User-assisted Inverse Procedural Facade Modeling and Compressed Image Rendering

Huilong Zhuo[1], Shengchuan Zhou[2], Bedrich Benes[1], and David Whittinghill[1]

[1] Purdue University, USA,
[2] Qingdao Geotechnical Investigation and Surveying Research Institute, China

Abstract. We take advantage of human intuition by encoding facades into a procedural representation. Our user-assisted inverse procedural modeling approach allows users to exploit repetitions and symmetries of facades to create a split grammar representation of the input. Terminal symbols correspond to repeating elements such as windows, window panes, and doors and their distributions are encoded as the production rules. Our participants achieved a compression factor that averaged 57% (min=12%, max=99%) while taking on average 7 minutes (min=1, max=25) to compress an image. The compressed facades do not suffer from occlusion problems present in the input, such as trees or cars. Our second contribution is a novel rendering algorithm that directly displays the compressed facades in their procedural form by interpreting the procedural rules during texture lookup. This algorithm provides considerable memory savings while achieving comparable rendering performance.

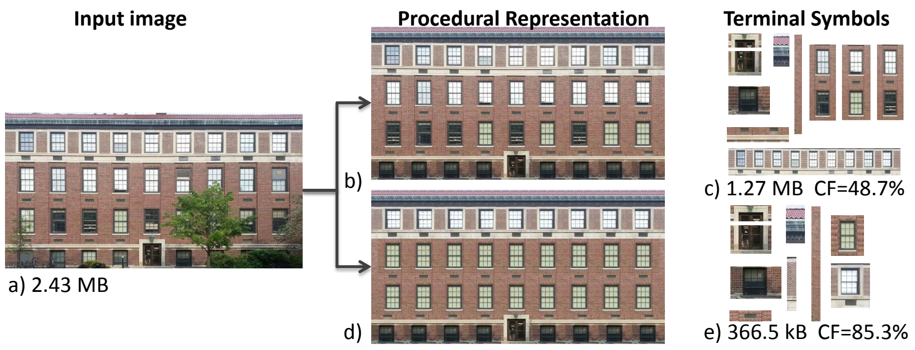


Fig. 1. The user encoded a facade (a) procedurally with attention paid to conveying the varying details (b) resulting in 44 procedural rules, 49% compression, and 10 terminal symbols (c). The same facade was encoded again with less precision (d) resulting in only 30 procedural rules, 85% compression, and 9 terminal symbols (e). The user removed the occluding trees and each encoding took under two minutes. The original image was rendered in 1.32 ms and the direct compressed facade rendering was 1.73 ms for (b) and 1.54 ms for (d).

1 Introduction

The most common facade representation is by texture images which occupy a large amount of memory, often even more than the actual model geometry. Although images can be compressed, few techniques exist for their compressed rendering. Facades contain repeated, structures, making them suitable for procedural representation [1–3]. The goal of inverse procedural modeling (IPM) is to find a procedural representation of an input model [4] and it can be thought of as image compression. Existing IPM methods require a preprocessed input in which the scene includes higher semantic information [5, 6]. This hinders the applicability of IPM in practice and is exacerbated by the large variability of details in facade images.

We present a user-assisted approach for inverse procedural facade modeling. Our key observation is that human visual perception provides global insight into structures and can detect subtle details at varying scales while simultaneously judging their importance. Humans are good at detecting repetitions and, due to previous experiential knowledge, can account for missing data or incoherencies. Humans may not achieve an (algorithmically) optimal facade compression such as [6], but they work directly with unstructured images, they can complete missing information, and they can generate representations that will be perceived as realistic.

We encoded the input image by indicating symmetries and repetitions during an interactive session. The procedural rules are immediately available and the system also shows the resulting compression of the facade as well as the original. The output of this process is a set of repeating elements (terminal symbols of the grammar), and a set of procedural rules that define the distribution of the terminals.

Our second contribution is an approach for direct *compressed* image rendering. We store only the terminal symbols of the grammar while pixel color is calculated using a direct lookup through the generated procedural rules. This presents considerable savings of the GPU memory while maintaining comparable efficiency of rendering. While individual image rendering is slower by an average of 23% depending on the rules' complexity. In an application testing the rendering of multiple houses the compressed facade rendering is slower an average 17%.

Figure 1 shows an input facade that includes missing parts, large variability, and obstacles. This facade was encoded procedurally in less than two minutes. In the upper row the user aimed for a precise representation of the input leading to the compression factor 49% and a high number of 44 procedural rules. The lower row shows the image with compression of 85% and only 30 rules. The higher compression is achieved by sacrificing various unique details. These choices derive directly from aesthetic objectives; an automated system would be poorly suited to achieve acceptable results. Moreover, the user implicitly accounts for missing details.

2 Related Work

Procedural representation has been addressed in Computer Graphics by work ranging from methods for pure procedural generation [2, 7–10] to inverse approaches that attempt to find procedural representation of real buildings [11–13] or even vegetation [14].

Recently, attention has been focused on procedural *facade* representation. Mueller et al. presented what is likely the first method for procedural representation of facades in Computer Graphics in [15]. Their approach builds on shape grammars and allows for an automatic extrusion of facade features into a 3D representation. This approach was later extended to interactive facade editing in [7]. Various approaches aim at automatic generation of a grammar representation based upon an input facade. They can be categorized into techniques that use images as input, or those that use Laser Interferometry Detection and Ranging (LiDAR) data sets. In the first class of algorithms that work with images, there are a number of techniques that attempt to find symmetries within the image's structures. Musialski et. al introduced an automatic approach for facade repair that exploits the facade symmetry in [16]. Teboul et al. addressed segmentation in [17], where they use a combination of grammars, supervised classification, and random walks to synthesize facades and recently a 3D reconstruction was presented in [18].

Grammar generation of facades from input images has been addressed only recently. Boulch et. al. use bottom-up automated parsing to discover attributes from a predefined grammar [19], and similarly Martinović et al. learns attributed context-free grammar using Bayesian networks in [20]. Zhang et al. use an automated approach for automatic grammar generation by using symmetry detection and evaluate different groups and layers in [3]. A similar approach for automatic grammar generation from segmented facades was presented in [6] in which the objectives were to generate minimal grammar and a new facade synthesis. We are also acquiring an inverse procedural representation, but rather than new facade synthesis, we focus on compressed facade rendering. Also, our approach uses human intuition to complete missing parts and irregularities. Moreover, previous approaches require some kind of pre-segmentation and user identification of terminal symbols. This is a common requirement for the majority of inverse procedural approaches such as [4].

LiDAR data have been addressed by a number of previous studies as well. A technique called adaptive recursive facade splitting has been used to reconstruct facades from LiDAR data in [21]. Vanegas et al. used a generalized rewriting rule to describe Manhattan-world reconstructed buildings [22]. Ceylan et. al introduced an image-based 3D building reconstruction framework that focused upon facades in [23]. Wan and Sharf used grammars to aid in 3D reconstruction of faces from scanned urban facades [24]. Kerber et al. used symmetry detection with a feature descriptor for large scale urban scenes in [25]. Li et al. used a combined approach in which images and LiDAR data were used to create depth layers to complete 3D facades in [26].

Close to our approach is the interactive system for facade modeling [27]. Our approach exploits symmetries and user intuition in order to create facade representations. The work [27] differs from ours in that it does not attempt to perform direct facade rendering, nor does it show the actual procedural representation of the input.

3 Split Grammar

The objective of our work is to describe an input image as a context-free attributed split grammar [2, 9, 15]. The split grammar is a quadruple

$$G = \langle \omega, N, T, R \rangle \,, \tag{1}$$

where $\omega \in N$ is the starting symbol of the grammar (the axiom) that usually corresponds to the entire facade. The non-terminal symbols $N = \{N_1, N_2, \ldots, N_{|N|}\}$ represent transformations and intermediate steps and can be thought of as groups or structures. The terminal symbols $T = \{T_1, T_2, \ldots, T_{|T|}\}$ are the final sub-images that compose the input and R is a non-empty set of production rules in the form

$$N(\mathbf{p}) \rightarrow op_1(\mathbf{p})\{x_1(\mathbf{p})\}op_2(\mathbf{p})\{x_2(\mathbf{p})\}\ldots. \tag{2}$$

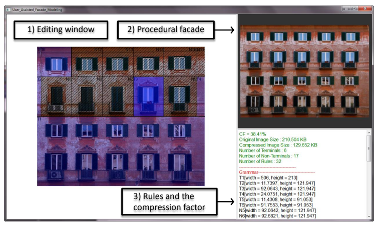
The rules denote rewriting of the non-terminal symbol $N(\mathbf{p})$ from the left-hand side with a sequence of nonterminal, and terminal symbols $x \in N \cup T$. Each symbol on the right-hand side can be modified by an operation op. The operation op can be one of the $splitX$, $splitY$, $subdivX$, $subdivY$, $flipX$, $flipY$, and $rotateX$.

Every (terminal and non-terminal) symbol is associated with a (rectangular) shape. Terminal symbols represent a certain part of the input image, whereas non-terminals correspond to areas that are composed of other non-terminals and/or terminals. We define the *compression* of the input image as

$$cf = \frac{1}{\sum scale(T_i)}, \tag{3}$$

where $scale(T_i)$ measures the relative size of the terminal symbol in the input image. Though we could also add the size of the grammar to the compression, its size is usually significantly smaller than the size of the images and can safely be ignored.

4 Interactive Application



```
CF = 38.41%
Original Image Size : 210.504 KB
Compressed Image Size : 129.652 KB
Number of Terminals : 6
Number of Non-Terminals : 17
Number of Rules : 32
------------------------------
Grammar------------------------------
T1[width = 506, height = 213]
T2[width = 11.7397, height = 121.947]
T3[width = 92.0643, height = 121.947]
T4[width = 24.0751, height = 121.947]
T5[width = 11.4308, height = 91.053]
T6[width = 91.7553, height = 91.053]
N5[width = 92.0642, height = 121.947]
N6[width = 92.6821, height = 121.947]
```

Fig. 2. The user represents the original facade (1) as a procedural representation. The application shows the result in the second window, while the third window (3) shows the procedural rules and the compression factor.

We have developed an application that allows the user to compress a facade during an interactive session and represent it as a split grammar, Eqn. (1). The application has three main windows (see Figure 2). The first is the editing window that shows the input facade and allows all user interaction. The second window displays the facade that is the result of the procedural representation. The image in the second window is always immediately interpreted from the rules. The grammar (non-terminals, terminals, rules) and grammar statistics (compression factor, number of rules) are displayed on the third window. This window is observed by the user when she attempts different procedural representations.

The application implements rules from Section 3, namely terminal and non-terminal node selection, facade horizontal $splitX$ and vertical $splitY$, symbol $flip$, symbol $rotation$, and regular distribution of symbols via the $subdivide$ command.

The user interface shows selected symbols with dark blue color, terminal symbols in lighter blue, and non-terminal symbols with a pattern. In this way the user has visual control over the amount of encoded parts of the input image and can therefore make a better estimate of the total area of terminal symbols that will need to be stored.

The interactive session starts with the complete facade that is represented as the axiom ω. The user selects a symbol and applies an operation that also creates a procedural rule. This process is repeated until the user decides that further modeling is not necessary at which point the procedural model is saved. The result of the session is the grammar G from Eqn. (1) where each terminal symbol is stored as a separate image. The user can create rules that may not be optimal, however we do not perform any rule optimization to avoid biasing the results.

5 Rendering

We present a new approach for direct procedural facade rendering. The terminal symbols and the rules are stored on the GPU and the (u, v) texture coordinates are evaluated directly from the grammar. Depending on the encoding, the compressed image may be significantly smaller than the input image. However, as rule interpretation is required, there is an overhead imposed by the texture color look-up.

There are two possible ways to implement this function: 1) the procedural rules could be interpreted once and the expanded form stored as a long array of terminal symbols and accompanying symbol operations, 2) the rules can be interpreted for every pixel. We have implemented both approaches. Although the first approach would save the overhead of rule evaluation, we observed performance to be significantly slower (2.97ms vs 1.54ms for the texture from Figure 1 d)) on the current GPU architectures (Shader Model 5.0, OpenGL 4.3) that are sensitive to branching and loops. For the fully expanded rules, the GPU must traverse the array to check if the searched pixel belongs inside the displayed region of terminals. This results in extensive branching. The looping and branching operations cannot be unrolled by the compiler and the expanded rule includes various repeating *if* statements that present a significant penalty for rule runtime execution performance. The second approach, where the split rules are stored on the GPU and interpreted for each pixel, was more efficient on current GPUs, but it can be different for future GPU architectures or rendering systems.

Let us recall (see Section 3) that the terminal symbols (images) are denoted by $T = \{T_1, T_2, \ldots, T_{|T|}\}$, non-terminals by $N = \{N_1, N_2, \ldots, N_{|N|}\}$ and the split rules have form $N(\mathbf{p}) \rightarrow op_1(\mathbf{p})\{x_1(\mathbf{p})\}op_2(\mathbf{p})\{x_2(\mathbf{p})\}\ldots$. Each rule is encoded in a data structure that stores the operation, parameters, and the set of terminal and non-terminal symbols on the right-hand side. Each rule has an ID and the rules are stored in a table that is indexed by the left hand side and has the right hand side as the table elements. This table is stored on the GPU.

The rule interpretation begins with the axiom ω and the texture coordinates (u, v) for which the color is queried. The rule with ω on the left-hand side is found in the table, the right-hand side is scanned, and each operation ($split$, $subdivide$, $flip$, etc.) is evaluated. As the operations lead to a smaller part of the ω we check if the searched (u, v) coordinates fall inside the evaluated operation. If yes, the symbol that was on the right-hand side is located and further interpreted. If the new symbol is a non-terminal, the interpretation continues with interpreting the right-hand side of the corresponding rule.

The scanning stops if a non-terminal symbol is found. The (transformed) coordinates of the (u, v) are used to look up the color.

The algorithm has theoretical complexity $\mathcal{O}\big(|R|\big)$, where $|R|$ is the number of rules. This case would correspond to the ω being replaced by the non-terminals covering the same area and eventually ending with an image that is the same size as the input. In a practical implementation the rules split the input image into smaller parts, so the behavior is $\approx \mathcal{O}\big(log_2|R|\big)$. This, of course, depends on the strategy for the creation of the procedural rules that is discussed in Section 6.1.

6 Implementation and Results

The system is implemented in C++ and uses an OpenGL for rendering, shaders were implemented in GLSL 5.0, the system has been tested on Intel i7 CPU clocked at 3.2 GHz with 16GB of memory and a NVIDIA K5000 Quadro GPU.

6.1 User Testing

We tested our approach on ten participants aged 20-30 years. All held undergraduate degree in computer science or a related discipline, and all had some knowledge of procedural modeling. The objective was to interactively encode eleven facades (images available on the project page at hpcg.purdue.edu) while maintaining visual similarity with the input and maximizing the compression factor. We selected test facades such that they 1) provided variability of structures, 2) allowed for comparison with previous work [3, 28], and 3) included facades with occlusion problems. The participants were first introduced to the goal of the task. Next, they completed a short training tutorial that led them step-by-step to compress a simple schematic facade and explained every action and result in detail. There was no time limit on the duration of the training and all participants finished the training in an average of 11 minutes.

The overall compression was 57% with the standard deviation equal to 10%. The maximum compression factor was 99% and the minimum 12%. Higher compression was achieved by visually degrading the input and vice versa. The image can be better compressed by replacing similar windows with some variation of a single instance. We demonstrate this in Figure 4 where the first column shows the original image, the middle column is the figure that corresponds to the highest compression factor, and the last column shows images that have the lowest compression factor. Note the results for the facade #8 are quite subtle, yet the compression went from 65% to 22%. The compression of 99% of the facade 10 is an extreme example and washes out the visual differences.

The results also show a degree of variability among individual subjects. The second column of Figure 4 shows the average compression per participant for the entire set of images (min=22%, max=69%).

The participants spent in average of 7 minutes compressing each image (min=1, max=25). The average number of production rules was 52 (min=4, max=146).

Rule Selection We analyzed the selected rules and discussed the strategy for image compression with the participants in a group discussion. All participants independently tested two different strategies. First they would attempted to isolate the most frequently repeating large elements, such as windows, and convert them into terminal symbols. That is also a common strategy in automated systems [3, 6]. Surprisingly this did not lead to the best compression strategy and most participants quickly discarded this approach as inefficient.

The common strategy to which all participants independently converged was to find the largest symmetrical parts of the image and divide it. This strategy was then repeated until small elements, such as windows, were found. The terminal symbols were then analyzed and some of them were merged to replace windows with small visual changes.

Occlusion Users use their intuition to quickly account for occluded parts of missing structures as can be seen in Figures 1 and 3. Interestingly, users can create similar facades by discarding obstacles as can be seen in Figure 3, where the right side of the facade was replaced with the left one that was completed by non-occluded windows.

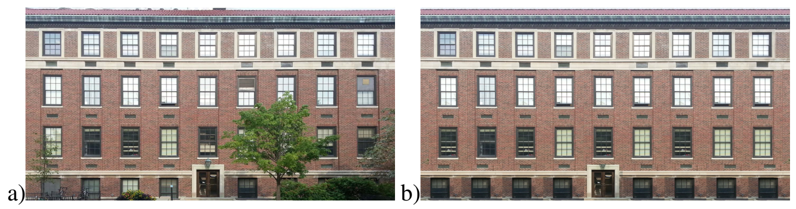


Fig. 3. An image with occluded parts (1) can be quickly resolved by user intuition (b).

Comparison to Previous Work Images 1 and 2 from our testing set (hpcg.purdue.edu) correspond to the images used by [28] (Figures 9 and 11) and Images 4, 6, 7, and 8 are from [3] (Figure 9). While both methods focused on an automatic detection of structural similarities in our work we have completely offloaded the detection of similarities to human. The compression factor of the previous work was not their main aim but a side effect. In order to find the compression factor we have manually calculated the number of terminal symbols and the compression factor of each facade from [28, 3]. Table 1 shows the comparison between A: [28], B: [3], and C: our approach. The compression factors reported by our method are the average numbers from the user study.

Our user-assisted approach leads to better results for all cases. The average CFs are A:32%, B:17%, and C:54%. The number of terminal symbols has the major effect on the size of the final representation and the averages were A:7, B:18, and C:13. It seems the main reason for a higher compression factor of the user-assisted approach is in the human tolerance to variability in the input images. It would likely be difficult to tune an automatic system to be as tolerant to the variance in input images as are humans. Yet, humans produce facades that are visually plausible.

	CF [%]			# of T		
Image	A	B	C	A	B	C
1	31		76	3		8
2	32		71	4		6
4		25	63		10	17
6		17	39		7	11
7		8	24		14	23
8		18	48		5	21

Table 1. Comparison of the compression factor and the number of terminal symbols among A: [28], B: [3], and C: our approach.

6.2 Compressed Image Rendering

Compressed facade rendering comes at an additional cost for extra computation. We have used the result from the user study to demonstrate the performance hit as a function of the number of rules. The rules are different and having the same number of rules can vary significantly in their composition. To account for this we have sorted the results according to the number of rules. We have then calculated the average over intervals of 5 rules, displayed the normalized rendering time.For the worst case of 110 procedural rules the rendering was two times slower than direct image rendering. For the average case of 50 rules the performance was 23% slower (1.234ms vs. 1.525ms).

As rendering an individual image is a rare case, we have also created an example of a real-world scenario. We created a set of 250 different buildings and rendered them with and without compressed facades from the user-generated examples. The average compression factor for the textures from the entire scene was 81%. The loading time was 31s for uncompressed textures and 6s for compressed ones; the rendering time was 1.39ms per frame for uncompressed and 1.62ms per frame for compressed (16% performance hit).

7 Conclusions

We have presented a user-assisted approach for inverse procedural modeling of facade images and their compressed rendering. Our approach treats the facade as a set of pixels and visual consistency is judged by the user. Users quickly account for occluded sections and use their own intuition to compress global symmetries and identify visual differences between similar elements. Images are compressed in an interactive session that takes approximately two minutes per facade depending upon its complexity. During our testing users achieved average compression of 57%. The original image is represented as a set of sub-images and directly rendered by the GPU from the procedural rules. The overhead for the image rendering depends upon the complexity of the procedural representation and was on average 23% for an image composed from 50 procedural rules. In a real-world scenario of a small city rendering, the performance hit was 16%. An interesting observation is that subjects used different strategies for splitting, such as dividing the elements that are usually considered atomic (i.e. windows or doors).

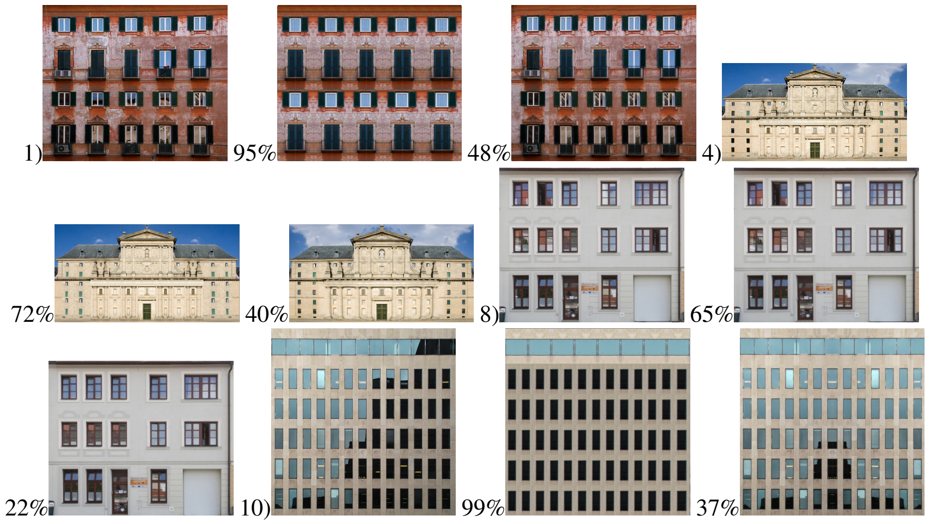


Fig. 4. Example results of the testing. The first column shows the original image, the middle column image with the highest compression factor, and the rightmost column shows the image with the lowest compression factor. The high compression factor is compensated for by losing variety of detail amongst structures (e.g., reflections in windows.)

One obvious limitation of our method is the need for manual editing. Though it would have been possible to directly render images compressed by some existing automatic lossless method, users nonetheless achieved a good lossy compression factor while preserving the quality of the output, since they were allowed to apply their own individual judgment as to what could be discarded. Another potential limitation of our study was the selection of our test group. It would have been interesting to see how a more heterogeneous group of users might have performed.

Many possible avenues for future work exist. Recently, a number of automatic methods for facade encoding have appeared. One is an approach [28] that allows an accounting for varying facade visual properties which are then encoded as a parameter. It would be interesting to implement their approach using direct GPU image rendering. Lastly, a potentially promising caching strategy derived from this work could be implemented in which similar pixels are identified and then processed by grammatical rules similar to those created using our approach.

References

1. Bao, F., Schwarz, M., Wonka, P.: Procedural facade variations from a single layout. ACM Trans. Graph. **32** (2013) 8:1–8:13
2. Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van Gool, L.: Procedural modeling of buildings. ACM Trans. Graph. **25** (2006) 614–623
3. Zhang, H., Xu, K., Jiang, W., Lin, J., Cohen-Or, D., Chen, B.: Layered analysis of irregular facades via symmetry maximization. ACM Trans. Graph. **32** (2013) 121:1–121:13

4. Stava, O., Benes, B., Mech, R., Aliaga, D.G., Kristof, P.: Inverse Procedural Modeling by Automatic Generation of L-systems. Comput. Graph. Forum **29** (2010) 665–674
5. Haegler, S., Wonka, P., Arisona, S.M., Gool, L.V., Müller, P.: Grammar-based encoding of facades. In: Proceedings of the EGSR, Eurographics Association (2010) 1479–1487
6. Fuzhang Wu, Dong-Ming Yan, W.D.X.Z.P.W.: Inverse procedural modeling of facade layouts. Technical report, arXiv:1308.0419 [cs.GR] (2013)
7. Lipp, M., Wonka, P., Wimmer, M.: Interactive visual editing of grammars for procedural architecture. ACM Trans. Graph. **27** (2008) 102:1–102:10
8. Parish, Y.I.H., Müller, P.: Procedural modeling of cities. In: Proceedings SIGGRAPH, ACM Press (2001) 301–308
9. Wonka, P., Wimmer, M., Sillion, F., Ribarsky, W.: Instant architecture. ACM Trans. Graph. **22** (2003) 669–677
10. Smelik, R.M., Tutenel, T., Bidarra, R., Benes, B.: A survey on procedural modelling for virtual worlds. Comp. Graph. Forum **33** (2014) 31–50
11. Aliaga, D.G., Rosen, P.A., Bekins, D.R.: Style grammars for interactive visualization of architecture. IEEE TVCG **13** (2007) 786–797
12. Hohmann, B., Krispel, U., Havemann, S., Fellner, D.: Cityfit - high-quality urban reconstruction by fitting shape grammars to image and derived textured point clouds. In: Proceedings of the International Workshop 3D-ARCH 2009. (2009)
13. Vanegas, C.A., Garcia-Dorado, I., Aliaga, D.G., Benes, B., Waddell, P.: Inverse design of urban procedural models. ACM Trans. Graph. **31** (2012) 168:1–168:11
14. Stava, O., Pirk, S., Kratt, J., Chen, B., Mch, R., Deussen, O., Benes, B.: Inverse procedural modelling of trees. Computer Graphics Forum **33** (2014) 118–131
15. Müller, P., Zeng, G., Wonka, P., Van Gool, L.: Image-based procedural modeling of facades. ACM Trans. Graph. **26** (2007)
16. Musialski, P., Wonka, P., Recheis, M., Maierhofer, S., Purgathofer, W.: Symmetry-based facade repair. In: Vision, Modeling, and Visualization Workshop 2009. (2009)
17. Teboul, O., Simon, L., Koutsourakis, P., Paragios, N.: Segmentation of building facades using procedural shape priors. In: Proceedings of CVPR. (2010) 3105–3112
18. Demir, I., Aliaga, D.G., Benes, B.: Coupled segmentation and similarity detection for architectural models. ACM Trans. Graph. **34** (2015) 104:1–104:11
19. Boulch, A., Houllier, S., Marlet, R., Tournaire, O.: Semantizing complex 3d scenes using constrained attribute grammars. Computer Graphics Forum **32** (2013) 33–42
20. Martinovic, A., Van Gool, L.: Bayesian grammar learning for inverse procedural modeling. In: Proceedings of CVPR. (2013) 201–208
21. Shen, C.H., Huang, S.S., Fu, H., Hu, S.M.: Adaptive partitioning of urban facades. ACM Trans. Graph. **30** (2011) 184:1–184:10
22. Vanegas, C.A., Aliaga, D.G., Beneš, B.: Building reconstruction using manhattan-world grammars. Proceedings of CVPR (2010) 358–365
23. Ceylan, D., Mitra, N.J., Li, H., Weise, T., Pauly, M.: Factored facade acquisition using symmetric line arrangements. Comp. Graph. Forum **31** (2012) 671–680
24. Wan, G., Sharf, A.: Applications of geometry processing: Grammar-based 3d facade segmentation and reconstruction. Comput. Graph. **36** (2012) 216–223
25. Kerber, J., Bokeloh, M., Wand, M., Seidel, H.P.: Scalable symmetry detection for urban scenes. Comput. Graph. Forum **32** (2013) 3–15
26. Li, Y., Zheng, Q., Sharf, A., Cohen-Or, D., Chen, B., Mitra, N.J.: 2d-3d fusion for layer decomposition of urban facades. In: Proceedings of ICCV. (2011) 882–889
27. Musialski, P., Wimmer, M., Wonka, P.: Interactive coherence-based facade modeling. Comp. Graph. Forum **31** (2012) 661–670
28. AlHalawani, S., Yang, Y.L., Liu, H., Mitra, N.J.: Interactive facades analysis and synthesis of semi-regular facades. Computer Graphics Forum **32** (2013) 215–224