optimal alignment is formulated as a shortest path problem on the *alignment decision network* ADN. To construct an ADN, we use the state transition graph of the automaton as defined in Section 1.2. We have defined the transition relation δ as a union of relations where each δ_i uses only the $i^{\textbf{th}}$ head for reading. Therefore, the transition graph defines a partial ordering of the set of states Q. Structure of the graph forms a directed acyclic m-dimensional square lattice. In order to allow reading more than a single letter by one or more heads at once, the ADN is constructed as the transitive closure of the transition relation δ.

In the following, we provide the explanation to $m = 2$. A generalization to an arbitrary m is achieved using a hierarchical approach with additional time complexity of $O\left(\log m\right)$. Let the initial state be placed top-left and the final state bottom-right in the square lattice. Then, the transitive closure causes that each state in the ADN can reach all other (and only those) states to the right and to the bottom, by a single edge. The set of reachable states is actually a rectangular subregion of the ADN. The number of incoming and outgoing edges for each node can be easily determined as the product of the respective subword lengths, roughly $O\left(n^m\right)$.

Linear traversal Weights for the edges are determined by the column alignment error ω (see Eq. (9) in the paper). In the previous paragraph we have shown that the number of edges leaving a node grows exponentially with the number of layers. Thus, their weights are computed on-demand only for visited nodes. The property of partial ordering of the state space allows to reduce the complexity even more. In the following we show that for a large portion of edges leaving the current node can be ignored. Please follow the Figure 6 for an illustration of the weighting process on the data from Figure 6 in the paper.

utilize dynamic programming to limit the weighting to a small subset of edges.

For $m = 2$, there are only two reading heads, i.e. two directions in the square lattice. The region to the lower-right from the current node contains all nodes directly reachable. Let us set a pointer to the current node. Using a dynamic programming method which accumulates the signed alignment error, the pointer can track the border of the region where $\omega \leq \mu$. In Figure 6 it is marked by a red line. It takes exactly $2(m + n)$ steps. The region mostly takes a narrow diagonal form. The quadratic complexity becomes nearly linear.

In more detail, the pointer starts toward bottom. It simulates reading with the second head, accumulates the size of each processed letter and computes ω. Once $\omega > \mu$ the heads switch and the pointer moves to the right. The ω now decreases. A switch is also performed once $\omega \leq \mu$. This way the lower border of the acceptable region is traced. After reading both words to the end, the pointer returns in the same manner tracing the upper border of the region. There is no need to compute ω for nodes outside of the region, as it is known to be higher than μ.

For $m > 2$, words of the candidate are processed pairwise with results combined in a hierarchical way. The number of edges needed to be weighted for a node of ADN is then $O(n \log m)$.

3.4 Placement of Non-Opaque Symbols

In the alignment constraints for the least squares solver listed in Section 5.3 of the paper, the last constraint

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

assures that all rows of any column j have to be sized equally. However, it can be relaxed for non-opaque symbols. Shapes produced by non-opaque symbols do not occlude everything behind them. Thus, it is possible for them to omit the requirement to stretch in the alignment column, and rather let them approximate their preferred size the best. Figure 3b in the paper shows the non-opaque letter P in L_1 placed in the middle of the opaque content from the layer below. We relax the stretching behavior by inserting pairs of additional gap letters $\gamma_1 \ldots \gamma_4$ as the first and last letter to those elements of $A_{i,j}$ with at least one non-opaque letter. The alignment matrix from Figure 3b changes as follows:

$$\begin{pmatrix} \varepsilon & a & \underline{P} & b & \varepsilon & \underline{P} & a & \underline{P} & b & \underline{P} & a & \varepsilon \\ \underline{A} & \underline{D} & \underline{B} & \underline{D} & \underline{A} & \underline{DB} & \underline{D} & \underline{A} & \underline{D} & \underline{B} & \underline{D} & \underline{A} \end{pmatrix}$$

$$\downarrow$$

$$\begin{pmatrix} \varepsilon & a & \gamma_1\underline{P}\gamma_1 & b & \varepsilon & \gamma_2\underline{P}\gamma_2 & a & \gamma_3\underline{P}\gamma_3 & b & \gamma_4\underline{P}\gamma_4 & a & \varepsilon \\ \underline{A} & \underline{D} & \underline{B} & \underline{D} & \underline{A} & \underline{DB} & \underline{D} & \underline{A} & \underline{D} & \underline{B} & \underline{D} & \underline{A} \end{pmatrix}$$

The γ and ε symbols are similar, but the instances of γ must be pairwise equally sized to achieve centered placement of the corresponding elements. As a further extension, it would be possible to allow to the local alignment for non-opaque symbols not only to *center*, but also to *left* and *right*.

3.5 Sizing and Placement Example

Figure 7 shows the final stage of the solver processing the best alignment of $(CeC, ABABA)$. Instances of symbols are first transferred to letters and a representative instance for each symbol is chosen and stored in the property p. Yellow boxes list constraints in the linear system. All instances of a symbol are constraint to the same size. The objective in the orange box minimizes the size deviation of the optimized symbols sizes to the preferred sizes given by the user.

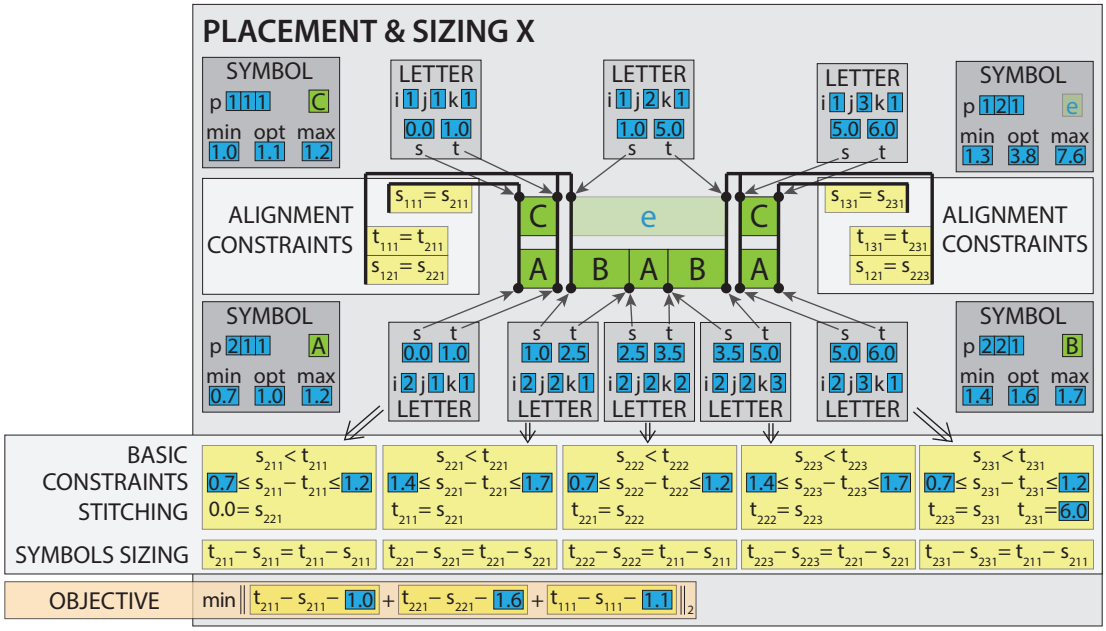


Figure 7: Linear system of equations for placement of $(CeC, ABABA)$. Letters are extracted from the alignment and a set of constraints is constructed. The s and t variables are referenced by the letter identifier with ijk addressing $a_{ij}\,[k]$. Note that redundant constraints and tautologies were not removed in this example. However, we did omit local constraints for letters 1-1-1, 1-2-1 and 1-3-1.

4 Further Comparisons with Related Concepts

4.1 Modeling with Split Based Grammars

Our approach provides even more significant results for facades with many details organized in both – regular and irregular patterns. Let us examine the top layer of paper's teaser containing an irregular dominant hall. Please follow Figure 1 to compare the design process.

4.1.1 Our Method

Without editing or understanding the remaining content, the hall can be modeled in a separate layer with a pair of new masked patterns. The hall stretches over two full floors and covers small parts of the last floor as well. In opposite to split methods, which are limited to convex scopes, our patterns mask out arbitrary shapes that can be represented as a union of rectangles. By setting AX and BX to transparent, the background layer stays visible on the last floor while IX stays solid. The hall position relative to the right wall is controlled by the counter n in $(BA)^n$. Unsized symbols g, h, π and β adapt automatically. A, B, P, Q, Δ, X and T have been already defined and used by lower layers. The horizontal pattern is assigned to a separate alignment group so that the tall windows will not align with the rest of the facade. The hall addition is simple to perform, without any interference with the rest of the facade design.

4.1.2 Split-Based Method

The user needs to understand and to edit the whole hierarchy of rules to achieve the same result using a split-based tool. The changes involve four new rules, five adapted copies and one edited rule In sum ten additional symbols are required. The split concept implicitly requires redundant structural information: After being derived from the G branch, the L branch encodes a very similar structure. It is easy to get confused in the hierarchy of split and repeat rules. Adding even more independent patterns using a split-based method would be very difficult to achieve and to control.

While our system binds the geometric size to the symbols and guarantees equal size all over the facade, split-based approaches are designed in a fully different way. Therefore, you can see various combinations of symbols and sizes in Figure 1. With increasing complexity, the user seeks automated sizing of shapes with help of relative values. By our experience, concurrent usage of relative and absolute sizing is difficult. The best choice is to define the size of the final elements absolutely, using global attributes which are fixed. Snapping, proposed by Müller et al. [MWH+06], offers only limited fine-tuning – not comparable to our alignment process. Absolute sizing also produces a lot of design complexity, since a high number of linear expressions is required for sizing of splits. For the hall structure, it is mainly G and K which are difficult to be sized. Consequently, the repeat rules are indirectly bound to the counter values m and n which are set manually. Contrary, our approach automatically seeks the best values for them.

4.1.3 Added Detail

The layout as shown in Figure 1 is a bare variation of a real bookshop facade in Vienna, built almost 180 years ago. A detailed reconstruction of the real building (Figure 8) requires 18 patterns with 29 symbols ordered in 12 layers. The design stays simple as the patterns are defined independently of each other. A CGA Shape reconstruction of the same facade requires

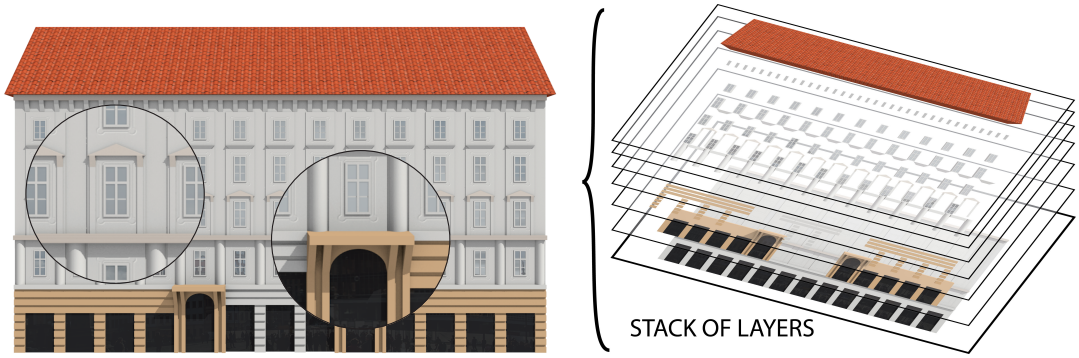


Figure 8: Detailed replica of the real-world bookshop facade with our framework. Note that for illustration, some layers were merged together in the stack on the right.

55 rules organized in a complex hierarchy. Each addition of new elements requires global changes and imposes high intellectual demands on the designer.

To solidify our claim that our layer-based description of facade layouts compares favorably to split-based grammars, we provide an overview in Figure 9. It lists the number of rules, layers, symbol and counters that were needed to create the facade that are shown in the top row of Figure 10 in the paper.

4.2 Detailed Comparison with Structure-Aware Concepts

Our main contributions over the most closely related approaches of Dang et al. [DCNP14] and Zhang et al. [ZXJ+13] are:

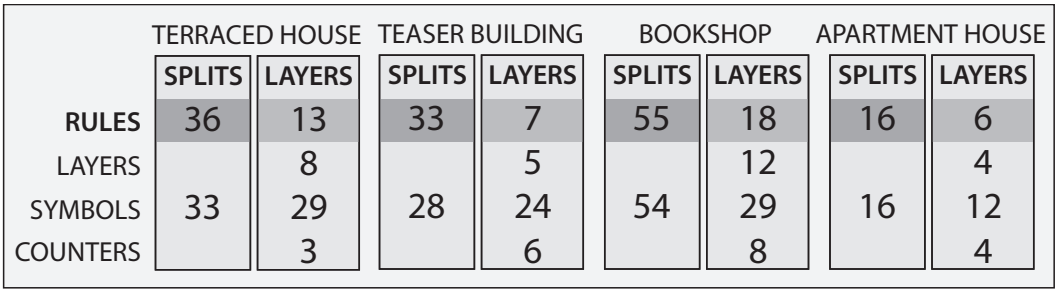
1. Different concept of structure representation
2. Global alignment solver

Layers. [DCNP14] utilize generalized grids, [ZXJ+13] implement a layering operation, but both are based on decompositions of the facades into hierarchical structures. Both consider this a serious limitation: Manipulation of hierarchical structures is not intuitive for the user [DCNP14], introduces additional constraints and requires to handle many special cases. We put the layers into a single stack, avoiding complex hierarchies. Users can focus on editing single layers while merging and alignment of the layers is solved by our system. Independent editing of horizontal and vertical patterns is simpler compared to split-based approaches like [DCNP14].

Overlapping. [DCNP14] create a non-overlapping hierarchy, the layers in [ZXJ+13] exist only in the volume of their parent. Contrary to [ZXJ+13] and [DCNP14] our method generates full 3D facade models including semi-occluded structures (arcades on Figure 11-bottom-middle).

Large Elements. Insertion of larger elements is difficult due to topology jumps [DCNP14]. Our layers span over the whole canvas, thus elements of any size can be easily added.

General Symmetries. Multi-way partitioning and complex structures beyond reflective symmetry are impossible in [ZXJ+13]. Our masked patterns allow to define complex repetitive and reflective structural symmetries.



	TERRACED HOUSE		TEASER BUILDING		BOOKSHOP		APARTMENT HOUSE	
	SPLITS	LAYERS	SPLITS	LAYERS	SPLITS	LAYERS	SPLITS	LAYERS
RULES	36	13	33	7	55	18	16	6
LAYERS		8		5		12		4
SYMBOLS	33	29	28	24	54	29	16	12
COUNTERS		3		6		8		4

Figure 9: Complexities of modeling the top row facades in Figure 11 in the paper by either split-based grammars (SPLITS) or our layer-based approach (LAYERS). Our approach requires less than half the amount of rules when compared to CGA [MWH+06] for all facades and even much fewer in presence of irregularities as in the teaser building.

Alignment Solver. [ZXJ+13] require manual locking for retargeting. Global alignment is beyond the scope of their paper. [DCNP14] use continuous modifications to repair alignment errors caused by discrete topological jumps. Our main contribution is the automated global alignment. Alignment groups allow intentional misalignments and also localized alignment. We use combinatorial optimization to select the best alignment. Continuous optimization only fine-tunes the symbol sizes without any topological changes.

Creativity. For obtaining variations of a facade [ZXJ+13] and [DCNP14] both require image input and manual segmentation. Elements not present in the original picture can not be added. Our framework allows to design facades from scratch. There is more user-input required, but at the same time more artistic freedom is offered to the designers.

Reusing. Data between nodes in a hierarchy can neither be shared nor be edited at once, resulting in repeated user actions. The concept of symbols allows to reuse the same elements in many patterns+layers, avoiding redundant interaction.

Workflow. The editing process in [ZXJ+13] and [DCNP14] is incremental and linear. Switching the layers on/off supports a non-linear workflow and offers a comfortable way to return to previous layer configurations.

References

[DCNP14] Minh Dang, Duygu Ceylan, Boris Neubert, and Mark Pauly. SAFE: Structure-aware Facade Editing. *Comp. Gr. F.*, 33(2):83–93, May 2014.

[LWW08] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive Visual Editing of Grammars for Procedural Architecture. *ACM Trans. Gr.*, 27(3):1, August 2008.

[MWH+06] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc van Gool. Procedural Modeling of Buildings. *ACM Trans. Gr.*, 25(3):614, July 2006.

[WWSR03] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant Architecture. *ACM Trans. Gr .*, 22(3):669, 2003.

[ZXJ+13] Hao Zhang, Kai Xu, Wei Jiang, Jinjie Lin, Daniel Cohen-Or, and Baoquan Chen. Layered Analysis of Irregular Facades via Symmetry Maximization. *ACM Trans. Gr.*, 32(4), 2013.