Additional Material for

Inverse Procedural Modeling of Facade Layouts

Fuzhang Wu[1,2] * Dong-Ming Yan[2,1] † Weiming Dong[1] ‡ Xiaopeng Zhang[1] § Peter Wonka[2,3] ¶
[1]LIAMA-NLPR, CAS Institute of Automation, China [2]KAUST, KSA [3]Arizona State Univ., USA

1 The Facade Model of Mueller et al.

Mueller et al. [Müller et al. 2007] assume that the facade can be split into a single irregular grid of rectangles, called tiles. In each tile, there is preferably a single element, e.g., window or door. This works well for a large set of facades, but it does not work as well for more complex facades or facades modeled in greater detail (as is the case in our test dataset). If a facade does not conform to the assumed facade model, facade elements have to be split. We do not consider this as valid solution in our paper. Here are some examples of layouts that cannot be represented well:

- facades with more than one grid of elements where the grids are not aligned
- facades with interleaved grids
- signs that span multiple floors or columns of elements
- facades with ornaments, e.g., pilasters, between windows
- facades with columns that span multiple floors
- facades with high doors that span multiple floors
- ornaments that go across the whole facade
- ornaments that have a higher repetition frequency than the base grid of tiles, e.g., triglyphs

In Fig. 1 we show examples that fit well into the model of Mueller et al. In Fig. 2 we show examples that do not work well.

Figure 1: *Facade examples that conform to the model of Mueller et al. These two facades can be split into a grid of tiles. However, if the letters on the first floor of the example to the right is modeled as separate element, the grid structure would also no longer work.*

2 Generating a single deterministic grammar vs. combining multiple deterministic grammars

There are two problems that can be distinguished in our context. The first problem is the following: Given a single layout as input, how can we automatically extract a deterministic split grammar to represent the input layout? The second problem is the following: Given a set of deterministic split grammars that each describe a facade layout, how can we combine the deterministic split grammars to generate a stochastic split grammar? The stochastic split grammar should be able to generate all input layouts as well as new plausible layouts. Our paper tackles the first problem. By contrast, Talton et al. [Talton et al. 2012] and Martinovic et al. [Martinovic and Van Gool 2013] tackle the second problem. However, the second problem requires a solution to the first problem as input. Therefore, Martinovic et al. also propose a solution to the first question. We compare our results to this solution in our paper. For the second question, our paper has a different philosophy. While Martinovic (and also Talton) show very nice initial results, the methods only work when certain assumptions about the input data are met. We were not able to get high-quality results by automatically merging grammars. The facade layouts need to be simple, have few details, and few variations. This is not the case for our test dataset. We therefore opted for a semi-automatic approach as our main modeling tool to generate variations.

3 Structure learning vs. parameter learning

We consider the question of determining the sizing parameters of a splitting rule as parameter learning. Structure learning is concerned with questions such as:

- What rule types do we choose from a set of possible rules? We have multiple different types, including the split rule and the repeat rule
- In what direction (x or y) do we split next?
- How many elements should a split rule split?

Since we have to address all the aforementioned problems, our approach has to determine the structure of the grammar. This is not a facade parsing approach.

4 Comparison to Teboul et al.

We cite Teboul et al. [Teboul et al. 2011] and recognize this paper as important previous work. Still, Teboul et al. and our paper actually deal with different problemes. In Teboul et al.'s paper, an image is labeled via parsing a given shape grammar. The structure of this

*Fuzhang.Wu@nlpr.ia.ac.cn
†yandongming@gmail.com
‡Weiming.Dong@nlpr.ia.ac.cn
§Xiaopeng.Zhang@nlpr.ia.ac.cn
¶pwonka@gmail.com

Figure 2: *Facade examples that show instances where the model of Mueller et al. would lead to the splitting of facade elements. (a) has large elements in the first floor and smaller windows in the top floors. There is no meaningful grid structure. (b) The door spans two floors and the columns on the top span two floors. The columns themselves also create an interleaved grid of elements that cannot be modeled. e) The irregular arrangements of the windows makes it impossible to find vertical splits that do not cut through a window. e) The structure around the door and the fine glass paneling make splitting into tiles impossible.*

grammar is almost completely pre-determined. In contrast, we attempt to solve an inverse problem. Our algorithm takes a labeled facade image (layout) as input and tries to automatically extract the smallest grammar. When finding the solution for our problem, we tried a Q-learning method to optimize our objective function. But since we extract an N-split grammar, the state always has too many actions to choose during the exploration. This leads to the algorithm taking a long time to converge or being unable to search for the smallest grammar. To avoid this drawback, we propose an approximate DP algorithm to solve our optimization problem and the results show that this algorithm works well.

5 Comparison to Zhang et al.

Here, we compare our work to the recent work of Zhang et al. [Zhang et al. 2013]. The goal of both papers is similar. They both take a two-dimensional box abstraction of a facade layout. Both papers attempt to infer a facade model that is useful for modeling and analysis.

- Representation: One main idea of Zhang et al. is the use of grids to structure a facade image. Particularly clever is the idea of using grids with irregular spacing as a design primitive. This way to structure the data has a lot of potential. By contrast, our paper does not propose a new representation for facade layouts; we simply rely on an existing one.
- Analysis: To derive the hierarchical structure, Zhang et al. introduce a new symmetry score. This is also a novel contribution of their paper, but it is controversial. It is unclear why mirror symmetry should be a driving force behind facade analysis. We would argue that our choice of translational symmetry is much more in line with how architecture is constructed and analyzed.
- Methodology: Zhang et al. propose to use greedy search and genetic algorithms. Our approach is based on approximate dynamic programming. We believe that genetic algorithms are much simpler to design than approximate dynamic programming algorithms, but they also have many drawbacks documented in the literature. A direct comparison of the algorithms is not possible however.
- Procedural Modeling: Zhang et al. have some nice initial examples on how to resize facade layouts. However, the necessary technical framework on how to resize / retarget / reshuffle layouts is not fully developed yet. This technical content might be beyond the scope of a single paper, but until it is developed, the challenging aspects of alignment, resizing, and overlap avoidance might be obstacles in adopting the proposed representation for procedural modeling.
- Test Datasets: Zhang et al. built a very nice test dataset that is one order of magnitude larger than ours. However, the amount of detail in the data is lower. Zhang et al. only extract windows, so that some of the simpler facades are constructed out of fewer than 10 boxes. By contrast, we consider ornaments, ledges, and pillars in the design. That makes the problem more challenging, because a lot of layout questions arise from the relative positioning of windows with respect to ornaments and pillars. It is not clear from the description in Zhang et al. how and if the method would actually be able to handle facade designs where there is not a 1:1 relationship between windows, pillars, and ornaments. Such a non-trivial relationship exists in at least half of our test dataset and we typically have an order of magnitude more details (boxes) in each facade that we consider to be part of the design.

6 More Extensive Parameter Evaluation

We present a more extensive parameter evaluation in Table 1.

7 Challenges

We would like to illustrate some challenges using example configurations. A very powerful idea for automatically generating compact string grammars is a simple greedy heuristic. Long subsequences that occur multiple times are replaced by a new non-terminal and a new rule is added for this non-terminal. The classical example is shown below:

```
Start → abcd abcd abce abef
```

The heuristic replacement considers the lengths of subsequences and the number of occurrences. In this example, *ab* occurs four times, *abc* occurs three times, and *abcd* occurs two times. One popular heuristic would suggest to replace *abc* as the best tradeoff between sequence length and number of occurrences:

```
Start → Ad Ad Ae abef
A → abc
```

There are many variations of this fundamental idea, but most algorithms use the idea of identifying and replacing long repeated substrings. When applying this idea to facade layouts, we encounter multiple problems. 1. Iteratively replacing substrings in 1d string grammars transforms one valid grammar to another valid grammar. This is not the case for facade layouts. A general region cannot be explained by a single rule (See Fig. 3). 2. The process of region replacement by itself does not directly lead to valid grammars. Most importantly, it is not clear how different rule types will emerge from a region replacement process. Additionally, region replacement can

param	*Expert Users*												*Non – expert Users*								
	User1			*User2*			*User3*			*User4*			*User5*			*User6*			*User7*		
	P	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*
$\|\alpha\|$=0	0.45	0.53	0.48	0.47	0.66	0.53	0.37	0.45	0.4	0.38	0.37	0.36	0.28	0.37	0.29	0.33	0.27	0.28	0.24	0.16	0.18
$cost_{OP}$ = 0	0.53	0.5	0.51	0.49	0.56	0.51	0.46	0.47	0.46	0.45	0.35	0.38	0.32	0.31	0.29	0.39	0.28	0.32	0.33	0.18	0.22
split = 0.1, *repeat* = 0.1	0.59	0.55	0.56	0.58	0.66	0.61	0.52	0.54	0.52	0.48	0.38	0.41	0.37	0.37	0.35	0.41	0.29	0.33	0.33	0.18	0.23
split = 0.1, *repeat* = 0.5	**0.73**	**0.72**	**0.71**	**0.73**	**0.85**	**0.77**	0.54	**0.57**	**0.55**	**0.57**	**0.46**	**0.49**	**0.42**	**0.44**	**0.4**	0.45	**0.32**	**0.37**	**0.35**	**0.19**	**0.24**
split = 0.1, *repeat* = 1	0.62	0.58	0.59	0.56	0.65	0.59	0.52	0.54	0.52	0.48	0.36	0.40	0.37	0.37	0.35	0.43	0.29	0.34	0.34	0.18	0.22
split = 0.1, *repeat* = 10	0.31	0.46	0.36	0.29	0.51	0.36	0.27	0.43	0.32	0.27	0.34	0.29	0.17	0.31	0.21	0.25	0.26	0.24	0.17	0.15	0.15
split = 1, *repeat* = 0.1	0.67	0.62	0.64	0.66	0.76	0.69	0.54	0.55	0.53	0.53	0.41	0.45	0.41	0.41	0.38	0.46	0.31	0.36	0.33	0.18	0.23
split = 1, *repeat* = 1	0.67	0.63	0.64	0.67	0.76	0.70	**0.55**	0.56	**0.55**	0.54	0.43	0.46	0.41	0.42	0.38	**0.47**	**0.32**	**0.37**	0.34	**0.19**	0.23
split = 1, *repeat* = 10	0.34	0.49	0.39	0.34	0.58	0.41	0.28	0.42	0.32	0.30	0.36	0.31	0.20	0.34	0.24	0.28	0.26	0.26	0.18	0.15	0.16
split = 10, *repeat* = 0.1	0.67	0.63	0.64	0.71	0.81	0.74	**0.55**	0.56	**0.55**	0.54	0.43	0.46	0.41	0.43	0.39	0.46	0.31	0.36	0.34	0.18	0.23
split = 10, *repeat* = 1	0.67	0.64	0.64	0.71	0.81	0.74	**0.55**	**0.57**	**0.55**	0.56	0.44	0.48	0.41	0.42	0.39	0.45	0.31	0.35	**0.35**	**0.19**	**0.24**
split = 10, *repeat* = 10	0.69	0.63	0.65	0.72	0.80	0.74	0.54	0.55	0.54	0.56	0.44	0.48	**0.42**	0.43	**0.4**	0.43	0.29	0.34	0.34	0.18	0.23

Table 1: *We use the precision-recall test to evaluate different parameters for our grammars. From the table, we can notice that the grammars become worse when we omit either the first term (cost of the rule) or the second term (cost per symbol in the rule) of the proposed cost function. Overall, the grammars tend to be more similar with the expert user grammars when the repeat rule is cheaper than the split rule.*

$cost(repeat)$ = 0.5				$cost(split)$ = 0.1			
split	*#rule*	*#split*	*#repeat*	*repeat*	*#rule*	*#split*	*#repeat*
0.2	23	18	5	0.2	25	20	5
1	20	15	5	2	22	19	3
2	19	14	5	5	22	21	1

Table 2: *Statistics for different parameter values. We use the seventh layout from the 10 selected facades in the main paper as input to generate the grammars.*

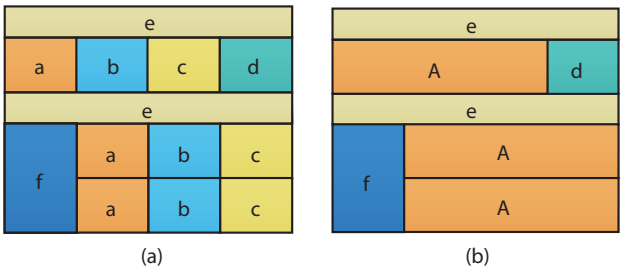


Figure 3: *Iteratively replacing substrings in one-dimensional string grammars transforms one valid grammar to another valid grammar. For two-dimensional facade layouts, this is no longer the case. Neither the region (a) nor the region after replacement (b) can be explained by a single rule.*

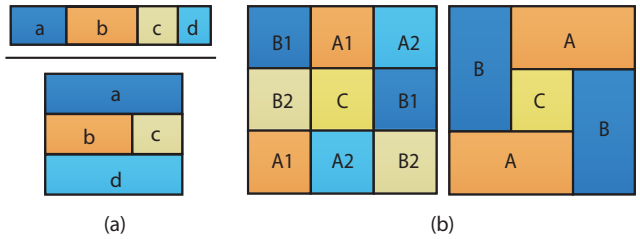


Figure 4: *a) An example of two regions that have the same number of terminal regions (4), but that require grammars of different cost due to the different two-dimensional arrangements. b) After replacing regions A1, A2, B1, and B2 the design becomes locked, i.e., no splitting rules can explain the layout.*

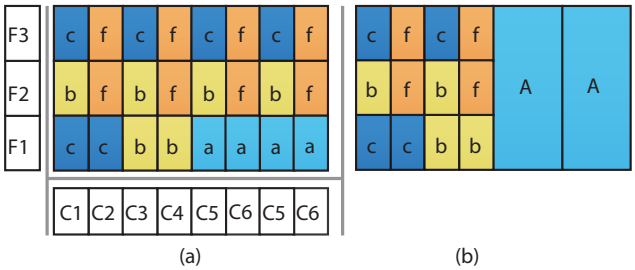


Figure 5: *We show a layout (left) with three floors and eight columns. Additionally, we show the abstraction of the layout as a sequence of columns $C1, C2, C3, C4, C5, C6, C5, C6$ and a sequence of floors $F1, F2, F3$. Analyzing the floors would result in a split into three separate floors. Analyzing the columns would detect a repetition of $C5C6$ and group these columns together (right). Treating the facade as two one-dimensional problems gives no hint at the correct solution, i.e., that the floors $F2$ and $F3$ have to be processed together to extract a repeat pattern that spans both floors.*

generate configurations that are locked and cannot be split in vertical or horizontal directions (See Fig. 4). 3. String length is a useful and straightforward measure to derive cost heuristics. In facade layouts, the analogue would be the number of terminal regions in a compound region. However, depending on the arrangement of the regions, the cost of processing a compound region differs. See Fig. 4. 4. While we ultimately solve the two-dimensional problem with rules that split in only one direction, the problem is inherently two-dimensional and cannot be solved by breaking it down into one-dimensional problems in a straightforward manner. In Fig. 5, we show an example layout together with its abstraction as column and floor sequence. Processing the design as either columns or floors does not lead to a meaningful solution. 5) Facade layouts are not regular grids but more general box layouts.

8 Performance on automatically generated facade segmentations

Since our algorithm only considers the labeled terminal regions as the input, any facade layout with correct labels can be used as the input, no matter what kinds of tools (automatic or semi-automatic tools) they are generated with. However, we pass all automatically generated input layouts to our QP algorithm to regularize the facade (improve alignment, spacing, and sizing of elements) in a preprocess. The regularization of the input layout is not a part of the core algorithm presented in the paper. In the following examples,

Figure 6: *Three facades that were segmented by an automatic segmentation algorithm [Teboul et al. 2010]. We use these three facades to automatically extract grammars.*

F	#T	PL		BGL		SM		Ours	
		cost	#rule	cost	#rule	cost	#rule	cost	#rule
(a)	147	149.4	24	152	20	172.3	33	**131.2**	36
(b)	71	77.5	15	76.9	9	77.3	13	**61.1**	11
(c)	98	115.9	19	105	10	143.6	36	**87.1**	19

Table 3: *We compare the compactness of the extracted grammars to other methods (PL [Weissenberg et al. 2013], BGL [Martinovic and Van Gool 2013], and SM [Zhang et al. 2013]). From the table, we can observe that our algorithm generates good results even with an automatic segmentation as input.*

we use the automatic facade segmentation generated by Teboul et al. [Teboul et al. 2010] as the input layout. We randomly choose three facades, see Fig. 6. Using our cost function, our algorithm still performs best compared with alternative methods, see Table 3). We believe that our algorithm is completely independent of the nature of the input layout, but it is not independent from the regularization done in a pre-process. The other algorithms also perform better with our regularization. We therefore use the regularized layout as input for all the different methods in our tests.

9 Note on Carrascosa et al.

A challenge of our formulation is that even though we structure the facade as nested one-dimensional structures, the problem is fundamentally two-dimensional and two of the dominant heuristics used in the one-dimensional case do not work directly [Carrascosa et al. 2010]. First, greedy grouping of repeated sub-regions does not usually lead to a valid grammar. Second, the nice dynamic programming solution to the word occurrence problem of Carrascosa et al. [2012] is NP-complete in our two-dimensional setting and cannot be solved in closed form.

	Automatic method					Expert				Non − expert		
	PL	BGL	SM	IPM	Ours	U1	U2	U3	U4	U5	U6	U7
S	5.11	6.31	4.03	7.03	8.23	7.69	8.86	7.2	6.8	6.4	5.31	3.57
E	0.2	0.34	0.11	0.49	0.77	0.51	0.83	0.43	0.43	0.29	0.23	0.06
C	136.3	123.5	173.1	109.1	77.4	86	79.4	98.8	103.4	126.8	126.1	154.2

Table 7: *We invited seven expert users to evaluate both the automatically and manually extracted grammars from five facades. Each user was asked to score each grammar on a scale from 1 to 10. We report the average score for all facades and all users in the row S. To calibrate the scores, we also asked each user if the result can be considered to be expert work / high quality (reported in row E).*

10 Robustness evaluation

In this section, we present more additional results to evaluate the robustness of our method. We first show a series of examples in which more and more noise is added into an initially good automatic parsing result of a facade (see Fig. 13). This will illustrate the point where the automatic regularization fails. We can do that for different imperfections (e.g. position and size).

Next, we show how the grammar size increases as the facade gets more complex and when the algorithm fails in case the input is not splitable (see Fig. 14). This is a global property of the input layout. If at least one splitting solution exists, the algorithm will produce some output.

References

Carrascosa, R., Coste, F., Gallé, M., and Infante-Lopez, G. 2010. Choosing word occurrences for the smallest grammar problem. In *Proceedings of the 4th international conference on Language and Automata Theory and Applications*, 154–165.

Carrascosa, R., Coste, F., Gallé, M., and Infante-Lopez, G. 2012. Searching for smallest grammars on large sequences and application to DNA. *Journal of Discrete Algorithms 11*, 62 – 72.

Martinovic, A., and Van Gool, L. 2013. Bayesian grammar learning for inverse procedural modeling. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, 201–208.

Müller, P., Zeng, G., Wonka, P., and Gool, L. V. 2007. Image-based procedural modeling of facades. *ACM TOG (SIGGRAPH) 26*, 3, 85:1–85:9.

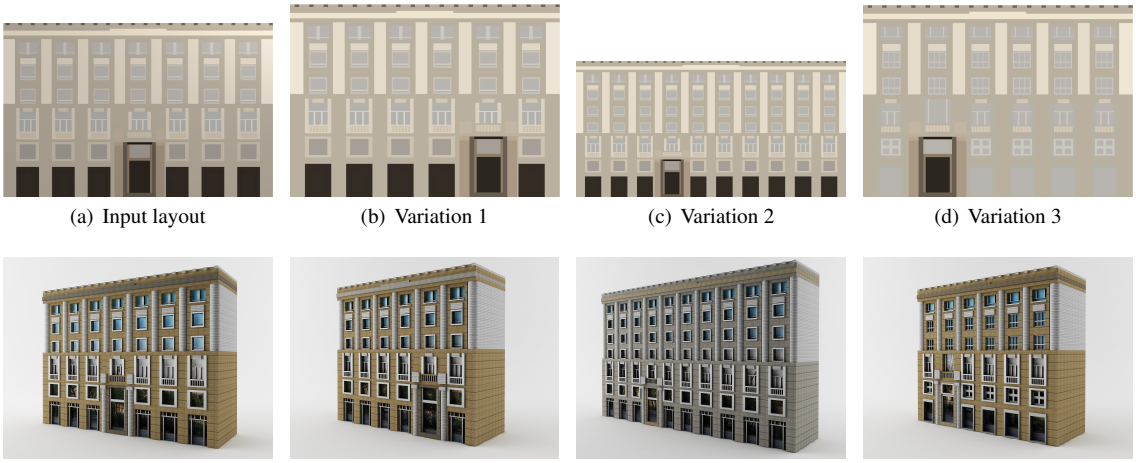
Talton, J. O., Yang, L., Kumar, R., Lim, M., Goodman, N. D., and Mech, R. 2012. Learning design patterns with bayesian grammar induction. In *UIST*, 63–74.

Teboul, O., Simon, L., Koutsourakis, P., and Paragios, N. 2010. Segmentation of building facades using procedural shape priors. In *CVPR*, 3105 –3112.

Teboul, O., Kokkinos, I., Simon, L., Koutsourakis, P., and Paragios, N. 2011. Shape grammar parsing via reinforcement learning. In *CVPR*, 2273–2280.

Weissenberg, J., Riemenschneider, H., Prasad, M., and Van Gool, L. 2013. Is there a procedural logic to architecture? In *CVPR*, 185–192.

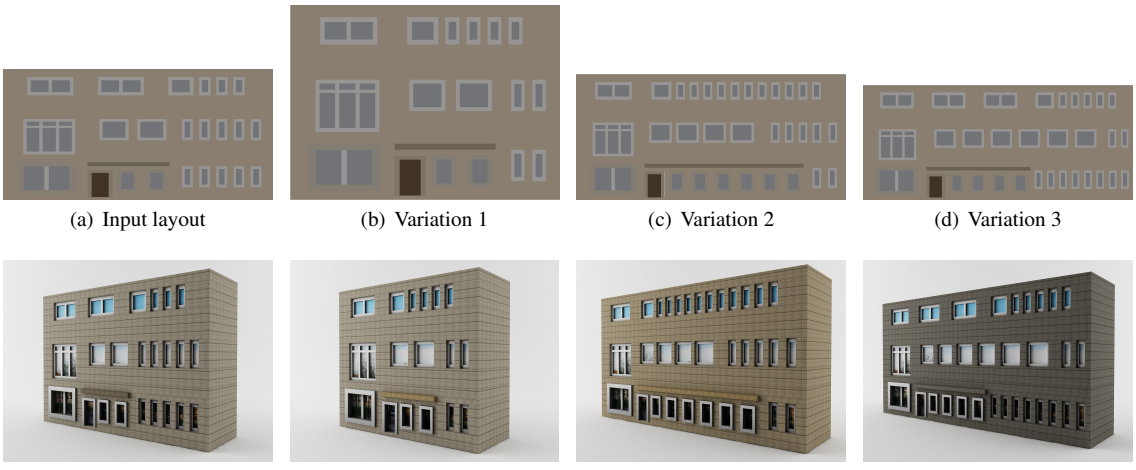
Zhang, H., Xu, K., Jiang, W., Lin, J., Cohen-Or, D., and Chen, B. 2013. Layered analysis of irregular facades via symmetry maximization. *ACM TOG (SIGGRAPH) 32*, 4, 121:1–121:10.



(a) Input layout (b) Variation 1 (c) Variation 2 (d) Variation 3

(e) Input layout (f) Variation 1 (g) Variation 2 (h) Variation 3

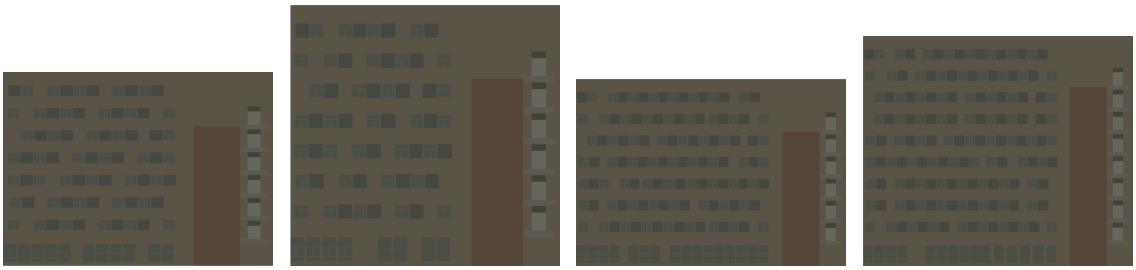
Figure 7: *An example of resizing variations generated from the input layout. (e)-(h) show the corresponding rendered results.*



(a) Input layout (b) Variation 1 (c) Variation 2 (d) Variation 3

(e) Input layout (f) Variation 1 (g) Variation 2 (h) Variation 3

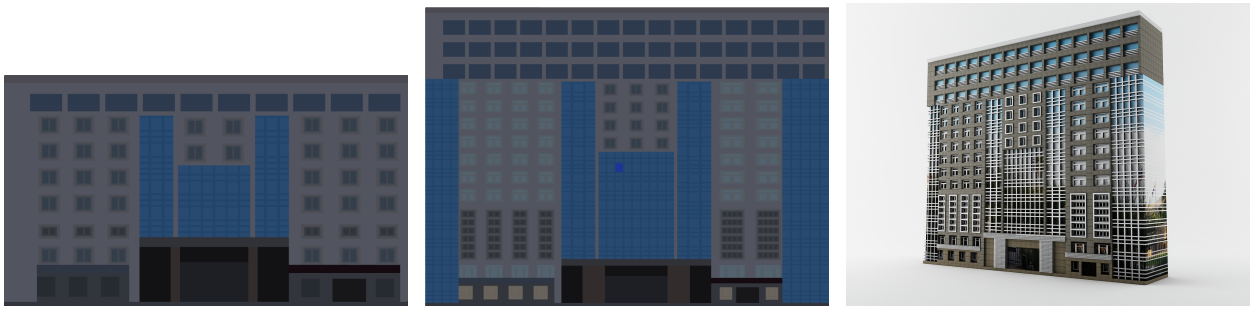
Figure 8: *An example of resizing variations generated from the input layout. (e)-(h) show the corresponding rendered results.*



(a) Input layout (b) Variation 1 (c) Variation 2 (d) Variation 3

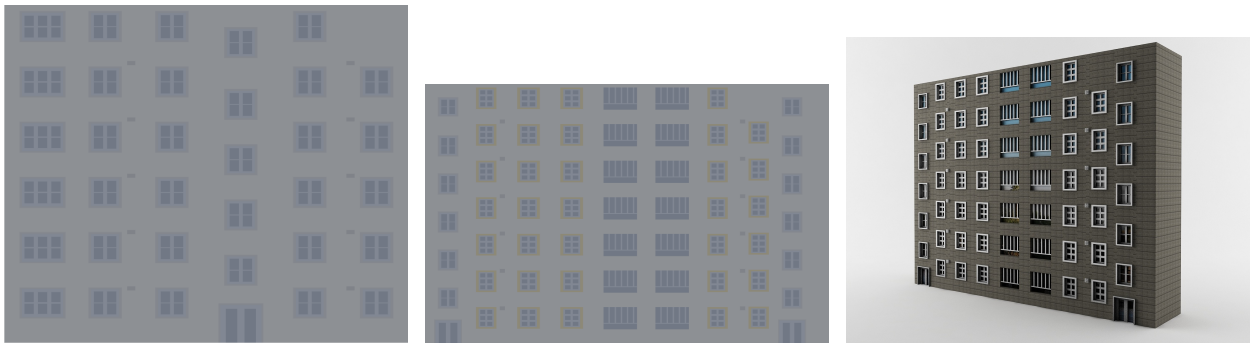
(e) Input layout (f) Variation 1 (g) Variation 2 (h) Variation 3

Figure 9: *An example of resizing variations generated from the input layout. (e)-(h) show the corresponding rendered results.*



(a) Input layout (b) Variation after grammar editing (c) Rendered result

Figure 10: *An example of grammar editing.*



(a) Input layout (b) Variation after grammar editing (c) Rendered result

Figure 11: *An example of grammar editing.*

F	#T	ADP(2K)			ADP(10K)			Importance Sampling (2K)			Importance Sampling (10K)			Greedy		
		cost	*#rule*	*time(s)*	*cost*	*#rule*	*time(s)*	*cost*	*#rule*	*time(s)*	*cost*	*#rule*	*time(s)*	*cost*	*#rule*	*time(s)*
(a)	357	**105.5/105.5**	33/33	1.7/1.8	**105.5/105.5**	33/33	8.1/8.9	**105.5/113**	33/37	1.9/2.0	105.6/113.1	34/37	9.5/10.1	113.8	36	0.03
(b)	409	**33.7/33.7**	11/11	0.8/0.8	**33.7/33.7**	11/11	3.6/3.7	**33.7/33.7**	11/11	0.8/0.8	**33.7/33.7**	11/11	3.8/3.9	**33.7**	11	0.02
(c)	313	72.2/81.3	20/24	1.7/1.8	**72.0/81.7**	18/23	8.3/8.6	80.9/86.4	23/26	2.0/2.0	78.2/83.8	20/24	9.6/9.9	83.8	20	0.02
(d)	636	138.3/139.8	47/48	3.5/3.6	**134.6/138.4**	44/47	17.0/17.6	144.6/151.6	50/54	4.0/4.3	143.3/152.0	47/52	20.5/20.9	151.4	48	0.05
(e)	950	**45.1/45.5**	17/17	0.7/0.7	**45.1/45.1**	17/17	3.7/3.7	**45.1/51.1**	17/21	0.8/0.8	47.2/50.0	18/20	3.9/4.0	55.5	21	0.02
(f)	294	51.9/52.6	15/16	0.8/0.9	**51.3/52.5**	15/16	4.0/4.3	51.4/59.6	16/20	0.8/0.9	54.1/67.7	19/24	4.8/4.9	58.6	20	0.01
(g)	348	**67/71.1**	23/24	1.1/1.1	**67/71**	23/24	5.3/5.6	70.4/75.1	24/26	1.2/1.2	69.3/73.4	23/25	5.9/6.1	83.3	27	0.05
(h)	823	**40.1/40.9**	13/13	0.8/0.8	**40.1/41.4**	13/13	3.8/4.0	44/46.8	14/16	0.8/0.9	44/46.4	13/15	3.7/4.2	40.8	12	0.04
(i)	1296	**61.9/61.9**	19/19	1.0/1.1	**61.9/61.9**	19/19	4.9/5.5	**61.9/65.6**	19/20	1.1/1.2	**61.9/66.2**	19/21	5.4/5.9	65.9	19	0.03
(j)	618	**50.0/50.0**	14/14	1.2/1.2	**50.0/50.0**	14/14	5.6/5.9	**50/50**	14/14	1.4/1.5	**50.0/50.0**	14/14	6.5/6.7	**50**	14	0.03
(k)	495	**30.6/33.4**	10/12	1.1/1.2	**30.6/32.9**	10/11	4.5/5.0	**30.6/38.0**	10/14	1.4/1.4	**30.6/37.1**	10/12	5.9/6.2	31.2	10	0.02
(l)	497	**50.8/53.3**	11/12	1.3/1.3	**50.8/51.67**	11/11	5.6/6.1	**50.8/53.8**	11/12	1.3/1.4	**50.8/53.1**	11/12	6.0/6.4	55.9	13	0.07
(m)	2040	**16.8/16.8**	8/8	1.2/1.2	**16.8/16.8**	8/8	3.5/4.1	21.6/22.5	10/10	0.9/0.9	21.6/22.8	10/10	3.2/3.5	25.7	11	0.2
(n)	185	111.3/125.5	34/41	5.2/5.5	**107.3/123.3**	30/40	23.7/25.1	110.7/132.2	33/43	7.0/7.2	110.3/127.3	29/40	32.8/34.3	112.3	29	0.08
(o)	634	**68.7/68.7**	17/17	1.1/1.2	**68.7/68.7**	17/17	5.4/5.9	76.9/85.0	19/24	1.3/1.4	71.8/86.6	18/24	6.2/6.8	74.8	18	0.04
(p)	816	**56.9/63.5**	19/19	0.8/0.9	**56.9/62.7**	19/19	3.9/4.3	61.6/65.2	20/21	0.9/0.9	61.5/64	19/20	4.3/4.7	79.8	22	0.01
(q)	655	**48/51.7**	13/14	0.9/0.9	**48/51.2**	13/14	4.2/4.4	50.2/55.6	14/16	0.9/1.0	50.2/55.52	14/15	5.0/5.1	48.9	11	0.02
(r)	514	**74.3/79.8**	15/18	2.3/2.3	**74.3/77.1**	15/16	10.8/11.3	**74.3/84.2**	15/20	2.6/2.8	**74.3/85.4**	15/20	13.3/13.9	119.8	30	0.02
(s)	312	**67.2/67.9**	16/17	1.0/1.1	**67.2/69.2**	16/18	5.3/5.7	72.5/81.0	19/23	1.2/1.3	74.7/82.8	21/24	6.3/6.4	69.2	16	0.02
(t)	60	**58/58.6**	18/18	2.8/3	**58/60.2**	18/19	13.6/15.9	**58/63.4**	18/21	4.6/4.8	**58/61.6**	18/20	22.6/23.5	96	28	0.03
(u)	68	59.1/65.9	13/16	2.5/2.6	**57.4/65**	13/15	12.8/13.2	59.4/85.4	16/26	3.3/3.3	65.3/87.4	15/25	15.7/16.2	61	12	0.02
(v)	57	**60/68.82**	10/16	1.6/1.7	**60/65.8**	10/14	7.9/8.1	63.1/75.9	11/20	2.0/2.1	64.4/80.39	14/23	9.9/10.3	101.5	25	0.01
(w)	267	78.6/82.8	17/19	1.1/1.2	**77.5/81.5**	17/19	5.6/5.9	77.7/86.2	17/21	1.2/1.3	78.6/92.5	18/24	6.1/6.3	80.8	20	0.01
(x)	1266	**112.8/114.9**	38/39	3.5/3.7	**112.8/115.1**	38/40	16.8/17.5	116/121.4	40/42	4.4/4.6	**112.8/119.12**	38/41	20.4/21.0	134.6	40	0.1
avg	579	64.9/68.1	18/20	1.6/1.7	**64.4/67.5**	18/19	7.8/8.3	67.1/74.2	19/23	1.9/2.08	67.1/74.6	19/23	9.6/10	76.1	21	0.03

Table 4: *Comparison details for different methods for the facades not evaluated in the paper. See Fig. 15 for the input layouts. Best values are highlighted. We report values for ten different runs. If a cell has two values, the first value is the optimum from among the ten runs. The second value is the average. The compared methods are the following. ADP(2K): approximate dynamic programming with 2K runs. ADP(10K): approximate dynamic programming with 10K runs. Importance Sampling(2K) and Importance Sampling(10K) and Greedy are described in the paper.*

F	*Automatic grammar*					*Manual grammar*						
	PL	*BGL*	*SM*	*IPM*	*Ours*	*User1*	*User2*	*User3*	*User4*	*User5*	*User6*	*User7*
(a)	125.5	130.9	141.9	163.2	**105.5**	110.8	**105.5**	141.3	140.4	167.7	149	169.7
(b)	65.5	49.6	127.9	55	**33.7**	37.9	**33.7**	63.8	44.4	52.2	128.1	64.5
(c)	122.9	124.2	137.7	95.1	**72**	78	76.2	77.1	105.8	145	113.6	157.3
(d)	183.2	174.6	328.9	171.6	134.6	168.8	**120.4**	148.7	156.4	209.5	197.3	225.3
(e)	76.4	70.8	84.5	60.8	**45.1**	49.2	48	59.8	52.4	64.6	55.6	68.4
(f)	65.2	73.1	71.7	63.2	**51.3**	52.3	52.3	53.4	55.6	60.7	65.5	71.2
(g)	106.2	93.5	147.9	97.5	67	**66.5**	69.6	78.6	75.9	105.7	116.6	130.9
(h)	68.9	59.6	77	58.2	**40.1**	42.3	40.9	41.3	43.5	53.7	43.5	76.8
(i)	105.3	97.1	127.5	77.3	**61.9**	78.6	67.2	83.9	70.3	83.7	69.2	90.5
(j)	116	125.1	141.4	74.6	**50**	53	56	50	51.1	176.3	73.3	66.9
(k)	52.4	49.8	74.2	48.2	**30.6**	**30.6**	32.2	32.2	75.5	45.1	123.2	140.9
(l)	139.4	148.6	134.4	65.1	**50.8**	55.7	59.8	57.8	62.2	90.6	76.1	84.8
(m)	61.7	60.4	98.1	31.8	**16.8**	18	18	20	20	27.8	25.6	24.9
(n)	166.3	153.2	219.1	116.1	**107.3**	126.8	**107.3**	119.6	122.8	126.7	154.2	145.1
(o)	105.4	97.7	104.7	83.7	**68.7**	75.4	**68.7**	74.7	84.2	84.3	91.2	95.5
(p)	82.4	81.7	110.9	83	**56.9**	57.2	69.7	74	86.4	86.5	81.8	101.8
(q)	83.5	61.7	84.9	87.6	**48**	54.2	55.6	77.7	82.9	59.1	80.7	84.1
(r)	92.3	86.1	141	100.6	**74.3**	74.6	78.3	77.2	89	119.4	93.4	159.1
(s)	86	73.9	83.8	73.3	**67.2**	71.1	68.7	72.2	75.8	73.3	73.6	101.4
(t)	88.8	75.1	93	101.7	**58**	61.2	59	58	61.2	61.6	107.2	102.3
(u)	75.4	62.6	113.2	66	**57.4**	58.3	60.9	59.1	62.4	64.2	119.1	73.2
(v)	69.2	63.7	86.7	64.8	**60**	63.7	63.7	97.7	101	63.7	70.4	129.9
(w)	93.4	90.2	133.5	88.4	**77.5**	80.8	85.3	84.7	94.5	100.4	93.9	104.3
(x)	255.7	227.9	336.1	293.1	**112.8**	140.5	114.2	121.4	202.6	203	211	390.3
(avg)	103.6	97.1	133.3	92.5	**64.5**	71.1	67.1	76	84	96.9	100.5	119.1

Table 5: *We compare the compactness of the grammar to manually generated grammars by seven users and other automatic algorithms: PL [Weissenberg et al. 2013], BGL [Martinovic and Van Gool 2013], SM [Zhang et al. 2013], and IPM [Müller et al. 2007]. We report the cost in each cell of the table. The best values are highlighted in bold.*

F	*#N − terminal*					*#Common*				*Precision*				*Recall*			
	Ours	*PL*	*BGL*	*SM*	*Expert*	*Ours*	*PL*	*BGL*	*SM*	*Ours*	*PL*	*BGL*	*SM*	*Ours*	*PL*	*BGL*	*SM*
(a)	59	89	45	57	58	43	23	23	21	**0.73**	0.26	0.51	0.37	**0.74**	0.4	0.4	0.36
(b)	21	31	20	47	20	12	2	1	1	**0.57**	0.06	0.05	0.02	**0.6**	0.1	0.05	0.05
(c)	53	48	23	36	53	30	3	3	8	**0.57**	0.06	0.13	0.22	**0.57**	0.06	0.06	0.15
(d)	79	95	48	181	89	45	43	41	29	0.57	0.45	**0.85**	0.16	**0.51**	0.48	0.46	0.33
(e)	29	23	20	80	30	27	15	15	5	**0.93**	0.65	0.75	0.06	**0.9**	0.5	0.5	0.17
(f)	33	22	19	34	31	31	9	19	9	0.94	0.41	**1**	0.26	**1**	0.29	0.61	0.29
(g)	41	31	24	44	39	39	13	9	11	**0.95**	0.42	0.38	0.25	**1**	0.33	0.23	0.28
(h)	19	14	14	71	23	17	8	3	1	**0.89**	0.57	0.21	0.01	**0.74**	0.35	0.13	0.04
(i)	34	59	20	91	48	27	5	13	8	**0.79**	0.08	0.65	0.09	**0.56**	0.1	0.27	0.17
(j)	48	63	35	37	56	45	1	11	8	**0.94**	0.02	0.31	0.22	**0.8**	0.02	0.2	0.14
(k)	22	47	43	50	22	22	1	2	6	**1**	0.02	0.05	0.12	**1**	0.05	0.09	0.27
(l)	24	59	14	114	26	24	13	14	3	**1**	0.22	**1**	0.03	**0.92**	0.5	0.54	0.12
(m)	136	21	18	62	155	124	19	18	44	0.91	0.9	**1**	0.71	**0.8**	0.12	0.12	0.28
(n)	99	140	37	96	98	32	7	1	11	**0.32**	0.05	0.03	0.11	**0.33**	0.07	0.01	0.11
(o)	32	20	18	41	54	25	11	18	13	0.78	0.55	**1**	0.32	**0.46**	0.2	0.33	0.24
(p)	39	20	10	38	27	17	3	6	4	0.44	0.15	**0.6**	0.11	**0.63**	0.11	0.22	0.15
(q)	29	24	11	40	25	19	6	6	4	**0.66**	0.25	0.55	0.1	**0.76**	0.24	0.24	0.16
(r)	27	40	24	55	45	11	10	13	1	0.41	0.25	**0.54**	0.02	0.24	0.22	**0.29**	0.02
(s)	24	27	16	35	30	16	3	16	13	0.67	0.11	**1**	0.37	**0.53**	0.1	**0.53**	0.43
(t)	23	37	23	38	25	23	4	4	1	**1**	0.11	0.17	0.03	**0.92**	0.16	0.16	0.04
(u)	18	30	10	33	20	14	1	10	6	0.78	0.03	**1**	0.18	**0.7**	0.05	0.5	0.3
(v)	21	32	7	35	19	16	1	7	1	0.76	0.03	**1**	0.03	**0.84**	0.05	0.37	0.05
(w)	38	60	16	35	37	24	8	16	15	0.63	0.13	**1**	0.43	**0.65**	0.22	0.43	0.41
(x)	70	173	124	142	67	52	1	6	9	**0.74**	0.01	0.05	0.06	**0.78**	0.01	0.09	0.13
avg	42	50	27	62	46	31	9	11	10	**0.74**	0.18	0.41	0.16	**0.67**	0.2	0.24	0.22

Table 6: *We ran a precision-recall experiment to calculate the similarity between our grammar and the expert grammar. For an input facade layout, we calculate the number of non-terminal regions that were extracted by both expert grammar and our automatic grammar, separately. For a comparison, the precision-recall results generated by three other kinds of grammar (procedural logic (PL)[Weissenberg et al. 2013], Bayesian grammar learning (BGL)[Martinovic et al. 2013], symmetry maximization (SM)[Zhang et al. 2013]) are also shown in the table. These precision-recall results imply that our grammars are truly consistent with the expert grammars (ground truth).*

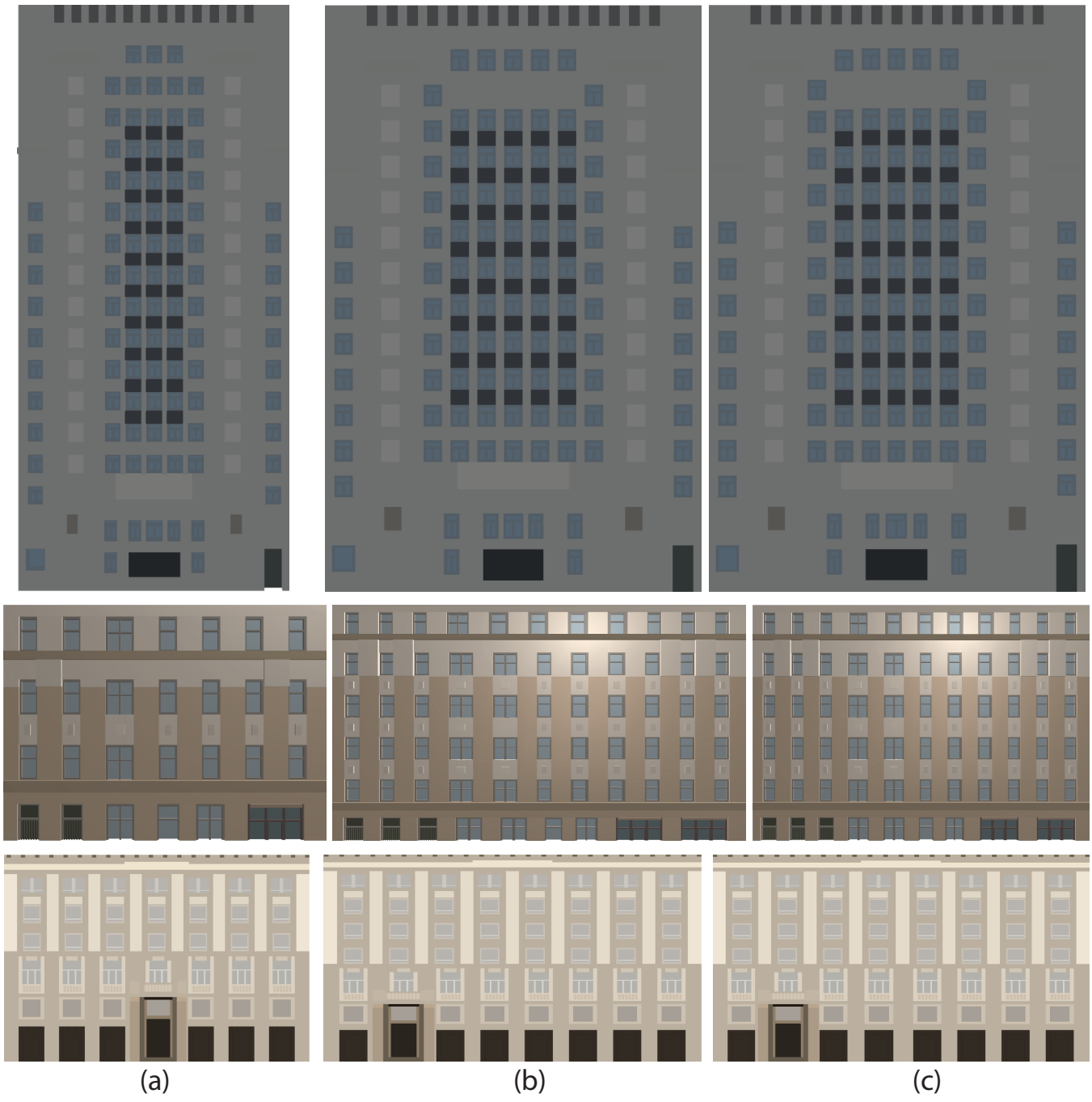


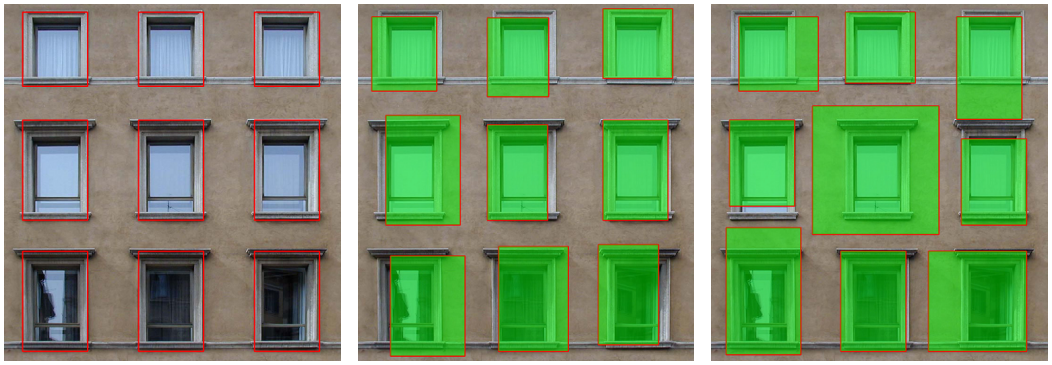
Figure 12: *Alignment Examples. (a) The input layout. (b) The variation result without alignment. (c) The variation result with the alignment post-process.*

(a) *cost* = 32 (b) *cost* = 43.3 (c) *cost* = 51.9

(d) *cost* = 75.7 (e) *cost* = 88.6 (f) *cost* = 82.1

(g) *cost* = 120.8 (h) *cost* = 125.1 (i) *cost* = 155.5

Figure 13: *The grammar sizes of a facade layout with different types of random noise. (a) Input facade layout. (b) One rectangle is randomly moved horizontally (marked as the green rectangle). (c) Two rectangles are randomly moved horizontally. (d) More rectangles are randomly moved vertically and horizontally. (e) Both the rectangles' positions and sizes are randomly changed. (f) Rectangles are randomly deleted (see the blue rectangles). (g) Half of the rectangles are randomly moved. (h) All of the rectangles are randomly moved. (i) All of the rectangles' positions and sizes ares changed.*



(a) *cost* = 18.3 (b) *cost* = 48.2 (c) *cost* = *null*

Figure 14: *The grammar size increases if the input layout (a) has too much noise (b). In extreme cases, the facade layout cannot even be split any more (c).*

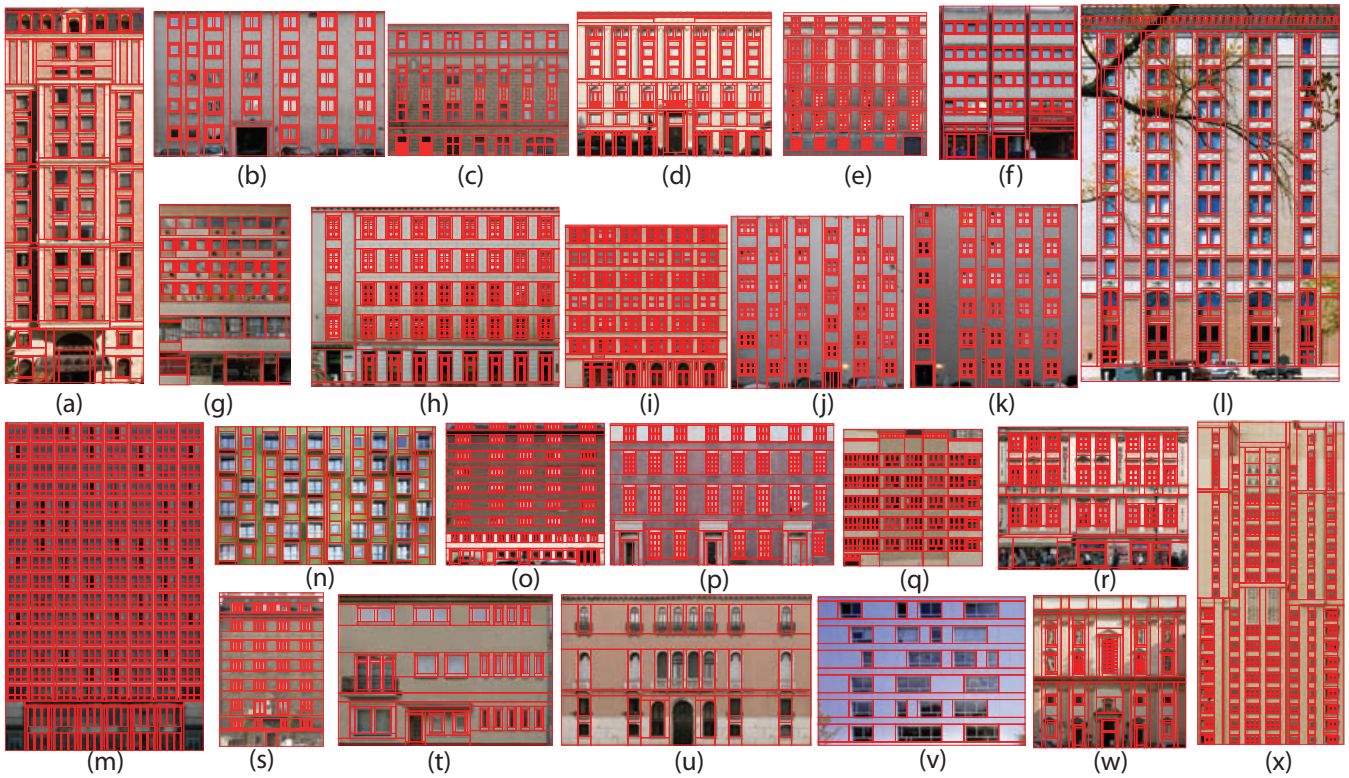


Figure 15: *The facade layouts used in Table 4.*