Proceduralization of Buildings at City Scale

Ilke Demir
Computer Science
Purdue University
West Lafayette, IN, US
idemir@purdue.edu

Daniel G. Aliaga
Computer Science
Purdue University
West Lafayette, IN, US
aliaga@purdue.edu

Bedrich Benes
Computer Graphics Technology
Purdue University
West Lafayette, IN, US
bbenes@purdue.edu

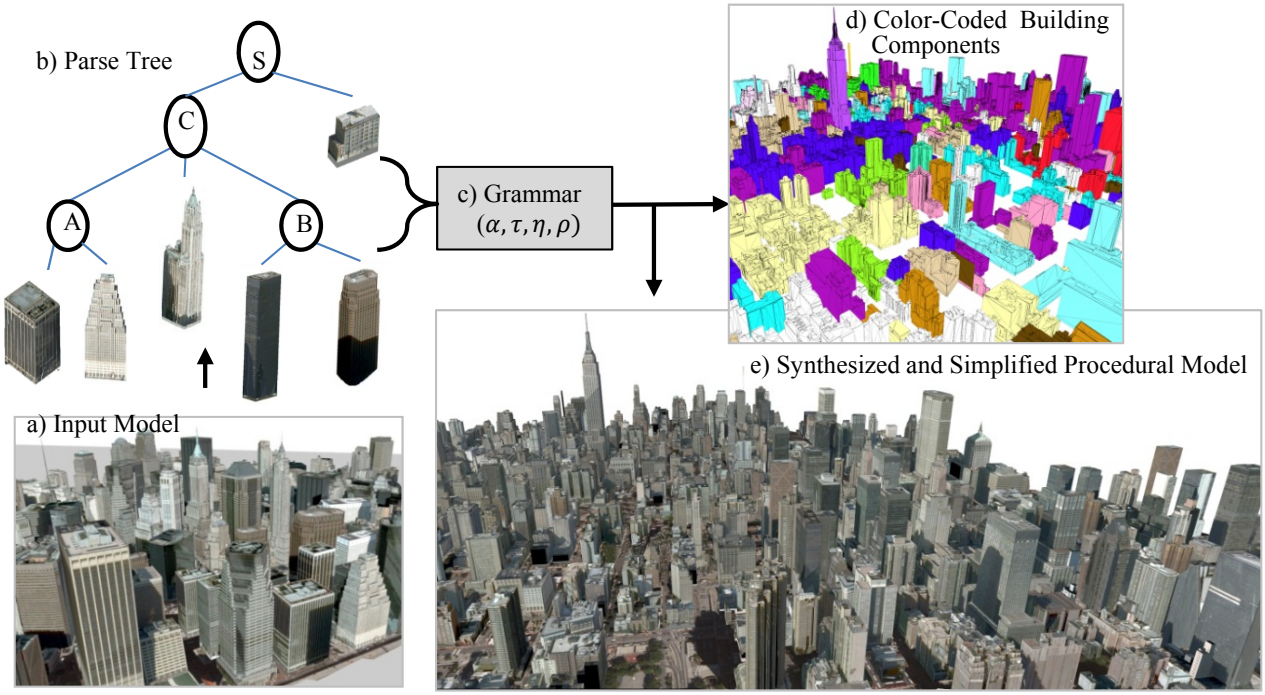

Figure 1. **City-Scale Building Proceduralization.** Our approach uses a) an unorganized 3D model as input, b) computes a hierarchical clustering of building components, and c) extracts a context-free grammar of the urban area. We can procedurally generate (d-e) structurally-similar cities, at a chosen de-instancing level

***Abstract*—We present a framework for the conversion of existing 3D unstructured urban models into a compact procedural representation that enables model synthesis, querying, and simplification of large urban areas. During the de-instancing phase, a dissimilarity-based clustering is performed to obtain a set of building components and component types. During the proceduralization phase, the components are arranged into a context-free grammar, which can be directly edited or interactively manipulated. We applied our approach to convert several large city models, with up to 19,000 building components spanning over 180 km squares, into procedural models of a few thousand terminals, non-terminals, and 50-100 rules.**

***Keywords*-Graphics, Procedural modeling, Proceduralization, Urban modeling, Reconstruction**

I. Introduction

The demand for city-scale 3D urban models has significantly increased due to the proliferation of urban planning, city navigation, and interactive applications. Some existing methods focus on automating 3D reconstruction from images (e.g., [17]) or from LiDAR (e.g., [28]). Other methods assume widespread human assistance (e.g., Trimble Sketchup), or expert modeling (e.g., CityEngine). The results of these approaches vary tremendously, ranging from unorganized triangle soups to highly-structured parameterized models. Manual and reconstructed models are detailed and realistic; but lack of structure causes inefficient editing, storage and rendering.

The key observation of our work is that there is a huge potential to exploit repetitions and to organize similarities into meaningful procedural models in existing urban spaces. While procedural models are known to be a powerful and compact parameterized representation, it requires significant manual expertise to procedurally code a city. The pioneering work of Parish and Mueller [18], and subsequent urban modeling papers (e.g., see surveys [23], [17]) have focused on city-scale procedural modeling and on building-scale inverse modeling. To the best of our knowledge, our methodology is the first to automatically proceduralize at city scale. We present a framework for finding geometric components, repetition,

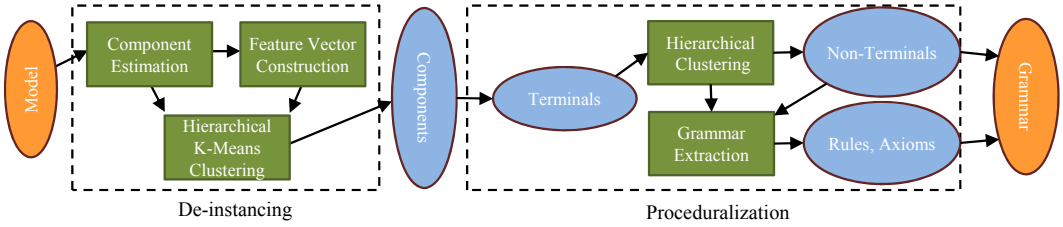

Figure 2. **Pipeline.** From the input model, we perform de-instancing to obtain component types which are then given to proceduralization (hierarchical clustering and grammar extraction) to output a grammar for the city.

and a high-level structural organization of a large collection of buildings and for creating a compact procedural representation that enables their synthesis, querying, and efficient rendering (Figure 1). Our input (Figure 2) is an existing 3D city model represented as an unorganized but renderable collection of 3D buildings (e.g., any combination of polygons, textured geometry, points, procedural content, or hand-crafted buildings; without a strict concern on building types). Our automatic pipeline has two phases.

1) De-Instancing: A similarity-based clustering of instances of building components is performed to obtain a set of components and component types.
2) Proceduralization: Repetitive spatial patterns are discovered amongst the components and result in a compact context-free grammar applicable to synthesis, querying, and other procedural applications.

Our work closely relates to Kolmogorov complexity and compression where the task is to find a smaller representation for an input [4]. Although finding the minimal representation is known to be incomputable, discovering repetitions, similarities, and instantiations are ways to find a smaller and more versatile representation.

We have used our approach to convert parts of several large cities (e.g., New York, Chicago, and San Francisco) into procedural models, containing 10,000 to 19,000 building components. The resulting grammars can be directly edited (e.g., as a text file), interactively manipulated (e.g., [14] or CityEngine), and several types of information queries can be performed (e.g., localizing oneself within a city based on the uniqueness of the nearby observed buildings, as in [2]). Moreover, proceduralized models have been demonstrated to be friendly to very efficient GPU-based rendering (e.g., [8], [11], [20]).

Our framework enables automatically reducing geometry and texture sizes by 2 to 21 times, producing grammars with 177 to a few thousand terminals and 5 to approximately 100 non-terminals. Preprocessing takes 2-4 hours for a city of 3,000 to 6,000 buildings. Editing and synthesis takes seconds.

Our main contributions include

- a framework to convert an existing collection of 3D building models into a procedural representation,
- a feature-based de-instancing algorithm to determine building components and component types,
- a hierarchical clustering algorithm that finds repetition and spatial patterns of building components and enables extracting a compact context-free grammar,

II. Previous Work

Our paper builds upon urban modeling and procedural modeling. While urban reconstruction work, such as Lafarge and Mallet [12] who segment the model into buildings, trees, and ground, while Lin et al. [13] who perform semantic reconstruction of medium-sized LiDAR scenes, generate 3D models, none of these reconstruction approaches focus on producing a detailed city-scale proceduralization of the underlying models.

Urban procedural modeling has become a powerful paradigm since the pioneering work of Parish and Mller [18] and several follow-on building modeling papers (e.g.,[15], [26]). Also Smelik et al. [19] provides a survey of procedural methods for virtual worlds. However, these methods define the procedural model and/or assume a provided procedural model.

More recently, several inverse procedural modeling methods have been proposed that attempt to extract a procedural model from the input. Initial works provided semi-automatic and automatic building solutions (e.g.,[1], [5], [24], [21]) and facade solutions (e.g., [3], [9], [16], [27]), most of which rely on pre-segmented components, user input, known grammar, or layout.

Most related to our work are the city-scale inverse modeling approaches of Toshev et al. [22] and Vanegas et al. [25]. The former focuses on detailed extraction of features from point clouds, while our method works with visual similarity by considering more complex feature descriptors. The latter describes an inverse modeling approach at city-scale but assumes a parameterized procedural model as input. Our method does not assume any a priori knowledge of the procedural model.

III. Overview

A. Component Definition

A building has one or more vertically stacked components, each being a different part of the building.

Components. We define an initial set of n_C components, each labeled C_i and being oriented bounding boxes rotated about the z-axis with minimum/maximum coordinate corners p_i and P_i and midpoint position $m_i = (P_i + p_i)/2$.

Feature Vector. Each component has a c-dimensional feature vector $F_i = f_i^1, f_i^2, \ldots, f_i^c$ where each scalar f_i^u represents a geometric and/or visual property of C_i similar to those of Doersch et al. [6] (see Section 4).

Component Types. We also define an initial set of n_T component types. Component type T_k is defined by a dissimilarity metric. For C_i and C_j, their dissimilarity is

$$d_{ij} = d_{ji} = ||W(F_i - F_j)|| \quad (1)$$

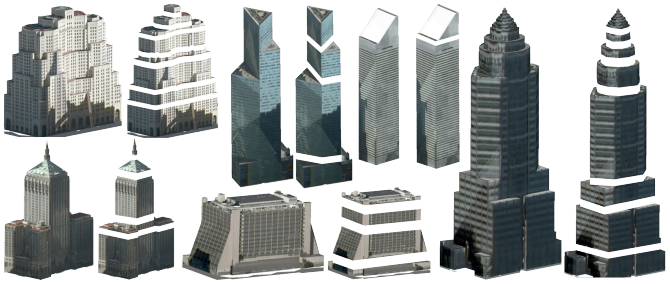

Figure 3. **Building Components and Styles.** Components of different styles of buildings are shown. Note that slanted roofs/facades are componentized as expected.

where $W \in R^c$ represents a per-feature unit-length weight vector. Thus, $C_i \in T_k$ and $C_j \in T_k$ if and only if $d_{ij} \leq t_d$ for user-defined de-instancing threshold t_d (Section 4D will expand upon the use of this threshold and weight vector W). The component instance closest to the cluster mean is selected as the component type representative C_k^*.

Additional Relations. We also define

$$dist(C_i, C_j) = ||m_i - m_j|| \tag{2}$$

$$nbr(C_i, C_j) = \begin{cases} 1 & dist(C_i, C_j) \leq \tau \\ 0 & otherwise \end{cases} \tag{3}$$

$$mht(C_i, C_j) = |m_i - m_j|_{L1} \tag{4}$$

Eqn (2) measures the unsigned Euclidean distance between (the midpoints of) two components; Eqn (3) indicates whether two components are considered neighbors; and Eqn (4) measures the Manhattan, or L1, distance between the components' midpoints, which is the sum of block distances in the xy plane.

B. Parse Tree Definition

The components are organized into a parse tree of n_P nodes. The tree organizes similar components and their repetition so as to enable grammar output.

Nodes. Each node N_k corresponds to a set of component types. A leaf node corresponds one-to-one to the initial component types; e.g., $N_k = \{T_k\}$. Nodes higher in the tree are created by clustering nodes; e.g., $N_k = \{N_{l1}, N_{l2}, \ldots, N_{n_{N_K}}\}$ where N_{l*} are children of N_k and n_{N_K} are the number of types clustered to node N_k. An instance of the component type of node N_k is labelled $\tilde{N}_k$, which corresponds to a $C_k \in T_k$.

Edges. A parse tree edge stores the relative transformation matrix E_{kl} from parent node N_k to child node N_l. Since multiple instances may exist of N_k and N_l, E_{kl} is obtained by computing the affine transformation from each $\tilde{N}_k$ to its corresponding $\tilde{N}_l$ and using the average rotation, translation, and scale change to a define a single matrix.

Grammar Definition Our approach converts the parse tree into a grammar defined as $G^* = \{\alpha, \tau, \eta, \rho\}$ where α is the starting axiom, τ is the set of terminals, η is a set of non-terminals, and ρ is a set of rewriting rules using the terminals and non-terminals.

The parse tree leaves, corresponding to representatives of the initial component types, form the terminal set:

$$\tau = \bigcup_{1 \leq k \leq n_T} C_k^* \tag{5}$$

All other nodes form the non-terminal set:

$$\eta = \bigcup_{n_T \leq k \leq n_P} N_k \tag{6}$$

where n_T and n_P are number of terminals and nodes, respectively. We collect all the children nodes of each node to define a single rewrite rule per node, namely:

$$\rho = \{\rho_k \quad k \in [n_T, n_P]\} \rho_k = (N_k \rightarrow \ \oplus_{N_l \ is \ child \ of \ N_k} E_{kl} N_l) \tag{7}$$

and $\oplus$ means concatenation and E_{kl} is the transformation matrix (see Figure 5).

IV. De-Instancing Phase

We describe how to classify the instances of all buildings into a set of components C_i and component types T_k. We estimate cut planes, fit component boxes, create feature vectors, and perform dissimilarity clustering (Figure 4 and Supplemental Figures S1 and S2).

A. Cut Plane Estimation

To estimate building components, our approach finds cut planes parallel to the ground plane. The volume between two cut planes defines a building component. We progressively move the near plane closer to the viewer along the $+z$ axis and render the building to framebuffer image I_i. Meanwhile the far plane starts from the ground level and progresses as the most recently detected cut plane. The process repeats until framebuffer image I_{i+1} is empty. A cut plane is created whenever there is a significant change in the pixel difference between the rendered contours of consecutive framebuffer images (i.e., a non-zero second derivative). Using the second derivative enables supporting a variety of facade and slanted roof geometries as seen in Figure 3.

B. Component Box Fitting

An oriented bounding-box (OBB) is fit to each component volume. The OBB is computed by rendering the component geometry (i.e., rendering all geometry and using the two cut planes as near and far planes), fitting an oriented rectangle to its projection on the framebuffer, and extruding it. Since the OBB is computed from the rendered result, it works with any renderable 3D representation.

C. Component Feature Vector Creation

To classify the component instances into component types, we compute an 11-dimensional feature vector (Supplemental Figure S1). We render each component from four viewing directions, each rotated by 90-degrees about the $+y$ axis, and then average them. While not all buildings necessarily have four sides at 90-degrees to each other, this assumption works well in practice. For each side, our technique applies an intensity threshold, image binarization (using Otsu's method) and morphological opening/closing (using a 3x3 kernel) to yield a processed mask. The features are listed below.

- Component shape: One efficient option is to use the number of vertices of a simplified polygon (using Douglas-Peucker

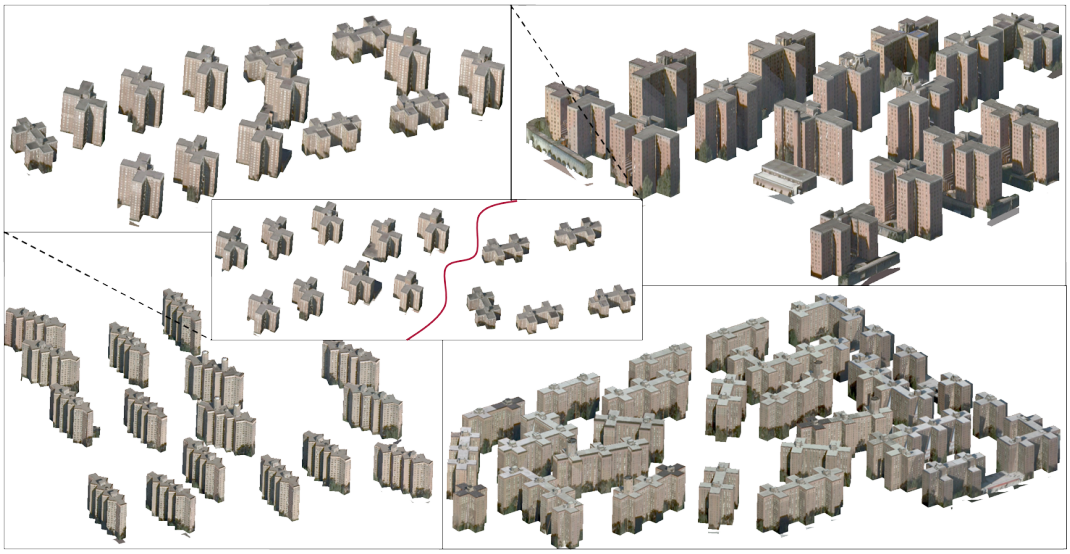

Figure 4. **Building Clusters.** Four building clusters computed automatically based on feature vector similarity. Inset: if the feature weights are adjusted such that building height is emphasized, the first cluster is divided into two separate categories.

algorithm) enclosing the foreground of the image as an indication of shape.

- Window size and spacing: Our approach calculates the average window width and height and average x-spacing and y-spacing. k-means clustering groups the connected regions in the processed mask by window area. Each cluster represents a particular window type. After removing outliers, the method computes a tight-fitting bounding box for each window in the dominant cluster, and computes their average width and height. Next, the method creates horizontally- and vertically-sorted arrays of window midpoints. The x-spacing (y-spacing) value is equal to the most frequently occurring distance between window midpoints, minus the window width (height).
- Emptiness: Our method computes the average ratio of background pixels to window pixels within each window's tightly fit bounding rectangle.
- Background intensity: Our algorithm computes the average grayscale intensity of the background pixels.
- Component size: Our vector includes the width, height, depth, and rotation angle of the component's box in world space.

D. Dissimilarity Clustering

Our algorithm uses k-means clustering to perform a dissimilarity clustering of the feature vectors and to determine the component types. Changing the de-instancing threshold t_d affords different levels of clustering; e.g., a larger t_d produces fewer component types, each of more instances. For example, it can be interpreted as increasing the simplification level of the city or as a trade off of expressiveness vs. compression of the grammar. As an implementation detail, the way we de-instance is to build a k-means hierarchy with k initially equal to $n_{c/2}$ and successively halved until $k = 1$. Each horizontal cut through the tree corresponds to a de-instancing result for a different t_d value. Further, we observed that increasing the weights of window-related features in W, benefits street-level rendering while giving relatively more

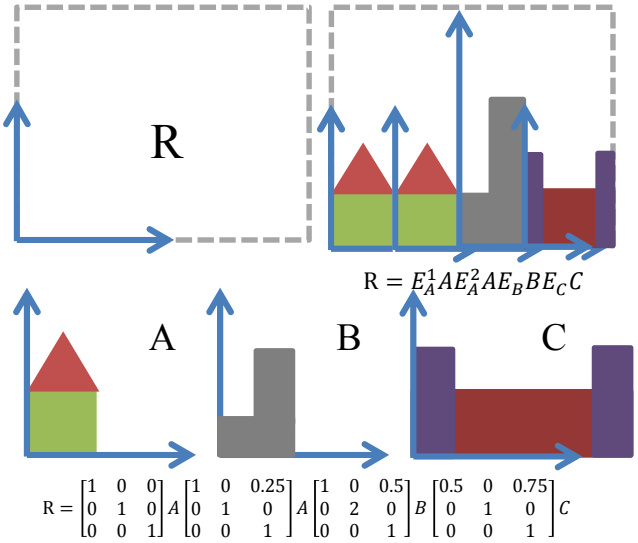

Figure 5. **Grammar Example.** A pictorial 2D example for a rule R→ AABC. The space of a non-terminal is filled with components A, B and C, using the average transformation matrices shown in Rule R.

weight to component size features behooves larger-scale scene rendering. Changing the other weights in W is also useful for re-assembling 3D urban data compactly based on a component-based approach (see Figure 4 inset). All component types always span input model. (See Supplementary Figure S2 for an example of de-instantiation.)

V. Proceduralization Phase

We describe how to define a proceduralization graph, which together with our distance metric, assists in creating a parse

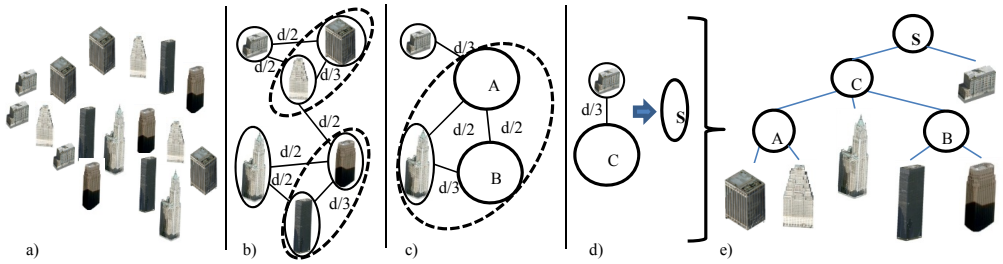

Figure 6. **Proceduralization.** a) The initial terminals in a city. b) The initial component type graph for Girvan-Newman clustering. For edge values $w_m = 1$ and $w_s = 0$ and for edges of buildings within "d" distance; we also do not show unique edges. Dashed ovals are the first clusters (i.e., nonterminals A and B). c) Another level producing nonterminal C. d) Final clustering producing the axiom S. e) The parse tree of the small city.

tree, extracting a grammar, and ultimately exploiting patterns and repetition of geometry (Figure 6).

A. Distance Metric

We use a custom distance metric for clustering based on three properties for a pair of nodes N_{k_1} and N_{k_2}.

Repetition: To identify frequent co-occurrences, we use the sum of pairwise occurrences:

$$r_{k_1\ k_2} = \sum_{\tilde{N}_{k_1} \in N_{k_1};\ \tilde{N}_{k_2} \in N_{k_2}} nbr(\tilde{N}_{k_1}, \tilde{N}_{k_2}) \tag{8}$$

Consistency: To find components separated by a nearly constant distance, we look at the standard deviation,

$$\sigma_{k_1\ k_2} = \tag{9}$$
$$\left(\frac{\sum_{\tilde{N}_{k_1} \in N_{k_1}; \tilde{N}_{k_2} \in N_{k_2}} \left(\overline{mht(\tilde{N}_{k_1}, \tilde{N}_{k_2})} - mht(\tilde{N}_{k_1}, \tilde{N}_{k_2}) \right)^2}{min(||N_{k_1}||, ||N_{k_2}||)} \right)^{0.5}$$

where $\overline{mht(\tilde{N}_{k_1}, \tilde{N}_{k_2})}$ is the mean Manhattan distance and $||N_k||$ implies the number of instances of the node.

Closeness: We also measure the normalized compactness of spatial grouping:

$$\mu_{k_1\ k_2} = \frac{\sum_{\tilde{N}_{k_1} \in N_{k_1};\ \tilde{N}_{k_2} \in N_{k_2}} dist(\tilde{N}_{k_1}, \tilde{N}_{k_2})}{min(||N_{k_1}||, ||N_{k_2}||)} \tag{10}$$

Hence, the complete distance metric for capturing consistently close patterns with high repetition is

$$D(T_k, T_l) = \frac{w_\mu \mu_{kl} + w_\sigma \sigma_{kl}}{r_{kl}} \tag{11}$$

The weights w_μ and w_σ are used for normalization and to explicitly alter the relative importance of the structural properties and thus support a variety of clustering styles; e.g., $w_\mu = 1$ and $w_\sigma = 0$ for frequently occurring spatially-near components, and $w_\mu = 0$ and $w_\sigma = 1$ for frequently occurring patterns for consistent but not necessarily close distances.(See Supplementary Figure S3).

B. Hierarchical Clustering

To compute the hierarchical parse tree, we iteratively cluster the graph vertices using Girvan and Newman clustering [7]. Initially, we create a graph of nodes with one node for each of the n_T component types. Graph edges store the value of our distance metric over all pairs of node types (note: edges for distant node type pairs are omitted). Since the total number of parse tree nodes is not known a priori, a k-means clustering or graph-partitioning algorithm is cumbersome. Instead, our clustering method progressively finds the communities [7] and removes edges causing graph vertices to be merged. The method converges to a set of clusters that are mutually close, in terms of our distance function, and with relatively few (small-valued) links to other communities. Then, we increase the clustering tolerance and perform another iteration of Girvan-Newman's method until one vertex remains (i.e., a single parse tree).

Simp Level	Poly (K)	Tex (MB)	Comps (K)	Geo Ratio	Tex Ratio	Terms	Non-Terms
NY 0	1700	1060	19	1.00	1.00		
1	989	515	9.5	1.72	2.06	2380	83
2	698	249	4.75	2.44	4.26	1428	
3	318	114	2.4	5.35	9.30	476	
4	175	49	1.2	9.71	21.63	256	11
SF 0	647	781	14	1.00	1.00		
1	380	462	7	1.70	1.69	1425	51
2	267	212	3.5	2.42	3.68	858	
3	147	90	1.75	4.40	8.68	288	
4	77	53	0.88	8.40	14.74	177	9
Chi 0	374	556	10	1.00	1.00		
1	246	233	5	1.52	2.39	1142	49
2	134	142	2.5	2.79	3.92	686	
3	77	72	1.25	4.86	7.72	230	
4	45	34	0.6	8.31	16.35	134	8

Table 1. **Model Statistics.** First row is the original size (e.g., NY 0); subsequent 4 rows use progressively larger de-instancing thresholds. Ratios are the reductions as compared to original.

C. Grammar Generation

To output the current instantiation G of the grammar G^* (Figure 6) a pre-order traversal of the parse tree is performed. All children of the root (i.e., the starting axiom) are placed after a transformation matrix which globally positions the node. Then, each rule ρ_k is applied relative to the coordinate frame defined by the applied node. An additional matrix is inserted in-between a pair of non-terminals

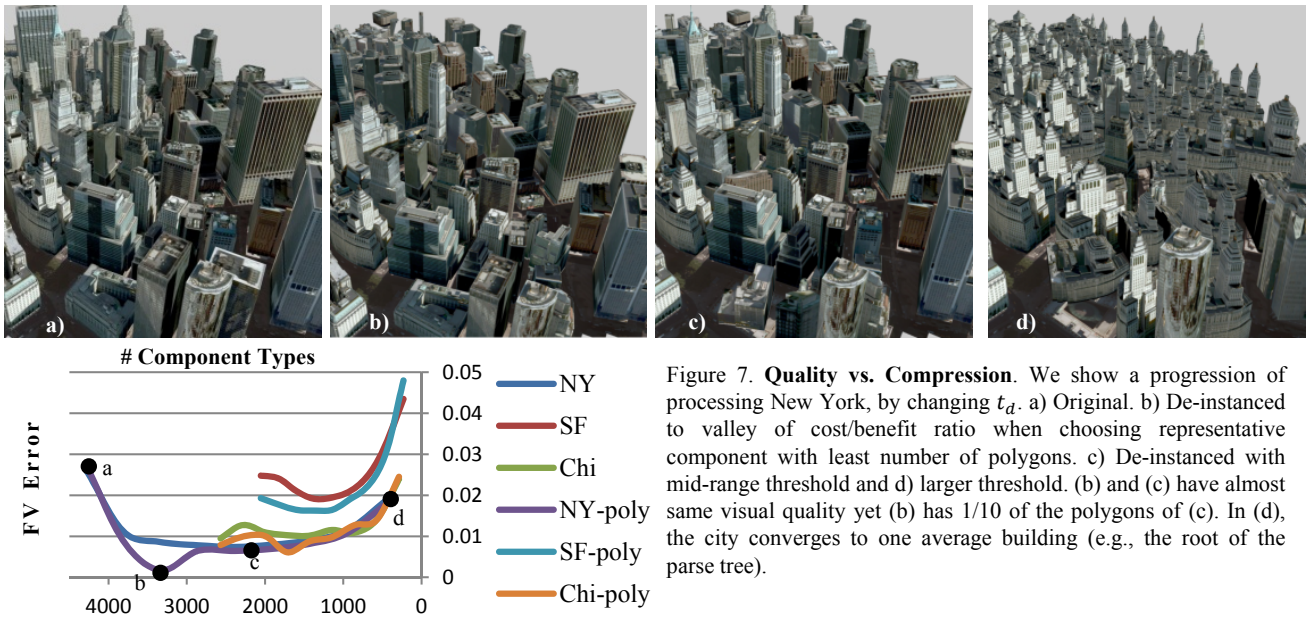

Figure 7. **Quality vs. Compression**. We show a progression of processing New York, by changing t_d. a) Original. b) De-instanced to valley of cost/benefit ratio when choosing representative component with least number of polygons. c) De-instanced with mid-range threshold and d) larger threshold. (b) and (c) have almost same visual quality yet (b) has 1/10 of the polygons of (c). In (d), the city converges to one average building (e.g., the root of the parse tree).

to ensure the second non-terminal is properly positioned (relative to the parent), as its edge indicates.

Repetitions. Our approach reproduces hierarchical (or sequential) pairwise repetitive patterns. Any observed pattern can be reproduced but the type of repetitive pattern (e.g., grid pattern, circular, etc.) is neither explicitly recognized nor parameterized. Nevertheless, our clustering approach can group the entire pattern into a single rule and thus enable applying it one or more times.

VI. Results and Applications

Our framework is implemented in C++ using Qt, OpenGL, OpenCV, dlib-ml library [10], Intel i7 Desktop PC, and NVIDIA GTX 680 graphics card. We have applied it to large triangle mesh models of New York, Chicago, and San Francisco obtained from a 3D CAD Browser.

Figures 4 and 7 show de-instancing. Figure 4 shows four building clusters, as a result of our distance metric. The inset in Figure 4 demonstrates how the clustering can be purposefully changed by, for example, giving more importance to height. Figure 7 contains a de-instancing progression. The graph in Figure 7a shows two cost/benefit curves for each model. The cost is the average standard deviation of all clusters (i.e., feature vector error) and the benefit is the number of polygons eliminated while de-instancing for various threshold values t_d. For the curve labels ending in "-poly", the representative component (per cluster) is chosen as the one with the least number of polygons for the other curves the representative component is the component closest to the mean of the cluster. The cost/benefit curves imply that a good trade off occurs near the bottom of the "u" shape. Moreover, the selection of the least cost representative components does yield an overall benefit (i.e., 10x less polygons at about same clustering error). Figures 7b-d demonstrate the model at various locations along the de-instancing progression.

Table 1 and Supplementary Table S1 contains statistics and comparisons of our work. Table 1 contains a numeric summary of size reductions and grammar statistics of four de-instancing threshold levels. It shows compression and simplification as a side effect of our method: lossless for level 0 and feature error dependent for other levels. Table S1 compares characteristics of our work to those of others. Supplemental Figure S1 focuses on the feature vectors of several component facades over the city. Our approach automatically handles incomplete facades and missing textures per face, provided that the missing parts are present in the repetitions of the same facade. Supplementary Figure S2 shows the components of a neighborhood, with similar and dissimilar building analysis.

Our approach enables a variety of applications. Figure 8 demonstrates original and proceduralized examples of Chicago and San Francisco. Figure 9 contains an example of user-controlled new model synthesis. Our system fills-in a new starting symbol as best as possible using the grammar from a proceduralized city. In this way, we have instanced a New York style city with similar building structures as per their feature vectors and with similar spatial arrangements, but with the shape of the letters "3DV" – inspired by inverse procedural modeling (e.g.,[21]) (See video).

Finally, Figure 10 demonstrates an interesting potential application of our system for localizing oneself within a city (see [2] for a more detailed explanation of this application).Our proceduralization work enables an alternative and simpler method to enable self-localization from ground-level images. Given a street view of New York (Figure 10a; from Google Street View), our prototype application rectifies the images using manually-estimated camera rotation parameters (Figure 10b). Then, our proceduralization method treats the images just like the input buildings, by extracting their feature vector and placing them into the parse tree. It finds the co-occurrence of all images from the grammar and automatically determines that the observed set of window distributions/styles only occurs adjacently in 4 locations within the city. Hence, the observer of Figure 10a is known to be at one of those locations.

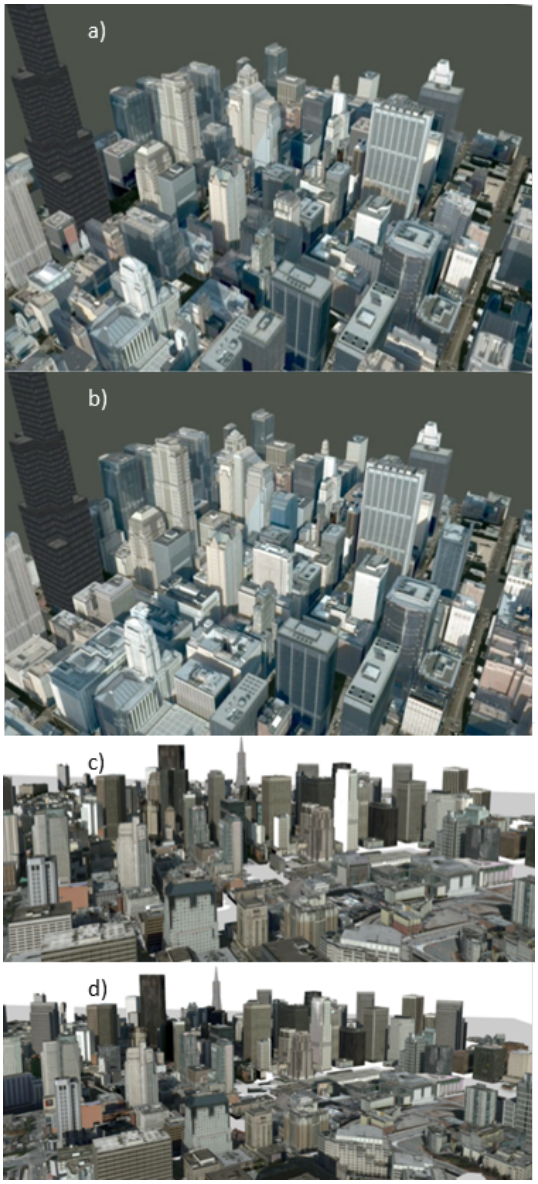

Figure 8. **City Views.** We show a-c) original Chicago and San Francisco, and b-d) de-instanced and proceduralized versions about 1/2 to 1/4 less component types

The view from our model is shown in Figure 10d while Figures 10e-g show the other 3 potential views (also using Google Street View). The demonstrative prototype does not determine camera orientation only its approximate position. The orientation of the camera, in all views, is done manually. The illumination effects are also not considered yet.

VII. Conclusions and Future Work

We have presented a framework to convert an existing large collection of unstructured 3D urban data of a city into a compact procedural representation. Our system has been applied to several large scenes and we have shown a variety of analysis and visual results in the paper, supplemental material, and in our video. As

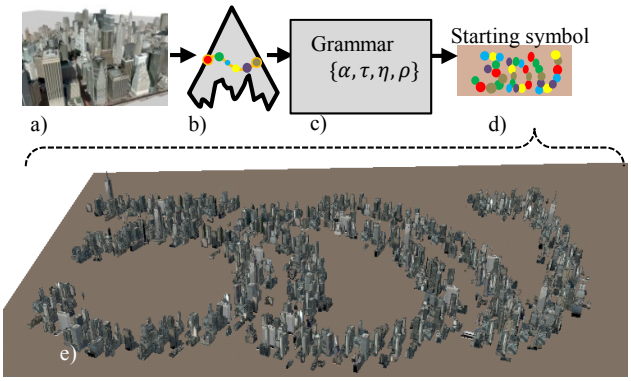

Figure 9. **User-Controlled Synthesis.** a) Input model is converted into b) a parse tree and c) a grammar is extracted. The user specifies d) a starting symbol, and our method best fills it with the grammar and generates e) new content keeping the style and neighborhoods, showing the expressive power of the output grammar.

future work, we would like to subdivide a component having two or more window distributions/styles (without any noticeable difference in the component's exterior contour), to apply our method to incomplete geometry and to include methods to decompose facades into their core smaller components

Acknowledgement

This research is partially supported by NSF grants 1250232, 0964302, and 1302172 and a Google Research grant. We sincerely thank the reviewers for their insightful feedback.

References

[1] D. Aliaga, P. Rosen, and D. Bekins. *Style grammars for interactive visualization of architecture*, IEEE Trans. on Vis. and Comp. Graphics, 13(4), 786797, 2007.

[2] G. Baatz, K. Koeser, D. Chen, R. Grzeszczuk, and M. Pollefeys. *Handling Urban Location Recognition as a 2D Homothetic Problem*, Europ. Conf. on Comp. Vision, 2010.

[3] F. Bao, D.-M. Yan, N. J. Mitra, and P. Wonka, *Generating and exploring good building layouts*, ACM Trans. Graph. 32, 4, Article 122 (July 2013), 10 pages, 2013.

[4] M. Blum. *On the size of machines*, Information and Control, 11(3), 257, 1967

[5] M. Bokeloh, M. Wand, And H.-P. Seidel. *A connection between partial symmetry and inverse procedural modeling*, ACM Trans. on Graphics., 29(4), 2010.

[6] Doersch, S. Singh, A. Gupta, J. Sivic, and A. Efros. *What makes Paris look like Paris?*, ACM Trans. On Graphics 31, 4, Article 101 (July 2012), 9 pages, 2012.

[7] M. Girvan, And M. E. J. Newman. *Community structure in social and biological networks*, Natl. Acad. Sci. USA, 2002.

[8] S. Haegler, P. Wonka, S. Arisona, L. Van Gool, And P. Mueller, *Grammar-based Encoding of Facades*, Computer Graphics Forum, 29(4), 1479 1487. 2010.

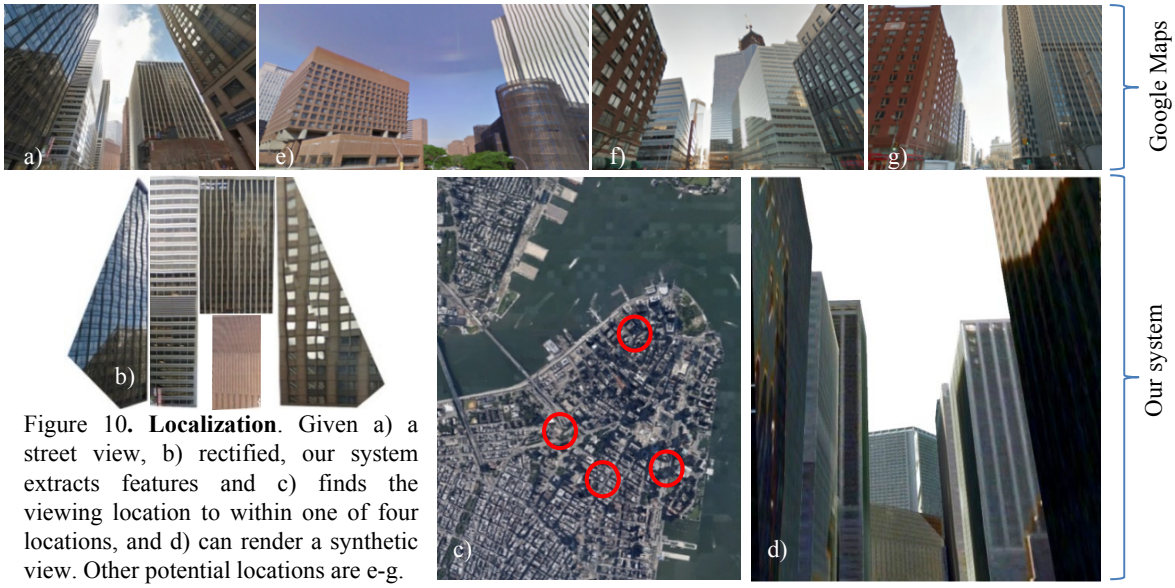

Figure 10. **Localization**. Given a) a street view, b) rectified, our system extracts features and c) finds the viewing location to within one of four locations, and d) can render a synthetic view. Other potential locations are e-g.

[9] B. Hohmann, U. Krispel, S. Havemann, And D. Fellner, *CityFit: High-quality urban reconstructions by fitting shape grammars to images and derived textured point clouds*, ISPRS Workshop, 8 pages, 2009.

[10] D. King, Dlib-ml: *A Machine Learning Toolkit*, Journal of Machine Learning Research, 10, 1755-1758, 2009.

[11] Z. Kuang, B. Chan, Y. Yu, And W. Wang. *A Compact Random-Access Representation for Urban Modeling and Rendering*, ACM Trans. on Graphics, 32(6), 172, 2013.

[12] F. Lafarge, And C. Mallet. *Building large scale urban environments from unstructured point data*, IEEE Intl Conference on Computer Vision, 1068-1075, 2011.

[13] H. Lin, J. Gao, Y. Zho., G. Lu, M. Ye, C. Zhang, L. Liu, And R. Yang. *Semantic decomposition and reconstruction of residential scenes from LiDAR data*. ACM Trans. on Graphics, 32(4), 66, 2013.

[14] M. Lipp, D. Scherzer, P. Wonka, And M. Wimmer. *Interactive Modeling of City Layouts using Layers of Procedural Content*, Computer Graphics Forum (Eu-rographics), 30(2), 345-354, 2011.

[15] P. Mueller, P. Wonka, S. Haegler, A. Ulmer, L. Van Gool. *Procedural modeling of buildings*, ACM Trans. on Graphics, 25(3), 614-623, 2006.

[16] P. Mueller, G. Zeng, P. Wonka, And L. Van Gool. *Image-based Procedural Modeling of Buildings*, ACM Trans. on Graphics, 24(3), 85, 2007.

[17] P. Musialski, P. Wonka, D. Aliaga, M. Wimmer, L. Van Gool, And W. Purgathofer. *A Survey of Urban Reconstruction*, Computer Graphics Forum, 32(6), 146, 2013.

[18] Y. I. Parish, And P. Mller. *Procedural modeling of cities*, ACM SIGGRAPH, 301-308, .2001.

[19] R. M. Smelik, T. Tutenel, R. Bidarra, And B. Benes. *A Survey on Procedural Modeling for Virtual Worlds*, Computer Graphics Forum, 33(6); 31-50, 2014.

[20] M. Steinberger, M. Kenzel, B. Kainz, P. Wonka, and D. Schmalstieg. *Combined Derivation and Rendering of Shape-Grammars on the GPU*, Computer Graphics Forum (Eurographics), 10 pages, 2014.

[21] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Mech, And V. Koltun. *Metropolis procedural modeling*. ACM Trans. on Graphics, 30(2), 11, 2010.

[22] A. Toshev, P. Mordohai, And B. Taskar. *Detecting and parsing architecture at city scale from range data*, IEEE Computer Vision and Pattern Recognition, 398-405. 2010.

[23] C. Vanegas, D. Aliaga, P. Mueller, P. Waddell, B. Watson, And P. Wonka. *Modeling the Appearance and Behavior of Urban Spaces*, Eurographics, STAR 17 pages, 2009.

[24] C. Vanegas, D. Aliaga, And B. Benes, *Building reconstruction using Manhattan-world Grammars*, IEEE Computer Vision and Pattern Recognition, 358-365, 2010.

[25] C. Vanegas, I. Garcia-Dorado, D. Aliaga, B. Benes And P. Waddell. *Inverse design of urban procedural models*. ACM Trans. Graph., 31(6), 168, 2012.

[26] P. Wonka, M. Wimmer, And F. Sillion, *Instant Architecture*, ACM Trans. on Graphics, 22(3), 669-677, 2003.

[27] J. Xiao, T. Fang, P. Tan, P. Zhao, E. Ofek, And L. Quan. *Image-based faade modeling*, ACM Trans. on Graphics, 27(5), 2008.

[28] Q. Zhou, And U. Neumann. *2.5D Dual Contouring: A Robust Approach to Creating Building Models from Aerial LiDAR Point Clouds*, Europ. Conf. on Comp. Vision, 2010.