Technical Section

Guided proceduralization: Optimizing geometry processing and grammar extraction for architectural models☆

İlke Demir*, Daniel G. Aliaga

Purdue University, West Lafayette, IN, US

ABSTRACT

We describe a guided proceduralization framework that optimizes geometry processing on architectural input models to extract target grammars. We aim to provide efficient artistic workflows by creating procedural representations from existing 3D models, where the procedural expressiveness is controlled by the user. Architectural reconstruction and modeling tasks have been handled as either time consuming manual processes or procedural generation with difficult control and artistic influence. We bridge the gap between creation and generation by converting existing manually modeled architecture to procedurally editable parametrized models, and carrying the guidance to procedural domain by letting the user define the target procedural representation. Additionally, we propose various applications of such procedural representations, including guided completion of point cloud models, controllable 3D city modeling, and other benefits of procedural modeling.

1. Introduction

The recent popularity of 3D environments and models for augmented and virtual reality environments puts high expectations on the complexity and quality of such assets, with a desire to replicate the real world. Urban planning, remote sensing, and 3D reconstruction researchers have been focusing on bringing the digitized and physical world together, with an emphasis on urban models. In parallel to the demand for city-scale 3D urban models, the availability of 3D data acquisition systems and image-based solutions have also increased. Although using the 3D data obtained from different sources such as images, laser scans, time-of-flight cameras, and manual modeling databases is an option, the results of these approaches usually do not expose an easily modifiable model with structural parts and thus obstructs architectural reconstruction and modeling tasks.

Aiming for automatic generation, procedural representations are highly parameterized, compact, and powerful, especially in urban domain [1,2]. The pioneering work of Parish and Mueller [3], and subsequent urban modeling papers (e.g., see surveys [4,5]) focused on forward and inverse procedural modeling approaches. While procedural modeling (PM) accomplishes providing architec-

tural structures of required detail, creating complex realistic building templates needs time, extensive coding, and significant domain expertise. Inverse procedural modeling (IPM) addresses the shortcomings of procedural modeling by controlling and adapting the procedural generation to a given target model [6,7]. In this sense, IPM can be regarded as an optimization problem over the space of derivations, to guide procedural modeling.

We want to carry this problem one step further by (1) removing the dependency on the space of derivations, and (2) switching the control domain. *Proceduralization* [8] takes care of the first motivation, by converting existing geometric models into a procedural representation, with no a priori knowledge about the underlying grammar or components.

However, as this procedural representation aims to serve as the minimal description of the model, evaluating for the best description (e.g., the description with the best expressiveness) requires determining its Kolmogorov complexity, which is uncomputable. Our solution is to let the user guide the system to find the best grammar per use case. This also handles the second motivation by enabling the user to control the characteristics of the extracted grammar. In other words, inverse procedural modeling enhances procedural modeling by producing the best instance, while guided proceduralization enhances proceduralization by producing the best grammar.

In this paper, we focus on guided proceduralization for 3D urban modeling and reconstruction of architectural meshes, building point clouds, and textured urban areas. Our framework provides
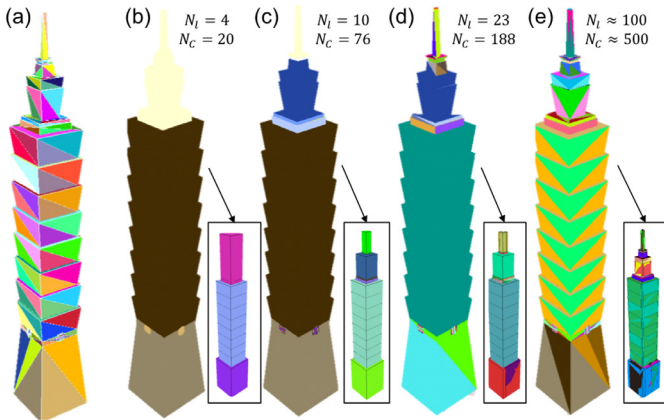
Fig. 1. *Guided proceduralization.* (a) The input model of *Taipei 101*, (b–e) colored procedural elements ordered by increasing target parameter values (e.g., number of components N_c and number of similarity groups N_l). As the user specification on the target grammar changes, different grammars of the model are revealed. Insets indicate representative instances of rules and terminals.

a feedback loop which under user control seeks out the hidden high-level hierarchical and structural information coherent with the target specification and the application objectives (Fig. 1). Although guided procedural modeling approaches [9,10], and proceduralization methods [11,12] have been introduced, we propose the first approach to guide the proceduralization process using specifications of a target grammar. We start with definitions and functions for guided proceduralization, then introduce geometry processing and grammar extraction steps of generalized proceduralization in the controlled setting. Afterwards, we demonstrate applications of the obtained grammars in completion, reconstruction, synthesis, modeling, querying, simplification, and rendering.

Altogether, our main contributions include:

- a generalized guided proceduralization framework that extracts procedural representations across different 3D data types,
- an optimization process to evaluate and diversify the grammars output by our proceduralization,
- a feedback loop to enable guidance to control proceduralization for obtaining the most expressive grammar and for various aims, and
- several applications of the guided proceduralization framework for editing and merging various models.

Using our controlled proceduralization framework, we have extracted procedural models from complex polygonal models (i.e., Turning Torso, Taipei 101, Saint Basil's Cathedral), from point clouds of architectural scenes (i.e., Staatsoper Hannover, Wilhelm Busch Museum), and from textured massive city models (i.e., 180 km^2 metropolitan area of New York with more than 4000 buildings, San Francisco, Chicago). We have used these models to create more complete reconstructions, controlled procedural generation, and easier, efficient, and structure-aware urban modeling.

2. Related work

2.1. Procedural modeling

Procedural modeling P generates a model M from a given grammar G ($P(G) = M$). Starting with the pattern language of Alexander et al. [13], PM has been utilized by many approaches [14–17] in urban settings. More PM approaches are surveyed by Smelik et al. [4]. However, coding procedural details is a cumbersome process needing domain expertise and codification skills. Interactive editing systems, such as Lipp et al. [18] and CityEngine, have been added on top of existing procedural systems to facilitate grammar editing. These can be considered as the first guidance solutions for procedural editing systems.

2.2. Inverse procedural modeling

In contrast, inverse procedural modeling discovers the set of parameters, probabilities, and rules from a given grammar to generate a target instance [5] ($P(G, T_M) = M_{opt}$ as per the second row of Fig. 2). Initial works provided semi-automatic and automatic building (e.g., [19–21]) and facade solutions (e.g., [22–26]). Given some exemplar derivations and labeled designs, Talton et al. [27] use Bayesian induction to capture probabilistic grammars. Similarly, Monte Carlo Markov Chain (MCMC) optimization is used to discover the optimized parameters for a target instance of a procedural representation of buildings (Talton et al. [28] and Nishida et al. [29]) and cities (Vanegas et al. [30]). Most of those solutions support synthesizing similar models that best fit the given guidance. However they rely on pre-segmented components, known grammars, and known layouts to generate the derivation space. This is an important drawback, since it constrains reconstruction and modeling to the limited space of the initial grammar. In contrast, we want to carry the guidance from the geometric space to the procedural space, thus the *desired control* is defined rather than the *desired model*.
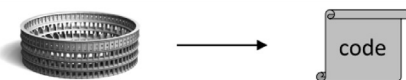
	Process	**Function**	**Approaches**
Procedural Modeling		$P(G) = M$	Wonka03, Muller06, Lipp08, Vanegas09, Schwarz15, …
Inverse Procedural Modeling		$P(G, T_M) = M_{opt}$	Aliaga07, Zhou12, Talton11, Vanegas12, Nishida16, …
Proceduralization		$P^{-1}(M) = G$	Bokeloh10, Toshev10, Demir15, Demir16, …
Guided Proceduralization		$P^{-1}(M, T_G) = G_{opt}$	This paper

Fig. 2. *Evolution of procedural modeling.* Differences between controlled and inverse procedural modeling approaches with references. P represents the process of procedural modeling with an input grammar G and output geometric model M. T_M indicates a geometric target instance (exemplar, silhouette, etc.), and T_G indicates a procedural target instance (grammar elements, pattern examples, etc.). $*_{opt}$ indicates that it is optimized for "the most representative of the target".
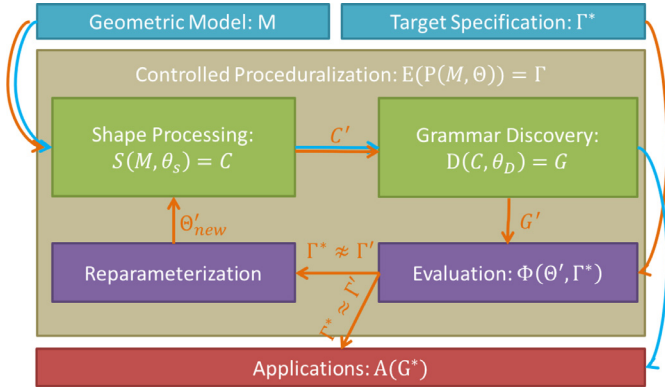
Fig. 3. *Pipeline.* Our system optimizes parameter values Θ of the proceduralization function P to generate a grammar G that fulfills user-provided grammar values Γ^*. Blue path shows traditional proceduralization pipeline and orange path shows guided proceduralization pipeline. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.3. Proceduralization

Proceduralization starts with only geometry and no knowledge on the grammar ($P^{-1}(M) = G$ as per the third row of Fig. 2). Some image based techniques use deep learning for extracting simple and-or template grammars [31], or generative schemes [32]. In urban scenes, Bokeloh et al. [33] use partially symmetric structures to search for transformations that map one partition to another based on *r*-similar surfaces. It enables building model synthesis though not formally yielding a procedural model. For point clouds, Toshev et al. [21] segment a building into planar parts and join them using a hierarchical representation that separates roof, rest of the building, and non-building structures. Demir et al. [11] focus on point clouds, and user control is explicit in the geometric domain at the semi-automatic segmentation step. Martinovic et al. [6] use Bayesian induction to obtain facade grammars, and Kalojanov et al. [34] divide the input structure into microtiles to detect partial similarities. Demir et al. [12] introduces proceduralization, automatically creating a set of terminals, non-terminals, and rules (blue path in Fig. 3). Although evaluating the expressiveness of this automatic encoding is uncomputable (see Section 6.1), the expressiveness of a fixed grammar per model is still limited with regard to the modeler's use case. Thus, their approach has smaller tolerance for noise, directed by the one-pass segmentation and labeling, is not flexible for different use-cases, works only on triangular models, and does not allow user control (for the grammar generation).

2.4. Guided proceduralization

In contrast, the key motivation behind our research is that, if we have some insights about the desired grammar, we can evaluate the proceduralization outputs to suggest candidate grammars. This creates a pioneering framework that is the first to provide guided proceduralization for synthesis of arbitrary 3D architectural structures ($P^{-1}(M, T_G) = G_{opt}$ as per the last row of Fig. 2). As guidance in procedural modeling enables finding *the best instance*, guidance in proceduralization enables finding *the best grammar*. This guidance enables the procedural representation to be optimized by user specification (orange path in Fig. 3), so that the resulting grammars are robust to model noise, flexible for different use cases, independent of segmentation and labeling, and supports more creative processes. In summary, we would like the artists to define the desired control, instead of the desired structure, while

Fig. 4. *Evaluation of proceduralization.* The input model is proceduralized using some input parameter set Θ. The resulting grammar is evaluated to obtain grammar specification values Γ.

still being able to derive procedural representations from existing models based on their use cases.

3. Definitions and functions for guidance

Our control loop optimizes the internal parameters of the proceduralization process to generate a procedural description (e.g., a grammar) that meets a user-provided set of grammar specification values. More precisely, proceduralization function P operates on a set $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ of m input parameter values and an input model M to produce a grammar G. The grammar is then assessed by an evaluation function E which produces a set $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ of n grammar specification values (or simply grammar values). Each parameter value and grammar value should be within a range of $[\theta_{m_{min}}, \theta_{m_{max}}]$ and $[\gamma_{n_{min}}, \gamma_{n_{max}}]$, respectively. Symbolically, we can define the relationship of input parameters to the target grammar values as follows in Eq. (1), which is also depicted in Fig. 4.

$$\Gamma = E(G) = E(P(\Theta, M)) = (E \circ P)(\Theta, M) \qquad (1)$$

Intuitively, guided proceduralization frees grammar discovery from any dependency to the underlying segmentation and grammar discovery. There is no concept of "good segmentation", "different tessellations", "noisy point clouds", "vertex displacements", because the whole idea of this paper is to let the optimization find the best internal parameters towards the target grammar specification. Previous proceduralization approaches failed to achieve that, because evaluating grammars for expressiveness is uncomputable. However, because we are provided with an estimate of "the best grammar" by the user, we keep optimizing the whole proceduralization process (including segmentation, labeling, pattern analysis, etc.) to converge for the specification.

3.1. Target grammar specification

Our optimization objective is to let the user specify the desired grammar values and then to automatically generate a grammar whose actual grammar values are close to the specified ones. Hence, we define the set $\Gamma^* = \{\gamma_1^*, \gamma_2^*, \ldots, \gamma_n^*\}$ to be the user-specified target grammar values. Then, our method computes a set of parameter values Θ' that leads the proceduralization function P to produce a grammar G' exhibiting grammar values closest to the target values Γ^*. In other words, we seek a set of parameter values Θ' such that $(E \circ P)(\Theta', M) \to \Gamma^*$. The equality is replaced by an arrow to indicate that we want the convergence, as absolute equality may not be satisfied for every case. The overall error is encapsulated in the function $\Phi(\Theta', \Gamma^*)$ and the computational goal is to find values of Θ' that minimizes Φ. Fig. 5 gives three examples for a toy facade model, where the user defines different grammar values to expose different grammars.

3.2. Error function approximation

Our control loop requires calculating $(E \circ P)(\Theta, M)$ for a large number of different values for Θ. Prior inverse procedural modeling and early proceduralization systems use predefined numbers for Θ and typically are not set up to generate such a large number of productions at reasonable rates. Hence, we simplify the $(E \circ P)$ by

a function $f(\Theta)$ that does not output the grammar but generates the set $\hat{\Gamma} = \{\hat{\gamma}_1, \ldots, \hat{\gamma}_n\}$ that approximate the grammar values. The function f is a good approximation of $(E \circ P)$ if $|\Gamma - \hat{\Gamma}| < \epsilon$ where ϵ is a small positive constant, meaning that each grammar value $\hat{\Gamma}_n$ generated by the approximator should be close enough to the corresponding actual grammar value Γ_n generated by forward proceduralization. Thus, given a good approximation f of the proceduralization system and grammar evaluation $(E \circ P)$, our optimization goal is to find sets of parameter values Θ' such that $f(\Theta') \rightarrow \Gamma^*$.

4. Guided proceduralization framework

In the proceduralization framework $G = P(M)$ (Fig. 3, blue path), the aim is to obtain the underlying grammar G that is representative of the input model M. The input to this pipeline can be images, meshes, point clouds, and almost any content that can be categorized. The granularity of the content for urban proceduralization can vary from facades, to buildings, and to entire cities. Also, the input can be some specific elements of an urban area such as parcels, roads, buildings, or a full city. The proceduralization pipeline is divided into two main steps: geometry processing and grammar discovery.

For guided proceduralization (Fig. 3, orange loop), the proceduralization function is parametrized such that different grammars can be extracted based on different parameters (i.e., $G = P(\Theta, M)$). Those grammars do not only differ by the composition of the rules and rule parameters, but the terminals and non-terminals also vary between different grammars, as the decomposition changes. The parametrization is carried over to the two main stages of proceduralization. Shape processing $S(\Theta_S, M) = C$ in turn is subdivided into segmentation and labeling, while grammar discovery $D(\Theta_D, C) = G$ is further separated into hierarchy construction and pattern extraction. We can re-write the proceduralization framework in detail as:

$$G = P(\Theta, M) = D(\Theta_D, C)$$
$$= D(\Theta_D, S(\Theta_S, M))$$
$$= (D \circ S)(\Theta, M) \tag{2}$$

4.1. Geometry processing

Our geometry processing step $S(\Theta_S, M)$ detects the repetitive nature in the underlying buildings, and uses that to compute a segmentation and perform a similarity labeling. The input model M will be segmented into components c_i, which are the smallest
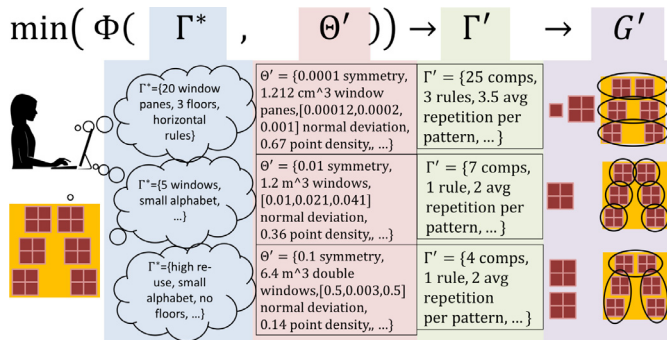

Fig. 5. *Optimizing for target grammars.* Minimization of the error function Φ, finds the optimum set of input parameters Θ' that generates Γ' which is the closest grammar specification to the user defined set Γ^*. As depicted, each Γ^* ends up generating a different grammar G', unlike traditional proceduralization. The first row has window panes as the terminals, windows as rules. The second row has a window as a terminal, and the last row has a double window with varying orientation, as a terminal. Example rules per grammar are also given per input Γ^*.
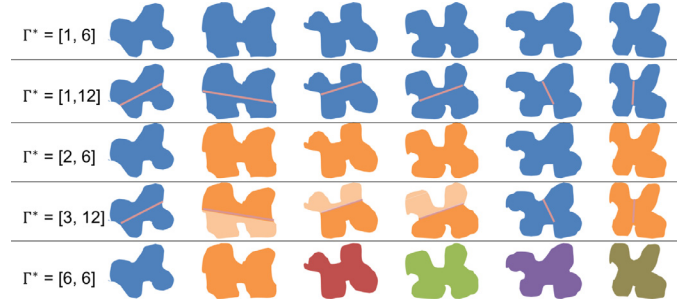

Fig. 6. *Shape processing.* The effect of reparameterization of Θ' based on user-specified grammar values Γ^* is illustrated on the segmentation on a set of components with six similar but irregular shapes. The colors indicate the similarity groups l_j. As an example, the target grammar values for row 4 are "3 component types and 12 components", ending up segmenting the set to 5 orange, 4 blue, and 3 light orange components. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

geometric units of decomposition, also candidates to become terminals of the grammar. Each component will have a corresponding label l_j, where the label is the indicator of the similarity group that the component belongs to. Thus, the full set of components is $C = \{(c_1, l_1), \ldots, (c_{N_c}, l_{N_l})\}$ (where $N_l \leq N_c$). Using the control loop, different Cs are produced depending on the Γ^* values (notice the different set of components and patterns found for the middle section of the tower in the top right corner of Fig. 11).

4.1.1. Parametrized shape segmentation
Our formulation for S depends on the type of input data $M = \{t_1, \ldots, t_{N_m}\}$. Prior segmentation/labeling work focuses on single data types and exploiting geometric similarities [35], analogies with the exact-cover problem [36], image contours [20], pre-defined semantics [21,23], or geometric primitives [1,37]. The aforementioned approaches do not organize the repetitions into patterns, use the patterns for modeling, or provide a way to control pattern granularity selection.

In contrast, our shape processing method extends prior work by adapting it to our guided proceduralization framework, by introducing weights as parameters to balance collective component properties such as number of components and number of similarity groups. As shown in Fig. 6, various (simplified) grammar values end up generating several versions of segmentation and labeling of the same set. Here, Γ^* only consists of the alphabet size and the component size, and components are color-coded by their labels.

4.1.2. Segmentation formulation
Our approach combines segmentation and labeling in a coupled algorithm, where the model is regarded as the combination of all elements of all labels (Eq. (3)), and shape processing outputs the components and their labels (Eq. (4)).

$$M = \Sigma_i^{|M|} t_i = \Sigma_j^{N_l} \Sigma_k^{N_c} (t_i, l_j) \mid t_i \in c_k \tag{3}$$

$$S(M, \Theta_S) = \{(\Sigma_k^{|c_1|} t_k, l_1), \ldots, (\Sigma_k^{|c_{N_c}|} t_k, l_{N_l})\} \tag{4}$$

Our parameter value vector contains similarity-metrics for the components, namely $\Theta_S = \{\theta_{geo}, \theta_{top}, \theta_{den}, \theta_{dir}, \theta_{num}\}$ for geometric similarity, topological similarity, density, orientation, and number of elements, respectively.

4.1.3. Input dependent implementation
Although the metric is unified into one representation, the segmentation method and the elements in the parameter value vector varies based on the data type of t_i. Our framework collectively optimizes over these methods, where traditionally hand-tuned parameters are replaced by optimization space parameters
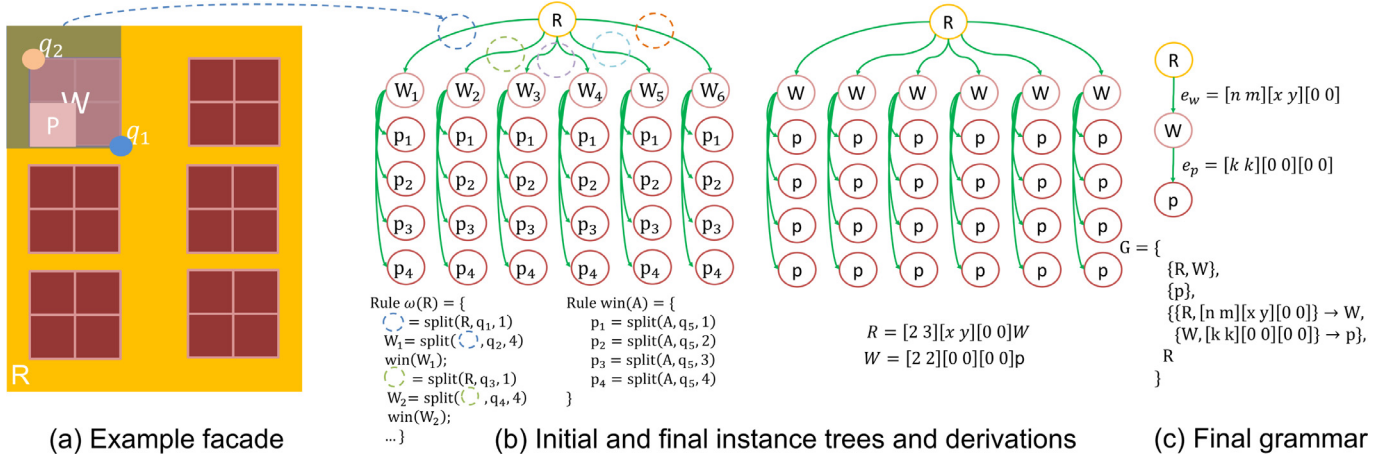
Fig. 7. *Split tree and grammar.* (a) Components of the example facade: p as a windowpane and W as a window. q_1 is the split point to create the invisible blue node, and q_2 is the second split to obtain w from the blue node. (b) initial and final split tree and corresponding rules: p recognized as a terminal and W recognized as a non-terminal, meanwhile rule parameters are discovered, and (c) the final grammar with a facade rule and a window rule. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

specific to the input type. In all cases, we will use our overall optimization framework to compute the parameter values Θ'_S that best yields a grammar with the desired values Γ^*, expressed as $(E \circ (D \circ S))(\Theta'_S, \Theta'_D, M) \to \Gamma^*$.

4.1.3.1. Triangular models

For example, for t_i as triangles, the similarity parameter θ_{geo} is defined as a weighted sum of similarity of convex-hulls and averaged similarity of triangles between two components. θ_{num} becomes number of initial component seeds for triangles. θ_{dir} becomes the similarity of orientations by a deviation of normal vectors. The segmentation methodology also changes based on the data type. For triangular meshes, we use synchronous volume growing [36] to establish a segmentation and similarity detection simultaneously, where the algorithm parameters are explored by the optimization implicitly.

4.1.3.2. Point clouds

For t_i as points, geometric similarity metric θ_{geo} is defined as the mean of correspondence distances normalized by point density, after an iterative closest point (ICP) [38] alignment between components, because correspondence distances without aligning the two components result in incorrect measurements for θ_{geo}. For the segmentation of point clouds, we use a loop of plane fitting and Euclidian clustering [39], where θ_{num} represents the iteration count for the expected number of segments. θ_{den} defines the minimum density of components, which is also instrumental in cleaning the point clouds during segmentation phase.

4.1.3.3. Textured meshes

Lastly, for t_i as textured polygons, θ_{geo} is defined as a similarity vector of components with multiple geometric and visual features (dimensions, albedo, color patterns, etc.). The segmentation incorporates rendering components in different main directions and measuring the rate of change for finding the cut planes at significant jumps. The components between the cut planes are re-rendered for extracting and clustering their feature vectors [40]. In this process, θ_{num} controls the camera displacement step size.

4.2. Grammar discovery

The second step of the pipeline organizes the segments c_i and their labels l_j into a hierarchy, which we will call an instance tree $T = \{(v_0, \varnothing), (v_1, e_1), \ldots, (v_{N_v}, e_{N_v})\}$, where v_i denotes a tree node (with v_0 being the root), and e_i denotes the incoming edge to node v_i. This hierarchy is needed to encode the relations of c_i and to discover the underlying rules. It also encodes an actual derivation

instance thus can be treated as an early procedural representation of the model. After the hierarchical structure is constructed, it is analyzed for patterns and for the distribution of the components to finally extract the grammar G.

4.2.1. Grammar definition

While G can be a shape grammar or an L-system, we define a split grammar of the following form:

$$G = \{N, \Sigma, R, \omega\}, \; where \tag{5}$$
$$N = \{v_i \mid (v_i, \_) \in T \; \wedge \; fanout(v_i) \neq 0 \wedge (v_i, \_) \in C\}$$
$$\Sigma = \{v_i \mid (v_i, \_) \in T \; \wedge \; fanout(v_i) = 0 \wedge (v_i, \_) \in C\}$$
$$R = \{(v_i, e_j, v_j) \mid (v_j, e_j) \in T \wedge v_i \in anc(v_j) \wedge \{(v_i, \_), (v_j, \_)\} \subset C\}$$
$$\omega = \{v_0 \mid (v_0, \varnothing) \in T \wedge (v_0, \_) \in C\}$$

where N is the set of non-terminals (root nodes of subtrees in T each representing a rule), Σ is the set of terminals (leaves), R is the collection of rules containing triplets of the application node v_i, the actual rule on the edges e_j and the produced node v_j (e.g., all split operations encoded in the edges e_j from a subtree root node v_i), ω is the starting axiom (root node), $fanout(v_i)$ is the number of edges from node v_i to its children, $v_x \in anc(v_i)$ means node v_x is an ancestor of node v_i, and $\_$ denotes a "don't care" edge, node, or label.

Fig. 7 depicts the grammar elements and some relations between the tree, the instance, and the grammar. The example facade in (a) contains six windows and four windowpanes per window. During the initial tree construction, the windows are produced from the wall (root) by two split operations. For the first window, R is split from q_1 to obtain the blue intermediate node, and the blue node is split from q_2 to obtain the window node W_1. Having no offset from the window, each window pane is produced from the window with one split operation. Leaf nodes become candidates for Σ, and subtree roots become candidates for N, because they mostly contain geometry. Intermediate nodes are embedded as individual split operations as a part of a rule. After we process the instance tree, we converge on the pattern structure to be discovered as the second subtree in (b), with window pattern parameters of "2 horizontal repetition with x spacing and 3 vertical repetition with y spacing" and windowpane parameters of "2 horizontal and 2 vertical repetitions with 0 spacing". Then those pattern parameters are generalized and exported as the grammar following the syntax introduced above.

We define the grammar specification values as $\Gamma = \{\gamma_{alp}, \gamma_{non}, \gamma_{fan}, \gamma_{rep}\}$ which are alphabet size $|\Sigma|$, number of non-terminals $|N|$, average repetitions per pattern $|\bar{R}_i|$, and average number of components $|C|$, respectively.

4.2.2. Tree Definition

The components in the set C of the previous section are represented by their bounding boxes $BB(c_i)$ and all elements $t_k \in c_i$ (e.g., triangles, points, textured polygons, etc.) are placed relative to the coordinate space defined by $BB(c_i)$. We start with putting all c_i into an instance tree T based on volumetric containment (Eq. (6)), based on point splits. The components are processed by decreasing size, and the tightest containing node is selected as the parent at each step. Thus, a component can only be completely inside one component, preventing T from becoming a DAG. Each c_i corresponds to a node v_i and each edge e_i corresponds to the spatial transformation matrix (translation and rotation) of node v_i relative to its parent $\alpha(v_i)$ (Eq. (7)): in this notation e_i is a matrix and $BB(v_i)$ is represented by two vectors containing the minimum and maximum points of the bounding box, thus the multiplication operator gives the transformed bounding box for the new node. Labels are also preserved (Eq. (8)) at the initiation of T. $L(v_i)$ operation retrieves label of the component c_i corresponding to v_i.

$$v_j = \alpha(v_i) \iff BB(v_i) \subset BB(v_j) \tag{6}$$

$$(v_i, e_i) \in T \Rightarrow BB(v_i) = e_i * BB(\alpha(v_i)) \tag{7}$$

$$L(v_i) = l_j \iff (v_i, l_j) \in C \tag{8}$$

4.2.3. Subtree similarity

Afterwards we process T to detect similar subtrees. Starting with initial labels, we compare all v_i, v_j pairs where $L(v_i) = L(v_j)$. The comparison is based on pairwise topological (θ_{sub} = subtree matching by BFS ordering) and geometrical (θ_{sym} = label matching) similarity, with the additional comparisons of number of elements (θ_{ele}) and size of bounding boxes (θ_{box}). According to these metrics, if v_i and v_j are not similar, we update their labels to reflect that $L(v_i) \neq L(v_j)$. We define the concept of representative instance v_x^*, which indicates the common representation for a repeating subtree, for all similar nodes ($v_x^* = \{v_i \mid \forall i, \ L(v_i) = x\}$). Note that representative instances are not decided based on initial labels, so Eq. (8) is only used during hierarchy construction phase. We also perform a special *join* operation to canonicalize the subtree. The derivation of nodes $v_i \in v_x^* \wedge v_j \in v_x^* \wedge \exists k| \ v_k \in anc(v_i) \wedge v_k \in anc(v_j)$ are unified to follow the same subtree structure, so that they have the same ancestor v_k as the application subtree root. In this case, v_k can be the parent or grandparent of joint nodes in v_x^*, but we will refer to it as $\alpha(v_x^*)$ for a simpler notation.

4.2.4. Pattern parameters

After all v_x^* are discovered, we reveal the distribution of the representative instances and the spatial parameters of the distributions – namely the patterns. R corresponds to an encoding of the patterns, however instead of representing each derivation with one rule on one selected edge (i.e., $R_x = (\alpha(v_x^*), e_i, v_i) \mid \exists \ i, \ v_i \in v_x^* \wedge L(v_i) = x$), the derivation of representative instances are combined under an averaged rule (i.e., $R_x = \{\alpha(v_x^*), \bar{e}_i, \bar{v}_i) \mid \forall v_i \in v_x^*, \ \bar{v}_i = mean(v_i) \wedge \bar{e}_i = mean(e_i)\}$. In other words, similar derivations in the parse tree are averaged or combined to obtain a representative rule that creates the example derivation. As mentioned in the previous section, this merging process is controlled by the feedback loop, thus we introduce two more parameter values as θ_{pat} for the pattern granularity and θ_{ins} for instance similarity. The first one
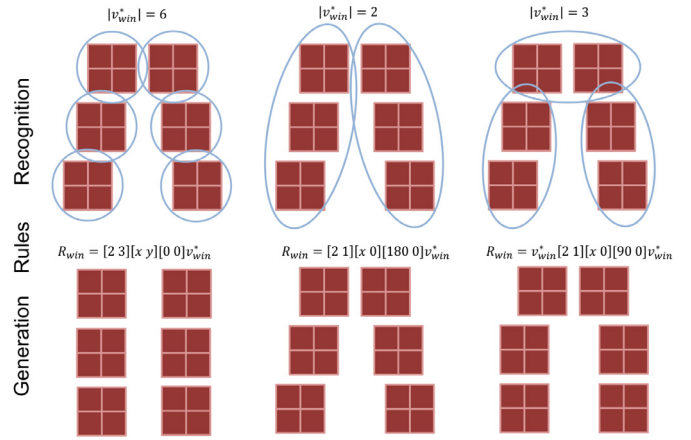

Fig. 8. *Grammar discovery.* Granularity of recognized rules R changes based on specified grammar values Γ^*. Extracted rules generate different derivations. The rule format is $R = [repetition][spacing][rotation][representative instance]$.

controls pattern granularity by adjusting the repetition count per instance and the second one controls similarity of transformation matrices to be averaged. Together, they allow the same geometrical pattern to be recognized into different rules (Fig. 8).

4.2.5. Grammar discovery formulation

The full parameter value vector contains similarity metric parameters for the tree nodes and for the patterns, namely $\Theta_D = \{\theta_{sub}, \theta_{sym}, \theta_{ele}, \theta_{box}, \theta_{pat}, \theta_{inst}\}$. The control loop executes the evaluation function E to determine the approximated values Γ' and the error function Φ decides whether $\Gamma' \approx \Gamma^*$. Thus, the loop explores for the best representative instance in the optimization stage, again solving for the closest parameter values Θ_D' for the desired grammar values Γ^*, expressed as $(E \circ (D \circ S))(\Theta_S', \Theta_D') \rightarrow \Gamma^*$. Also, instead of running the whole framework for each optimization step, we simplified $(E \circ P)$ with f, which performs a reduced set of segmentation and hierarchy building operations (i.e., instead of creating a tree, numerically deciding the relationships), still yielding the same grammar values without processing the entire geometry.

4.2.6. Grammar output

Finally, G is exported using a common split grammar syntax where leaf representative instances are converted to terminals Σ, and non-leaf representative instances are converted to non-terminals N. The root of the instance tree is output as the starting axiom ω, followed by other grammar elements concluding the grammar export. To save time, grammar exportation is delayed until the control loop finds the most descriptive grammar based on user specifications.

5. Results and applications

Our framework is implemented in C++ using Qt, OpenGL, OpenCV, PCL, and dlib on an Intel i7 Desktop PC with a NVIDIA GTX 680 graphics card. We used publicly available 3D databases (e.g., Trimble Sketchup, [41,42]) and some manually created models. We applied our guided proceduralization system to polygonal, point-based, and textured models of varying scales, from simple buildings (e.g., 2000 polygons) to large cities (e.g., 4000 buildings) and laser scans (e.g., 2M points), including over 70 individual buildings and 3 large cities. We demonstrate the applications of our controlled grammars in three areas: completion and reconstruction, architectural modeling, and procedural generation.
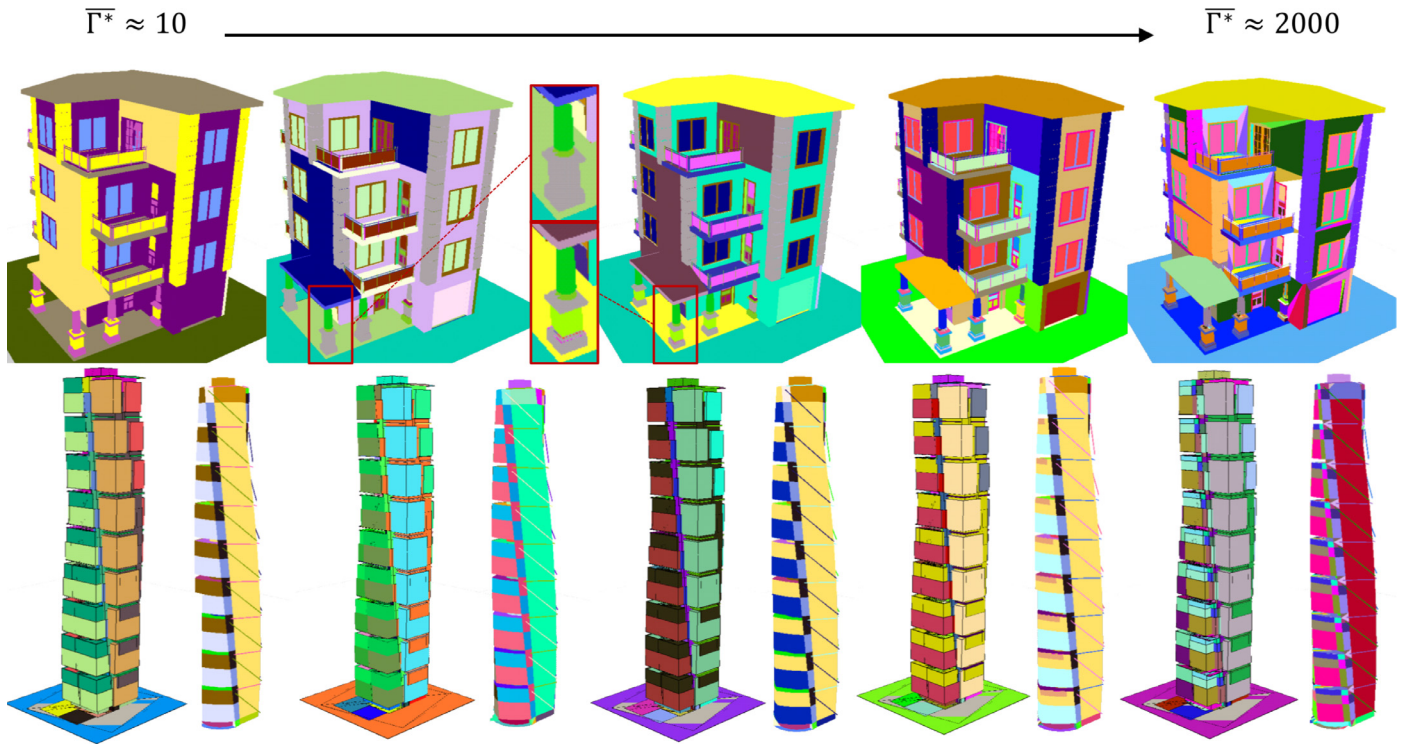
$\overline{\Gamma^*} \approx 10$ ⟶ $\overline{\Gamma^*} \approx 2000$

Fig. 9. *Varying grammar values* Γ^*. Grammar specification varies from small (left) to larger values (right) producing different components, labels, rules, and grammars. As can be observed in the inset of the top row, even the slight changes in specification can converge to a significantly different set of parameter values Θ'. Bottom row shows a similar analysis on *Turning Torso*, each instance (left) paired with its node distribution (right).

5.1. Analysis

We show an example analysis of varying grammar values in Fig. 9 on a complex model of 23K polygons. We sample and demonstrate a subset of the domain $[\Gamma^*_{min}, \Gamma^*_{max}]$ yielding visually recognizable patterns increasing from left to right. Notice that even slight changes in the grammar specification can yield new patterns, not only with different rules, but also with different geometric components and similarity labels. The change in the granularity of the rule elements to define the column rule is shown in the insets. The right-most example shows an extreme case where $\Gamma^* > \Gamma^*_{max}$ which is not desirable and over-segments some nonterminals near the inner walls, visualized as colored triangles. The bottom row also shows a similar sampling of models for Γ^* for the rotating building pattern of *Turning Torso* landmark with 11K polygons, each instance (right) is paired with its node distribution (left). We observed that the grammar can capture more details based on the user specification, as seen in the white nodes on the rightmost model.

5.2. Robustness

We also evaluated the robustness of our approach in case of noisy models. Previous approaches tend to capture perfect repetitions and then convert them to grammars, but they fail to do so when the repetitions slightly differ in geometry and/or in distribution. We observed that guided proceduralization is more robust against such imperfections. We compare our work against work of Demir et al. [36], to evaluate the robustness of two approaches on models with different noise levels generated by vertex displacements. Compared to their segmented and labeled model (Fig. 10a), $\rho = 0.1\%$ full model vertex displacement (meaning that each vertex arbitrarily moves $x \in (0, \rho D]$, where D is the model diagonal) breaks their labeling (Fig. 10b), and $\rho = 1\%$ vertex dis-

placement breaks the overall algorithm (Fig. 10c) (The images are taken from the paper with permission). We input the undisturbed model (Fig. 10d) to the forward proceduralization algorithm P and obtained the grammar values Γ. Then we used the output Γ as target grammar values Γ^* to control the proceduralization of the disturbed models ($\rho = 0.1\%$ in (e) and $\rho = 1\%$ in (f)). As shown, our approach was able to find a grammar with the same specification (within ϵ) even for the most noisy model. This also proves the superiority of guided proceduralization over traditional proceduralization in case of noisy input models. Note that the
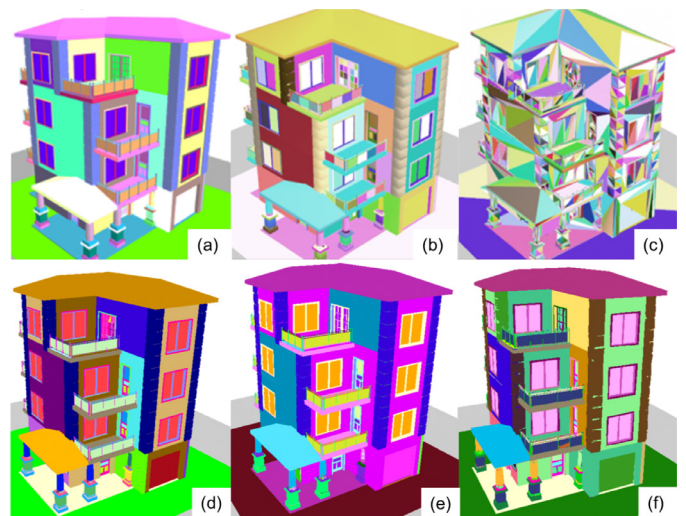

Fig. 10. *Robustness* comparison between [36] (a-c) and our approach (d-f) against full model vertex displacements (a, d have 0%, b,e have 0.1%, and c,f have 1%). (a,b,c is courtesy of [36]).

segmentation approaches in both systems are the same, thus the evaluation purely compares the overall frameworks.

5.3. Family of grammars

We observe that different grammars for different purposes can be obtained using guided proceduralization. In Fig. 11, different rules and different decompositions are colored as repeating patterns on the models. Moreover, we can guide the user towards different use cases by suggesting different grammars per model. This visual suggestion mechanism helps the user to approximate the grammar values Γ^* visually, instead of guessing numerically (as explained in Section 6.4). For example, the system suggests horizontal patterns on the left end of the tower body in Fig. 11 and vertical patterns on the right end. Similarly, the granularity of captured patterns decreases from left to right for the patio model. We also show that our approach is applicable to models with different structure (i.e., curved buildings), different scale (i.e., neighborhoods), and different complexity (tower with 2K polygons to the union building with 80K polygons).

5.4. Completion and architectural reconstruction

Although similarities and segmentations are exploited for some reconstruction techniques as in previous methods of Pauly et al. [43] and Simon et al. [44], the controlled discovery of grammars is not a focus of such papers. In contrast to prior reconstruction methods, we can improve the completion of point clouds before triangulation, without using any priors. We run our controlled proceduralization system on building point clouds, where the user roughly sets Γ^* to indicate the desired grammar values. The optimization finds the parameter values Θ'_S that best segments and labels the point cloud and Θ'_D that extracts the best grammar. Afterwards, we use all representative instances v_x^* to create consensus models (CM) [11] that are used to improve the quality per instance. The key innovation in using proceduralization for reconstruction emerges from i) exploiting repetitions for completion in a procedural setting, and ii) controlling proceduralization to get rid of the tedious task of semi-automatic segmentation for deciding the segments (as mentioned in Section 4.1.3). At a first glance, our reconstruction may seem blurrier, however this occurs from the fact that our approach combines all instances of the same terminal into a
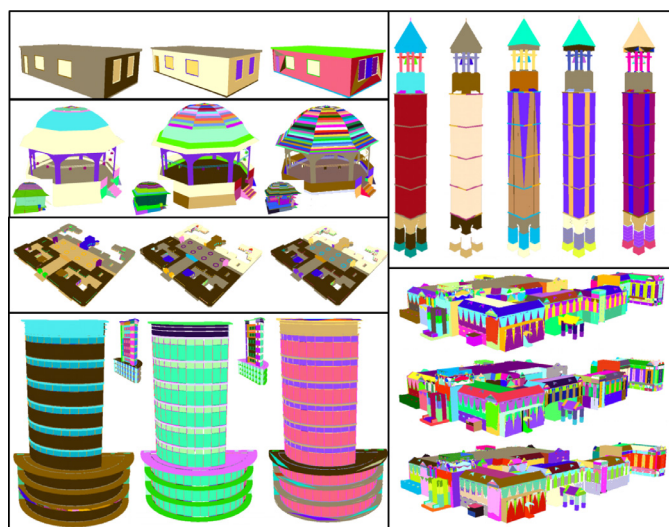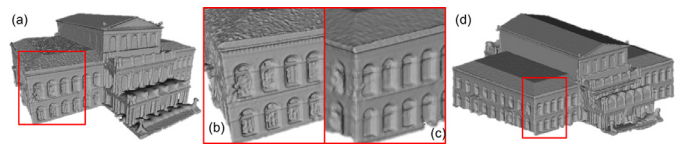

Fig. 12. *Poisson reconstructions* of (a,b) input and of (c,d) proceduralized point clouds. The identification of representative instances and exploitation of repetitions improve the reconstruction.
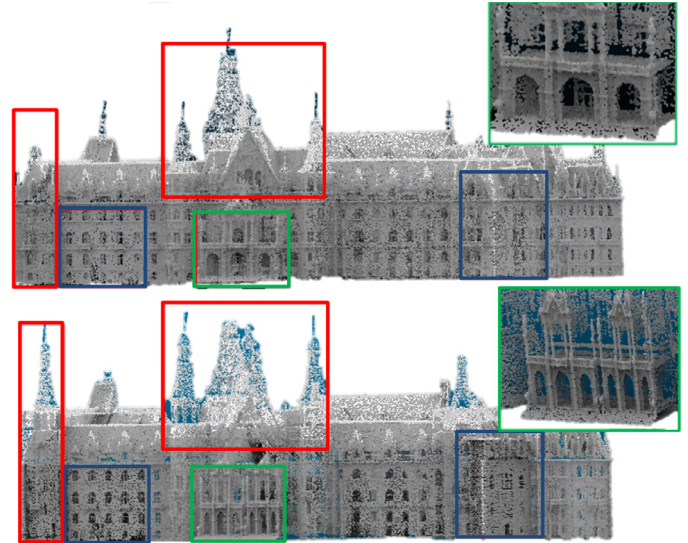
Fig. 13. *Completion.* Original model (top) and proceduralized and edited model (bottom) are compared for completion and synthesis. Blue boxes focus on improvements for window details, provided by consensus model based proceduralization. Red and green boxes focus on edited parts and their smooth blending, the green box zooming into the entrance of the building. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

more complete CM, and overall noise is reduced by edge-aware resampling. The smoothness of the components should not be perceived as blurriness.

Fig. 13 compares an input point cloud of 1.2M points vs. its proceduralized and edited version. Blue boxes emphasize the regions improved by only consensus models, prior to any editing sessions. Red boxes emphasize the parts after both automatic completion and procedural editing sessions, and how the blending is seamless. The green inset shows an extreme case where the modeler changes the structure is completely after the completion, but the underlying procedural engine is still able to preserve the style. Fig. 12 also compares a model with 2M points with its proceduralized version, after reconstruction of the point clouds. As shown in the insets, controlled proceduralization as a pre-construction step produces more complete models. The completion is evaluated by comparing the amount of additional points in the new model, after CM models are generated and placed using edge-aware resampling to keep the point density similar – thus additional points most likely means more surfaces are complete or filled. For example, the completed model in Fig. 12 has 24% more points, compared to the original model with a similar local point density in an epsilon.

5.5. Structure-aware urban modeling

As mentioned in the introduction, the raw 3D data coming from manual modeling or acquisition systems does not contain any structural or semantic organization. However procedural representations inherently solve that problem and enable faster modeling. We implemented an interactive structure-preserving editing system that uses our procedural engine enhanced with some at-

Fig. 11. *Grammar variety*. Instead of representing a structure with a unique grammar, we are able to create a family of grammars, for architecturally varying structures.
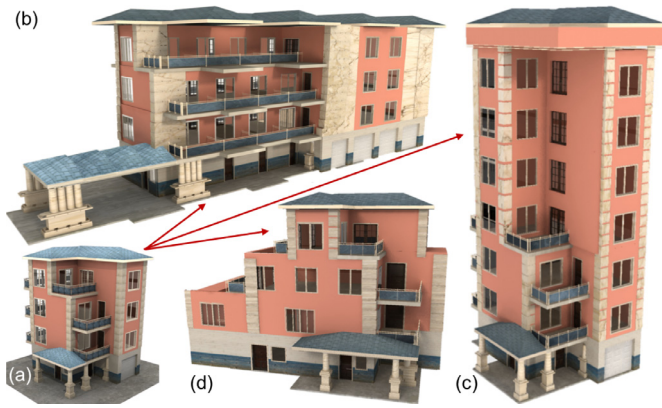
Fig. 14. *Urban modeling.* Our guided proceduralization allows creation of complex architectures using simple operations, with an average editing time of 15 min.

Fig. 15. *Procedural generation.* (a) Input map $\omega =$ "C&G", followed by the rendered model in (b) color-coded tree nodes, (c) labeled buildings, (d) textured buildings, and (e) street view.

tachment rules to preserve the adjacencies of grammar elements across rules (similar to [45]). We convert the instance tree into an instance graph, where the non-tree edges (newly included edges) contain the parameters of spatial adjacencies between not directly related nodes. When the user performs an edit, our framework optimizes the derivation for the propagation of the operation from the edited node to other nodes in the graph, preserving the attachment rules. After the optimization converges, the parameters are re-computed and the model is ready to be exported. Fig. 14 shows variations of a complex building structure synthesized by our editing system built upon the controlled proceduralization framework emphasizing (b) horizontal patterns, (c) vertical patterns, and (d) mixed patterns. The processing takes less than an hour to process and editing sessions are kept under 5 min for this model.

Without guided proceduralization, some users were confused about which patterns to use while editing. However, enabling user influence while generating patterns helped users to more intuitively edit the patterns that they defined, and reduced the overall editing time. Another benefit of guided proceduralization over proceduralization is that it reveals some patterns that would never have been discovered by prior proceduralization methods. Consider the last row of Fig. 5 and the last column of Fig. 8. The pattern parameters for those non-terminals are highly noisy, but guided proceduralization still captures the essence of double windows. Such cases are especially important in editing systems where the granularity of patterns should follow user's intuition. Our system enables such implicit granularity declarations over previous work converging on the smallest component configuration. A real-world example of this improvement can be observed at the roof of the gazebo in Fig. 11. Previous approaches would have discovered only the right-most configuration with too many rules and terminals (each indicated in a different color), in search of perfect repetitions. In contrast, guided proceduralization enables more compact configurations by auto-adjusting the granularity and rule/terminal similarity following the guidance (instances are colored with the same color), adding the flexibility to fit imperfect repetitions into rules.

5.6. Procedural content generation

Finally, we use our controlled proceduralization framework to create procedural models for large content creation, which is used for compression and efficient rendering. Previous approaches (e.g., [14,28,30,40,46]) either need a grammar for inverse procedural modeling, or use thresholds to create derivation trees. However our approach keeps the details of the procedural model under the hood and enables content generation based only on a geometric model, and optionally grammar specification values. We used
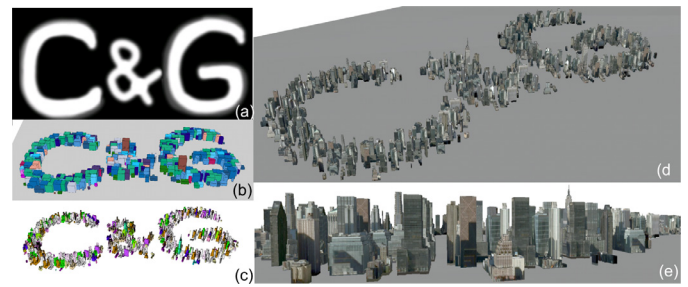
our controlled proceduralization framework to convert 4K buildings in the 180 km^2 metropolitan area of New York into a procedural representation, achieving 95% compression in size (reducing from 1.7M polygons to 175K polygons, and from 1060 MB of textures to 49 MB). We show a new city (called "CG York") generated using the extracted grammar in Fig. 15. We also demonstrate input map, nodes of the derivation tree, color-coded components, and a street view from our new city.

6. Discussions

6.1. Expressiveness

At first glance, one can claim that finding the most effective grammar (i.e., compact) is not possible, because finding the minimal description $|d(.)|$ covering all instances of all possible grammars requires computing their Kolmogorov complexity (i.e., $K(.)$), which is not a computable function. However, in our case i) grammars can be regarded as compressors that implicitly eliminate some redundant information and simplify the description, and ii) we do not seek a minimal description (i.e., $|d(.)| = K(.)$) but a description that approximates our target grammar specification values ($|d(.)| \approx K(.)$). We incorporate user specified Γ^* to decide on the level of simplification and to produce the most useful grammar based-on-need, as shown in different decompositions and rules of Fig. 9. For example, if we were to look for the minimal description of "abcabababab", we would have to find a description including "c". But if the user indicates that single occurrences are noise, or the label "c" can be simplified, or wants four instances, then an easy and efficient description would be "(ab)*5". In Fig. 9, such user specification is incorporated as different Γ^* values, and the results indicate how much variance is tolerated within each terminal group (similarity noise) and within each rule (like the "ab" example).

6.2. Guided proceduralization vs. controlled IPM

One option that some exploration algorithms in inverse procedural modeling use is providing the user with some example derivations and letting them guide the generation. In prior work (e.g., [28,29,46,47]) this approach assumed a known grammar and a discretization of the derivation space. In our case for proceduralization, we let the user specify desired grammar values (Γ^*) beforehand and perform an optimization. The distinction of those processes are more clear when visualized: Bottom row of Fig. 10 shows guided proceduralization output (the desired grammar), and Fig. 14 shows controlled IPM outputs (the desired instances of a grammar). For clarification, we do not input a grammar (e.g. $G = E(\Gamma)$), but we input perceptible grammar values (Γ^*), i.e., the alphabet size, repetition per rule, etc. (Section 4.2.1).

6.3. Optimization space

The optimization search space $\mathcal{P}(\Theta)$ has variables of geometric and topological similarity as discussed in definitions of Θ_S (Section 4.1.2) and Θ_D (Section 4.2.5). We set appropriate bounds for the search space parameters and run multiple iterations of the proceduralization process until we converge on the desired grammar specification values. Also, instead of a brute force exploration, we use BOBYQA algorithm [48] to converge faster in this well-bounded space without derivatives. This reduces the execution time of the optimization, from 1–1.5 days to 4–5 h on a single machine with 8 CPUs, for a model of 20K triangles. The optimization is also flexible enough to include other parameters Θ and specifications Γ if needed. We implemented an import/export mechanism for our intermediate steps. Thus, the current grammar can be exported at any stage of the optimization, which provides flexibility to start from near-optimum grammars for faster convergence. Also, the interactive user interface allows the underlying system to use the grammar from the last step of the optimization whenever new target values are set, instead of starting from scratch.

Although our prototype optimization may take a couple of hours, converting the models into the desired procedural representation is a one-time offline operation that does not consume any human labour. In addition, the procedural editing engine is completely online, enabling the user synthesize many complex models in minutes, following the procedural control that they defined. It is also possible to make the offline part more efficient. To converge faster, one can structure Θ to include differentiable features, or to create a convex optimization space, and then change the algorithm to a gradient-descent variation. Carrying the computation to GPUs and introducing parallelization for the optimization is also a suggested option. However in our experiments, we targeted for the most flexible setup to support all possible features.

6.4. User interaction

We have implemented two means of defining target grammar values Γ^* for users. The first one presents some GUI elements (sliders and input boxes) for defining the grammar elements defined at the end of Section 4.2.1. Users can set these relative values using the GUI, and the system optimizes for the grammar that best satisfies these values. The second setting visually provides suggestions as a variety of grammars where elements of each grammar choice are color-coded on the re-generated input model (Fig. 11). We achieve that by sampling the multi-dimensional grammar value space $[\Gamma_{m_{min}}, \Gamma_{m_{max}}]$ and filtering the results that converge to significantly different grammars. Then, we re-generate the input model using each of the candidate grammars, and demonstrate these instances of candidate grammars. The user then selects the best grammar by choosing the instance that visually contains patterns and components applicable to the use case.

After the grammar is generated, the procedural editing engine provides operations like push-pull, copy-paste, and replicate-join. All of those operations are conducted visually on the model via mouse/keyboard interactions. The user can copy terminals and non-terminals by clicking on them, which are adaptively pasted into their new nodes to fill its volume. Replicate and join operations change the application counts of patterns by simple strokes on the terminals and non-terminals. Finally, the user can pull a model and the procedural engine resizes the model as explained in Section 5.5. Note that this operation is different than just changing pattern parameters, because the adjacency graph of the nodes are preserved during editing.

7. Conclusion

In summary, our research shows that extending proceduralization frameworks to include user guidance discovers procedural representations for existing models so as to improve reconstruction and modeling. As the graphics and vision communities converge in automatic asset creation in almost all domains, many fascinating problems await to be solved by procedural approaches instead of manual and semi-automatic organization of geometry.

Looking forward, we have identified the following items of future work: (i) using a neural network model to approximate function $f(\Theta') \rightarrow \Gamma^*$ for better performance (to replace hours of optimization with minutes of prediction), (ii) bringing our controlled proceduralization framework to other domains, and (iii) proposing evaluation tools for output grammars to assess their generative capacity in a general fashion.

Acknowledgments

References

[1] Mathias M, Martinovic A, Weissenberg J, Gool LV. Procedural 3D building reconstruction using shape grammars and detectors. In: Proceedings of the 3DIMPVT. In: 3DIMPVT. Washington, DC, USA: IEEE Computer Society; 2011. p. 304–11. doi:10.1109/3DIMPVT.2011.45. ISBN 978-0-7695-4369-7.

[2] Vanegas CA, Aliaga DG, Benes B. Building reconstruction using Manhattan-world grammars. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR); 2010. p. 358–65. doi:10.1109/CVPR.2010.5540190.

[3] Parish YIH, Müller P. Procedural modeling of cities. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques. In: SIGGRAPH. New York, NY, USA: ACM; 2001. p. 301–8. doi:10.1145/383259.383292. ISBN 1-58113-374-X.

[4] Smelik RM, Tutenel T, Bidarra R, Benes B. A survey on procedural modelling for virtual worlds. Comput Graph Forum 2014;33(6):31–50. doi:10.1111/cgf.12276.

[5] Aliaga DG, Demir I, Benes B, Wand M. Inverse procedural modeling of 3D models for virtual worlds. In: Proceedings of the ACM SIGGRAPH Courses. In: SIGGRAPH. New York, NY, USA: ACM; 2016. p. 16:1–16:316. doi:10.1145/2897826.2927323. ISBN 978-1-4503-4289-6

[6] Martinovic A, Gool LV. Bayesian grammar learning for inverse procedural modeling. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR); 2013. p. 201–8. doi:10.1109/CVPR.2013.33.

[7] Weissenberg J, Riemenschneider H, Prasad M, Van Gool L. Is there a procedural logic to architecture?. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), Portland, Oregon, USA. Piscataway, NJ: IEEE; 2013. p. 185–92.

[8] Demir I. A generalized proceduralization framework for urban models with applications in procedural modeling, synthesis, and reconstruction. Purdue University; 2017. Ph.D. thesis.

[9] Ritchie D, Thomas A, Hanrahan P, Goodman ND. Neurally-guided procedural models: amortized inference for procedural graphics programs using neural networks. In: Lee DD, Sugiyama M, von Luxburg U, Guyon I, Garnett R, editors. Advances in neural information processing systems (NIPS); 2016.

[10] Beneš B, Šťáva O, Měch R, Miller G. Guided procedural modeling. In: EG full papers. Eurographics Association; 2011. p. 325–34.

[11] Demir I, Aliaga DG, Benes B. Procedural editing of 3D building point clouds. In: Proceedings of the IEEE international conference on computer vision (ICCV); 2015a. p. 2147–55. doi:10.1109/ICCV.2015.248.

[12] Demir I, Aliaga DG, Benes B. Proceduralization for editing 3D architectural models. In: Proceedings of the international conference on 3d vision (3DV); 2016. p. 194–202. doi:10.1109/3DV.2016.28.

[13] Alexander C, Ishikawa S, Silverstein M. A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press; 1977.

[14] Wonka P, Wimmer M, Sillion F, Ribarsky W. Instant architecture. ACM Trans Graph 2003;22(3):669–77. doi:10.1145/882262.882324.

[15] Müller P, Wonka P, Haegler S, Ulmer A, Van Gool L. Procedural modeling of buildings. ACM Trans Graph 2006;25(3):614–23. doi:10.1145/1141911.1141931.

[16] Schwarz M, Muller P. Advanced procedural modeling of architecture. ACM Trans Graph 2015;34(4):107:1–107:12. doi:10.1145/2766956.

[17] Vanegas CA, Aliaga DG, Beneš B, Waddell PA. Interactive design of urban spaces using geometrical and behavioral modeling. ACM Trans Graph 2009;28(5):111:1–111:10. doi:10.1145/1618452.1618457.

[18] Lipp M, Wonka P, Wimmer M. Interactive visual editing of grammars for procedural architecture. ACM Trans Graph 2008;27(3):102:1–102:10. doi:10.1145/1360612.1360701.

[19] Aliaga DG, Rosen PA, Bekins DR. Style grammars for interactive visualization of architecture. IEEE Trans Vis Comput Graph 2007;13(4):786–97. doi:10.1109/TVCG.2007.1024.

[20] Zhou Q-Y, Neumann U. 2.5d building modeling by discovering global regularities. In: Computer vision and pattern recognition (CVPR); 2012. p. 326–33. doi:10.1109/CVPR.2012.6247692.

[21] Toshev A, Mordohai P, Taskar B. Detecting and parsing architecture at city scale from range data. In: Computer vision and pattern recognition; 2010. p. 398–405. doi:10.1109/CVPR.2010.5540187.

[22] Bao F, Schwarz M, Wonka P. Procedural facade variations from a single layout. ACM Trans Graph 2013;32(1):8:1–8:13. doi:10.1145/2421636.2421644.

[23] 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.

[24] Musialski P, Wonka P, Aliaga DG, Wimmer M, van Gool L, Purgathofer W. A survey of urban reconstruction. Comput Graph Forum 2013;32(6):146–77. doi:10.1111/cgf.12077.

[25] Gadde R, Marlet R, Paragios N. Learning grammars for architecture-specific facade parsing. Int J Comput Vis 2016;117(3):290–316.

[26] Teboul O, Kokkinos I, Simon L, Koutsourakis P, Paragios N. Parsing facades with shape grammars and reinforcement learning. IEEE Trans Pattern Anal Mach Intell 2013;35(7):1744–56.

[27] Talton J, Yang L, Kumar R, Lim M, Goodman N, Měch R. Learning design patterns with bayesian grammar induction. In: Proceedings of the 25th annual ACM symposium on user interface software and technology. In: UIST. New York, NY, USA: ACM; 2012. p. 63–74. doi:10.1145/2380116.2380127. ISBN 978-1-4503-1580-7.

[28] Talton JO, Lou Y, Lesser S, Duke J, Měch R, Koltun V. Metropolis procedural modeling. ACM Trans Graph 2011;30(2):11:1–11:14. doi:10.1145/1944846.1944851.

[29] Nishida G, Garcia-Dorado I, Aliaga DG, Benes B, Bousseau A. Interactive sketching of urban procedural models. ACM Trans Graph 2016;35(4):130:1–130:11. doi:10.1145/2897824.2925951.

[30] Vanegas CA, Garcia-Dorado I, Aliaga DG, Benes B, Waddell P. Inverse design of urban procedural models. ACM Trans Graph 2012;31(6):168:1–168:11. doi:10.1145/2366145.2366187.

[31] Si Z, Zhu SC. Learning and-or templates for object recognition and detection. IEEE Trans Pattern Anal Mach Intell 2013;35(9):2189–205. doi:10.1109/TPAMI.2013.35.

[32] Demir I, Hughes F, Raj A, Dhruv K, Muddala SM, Garg S, et al. Generative street addresses from satellite imagery. ISPRS Int J Geo Inf 2018;7(3). doi:10.3390/ijgi7030084.

[33] Bokeloh M, Wand M, Seidel H-P. A connection between partial symmetry and inverse procedural modeling. ACM Trans Graph 2010;29(4):104:1–104:10. doi:10.1145/1778765.1778841.

[34] Kalojanov J, Bokeloh M, Wand M, Guibas L, Seidel H-P, Slusallek P. Microtiles: extracting building blocks from correspondences. Comput Graph Forum 2012;31(5):1597–606. doi:10.1111/j.1467-8659.2012.03165.x.

[35] Lipman Y, Chen X, Daubechies I, Funkhouser T. Symmetry factored embedding and distance. ACM Trans Graph 2010;29(4):103:1–103:12. doi:10.1145/1778765.1778840.

[36] Demir I, Aliaga DG, Benes B. Coupled segmentation and similarity detection for architectural models. ACM Trans Graph 2015;34(4):104:1–104:11. doi:10.1145/2766923.

[37] Attene M, Falcidieno B, Spagnuolo M. Hierarchical mesh segmentation based on fitting primitives. Vis Comput 2006;22(3):181–93. doi:10.1007/s00371-006-0375-x.

[38] Holz D, Ichim AE, Tombari F, Rusu RB, Behnke S. Registration with the point cloud library: A modular framework for aligning in 3-d. IEEE Robot Autom Mag 2015;22(4):110–24. doi:10.1109/MRA.2015.2432331.

[39] Rusu RB, Cousins S. 3D is here: point cloud library (PCL). In: Proceedings of the IEEE international conference on robotics and automation (ICRA); 2011. p. 1–4. doi:10.1109/ICRA.2011.5980567.

[40] Demir I, Aliaga DG, Benes B. Proceduralization of buildings at city scale. In: Proceedings of the 2nd international conference on 3d vision (3DV), vol. 1; 2014. p. 456–63. doi:10.1109/3DV.2014.31.

[41] Lafarge F, Alliez P. Surface reconstruction through point set structuring. Comput Graph Forum 2013;32(2pt2):225–34.

[42] Institute fur Kartographie und Geoinformatik - www.ikg.uni-hannover.de/index.php?id=413A. 2015.

[43] Pauly M, Mitra NJ, Wallner J, Pottmann H, Guibas LJ. Discovering structural regularity in 3D geometry. ACM Trans Graph 2008;27(3):43:1–43:11. doi:10.1145/1360612.1360642.

[44] Simon L, Teboul O, Koutsourakis P, Gool LV, Paragios N. Parameter-free/pareto-driven procedural 3d reconstruction of buildings from ground-level sequences. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2012. p. 518–25.

[45] Bokeloh M, Wand M, Seidel H-P, Koltun V. An algebraic model for parameterized shape editing. ACM Trans Graph 2012;31(4):78:1–78:10. doi:10.1145/2185520.2185574.

[46] Ritchie D, Mildenhall B, Goodman ND, Hanrahan P. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. ACM Trans Graph 2015;34(4):105:1–105:11. doi:10.1145/2766895.

[47] Yumer ME, Asente P, Mech R, Kara LB. Procedural modeling using autoencoder networks. In: Proceedings of the 28th annual ACM symposium on user interface software & technology. UIST. New York, NY, USA: ACM; 2015. p. 109–18. ISBN 978-1-4503-3779-3.

[48] Powell MJ. The BOBYQA algorithm for bound constrained optimization without derivatives. Cambridge NA Report NA2009/06. University of Cambridge, Cambridge; 2009.