Interactive Sketching of Urban Procedural Models

Gen Nishida[1] Ignacio Garcia-Dorado[1] Daniel G. Aliaga[1] Bedrich Benes[1] Adrien Bousseau[2]
[1]Purdue University [2]Inria

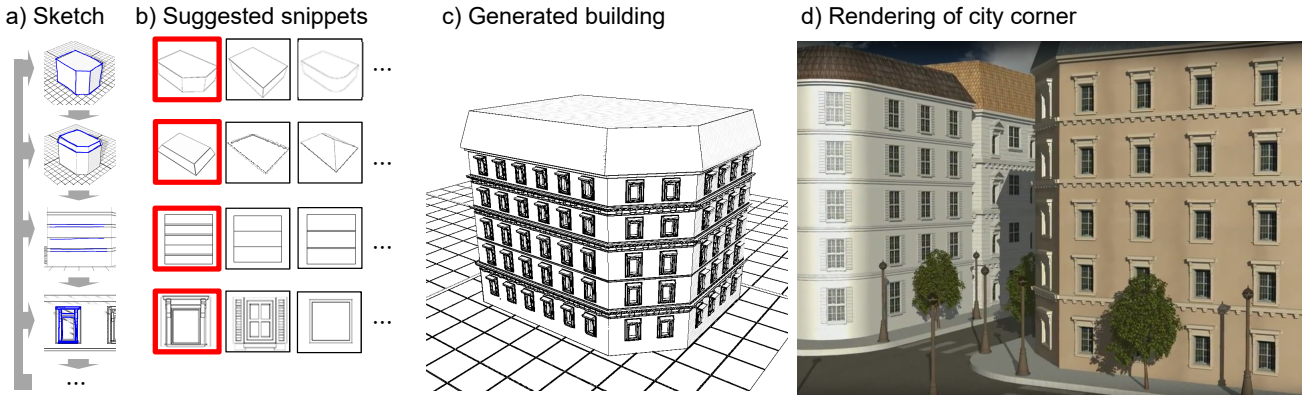

Figure 1: ***Procedural model sketching.*** *Our interactive sketching tool allows the user to quickly and easily author procedural 3D building models. The user only performs interactive sketching and our system automatically generates the procedural model and its parameters yielding the sketched shape. Our system partitions the design process of a 3D model into sketching various object parts that together form the overall model. a) The user sketches a few strokes of the current object type to specify the desired shape. For instance, (top row) the user sketches the shape of the building mass. b) Using a deep-learning based method, the system finds which pre-defined grammar snippets yield visually similar shapes and chooses the best one by default (red square). c) The selected snippets and their parameter values are merged into a single grammar that represents the entire building. d) An offline rendering of variations of the procedural building model generated automatically by changing the snippets and/or parameters that encode them.*

Abstract

3D modeling remains a notoriously difficult task for novices despite significant research effort to provide intuitive and automated systems. We tackle this problem by combining the strengths of two popular domains: sketch-based modeling and procedural modeling. On the one hand, sketch-based modeling exploits our ability to draw but requires detailed, unambiguous drawings to achieve complex models. On the other hand, procedural modeling automates the creation of precise and detailed geometry but requires the tedious definition and parameterization of procedural models. Our system uses a collection of simple procedural grammars, called snippets, as building blocks to turn sketches into realistic 3D models. We use a machine learning approach to solve the inverse problem of finding the procedural model that best explains a user sketch. We use non-photorealistic rendering to generate artificial data for training convolutional neural networks capable of quickly recognizing the procedural rule intended by a sketch and estimating its parameters. We integrate our algorithm in a coarse-to-fine urban modeling system that allows users to create rich buildings by successively sketching the building mass, roof, facades, windows, and ornaments. A user study shows that by using our approach non-expert users can generate complex buildings in just a few minutes.

Keywords: Inverse Procedural Modeling, Sketching, Machine Learning

Concepts: •**Computing methodologies** → **Computer graphics;** *Shape modeling;* •**Theory of computation** → *Computational geometry;*

1 Introduction

Object design and modeling is a crucial skill in various areas of entertainment, science, and engineering. Designers wish to create custom objects, seek variations of existing models, and explore new uncharted designs. However, designing geometric models is notoriously hard and unintuitive. It often requires extensive learning that makes it difficult especially for novices. Although humans can quickly draw and sketch in 2D and we are well-trained to do those tasks since childhood, the transition to 3D is difficult for automated processes.

Prior work has addressed the modeling problem from various directions. Sketch-based modeling [Olsen et al. 2009] exploits human intuition and experience in drawing objects. Nevertheless, the quality of the 3D model depends on the sketching skills of the user, the details added to the drawing, and the ability to resolve inherent ambiguities of the sketching process. Another important and popular direction is procedural modeling [Smelik et al. 2010; Vanegas et al. 2010] that has been successfully applied to the creation of detailed and complex cities [Parish and Müller 2001], realistic and growing vegetation [Prusinkiewicz and Lindenmayer 2012], and other objects [Ritchie et al. 2015]. However, procedural modeling is difficult to control and thus hard to use as an exploratory design tool making it accessible only to experts.

Our approach is to use machine learning to seamlessly merge procedural modeling and interactive sketching, thus enabling an interactive design process leveraging both the intuitiveness, freedom, and flexibility of sketching and the precision, exactness, and detail amplification [Smith 1984] of procedural modeling. The user does not need to specify tedious procedural rules or rule parameters; instead, they are recognized from the sketch automatically thus enabling untrained users to quickly create complex procedural models.

Consider a user who begins sketching a 3D model in 2D by using a mouse or a digital pen on a tablet. Inspired by the way artists draw the general structure before adding details [Loomis and Ross 2014], our approach allows the user to progressively add details by sketching various object types (e.g., building mass, roof, windows) from coarse to fine. Each object type is supported by multiple pre-defined *grammar snippets*. The partitioning into object types is not explicitly predetermined by our approach; rather it results from the types associated with each of the provided snippets. The user can select the next object type to sketch using gestures or via GUI buttons. For each object type, the user begins sketching a few strokes until a match with a snippet is found.

In order to achieve interactivity, we use convolutional neural networks (CNNs) to identify the grammar snippet and snippet parameter values that best explain the current user sketch. The CNNs are trained by generating many instances of each snippet and by sampling its parameters and rendering them in a sketch style. We employ a cascade of networks, where a first network recognizes the snippet, while a second snippet-specific network estimates its parameters. At run-time, we feed the CNNs with the user sketch to almost instantaneously recover the intended snippet and parameter values.

The final output is a single *generated grammar* composed of an assembly of grammar snippets collectively representing the sketched 3D model. The user can render the procedurally-generated model at any moment in one of a variety of rendering styles (e.g., sketch-style rendering, ambient occlusion, or shaded and textured rendering). Further, the result is not limited to a single instantiation of a model. Rather, we exploit the procedural nature of our system to create variations that are inspired by the originally-sketched object.

We demonstrate our approach in the context of urban modeling. Nonetheless, our system is applicable to other modeling domains as well. Our approach does not make any particular assumptions about the 3D model except assuming it can be decomposed into a set of object types and a set of pre-defined snippets per type.

Our main contributions include

- combining sketching and procedural modeling for designing complex 3D models without having to explicitly sketch all details and without having to write the underlying procedural grammars, and
- using a cascade of CNNs that enables interactive classification of partial sketches to one of multiple possible grammar snippets and estimating their parameters.

2 Previous work

Our work relates to research in sketch-based modeling, example-based modeling, and forward and inverse procedural modeling. Sketching attempts to make 3D modeling as direct and intuitive as drawing [Olsen et al. 2009]. However, recovering a 3D model from a 2D drawing is fundamentally ill-posed because strokes do not provide depth information. Early approaches made this problem well-posed by assuming that the lines in the drawing obey specific geometric constraints in 3D, or by using a well-defined set of gestures to specify one of a set of primitive shapes (e.g., [Zeleznik et al. 1996]). For smooth shapes, the lines can be assumed to denote contours and silhouettes [Igarashi et al. 1999], while for polyhedrons geometric relationships such as parallelism, orthogonality and planarity can be detected and imposed [Lipson and Shpitalni 1996], or even learned from line-renderings of 3D models [Lipson and Shpitalni 2000]. Unfortunately, such assumptions only hold for a limited family of shapes. Recent methods allow the creation of complex freeform shapes by exploiting geometric constraints present in professional design drawings, such as polyhedral scaffolds [Schmidt et al. 2009] and cross-section lines [Xu et al. 2014]. Interactive systems also rely on axis-aligned planes and other transient surfaces such as 3D canvases to support 3D strokes [Bae et al. 2008; Zheng et al. 2016]. Since all the above methods derive constraints from the drawn lines, they require relatively accurate drawings as input. In addition, these methods only reconstruct what is drawn, which means that users have to draw very detailed sketches to obtain detailed 3D models. In contrast, our system relies on procedural grammars as a strong prior to regularize inaccurate and ambiguous sketches as well as to suggest intricate details from a handful of lines.

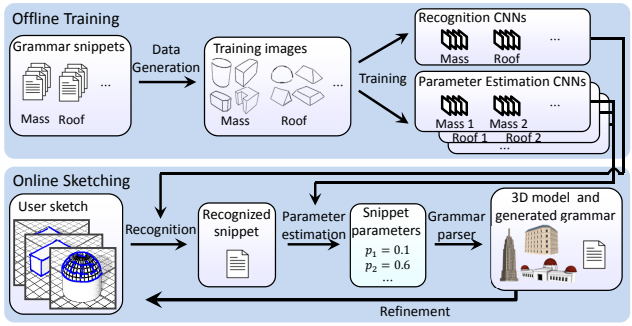

Figure 2: ***System Pipeline.*** *Our system consists of offline training and interactive online sketching. During training, a set of pre-defined grammar snippets are used to generate a large number of training images. Training and validation datasets are used to train the CNNs. During runtime, the pre-trained CNNs are used to find the best snippets and parameter values that match the current sketch. Then, the grammar parser generates an instance of the entire 3D model.*

Our approach is closer to example-based methods, which reconstruct drawings by fitting parts from a database of 3D models. The database can be composed of generic parameterized shapes such as cubes, cones and cylinders [Xue et al. 2012; Shtof et al. 2013], or detailed 3D models from a particular domain, such as furniture and vehicles [Xie et al. 2013] or building parts [Chen et al. 2008b]. Similarly to our offline pre-training, Xie et al. [2013] and Eitz et al. [2012] pre-compute non-photorealistic renderings of the 3D models to perform shape retrieval during sketching. The expressiveness of these systems greatly depends on the size of the 3D model database. On the one hand, while generic shapes can be represented compactly with a few parameters, many basic shapes need to be combined to create complex models. On the other hand, large databases of detailed 3D models are expensive to collect and store, and the resulting model is not editable. We offer a middle-ground between these two extremes by matching drawings to procedural grammars, which are compact and editable. Moreover, we do not match a complete model but progressively match parts which are then combined into the final model. This has the additional benefit of needing a relatively small database yet our approach is capable of generating a large variety of detailed 3D models.

Procedural modeling offers an effective way of generating complex, parameterized 3D models [Wonka et al. 2003; Müller et al. 2006; Smelik et al. 2014]. Among procedural models, grammar-based models are commonly used in urban modeling and vegetation. Procedural systems can quickly generate many 3D models with wide variation by either changing the grammar or by varying its attributes. However, creating a grammar requires programming expertise and domain knowledge to be able to write compact rules, and setting the parameters of a grammar is non-trivial be-

cause of the intricate relationship between the procedural parameters and the output. To address this issue, Lipp et al. [2008] introduce a visual editor akin to standard 3D modeling software, allowing direct editing of architectural models by selecting and dragging procedural components. Several sketch-based systems have also been proposed for specific domains, such as trees [Ijiri et al. 2006], terrains [Smelik et al. 2010], and roads [Applegate et al. 2012; Chen et al. 2008a]. However, these methods rely on application-dependent heuristics rather than on a generic sketch-recognition algorithm like ours.

Inverse procedural modeling estimates the parameters of procedural models by minimizing an objective function defined by the user input and the parameter values. Monte Carlo Markov Chain (MCMC) [Talton et al. 2011; Vanegas et al. 2012; Stava et al. 2014] and Sequential Monte Carlo (SMC) [Ritchie et al. 2015] are so far the most promising solutions to explore the large parameter space and find near-optimal parameter values. However, these iterative sampling algorithms require many steps to converge, preventing their use in an interactive context. Recently, Emilien et al. [2015] learned localized procedural models from examples and reused them for sketching virtual worlds. However, their approach is suited for stochastic models and fails to represent structure and its repetition. We exploit recent advances in machine learning to perform inverse procedural modeling at runtime without the cost of iterative optimization. This approach is inspired by recent work on *informed samplers* that rely on discriminative inference to accelerate MCMC for computer vision tasks [Jampani et al. 2015].

Our machine learning approach achieves high accuracy in a fraction of the time required by standard MCMC, opening the door to interactive application of inverse procedural modeling. Our approach is enabled by the strength of CNNs. CNN is a type of artificial neural network and has been applied to many areas, such as recognizing hand-written characters [Lecun et al. 1998], image classification [Krizhevsky et al. 2012], single-image depth estimation [Eigen et al. 2014], and cross-domain image matching [Bell and Bala 2015]. CNN has also been applied to 3D shape retrieval [Wang et al. 2015], which significantly improves the accuracy compared to a feature-based approach [Eitz et al. 2012].

3 Overview

We provide an overview of our offline training and online sketching phases (Figure 2). The input to our system is a collection of pre-defined grammar snippets (Figure 3). During an interactive design session, the user draws partial or complete sketches which are automatically matched and completed with procedurally-generated content. The final output is a complete 3D model and the corresponding generated grammar.

Snippets: To provide design flexibility, our system supports multiple grammar snippets for each object type of the sketching process. One usual partitioning of buildings into object types is to provide snippets for different building shapes (i.e., building masses), roofs, window styles, ledges, and other facade ornaments (Figure 3). These snippets are arbitrarily combined during sketching thus enabling the construction of a wide variety of building models.

Training: As a preprocess, our system trains a collection of CNNs. The snippets are fed to a data generator which then generates a large number of variations by randomly sampling the snippet's parameter values. The rendered images for each variation are then used to train two types of CNNs: one type for snippet recognition and the other type for snippet-specific parameter estimation.

Sketching: Online sketching is done via a mouse or 3D pen. The designer may interactively draw a new sketch on an empty canvas or use a *lasso* to select a region (e.g., a face) where to anchor the next sketched part. For a new sketch, the anchored region corresponds to the ground plane and the object type defaults to an agreed upon first object type. In all other cases, the object type depends on the label specified in the snippet for the selected anchor region. Nonetheless, the object type to sketch can be explicitly expressed by the designer. As the user sketches, the system uses the snippet recognition CNN to match the sketch to an instance of a particular snippet type and then uses that snippet's parameter estimation CNN to recover its parameter values. The procedurally-generated model is visualized in one of a variety of rendering styles. At any time the generated grammar of the entire 3D model can be output.

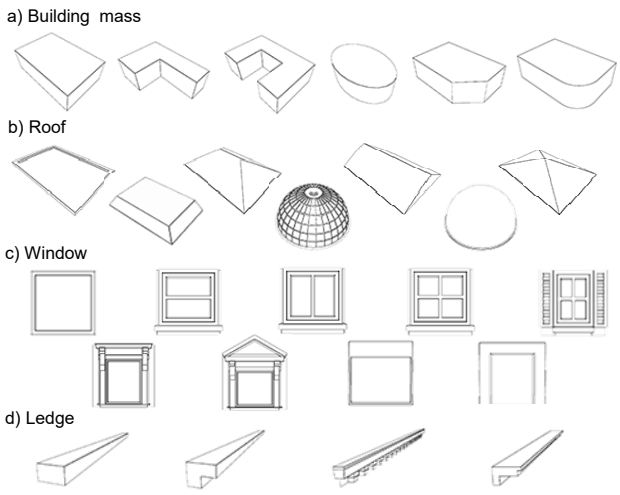

Figure 3: ***Example Snippets and Object Types.*** *As opposed to a standard 3D model, each snippet contains parameters defining a range of potentially generated geometry. Further, each snippet is associated with an object type. The set of shapes in each row are example outputs generated by randomly selecting snippets and parameter values.*

4 Representation

Without loss of generality, our system implements the grammar snippets and the generated grammar as XML-based split grammars. While our current implementation makes use of split grammars (i.e., as used by [Wonka et al. 2003] and [Müller et al. 2006]), the approach does not depend on the exact grammar type.

4.1 Split Grammar

Our system uses split grammars to represent each snippet as well as the entire model. The i'th snippet is defined as $G_i = \{\alpha, \tau, \eta, \rho\}$ where α is the axiom, τ is the set of terminal symbols, η is a set of non-terminals, and ρ is a set of rewriting rules using the terminals and non-terminals. Moreover, each snippet is labeled with an object type and the axiom α is named *Start*. The snippet may also contain one or more object-type labeled non-terminals (Figure 4).

4.2 Snippet Combination

Every time the user draws new strokes, a best fitting snippet is recognized and added to the grammar representing the 3D model. At this moment, the non-terminal of the current anchor is replaced by the recognized snippet and its automatically determined parameters. For example, when the name of the selected anchor non-

```
<!-- grammar for a building mass -->
<param building_height="20"/>
<rule name="Start">
   <extrude height="building_height"/>
   <comp>
      <top name="TopFace"/>   <!-- roof -->
      <side name="Façade"/>   <!-- façade -->
      <bottom name="Base"/>
   </comp>
</rule>
```

Figure 4: ***Snippet Grammar.*** *Each snippet has an axiom named Start and contains one or more non-terminals that can be rewritten by procedural rules (i.e., replaced by other snippets). In this example, TopFace will be used by a roof-type snippet, and Facade will be used by a facade-type snippet.*

```
<!-- grammar for a building mass -->
<param building_height="20"/>
<rule name="Start">
   <extrude height="building_height"/>
   <comp>
      <top name="TopFace"/>
      <side name="Façade"/>
      <bottom name="Base"/>
   </comp>
</rule>

<!-- grammar for a roof -->
<param roof_slope="50"/>
<rule name="TopFace">
                <!-- originally was "Start" -->
   <roofGable slope="roof_slope"/>
</rule>
```

Figure 5: ***Grammar Combination.*** *When a snippet is recognized, the name of its starting axiom is modified to the name of the non-terminal to which the snippet is anchored. In this way, multiple snippets can be linked into one generated grammar for the entire 3D model.*

terminal is *TopFace* and the user sketches a grammar for a roof, the axiom of a copy of the snippet grammar is modified to *TopFace* (Figure 5). Also, the object type of this axiom is set accordingly so that the system knows that the added snippet is for a roof in the generated grammar. Afterwards, the derivation tree [Müller et al. 2006] is regenerated producing a new instance of the 3D model.

5 Training

Our system trains one CNN per object type to recognize snippets and one CNN per pre-defined snippet to estimate parameters. The training process is performed once as a preprocess and during runtime the trained CNNs are loaded and used.

5.1 Convolutional Neural Networks

CNNs can be used for both recognition/classification problems [Krizhevsky et al. 2012; Abdel-Hamid et al. 2014], in which the output is the probability distribution of discrete values for each category, and regression problems [Pfister et al. 2015], in which the output are continuous valued parameters. Even though it may require a one-time several hours preprocess, once the network is trained, the aforementioned problems can be solved almost instantaneously. Thus, it is suitable for interactive usage (Figure 6).

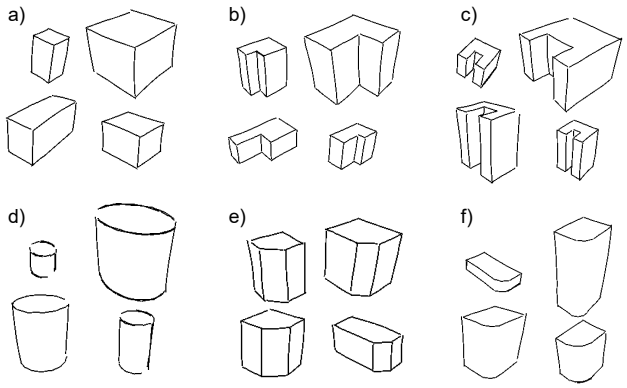

Figure 6: ***CNN Training.*** *An example set of objects generated by our building mass snippet for CNN training. The objects were rendered as sketches to optimize the CNN for human-drawn strokes. For other object types, we generated the training images in a similar manner.*

Table 1: ***Recognition CNNs.*** *We compared the accuracy after 20,000 iterations of recognition CNN training using pretrained weights as initial weights and training from scratch. The former training approach improves the accuracy on our dataset.*

	Training using the pretrained weights	Training from scratch
Building mass	0.99	0.92
Roof	0.98	0.90
Window	0.98	0.87
Ledge	0.95	0.86

5.2 Training Process

For the recognition CNNs, we use the BVLC AlexNet architecture [Krizhevsky et al. 2012] starting from the weights obtained from the Caffe Model Zoo [Jia et al. 2014]. The initial weights are obtained by training with hundreds of thousands of images so as to learn to discriminate the features [Russakovsky et al. 2015]. We fine-tune the networks by using our training images so as to optimize the classifiers to achieve high accuracy on our recognition application (Table 1).

For each parameter estimation CNN, we use a modified architecture with three convolutional layers followed by two fully connected layers. Finding the best network architecture is still an open problem especially for regression problems. In practice, deeper convolutional networks can extract more features but are more difficult to train. We use three convolutional layers that achieved satisfactory results. Since there are no pre-trained weights for this network, we train CNNs from scratch, which requires many more iterations to converge (Figure 7). We use normalized parameter values as input for the CNNs. The j'th parameter has a value $v_j \in [v_{min}, v_{max}]$ and the normalized value is computed based on this range by $\frac{v_j - v_{min}}{v_{max} - v_{min}}$.

Note that the output of the first CNN type is a probability distribution of the snippet types, whereas the outputs of the second CNN types are normalized parameter values.

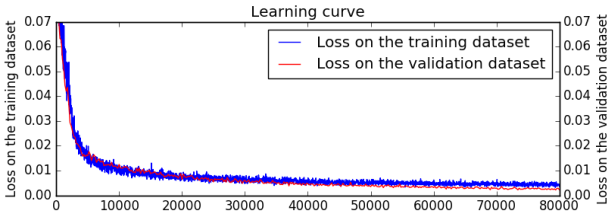

Figure 7: ***Parameter Estimation CNN.*** *Training for parameter estimation requires a large number of iterations to converge because no initial pre-trained weights are used.*

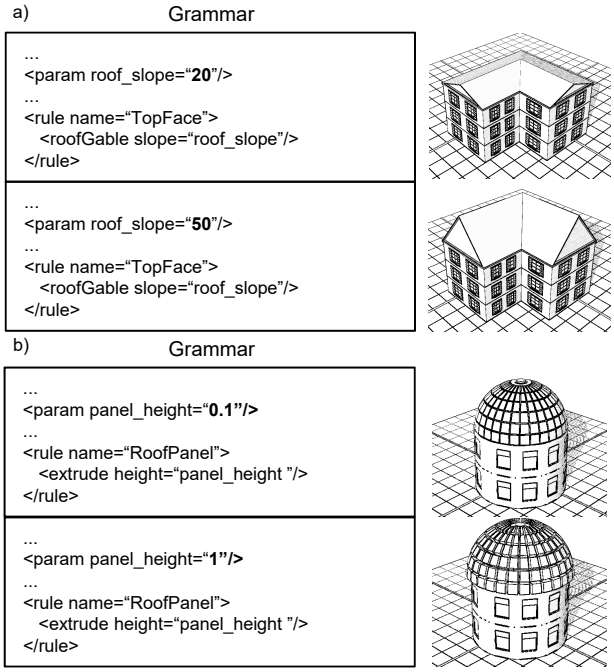

Figure 8: ***Grammar Editing.*** *We also enable the user to directly change the parameter values, such as a) the slope of the roof, and b) the height of the extrusion of roof panels.*

5.3 Data Generation

Our preprocessing tool generates a large number of training images (over 40,000 images for the first object type to sketch, building mass, and around 7,000 images for other object types) for each snippet by randomly changing its parameter values. In an early version of our system, we trained the network with multiple viewpoints. However, we found in practice that most people use a predefined vantage point, such as a *three quarter view*. Thus, in order to reduce the complexity of snippet recognition and parameter estimation, our system fixes the viewing orientation and position of the sketched objects. We consider the XY plane to be the ground plane in the world coordinate system, θ be the rotation angle around X axis, and ϕ be the rotation angle around Z axis. Our system enables specifying the viewing orientation for each object type (e.g., for building mass and roof, $\theta = 30°$ and $\phi = 45°$; for ledge, $\theta = 0°$, $\phi = \rho - 72°$; for other object types $\theta = 0°$, $\phi = \rho$, where ρ is the Z rotation such that the anchor plane is perpendicular to the camera view direction). The viewing position is fixed such that the projection of the center of the bounding box of the anchored region is

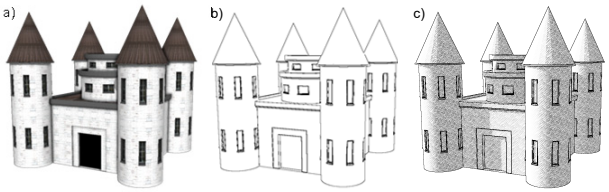

Figure 9: ***Rendering Styles.*** *Our tool provides several rendering options, a) screen space ambient occlusion (SSAO), b) sketch lines, c) sketch lines + hatching texture.*

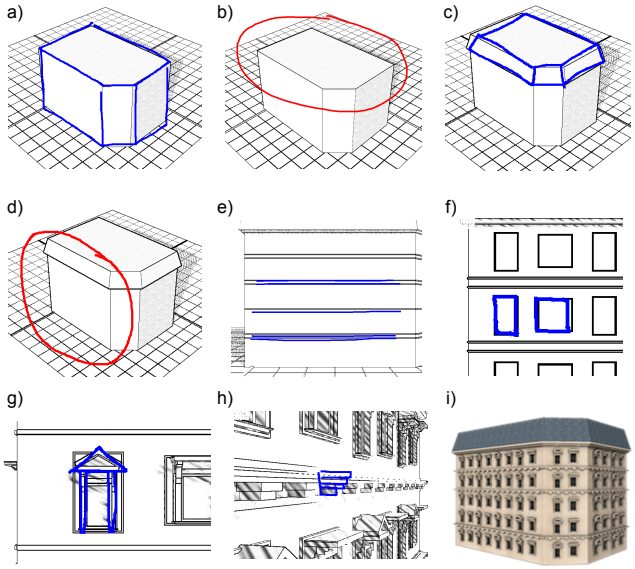

Figure 10: ***Example Sketching Sequence.*** *Our system allows the user to intuitively create a 3D model. We show snapshots of the sketching sequence of a building. See main text and video for more details.*

located at the center of the drawing canvas. Note that for the agreed-upon first object type to sketch (e.g., building masses), the system considers the ground plane as the anchor. Since the anchor can be any shape at any location on the ground plane, the preprocessing tool for training changes the originating position of the snippet on the ground plane to a sampling of potential positions.

6 Sketching

At run-time, our system uses the pre-trained CNNs to recognize which snippet is being partially sketched. Moreover, additional tools are provided to support object repetition, to generate a variety of similar 3D models, to render using one of several styles, and to provide standard move/resize/copy/paste/delete/undo editing operations.

6.1 Recognition and Parameter Estimation

During sketching, the snippet with the highest probability is selected, and then the system performs snippet parameter estimation. When the user selects an anchor-region for the next sketch, the system automatically alters the viewpoint (Section 5.3) so that the user sketch is best aligned to the training images. Our system resizes the

user sketch to the same resolution as the training images and the recognition CNN produces a normalized probability distribution of the snippet types.

Afterwards, the system resizes the user sketch to the training image resolution for the parameter estimation CNN of the recognized snippet type, and the parameter estimation CNN outputs the normalized parameter values. As an extra step of refinement, the system also uses a few iterations (i.e., 10 in our current implementation) of a MCMC engine to further improve the parameter values. We use a single thread and Metropolis-Hastings algorithm to accept or reject randomly-generated state change proposals. Sketch-style rendering is used to generate a 2D image I_{prop}, and state change proposal score s_{prop} is computed based on the user sketch I_{sketch} by

$$s_{prop} = \exp\left(-\left\|D(I_{prop}) - D(I_{sketch})\right\|_F\right) \quad (1)$$

where $D(*)$ denotes distance transform and $\|*\|_F$ denotes Frobenius norm.

Once the parameter values are estimated, the system inserts or replaces the new snippet into the generated grammar (Section 4.2) and executes the derivation to update the 3D model.

6.2 Repetition Management

In some cases, the user may want to apply an object type repetitively (e.g., a row of windows, stack of floors). Instead of sketching the object many times or making extensive use of copy and paste, we provide a quick mechanism to subdivide the anchor region, sketch an object type, and generate repetitions. Our tool infers the intended horizontal or vertical 1D repetitive application from just a few user-drawn line segments. Two examples of currently supported patterns are 1) $(A)^*$ that evenly splits the anchor into multiple anchors (or faces) of the same shape and 2) $(AB)^*A$ that splits the anchor into shapes A with shape B in between (e.g., shape B might be a thin separator between shape A's). Both A and B correspond to non-terminals of different object type labels than the anchor region. Note that our system does not require any additional action from the user to select the repetition mode; it is automatically selected when drawing facades and floors.

6.3 Design Variability

Since our end result is a grammar, we can exploit procedural modeling not just to create one model but to create a wide variation of buildings that are inspired by the original sketch. This is akin to the notion of concept drawing used in sketching to explore a design space: an artist draws subtle variations to an initial sketch and then potentially a more attractive option than the initial sketch is discovered [Eissen and Steur 2009]. We support two approaches to provide variability: manual swap of snippets (for complete control; Figure 8) and stochastic snippet/parameter replacement. For the latter option, we use a similarity value between snippets (provided as input) to define the probability of the swaps. Moreover, we randomly explore parameter value changes to create more variability. We show results in Section 7 and Figure 16.

6.4 Rendering and Editing Tools

Our system provides several rendering and editing tools. The 3D model can be rendered using sketch style, screen space ambient occlusion (SSAO), pen-and-ink style, or shaded texture-mapping (Figure 9). In addition, our framework supports selecting, moving, resizing, deleting, copying and pasting subsets of the entire model as well as an undo function. For example, when the user selects a building mass, control points appear on its top and side faces, and the user can resize it by moving the control points.

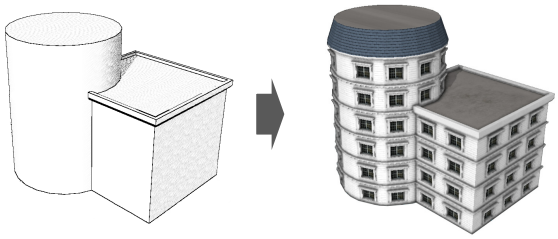

Figure 11: ***Final Model Previewing.*** *Our tool allows the user to see the final expected model at any moment even if the user has not selected a grammar for some stages. The system uses a default grammar for those stages and generates a complete building model. The user can optionally stop modeling if the result is satisfactory.*

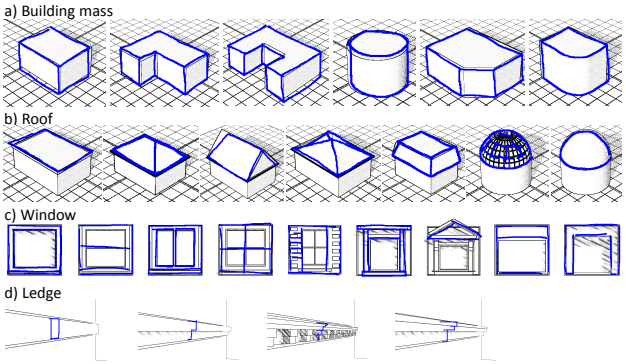

Figure 12: ***Objects Generated from Snippets.*** *Our approach uses CNN to find the most appropriate snippet and its parameter values. We show a few sketches for four different object types.*

7 Results and discussion

We implemented our system on an i7-based PC Workstation with 24GB of memory and NVidia GTX980 graphics card. Our tool was implemented in C++ using OpenGL/GLSL and supports mouse and tablet input. We used the Caffe library [Jia et al. 2014] as the framework to train the CNNs using the GPU and to recognize snippets and estimate snippet parameters during run-time.

The pre-trained CNNs and all the snippets are loaded at system startup. Our implementation includes 4 object types and 26 snippets, which results in 30 CNNs in total. We used images of resolution 256x256 for training the recognition CNNs and we used lower resolution 128x128 images to decrease the size of the input data and increase the batch size for the parameter estimation CNNs. We trained the recognition CNNs for 20,000 iterations using the pre-trained network as initial weights [Krizhevsky et al. 2012] and trained the parameter estimation CNNs for 80,000 iterations each. Every 10,000 iterations of training took around 30 minutes on our computer. Note that once trained, we can use these trained CNNs to recognize and estimate the parameters within 400ms.

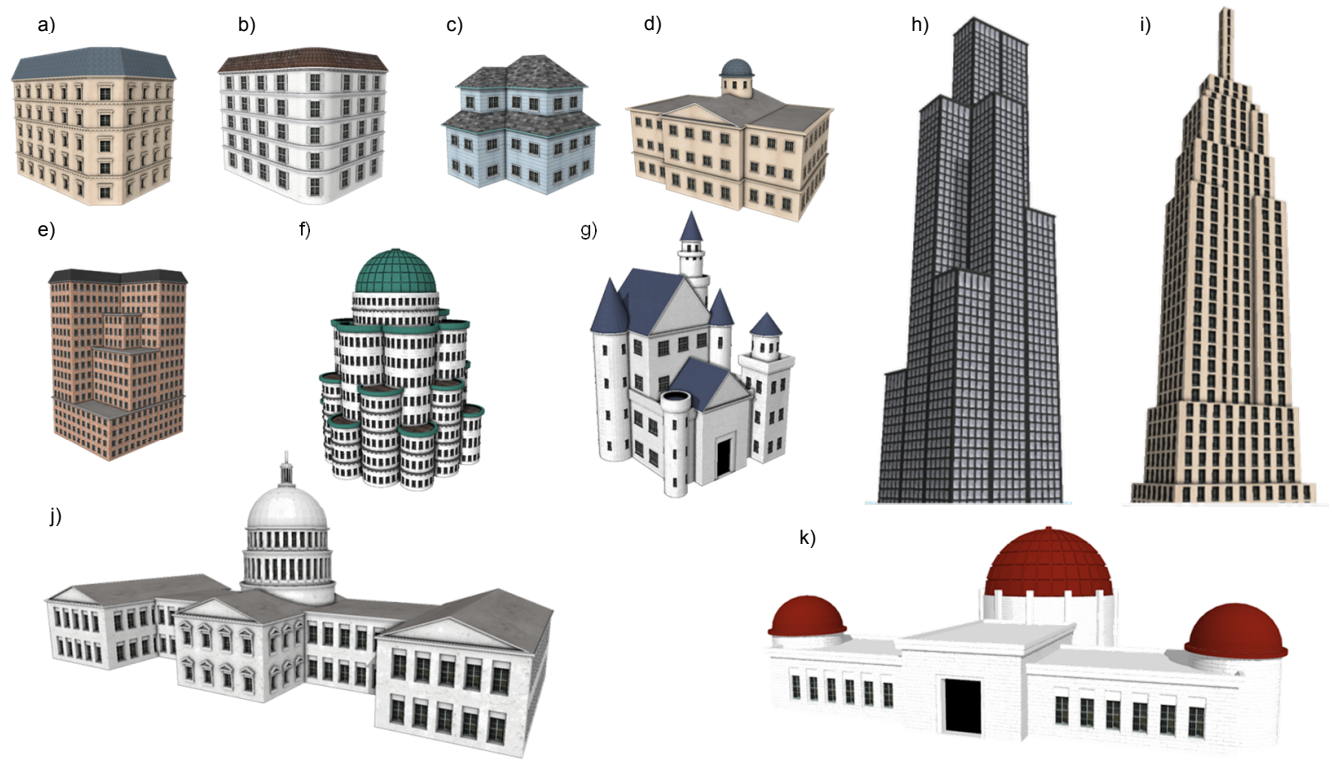

Figure 13: ***Example Buildings.*** *Some of the buildings generated by our tool: Paris-style buildings (a and b), complex shape of buildings (c, d, e, f, g, j, and k), and skyscrapers (h and i). The creation time was around 5 minutes for each building.*

7.1 Sketching Examples

Example sketches are shown in Figure 1, Figures 10-14 and the accompanying video. In Figure 10a, the user draws a sketch of a building mass. After selecting the top face of the generated building mass using the lasso tool (Figure 10b), the user draws a sketch of a roof (Figure 10c). The user selects a facade (Figure 10d) to subdivide it into a repeating arrangement of floors (Figure 10e). Similarly, the user selects a floor to subdivide it into walls and windows (Figure 10f). Then, the user selects one of the windows and sketches more details of the window (Figure 10g). Lastly the user sketches a ledge (Figure 10h). Further, the user can select color and texture that will be a part of the output grammar (Figure 10i). Also, the user can choose to preview a final expected model at any moment (Figure 11). In this building, the system has filled-in the details using the default snippets for the object types the user has not yet selected.

Figure 12 shows some examples of the object geometry generated by our tool for several exemplary object types. Notice how well our system can estimate parameter values. Figure 13 shows examples of buildings generated by our tool, including Paris-style buildings, skyscrapers, and complex shaped buildings. As a mean of evaluation, we used our system to reproduce a random set of buildings by using the 40 first "office" images from ImageNet and Flickr. We provide the full set of results as supplemental materials, and show a subset of representative ones in Figure 14. It took 5-15 minutes to create each of the buildings. Note that we managed to capture the overall shape and look of most buildings, although some details differ when they cannot be expressed by the snippets we've included in our current implementation (highlighted by red arrows in Figure 14).

Table 2: ***Training Performance and Computation Time.*** *The table shows the training performance and computation time for recognition and parameter estimation of the various models and snippets shown in this paper. The high accuracy and the relatively small root mean squared error (RMSE) show that our approach can find an appropriate grammar and parameter values precisely within a very short time.*

	Recognition		**Parameter estimation**	
	Accuracy	Time [ms]	RMSE	Time [ms]
Mass	0.99	25.1	0.03	305
Roof	0.98	25.2	0.06	318
Window	0.98	25.4	0.09	329
Ledge	0.95	25.6	0.08	304

7.2 Performance and Comparison

In this section, we show training performance, compute time, and system comparisons. Table 2 shows the training performance and computation time for recognition and parameter estimation of the various models and snippets shown in this paper. The total training accuracy averages to 97.5% and the recognition plus parameter estimation time is about 340ms.

We compared our approach to MCMC, the de-facto approach for inverse procedural modeling (Figure 15). In this experiment, our approach can precisely find the parameter values within 320ms, whereas MCMC cannot find a good estimate in such a short time. Also, note that we used only one grammar for MCMC in order to skip finding the best grammar among others. The "CNN only" approach already produces very accurate results, but we add a few MCMC steps in order to further improve parameter estima-

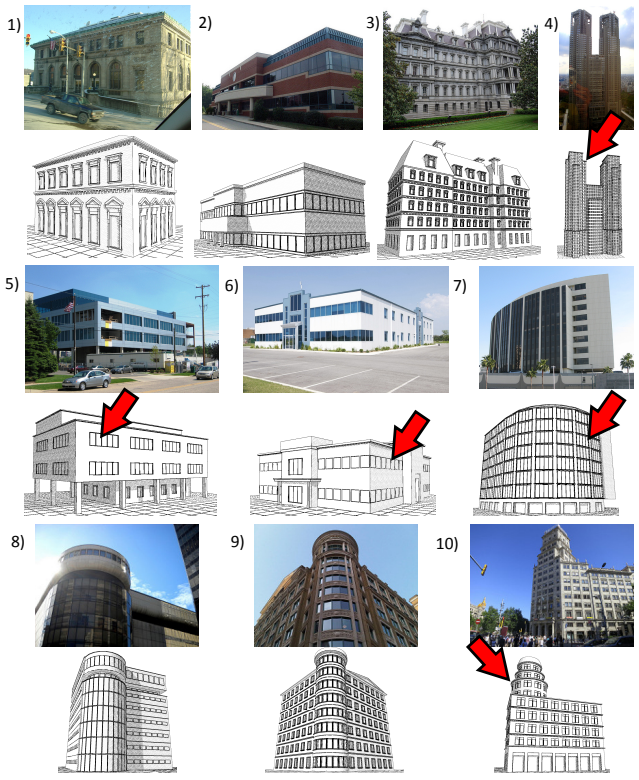

Figure 14: ***Expressivity.*** *We have created 40 buildings using the images from ImageNet and Flickr as inspiration. Only a subset of buildings is shown here. Please refer to the supplemental materials for all the created buildings and the source images. The red arrows highlight some building features and shapes not well supported by our currently implemented set of snippets, such as non-axis aligned shapes (4 and 10), certain window shapes (5), and irregular facade patterns (6 and 7).*
Photo Credits: 8) © Jennifer C., 9) © rjp, and 10) © Alper Çuğun.

tion while remaining fast enough for interactive use. While the improvement is subtle, it can be noticed on the second column in Figure 15, for example. Another naive approach for parameter estimation would be to generate a very large number of training images by changing parameter values and find the best match image given a user sketch. However, this approach can only choose the best one from a discrete set of options. In contrast, our approach can estimate continuous values.

To improve accuracy, we experimented with several rendering strategies for use in training: 1) sketch style rendering, 2) blurry rendering, and 3) incomplete renderings (e.g., some edges removed). For the first option, we implemented our custom rendering algorithm. In a first pass, it renders an image by using a standard line rendering algorithm. In a second pass, it detects edges by Edlines [Akinlar and Topal 2011], and replaces them by predefined polylines that imitate hand-drawn lines. For the second option, we use Gaussian filtering with σ =image size x 0.02. For the third option, we use a similar algorithm to the first one but remove half of the detected edges. Our experiments showed that sketch style rendering achieved the best accuracy (Table 3).

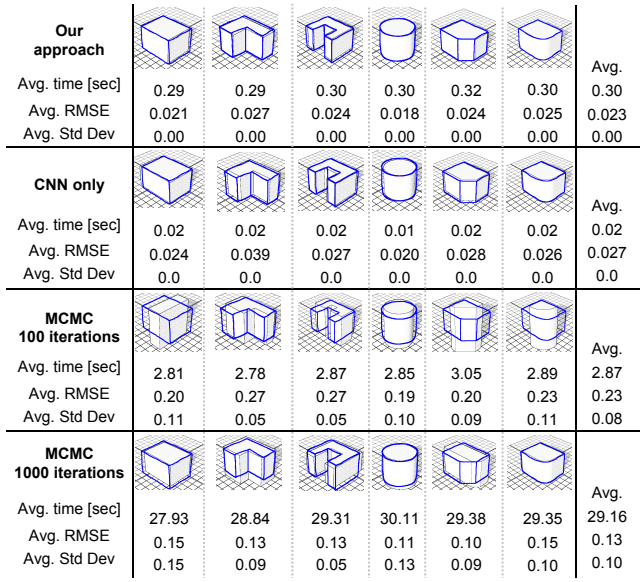

Figure 15: ***MCMC Comparison.*** *Our approach uses both CNN and 10 iterations of MCMC-based refinement to yield much better parameter estimation than MCMC alone. Even if the MCMC iterations are removed from our approach, CNN still yields much better accuracy as compared to MCMC and in a fraction of the time. The time, RMSE, and standard deviation for each grammar are the averaged values of ten executions.*

Table 3: ***Comparison of accuracy with different rendering options.*** *We experimented with several rendering options to generate training images. Our sketchy rendering achieved the best accuracy in recognizing parts of human-drawn sketches.*

	Building mass	Roof	Window	Ledge
Sketchy rendering	0.94	0.83	0.89	0.75
Blurred image	0.59	0.46	0.37	0.38
Missing edges	0.75	0.50	0.41	0.50

7.3 Model Variation

Since the output of our tool is always a consistent grammar, the user can easily produce a wide variety of geometry by changing snippets and snippet parameter values. Figure 16 shows some of the automatically generated buildings after the user completes the design sessions. A user-provided probability density function based on the visual similarity between grammar snippets is used to change grammars, while the parameter values are randomly selected. In this example, we manually defined a similarity value between different snippets of the same object type. Then, when so desired the designer can use this function to randomly select an alternative rule; upon selection the system then performs parameter estimation.

7.4 Robustness

Our system tolerates significant sketch inaccuracy and incompleteness and always generates a valid procedural model. Figure 17a shows examples of how our system is able to find the correct grammar snippet despite of inaccurate sketches and different sketching styles. Note that if the user draws a novel shape, i.e., a shape that has not been introduced in our grammar snippet set, the CNN will select the closest shape expressible by the available snippets (Figure 17b).

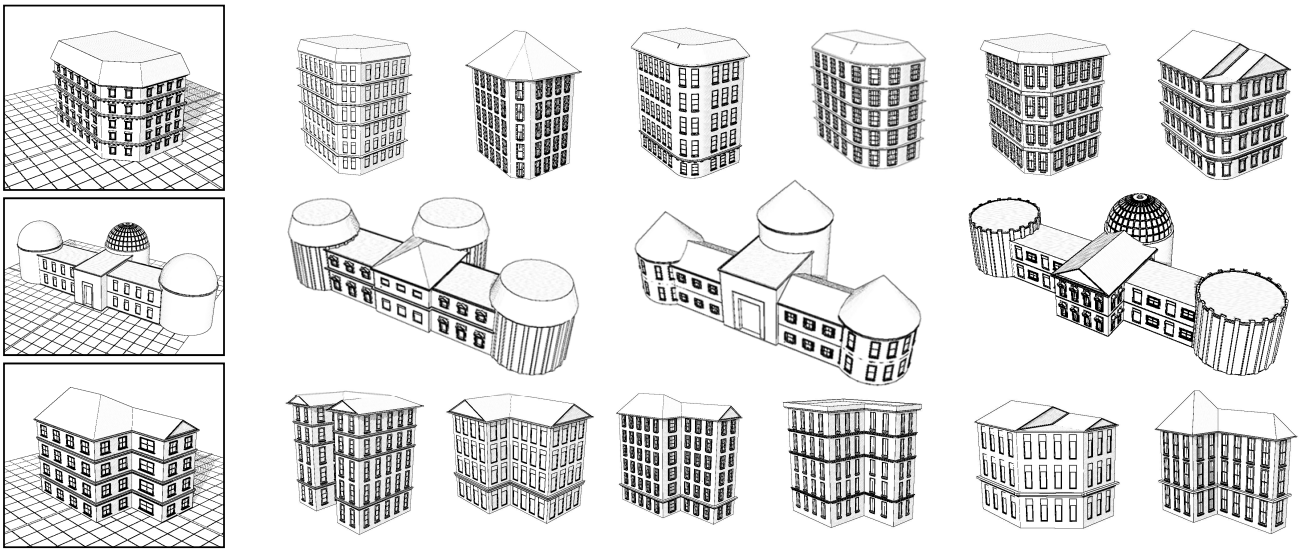

Figure 16: *Variability. Our system can automatically change snippets and snippet parameters to yield model variation. After the user completes a design (leftmost images), our system generates a wide variety of geometry by changing snippets using a probability density function and by randomly changing parameter values within a predefined range.*

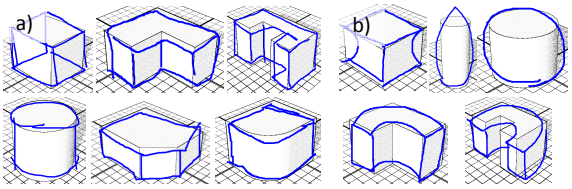

Figure 17: ***Robustness to sketch inaccuracy and style.*** *a) Our system can find the best grammar snippet and its parameter values despite inaccurate sketches and different sketching styles. b) If the user draws a shape that is not covered by a snippet, the CNN will select the closest shape expressible by the available snippets.*

Target building	#clicks	Creation time [min]	#rules
	38.5	6.7	94
	42	6.2	72

Figure 18: ***Quantitative evaluation.*** *All the subjects could successfully create a building that looks very close to the target in less than 7 minutes. The number of rules in the output grammar indicates its complexity*

7.5 User Experience

We had a group of eight users aged from 25 to 34 evaluate our system. All of them are graduate students who received technical education, but they all reported limited knowledge of 3D modeling tools and basic drawing skills. Each participant followed a simple tutorial about our system, which took them around 10 minutes to complete. They then had to reproduce two target buildings using our system, without any time limit. The two targets were both in a Parisian style, but contained subtle differences of mass, roof, windows and ledge to assess user's ability to specify different details. Figure 18 shows the quantitative evaluation of our tool. For each of two target buildings, it shows the number of clicks, the creation time, and the number of rules in the generated grammar which is an indication of generated building complexity. All participants successfully created buildings that look similar to the targets in a very short time (Figure 19). We also asked some participants to create some original buildings after showing them inspirational examples (Figure 20). On a Likert scale from 1 (strongly disagree) to 5 (strongly agree), five participants strongly agreed and three agreed that our system allowed them to achieve the buildings they wanted. They also all agreed that the system interpreted well the shapes they drew. Finally, several participants suggested areas of improvement. In particular, some participants had limited skills in perspective drawing and suggested the use of more traditional front/top/side views to sketch the building mass. It is important to note that even for those who are not good at sketching, our system interpreted their intended shapes and enabled the participants to achieve the desired output.

7.6 Limitations

Our proposed method cannot recover the procedural grammar from an existing sketch. Although our coarse-to-find sketching workflow reflects the way people draw complex objects, the sketching procedure requires the user to mentally decompose a target building. Also, the output variability is limited to that provided by the grammar snippets currently implemented in the system. One option to ameliorate the highlighted differences of Figure 14 is to increase the number of supported snippets. While this may increase the range of buildings features and shapes that can be supported, it will eventually also decrease the CNN's accuracy in recognizing which snippet is currently being sketched. Also, for some irregular shapes using procedural modeling might not be suitable – in those cases, it might be preferable to integrate non-procedural modeling methods such as example-based and/or pure sketch-based methodologies.

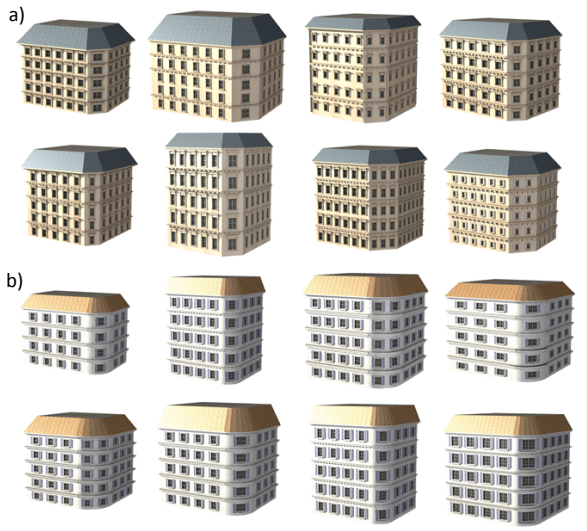

Figure 19: ***User trial output.*** *We show the buildings created by the users for each of the two target buildings in Figure 18.*

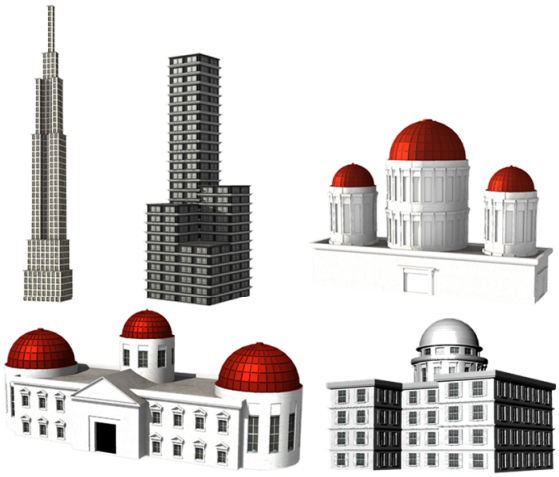

Figure 20: ***Original buildings.*** *We asked some of the participants to create more complex buildings after showing them examples. These imaginary buildings were sketched in less than 15 minutes.*

8 Conclusions and Future Work

We have presented a procedural modeling sketching system that enables a novice user to create a grammar and a corresponding 3D geometry quickly and easily, without the need of writing procedural rules. We achieve interactive sketching of complex grammars by both decomposing the creation process of hierarchical models into coarse-to-fine stages, and by leveraging modern machine-learning algorithms to recognize grammar snippets and estimate their parameters in less than a second. The results indicate that our approach easily and efficiently creates a complex grammar of a desired building shape even for people who do not have any knowledge of procedural modeling grammars.

As future work, we will pursue several items. Currently, our system requires grammar snippets as input. In the future, we would like to automatically partition large grammar definitions into a suitable set of grammar snippets. Also, our system currently uses a fixed-view for sketching. Supporting arbitrary viewpoints significantly increases the complexity of data generation for training, while reducing the accuracy of snippet recognition and parameter estimation. Nevertheless, recent work on CNN-based viewpoint estimation suggests that such an ability is within reach [Su et al. 2015]. Finally, our current machine-learning approach requires a significant amount of memory (e.g., up to few hundred MB per network). We implemented our system using standard deep CNNs using Caffe and have not experimented with the performance of shallower CNNs to find the best tradeoff between memory and accuracy. Another possible solution would be to use a single CNN for all object types, which saves memory but might sacrifice accuracy.

Acknowledgements

We would like to thank Jennifer Neville and Hogun Park for their comments and suggestions about the CNN architectures, our user study participants for their time and feedbacks, and the anonymous reviewers for their constructive suggestions. This research was partially funded by NSF CBET 1250232, NSF IIS 1302172, a Google Research Award, and the French ANR project SEMAPOLIS (ANR-13-CORD-0003).

References

ABDEL-HAMID, O., MOHAMED, A.-R., JIANG, H., DENG, L., PENN, G., AND YU, D. 2014. Convolutional neural networks for speech recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc. 22*, 10, 1533–1545.

AKINLAR, C., AND TOPAL, C. 2011. Edlines: Real-time line segment detection by edge drawing (ed). In *ICIP*, 2837–2840.

ANASTACIO, F., PRUSINKIEWICZ, P., AND SOUSA, M. C. 2009. Sketch-based parameterization of l-systems using illustration-inspired construction lines and depth modulation. *Comp. & Graph. 33*, 4, 440–451.

APPLEGATE, C. S., LAYCOCK, S. D., AND DAY, A. 2012. A sketch-based system for highway design with user-specified regions of influence. *Comp. & Graph. 36*, 6, 685–695.

BAE, S., BALAKRISHNAN, R., AND SINGH, K. 2008. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *User Interface Software and Technology*.

BELL, S., AND BALA, K. 2015. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. Graph. 34*, 4, 98:1–98:10.

CHEN, G., ESCH, G., WONKA, P., MÜLLER, P., AND ZHANG, E. 2008. Interactive procedural street modeling. *ACM Trans. Graph. 27*, 3, 103:1–103:10.

CHEN, X., KANG, S. B., XU, Y.-Q., DORSEY, J., AND SHUM, H.-Y. 2008. Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2, 11:1–11:15.

EIGEN, D., PUHRSCH, C., AND FERGUS, R. 2014. Depth map prediction from a single image using a multi-scale deep network. In *Neural Information Processing Systems (NIPS)*.

EISSEN, K., AND STEUR, R. 2009. *Sketching: Drawing Techniques for Product Designers*. BIS Publishers.

EITZ, M., RICHTER, R., BOUBEKEUR, T., HILDEBRAND, K., AND ALEXA, M. 2012. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4, 31:1–31:10.

EMILIEN, A., VIMONT, U., CANI, M.-P., POULIN, P., AND BENES, B. 2015. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Trans. Graph. 34*, 4, 106:1–106:11.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proc. of Siggraph*, 409–416.

IJIRI, T., OWADA, S., AND IGARASHI, T. 2006. The sketch l-system: Global control of tree modeling using free-form strokes. In *Smart Graphics*, Springer, 138–146.

JAMPANI, V., NOWOZIN, S., LOPER, M., AND GEHLER, P. V. 2015. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *Computer Vision and Image Understanding 136*, 32 – 44.

JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11, 2278–2324.

LIPP, M., WONKA, P., AND WIMMER, M. 2008. Interactive visual editing of grammars for procedural architecture. *ACM Trans. Graph. 27*, 3, 102:1–102:10.

LIPSON, H., AND SHPITALNI, M. 1996. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design 28*, 651–663.

LIPSON, H., AND SHPITALNI, M. 2000. Conceptual design and analysis by sketching. *Artif. Intell. Eng. Des. Anal. Manuf. 14*, 5, 391–401.

LONGAY, S., RUNIONS, A., BOUDON, F., AND PRUSINKIEWICZ, P. 2012. Treesketch: Interactive procedural modeling of trees on a tablet. In *SBIM*, Eurographics Association, 107–120.

LOOMIS, A., AND ROSS, A. 2014. *I'd Love to Draw*. Titan Books.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3, 614–623.

OLSEN, L., SAMAVATI, F. F., SOUSA, M. C., AND JORGE, J. A. 2009. Sketch-based modeling: A survey. *Comp. & Graph. 33*, 1, 85–103.

PARISH, Y. I., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Comp. graphics and interactive techniques*, ACM, 301–308.

PFISTER, T., CHARLES, J., AND ZISSERMAN, A. 2015. Flowing convnets for human pose estimation in videos. In *ICCV*.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 2012. *The algorithmic beauty of plants*. Springer Science & Business Media.

RITCHIE, D., MILDENHALL, B., GOODMAN, N. D., AND HANRAHAN, P. 2015. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Trans. Graph. 34*, 4, 105:1–105:11.

RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., AND FEI-FEI, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV 115*, 3, 211–252.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3d scaffolds. *ACM Trans. Graph. 28*, 5, 149:1–149:10.

SHTOF, A., AGATHOS, A., GINGOLD, Y., SHAMIR, A., AND COHEN-OR, D. 2013. Geosemantic snapping for sketch-based modeling. *Comp. Graph. Forum 32*, 2, 245–253.

SMELIK, R., TUTENEL, T., DE KRAKER, K. J., AND BIDARRA, R. 2010. Interactive creation of virtual worlds using procedural sketching. In *Proceedings of eurographics*.

SMELIK, R. M., TUTENEL, T., BIDARRA, R., AND BENES, B. 2014. A survey on procedural modelling for virtual worlds. In *Comp. Graph. Forum*, vol. 33, 31–50.

SMITH, A. R. 1984. Plants, fractals, and formal languages. *SIGGRAPH Comput. Graph. 18*, 3, 1–10.

STAVA, O., PIRK, S., KRATT, J., CHEN, B., MCH, R., DEUSSEN, O., AND BENES, B. 2014. Inverse procedural modelling of trees. *Comp. Graph. Forum 33*, 6, 118–131.

SU, H., QI, C. R., LI, Y., AND GUIBAS, L. J. 2015. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *ICCV*.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2, 11:1–11:14.

VANEGAS, C. A., ALIAGA, D. G., WONKA, P., MÜLLER, P., WADDELL, P., AND WATSON, B. 2010. Modelling the appearance and behaviour of urban spaces. *Comp. Graph. Forum 29*, 1, 25–42.

VANEGAS, C. A., GARCIA-DORADO, I., ALIAGA, D. G., BENES, B., AND WADDELL, P. 2012. Inverse design of urban procedural models. *ACM Trans. Graph. 31*, 6, 168:1–168:11.

WANG, F., KANG, L., AND LI, Y. 2015. Sketch-based 3d shape retrieval using convolutional neural networks. *arXiv preprint arXiv:1504.03504*.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. Graph. 22*, 3, 669–677.

XIE, X., XU, K., MITRA, N. J., COHEN-OR, D., GONG, W., SU, Q., AND CHEN, B. 2013. Sketch-to-design: Context-based part assembly. *Comp. Graph. Forum 32*, 8, 233–245.

XU, B., CHANG, W., SHEFFER, A., BOUSSEAU, A., MCCRAE, J., AND SINGH, K. 2014. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. on Graph. 33*, 4, 131:1–131:13.

XUE, T., LIU, J., AND TANG, X. 2012. Example-based 3d object reconstruction from line drawings. In *CVPR*, IEEE, 302–309.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. Sketch: An interface for sketching 3d scenes. In *Computer Graphics, Proceedings of Siggraph 1996*, 163–170.

ZHENG, Y., LIU, H., DORSEY, J., AND MITRA, N. J. 2016. Smartcanvas; context-inferred interpretation of sketches for preparatory design studies. *Comp. Graph. Forum*.